DOES STATE-SPACE EXPANSION INCREASE THE GENERALISATION ABILITY OF A PATTERN RECOGNITION SIGMA-PI NEURON [OR NET]?

A dissertation submitted to The University of Manchester for the degree of Master of Science in the Faculty of Science and Engineering

2016

David Toluhi
9673244
SCHOOL OF COMPUTER SCIENCE

TABLE OF CONTENTS

Final word count: 16,881

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This is dissertation presents an investigation into optimization of neural networks, specifically sigma-pi neural networks. Expansion of neural networks is a process that oversamples the input to the network in order to improve the networks performance which is measured using the generalization performance of the network. This study investigates a methodology known as state-space expansion which is another method that may be used to oversample the input to a sigma-pi neural network as sigma-pi neural networks are the only viable candidates for the methodology. The motivation for the study is to replicate similar results to those observed by (Neville R et al., 1999). We evaluate the performance of the model using the generalisation performance to see if the generalisation does improve the network's performance. My findings suggest that it does, but not with a guarantee as my findings suggest that the methodology results in an erratic behaviour of the models, this gives room for further research of the methodology.

DECLARATION

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

COPYRIGHT

i. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the dissertation, for example graphs and tables ("Reproductions"), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy, in any relevant Dissertation restriction declarations deposited in the University Library, and The University Library's regulations.

ACKNOWLEDGEMENT

I would sincerely like to thank Dr. Richard Neville for being gracious and patient enough to mentor me using his mentoring scheme; his insights and methodologies which he taught me positively influenced the quality of the study. Thank you Richard for teaching me how to think and act like a proper scientist.

I would also acknowledge and thank the School of Computer Science at the University of Manchester, and all my teachers, especially Professor Ulrike Sattler and Dr. Bijan Parsia, for providing me with an opportunity that aided my wellbeing during the period of the dissertation.

I would also like to express my gratitude to friends and family, especially my parents for providing all their support during one of the more challenging moments in my career without which this project could not have been completed.

I. INTRODUCTION

A. Overview

The domain that this study is concerned with is that of artificial intelligence. Artificial Intelligence is a field in computer science concerned with the understanding and modelling of intelligent entities (Russell, 1995).

There are three main research domains, #RDn, that need to be explored in order to successfully carry out the study:

#RD1: The first and most important of the three RD's is *neural networks*;

#RD2: The second RD is *function approximation*, one of the areas in which neural networks are applied.

#RD3: The third and last RD is the expansion of neural networks.

All these fall under the field of machine learning: a subfield of artificial intelligence concerned with modelling the intelligent characteristic of learning or in more concrete terms "*adaptation in response to observed data.*" (Brown, 2015) by using algorithms and mathematical models; one may also consider machine learning as a field of deductive reasoning: In the context of machine learning "data" is analogous to "experience" and mathematical models are built to extract information from this data by utilising appropriate data structures and algorithms. The advent of the internet and the increased computational power of computer systems have brought about the information age and as a result, there is an abundance of data, so much so that humans are incapable of processing. Machine Learning, with the help of processing power of modern computing, is used to process information embedded in this data. Neural networks are one of the mathematical models that can be used to learn from data.

An artificial neural network is a learning paradigm which is based on the operation of neural networks in mammalian brains. Neurons are the elementary units of a neural network, and are standardly mathematically modelled in machine learning by a perceptron, "*a probabilistic model for information storage*" (Rosenblatt, 1958). A perceptron models the biological neuron's behaviour by treating an input data vector as incoming signals, multiplying the input vector by the weights to simulate the effect of synapses in biological neurons, the resulting product is passed through an activation-output transfer function and then onto other neurons in the network and the information extracted from the data is stored in the weights/synapses. The sigma-pi model of the neuron is a more accurate model of a biological neuron as it models the presynaptic inhibitory property of biological neurons (Neville R et al., 1999). The artefact of interest is sigma-pi neural networks as the state-space expansion methodology is not compatible with standard neural networks. These neurons are endowed with the ability to expand

their state-space (Neville R et al., 1999). This study is an attempt to reproduce similar observations of the methodology to the ones presented by (Neville R et al., 1999)

B. Aims and Objectives

The primary aim of this study is to investigate whether state-space expansion can improve the generalization performance of pattern recognition sigma-pi neurons and in so doing observe results similar to those observed by (Neville R et al., 1999). The objectives associated with this aim are as follows:

1. Understanding of Design Science which is to be used as the primary research methodology in this study and may evolve over time.
2. Research and understanding of artificial neural networks, involving but not limited to the investigation of topics/artefacts such as: *threshold-logic-unit, the perceptron, the sigma-pi neural networks, backpropagation, automatic differentiation* and *state-space expansion*.
3. Research and understanding of state-space expansion theory as presented by Richard in 2016 (Neville, 2016).
4. Implementation of a standard neural network
5. Implementation of a sigma-pi neural network
6. Implementation of state-space expansion in sigma-pi neural networks.

C. Report Outline

This report is structured as follows: chapter 2 provides a history and background of neural networks. Chapter 3 provides the background on sigma-pi neural networks providing their history and a description of their operation. Chapter 4 provides a description of state-space expansion which is the artefact which this study evaluates; it also highlights the motivations for the state-space expansion methodology. Chapter 5 provides an overview of related studies. Chapter 6 describes the implementations of the artefacts presented in the previous chapters. Chapter 7 presents the evaluation metrics used to evaluate the artefacts presented, chapter 8 presents the results of the experiments and the report concludes in chapter 9.

II. BACKGROUND ON NEURAL NETWORKS

This chapter provides the necessary context and motivation that surrounds the conducted study. The chapter starts by providing a brief historical account of artificial intelligence with regards to how the domain of neural networks arose from it. We then delve deeper into the domain of machine learning in order to discuss the applications and practicality of neural networks and then describe the architecture of neural networks, its mathematical model, and the algorithms that may be used to train neural networks to learn.

A. Artificial Intelligence and it's two approaches

Artificial Intelligence is a subfield of computer science concerned with the understanding and modelling of intelligent entities in machines (Russell, 1995). There are several approaches and viewpoints through which this issue can be addressed; this study adopts the *connectionist paradigm* viewpoint as described by (Gurney, 1989) , (Bechtel and Abrahamsen, 1991) and (Sun, 1999).

1) Artificial Intelligence: The Symbolic Paradigm

The central idea of this paradigm is representation (Sun, 1999) and construes "cognition as involving symbol manipulation" (Bechtel and Abrahamsen, 1991). Newell and Simon in 1976 argue that "Symbols lie at the root of intelligent action", as such the main development of intelligent entities in the symbolic paradigm revolve around symbol manipulation (Newell and Simon, 1976) , (Sun, 1999)

The developments of cognitive sciences in the 1970s influenced the existing views on artificial intelligence (Bechtel and Abrahamsen, 1991); as the artificial intelligence community "developed, the idea that cognition involved the manipulation of symbols became increasingly central" (Bechtel and Abrahamsen, 1991). It was believed that symbols can "refer to external phenomena and so have semantics" (Bechtel and Abrahamsen, 1991).

(Dreyfus, 1979) argues against the assumption that symbols lie at the root of intelligent action and claims that this approach to modelling intelligence may fail in the aspect of representing intelligence (Gurney, 1989). The paradigm is built upon concepts such as state representation and further goes to encode intelligent-processing abilities by using rule-based reasoning systems (Sun, 1999).

The primary shortcoming of the paradigm is its focus on representation which results in a neglect on learning, a characteristic which is fundamental to cognition/intelligence (Sun, 1999). Despite these shortcomings, there are still developments in the paradigm in fields such as automated reasoning whose applications span domains such as hardware verification and creation of video games.

2) Artificial Intelligence: the Connectionist Paradigm

This paradigm is motivated by re-modelling the processing features of the brain in machines(Gurney, 1989). Biological brains consist of billions of neurons communicating via electrical signals that are transmitted using synapses, dendrites and axons; this is illustrated in Figure 1.

"*Each neuron is continually summing or integrating the effected of all its incoming pulses and, depending on whether the result is excitatory or inhibitory, an output pulse may or may not be generated*" (Gurney, 1989). This is modelled in the connectionist paradigm by creating an artificial network of artificial neurons.

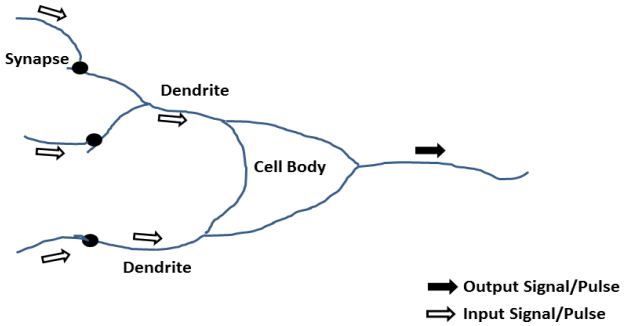


Figure 1: Schematic diagram of a biological neuron. The arrows in the diagram represent the flow direction of the electric impulses.

The strength of the connectionist paradigm lies in its focus on learning which is a key characteristic of intelligent behaviour (Sun, 1999); and recent developments that adopt the ideologies of the paradigm suggest that modelling the behaviour of learning is an important aspect of simulating intelligence; Machine Learning is a subfield of artificial intelligence that deals with modelling the learning capabilities in machines using statistical methods, this is expanded upon in the next section.

Artificial neural networks are the core of the connection paradigm; there are several variants of artificial networks, however, Gurney in 1989 identifies characteristics prevalent in a standard artificial neural network. They are as follows:

- The parameters of every neuron "*are trained to their final values by continually presenting members from a set of patterns or training vectors to the net, allowing the net to respond to each presentation and altering the weights accordingly* " (Gurney, 1989).

- The action of a neuron[or net] when fed input "*is often best thought of as the time evolution of a dynamical physical system and there may even be an explicit description of this in term of a set of differential equations*" (Gurney, 1989).
- "*They are robust under the presence of noise on the inter-unit signal paths and exhibit graceful degradation under hardware failure*" (Gurney, 1989). This means that neurons have the capability to perform well when presented with a corrupt dataset.
- "*A characteristic feature of their operation is that they work by extracting statistical regularities or features from the training set*." (Gurney, 1989).
- "*Standard modes of operation are as associative memories, retrieving complete patterns from partial data, and as pattern classifiers*." (Gurney, 1989).
- "*There is no simple correspondence between nodes and high semantic objects.*" (Gurney, 1989).

3) A brief history of Neural Networks

The concept of neural networks were present during the early stages of the development of modern PCs (personal computers) (Gurney, 1989); this is made apparent in the first draft report on the EDVAC PC by Jon von Neumann (Von Neumann, 1993) as he makes structural correlations between a biological neuron and the proposed circuit elements for the modern PC. (Gurney, 1989).

In 1943, Warren McCulloch and Walter Pitts wrote what is widely considered the first formal document about how biological neurons operate (Gurney, 1989) , (McCulloch and Pitts, 1943).

In 1949, Donal Hebb illustrated that the usage of neural pathways was a fundamental element in the learning ability of animals (Gurney, 1989).

Unfortunately for these pioneers, the computing power available was too little and simplistic to test and simulate their ideas, but as the 1950's approached, computers became more and more advanced and the next major development in neural networks was the invention of the perceptron by Rosenblatt (Gurney, 1989).

Several other developments continued in the domain from that point on; in 1959 , Bernard Widrow and Marcian Hoff of Stanford were developing the ADALINE and MADALINE models: acronyms for ADAptive LINear Elements and Multiple ADAptive LINear Elements respectively around the same time Rosenblatt was developing the perceptron (Bishop, 1995). In 1972, Kohenen and Anderson developed the self-organizing map which involved "*investigation of neural networks that used topological feature maps*" (Gurney, 1989).

Fortunately, Moore's law[1] prevailed over the coming years and computers became faster and more powerful. Also parallel to his was the advent of the internet which brought about the information age resulting in an abundance of data and increased connective this resulting in massive strides in the field; perhaps the most significant recent development is "*Human-level control through deep reinforcement learning*" by (Mnih et al., 2015). Today, neural networks are applied in many domains such as bioinformatics, image processing, robotics, etc.

B. Machine Learning

As discussed previously in chapter 1, "data" is analogous to "experience" in the context of machine learning; Machine Learning is a domain that concerned with algorithms that build mathematical models used to extract information from said data, these are called *learning algorithms*; as such, data is critical to the field and it can further be concluded that the fields applications span any domain/industry where data can be extracted from. Data in this context may entail, but is not limited to birth records at a hospital, sales records from a car dealership, DNA sequences, and pixels of an image. A dataset in the context of machine learning can fall into one of the following categories #Cn:

#C1: Labelled data: Datasets that fall into this consist of samples that are labelled; the labels given to these samples are sometimes referred to as classes.

#C2: Unlabelled data: Datasets that fall into this category consist of samples that have no labels or classes and the task of the learning in this context is to detect patterns that may exist in the dataset

The viewpoint of categorized datasets as either labelled or unlabelled, result in the realization of two learning paradigms #Pn:

#P1: the first is supervised learning which entails learning from labelled-data; and

#P2: the second is unsupervised learning which entails learning from unlabelled-data.

It should be noted that a third domain known as *semi-supervised learning* has been realized in recent times, but is not recognized widely across the community. Semi-supervised learning involves learning from both labelled and unlabelled data. This study falls into the domain of supervised learning.

Datasets in the context of machine learning are matrices of numerical values. Each row in the matrix is a vector representing a sample[2] of the data that is being represented; for example, a dataset can consist of houses, their characteristics and their prices, each row in the dataset would be a vector of numerical values such as the living area size, number of bedrooms and the price would be labels of that row in a task of predicting house prices. These characteristics are referred to as features and the number of

[1] Moore's law is the observation that the number of transistors in a dense integrated circuit doubles approximately every two years. This increases the computation capabilities of machines.

[2] A data-sample is also sometimes referred to as an "Example".

features and samples in a dataset represents the dimension of the dataset. The goal of a learning algorithm is its ability to encode the underlying form or function in a dataset into a mathematical model that can be used to predict values for samples that aren't included in said dataset, this is also known as the model's ability to *generalize* and is used as a metric in evaluating the performance of models and learning algorithms, this is expanded upon in chapter 6. In the context of supervised learning, one may consider the task of a learning algorithm to map an arbitrary vector form the input space to the space of outputs. The space of outputs is the space containing the labels associated with every data vector in the input space. Mathematically speaking, given a "*vector mapping from the space of inputs*" (Stulp and Sigaud, 2015), the task of a learning algorithm would be to create a model that can predict the label of that vector in the output space; A neural networks task is, therefore, to modify the weights[parameters] of all its neuron in order to achieve this task. "*The process of determining the values for these parameters on the basis of the dataset is called learning or training and for this reason, the dataset of examples is generally referred to as a training set*" (Bishop, 1995). A neural network learns by the use of a learning algorithm. The learning algorithm used to train a neural network depends on the architecture of the network.

The examples presented above mainly have to deal with learning algorithms finding or understanding patterns in dataset, this can be considered as subdomain of supervised learning which is pattern recognition. There is another domain in which learning algorithms may apply which is *function approximation*. Function approximation is concerned with approximating the values of a mathematical function and in the domain of function approximation, the input vectors of the dataset is the input to a mathematical function and the output is the output of said mathematical function. "*Both in econometric and in numerical problems, the need for an approximating function often arises. One possibility is that one has a finite set of data points and wants to determine the underlying functional form*."(Wouter Den Haan, 2016). A simplified example to illustrate this is a dataset whose samples consists of some points from an x^2 function as depicted in Figure 2 the goal of approximation is therefore to understand the underlying trend or form of the rest of the function.

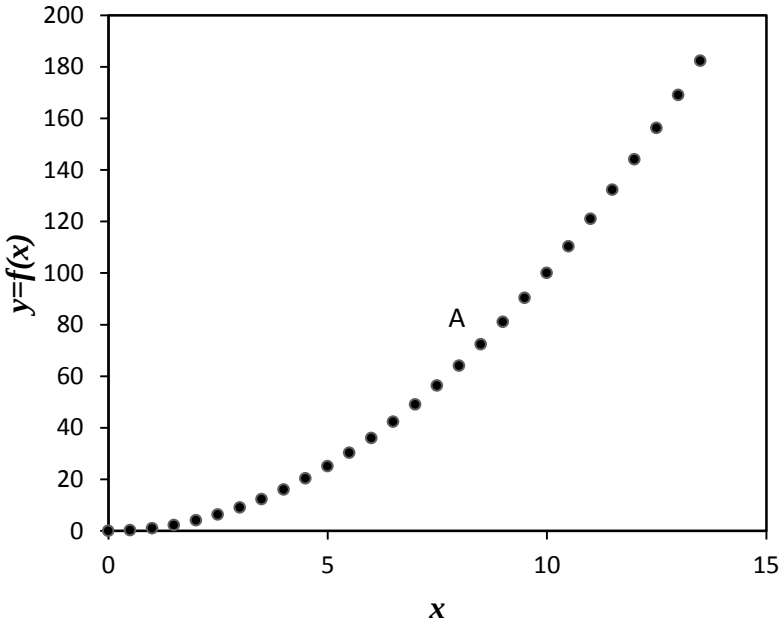


Figure 2: An illustration of a function that may be approximated by a neural network. The graph depicts the x^2 function over the interval [0, 15].

This may be accomplished by providing the available points as a dataset to a learning algorithm, once the algorithm trains on this set, the resulting model can be provided an arbitrary set of data points and their corresponding labels, or y-values in this case can be predicted as a result, the trend of the rest of the function can be detected by providing a continuous set of arbitrary points. The experiments that are pertinent to this study fall into the domain of function approximation; this is discussed further in the implementation and experimental chapters.

C. Architecture of Neural Networks

1) Overview

This section provides an overview of the architecture of how artificial neural networks model biological ones. The operation of a neuron is described in first, before moving onto the how the neurons interoperate in a network and are trained.

Neural networks have several architectures, but usually fall into one of the following categories:

- Feed-forward Neural Networks: In a feed-forward neural network, the neurons are linked in a unidirectional manner, and there are no cycles (Russell, 1995).
- Recurrent Neural Networks: "*In a recurrent neural network, the links can form arbitrary topologies*." (Russell, 1995). This means that cycles can exist in a recurrent neural network which is the case in most recurrent neural networks (Russell, 1995).

The focus is mainly on the architecture of a feed forward neural network because recurrent neural networks are not relevant to this study.

2) Architecture of Neurons

The architecture of a standard artificial neuron is loosely based on that of a biological neuron. A biological neuron consists of a cell body also known as the soma, dendrites, synapses and an axon (Gurney, 1997). A standard biological neuron usually has several dendrites and one axon which can branch out into several branches to connect to the dendrites of neighbouring neurons (Gurney, 1997). Biological neurons communicate via electrochemical signals. The dendrites of a neuron are responsible for receiving electrochemical signals from other neurons and are integrated into the cell body of the receiving neuron (Gurney, 1989). The neuron fires a signal to other neurons in the network if the action potential on the surface its cell body which is defined an aggregate sum of all the incoming signals to the neuron a certain threshold, if this happens, the neuron is said to be activated and fires a signal. Dendrites and axons make contact at junctions called *synapses* (Gurney, 1997). It should be noted that the effect an incoming signal has on the neuron is affected by the synapse. Synapses possess physicochemical properties which dictate the degree of the effect the incoming signal has on the action potential (Gurney, 1997). In a standard artificial neuron, this physicochemical property is rather trivialized, but is more accurately a captured in a sigma-pi neuron; the operation of a sigma-pi neuron is discussed further in chapter 3.

To illustrate how the characteristics of a biological neuron are captured by an artificial neuron, a sample task that an artificial neuron may be used for is described as follows: An artificial neuron may be used to map a logical AND function which is represented in Figure 3; this is an example of a dataset which is previously discussed. To understand the rest of this section, it is sufficient to only understand that every row in the input space as depicted in a picture is a data vector and the corresponding value in the output space on the same row is the label for the data vector.

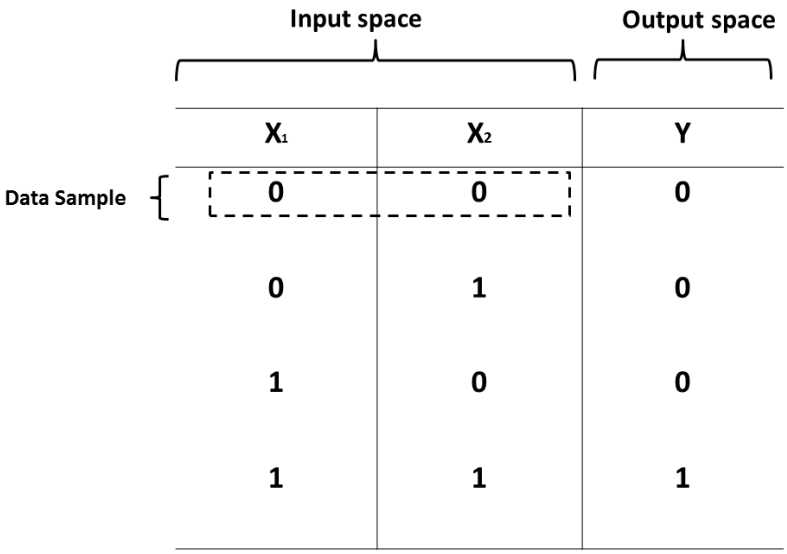


Figure 3: A sample dataset. The rows in the data are data-samples. The rows in the input space constitute of the data vectors fed into the neuron[or network]; The rows in the output space is the mapped value of the corresponding input data vector from the input space.

The task of this neuron would be to model the weights of a linear function that can correctly map an input vector to the output space. The input data vector is analogous to incoming signals in biological neurons. The synapse is simulated using weights in artificial neurons, the input data is multiplied by these weights and the values are summed and passed through an activation function to normalize the output of the neuron to a defined range, this is because the weights of the neuron are initialized to random numerical values and the range of the values in the dataset is usually arbitrary. The activation function of a standard artificial neuron serves to squash the output of the neuron to a range that has defined threshold for determining whether the neuron has been activated or not. In most standard neural networks, the activation function is the sigmoid function shown in (2) which squashes the output of the neuron to a range between 0 and 1, threshold of this range is 0.5, as such if the squashed output is greater than 0.5, the neuron is said to be activated. This process is illustrated in Figure 4.

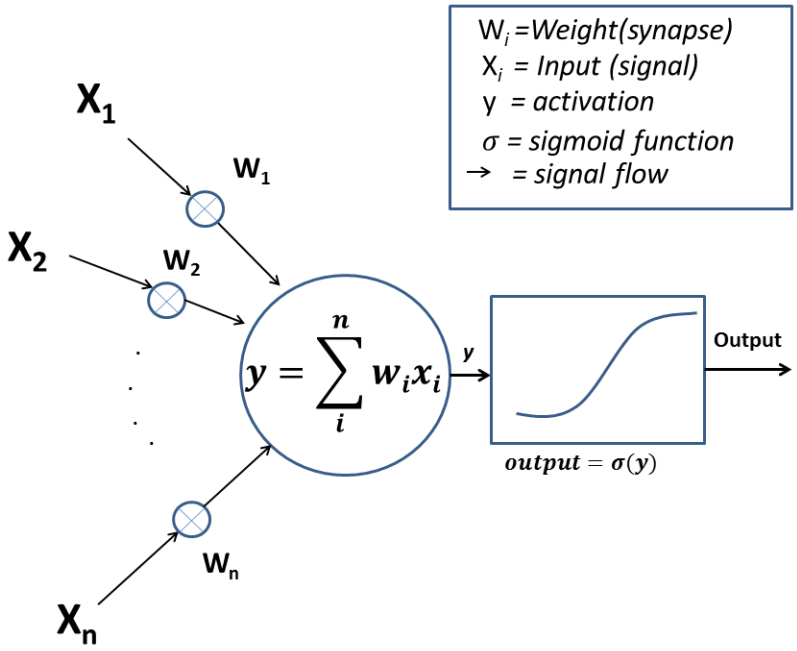


Figure 4: Schematic diagram of an artificial neuron. The arrows in the diagram represent the flow direction of the electric impulses.

$$activation = \sum_{i}^{n} w_i * x_i \tag{1}$$

$$f(x) = \frac{1}{1 + e^{-x/\rho}} \tag{2}$$

As can be seen in (1), the output of a standard neuron is a sum of linear weighted inputs. . The weights of the neuron are adjusted continually through the use of a learning algorithm until it reaches a minimum error; this is expanded upon when the training of feed forward neural networks is discussed.

3) Architecture of a feed forward neural network

A neural network is a network of neurons as the name suggests. A standard neural network consists of layers of interconnected neurons as depicted in Figure 6.

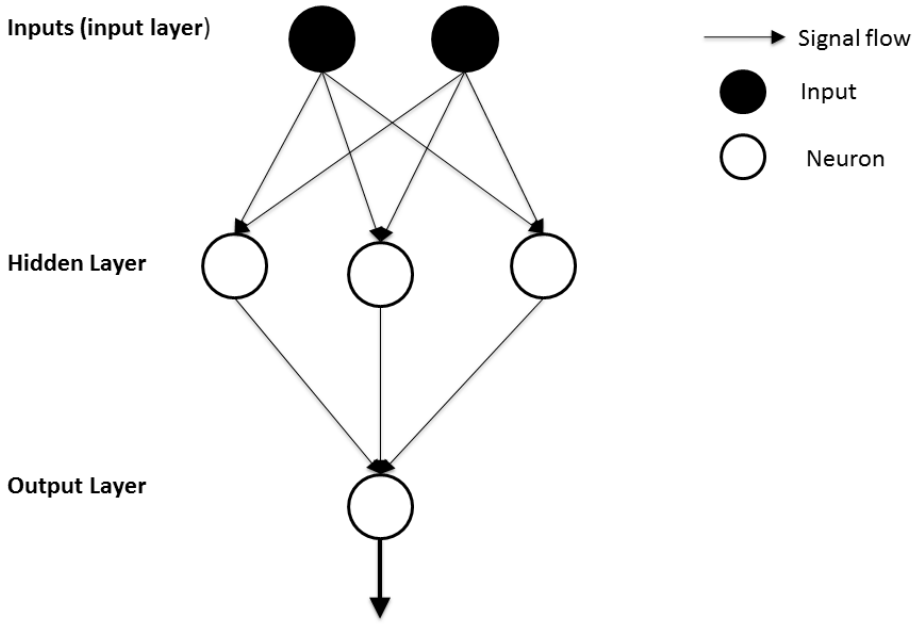


Figure 5: Schematic diagram of an artificial feed forward neural network. The input layer represents and input vector to the network with each node having a numerical value. The neuron in the hidden layer multiply the values from the input layer with its weights, sums the result, passes it through an activation function and passes it onto the output layer. The output layer receives the output from the input layer and treats it as an input vector does the same processing as any other neuron.

The first layer also called the input layer consists of the input vector which is fed into the next layer. Each neuron consists of a weight vector which is whose size is equal to the number of neurons in the layer preceding the layer that the neuron is in. The output of every neuron in one layer is a vector which is the input for each neuron in the next layer. The last layer of the network is also referred to as the output layer and the intermediate layers are referred to as hidden layers. The output of the network is the network's mapping of the input vector to the output space.

4) Training Artificial Neural Networks

The learning algorithm used to train a neural network depends on the domain of learning in which it is being used. The focus is mainly on the training algorithms in the supervised learning domain as it is pertinent to this study, and briefly describe those in the unsupervised learning domain.

a) Supervised Learning algorithms

Supervised learning algorithms for artificial neural networks include the reward penalty algorithm and the gradient descent learning algorithm. The focus lies mainly on the gradient descent algorithm as it is pertinent to this study.

(1) Gradient Descent Learning algorithm

A feed forward neural network is trained using the gradient descent algorithm developed by (Werbos, 1974) , (Stulp and Sigaud, 2015). The methodology this algorithm implements is an iterative minimization of the network's error, shown in (3), according to the delta rule which roughly states that in order to minimize the error of a model, the updates to the model's parameters, in this case the weight space of the neuron, should be in the direction of the negative gradient of the error with respect to its weights (Battiti, 1989).The updates to the weights can either be done iteratively i.e. after each input vector to the weight is presented to the network this is also known as *stochastic gradient descent* or in batch i.e. after the network goes through all the input vectors this also known as *batch gradient descent*. The gradient descent algorithm is also sometimes referred to as the backpropagation algorithm. These terms are used interchangeably during the rest of this document.

The gradient descent algorithm consists of two phases, #Pn:

#P1: Feed-forward phase.

#P2: Backpropagation.

The feed-forward phase: During the feed-forward phase, data is "fed" into the network [or neuron]. Each neuron in the first layer of the network processes said input and passes the output onto the next layer as illustrated in Figure 5. The generalisation error of the network's mapping of the input vector is calculated using the mean squared error shown in (3) at the output layer, where t^p denotes the mapping of the input vector in the output space also known as the target vector and y^p denotes the network's mapping of the input space to the output space (Stulp and Sigaud, 2015). This error is then propagated backward through all the layers of the network (Stulp and Sigaud, 2015). The whole process is repeated until the network converges to a minimal error.

$$mean\ squared\ error = \frac{1}{n}\sum_{p=1}^{P=n}(t^p - y^p)^2 \tag{3}$$

Backpropagation phase: The error computed at the output layer is propagated through all the layers of the network (Stulp and Sigaud, 2015). Backpropagation involves calculating the error gradient of the network's output as a whole (which is the error of the output nodes) and propagating that gradient throughout all the previous layers. The error gradient of the output layer is the derivative of the error function.

The weights of the neurons are then updated using the according to the delta rule (4) where $weight_{ji}$ is the current value of the weight, α denotes the learning rate, $\delta_j$denotes the error between the actual output and the target output and y_i denotes the input from neuron i to j as depicted in Figure 6 (Stulp and Sigaud, 2015).

$$weight_{ji} = weight_{ji} + \alpha\delta_j y_i \tag{4}$$

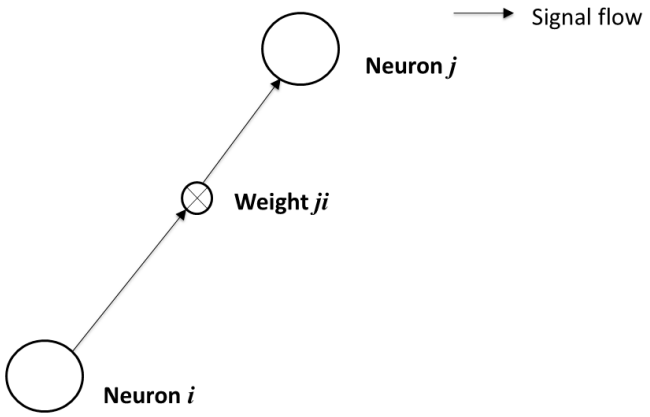


Figure 6: Credit assignment in a link between a neuron in layer i and the subsequent layer j. $Weight_{ji}$ represents the weight/synapse between both $neuron_i$ and $neuron_j$.

If the neuron being updated is in the output layer the error delta is calculated as the difference between the output and the target output as shown in (5) (Stulp and Sigaud, 2015), where $\sigma'(y_j)$ is the sigmoid derivative of the neuron's output.

$$\delta_j = \sigma'(y_j)(t^p - y^p) \tag{5}$$

If the neuron is in the hidden layer, the error delta is calculated as a dot product of the errors of the neurons in the subsequent layer and the respective weights as shown in (6) (Stulp and Sigaud, 2015).

$$\delta_j = \sigma'(y_j) \sum_{k=1}^{K} \delta_j w_{kj} \, (t^p - y^p) \tag{6}$$

This is the standard form of the update rule to the weights and proves difficult to understand because the representation adopted is used to compact the mathematical operations involved, as such we use a method called *automatic differentiation* to provide a more explicit explanation of how the network is trained. Automatic differentiation is a technique used to illustrate the how derivative of a function is mapped or calculated by a computer program or algorithm, in this we address is the error function/delta rule of the network with respect to its weights and the gradient descent algorithm.

