TRAINING METHODOLOGY FOR A MULTIPLICATION FREE IMPLEMENTABLE OPERATOR BASED NEURAL NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

OZAN YILDIZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2017

Approval of the thesis:

TRAINING METHODOLOGY FOR A MULTIPLICATION FREE IMPLEMENTABLE OPERATOR BASED NEURAL NETWORKS

submitted by **OZAN YILDIZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. Fatoş T. Yarman Vural
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Uluç Saranlı
Computer Engineering Department, METU _____

Prof. Dr. Fatoş T. Yarman Vural
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Murat Manguoğlu
Computer Engineering Department, METU _____

Assist. Prof. Dr. Emre Akbaş
Computer Engineering Department, METU _____

Assist. Prof. Dr. Tolga Çukur
Electrical and Electronics Engineering Dept., Bilkent University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: OZAN YILDIZ

Signature :

ABSTRACT

Training Methodology for a Multiplication Free Implementable Operator Based Neural Networks

Yıldız, Ozan
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Fatoş T. Yarman Vural

August 2017, 142 pages

Technological advances opened new possibilities for computing environments including smart phones, smart appliances, and drones. Engineers try to make these devices smart, self-sustaining through usage of machine learning techniques. However, most of the mobile environments have limited resources like memory, computing power and battery, and consequently traditional machine learning algorithms which require relatively high resources might not be suitable for them. Therefore, efficient versions of traditional machine learning algorithms receives interest for these kinds of environments. Recently, an operator named the ef-operator, which avoids multiplication is proposed as an alternative to classic vector multiplication. Recent studies, showed that ef-operator can be used on machine learning problems with small degradation on performance to gain energy efficiency. This thesis concerns with the application of this ef-operator over artificial neural networks. An artificial neural network architecture based of this ef-operator proposed which can approximate any Lebesgue integrable function. Applicability of standard backpropagation algorithm for this new

network architecture is analyzed and a modified version of backpropagation algorithm with a line search step proposed for training this network architecture.

Keywords: Artificial Neural Networks, Mulitplication Free Operator, Backpropagation, Line Search

ÖZ

ÇARPMASIZ HESAPLANABİLEN OPERATÖR BAZLI YAPAY SİNİR AĞLARI İÇİN ÖĞRENME METODU

Yıldız, Ozan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Fatoş T. Yarman Vural

Ağustos 2017, 142 sayfa

Teknolojik gelişmeler sonucu drone, akıllı telefon ve ev aletleri gibi cihazlarda hesaplamalar yapılmaya başlandı. Mühendisler, bu cihazları makine öğrenme teknikleri yardımıyla akıllı ve kendi kendine idare eden bir hale getirmeye çalışıyor. Ancak, bu cihazların büyük çoğunluğu sınırlı hafıza, hesaplama kapasitesi ve bataryaya sahip. Dolayısıyla, fazla kaynak gerektiren standart makine öğrenme teknikleri bu cihazlar için çok uygun değil. Bu nedenle, enerji verimli makine öğrenme teknikleri bu cihazlarda kullanılmak üzere ilgi görmektedir. Yakın zamanda, klasik vektör çarpımı yerine ef-operatör isimli çarpmadan kaçınan bir operatör önerildi. Bu operatörün makine öğrenme tekniklerinde, ufak bir performans kaybıyla, enerji verimliliğini arttırmak için kullanılabileceği geçmiş çalışmalarda gösterilmiş. Bu tezde, ef-operatörünün yapay sinir ağlarında kullanılabilirliği irdelendi. Lebesgue integrallenebilir fonksiyonları yakınsama yeteneğine sahip ef-operatörüne dayalı bir yapay sinir ağı yapısı önerisi önerildi. Önerilen bu yapay sinir ağı yapısını eğitmek için standart geri yayılım

algoritması incelendi ve bu yapay sinir ağlarının eğitmek için doğrusal arama içeren bir geri yayılım algoritması önerildi.

Anahtar Kelimeler: Yapay Sinir Ağları, Çarpmasız Hesaplanabilen Operatör, Geri Yayılım, Doğrusal Arama

To my family

ACKNOWLEDGMENTS

Foremost, I would like thank to my thesis advisor, Prof. Dr. Fatoş T. Yarman-Vural, for her endless support. I am thankful for the freedom she gave me during research and her comments when I am stuck. I would not be able to complete this work without her support, insights and helpful comments.

I would like to also thank to Prof. Dr. Enis A. Çetin for his guidance over energy efficient neural networks and ef-operator. This work would not be possible without his suggestions.

Beside my thesis advisor, Prof. Dr. Fatoş T. Yarman-Vural and Prof. Dr. Enis A. Çetin, I would like to thank rest of my jury committee members, Assoc. Prof. Dr. Uluç Saranlı, Assoc. Prof. Dr. Murat Manguoğlu, Assist. Prof. Dr. Emre Akbaş, and Assist. Prof. Dr. Tolga Çukur for their encouragement, helpful comments, and challenging questions.

In addition to my jury committee members, I would like to thank all professors, whom I took a course from during my studies at METU, for the knowledge and perspective they given to me. I would like to thank Dr. Muhiddin Uğuz, Prof. Dr. Gülin Ercan, Assoc. Prof. Dr. Ali Ulaş Özgür Kişisel and other professors from METU mathematics deparment for the mathematical background they gave me during my undergraduate education and I would like to thank Prof. Dr. Faruk Polat, Prof. Dr. İsmail Hakkı Toroslu, Assoc. Prof. Dr. Sinan Kalkan and other professors from METU computeger engineering department for the computer science background they gave during my undergraduate and master education. Lastly, I would like to Assoc. Prof. Dr. Azer Kerimov, Prof. Dr. Şahin Emrah for their continuous guidance over years.

I would like to also thank my friends for their support. I would like to specifically thank to Imagelab members, Barış Nasır, Arman Afrasiyabi, Dr. Itır Önal Ertuğrul, Hazal Moğultay, Burak Velioğlu, Baran Barış Kıvılcım, and Abdullah Al-shihabi. Working with them for last two years was a honor. I am grateful to my friends Hüseyin Aydın and Irmak Doğan for their help during my master studies. I also want to Cansu Yılmaz, Bensu Koçyiğitoğlu and Hande İbili for their support. I would not make it most of the challenges I encountered without their moral support. Lastly, I would like to thank my teammates from METU Orienteering and Navigation Team. I do not believe, I would make the college without their presence, continuous support and fun times we spent with.

Finally, I would like to my family, my dear father Kanber, my dear mother Esma and my dear sister Burcu for their endless support and encouragements. Without their continuous support through all my life, I would not be able to achieve half of the things I achieved.

Also, I acknowledge the support of TÜBİTAK (The Scientific and Technological Research Council of Turkey) BİDEB through 2210-E graduate student fellowship during my M.Sc. education.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

ADAM	Adaptive Modment Estimation Method
ANN	Artificial Neural Network
BLS	Backpropagation with Line Search
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transformation
EFNN	EF-operator based Neural Network
FNN	Feed-forward Neural Network
FPGA	Field-programmable gate array
MSE	Mean Square Error
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Netowrk
SCU	Sign computing unit
SGD	Stochastic Gradient Descent

CHAPTER 1

INTRODUCTION

Artificial neural networks are biologically inspired powerful machine learning algorithms with a variety of application areas with relatively high success rates compared to many popular statistical learning methods. ANNs can be used on various machine learning problems including classification, regression, prediction and function approximation [58]. Improvements in the hardware capabilities and increase of available data allow ANNs to be used on many areas with human level precision. Recently, ANNs are successfully applied to many real world learning problems including computer vision, natural language processing, recommendation systems [51]. Increased application areas of ANNs created demand for ANNs on mobile environments such as mobile phones and drones which have limited resources. There are some applications of ANNs on mobile environments, such as flight control, path estimation [13], obstacle avoidance and human recognition [32] for unmanned aerial vehicles and drones. There are ongoing research on making ANNs efficient so that they can be used on systems with limited resources in addition to researches aiming to perform better than state-of-the art techniques on systems designed to be used for computationally intensive tasks.

The studies about making ANNs energy efficient are focusing on approximating multiplication operations and complex mathematical functions. A popular way of approximating multiplication operation is usage of low-precision floating-point numbers [84, 15] for the parameters of ANNs. The extreme case of low-precision is binary parameters also received a lot of attention [16, 18, 20, 19, 70]. Approximations of complex mathematical functions are based on approximations with polyno-

mials and piece-wise linear functions [82, 46, 67, 14]. Approximations of complex mathematical functions These studies showed that approximate calculations of ANNs allows energy efficiency without sacrificing performance.

Recently, some energy efficient operator which avoids multiplication is proposed to replace the classical operators such as inner product [83]. Ef-operator is proposed in [83] is a binary operator designed to be energy-efficient through avoidance of floating-point multiplication. It is initially proposed in the context of signal processing [83] and afterwards, it is applied to various machine learning problems. Previous work related to this operator shows interesting possibilities and usage areas including artificial neural networks. However, most of these previous works focused on applicability of this operator to existing machine learning methods and evaluated these methods through experiments. This thesis aims to reinforce results of previous studies with theoretical analysis and suggests a new training method for neural networks based on ef-operator.

1.1 Main Contributions

The main contribution of this thesis is the inspection of applicability of the ef-operator on artificial neural networks. We did this inpsection by defining ef-operator based neural network, EFNN, and analyzing this new network. Analyze of this new network includes inspection of the representation capabilities of this network, inspection of the dependence between mappings computed by these networks and their parameters, and inspection of the applicability of standard backpropagation algorithm for the training EFNNs.

This thesis is not the first study which applies ef-operator to artificial neural networks. Previously, Akbaş et al. [5] introduced multiplication-free neural networks in which they replaced standard vector multiplication with ef-operator. Multiplication-free neural networks failed to match the state-of-the art classification accuracies in their experiments. Their experiments showed that classification accuracy can drop as much as 10% with these networks. Recently, we [4] proposed an improved version of

multiplication-free neural networks, called additive neural networks, with additional parameters. Additive neural networks achieved near state-of-the art classification accuracy on MNIST [52] data set. The drop on classification accuracy was less than 2%. Also, we showed that these networks are capable of approximating any Lebesgue integrable function arbitrarily well.

- This thesis proposes a simpler version of additive neural networks with removal of extra parameters introduced by additive neural networks. This proposed network is called EF-operator based Neural Network, EFNN. EFNN achieves similar representation capabilities through carefully designed hidden layer structures. EFNNs employ different type of hidden layers in different levels, unlike the multiplication-free neural networks which uses same layer structure for each hidden layers. Representation capabilities of EFNNs are inspected for identity and ReLU activation functions which can be implemented without any complex mathematical operations. Furthermore, dependence between mappings computed by the EFNNs and their parameters for constant set of input vectors and activation function inspected.

- The applicability of standard backpropagation for the training EFNNs analyzed. This analysis showed that standard backpropagation may fail to train EFNNs due to non-differentiable nature of ef-operator. A simple modification over standard backpropagation which introduces a line search step, proposed to overcome training problems.

1.2 Outline of the Thesis

A brief summary of artificial neural networks (ANN) is provided in Chapter 2. This summary starts with explaining what is an ANN and when they are used. Then, different types of ANNs are surveyed with a focus on efficient variants. Moreover, representation capabilities of ANNs and training methodologies for ANNs are discussed.

Ef-operator is introduced in Chapter 3. Basic mathematical properties of ef-operator are explained. Similarities and differences between this new operator and classic vector multiplication are discussed. An ANN model based on ef-operator, EFNN also introduced in Chapter 3. Representational capabilities of this model is inspected. Finally, dependence between parameters of EFNNs' and the mappings computed by EFNNs is inspected. This inspection constitutes a basis for the proposed training algorithm at chapter 4.

A modified backpropagation algorithm with line search for the training EFNNs is introduced in Chapter 4. A brief discussion of why standard backpropagation algorithm is not enough to train EFNNs are given. This discussion is followed by the explanation of the proposed algorithm. Next, the convergence of the proposed algorithm is discussed. Finally, the quality of solutions founded by the new algorithm is discussed.

An experimental analysis of proposed algorithm at Chapter 4, is given in Chapter 5. Proposed algorithm experimentally is compared with standard backpropagation algorithm. Both algorithm are used for training various data sets with various hyper-parameters. Comparison of training performances, as well as classifier performances between two algorithms depending on hyper-parameters is given.

A brief overview of what is done by this thesis is given in Chapter 6. Possible future directions for this work is also included in this chapter.

CHAPTER 2

BRIEF OVERVIEW OF ARTIFICIAL NEURAL NETWORKS

Since this thesis about representing a new energy efficient neural network, we devote this chapter to a brief analysis of Artificial Neural Networks (ANN). In the following sections a brief history of ANNs and their application areas will be summarized. Then, artificial neuron and various network architectures will be studied. Efficient neural network variants in addition to state-of-the-art architectures will be explained and downsides of efficient variants will be discussed. Chapter will continue with the discussion of representation capabilities of neural networks. Finally, chapter will be concluded with overview of gradient based training methods for ANNs.

2.1 A Brief History of Artificial Neural Networks

The history of artificial neurons can be traced back to the pioneering work of McCulloch and Pitts, in 1941 [57]. In this architecture the neurons are connected by weighted and directed arcs, where each neuron outputs weighted sum of its inputs. A threshold function is applied on these weighted sums to determine if a neuron fires or not. Therefore, the output of each neuron is binary in this set-up. Weights of these directed arcs between neurons are set by the researcher to implement a specific functionality, unlike the modern neural networks. Hebb proposed first training algorithm for artificial neural networks [36]. This algorithm was based on the assumption that if two neurons fires simultaneously then the connection between these two neurons should be stronger than that of not fire together. Frank Rosenblatt with the help of several other scientists, including Block [10], Minsky and Papert [60] developed

the perceptron [72]. Perceptrons are connected by weighted and directed arcs, and produce an output using their weighted inputs, similar to neurons suggested by McCulloch and Pitts [57]. Rosenblatt also suggested perceptron learning rule when he suggested the perceptron. Perceptron rule adjusts weights between perceptrons iteratively to implement desired functionality. Although, perceptron rule is more powerful than Hebb rule, it is applicable to single layer networks which caused a pause on the development of ANNs until the emergence of backpropagation algorithm [29]. The perceptron learning rule is failed to train multi-layered networks which A training method for ANNs through propagation of error from output layer to hidden layers is proposed by Werbos in 1974 [87], However, this algorithm did not receive much attention until mid 80s when Parker [68] and LeCun [50] independently rediscovered the algorithm. Although, ANNs received a renewed interest after backpropagation algorithm, interest disappeared quickly after realization of ANNs do not scale well to large problems; partly because of the hardware limitations [8]. ANNs received attention after the appearance of deep belief networks, which uses an unsupervised learning method to train weights in a greedy fashion to construct deep network with initial weights coming from initial greedy training [38]. This new method allowed training of deeper networks and made a breakthrough on ANN literature enabling ANNs to achieve very high performance levels in some problems, such as OCR [12] and face recognition [90]. Development of new hardwares allowed training of larger networks than networks used on 80s and 90s, allowing more representational capacity and increasing precision levels. Currently, ANNs running on GPUs in use for greatly diverse set of problems with great success [8]. These problems include object recognition [49, 81], speech recognition [59, 37], and language translation [45, 79].

2.2 Perceptron

A perceptron is a mathematical model which takes a real-valued input vector and produce binary outputs of +1 and -1 according to linear combination of inputs being greater than some threshold value or not. Formally, perceptron is a function p which

takes input vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_{d_0}]$ and outputs

$$p(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\boldsymbol{x} + b > 0, \\ -1 & \text{otherwise,} \end{cases} \tag{2.1}$$

where $\mathbf{w} = [\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_{d_0}]$ is weights of the perceptron determining individual contribution of each input and b is bias of the perceptron determining a threshold value for the activation of perceptron [61]. A graphical representation of perceptron is given in Figure 2.1.

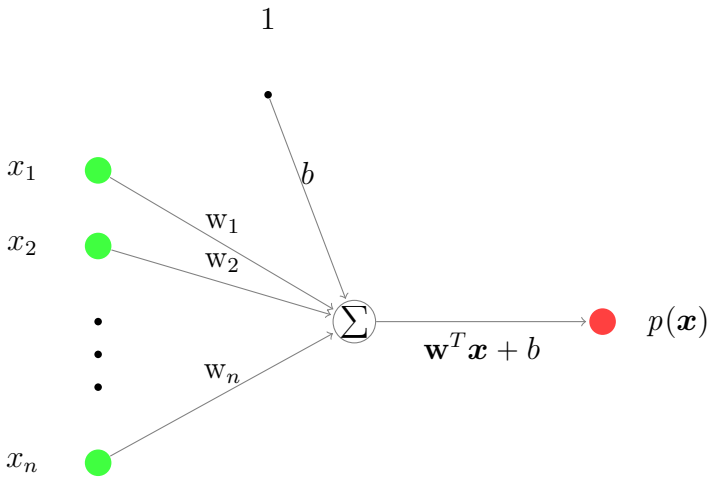


Figure 2.1: Visual representation of a perceptron which maps an input d_0-dimensional real input vector $\boldsymbol{x}$ to an output vector $p(\boldsymbol{x}) = \mathbf{w}^T\boldsymbol{x} + b$.

ANNs based on perceptrons are created connecting the outputs of several perceptrons to the input of other perceptrons.

2.3 Feed-forward Neural Networks

Feed-forward neural networks (FNN) are special kind of neural networks in which neurons are partitioned into disjoint sets called layers. Layers are indexed from 1 to l, and output of a neuron at layer i can be connected only to neurons at layer $i + 1$ as input. Inputs are considered as layer 0. Each layer computes linear combinations of outputs of previous layer's and outputs these linear combinations after applying activation function. Formally, each layer is a function $\mathbf{l}_i$ which takes output of previous

layer $\mathbf{l}_{i-1}(\boldsymbol{x};\boldsymbol{\theta})$ and outputs

$$\mathbf{l}_i(\boldsymbol{x};\boldsymbol{\theta}) = f(\mathbf{W}_i{}^T\mathbf{l}_{i-1}(+;\mathbf{b})[i]), \tag{2.2}$$

where $\boldsymbol{x}$ is d_0-dimensional input vector, f is the activation function, $\boldsymbol{\theta}$ is the parameters of network, $\mathbf{W}_i$ is weights of i^{th} layer, $\mathbf{b}_i$ is biases of i^{th} layer, and $\mathbf{l}_0(\boldsymbol{x};\boldsymbol{\theta})$ is the input vector [29].

2.4 Energy Efficient Neural Network Variants

Neural networks are powerful tools, however, to a certain extent their power comes with a high computational cost. They require computationally costly numeric calculations such as multiplication, present in the linear transformation of perceptrons, and complex mathematical functions, necessary for the calculation of activation functions. There are various approaches to make ANNs efficient from different perspective. Two of the most popular approaches are limiting weights to some special set so that computation can be made more efficiently, and approximating multiplications and other complex mathematical functions with computationally more efficient functions.

2.4.1 Neural Networks with Restricted Weights

One popular way of decreasing the computational cost of a ANN is to put some restrictions on the weights. If the weights are limited to a special subset of floating-points, then it is possible to compute them efficiently without using hardware implementation of general purpose multiplication instruction. These special subsets can be powers of 2, low-precision floating-points or simple set of $\{-1,\ 0,\ 1\}$. One of the first example of these kinds of networks is proposed by Marchesi et al. [56]. They showed that restriction of weights to subset of integers which are power of two or sums of power of twos can be used on feed-forward forward networks with binary outputs. The computational and energy efficiency of such weights on multiplication, which can be computed with addition and shifting shown by White and Elmasry [88], and Lim and Liu [54].

Usage of low precision floating-point numbers as weights, originates from the error-resilience of ANNs to small noise[34]. Both theoretically and experimentally usability of low precision floating-points can be used on ANNs. Holi and Hwang [41] gave a theoretical analysis of effects of low precision floating-point numbers on usage of neural networks. Hammerstrom [35] used 16-bit fixed floating-point computation for computing and later Höhfeld and Fahlman [40] used 8-bit floating-point computation. Recent studies using low precision arithmetic include the study done by Vanhoucke et al. [84] in which 8-bit approximations of high precision weights used on deployment, study done by Chen et al. [15] in which 32-bit fixed point floating-point arithmetic used, and study done by Gupta et al. [34] in which 16-bit fixed point floating-point arithmetic used.

Binary weights are extreme case of low precision arithmetic, where weights are limited to set $\{-1, 1\}$. Binary weighted ANNs based on the work of Hwang and Sung [43], in which they showed restriction of weights to the set $\{-1, 0, 1\}$ can be used with minimal loss in the performance. The set $\{-1, 0, 1\}$ requires 3-bit fixed point floating-point arithmetic allowing these networks to achieve both energy and memory efficiency. Their work is later applied on binary setting where weights can be represented with single bit and any arithmetic operation can be carried with logical instructions which are more energy efficient than usual arithmetic instructions [47]. Binary multi-layer neural networks are suggested by Cheng et al. [16] and followed by bitwise neural networks [47], BinaryNet [18], binarized neural networks [20], BinaryConnect [19], and XNOR-Net [70]. All of these works attained near state-of-the art performances in some problems, while saving large amount of energy using logical instructions instead of multiplication and memory using single bit to represent weights.

2.4.2 Approximate Computations

Approximate calculation of complex mathematical functions is another method to efficiently compute ANNs. Idea behind the usage of approximate computations is resilience of neural networks to errors [17], similar to usage of low precision arith-

metic. Two popular approximation approaches are approximation of the activation function and approximation of multiplication operation inside neurons. Studies using both approaches summarized at below paragraph.

Sigmoidal activation functions such as sigmoid function and hyperbolic tangent function ($\tanh$) requires computation of complex mathematical operations which are not energy efficient. There are various approaches for approximating these functions using piece-wise linear functions, polynomials and discrete cosine transformation (DCT). Recent works include Torki et al. [82], Khodja et al. [46], Panicker and Babu [67], Callejas-Molina et al. [14], Abdelsalam et al. [1, 2]. Torki et al. [82] proposed a digital hardware implementation which computes sigmoidal activation function approximately with polynomials. Similar to Faiedh et al.[82], Khodja et al. [46], proposed a field-programmable gate array (FPGA) implementation, which computes sigmoid function approximately with polynomials. Panicker and Babu [67] proposed a FPGA implementation which computes sigmoid function approximately with piece-wise linear functions. Callejas-Molina et al. [14], proposed a digital hardware implementation which computes $\tanh$ approximately with piece-wise linear functions. Abdelsalam et al. [1, 2] proposed a FPGA implementation, which computes $\tanh$ approximately with DCT interpolation.

Approximation of multiplication operations based on the fact that artificial neurons resilient to noise and approximation of multiplication operation is just another source of error. A neuron is called resilient to error if the small changes on the input of the neuron causes small changes on the output of the ANN [85]. Venkataramani et al. [85] proposed a method to determine neurons resilient to error based on gradients computed during backpropagation. They approximate the multiplication operation on these neurons through low precision multiplication and they also approximate activation functions through piece linear functions. Zhang et al. [92] theoretically analyzed approximating error resilient neurons and proposed an iterative method based on the ranking of neurons to determine neurons whose output does not affect the performance of ANN significantly. They approximated multiplier circuits for energy efficiency. Du et al. [24] proposed an inexact computation of multiplication

through inexact logic units using inexact logic minimization approach proposed by Lingamneni et al. [55]. Sarwar et al. [74] proposed approximation of multiplication using alphabet set. Suggested multiplication method splits the weights into parts, so that multiplication of corresponding parts can be computed using a look up table and results can be combined using shift and addition operations. Mrazek el al. [62] proposed approximation of multiplication on all neurons using the same approximation method. Their results show that large amount of power reduction can be gained through approximating the multiplication operations with small loss on classification accuracy.

2.5 Representation Capabilities of Artificial Neural Networks

A neural network can be considered as a mapping from an input space to an output space. Although, there might be mappings between these spaces, which cannot be represented by classical neural networks, Cybenko showed that depending on the activation function, neural networks is capable of representing large variety of functions.

An activation function σ is called sigmoidal, if

$$\begin{aligned} \lim_{x \to \infty} \sigma(x) &= 1, \\ \lim_{x \to -\infty} \sigma(x) &= 0. \end{aligned} \tag{2.3}$$

The definition of sigmoidal functions does not impose strict restrictions on a function, and consequently large variety of functions are classified as sigmoidal. Although, some popular activation functions such as sigmoid and tanh are not sigmoidal by definition, they can be converted to one with affine transformations. Cybenko proved the following two theorems in his pioneering paper[21] about sigmoidal functions, which shows the function approximation capabilities of neural networks when activation function is an affine transformation of a sigmoidal function.

Theorem 2.5.1. *Let σ_c be a continuous sigmoidal function. Then, finite sums of the form*

$$\mathcal{G}_{\sigma_c} = \left\{ \sum_{i=1}^{n} \alpha_i \sigma(\mathbf{w}_i^T \boldsymbol{x} + b_i) | \mathbf{w}_i \in \mathbb{R}^{d_0},\ \alpha_i, b_i \in \mathbb{R},\ n \in \mathbb{Z}^+ \right\} \tag{2.4}$$

are dense in $C(I_{d_0})$ where $C(I_{d_0})$ is the metric space of continuous functions over d_0-dimensional unit cube with supremum norm.

Theorem 2.5.1 implies that, when there is no restriction on number of neurons, the network can approximate any continuous function arbitrarily well, provided that the activation function is an affine transformation of continuous sigmoidal function. This theorem applies to activation functions such as sigmoid and tanh.

Theorem 2.5.2. *Let σ_b be a bounded, measurable sigmoidal function. Then finite sums of the form*

$$\mathcal{G}_{\sigma_b} = \left\{ \sum_{i=1}^{n} \alpha_i \sigma(\mathbf{w}_i^T \mathbf{x} + b_i) | \mathbf{w}_i \in \mathbb{R}^{d_0},\ \alpha_i, b_i \in \mathbb{R},\ n \in \mathbb{Z}^+ \right\} \tag{2.5}$$

are dense in $L^1(I_{d_0})$ where $L^1(I_{d_0})$ is the metric space of Lebesgue integrable functions over d_0-dimensional unit cube with L1 norm.

Theorem 2.5.2 is slightly weaker form of 2.5.1, which states that, when there is no restriction on number of neurons, network can approximate any bounded, measurable function arbitrarily well, provided that the activation function is an affine transformation of bounded, measurable sigmoidal function. This theorem applies to any continuous sigmoidal function, as well as some non-continuous functions such as sign.

2.6 Training Methods for Artificial Neural Networks

Training an ANN means finding a "desirable" set of parameters, so that the error between desired outputs and actual outputs of network with respect to an objective function, L, is minimized. Until 1980s, multiple algorithms suggested for training ANNs [36, 72, 89]. However, none of them scaled to networks with more than two

layers until the appearance of backpropagation algorithm [29]. Currently, backpropagation algorithm with gradient descent optimization techniques are one of the most popular training method for ANNs [73].

2.6.1 Back Propagation

Backpropagation algorithm is a training method for ANNs. It was designed to be used on supervised learning problems, where desired outputs are available, which is the label of each input in case of classification problem. Backpropagation algorithm mainly consists of four stages, (i) computing outputs of network, (ii) computing error between desired outputs and the outputs computed by the network with respect to objective function, (iii) computing update values minimizing error, and (iv) updating parameters with previously computed update values [29].

Formally, let $\mathcal{X}$ be set of input vectors and $\mathcal{Y}$ their corresponding desired output vectors, L be objective function which will be minimized through training, M be any optimization algorithm which returns update values for parameters given error, $o(\boldsymbol{x};\, \boldsymbol{\theta})$ be the output of ANN with parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_0$ be initial set of parameters. Backpropagation method is proved in algorithm 2.1 using these notations.

Algorithm 2.1 Backpropagation algorithm

procedure BACKPROP($\mathcal{X}$, $\mathcal{Y}$, $\mathbf{u}_0$)
  $t \leftarrow 0$
  while not converged **do**
    $t \leftarrow t + 1$
    Compute outputs with forward pass, $\mathcal{O}_t \leftarrow o(\mathcal{X};\, \mathbf{u}_{t-1})$
    Compute error between desired output and actual output, $L_t \leftarrow L(\mathcal{O}_t, \mathcal{Y})$
    Compute update values, $\Delta\mathbf{u}_{t-1} \leftarrow M(L,\, \mathbf{u}_{t-1})$
    Update parameters, $U_t \leftarrow \mathbf{u}_{t-1} + \Delta\mathbf{u}_{t-1}$
  end while
  return $\mathbf{u}_t$
end procedure

2.6.2 Gradient Descent Optimization Methods

In order to update the objective functions L at each iteration a popular method, called gradient descent, is used. These methods can be easily generalized to multi-layer architectures from single layer using chain rule [58]. Gradient descent method iteratively minimizes the objective function L with respect to its parameters $\boldsymbol{\theta}$ by updating parameters in the opposite direction of the gradient of objective function $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$, where the change is maximal. Magnitude of each update determined by a hyper-parameter called learning rate [73].

There are many variants of gradient descent method on use. These include variants of plain steepest descend method, which differ by how much input data used to compute update values. Stochastic gradient descent method (SGD) is one of the popular variants of plain steepest descent method. There are also more sophisticated variants of gradient descent method, which decide update direction using not only the current gradient, but using previous update values as well. Adaptive moment estimation (ADAM) is one of these more sophisticated variants which is commonly used in practice. Stochastic gradient descent (SGD) and adaptive moment estimation (ADAM) methods will be explained in this section.

2.6.2.1 Stochastic Gradient Descent

Gradient descent method employs equation (2.6) as update rule.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mu \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}), \tag{2.6}$$

where $\boldsymbol{\theta}_t$ is the current parameters, L is the objective function to minimize, μ is learning rate and $\boldsymbol{\theta}_{t+1}$ is the updated parameters. Computation of L from inputs and desired outputs will be discussed momentarily. This update rule moves at the direction, where change on L is maximal [76]. Although, there are algorithms with better convergence rates, this algorithm still receives interest due to its simplicity.

There are variations of gradient descent algorithm depending on how much of input

data is used on computation of L in a single update. Stochastic gradient descent version computes L using a single input vector at each update, in other words at each update stochastic gradient descent optimizes L with respect to the error between network's output on single sample and desired output of that sample. On the other hand, batch gradient descent method computes L with respect to error between network's output on all samples and desired output of those samples. There is also mini-batch gradient descent method which computes L using a small subset of training samples at each update. Both stochastic gradient descent and batch gradient descent are special cases of mini-batch gradient descent algorithm with batch sizes 1 and N, where N is number of training samples. Usually, the term SGD is used when mini-batch gradient descent method used on the literature [73].

2.6.2.2 Adaptive Moment Estimation

Stochastic gradient descent method relies solely on the current gradient and uses fixed learning rate. However, several studies showed that using past gradients in conjunction with current gradient can improve convergence rate [80, 69]. There are also methods, which changes learning rate to accelerate learning such as AdaGrad [25], AdaDelta [91]. Both of these methods increase the stability of gradient descent and improve the convergence rate. Adaptive moment estimation (ADAM) is a popular optimization method which uses both past gradients to compute update direction and dynamically adjusts the learning rate. ADAM method relies on both exponentially decaying average past gradients, $\mathbf{m}_t$, and exponentially decaying average past squared gradients, $\mathbf{v}_t$. These terms calculated according to equations (2.7) and (2.8), as follows;

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t, \tag{2.7}$$

$$\mathbf{v}_t = \beta_2 \mathbf{g}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2, \tag{2.8}$$

where β_1 and β_2 are exponential decay rates, and $\mathbf{g}_t$ is

$$\nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{\theta}_t), \tag{2.9}$$

which is gradient of objective function with respect to current parameters. $\mathbf{m}_t$ and $\mathbf{v}_t$ are initialized with 0 vectors. Afterwards, ADAM method computes bias-corrected estimates from these values using equations (2.10) and (2.11).

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \tag{2.10}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}. \tag{2.11}$$

$$\tag{2.12}$$

Update rule for ADAM method can be formulated using bias-corrected values as in equation (2.13).

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\mu}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon}\hat{\mathbf{m}}_t, \tag{2.13}$$

where μ is learning rate, ε is a scalar, $\boldsymbol{\theta}_t$ is current parameters and $\boldsymbol{\theta}_{t+1}$ is updated parameters. The authors of ADAM method suggested 0.9 for β_1 as default value, 0.999 as default value for β_2 and 10^{-8} as default value for ε. They also showed experimentally that ADAM method works well in practice and produce comparable results with other adaptive learning algorithms [48, 73].

2.6.3 Initialization of Neural Networks and Training

There are various parameters, which needs to be selected before training an ANN and there also various parameters which needs to be selected to employ an optimization method. Important issues regarding the design on an ANN architecture are the activation function, defining number of hidden layers and the number of neurons at each layer. Also, selection of initial parameters is an important problem.

There are well-known algorithms which are guaranteed to successfully train ANNs with single hidden layer [66]. However, training an ANN with at least two hidden layers through backpropagation algorithm is NP-complete problem regardless of selected optimization method [11]. Unfortunately, most of the above problems do not

have formal solutions, instead there are some heuristics, which work in a wide range of practical problems.

2.6.3.1 Selection of Topology

The number of neurons used at hidden layers, as well as how they are distributed into hidden layers directly affects the representation capacity of an ANN. There are theorems, such as universal approximation theorem suggesting ANNs can approximate the desired function arbitrarily well, given a sufficient number of neurons used in hidden layers and appropriate activation function used. However, increasing the number of neurons in a layer increases both complexity of computing layer's output and training process of the ANN [58]. There are various studies showing that the same approximation capabilities can be achieved using deeper networks instead of shallow but large networks [44, 26]. Current trend is toward ANNs with several layers each of which containing small number of neurons. These architectures allow networks to have sufficient capacity to represent most function with relatively small number of parameters. Usually, one employs several topologies to solve a problem and selects the "best" performing topology among these topologies, empirically.

2.6.3.2 Selection of Activation Function

The activation function directly affects the representation capacity of ANNs similar to choice of network topology. For example, if a linear activation function is used, then ANNs can only represent linear functions. Although, choice of activation function restricts the representation capacity of ANNs, large class of activation functions can be effectively used on practice. One such class is the continuous sigmoidal functions, as shown by universal approximation theorem [21]. Activation function also affects the training procedure. If a non-differentiable function is used as the activation function, then the optimization method which requires the gradients can only be applied by an approximation of gradient. Otherwise, they cannot be used on training these ANNs. Similar to selection of topology, one employs several activation functions and

selects the one performing "best" among these functions, empirically. However, one important point on selection of activation functions is, the properties of the activation function such as continuity, differentiability, are shared with the functions computed by neural networks at the end.

2.6.3.3 Selection of Initial Parameters

ANNs usually initialized with randomly chosen small initial parameter, typically from normal distribution. There are some theoretical inspection on initialization of parameters according to the activation function instead of sampling from a random distribution. Glorot et al. [33] suggests that weights, $\mathbf{W}$, of a layer can be initialized with random numbers sampled from the following normal distribution;

$$\mathcal{N}\Big(0,\, \frac{2}{n_{in} + n_{out}}\Big), \tag{2.14}$$

where n_{in} number of neurons in the previous layer, n_{out} number of neurons in the current layer, and $\mathcal{N}$ represents the normal distribution.

2.6.3.4 Selection of Objective Functions

The selection of objective function depends on the type of the problem. Some problems require exact outputs, such as function approximation problems and prediction problems, while other problems can use inexact outputs as long as order of outputs are preserved, such as probability estimation for individual classes for classification problem. Two popular objective functions are mean square error (MSE) and cross-entropy loss. Mean square error is defined between two set of vectors as follows,

$$\frac{1}{N}\sum_{i=1}^{N} \left\|\boldsymbol{o}_i - \boldsymbol{y}_i\right\|_2, \tag{2.15}$$

where $\boldsymbol{o}_i$ is actual output vectors and $\boldsymbol{y}_i$ is desired output vectors for $i = 1,\, 2,\, \ldots,\, N$. Cross-entropy loss is defined between two set of vectors as follows,

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,\,j}\log(o_{i,\,j}), \tag{2.16}$$

where $\boldsymbol{o}_i$ and $\boldsymbol{y}_i$ are C-dimensional vectors for each i [65]. Both vectors assumed to be a probability distribution or equivalently each entry on vectors should be non-negative and sum of entries should be 1. An arbitrary vector can be converted into a probability distribution using softmax function. Outputs of softmax function can be calculated as follows,

$$\frac{\exp(o_i)}{\sum_{j=1}^{C} \exp(o_j)} \tag{2.17}$$

for i^{th} entry of $\boldsymbol{o}$.

