An Empirical Study Towards Efficient Learning in Artificial Neural Networks by Neuronal Diversity

Abdullahi S. Adamu

Department of Computer Science

University of Nottingham

A thesis submitted for the degree of

Doctor of Philosophy

November 2015

for my mother and father ...

Acknowledgements

There are a lot of people I will like to acknowledge for their support and friendship during my Ph.D., and not all of them might be mentioned in my acknowledgments, regardless I treasure my time with them during my journey as a Ph.D. student.

Firstly, I will like to start by thanking my family, and friends as they have played the role of being amazing people to be with that made this experience memorable. I will also like to reiterate my deep appreciation of the supportive role my parents had during my time. I also really appreciate my wifes role in being both understanding and supportive, and my child's role in giving me lots of reasons to laugh.

I will like to thank my supervisors Dr. Tomas Maul and Prof. Bargiela, for their guidance, patience, insights, valuable advice, and support. They have offered words of encouragement at much needed times. I really appreciate their help in reviewing and giving me valuable feedback on various works during my study, including this thesis. Dr. Christopher Roadknight and Dr. Chong Siang Yew have also played a supporting role during my time as research student; they offered their insights, valuable advice at various times, which I appreciate greatly. I'd not forget Mr. Hoo Soi Hock and Mr. K.R Selveraj, who have both been very helpful, since my time as an undergraduate at the University. I appreciate their role in giving me the opportunity to connect back to my days as an undergraduate through the teaching assistantship program. I have benefited greatly from them. Finally, I'd also like to thank Richard Crossland, who helped me in the final checks of this thesis by proofreading for minor errors.

There have been lots of people who have written in an amazing chapter of my life through this journey, and I am glad they were part of it.

Abstract

Artificial Neural Networks (ANN) are biologically inspired algorithms, and it is natural that it continues to inspire research in artificial neural networks. From the recent breakthrough of deep learning to the wake-sleep training routine, all have a common source of drawing inspiration: biology. The transfer functions of artificial neural networks play the important role of forming decision boundaries necessary for learning. However, there has been relatively little research on transfer function optimization compared to other aspects of neural network optimization. In this work, neuronal diversity - a property found in biological neural networks- is explored as a potentially promising method of transfer function optimization.

This work shows how neural diversity can improve generalization in the context of literature from the bias-variance decomposition and meta-learning. It then demonstrates that neural diversity - represented in the form of transfer function diversity- can exhibit diverse and accurate computational strategies that can be used as ensembles with competitive results without supplementing it with other diversity maintenance schemes that tend to be computationally expensive.

This work also presents neural network meta-features described as problem signatures sampled from models with diverse transfer functions for problem characterization. This was shown to meet the criteria of basic properties desired for any meta-feature, i.e. consistency for a problem and discriminatory for different problems. Furthermore, these meta-features were also used to study the underlying computational strategies adopted by the neural network models, which lead to the discovery of the strong discriminatory property of the evolved transfer function.

The culmination of this study is the co-evolution of neurally diverse neurons with their weights and topology for efficient learning. It is shown to achieve significant generalization ability as demonstrated by its average MSE of 0.30 on 22 different benchmarks with minimal resources (i.e. two hidden units). Interestingly, these are the properties associated with neural diversity. Thus, showing the properties of efficiency and increased computational capacity could be replicated with transfer function diversity in artificial neural networks.

List of Publications

The following were the publications made during the course of the thesis [73, 4, 5, 6, 7]:

- Maul, T., Bargiela, A., Chong, S-Y., **Adamu, A.**,"Towards evolutionary deep neural networks", ECMS 2014 Proceedings of the European Conference for Modeling and Simulation, Brescia, Italy, 27-30 May 2014. doi:10.7148/2014-0319.

- **Adamu A.S**, Maul T, Bargiela A, "Efficient Learning by Co-evolution of Neurally Diverse Artificial Neural Networks", In the 2014 IEEE Symposium on Computers & Informatics, Kota Kinabalu, Malaysia. (*In Press*)

- **Adamu A.S**, Maul T, Bargiela A, "On Training Neural Networks with Transfer functions diversity", In the proceedings of the third International Conference on Computational Intelligence and Information Technology, CIIT&ITC 2013 (pp. 295–304). Elsevier - ISBN 978-81-910691-6-3. (**Print**)

- **Adamu A.S**, Maul T, Bargiela A, C. Roadknight "Preliminary Experiments with Ensembles of Neurally Diverse Artificial Neural Networks for Pattern Recognition", Recent Advances in Information and Communication Technology 2015, Springer International Publishing, 2015, 361, 85-96. doi:10.1007/978-3-319-19024-2_9.

- **Adamu A.S**, Maul T, Bargiela A,"Assessing the feasibility of approximating higher-order problem signatures in Artificial Neural Networks with hybrid transfer functions", International Journal of Computer Science Issues, 11(2), 8–18. ISSN (Online): 1694-078.

- **Adamu A.S**, Maul T, Bargiela A, Chong S.Y, "Cooperative Co-evolution of Neuronal Diversity Towards Self-adaptive Artificial Neural Networks", Neural Processing Letters *(undergoing revision).*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Neural Networks are biologically inspired machine learning algorithms, which are made up of a set of interconnected units known as neurons. The artificial neurons, just as their biological counterparts, are responsible for cooperatively coordinating a response which translates into a unanimous response to some input. Neurons are designated into three classes of layers: the input layer, hidden layer, and the output layer. The Input layer consists of neurons that relay information to the hidden layer, much like sensory neurons found in biological neural networks. The hidden layers consist of neurons that connect to other neurons on an adjacent hidden layer or the output layer and is where most of the computation takes place. The output layer is essentially a layer of neurons that presents the output in response to the given inputs. The response will differ depending on how these neurons are connected, and the various connection strengths between them - modeled as weighted connections. Much like how the biological neural networks adapt and learn, artificial neural networks optimize the various connection strengths between neurons, and in some cases even prune some.

In artificial neural networks, this is done using optimization algorithms that adapt these weights with respect to the error of the output neurons, typically measured by the mean squared error (MSE). However, while weight optimization is one of the leading aspects that contribute to learning in artificial neural networks, there are other aspects of learning in biological neural networks that also contribute significantly to learning. These include depth of the hidden layer, the connection pattern of neurons between layers, self-connections (i.e. recurrent connections), and much more. All these findings have been incorporated into neural networks with significant successes, from deep learning to recurrent neural networks.

Among all the components modeled into the architecture of artificial neural networks, transfer function optimization has received relatively little attention as also highlighted by W. Duch [37]. The transfer functions of neurons models the firing behavior of the biological neurons and is responsible for projecting decision boundaries critical for learning.

Traditional transfer functions such as the ones used in the Multilayer Perceptron (MLP) and the Radial Basis Network (RBF) are used as an informal standard across all sort of problems, even though there can be significant differences between one choice of a transfer function to the other.

The main argument of this work is that incorporating neural diversity as a means to optimizing transfer functions in artificial neural networks can result in the same efficiency associated with it in biological neural networks. Neural diversity is believed to be one of the major factors that behind efficient learning in biological neural networks [69, 70, 18]. It is only natural that it inspires a research direction for optimizing the transfer function of neurons in artificial neural networks. This is the major difference with other related works [61, 47, 48, 3, 54, 55, 38, 28] which are motivated by either the theory of duality of functions or the universal approximation capabilities of the units in RBF and MLP networks. As such, these works use a set of projection, and radial-basis functions alone. However, in our study, we have a variety of other classes of transfer functions that include higher-order units, and statistical units to replicate diversity in the classes of transfer functions. The resulting effect is that the neural networks can exhibit a wider variety of computational strategies for any given problem. The three major contributions of this thesis can be summarized as follows:

Firstly, this study explores how neural diversity can improve generalization ability in the context of the literature that links generalization to bias and variance, i.e. the bias-variance decomposition and meta-learning. Experiments with ensembles of neurally diverse artificial neural networks were then used to show that it can produce diversity in the form of various computational strategies, which is an essential part of ensembles. The resulting members of the ensemble were both compact and accurate and had competitive results without the need for explicit diversity maintenance - which tends to be computationally expensive.

Secondly, the study then explores the problem of dimensionality as a result of the increased number of possible computational strategies when working with neural diversity in the next chapter (Chapter 5). The hypothesis is that it is possible to characterize problems by their computational strategies, which is made up of their learned topology, weights, and choice of transfer functions. It was shown that their pattern of computation can characterize problems, and some of these patterns were explored using neuroscience-inspired measures to unveil their insights and uniqueness to different problems. The chapter also presented the possibility of these being used as meta features which can help guide the choice of transfer functions, and as a result, significantly reduce the dimensionality problem.

Finally, the study culminates the findings of the thesis in the following chapter (Chapter 6) by presenting a holistic approach to neural network optimization that trains the three major components of the artificial neural network: weights, connectivity/topology, and transfer function choice (with their respective parameters) simultaneously. The objective was to tackle the problems associated with the bias as a result of neglecting other components of the neural networks during training, which accumulates and contributes to the error of the neural network on patterns, and its generalization ability in general.

The subsequent sections of this chapter will highlight in more detail the motivation, major contributions, and overview of this thesis.

1.1 Motivation

There are various approaches to training neural networks. Among the least researched - as also expressed by Duch [37] - are approaches that depend on optimizing the choice of transfer functions simultaneously with other essential components like topology and their weights. The notion of transfer function diversity is one of the nature inspired facets of transfer function optimization. Transfer function optimization is important for many reasons, one of which includes its important role in forming decision boundaries necessary for learning. Other motivations for researching ways of optimizing transfer functions are to improve the efficiency and robustness of learning in artificial neural networks (ANNs). Interestingly apart from its biological plausibility, results from neuroscience have found that neural diversity is one of the reasons why biological neural networks are both efficient and robust [69, 70, 18]. In this chapter, we introduce the critical role of transfer functions in learning as well as the motivations for researching on the benefits of neural diversity in ANNs.

The transfer function of an artificial neuron, h is composed of an activation function, $j_h = g_h(x, W)$ and an output function, $f_h(j_h)$; the activation function computes the action potential, j_h from the input and weight values (i.e. $x = \{i_1...i_k\}$, and $W = \{w_1...w_k\}$) of the incoming connections. The output function computes the output of the neuron in response to the action potential. This transfer function $z_h = f_h(g_h(x, W))$ is responsible for forming the decision boundaries in the input space among other tasks such as feature selection. The shape and form of these decision boundaries vary with the types of transfer functions. Radial basis functions (RBF), for example, have a cluster-like behavior of creating decision boundaries: points in the input space closer to its center have a higher response while points further away from the center tend towards progressively lower responses. Linear functions are more discrete in the way they form boundaries: they form a hyperplane in the input space, which divides the points. Sigmoid functions are similar but differ because they tend to be continuous, rather than discrete. Any learning algorithm attempting to learn a classification problem has to be able to form boundaries that classify observations (i.e. points in the input space) into their respective classes. This has to be done on a sub-sample of the data set that is representative of the whole sample to avoid the learning algorithm from forming an incorrect generalization of the problem. Ideally, the neural network should learn the underlying function of the problem and be able to predict unseen data with significant accuracy, and that is the essence of *generalization*. Although transfer functions have a critical role in learning, there has been relatively little research in this direction compared to other aspects such as the training algorithm, as Duch also explained in [37].