We start by illustrating the feed forward phase of the network using automatic differentiation

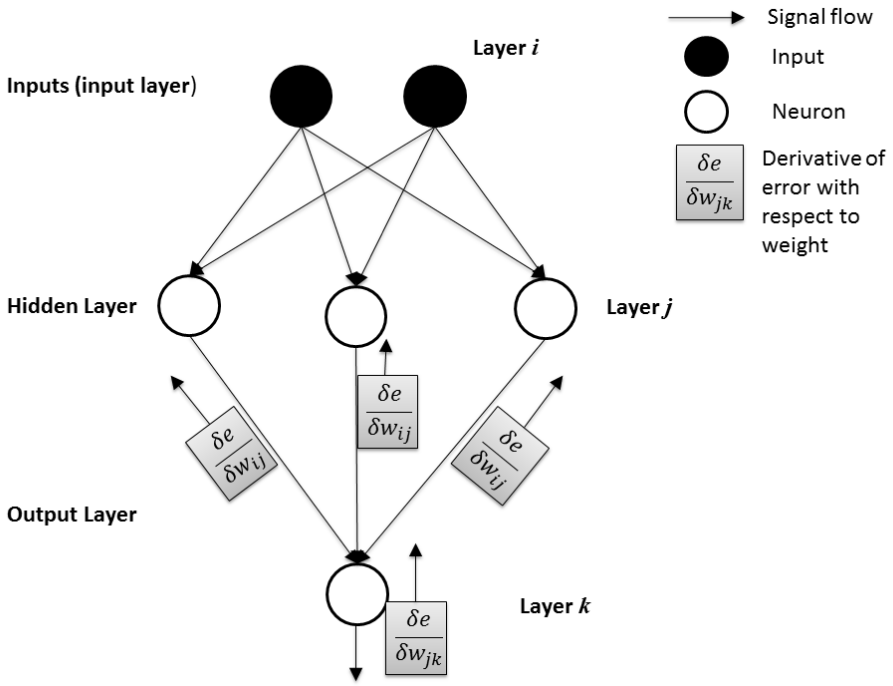


Figure 7: Diagram illustrating the backpropagation phase of the gradient descent algorithm. The error calculated at the output layer is propagated throughout the rest of the layers with respect to each neurons weight. *e* represents the error of the network.

techniques to illustrate all the mathematic functions involved during the feed-forward phase so as to make the illustration of each functions respective derivative during the backpropagation phase clear.

Automatic differentiation is used to evaluate the derivative of a function with a computer program (Griewank, 1989). It is based on the idea that we can break any composite function into littler sequences of operations that can be carried out by a computer program (Griewank, 1989). We leverage this idea to

overcome the hindrances associated with converting symbolic mathematics to an algorithm to be implemented by the computer program (Griewank, 1989); the major issue being that the standard methods of doing so such as *Symbolic differentiation* and *Numerical differentiation* are likely to suffer from round-off errors over small intervals abridged errors over large intervals (Griewank, 1989).

Automatic Differentiation repeatedly applies the chain rule to overcome these issues. It repeatedly applies the chain rule to calculate the derivative of a composite function. In this case, the interest lies in the derivative of the network's error function with respect to each respective neuron's weights. Automatic Differentiation repeatedly applies the chain rule to computational representation of a function in order to calculate the values for the function and its derivatives.

The chain rule is used to simplify the derivative of composite functions; we compute the derivatives of the individual functions and multiply them. For instance, to calculate the derivative of (7), we have to evaluate the derivative of each individual function in (7) and multiply them. The derivative for (7) with respect to *x* is then the multiplication of (8), (9) and (10) as shown in (11).

$$f(g(h(x))) \tag{7}$$

$$\frac{dh}{dx} \tag{8}$$

$$\frac{dg}{dh} \tag{9}$$

$$\frac{df}{dg} \tag{10}$$

$$\frac{df}{dx} = \frac{df}{dg} * \frac{dg}{dh} * \frac{dh}{dx} \tag{11}$$

Automatic Differentiation has two modes: forward accumulation phase, reverse accumulation phase. The forward accumulation phase of automatic differentiation is a step by step the evaluation of the mathematical sequence of operations for composite function of which it is attempting to calculate a derivative for. In the context of neural networks, we may view the whole neural network as a composite function on its own. For instance, in a two layer network, the mathematical model is a composite function of every layer i.e. the function of the network is the function of the output layer, the function of the output layer is the function of the vector outputs of the hidden layer, and the function of the hidden layer is a function of the vector of inputs to the networks also known as the input layer, this is illustrated in Figure 8. The forward accumulation phase can also be considered as the feed forward phase in the

gradient descent learning algorithm. We also illustrate the forward accumulation phase for a hypothetical two-layer network in the Table 1.

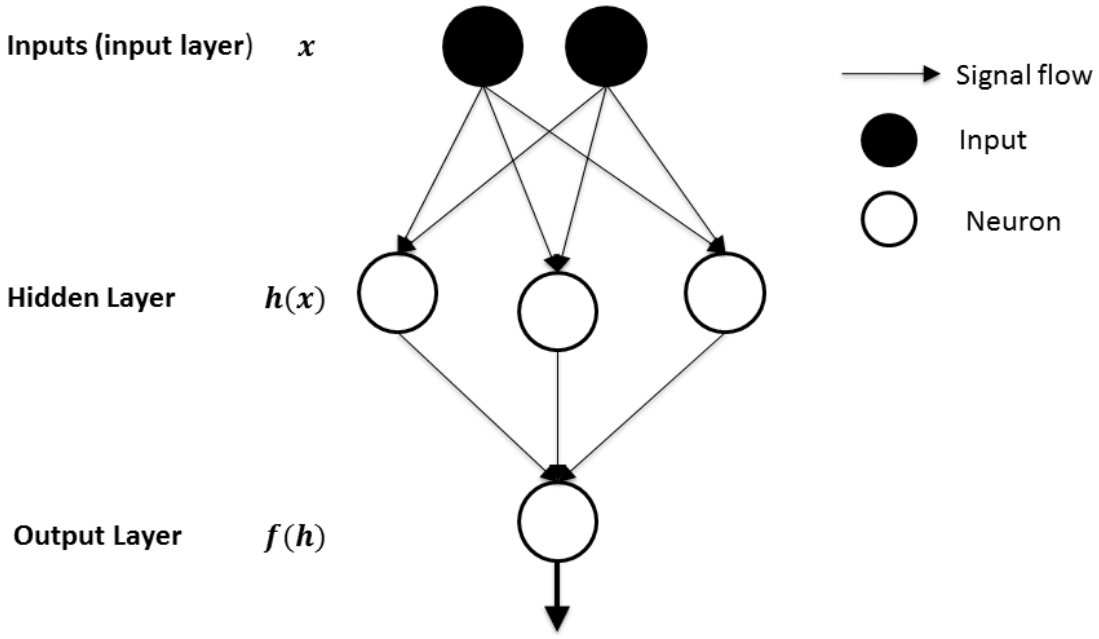


Figure 8: Depiction of a feed forward neural network as a composite function. The input layer represents the input vector to the network, the hidden layer is the function of the input vectors the output layer is a function of the hidden layer function.

Table 1 illustrates the feed forward phase of a two layer artificial neural network, the table is built sequentially from the most elementary units to more complex ones as the input variable is fed through all the layers of the network, vertices are used to keep track of and represent the variables/units represents a mathematical operation used in evaluating the composite function that represents the neural network and are used as references in in the subsequent table that illustrates the backpropagation phase of the network.

Table 1: A Depiction of forward accumulation phase of a two layer Standard Neural Network using automatic differentiation

Network Layer	Vertex	Value	Comment
Input	v_0	x_i	This is the input vector to the network(also considered as the first layer of the network) as illustrated in Figure 5, in this case, layer *i*.
	v_1	$w_{i,j}^h$	This illustrates the weights of the neurons in the hidden layer *j* that
Hidden	v_2	$a_j^h = \sum_{i,j} w_{i,j}^h * x_i^h$	This is the activation of one of the units in the hidden layer of the network
	v_3	$y_j^h = \sigma(a_j^h)$	This is the final output of one of the nodes in the output layer, to be used as one the values in the input vector for subsequent layer *k* which is the output layer. 'σ' denotes the sigmoid function.
Output	v_4	$a_k^o = \sum_{j,k} w_{j,k}^o * y_j^h$	y_j^h denotes the output value from the hidden layer.
	v_5	$y_k^0 = \sigma(a_k^o)$	
	v_6	$e_k^o = \frac{1}{2}(t_k^o - y_k^o)^2$	The error function or delta rule for the output layer: *o*. The gradient of the function is to be calculated and back propagated through the rest of the network.
	v_7	e_k^o	

The reverse accumulation phase of automatic differentiation repeatedly applies the chain rule of calculus to calculate the derivative to the steps/vertices of the evaluated function in the forward

accumulation phase. The reverse accumulation phase of automatic differentiation can also be considered as the backpropagation phase in the gradient descent learning algorithm. In the case of feed forward neural networks, we evaluate the partial derivative of the network as a whole with respect to the weights of the network; this is illustrated in Figure 7 and Figure 8. In Table 2, we illustrate the reverse accumulation phase of automatic differentiation in the second column from the bottom to the top of the table (in reverse order) i.e. the steps it takes to evaluate the derivative of function of the network as a whole, as a sequence of mathematical operations.

The chain rule is applied in the sense that the derivative for each vertex (the same row in the first column) is calculated and multiplied by the derivative of the subsequent layer, just like in the chain rule. The update rule is to multiply the derivative of the weights, denoted by Δw with the learning rate and add the resulting value to the previous weights as shown in (12), where α depicts the learning rate.

$$w = w + \alpha \Delta w \tag{12}$$

Table 2 illustrates the both the feed forward phase and backpropagation phase of a two layer artificial neural network. The backpropagation phase is illustrated in the 'reverse accumulation' column of the table and illustrates derivatives of the vertices during the feed forward phase, all of which are used to calculate the weight updates of the network.

Table 2: A depiction of Derivatives for the backpropagation of the error gradient in a 2-Layer standard neural network using automatic differentiation

Layer	Vertices	Value(feed forward phase)	Reverse Accumulation (Backpropagation phase)
Input	v_0	x_i	
	v_1	$w_{i,j}^h$	
Hidden	v_2	$a_j^h = \sum_{i,j} w_{i,j}^h * x_i^h$	$\Delta w_{i,j}^h = x_i^h * \Delta a_j^h$ $\Delta x_i^h = \sum_{i,j} w_{i,j}^h * \Delta a_j^h$
	v_3	$y_j^h = \sigma(a_j^h)$	$\Delta a_j^h = \sigma'(a_j^h) * \Delta y_j^h$
	v_4	$w_{j,k}^o$	$\Delta w_{j,k}^o = y_j^h * \Delta a_k^o$
	v_5	$a_k^o = \sum_{j,k} w_{j,k}^o * y_j^h$	$\Delta y_j^h = \sum_{j,k} w_{j,k}^o * \Delta a_k^o$
	v_6	$y_k^0 = \sigma(a_k^o)$	$\Delta a_k^o = \sigma'(a_k^o) * \Delta y_k^o$
Output	v_7	$e_k^o = \frac{1}{2}(t_k^o - y_k^o)^2$	$\Delta y_k^o = (t_k^o - y_k^o) * \Delta e_k^o$
	v_8	e_k^o	$\Delta e_k^o = 1$

Algorithm 1: Stochastic gradient descent

1: begin

2: initialize weights $w_{index,layer}$ of network to random value

3: $t \leftarrow 0$

4: $i \leftarrow 0$

5: ***for*** *each sample* $\vec{s}$ in *dataset* ***do***

6: $a \leftarrow feed\ \vec{s}\ though\ network$

7: $e \leftarrow labels(i) - a$

8: $\Delta w = backpropagate\ e\ through\ network\ in\ reverse\ order\ of\ layers$

9: $update\ weights\ of\ all\ neurons\ with\ respective\ \Delta w$

10: **end for**

11: **if** $t <$ number of iterations **then**

12: $t = t + 1$

13: **end if**

14: end

The idea behind backpropagation is that the error of the network is influenced by the contribution of all the nodes in the network, the weights of the nodes need to be updated based on the error of the network as a whole (Gurney, 1997). For each node in the network, we would like to update its weights based on the degree of influence it had on the output of the whole network (Gurney, 1997). This is sometimes referred to as the credit assignment problem which is why the weight update as described in (6) is a produced derivative of the error with respect to the weight associated with that nodes' output (Gurney, 1997).

When this process is repeated over several iterations, the network would eventually converge to a minimal/satisfactory error.

(2) Geometric interpretation of Neural Networks

The input vectors that are fed into standard neurons usually have a bias value which is usually '1' appended to the end of the vector, as such the size of the vector of weights in the neuron is 1 plus the size of the input vector; If we consider a one input vector, weight vector of the neuron would consist of two values; one to handle the input, and one to handle the bias unit, the weight associated with the bias unit is also referred to as the threshold of the neuron. In a pattern recognition task, the neuron goal is find the parameters of the neuron that can act as decision boundary in Euclidean space that serves to separate classes in the dataset. The decision boundary of the standard neuron is the plane in Euclidean space in which the weight associated with input vector is equal to the weight associated with the bias unit or the threshold unit.

In a function approximation task, the standard neuron serves to adjust its weights to simulate the effect of that function represented by the dataset, in such instances, the only option available to the weight is a first order polynomial because the only calculation it performs is a dot product between the input vector and the neurons weight vector, this places a limit on the neuron's functionality (Neville R et al., 1999), because of this, the only way to achieve a higher order polynomial functionality is to increase the

number of neurons in the network. This constitutes as part of the motivation for this study as it turns out that sigma-pi neurons are a more powerful variant of the standard artificial neuron that can be used to overcome this limitation. This is expanded upon this in Chapter 4.

b) Unsupervised learning Algorithms

There are two main types of neural networks associated with unsupervised learning they are as follows:

- Autoencoders: These neural networks are mostly used for reducing the dimensions[3] of a dataset. The network extracts useful features from the datasets they are provided with by encoding a representation of the dataset in the hidden layers in the network and then use this information to reduce the datasets dimensionality (Hinton and Salakhutdinov, 2006).
- Self-Organizing Maps: These neural networks are used for visualizing patterns high-dimensional datasets in a Euclidean space. The goal of the network is to create a structure that mimics the patterns that it discovers in a dataset (Kohonen, 1990).

c) Weights as a storage mechanism

As the weights are the updated components in the neural networks, one can view them where all the information is stored after learning has taken place. As articulated by Rosenblatt, an artificial neuron can be viewed as "*a probabilistic model for information storage*" (Rosenblatt, 1958).

[3] Dimensions in the context of machine learning refers to the size of the dataset either in terms of the features of the data-samples or in terms of the amount of data-samples in the dataset.

III. BACKGROUND ON SIGMA-PI NEURONS

A sigma-pi neuron is a variant of a standard artificial neuron. The primary difference between a sigma-pi neuron and a standard artificial neuron is that the architecture of sigma-pi neuron takes into account the pre-synaptic inhibitory property of synapses in biological neurons (Gurney, 1997).

The pre-synaptic inhibitory property occurs in most synapses of biological neurons. The synapse modelled by standard artificial neuron as illustrated in Figure 1 and Figure 4 suggests that there is a straightforward and standalone connection between an axon and a dendrite at a synapse, while this may be the case in some instances; there are other neurons that possess the architecture illustrated in Figure 9. Figure 9 illustrates a presynaptic inhibitory synapse, here *axon* 2 bears upon *axon* 1, and as such has the ability to influence the "*efficiency of the synapse with axon 1 by inhibiting it's action*" (Gurney, 1997)

Given that the synapse of a biological neuron is analogous to the weight in an artificial one, we may denote the post-synaptic effect of an input; or axon (Gurney, 1997); as wx where *w* denotes the weight of artificial neuron and *x* denotes the input corresponding with that weight.

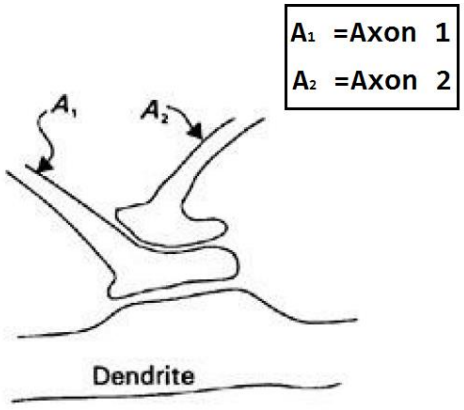


Figure 9 Schematic diagram of an pre-inhibitory synapse (Gurney, 1997). Where A_1 can be denoted as input x_1 to the neuron and A_1 denoted as input x_2 in mathematical terms.

If there were two inputs to a neural net with presynaptic inhibitory properties in its neurons, Figure 8 illustrates the pre-inhibitory synapse where Axon 1 can be considered as input x_1 and Axon 2 as x_2. To model the inhibitory effect of axon 2 on axon 1 mathematically, the previously described postsynaptic effect may be simply multiplied by $(1-wx_2)$ where *w* is less than or equal to 1 and x_2 represents the other axon impinging on inputx_1; this resulting postsynaptic effect is now summarized in (13)

$$postsynaptic\ effect = wx_1(1 - wx_2) \tag{13}$$

We can deduce from (12) that when $x_2=0$, the axon/input represented by $x_2$has no influence on the efficiency of the synapse and (12) is then reduced to *wx* (Gurney, 1997); however the axon/input has a modular effect when greater 0, this is an approach to model the presynaptic inhibitory property of neurons(Gurney, 1997).

When presented with an input vector of size n, the sigma-pi neuron [or network] initializes its weight vector to 2^n random continuous values, the input to the neuron is then modified to 2^n variables, otherwise known as sites, the values in each site is a mathematical model of the presynaptic-inhibitory property of neurons and is a function of the input vector to the neuron. The site addresses of the neuron may be interpreted to model the probability of addressing the weight associated with each respective site (Neville, 2016); in a standard neuron, the probability associated with addressing the weights of the neuron is always equal to 1, which places a polynomial limitation on the neuron as discussed in Chapter 2. To calculate the values in each site sigma-pi neurons use binary components to formalize the presynaptic-inhibitory property. Each site has a binary address associated with it and the binary address is the binary value of the site's index, i.e. for index 0, the binary value is 0 and for index 2, the binary value is 10. The length of the binary string(or the bits in the address) for each site's address is the equal to the length of the original input vector, therefore, for an input vector of size 2, we would have 4 sites, 4 binary addresses, each of string length 2(or 2 bits) and 4 weights associated with each address. The value in each site is a function of the input vector calculated using the binary address associated with it. Each bit in each binary address is associated with the same input index in the input vector i.e. the i^{th} bit in an address is associated with the ith value in the input vector; Therefore, for an input vector of size n, the site value for each site is a product of all n interpretations of the input vector values. The interpretations of the input vector values is as such: for any value x with index i in the input vector, if the i^{th} bit of the binary address associated with the site being addressed is 0 we interpret the input as $1-x$ and otherwise, the i^{th} bit is 1 and we interpret x as x, this is illustrated in Figure 10 and Tables Table 3 and Table 4.

Table 3: A depiction of the calculation of site-address values a sigma-pi neuron when fed a 1-input vector.

Input vector	*Site Address Index*	*Binary Value*	*Site address Value*	*Value*
[2]	0	0	(1-2)	-1
	1	1	2	2
			Total:	1

Table 4: A depiction of the calculation of site-address values in a sigma-pi neuron when fed a 2-input vector.

Input vector	*Site Address Index*	*Binary Value*	*Site address Value*	*Value*
[2, 3]	0	00	(1-2)*(1-3)	2
	1	01	(1-2)*3	-3
	2	10	2*(1-3)	-4
	3	11	2*3	6
			Total:	1

If we represent values of the site addresses as a vector, we have $P_\mu = [p_0, \dots, p_n]^T$ where p_i is the site value for index i and the represent the weights as a vector $W_\mu = [w_0, \dots, w_n]$ where w_i is the weight associated with site value at index i the activation function of a sigma-pi neuron is shown in (3).

$$activation = \sum W_\mu * P_\mu \tag{3}$$

As shown in Figure 10, μ_i represents the Binary version of the site address for a network with an input vector of length 1. The calculation of each site value P_μ can then be formalized mathematically using (4) where M represents the bit in the Binary representation of the site address.

$$P_\mu = \prod_{j \epsilon M_{\mu 0}} (1 - x_i\ ) \prod_{j \epsilon M_{\mu 1}} (x_i\ ) \tag{4}$$

The activation of the neuron is then passed through a sigmoid to get the output of the node.

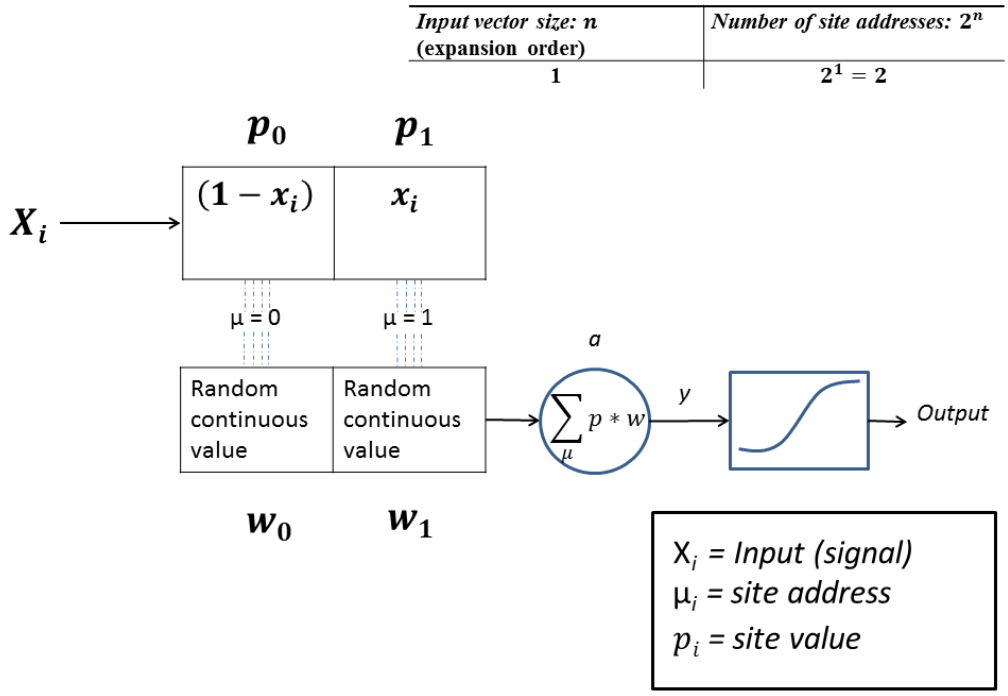


Figure 10 Schematic diagram of a sigma pi unit. Also serves as a depiction of 1st order state-space expansion implemented in a sigma pi unit. Here, we can see that the interpretations of the site input in each site address in the boxes denoted by p_n.

This architecture captures the presynaptic inhibitory property of biological neurons and thus lifts the linearity off the neuron, making it's activation of a higher order polynomial equation, thus making it more powerful than a standard neuron (Neville, 2016).

(Rumelhart et al., 1988) were the first to introduce the sigma-pi neurons, they posit the notion that several situations may arise when we want input signals to a neuron are multiplied before entering the sum so as to have a gate-effect on each other (Rumelhart et al., 1988). The gate-effect they describe is what is referred to as the pre-synaptic inhibitory property of neurons [both artificial and biological]. They argue that these neurons "*can be used to convert the output level of a unit into a signal that acts like a weight connecting between two units*" and as such increases the compactness of the network, this concept is what state-space expansion takes advantage of and is further discussed in Chapter 5

Binary components of the sigma-pi neuron derive their origins from the work of, (Aleksander, 1965); who created a digital network which replaced weights in a traditional neural network with Boolean functions, these networks were also known as discriminators and allowed the possibility of pattern recognition by using the Boolean functions to "act as a recorder of features seen during

training" (Gurney, 1997), these networks however lose a lot of information during training as it only uses two values (0,1) for each feature of the training data. Gurney in 1989 presented a unified model/definition for the neurons, and defined the Boolean components using bit-streams to define the Boolean components, which is different from the methodology described and used in this research; the methodology used in this research was presented by (Neville R et al., 1999). Gurney was also the first to present training techniques for the neurons in his thesis (Gurney, 1989), he proposed two algorithms to train the neurons:

i. The Backpropagation Algorithm
ii. The Reward-Penalty Algorithm

The backpropagation method algorithm is the algorithm implemented in this research as it is faster than the reward penalty algorithm, but is more mathematically complex.

1) Training Sigma-pi Neurons using the Gradient Descent Algorithm

Feed forward Sigma-pi neural networks are trained using the backpropagation algorithm described in Chapter 2, the weight updates are the same, but the derivatives of the site addresses which is used to calculate the weight update is different, as such, we use an automatic differentiation table as previously discussed in Chapter 2 to illustrate the processes involved during both the feed forward and backpropagation phase of the gradient descent algorithm. The automatic differentiation table for backpropagation in sigma-pi neural networks is shown in Table 5, P_{μ}^{h} denotes the site addresses for the neuron(s) in the hidden layer, Q_{μ}^{o} denotes the site address for the neuron(s) in the output layer, we denote the network error by e, and the activations by a. The inputs to each layer is denoted by y , the hidden layer is denoted by h as a superscript to all the respective variables and the output layer is denoted by o as a superscript for all the respective variables associated with the neurons in the output layer. The update rule is the same as that in training standard neural networks: to multiply the derivative of the weights, denoted by Δw with the learning rate and add the resulting value to the previous weights as shown in (12) repeated below for convenience, where α depicts the learning rate.

$$w = w + \alpha \Delta w \tag{12}$$

Table 5: A depiction of derivatives for the backpropagation of the error gradient for a 2-layer sigma-pi neural network using automatic differentiation

LAYER	VERTICES	VALUE	REVERSE ACCUMULATION
Input	v_0	x_i	$\Delta x_i = \sum_{j \epsilon M_{\mu 1}} \frac{P^h}{x_i} \Delta P^h - \sum_{j \epsilon M_{\mu 0}} \frac{P^h}{(1 - x_{i)}} \Delta P^h$
Hidden	v_1	$P_\mu^h = \prod_{j \epsilon M_{\mu 0}} (1 - x_i) \prod_{j \epsilon M_{\mu 1}} (x_i)$	$\Delta P_\mu^h = \sum_h W_\mu^h * \Delta a_j^h$
			$\Delta W_\mu^h = P_\mu^h * \Delta a_j^h$
	v_2	$a_j^h = \sum_{i,j} W_\mu^h * P_\mu^h$	$\Delta a_j^h = \sigma'(a_j^h) * \Delta y_j^h$
	v_3	$y_j^h = \sigma(a_j^h)$	$\Delta y_j^h = \sum_{j \epsilon M_{\mu 1}} \frac{Q^o}{y_j^h} \Delta Q^o - \sum_{j \epsilon M_{\mu 0}} \frac{Q^o}{(1 - y_{j)}^h} \Delta Q^o$
Output	v_4	$Q_\mu^o = \prod_{j \epsilon M_{\mu 0}} (1 - y_i^h) \prod_{j \epsilon M_{\mu 1}} (y_i^h)$	$\Delta Q_\mu^o = \sum_k W_\mu^o * \Delta a_k^o$
	v_5	$a_k^o = \sum_{j,k} W_\mu^k * Q_\mu^o$	$\Delta W_\mu^o = Q_\mu^o * \Delta a_k^o$
	v_6	$y_k^0 = \sigma(a_k^o)$	$\Delta a_k^o = \sigma'(a_k^o) * \Delta y_k^o$
	v_7	$e_k^o = \frac{1}{2}(t_k^o - y_k^o)^2$	$\Delta y_k^o = (t_k^o - y_k^o) * \Delta e_k^o$
	v_8	e_k^o	$\Delta e_k^o = 1$

IV. BACKGROUND ON STATE-SPACE EXPANSION

A. Overview

In this chapter, the concept of state-space expansion is introduced. The reason why sigma-pi neurons are viable candidates for state-space expansion and why standard neurons aren't is demonstrated. The concept of state-space expansion is then introduced and the reason why it may improve the generalization performance of a sigma-pi neuron is explained.

B. State-space expansion and motivation

State-space can be described as the number of states that a system can hold; in the context of function approximation, for a function $y = f(x)$, variable x in a may hold several values or in this case states.

The motivation for expanding the state-space of an input vector is that it enables us to essentially oversample the input vector (Neville, 2016). When training a neural network with appropriate parameters, if the desired minimum generalization error isn't reached, the next step is usually to increase the number of neurons in the network, which can be interpreted as adding more weights to oversample the input (Neville, 2016). Adding more neurons also has the effect of providing the network with a higher polynomial order that it can use to its advantage. Another constraint the standard neuron has is that it does not have any inbuilt mechanism to enable state-space expansion. The structure of the sigma-pi neuron however makes it possible to implement state-space expansion because the presynaptic inhibitory property of biological neurons is encoded into its architecture, this enables us to make more use of the input vector and oversample it.

State-space expansion in sigma-pi neurons involves increasing the number of sites in the neuron in an effort to oversample the input vector to the neuron and increase its generalization performance. The methodology was first presented by (Neville R et al., 1999).

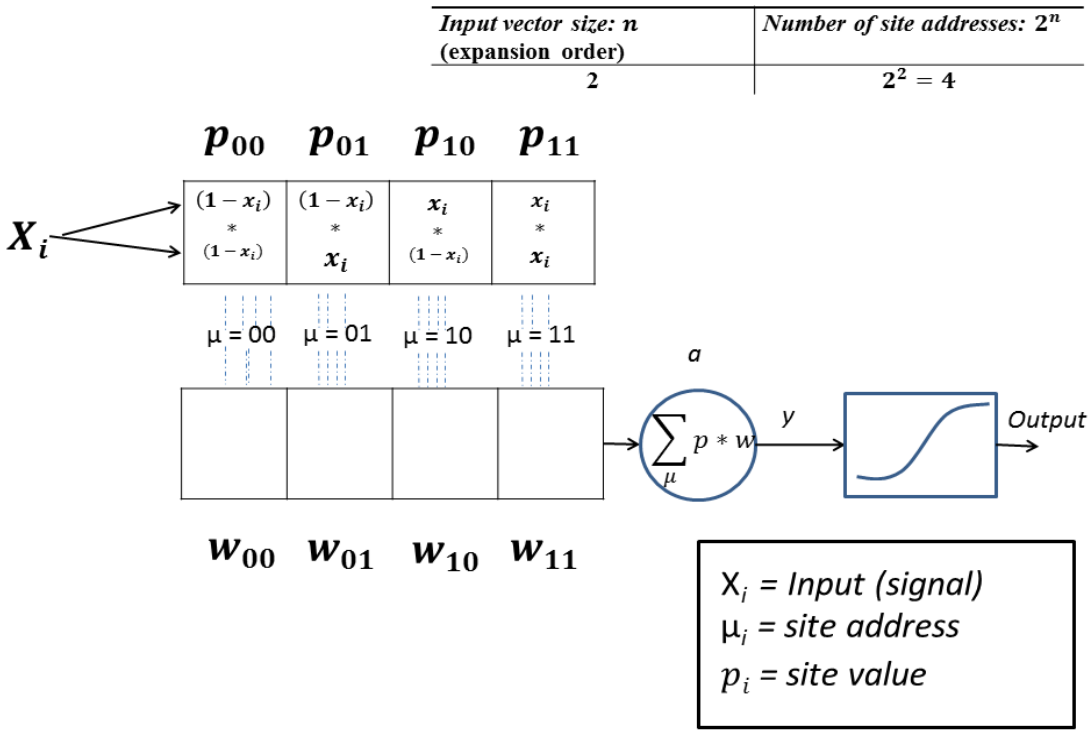


Figure 11: A depiction of 2nd order state-space expansion implemented in a sigma pi unit. The input vector is of size 1 and is replicated twice times as an input to the unit which makes it intrinsically expand the number of sites in the unit to $4(2^n)$ sites.

To increase the number of sites the neuron we oversample the values in the input vector by replicating the input by the order of expansion which intrinsically increases the sites in by the neuron, as the number of sites are initialized by 2^n where n denotes the size of the input vector. An example of this is illustrated in for a one input vector in Figure 11.

The number of times the values in the input vector is replicated is called the order of expansion. To illustrate this further, the expansion of a sigma-pi neuron from 1st order to third order expansion, using a 1-input vector is demonstrated.

1) Zeroth-order Expansion

Under the framework of state-space expansion, we may view standard neurons as sigma-pi neurons with an expansion order of 0 as the there is only one site; this is illustrated in Figure 12.

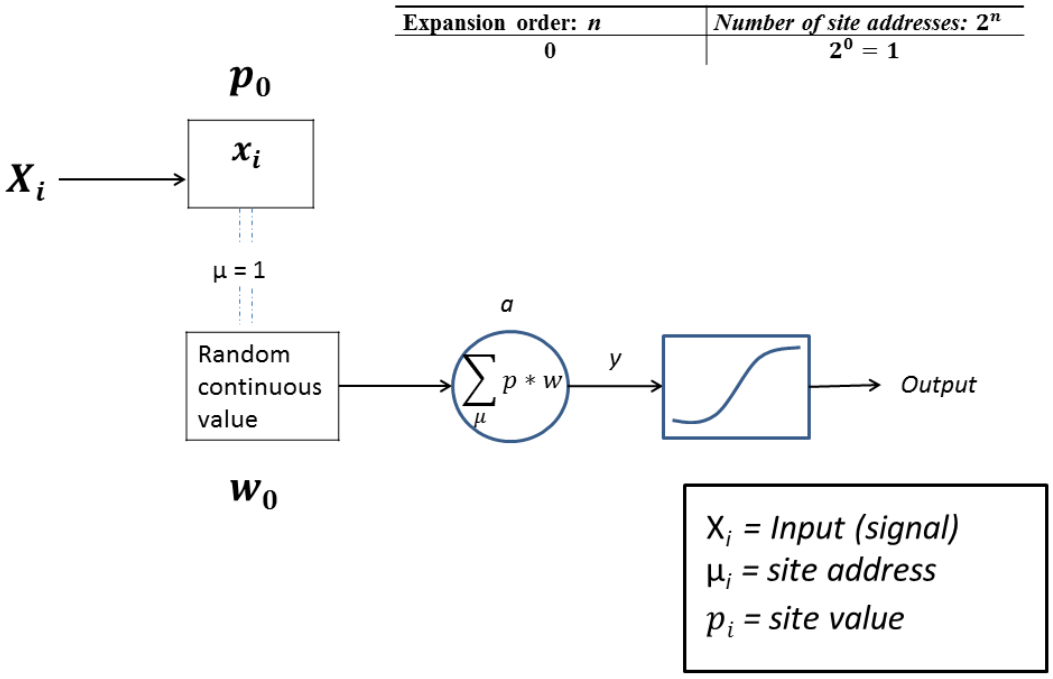


Figure 12: A depiction of a hypothetical 0 order state-space expansion implemented in a sigma pi unit. The unit interprets the order of expansion to be 0 and such intrinsically initializes it's number of sites to 1.