2.7 Summary

This chapter presents a brief overview of ANNs. A brief history of ANNs presented at Section 2.1 with focus on development of artificial neurons and training methodologies. Artificial neurons and feed-forward neural networks are discussed at the same section. Then, Section 2.4 summarizes the energy efficient techniques used on neural networks. The representational power of neural networks and universal approximation theorem for feed-forward neural networks with sigmoidal activation functions, is discussed in Section 2.5. Finally, training methodologies for ANNs with focus on back propagation algorithm and optimization methods for weight updates specifically SGD and ADAM, is summarized in Section 2.6.

CHAPTER 3

EF-OPERATOR AND NOVEL NEURAL NETWORK ARCHITECTURE BASED ON EF-OPERATOR

The major motivation of this thesis is to develop an efficient neural network, which can be used by the limited-resource computing environments. We approach this problem by designing a new type of neuron based on an additive operator, called ef-operator, suggested by Çetin et al. [83]. This chapter studies the ef-operator, a binary vector operator which can be computed without multiplication. It is initially designed to be used on machine learning algorithms at the computing environments with limited energy, such as mobile devices, instead of vector product to save energy. Focus of this chapter is how ef-operator can be used on neural networks and how they behave with respect to the parameters of neural networks. We will start with the definition of ef-operator. Then, we analyze its properties including brief comparison with vector product. Next, a feed-forward neural network architecture based on ef-operator is introduced. Afterwards, we study the representation capabilities of ef-operator based feed-forward neural network architecture. Finally, we analyze this feed-forward neural network architecture. The analysis is conducted with respect to the network parameters and how these parameters are affected by the introduction of ef-neurons.

3.1 EF-operator

EF-operator is designed to be computable without any multiplication operation. It is first introduced in 2009 [83] to gain energy efficiency in image processing algo-

rithms. Afterwards, it is used on several other image processing tasks, such as object tracking for infrared videos [22], classification of biomedical images [78] for energy efficiency. It is also used in signal processing tasks to approximate cosine similarity in an energy efficient way [6]. Recently, we applied ef-operator to artificial neural network methods for possible improvements of energy efficiency in artificial neural networks [5], [4].

Definition 3.1.1. *Ef-operator, $\diamond$, between two vectors is defined as follow,*

$$\mathbf{x} \diamond \mathbf{y} := \sum_{i=1}^{d} sign(\mathrm{x}_i \mathrm{y}_i)(|\mathrm{x}_i| + |\mathrm{y}_i|), \tag{3.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two d-dimensional real vectors.

Ef-operator can be written in terms of regular vector multiplications. Although, using regular vector multiplication to represent ef-operator contradict with the main motivation behind the proposition of this operator, usage of well-understood regular vector multiplication makes manipulation of ef-operator easier, for the development of artificial neural network algorithms.

Proposition 3.1.2. *Ef-operator can be computed using vector multiplication as follows,*

$$\mathbf{x} \diamond \mathbf{y} = \mathbf{sign}(\mathbf{x})^T \mathbf{y} + \mathbf{x}^T \mathbf{sign}(\mathbf{y}), \tag{3.2}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two d-dimensional real vectors, and $\mathbf{sign}$ is element-wise $sign$ function.

Proof. The equation (3.2) is a natural outcome of two well-known properties of $sign$ function. Let x and y be two real numbers, then,

1. $sign$ function is multiplicative, i.e

$$sign(xy) = sign(x)sign(y). \tag{3.3}$$

2. A real number can be written in terms of its sign and its magnitude, i.e

$$x = sign(x)|x|. \tag{3.4}$$

Individual sums on the equation (3.1),

$$sign(\mathrm{x}_i\mathrm{y}_i)(|\mathrm{x}_i| + |\mathrm{y}_i|) \tag{3.5}$$

becomes

$$sign(\mathrm{x}_i)sign(\mathrm{y}_i)|\mathrm{x}_i| + sign(\mathrm{x}_i)sign(\mathrm{y}_i)|\mathrm{y}_i|, \tag{3.6}$$

by equation (3.3), which is equivalent to

$$sign(\mathrm{y}_i)\mathrm{x}_i + sign(\mathrm{x}_i)\mathrm{y}_i, \tag{3.7}$$

by equation (3.4), for all $i = 1, 2, \ldots, d$. Hence equation (3.1) becomes,

$$\mathbf{x} \diamond \mathbf{y} = \sum_{i=1}^{d} sign(\mathrm{x}_i)\mathrm{y}_i + \mathrm{x}_i sign(\mathrm{y}_i), \tag{3.8}$$

which is equivalent to

$$\mathbf{x} \diamond \mathbf{y} = \mathbf{sign}(\mathbf{x})^T\mathbf{y} + \mathbf{x}^T\mathbf{sign}(\mathbf{y}), \tag{3.9}$$

as proposed. □

One important consequence of Proposition 3.1.2 is that, ef operator can be easily generalized to matrix-vector form, as follows.

Definition 3.1.3. *Ef-operator, $\diamond$, between a matrix and a vector defined as follow,*

$$\mathbf{X} \diamond \mathbf{y} := \mathbf{sign}(\mathbf{X})^T\mathbf{y} + \mathbf{X}^T\mathbf{sign}(\mathbf{y}), \tag{3.10}$$

where $\mathbf{X}$ is a $d' \times d$-dimensional real matrix and $\mathbf{y}$ is a d-dimensional real vector.

In order to gain some visual insight on ef-operator, we plot the ef-operator applied on two scalars in figure 3.1.

3.1.1 Partial Derivatives of EF-operator

Most of the training methods for ANNs uses partial derivatives of some cost function with respect to their parameters. In order to employ ef-operator to define a new

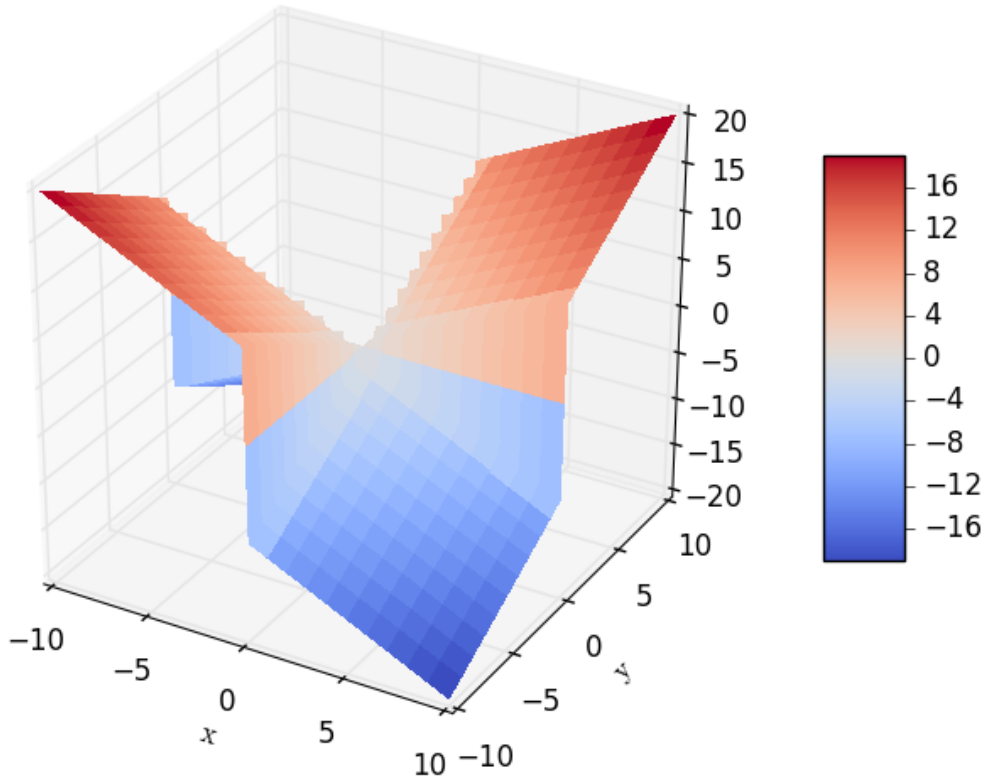


Figure 3.1: Behavior of EF-operator between two 1-D vectors in the interval [-10, 10] as 3-D graph.

ANNs, the partial derivatives are necessary to train these ANNs using well-known algorithms for standard ANNs. One crucial problem about ef-operator is that it is not continuous due to sign function and consequently does not have proper partial derivatives. However, partial derivatives of ef-operator are undefined only at 0 and this is a removable discontinuity. Hence, it can be approximated during numeric calculations using the approach below.

Partial derivatives of ef-operator can be defined as follows;

$$\begin{aligned}\frac{d}{d\mathrm{x}_i}\mathbf{x}\diamond\mathbf{y} =& \frac{d}{d\mathrm{x}_i}(\mathbf{sign}(\mathbf{x})^T\mathbf{y} + \mathbf{x}^T\mathbf{sign}(\mathbf{y})) \\ =& \frac{d}{d\mathrm{x}_i}(sign(\mathrm{x}_i)\mathrm{y}_i) + sign(\mathrm{y}_i) \\ \approx& sign(\mathrm{y}_i),\end{aligned} \tag{3.11}$$

for all $i = 1, 2, \ldots, d$ as

$$\lim_{\mathrm{x}\to 0^+} \frac{d}{d\mathrm{x}} sign(\mathrm{x}) = 0 = \lim_{\mathrm{x}\to 0^-} \frac{d}{d\mathrm{x}} sign(\mathrm{x}). \tag{3.12}$$

Similarly,

$$\frac{d}{d\mathrm{y}_i}(\mathbf{x} \diamond \mathbf{y}) \approx sign(\mathrm{x}_i) \tag{3.13}$$

for all $i = 1, 2, \ldots, d$.

3.1.2 Multiplication-free Implementation of EF-operator

One of the main idea behind the ef-operator is to compute a transformation without using multiplications. Proposition 3.1.2 gives us a way to compute it effectively with existing matrix multiplications. On the other hand, original definition 3.1.1 can be implemented without multiplication. Pseudo-code for this naive implementation which does not make any assumption on the representation of numbers, is given in Algorithm 3.1. A more efficient version for single precision floating numbers represented with IEEE 754 floating-point standard given in Algorithm 3.2

Implementation of ef-operator without multiplication requires more instruction compared to hardware which supplies floating-point multiplication instruction due to calculation of conditionals and extra addition instruction. However, avoidance of floating-point multiplication might improve energy consumption, since floating-point multiplication consumes relatively more energy than floating-point addition in most processors [28], [9]. The improvement in energy consumption is useful especially in mobile computing environments, where energy is a limited resource. The improvement in energy consumption depends on the hardware, as the difference between energy consumption of floating-point addition instruction and floating-point multiplication instruction varies between different hardwares [9]. If the difference between energy consumptions of these two instructions is large enough to compensate the additional instructions, then usage of ef-operator improves the energy consumption.

Algorithm 3.1 Naive multiplication-free computation of ef-operator between two d-dimensional vectors

```
1:  procedure EF(x, y, d)
2:      out ← 0
3:      for i = 1, 2, ..., d do
4:          if x_i = 0 or y_i = 0 then
5:              continue;
6:          end if
7:          if x_i > 0 then
8:              out ← out + y_i
9:          else
10:             out ← out − y_i
11:         end if
12:         if y_i > 0 then
13:             out ← out + x_i
14:         else
15:             out ← out − x_i
16:         end if
17:     end for
18:     return out
19: end procedure
```

Theoretically, expected energy consumption of Algorithm 3.2 based implementation and theoretical expected energy consumption of classic vector multiplication can be computed as follows: We assume that the expected energy consumptions of floating-point multiplication is P_{fm}, floating-point addition is P_{fa}, binary and is P_{ba}, binary or is P_{bo}, binary xor is P_{bx} and branching is P_b, the energy consumption of lines 4-6 in Algorithm 3.2 is P_c. We, also assume that probability of an entry being non-zero in the $\mathbf{x}$ vector is p_x and probability of an entry being non-zero in the $\mathbf{y}$ is p_y. We exclude the extra power consumption caused by the for loop as it is shared by both implementation. The expected power consumption of classic vector multiplication is

$$d(P_{fm} + P_{fa}), \tag{3.14}$$

Algorithm 3.2 Multiplication-free computation of ef-operator between two d-dimensional vectors for single precision floating-point numbers

1: **procedure** EF($\mathbf{x}$, $\mathbf{y}$, d)
2: $out \leftarrow 0$
3: **for** $i = 1, 2, \ldots, d$ **do**
4: **if** $\mathrm{x}_i = 0$ or $\mathrm{y}_i = 0$ **then**
5: continue;
6: **end if**
7: $s \leftarrow (\mathrm{x}_i \& 0\mathrm{x}80000000) \oplus (\mathrm{y}_i \& 0\mathrm{x}80000000)$
8: $out \leftarrow out + s|((\mathrm{x}_i \& 0\mathrm{x}7fffffff) + (\mathrm{y}_i \& 0\mathrm{x}7fffffff))$
9: **end for**
10: **return** out
11: **end procedure**

as d many floating-point multiplications required between d entries and d many floating-point additions are required to combine these multiplications. The expected power consumption of ef-operator is

$$d(P_c + p_x p_y(P_{bx} + 4P_{ba} + P_{bo} + 2P_{fa})), \tag{3.15}$$

as lines 7 and 8 executed only if both x_i and y_i are non-zero which happens with probability $p_x p_y$. Hence, ef-operator improves energy consumption for hardwares where equation (3.15) is less than (3.14). More efficient implementations of ef-operator can be suggested to save energy. Usually, one of the vectors is a weight vector. Consequently, usage of sparse weights can improve energy consumption of ef-operator further.

3.1.3 Properties of EF-operator

In order to gain insights about the power and weaknesses of ef-operator, one needs to study various properties, such as commutativity and distributivity. It is also worth to compare ef-operator with regular vector multiplication and inner product operators. These comparisons will enable us to replace regular vector multiplication and inner

products by ef-operators, when we define a neuron. Let us start by remarking that while vector multiplication is an inner product [39], the ef-operator fails to satisfy the inner product properties. In other words, ef-operator does not satisfy the first two requirements of the inner product which are,

$$(\mathbf{x}+\mathbf{y})\diamond\mathbf{z} = \mathbf{x}\diamond\mathbf{z}+\mathbf{y}\diamond\mathbf{z}, \tag{3.16}$$

$$(\mathrm{c}\mathbf{x})\diamond\mathbf{y} = \mathrm{c}(\mathbf{x}\diamond\mathbf{y}), \tag{3.17}$$

for any $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z} \in \mathbb{R}^d$ and for any $\mathrm{c} \in \mathbb{R}$. Let us, consider the simple case $d = 1$. Equation (3.16) implies that

$$sign(\mathbf{x}+\mathbf{y})\mathbf{z} = sign(\mathbf{x})\mathbf{z} + sign(\mathbf{y})\mathbf{z}, \tag{3.18}$$

which can hold if either $\mathbf{z} = 0$ or $\mathbf{x} = -\mathbf{y}$ is satisfied. Similarly, equation (3.17) implies that

$$\mathrm{c}sign(\mathbf{x})\mathbf{y} = sign(\mathrm{c}\mathbf{x})\mathbf{y}, \tag{3.19}$$

which can hold if either $\mathrm{c} = 0$ or $|\mathrm{c}| = 1$ is satisfied. However, ef operator satisfies similar identity,

$$(\mathrm{c}\mathbf{x})\diamond(\mathrm{c}\mathbf{y}) = \mathrm{c}(\mathbf{x}\diamond\mathbf{y}). \tag{3.20}$$

On the other hand, last two requirements are satisfied,

$$\mathbf{x}\diamond\mathbf{y} = \overline{\mathbf{y}\diamond\mathbf{x}}, \tag{3.21}$$

$$\mathbf{x}\diamond\mathbf{x} > 0 \text{ if } \mathbf{x} \neq \mathbf{0}, \tag{3.22}$$

where $\bar{\cdot}$ is complex conjugate. Equation (3.21) is equivalent to ef operator being commutative as both $\mathbf{x}$ and $\mathbf{y}$ are real vectors and equation (3.22) is equivalent to

$$\mathbf{x}\diamond\mathbf{x} = 2\left\|\mathbf{x}\right\|_1 > 0 \text{ if } \mathbf{x} \neq \mathbf{0}. \tag{3.23}$$

3.2 EF Neuron and EF Neural Networks

This thesis proposes a feed-forward neural network based on ef-operator, called ef-operator based neural network (EFNN). Proposed architecture is similar to previously

suggested versions in [5] and [4]. Network suggested on [5] simply replaces matrix-vector multiplication at each layer of classic feed-forward neural network with ef-operator. Hence, each hidden layer computes the equation (3.24),

$$f(\mathbf{W} \diamond \boldsymbol{h} + \mathbf{b}), \tag{3.24}$$

instead of the equation (3.25) which is computed by standard feed forward neural networks,

$$f(\mathbf{W}^T \boldsymbol{h} + \mathbf{b}), \tag{3.25}$$

where $\mathbf{W}$ is weights of the layer, $\mathbf{b}$ is the bias of the layer, f represents the activation function and $\boldsymbol{h}$ is the output of previous layer if the hidden layer is not the first one, otherwise $\boldsymbol{h}$ is just the input vector. In this study, the function classes which are representable by the networks is not investigated.

Artificial neural networks suggested in [4], called additive neural networks, employ the same structure for each hidden layer like the network suggested on [5]. These network introduce a new parameter for each hidden layer and replace the output layer with a linear layer. Hidden layers of these networks compute equation (3.26) as follows;

$$f(\mathbf{a} \odot (\mathbf{W} \diamond \boldsymbol{h}) + \mathbf{b}), \tag{3.26}$$

where $\mathbf{a}$ is a scaling coefficient and $\odot$ is element-wise multiplication. These networks can approximate any Lebesgue integrable function arbitrarily well, given enough neurons if specific activation functions are used. [4] shows this property in case of identity and rectified linear unit (ReLU).

EFNN is a simplified version of the additive neural networks which was improvement over [5] with provable representation capabilities. This new artificial neural network architecture eliminates the necessity of matrix multiplications at all layers except for the first hidden layer and the output layer. Consequently, number of multiplication operation required to compute EFNNs does not depend on the number of hidden layers while number of multiplication operation required to compute additive neural networks depends on the number of hidden layers. Furthermore, EFNNs preserves representation capacities of additive neural networks. Loosely speaking, EFNNs em-

ploy linear layer as the first hidden layer and output layer, and other hidden layers computes equation (3.24).

Definition 3.2.1. *An l-layered EFNN with layer sizes $d_0,\ d_1,\ \ldots,\ d_l$ is an ANN whose layers computes $\mathbf{h}_i$ which will be recursively defined at the equation 3.27.*

$$\mathbf{h}_i(\boldsymbol{x},\ \mathbf{u},\ f) := \begin{cases} f(\mathbf{W}_{\mathbf{u},i}^T \boldsymbol{x} + \mathbf{b}_{\mathbf{u},i}) & \text{if } i = 1, \\ f(\mathbf{W}_{\mathbf{u},i} \diamond \mathbf{h}_{i-1}(\boldsymbol{x},\ \mathbf{u},\ f) + \mathbf{b}_{\mathbf{u},i}) & \text{if } i < l, \\ \mathbf{W}_{\mathbf{u},i}^T \mathbf{h}_{i-1}(\boldsymbol{x},\ \mathbf{u},\ f) + \mathbf{b}_{\mathbf{u},i} & \text{if } i = l, \end{cases} \tag{3.27}$$

where $\boldsymbol{x} \in \mathbb{R}^{d_0}$ is the input vector, f represents the activation function and $\mathbf{u}$ is the parameters defined as

$$\mathbf{u} := (\mathbf{W}_{\mathbf{u},1},\ \mathbf{b}_{\mathbf{u},1},\ \ldots,\ \mathbf{W}_{\mathbf{u},l},\ \mathbf{b}_{\mathbf{u},l}) \in \mathcal{R}, \tag{3.28}$$

where $\mathcal{R}$ is parameter space defined as

$$\bigtimes_{i=1}^{l} \left(\mathbb{R}^{d_{i-1} \times d_i} \times \mathbb{R}^{d_i} \right), \tag{3.29}$$

where $\times$ is cartesian-product.

A visual representation of this architecture can be seen at figure 3.2.

The main idea behind ef-operator is to avoid multiplications as much as possible. Usage of any activation function requiring complex mathematical functions defeats this purpose. Consequently, focus will be on activation functions which does not require multiplication or any other complex mathematical functions throughout this thesis. Specifically, rectified linear unit, ReLU, and identity function, I, will be considered most of the time.

What does an EFNN represent is a crucial question which needs to be answered. We will discuss the properties of EFNN formally in the next section. In this section, we suffice to provide, Figure 3.3 a visualization of the functions computable with EFNNs over 2-dimensional input vectors. In order to display figures with uniform coloring, output of each network is scaled to $[-1,\ 1]$ range with a simple linear transformation.

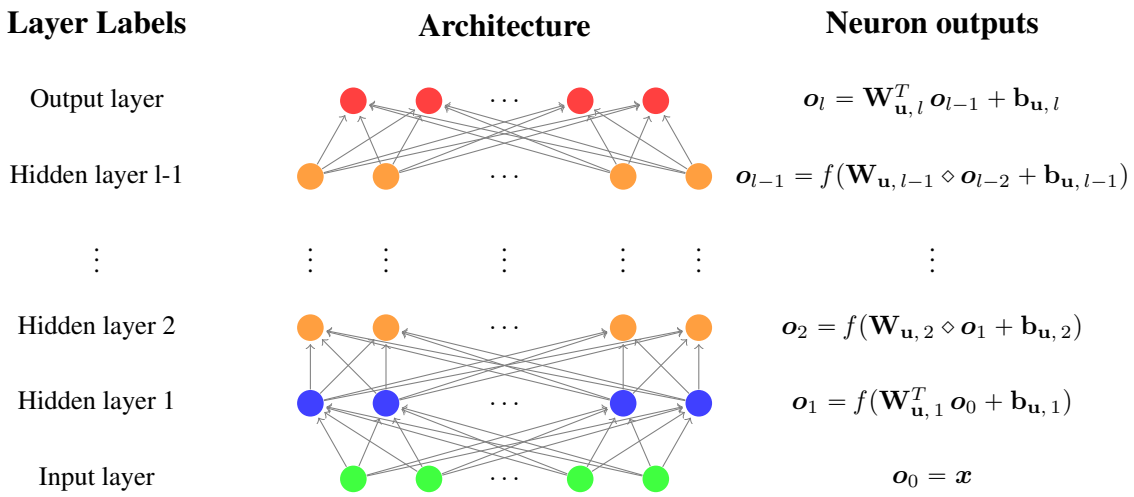


Figure 3.2: Visual representation ef-operator based neural networks (EFNN). Note that, input and output layers perform standard vector multiplication, whereas hidden layers perform ef-operator.

These networks are randomly generated and they are not trained for approximating any functions. Although, one can produce much more complex outputs with training, these networks are selected to visualize sample functions which are computable by EFNNs. A simple observation from these graphs is that when the number of neurons in hidden layers is increased, the mappings become increasingly non-linear even if very simple identity function is used as the activation function. However, this inherent non-linearity of ef-operator comes with a price. Any non-linear function computed by EFNNs are non-differentiable. Non-differentiable nature of computing functions may restrict the usage areas of EFNNs when contiunity or differentiability is required.

3.3 Representation Capabilities of EF Neural Networks

In this section, we analyze the representation capabilities of EFNNs with respect to the activation function, which is restricted to identity and ReLU. This section starts with necessary definitions required for the analysis. Then, we prove the major theorem of this thesis which show that EFNNs are capable of computing functions which are dense in Lebesgue integrable function space, when activation function is identity.

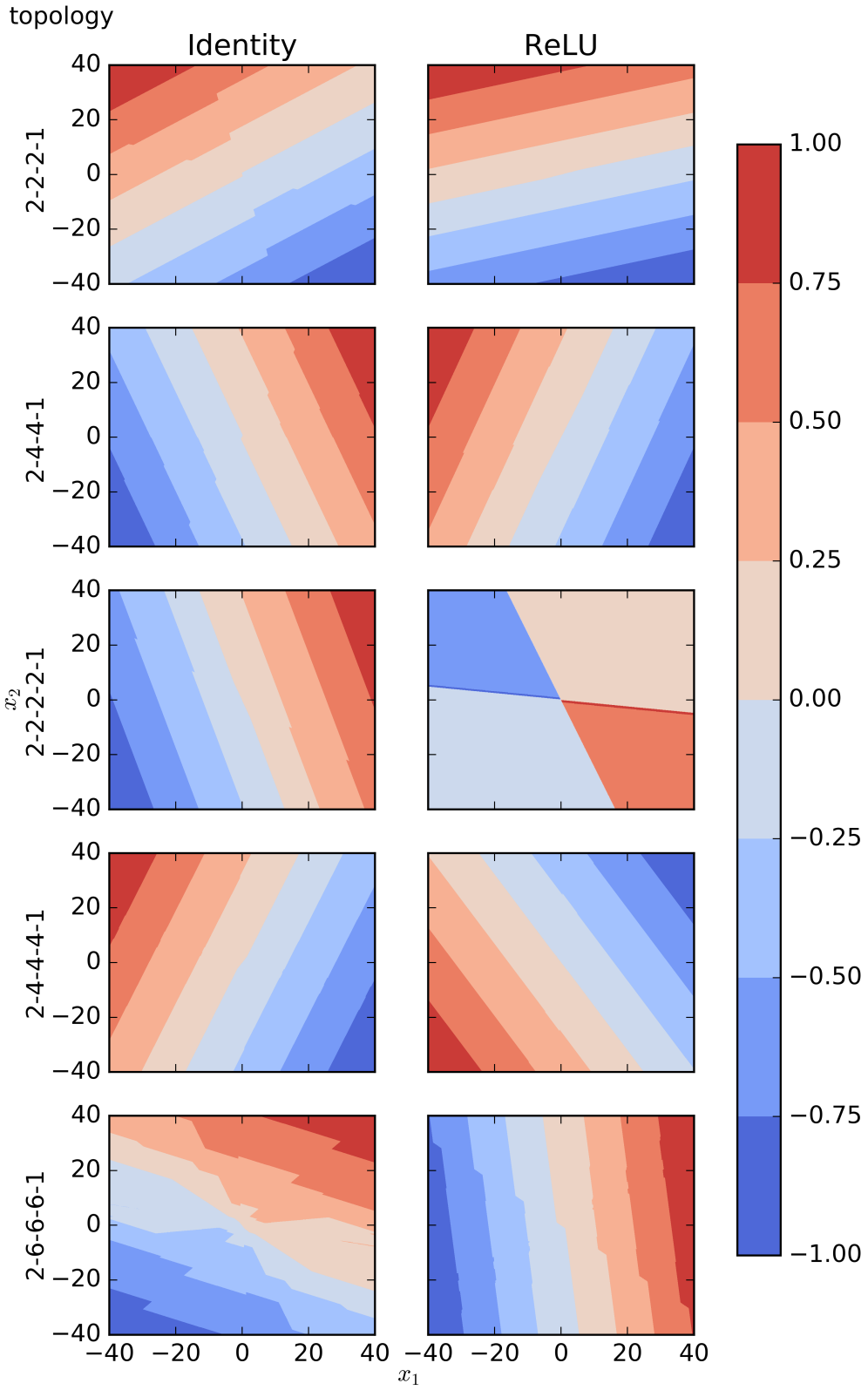


Figure 3.3: 2-D mappings computed by randomly generated EFNNs using activation functions identity and ReLU with different network topologies. Topologies given in left column shows the number of neurons used on each layer, including input neurons.

Theorem 3.3.1. *EFNNs can approximate any bounded measurable function arbitrarily well, given enough neurons if activation function is either identity or ReLU.*

Lastly, we investigate the representation power of EFNNs when activation function is ReLU. This theorem will be proven with the help of the propositions, given below. Identity and ReLU activation functions will be handled separately.

Definition 3.3.2. *Consider an EFNN mapping d_0-dimensional input vectors to 1-dimensional values. Let $\mathcal{G}_I$ be the set of functions computed by this EFNN with identity activation function and $\mathcal{G}_R$ be the set of functions computed by this EFNN with ReLU activation function.*

Proposition 3.3.3. *$\mathcal{G}_I$ is dense in $L^1(I_{d_0})$, metric space of Lebesgue integrable functions with L1 norm over d_0-dimensional unit cube.*

This proposition will be proved using Theorem 2.5.2 which shows that $\mathcal{G}_\sigma$ is dense in $L^1(()I_{d_0})$ for any bounded, measurable sigmoidal function, σ. Firstly, a set of functions satsifying Theorem 2.5.2 will be defined. Afterwards, it will be shown that this set is a subset of $\mathcal{G}_I$.

Let σ' be a function of the following form,

$$\sigma'(x) = \frac{sign(x) + 1}{2}, \tag{3.30}$$

and $\mathcal{G}_{\sigma'}$ be the set of functions of the following form,

$$\mathcal{G}_{\sigma'} = \Big\{ \sum_{i=1}^{n} \alpha_i \sigma'(\mathbf{w}_i^T \boldsymbol{x} + b_i) | \mathrm{w}_i \in \mathbb{R}^{d_0},\ \alpha_i, b_i \in \mathbb{R},\ n \in \mathbb{Z}^+ \Big\}. \tag{3.31}$$

Lemma 3.3.4. *$\mathcal{G}_{\sigma'}$, given in equation* (3.31)*, is dense in $L^1(I_{d_0})$.*

Proof. If σ' is a sigmoidal function, then by Theorem 2.5.2 $\mathcal{G}_{\sigma'}$ is dense in $L^1(I_{d_0})$.

$$\sigma'(x) = \frac{sign(x) + 1}{2} = \frac{-1 + 1}{2} = 0 \forall \mathrm{x} < 0, \tag{3.32}$$

and

$$\sigma'(x) = \frac{sign(x) + 1}{2} = \frac{+1 + 1}{2} = 1 \forall x > 0. \tag{3.33}$$

Equations (3.32) and (3.33) implies that

$$\begin{aligned} \lim_{x\to\infty} \sigma'(x) =& 1, \\ \lim_{x\to-\infty} \sigma'(x) =& 0. \end{aligned} \tag{3.34}$$

Equation (3.34) implies that σ' is a sigmoidal function and consequently $\mathcal{G}_{\sigma'}$ is dense in $L^1(I_{d_0})$.

□

Proof of Proposition 3.3.3. If $\mathcal{G}_{\sigma'}$ is subset of $\mathcal{G}_I$, then $\mathcal{G}_I$ is dense in $L^1(I_{d_0})$ as one of its subset is dense in $L^1(I_{d_0})$ by claim 3.3.4. Hence, this proof will focus on showing that $\mathcal{G}_{\sigma'}$ is a subset of $\mathcal{G}_I$.

An arbitrary member of $\mathcal{G}_{\sigma'}$ is of the form,

$$g(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \sigma'(\mathbf{w}_i^T \boldsymbol{x} + b_i) = \mathrm{b} + \sum_{i=1}^{n} \alpha_i' sign(\mathbf{w}_i^T \boldsymbol{x} + b_i), \tag{3.35}$$

where $\boldsymbol{x}$ is d_0-dimensional vector. $\mathcal{G}_{\sigma'}$ is a subset of $\mathcal{G}_I$ if $g(\boldsymbol{x}) \in \mathcal{G}_I$ or equivalently, there exists an EFNN with identity activation function such that its output is $g(\boldsymbol{x})$. Constructing such a network and verifying its output is $g(\boldsymbol{x})$ is enough to show that $\mathcal{G}_{\sigma'} \subset \mathcal{G}_I$.

Although EFNNs can be constructed in multiple hidden layers without any restrictions on the number of hidden layers besides the burden of computational complexity, constructing an EFNN with 4 layers suffice to prove our proposition. One such network, $\mathcal{N}$, consists of 4 layers each having n, $2n$, $n\,1$ many neurons, respectively. Let $\mathbf{u}$ be the parameters of $\mathcal{N}$, then $\mathbf{u}$ can be computed from g as follows,

$$\mathbf{W}_{\mathbf{u},1} = \Big[\mathbf{W}1 \quad \mathbf{W}2 \quad \dots \quad \mathbf{W}n\Big]_{d_0 \times n}, \tag{3.36}$$

$$\mathbf{b}_{\mathbf{u},1} = \Big[\mathbf{b}1 \quad \mathbf{b}2 \quad \dots \quad \mathbf{b}n\Big]_{n \times 1}, \tag{3.37}$$

$$\mathbf{W}_{\mathbf{u},2} = \Big[\mathbf{e}_1 \quad \mathbf{e}_2 \quad \dots \quad \mathbf{e}_n \quad 2\mathbf{e}_1 \quad 2\mathbf{e}_2 \quad \dots \quad 2\mathbf{e}_n\Big]_{n \times 2n}, \tag{3.38}$$

$$\mathbf{b}_{\mathbf{u},2} = \mathbf{0}_{2n \times 1}, \tag{3.39}$$

$$\mathbf{W}_{\mathbf{u},3} = \Big[\mathbf{e}_{n+1} - \mathbf{e}_1 \quad \mathbf{e}_{n+2} - \mathbf{e}_2 \quad \dots \quad \mathbf{e}_{2n} - \mathbf{e}_n\Big]_{2n \times n}, \tag{3.40}$$

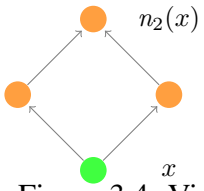
$$\mathbf{b}_{\mathbf{u},3} = \mathbf{0}_{n \times 1}, \tag{3.41}$$

$$\mathbf{W}_{\mathbf{u},4} = \Big[\alpha'_1 \quad \alpha'_2 \quad \dots \quad \alpha'_n\Big]^T_{n \times 1}, \tag{3.42}$$

$$\mathbf{b}_{\mathbf{u},4} = \Big[\mathrm{b}\Big]_{1 \times 1}, \tag{3.43}$$

where $\mathbf{e}_i$ is the i^{th} standard basis element for each i.

This construction is based on a crucial observation, that is the $sign$ function can be computed using combination of neurons with EF-operators. In fact, only three neurons in two layers are enough to compute $sign$ function, where one of them combines the output of the other two. This structure can be seen in figure 3.4 which will be called *sign computing unit* (SCU).



$$\mathbf{W}2' = \Big[-1 \quad 1\Big]^T_{1\times 2}, \ \mathbf{b}2' = \Big[0\Big]_{1\times 1}$$

$$\mathbf{W}1' = \Big[1 \quad 2\Big]_{1\times 2}, \ \mathbf{b}1' = \Big[0 \quad 0\Big]^T_{1\times 2}$$

Figure 3.4: Visual representation of a sign computing unit.