Studies have shown that neural networks with transfer function optimization are more efficient [54, 55, 38, 49, 71, 6, 5]. By *efficient* we are referring to models of relatively low complexity (i.e. regarding the number of hidden neurons) that learn a problem with possibly better generalization ability. Traditional and well established ANNs such as Radial Basis Function Networks (RBFN) and Multilayer Perceptrons (MLP) use predetermined classes of transfer functions for the neurons of each layer as part of their distinct architectural properties [40]. Often, the neurons of each layer are homogeneous (i.e. are the same) in their choice of transfer functions. An RBFN typically uses a combination of a distance-based activation function, and a radial basis function such as a Gaussian or multiquadratic function. In the case of Multilayer Perceptron, it is typically an inner-product activation function accompanied with a sigmoid output function such as the hyperbolic tangent. Though it is proven that both Radial Basis Functions Networks and Multilayer Perceptrons can approximate any function, given the complexity of the network's model is matched with the complexity of the problem [17]; this does not guarantee that the models produced are the most optimal or practical for all problems. This is because matching the complexity of unusually complicated problems typically involves growing the neural network's hidden layer. Although the computational capabilities of computers are continually being improved especially with the advent of cloud computing; training can become more difficult as the neural network gets larger. Besides, it is preferable to have neural networks of lower complexity because they are often associated with good generalization ability [33], and are much easier to analyze.

Furthermore, transfer function optimization can lead to more robustness in learning [54, 55, 38, 49, 71, 6]. This all boils down to the no free lunch theorem; no algorithm is universally the most optimal for all possible problems. However, for a small subset of challenges, there can be an algorithm that is more robust. In other words, learning algorithms can't be perfect for all problems. However, the interest in machine learning is usually not to develop learning algorithms that can solve all problems; it's to develop algorithms that are robust enough to learn as many practical and useful problems as possible - which is a much smaller set of challenges, such as pattern recognition, autonomous control, and more. For a learning algorithm to be robust, it should be able to adapt its *bias* to search for the computational strategy or hypothesis that best describes the underlying function of problems. In the case of ANNs, the weights, topology and transfer functions are among the components that have an inherent bias [75, 102], which will need to be optimized according to the problem to enable the neural network models to adapt to a variety of problems. The increased computational capacity as a result of transfer function optimization and appropriate training leads to a robust learner. In this work, we especially focus on the study of transfer functions and topology because of their critical role in learning.

Finally, ANNs are nature inspired, so it's only natural that discoveries from neuroscience continue to guide and inspire the research direction of ANNs. One of those discoveries is neural diversity. There is a lot of diversity in biological neurons; in the retina alone, there are more than 50 different types of neurons. Marder [69, 70] also found that even in neurons that are anatomically identical, there are some behavioral differences between them when they are subjected to the same stimuli. Briggman et. al. [18] also found that diversity in biological neural networks is responsible for their efficiency, making them more compact. In addition to that, they also believed it enables biological neural networks to exhibit a wider range of responses without the need for more neurons. In other words, it is one of the reasons behind the enormous computational and representational capacity of biological neural networks.

In summary, transfer functions perform a critical role in learning. They are behind a significant portion of learning which involves approximating the underlying function that describes problems, by forming decision boundaries. However, to be robust and efficient, ANNs have to adapt their bias according to the nature of the problem. Transfer function optimization is one such approach for optimizing the transfer function according to the problem among other components such as weights and topology. Unfortunately, there has been relatively little research in this direction [37]. This work is particularly novel as it is inspired by the property known to be responsible for efficiency, robustness, and compactness in biological neural networks; i.e. *neural diversity*.

1.2 Major Contributions

This thesis presents contributions across various aspects of efficient and robust learning using the property of neural diversity found in biological neural networks. The major contributions can be summarized as follows:

- Establishing the relationship of neural diversity to generalization ability in the context of the literature, and providing empirical results to support that.
- Showing that neural network meta-features (described as *problem signatures*) can characterize problems, and can be a valuable tool in analyzing underlying computational strategies evolved by neural networks.
- Finally, a holistic approach to optimizing artificial neural networks that involve the co-evolving neural diversity with topologies and weights to reduce bias, and increase the ability to perform more efficient information transfer between components using evolutionary operators such as crossover.

1.2.1 Establish the Relationship of Neural Diversity to Generalization Ability

The first contribution of the thesis was to review the various related works done in transfer function optimization, classify them according to their motivation, and establish a notion of how neural diversity could lead to better generalization in the context of the bias-variance decomposition. There were three major motivations behind works found in the literature [61, 47, 48, 3, 54, 55, 38, 28]: (i) Donoho's theory of duality of functions; (ii) meta-learning; and (iii) transfer function flexibility. Though these works presented promising results, there are still relatively few works on transfer functions in general. Also, most of these works were mostly not extensive. Furthermore, as per the writing of this thesis, there is a lack of works that specifically and explicitly relates transfer function optimization to generalization. This thesis contributes to the literature by presenting a biologically inspired facet of transfer function optimization, which has been found to have promising results in preliminary experiments [71]. Furthermore, it also explains specifically how the bias of components of neural networks affects generalization ability in the context of the bias-variance decomposition. It then explains how neural diversity, represented as transfer function diversity in this thesis, can enable the neural network to adopt more forms of bias which result in being able to access and search more regions of the hypothesis space. In other words, neural diversity increases the computational capacity of artificial neural networks. Consequently, this increases the probability of finding the most appropriate solution that describes the problem's underlying function. Additionally, experimental results are used to show that transfer functions can be used as a diversity maintenance scheme for neural network ensembles. This is due to the ability of neurally diverse artificial networks to adopt different inherent biases by adapting the choice of transfer functions. This argument is further emphasized by analyzing the computational strategies which lead to the unveiling of two distinct computational strategies evolved for the diabetes dataset. Those findings pave the way for the next contribution which exploits the performance differences of transfer functions to characterize problems, among other tasks.

1.2.2 Meta-features for Problem Characterization and Analysis of Computational Strategies

The second major contribution of this thesis, which constitutes the computational signatures chapter contributes to the literature in several forms. Firstly, it explains the idea of computational signatures, which could be regarded as meta-features. Meta-features in the literature have typically been considered to be either a set of features resulting from analyzing a dataset [102, 103], or performance information of various learning algorithms on a dataset which also provides some degree of insight into the nature of the problem [61, 103, 3]. Though performance data of different learning algorithm could arguably provide some level of information regarding the properties of problems, it was the opinion of the author that more reliable features can be unveiled by studying the models that learn the problem. Related works include the use of features such as the depth of decision trees after training on a dataset as a meta-feature [81]. Unfortunately, just like the study of transfer functions: there seems to have been very limited work in this direction, specifically for artificial neural networks. This works contributes to the literature, by specifically developing meta-features based on neural network models. It also shows that these meta-features have the desirable properties of features, i.e. consistency in being discriminative between problems that are different, as well as a significant degree of invariance for a problem regardless of changes to various parameters. Additionally, it was also shown that these signatures can be used to form digraphs representing the building blocks of the model that learned the problem, which in essence represents some information about the nature of the problem. These graphs are then used to verify the feasibility of the neural network computational signatures as meta-features.

This was also partially motivated by the tendency of neural networks to be used as black box learning algorithms. There is yet to be a significant contribution in the direction of techniques for analyzing the hypotheses learned by ANN models. However, neuroscience has a variety of established methods used for studying biological networks. Another contribution of this thesis is the development of neuroscience-inspired analysis techniques for the study of ANNs. The graph-based approach which uses the meta-features to form an expectation of the computational strategies found to be most appropriate for the problem also provided a method of not only understanding the neural network models, but also the nature of the problem itself. The results of these have been used to show interesting computational strategies, including one that shows how a neural network model creatively used *min* and *max* activation functions to filter the most important features of the diabetes problem, namely *age*, and *skin fold thickness*. Interestingly, these features were found to be some of the leading factors that increase the risk of diabetes by the American Diabetes Association (ADA) [10].

1.2.3 Co-evolution of Neural Diversity and their Topologies for Self-Adaptive and Efficient Learning

The third contribution of this thesis was the co-evolution of neural diversity with the topology and weights. In traditional representations of artificial neural networks meant to be optimized using evolutionary algorithms (EAs), a single genetic string is usually used. Though there have been very promising results from this approach [108, 110, 12, 34, 97, 98]. It has also presented problems such as the cross-over operations being particularly detrimental to the performance of the neural networks. This thesis contributes to the field by particularly co-evolving a representation of the model space regarding its major components, i.e. weight, topology and transfer functions to achieve better information transfer during evolutionary operations. Transfer function diversity, the flexibility to evolve any form of topology, and the improvement in knowledge transfer are all meant to reduce the bias of the neural network thereby increasing the chances of achieving better generalization ability. Though there are related works that have co-evolved neural networks using various representation, evaluation, and optimization techniques, such as SANE [78], COVNET [44] CoSyNE [46], and EPNET [111], this work differs as it co-evolves the three primary components of the neural network represented as three interdependent sub-spaces. The third sub-space, unlike the aforementioned approaches, lends itself to the property of neural diversity incorporated into the neural networks for transfer function optimization. In other words, it was a holistic approach to optimizing artificial neural networks.

The results show that co-evolving the components of neural networks with neural diversity can enable them to achieve significant generalization ability with comparatively less complex models consisting of two hidden units [6]. In particular, the results tested on 22 benchmarks from the machine learning repository (UCI) [13] and on the PROBEN1 [89] benchmark show an average mean squared error (MSE) of 0.30, which is significant for a neural network of that complexity. Finally, another contribution was to show that injecting more neural diversity during the co-evolutionary process also significantly increased both the generalization ability of the neural network and its convergence.

1.3 Overview

The thesis constitutes four main chapters while the remaining chapters present the literature review of the subject area and the conclusion of the thesis.

Chapter 2 reviews the literature on the subject. Apart from presenting some of the basic knowledge of both biological and artificial neural networks, it presents some of the works from neuroscience on neural diversity. In particular, it highlights the works which have associated neural diversity with efficiency and increased representational capacity. It also highlights related works from the field of artificial neural networks, particularly on hybrid neural networks, that were motivated by a variety of reasons for optimizing transfer functions.

Chapter 3 explains the nature of the search space of artificial neural networks. It then highlights the related works in transfer function optimization and the motivation for using neural diversity in artificial neural networks. In particular, it shows how transfer function diversity can improve generalization ability in the context literature that links generalization to bias. Specifically, the bias-variance decomposition and the meta-learning theory on the search space of learning algorithms.

Chapter 4 then shows conclusive results of neural diversity leading to a repertoire of accurate computation strategies for neural networks in an ensemble. Neurally Diverse Artificial Neural Networks (NeuDiME) was used to show that neural diversity is also an effective diversity maintenance scheme for ensembles as it exhibits a broad spectrum of accurate computational strategies, which is an essential part of ensembles. The indication of differences in the use of transfer functions for different problems in the results of this chapter paved the way for the next chapter on *computational signatures*.

Chapter 5 highlights the problem of increased dimensionality of the search space, which results from transfer function optimization. Computational signatures are presented as a means to both initializing and managing the set of transfer functions during optimization, thus strategically overcoming the complexity of the search space. The chapter presents related works from meta-learning. It then highlights how neuroscience-inspired measures can be used to examine artificial neural networks. Furthermore, it shows how the results of the computational signatures produced from the analysis of neural network models with the neuroscience-inspired measures can be consistent for a problem and discriminatory between different problems. It also shows results of using a graph-based analysis technique developed to study the expected computational strategy for various problems.