2) First-order state-space expansion

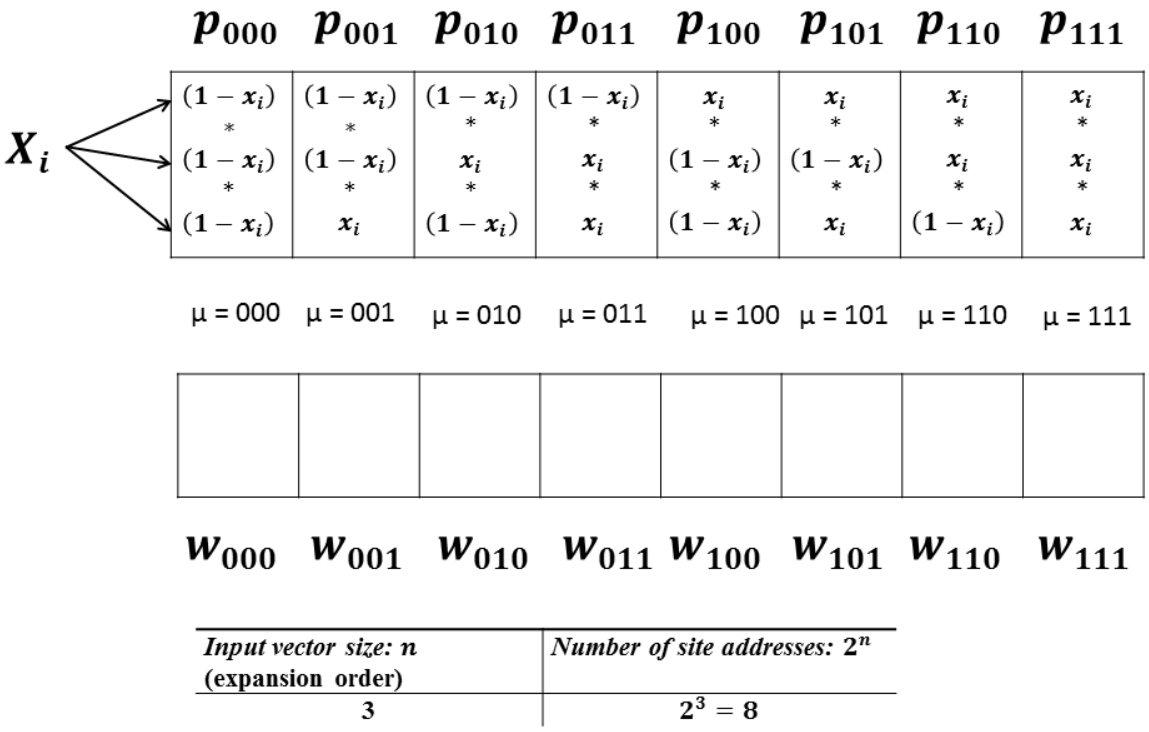
This is the default state of the neuron; Here the values in the input vector aren't replicated because the neurons intrinsically expands the input given because of the presynaptic inhibitory property given encoded into the neuron, given input x_i, the neuron expands it into 2 sites as illustrated in Figure 10.

3) Second-order state-space expansion

As can be seen in Figure 11, the input vector is replicated and as a result, the neuron interprets the input vector to the network to be of size 2 and thus expands the number of sites in the neuron by 2^2 and in so doing expands weights that sample the input to the neuron; This creates a higher polynomial that may be used to address the input to the network and should result in a performance better than that of first order expansion

4) Third-order state-space expansion

With third order state-space expansion, we replicate the input vector to the neuron 3 times and the neuron interprets the input vector to be size 3 and expands its sites to 2^3 addresses as illustrated. This should result in a performance better than that of second order expansion.



Input vector size: n **(expansion order)**	*Number of site addresses:* 2^n
3	$2^3 = 8$

Figure 13: A depiction of 3rd order state-space expansion implemented in a sigma pi unit. The input vector is of size 1 and is replicated three times as an input to the unit which makes it intrinsically expand the number of sites.

V. RELATED WORK

A. *Expansion of Neural Networks*

The activation of a standard neuron is a linear weighted sum of inputs pass through a sigmoid squashing function and as such "*the linearity required immediately places restriction on the node functionality and although it is possible to implement any continuous function using two layers of such nodes, the resources required in terms of hardware and time can be prohibitive.*" Simply put, in order to implement any continuous function using a standard network consisting of one neuron, one has to expand the network, either by adding layers or adding nodes. Using this method, the complexity of the function attempted to be mapped determines the size of the neural network; the more complex the function, the larger the network, the larger the network, the more computational and time resources required.

As sigma-pi neurons capture the pre-synaptic inhibitory property of biological neurons, they "*contain polynomial terms of order greater than one*" as opposed to the standard neuron (Neville R et al., 1999). This makes the sigma-pi neurons more powerful; they have the potential to map functions with polynomial terms which is usually a characteristic of complex functions.

This work mainly draws its inspiration from the work done by (Neville R et al., 1999), in their paper they present a methodology "that partially pre-calculates the weight updates of the backpropagation learning regime and obtains high accuracy function mapping"(Neville R et al., 1999) the aspect of the methodology which obtains higher accuracy function mapping is state-space expansion, as such this study is concerned with reproducing similar results to the ones produced in their paper.

B. *Other Related Work*

There are several algorithms which expand neural network structures by adding neurons to a hidden layer or adding hidden layers to the network structure. Some examples are as follows:

- The "*Pocket Algorithm"* (Gallant, 1990): Gallant in 1990 describes another variant of the perceptron algorithm which he refers to as the pocket algorithm (Gallant, 1990). The pocket algorithm is an optimized version of the perceptron; the mathematical model remains the same, however, the pocket algorithm keeps record of the best weights during iterations and ensures that no updates are made to the weights unless a better set of weights are discovered (Gallant, 1990). This methodology is similar to implementing *state-space expansion* in a single sigma-pi neuron.
- The "*Tower Algorithm"* (Gallant, 1990): In the same paper (Gallant, 1990), the "*tower algorithm"* is described. The first step in this algorithm is to train a neuron/perceptron using

the pocket algorithm and then freeze the weights of that neuron (Gallant, 1990); the second step is to create a new neuron based on the parameters of the previously trained neuron and feed the inputs to the newly created neuron and then run the pocket algorithm on the newly created neuron (Gallant, 1990). The previous step is repeated if the performance of the network's performance improves after the addition of the new neuron, if the performance doesn't improve, the recently added neuron is removed from the network and the algorithm terminates (Gallant, 1990).

- (Schmidhuber, 2015) gives an overview of several deep learning approaches viewing credit assignment which was previously discussed in Chapter 2, as the problem deep learning tries to solve. The credit assignment problem involves determining the amount of credit or blame to attribute to a neuron in a network for the network's overall performance in order to update the weights of the neurons to make the network attain the desired performance. Depending on the problem, training the neurons to attain the desired performance "*may require long causal chains of computational stages.*" (Mnih et al., 2015). (Mnih et al., 2015) describe deep learning as a field concerned with "*accurately assigning credit across many such stages*." (Mnih et al., 2015). In their paper, they present an overview of several deep learning methodologies that handle the credit assignment problem.

VI. IMPLEMENTATION

A. *Overview*

This chapter describes the how the mathematical models described in the background chapters were implemented and highlights some considerations and issues that were realized during the process.

I thought it important to implement all models personally in order to increase my understanding of the subject matters at hand, as such; all models were implemented from scratch.

B. *Experiment Environment*

All models were implemented in MATLAB, a fourth-generation programming language whose target programming demographic is numeric computation. It provides the programmer with tools to easily perform tasks such as matrix manipulation, plotting of data and other numerical computational tasks. One of its advantages is its matrix manipulation syntax which makes code more readable and easier to debug. It has a parallelization feature which can be used to speed up the execution time of programs, in addition, it has libraries which enable creation and manipulations of Microsoft excel documents, making it convenient to keep track of numerical results. These attributes make it a good candidate for the machine learning tasks involved in this study. A good alternative to MATLAB is Python as it offers almost all the same features and has a large community dedicated to managing scientific and numerical libraries.

C. *Implementation of A standard feed forward neural network*

The parameters used to initialize the implemented standard neural network are the following #Pn:

#P1: Learning rate

#P2: A vector representing the number of neurons in the hidden layers; the size of this vector is the number of hidden layers in the network

#P3: Rho

#P4: Number of iterations or epochs

Initially, an object-oriented methodology was used to implement the artificial neural network, there was a class for the neuron and one for the network as a whole, however, during the initial phase of experiments, it was realized that the performance of the network was subpar and posed a risk to the research given the time constraint, as such the object-oriented methodology was disposed and a vectorised methodology was sought. In the final implementation, a vector of weights is used to represent a neuron as opposed to using an object oriented methodology where one may use a class to represent a

neuron; this is because it makes the network easier and faster to train due to the matrix manipulation features of MATLAB.

To provide a framework for creating dynamic structures of the neural network, my implementation requires a vector representing the number of neurons in the hidden layers of the network. This then enables us to run scripts that automatically record the performance of different neural net structures.

Algorithm 2: Stochastic gradient descent for a standard neural network

input: h → vector of hidden layer neuron count
input: α → learning rate
input: ρ→ Rho
input: I → Number of iterations
input: D → Dataset

for *each layer in network* **do**
    initialize neuron weights in layer to a random number using a normal distribution
end for

for *i from 1 to I* **do**
    for *j from 1 to number of samples in dataset* **do**
        $\vec{s}$ *= D(j) //data-sample at index j represented as a vector* $\vec{s}$
        Append bias '1' to $\vec{s}$
        $a \leftarrow feed\ \vec{s}\ though\ network$
        $e \leftarrow labels(j) - a$
        for *k from output layer to input layer* **do**
            if *k is the output layer* **then**
                calculate error gradient using e
                compute deltas for each neuron
                compute weight updates for neurons in k using deltas
            else
                calculate error gradient using deltas in layer k+1
                compute deltas for each neuron

```
15:                 compute weight updates for neurons in k using deltas
16:             end if
17:         end for
18:         update all weights of neurons in network
19:     end for
20: end for
```

D. Implementation of a sigma-pi neural network

The implementation for a sigma-pi neural network differs from that of a standard feed forward network. The differences are as follows:

1. The size of the weight vector for every neuron is 2 raised to the power of the size of the input vector.
2. The input to each neuron is modified into site addresses that model the probability of addressing the weight associated with each site.

Algorithm 3: Stochastic gradient descent for a sigma-pi neural network

input: h → vector of hidden layer neuron count
input: α → learning rate
input: ρ→ Rho
input: I → Number of iterations
input: D → Dataset

for *each layer in network* **do**
initialize neuron weights in layer to a random vector of size 2 raised to the number of inputs in the previous layer
end for

```
1: for i from 1 to I do
2:     for j from 1 to number of samples in dataset do
3:         s⃗ = D(j) //data-sample at index j represented as a vector s⃗
```

```
    set s⃗ to be the input layer //also the first layer
    for i from first hidden layer to output layer do
      Calculate site address vector using vector outputs from previous layer
      Use site addresses to calculate vector outputs for layer i
    end for
    a ← vector ouputs from output layer
    e ← labels(j) − a
    for k from output layer to input layer do
      if k is the output layer then
        calculate error gradient using e
        compute deltas for each neuron
        compute weight updates for neurons in k using deltas
      else
        calculate error gradient using deltas in layer k+1
        compute deltas for each neuron
        compute weight updates for neurons in k using deltas
      end if
    end for
    update all weights of neurons in network
  end for
end for
```

Algorithm 4: calculating the site address vector for a given input vector

input: v → input vector

```
b← site address vector of size 2 raised to power of input vector size
for i from 0 to size of b do
  μ ← boolean conversion of i with string length of input vector
  b(i) ← 1
  for j from 1 to size of input vector do
    if μ(j) equals 0 then
      b(i) ← b(i) ∗ (1 − v(j))
```

8: **else**

9: $b(i) \leftarrow b(i) * v(j)$

10: **end if**

11: **end for**

12: *return b*

13: **end for**

E. Implementation of state-space expansion

In order to expand the state-space of the input vector, we simply duplicate the features of the dataset according to the order of expansion as discussed previously in Chapter 4. The only difference between the algorithms is that we modify the weight space during the initialization as shown in Algorithm 5 before we begin the training phase and modify the function that calculates the site-address values as shown in Algorithm 6.

Algorithm 5: sigma-pi neural network modification with state-space expansion implemented

input: h → vector of hidden layer neuron count

input: α → learning rate

input: ρ→ Rho

input: I → Number of iterations

input: D → Dataset

input: S → Expansion order

for *each layer in network* **do**

initialize neuron weights in layer to a random vector of size 2 raised to the number of inputs in the previous layer times expansion order

end for

Algorithm 6: calculating the site address vector for a given input vector with state-space expansion implemented

input: S → Expansion order
input: v → input vector

for *i from 1 to S* **do**
  for *each feature in* **v do**
    add new value of current vector value next to current vector value in vector
  end do
end do

1:
2: $b \leftarrow$ *site address vector of size 2 raised to power of input vector size times the expansion order*
3: **for** i *from 0 to size of b* **do**
4: $\mu \leftarrow boolean\ conversion\ of\ i\ with\ string\ length\ of\ input\ vector$
5: $b(i) \leftarrow 1$
6: **for** *j from 1 to size of input vector* **do**
7: **if** $\mu(j) equals\ 0$ **then**
8: $b(i) \leftarrow b(i) * (1 - v(j))$
9: **else**
10: $b(i) \leftarrow b(i) * v(j)$
11: **end if**
12: **end for**
13: *return b*
14: **end for**
15:

Because some of my experiments only deal with function approximation tasks, the output space of the dataset may not be limited to a range between 0 and 1, as such, the sigmoid function will not suffice, and

therefore an identity function is used as the activation function of the neurons in the output layer so as to eliminate the squashing effect of the sigmoid function as illustrated in Figure 14 as recommended by (Nguyen-Thien and Tran-Cong, 1999). For experiments dealing with pattern recognition, we use a sigmoid function throughout all layers.

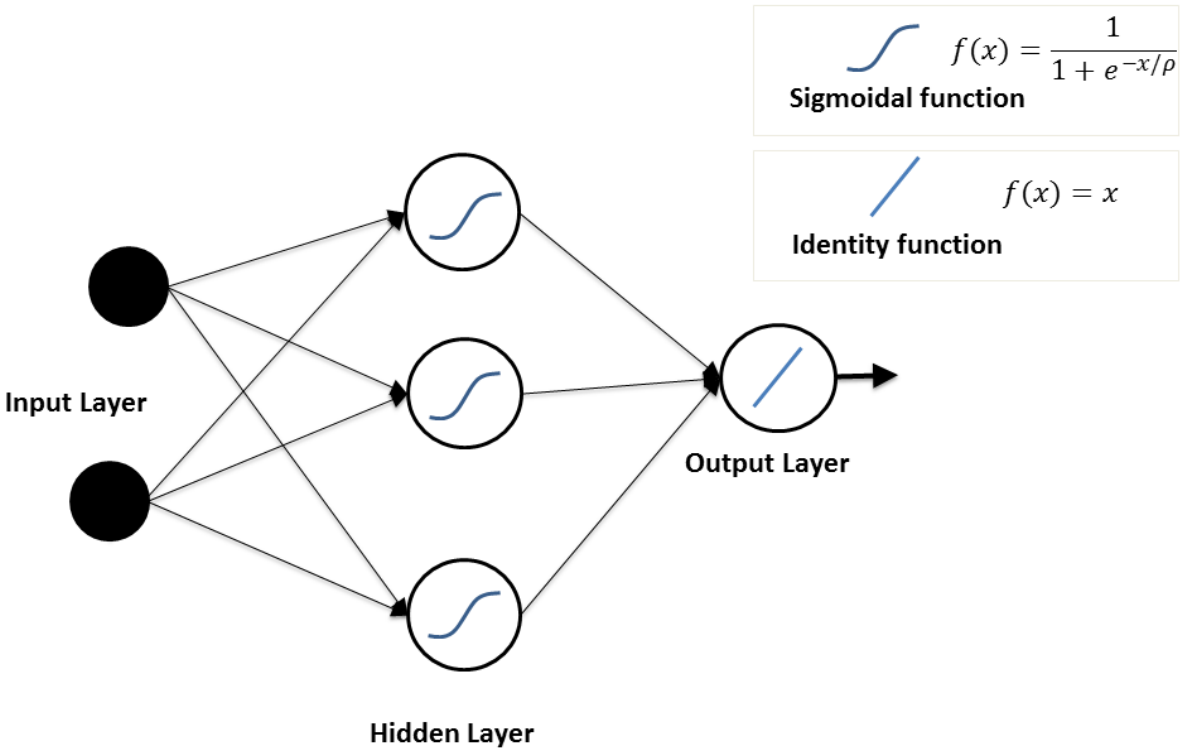


Figure 14: An illustration of the activation functions used in a neural network for function approximation. The sigmoid function will not suffice our needs, and therefore we use an identity function as the activation function of the neurons in the output layer

VII. EVALUATING THE PERFORMANCE OF NEURAL NETWORKS

A. Overview

This chapter provides a description on how the performance of neural networks is evaluated. It starts by describing the generalization performance of a neural network which is the standard way to evaluate neural networks and also provides a description of sensitivity analysis and why it is relevant to this study is provided.

B. Generalization

In the context of supervised learning, a neural network's task is to find the underlying form/pattern of a dataset by using an appropriate algorithm to adjust the network's weights in order to correctly map the vectors from the input space of the dataset to the output space as accurately as possible, as such we train a network on a dataset until it reaches a satisfactory minimum generalization error also known as the mean squared error, defined by equation (3) repeated below for convenience.

$$mean\ squared\ error = \frac{1}{n}\sum_{p=1}^{P=n}(t^p - y^p)^2 \tag{3}$$

However, the purpose of training the neural network is to use it as a tool for prediction of vectors in the input space that are not present in the dataset, therefore, the standard way of training the neural network is to partition the entire dataset into 3 parts #Pn:

#P1: A training set that consists of the vectors of the dataset that we wish to use to train the network.

#P2: A validation set that consists of vectors that are used to evaluate the performance of the network; none of which are present in the training set and testing set.

#P3: A test set, consisting of vectors that are used to evaluate the overall performance of the neural network; none of which are present in both the training set and the validation set.

This method of evaluating the performance of neural networks is called *cross- validation* (Gurney, 1997)and is valid for both pattern recognition tasks and function approximation tasks.

In the context of pattern recognition, a network is said to generalize well if it is able to adjust its parameters[weights] well enough to create a mathematical model represented by the neural network that captures the form or pattern of the dataset, as such there are instances where we may not be too keen on the network's accuracy. Consider the pattern recognition task presented in Figure 15, the task of a neural network is to come up with a decision boundary as shown in Figure 16 because it follows the trend of the data shown in Figure 17, however, if we train the network using a large number of iterations, it is

said to over-fit the dataset and comes up with a decision boundary that over-fits the data as shown in Figure 18.

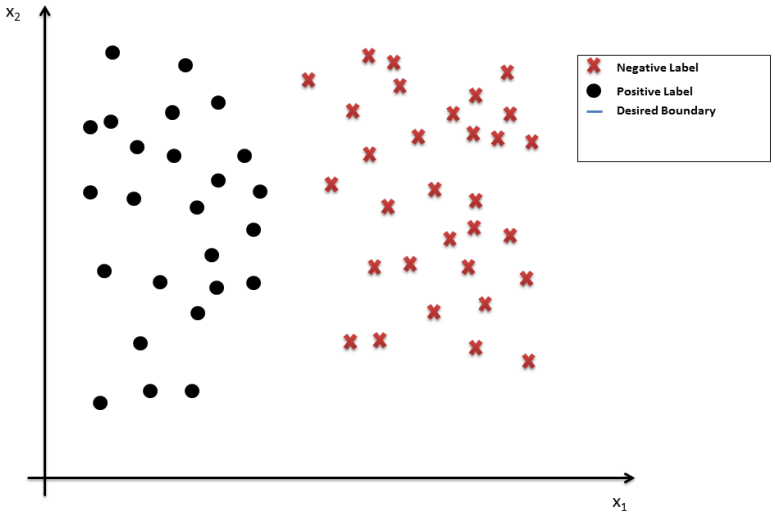


Figure 15: Depiction of a hypothetical pattern recognition dataset whose pattern is intended to be learned by a neural network. The red crosses and circles denote the different classes[or labels] in the dataset. All points on the graph are data-samples the axes of the graph represent different features of the data-samples. The goal of a network would be to create a decision boundary that separates the classes.

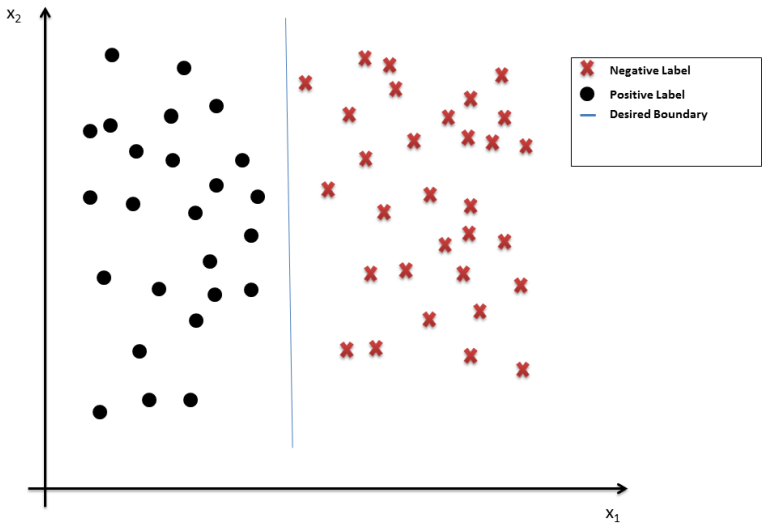


Figure 16: Depiction of the desired decision boundary that models the trend of the data-samples in a hypothetical pattern recognition dataset whose pattern is intended to be learned by a neural network. The image serves to illustrate the directions in which the data-samples of the dataset trend; this is obvious to the human observer, but is not to a machine. The task of a neural network is to find a decision boundary encoded in a mathematical model that manages to preserve the trend of the distinct classes for the data-samples.

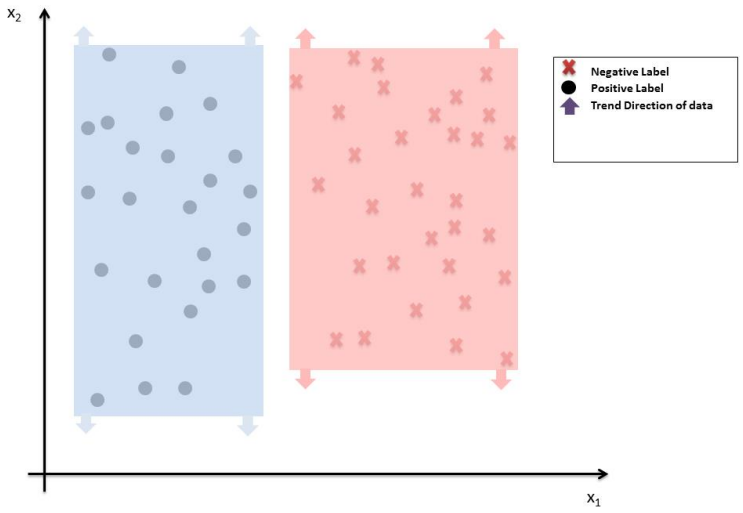


Figure 17: Depiction of the trend of the data-samples in a hypothetical pattern recognition dataset whose pattern is intended to be learned by a neural network. The image serves to illustrate the directions in which the data-samples of the dataset trend; this is obvious to the human observer, but is not to a machine. The task of a neural network is to find a decision boundary encoded in a mathematical model that manages to preserve the trend of the distinct classes for the data-samples.

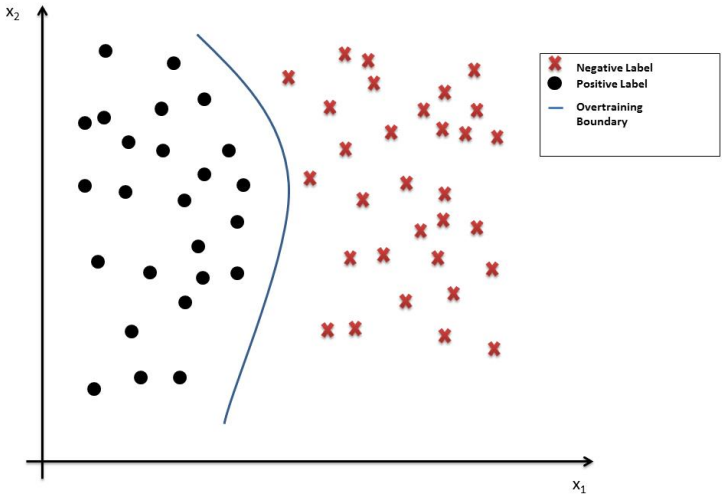


Figure 18: Depiction of a decision boundary that may result from overfitting a neural network on a hypothetical pattern recognition dataset. This decision boundary is not desirable as it does not preserve the trend of the classes in the data-samples.

This methodology of evaluating the performance of neural networks is known as cross-validation and is very useful when dealing with datasets that pose a pattern recognition task; it is worth noting that there are more methods that can be used to evaluate the performance of a neural network, in pattern recognition such as *receiver operating characteristics(ROC) analysis* (Fawcett, 2006) which involves performing an analysis on the data-samples that were misclassified by the decision boundary arrived at by the neural network, this is does not prove to be useful in function approximation tasks as there is no classification being made, only approximation of functions.

However, with regards to datasets that pose a function approximation task, ROC analysis may not suffice as the geometric space problems are quite different; in pattern recognition tasks, the goal of the network, from a high level point of view, is to find the best parameters of the network that create hyperplane to separate all the classes of the dataset however with function approximation, the task of the network is to predict the value of an input to a function such that the networks predicted value follows the trend of the function in the Euclidean space as illustrated in Figure 19. This can be either done by interpolating or extrapolating points on the function trend in the Euclidean space; neural networks are not expected to perform properly on extrapolation tasks as the activation functions of the neurons provide a limitation on the network, as such, my experiments focus on using interpolation to evaluate the networks performance.

It should be noted that overfitting is not a problem in the function approximation as there are no decision boundaries in the context of function approximation and we do want the neural network to come up with a mathematical model learns the function as accurately as possible.

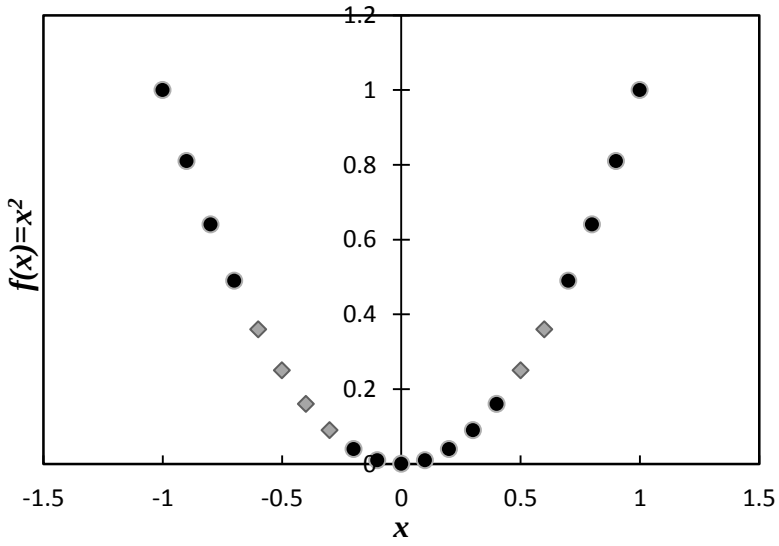


Figure 19: Interpolation and extrapolation of points as a function approximation task. The dataset represented is an x^2function. The goal of a neural network would be to interpolate the diamond data-samples, given a training set consisting of the circular data-samples.

As the study at hand evaluates the state-space expansion methodology, the generalisation performance of the network is used as a metric of evaluation; I pay attention to the trend of the generalisation error of the network as the state-space of a sigma-pi neural network is expanded. I also compare this form of expansion with expanding the hidden layer of a standard neural network using the generalisation performance observed during both phases of expansion as a metric of comparison.

An important consideration to be made when evaluating the performance of neural networks is how sensitive the networks is to the parameters that are set, how the parameters are set and how to sample the performance in order to ensure that the results are not custom to a particular initialization of parameters, this is covered in the next section.

C. Hyper-parameters and sensitivity analysis

In the context of machine learning, a hyper-parameter is a parameter that cannot be obtained from the training phase of a model, such parameters have to be manually set by humans and as such, to evaluate the performance of any machine learning model, it is necessary to analyse and understand how sensitive the model is to all hyper-parameters of the model, this usually involves keep all but one of the parameters fixed, changing the range of the unfixed parameter and then observing the performance of the network, this is discussed in the next section, but before then, we explain the hyper-parameters that are associated with a feed forward neural network that using gradient descent/backpropagation to train as it pertinent to the experiments conducted in this study

1) Hyper-parameters for feed-forward neural networks

1. Learning Rate: The learning rate determines how large the update to the weights is. From a high-level point of view, it determines how large the steps are in gradient descent.
2. Iterations/Epochs: This parameter determines the number of times an update is made to the weights in the network. Usually, the higher it is, the more likely it is for the network to improve its accuracy.
3. Rho: This parameter determines the shape of the sigmoid function and in so doing determines how steep the threshold for activation is. Smaller values tend to make for a steeper/stricter slope while higher values tend to make for a more relaxed/lenient slope.

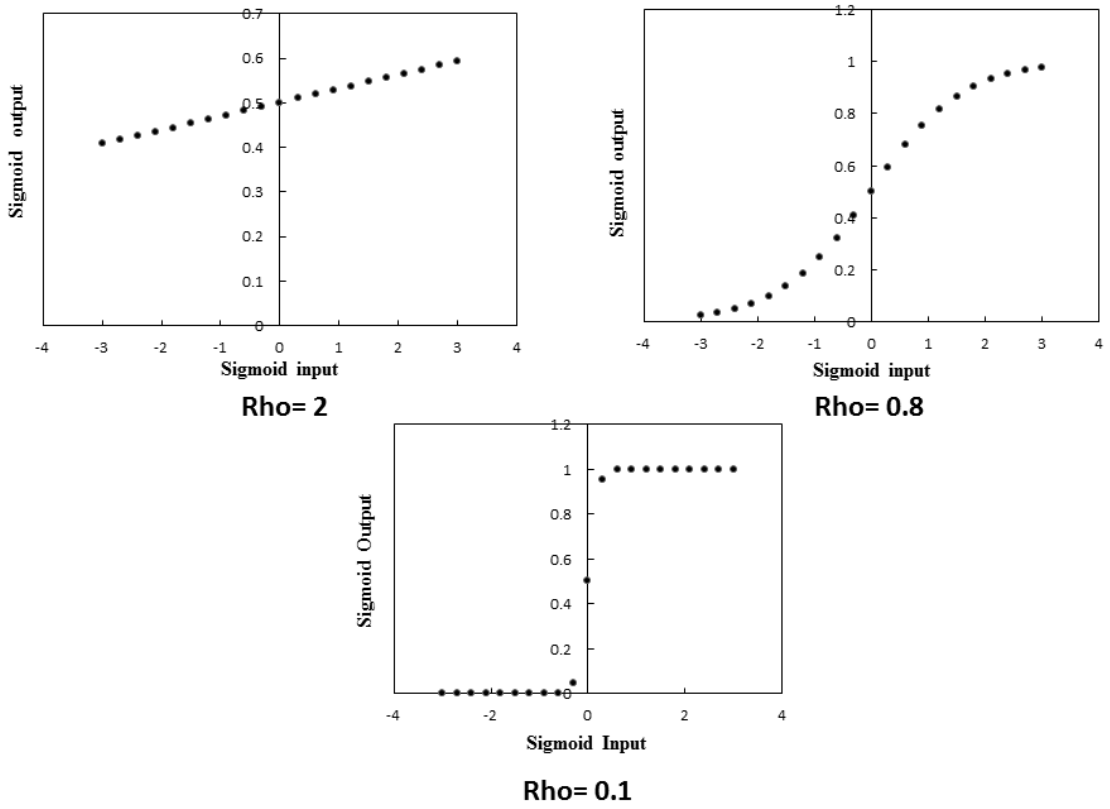


Figure 20: An illustration of the effects of the rho parameter on the shape of the sigmoid function. As can be seen by the shape of the sigmoid when rho is equal to 2, there is lenient threshold for the activation function as opposed to the shape when rho is 0.1 which makes for a very steep threshold.