Let the weight of the first layer of sign computing unit be $\mathbf{W}1'$, the bias of the first layer of SCU be $\mathbf{b}1'$ and the output of the first layer of SCU be $n_1(x)$. Let the weight of the second layer of SCU be $\mathbf{W}2'$, the bias of the second layer of SCU be $\mathbf{b}2'$ and the output of the second layer of SCU be $n_1(x)$. If these weights and biases selected according to equation (3.44),

$$\begin{aligned} \mathbf{W}'_1 &= \Big[1 \quad 2\Big]_{1\times 2}, \quad \mathbf{b}'_1 = \Big[0 \quad 0\Big]^T_{1\times 2}, \\ \mathbf{W}'_2 &= \Big[-1 \quad 1\Big]^T_{1\times 2}, \quad \mathbf{b}'_2 = \Big[0\Big]_{1\times 1}, \end{aligned} \tag{3.44}$$

then outputs of these neurons becomes,

$$\begin{aligned} n_1(x) =& \mathbf{W}_1' \diamond \begin{bmatrix} x \end{bmatrix} + \mathbf{b}_1' \\ =& \begin{bmatrix} sign(x) + x \\ 2sign(x) + x \end{bmatrix}, \end{aligned} \tag{3.45}$$

$$\begin{aligned} n_2(x) =& \mathbf{W}_2' \diamond n_1(x) + \mathbf{b}_2' \\ =& - sign(x + sign(x)) + sign(x + 2sign(x)) \\ & - (x + sign(x)) + (x + 2sign(x)) \\ =& - sign(x) + sign(x) + sign(x) \\ =& sign(x), \end{aligned} \tag{3.46}$$

usign the fact that

$$sign(x) = sign(sign(x)), \tag{3.47}$$

and consequently

$$sign(x) = sign(x + sign(x)) = sign(x + 2sign(x)). \tag{3.48}$$

The first layer of $\mathcal{N}$ computes

$$\mathbf{h}_1(\mathbf{x},\, \mathbf{u},\, I) = \begin{bmatrix} \mathbf{w}_1^T \mathbf{x} + b_1 \\ \mathbf{w}_2^T \mathbf{x} + b_2 \\ \vdots \\ \mathbf{w}_n^T \mathbf{x} + b_n \end{bmatrix}. \tag{3.49}$$

The second and third layer contains n SCU in parallel. These SCUs are composed of i^{th} neuron of the third layer with the i^{th} and $(n+i)^{\text{th}}$ neurons of the second layer constructs for each $i = 1,\, 2,\, \ldots\, n$. The SCU whose output is i^{th} neuron of the third layer, takes the output of i^{th} neuron of the first layer as input. Therefore, the third layer of $\mathcal{N}$ computes

$$\mathbf{h}_3(\boldsymbol{x},\, \mathbf{u},\, I) = \begin{bmatrix} sign(\mathbf{W}1^T \boldsymbol{x} + \mathbf{b}1) \\ sign(\mathbf{W}2^T \boldsymbol{x} + \mathbf{b}2) \\ \vdots \\ sign(\mathbf{W}n^T \boldsymbol{x} + \mathbf{b}n) \end{bmatrix}. \tag{3.50}$$

The fourth layer of $\mathcal{N}$ computes

$$\mathbf{h}_4(\boldsymbol{x},\, \mathbf{u},\, I) = \mathrm{b} + \sum_{i=1}^{n} \alpha_i' sign(\mathbf{W}i^T\boldsymbol{x} + \mathbf{b}i), \tag{3.51}$$

which is exactly $g(\mathbf{x})$. Hence $\mathcal{G}_{\sigma'}$ is a subset of $\mathcal{G}_I$ and consequently $\mathcal{G}_I$ is dense $L^1(I_{d_0})$. □

Remark. The proof of Proposition 3.3.3 is based on the construction of 4 layered EFNNs. Deeper networks can also be used instead of the 4 layer network. If number of layers is even, then extension can be done using sign computing units. If number of layers is odd, then we need an additional observation. This observation is for sufficiently small $\epsilon > 0$, ($\epsilon\mathbf{I} \diamond \mathbf{x}$ is an approximation of $\mathbf{x}$ as,

$$(\epsilon\mathbf{I}) \diamond \mathbf{x} = \begin{bmatrix} \mathrm{x}_1 + sign(\mathrm{x}_1)\epsilon \\ \mathrm{x}_2 + sign(\mathrm{x}_2)\epsilon \\ \vdots \\ \mathrm{x}_d + sign(\mathrm{x}_d)\epsilon \end{bmatrix}_{d\times 1} \approxeq \mathbf{x}, \tag{3.52}$$

, where $\mathbf{I}$ is identity matrix. Hence, networks whose number of layers are odd, can be constructed using networks having even number of layers and approximation of $sign(\mathbf{x})$ usign above observation. However, this construction is not applicable to 3 layered EFNNs and we do not know their representation capacities. A final remak is that the proof uses explicit construction, which may lead readers to think that training is not necessary for EFNNs using this explicit construction. However, the construction relies on parameters of already trained standard ANN and consequently cannot be constructed without training.

Proposition 3.3.5. *$\mathcal{G}_I$ is a subset of $\mathcal{G}_R$.*

Lemma 3.3.6. *If*

$$\mathbf{X} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}, \tag{3.53}$$

and

$$\mathbf{y} = \begin{bmatrix} \mathbf{e} \\ \mathbf{f} \end{bmatrix}, \tag{3.54}$$

then

$$\mathbf{X} \diamond \mathbf{y} = \begin{bmatrix} \mathbf{A} \diamond \mathbf{e} + \mathbf{C} \diamond \mathbf{f} \\ \mathbf{B} \diamond \mathbf{e} + \mathbf{D} \diamond \mathbf{f} \end{bmatrix} \tag{3.55}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ *are matrices of size* $a \times c$, $a \times d$, $b \times d$, $b \times d$, *respectively and* $\mathbf{e}$ *and* $\mathbf{v}$ *are vectors of size* c *and* d, *respectively.*

Proof. Recall that,

$$\mathbf{X} \diamond \mathbf{y} = \mathbf{sign}(\mathbf{X})^T \mathbf{y} + \mathbf{X}^T \mathbf{sign}(\mathbf{y}) \tag{3.56}$$

by equation (3.10). This equation is equivalent to

$$\mathbf{X} \diamond \mathbf{y} = \begin{bmatrix} \mathbf{sign}(\mathbf{A})^T \mathbf{e} + \mathbf{sign}(\mathbf{C})^T \mathbf{f} + \mathbf{A}^T \mathbf{sign}(\mathbf{e}) + \mathbf{C}^T \mathbf{sign}(\mathbf{f}) \\ \mathbf{sign}(\mathbf{B})^T \mathbf{e} + \mathbf{sign}(\mathbf{D})^T \mathbf{f} + \mathbf{B}^T \mathbf{sign}(\mathbf{e}) + \mathbf{D}^T \mathbf{sign}(\mathbf{f}) \end{bmatrix} \tag{3.57}$$

by equations (3.53) and (3.54). Finally, equation (3.57) becomes

$$\mathbf{X} \diamond \mathbf{y} = \begin{bmatrix} \mathbf{A} \diamond \mathbf{e} + \mathbf{C} \diamond \mathbf{f} \\ \mathbf{B} \diamond \mathbf{e} + \mathbf{D} \diamond \mathbf{f} \end{bmatrix} \tag{3.58}$$

by the following equivalences;

$$\mathbf{A} \diamond \mathbf{e} = \mathbf{sign}(\mathbf{A})^T \mathbf{e} + \mathbf{A}^T \mathbf{sign}(\mathbf{e}), \tag{3.59}$$

$$\mathbf{B} \diamond \mathbf{e} = \mathbf{sign}(\mathbf{B})^T \mathbf{e} + \mathbf{B}^T \mathbf{sign}(\mathbf{e}), \tag{3.60}$$

$$\mathbf{C} \diamond \mathbf{f} = \mathbf{sign}(\mathbf{C})^T \mathbf{f} + \mathbf{C}^T \mathbf{sign}(\mathbf{f}), \tag{3.61}$$

$$\mathbf{D} \diamond \mathbf{f} = \mathbf{sign}(\mathbf{D})^T \mathbf{f} + \mathbf{D}^T \mathbf{sign}(\mathbf{f}). \tag{3.62}$$

□

Lemma 3.3.7. *The ef-operator between two vectors can be defined in therms of ef-operator between the vector and ReLU function of the other vector;*

$$\mathbf{x} \diamond \mathbf{y} = \mathbf{x} \diamond \mathit{ReLU}(\mathbf{y}) + (-\mathbf{x}) \diamond \mathit{ReLU}(-\mathbf{y}) \tag{3.63}$$

for any $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^d$.

Proof. The ef-operator can be written in the following form,

$$\mathbf{x} \diamond \mathbf{y} = \sum_{i=1}^{d} \mathrm{x}_i \diamond \mathrm{y}_i. \tag{3.64}$$

This lemma can be proved by showing that

$$\mathrm{x}_i \diamond \mathrm{y}_i = \mathrm{x}_i \diamond \mathrm{ReLU}(\mathrm{y}_i) + (-\mathrm{x}_i) \diamond \mathrm{ReLU}(-\mathrm{y}_i), \tag{3.65}$$

for each $i = 1, 2, \ldots, n$, using the equation (3.64) with the help of following two observations,

Observation 1:

$$\begin{aligned} \mathrm{x}_i \diamond 0 =& sign(\mathrm{x}_i)0 + \mathrm{x}_i sign(0) \\ =& 0. \end{aligned} \tag{3.66}$$

Observation 2:

$$\begin{aligned} (-\mathrm{x}_i) \diamond (-\mathrm{y}_i) =& sign(-\mathrm{x}_i)(-\mathrm{y}_i) + (-\mathrm{x}_i) sign(-\mathrm{y}_i) \\ =& sign(\mathrm{x}_i)\mathrm{y}_i + \mathrm{x}_i sign(-\mathrm{y}_i) \\ =& \mathrm{x}_i \diamond \mathrm{y}_i. \end{aligned} \tag{3.67}$$

If $\mathrm{y}_i \leqslant 0$, then the equation (3.65) becomes

$$\mathrm{x}_i \diamond 0 + (-\mathrm{x}_i) \diamond (-\mathrm{y}_i) = \mathrm{x}_i \diamond \mathrm{y}_i. \tag{3.68}$$

Otherwise, the equation (3.65) becomes

$$\mathrm{x}_i \diamond \mathrm{y}_i + (-\mathrm{x}_i) \diamond 0 = \mathrm{x}_i \diamond \mathrm{y}_i. \tag{3.69}$$

Hence, equation (3.63) holds. □

Proof of Proposition 3.3.5. The set of functions computed by EFNNs with identity activation function, $\mathcal{G}_I$, is a subset of the set of functions computed by EFNNs with ReLU activation function, $\mathcal{G}_R$, if for any network, $\mathcal{N}$, with identity activation function whose output is g, there exists a network, $\mathcal{N}'$, with ReLU activation function whose output is also g.

Let the number of hidden layers of $\mathcal{N}$ is l and number of neurons at i^{th} layer is d_i for $i = 1, 2, \ldots, l$, then $\mathcal{N}'$ can be constructed with l hidden layers, each containing ${d_i}'$ neurons for $i = 1, 2, \ldots, l$, where

$${d_i}' = \begin{cases} 2d_i & \text{if } i < l, \\ d_i & \text{if } i = l. \end{cases} \tag{3.70}$$

Let input dimension be d_0, $\mathbf{u}'$ be the parameters of $\mathcal{N}'$ and $\mathbf{u}$ be the parameters of $\mathcal{N}$. $\mathbf{u}'$ can be computed from $\mathbf{u}$ at the first and last layers, as follows,

$$\mathbf{W}_{\mathbf{u}',1} = \begin{bmatrix} \mathbf{W}_{\mathbf{u},1} & -\mathbf{W}_{\mathbf{u},1} \end{bmatrix}_{d_0 \times d_1'}, \tag{3.71}$$

$$\mathbf{b}_{\mathbf{u}',1} = \begin{bmatrix} \mathbf{b}_{\mathbf{u},1} & -\mathbf{b}_{\mathbf{u},1} \end{bmatrix}_{d_1' \times 1}, \tag{3.72}$$

$$\mathbf{W}_{\mathbf{u}',l} = \begin{bmatrix} \mathbf{W}_{\mathbf{u},l} \\ -\mathbf{W}_{\mathbf{u},l} \end{bmatrix}_{d_{l-1}' \times d_l'}, \tag{3.73}$$

$$\mathbf{b}_{\mathbf{u}',l} = \mathbf{b}_{\mathbf{u},l}, \tag{3.74}$$

for the hidden layers, $i = 2, \ldots, l-1$

$$\mathbf{W}_{\mathbf{u}',i} = \begin{bmatrix} \mathbf{W}_{\mathbf{u},i} & -\mathbf{W}_{\mathbf{u},i}, \\ -\mathbf{W}_{\mathbf{u},i} & \mathbf{W}_{\mathbf{u},i} \end{bmatrix}_{d_{i-1}' \times d_i'}, \tag{3.75}$$

$$\mathbf{b}_{\mathbf{u}',i} = \begin{bmatrix} \mathbf{b}_{\mathbf{u},i} & -\mathbf{b}_{\mathbf{u},i} \end{bmatrix}_{d_i' \times 1}. \tag{3.76}$$

This construction is done, so that

$$\mathbf{h}_i(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}) = \begin{bmatrix} \text{ReLU}(\mathbf{h}_i(\boldsymbol{x},\, \mathbf{u},\, I)) \\ \text{ReLU}(-\mathbf{h}_i(\boldsymbol{x},\, \mathbf{u},\, I)) \end{bmatrix}_{d_i' \times 1}, \tag{3.77}$$

for $i = 1,\, 2,\, \ldots,\, l-1$ and

$$\mathbf{h}_l(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}) = \mathbf{h}_l(\boldsymbol{x},\, \mathbf{u},\, I) = g(\boldsymbol{x}), \tag{3.78}$$

where ReLU is rectified linear unit and I is identity function. Equations (3.77) and (3.78) can be verified as follow,

The output of the first layer is

$$\begin{aligned} \mathbf{h}_1(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}) =& \text{ReLU}(\mathbf{W}_{\mathbf{u}',1}^T \boldsymbol{x} + \mathbf{b}_{\mathbf{u}',1}) \\ =& \begin{bmatrix} \text{ReLU}(\mathbf{W}_{\mathbf{u},1}^T \boldsymbol{x} + \mathbf{b}_{\mathbf{u},1}) \\ \text{ReLU}(-\mathbf{W}_{\mathbf{u},1}^T \boldsymbol{x} - \mathbf{b}_{\mathbf{u},1}) \end{bmatrix} \\ =& \begin{bmatrix} \text{ReLU}(\mathbf{h}_1(\boldsymbol{x},\, \mathbf{u},\, I)) \\ \text{ReLU}(-\mathbf{h}_1(\boldsymbol{x},\, \mathbf{u},\, I)) \end{bmatrix}. \end{aligned} \tag{3.79}$$

Hidden layers except the last one requires the calculation of

$$\mathbf{W}_{\mathbf{u}',i} \diamond \mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}), \tag{3.80}$$

which is equivalent to

$$\begin{bmatrix} \mathbf{W}_{\mathbf{u},i} & -\mathbf{W}_{\mathbf{u},i}, \\ -\mathbf{W}_{\mathbf{u},i} & \mathbf{W}_{\mathbf{u},i} \end{bmatrix} \diamond \begin{bmatrix} \text{ReLU}(\mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I)) \\ \text{ReLU}(-\mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I)) \end{bmatrix}, \tag{3.81}$$

assuming

$$\mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}) = \begin{bmatrix} \text{ReLU}(\mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I)) \\ \text{ReLU}(-\mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I)) \end{bmatrix}, \tag{3.82}$$

for $i = 2,\, \ldots,\, l-1$. Considering the Lemma 3.3.6, equation (3.81) becomes

$$\begin{bmatrix} \mathbf{W}_{\mathbf{u},i} \diamond \text{ReLU}(\mathbf{h}_{i-1}) + (-\mathbf{W}_{\mathbf{u},i}) \diamond \text{ReLU}(-\mathbf{h}_{i-1}) \\ (-\mathbf{W}_{\mathbf{u},i}) \diamond \text{ReLU}(\mathbf{h}_{i-1}) + \mathbf{W}_{\mathbf{u},i} \diamond \text{ReLU}(-\mathbf{h}_{i-1}) \end{bmatrix}, \tag{3.83}$$

where for the sake of simplicity we use $\mathbf{h}_{i-1} = \mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I)$. This equation becomes

$$\begin{bmatrix} \mathbf{sign}(\mathbf{W}_{\mathbf{u},i})^T \mathbf{h}_{i-1} + \mathbf{W}_{\mathbf{u},i}^T \mathbf{sign}(\mathbf{h_{i-1}}) \\ -\mathbf{sign}(\mathbf{W}_{\mathbf{u},i})^T \mathbf{h}_{i-1} - \mathbf{W}_{\mathbf{u},i}^T \mathbf{sign}(\mathbf{h_{i-1}}) \end{bmatrix}, \tag{3.84}$$

or equivalently, using Lemma 3.3.7,

$$\begin{bmatrix} \mathbf{W}_{\mathbf{u},i} \diamond \mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I) \\ -\mathbf{W}_{\mathbf{u},i} \diamond \mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I) \end{bmatrix}. \tag{3.85}$$

Therefore,

$$\begin{aligned} \mathbf{h}_i(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}) =& \text{ReLU}(\mathbf{W}_{\mathbf{u}',i} \diamond \mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}) + \mathbf{b}_{\mathbf{u}',i}) \\ =& \text{ReLU}\left(\begin{bmatrix} \mathbf{W}_{\mathbf{u},i} \diamond \mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I) \\ -\mathbf{W}_{\mathbf{u},i} \diamond \mathbf{h}_{i-1}(\boldsymbol{x},\, \mathbf{u},\, I) \end{bmatrix} + \begin{bmatrix} \mathbf{b}_{\mathbf{u},i} \\ -\mathbf{b}_{\mathbf{u},i} \end{bmatrix} \right) \\ =& \begin{bmatrix} \text{ReLU}(\mathbf{h}_i(\boldsymbol{x},\, \mathbf{u},\, I)) \\ \text{ReLU}(-\mathbf{h}_i(\boldsymbol{x},\, \mathbf{u},\, I)) \end{bmatrix}. \end{aligned} \tag{3.86}$$

Finally, the output of the last layer is

$$\begin{aligned} \mathbf{h}_l(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}) =& \mathbf{W}_{\mathbf{u}',l}^T \mathbf{h}_{l-1}(\boldsymbol{x},\, \mathbf{u}',\, \text{ReLU}) + \mathbf{b}_{\mathbf{u}',l} \\ =& \mathbf{W}_{\mathbf{u},l}^T \text{ReLU}(-\boldsymbol{o}) + (-\mathbf{W}_{\mathbf{u},l})^T \text{ReLU}(-\boldsymbol{o}) + \mathbf{b}_{\mathbf{u},l} \\ =& \mathbf{W}_{\mathbf{u},l}^T \boldsymbol{o} + \mathbf{b}_{\mathbf{u},l} \\ =& \mathbf{h}_l(\boldsymbol{x},\, \mathbf{u},\, I), \end{aligned} \tag{3.87}$$

where $\boldsymbol{o}$ is $\mathbf{h}_{l-1}(\boldsymbol{x},\ \mathbf{u},\ I)$, as desired. □

Proof of Theorem 3.3.1. Proposition 3.3.5 shows that the set of functions computed by EFNNs with identity activation function, $\mathcal{G}_I$, is a subset of the set of functions computed by EFNNs with ReLU activation function, $\mathcal{G}_R$, and Proposition 3.3.3 shows that $\mathcal{G}_I$ is dense in $L^1(I_{d_0})$. Consequently $\mathcal{G}_R$ is dense in $L^1(d_0)$ as one of its subsets is dense in $L^1(d_0)$. Density of these sets implies that for any function $g \in L^1(I_{d_0})$ and $\epsilon > 0$, there is a function $G \in \mathcal{G}_I$ which is computed by $\mathcal{N}_I$ and $\mathcal{N}_R$, EFNNs with activation functions identity and ReLU, respectively, such that

$$|G(\mathbf{x}) - g(\mathbf{x})| < \epsilon, \tag{3.88}$$

for all $\mathbf{x} \in I_{d_0}$. This implies that EFNNs can approximate any bounded measurable function arbitrarily well given sufficiently large number of neurons used if activation function is either identity or ReLU. The question of how much neuron is sufficient to approximate a function with given the degree of error, is not investigated. □

3.4 Behavior of EF Neural Networks from Parameters Perspective

Discontinuity of ef-operator causes EFNNs to compute very different functions depending on the parameters. This phenomenon can be understood through a simple example containing three neurons.

Example 3.4.1. *Let us assume that activation function is identity for the simplicity. Also, let us assume that* $\mathbf{u}_0 = [(w_1,\ b_1),\ (w_2,\ b_2),\ (w_3,\ b_3)]$ *is the parameters of this network. The functions computed by this network given at below:*

$$\mathbf{h}_3(x,\ \mathbf{u}_0,\ I) = \begin{cases} -w_3w_1x + w_3w_2sign(w_1x + b_1) + w_3(b_2 - b_1) + b_3 & \textit{if } w_2 < 0, \\ w_3b_2 + b_3 & \textit{if } w_2 = 0, \\ w_3w_1x + w_3w_2sign(w_1x + b_1) + w_3(b_2 + b_1) + b_3 & \textit{otherwise}. \end{cases} \tag{3.89}$$

An important observation is that

$$\lim_{w_2 \to 0^-} \mathbf{h}_3(x,\, \mathbf{u}_0,\, I) = -\, w_3 w_1 x + w_3(b_2 - b_1) + b_3, \tag{3.90}$$

$$\lim_{w_2 \to 0^+} \mathbf{h}_3(x,\, \mathbf{u}_0,\, I) = w_3 w_1 x + w_3(b_2 + b_1) + b_3. \tag{3.91}$$

Any neighborhood of $\mathbf{u}_0$ *when* w_2 *is zero, will contain parameters giving rise to very different functions, two linear functions with slopes* $-w_3 w_1$ *and* $w_3 w_1$*, unless*

$$w_3(w_1 x + b_1), \tag{3.92}$$

is 0. Left and right limits are different for this particular example, this means that if the desired function lies in one of the limits, then most of the optimization techniques may fail to converge unless they take this information into account, especially if w_2 *changes sign during optimization as this change will cause significant changes on derivatives of other parameters.*

Let us now inspect how EFNNs behave on different sets of parameters with respect to finite set of sample vectors. Firstly, an equivalence relation which relates parameters causing EFNNs to compute similar functions, will be defined. Afterwards, partitions induced by this relation will be inspected with respect to activation function.

The EFNNs which will be inspected throughout this section assumed to have l hidden layers with i^{th} layer containing d_i many neurons for all $i = 1,\, 2,\, \ldots,\, l$ and inputs are d_0-dimensional vectors. Also, $\mathcal{X}$ will denote the set of finitely many d_0-dimensional real vectors which represents the training samples.

Definition 3.4.2. *Let* $\mathbf{u}$ *and* $\mathbf{v}$ *be elements of* $\mathcal{R}$*. They are related by the relation* $\sim_{\mathcal{X},f}$ *if and only if*

$$\mathbf{sign}(\mathbf{W}_{\mathbf{u},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v},i}), \tag{3.93}$$

for all $i = 2,\, \ldots,\, l-1$ *and*

$$\mathbf{sign}(\mathbf{h}_j(\boldsymbol{x},\, \mathbf{u},\, f)) = \mathbf{sign}(\mathbf{h}_j(\boldsymbol{x},\, \mathbf{v},\, f)), \tag{3.94}$$

for all $j = 1,\, 2,\, \ldots,\, l-2$ *and for any* $\boldsymbol{x} \in \mathcal{X}$*.* $\sim$ *will be used instead of the* $\sim_{\mathcal{X},f}$ *for the brevity at the remainder of this section.*

Proposition 3.4.3. *The relation $\sim_{\mathcal{X},f}$ is an equivalence relation.*

Proof. Recall that, a relation is an equivalence relation if it is reflexive, symmetric and transitive [77]. Let us check the properties of $\sim$:

- **Reflexivity**: The relation is reflexive if $\mathbf{u} \sim \mathbf{u}$ for any $\mathbf{u} \in \mathcal{R}$ [77], which is trivially follow from the definition.

- **Symmetry**: The relation is symmetric if $\mathbf{u} \sim \mathbf{v}$ implies $\mathbf{v} \sim \mathbf{u}$ for any $\mathbf{u}$, $\mathbf{v} \in \mathcal{R}$ [77]. $\mathbf{u} \sim \mathbf{v}$ implies $\mathbf{v} \sim \mathbf{u}$ as

$$\mathbf{sign}(\mathbf{W}_{\mathbf{u},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v},i}), \tag{3.95}$$

  implies

$$\mathbf{sign}(\mathbf{W}_{\mathbf{v},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{u},i}), \tag{3.96}$$

  for all $i = 2, \ldots, l-1$, and

$$\mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{u}, f)) = \mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{v}, f)), \tag{3.97}$$

  implies

$$\mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{v}, f)) = \mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{u}, f)), \tag{3.98}$$

  for all $j = 1, 2, \ldots, l-2$ and for any $\boldsymbol{x} \in \mathcal{X}$.

- **Transitivity**: The relation is transitive if $\mathbf{u} \sim \mathbf{v}$ and $\mathbf{v} \sim \mathbf{v}'$ implies $\mathbf{u} \sim \mathbf{v}'$ for any $\mathbf{u}$, $\mathbf{v}$, $\mathbf{v}' \in \mathcal{R}$ [77]. $\mathbf{u} \sim \mathbf{v}$ and $\mathbf{v} \sim \mathbf{v}'$ implies $\mathbf{u} \sim \mathbf{v}'$ as

$$\mathbf{sign}(\mathbf{W}_{\mathbf{u},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v},i}), \tag{3.99}$$

  and

$$\mathbf{sign}(\mathbf{W}_{\mathbf{v},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v}',i}), \tag{3.100}$$

  implies

$$\mathbf{sign}(\mathbf{W}_{\mathbf{u},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v}',i}), \tag{3.101}$$

  for all $i = 2, \ldots, l-1$, and

$$\mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{u}, f)) = \mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{v}, f)), \tag{3.102}$$

and

$$\mathbf{sign}(\mathbf{h}_j(\boldsymbol{x},\, \mathbf{v},\, f)) = \mathbf{sign}(\mathbf{h}_j(\boldsymbol{x},\, \mathbf{v}',\, f)), \tag{3.103}$$

implies

$$\mathbf{sign}(\mathbf{h}_j(\boldsymbol{x},\, \mathbf{u},\, f)) = \mathbf{sign}(\mathbf{h}_j(\boldsymbol{x},\, \mathbf{v}',\, f)), \tag{3.104}$$

for all $j = 1,\, 2,\, \ldots,\, l-2$ and for any $\boldsymbol{x} \in \mathcal{X}$.

Hence, $\sim_{\mathcal{X},f}$ is an equivalence relation. □

Proposition 3.4.3 shows that $\sim$ is an equivalence relation. Consequently, it induces a partitioning scheme for $\mathcal{R}$ [77].

Definition 3.4.4. *Let $\mathcal{P}_{\mathcal{X},f,\mathbf{u}}$ be the set of points in $\mathcal{R}$ which are equivalent to $\mathbf{u}$ with respect to $\sim$ for each $\mathbf{u} \in \mathcal{R}$. The sets $\mathcal{P}_{\mathcal{X},f,\mathbf{u}}$ and $\mathcal{P}_{\mathcal{X},f,\mathbf{v}}$ for any $\mathbf{u},\, \mathbf{v} \in \mathcal{R}$ is either disjoint or same set as $\sim$ is an equivalence relation. Consequently, the family of sets*

$$\{\mathcal{P}_{\mathcal{X},f,\mathbf{u}} | \mathbf{u} \in \mathcal{R}\} \tag{3.105}$$

is a partition of $\mathcal{R}$. $\mathcal{P}_{\mathbf{u}}$ will be used instead of the $\mathcal{P}_{\mathcal{X},f,\mathbf{u}}$ for the brevity at the remainder of this section.

One important issue about these partitions are some of them may not be closed. Consequently, local infimum may not be attainable in some partitions. Furthermore, the mappings computed in neighbor partitions can be drastically different. Reader can refer to example 3.4.1 to see that the difference between mappings computed by neighbor partitions. Let us assume that there is no input vector, in other words $\mathcal{X} = \varnothing$ for the simplicity. Then, there are three partitions in example 3.4.1, which are $\mathbb{R} \times \mathbb{R}^+ \times \mathbb{R}$, $\mathbb{R} \times \{0\} \times \mathbb{R}$, and $\mathbb{R} \times \mathbb{R}^- \times \mathbb{R}$. First and third partitions are open and second partition is closed in this example. Furthermore, second partition is actually boundary of first and third partitions. Consequently, all three partitions are neighbor. Example 3.4.1 showed that there are points from both first and third partitions in neighborhood of points in the second partition, such that the difference between the mappings computed by these points are not bounded by a function of distance between these points.

Definition 3.4.5. *A function g will be called sign-wise linear if*

$$g(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) = \alpha g(\mathbf{x}) + (1-\alpha)g(\mathbf{y}) \tag{3.106}$$

for any $\alpha \in [0,\, 1]$ *when*

$$sign(g(\mathbf{x})) = sign(g(\mathbf{y})) \tag{3.107}$$

for any $\mathbf{x}$ *and* $\mathbf{y}$.

Proposition 3.4.6. *If g is a sign-wise linear function, then*

$$sign(g(\alpha\mathbf{x} + (1-\alpha)\mathbf{y})) = sign(g(\mathbf{x})), \tag{3.108}$$

for any $\alpha \in [0,\, 1]$*, when*

$$sign(g(\mathbf{x})) = sign(g(\mathbf{y})), \tag{3.109}$$

for any $\mathbf{x},\, \mathbf{y} \in \mathbb{R}^d$.

Lemma 3.4.7.

$$sign(\alpha \mathrm{x} + (1-\alpha)\mathrm{y}) = sign(\mathrm{x}), \tag{3.110}$$

if

$$sign(\mathrm{x}) = sign(\mathrm{y}), \tag{3.111}$$

for any $\mathrm{x},\, \mathrm{y} \in \mathbb{R}$ *and for all* $\alpha \in [0,\, 1]$.

Proof. For any $\mathrm{x},\, \mathrm{y} \in \mathbb{R}$

$$\alpha \mathrm{x} + (1-\alpha)\mathrm{y} \in [\min(\mathrm{x},\, \mathrm{y}),\, \max(\mathrm{x},\, \mathrm{y})], \tag{3.112}$$

and $sign(\mathrm{x}) = sign(\mathrm{y})$ implies one of the following three cases;

$$0 \leqslant \min(\mathrm{x},\, \mathrm{y}) \leqslant \alpha \mathrm{x} + (1-\alpha)\mathrm{y} \leqslant \max(\mathrm{x},\, \mathrm{y}) \qquad \text{if } \mathrm{x} > 0, \tag{3.113}$$

$$0 = \min(\mathrm{x},\, \mathrm{y}) \leqslant \alpha \mathrm{x} + (1-\alpha)\mathrm{y} \leqslant \max(\mathrm{x},\, \mathrm{y}) = 0 \qquad \text{if } \mathrm{x} = 0, \tag{3.114}$$

$$0 \geqslant \min(\mathrm{x},\, \mathrm{y}) \geqslant \alpha \mathrm{x} + (1-\alpha)\mathrm{y} \leqslant \max(\mathrm{x},\, \mathrm{y}) \qquad \text{if } \mathrm{x} < 0, \tag{3.115}$$

in any case $sign(\mathrm{x}) = sign(\alpha \mathrm{x} + (1-\alpha)\mathrm{y})$, as desired. □

Proof of Proposition 3.4.6. This is immediate result of Lemma 3.4.7. Replacing x with $g(\mathbf{x})$ and y with $g(\mathbf{y})$ at equations (3.110) and (3.111) proves this proposition. □

Proposition 3.4.8. *Partition $\mathcal{P}_{\mathbf{u}}$ is convex for each $\mathbf{u} \in \mathcal{R}$ if activation function f is sign-wise linear.*

Lemma 3.4.9. *If f is a sign-wise linear function, then*

$$\mathbf{h}_i(\boldsymbol{x},\, \alpha\mathbf{v} + (1-\alpha)\mathbf{v}',\, f) = \alpha\mathbf{h}_i(\boldsymbol{x},\, \mathbf{v},\, f) + (1-\alpha)\mathbf{h}_i(\boldsymbol{x},\, \mathbf{v}',\, f), \tag{3.116}$$

for any $\mathbf{u} \in \mathcal{R}$, for any $\mathbf{v},\, \mathbf{v}' \in \mathcal{P}_{\mathbf{u}}$, for any $\boldsymbol{x} \in \mathcal{X}$, for any $\alpha \in [0,\, 1]$, and for all $i = 1,\, 2,\, \ldots,\, l-1$ and if $i < l-1$, then

$$\mathbf{sign}(\mathbf{h}_i(\boldsymbol{x},\, \alpha\mathbf{v} + (1-\alpha)\mathbf{v}',\, f)) = \mathbf{sign}(\mathbf{h}_i(\boldsymbol{x},\, \mathbf{u},\, f)). \tag{3.117}$$

Proof. $\mathbf{v}_\alpha$ will denote

$$\mathbf{v}_\alpha = \alpha\mathbf{v} + (1-\alpha)\mathbf{v}', \tag{3.118}$$

throughout this proof. Let

$$\mathbf{h}_i(\boldsymbol{x},\, \mathbf{v}_\alpha,\, f) = f(\mathbf{n}_{i,\,\alpha}(\boldsymbol{x})), \tag{3.119}$$

for $i = 1,\, 2,\, \ldots,\, (l-1)$. If

$$\mathbf{n}_{i,\,\alpha}(\boldsymbol{x}) = \alpha\mathbf{n}_{i,\,0}(\boldsymbol{x}) + (1-\alpha)\mathbf{n}_{i,\,1}, \tag{3.120}$$

for $\boldsymbol{x} \in \mathcal{X}$, then equation (3.116) is satisfied for $i = 1,\, 2,\, \ldots,\, (l-1)$ as f is a sign-wise linear function. If equation (3.116) is satisfied, then equation (3.117) is, also, satisfied by Proposition 3.4.6 as

$$\mathbf{sign}(\mathbf{h}_i(\boldsymbol{x},\, \mathbf{u},\, f)) = \mathbf{sign}(\mathbf{h}_i(\boldsymbol{x},\, \mathbf{v},\, f)) \tag{3.121}$$

$$\mathbf{sign}(\mathbf{h}_i(\boldsymbol{x},\, \mathbf{v},\, f)) = \mathbf{sign}(\mathbf{h}_i(\boldsymbol{x},\, \mathbf{v}',\, f)) \tag{3.122}$$

as $\mathbf{v},\, \mathbf{v}' \in \mathcal{P}_{\mathbf{u}}$ implies $\mathbf{v} \sim \mathbf{u}$ and $\mathbf{v}' \sim \mathbf{u}$, and by transitivity $\mathbf{v} \sim \mathbf{v}'$ if $i < l-1$.

If $i = 1$, then

$$\begin{aligned}
\mathbf{n}_{1,\,\alpha}(\boldsymbol{x}) =& \mathbf{W}_{\mathbf{v}_\alpha,\,1}^T\boldsymbol{x} + \mathbf{b}_{\mathbf{v}_\alpha,\,1} \\
=& (\alpha\mathbf{W}_{\mathbf{v},\,1} + (1-\alpha)\mathbf{W}_{\mathbf{v},\,1})^T\boldsymbol{x} + (\alpha\mathbf{b}_{\mathbf{v},\,1} + (1-\alpha)\mathbf{b}_{\mathbf{v},\,1}) \\
=& \alpha\mathbf{n}_{1,\,0}(\boldsymbol{x}) + (1-\alpha)\mathbf{n}_{1,\,1}(\boldsymbol{x}),
\end{aligned} \tag{3.123}$$

as desired.

If $i < l$, then

$$\mathbf{n}_{i,\alpha}(\boldsymbol{x}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v}_\alpha,i})^T\mathbf{h}_{i-1}(\boldsymbol{x}, \mathbf{v}_\alpha, f) + \mathbf{W}^T_{\mathbf{v}_\alpha,i}\mathbf{sign}(\mathbf{h}_{i-1}(\boldsymbol{x}, \mathbf{v}_\alpha, f)) + \mathbf{b}_{\mathbf{v}_\alpha,i} \tag{3.124}$$

First term on the right side of the equation (3.124) can be written as,

$$\alpha\mathbf{sign}(\mathbf{W}_{V,i})^T\mathbf{h}_{i-1}(\boldsymbol{x}, \mathbf{v}, f) + (1-\alpha)\mathbf{sign}(\mathbf{W}_{V',i})^T\mathbf{h}_{i-1}(\boldsymbol{x}, \mathbf{v}', f), \tag{3.125}$$

assuming the claim shown for $i-1$ and using the fact that $\mathbf{u} \sim \mathbf{v} \sim \mathbf{v}'$ with Claim 3.4.7.