Chapter 6 culminates the observations of chapters 3 and 5 into a holistic approach to optimizing artificial neural networks by co-evolving neurally diverse neural networks with their weights and topologies. It highlights the issues of compounding bias and information transfer and how they can affect generalization ability. The chapter then highlights the framework design and justifications. It also presents the results of using co-evolving neurally diverse artificial neural networks with their weights and topologies on 22 benchmarks and compares them to some of the state of the art learning algorithms in addition to learning algorithms such as the MLP, RBF, and SVM (Support Vector Machines). The chapter also compares the results from NeuDiME to the results obtained using the framework. It also highlights interesting results of hybrids and the variance of these learning algorithms such as the MLP-RBF, which consists of a projection layer and a radial basis layer as used in some related works.

Chapter 7 then concludes with the implications of the findings made in the thesis as well as their contributions.

Chapter 2

Background

"These networks represent functions in much the same way that circuits consisting of simple logic gates represent Boolean functions." - Russel & Norvig (1995)

In this chapter, a brief background is given for the topics involved in the scope of this thesis. Some basic familiarity with neural networks and evolutionary algorithms is assumed; thus, the background does not highlight all the aspects of each topic.

2.1 The Biological Neural Network

In general, the most fundamental unit in a nervous system is a single cell, called the *neuron* [68]. We know that each neuron is made up of a cell body, also known as the *soma*, which like other types of cells contains a nucleus within the walls of the cell (i.e. the membrane). There are strands of fibers that extend from this cell body to the proximity of other neurons, known as *dendrites*. These are responsible for receiving signals from other neurons. Furthermore, each neuron also has a long branch-like fiber that extends farther than the dendrite to the proximity of the dendrites of other neurons, where it fans out in branches. This branch-like fiber, known as the *axon*, is specialized at conducting a certain type of electric impulse and is insulated by a membrane, known as the *myelin sheath*. As illustrated in Fig.2.1, the axon is much thicker than the dendrites. Axons conduct electric impulses from somewhere between the cell body and the *axon hillock*, where the impulses are generated and transmitted through the length of the axon and then along its branches to its tips.

There are various ways in which neurons communicate; the connection illustrated in Fig.2.1 is the tip of each axon ending (or *axon terminal*) with one neuron being really close to the cell body of another cell, though not quite being in contact. The connection of an *axon termini* to a cell body or dendrite is known as a *synapse*, and the gap in the synapse where the fibres do not quite make contact is called the *synaptic cleft*. It is known that neurons are not restricted to making connections with neurons alone; they also make connections to various other types of cells within the nervous system. Essentially, neurons like the ones illustrated in Fig.2.1 connected in a neighborhood are defined as *neural networks*.

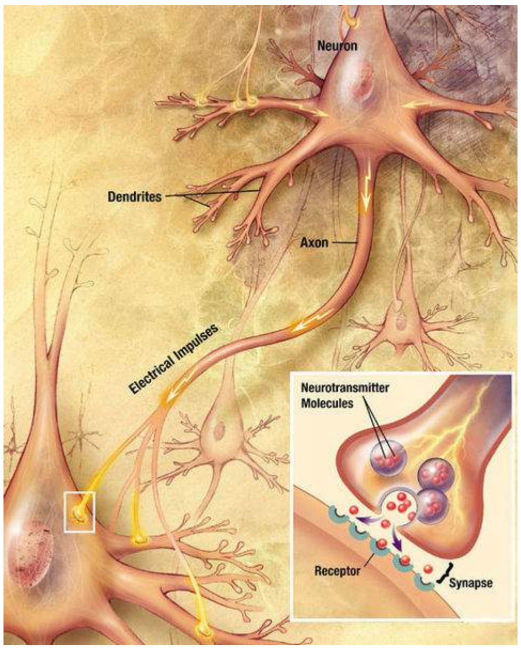


Figure 2.1: An axon terminus of a neuron connecting to the body of another neuron. *(Courtesy of Wikimedia Commons)*

Some of the important agents that aid the transmission of signals between cells in the network are *neurotransmitters*. Neurotransmitters are chemicals stored in vesicles found at axon terminals that are released by the cell at the transmitting side of the synapse (i.e. the pre-synaptic cell) and diffuse to the cell at the receiving end of the synapse (i.e. the post-synaptic cell). It's release is caused by a series of actions initiated by an electric impulse, known as the *action potential*. The action potential is a type of electric impulse that is characterized by a series of changes in the electric voltage in the axon body which result in spike-like graphs when measured. It is generated as a result of significant stimulation caused by the signals of other cells in the network. This results in an impulse being generated between the cell body and the axon of the stimulated cell, which quickly gets conducted along the length of the axon. It then fans out to all the terminals of the axon where it causes a rise in calcium ions (Ca^{2+}) concentration. This in turn causes the vesicles containing the neurotransmitters to fuse with the plasma membrane (i.e. the *cell wall*) leading to them being opened in the direction of the synaptic cleft. The neurotransmitters travel to the post-synaptic neuron's where it increases the electric potential of the cell at that moment (the membrane electric potential) by changing the ion permeability of the membrane. Given that the cell at the end of the of connection is a neuron, and that the stimulation is sufficient, this will result in the neuron generating an action potential. This process is then repeated at the end of the axon termini of the excited neuron. Depending on the properties as well as the presence of hormones, the response of each neuron will be different [101, 18].

The chemical signals at the post-synaptic neuron are eliminated in various ways; one method is through the use of enzymes attached to the network which destroys the neurotransmitter [68]; others include either leaving chemical signals to fade away by diffusion or to be taken back by the neuron that initiated the signal.

2.1.1 Learning in Biological Neural Networks

There are a variety of things that aid learning in the brain. As highlighted in the earlier section, these include the use of hormones, enzymes, neurons and various other forms of communication that lead to learning and a coordinated response.

Learning in the brain is a highly complex process. Generally, stimuli from experiences form, reinforce or prune connections between neurons, which results in an adaptation of the neural circuitry [104, 24]. The process of refining connections between neurons is lifelong; the most active connections are continually strengthened, while connections that are relatively inactive get weaker. At some point connections that are not active get pruned. This process results in both specialization and efficiency [104, 24]. This finding influenced the concept of the McCulloch Pitts neuron, which eventually gave rise to the Perceptron.

Other mechanisms that are believed to be important for learning include components within the biological network - such as neurotransmitters, and feedback signals - and elements in the form of behavior, such as sleeping. The neurotransmitter is also a complex signal pre-processor and is crucial for learning and adaptation [104]. Another role of neurotransmitters is sending feedback signals, after activation of a post-synaptic neuron. These feedback signals - sometimes in the form of soluble gasses - are sent backward to modify the behavior of the pre-synaptic neuron [104, 24]. In the category of mechanisms that aid learning from a behavioral perspective is sleep, where our brain has been found to strengthen memory, by stimulating itself in the form of dreams [23]. In general, there are various mechanisms both from behavior and within the biological neural network that are related to learning, either in part of full.

In the context of the human brain, learning can be characterized by its developmental stages. At infancy the brain has an almost fully connected set of neurons relative to its adulthood stage [104, 24], which enables it to adapt to almost any environment. To be specific there are an estimated 15 - 32 billion neurons [24], with each having about 10,000 connections. Towards its later stages, the human brain is shaped and refined by experiences.

In hindsight, it is interesting to see how all these neuroscience findings have directed research in the direction of Artificial Neural Networks. The findings of neuroscience and their inspired artificial neural network research can be listed as follows: Firstly, the findings of the Hebbian learning theory and the excitation of neurons by McCulloch-Pitts inspired Perceptrons. Secondly, the finding of hormones that affect the behavior of neighboring neurons inspired ART (Adaptive Resonance Theory) by Grossberg (1976). In our current time, deep learning has been inspired by findings relating to the deep nature of the brain, which has also had a lot of success [14]. Also, findings related to sleeping have also inspired training routines for neural networks that stimulate them with random stimuli to consolidate learned patterns in a sleep-wake learning cycle [51].

In summary, it can be said that neuroscience is likely to play an even greater roles in Artificial Neural Network research as more technologies for studying the brain and unraveling its mysteries are developed.

2.1.2 Neural Diversity in Biology

This thesis presents research in the direction inspired by the neural diversity property [18, 101, 99] of biological neural networks towards artificial neural networks that are more efficient and with better generalization ability.

Neural diversity is one of the characteristic properties found in biological systems; for example, there are no less than twenty classes of interneurons in the hippocampus alone. In areas such as the retina, there are more than fifty neuronal types. And it is not just coincidental, as research shows that neuronal diversity in these systems has some intrinsic advantages [101, 18, 99].

Apart from neurons being distinct by structure, it is also proven that neurons of the same group are also diverse in terms of behavior. Initially, slight differences in the responses of neurons to incoming stimuli were believed to be noise. However, isolating the neurons and testing their responses to stimuli showed that the response of each neuron - though anatomically similar - varied. Further research at Carnegie Mellon University has taken the research further to alter the stimuli delivered to the neurons; the results showed that neurons responded based on the characteristics of the stimuli [101]; for example, while stimuli characterized by rhythms triggered responses in some, it did not in others, and while some responded to stimuli characterized by rapid changes, others did not. As this research explains, the variations in neuronal responses in a group of heterogeneous neurons are many folds more than those found in a group of homogeneous neurons, which suggests that the variations of this diversity are not just coincidental.

The concluding finding of the work [101] was that neural diversity is essential for the efficient and effective functionality of biological systems such as the brain. Additional research by Thivierge [99] suggested that diversity plays complex roles in aspects concerning our representation and interpretation of the world. He further proposed that the diversity of neurons in a network enables a wider variety of responses with less effort.

In summary, neural diversity is another prevalent property of biological neural networks, whose constituent neurons have shown to vary in their responses even when they belong to the same group. The variance allows for a broader range of output responses from the network than if they were made up of a homogeneous group. In addition, it also enhances efficiency in terms of the relatively fewer number of neurons required, and to improve the effectiveness of neural networks in terms of the representational capacity associated with it.

2.1.3 Multifunctional Neurons

Apart from the diversity between neurons even of the same anatomy, which was highlighted in the previous section, neurons also foster more representational capacity by their ability to be *polymorphic*.

The early belief about the nervous system was that it stimulates a pool of interneurons, known as *command neurons*, which were responsible for the intended behavior. Eventually, it was found that interneurons - a class of neurons usually linking neurons to motor neurons, glands, or organs - can exhibit a variety of responses depending on many factors; including synaptic input, modulation state, and plasticity. These neurons are classed as multifunctional neurons [18]. Multifunctional neurons can also, therefore, be regarded as one of the facets of neural diversity. This is because they are not restricted in their behavior, which would have made them strictly homogenous; but they exhibit a variety of behaviors depending on the incoming signal and their individual properties.

There have been efforts to understand this property of neurons better; for example, work by Marder et. al. [69] has shown that interneurons that activated for some motor behaviors such as respiratory gill movements in invertebrates are also activated for other actions such as spontaneous gill movements and sensory induced gill movements. The findings showed that these interneurons are being shared to exhibit various behaviors. Further study by Briggman [18] which focused on the multifunctional nature of neural circuits summarized and inferred from earlier research, the factors that cause these several functional switches.

These factors were reported to include: inputs from sensory or projection neurons, the influence of neuromodulatory substances, and movement constraints on the organism's body.