4. Structure of the Neural net: This parameter usually depends on how the network is implemented. Some implementations of a neural network are fixed, while others are made to be dynamic. As expansion of neural networks is pertinent to this study, it is important that this parameter is dynamic so as to evaluate its performance. This parameter consists of two variables #Vn:

    #V1: the number of layers in the network

    #V2: the number of neurons in each layer

Usually, the bigger the network is, the more likely it is to be accurate. However, the bigger the network, the more iterations required to train it to reach a satisfactory minimal error.

Despite the several hyper-parameters involved with neural networks discussed above, the learning rate seems to affect the performance of the network most significantly most likely because it directly affects the updates to the weights.

2) Sensitivity Analysis

Sensitivity analysis involves finding how sensitive a model is to its parameters. Sensitivity analysis is a more general term, in the context of machine learning it is usually referred to as hyper-parameter optimization (Bergstra and Bengio, 2012). There are several methods that may be used to perform sensitivity analysis they are as follows #Mn:

#M1: Grid search

#M2: Bayesian optimization

#M3: Random search

The grid search is widely used and preferred (Bergstra and Bengio, 2012) and is used in this study as it suffices to meet the needs of the experiments involved. Grid search involves sampling the generalization performance of the network over a manually specified subset of the hyper-parameter space and taking note of the value or range of the parameter that gives the minimum error over all the recorded samples. This approach is used in all the experiments to find the best parameters to train the networks.

VIII. EXPERIMENTS AND RESULTS

A. Overview

This chapter describes the experiments and results observed during the course of the study; experiments were performed using three datasets #Dn:

#D1: The AND datasets: this dataset represents a binary/logical function which outputs the value of 1 only if the input vector to the function consists of only values of 1; otherwise, it outputs 0.

#D2: The XOR dataset: this dataset represents a binary/logical function which outputs the value of 1 only if the input vector to the function consists of only one value of 1 otherwise, it outputs 0.

#D3: The x^2 dataset: this dataset consists of data-samples from the x^2 function.

The AND and XOR datasets are Boolean functions but are treated as pattern recognition tasks. The datasets then serve to represent 3 types of tasks and problems #Pn

#P1: A linearly separable dataset: in terms of the Euclidean geometry of the dataset. This is the AND dataset.

#P2: A linearly inseparable dataset: in terms of the Euclidean geometry of the dataset this is the XOR dataset

#P3: A function approximation dataset.

Where #P1 and #P2 can be considered as pattern recognition tasks, as a decision boundary can be formed by the neural network to learn the function.

During all experiments, the networks were trained using the *stochastic gradient descent algorithm*.

Before running the expansion experiments on any dataset, a sensitivity analysis of the network with respect to its parameters is carried out in order to select ideal values to train the network. It should be emphasized that during all experiments, the performance of a standard neuron is regarded as the same performance of a sigma-pi neuron with 0^{th} order expansion as previously discussed in chapter 4.

1) Note on graph representations

The results of all experiments are presented using surface area graphs. On each graph, the trend of worst performance of the model/network being analysis in annotated by **A**, the average annotated by **B** and the worst is annotated by **C**. Most experiments presented involve observing the performance of a model/network as one its parameters are varied; for each parameter value, the network is initialised with random weights and we record its performance over 40 individual runs, for each run, we record the best, worst and average performances as previously mentioned. The trend lines of the best, worst and average performances are then plotted on the surface area graph with each trend line represented as the threshold of a distinct area of the graph. This serves to illustrate the distribution of the model/networks

performance over the 40 individual sample runs. The graphs are initially presented in relatively small sizes for the convenience of the reader, however clearer and larger images can be found in the appendix.

B. AND dataset experiments

The AND dataset consists of only four data-samples as it is a binary function. This is illustrated in Figure 21 and Figure 22. This dataset consists of very few data points, as a result, cross-validation cannot be used to evaluate the performance of the network; to evaluate the performance, the networks are trained and tested on all data-samples and the generalisation performance of the network is calculated using the networks generalisation error shown in (14) where n represents the number of data-samples, t^p represents the actual output of the AND binary/logical function given a data-sample at p , y^p is the predicted output by the network

$$generalisation\ error\ e_g = \frac{1}{n}\sum_{p=1}^{P=n}(t^p - y^p) \tag{14}$$

The generalisation error is inversely proportional to the networks performance i.e. the lower the generalisation error, the better the performance of the model and vice versa. Figure 22 illustrates the labels of the data-samples in a Euclidean space and the desired decision boundary

	Input space		Output space
	X_1	X_2	Y
Data Sample	0	0	0
	0	1	0
	1	0	0
	1	1	1

Figure 21: Depiction of the AND dataset.

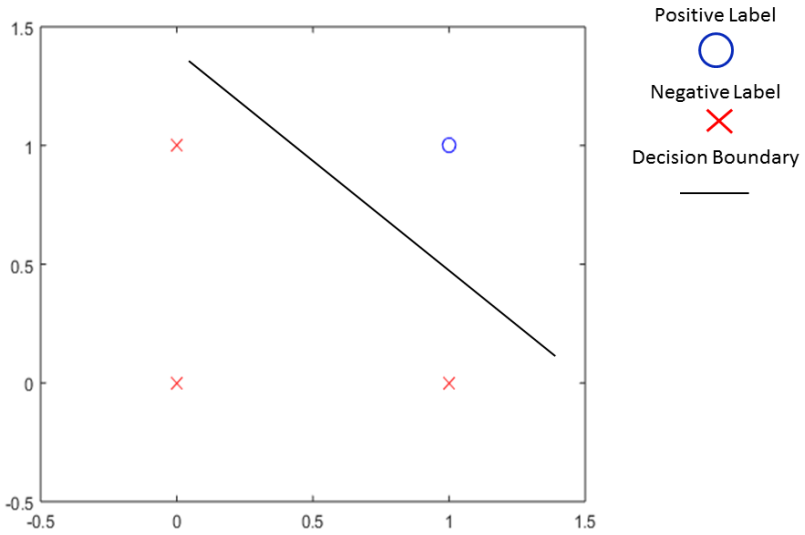


Figure 22: Depiction of the labels of the data-samples in the AND dataset in a Euclidean space. The axes represent the features of the dataset. The crosses denote the data-samples with the label/class 0 and the circles denote the data-samples with the label/class 1. The figure also serves to illustrate the desired decision boundary that can be used to fit the dataset.

1) Sensitivity analysis for ideal learning rate

A grid search sensitivity analysis was performed to find an ideal value for the learning rate, during sensitivity analysis, the model is trained with 100 iterations and tested on the validation set as the learning rate parameter is varied from 1 to 0.001; for each iteration, the process is repeated over 40 individual runs and best, worst and average performance observed are recorded; the trend of the best, worst and average performance over all 40 runs for each iteration is then annotated on the graph by **C, A** and **B** respectively. As can be seen from Figure 23, any learning rate between the range of 0.04 and 0.46 would suffice to make the network converge to a minimum error.

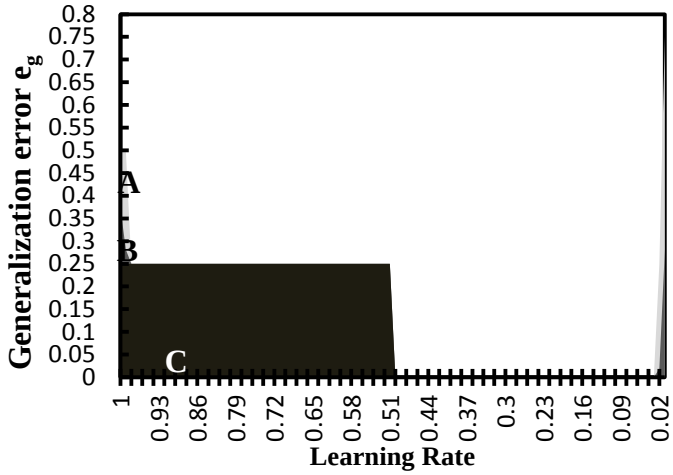
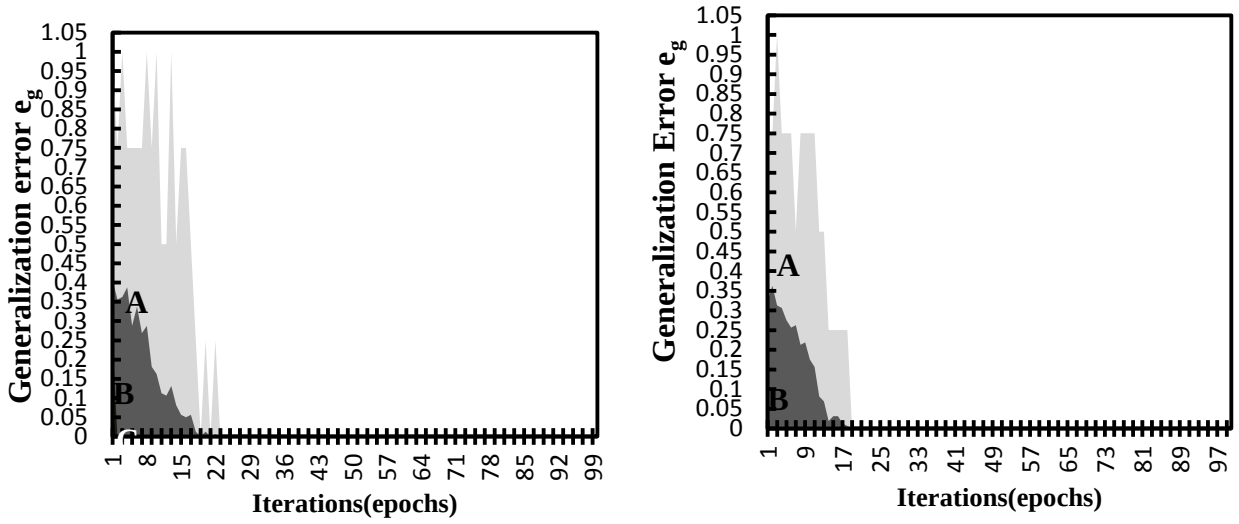


Figure 23: Sensitivity Analysis for the best learning rate value that can be used to fit AND dataset. The horizontal axis represents range of learning [1, 0.01] which was used in testing, the vertical axis represents the range of the generalisation error over a sample of 40 independent runs and 100 iterations during each run; **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 independent runs; **C** annotates the best performance of the model over 40

2) Experiments

From the experiments performed, it was observed that single standard neuron suffices to fit the AND dataset, as such no expansion experiments with a standard neural network is performed. Figure 24 illustrates the performance of both a standard neuron and a sigma-pi neuron with first order state-space expansion as the number of iterations is increased from 1 to 100; for each iteration, a sample of 40 individual runs are performed and the best, average and worst performance over the 40 runs is recorded. From Figure 24 **(1)**, we can observe that the standard neuron converges to a minimum error with only 25 iterations and Figure 24 **(2)**, we can observe that the sigma-pi neuron with first order state-space expansion converges to a minimum error with only 19 iterations. In both graphs, **A** annotates the worst performance of the model, **B** annotates the average performance of the model and **C** annotates the best performance of the model over 40 independent runs as the number of iterations is increased. It is also observed that the sigma-pi neuron is capable of converging to a minimum error over all 40 independent sample runs regardless of the number of iterations, because the worst performance trend of the neuron that is intended denoted by C is non-existent.



(1) **(2)**

Figure 24: Depiction of the performance of a standard neuron and a sigma-pi neuron with 1st order state-space expansion over the AND dataset. **(1)** denotes the performance of the standard neuron and **(2)** denotes the performance of a sigma-pi neuron. In both graphs, **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best

C. XOR dataset experiments

The XOR dataset consists of only four data-samples as it is a binary function. This is illustrated in Figure 25 and Figure 26. This dataset also consists of few data points, as a result, cross-validation cannot be used to evaluate the performance of the network; to evaluate the performance, the network is trained on all data-samples and tested on the same data-samples. The network's generalisation error is then calculated using (14).

	Input space		Output space
	X_1	X_2	Y
Data Sample	0	0	0
	0	1	1
	1	0	1
	1	1	0

Figure 25 Depiction of the XOR dataset.

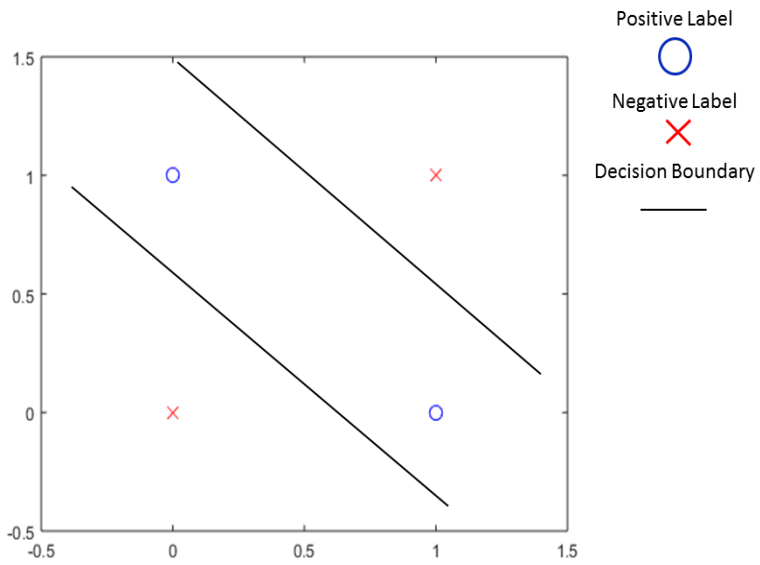


Figure 26: Depiction of the labels of the data-samples in the XOR dataset in a Euclidean space. The axes represent the features of the dataset. The crosses denote the data-samples with the label/class 0 and the circles denote the data-samples with the label/class 1. The figure also serves to illustrate the desired decision boundary that can be used to fit the dataset.

1) Sensitivity analysis for ideal learning rate

A grid search sensitivity analysis was performed to find an ideal value for the learning rate. When testing the model for the rho and learning parameters, the model is trained with 100 iterations and validated with the validation set over 40 individual runs and observe the trend of the model's performance. As can be seen from Figure 27, the performance of the network improves from a learning rate of 0.3 to 0.01; I decide on using a learning rate of 0.1 to train the networks in order to make larger updates to the weights of the network given that the dataset is small.

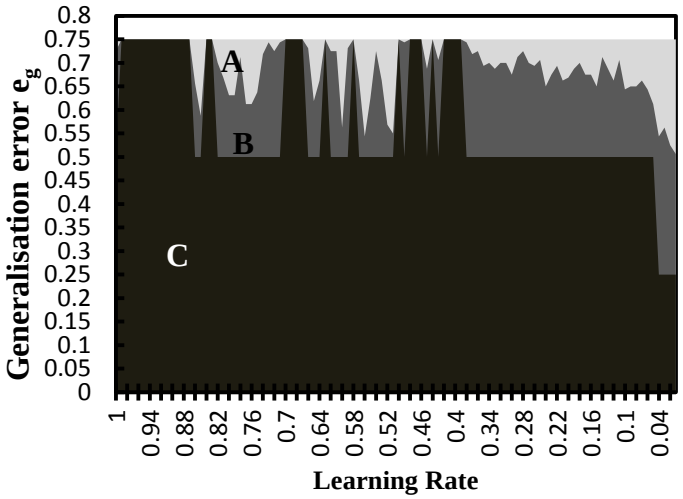


Figure 27: Sensitivity Analysis for the best learning rate value that can be used to fit AND dataset. The horizontal axis represents range of learning [1, 0.01] which was used in testing, the vertical axis represents the range of the generalisation error over a sample of 40 independent runs and 100 iterations during each run; **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs.

2) Experiments

The results of the experiments performed are shown in Figure 28 and Figure 29. The results show that in order to make a standard neural network converge to a minimum error, we need to expand its hidden layer to 2 neurons. This is not the case for sigma-pi neurons as we can see from Figure 29 that we can fit the function using just one sigma-pi neuron with first order state-space expansion and 19 iterations. It should also be noted that even though we can converge a standard neural network to a minimum error with 2 hidden layer neurons and 500 iterations, it's behaviour still proves to be erratic as illustrated by the trend of the worst performance over 40 independent runs for every iteration. This is most likely due to the random initialisation of weights in the network. Unlike the standard neuron, the sigma-pi neuron seems to be more stable; we can see it requires about 19 iterations to fit the dataset with a guarantee at all times because after 19 iterations, we do not observe the worst performance trend line denoted by A anymore over any of the 40 independent runs.

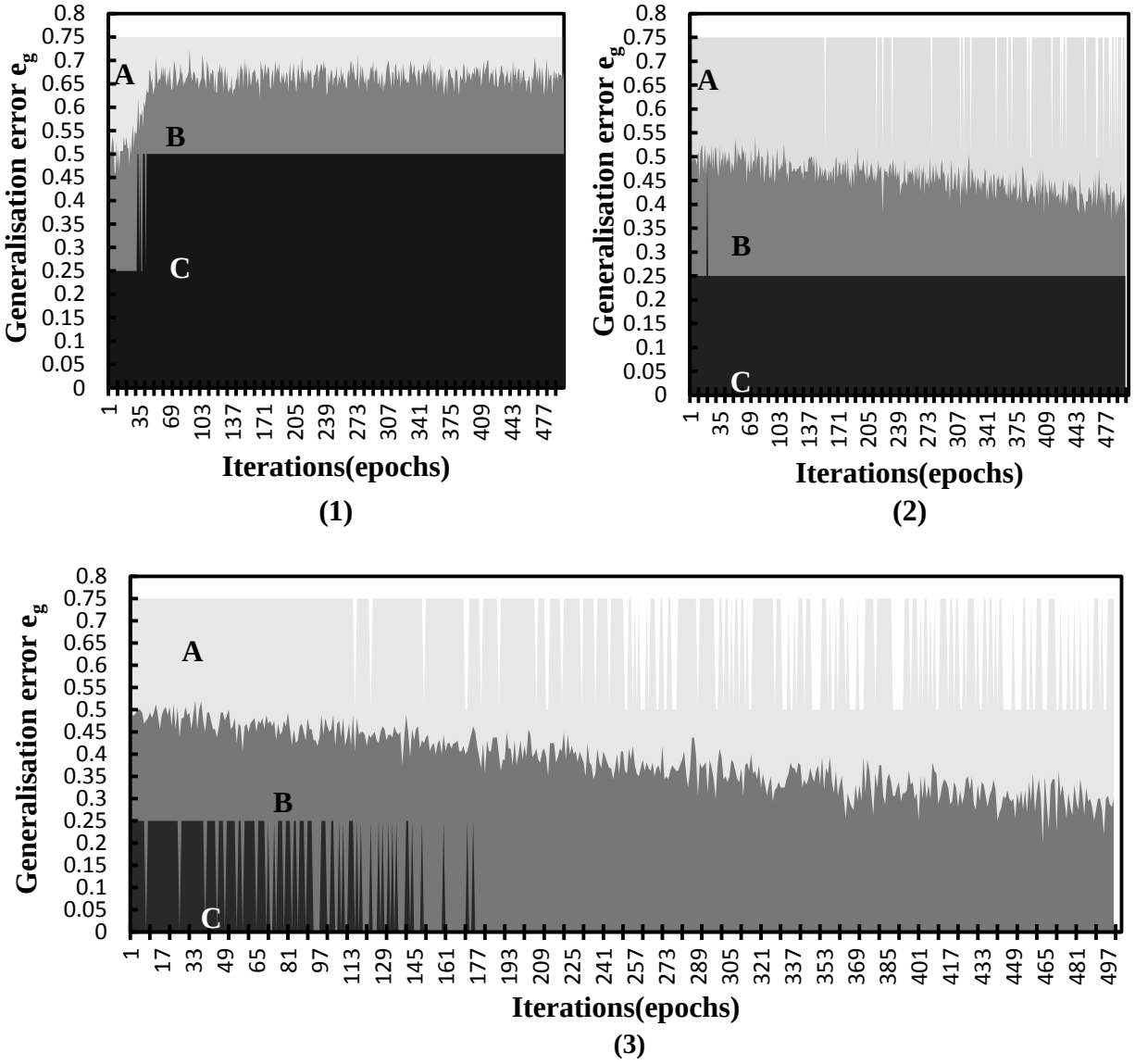


Figure 28: Depiction of the performance of a standard neural when trained on the XOR dataset a standard neural network. **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs. **(1)** illustrates the convergence of the generalisation error when using one neuron to train the function (**2)** illustrates the convergence of the generalisation error when using a standard neural network with one neuron in the hidden layer and **(3)** illustrates the convergence of the generalisation error when using a standard neural network with two neurons in the hidden layer. **(3)** illustrates that we need at least 2 neurons in the hidden layer to fit the function as it can be seen from **(1)** and **(2)** that both networks get stuck in a local minima.

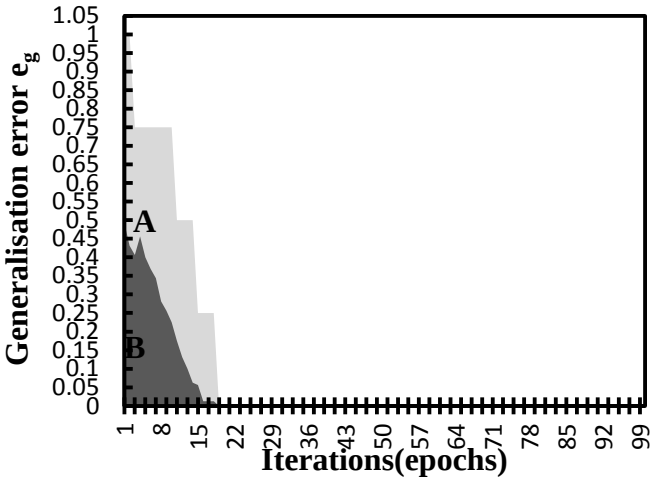


Figure 29: Depiction of the performance of a sigma-pi neuron with 1st order state-space expansion when trained on the XOR dataset. As we can see it requires about 19 iterations to fit the dataset with a guarantee at all times because after 19 iterations, we do not observe the worst performance trend line denoted by A anymore. The performance of sigma-pi neuron seems to be more stable compared to that of a standard neural network.

D. X^2 dataset experiments

These experiments involve using data-samples from the x^2 function to evaluate the performance of the networks as they expanded. This section begins by providing an overview of how the parameters for each experiment by showing the sensitivity analysis carried out for the dataset and then describing each experiment performed.

The experiments carried with the x^2 function dataset are in twofold: #Fn:

#F1: An observation on how well the trained models perform when trained with a sparse dataset.

#F2: An observation on how well the trained models perform when trained with a smoother dataset.

I observe how the generalisation ability of the models changes as the networks are expanded. My findings suggest that state-space expansion does improve the generalization performance of sigma-pi neurons under certain conditions, however, the overall behaviour observed is erratic due to random initialisation of weights; this is expanded upon in the next chapter where the results are summarised and discussed.

The performance of a typical standard neural network is used as a benchmark to evaluate the performance of the sigma-pi neural network; as such no any excess tuning of parameters for the sigma-pi neural networks is used, as the methodology is intended to improve the performance of the models.

I use the stochastic gradient descent algorithm to train the network models and use the error modulus depicted in (15) used originally by (Neville R et al., 1999) as a measure of performance of each models performance. The error modulus is what is plotted in the graphs presented. In (15), n represents the

number of data-samples, t^p represents the actual output of the x^2 function given a data-sample at p , y^p is the predicted output by the network.

$$error\ modulus\ |e| = \sqrt{\frac{1}{n}\sum_{p=1}^{P=n}(t^p - y^p)^2} \tag{15}$$

The datasets used for testing are in two fold, one dataset for experiments with sparse data points, and one with closer and more data points, so as to observe if the topology of the samples of the function provided to the network affects the performance of the network.

1) Sparse Dataset

The training data for the sparse dataset consists of 15 vectors over the range of [-1, +1] with an interval of 0.1 with a set of missing samples, called the test set which is what the trained model is evaluated on. The task of a trained model would be then to interpolate the missing points correctly; this is illustrated in Figure 31.

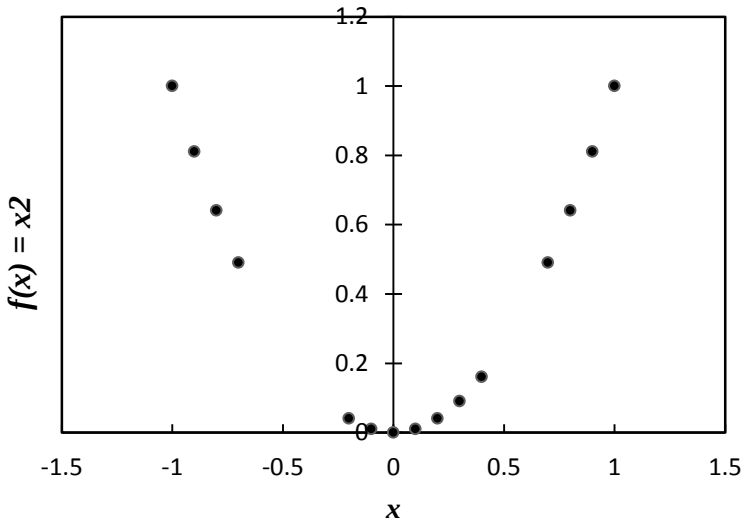


Figure 30: Training and validation data for sparse dataset. The figure illustrates the data-samples that were used to train the model. The horizontal axis represents the range of *x* the input the function: x^2 we are trying to approximate with our model. The vertical axis represents the range of the output of the function we are trying to approximate. During the training phase, we train the network using some of the data-samples shown above, during the validation phase, we test the network on the data-samples that were not used during the training phase.

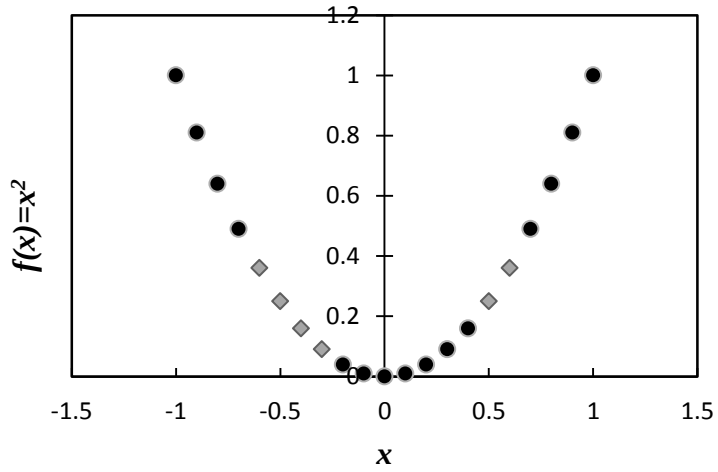


Figure 31: Training, validation and test data-samples for the sparse dataset. The data-samples denoted by the circles are the samples that are used to train the model. The samples that are denoted by the diamonds are the samples that are used to evaluate the generalization performance of the model. The model task of the model is to interpolate/approximate the samples denoted by a diamond which is also the test set.

2) Smooth dataset

The training data for the smooth dataset consists of 21 vectors over the range of [-1, +1] with an interval of 0.1. When evaluating with this dataset, we test on an entirely different set which is still over the range of [-1, +1] with an interval of 0.3, this is still an interpolation task to be carried out by the model. This is illustrated further in Figure 33.

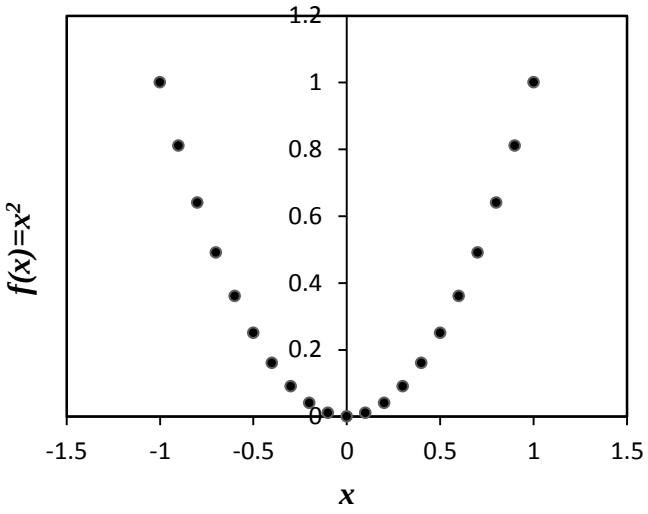


Figure 32: Training and validation data-samples for smooth dataset. The figure illustrates the data-samples that were used to train the model. The horizontal axis represents the range of x the input the function: x^2 we are trying to approximate with our model. The vertical axis represents the range of the output of the function we are trying to approximate. During the training phase, we train the network using some of the data-samples shown above, during the validation phase, we test the network on the data-samples that were not used during the training phase.

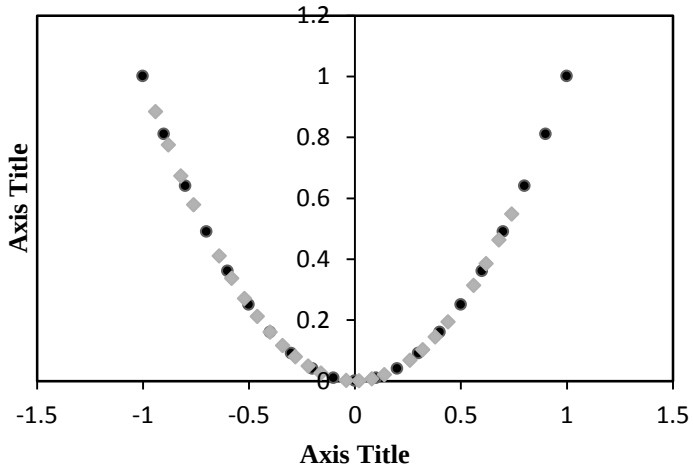


Figure 33: Training, validation and test data-samples for smooth dataset. The data-samples denoted by the circles are the samples that are used to train the model. The samples that are denoted by the diamonds are the samples that are used to evaluate the generalization performance of the model. The model task of the model is to interpolate/approximate the samples denoted by a diamond.

3) Sensitivity analysis for the standard neural network implementation

A grid search sensitivity analysis was performed for the following parameters which the model's accuracy/generalisation performance is sensitive to #Pn:

#P1: Rho: this parameter affects

#P2: Learning Rate:

#P3: Iterations:

When testing the model for the rho and learning parameters, a low number of iterations over a sample 40 individual runs is used to observe the trend of the model's performance as it suffices to only pay attention to the parameter values that give the lowest error modulus i.e. the network's best performance during the experiment. The figures plot the best worst and average performance of the model over the sample of runs for each parameter of the value in order to illustrate the spread of data so as not to come up with biased interpretations of the models performance; As a convention, the worst performance trend of the network is annotated by **A** on each figure, the average performance is annotated by **B** and the worst performance is annotated by **C.**

a) Sensitivity Analysis for Rho

From Figure 34, we can deduce that the lower the value of rho is, the higher error modulus is on all best, average and worst performances of the model; as such do not alter the value of rho from 1 when evaluating the performance of the model. Table 6 provides an overview of the constant variables used during the process.

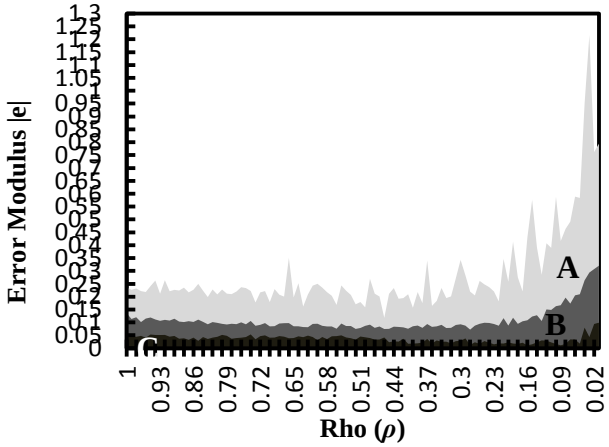


Figure 34: Sensitivity Analysis for the best value of rho used to fit an x^2 function with a standard neural network. The horizontal axis represents range of rho which is [1, 0.01], the vertical axis represents the range of the error modulus over a sample of 40 independent runs and 100 iterations during each run; the rest of the parameters used during each run are illustrated in Table 6. **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs.

Table 6: Constant parameters used during Sensitivity Analysis of rho iterations to train a standard neural network

Parameter	Value
Sample size for each value of rho	40
Layer Count	2
Neurons in Layer 1	5
Neurons in Layer 2	1
Learning Rate	0.1
Iterations/Epoch	100

b) Sensitivity Analysis for Learning Rate

From Figure 35, we can deduce that the model performs at its best when the learning rate is in a range of [0.6, 0.9]. I conclude on a learning rate of 0.6 for all experiments.