Second term on the right side of the equation (3.124) can be written as

$$\alpha\mathbf{W}^T_{V,i}\mathbf{sign}(\mathbf{h}_{i-1}(\boldsymbol{x}, \mathbf{v}, f)) + (1-\alpha)\mathbf{W}^T_{V',i}\mathbf{sign}(\mathbf{h}_{i-1}(\boldsymbol{x}, \mathbf{v}', f)), \tag{3.126}$$

assuming the claim shown for $i-1$ and using the equation (3.117) with Claim 3.4.7.

Third term on the right side of the equation (3.124) can be written as

$$\alpha\mathbf{b}_{\mathbf{v},i} + (1-\alpha)\mathbf{b}_{\mathbf{v}',i}. \tag{3.127}$$

Combining all three terms results

$$\mathbf{n}_{\alpha\, i} = \alpha\mathbf{n}_{0,i} + (1-\alpha)\mathbf{n}_{1,i}, \tag{3.128}$$

as desired.

□

Proof of Proposition 3.4.8. The partition $\mathcal{P}_\mathbf{u}$ is convex if

$$\mathbf{v}_\alpha = \alpha\mathbf{v} + (1-\alpha)\mathbf{v}' \in \mathcal{P}_\mathbf{u}, \tag{3.129}$$

for any $\mathbf{v}, \mathbf{v}' \in \mathcal{P}_\mathbf{u}$ and for any $\alpha \in [0, 1]$. $\mathbf{v}_\alpha \in \mathcal{P}_\mathbf{u}$ if $\mathbf{v}_\alpha \sim \mathbf{u}$ by definition or equivalently

$$\mathbf{sign}(\mathbf{W}_{\mathbf{v}_\alpha,i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{u},i}) \tag{3.130}$$

$$\mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{v}_\alpha, f)) = \mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{u}, f)) \tag{3.131}$$

for all $i = 2, \ldots, l-1$, $j = 1, 2, \ldots, l-2$, and for any $\boldsymbol{x} \in \mathcal{X}$.

$\mathbf{v}, \mathbf{v}' \in \mathcal{P}_{\mathbf{u}}$ implies

$$\mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{v}_\alpha, f)) = \mathbf{sign}(\mathbf{h}_j(\boldsymbol{x}, \mathbf{u}, f)) \tag{3.132}$$

for any $\boldsymbol{x} \in \mathcal{X}$ and for all $i = 1, 2, \ldots, l-2$ by Lemma 3.4.9 and it also implies $\mathbf{v} \sim \mathbf{u}$ and $\mathbf{v}' \sim \mathbf{u}$. By transitivity of $\sim$, $\mathbf{v} \sim \mathbf{v}'$ which implies

$$\mathbf{sign}(\mathbf{W}_{\mathbf{u},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v},i}), \tag{3.133}$$

$$\mathbf{sign}(\mathbf{W}_{\mathbf{v},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v}',i}), \tag{3.134}$$

for any $i = 2, \ldots, l-1$. Equation (3.134) implies

$$\mathbf{sign}(\mathbf{W}_{\mathbf{v},i}) = \mathbf{sign}(\mathbf{W}_{\mathbf{v}_\alpha,i}), \tag{3.135}$$

by Lemma 3.4.7 and when combined with equation (3.133) results equation (3.134). Therefore, $\mathbf{v}_\alpha \sim \mathbf{u}$ and consequently, $\mathcal{P}_{\mathbf{u}}$ is convex. □

3.5 Chapter Summary

This chapter starts with defining ef-operator and discussing some of its prominent properties, such as multiplication free-implementability similarities and dissimilarities to vector multiplication. Afterwards, a feed forward network architecture is proposed based on ef-operator. Remainder of the chapter analyzes the properties of proposed network. Analysis shows that 4 layered or deeper proposed networks have universal approximation capacity if activation function is selected as either identity or ReLU. Analysis also includes the dependency of EFNNs on the parameters with respect to some set of input vectors. An equivalence relation between parameters is defined with respect to input vectors and the activation function. Analysis of this relation showed that partitions induced by this relation splits the parameter space into convex sets if the activation function behaves linearly between two points whose sign under the activation function is same. Analysis also shows that, mappings computed by the last hidden layer of EFNNs are linear with respect to parameters in each partition at input vectors.

CHAPTER 4

MODIFIED BACKPROPAGATION ALGORITHM WITH LINE SEARCH FOR TRAINING EF NEURAL NETWORKS

Classic ANNs employ generally well-behaving activation functions, which have continuous partial derivatives on the whole space. However, this is not the case for EFNNs. In Section 3.4, we showed that parameter space of EFNNs can be splited into convex partitions where the resulting functions are similar to each other, while different partitions may contain different functions. Some of these partitions are not closed and consequently, local optimum may not be attainable in these partitions. Also, transition between different partitions may cause unbounded changes on partial derivatives causing the search algorithms to diverge from desired local optimum. This section introduces a modified version of backpropagation algorithm, called backpropagation with line search (BLS), which considers the properties of EFNNs mentioned in the previous chapter. In the following subsections, we first analyze the training of EFNNs to approximate linear functions with standard backpropagation algorithms. Next, we discuss the problem of non-attainable local optimum through this analysis. Afterwards, we propose a possible solution to this problem, namely the BLS algorithm. Finally, we analyze the convergence properties of BLS algorithm.

4.1 A case study: Learning Linear Functions

Linear functions are one of the simplest class of functions and usually can be represented by most of the machine learning algorithms. However, training an EFNN to represent a linear function with standard backpropagation algorithm is quite dif-

ficult. A simple $efnn$ containing 4 neurons with identity activation function trained to approximate a simple linear function $y = 2x$. 100 data points uniformly sampled from the interval $[-100,\ 100]$ as training samples. Training is done using adaptive moment estimation (ADAM) optimization algorithm without minibatches. Learning rate is taken as 1 during training trials. Suggested values used for other hyper parameters which are 0.9 for β_1, 0.999 for β_2 and 10^{-8} for ε. Objective function is taken as mean square error between predicted values and expected values. 10 training trials using ADAM optimizers and other hyper parameters are performed with 10 different sets of initial parameters sampled according to following normal distributions,

$$\mathcal{N}\Big(0,\ \frac{2}{n_{in} + n_{out}}\Big), \tag{4.1}$$

$$\mathcal{N}(0,\ 10^{-4}), \tag{4.2}$$

where n_{in} is the number of neurons in the previous layer, and n_{out} is the number of neurons in the current layer as suggested by Glorot et al. [33]. Weights are sampled independently from the distribution (4.1) and biases are sampled independently from the distribution (4.2). Each training trial is halted after 100,000 iterations unless algorithm terminated earlier. Results of these training trials can be seen in figure 4.1.

One important phenomenon to notice in figure 4.1 is that training procedure repeatedly minimizes the objective function by converging to global optimum, and diverging from this point when it gets too close. ADAM, optimization method is a robust method which is designed to be wide range of optimization problems. In the case of optimizing difference between linear function and approximation calculated by the EFNNs, ADAM, method generates a sequence of update points toward the global optimum. However, when the distance between the current parameters and global optimum becomes sufficiently small, next update value causes transition to another partition resulting unbounded increase in MSE. ADAM, method generates another sequence of update points toward the global optimum until it diverges again. Note that, even for linear functions, which are exceedingly simple functions, standard backpropagation algorithm cannot converge when training EFNNs. Considering the complex data sets, EFNNs may result in more complex problems, which cannot be solved without considering properties of EFNNs.

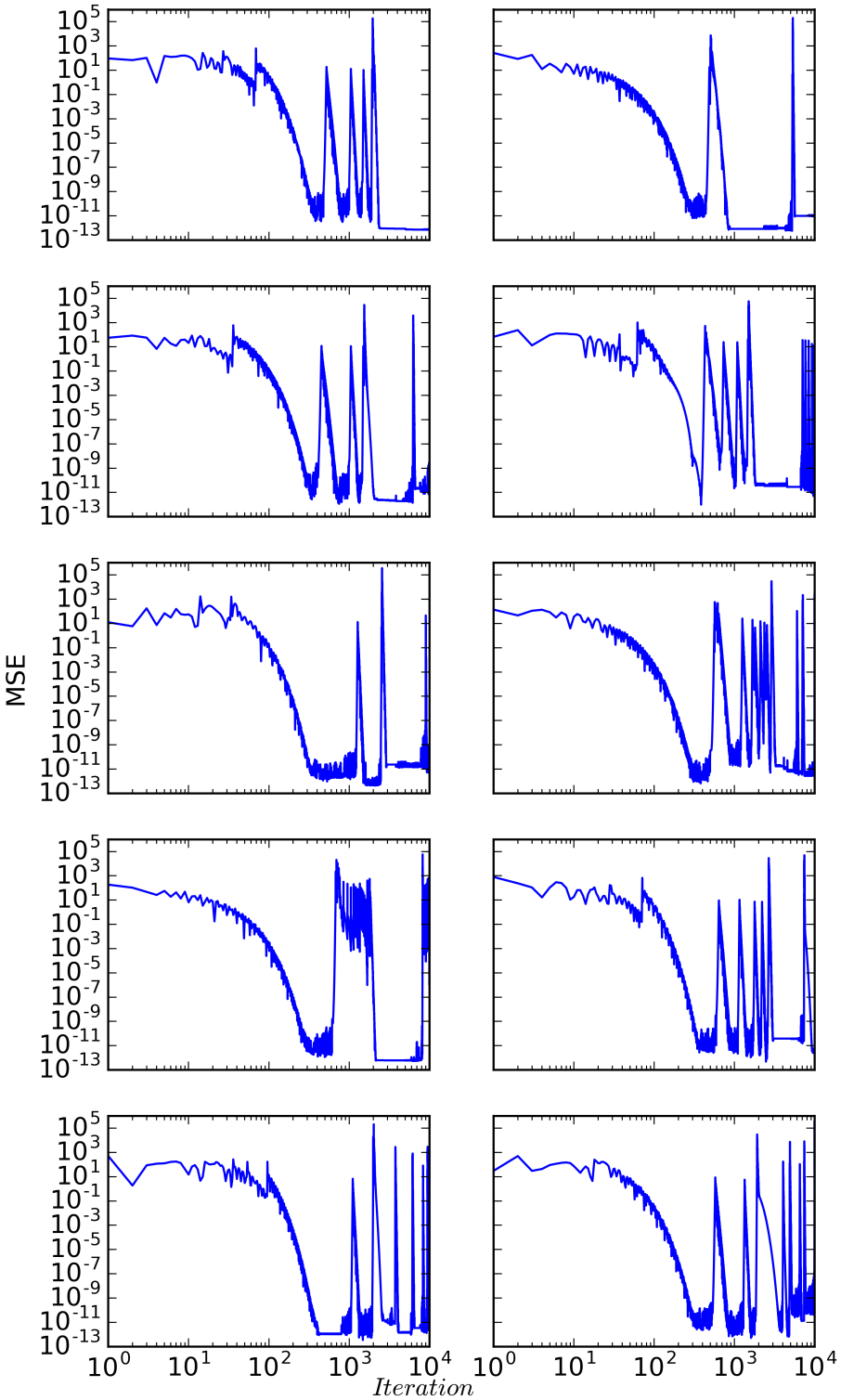


Figure 4.1: Change of mean squared error during training EFNN with standard back-propagation algorithm to approximate the linear function $y = 2x$.

4.2 Backpropagation with Line Search

Standard backpropagation algorithm given in algorithm 2.1, can be divided into three sub-procedure. i) Calculating objective function with a forward pass, ii) computing minimizers for parameters and iii) updating parameters. Proposed modification is to add a line search between minimizer, u_t' and current parameters u_t before updating them.

Let L be the objective function to minimize, M be an optimizer which returns minimizer vector for the objective function, and $\mathcal{X} \times \mathcal{Y}$ be the input vectors with their expected output. Assuming that training is done for a l-layered EFNN with starting point $\mathbf{u}_0$, proposed algorithm can be seen at 4.1. $\boldsymbol{\lambda} = \{\lambda_t\}_{t=1}^{\infty}$ on the algorithm is a sequence with $\lambda_t \geqslant 1$ for each $t = 1, 2, \ldots$, and

$$\lim_{T\to\infty} \prod_{t=1}^{T} \lambda_t, \tag{4.3}$$

exists and it is finite. The backpropagation with line search (BLS) algorithm usign these notations given in pseudo-code 4.1.

Suggested modification is based on the observation that partitions, $\mathcal{P}_{\mathbf{u}_t}$, are convex by proposition 3.4.8, which means that any point over a line segment between $\mathbf{u}_t$ and another point $\mathbf{u}_t'$ will reside in $\mathcal{P}_{\mathbf{u}_t}$, if $\mathbf{u}_t' \in \mathcal{P}_{\mathbf{u}_t}$. The main idea is, if the minimizer causes an increase on the objective function, then there is two possibilities. First possibility is that minimizer is still in the same partition, however magnitude of the update was too much that local optimum is missed. Second possibility is that minimizer resides in another partition and mappings computed in this partition are different than mappings computed in previous partition and the distance between these two sets of mappings is not bounded by the magnitude of the change of parameters. Reader can refer to example 3.4.1 for an example. Line search will find a point decreasing objective function in the first case, assuming local optimum was on the direction of the minimizer. However, the latter case is more problematic because of the following reasons: There are two possibilities, either line search can return to original partition which means the goal is still at the same local optimum, or line search terminate

Algorithm 4.1 Backpropagation with line search algorithm

procedure BLS($\mathcal{X}$, $\mathcal{Y}$, $\boldsymbol{\lambda}$, $\mathbf{u}_0$)

$t \leftarrow 0$

while not converged **do**

$\alpha \leftarrow 2$

$\mathrm{L}_t \leftarrow L(\mathbf{h}_l(\mathcal{X},\, \mathbf{u}_{t-1},\, f), \mathcal{Y})$

$t \leftarrow t + 1$

Compute minimizer, $\mathbf{u}'_{t-1} \leftarrow M(L,\, \mathbf{u}_{t-1})$

$\Delta\mathbf{u}_{t-1} \leftarrow \mathbf{u}'_{t-1} - \mathbf{u}_{t-1}$

repeat

$\alpha \leftarrow \alpha/2$

$L'_{t+1} \leftarrow L(\mathbf{h}_l(\mathcal{X},\, \mathbf{u}_{t-1} + \alpha\Delta\mathbf{u}_{t-1},\, f), \mathcal{Y})$

until $\mathrm{L}'_{t+1} \leqslant \lambda_t \mathrm{L}_t$ or $\alpha < \varepsilon$

if $\alpha < \varepsilon$ **then**

Break

else

$\mathbf{u}_t \leftarrow \mathbf{u}_{t-1} + \Delta\mathbf{u}_{t-1}$

end if

end while

return $\mathbf{u}_t$

end procedure

on another partition. Although, line search terminating on another partition does not mean that this partition have a lower local infimum. The new partition contains at least one parameter which is not worse at optimizing objective function than current parameter. However, this is also the case for standard backpropagation algorithm. In fact, there is no guarantee that new partition will contain at least one parameter which minimizes objective function better than current parameter. An additional important point is that BLS can create a converging sequence to the boundary of a partition if a local optimum is on the boundary, even if local infimum is not attainable.

A remark on the selection of $\boldsymbol{\lambda}$. Although, equation (4.3) restricts the selection of $\boldsymbol{\lambda}$, most sequences can be used as $\boldsymbol{\lambda}$ in practice, assuming training will be done for limited number of iterations. Any sequence of positive real numbers equal or greater than one with finitely many entries greater than one, satisfies equation (4.3). Small constant λ_t values will cause algorithm to exploit current partition to converge to a point inside the initial partition or another neighbor partition. Exponentially decaying sequences can be used to search different partitions at the start of training and restrict search to few partitions near the end of training. Similarly, sufficiently small constant values can be used to search different partitions without restricting search to current partition. Smallness of the constant value depends on the objective function. Change of partition requires change of sign on either some weight or some output. Any constant value limiting the change from positive to non-positive output due to increase on objective function can be treated as small. Small constant values will prevent from current partition to another partition where mappings differ greatly from target mapping. On the other hand, large constant values will result non-restricted search causing divergence problem similar to the standard backpropagation algorithm.

4.3 Computational Complexity of BLS

This algorithm computationally is not less efficient than standard backpropagation algorithm. Modification requires multiple forward pass and multiple parameter updates. However, number of forward passes is bounded with $\log_2(\varepsilon)$, which can be set according to machine precision and can be treated as constant. Furthermore, BLS does not require external early stopping criteria to catch the local minimum as algorithm terminates if there is no viable update on the direction of the minimizer. Next iteration will not produce a new minimizer and consequently algorithm will not continue to update parameters, assuming optimizer will return the same minimizer at the same point regardless of iterations.

4.4 Convergence of Backpropagation with Line Search Algorithm

Let us now, investigate the convergence properties of BLS, and show that it cannot diverge.

Theorem 4.4.1. *Backpropagation algorithm with proposed modification will create a converging sequence of parameters for EFNNs, regardless of the initial parameters,* $\mathbf{u}_0$*, quality of the minimizer,* M*, and* $\{\lambda_t\}_{t=1}^{\infty}$ *sequence, as long as*

$$\lim_{T\to\infty}\prod_{t=1}^{T}\lambda_t, \tag{4.4}$$

exists and it is finite.

Proof. Let

$$\mathbf{L} = \{\mathrm{L}_t = L(\mathbf{h}_l(\mathcal{X},\, \mathbf{u}_t,\, f))|t = 0,\, 1,\, \ldots\}, \tag{4.5}$$

be the sequence of outputs of objective function after each update.

$$\mathrm{L}_{t+1}{}' \leqslant \lambda_t \mathrm{L}_t, \tag{4.6}$$

requirement in the algorithm 4.1 implies

$$\mathrm{L}_t \leqslant \prod_{i=1}^{t}\lambda_i \mathrm{L}_0 \leqslant \lambda \mathrm{L}_0, \tag{4.7}$$

for $t = 1,\, 2,\, \ldots$, where

$$\lambda = \lim_{T\to\infty}\prod_{i=1}^{T}\lambda_i. \tag{4.8}$$

There are three cases to be considered. In the first case, there are finitely many t such that

$$\mathrm{L}_t \leqslant \mathrm{L}_{t+1}. \tag{4.9}$$

In the second case, there are finitely many t' such that

$$\mathrm{L}_{t'} \geqslant \mathrm{L}_{t'+1}. \tag{4.10}$$

In the third case, there are infinitely many t satisfying equation (4.9) and there are infinitely many t' satisfying equation (4.10).

The first case implies that there is a T such that

$$\mathrm{L}_t > \mathrm{L}_{t+1}, \tag{4.11}$$

for all $t > T$. This means that the sequence $\{\mathrm{L}_{T+1}, \mathrm{L}_{T+2}, \ldots\}$ is a monotonically increasing sequence bounded above with $\lambda \mathrm{L}_0$. Hence, this sequence converge by the monotone convergence theorem [76].

The second case implies that there is a T', such that;

$$\mathrm{L}_t < \mathrm{L}_{t+1}, \tag{4.12}$$

for all $t > T'$. This means that the sequence $\{\mathrm{L}_{T'+1}, \mathrm{L}_{T'+2}, \ldots\}$ is a monotonically decreasing sequence bounded below with 0. Hence, this sequence converge by the monotone convergence theorem [76].

The third case implies that there is no T'', such that the sequence $\{\lambda_t\}_{T''}{}^{\infty}$ is monotone, and consequently monotone convergence theorem cannot be used in this case. However, sequence is still bounded both from above, $\lambda \mathrm{L}_0$ and from below, 0. Therefore, there exists a subsequence, $\{\mathrm{L}_{a_1}, \mathrm{L}_{a_2}, \ldots\}$ of $\mathbf{L}$ which converges by Weirstrass-Bolzano theorem [76]. Let us say they converge to l. Showing for any $\epsilon > 0$, there exists T_ϵ, such that

$$|\mathrm{L}_t - \mathrm{l}| < \epsilon \tag{4.13}$$

for any $t > T_\epsilon$, is enough to show that $\mathbf{L}$ converges to l [76]. Equation (4.8) implies that

$$\begin{aligned} \log(\lambda) &= \log\left(\lim_{T\to\infty} \prod_{i=1}^{T} \lambda_i\right) \\ &= \lim_{T\to\infty} \sum_{i=1}^{T} \log(\lambda_i). \end{aligned} \tag{4.14}$$

Let

$$\mathrm{s}_t = \sum_{i=1}^{t} \log(\lambda_i), \tag{4.15}$$

for all t. Equation (4.14) implies that

$$\lim_{t\to\infty} \mathrm{s}_t = \log(\lambda), \tag{4.16}$$

and as a convergence sequence, this sequence is a Cauchy sequence [76]. Consequently, there exists $T_{0,\epsilon}$ such that

$$|\mathrm{s}_m - \mathrm{s}_n| < \log\left(1 + \frac{\epsilon}{2\mathrm{l} + \epsilon}\right) \tag{4.17}$$

for any $n,\ m > T_{0,\epsilon}$ or equivalently

$$\prod_{i=n+1}^{m} \lambda_i < \frac{\mathrm{l} + \epsilon}{\mathrm{l} + \epsilon/2}, \tag{4.18}$$

assuming $m > n$. Similarly, there exists $T_{1,\epsilon}$ such that

$$|\mathrm{s}_m - \mathrm{s}_n| < \log\left(1 + \frac{\epsilon}{2\mathrm{l} - 2\epsilon}\right) \tag{4.19}$$

for any $n,\ m > T_{0,\epsilon}$ given $\epsilon < \mathrm{l}$ or equivalently

$$\prod_{i=n+1}^{m} \lambda_i < \frac{\mathrm{l} - \epsilon/2}{\mathrm{l} - \epsilon}, \tag{4.20}$$

assuming $m > n$.

The sequence $\{\mathrm{L}_{a_1},\ \mathrm{L}_{a_2},\ \ldots\}$ converges to l implies that there exists $T_{2,\epsilon}$ such that

$$|\mathrm{L}_{a_t} - \mathrm{l}| < \epsilon/2, \tag{4.21}$$

for any $t > T_{1,\epsilon}$.

If

$$\epsilon' = \min(\epsilon,\ \mathrm{l}/2), \tag{4.22}$$

and

$$T = \max(T_{0,\epsilon'}, T_{1,\epsilon'}, T_{2,\epsilon'},\ a_1\}), \tag{4.23}$$

then, for any $t > T$, either there exists t' such that $a_{t'} = t$ or $a_{t'} < t < a_{t'+1}$. If $a_{t'} = t$, then, $|\mathrm{l}_t - \mathrm{l}| < \epsilon$ by equation (4.21), and otherwise

$$\mathrm{L}_t \leqslant \prod_{i=a_{t'}+1}^{t} \lambda_i \mathrm{L}_{a'_t} < \left(\frac{\mathrm{l} + \epsilon}{\mathrm{l} + \epsilon/2}\right)(\mathrm{l} + \epsilon/2) = \mathrm{l} + \epsilon, \tag{4.24}$$

$$\mathrm{L}_t \geqslant \frac{\mathrm{L}_{a_{t'+1}}}{\prod_{i=t+1}^{a_{t'+1}} \lambda_i} > \frac{\mathrm{l} - \epsilon/2}{\frac{\mathrm{l}-\epsilon/2}{\mathrm{l}-\epsilon}} = \mathrm{l} - \epsilon. \tag{4.25}$$

Therefore, $\mathbf{L}$ converges to l.

$\mathbf{L}$ converges at all three cases, and so it converges. □

Theorem 4.4.1 states that BLS will converge for any given set of initial parameters and for any minimizer. However, convergence does not guarantee a viable solution. The final value of objective function is strongly depends on the selection of both initial parameters and the minimizer. The algorithm can converge to non-infimum points depending on the choice of initial parameters and update values generated by the minimizer.

4.5 Chapter Summary

This chapter starts by analyzing the convergence properties for training EFNNs with the standard backpropagation algorithm. The analysis is exemplified through a simple problem of learning approximation of a linear function, $y = 2x$. A modification to backpropagation is proposed, which adds a line search step before updating parameters to prevent EFNNs from diverging. Next, the proposed modification is investigated for training EFNN and motivation behind the suggested modification discussed, Finally, convergence of training procedures with proposed modification is shown to be independent from selection of initial parameters and hyper-parameters of optimization technique used. Although, the convergence does not depend on selection of hyper-parameters and initial parameters, quality of end result depends on the selection of these parameters. Consequently, proposed modification does not eliminate the necessity of parameter search for converging to a better local optimum.

CHAPTER 5

EXPERIMENTS

This section experimentally compares the standard backpropagation algorithm and backpropagation with line search (BLS) in training EFNN. Experiments are done on XOR problem, several multi-class classification problems from UCI data sets [53], and on hand-written digit classification problem, MNIST [52]. We analyze the classification accuracies of trained networks with both algorithms using multiple hyper-parameters, initial parameters and different selections of $\boldsymbol{\lambda}$, limiting factor on the increase in objective function. Comparisons are done over a variety of network topologies, activation functions, and optimization methods for each data set.. The code for experiments is implemented using python with theano library [7]. 32-bit floating point numbers are used during experiments and ε is taken as 2^{-32} considering precision of 32-bit floating point numbers.

5.1 Learning Linear Functions Revisited

In Section 4.1, we inspect the problems of training an EFNN with standard backpropagation results. One important problem was that whenever objective function becomes sufficiently small, the next update on parameters causes jump on output of objective function. Same experiments are repeated using BLS algorithm. Shared parameters are not changed during these experiments. BLS related parameter λ_t is taken as

$$1 + \delta^t, \tag{5.1}$$

where $\delta = 1 - 10^{-8}$. Figure 5.1 shows comparison between training of EFNN with standard backpropagation algorithm and BLS algorithm.

Figure 5.1 shows that BLS algorithm can either achieve the same loss as standard backpropagation algorithm, or terminates early without training network properly. Choice of $\boldsymbol{\lambda}$ does not allow large increases on the loss function. This can be restrictive when training procedure requires sequence of points causing increase on the loss to converge. The sequence generated by the training procedure depends on hyper-parameters as well as initial parameters. Experiments show that usage of different initial parameters is enough to overcome this problem. Although, these phenomena can be observed commonly, proposed algorithm stops early in these cases, usually around 100^{th} iteration, allowing multiple initial points to be tried without losing much time on a "bad" initial parameter set. On the other hand, if BLS train the network properly, then optimum parameters usually found earlier than standard backpropagation, and the algorithm terminates training procedure automatically.

5.2 Learning XOR Problem by EFNN

XOR problem is relatively simple non-linear function approximation problem. It is one of the two binary Boolean function out of sixteen possible binary Boolean function whose output cannot be linearly separated. Although, it is a simple problem, it is complex enough not to be classified correctly by linear classifiers. Table 5.1 describes the XOR operator.

Table 5.1: Description of XOR operator.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

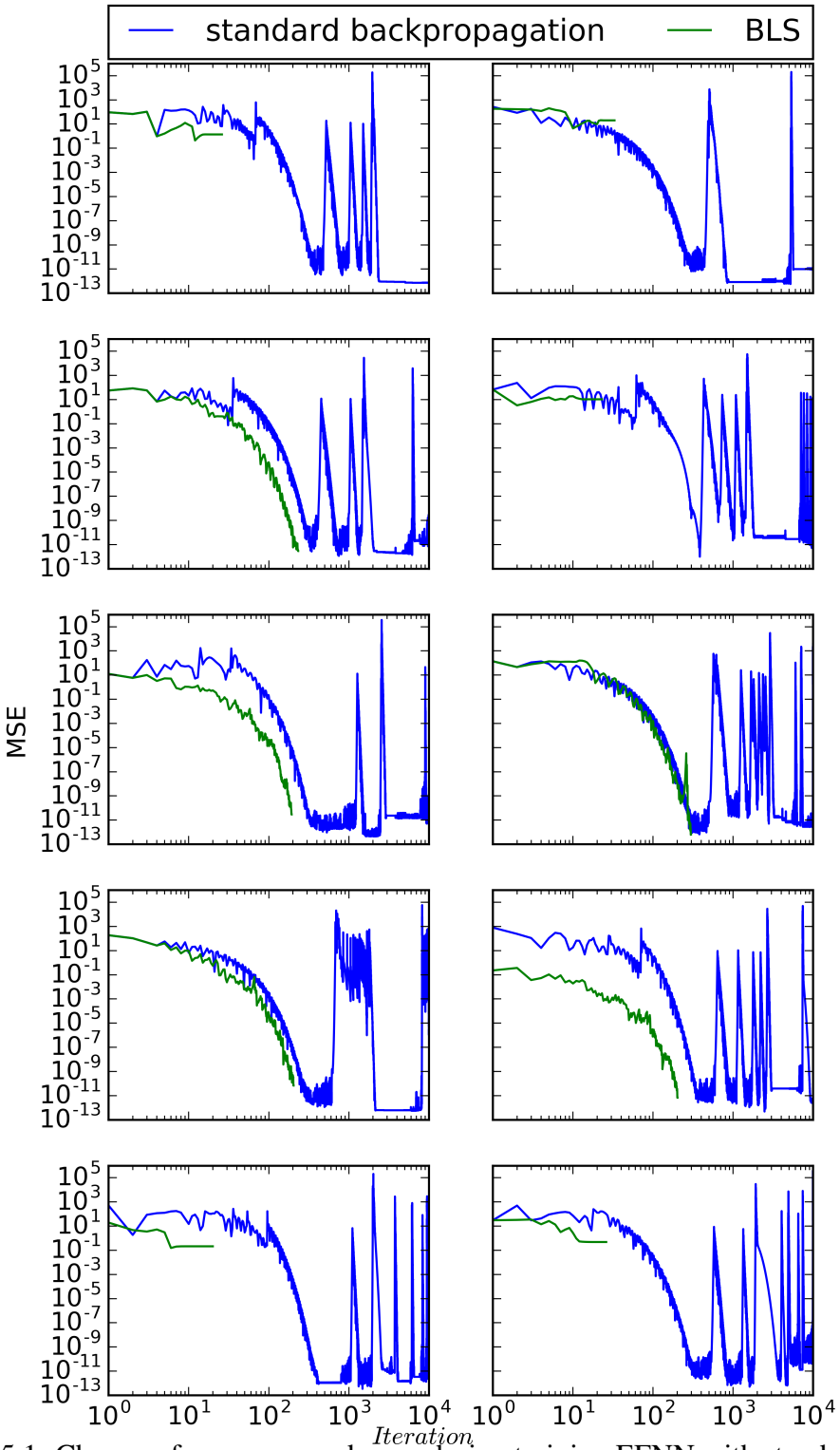


Figure 5.1: Change of mean squared error during training EFNN with standard back-propagation algorithm and BLS to learn the linear function $y = 2x$.

5.2.1 Experimental Design

Training of EFNNs is done using both standard backpropagation algorithm and the suggested backpropagation with line search (BLS) algorithm. Two different $\boldsymbol{\lambda}$s are used during the experiments. One $\boldsymbol{\lambda}$ is taken a constant value 1, in other words, BLS algorithm does not allow any increase on the loss function. Other $\boldsymbol{\lambda}$ is selected as following exponentially decaying sequence,

$$\lambda_t = 1 + \delta^{2^t}, \tag{5.2}$$

where δ is a constant in the interval $[0,\ 1)$. δ value taken as $(1 - 10^{-8})$ during experiments. Each version will be called with λ_t value when results reported in the figures and tables. Standard back propagation algorithm is a special case of BLS algorithm with $\lambda_t = \infty$. The selection of exponentially decaying sequence done according to following observation,

$$\frac{1 - \delta^{2^{T+1}}}{1 - \delta} = \prod_{t=0}^{T}(1 + \delta^{2^T}). \tag{5.3}$$

Experiments are performed by using four different network topologies and two different activation functions. Two different optimizers applied to minimize the objective function. These two optimizers are stochastic gradient descent (SGD) and adaptive moment estimation (ADAM). These two different activation functions are identity and ReLU functions. And, these four network topologies tested in the experiments are as follows,

- 2 hidden layers each of which containing 2 neurons,
- 2 hidden layers each of which containing 3 neurons,
- 3 hidden layers each of which containing 2 neurons,
- 3 hidden layers each of which containing 3 neurons,

These networks are trained with respect to mean square error at 4 points given in table 5.1 using each optimizer and activation function combination.

Each training experiment is repeated with multiple hyper-parameters including different initial parameters, learning rates, and batch sizes. 4 different learning rates are used during experiments. These learning rates are 1e-4, 1e-6, 1e-8, and 1e-10. 2 different batch sizes are used during experiments, which are 1 and 4. Each topology, activation function, optimizer, learning rate and batch size combination is used on training of EFNNs with different $\boldsymbol{\lambda}$ sequences starting from 25 different initial parameters sampled according to following distributions,

$$\mathcal{N}\Big(0,\ \frac{2}{n_{in} + n_{out}}\Big), \tag{5.4}$$

$$\mathcal{N}(0,\ 10^{-4}), \tag{5.5}$$

where n_{in} is number of neurons in the previous layer, and n_{out} is number of neurons in the current layer as suggested by by Glorot et al. [33]. Weights are sampled independently from the distribution (5.4) and biases are sampled independently from the distribution (5.5). Each training trial halted after 100,000 iterations unless algorithm terminated earlier. ADAM method specific parameters β_1, β_2 and ε taken as default values suggested by [48], which are 0.9, 0.999, 10^{-8}, respectively.

5.2.2 Performances of EFNN with Standard Backpropagation and BLS

Table 5.2 presents the least mean square error obtained by standard backpropagation and the suggested BLS method using different hyper-parameters and initial parameters for each network topology, activation function, and optimizer triplet. Figure 5.2 shows the change of MSE during the training of one particular triplet consisting ADAM optimizer, ReLU activation function and topology consisting of 3 hidden layers each of which containing 3 neurons for each backpropagation version. Each graph includes results of 20 trials selected according to least mean square error they achieved.

Figure 5.2 shows that while most of the trials with standard backpropagation achieves the small MSE, they continue to oscillate greatly even after reaching to optima allowing MSE to increase as much as $10^8\%$. However, there are trials among these 20 trials which cannot achieve acceptable MSE levels, even though each hyper-parameter set

Table 5.2: Least mean squared error achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λ, $\lambda_t = 1$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on XOR problem using ADAM and SGD optimizers with ReLU and identity activation functions for networks consisting 2 and 3 hidden layers which contain either 2 or 3 neurons.