Two distinct projection neurons responsible for distinct behaviors (i.e. mill rhythms) found in lobsters that were activated in parallel resulted in a switch in the configuration of the network, enabling two distinct rhythms. Similarly sensory neurons, specifically mechanosensory neurons, when activated simultaneously can cause a reconfiguration that can result in different rhythm behaviors [18].

In addition to synaptic inputs, another influencing factor that can cause a neuron to switch between behaviors is neuromodulator type. Neuromodulators are chemical substances that are released either to alter the properties of neurons, thus changes the way they react to stimuli, or to alter the synaptic strength of a population of neurons. The release of the neuromodulator could be targeted at specific cells, thus being released locally by projection neurons or released in a more hormonal pattern to affect the whole population of neurons. When released, these substances can alter the movement in organisms, switch between movements or even create new movements [18].

In summary, multifunctional neurons could be considered a neural feature that enables a smaller population of neurons to carry out tasks that otherwise would have required a lot more neurons. In other words, it can be regarded as an efficiency-enhancing property.

2.2 Artificial Neural Network(ANN)

Artificial Neural Networks incorporate a variety of the properties of biological neural networks associated with learning and generalization. Typical examples of this include weight adjustments to simulate the strengthening and weakening of connections, and transfer functions to model the firing response of neurons based on incoming stimuli. Additional examples include weight decay to replicate the pruning of relatively unused connections over time, which has been used as a regularization method. More recent examples of biologically inspired learning include the sleep-wake training routine modeled after the process of memory refinement that occurs during sleeping in the human brain [51, 23], and the property of depth (i.e. several interconnected layers of units) found in the brain, which inspired deep neural networks [14].

In general, the architecture of artificial neural networks consists of neurons, topology, and weights. The neurons are typically classified by their role as input neurons, hidden neurons or output neurons. In analogy to the biological neural network, input neurons could be regarded as the sensory neurons, hidden neurons as the inter-neurons which in artificial

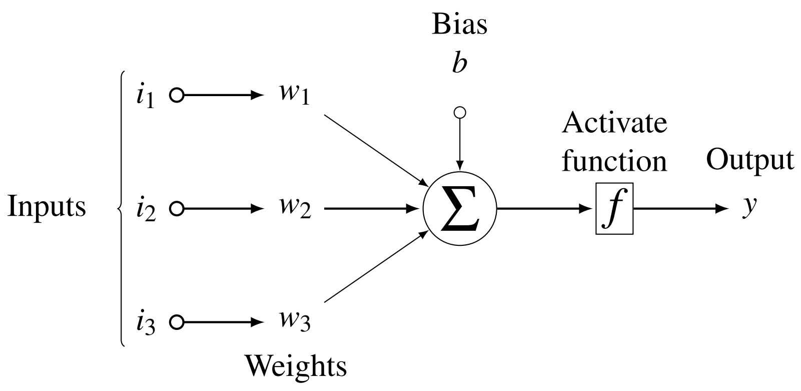


Figure 2.2: Components of a typical neuron.

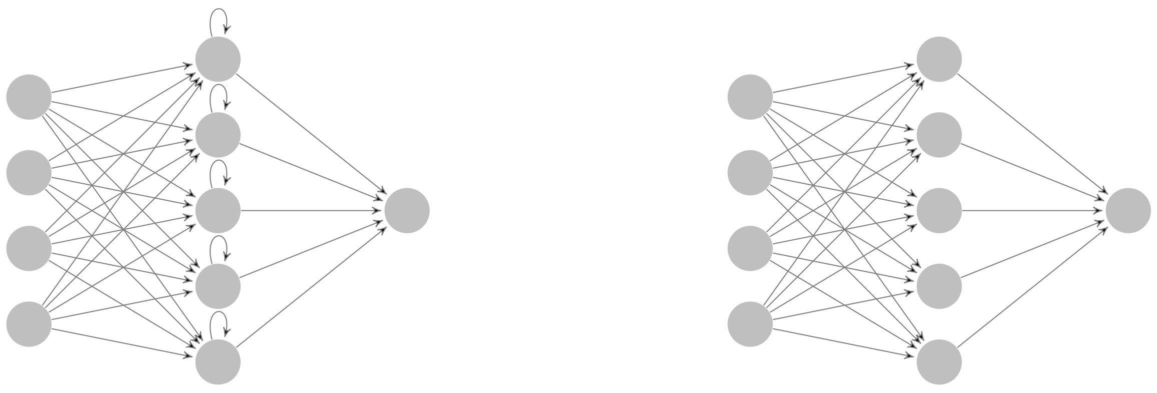
neural networks do most of the computations, and finally, the output neurons - which are the equivalent of either a motor neuron or the terminal neuron for the sub-network.

In artificial neural networks, the input neurons are typically not involved in any computation of the incoming signals. However, there are exceptions such as in relational neurons, where input neurons have been used to produce additional sets of features by performing some computation on the original set of features [92, 40]. The hidden and output neurons, on the other hand, are typically involved in most of the computation. This is made possible by the transfer functions assigned to each of the neurons in the hidden and output layer, as well as by their topology and weights. The transfer functions in conjunction with the topology to a large extent define the architecture of the neural network.

The transfer function is made up of an input combination function, also commonly referred to as *activation function* or *weight function*, which essentially computes an activation value from incoming signals ($x = \{i_1, i_2, i_3\}$) and the weights of their connections($W = \{w_1, w_2, w_3\}$) (see Fig. 2.2). There are a variety of activation functions, some of the most popular ones include distanced based functions such as Euclidean distance, Manhattan distance, maximum distance, and dot product based functions such as the inner-product [35, 36]. Table 2.1 showcases some of the popular activation functions. In conjunction with the activation function, there is an output function which computes the output signal of the neuron from the already computed activation value. There is a variety of these as well (see Table 2.1), from projection-basis functions, which include the sigmoid, and the hyperbolic tangent, and from radial basis functions, which include the Gaussian function, and the multi-quadratic Gaussian [35, 36]. The activation function and the output function make up the transfer function for a neuron and determine the type of transformation applied to signals from other neurons. The component responsible for routing the signals produced is the topology, which defines the connectivity between the neurons.

Table 2.1: List of activation (input combination) and output functions found in the literature [36].

Activation functions	Output functions
Inner-Product	Linear
Euclidean Distance	Step
Maximum Distance	Hyperbolic Tangent
Manhattan Distance	Sigmoid
Minkovski Distance	Gaussian
	Multiquadratic
	Bi-radial [38]


(a) A recurrent neural network topology.

(b) A feed-forward neural network topology.

Figure 2.3: Common neural network topologies.

The most common topologies of artificial neural networks are feed-forward, or recurrent. A feed-forward topology connects all the neurons from the previous layer to all the neurons in the next layer, thus feeding forward the signals through the neural network (see Fig. 2.3b). A canonical representation of recurrent topology, on the other hand, has self-connections, typically used for the hidden units (see Fig. 2.3a). The self-connection enables the neural network to have a temporal dimension/depth, which gives the neural network memory that is useful for pattern recognition where having some context of other trends is necessary. However, topologies are not strictly confined to these type of connections, especially in artificial neural networks that evolve their topologies [42, 97, 98, 6, 71, 5, 44, 46].

Learning in artificial neural networks involves optimizing each of its components, his minimally includes adjusting the topology, weights and functional parameters of the transfer functions. Relative to other components of the artificial neural networks, the least researched direction is transfer function optimization. Typically, in the case of weight optimization, the weights of the connections are adjusted using learning algorithms that iteratively adjust the weights. Examples of these include gradient-based approaches such as Least Mean Squared (*LMS*), Backpropagation (*BP*) or Evolutionary Algorithm (*EA*) based approaches [22, 112, 16, 12, 34, 109, 41, 25, 67]. In both cases, an objective function such as a Mean Squared Error (MSE) approximates the generalization performance on the dataset. The objective function also forms an error surface which needs to be minimized. In the case of gradient-based approaches such as back-propagation, weight adjustments are made by computing the gradient of the error surface in relation to the weights, and adjustments are then made accordingly. In the case of evolutionary algorithms, the cost of the objective function is typically assigned as the fitness of the weights, and evolutionary operators are applied to genetic representations of the more promising weights candidates to produce variations. Both algorithms are essentially traversing the search space for the set of weights with the least cost. In general, evolutionary algorithms - before the advent of deep learning - have tended to be more robust due to their ability to escape local minima.

Topologies are also among the principal components optimized during learning. This involves making or pruning connections. However, a fixed topology such as a feed-forward architecture can be maintained, which makes it easier to train and significantly reduces the complexity of the search space. Furthermore, it is also easier to study the underlying computational strategy learned.

Transfer functions, on the other hand, are optimized by adjusting parameters such as the bias weight, and output function parameters. In later chapters, we shall explore in more detail some of the approaches of transfer function optimization found in the literature.

During training, a dataset is used to train the neural network on possible inputs; and depending on the method of training; the desired output might be given to the neural network. In which case, it becomes a *supervised learning* model. Alternatively, only the feedback of the network's output is given, in which case it becomes an *unsupervised learning* model.

Regardless of the training method, each pattern from a dataset is used to stimulate the input neurons of the neural network. This pattern is then propagated through the network through the connections between the neurons to the output neurons. In the case of a neural network having more than one output neuron - such as in a multi-class classification; the output of the neuron with the largest output value is typically selected, in what is known as a *winner-take-all* method. Errors of each output neuron for all the patterns are then calculated afterward using objective functions such as the Sum of Squared Error (SSE) difference or Mean Squared Error (MSE).

In summary, there are many design parameters to consider, keeping in mind the intended application of the Artificial Neural Network. These include: the number of neurons in each layer, and if multiple layers would be needed. Connections are usually directly proportional to the number of neurons used in a neural network; so a large network is likely to have a greater number of connections, and consequently, a large search space -

if not constrained. The number of nodes and connections in a network essentially defines its complexity. For optimal generalization ability of a neural network to be achieved, the complexity of the neural network has to be matched with the complexity of the problem [17]. This tends to be a delicate task which requires balancing between underfitting and overfitting the problem.

Another crucial choice in the design of neural networks is the choice of transfer functions for each layer, and if there are going to be layers that have hybrid neurons (i.e. different combinations of transfer functions). As several studies have found [71, 6, 55, 49], some choices of activation functions work more effectively than others for a subset of problems: there is no one-size-fits-all choice for all problems. For example, linear output functions work well as output nodes for regression problems, but might not be as suitable for probability estimation as a sigmoidal node function. Therefore, for every problem an appropriate configuration of transfer functions for the problem is required. Although there has been research towards transfer functions that adapt to the nature of problems [78, 98, 116, 115], most of the popular transfer functions have usually been used as informal standards. This is perhaps because of their use in various architectures such as the Multilayer Perceptron and Radial Basis Function Network that are proven universal approximators [17]. Regardless, the no free lunch theorem applies to all the components of the artificial neural network.

2.2.1 The Perceptron

The perceptron can be regarded as a simplified model of a single biological neuron, and is the simplest form of an Artificial Neural Network. It is characterized by both input (x) and output pathways (y). The input pathways act like dendrites, receiving features from a given pattern, which is then operated on by the activation function. In the case of the perceptron, this is an inner-product of the weights and input pattern vectors. Thus, the weight vector W is given by $\{w_1, w_2 \ldots w_k, w_{bias}\}$, and the input pattern x is represented as $\{i_1, i_2 \ldots i_k, i_{bias}\}$. Each input pattern can be thought of as a *feature vector* which holds various measurements of a given state (such as car instrument readings, medical readings, shuttle instrument readings), objects (facial features measurements), or any other form of data that can be represented as a pattern recognition problem.