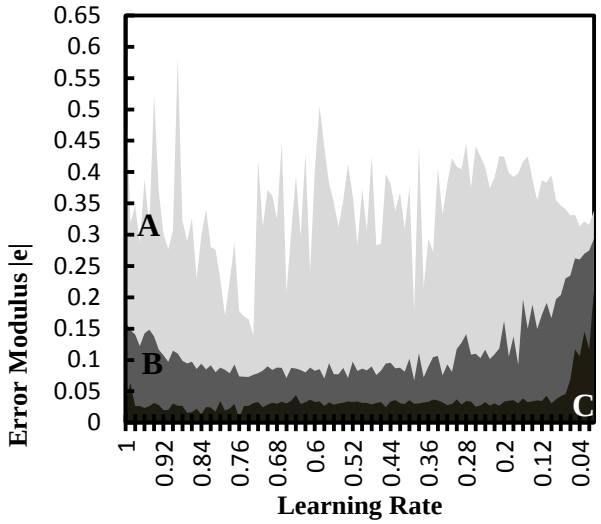


Figure 35: Sensitivity Analysis for the best learning rate value that can be used to fit an x^2 function with a standard neural network. The horizontal axis represents range of rho which is [1, 0.01], the vertical axis represents the range of the error modulus over a sample of 40 independent runs and 100 iterations during each run; the rest of the parameters used during each run are illustrated in Table 7. **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs.

Table 7: Constant parameters used during sensitivity analysis of learning rate iterations to train a standard neural network

Parameter	Value
Sample size for each learning rate value	40
Layer Count	2
Neurons in Layer 1	5
Neurons in Layer 2	1
Rho	1
Iterations/Epoch	100

4) Generalisation performance

Figure 36 illustrates the generalisation error after training and testing a standard neural network with 5 hidden neurons it should be re-emphasized that we view the generalisation performance of standard neuron as that of a sigma-pi neuron with 0^{th} expansion order.

From Figure 36, we can deduce that the model performs better as the number of iterations increase as is always expected with neural networks and converges to a minimum error around the $4{,}500^{th}$ iteration.

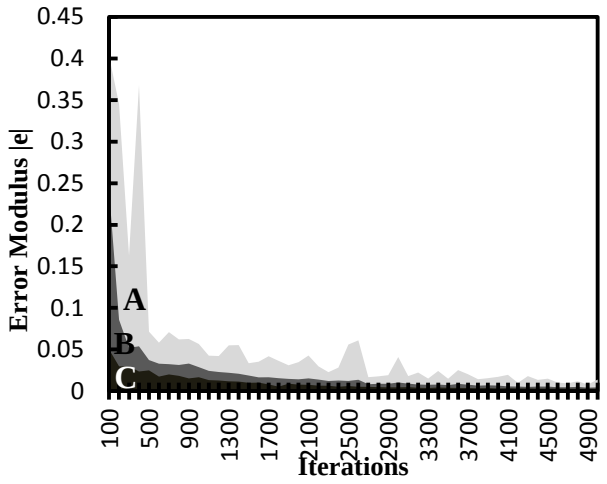


Figure 36: Generalisation performance of a standard neural network with 5 neurons in the hidden layer over the sparse dataset. The horizontal axis represents range of rho which is [100, 5000], with an interval of 100 the vertical axis represents the range of the error modulus over a sample of 40 independent runs; the rest of the parameters used during each run are illustrated in Table 8. **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs.

Table 8: Constant parameters used during sensitivity analysis of iterations to train a standard neural network

Parameter	Value
Sample size for each iteration	40
Layer Count	2
Neurons in Layer 1	5
Neurons in Layer 2	1
Learning Rate	0.1
Rho	1

5) Expansion Experiments

a) Overview

For each model, a description of results is provided observed as the network is expanded on the test set of each dataset and also an illustration of the trend of the error modulus of the model over a range of iterations [10, 500] with an interval of 10 for each stage of expansion. The performance of a standard neural network is first discussed before going on to discuss that of a sigma-pi neural network as the performance of the standard neural network is used as a benchmarks of comparison for sigma-pi neural networks.

b) Performance Benchmarks

The performance of sigma-pi neural networks with expanded states is of prime importance. The expansion of standard neural networks is also of interest; to benchmark the performance of standard neural networks, a model that has one hidden layer neuron is used as illustrated in Figure 37, and the performance of the model is expected to improve as more neurons are added to its hidden layer. To benchmark the performance of sigma-pi neural networks with expanded states, a sigma-pi network with first order expansion is used to compare the performance of expanded versions of the model. I also compare the performance of a sigma-pi neural network with a standard neural network as a standard neural network is technically a sigma-pi neural network with 0^{th} order expansion. I record the best, worst and average performance of the models over a sample of 40 individual runs and define the satisfactory minimum error modulus to be any value below the threshold of 0.05. All results shown are the results from evaluating the models on the test set of the respective dataset.

Table 9: Benchmarks for model performance

Model/Measure	Benchmark(s)
Expanded Standard Neural Network	Standard neural network with one hidden layer neuron
Sigma-pi Neural Network with expanded state-space	Sigma-pi Neural Network with first order expanded state-space and also standard neural network.
Error Modulus for model satisfactory best performance	0.05

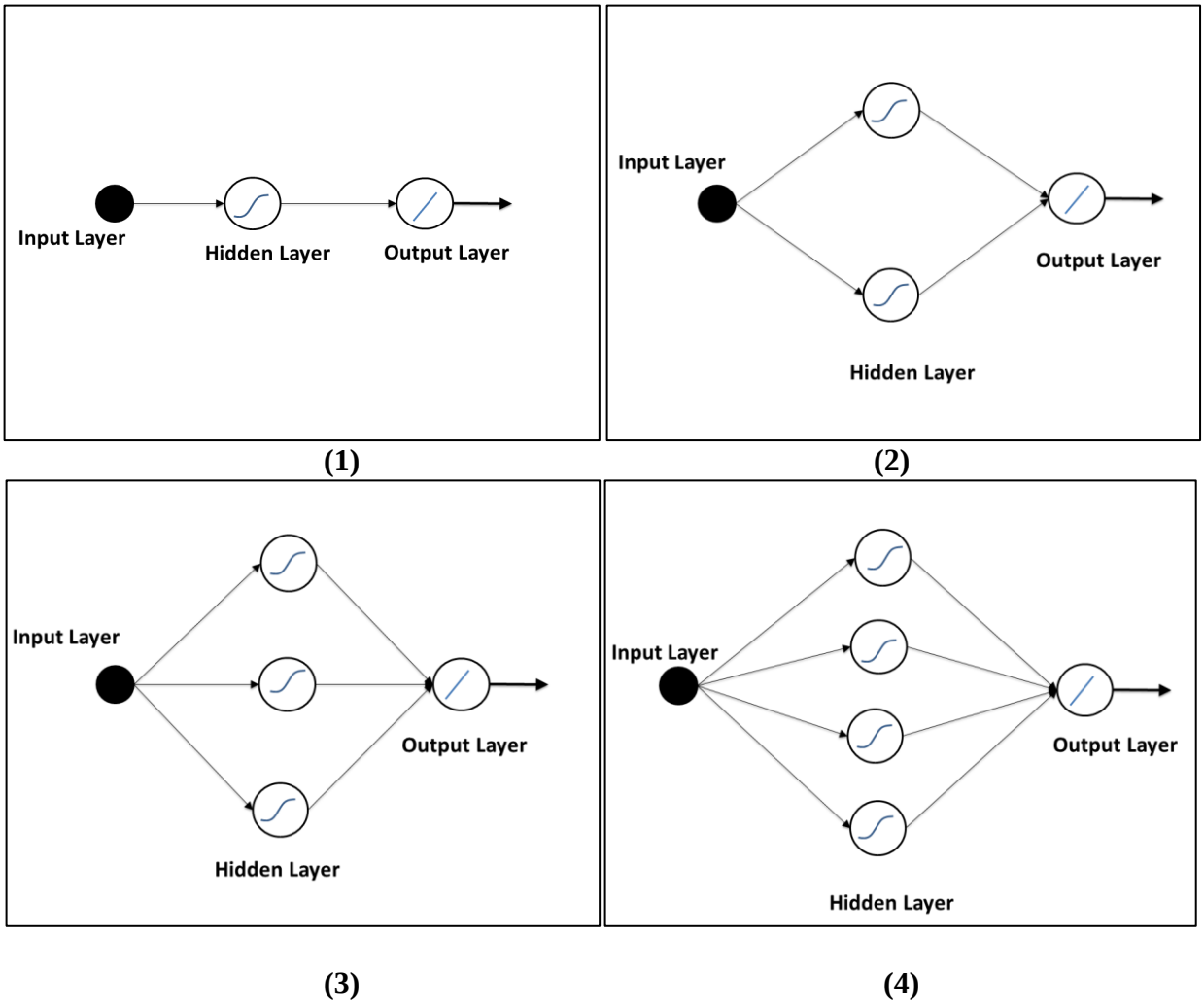


Figure 37: Network topologies for expanding standard neural networks. **(1)** illustrates a standard neural network with one neuron in the hidden layer, **(2)** illustrates a standard neural network with two neurons in the hidden layer, **(3)** illustrates a standard neural network with three neurons in the hidden layer and **(4)** illustrates a standard neural network with four neurons in the hidden layer.

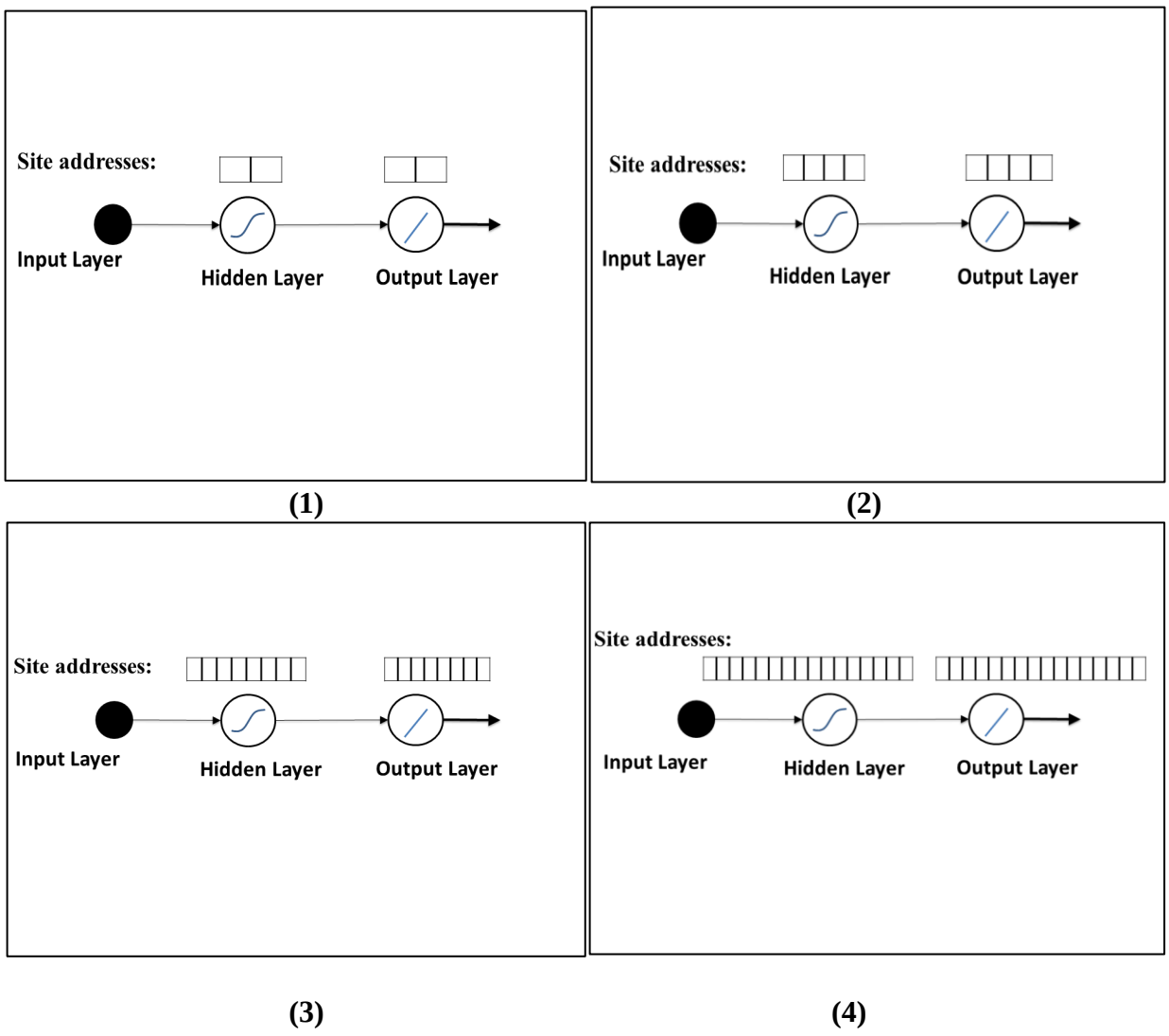


Figure 38: Network topologies for expanding state-space of a sigma-pi neural network with on neuron in its hidden layer. **(1)** illustrates a sigma-pi neural network with first order expansion of state-space, **(2)** illustrates a sigma-pi neural network with second order expansion, **(3)** illustrates a sigma-pi neural network with third order expansion and **(4)** illustrates a illustrates a sigma-pi neural network with fourth order expansion

The performance of the neural network is expected to improve when the input to the network is oversampled. The input to the network can be oversampled by adding more neurons to the network; this applies to both standard neural networks and sigma-pi neural networks. A sigma-pi neural network may also be expanded by increasing the number of site addresses, which is what is done intrinsically by the network when its state-space is expanded. If the generalisation performance does improve, all best, average and worst performances of the network should improve as the number of iterations is increased.

c) Results for sparse dataset

(1) Expanding Standard Neural Networks

Figure 39 illustrates the performance of the standard neural network models over a range of [10, 500] iterations with an interval of 10 and a sample of 40 runs on each iteration. Figure 39 **(1)** illustrates the performance of a model that only has one hidden layer neuron and we observe that on hidden neuron does not suffice to fit the function as it seems to get stuck in a local minima at about the 330^{th} iteration. Figure 39 **(2)** illustrates a model with 2 hidden layer neurons, we observe that the best performance of the network converges to a satisfactory error of 0.006. Figure 39 **(3)** illustrates the performance of the model with 3 hidden layer neurons we observe that the best performance of the network converges to a satisfactory error of 0.006. Figure 39 **(4)** illustrates the performance of the model with 4 hidden layer neurons we observe that the best performance of the network converges to a satisfactory error of 0.008. Table 14 illustrates the minimum actual errors resulting from each experiment. We observe that all expanded networks do perform significantly better than the benchmark model which is the standard neural network with one hidden layer neuron and we can fit the network with 2 neurons in the hidden layer; also, the best worst and average performances annotated by **C**, **A** and **B** respectively, of each trained converge to a low error as the iterations increase. This is because the input to the network is oversampled which results in the network having a higher polynomial order to process the input. However, we observe that as we expand the network further i.e. have more than two neurons in the input layer, the performance degrades slightly; this is most likely due to the fact that the more resources/weights you have in the network i.e. the larger the network, the more iterations it takes to train the network to a reach a satisfactory performance.

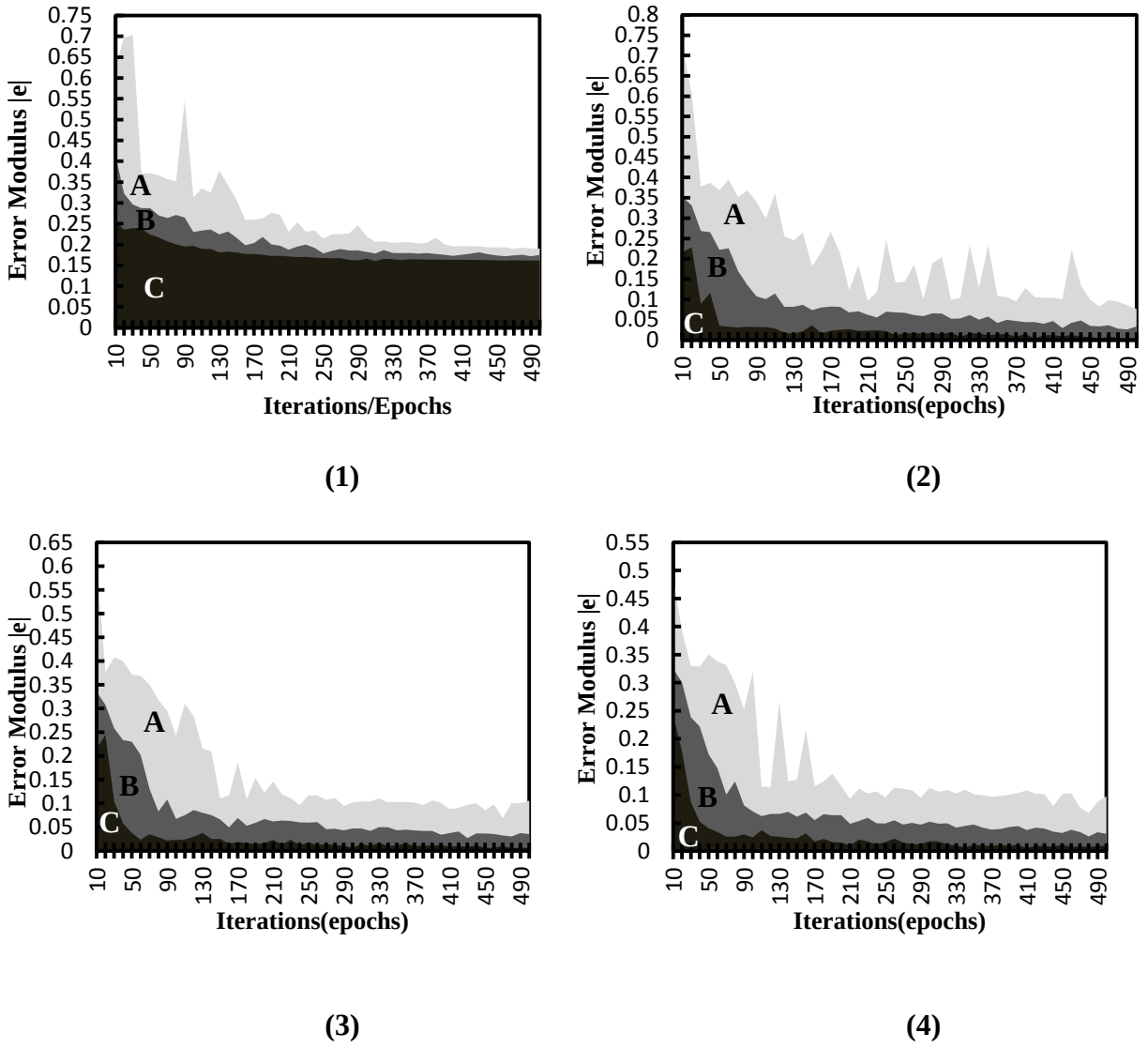


Figure 39: Performance of a standard neural network as its hidden layer is expanded when tested on the sparse dataset. The horizontal axis of each figure represents range of iterations the model uses over a sample of 40 runs, the range of iterations [10, 500], with an interval of 10. The vertical axis represents the range of the error modulus over a sample of 40 independent runs; the rest of the parameters used during each run are illustrated in table 10. In each figure, **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs. **(1)** illustrates the convergence of the error modulus of a standard neural network with one neuron in the hidden layer, **(2)** illustrates the convergence of the error modulus of a standard neural network with two neurons in the hidden layer, **(3)** illustrates the convergence of the error modulus of a standard neural network with three neurons in the hidden layer and **(4)** illustrates the convergence of the error modulus of a standard neural network with four neurons in the hidden layer.

Table 10: Constant parameters used during each training phase of standard neural network over the sparse dataset for all expansion stages

Parameter	Value
Sample size for each iteration	40
Rho	1
Learning Rate	0.5
Number of neurons in output layer	1

(2) Expanding Sigma-Pi Neural networks using state-space expansion.

Figure 40 illustrates the performance of the sigma-pi neural network models over a range of [10, 500] iterations with an interval of 10 and a sample of 40 runs on each iteration. Table 15 illustrates the minimum actual errors resulting from each experiment. We observe that all expanded networks do perform significantly better than the benchmark model which is the sigma-pi neural network with first order expansion, however, from the worst performances of these models, annotated by **A** on the graphs, we can see that its behaviour is very erratic, but the trends of the best performance of each model during each expansion order, annotated by **C** seems consistent. We also observe as we expand the state-space, the model is capable of achieving a desirable performance with only 10 iterations as seen in the best performances annotated by **C** on the graphs, unlike standard neural networks. It should also be emphasized that all the experiments performed only using a sigma-pi neural network with one hidden layer neuron which we expand the state-space for unlike standard neural networks which require 2 hidden layer neurons to fit the x^2 function. The best performance of the model is observed when we apply a second order expansion. We draw attention to the results of the model with a fourth order expansion illustrated in Figure 40 **(4)**; It has a significantly poor performance on almost all sample runs for each iteration, we trace this to instances during the training phase where the weights of some of the neurons approach an impossible value due to indeterminate arithmetic operations encountered; this is most likely because it involves extremely small or large numbers and as such, the network gets stuck in a local minima; these values are denoted by MATLAB as NaN which is the Institute of Electrical and Electronics Engineers (IEEE) arithmetic representation for *not a number* this results in the network predicting the value for any input vector to be NaN, we then substitute the results of these instances with 0.9 for presentation purposes. The erratic behaviour that is observed [increasingly as the state-space is expanded] is most likely as a result of initializing the weights of the neuron randomly; it is also likely due to the parameters used to train the network- it is possible that different expansion orders require a different set of parameters to perform properly. It should be noted that this erratic behaviour is observed occasionally even when the model is trained with a higher number of iterations.

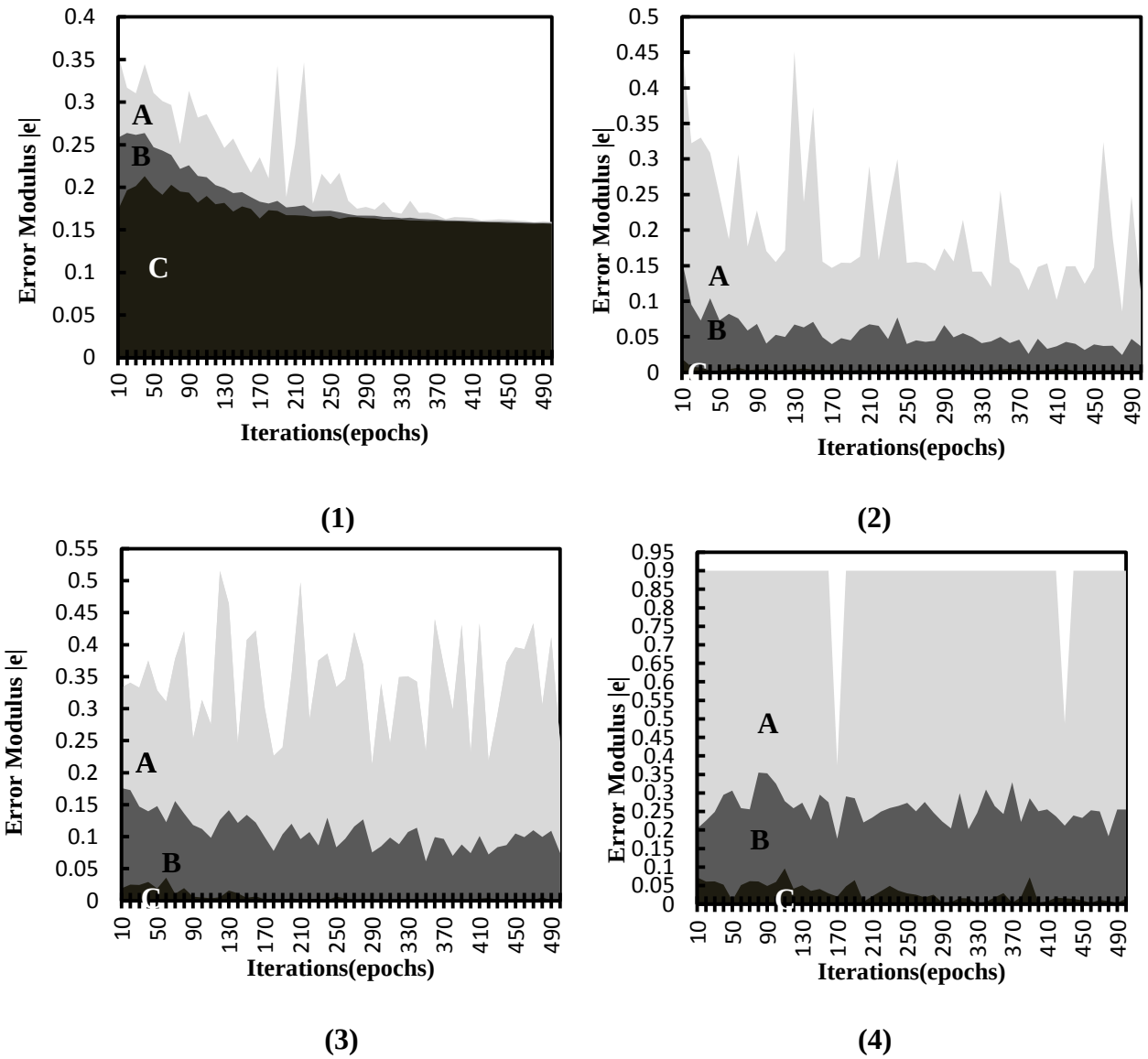


Figure 40: Performance of a sigma-pi neuron as its state-space is expanded when tested on the sparse dataset. The horizontal axis of each figure represents range of iterations the model uses over a sample of 40 runs, the range of iterations [10, 500], with an interval of 10. The vertical axis represents the range of the error modulus over a sample of 40 independent runs; the rest of the parameters used during each run are illustrated in table 11. In each figure, **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs. **(1)** illustrates the convergence of the error modulus of a sigma-pi neuron with first order expansion, **(2)** illustrates the convergence of the error modulus of a sigma-pi neuron with second order expansion, **(3)** illustrates the convergence of the error modulus of a sigma-pi neuron with third order expansion **(4)** illustrates the convergence of the error modulus of a sigma-pi neuron with fourth order expansion.

Table 11: Constant parameters used during each training phase of the sigma-pi neural network model on the sparse dataset for all expansion orders

Parameter	Value
Sample size for each iteration	40
Rho	1
Learning Rate	0.5
Number of neurons in hidden and output layer	1

d) Results for smooth dataset

The experiments run on the dataset with a smoother topology suggest that the topology of data-samples of the function being approximated does not affect the performance of any of the models as we can see from Figure 41 and 42, the performance is very similar to that of the models trained on a dataset with a sparse dataset.

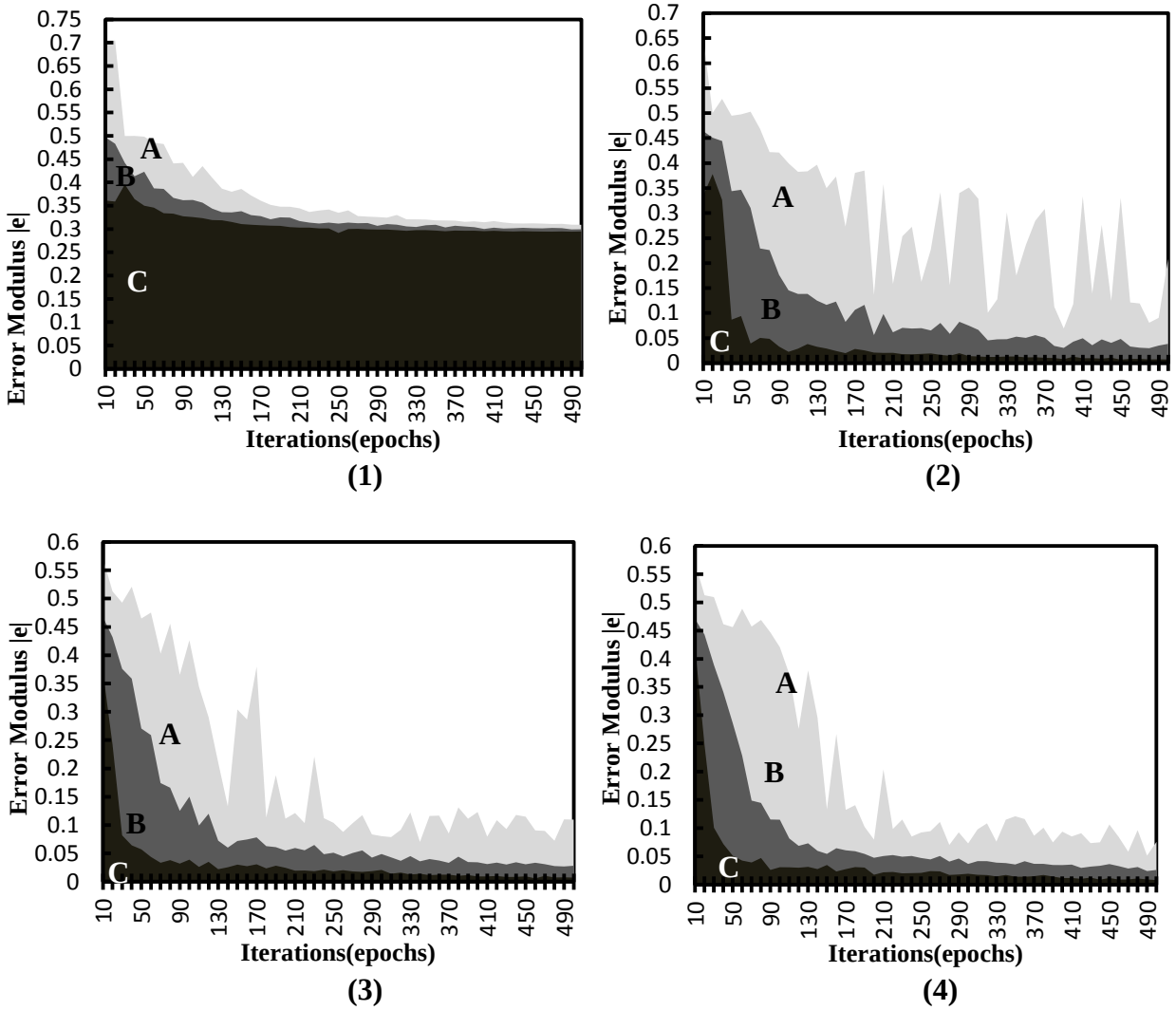


Figure 41: Performance of a standard neural network as its hidden layer is expanded when tested on the smooth dataset. The horizontal axis of each figure represents range of iterations the model uses over a sample of 40 runs, the range of iterations [10, 500], with an interval of 10. The vertical axis represents the range of the error modulus over a sample of 40 independent runs; the rest of the parameters used during each run are illustrated in table 12. In each figure, **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs. **(1)** illustrates the convergence of the error modulus of a standard neural network with one neuron in the hidden layer, **(2)** illustrates the convergence of the error modulus of a standard neural network with two neurons in the hidden layer, **(3)** illustrates the convergence of the error modulus of a standard neural network with three neurons in the hidden layer and **(4)** illustrates the convergence of the error modulus of a standard neural network with four neurons in the hidden layer.