Network	λ_t	SGD	ADAM
2 hidden layers	∞	$1.01e-09$	$0.00e+00$
2 neurons	1	$2.11e-02$	$3.02e-06$
Identity	$1+\delta^{2^t}$	$1.18e-02$	$3.02e-06$
2 hidden layers	∞	$4.00e-12$	$0.00e+00$
2 neurons	1	$7.21e-12$	$2.32e-14$
ReLU	$1+\delta^{2^t}$	$7.21e-12$	$1.34e-13$
2 hidden layers	∞	$7.29e-10$	$0.00e+00$
3 neurons	1	$2.75e-09$	$1.12e-03$
Identity	$1+\delta^{2^t}$	$2.75e-09$	$1.48e-05$
2 hidden layers	∞	$7.98e-12$	$0.00e+00$
3 neurons	1	$2.82e-11$	$9.15e-17$
ReLU	$1+\delta^{2^t}$	$2.82e-11$	$4.88e-18$
3 hidden layers	∞	$1.57e-08$	$0.00e+00$
2 neurons	1	$5.07e-03$	$4.00e-11$
Identity	$1+\delta^{2^t}$	$1.42e-09$	$3.63e-11$
3 hidden layers	∞	$2.72e-11$	$0.00e+00$
2 neurons	1	$1.22e-11$	$1.07e-13$
ReLU	$1+\delta^{2^t}$	$1.22e-11$	$8.39e-14$
3 hidden layers	∞	$5.73e-11$	$0.00e+00$
3 neurons	1	$1.38e-10$	$2.21e-12$
Identity	$1+\delta^{2^t}$	$1.38e-10$	$2.21e-12$
3 hidden layers	∞	$1.30e-11$	$0.00e+00$
3 neurons	1	$5.06e-12$	$3.91e-16$
ReLU	$1+\delta^{2^t}$	$4.77e-12$	$2.44e-15$

tried using 25 different initial parameter set. On the other hand, although, both versions of BLS algorithm can achieve acceptable levels of losses only on 3 trials out of 20 trials, they terminated automatically instead of allowing loss to diverge. BLS seems to perform worse than standard backpropagation. However, effects of choice

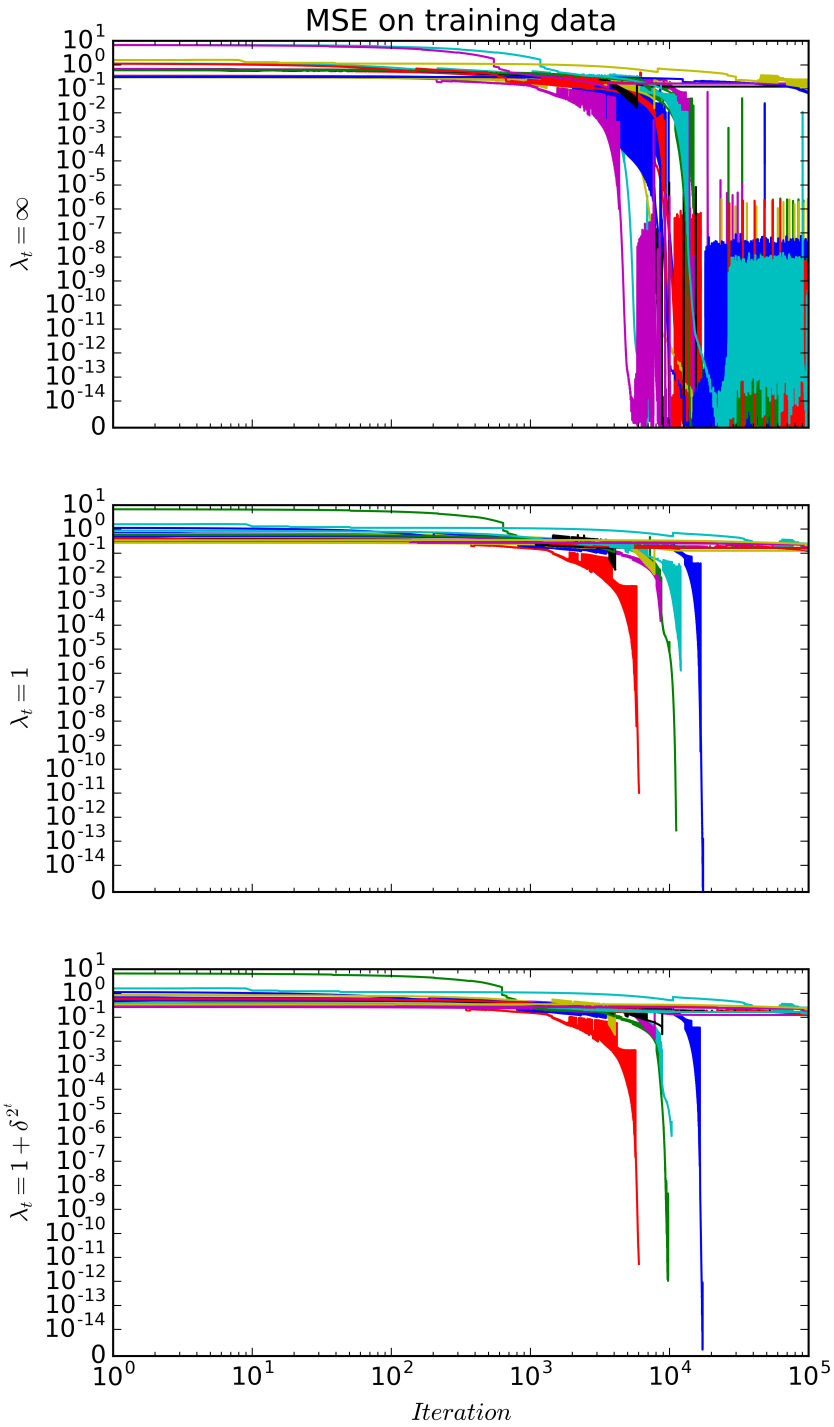


Figure 5.2: The change of mean squared error with respect to iteration for trials achieving minimal mean squared error when standard backpropagation, $\lambda_t = \infty$, BLS with constant $\boldsymbol{\lambda}$, $\lambda_t = 1$, and BLS with exponentially decaying $\boldsymbol{\lambda}$, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM used for training 3 hidden layered EFNNs whose hidden layers consist of 3 neurons with ReLU activation function to approximate XOR function.

of initial parameters on the success of training should be noted. Selection of $\boldsymbol{\lambda}$ as either exponentially decaying sequence or constant 1, does not permit optimizer to search hyper-parameter space effectively if initial parameter set is bad. This problem can be solved with increasing the number of initial parameters.

5.3 Learning UCI Data Sets

The UCI data sets [53] usually used as benchmark data sets to test a novel machine learning method. These data sets comes from real world problems and include different types machine learning problems including binary-classification, multi-class classification and regression. Data sets may contain continuous attributes as well as categorical attributes. In addition to, comparison of BLS and standard backpropagation algorithm, each of these 9 data sets is used for comparing classic feed-forward neural networks and ef operator based neural networks (EFNN). These data sets are **abalone**, **connectionist bench**, **ecoli**, **glass identification**, **iris**, **leaf**, **letter recognition**, **wine**, and **yeast data sets**. Each data set is used for evaluating the proposed algorithm by training different EFNNs using different hyper-parameters.

5.3.1 Overview of Data Sets

This section gives a brief overview of each data sets, which includes types of feature vectors and their dynamic ranges, as well as their mean values and standard deviations. In addition to description of input vectors, number of classes in the data sets, and number of samples for each of these classes are also provided.

5.3.1.1 Overview of Abalone Data Set

The main task of abalone data set is to predict the age of abalones. Data set contains 4177 samples and each of the samples is represented by a 8-dimensional feature vector. The type of the entries of feature vector is given in Table 5.3. Detailed information regarding the feature space of this data set can be found at [64].

Table 5.3: The general structure of the entries of feature vectors of abalone data set.

Feature entry index	Type	Dynamic range Min value	Dynamic range Max value	Mean of each entry	Standard deviation of each entry
0	Categorical	0	2	0.992	0.796
1	Continuous	0.075	0.815	0.524	0.12
2	Continuous	0.055	0.65	0.408	0.099
3	Continuous	0	1.13	0.14	0.042
4	Continuous	0.002	2.826	0.829	0.49
5	Continuous	1.0e-03	1.488	0.359	0.222
6	Continuous	5.0e-04	0.76	0.181	0.11
7	Continuous	0.002	1.005	0.239	0.139

The number of samples across 28 classes are given in Table 5.4, in addition to the structure of feature vectors, given in Table 5.3. The distribution of samples for different classes of abalone is highly unbalanced which can be seen from Table 5.4. There is no prior division of samples as training and test samples. Therefore experiments done using 5-fold stratified cross validation for abalone data set [71]. Classes with indices 0, 1, 21, 22, 23, 24, 25, 26, and 27 have less than 10 samples and consequently these classes can cause high variance between different folds.

Table 5.4: The number of samples of all 28 classes of abalone data set.

Class index	Count	Class index	Count	Class index	Count
0	1	10	487	20	14
1	1	11	267	21	6
2	15	12	203	22	9
3	57	13	126	23	2
4	115	14	103	24	1
5	259	15	67	25	1
6	391	16	58	26	2
7	568	17	42	27	1
8	689	18	32		
9	634	19	26		

5.3.1.2 Overview of Connectionist Bench Data Set

The main task of connectionist bench data set is to predict correct vowel whose utterance given by different speakers. Data set contains 990 samples and each of the samples is represented by a 10-dimensional feature vector. The type of the entries of feature vector is given in Table 5.5. Detailed information regarding the feature space of this data set can be found at [23].

Table 5.5: The general structure of the entries of feature vectors of connectionist bench data set.

Feature entry index	Type	Dynamic range Min value	Dynamic range Max value	Mean of each entry	Standard deviation of each entry
0	Continuous	-5.211	-0.941	-3.204	0.869
1	Continuous	-1.274	5.074	1.882	1.175
2	Continuous	-2.487	1.431	-0.508	0.712
3	Continuous	-1.409	2.377	0.515	0.759
4	Continuous	-2.127	1.831	-0.306	0.664
5	Continuous	-0.836	2.327	0.63	0.604
6	Continuous	-1.537	1.403	-0.004	0.462
7	Continuous	-1.293	2.039	0.337	0.573
8	Continuous	-1.613	1.309	-0.303	0.57
9	Continuous	-1.68	1.396	-0.071	0.604

The number of samples across 11 classes are given in Table 5.6, in addition to the structure of feature vectors, given in Table 5.5. The distribution of samples for different classes of connectionist bench is balanced which can be seen from Table 5.6. There is a prior division of samples as training and test samples. 16 of 30 samples from each class labeled as training samples and remaining 14 samples labeled as test samples. This prior division used during experiments instead of 5-fold cross validation.

Table 5.6: The number of samples of all 11 classes of connectionist bench data set.

Class index	Count	Class index	Count	Class index	Count
0	90	4	90	8	90
1	90	5	90	9	90
2	90	6	90	10	90
3	90	7	90		

5.3.1.3 Overview of Ecoli Data Set

The main task of ecoli data set is to predict the localization site of proteins. Data set contains 336 samples and each of the samples is represented by a 7-dimensional feature vector. The type of the entries of feature vector is given in Table 5.7. Detailed information regarding the feature space of this data set can be found at [63].

Table 5.7: The general structure of the entries of feature vectors of ecoli data set.

Feature entry index	Type	Dynamic range		Mean of each entry	Standard deviation of each entry
		Min value	Max value		
0	Continuous	0	0.89	0.5	0.194
1	Continuous	0.16	1	0.5	0.148
2	Continuous	0.48	1	0.495	0.088
3	Continuous	0.5	1	0.501	0.027
4	Continuous	0	0.88	0.5	0.122
5	Continuous	0.03	1	0.5	0.215
6	Continuous	0	0.99	0.5	0.209

The number of samples across 8 classes are given in Table 5.8, in addition to the structure of feature vectors, given in Table 5.7. The distribution of samples for different classes of ecoli is highly unbalanced which can be seen from Table 5.8. There is no prior division of samples as training and test samples. Therefore experiments done using 5-fold stratified cross validation for ecoli data set [71].

Table 5.8: The number of samples of all 8 classes of ecoli data set.

Class index	Count	Class index	Count	Class index	Count
0	20	3	5	6	77
1	52	4	2	7	143
2	2	5	35		

5.3.1.4 Overview of Glass Identification Data Set

The main task of glass identification data set is to predict the type of glasses. Data set contains 214 samples and each of the samples is represented by a 9-dimensional feature vector. The type of the entries of feature vector is given in Table 5.9. Detailed information regarding the feature space of this data set can be found at [27].

Table 5.9: The general structure of the entries of feature vectors of glass identification data set.

Feature entry index	Type	Dynamic range Min value	Dynamic range Max value	Mean of each entry	Standard deviation of each entry
0	Continuous	1.511	1.534	1.518	0.003
1	Continuous	10.73	17.38	13.41	0.815
2	Continuous	0	4.49	2.685	1.439
3	Continuous	0.29	3.5	1.445	0.498
4	Continuous	69.81	75.41	72.65	0.773
5	Continuous	0	6.21	0.497	0.651
6	Continuous	5.43	16.19	8.957	1.42
7	Continuous	0	3.15	0.175	0.496
8	Continuous	0	0.51	0.057	0.097

The number of samples across 6 classes are given in Table 5.10, in addition to the structure of feature vectors, given in Table 5.9. The distribution of samples for different classes of glass identification is highly unbalanced which can be seen from Table 5.10. There is no prior division of samples as training and test samples. Therefore experiments done using 5-fold stratified cross validation for this data set [71].

Table 5.10: The number of samples of all 6 classes of glass identification data set.

Class index	Count	Class index	Count	Class index	Count
0	70	2	17	4	9
1	76	3	13	5	29

5.3.1.5 Overview of Iris Data Set

The main task of iris data set is to predict the type of iris plants. Data set contains 150 samples and each of the samples is represented by a 3-dimensional feature vector. The type of the entries of feature vector is given in Table 5.11. Detailed information regarding the feature space of this data set can be found at [30].

Table 5.11: The general structure of the entries of feature vectors of iris data set.

Feature entry index	Type	Dynamic range		Mean of each entry	Standard deviation of each entry
		Min value	Max value		
0	Continuous	4.3	7.9	5.843	0.825
1	Continuous	2	4.4	3.054	0.432
2	Continuous	1	6.9	3.759	1.759
3	Continuous	0.1	2.5	1.199	0.761

The number of samples across 3 classes are given in Table 5.12, in addition to the structure of feature vectors, given in Table 5.11. The distribution of samples for different classes of iris is balanced which can be seen from Table 5.12. There is no prior division of samples as training and test samples. Therefore experiments done using 5-fold stratified cross validation for iris data set [71] where each fold contains 10 test samples from each class and 40 training samples from each class.

Table 5.12: The number of samples of all 3 classes of iris data set.

Class index	Count	Class index	Count	Class index	Count
0	50	1	50	2	50

5.3.1.6 Overview of Leaf Data Set

The main task of leaf data set is to predict the specie of plant which the leaf belong to. Data set contains 340 samples and each of the samples is represented by a 14-dimensional feature vector. The type of the entries of feature vector is given in Table 5.13. Detailed information regarding the feature space of this data set can be found at [75].

Table 5.13: The general structure of the entries of feature vectors of leaf data set.

Feature entry index	Type	Dynamic range		Mean of each entry	Standard deviation of each entry
		Min value	Max value		
0	Continuous	0.117	0.999	0.72	0.208
1	Continuous	1.007	19.04	2.44	2.595
2	Continuous	0.108	0.948	0.514	0.195
3	Continuous	0.485	0.994	0.904	0.114
4	Continuous	0.396	1	0.944	0.115
5	Continuous	0.078	0.858	0.531	0.217
6	Continuous	0.003	0.199	0.037	0.039
7	Continuous	0.001	7.206	0.524	1.038
8	Continuous	0.005	0.191	0.051	0.036
9	Continuous	0.033	0.281	0.125	0.052
10	Continuous	0.001	0.073	0.018	0.014
11	Continuous	2.3e-04	0.03	0.006	0.005
12	Continuous	6.9e-06	0.003	3.9e-04	4.3e-04
13	Continuous	0.169	2.708	1.163	0.584

The number of samples across 30 classes are given in Table 5.14, in addition to the structure of feature vectors, given in Table 5.13. The distribution of samples for

different classes of leaf is balanced which can be seen from Table 5.14. However, each of the classes has small number of samples. There is no prior division of samples as training and test samples. Therefore experiments done using 5-fold stratified cross validation for leaf data set [71].

Table 5.14: The number of samples of all 30 classes of leaf data set.

Class index	Count	Class index	Count	Class index	Count
0	12	10	16	20	11
1	10	11	12	21	12
2	10	12	13	22	12
3	8	13	12	23	12
4	12	14	10	24	11
5	8	15	12	25	11
6	10	16	11	26	11
7	11	17	13	27	11
8	14	18	9	28	11
9	13	19	12	29	10

5.3.1.7 Overview of Letter Recognition Data Set

The main task of letter recognition data set is to predict the correct capital letter in the English alphabet written in one of the twenty different fonts. Data set contains 20000 samples and each of the samples is represented by a 16-dimensional feature vector. The type of the entries of feature vector is given in Table 5.15. Detailed information regarding the feature space of this data set can be found at [31].

The number of samples across 26 classes are given in Table 5.16, in addition to the structure of feature vectors, given in Table 5.15. The distribution of samples for different classes of letter recognition is balanced which can be seen from Table 5.16. There is no prior division of samples as training and test samples. Therefore experiments done using 5-fold stratified cross validation for letter recognition data set [71].

Table 5.15: The general structure of the entries of feature vectors of letter recognition data set.

Feature entry index	Type	Dynamic range Min value	Dynamic range Max value	Mean of each entry	Standard deviation of each entry
0	Categorical	0	15	4.024	1.913
1	Categorical	0	15	7.036	3.304
2	Categorical	0	15	5.122	2.015
3	Categorical	0	15	5.372	2.261
4	Categorical	0	15	3.506	2.19
5	Categorical	0	15	6.898	2.026
6	Categorical	0	15	7.5	2.325
7	Categorical	0	15	4.629	2.7
8	Categorical	0	15	5.179	2.381
9	Categorical	0	15	8.282	2.488
10	Categorical	0	15	6.454	2.631
11	Categorical	0	15	7.929	2.081
12	Categorical	0	15	3.046	2.332
13	Categorical	0	15	8.339	1.547
14	Categorical	0	15	3.692	2.567
15	Categorical	0	15	7.801	1.617

Table 5.16: The number of samples of all 26 classes of letter recognition data set.

Class index	Count	Class index	Count	Class index	Count
0	789	9	739	18	758
1	736	10	747	19	813
2	766	11	792	20	796
3	768	12	761	21	752
4	805	13	753	22	764
5	773	14	783	23	786
6	775	15	783	24	787
7	755	16	803	25	734
8	734	17	748		

5.3.1.8 Overview of Wine Data Set

The main task of wine data set is to predict the cultivar which the wine derived from. Data set contains 178 samples and each of the samples is represented by a 13-dimensional feature vector. The type of the entries of feature vector is given in Table 5.17. Detailed information regarding the feature space of this data set can be found at [3].

Table 5.17: The general structure of the entries of feature vectors of wine data set.

Feature entry index	Type	Dynamic range Min value	Dynamic range Max value	Mean of each entry	Standard deviation of each entry
0	Continuous	11.03	14.83	13	0.81
1	Continuous	0.74	5.8	2.336	1.114
2	Continuous	1.36	3.23	2.367	0.274
3	Continuous	10.6	30	19.5	3.33
4	Categorical	70	162	99.74	14.24
5	Continuous	0.98	3.88	2.295	0.624
6	Continuous	0.34	5.08	2.029	0.996
7	Continuous	0.13	0.66	0.362	0.124
8	Continuous	0.41	3.58	1.591	0.571
9	Continuous	1.28	13	5.058	2.312
10	Continuous	0.48	1.71	0.957	0.228
11	Continuous	1.27	4	2.612	0.708
12	Categorical	278	1680	746.9	314

The number of samples across 3 classes are given in Table 5.18, in addition to the structure of feature vectors, given in Table 5.17. The distribution of samples for different classes of wine is unbalanced which can be seen from Table 5.18. There is no prior division of samples as training and test samples. Therefore experiments done using 5-fold stratified cross validation for wine data set [71].

Table 5.18: The number of samples of all 3 classes of wine data set.

Class index	Count	Class index	Count	Class index	Count
0	59	1	48	2	71

5.3.1.9 Overview of Yeast Data Set

The main task of yeast data set is to predict the localization site of proteins. Data set contains 1484 samples and each of the samples is represented by a 8-dimensional feature vector. The type of the entries of feature vector is given in Table 5.19. Detailed information regarding the feature space of this data set can be found at [42].

Table 5.19: The general structure of the entries of feature vectors of yeast data set.

Feature entry index	Type	Dynamic range		Mean of each entry	Standard deviation of each entry
		Min value	Max value		
0	Continuous	0.11	1	0.5	0.137
1	Continuous	0.13	1	0.5	0.124
2	Continuous	0.21	1	0.5	0.087
3	Continuous	0	1	0.261	0.137
4	Continuous	0.5	1	0.505	0.048
5	Continuous	0	0.83	0.008	0.076
6	Continuous	0	0.73	0.5	0.058
7	Continuous	0	1	0.276	0.106

The number of samples across 10 classes are given in Table 5.20, in addition to the structure of feature vectors, given in Table 5.19. The distribution of samples for different classes of yeast is highly unbalanced which can be seen from Table 5.20. There is no prior division of samples as training and test samples. Therefore experiments done using 5-fold stratified cross validation for yeast data set [71]. Class with index 8 has less than 10 samples and consequently this class can cause high variance between different folds.

Table 5.20: The number of samples of all 10 classes of yeast data set.

Class index	Count	Class index	Count	Class index	Count
0	35	4	429	8	5
1	20	5	44	9	244
2	463	6	163		
3	30	7	51		

5.3.2 Experimental Design

Selected UCI data sets are multi-class classification problems. Each classification problem is formulated as the function approximation from feature vectors to one-hot-vector-encoding of class labels. Experiments for UCI data sets consists of training trials which are performed by using multiple network topologies, two activation functions, two optimizers and two objective functions, to approximate these functions. Experiments compare the standard backpropagation algorithm and proposed algorithm using different $\boldsymbol{\lambda}$ sequences on EFNNs. Five different $\boldsymbol{\lambda}$ sequences are used for experiments. 4 of 5 different $\boldsymbol{\lambda}$ sequences are constants and other one is exponentially decaying sequence. 4 constant values are 1, 2, 10, and 100. The constant value 1 does not allow any increase on loss which results strict decrease on loss, if no mini-batch is used. The constant value 2 allows small increments on loss which may allow algorithm to explore more regions during any part of the training without allowing divergence through prevention of visiting "bad" partitions. The loss should increase drastically when parameters changed from "good" partition to "bad" partition. The constant value 10 is more relaxed version of constant value 2 and the constant value 100 should behave almost the same as the standard backpropagation in most situations. Exponentially decaying sequence employed, is

$$\lambda_t = 1 + \delta^{2^t}, \tag{5.6}$$

where δ is $1 - 10^{-8}$. The exponentially decaying sequence satisfies the restriction

$$\lim_{T \to \infty} \prod_{t=1}^{T} \lambda_t, \tag{5.7}$$

exists and finite, whenever $\delta < 1$ as

$$1 - \delta^{2^{N+1}} = (1 - \delta) \prod_{t=0}^{N} (1 + \delta^{2^t}), \tag{5.8}$$

for every $N \in \mathbb{Z}^+$. Also, the sequence consisting of only 1s satisfies the restriction. Although, other three sequences do not satisfy the restriction, violation of the restriction is not an issue as training trials done for at most 1,000 iterations.

Two different network topologies, consisting of 2 hidden layers and 3 hidden layers are used for experiments. There is no prior work on these data sets, which uses an ef operator based algorithm. Henceforth, large number of hidden neuron combinations tested to determine "best" working architectures. All topologies used during experiments contain the same number of neurons in hidden layers. These numbers are selected as 5, 10, 20, 50, and 100. Identity and ReLU functions are considered for the activation function. Resulting EFNNs are trained with respect to mean square error and average cross entropy loss functions using ADAM and SGD optimizers. If average cross entropy used as loss function, then outputs of EFNNs normalized to interval [0, 1] using softmax function.

Hyper-parameters for SGD are the learning rate and the number of mini-batches and hyper-parameters for ADAM are the learning rate, the number of mini-batches, β_1, β_2, and ε. β_1, β_2, and ε is not optimized, instead default values, suggested by original authors [48] are used as 0.9, 0.99, and 10^{-8}, respectively. Learning rates for the optimizers are selected as 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10} using classification accuracy over training data. Number of mini-batches are selected from 1 and 4 using classification accuracy over training data. 10 randomly sampled initial parameter sets are selected for each network topology using following normal distributions

$$\mathcal{N}\Big(0,\ \frac{2}{n_{in} + n_{out}}\Big), \tag{5.9}$$

and

$$\mathcal{N}(0,\ 10^{-4}), \tag{5.10}$$

where n_{in} is number of neurons in the previous layer, and n_{out} is number of neurons in the current layer as suggested by by Glorot et al. [33]. Weights are sampled inde-

pendently from the distribution (5.9) and biases are sampled independently from the distribution (5.10).

Full grid-search over hyper-parameters is done with two different initial points for standard backpropagation algorithm to determine the best working network architecture, activation function, loss function and optimizer combinations. Among these trials 100 of them are selected for further trials according to average maximal training accuracy achieved over two initial parameter sets including all network topology, activation function, loss function and optimizer quadruples, due to time and resource constraints. Other initial parameters are tested using configurations of these trials for standard backpropagation algorithm. BLS with different $\boldsymbol{\lambda}$ sequences are also tested using configurations of these 100 trials over 10 initial parameter sets.

5.3.3 Results

Each subsubsection contains a table presenting the average test accuracy obtained from different trials by each $\boldsymbol{\lambda}$ sequences using different hyper-parameters and initial parameters for different network topology, activation function, optimizer and loss function quadruples. Each table contains a single two hidden layered network with two activation functions (identity and ReLU) and single three hidden layered network with two activation functions (identity and ReLU). These topologies are selected according to maximal average training classification accuracy achieved by each topology. The maximal average training classification performance achieved by a topology is calculated as follow: Hyper-parameters and initial parameter sets maximazing training classification accuracy for each optimizer, objective function, activation function, and $\boldsymbol{\lambda}$ sequence combinations on each fold are determined for each topology. Maximum training classification accuracy is achieved during the training using these hyper-parameters and initial parameter sets for each optimizer, objective function, activation function, and $\boldsymbol{\lambda}$ sequence combinations on each fold is computed for each topology. Average of these training classification accuracies is computed over each fold for each optimizer, objective function, activation function, $\boldsymbol{\lambda}$ sequence, and topology. Maximal average training performance achieved by each topology is com-

puted as maximum among average training classification accuracy computed for each optimizer, objective function, activation function, and λ sequence for that topology.

Average test accuracies presented in tables are calculated using different trials from each fold. Hyper-parameters achieving training classification accuracy for each optimizer, objective function, activation function, and λ sequence combinations on each fold which is not less than 0.99 times the maximum training classification accuracy achieved by these combinations determined for each topology. Test accuracy at the iteration which achieves maximum training classification accuracy, for each of these hyper-parameters, optimizer, activation function, λ sequence, fold, and topology combinations computed. If there are more than one such iterations, then the iteration achieving the smallest training loss is used. Average of these test accuracies is presented for each optimizer, objective function, activation function, λ sequence, and topology combination.

Each subsubsection also contains a figure presenting the change of objective function with respect to iterations for selected optimizer, objective function, activation function, topology and λ sequence combination. The change of objective function with respect to iterations is given for the trials which are used for calculation of average test accuracy.

Finally, each subsection contains a comparison between average classification performances achieved by classic feed-forward neural networks (FNN) and average classification performances achieved by EFNNs. These comparison is done using network topologies selected for comparison of BLS and standard backpropagation algorithm, applied to EFNNs. FNNs are trained using activation functions, objective functions and optimizers used when BLS and standard backpropagation algorithm compared. Average classification performances achieved by EFNNs are calculated using networks trained with standard backpropagation algorithm and BLS algorithm.

5.3.3.1 Abalone Data Set

Table 5.21 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on abalone data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.21 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.21 contains two different network topologies which are 2 hidden layers each of which contains 100 neurons and 3 hidden layers each of which contains 100 neurons.

Table 5.21: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 100 many neurons, and 3 hidden layered networks whose hidden layers contain 100 many neurons employing identity and ReLU activation functions trained on abalone data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	FNN	17.2 ± 0.8	23.1 ± 0.9	17.2 ± 0.8	23.0 ± 3.0
	EFNN	$\mathbf{23.0 \pm 1.0}$	$\mathbf{24.8 \pm 1.8}$	$\mathbf{18.0 \pm 1.6}$	$\mathbf{27.3 \pm 2.3}$
2 hidden layers 100 neurons ReLU	FNN	17.8 ± 0.8	25.3 ± 1.5	17.9 ± 0.7	23.7 ± 2.9
	EFNN	$\mathbf{20.9 \pm 0.5}$	$\mathbf{25.4 \pm 2.9}$	$\mathbf{19.4 \pm 1.8}$	$\mathbf{27.2 \pm 1.2}$
3 hidden layers 100 neurons Identity	FNN	19.7 ± 2.0	23.0 ± 1.7	20.0 ± 1.8	20.8 ± 1.5
	EFNN	$\mathbf{25.1 \pm 2.0}$	$\mathbf{26.8 \pm 1.8}$	$\mathbf{25.2 \pm 2.1}$	$\mathbf{26.4 \pm 2.4}$
3 hidden layers 100 neurons ReLU	FNN	20.2 ± 1.3	$\mathbf{26.6 \pm 1.9}$	20.1 ± 1.2	24.4 ± 2.2
	EFNN	$\mathbf{24.6 \pm 1.5}$	24.8 ± 1.4	$\mathbf{25.0 \pm 1.1}$	$\mathbf{27.3 \pm 1.3}$

Table 5.22: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on abalone data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 100 many neurons and 3 hidden layered networks whose hidden layers contain 100 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	∞	**23.1 ± 1.0**	25.7 ± 1.8	**18.0 ± 1.6**	27.4 ± 2.2
	1	21.3 ± 1.8	25.2 ± 1.4	**18.0 ± 1.6**	27.0 ± 1.6
	$1 + \delta^{2^t}$	21.3 ± 1.8	25.2 ± 1.5	**18.0 ± 1.6**	26.7 ± 1.5
	2	**23.1 ± 1.0**	25.9 ± 1.5	**18.0 ± 1.6**	**27.9 ± 2.0**
	10	**23.1 ± 1.0**	25.6 ± 1.5	**18.0 ± 1.6**	27.2 ± 2.2
	100	**23.1 ± 1.0**	**26.2 ± 1.8**	**18.0 ± 1.6**	27.8 ± 2.2
2 hidden layers 100 neurons ReLU	∞	21.1 ± 0.8	26.9 ± 2.2	18.9 ± 2.1	27.0 ± 1.6
	1	18.9 ± 2.0	26.3 ± 2.1	18.7 ± 1.7	26.9 ± 1.5
	$1 + \delta^{2^t}$	19.4 ± 2.3	26.6 ± 2.4	18.4 ± 1.2	**27.2 ± 1.7**
	2	20.4 ± 0.7	**27.1 ± 1.9**	18.7 ± 1.5	26.6 ± 1.6
	10	**21.7 ± 1.9**	**27.1 ± 1.7**	18.6 ± 1.4	26.9 ± 1.4
	100	20.9 ± 1.8	26.2 ± 2.6	**19.0 ± 1.8**	26.9 ± 1.4
3 hidden layers 100 neurons Identity	∞	24.9 ± 1.3	25.9 ± 1.9	**25.3 ± 2.2**	26.7 ± 2.0
	1	21.5 ± 2.6	20.9 ± 2.2	25.2 ± 2.2	**27.1 ± 2.2**
	$1 + \delta^{2^t}$	21.1 ± 3.0	23.5 ± 2.2	25.1 ± 2.3	26.0 ± 2.0
	2	24.9 ± 2.2	**26.7 ± 1.8**	25.2 ± 2.0	26.3 ± 1.6
	10	**25.5 ± 1.6**	**26.7 ± 1.9**	25.2 ± 2.0	26.5 ± 2.1
	100	25.3 ± 1.9	25.0 ± 2.4	25.2 ± 2.0	26.4 ± 1.5
3 hidden layers 100 neurons ReLU	∞	24.8 ± 1.6	**25.0 ± 2.2**	**24.2 ± 1.5**	27.1 ± 1.9
	1	23.3 ± 1.9	**25.0 ± 2.6**	23.1 ± 1.7	26.7 ± 1.6
	$1 + \delta^{2^t}$	23.2 ± 0.8	**25.0 ± 2.8**	23.1 ± 1.7	26.5 ± 1.6
	2	24.3 ± 1.5	24.8 ± 1.4	24.1 ± 1.4	**27.3 ± 1.4**
	10	24.9 ± 1.6	24.8 ± 1.4	23.8 ± 1.4	**27.3 ± 1.4**
	100	**25.1 ± 1.4**	24.8 ± 1.4	**24.2 ± 1.4**	**27.3 ± 1.4**

In addition to Table 5.21, Table 5.22 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances

computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and SGD optimization methods. The network topologies presented in Table 5.22 are 2 hidden layers each of which containing 100 neurons and 3 hidden layers each of which contains 100 neurons, similar to Table 5.21.

Table 5.23: The number of different hyper-parameter sets used on trials given in Figure 5.3

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	1	5
		4	2
1	10^{-4}	4	7
$1+\delta^{2^t}$	10^{-4}	4	7
2	10^{-4}	1	4
		4	3
10	10^{-4}	1	4
		4	3
100	10^{-4}	1	4
		4	3

Lastly, Figure 5.3 and Table 5.23 presented. Figure 5.3 shows the change of average cross entropy with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 3 hidden layers each of which contains 100 many neurons, and ReLU activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.22. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.3 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.23 instead of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

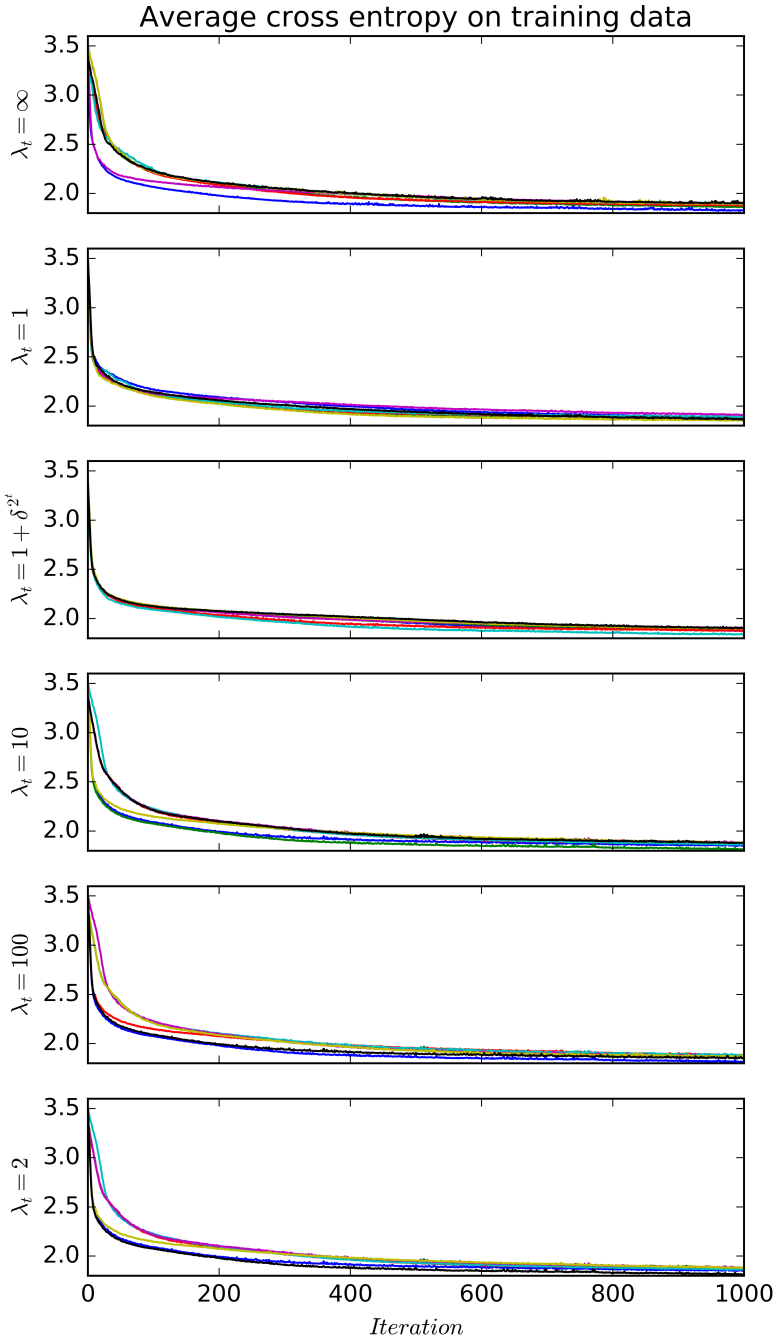


Figure 5.3: The change of average cross entropy with respect to iteration for trials achieving maximal classification performance on training data when standard back-propagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 3 hidden layered EFNNs whose hidden layers consist of 100 neurons with ReLU activation function over abalone data set.

5.3.3.2 Connectionist Bench Data Set

Table 5.24 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on connectionist bench data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.24 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.24 contains two different network topologies which are 2 hidden layers each of which contains 100 neurons and 3 hidden layers each of which contains 100 neurons.