There is a *bias* as illustrated above in the perceptron, which is another parameter that gives the learning algorithm more control of how it projects the decision boundary [17, 58]. A neuron in a neural network is usually accompanied by a bias value, i_{bias}, which is usually set to one, and its weight, w_{bias}, is optimized as a parameter by the learning algorithm.

$$j = \sum_{i}^{k} w_i i_i + w_{bias} i_{bias} \tag{2.1}$$

The inner product activation function, given by Eq. 2.1, of the weight vector and input pattern results in the action value. It is then applied to the output function that determines the networks output. This can come in various forms; from step functions which produce binary outputs, to sigmoids which can be regarded as smoothed step-functions that produce a continuous output over a range of values.

The perceptron can learn with great accuracy various linearly separable problems. However for problems such as the XOR that are classified as non-linearly separable problems; it not capable of learning, and performs poorly. In the following section, the reason behind this is explained.

2.2.2 Decision Boundaries and Linear Separability

The perceptron is applicable to classification problems which require that given a set of input patterns $x_i = \{i_1, ...i_K\}$ be classified into their respective categories (or classes) $y_i \in c_1, ..c_N$. Such problems might involve more than two classes and can be categorized as either *linearly separable* and *non-linearly separable* problems. Basically, this to a large extent defines the difficulty of the problem.

Classification problems of N classes are essentially a class of problems that involves forming *decision boundaries* projected by transfer functions onto the K dimensions of the input space (each dimension representing a feature of the input pattern x_i), which generalize the membership of patterns based on the training dataset $(x_i, y_i) \in D$. Depending on the transfer functions, these boundaries will differ in shape and form. Thus, the goal of training is to fit the decision boundaries to the training dataset while maintaining the delicate balance between underfitting and overfitting. In the next chapter, this is explained more in the context of the bias-variance decomposition.

In the case of linearly separable problems, the members of each class can be dissected into their respective classes linearly. However, in most cases the hyperspace of classification problems is non-linear and cannot be separated with a linear hyperplane alone. This is essentially the core argument of Minsky's paper that discredited the perceptron as highlighted in [17]. Undoubtedly, this is the nature of real world problems which are complex. The hyperspace of such problems would require increased network complexity to be better able to separate members of each class appropriately [17]; specifically, regarding the network architecture such as number of nodes, the number of hidden layers, the number of nodes per layer, and transfer functions.

In the next few sections, we present neural networks that have been proven to be able to approximate any arbitrary function - provided they are allowed to match their complexity with the complexity of the problem. These are Multilayer Perceptron (MLPs), and Radial Basis Function Network (RBFNs). We also briefly highlight the topic of generalization and model complexity.

2.2.3 Multilayer Perceptron (MLP)

The Multilayer Perceptron is an example of a neural network architecture that is theoretically proven to be able to approximate any arbitrary decision boundaries; thus, it is described as *universal approximator*. A universal approximator can approximate any arbitrary function. However, this is provided it can adopt sufficient complexity - in the form of more hidden units and layers [17]- that matches the problems complexity. However in practical terms, network size is limited by the computational capacity. Also, large models are computationally expensive to train.

The Multilayer Perceptron differs from the Perceptron as it has multiple layers of neurons. Training the connection weights of the layers is traditionally done using the Back-propagation (BP) algorithm: a gradient descent algorithm which is typically characterized with some drawbacks such as its vulnerability to getting stuck in local minima. The basic framework of the Multilayer Perceptron is that a network uses linear or sigmoidal output functions, and should consist of at-least one hidden layer. It is an architecture that revolves around having a hierarchy of processing units (neurons) in the hidden layer and adapting the number of layers in this hierarchy, and the number of nodes contained in each according to the problem.

Each node in the hidden layer of the network projects a decision boundary, as does a perceptron in the hyperspace. The form of the decision boundary projected depends on the activation function adopted by the neuron; taking the form of a linear hyperplane, if it's a linear activation function, and the form of a continuous ramp, if it is a sigmoid. The hierarchy of layers in the Multilayer Perceptron enables it to combine these decision boundaries into more complex non-linear decision boundaries.

The Multilayer Perceptron is a model that has had various extensions; one example of such an extension is the proposition of using a multiplicative rather than a summative combination of inputs, which increases the representational capacity of the neural network. Such a unit will perform input signal combination as a product operation, instead of a summation (see equation 2.2).

$$j = \prod_{i}^{k} i_i^{w_i} \tag{2.2}$$

Given that the weights of input connections to this node are binary, this unit is referred to as a *higher order* processing unit or *sigma-pi* unit. However, in cases where the weights are not fixed and can be real values, it becomes a *product unit(PU)* activation function. The advantage of using higher order units such as the product unit is increased information capacity [17]. Thus, fewer units would be required as compared to when a summation unit is used resulting in smaller network architectures. However, the use of higher order units causes a disturbance in the weight space [40]. Specifically, it is characterized with more local minima. This is a consequence associated with increased computational capacity which presents various alternatives of sub-optimal computational strategies.

MLPs have been used in a wide variety of problems [72, 116, 92, 22, 59, 76], some of which include segmentation problems, an example being an implementation by Magnotta et al. (1999), where image intensities of neighboring voxels were used as image features. This was further developed by Powell et al. (2008) for the segmentation of brain structures such as putamen, caudate, thalamus, and cerebellar regions of interest as highlighted in [76]. While an extension of it such as the higher order neural network has been applied in areas like invariant pattern recognition by structure [9], where it was modified to improve computational time.

2.2.4 Radial Basis Functions Networks (RBFN)

An RBF is another neural network architecture that has found several applications in real world problems. Like the MLP, it has also been proven by Hartman et al.(1990) to be a universal approximator, provided there is a sufficient number of hidden nodes [17]. It is a model that implements hidden nodes that use non-linear transformation functions on input signals. The transfer functions of the hidden units typically consist of a distance-based activation function, and a radial-basis output function.

RBFs are functions that are generally characterized by a center, which is the peak of the function. Points away from the base of this peak decrease rather rapidly, forming an almost bell-shaped curve. It is also characterized by a width, which determines how wide the bell spans. Examples of some of these types of functions include: (i) the Gaussian (ii) multi-quadratic, and (iii) inverse multi-quadratic functions.

The Radial Basis Function Network is a three-layer network [17, 62]. The first layer of connection weights between the inputs and the hidden units, act as a vector that represents the centers of the radial basis units. For each of the hidden units, the distance between each feature value $i_i \in x$ and its corresponding weight $w_i \in W$ is calculated, usually with a distance-based function such as the Euclidean distance given by equation (2.3).

$$j = \sqrt{\sum_{i}^{k} (w_i - i_i)^2} \tag{2.3}$$

where,

- W - is the weight vector of incoming connections.
- x - is the input vector.

$$g(x) = e^{-(j-\mu)^2 \big/ 2\sigma^2} \tag{2.4}$$

Afterward, the differences are summed to an activation value which represents the distance between the feature vector (pattern) and the center of the radial basis unit. This is then transformed by a radial basis function - such as a Gaussian function, $g(.)$ (see equation 2.4). Thus, the resulting output value for each node in the hidden layer of a radial basis function is given by equation (2.5).

$$z = g(j) \tag{2.5}$$

The variance (σ) of the function specifies the region covered by the function in the input space while the mean (μ) determines the peak of the Gaussian function. The centers of each of the basis functions have to be evenly spread across the input space for more optimal performance. In addition to that, there has to be a sufficient number of basis functions to cover the input space [40, 17]. Thus, the more the number of basis functions; the better the approximation of the problem's underlying mapping function. However, it is a balancing act; if the basis functions are more than necessary, it can cause an increase in the computation complexity, and risk overfitting the dataset. Radial Basis function Networks have been used in many applications [113, 114, 115, 56].

2.2.5 Generalization and Model Complexity

The Artificial Neural Networks are noted for robustness in their learning ability, and ability to generalize. However, designing an ANN requires choosing parameters such as the transfer functions, topology of the neural network (i.e. feedforward or recurrent), and the number of layers and nodes, which is a delicate task. This is because these define the computational capacity of the neural networks, which subsequently directly affect its ability to generalize. If it is designed with relatively less complexity than the problem, then it may underfit the problem, and if it is relatively more complex than the problem it may lose its ability to generalize to unseen datasets. Considering also that there is yet to be a defined method for designing neural networks, and knowing the role of that their architecture plays in its successful application for a given task; it makes the task of finding approaches to optimizing neural network architectures essential.

It is known that matching the complexity of neural network's architecture to the problem's complexity is one way optimal generalization ability can be achieved by a neural network [17], and various methods have been suggested to achieve this. These were summarized to be classifiable into three categories according to Jankowski [55], and include (i) regularization (ii) ontogenic grow/shrink network, and (iii) choice of appropriate transfer functions (i.e. transfer function optimization). However, in this section, only the first two methods mentioned will be explained, while the approach dealing with the choice of transfer functions is left to be discussed in more detail in later chapters.

In regularization methods, a penalisation scheme is applied to the objective function, which typically factors in the neural network's size and penalizes accordingly. There are various forms of penalization that have been used in studies [55]. One approach is weight decay, which diminishes the weights of connections towards zero. The effect of this is that weaker connections, which are assumed to be of less importance, are pruned in the course of training. However, this does not consider that stronger connection weights are also being equally penalised as weaker connections, making it counterproductive as explained by Engelbrecht [40]. However, a variant by Hansen - highlighted in [40] - used functions such as hyperbolic and exponential functions to determine the amount of penalization to be made for each weight; weaker connection weights are penalized more than relatively stronger connection weights. Some other approaches similar to regularization include penalizing based on the number of weights and a regulatory constant, proposed by Weightend et al, minimisation of networks energy, measured as the sum of each hidden unit's activation squared as proposed by Chauvin, and regulating the sharing of hidden units between output nodes studied by Yasui - as highlighted in [40].

The second approach to complexity control is the use of grow/prune methods in an attempt to find the appropriate network size for a given problem. In constructive or growing methods, operations that add nodes to the network are included in the training process, and usually, rules are implemented to govern this growth. The network starts off with a small network architecture and grows the network if the networks architectural complexity is still not able to learn the problem with a desired estimated generalization error. Engelbrecht [40] explains that the rule that governs when to increase the network size and when to stop is crucial for obtaining optimal network architectures. This is due to the rule directly interacting with the network's complexity and in consequence, its generalization ability. Its counterpart are the pruning methods that remove nodes and connections to shrink the size of the network from a predetermined size. The decision of which node or connection to prune is usually based on a rule that relies on measurements of how useful the node or connection is to the network performance. It could also involve statistics based methods such as saliency tests. This approach towards complexity regularization could be expected to have faster convergence. Also, though Engelbrecht argues that this approach is guaranteed to learn the underlying mapping function for the data; this is only under the precondition that the network's initial size is sufficiently complex. If the initial complexity of the network is relatively less than that of the problem; it might result in the training process not being able to learn the underlying function of the problem. Bishop's analogy on polynomial curve fitting describes the delicate nature of finding the balance between overfitting and underfitting problems [17].