Table 12: Constant parameters used during each training phase of a standard neural network over the smooth dataset for all expansion stages

Parameter	Value
Sample size for each iteration	40
Rho	1
Learning Rate	0.5
Number of neurons in output layer	1

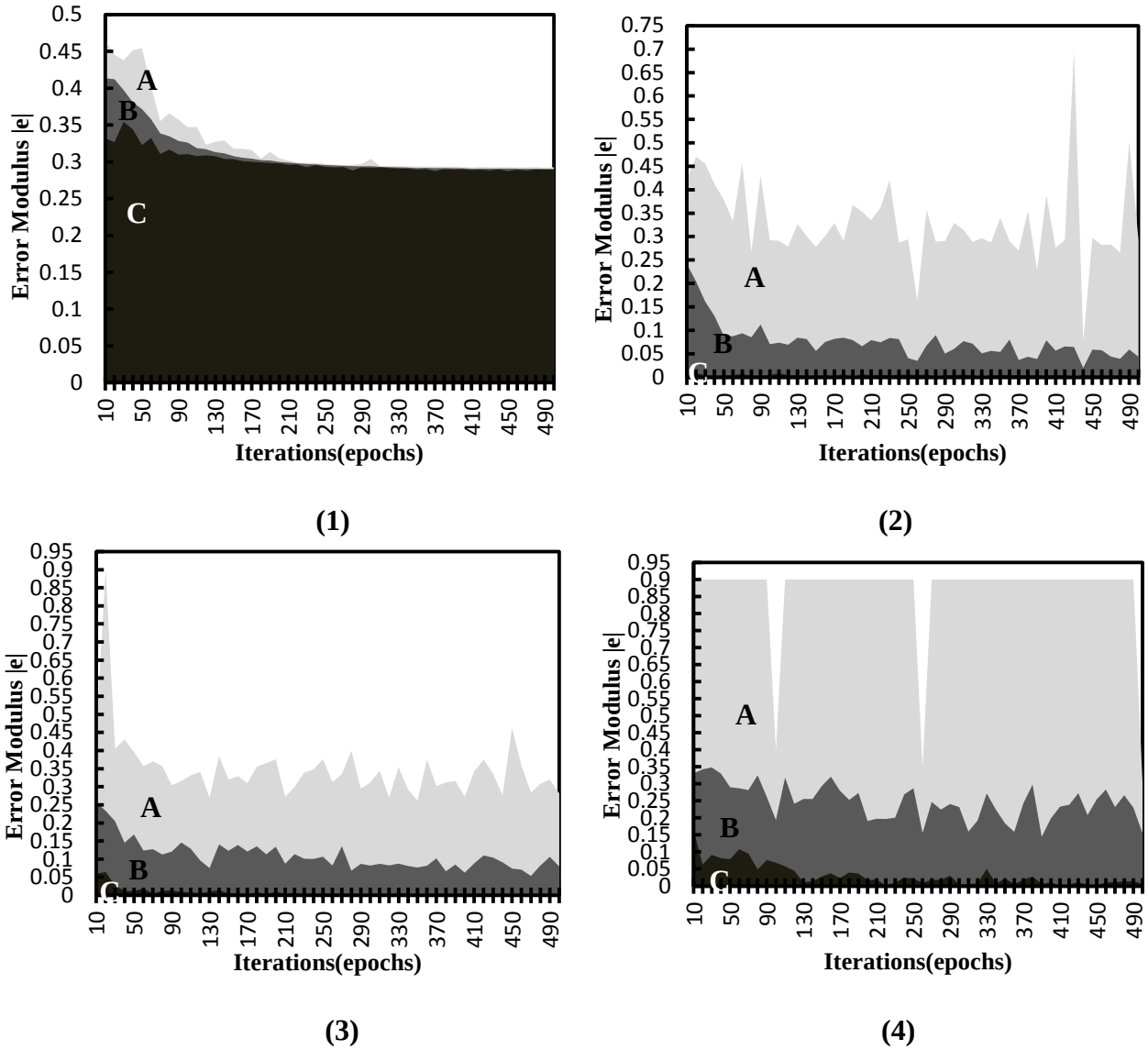


Figure 42: Performance of a sigma-pi neuron as its state-space is expanded when tested on the smooth dataset. The horizontal axis of each figure represents range of iterations the model uses over a sample of 40 runs, the range of iterations [10, 500], with an interval of 10. The vertical axis represents the range of the error modulus over a sample of 40 independent runs; the rest of the parameters used during each run are illustrated in table 13. In each figure, **A** annotates the worst performance of the model over 40 independent runs; **B** annotates the average performance of the model over 40 runs; **C** annotates the best performance of the model over 40 runs. **(1)** illustrates the convergence of the error modulus of a sigma-pi neuron with first order expansion, **(2)** illustrates the convergence of the error modulus of a sigma-pi neuron with second order expansion, **(3))**illustrates the convergence of the error modulus of a sigma-pi neuron with third order expansion **(4)** illustrates the convergence of the error modulus of a sigma-pi neuron with fourth order expansion.

Table 13: Constant parameters used during each training phase of the sigma-pi neuron model over the smooth dataset for all expansion orders

Parameter	Value
Sample size for each iteration	40
Rho	1
Learning Rate	0.5
Number of neurons in hidden and output layer	1

IX. DISCUSSIONS AND CONCLUSIONS

A. Development of ideas

The whole idea behind the study was to explore how the expansion of neural networks can improve their performance. Repeating results similar to those observed by (Neville R et al., 1999) is of prime interest. The purpose of expanding neural networks is to oversample to the input vector to the network so as to increase the networks accuracy; I investigate this by increasing the neurons in the hidden layer of standard neural networks and by using the state-space expansion methodology presented by (Neville R et al., 1999) in sigma-pi neural networks as sigma-pi neurons capture the presynaptic inhibitory property of biological neurons which enables them to expand their state-space unlike standard neurons. The main artefact is to deliver an evaluation of the state-space expansion methodology, which is presented by an analysis of the results observed from the experiments performed and further summarized below.

B. Discussion

The method of evaluation involved sampling the performance of each model over 40 individual runs as I varied the iterations of the network; during each run, I randomly initialize the weights of the network using a normal distribution. I record the best, average and worst performances over every sample of runs so as to observe the spread of the models performance. The results recorded are the performances of the network on the test set of each respective dataset. Because the methodology being evaluated has to do with the expansion of neural networks, I use neural networks that are not expanded as a benchmark of comparison. With each dataset, I observe the performance of both standard neural networks and sigma-pi neural networks as I expand them; I pay special attention to the stage of expansion at which the network converges to a satisfactory error and also the behaviour of the network as it is expanded further with a fixed set of parameters. A recurring theme in my observed results is that the performance of the networks seems to be very erratic; this is most likely because the convergence of the network's error depends on the initialisation of the weights of the network and I generate the weights of the network randomly and as such, there are varying rates convergence of the error of the models. A possible solution may be to restrict the initialization of the weights by the random number generator to a fixed or predictable range; however, in practical real world scenarios, this may not suffice as in practical real world scenarios, as the task of a neural network is to predict the output of unfamiliar data-samples in the output space of the underlying form of dataset; in addition, using a predictable range for initialization of weights during evaluation would result in biased observations, because all results observed would be dependent on the initialization of the networks weights that same range, resulting in results that cannot be reproduced.

a) Discussion of experiments with the AND and XOR datasets

The AND and XOR datasets are binary/logical functions that we try to approximate with the neural networks, however, despite them being functions, they are treated as pattern recognition tasks because the range of the output is [0, 1] which we can consider as classes that the data-samples of the datasets belong to. With the AND dataset, I perform no expansion experiments as a single neuron suffices to fit the function. However with the XOR dataset, I observe that at least 2 neurons are needed in the hidden layer of a standard neural network for it to converge to a minimum error and even then, the performance of the network is very erratic; with sigma-pi neurons, however, I observe that it takes only about 19 iterations to for a single neuron to converge to a minimum error with 1^{st} order state-space expansion and its performance seems to be stable. This illustrates the strength and advantage of capturing the presynaptic inhibitory property of biological neurons; the sigma-pi neuron with first-order state-space expansion has a higher polynomial order and a higher weight space at its disposal to fit the function properly.

b) Discussion of experiments with the X^2 dataset

With the x^2 dataset, the interest lies in both how the expansion of the network affects the generalisation performance and how the topology of data-samples of the function affects the generalisation performance of the network.

When expanding standard neural networks, I use a standard neural network with one neuron in its hidden layer as a benchmark of comparison for the performance of expanded forms of the neural network. My results show that expanding a standard neural network does improve its generalisation performance measured by the error modulus and also the overall performance, including the best, worst and average performance does seem to stabilise as the network is expanded unlike the behaviour observed with from its performance on the XOR dataset. I observed that it takes at least two neurons in the hidden layer of a standard neural network to converge to a satisfactory minimum error when dealing with the x^2 dataset; compared to the benchmark of one hidden layer neuron which gets stuck in local minima, it is safe to conclude that expansion does improve the generalisation performance of the network.

To evaluate the performance of the state-space expansion methodology, I use the performance of a sigma-pi neural network with one hidden neuron with first order expansion as a benchmark of comparison. I also use the performance of standard neural networks observed from training on the same dataset as standard neural networks may be seen as sigma-pi neural networks with 0^{th} order expansion under the framework of the methodology. I observe the performance of the model as I expand its state-space; my results suggest that the methodology does improve the performance of a sigma-pi neuron as it

takes an expansion order of 2 for the network to converge to a satisfactory minimum error. Compared to the performance of a sigma-pi neural network of the same structure with 1st order expansion, all sigma-pi networks with the same architecture but with expanded state-spaces do perform better, however their behaviour seems to be erratic. I attribute the erratic behaviour to randomly initializing the weights of the network which the convergence of error is dependent on. Compared to the method of expanding standard neural networks by adding neurons to the network, state-space expansion does seem to be competent but more unstable with the x^2 dataset. I consider the methodology to be competent, because it enables sigma-pi neural networks to converge to a minimum error with just one neuron in the hidden layer of the network as opposed to standard neural networks that require at least two neurons in the hidden layer. Also, the best performance of the sigma-pi neuron with respect to iterations used is significantly better than that of standard neural networks as all expansion orders can enable the network to converge to a satisfactory minimum error with just 10 iterations as opposed to standard neural networks that require at least 170. The trade-off, however, is that the performance of standard neural networks is more stable. A summary of the performances of the entire model evaluated is presented below from Table 14 to Table 19.

Table 14: Error modulus at the 500th iteration for each standard neural network model over the sparse dataset for all expansion stages

Number of neurons in hidden layer	Error Modulus for worst performance	Error Modulus for average performance	Error Modulus for best performance
1	0.19	0.18	0.160
2	0.08	0.03	0.006
3	0.11	0.04	0.006
4	0.10	0.03	0.008

Table 15: Error modulus at the 500th iteration for each sigma-pi neural network model over the sparse dataset for all expansion orders

Expansion Order	Error Modulus for worst performance	Error Modulus for average performance	Error Modulus for best performance
1	0.16	0.16	0.160
2	0.15	0.04	0.002
3	0.25	0.07	0.003
4	NaN	0.26	0.015

Table 16: Error modulus at the 500th iteration for each standard neural network model over the smooth dataset for all expansion stages

Number of neurons in hidden layer	Error Modulus for worst performance	Error Modulus for average performance	Error Modulus for best performance
1	0.31	0.40	0.295
2	0.21	0.04	0.007
3	0.11	0.03	0.008
4	0.08	0.03	0.009

Table 17: Error modulus at the 500th iteration for each sigma-pi neuron model over the smooth dataset for all expansion orders

Expansion order	Error Modulus for worst performance	Error Modulus for average performance	Error Modulus for best performance
1	0.29	0.29	0.290
2	0.29	0.04	0.003
3	0.28	0.08	0.002
4	0.30	0.15	0.006

Table 18: Summary of performance for standard neural networks over all experiments

Dataset	Number of hidden layer neurons required to fit dataset	Performance
AND	0	stable
XOR	2	erratic
X^2	2	stable

Table 19: Summary of performance for sigma-pi neural networks over all experiments

Dataset	Number of hidden layer neurons required to fit dataset	Expansion order required to fit dataset	Performance
AND	0	0	stable
XOR	0	1	stable
X^2	1	2	erratic

c) Remarks

The previous work with expansion of state-space in neural networks performed by (Neville R et al., 1999), does show that state-space expansion does improve the generalization performance of sigma-pi neural networks however, the experiments they present only involves expanding the networks from zeroth to second order. I observe a similar behaviour in my experiments; however, as I increase the order of expansion further, I realize that the state-space expansion methodology may not be as stable as previously thought. The results above illustrate that the state-space expansion does have the potential to improve the generalization performance of a sigma-pi neural network. It is expected that all average, best and worst performances of the network to improve as the number of iterations are increased i.e. the higher the number of iterations, the lower the generalisation error, however, results observed suggest that the performance sigma-pi neural networks with expanded state-spaces tend to be erratic because even with higher iterations in my experiments with the x^2 dataset, the model still has the potential to perform very poorly as we still observe high errors on the result sample for every iteration; This behaviour is observed more so as the state-space is expanded. Another possible reason may be due to the choice of activation functions used in the network. For standard neural networks, I observe that its performance does improve and become more stable as I expand the networks. My findings also suggest that the topology of the data-samples of the function used to train a neural network does not significantly

affect the network's ability to approximate the function of interest. This however does not discredit the superior ability of sigma-pi neurons as it still fits the function with only one neuron in its hidden layer, unlike standard neurons which require 2 neurons in its hidden layer to fit the function, which is defined by the criteria of the network converging to an error modulus of below 0.05.

C. Limitations

It should also be noted that aside from the unpredictable performance of sigma-pi neural networks with expanded state-spaces, another significant limitation is that the computational resources required to train sigma-pi networks with expanded space is significantly high, this is because of the arithmetic operations required to calculate the site address values during the feed forward phase, and the derivative of each neuron's input during the backpropagation phase. An alternative to combat this limitation is to train the networks using reward-penalty training algorithm. As we expand the state-space of a sigma-pi network, we increase the number of weights required by the network to train; this process requires a lot of allocation in the virtual memory of the computer being used. Future work concerning sigma-pi networks would require powerful computers. Another major limitation is the fact that neural networks cannot perform extrapolation tasks when dealing with function approximation problems; this places a significant limitation on their applications.

D. Future work

Future work may involve any of the following:

- Investigating the performance of state-space expansion technique in a pattern-recognition domain.
- Performing a sensitivity analysis of the performance of sigma-pi networks with expanded states with regard to different activation functions used in various layers of the network order to observe how this affects the generalisation performance of the network especially with regards to function approximation.
- The state-space expansion methodology may be applied using alternative network architectures such as recurrent neural networks in order to see if the architecture of the network affects its performance.
- Performing a comparison of the state-space expansion methodology in pattern recognition domains and function approximation domains.

E. Conclusion

The motivation for the study is to replicate similar results to those observed by Neville et al in (Neville R et al., 1999) so I sought to implement the necessary artefacts to reproduce the results. The primary objective was to investigate whether state-space expansion can improve the generalization performance of sigma-pi neural networks. The aims and objectives were developed as a result and were delivered as a result of completing the following activities:

- Understanding of the necessary subjects relevant to the study. This includes *neural networks, sigma-pi neural networks, training neural networks, application domains of neural networks, automatic differentiation, and expansion of neural networks* and *evaluating the performance of neural networks.*
- An implementation of the necessary artefacts used to carry out the study which includes an implementation of a standard neural network, and implementation of a sigma-pi neural network and an implementation of state-space expansion.
- An evaluation methodology to measure the performance of the state-space expansion methodology. This involved an understanding of sensitivity analysis: a methodology that I use to select ideal parameters I order to ensure I do not provide a biased analysis. The benchmarks chose were the performances of standard neural networks and sigma-pi neural networks with first order expansion.
- An analysis of the results observed. This was discussed in chapter 8, where I present my observations. My observations suggest that the methodology works and the results seem similar to the ones presented by (Neville R et al., 1999) however, I also come to the realisation that the state-space expansion methodology may become unstable as the expansion order is increased.

X. REFERENCES

[1] ALEKSANDER, I. 1965. Fused logic element which learns by example. *Electronics Letters,* 6**,** 173-174.

[2] BATTITI, R. 1989. Accelerated backpropagation learning: Two optimization methods. *Complex systems,* 3**,** 331-342.

[3] BECHTEL, W. & ABRAHAMSEN, A. 1991. *Connectionism and the mind: An introduction to parallel processing in networks,* Basil Blackwell.

[4] BERGSTRA, J. & BENGIO, Y. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research,* 13**,** 281-305.

[5] BISHOP, C. M. 1995. *Neural networks for pattern recognition,* Oxford university press.

[6] BROWN, G. 2015. COMP61011, Foundations of Machine LearningThe Not so scary guide to Machine Learning [lecture handout].

[7] DREYFUS, H. L. 1979. *What computers can't do: The limits of artificial intelligence,* Harper & Row New York.

[8] FAWCETT, T. 2006. An introduction to ROC analysis. *Pattern recognition letters,* 27**,** 861-874.

[9] GALLANT, S. I. 1990. Perceptron-based learning algorithms. *Neural Networks, IEEE Transactions on,* 1**,** 179-191.

[10] GRIEWANK, A. 1989. On automatic differentiation. *Mathematical Programming: recent developments and applications,* 6**,** 83-107.

[11] GURNEY, K. 1989. Learning in networks of structured hypercubes. Brunel Univ., Uxbridge (UK).

[12] GURNEY, K. 1997. An introduction to neural networks. *CRC press*.

[13] HINTON, G. E. & SALAKHUTDINOV, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science,* 313**,** 504-507.

[14] KOHONEN, T. 1990. The self-organizing map. *Proceedings of the IEEE,* 78**,** 1464-1480.

[15] MCCULLOCH, W. S. & PITTS, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics,* 5**,** 115-133.

[16] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K. & OSTROVSKI, G. 2015. Human-level control through deep reinforcement learning. *Nature,* 518**,** 529-533.

[17] NEVILLE, R. 2016. Expansion of State Space. *Technical memorandum 2016-2, The University of Manchester, Tuesday, dated: 12 January 2016*.

[18] NEVILLE R, STONHAM T & R., G. 1999. Partially pre-calculated weights for the backpropagation learning regime and high accuracy function mapping using continuous input RAM-based sigma–pi nets. *Neural networks,* 13**,** 19.

[19] NEWELL, A. & SIMON, H. A. 1976. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM,* 19**,** 113-126.

[20] NGUYEN-THIEN, T. & TRAN-CONG, T. 1999. Approximation of functions and their derivatives: a neural network implementation with applications. *Applied Mathematical Modelling,* 23**,** 687-704.

[21] ROSENBLATT, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review,* 65**,** 386.

[22] RUMELHART, D. E., MCCLELLAND, J. L. & GROUP, P. R. 1988. *Parallel distributed processing*, IEEE.

[23] RUSSELL, S., NORVIG, P. AND INTELLIGENCE, A., 1995. A modern approach.Artificial Intelligence. *Prentice-Hall, Egnlewood Cliffs, 25, p.27.* .

[24] SCHMIDHUBER, J. 2015. Deep learning in neural networks: An overview. *Neural Networks,* 61**,** 85-117.

[25] STULP, F. & SIGAUD, O. 2015. Many regression algorithms, one unified model: a review. *Neural Networks,* 69**,** 60-79.

[26] SUN, R. 1999. Artificial intelligence: Connectionist and symbolic approaches.

[27] VON NEUMANN, J. 1993. First Draft of a Report on the EDVAC. *IEEE Annals of the History of Computing***,** 27-75.

[28] WERBOS, P. 1974. Beyond regression: New tools for prediction and analysis in the behavioral sciences.

[29] WOUTER DEN HAAN 2016. Function Approximation [online] Available at: http://econ.lse.ac.uk/staff/wdenhaan/numerical/functionapproximation.pdf (Accessed: 5 May 2016).

APPENDIX

The purpose of this appendix is to provide an enlarged depiction of the results observed during the experiments.

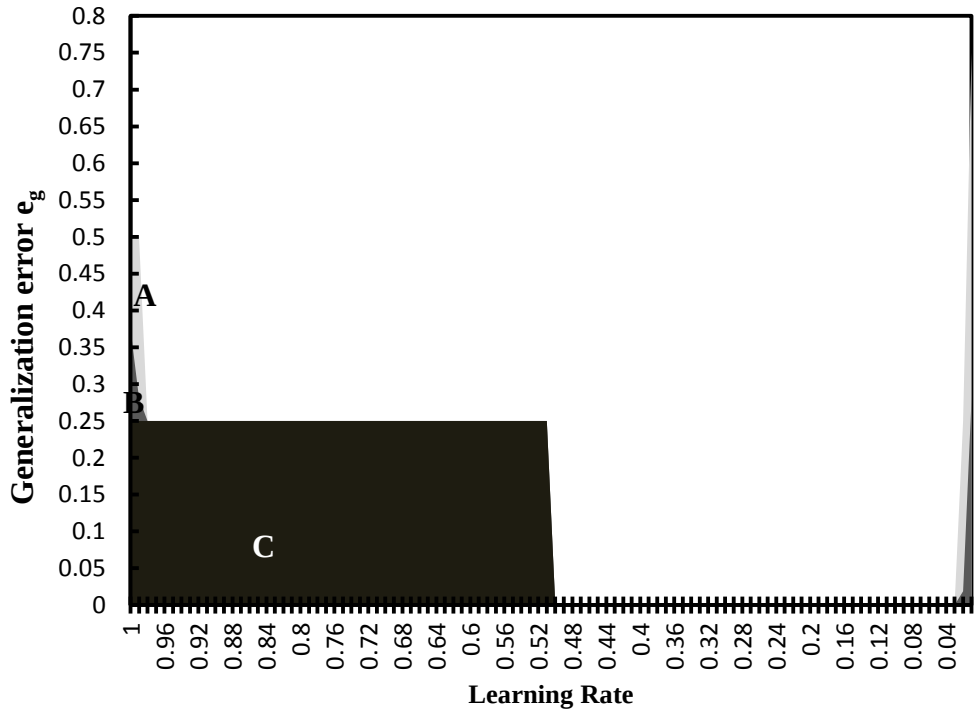


Figure 43: Enlarged graph of the sensitivity analysis for the AND dataset using a standard neuron.

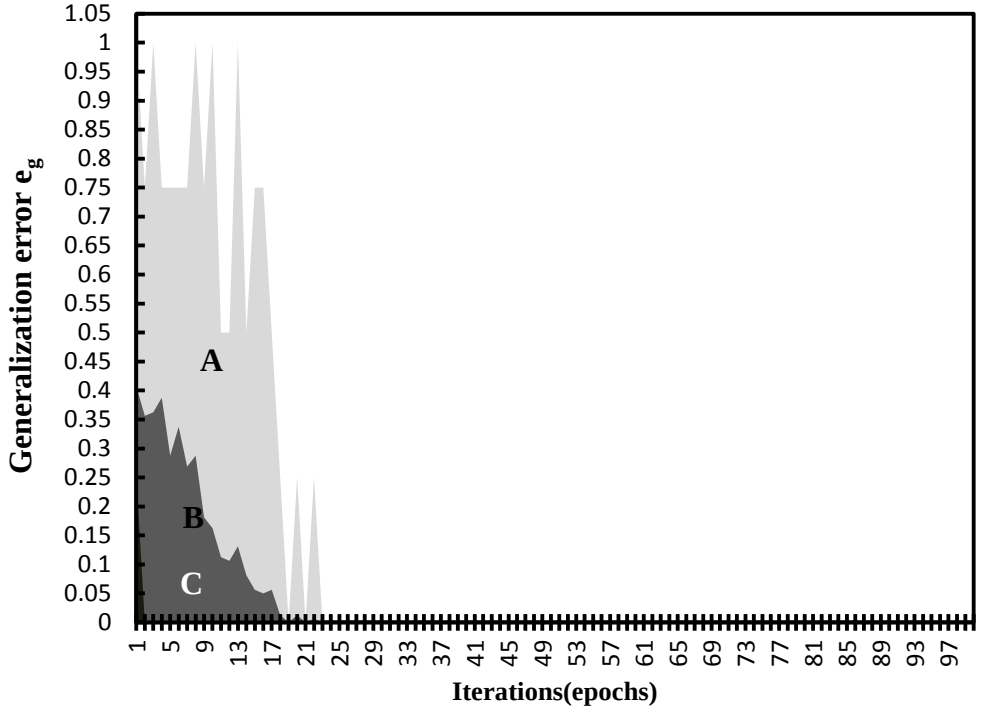


Figure 44: Enlarged graph of the generalisation performance of a standard neuron on the AND dataset.

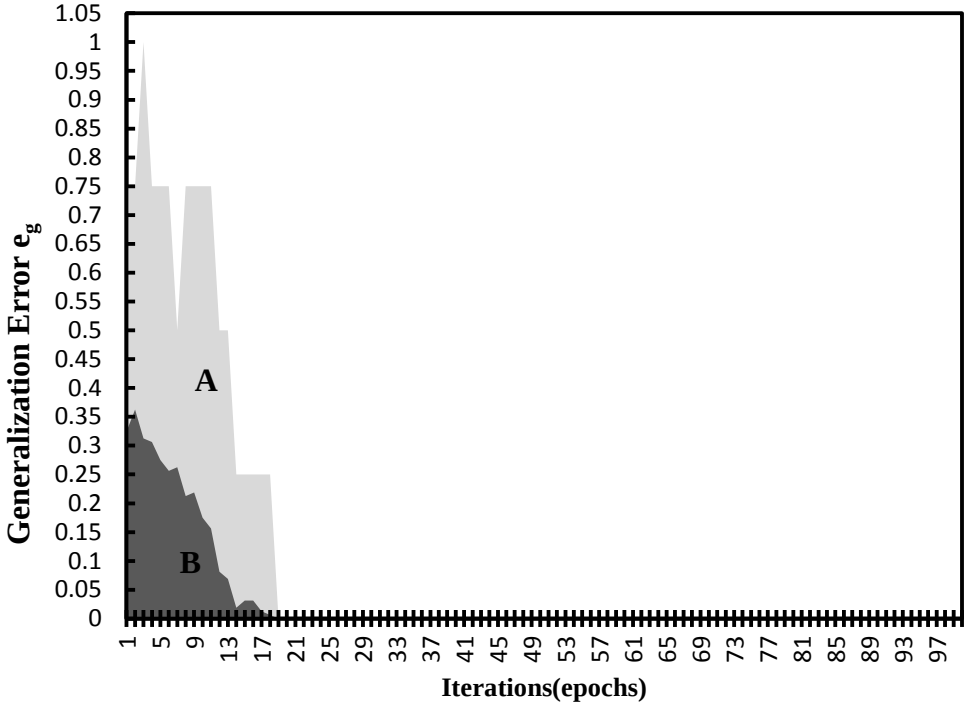


Figure 45: Enlarged graph of the generalisation performance of a sigma-pi neuron on the AND dataset.

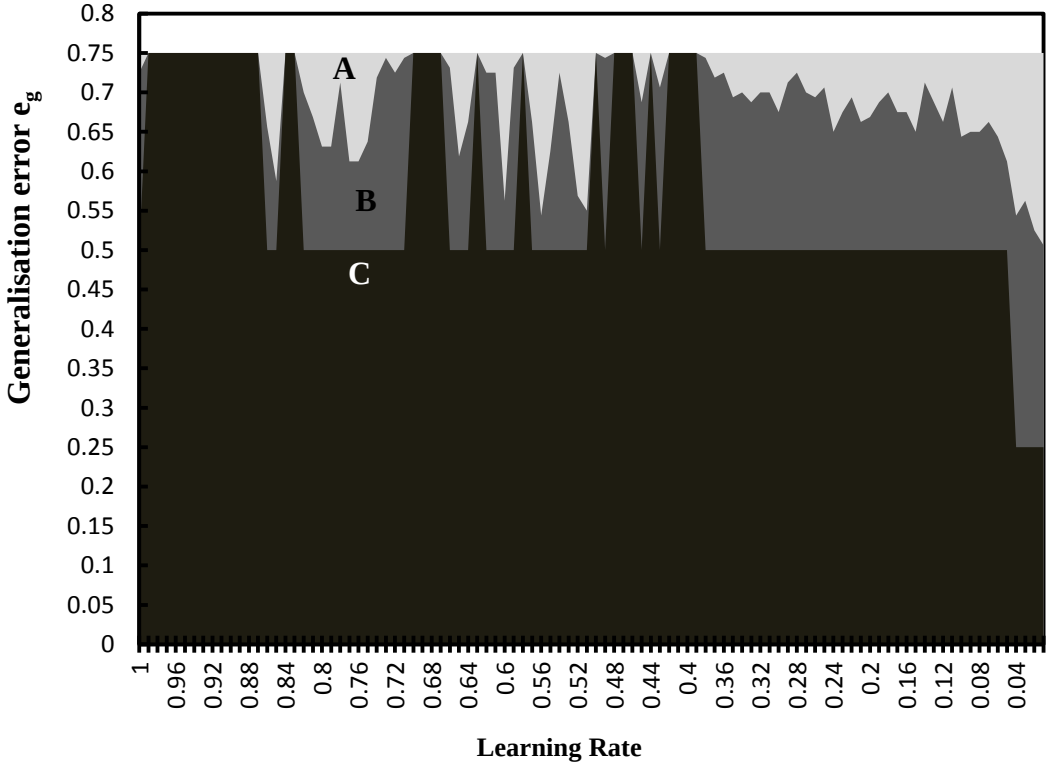


Figure 46: Enlarged graph of Sensitivity Analysis for the best learning rate value that can be used to fit XOR dataset with a standard neuron.

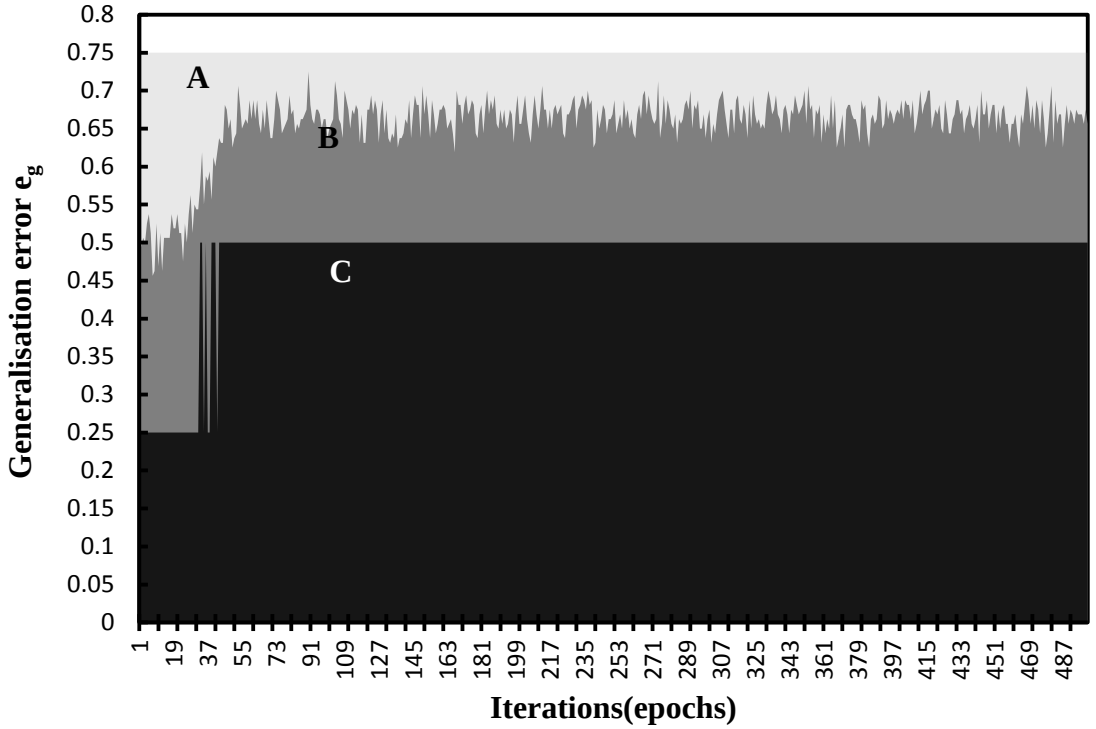


Figure 47: Enlarged graph of the performance of a standard neuron when trained on the XOR dataset.

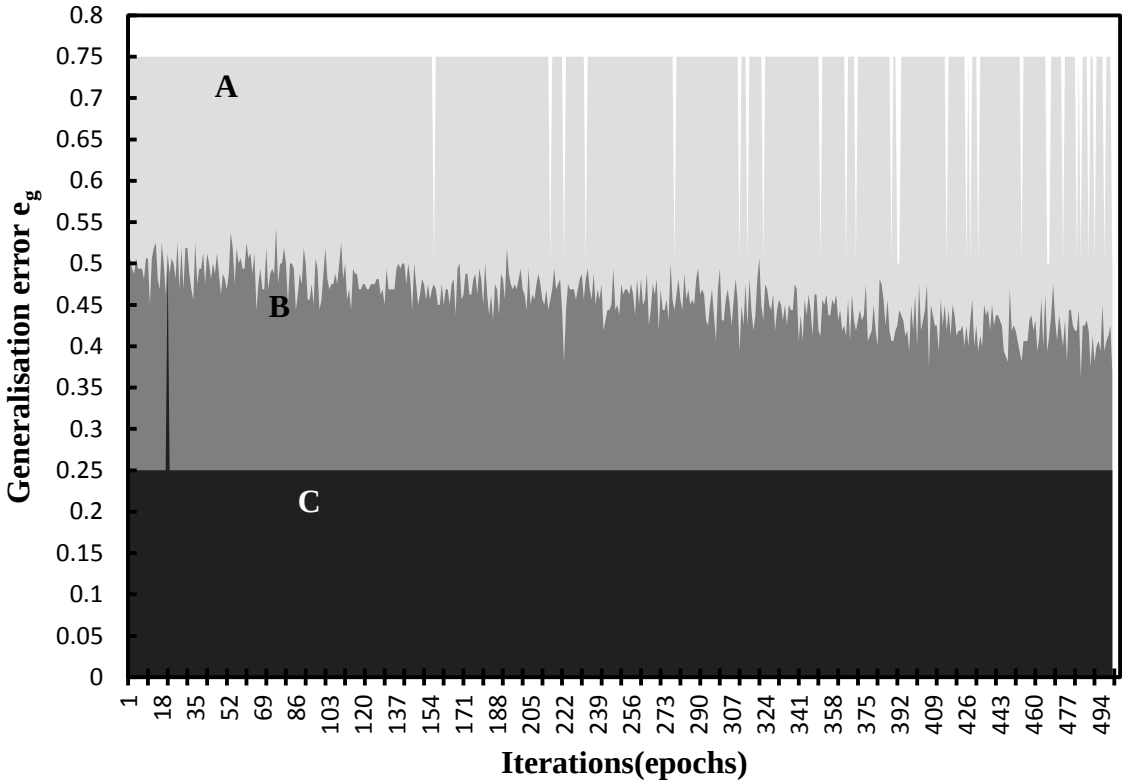


Figure 48: Enlarged graph of the performance of a standard neural network with one hidden layer neuron when trained on the XOR dataset.