Table 5.24: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 100 many neurons, and 3 hidden layered networks whose hidden layers contain 100 many neurons employing identity and ReLU activation functions trained on connectionist bench data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons	FNN	14.7 ± 0.0	32.1 ± 0.7	14.3 ± 0.0	$\mathbf{47.3 \pm 0.1}$
Identity	EFNN	$\mathbf{29.0 \pm 0.0}$	$\mathbf{40.7 \pm 0.0}$	$\mathbf{18.4 \pm 0.3}$	45.0 ± 0.4
2 hidden layers 100 neurons	FNN	15.4 ± 0.0	$\mathbf{55.2 \pm 0.5}$	$\mathbf{13.2 \pm 0.0}$	50.4 ± 0.0
ReLU	EFNN	$\mathbf{24.2 \pm 0.0}$	51.5 ± 1.0	11.5 ± 0.0	$\mathbf{52.7 \pm 1.2}$
3 hidden layers 100 neurons	FNN	13.9 ± 0.0	31.0 ± 1.7	11.3 ± 0.0	$\mathbf{47.2 \pm 1.1}$
Identity	EFNN	$\mathbf{39.3 \pm 2.3}$	$\mathbf{45.0 \pm 0.0}$	$\mathbf{39.0 \pm 0.2}$	35.5 ± 0.0
3 hidden layers 100 neurons	FNN	13.6 ± 0.2	$\mathbf{58.7 \pm 0.0}$	12.3 ± 0.0	51.3 ± 0.4
ReLU	EFNN	$\mathbf{36.1 \pm 1.0}$	48.6 ± 3.0	$\mathbf{33.3 \pm 1.1}$	$\mathbf{56.0 \pm 3.1}$

Table 5.25: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on vowel data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 100 many neurons and 3 hidden layered networks whose hidden layers contain 100 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	∞	**29.0 ± 0.0**	37.0 ± 1.7	**18.4 ± 0.3**	44.4 ± 0.0
	1	18.2 ± 0.0	25.8 ± 0.0	13.2 ± 0.0	41.8 ± 0.0
	$1 + \delta^{2^t}$	18.2 ± 0.0	31.0 ± 0.0	13.2 ± 0.0	41.8 ± 0.0
	2	**29.0 ± 0.0**	33.8 ± 0.0	**18.4 ± 0.3**	**45.2 ± 0.0**
	10	**29.0 ± 0.0**	33.8 ± 0.0	**18.4 ± 0.3**	**45.2 ± 0.0**
	100	**29.0 ± 0.0**	**40.7 ± 0.0**	**18.4 ± 0.3**	**45.2 ± 0.0**
2 hidden layers 100 neurons ReLU	∞	24.2 ± 0.0	**54.3 ± 0.0**	11.5 ± 0.0	52.5 ± 1.5
	1	17.5 ± 0.0	37.7 ± 0.0	10.6 ± 0.0	35.1 ± 0.0
	$1 + \delta^{2^t}$	13.2 ± 0.0	27.9 ± 0.0	10.6 ± 0.0	35.1 ± 0.0
	2	24.0 ± 0.0	50.6 ± 0.9	**11.9 ± 0.0**	**53.3 ± 0.9**
	10	**27.1 ± 0.0**	50.6 ± 0.9	**11.9 ± 0.0**	52.2 ± 0.7
	100	24.0 ± 0.0	50.6 ± 1.0	**11.9 ± 0.0**	52.8 ± 1.0
3 hidden layers 100 neurons Identity	∞	**41.1 ± 1.0**	36.6 ± 3.3	**39.0 ± 0.2**	35.5 ± 0.0
	1	20.8 ± 0.0	19.0 ± 0.0	36.6 ± 0.0	30.5 ± 0.0
	$1 + \delta^{2^t}$	25.5 ± 0.0	25.5 ± 0.0	36.6 ± 0.0	31.0 ± 0.0
	2	36.4 ± 0.0	**45.0 ± 0.0**	**39.0 ± 0.2**	**39.0 ± 4.2**
	10	39.4 ± 2.1	**45.0 ± 0.0**	**39.0 ± 0.2**	**39.0 ± 3.2**
	100	39.4 ± 2.1	**45.0 ± 0.0**	**39.0 ± 0.2**	37.7 ± 3.7
3 hidden layers 100 neurons ReLU	∞	34.6 ± 0.0	48.7 ± 2.3	**33.8 ± 0.1**	55.9 ± 2.9
	1	24.2 ± 0.0	29.2 ± 0.0	18.7 ± 1.8	34.2 ± 0.0
	$1 + \delta^{2^t}$	24.2 ± 0.0	28.4 ± 0.0	18.1 ± 1.7	34.2 ± 0.0
	2	36.8 ± 0.0	48.6 ± 3.2	33.3 ± 1.2	56.4 ± 2.6
	10	36.8 ± 0.0	47.8 ± 3.4	33.3 ± 1.2	55.8 ± 3.2
	100	**40.2 ± 1.2**	**49.4 ± 2.6**	33.0 ± 1.3	**56.8 ± 3.0**

In addition to Table 5.24, Table 5.25 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm

and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and SGD optimization methods. The network topologies presented in Table 5.25 are 2 hidden layers each of which containing 100 neurons and 3 hidden layers each of which contains 100 neurons, similar to Table 5.24.

Table 5.26: The number of different hyper-parameter sets used on trials given in Figure 5.4

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	1	5
		4	5
1	10^{-4}	1	1
		4	1
$1+\delta^{2^t}$	10^{-4}	1	1
2	10^{-4}	1	4
		4	3
10	10^{-4}	1	3
		4	4
100	10^{-4}	1	2
		4	3

Lastly, Figure 5.4 and Table 5.26 presented. Figure 5.4 shows the change of average cross entropy with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 3 hidden layers each of which contains 100 many neurons, and ReLU activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.25. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.4 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.26 instead

of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

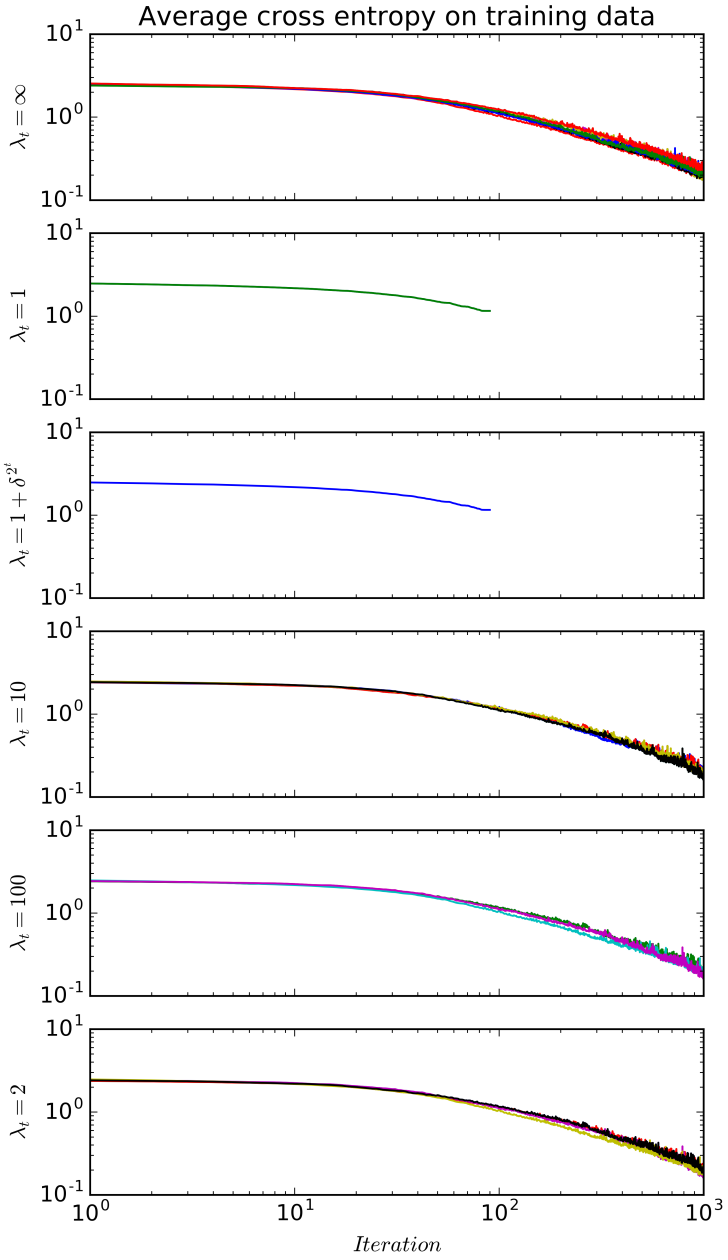


Figure 5.4: The change of average cross entropy with respect to iteration for trials achieving maximal classification performance on training data when standard back-propagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 3 hidden layered EFNNs whose hidden layers consist of 100 neurons with ReLU activation function over connectionist bench data set.

5.3.3.3 Ecoli Data Set

Table 5.27 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on ecoli data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.27 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.27 contains two different network topologies which are 2 hidden layers each of which contains 100 neurons and 3 hidden layers each of which contains 100 neurons.

Table 5.27: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 100 many neurons, and 3 hidden layered networks whose hidden layers contain 100 many neurons employing identity and ReLU activation functions trained on ecoli data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	FNN	**60.1 ± 2.0**	75.3 ± 5.1	**58.5 ± 3.7**	63.1 ± 4.9
	EFNN	53.3 ± 2.6	**85.3 ± 4.3**	52.7 ± 2.4	**84.4 ± 3.5**
2 hidden layers 100 neurons ReLU	FNN	62.8 ± 3.0	75.1 ± 5.1	62.8 ± 3.0	61.6 ± 3.7
	EFNN	**64.2 ± 1.2**	**84.1 ± 4.4**	**64.0 ± 1.4**	**83.7 ± 3.9**
3 hidden layers 100 neurons Identity	FNN	53.1 ± 2.7	74.3 ± 4.7	52.2 ± 3.0	61.2 ± 2.0
	EFNN	**85.5 ± 4.2**	**84.0 ± 5.0**	**79.3 ± 1.8**	**82.6 ± 4.4**
3 hidden layers 100 neurons ReLU	FNN	64.1 ± 1.1	66.6 ± 4.2	61.0 ± 2.7	60.2 ± 3.8
	EFNN	**79.8 ± 3.2**	**83.0 ± 5.0**	**73.8 ± 4.1**	**84.4 ± 3.2**

Table 5.28: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on ecoli data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 100 many neurons and 3 hidden layered networks whose hidden layers contain 100 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	∞	53.2 ± 2.7	84.2 ± 4.0	**52.7 ± 2.4**	84.5 ± 3.7
	1	**53.9 ± 2.1**	81.0 ± 2.4	**52.7 ± 2.4**	84.8 ± 4.4
	$1 + \delta^{2^t}$	**53.9 ± 2.1**	80.5 ± 1.3	**52.7 ± 2.4**	84.0 ± 4.2
	2	53.2 ± 2.7	84.8 ± 4.4	**52.7 ± 2.4**	**85.0 ± 3.5**
	10	53.2 ± 2.7	**85.1 ± 4.4**	**52.7 ± 2.4**	84.9 ± 3.5
	100	53.2 ± 2.7	85.0 ± 4.4	**52.7 ± 2.4**	84.8 ± 3.4
2 hidden layers 100 neurons ReLU	∞	63.7 ± 0.9	83.5 ± 4.8	**64.1 ± 1.4**	82.5 ± 4.3
	1	62.3 ± 4.9	72.4 ± 4.2	55.7 ± 9.4	77.6 ± 5.5
	$1 + \delta^{2^t}$	63.1 ± 4.3	73.6 ± 5.0	62.3 ± 4.5	77.3 ± 6.3
	2	64.0 ± 1.2	**84.4 ± 3.2**	64.0 ± 1.4	83.8 ± 3.0
	10	64.3 ± 1.2	83.2 ± 5.1	64.0 ± 1.4	84.1 ± 3.9
	100	**64.5 ± 1.2**	84.3 ± 3.4	63.9 ± 1.3	**84.3 ± 2.5**
3 hidden layers 100 neurons Identity	∞	85.2 ± 4.5	82.8 ± 4.0	**79.3 ± 1.8**	81.4 ± 5.4
	1	69.9 ± 6.1	81.4 ± 2.3	64.0 ± 4.9	86.0 ± 3.1
	$1 + \delta^{2^t}$	75.0 ± 3.3	83.2 ± 2.6	64.0 ± 4.9	**86.6 ± 3.2**
	2	85.5 ± 3.9	84.4 ± 5.3	**79.3 ± 1.8**	80.5 ± 5.3
	10	85.5 ± 3.9	**84.6 ± 4.3**	**79.3 ± 1.8**	79.0 ± 5.7
	100	**85.6 ± 4.0**	84.5 ± 4.9	**79.3 ± 1.8**	81.2 ± 5.2
3 hidden layers 100 neurons ReLU	∞	**80.6 ± 3.7**	83.1 ± 3.4	73.8 ± 3.7	82.0 ± 3.7
	1	58.2 ± 4.1	76.9 ± 1.2	55.6 ± 2.1	75.9 ± 2.4
	$1 + \delta^{2^t}$	60.5 ± 3.4	75.5 ± 3.4	57.9 ± 3.5	76.8 ± 1.8
	2	79.2 ± 2.4	**84.4 ± 5.0**	73.7 ± 4.4	83.4 ± 4.0
	10	79.1 ± 3.7	83.8 ± 4.3	73.6 ± 4.7	**83.5 ± 3.9**
	100	79.1 ± 2.8	**84.4 ± 4.4**	**73.9 ± 3.7**	83.3 ± 4.0

In addition to Table 5.27, Table 5.28 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances

computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and SGD optimization methods. The network topologies presented in Table 5.28 are 2 hidden layers each of which containing 100 neurons and 3 hidden layers each of which contains 100 neurons, similar to Table 5.27.

Table 5.29: The number of different hyper-parameter sets used on trials given in Figure 5.5

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	1	17
		4	16
1	10^{-4}	1	7
		4	5
$1+\delta^{2^t}$	10^{-4}	1	4
		4	5
2	10^{-4}	1	23
		4	21
10	10^{-4}	1	20
		4	21
100	10^{-4}	1	22
		4	22

Lastly, Figure 5.5 and Table 5.29 presented. Figure 5.5 shows the change of mean squared error with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 2 hidden layers each of which contains 100 many neurons, and identity activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.28. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.5 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.29 instead

of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

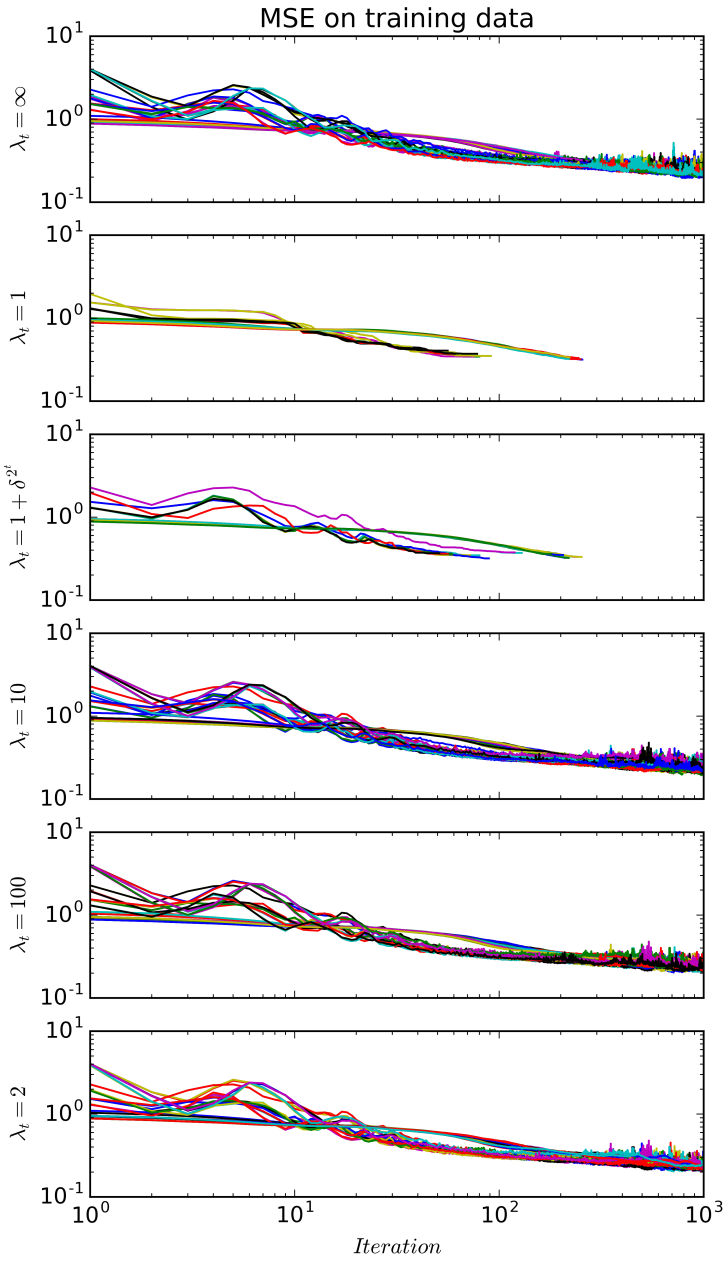


Figure 5.5: The change of mean squared error with respect to iteration for trials achieving maximal classification performance on training data when standard back-propagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 2 hidden layered EFNNs whose hidden layers consist of 100 neurons with identity activation function over ecoli data set.

5.3.3.4 Glass Identification Data Set

Table 5.30 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on glass identification data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.30 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.30 contains two different network topologies which are 2 hidden layers each of which contains 100 neurons and 3 hidden layers each of which contains 100 neurons.

Table 5.30: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 100 many neurons, and 3 hidden layered networks whose hidden layers contain 100 many neurons employing identity and ReLU activation functions trained on glass identification data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	FNN	40.8 ± 7.9	48.3 ± 8.7	42.1 ± 7.4	57.7 ± 8.2
	EFNN	$\mathbf{50.8 \pm 10.2}$	$\mathbf{60.7 \pm 6.0}$	$\mathbf{47.9 \pm 10.0}$	$\mathbf{59.6 \pm 5.0}$
2 hidden layers 100 neurons ReLU	FNN	43.6 ± 4.7	50.1 ± 9.7	$\mathbf{48.1 \pm 2.8}$	$\mathbf{57.1 \pm 6.3}$
	EFNN	$\mathbf{51.0 \pm 4.0}$	$\mathbf{55.0 \pm 7.2}$	42.1 ± 11.4	56.4 ± 3.8
3 hidden layers 100 neurons Identity	FNN	36.7 ± 2.9	50.2 ± 10.3	37.0 ± 3.4	58.5 ± 7.0
	EFNN	$\mathbf{44.4 \pm 5.4}$	$\mathbf{57.5 \pm 5.8}$	$\mathbf{55.4 \pm 11.7}$	$\mathbf{60.1 \pm 7.2}$
3 hidden layers 100 neurons ReLU	FNN	37.7 ± 8.5	$\mathbf{56.2 \pm 5.1}$	39.2 ± 9.3	56.2 ± 7.0
	EFNN	$\mathbf{55.2 \pm 7.3}$	54.8 ± 7.7	$\mathbf{53.0 \pm 5.1}$	$\mathbf{59.1 \pm 5.5}$

In addition to Table 5.30, Table 5.31 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm

Table 5.31: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on glass data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 100 many neurons and 3 hidden layered networks whose hidden layers contain 100 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	∞	41.0 ± 7.3	61.6 ± 5.6	46.5 ± 9.9	58.5 ± 4.6
	1	48.0 ± 13.1	31.5 ± 8.3	$\mathbf{47.9 \pm 9.3}$	41.8 ± 3.8
	$1 + \delta^{2^t}$	47.8 ± 6.6	47.7 ± 12.1	$\mathbf{47.9 \pm 9.3}$	41.8 ± 4.8
	2	$\mathbf{50.7 \pm 10.1}$	60.3 ± 6.3	46.5 ± 9.9	$\mathbf{60.8 \pm 4.8}$
	10	41.0 ± 7.3	$\mathbf{61.7 \pm 3.5}$	46.5 ± 9.9	60.7 ± 4.7
	100	41.0 ± 7.3	61.4 ± 5.6	46.5 ± 9.9	59.1 ± 5.0
2 hidden layers 100 neurons ReLU	∞	50.0 ± 7.5	57.2 ± 7.8	37.6 ± 10.6	57.5 ± 3.1
	1	37.3 ± 10.0	40.9 ± 8.1	39.2 ± 9.7	41.5 ± 6.4
	$1 + \delta^{2^t}$	44.7 ± 4.2	43.5 ± 3.9	39.4 ± 8.2	40.2 ± 9.4
	2	48.9 ± 5.9	55.4 ± 8.3	$\mathbf{39.8 \pm 9.1}$	55.7 ± 3.6
	10	50.3 ± 3.5	55.4 ± 6.4	37.6 ± 8.8	56.4 ± 3.9
	100	$\mathbf{53.7 \pm 4.1}$	$\mathbf{59.0 \pm 8.6}$	$\mathbf{39.8 \pm 9.1}$	$\mathbf{58.6 \pm 4.1}$
3 hidden layers 100 neurons Identity	∞	34.4 ± 9.5	54.5 ± 6.3	52.3 ± 12.9	59.7 ± 7.9
	1	31.1 ± 9.0	38.9 ± 6.8	48.0 ± 7.9	43.6 ± 3.8
	$1 + \delta^{2^t}$	$\mathbf{44.6 \pm 4.8}$	39.3 ± 6.7	47.9 ± 2.4	50.1 ± 9.3
	2	43.9 ± 6.1	44.6 ± 7.9	55.8 ± 13.5	58.8 ± 3.2
	10	35.0 ± 11.0	55.7 ± 6.1	$\mathbf{58.9 \pm 10.3}$	60.3 ± 5.6
	100	34.4 ± 9.5	$\mathbf{57.9 \pm 5.8}$	56.3 ± 12.3	$\mathbf{62.8 \pm 6.7}$
3 hidden layers 100 neurons ReLU	∞	47.8 ± 14.6	55.6 ± 7.1	52.8 ± 4.9	59.0 ± 6.7
	1	39.2 ± 9.1	37.9 ± 8.3	46.2 ± 11.9	45.7 ± 5.2
	$1 + \delta^{2^t}$	47.6 ± 17.2	41.2 ± 7.2	47.1 ± 11.9	43.4 ± 5.4
	2	$\mathbf{55.2 \pm 7.3}$	49.8 ± 16.7	$\mathbf{58.3 \pm 5.8}$	$\mathbf{59.1 \pm 3.3}$
	10	48.3 ± 15.6	$\mathbf{56.1 \pm 8.1}$	54.1 ± 5.7	57.8 ± 4.7
	100	47.9 ± 14.9	55.5 ± 10.1	55.3 ± 5.7	57.4 ± 4.3

and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and

SGD optimization methods. The network topologies presented in Table 5.31 are 2 hidden layers each of which containing 100 neurons and 3 hidden layers each of which contains 100 neurons, similar to Table 5.30.

Table 5.32: The number of different hyper-parameter sets used on trials given in Figure 5.6

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	1	6
		4	4
1	10^{-4}	1	3
		4	3
	10^{-6}	1	2
		4	2
$1+\delta^{2^t}$	10^{-4}	1	4
		4	5
2	10^{-4}	1	8
		4	3
10	10^{-4}	1	3
		4	5
100	10^{-4}	1	4
		4	5

Lastly, Figure 5.6 and Table 5.32 presented. Figure 5.6 shows the change of mean squared error with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 2 hidden layers each of which contains 100 many neurons, and identity activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.31. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.6 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.32 instead

of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

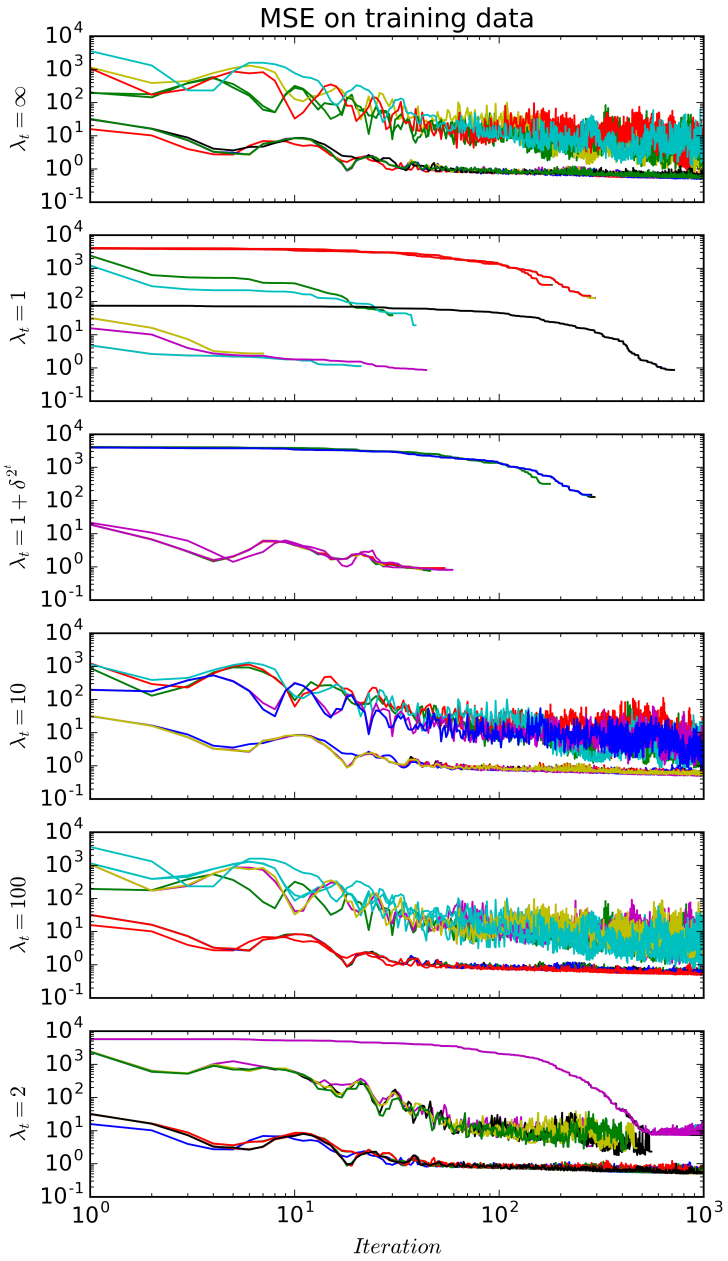


Figure 5.6: The change of mean squared error with respect to iteration for trials achieving maximal classification performance on training data when standard back-propagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 2 hidden layered EFNNs whose hidden layers consist of 100 neurons with identity activation function over glass identification data set.

5.3.3.5 Iris Data Set

Table 5.33 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on iris data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.33 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.33 contains two different network topologies which are 2 hidden layers each of which contains 50 neurons and 3 hidden layers each of which contains 100 neurons.

Table 5.33: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 50 many neurons, and 3 hidden layered networks whose hidden layers contain 100 many neurons employing identity and ReLU activation functions trained on iris data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 50 neurons Identity	FNN	90.0 ± 6.3	67.6 ± 8.5	54.7 ± 4.5	$\mathbf{97.6 \pm 2.2}$
	EFNN	$\mathbf{94.3 \pm 3.2}$	$\mathbf{94.0 \pm 5.1}$	$\mathbf{90.0 \pm 4.7}$	96.9 ± 3.5
2 hidden layers 50 neurons ReLU	FNN	78.0 ± 7.5	94.4 ± 4.2	66.3 ± 1.0	95.2 ± 3.3
	EFNN	$\mathbf{93.3 \pm 3.3}$	$\mathbf{96.7 \pm 3.8}$	$\mathbf{94.7 \pm 3.8}$	$\mathbf{96.9 \pm 3.3}$
3 hidden layers 100 neurons Identity	FNN	65.5 ± 1.6	66.6 ± 0.5	66.7 ± 0.0	$\mathbf{96.8 \pm 3.1}$
	EFNN	$\mathbf{91.2 \pm 4.9}$	$\mathbf{95.5 \pm 4.3}$	$\mathbf{97.3 \pm 3.1}$	96.0 ± 4.0
3 hidden layers 100 neurons ReLU	FNN	65.9 ± 1.4	$\mathbf{97.1 \pm 3.1}$	65.3 ± 2.7	96.9 ± 3.1
	EFNN	$\mathbf{97.5 \pm 3.4}$	96.6 ± 3.6	$\mathbf{97.3 \pm 3.3}$	$\mathbf{97.4 \pm 3.2}$

Table 5.34: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on iris data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 50 many neurons and 3 hidden layered networks whose hidden layers contain 100 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 50 neurons Identity	∞	**94.3 ± 3.2**	**94.1 ± 4.9**	**90.0 ± 4.7**	**96.9 ± 3.6**
	1	67.1 ± 5.5	84.0 ± 9.8	80.0 ± 7.0	88.1 ± 4.7
	$1 + \delta^{2^t}$	67.1 ± 5.5	91.6 ± 4.1	80.0 ± 7.0	94.7 ± 4.0
	2	**94.3 ± 3.2**	94.0 ± 5.2	**90.0 ± 4.7**	**96.9 ± 3.6**
	10	**94.3 ± 3.2**	94.0 ± 5.2	**90.0 ± 4.7**	**96.9 ± 3.5**
	100	**94.3 ± 3.2**	**94.1 ± 5.2**	**90.0 ± 4.7**	**96.9 ± 3.6**
2 hidden layers 50 neurons ReLU	∞	93.1 ± 3.8	**96.8 ± 3.8**	94.0 ± 3.7	**97.1 ± 3.2**
	1	89.2 ± 6.0	90.3 ± 3.5	66.5 ± 0.8	92.8 ± 4.0
	$1 + \delta^{2^t}$	83.9 ± 7.0	91.1 ± 1.6	90.4 ± 6.2	93.0 ± 2.5
	2	92.9 ± 4.2	**96.8 ± 3.9**	94.9 ± 3.8	97.0 ± 3.1
	10	**94.6 ± 4.4**	96.7 ± 3.6	**96.3 ± 3.1**	96.9 ± 3.4
	100	93.1 ± 3.7	**96.8 ± 3.6**	94.2 ± 3.9	96.9 ± 3.5
3 hidden layers 100 neurons Identity	∞	78.1 ± 6.1	95.3 ± 4.3	**97.4 ± 3.0**	95.8 ± 4.1
	1	80.0 ± 7.9	84.0 ± 6.8	96.7 ± 3.3	**96.2 ± 3.6**
	$1 + \delta^{2^t}$	87.2 ± 5.9	95.5 ± 3.6	96.1 ± 3.6	94.7 ± 4.7
	2	**91.2 ± 4.9**	**96.0 ± 4.5**	**97.4 ± 3.0**	95.9 ± 4.2
	10	78.1 ± 6.1	95.5 ± 4.2	**97.4 ± 3.0**	95.9 ± 4.0
	100	78.1 ± 6.1	95.6 ± 4.1	**97.4 ± 3.0**	96.0 ± 4.0
3 hidden layers 100 neurons ReLU	∞	**97.5 ± 3.4**	**96.9 ± 3.4**	**97.8 ± 2.8**	97.4 ± 3.3
	1	96.2 ± 3.1	92.1 ± 4.5	95.2 ± 4.8	95.7 ± 4.4
	$1 + \delta^{2^t}$	94.4 ± 3.8	95.8 ± 3.5	96.2 ± 3.9	95.3 ± 3.7
	2	**97.5 ± 3.3**	96.6 ± 3.4	97.1 ± 3.6	**97.8 ± 3.1**
	10	97.4 ± 3.4	96.7 ± 4.0	97.0 ± 3.5	97.5 ± 3.0
	100	97.3 ± 3.5	96.7 ± 3.7	97.3 ± 3.4	97.7 ± 3.0

In addition to Table 5.33, Table 5.34 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances

computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and SGD optimization methods. The network topologies presented in Table 5.34 are 2 hidden layers each of which containing 50 neurons and 3 hidden layers each of which contains 100 neurons, similar to Table 5.33.

Lastly, Figure 5.7 and Table 5.35 presented. Figure 5.7 shows the change of mean squared error with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 3 hidden layers each of which contains 100 many neurons, and ReLU activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.34. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.7 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.35 instead of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

Table 5.35: The number of different hyper-parameter sets used on trials given in Figure 5.7

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	1	37
		4	36
	10^{-6}	1	1
		4	1
1	10^{-4}	1	7
		4	7
$1 + \delta^{2^t}$	10^{-4}	1	6
		4	9
2	10^{-4}	1	36
		4	33
	10^{-6}	1	2
		4	1
10	10^{-4}	1	36
		4	35
	10^{-6}	1	2
		4	1
100	10^{-4}	1	36
		4	35
	10^{-6}	1	1
		4	1

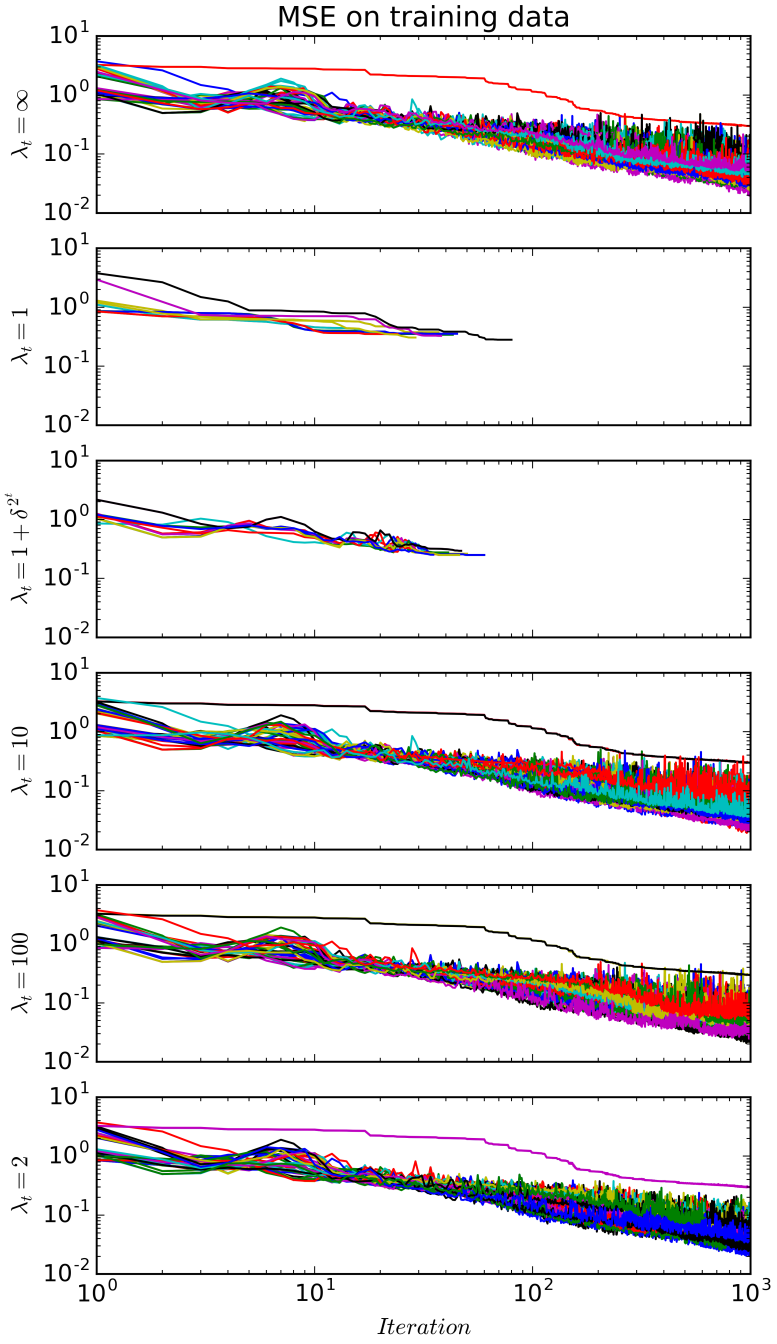


Figure 5.7: The change of mean squared error with respect to iteration for trials achieving maximal classification performance on training data when standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 3 hidden layered EFNNs whose hidden layers consist of 100 neurons with ReLU activation function over iris data set.

5.3.3.6 Leaf Data Set

Table 5.36 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on leaf data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.36 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.36 contains two different network topologies which are 2 hidden layers each of which contains 100 neurons and 3 hidden layers each of which contains 100 neurons.