Various approaches for pruning in artificial neural networks have also been studied, and can be classified into: pruning by evolutionary algorithms [12, 34, 98, 71], and pruning by statistical test [32]. Intuitive pruning methods assume that nodes that are frequently activated and have larger connection weights are more important, and nodes that lack this are less important. However, this assumption is not without its flaws; weights that are weak are important between hidden and output nodes [40]. Evolutionary pruning methods allow pruning of nodes to be done by the evolutionary algorithm; it can be by adding operations in the form of mutation operators [12, 34, 98], or encoding binary values representing the status of the connections onto the gene string to be switched off or on [71]. However, neural networks are sensitive to a node or weight removal. This is the motivation for the use of sensitivity analysis by other approaches [32].

In summary, model complexity plays a significant role in determining the generalization ability of neural networks. This relationship is further explained in a later chapter in relation to the bias-variance decomposition. In the following sections, we highlight some evolutionary algorithms and hybrids of artificial neural networks which evolve the architectural components of artificial neural networks, and as a result, control their complexity.

2.3 Evolutionary Computation

Evolutionary Algorithms (EAs) are a set of algorithms that model problems as an evolutionary process and implement the characteristic events such as mutation, recombination, and population seeding. Evolutionary Algorithms (EAs) are ideal for exploring complex search spaces.

In this section, we highlight only some evolutionary algorithms briefly, as it is not the primary focus of the research. Firstly, the Genetic Algorithm (GA) which is one of the most widely used in optimization problems, with so many variations that extend from it [39, 74]. Afterward, the working principles of another evolutionary algorithm, Differential Evolution (DE) is explained.

2.3.1 Genetic Algorithms (GA)

The Genetic Algorithm (GA) is a widely popular population-based algorithm that is biologically plausible. The concept of this algorithm is basically to form a representation of the problems parameters that are then encoded onto an array, commonly referred to as the *chromosome*, and adopt evolutionary operators such as mutation and cross-over to find an optimal solution. There are various representation schemes possible, though these can all be classified as binary or real value representations [107, 109]. In binary representational schemes, these chromosomes will represent the parameters of the problem in binary, while in real values representations, real values are used to represent parameters. Depending on the number of parameters and encoding approach, the chromosomes will vary in their length. However, canonical genetic algorithms use binary representations of a problem, so the parameters have to adopt an approach that encodes parameters onto a chromosome as binary representations. Such representations typically have to compensate with relatively complicated encoding methods so that the length of the chromosome is not too long -as this complicates the search space, and not too short that it causes a loss of weight precision, which will significantly affect the accuracy of approximating the mapping function (i.e underlying function of the problem) [112, 109, 108, 111].

However, in the case of real value representations [98, 4, 11], the precision of weights can be modified without altering the length of the chromosome. Standard evolutionary operators have to be modified to deal with these changes.

Once the problem has been represented in the form of a chromosome, a finite population of m chromosomes with different parameter variations is generated randomly and encoded, $P = \{c_1, c_2, c_3 \ldots c_m\}$. However, to ensure a better performance advantage, prior knowledge of the context of an application can be used to optimize properties of the initialization process such as the distribution of the random number generated, and range. This significantly reduces the search space complexity, thereby increasing the convergence speed. The next step in the process is to evaluate the solutions using an evaluation function, $f(.)$. This function is another crucial element of the evolutionary process which can affect its performance in a number of ways. One of such ways includes its ability to find optimal solutions. There is no general evaluation function for all problems, and one would ideally want to carefully craft a problem specific evaluation function, which takes into account the nature of the domain [107, 108, 112, 109]. All chromosomes or candidate solutions are scored using the evaluation function. This scoring process is the biological equivalent of the process that determines the fitness of individuals in the population. The next process in line is the selection process. It relates to the survival of the fittest trait found in biology and has two steps; the first step, *selection*, selects candidates that will make up the next generation of candidates solutions based on fitness. The other step, *reproduction*, creates new individuals that will replace weaker candidates using evolutionary operators (i.e. *mutation* and *cross-over*). The mutation operation alters fitter individuals selected typically at random to alter their chromosomes while the cross-over operation forms new candidate solutions by exchanging chunks of genes from the gene strings (chromosomes) of two parent individuals. The choice of which operator to apply can be selected on a probabilistic basis.

These processes are repeatedly executed until a stopping criterion is met, which could be after a maximum number of generations (i.e. iterations), or/and if a solution in the population has reached the desired fitness level.

2.3.2 Differential Evolution (DE)

Differential Evolution is also a population-based evolutionary algorithm developed by Storn and Price (1996) as highlighted in [40]. Although, its efficiency at converging on near optimal solutions has yet to be formally proven as explained by Jones [58]; it has been found to be efficient and effective for a wide variety of problems. It can be considered an extension of the Genetic Algorithm. The only significant difference between differential evolution and other classical evolutionary algorithms is its approach towards mutation and recombination of individuals in the population. It also introduces two new parameters: (1) a weighting factor and, (2) a probability of cross-over.

In differential evolution, the mutation operation starts off by randomly selecting three solution vectors from the population. Assuming these solution vectors are labeled as V_x, V_y and V_z; then the next step is to find the difference between the first two vectors, i.e. V_x and V_y, which is multiplied by the weighting factor, F, and then added to the solution vector of the third solution, i.e. V_z, resulting in the mutant solution's vector, V_n (refer to equation 2.6). This not only helps explore more regions of the search space as a result of its stochastic property but also results in a property found in particle swarm optimisation - another population-based algorithm that accounts for the effects of other members of the population on each individual vector - known as the flocking behavior.

$$V_n = V_z + F\left(V_x - V_y\right) \tag{2.6}$$

Extensions of this algorithm such the one used to evolve Neural Diversity Machines [71] implements a recombination stage which follows afterward, where the mutant solution and the third solution, V_z, create a new vector (or solution) by competing on a probabilistic basis (i.e. probability of cross-over) to contribute to the new solution V_n. In other words, if a random number generated is less than the probability of cross-over; a gene from the mutant vector is copied. However, If it is greater, then the gene from the third solution is copied instead. This vector is then evaluated and is only copied into the next generation if it has better fitness than the third solution. This behavior helps to relief pressure that might otherwise lead to premature convergence.

2.4 Hybrid Artificial Neural Networks

In this section, we highlight some of the works on evolutionary artificial neural networks. However, chapter-specific literature reviews provide more in depth reviews of some of the literature, and the contributions to the literature.

2.4.1 Evolutionary Artificial Neural Networks

A relatively new direction for designing the architecture of Artificial Neural Networks using Evolutionary Algorithms (EA) has gained some attention from its potential in finding near-optimal architectural designs for ANN [110, 108, 42, 71, 78, 46, 44, 39]. The idea is to use evolutionary algorithms in the task of optimizing design and training parameters of ANN such as weights, transfer functions, and architecture (topology), among other components. In general, the significant developments in the field of evolutionary ANN have been on a number of aspects of neural networks [107, 108, 110], including - but not limited to: (1) weight optimization (2) transfer function optimization, and (3) architectural optimization. The task of optimizing these aspects was explored using evolutionary methods, such as Genetic Algorithm (GA), and Genetic Programming (GP).

2.4.1.1 Weight and Architecture Optimization using Evolutionary Algorithms

The classical approach to optimizing the weights of an ANN is to use gradient-descent based algorithms such as Backpropagation (BP). Backpropagation tackles the task of optimizing the weight by reducing the error associated with the weights relative to the networks error by computing their respective gradients, which is followed then by proportionate adjustments. It does this iteratively until there is an acceptable difference between the actual error and the desired error. However, due to its reliance on the gradient, the canonical Backpropagation was known for its vulnerability to being stuck at local minima [17]. Thus, somewhat limiting its potential. However, with Evolutionary Algorithms such as Genetic Algorithms (GA), its feasible to make near-optimal weight optimizations, and also have relatively less worry about local minima, due to its stochastic property. An example of the implementation of an Evolutionary Algorithm as highlighted in Azzini, A. & Tettamanzi's [11] survey paper was the use of Genetic Algorithms (GA) by binary encoding weights, GENITOR, by Whitley et al. [105], which ranks genetic strings according to their fitness values. A recombination routine did this with a bias towards selecting genetic strings with higher ranks as parents. The parents are then recombined, and the offspring is evaluated and replaces the weaker parent. Another related work is by Montana and Davis [77], where they used an array of real numbers, which were initialized randomly using a distribution function. Their results showed that using their evolutionary algorithm training process was much faster than the traditional BP (Backpropagation) for the problems they considered [11].

In terms of architecture optimization, one of the prominent works was by Stanley and Miikkulainen [97, 98] referred to as NEAT (NeuroEvolution of Augmenting Topologies). It augmented the topologies of an evolutionary neural network using structural mutation and crossover operators, and population seeding in the initial phase using uniform distribution. One of the unique contributions of this work was the "historic marking" of the neural networks, which influenced cross-over without computationally expensive topological analysis. In addition to serving as a more efficient means to crossing over, it also doubled as a diversity maintenance scheme. However, fitness sharing was also used as a more explicit diversity maintenance scheme. The resulting effect was a speciation effect which protected various computational strategies from being dominated. Another work related to NEAT which also modified topologies by cross-over was SimBa [34] and SimBa2 [12], which relied on topological similarity analysis for cross-over operations. In comparison to NEAT this approach has a computational overhead introduced by the analysis. Other relatively recent works discussed in more detail in the final contributory chapter of this thesis, use co-evolutionary algorithms to represent neural networks. These include COVNET [44], SANE [78], and CoSyNE [46] which have all shown promising results.

Chapter 3

The Notion of Transfer Functions Diversity

This chapter explores how neural diversity can improve generalization ability in the context of theoretical proof from the literature [17] that links generalization to bias and variance, i.e. the bias-variance decomposition and the more qualitative meta-learning theories [102, 103, 75]. It does this by presenting both definitions of bias from both the statistical and meta-learning perspectives in the literature [17, 102, 103, 75], which form the basis of the arguments that we use to relate generalization to transfer function diversity, and how it can improve it.

The following introductory sections will present the motivations behind transfer function optimization, related works regarding transfer function optimization, and the contributions of this chapter. This is then followed by a detailed chapter overview.

3.1 Motivation

The transfer function of an artificial neuron, h is composed of the activation function, $g_h(x,W)$ and output function, $f_h(g_h(x,W))$; the activation function computes the action potential, a_h from the input and weight values (i.e. $x = \{i_1...i_k\}$, and $W = \{w_1...w_k\}$, respectively) of the incoming connections. The output function computes the output of the neuron in response to the action potential. The transfer functions $y_h = f_h(g_h(x,W))$ are responsible for forming the decision boundaries in the input space among other tasks such as feature selection. The shape and form of these decision boundaries vary with the type of transfer function. Radial-basis functions, for example, have a cluster-like behavior of forming decision boundaries: points in the input space closer to the center of a radial basis function generate a higher response while points further away from the center tend towards progressively lower responses. Linear functions form a hyperplane in the input space which separates the points. Sigmoid functions are similar but differ because they form sinusoidal-like hyperplanes. Neural Networks learn classification problems by optimizing these decision boundaries to classify observations (i.e. points in the input space) into their respective classes. Training is typically done on a subset of the data set that should be representative of the whole sample. Ideally, the neural network should learn the underlying function of the problem and be able to predict unseen data with significant accuracy. This is the essence of *generalization*.