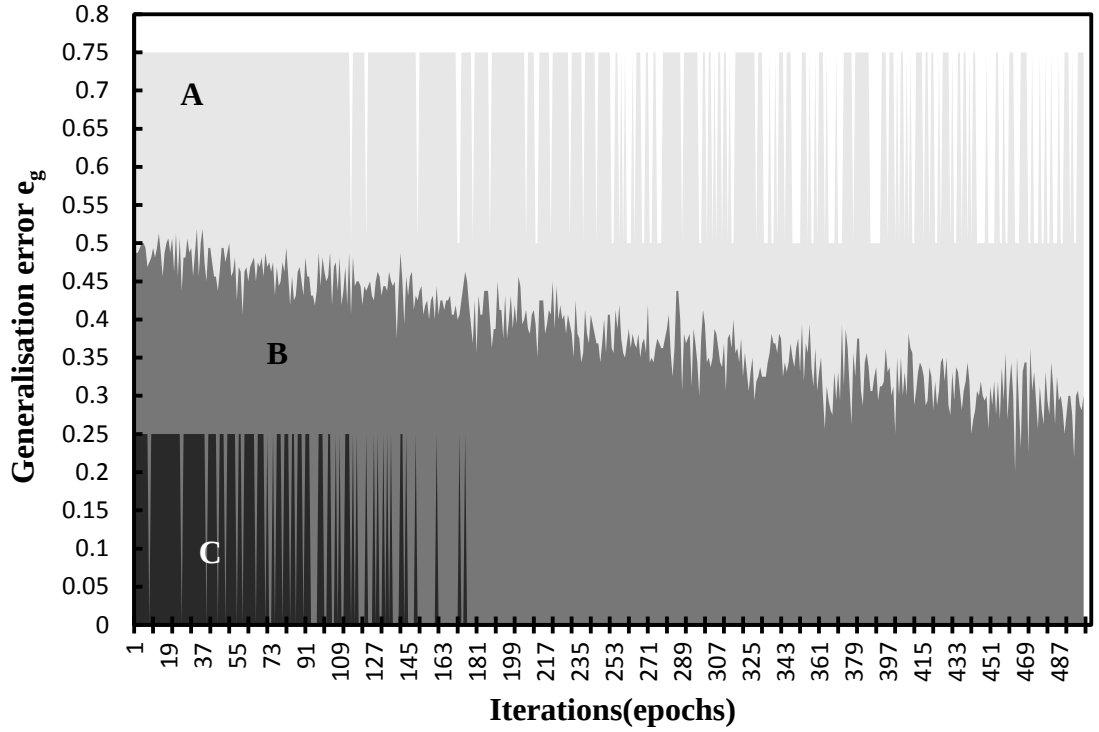


Figure 49: Enlarged graph of the performance of a standard neural network with two hidden layer neurons when trained on the XOR dataset.

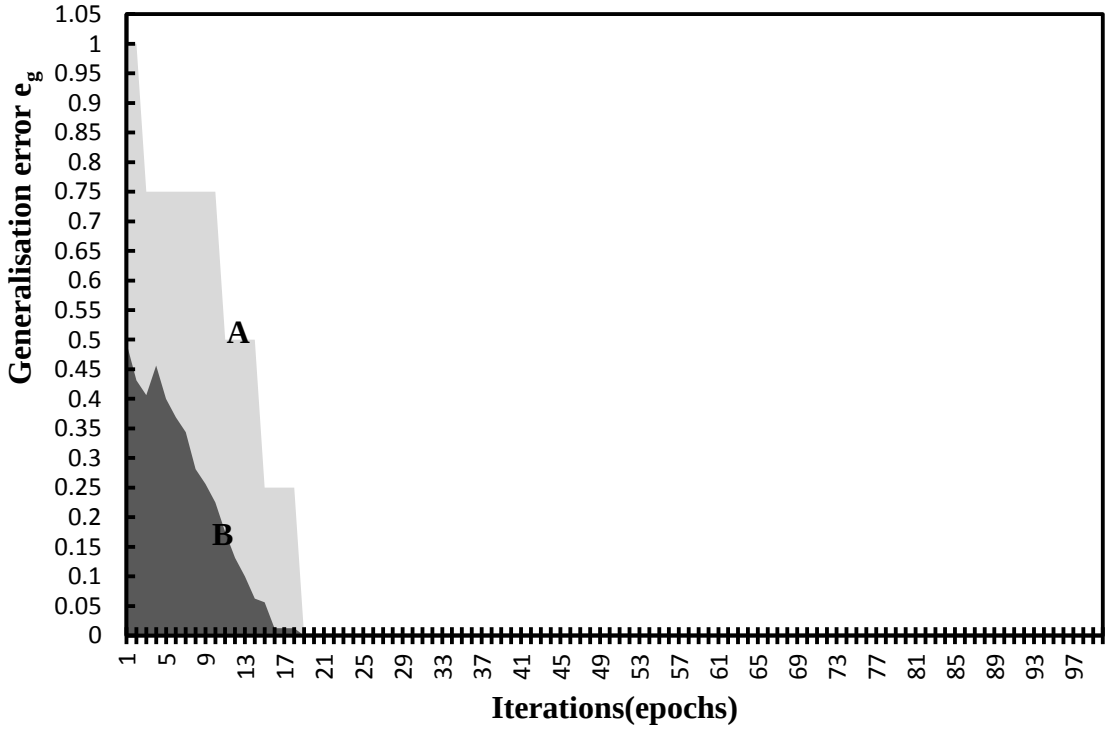


Figure 50: Enlarged graph of the performance of a sigma-pi neuron with 1st order state-space expansion when trained on the XOR dataset.

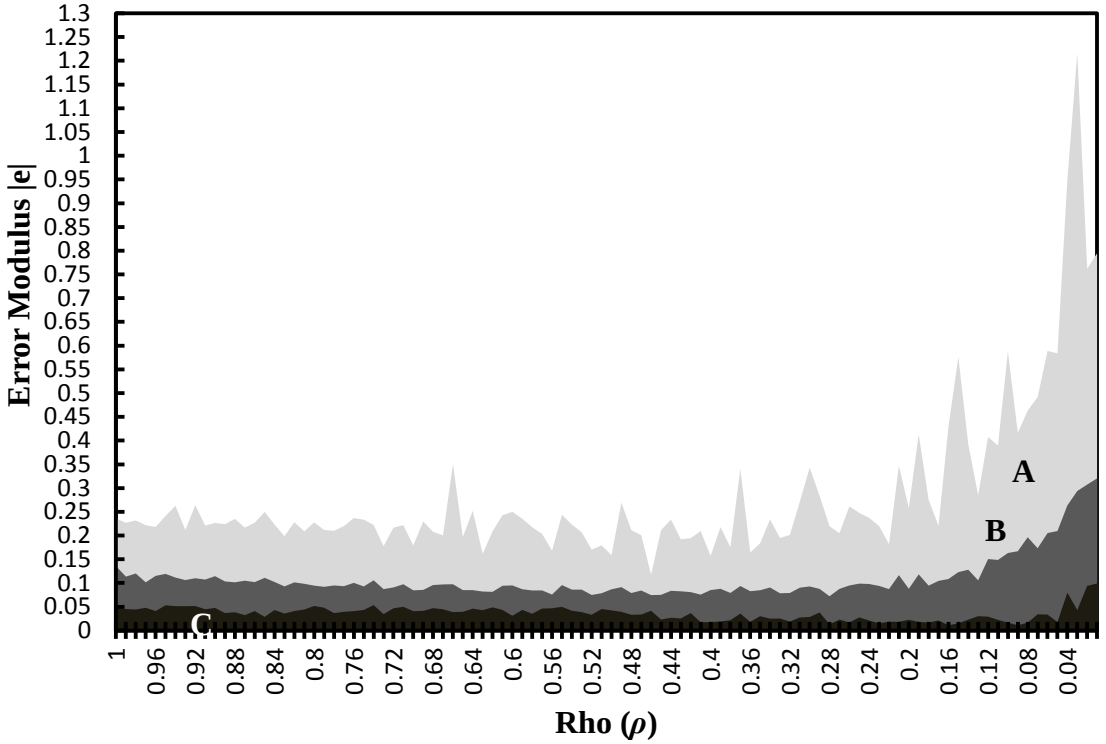


Figure 51: Enlarged graph of Sensitivity Analysis for the best value of rho used to fit an x^2 function with a standard neural network.

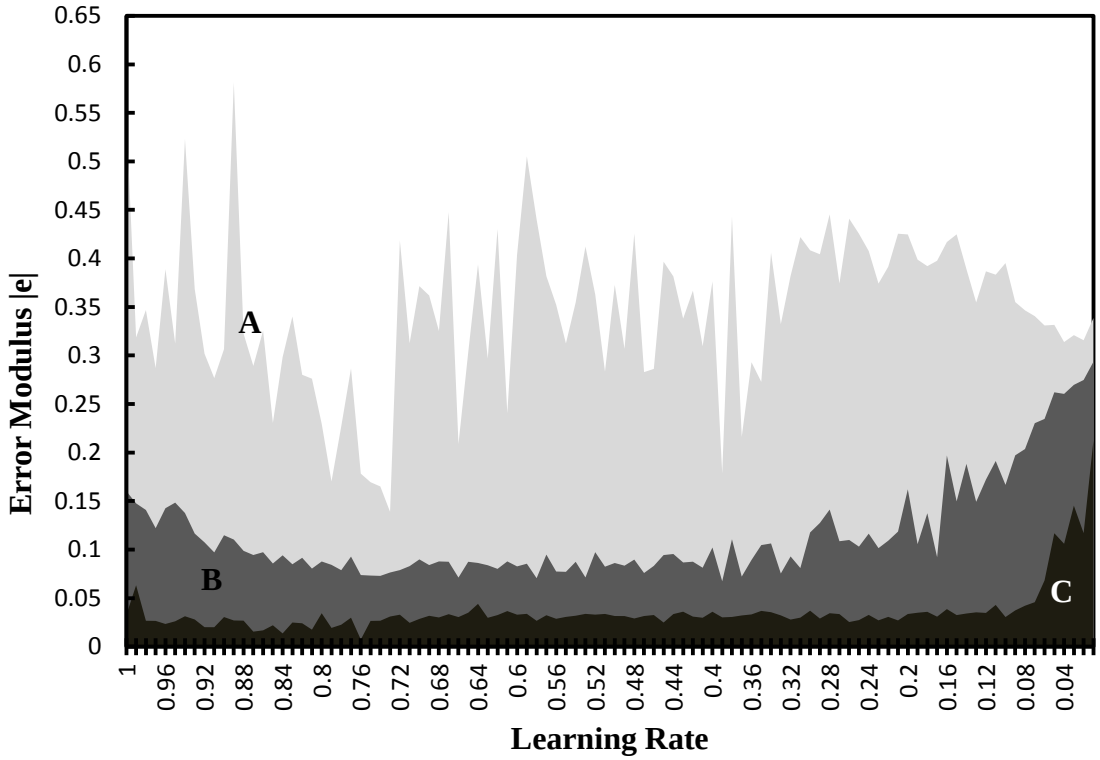


Figure 52: Enlarged graph Sensitivity Analysis for the best learning rate value that can be used to fit an x^2 function with a standard neural network

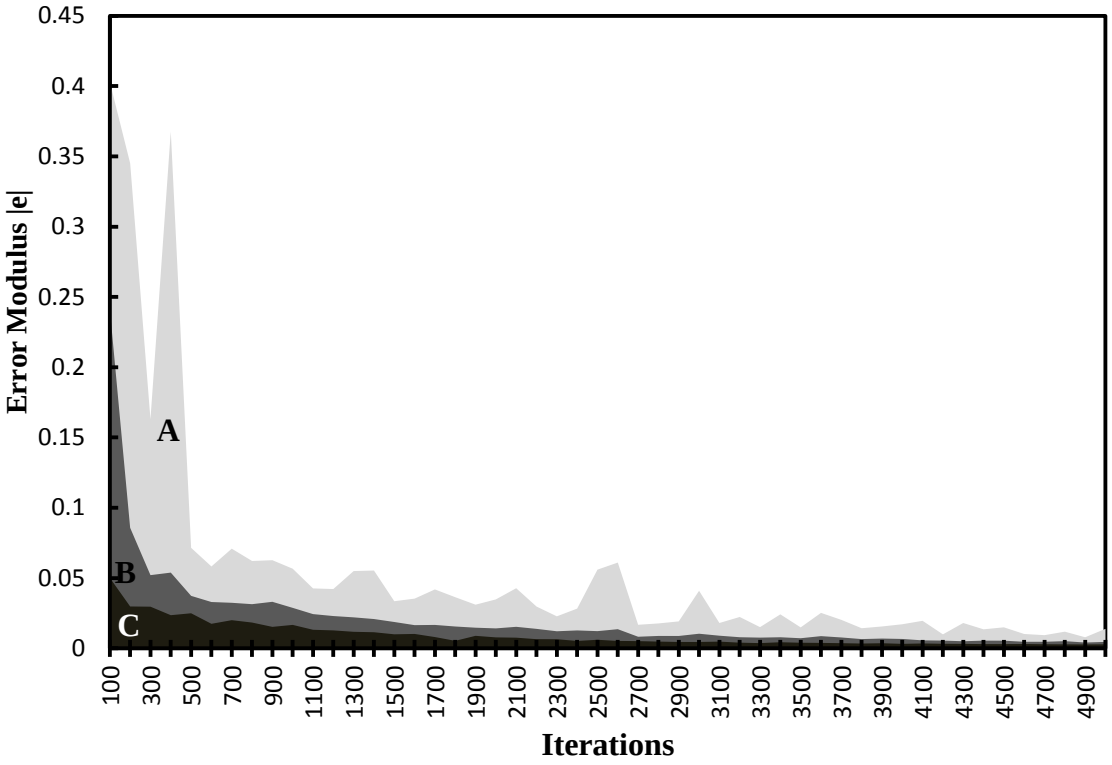


Figure 53: Enlarged graph of Generalisation performance of a standard neural network with 5 neurons in the hidden layer over the sparse dataset.

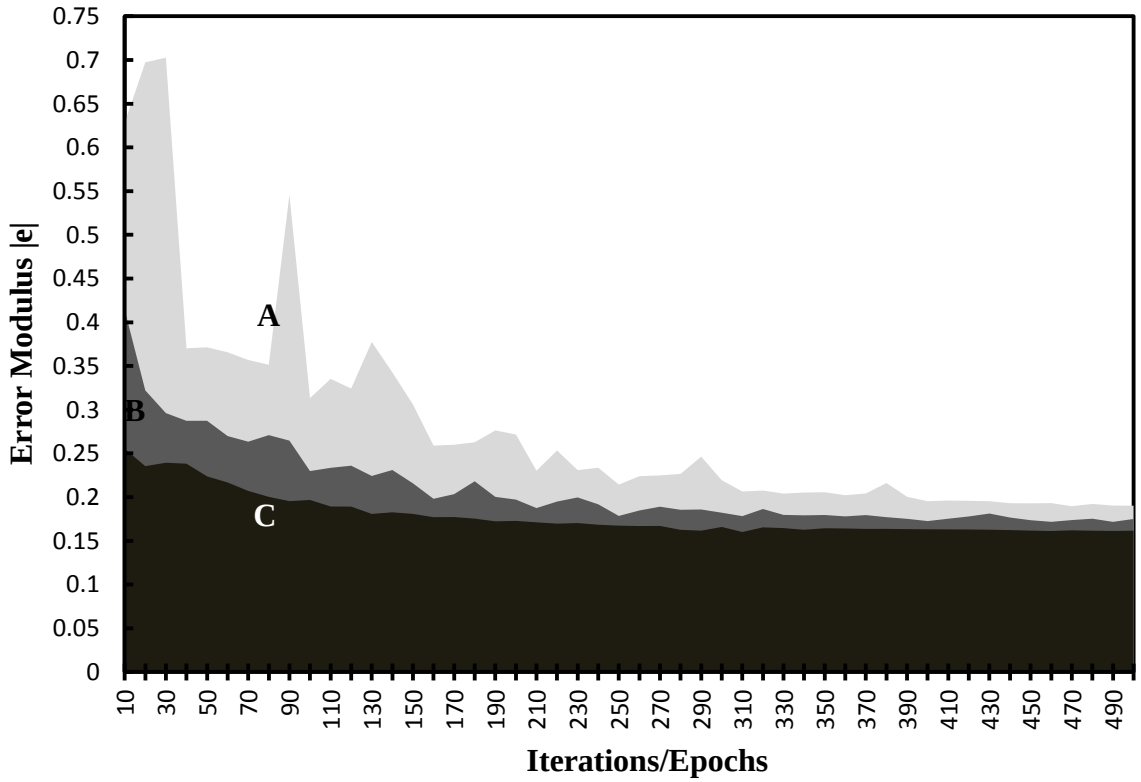


Figure 54: Enlarged graph of the Performance of a standard neural network with one hidden layer neuron when tested on the sparse dataset.

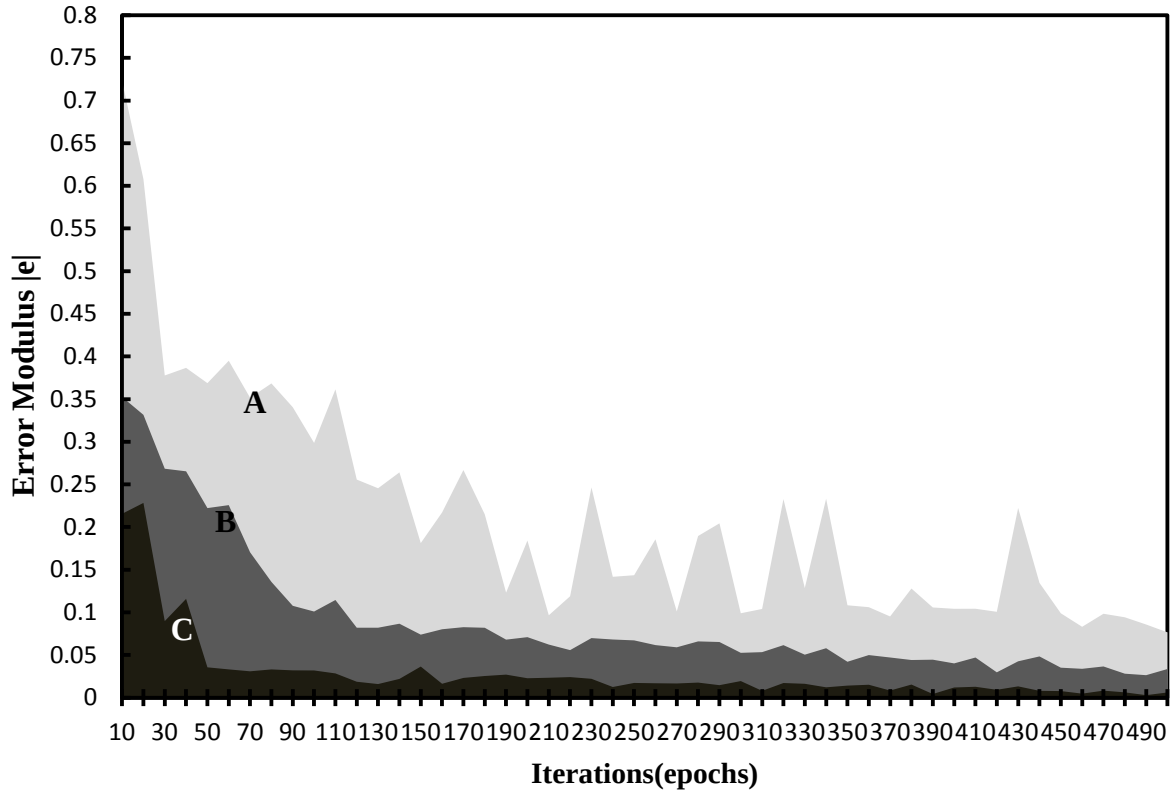


Figure 55: Enlarged graph of the Performance of a standard neural network with two hidden layer neurons when tested on the sparse dataset.

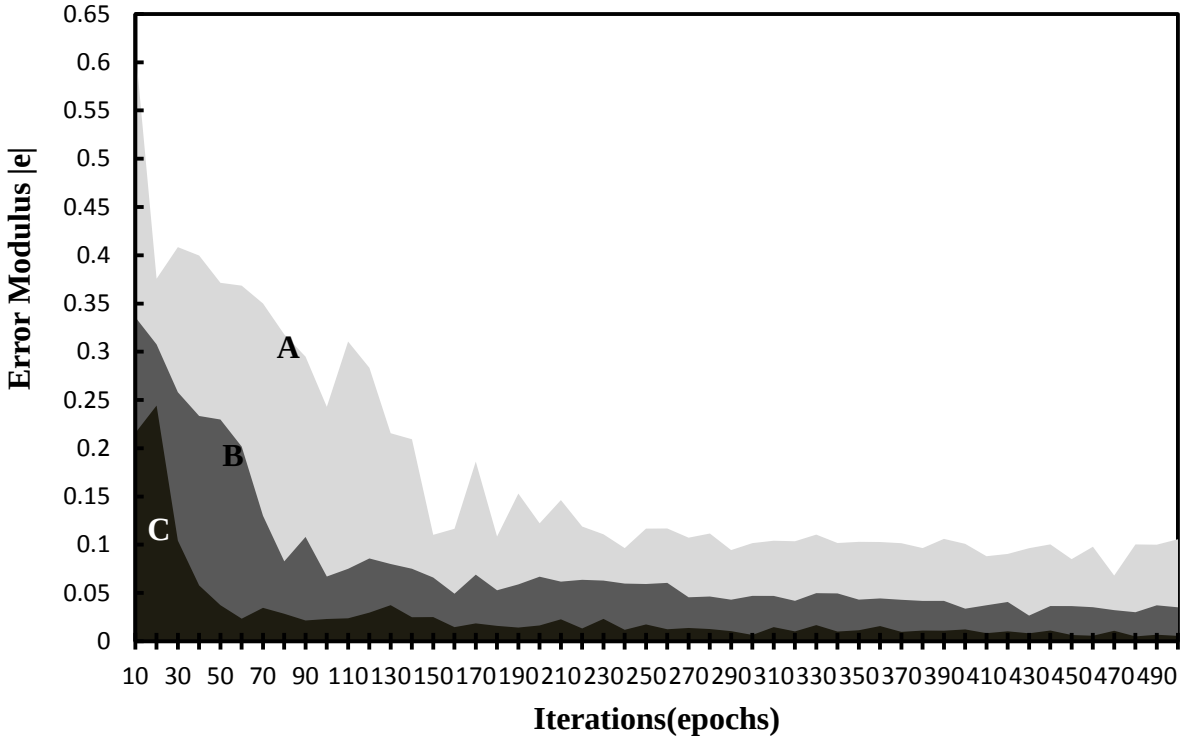


Figure 56: Enlarged graph of the Performance of a standard neural network with three hidden layer neurons when tested on the sparse dataset.

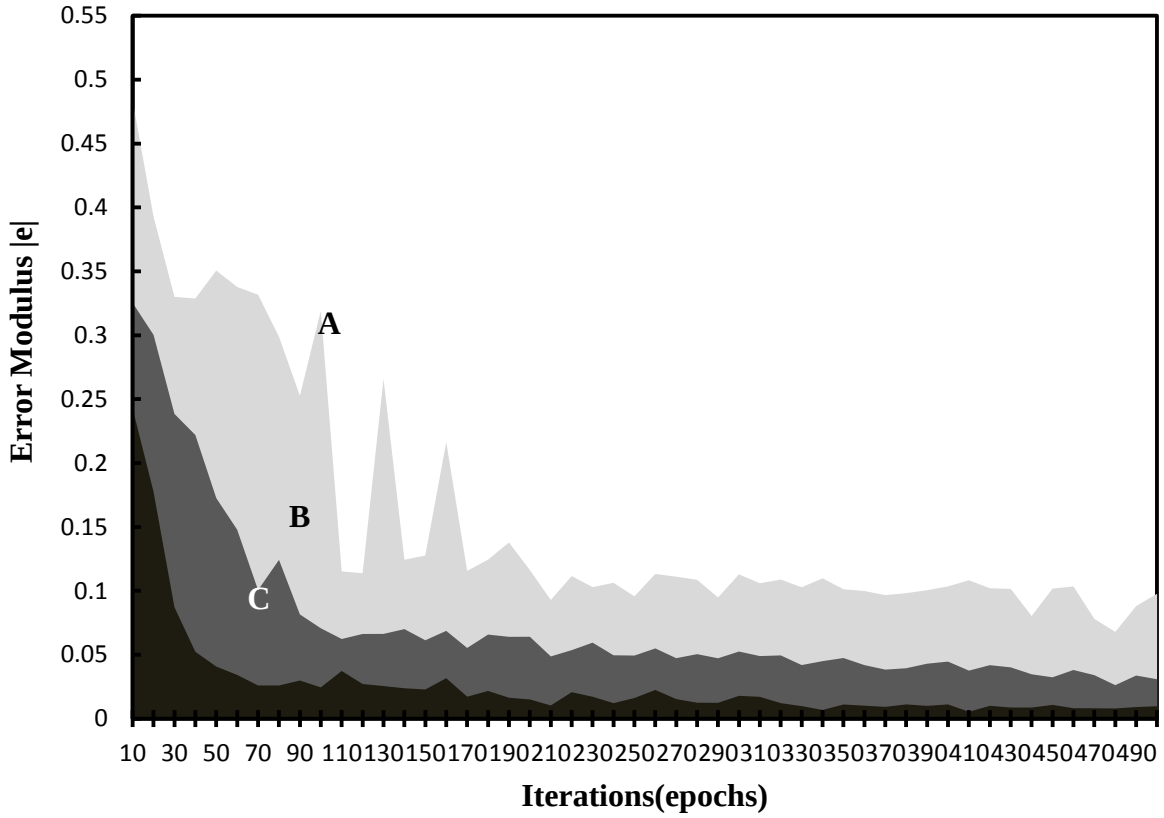


Figure 57: Enlarged graph of the Performance of a standard neural network with four hidden layer neurons when tested on the sparse dataset.

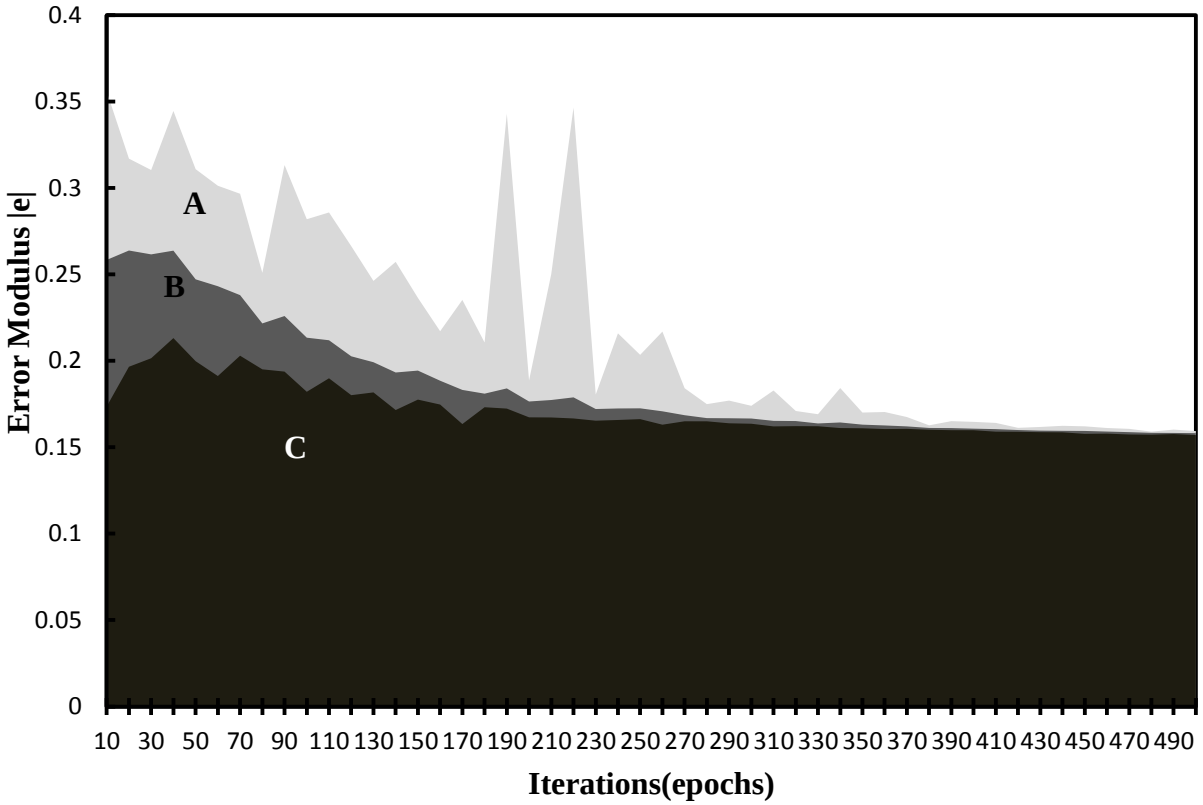


Figure 58: Enlarged graph of the Performance of a sigma-pi neural network with one hidden layer neuron and first-order expansion when tested on the sparse dataset.

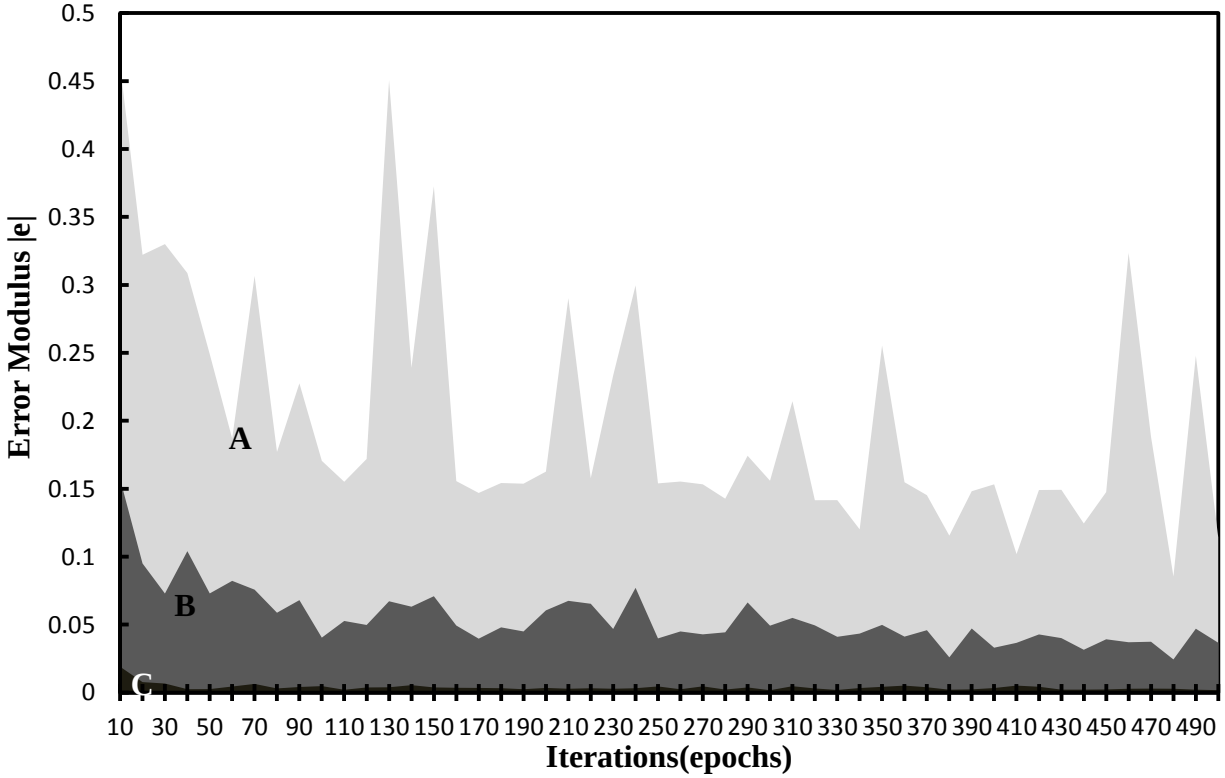


Figure 59: Enlarged graph of the Performance of a sigma-pi neural network with one hidden layer neuron and second-order expansion when tested on the sparse dataset.