Table 5.36: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 100 many neurons, and 3 hidden layered networks whose hidden layers contain 100 many neurons employing identity and ReLU activation functions trained on leaf data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	FNN	10.0 ± 2.0	32.2 ± 4.8	11.0 ± 0.8	48.0 ± 4.3
	EFNN	$\mathbf{21.9 \pm 3.9}$	$\mathbf{58.8 \pm 3.1}$	$\mathbf{14.2 \pm 1.4}$	$\mathbf{64.1 \pm 4.6}$
2 hidden layers 100 neurons ReLU	FNN	11.8 ± 0.4	51.6 ± 4.4	$\mathbf{11.9 \pm 0.7}$	46.8 ± 2.7
	EFNN	$\mathbf{19.2 \pm 3.5}$	$\mathbf{58.4 \pm 4.8}$	9.5 ± 1.4	$\mathbf{60.5 \pm 6.1}$
3 hidden layers 100 neurons Identity	FNN	8.0 ± 0.6	29.6 ± 2.5	8.0 ± 0.6	46.1 ± 5.7
	EFNN	$\mathbf{50.7 \pm 3.4}$	$\mathbf{58.3 \pm 4.4}$	$\mathbf{45.6 \pm 2.4}$	$\mathbf{71.0 \pm 5.2}$
3 hidden layers 100 neurons ReLU	FNN	8.4 ± 1.1	54.3 ± 3.2	9.0 ± 1.8	46.0 ± 5.8
	EFNN	$\mathbf{42.5 \pm 3.3}$	$\mathbf{61.3 \pm 3.9}$	$\mathbf{27.6 \pm 2.4}$	$\mathbf{64.2 \pm 3.5}$

Table 5.37: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on leaf data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 100 many neurons and 3 hidden layered networks whose hidden layers contain 100 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	∞	**21.9 ± 3.9**	59.1 ± 3.3	**14.0 ± 1.4**	63.1 ± 4.6
	1	13.0 ± 2.0	32.4 ± 3.0	13.9 ± 2.4	58.7 ± 4.3
	$1 + \delta^{2^t}$	13.0 ± 2.0	33.0 ± 2.6	13.9 ± 2.4	59.3 ± 5.3
	2	**21.9 ± 3.9**	58.8 ± 2.8	**14.0 ± 1.4**	**64.5 ± 4.8**
	10	**21.9 ± 3.9**	**60.3 ± 2.4**	**14.0 ± 1.4**	63.5 ± 5.6
	100	**21.9 ± 3.9**	60.1 ± 2.2	**14.0 ± 1.4**	63.1 ± 5.5
2 hidden layers 100 neurons ReLU	∞	17.4 ± 1.7	59.5 ± 3.4	9.4 ± 1.4	**64.6 ± 3.4**
	1	11.2 ± 3.6	29.0 ± 5.1	8.8 ± 2.1	47.9 ± 4.2
	$1 + \delta^{2^t}$	10.3 ± 2.9	28.8 ± 4.8	9.5 ± 2.7	47.5 ± 2.3
	2	**19.9 ± 3.2**	**59.7 ± 4.6**	9.4 ± 1.0	63.7 ± 6.5
	10	19.1 ± 2.7	58.8 ± 4.6	9.8 ± 1.4	63.4 ± 6.6
	100	18.3 ± 3.9	58.9 ± 4.5	**9.9 ± 1.3**	64.1 ± 6.6
3 hidden layers 100 neurons Identity	∞	**50.1 ± 3.3**	**60.6 ± 5.2**	**45.6 ± 2.4**	70.1 ± 4.6
	1	12.0 ± 2.1	16.4 ± 2.8	35.8 ± 3.0	63.5 ± 3.6
	$1 + \delta^{2^t}$	20.3 ± 2.9	15.5 ± 3.2	35.8 ± 3.0	62.0 ± 4.0
	2	48.1 ± 5.7	57.0 ± 4.4	**45.6 ± 2.4**	**72.2 ± 4.6**
	10	48.3 ± 3.7	56.8 ± 4.2	**45.6 ± 2.4**	71.8 ± 4.4
	100	49.3 ± 6.3	58.2 ± 2.9	**45.6 ± 2.4**	71.1 ± 5.6
3 hidden layers 100 neurons ReLU	∞	**42.8 ± 3.3**	**62.2 ± 5.2**	**26.6 ± 2.7**	63.3 ± 4.1
	1	10.3 ± 2.7	22.2 ± 3.3	13.1 ± 4.7	48.2 ± 3.7
	$1 + \delta^{2^t}$	14.9 ± 1.4	19.9 ± 3.8	12.2 ± 4.2	48.2 ± 2.9
	2	41.1 ± 3.6	61.8 ± 3.6	25.9 ± 1.4	64.0 ± 3.7
	10	39.8 ± 3.8	61.4 ± 3.5	26.5 ± 2.4	64.3 ± 3.5
	100	41.1 ± 3.6	61.2 ± 3.6	26.5 ± 2.4	**64.9 ± 3.2**

In addition to Table 5.36, Table 5.37 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances

computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and SGD optimization methods. The network topologies presented in Table 5.37 are 2 hidden layers each of which containing 100 neurons and 3 hidden layers each of which contains 100 neurons, similar to Table 5.36.

Table 5.38: The number of different hyper-parameter sets used on trials given in Figure 5.8

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	1	8
		4	11
1	10^{-4}	1	8
		4	9
$1+\delta^{2^t}$	10^{-4}	1	9
		4	8
2	10^{-4}	1	17
		4	17
10	10^{-4}	1	17
		4	16
100	10^{-4}	1	9
		4	9

Lastly, Figure 5.8 and Table 5.38 presented. Figure 5.8 shows the change of average cross entropy with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 3 hidden layers each of which contains 100 many neurons, and identity activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.37. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.8 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.38 instead

of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

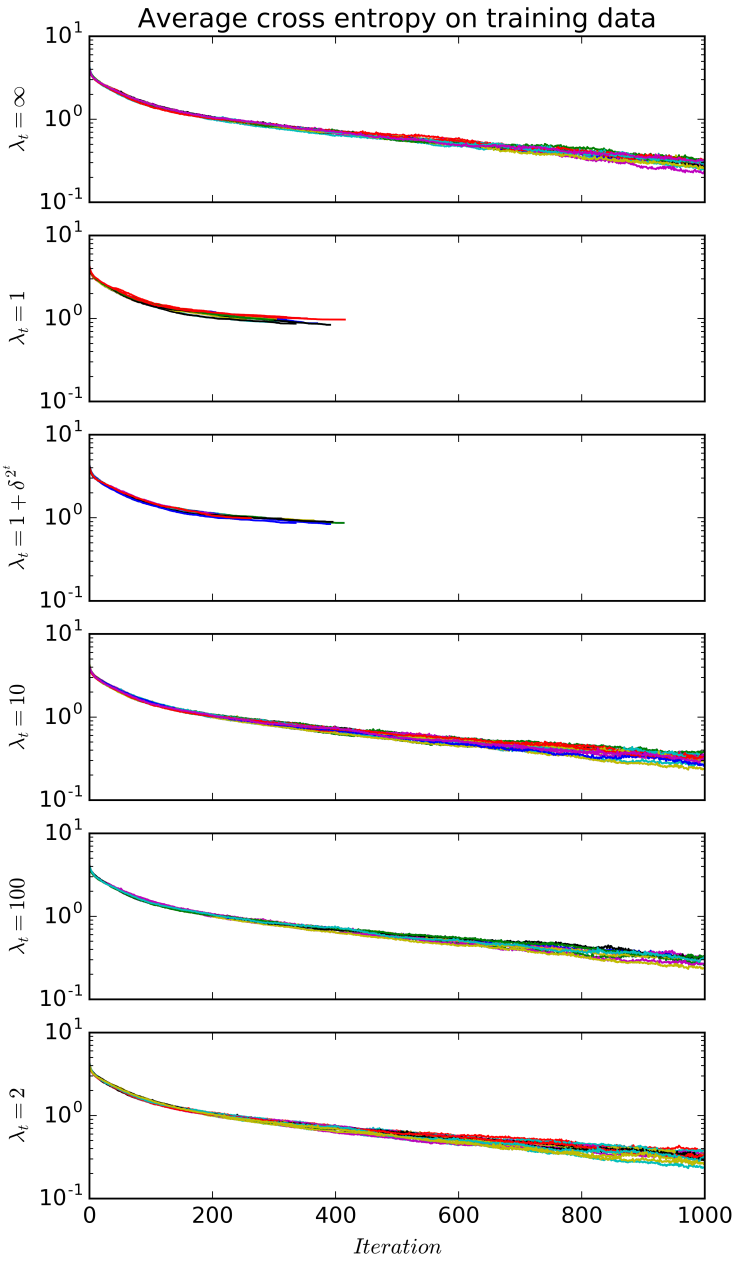


Figure 5.8: The change of average cross entropy with respect to iteration for trials achieving maximal classification performance on training data when standard back-propagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 3 hidden layered EFNNs whose hidden layers consist of 100 neurons with identity activation function over leaf data set.

5.3.3.7 Letter Recognition Data Set

Table 5.39 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on letter recognition data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.39 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.39 contains two different network topologies which are 2 hidden layers each of which contains 100 neurons and 3 hidden layers each of which contains 100 neurons.

Table 5.39: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 100 many neurons, and 3 hidden layered networks whose hidden layers contain 100 many neurons employing identity and ReLU activation functions trained on letter recognition data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	FNN	7.7 ± 0.3	55.7 ± 0.7	6.8 ± 0.8	72.1 ± 1.8
	EFNN	$\mathbf{54.2 \pm 3.4}$	$\mathbf{66.2 \pm 1.2}$	$\mathbf{72.5 \pm 1.0}$	$\mathbf{80.1 \pm 1.0}$
2 hidden layers 100 neurons ReLU	FNN	7.7 ± 1.3	$\mathbf{79.5 \pm 2.0}$	6.8 ± 0.6	71.1 ± 2.4
	EFNN	$\mathbf{66.8 \pm 2.5}$	79.0 ± 0.6	$\mathbf{70.6 \pm 1.0}$	$\mathbf{83.3 \pm 1.1}$
3 hidden layers 100 neurons Identity	FNN	6.0 ± 0.4	55.4 ± 0.8	6.0 ± 0.5	69.5 ± 3.0
	EFNN	$\mathbf{51.0 \pm 6.1}$	$\mathbf{64.6 \pm 1.3}$	$\mathbf{76.2 \pm 1.1}$	$\mathbf{80.4 \pm 0.8}$
3 hidden layers 100 neurons ReLU	FNN	6.5 ± 0.3	$\mathbf{79.0 \pm 1.8}$	6.4 ± 0.2	68.6 ± 2.6
	EFNN	$\mathbf{56.7 \pm 3.9}$	77.1 ± 1.2	$\mathbf{84.5 \pm 1.2}$	$\mathbf{89.6 \pm 0.8}$

Table 5.40: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on letter data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 100 many neurons and 3 hidden layered networks whose hidden layers contain 100 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	∞	54.4 ± 3.6	66.1 ± 1.6	$\mathbf{72.6 \pm 1.0}$	$\mathbf{80.2 \pm 0.8}$
	1	49.1 ± 4.2	30.3 ± 8.2	72.5 ± 1.0	79.8 ± 0.8
	$1 + \delta^{2^t}$	49.1 ± 3.8	50.4 ± 1.6	72.5 ± 1.0	79.7 ± 0.9
	2	54.3 ± 3.4	66.4 ± 1.7	$\mathbf{72.6 \pm 0.9}$	80.0 ± 1.1
	10	$\mathbf{54.6 \pm 3.3}$	$\mathbf{66.8 \pm 1.3}$	$\mathbf{72.6 \pm 0.9}$	80.0 ± 1.1
	100	54.3 ± 3.4	66.7 ± 1.2	$\mathbf{72.6 \pm 0.9}$	80.0 ± 1.1
2 hidden layers 100 neurons ReLU	∞	66.6 ± 2.4	78.0 ± 1.2	$\mathbf{70.8 \pm 0.8}$	83.4 ± 0.7
	1	55.9 ± 4.1	29.7 ± 9.4	68.8 ± 0.9	77.5 ± 1.5
	$1 + \delta^{2^t}$	55.9 ± 1.9	59.6 ± 8.4	68.7 ± 0.9	77.4 ± 2.5
	2	66.5 ± 2.4	$\mathbf{79.0 \pm 0.6}$	70.5 ± 1.0	$\mathbf{83.5 \pm 0.9}$
	10	$\mathbf{67.2 \pm 2.2}$	78.8 ± 0.6	70.5 ± 1.0	83.2 ± 1.0
	100	66.8 ± 2.5	78.9 ± 0.5	70.5 ± 1.0	83.2 ± 0.9
3 hidden layers 100 neurons Identity	∞	42.3 ± 2.8	63.2 ± 1.7	$\mathbf{76.3 \pm 1.1}$	80.3 ± 0.8
	1	12.1 ± 2.0	11.3 ± 1.3	76.1 ± 1.1	78.4 ± 2.6
	$1 + \delta^{2^t}$	31.5 ± 4.4	12.7 ± 0.9	76.2 ± 1.1	80.0 ± 0.8
	2	$\mathbf{51.0 \pm 6.1}$	$\mathbf{64.6 \pm 1.1}$	76.1 ± 1.2	$\mathbf{80.4 \pm 0.8}$
	10	42.5 ± 2.8	64.1 ± 1.5	76.1 ± 1.2	80.2 ± 0.9
	100	42.6 ± 2.9	64.0 ± 1.4	76.1 ± 1.2	80.3 ± 0.9
3 hidden layers 100 neurons ReLU	∞	26.9 ± 1.8	77.5 ± 1.2	$\mathbf{84.6 \pm 1.1}$	$\mathbf{89.5 \pm 0.7}$
	1	27.9 ± 8.6	$\mathbf{9.1 \pm 0.3}$	75.3 ± 4.3	83.3 ± 5.6
	$1 + \delta^{2^t}$	22.6 ± 8.6	15.4 ± 1.7	73.9 ± 5.7	86.1 ± 2.0
	2	$\mathbf{56.7 \pm 3.9}$	75.6 ± 2.2	84.4 ± 1.1	89.3 ± 0.8
	10	26.6 ± 1.6	75.0 ± 1.9	84.5 ± 1.1	89.3 ± 0.9
	100	26.2 ± 1.3	75.2 ± 2.1	$\mathbf{84.6 \pm 1.2}$	$\mathbf{89.5 \pm 0.8}$

In addition to Table 5.39, Table 5.40 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances

computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and SGD optimization methods. The network topologies presented in Table 5.40 are 2 hidden layers each of which containing 100 neurons and 3 hidden layers each of which contains 100 neurons, similar to Table 5.39.

Table 5.41: The number of different hyper-parameter sets used on trials given in Figure 5.9

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	4	31
1	10^{-4}	4	8
$1+\delta^{2^t}$	10^{-4}	4	12
2	10^{-4}	4	22
10	10^{-4}	4	23
100	10^{-4}	4	24

Lastly, Figure 5.9 and Table 5.41 presented. Figure 5.9 shows the change of average cross entropy with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 3 hidden layers each of which contains 100 many neurons, and ReLU activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.40. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.9 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.41 instead of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

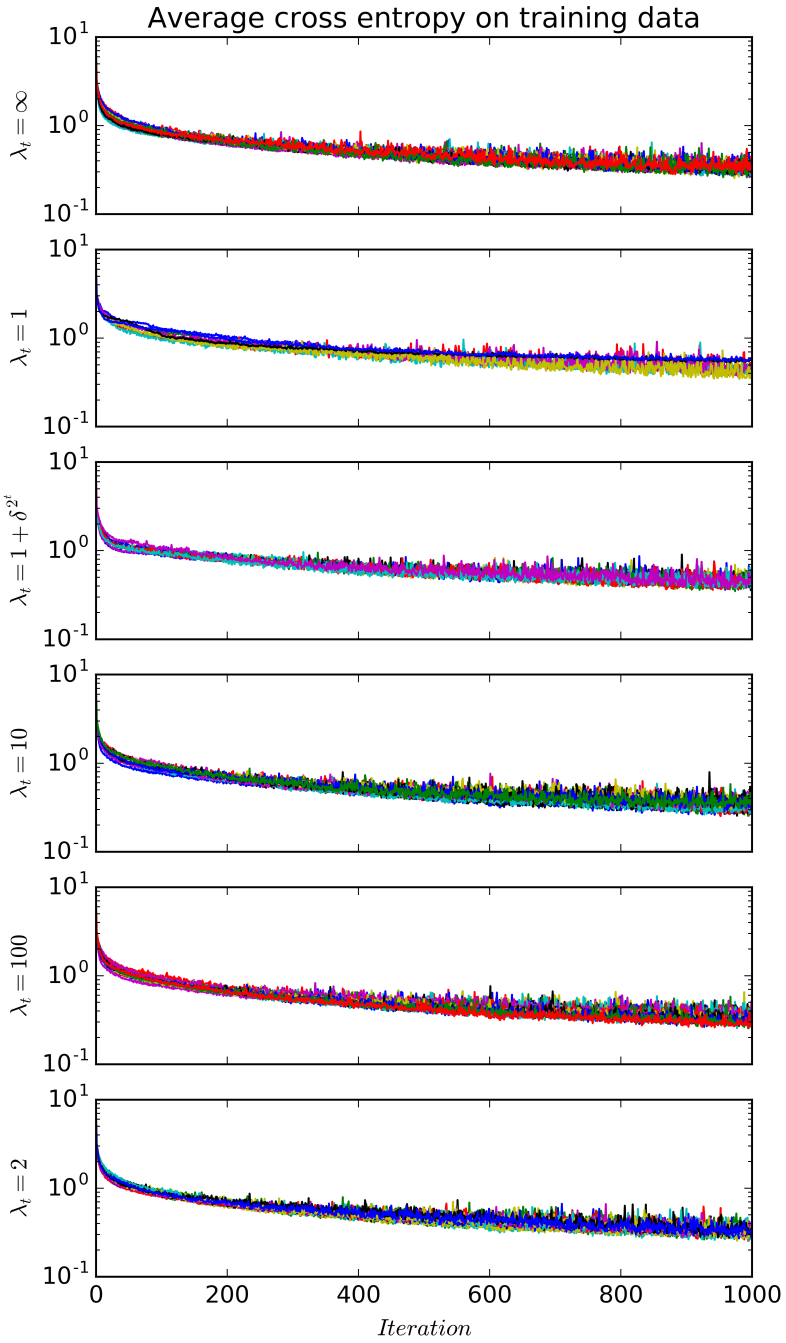


Figure 5.9: The change of average cross entropy with respect to iteration for trials achieving maximal classification performance on training data when standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 3 hidden layered EFNNs whose hidden layers consist of 100 neurons with ReLU activation function over letter recognition data set.

5.3.3.8 Wine Data Set

Table 5.42 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on wine data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.42 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.42 contains two different network topologies which are 2 hidden layers each of which contains 100 neurons and 3 hidden layers each of which contains 50 neurons.

Table 5.42: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 100 many neurons, and 3 hidden layered networks whose hidden layers contain 50 many neurons employing identity and ReLU activation functions trained on wine data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	FNN	66.9 ± 4.4	$\mathbf{95.5 \pm 3.3}$	65.8 ± 3.5	$\mathbf{93.8 \pm 4.4}$
	EFNN	$\mathbf{67.2 \pm 7.9}$	90.4 ± 5.7	$\mathbf{82.8 \pm 8.0}$	90.7 ± 5.2
2 hidden layers 100 neurons ReLU	FNN	66.1 ± 6.0	$\mathbf{92.2 \pm 4.1}$	63.4 ± 6.7	$\mathbf{92.4 \pm 4.7}$
	EFNN	$\mathbf{72.1 \pm 8.6}$	89.3 ± 6.0	$\mathbf{75.8 \pm 6.3}$	89.2 ± 5.9
3 hidden layers 50 neurons Identity	FNN	61.8 ± 6.6	$\mathbf{93.4 \pm 3.9}$	52.8 ± 1.9	$\mathbf{93.1 \pm 4.4}$
	EFNN	$\mathbf{65.3 \pm 12.8}$	65.1 ± 6.2	$\mathbf{67.8 \pm 11.4}$	89.1 ± 5.0
3 hidden layers 50 neurons ReLU	FNN	64.3 ± 6.0	$\mathbf{92.5 \pm 4.3}$	61.9 ± 7.0	$\mathbf{90.9 \pm 7.6}$
	EFNN	$\mathbf{81.0 \pm 6.7}$	71.0 ± 3.9	$\mathbf{87.7 \pm 8.8}$	89.0 ± 7.4

In addition to Table 5.42, Table 5.43 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm

Table 5.43: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on wine data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 100 many neurons and 3 hidden layered networks whose hidden layers contain 50 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	∞	57.3 ± 6.6	89.7 ± 5.9	67.8 ± 8.6	91.1 ± 5.1
	1	65.8 ± 7.9	55.6 ± 2.3	67.7 ± 8.8	63.7 ± 8.7
	$1 + \delta^{2^t}$	66.5 ± 8.3	59.1 ± 7.1	$\mathbf{83.3 \pm 8.2}$	65.6 ± 5.5
	2	$\mathbf{66.6 \pm 8.4}$	67.8 ± 9.7	70.4 ± 10.9	$\mathbf{92.1 \pm 4.5}$
	10	54.5 ± 2.7	89.8 ± 6.6	71.8 ± 11.3	90.9 ± 4.5
	100	54.5 ± 2.7	$\mathbf{91.9 \pm 3.5}$	68.4 ± 7.5	91.3 ± 5.1
2 hidden layers 100 neurons ReLU	∞	72.8 ± 7.7	$\mathbf{90.8 \pm 4.9}$	76.7 ± 5.7	90.0 ± 5.8
	1	70.9 ± 5.0	63.5 ± 4.5	69.6 ± 7.7	68.0 ± 9.0
	$1 + \delta^{2^t}$	$\mathbf{73.1 \pm 6.8}$	63.5 ± 4.5	68.1 ± 9.9	71.1 ± 8.5
	2	70.4 ± 9.4	74.4 ± 12.4	$\mathbf{77.1 \pm 6.7}$	90.6 ± 4.4
	10	72.4 ± 8.0	87.3 ± 8.3	75.8 ± 5.0	90.1 ± 5.9
	100	72.9 ± 8.0	86.9 ± 6.4	75.8 ± 4.3	$\mathbf{90.8 \pm 6.2}$
3 hidden layers 50 neurons Identity	∞	66.9 ± 4.7	65.2 ± 4.9	64.6 ± 6.2	89.2 ± 4.2
	1	65.9 ± 2.9	66.6 ± 4.1	68.2 ± 6.9	64.0 ± 3.2
	$1 + \delta^{2^t}$	$\mathbf{68.0 \pm 11.6}$	66.6 ± 4.1	$\mathbf{71.1 \pm 10.2}$	64.3 ± 5.4
	2	64.1 ± 10.7	$\mathbf{66.7 \pm 4.1}$	67.0 ± 9.7	65.8 ± 9.7
	10	66.9 ± 4.7	65.9 ± 3.3	65.2 ± 5.6	$\mathbf{89.9 \pm 4.7}$
	100	66.9 ± 4.7	66.5 ± 6.0	65.1 ± 6.9	88.6 ± 5.0
3 hidden layers 50 neurons ReLU	∞	68.6 ± 5.6	71.3 ± 4.2	81.4 ± 11.1	88.9 ± 7.5
	1	$\mathbf{81.0 \pm 6.7}$	64.6 ± 4.2	$\mathbf{87.1 \pm 8.0}$	68.0 ± 3.1
	$1 + \delta^{2^t}$	69.1 ± 6.1	59.2 ± 5.4	79.4 ± 12.3	67.2 ± 4.1
	2	79.7 ± 9.4	67.3 ± 4.8	70.3 ± 18.4	67.6 ± 4.5
	10	69.7 ± 5.8	63.9 ± 6.7	80.9 ± 8.4	85.5 ± 6.8
	100	69.1 ± 5.7	$\mathbf{75.6 \pm 7.4}$	80.9 ± 8.4	$\mathbf{89.6 \pm 6.3}$

and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and

SGD optimization methods. The network topologies presented in Table 5.43 are 2 hidden layers each of which containing 100 neurons and 3 hidden layers each of which contains 50 neurons, similar to Table 5.42.

Table 5.44: The number of different hyper-parameter sets used on trials given in Figure 5.10

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	1	4
		4	5
1	10^{-4}	1	5
		4	5
$1+\delta^{2^t}$	10^{-4}	1	5
		4	5
2	10^{-4}	1	4
		4	3
	10^{-6}	1	3
		4	3
10	10^{-4}	1	6
		4	2
100	10^{-4}	1	9
		4	6

Lastly, Figure 5.10 and Table 5.44 presented. Figure 5.10 shows the change of average cross entropy with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 3 hidden layers each of which contains 50 many neurons, and ReLU activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.43. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.10 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.44 instead

of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

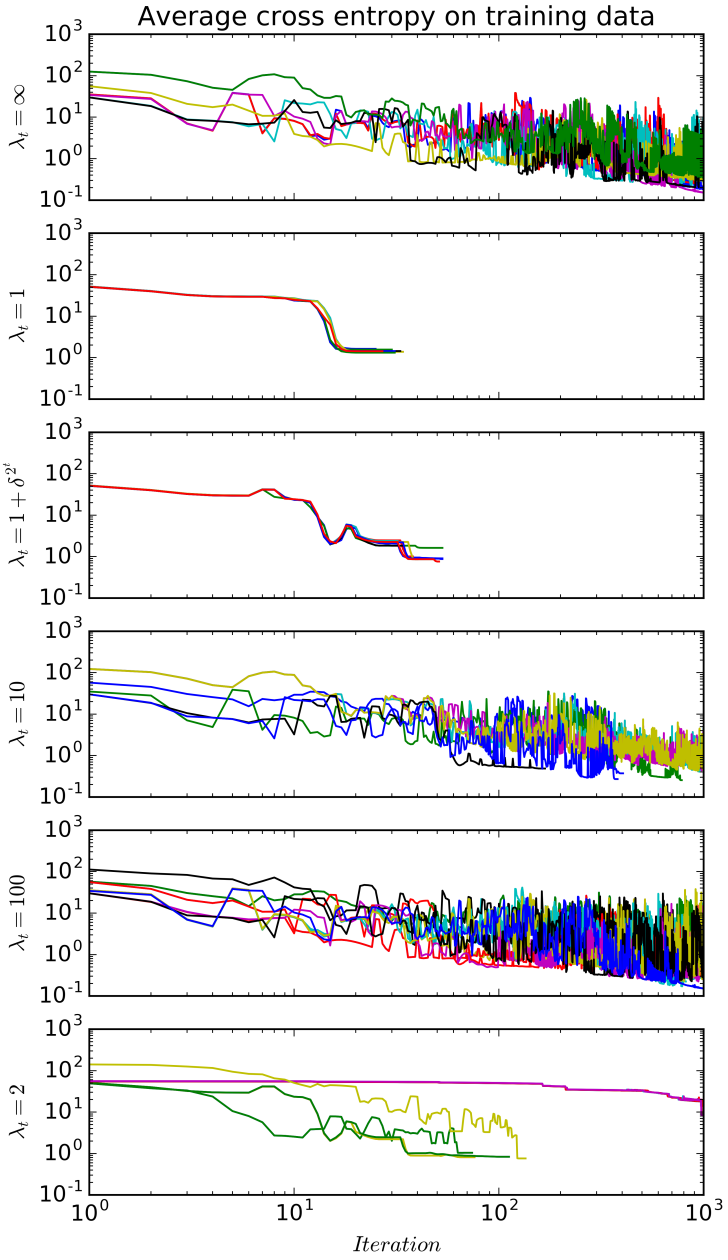


Figure 5.10: The change of average cross entropy with respect to iteration for trials achieving maximal classification performance on training data when standard back-propagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 3 hidden layered EFNNs whose hidden layers consist of 50 neurons with ReLU activation function over wine data set.

5.3.3.9 Yeast Data Set

Table 5.45 presents the average classification accuracies are achieved by feed-forward neural networks (FNN) and ef-operator based neural networks (EFNN) on yeast data set. FNNs trained using only standard backpropagation algorithm, while EFNNs trained with backpropagation with line search (BLS) with different $\boldsymbol{\lambda}$ sequences in addition to standard backpropagation algorithm. The average classification accuracy presented on table 5.45 for different network topology, optimizer, and objective function triplets for EFNNs calculated using results of both standard backpropagation algorithm and BLS algorithm. The table 5.45 contains two different network topologies which are 2 hidden layers each of which contains 100 neurons and 3 hidden layers each of which contains 100 neurons.

Table 5.45: Average test classification accuracy achieved by classic feed-forward neural networks (FNN) and ef operator based neural networks (EFNN) when 2 hidden layered networks whose hidden layers contain 100 many neurons, and 3 hidden layered networks whose hidden layers contain 100 many neurons employing identity and ReLU activation functions trained on yeast data set.

		MSE		Cross entropy	
Network	Architecture	SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	FNN	36.2 ± 2.5	47.2 ± 2.3	$\mathbf{36.0 \pm 2.1}$	41.0 ± 1.4
	EFNN	$\mathbf{37.4 \pm 3.5}$	$\mathbf{56.5 \pm 2.9}$	35.9 ± 2.0	$\mathbf{56.9 \pm 2.1}$
2 hidden layers 100 neurons ReLU	FNN	34.6 ± 2.3	44.0 ± 1.3	35.2 ± 2.1	41.0 ± 2.6
	EFNN	$\mathbf{36.8 \pm 2.9}$	$\mathbf{53.6 \pm 2.9}$	$\mathbf{37.0 \pm 2.0}$	$\mathbf{52.7 \pm 3.1}$
3 hidden layers 100 neurons Identity	FNN	32.6 ± 2.5	42.2 ± 2.3	31.7 ± 1.6	38.7 ± 2.3
	EFNN	$\mathbf{54.9 \pm 3.8}$	$\mathbf{57.3 \pm 2.3}$	$\mathbf{54.2 \pm 3.4}$	$\mathbf{57.0 \pm 2.2}$
3 hidden layers 100 neurons ReLU	FNN	34.2 ± 2.0	44.7 ± 1.4	33.1 ± 1.5	43.8 ± 1.2
	EFNN	$\mathbf{55.6 \pm 3.5}$	$\mathbf{57.1 \pm 2.5}$	$\mathbf{36.6 \pm 3.4}$	$\mathbf{58.4 \pm 1.9}$

Table 5.46: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on yeast data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 100 many neurons and 3 hidden layered networks whose hidden layers contain 100 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 100 neurons Identity	∞	**37.4 ± 3.3**	**56.9 ± 2.3**	**35.9 ± 2.0**	56.8 ± 2.2
	1	**37.4 ± 3.5**	55.3 ± 2.6	**35.9 ± 2.0**	56.7 ± 4.2
	$1 + \delta^{2^t}$	**37.4 ± 3.5**	56.4 ± 3.4	**35.9 ± 2.0**	56.4 ± 3.6
	2	**37.4 ± 3.3**	55.2 ± 3.5	**35.9 ± 2.0**	56.8 ± 2.1
	10	**37.4 ± 3.3**	55.4 ± 3.4	**35.9 ± 2.0**	**57.0 ± 2.3**
	100	**37.4 ± 3.3**	55.0 ± 3.4	**35.9 ± 2.0**	**57.0 ± 2.1**
2 hidden layers 100 neurons ReLU	∞	36.4 ± 2.0	54.1 ± 3.7	**37.6 ± 2.5**	52.8 ± 3.1
	1	35.2 ± 2.9	55.8 ± 2.9	37.3 ± 2.6	50.9 ± 3.0
	$1 + \delta^{2^t}$	35.3 ± 3.1	**56.6 ± 3.7**	37.0 ± 2.9	51.7 ± 4.0
	2	36.8 ± 3.1	54.9 ± 3.1	37.4 ± 2.9	53.7 ± 3.6
	10	**37.1 ± 3.2**	55.5 ± 1.7	37.4 ± 2.8	54.4 ± 4.1
	100	36.7 ± 3.1	53.2 ± 3.1	37.1 ± 2.7	**54.8 ± 3.5**
3 hidden layers 100 neurons Identity	∞	55.3 ± 3.3	56.7 ± 2.5	54.3 ± 3.3	57.5 ± 2.9
	1	54.6 ± 3.7	**56.8 ± 3.0**	**54.5 ± 3.3**	55.6 ± 3.5
	$1 + \delta^{2^t}$	54.2 ± 3.0	54.3 ± 3.1	54.4 ± 3.4	55.8 ± 2.8
	2	**55.4 ± 3.4**	55.4 ± 3.7	54.0 ± 3.5	57.6 ± 2.9
	10	55.0 ± 3.3	55.1 ± 3.3	54.0 ± 3.4	58.0 ± 2.8
	100	55.3 ± 3.3	55.0 ± 3.5	54.0 ± 3.4	**58.4 ± 2.8**
3 hidden layers 100 neurons ReLU	∞	54.9 ± 2.9	57.3 ± 2.1	**38.3 ± 3.2**	57.3 ± 2.4
	1	55.8 ± 2.2	56.6 ± 2.5	37.5 ± 3.9	56.4 ± 3.7
	$1 + \delta^{2^t}$	55.1 ± 1.7	57.1 ± 3.4	37.9 ± 3.5	57.3 ± 3.3
	2	55.7 ± 3.0	56.0 ± 2.8	36.8 ± 3.5	**58.8 ± 1.8**
	10	**55.9 ± 3.0**	**57.8 ± 2.4**	37.1 ± 3.4	57.1 ± 3.1
	100	55.5 ± 3.4	56.1 ± 2.6	36.8 ± 3.4	57.2 ± 2.6

In addition to Table 5.45, Table 5.46 is given which presents the average classification performances achieved by EFNNs trained with standard backpropagation algorithm and BLS algorithm using different $\boldsymbol{\lambda}$ sequences. Average classification performances

computed by employing ReLU and identity activation functions trained with mean squared error (MSE) and average cross entropy objective functions using ADAM and SGD optimization methods. The network topologies presented in Table 5.46 are 2 hidden layers each of which containing 100 neurons and 3 hidden layers each of which contains 100 neurons, similar to Table 5.45.

Table 5.47: The number of different hyper-parameter sets used on trials given in Figure 5.11

λ_t	μ	Mini Batch Count	Trial Count
∞	10^{-4}	1	11
		4	2
1	10^{-4}	4	7
$1+\delta^{2^t}$	10^{-4}	4	5
2	10^{-4}	1	6
		4	1
10	10^{-4}	1	7
		4	1
100	10^{-4}	1	7
		4	2

Lastly, Figure 5.11 and Table 5.47 presented. Figure 5.11 shows the change of mean squared error with respect to iterations, when the training trials are done with standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences to train EFNNs with network topology consisting 3 hidden layers each of which contains 100 many neurons, and ReLU activation function using ADAM optimization method. These trials achieve maximal training performances when training the EFNNs using standard backpropagation algorithm and BLS algorithm with different $\boldsymbol{\lambda}$ sequences. Also, these trials are used to compute average classification accuracy presented at Table 5.46. Trials for a specific $\boldsymbol{\lambda}$ sequences differ by the hyper-parameters of optimization method and initial parameter set. Figure 5.11 contains a lot of training trials resulting hard to read legend. Consequently, we decide to present Table 5.47 instead of a legend which presents the number of trials employing specific hyper-parameter set for each $\boldsymbol{\lambda}$ sequences.