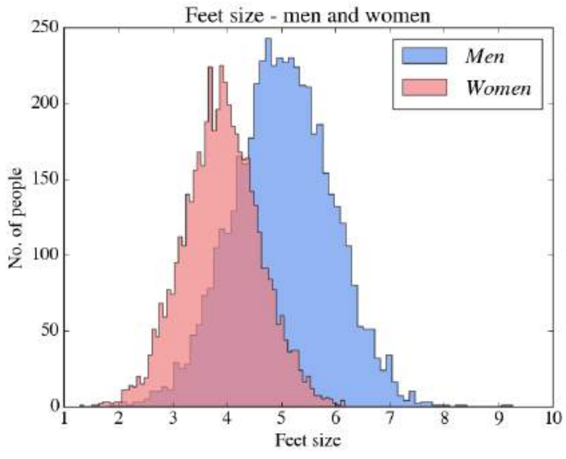
However, though transfer functions have a critical role in learning, there has been relatively little research in this direction as compared to other aspects - such as weight optimization methods, as explained by Duch [37]. A closely related aspect of neural networks - topological optimization - also has had relatively less research as well. By comparison, other aspects such as training methods and regularization methods have received relatively more attention as compared to transfer function optimization [49, 54, 55, 38, 49, 71, 6, 5].

Furthermore, there is also the heavy reliance on using projection basis functions such as inner-products with sigmoids, and radial basis functions such as Euclidean distance with Gaussian function as informal standards for transfer functions. These transfer functions are used as generic transfer functions for all problems. This is to a large extent due to the theoretical work with regards to their universal approximation capabilities [17]. However, this does necessarily guarantee that the models produced are efficient and effective. By *efficient*, we are referring to using a minimal number of hidden units to learn a problem, and by *effective* we are denoting significant generalization ability. Usually, the complexity of the neural network is increased to reduce the training error. However, some portion of the error could be as a result of the inherent bias of the chosen transfer function which might not be suitable for the problem.

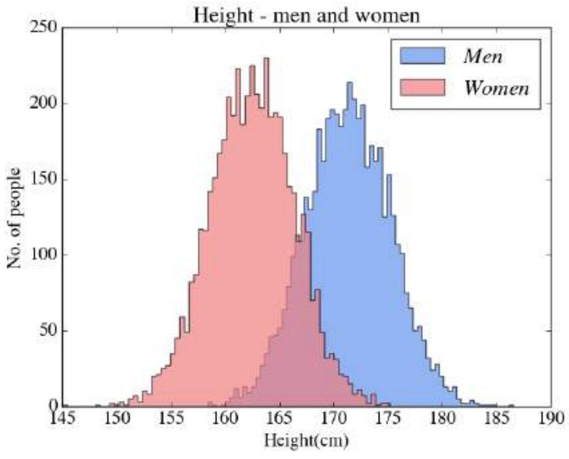
Like other components of a neural network such as its weight optimization algorithm and constraints; transfer functions have a bias usually based on assumptions [103, 102, 75, 3, 61]. In the case of radial basis units, such as Gaussian functions, it is assumed that the points representing instances of the classes in the input space have a normally distributed class membership, and as such can be modeled with adjustments to a Gaussian function. It is more biased to producing higher responses towards features close to its center, but this progressively decreases further away from the center. In the case of inner-product with sigmoid transfer functions, the assumption is that the points in the input space can be classified using discriminatory hyperplanes.

Moreover, the bias of the transfer functions can compound with other biases, such as the bias of constraints on optimization parameters or even regularization methods, to have a limiting effect on the neural network's ability to find a solution with the most optimal generalization ability [102, 103, 75]. Notably, constraints such as the maximum number of hidden units can compound with the bias of the transfer function to limit or even prohibit learning. This can have a significant effect especially in cases where the transfer function being used has an inappropriate bias for the given problem.

An unsuitable transfer function bias can affect learning accuracy, and generalization ability (approximated by the testing error). This can be illustrated with an example of a simple classification problem. Suppose the intent was to classify the gender of people based on their height and feet size, and we built up a balanced dataset. Upon visualizing the distributions of feet size according to gender, we should observe normal distributions for both men and women (as illustrated in Figure 3.1(a)). Likewise, we should also expect to see a normal distribution for the height of both men and women (as illustrated in Figure.3.1(b)).



(a)



(b)

Figure 3.1: Scatter plots of feet size and height for both genders.

If we are experimenting with linear functions, then the set of hypotheses available for such a learner, H_L will be solutions that are linear functions or made up of linear functions. This includes the hypothesis set $\{h_1, h_2, h_3\}$ as illustrated in Fig. 3.2. The hypothesis h_2 seems to have the best decision boundary among the three. We can expect this to result in better generalization ability as compared to the other hypothesis.

However, the bias of linear functions is clearly not suitable for this problem. The chances are that h_2 might have a generalization ability that gets worse as the testing set gets larger. This is because points in the overlap region which have been previously unknown are likely to increase, and the rigid boundary that divides the two classes does not account for this overlap. If the neural network's complexity is increased such that it can access more complex hypotheses from the global Hypothesis Space H_G, then it is likely to

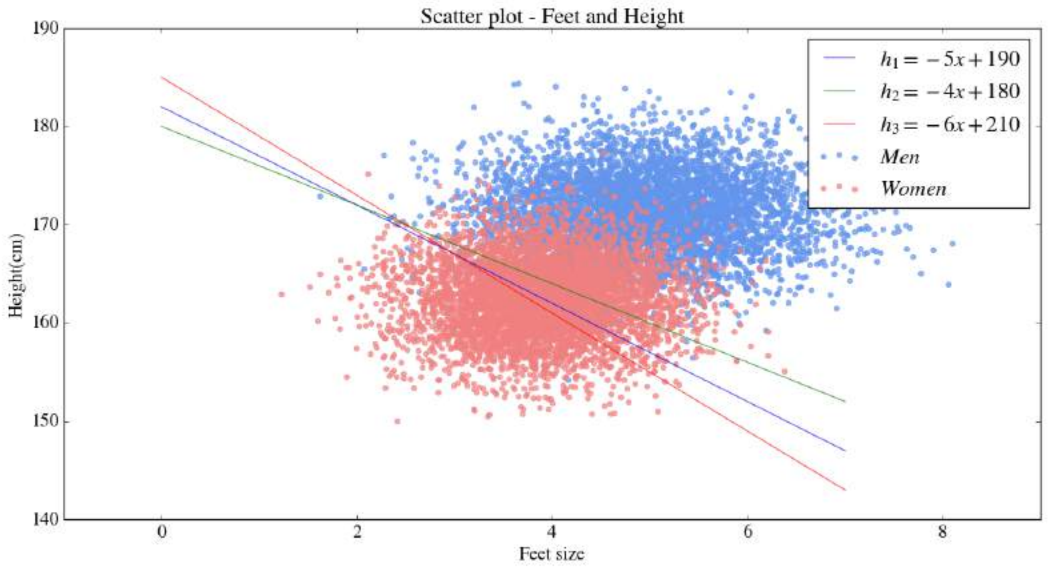


Figure 3.2: Some of the various possible hypotheses with linear functions.

find a decision boundary that reduces the training error. Though, this increases the risk of over-fitting. The bottom line is that the bias of the transfer function is clearly not suited to the problem.

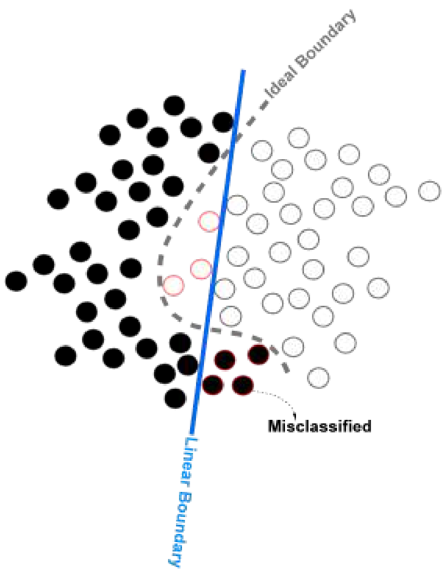


Figure 3.3: Linear decision boundary not accounting for overlap showing relatively more misclassified patterns as a result of the inherent bias (i.e. bias-error).

As illustrated in Fig. 3.3, although the classifier has separated a majority of the points from the two classes into their respective classes, there are still points that remain misclassified as a result of the bias of the decision boundary. In this case, the bias of the decision boundary is a consequence of the assumption that the classes should be linearly separable.

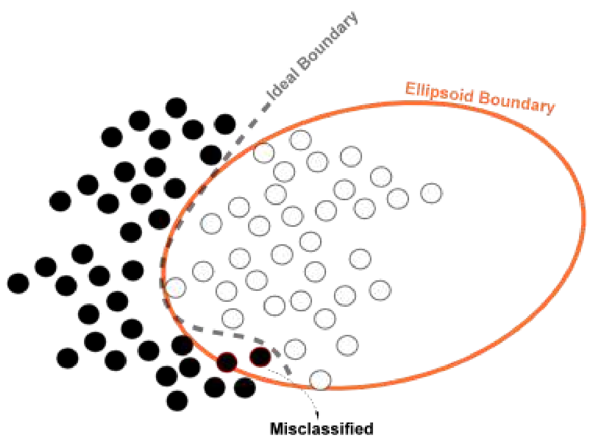


Figure 3.4: Ellipsoid decision boundary showing relatively fewer instances misclassified.

However, if we consider an ellipsoid decision boundary (Fig. 3.4) - such as one that can be exhibited by a radial basis function unit. We can see that it has relatively fewer points that have been misclassified, which might not purely be as a result of the bias of the transfer function, but as a consequence of the noise in the dataset. In this case, we have a more appropriate form of bias that assumes that the instances in the input-space produced by the classes can be modeled with a Gaussian function. Indeed, that was the case.

In a nutshell, transfer functions like any other neural network building block contributes to its generalization error with either some level of bias-error, such as the transfer functions not having the right sort of bias, or having very little of it as a result of its flexibility, resulting in over-fitting: in which case, it results in increased variance error. As such, in retrospect to the role of the transfer function in learning and generalization, there is a need for more approaches towards transfer function optimization oriented approach towards efficient learning.

Transfer function diversity is a nature inspired facets of transfer function optimization [71]. Transfer function optimization is important for many reasons: firstly, it the important role in forming decision boundaries necessary for learning. Secondly, another reason for researching ways of optimizing transfer function is to improve the efficiency of neural networks, since optimized decision boundaries are likely to result in more efficient and effective neural network models -given appropriate training [37].

Thirdly, transfer function optimization can lead to more robustness in learning. This all boils down to the no free lunch theorem; no algorithm is universally the most optimal for all possible problems. However, for a small subset of problems, there could be an algorithm that is more robust. For a learning algorithm to be robust, it will have to be able to adapt its *bias* to enable the search for the computational strategy that best describes the underlying function. In the case of artificial neural networks, the weights, topology and transfer functions among other components will need to be optimized according to the problem to enable the neural networks models to be applied to a variety of problems. The increased computational capacity as a result of transfer function optimization and appropriate training leads to a robust learner. In this work, we specifically emphasize on the study of transfer functions and topology because of their critical role in learning.

Finally, the field of artificial neural networks has been driven by a variety of neuroscience findings. Artificial neural networks are nature inspired after all, so it is only natural that discoveries from neuroscience continue to guide research of artificial neural networks. There is a lot of diversity in biological neurons, in the retina alone, there are more than 50 different types of neurons. Marder [69, 70] also found that even in neurons that were anatomically identical, there were some behavioral differences between them when they were subjected to the same stimuli. Briggman [18] and Marder [69, 70] have found that diversity of biological neural networks is responsible for the efficiency of biological neural networks, making them more compact. In addition to that, they also stated that this diversity enables biological neural networks to exhibit a wider range of responses without the need for more neurons. In other words, neural networks are likely to have more computational and representational capacity.