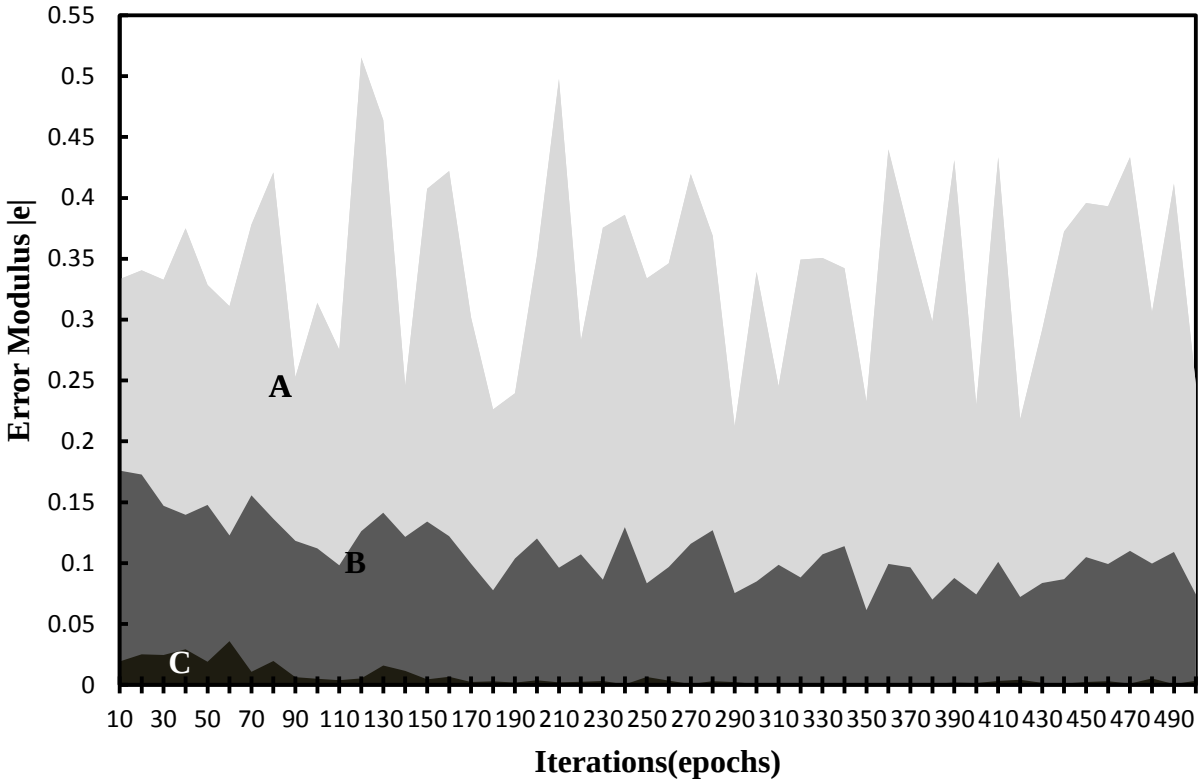


Figure 60: Enlarged graph of the Performance of a sigma-pi neural network with one hidden layer neuron and third-order expansion when tested on the sparse dataset.

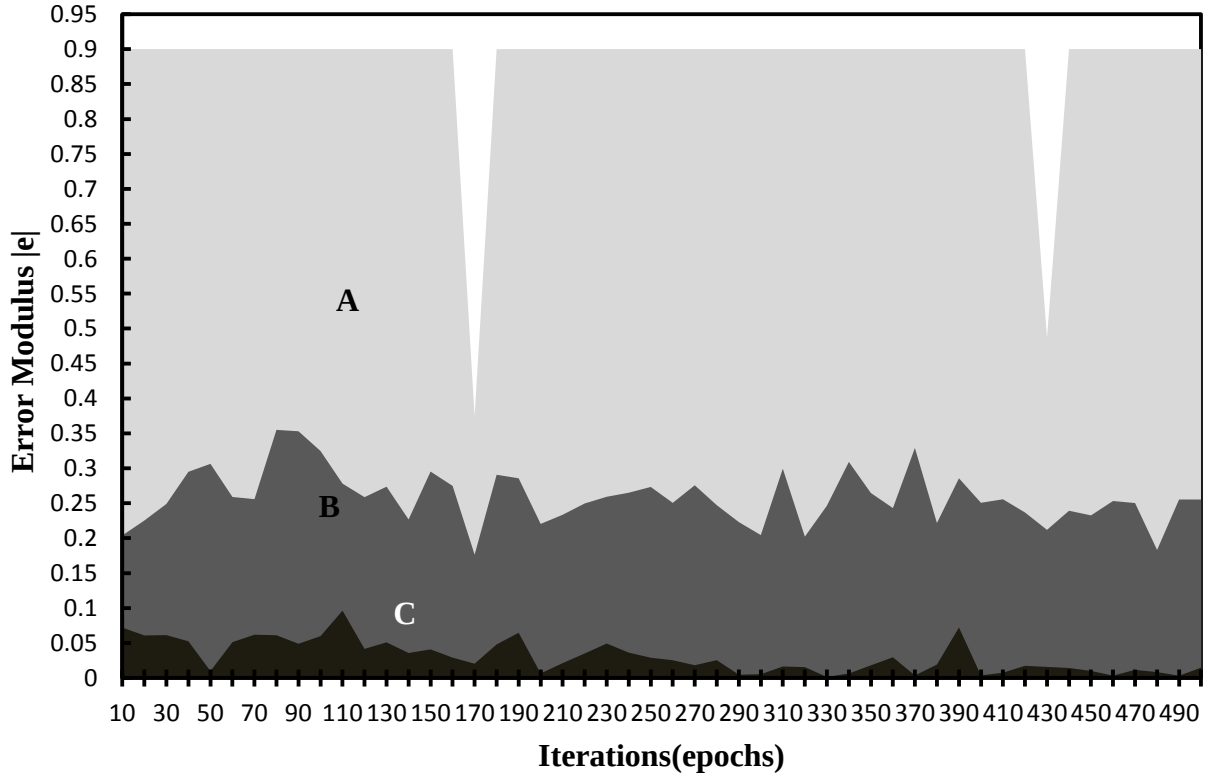


Figure 61: Enlarged graph of the Performance of a sigma-pi neural network with one hidden layer neuron and fourth-order expansion when tested on the sparse dataset.

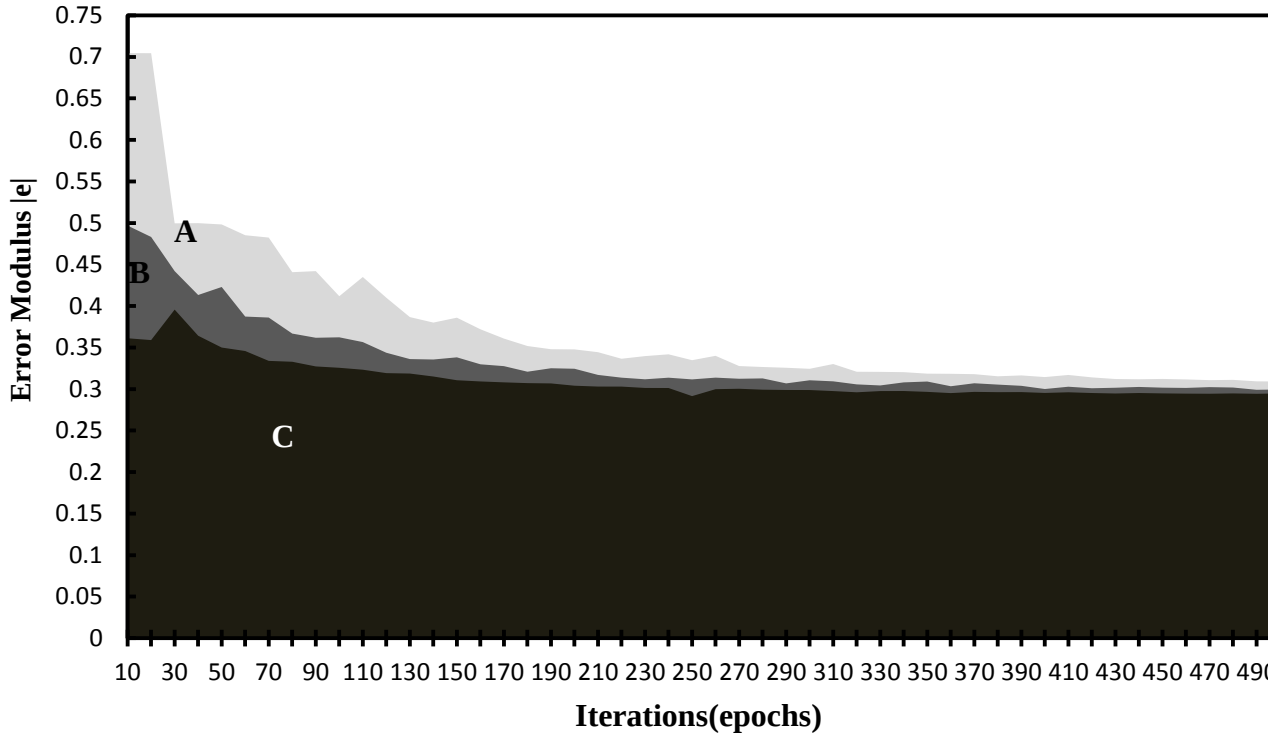


Figure 62: Enlarged graph of the Performance of a standard neural network with one hidden layer neuron when tested on the smooth dataset.

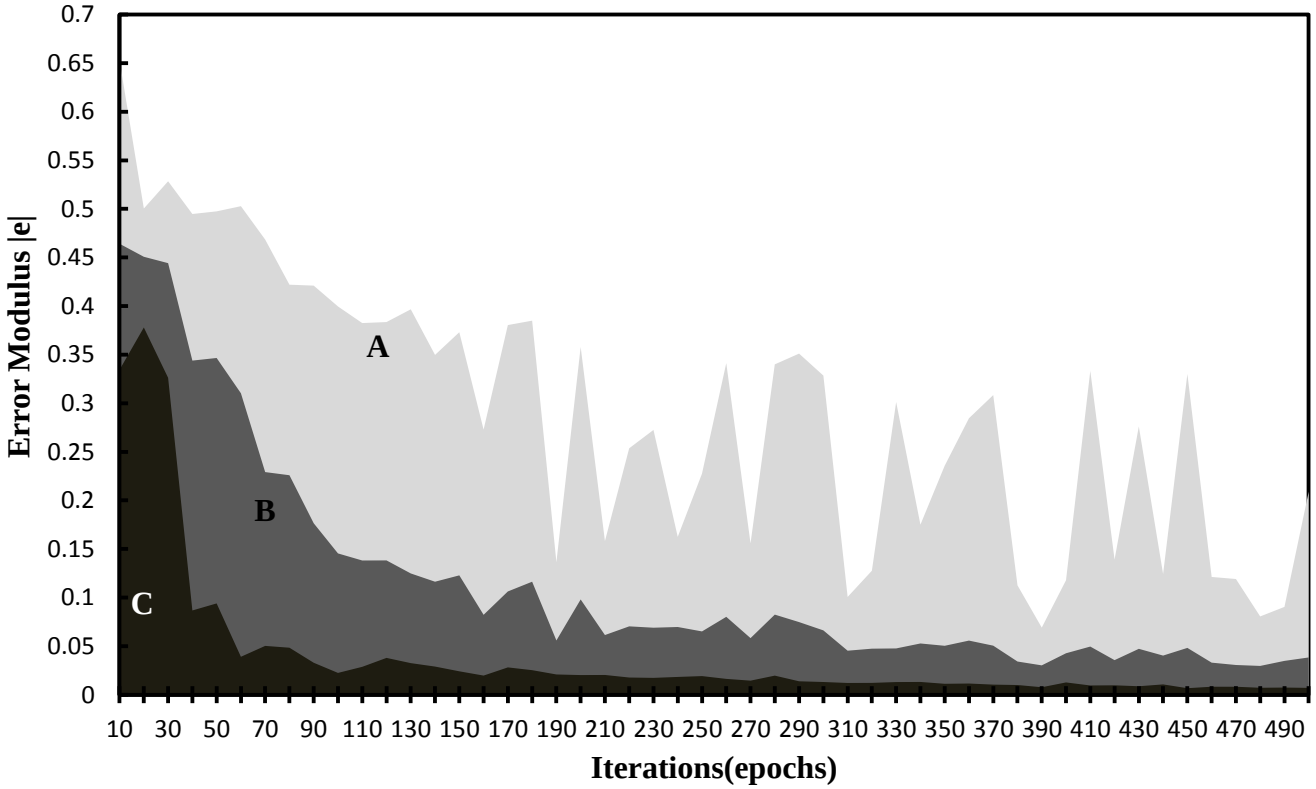


Figure 63: Enlarged graph of the Performance of a standard neural network with two hidden layer neurons when tested on the smooth dataset.

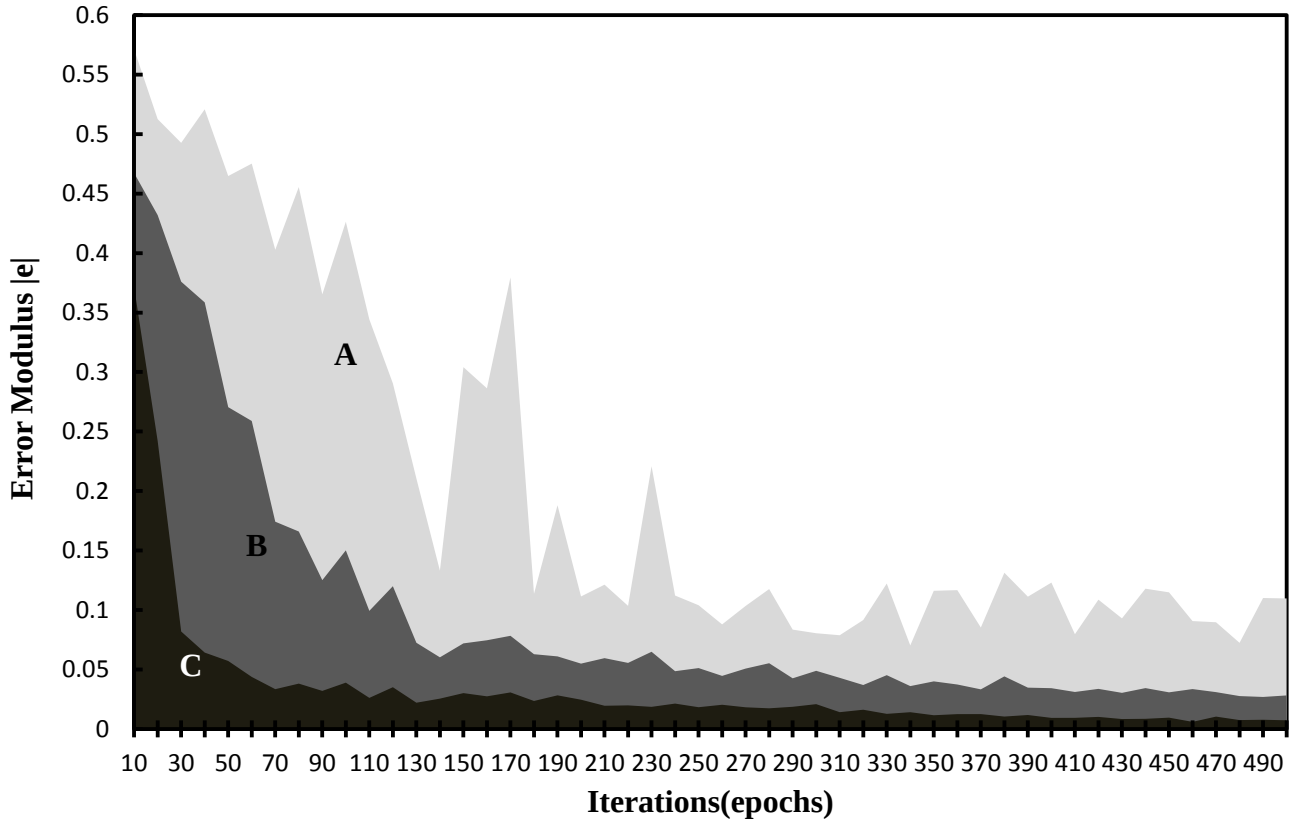


Figure 64: Enlarged graph of the Performance of a standard neural network with three hidden layer neurons when tested on the smooth dataset.

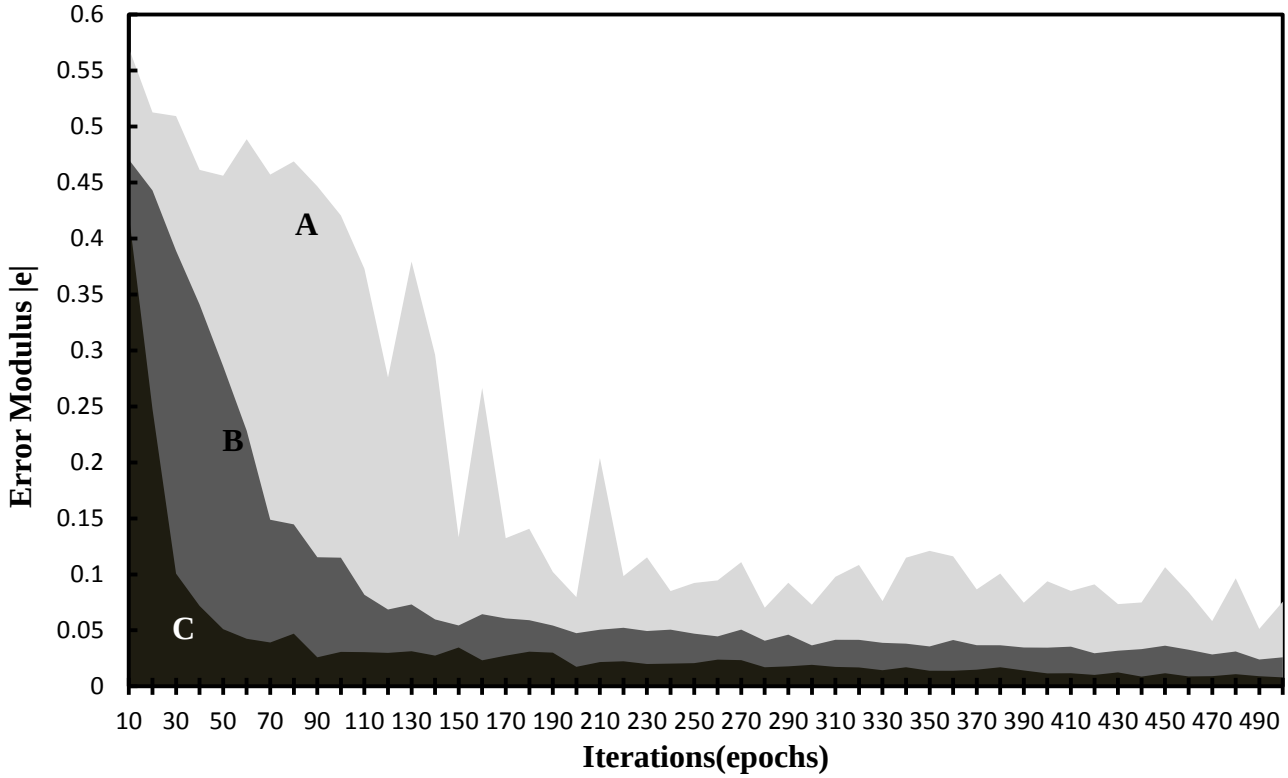


Figure 65: Enlarged graph of the Performance of a standard neural network with four hidden layer neurons when tested on the smooth dataset.

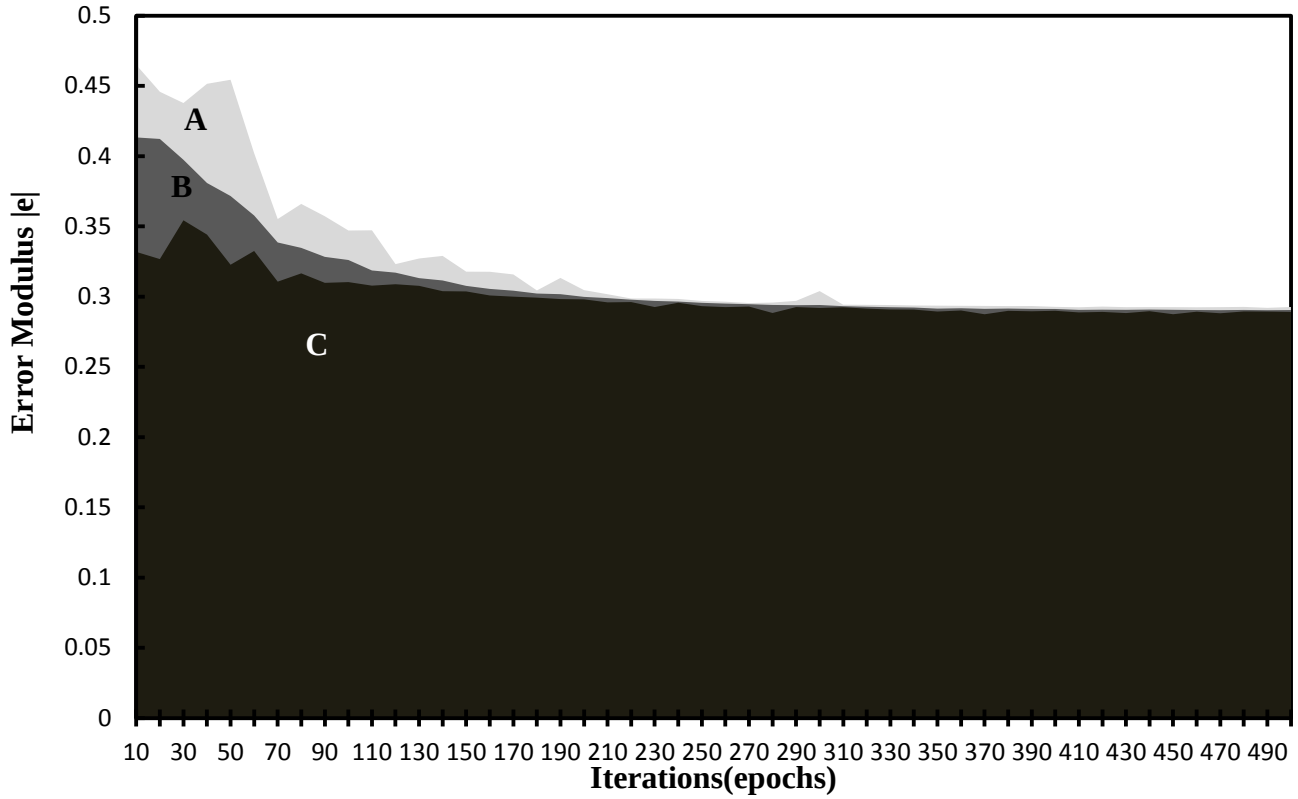


Figure 66: Enlarged graph of the Performance of a sigma-pi neural network with one hidden layer neuron and first-order expansion when tested on the smooth dataset.

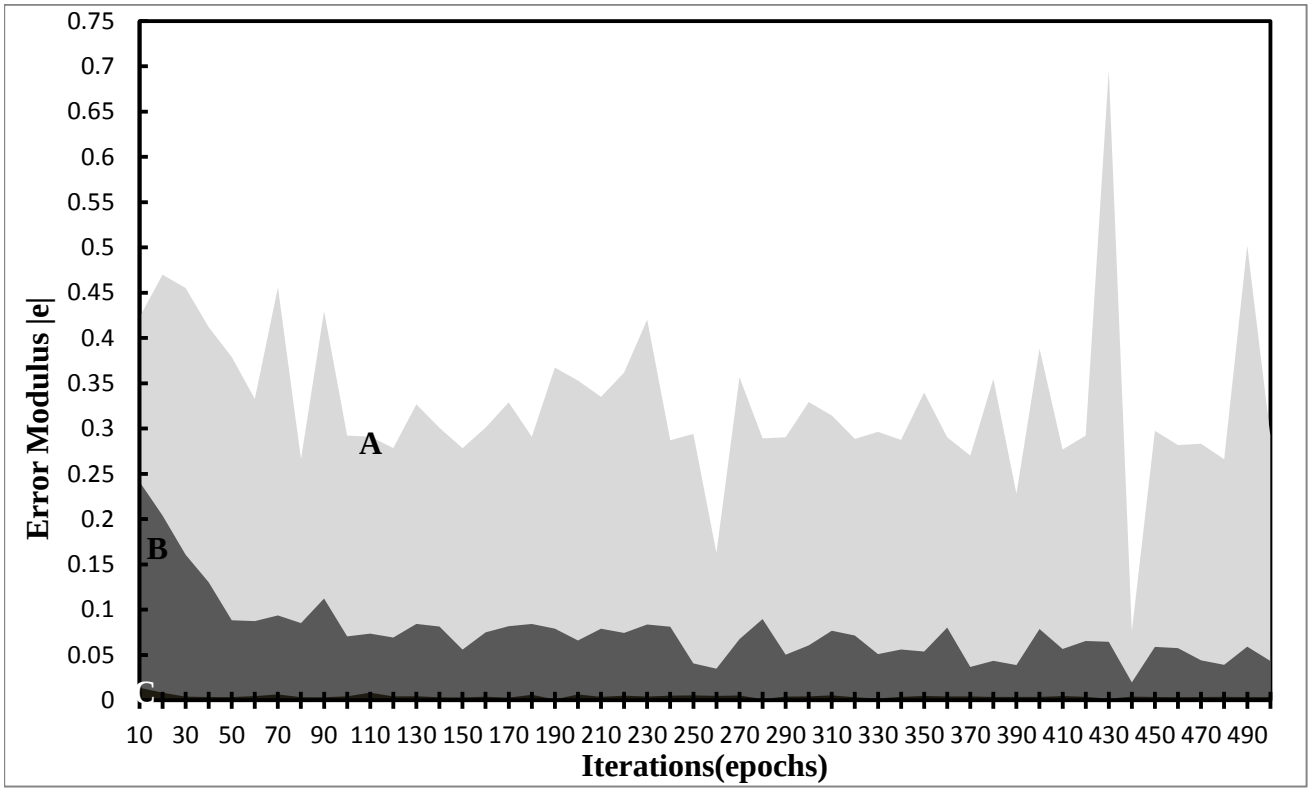


Figure 67: Enlarged graph of the Performance of a sigma-pi neural network with one hidden layer neuron and second-order expansion when tested on the smooth dataset.

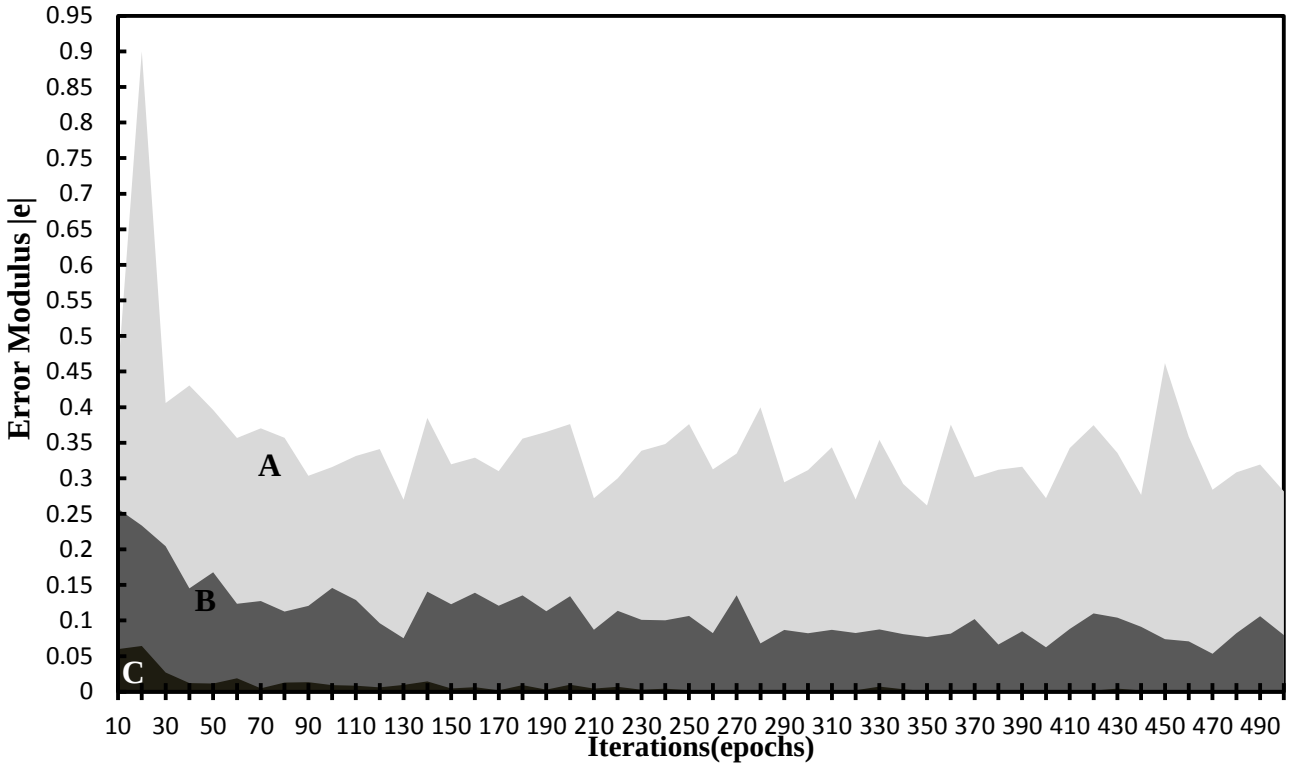


Figure 68: Enlarged graph of the Performance of a sigma-pi neural network with one hidden layer neuron and third-order expansion when tested on the smooth dataset.

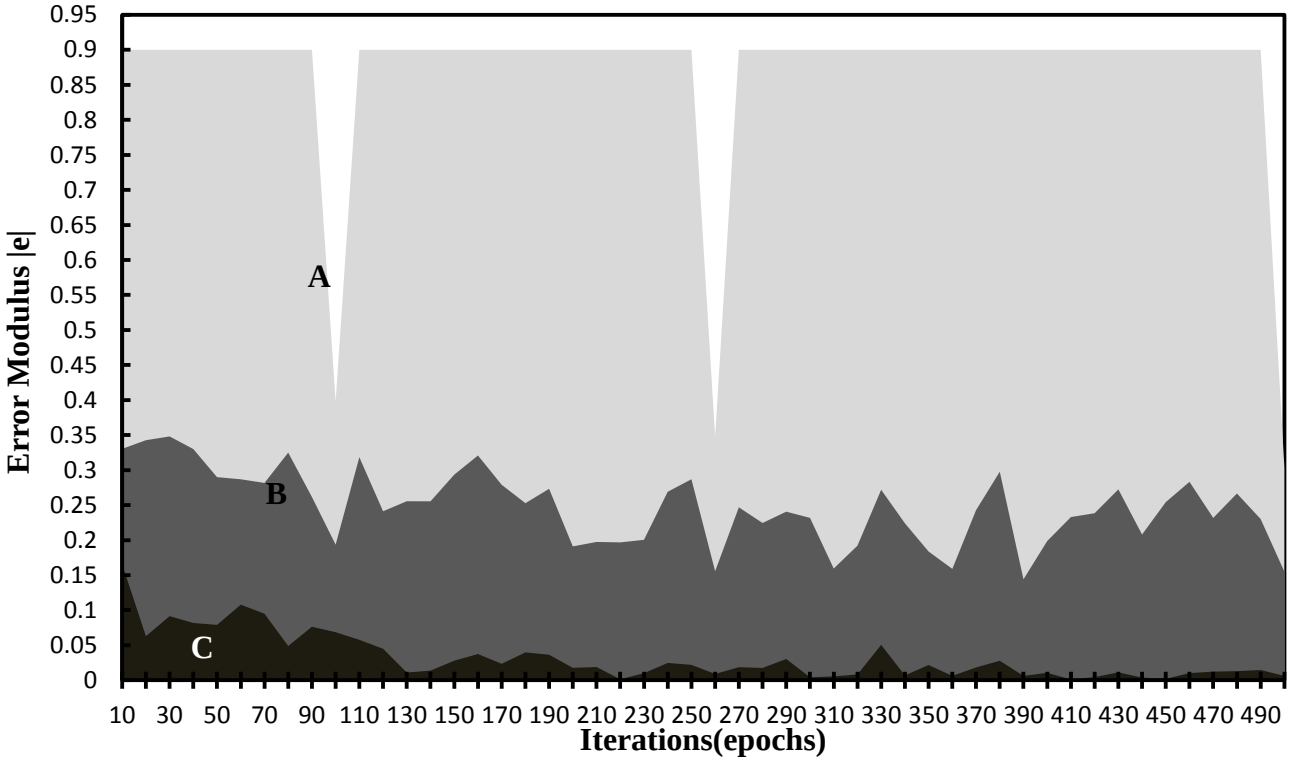


Figure 69: Enlarged graph of the Performance of a sigma-pi neural network with one hidden layer neuron and fourth-order expansion when tested on the smooth dataset.