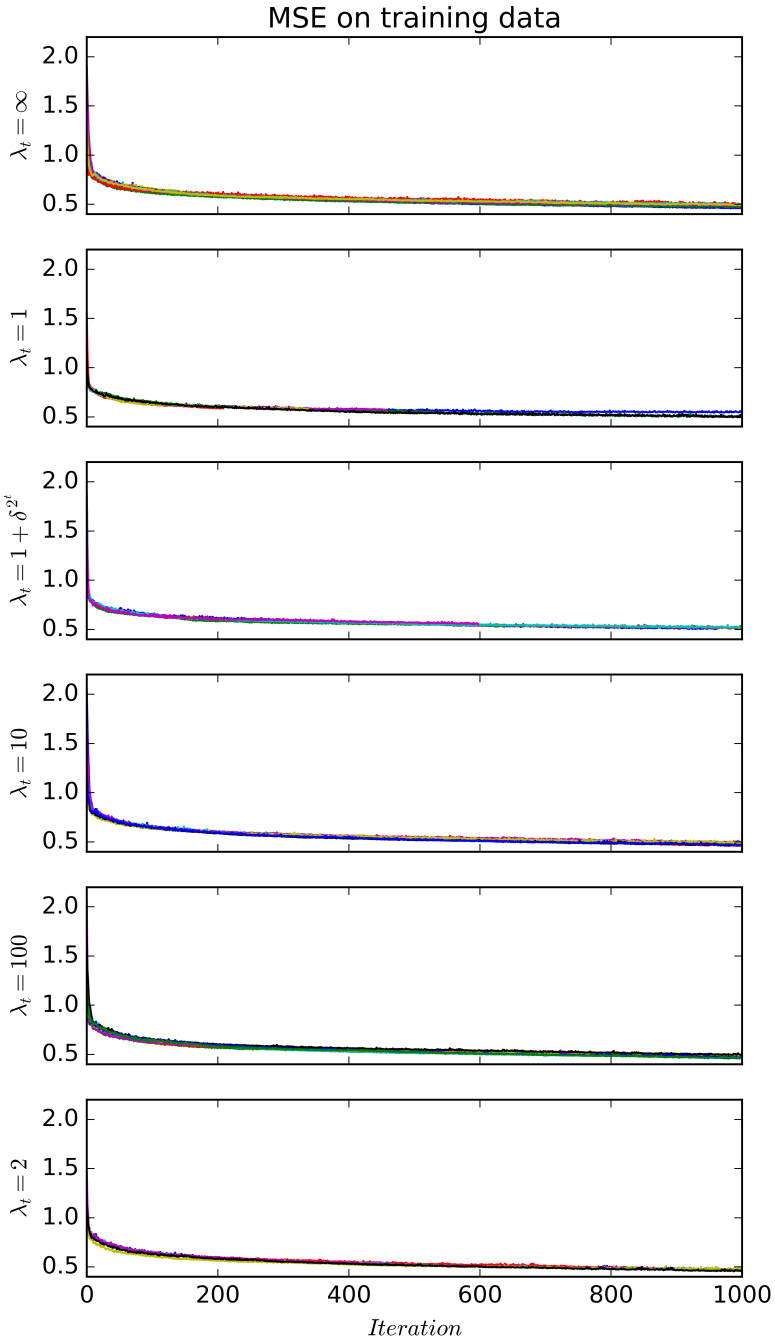


Figure 5.11: The change of mean squared error with respect to iteration for trials achieving maximal classification performance on training data when standard back-propagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, and $\lambda_t = 100$, and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 1 - 10^{-8}$, algorithms with ADAM are used for training 3 hidden layered EFNNs whose hidden layers consist of 100 neurons with ReLU activation function over yeast data set.

5.3.4 Discussion

Experiments showed that proposed version of backpropagation algorithm can match the classification accuracies of standard backpropagation algorithm depending on choice of $\boldsymbol{\lambda}$ sequence. Results indicates that proposed version fails mostly either when ADAM optimizer used or when $\boldsymbol{\lambda}$ values close to 1 used. Change of loss function graphics, figures 5.5, 5.6, 5.7, 5.8, 5.4, 5.10, indicates that when $mbda\boldsymbol{lambda}$ values close to 1, decrease on loss function is greater than standard backpropagation. However, these sequences cause algorithm to converge a non-desired local optimum, decreasing loss too quickly and preventing update to neighbor partitions which can lead to better optima points. The problem may solved using different initial parameters. However, this may not solve the problem unless initial parameters sampled from different distributions. Experiments showed that this problem can also be solved with usage constants greater than 1 instead of exponentially decaying sequences. Failure of the algorithm when ADAM optimizer is used can be explained by the fact that implementation of ADAM optimizer used on training trials is not aware of the line search step. ADAM method relies previous gradients to estimate new update values including magnitude of previous gradients as they affect the first and second moments. However, the line search step changes the magnitude of the update invalidating the moments calculated by the ADAM method using previous gradients. This is especially true, when exponentially decaying sequences or sequence of 1s are used, as most update requires shrinkage on update magnitude to prevent loss function from increasing. This problem can be solved with modification on ADAM method. However, necessary calculations are not available at the moment.

Comparisons between average classification performances achieved by FNNs and EFNNs indicates that EFNNs might be more suitable for these data sets than FNNs. However, parameters used for training FNNs are selected using performances of EFNNs. Consequently, FNNs may not be trained properly. FNNs should be trained with more hyper-parameters, before a conclusion can be drawn.

5.4 MNIST Data Set

MNIST data set is a popular data set used on testing ANNs. It consists of 70,000 hand-written digit images divided as 60,000 training sample and 10,000 test sample. Each image in the dataset is a 28x28 gray scale image with intensities represented with real numbers in the range [0, 1]. Images converted to 1-d vectors of size 784 for experiments. Class distribution in training samples and test samples given in table 5.48. The state of the art classification accuracy on the MNIST data set is 99.79%. This classification accuracy achieved by the DropConnect, a special type of ANN [86].

Table 5.48: Class distribution for mnist data set

Class index	Count in traning data	Count in test data
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009

5.4.1 Experimental Design

MNIST is a multi-class classification problems. This classification problem formulated as the function approximation from feature vectors to one-hot-vector-encoding of class labels. Experiments for MNIST data sets consists of training trials which are performed using multiple network topologies, activation functions, optimizers and loss functions,to approximate these functions. These experiments uses same parameters as experiments for UCI data sets except topologies and number of mini-batches. The selected topologies are limited to following for these training trials,

- 2 hidden layers each of which containing 500 neurons,
- 2 hidden layers each of which containing 800 neurons,
- 3 hidden layers each of which containing 800 neurons.

Number of mini-batches selected from 1 and 60 instead of 1 and 4. Experiments compares the standard backpropagation algorithm and BLS using different $\boldsymbol{\lambda}$ sequences. Five different $\boldsymbol{\lambda}$ sequences used for evaluation. 4 of 5 different $\boldsymbol{\lambda}$ sequences are constants and other one is exponentially decaying sequence. Constant values are 1, 2, 10, and 100. The constant value 1 does not allow any increase on loss, which results strict decrease on loss, if no mini-batch is used. The constant value 2 allows small increments on loss, which may allow algorithm to explore more regions during any part of the training without allowing divergence through prevention of visiting "bad" partitions. The constant value 10 is more relaxed version and the constant value 100 should behave almost same as standard backpropagation in most situations. Exponentially decaying sequence employed is

$$\lambda_t = 1 + \delta^{2^t}, \tag{5.11}$$

where δ is $1 - 10^{-8}$. The exponentially decaying sequence satisfies the restriction

$$\lim_{T\to\infty} \prod_{t=1}^{T} \lambda_t, \tag{5.12}$$

is exists and finite whenever $\delta < 1$ as

$$1 - \delta^{2^{N+1}} = (1-\delta) \prod_{t=0}^{N} (1 + \delta^{2^t}), \tag{5.13}$$

for every $N \in \mathbb{Z}^+$. Also, sequence consisting of only 1s satisfies the restriction. Although, other three sequences do not satisfy the restriction, violation of the restriction is not an issue as training trials done for at most 1,000 iterations.

Similar to UCI data sets, two activation functions used with these networks which are identity and ReLU functions. Resulting EFNNs trained with respect to mean square error and average cross entropy loss functions using ADAM and SGD optimizers. If

average cross entropy used as loss function, then outputs of EFNNs normalized to interval [0, 1] using softmax function.

Hyper-parameters for SGD are learning rate and number of mini-batches and hyper-parameters for ADAM are learning rate, number of mini-batches, β_1, β_2, and ε. β_1, β_2 and ε is not optimized, instead default values, suggested by original authors [48] used which are 0.9, 0.99, and 10^{-8}, respectively. Learning rates for optimizers are selected from 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10} using classification accuracy over training data. 10 randomly sampled initial parameter sets are selected for each network topology using following normal distributions

$$\mathcal{N}\Big(0,\ \frac{2}{n_{in} + n_{out}}\Big), \tag{5.14}$$

and

$$\mathcal{N}(0,\ 10^{-4}), \tag{5.15}$$

where n_{in} is number of neurons in the previous layer, and n_{out} is number of neurons in the current layer as suggested by by Glorot et al. [33]. Weights are sampled independently from the distribution (5.14) and biases are sampled independently from the distribution (5.15).

Full grid-search over hyper-parameters is done with two different initial points for standard backpropagation algorithm to determine best working network architecture, activation function, loss function and optimizer combinations. Among these trials 100 of them are selected for further trials according to average maximal training accuracy achieved over two initial parameter sets including all network topology, activation function, loss function and optimizer quadruples, due to time and resource constraints. Other initial parameters are tested using configurations of these trials for standard backpropagation algorithm. BLS with different $\boldsymbol{\lambda}$ sequences are also tested using configurations of these 100 trials over 10 initial parameter sets.

5.4.2 Results

Table 5.49 presents average test accuracy obtained from different trials by each backpropagation version using different hyper-parameters and initial parameters for each

network topology, activation function, optimizer and loss function quadruple. Trials selected according to maximum training performance achieved during training. The selection done as follow, maximal training accuracy computed for each trial. Afterwards, maximum value, M_i, among these accuracies selected for each loss function, network topology, activation function and optimizer quadruple. The trials achieving maximal training accuracy at least $0.99M_i$ included in the calculation of averages. The test accuracy for each of these trials, taken from the iteration where maximal training accuracy achieved. If there exists more than one such iteration, then the iteration with lowest training loss is taken. Networks with 2 hidden layers each containing 500 neurons omitted as their performance was inferior to networks with 2 hidden layers each containing 800 neurons from Table 5.49.

Figure 5.12 shows the change of MSE during training of one particular triplet consisting ADAM optimizer, ReLU activation function and topology consisting 2 hidden layers each containing 800 neurons for each backpropagation version. Graph includes only the trials selected for Table 5.49.

5.4.3 Discussion

The experiments for MNIST data set, unlike the experiments for UCI data sets, indicate very similar test accuracies to standard backpropagation. In fact, BLS performs slightly better than standard backpropagation algorithm. However, effects of selection of $\boldsymbol{\lambda}$ over different network topologies is not consistent. Figure 5.12 shows that change of MSE on training data is very similar on each version. The loss initially decreases and achieves a minimum value around 200th iteration, before MSE begins to fluctuate. Fluctuations exists, even when $\boldsymbol{\lambda}$ taken as constant 1, due to usage of mini-batches. However, one important issue to notice that hyper-parameters and initial parameters used on trials of BLS are selected according to the performances of trials with standard backpropagation. Consequently, standard backpropagation expected to work well with these parameters, while BLS could have failed. BLS may also work with other parameters where standard backpropagation failed. However, this claim requires further experiments for verification.

Table 5.49: Test classification accuracy achieved by standard backpropagation, $\lambda_t = \infty$, BLS with constant λs, $\lambda_t = 1$, $\lambda_t = 2$, $\lambda_t = 10$, $\lambda_t = 100$ and BLS with exponentially decaying λ, $\lambda_t = 1 + \delta^{2^t}$ where $\delta = 10^{-8}$ on MNIST data set, while optimizing mean squared error (MSE) and cross entopy using ADAM and SGD optimizers with ReLU and identity activation functions for 2 hidden layered networks whose hidden layers contain 800 many neurons and 3 hidden layered networks whose hidden layers contain 800 many neurons.

Network	λ_t	MSE		Cross entropy	
		SGD	ADAM	SGD	ADAM
2 hidden layers 800 neurons Identity	∞	87.9 ± 0.1	91.0 ± 0.2	**92.5 ± 0.0**	93.3 ± 0.2
	1	87.8 ± 0.1	**91.1 ± 0.2**	**92.5 ± 0.0**	93.2 ± 0.2
	$1 + \delta^{2^t}$	87.8 ± 0.1	91.0 ± 0.2	**92.5 ± 0.0**	93.3 ± 0.2
	2	**88.0 ± 0.1**	91.0 ± 0.2	**92.5 ± 0.0**	93.3 ± 0.2
	10	**88.0 ± 0.1**	91.0 ± 0.2	**92.5 ± 0.0**	**93.4 ± 0.2**
	100	**88.0 ± 0.1**	91.0 ± 0.2	**92.5 ± 0.0**	**93.4 ± 0.2**
2 hidden layers 800 neurons ReLU	∞	**97.3 ± 0.0**	98.3 ± 0.2	**95.2 ± 0.1**	**97.8 ± 0.2**
	1	**97.3 ± 0.1**	**98.6 ± 0.0**	**95.2 ± 0.0**	**97.8 ± 0.1**
	$1 + \delta^{2^t}$	**97.3 ± 0.1**	**98.6 ± 0.0**	**95.2 ± 0.0**	**97.8 ± 0.1**
	2	**97.3 ± 0.1**	98.3 ± 0.1	**95.2 ± 0.0**	97.6 ± 0.4
	10	**97.3 ± 0.1**	98.3 ± 0.2	**95.2 ± 0.0**	97.7 ± 0.2
	100	**97.3 ± 0.1**	98.3 ± 0.2	**95.2 ± 0.0**	97.7 ± 0.3
3 hidden layers 800 neurons Identity	∞	88.0 ± 0.1	90.7 ± 0.1	**92.7 ± 0.1**	**92.7 ± 0.2**
	1	**90.4 ± 0.1**	**91.1 ± 0.4**	**92.7 ± 0.1**	**92.7 ± 0.2**
	$1 + \delta^{2^t}$	87.9 ± 0.1	**91.1 ± 0.2**	**92.7 ± 0.1**	**92.7 ± 0.1**
	2	87.9 ± 0.1	90.7 ± 0.2	**92.7 ± 0.1**	**92.7 ± 0.3**
	10	87.9 ± 0.1	90.7 ± 0.2	**92.7 ± 0.1**	**92.7 ± 0.3**
	100	87.9 ± 0.1	90.7 ± 0.2	**92.7 ± 0.1**	**92.7 ± 0.3**
3 hidden layers 800 neurons ReLU	∞	**98.4 ± 0.0**	97.7 ± 0.6	**98.2 ± 0.1**	**98.0 ± 0.3**
	1	97.6 ± 0.2	97.9 ± 0.3	**98.2 ± 0.0**	**98.0 ± 0.3**
	$1 + \delta^{2^t}$	97.8 ± 0.1	97.9 ± 0.3	**98.2 ± 0.0**	**98.0 ± 0.3**
	2	98.3 ± 0.1	**98.0 ± 0.4**	**98.2 ± 0.1**	**98.0 ± 0.4**
	10	98.3 ± 0.1	97.9 ± 0.4	**98.2 ± 0.1**	**98.0 ± 0.3**
	100	98.3 ± 0.1	97.7 ± 0.5	**98.2 ± 0.1**	**98.0 ± 0.3**

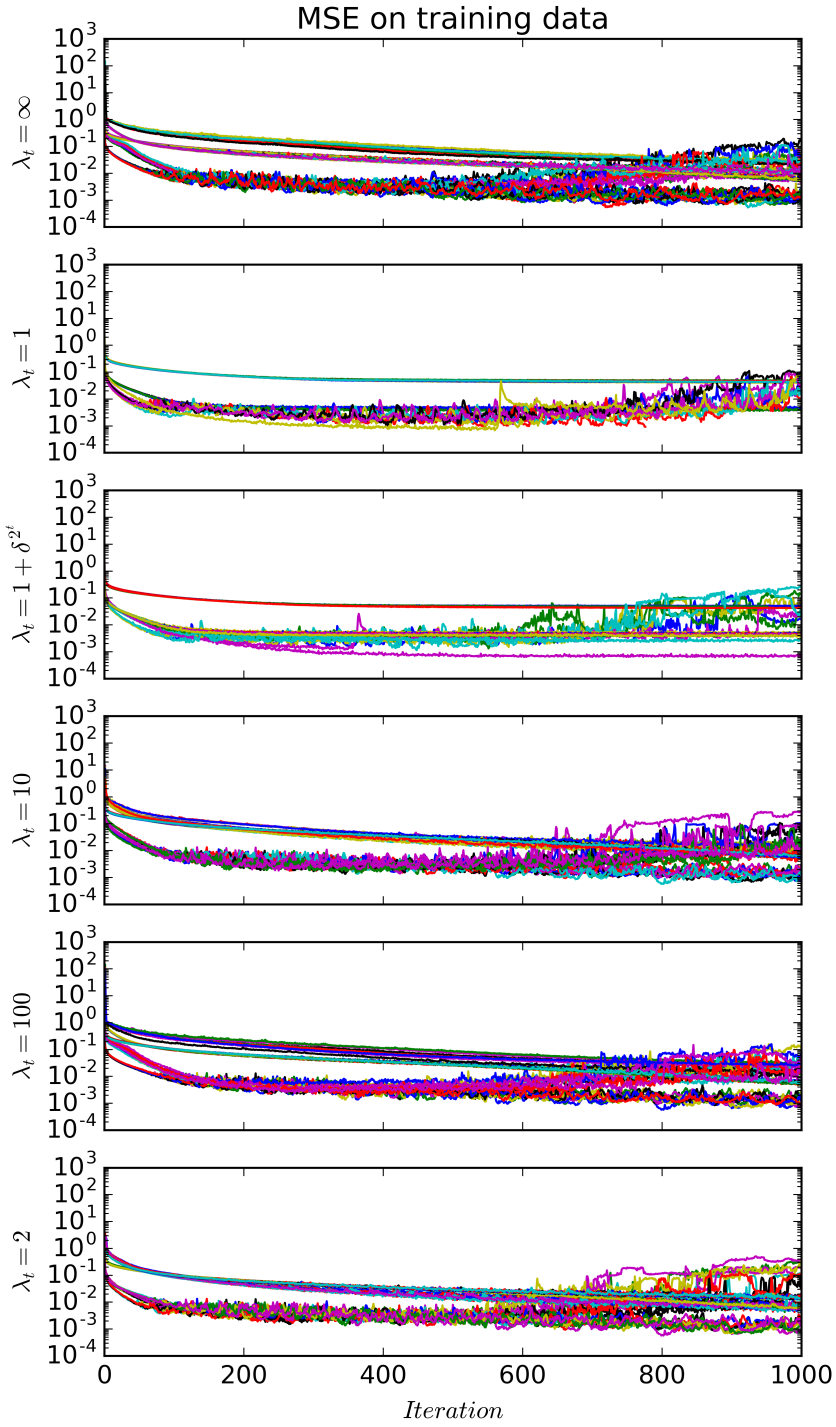


Figure 5.12: The change of mean squared error on training data with respect to iterations for trials with maximal classification accuracy on training data with standard backpropagation using 2 hidden layers, each of which containing 800 neurons. ReLU activation function and ADAM optimizer is used for MNIST data set.

5.5 Chapter Summary

Standard backpropagation algorithm and backpropagation with line search algorithm (BLS) with various $\boldsymbol{\lambda}$ sequences are empirically compared in this chapter. Experiments are done over linear function learning problem which is discussed in Section 4.1, XOR problem, 9 multi-class classification tasks from UCI data sets and MNIST data set to compare the performances of EFNN with classical backpropagation and BLS. Experiments verified that proposed algorithm converges. However, this can be a limiting factor especially when exponentially decaying sequences or constant 1 are used. Experiments showed that these kinds of sequences limits the training procedure to few partitions causing early termination. The early terminations may fail the training of neural networks. Experiments also showed that this problem can be solved using constant $\boldsymbol{\lambda}$ sequences greater than 1. Usage of relatively small constants may sometimes fail to train. However, they are usually more successful than sequences which do not allow increase on loss in later iterations. Experiments also verified that, BLS becomes exactly like standard algorithm when larger values of $\boldsymbol{\lambda}$ are used, as expected.

This chapter also includes a brief comparison between performances of feed forward neural networks (FNN) and ef-operator based neural networks (EFNN). Comparisons are done over the selected UCI data sets and comparisons indicates that EFNNs can achieve similar classification performance to FNNs, when the same number of neurons and hidden layers are used. In fact, our experiments indicate that EFNNs might perform slightly better than FNNs in some cases.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis concerns with neural network architectures based on a multiplication free implementable operator, called ef operator. Multiplication free implementable nature of the network has a potential to provide energy efficiency, and makes the network suitable for systems where energy is a limited resource. Previous works on the ef operator mostly concerned with applicability of the operator to image and signal processing problems. Some preliminary studies [5, 4] applied ef-operators to neural networks. Initial study [5] failed to attain state-of-the art classification accuracy with 10% loss on classification accuracy. A recent study in [4] reports improved results of [5] and attained near state-of-the art classification accuracies. Ef-operator and methods relying on ef-operator become viable option for an energy efficient replacement of artificial neural networks in light of the results of [5] and [4]. These studies simply replaces the inner product the argument of the activation functions and applied the standard training methodologies of classical neural networks. Although, the suggested training methodologies work in the practice, they have a fundamental problems which may make impossible to use these networks on more complicated data sets. One of the major problems emerge from the non-differentiable nature of ef-operator. Ef-operator introduces a jump when input is changed from a positive valued number to a negative valued number, which causes a sudden change on computed mappings in small neighborhoods. This thesis inspects the dependency between the

parameters and mappings computed by the suggested networks, and proposed a possible solution to solve the discontinuity problem.

In this thesis, A new neural network, called ef-operator based neural network (EFNN) based on ef-operator is proposed. The suggested network is based on the already existing architecture based on ef-operator, called additive neural networks [4]. Additive neural networks [4] introduced scaling parameters to approximate arbitrary Lebesgue integrable functions. Scaling parameters requires multiplication on each hidden layers which decreases the efficiency gained by avoidance of classic vector multiplication. EFNNs use the classic vector multiplication in the first and last hidden layers, and ef-operator on remaining hidden layers. This architecture allows EFNNs to approximate any Lebesgue integrable function with a right choice of activation function and energy efficiency increases with the increasing number of hidden layers. The suggested design is more suitable for deeper networks than additive neural networks. The representation capabilities of EFNNs is inspected specifically for identity and ReLU activation functions. We show that if sufficiently large number of neurons are used, then any Lebesgue integrable function can be approximated arbitrarily well. The multiplication free implementable nature of ef operator and representation power of EFNNs with simple activation functions, such as, identity and ReLU, allows EFNNs to be energy efficient without usage of low precision floating points and approximations. In addition to, representation capabilities of EFNNs, the relation between parameters of EFNNs and resulting mappings analyzed. Our analyze showed that parameter space can be splited into partitions, so that mappings computed in each partition computes similar functions with respect to the input data. We showed that, if the activation function satisfies a weaker version of linearity which we call sign-wise linearity, then these partitions becomes convex. Both identity and ReLU functions which are our focus as activation function, satisfies sign-wise linearity conditions. Furthermore, output of the last hidden layer immediately before output layer depends linearly on the parameters at input data. Also, our inspection showed that mappings computed in neighbor partitions can be drastically different. This observation combined with the observation of some of these partitions are not closed, presents a drawback for convergence during the training. If local optimum

located on the boundary, then it is not attainable and slightly large updates can result change of partition causing drastic changes on loss. Sudden increase on the loss can cause divergence in backpropagation algorithm as sudden increase caused by drastic change on computed mapping which can also result in drastic change on directions of derivatives.

We propose a modified version of backpropagation algorithm, called backpropagation with line search, (BLS). BLS algorithm makes a line search in the direction of update vector to find an update value whose loss is bounded by $\lambda_t \mathrm{L}_t$ where L_t is current loss. The convergence and performance of BLS algorithm compared experimentally with standard backpropagation algorithm, in Chapter 5, using various data sets and various choice of $\boldsymbol{\lambda}$ sequences. Experiments show that effectiveness of suggested method heavily depends on the problem, architecture as well as the choice of optimizer. Suggested method is capable of matching the performances achieved by standard backpropagation in most cases, even though experiments biased toward the standard backpropagation algorithm. However, there are cases, where the suggested method cannot train EFNNs properly. Experiments showed that training failures usually happens when small $\boldsymbol{\lambda}$ values are used in conjunction with ADAM optimizer. When BLS algorithm fail to train EFNNs, algorithm usually terminates very early around 100^{th} iteration. The early termination of BLS algorithm in case of failure to train, allows us to try more parameters instead of waiting a bad result. If decaying $\boldsymbol{\lambda}$ values fail to train EFNN, then small constant $\boldsymbol{\lambda}$ values can be used instead. Usage of small constant $\boldsymbol{\lambda}$ values permits training procedure to explore other regions, always allowing to it to reach more optimal points.

6.2 Future Work

Ef operator is a relatively new operator and usage of it in the context of the neural networks was very limited until recently. This work suggests and analyzes a neural network architecture based on ef-operator, called EFNN. However, there are many theoretical and practical questions remained to be answered. We showed that EFNNs

are capable of approximating Lebesgue integrable functions for specific activation functions. However, we do not know the representation capabilities of EFNNs in case of other activation functions. Also, we manage to show the representation capabilities of EFNNs for 4 or more hidden layered architectures. 2 hidden layered architectures do not use ef-operator, so we are not interested on them, but 3 hidden layered architectures requires more attention. There are two important questions to be answered regarding the representation capabilities of EFNNs. These questions are, can EFNNs approximate continuous functions arbitrarily well with a suitable choice of the activation function, and how much neuron is necessary to approximate a function with given degree of error. Practical concerns include the comparison between performances of EFNNs and performances of other efficient neural networks, improving training methods to estimate better local minima and generalizing the EFNN to different neural network architectures, such as convolutional networks and recurrent neural networks.

This thesis experimentally evaluated the proposed training algorithms. However, data sets used on the experiments were relatively small and simple. Experiments should be repeated with more complex data sets to demonstrate effectiveness of proposed algorithm. Also, experiments on this thesis solely focused on comparison between standard backpropagation algorithm and BLS. Future works may include experimental comparisons between EFNNs and other energy efficient neural network variants, such as, neural networks employing binary weights and neural networks relying on approximate computation of neurons and activation functions. Another set of possible experiments can be done to determine how much energy gained by the usage of EFNNs with respect to classical ANNs.

An important direction for the future works is developing an algorithm to train EFNNs using properties of partitions defined on Section 3.4. We demonstrated that the output of last hidden layer before the output layer linearly depends on the parameters at training data, in lemma 3.4.9. Also, the output of the output layer linearly depends on the parameters by definition. These two observation can be combined to develop a two-phase training algorithm which trains output layer in one phase and trains re-

maining layers in other phase in a partition. This kind of algorithms will require efficient sampling of partitions which might be hard as there is exponentially many partitions exist. A probabilistic selection algorithm can efficiently train EFNNs in multiple partitions and select "best" trained network. This algorithm is not presented in this thesis, because we do not know any efficient probabilistic selection algorithm for these partitions, at the moment.

REFERENCES

[1] A. M. Abdelsalam, J. Langlois, and F. Cheriet. Accurate and efficient hyperbolic tangent activation function on fpga using the dct interpolation filter. *arXiv preprint arXiv:1609.07750*, 2016.

[2] A. M. Abdelsalam, J. P. Langlois, and F. Cheriet. A configurable fpga implementation of the tanh function using dct interpolation. pages 168–171, 2017.

[3] S. Aeberhard, D. Coomans, and O. De Vel. Comparison of classifiers in high dimensional settings. *Dept. Math. Statist., James Cook Univ., North Queensland, Australia, Tech. Rep*, (92-02), 1992.

[4] A. Afrasiyabi, O. Yildiz, B. Nasir, F. T. Yarman-Vural, and A. E. Çetin. Energy saving additive neural network. *CoRR*, abs/1702.02676, 2017.

[5] C. E. Akbaş, A. Bozkurt, A. E. Çetin, R. Çetin-Atalay, and A. Üner. Multiplication-free neural networks. In *2015 23th Signal Processing and Communications Applications Conference (SIU)*, pages 2416–2418. IEEE, 2015.

[6] C. E. Akbaş, O. Günay, K. Taşdemir, and A. E. Çetin. Energy efficient cosine similarity measures according to a convex cost function. *Signal, Image and Video Processing*, 11(2):349–356, 2017.

[7] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V.

Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[8] A. L. Beam. Deep learning 101 - part 1: History and background, Feb 2017.

[9] P. Behrooz. Computer arithmetic: Algorithms and hardware designs. *Oxford University Press*, 19:143–204, 2000.

[10] H.-D. Block. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123, 1962.

[11] A. Blum and R. L. Rivest. Training a 3-node neural network is np-complete. In *Advances in neural information processing systems*, pages 494–501, 1989.

[12] T. M. Breuel, A. Ul-Hasan, M. A. Al-Azawi, and F. Shafait. High-performance ocr for printed english and fraktur using lstm networks. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 683–687. IEEE, 2013.

[13] A. J. Calise and R. T. Rysdyk. Nonlinear adaptive flight control using neural networks. *IEEE control systems*, 18(6):14–25, 1998.

[14] R. A. Callejas-Molina, V. M. Jimenez-Fernandez, and H. Vazquez-Leal. Digital architecture to implement a piecewise-linear approximation for the hyperbolic tangent function. pages 1–4, 2015.

[15] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[16] Z. Cheng, D. Soudry, Z. Mao, and Z. Lan. Training binary multilayer neural networks for image classification using expectation backpropagation. *arXiv preprint arXiv:1503.03562*, 2015.

[17] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, page 113. ACM, 2013.

[18] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[19] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. pages 3123–3131, 2015.

[20] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

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

[22] H. S. Demir and A. E. Cetin. Co-difference based object tracking algorithm for infrared videos. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 434–438. IEEE, 2016.

[23] D. H. Deterding. *Speaker normalisation for automatic speech recognition.* PhD thesis, University of Cambridge, 1990.

[24] Z. Du, A. Lingamneni, Y. Chen, K. Palem, O. Temam, and C. Wu. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 201–206. IEEE, 2014.

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

[26] R. Eldan and O. Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.

[27] I. W. Evett and J. S. Ernest. Rule induction in forensic science. central research establishment. home office forensic science service. aldermaston. *Reading, Berkshire RG7 4PN*, 1987.

[28] A. W. C. Faria, L. P. de Aguiar, D. d. S. D. Lara, and A. A. F. Loureiro. Comparative analyses of power consumption in arithmetic algorithms implementation. *Revista de Informática Teórica e Aplicada*, 18(2):234–250, 2011.

[29] L. V. Fausett. *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, 1994.

[30] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of human genetics*, 7(2):179–188, 1936.

[31] P. W. Frey and D. J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine learning*, 6(2):161–182, 1991.

[32] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[33] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[34] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. pages 1737–1746, 2015.

[35] D. Hammerstrom. A vlsi architecture for high-performance, low-cost, on-chip learning. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 537–544. IEEE, 1990.

[36] D. O. Hebb. *The organization of behavior: A neuropsychological approach*. John Wiley & Sons, 1949.

[37] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

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

[39] K. Hoffman and R. Kunze. *Linear Algebra*. Prentice-Hall, 1962.

[40] M. Höhfeld and S. E. Fahlman. Probabilistic rounding in neural network learning with limited precision. *Neurocomputing*, 4(6):291–299, 1992.

[41] J. L. Holi and J.-N. Hwang. Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, 42(3):281–290, 1993.

[42] P. Horton and K. Nakai. A probabilistic classification system for predicting the cellular localization sites of proteins. In *Ismb*, volume 4, pages 109–115, 1996.

[43] K. Hwang and W. Sung. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pages 1–6. IEEE, 2014.

[44] Y. Ito. Approximation capability of layered neural networks with sigmoid units on two layers. *Neural Computation*, 6(6):1233–1243, 1994.

[45] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[46] D. E. Khodja, A. Kheldoun, and L. Refoufi. Sigmoid function approximation for ann implementation in fpga devices. 2010.

[47] M. Kim and P. Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

[48] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[50] Y. Le Cun. Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240. Springer, 1986.

[51] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[52] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[53] M. Lichman. UCI machine learning repository, 2013.

[54] Y. C. Lim and B. Liu. Design of cascade form fir filters with discrete valued coefficients. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(11):1735–1739, 1988.

[55] A. Lingamneni, A. Basu, C. Enz, K. V. Palem, and C. Piguet. Improving energy gains of inexact dsp hardware through reciprocative error compensation. In *Proceedings of the 50th Annual Design Automation Conference*, page 20. ACM, 2013.

[56] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Fast Neural Networks Without Multipliers. *IEEE Transactions on Neural Networks*, 4(1):53–62, 1993.

[57] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[58] K. Mehrotra, C. K. Mohan, and S. Ranka. *Elements of artificial neural networks*. The MIT Press, 1997.

[59] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ. Strategies for training large scale neural network language models. In *Automatic Speech*

Recognition and Understanding (ASRU), 2011 IEEE Workshop on, pages 196–201. IEEE, 2011.

[60] M. L. Minsky and S. Papert. *Perceptions: An Introduction to Computational Geomry*. MIT press, 1969.

[61] T. M. Mitchell. Machine learning. *Burr Ridge, IL: McGraw Hill*, 45(37):81–126, 1997.

[62] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*, pages 1–7. IEEE, 2016.

[63] K. Nakai and M. Kanehisa. Expert system for predicting protein localization sites in gram-negative bacteria. *Proteins: Structure, Function, and Bioinformatics*, 11(2):95–110, 1991.

[64] W. Nash, T. Sellers, S. Talbot, A. Cawthorn, and W. Ford. The population biology of abalone (haliotis species). *Blacklip Abalone (H. rubra) from the North Coast and Islands of Bass Strait. Sea Fisheries Division Technical Report*, 48, 1994.

[65] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[66] A. B. Novikoff. On convergence proofs for perceptrons. Technical report, STANFORD RESEARCH INST MENLO PARK CALIF, 1963.

[67] M. Panicker and C. Babu. Efficient fpga implementation of sigmoid and bipolar sigmoid activation functions for multilayer perceptrons. *IOSR Journal of Engineering*, pages 1352–1356, 2012.

[68] D. B. Parker. Learning logic. 1985.

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

[70] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

[71] P. Refaeilzadeh, L. Tang, and H. Liu. Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer, 2009.

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

[73] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[74] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy. Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 145–150. IEEE, 2016.

[75] P. F. Silva, A. R. Marcal, and R. M. A. da Silva. Evaluation of features for leaf discrimination. In *International Conference Image Analysis and Recognition*, pages 197–204. Springer, 2013.

[76] M. R. Spiegel. *Schaum's outline of Theory and problems of advanced calculus*. McGraw-Hill, 1963.

[77] I. Stewart and D. O. Tall. *The foundations of mathematics*. Oxford University Press, 2 edition, 2015.

[78] A. Suhre, F. Keskin, T. Ersahin, R. Cetin-Atalay, R. Ansari, and A. E. Cetin. A multiplication-free framework for signal processing and applications in biomedical image analysis. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1123–1127. IEEE, 2013.

[79] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[80] R. S. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proc. 8th annual conf. cognitive science society*, pages 823–831. Erlbaum, 1986.

[81] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[82] K. Torki, H. Faiedh, C. Souani, and K. Besbes. Digital hardware implementation of a neural system used for nonlinear adaptive prediction. *Journal of Computer Science*, 2:355–362, 2006.

[83] H. Tuna, I. Onaran, and A. E. Cetin. Image description using a multiplier-less operator. *IEEE Signal Processing Letters*, 16(9):751–753, 2009.

[84] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, volume 1, page 4, 2011.

[85] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. Axnn: energy-efficient neuromorphic systems using approximate computing. In *Proceedings of the 2014 international symposium on Low power electronics and design*, pages 27–32. ACM, 2014.

[86] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1058–1066, 2013.

[87] P. J. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Doctoral Dissertation, Applied Mathematics, Harvard University, MA*, 1974.

[88] B. A. White and M. I. Elmasry. The digi-neocognitron: a digital neocognitron neural network model for vlsi. *IEEE transactions on Neural Networks*, 3(1):73–85, 1992.

[89] B. Widrow and M. E. Hoff. Adaptive switching circuits. Technical report, STANFORD UNIV CA STANFORD ELECTRONICS LABS, 1960.

[90] S. G. Wysoski, L. Benuskova, and N. Kasabov. Evolving spiking neural networks for audiovisual information processing. *Neural Networks*, 23(7):819–835, 2010.

[91] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[92] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. Approxann: an approximate computing framework for artificial neural network. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 701–706. EDA Consortium, 2015.