This work essentially introduces neural diversity in the form of transfer function diversity in artificial neural networks, as a facet of transfer function optimization for efficient learning. This has shown promising results in preliminary work done by Maul [71], which showed similar properties associated with neural diversity in biological neural networks. Specifically, the neural network models produced showed efficiency and promising generalization ability.

3.2 Related Works

There are various works that have been done in relation to transfer function optimization. These works can be classified by their main argument as : (i) meta-learning [3, 61], (ii) theory of duality of functions [47, 55, 30, 31], (iii) nature inspired computing [71], and (iv) flexibility enhancements of transfer functions [28, 41].

From the meta-learning perspective, it is believed that since there is no one-size-fits-all for the choice of artificial neural networks, then it makes sense to make as many of the components adaptable. This also involves the use of meta-rules or a meta-learner, to determine which architecture should be used. Often meta-features are used to measure the similarity between the problem at hand, and the knowledge-base of models and their performance on problems. The model architecture that works best for the most similar problem encountered before is then used. Usually, this knowledge of the performance of architectures, and the features of the problems are accumulated as the learning algorithm encounters more problems [3, 61, 103]. In the context of transfer function optimization, works done in this respect include [3, 61], where transfer functions are selected from a pool of transfer functions made up radial-basis and projection-basis transfer functions.

The second class of works [47, 54, 55, 38, 49] is based on the theory of duality of functions by Donoho (Donoho, 1992). The theory proves that any continuous function can be decomposed into two functions: a projection basis function and a radial basis function. However, there is yet to be any standard methods for approximating the underlying function of problems with the component functions as Gutiérrez highlighted [47]. Works done in this respect also choose the transfer functions of the target neurons from a pool of transfer functions from both classes of these component functions, i.e. radial basis and projection-basis functions. In essence, these models are related to the works from the meta-learning perspective [3, 61] regarding their choice of transfer functions.

Biological neural networks inspire the third class of works [71] on transfer function optimization. One of such inspirations is neural diversity. Neural diversity is one property of biological neural networks which is believed to be one of the reasons why they are so efficient [18, 101]. Works done in this respect are quite a few, among which is that of Maul [71], where a pool with a variety of transfer functions is used to optimize the choice of transfer functions for each hidden and output layer neurons. Unlike works from the other classes from the literature, these include higher-order, and statistical transfer functions - which bears resemblance to transfer functions being used for deep learning. These were found to evolve feature filters frequently.

The final class of works [38] is to have universal transfer functions that are capable of morphing into different forms (i.e. polymorphic/flexible) according to the nature of the problem. Examples of this include the bi-radial transfer function [38] which can exhibit a wider range of shapes and forms. It can even take the form of a Gaussian transfer function. Other works, in this regard, include the q-Gaussian transfer function [41], and a self-adaptive sigmoid function [95]. Both introduce parameters that give the learning algorithm additional representational capacity.

It is worth mentioning that most of the works from these diverse perspectives obtained similar findings. Especially in the case of transfer function pooling [47, 71, 5, 6, 61, 3] and polymorphic/flexible transfer functions [55, 28, 41, 38, 54], where they found that the models were relatively less complex, yet had better generalization ability. Interestingly, the polymorphism and diversity in the biological neuron are believed to be responsible for the same property of efficiency and representational capacity in biological neural networks [18, 101].

3.3 Contributions

There is still need for extensive research particularly in the area of transfer functions optimization. Related works [47, 71, 61, 3, 55, 38, 28, 41], have already shown the exceptional properties of generalization and relatively lower complexity that emerges in models of artificial neural networks that have their transfer functions optimized one way or another.

The gap in the literature is that there is hardly any studies that draw from findings of neuroscience for the purpose of transfer function optimization, even though it has been one of the primary sources of inspiration for the field since it's inception. Furthermore, there has been relatively little research on studying the underlying strategies that result from these hybrid artificial neural networks to uncover insights as to how they achieve significant generalization ability with significantly less complexity in terms of hidden layer size.

This chapter's main contribution is that it explicitly shows that neural diversity in the form of transfer function diversity can improve generalization performance in the context of literature [17, 103] that links generalization ability to bias and variance. It does this by using works from the literature [17, 102, 103, 75] that relates bias and variance to generalization performance as the basis for the argument of how transfer function diversity can improve generalization ability.

3.4 Chapter Overview

In this chapter, we shall look at the literature related to transfer function diversity. In particular, we will show the critical role of bias in learning and how transfer functions -among other neural network components- introduce a bias that directly affects the generalization ability of artificial neural networks. This is explained in the light of both the meta-learning and statistical perspectives and presents one of the contributions of this chapter. The main contribution of this chapter is then presented, which shows that transfer function diversity can improve learning by resulting in a neural network with a self-adaptive bias, and why that is essential for generalization in learning. This is elaborated formally with the help of a state-graph.

In this chapter, the nature of the search space of learning algorithms is described in general. This specifically includes sections that explain the input space, computational strategies space, and the bias space. This is followed by an explanation of the role of bias in learning. This consists of sections that consider the definition of bias from both the meta-learning and statistical perspectives and shows that both essentially explain the same effects. It also contains sections that illustrate how transfer function diversity can improve diversity by modeling the computational strategies space as a state-graph. Afterward, it is shown that bias substantially determines the set of hypotheses or computational strategies available to the neural network, and as such needs to be self-adaptive to ensure better generalization ability and robustness.

3.5 Chapter Background

3.5.1 The Nature of the Search Space

Literature from the meta-learning perspective presents a stacked representation of the input space, where there are three layers: the input space, hypothesis space, and the bias space. In our work, we describe the hypothesis space as the *computational strategy space*. In this section, we shall highlight the role of these search spaces in learning.

3.5.1.1 Input Space

The input space (feature space) is a hyperspace of k dimensions, where the number of the dimensions is determined by the number of features for the problem, $x = \{i_1, i_2 ... i_k\}$. Ideally, the points in this search space - which are meant to represent a case of the problem - should be representative of the problem. This includes having a balanced set of points in the dataset D such that each outcome (i.e. class) has about the same number of cases. An unbalanced set of points for the outcomes $C = \{c_1 ... c_n\}$ (where n is number of classes) of these cases will not be entirely representative of the nature of the problem. This can mislead the neural network to form an incorrect hypothesis about the nature of the problem, resulting in an inaccurate generalization. This happens when computational strategies that ought to describe the problem are incorrectly scored due to the imbalance in the data set. In addition to a balanced dataset, the set of features x used to describe the problem also plays a significant role in the choice of the computational strategy used by the trained model. An inadequate set of features is likely to fail in capturing certain dimensions of the problem, and consequently, it's true nature. This incomplete information is likely to result in an inaccurate hypothesis of the underlying function of the problem. As a consequence of this, the neural network learner is likely to adopt a computational strategy that will be unable to make the correct generalization of the problem. Likewise, a feature set that has higher dimensionality than required is likely to make learning more difficult. This is because the additional dimensions introduced by the additional features of the problem might clutter the true nature of the problem by providing information that might not be related to the required prediction. This makes it difficult to select a computational strategy that forms the correct decision boundaries which separate the outcomes of the problem.

In summary, the input space is a hyperspace that represents the dataset. This points to the significant role of dataset gathering in machine learning. It has a role that is as significant as the learning algorithm.

3.5.1.2 Computational Strategy Space

The computational strategies space consists of all the possible strategies $H_G = \{h_1(.)...h_p(.)\}$ of forming the decision boundaries that split the points in input space, which might or might not divide them into their respective classes. This can be related to the various ways of shattering the points used in the measure of the VC dimension. This space includes strategies that might not describe the underlying function of the problem. As such applying the evaluation function on this space will result in a fitness surface, with the most appropriate strategies being the lowest points (forming valleys), while the least appropriate strategies being the higher points (forming hills). One can speculate that as a model becomes more complicated, the number of local minima in this space increases. This is because an increased complexity should introduce more computational strategies, including those that don't describe the most appropriate decision boundaries for the given problem. Others might fit the dataset completely, resulting in over-fitting. This increase in the number of potential strategies results in the formation of more local minima. However, because every neural network model is limited by constraints such as the range of weights to search, maximum number of hidden units, and the transfer functions being used; the model l is limited to only a subset of this search space, as such the search space available for the model $H_l = \{h_1(.)...h_q(.)\}$ is a fraction of the global search space containing all the possible strategies H_G. Thus, $H_l \subset H_G$. This is to some extent, can be considered to be a good thing, since the model is exposed to relatively less local minima as a result. However, from the standpoint of the likelihood of the model finding the most appropriate computational strategy, it is a disadvantage.

A model of a neural network is, in essence, choosing a computational strategy from this space based on the dataset D presented to it. The choice of which hypothesis to choose is heavily dependent on the data set, and the evaluation function. Noise or dataset imbalance can mislead the choice of the most appropriate computational strategy. Likewise, the evaluation function also affects the choice of the hypothesis adopted by the neural network model; an evaluation function with a penalty for complex models is likely to bias the choice of the computational strategy adopted.

3.5.1.3 Bias Space

In the earlier section, we have discussed how the evaluation function and the dataset affect the choice of a models' computational strategy. Thus, the model of a neural network is biased towards a certain set of strategies over others. This effect and others similar to it such as constraints on optimization parameters limit the region of the computational strategy space being searched for solutions. The set of biases imposed on a neural network (i.e. $B_l = \{b_1 ... b_m\}$) compound and dictate the region of the computational strategy space available to the model. The VC dimension of a model d_{VC} could be a suitable measure that should be able to roughly approximate the size of the search space available to the neural network model, i.e. $d_{VC} \simeq q = |H_l|$. The size of this search space $|H_l|$ gives us an idea of the computational complexity of the neural network model and consequently tells us about the degree of bias the model has. This is because a coverage of a large space is actually a consequence of the model having a considerable complexity which enables it to exhibit a wider range of decision boundary forms. Also, this increased capacity to form a broader range of decision boundaries is as a result of a small degree of bias. This is also true, vice-versa.

The bias space is thus a search space containing all the possible bias sets that any learning algorithm can have, i.e. $B_G = \{b_1 ... b_M\}$, which both influence the choice of the computational strategy, as well as dictate the size (i.e. $|H_l|$) and region of the search space available for searching (i.e. H_l). This is explained later on in more detail.

In a nutshell, the computational strategies space H_G can be described as a search space containing all the possible functions that map the input space into classes. These mappings vary in the degree of correctness, which is why a fitness surface has hills (representing relatively worse solutions) and valleys (representing relatively better solutions) projected by the evaluation function. Finally, and most importantly, we have seen that the input space (projected by the dataset), and the fitness landscape generated by the evaluation function influence the choice of the models strategy. In other words, they *bias* the model towards choosing some strategies over others.

3.5.2 The Role of Bias in learning

In this section we present the role of bias in learning; firstly we start by providing the definitions of bias both from literature from both the statistical and meta-learning point of view. We illustrate that these definitions describe the same effect. Afterward, these definitions of bias and their established relationship in the literature is used to show how transfer function diversity can result in increased generalization ability.

3.5.2.1 Definition of Bias

From a statistical point of view, as in the bias-variance decomposition; the bias in relation to a given data point $x_i \in D$ is the expected difference between the predictions of the trained model for the data point $h(x_i)$ and the target value $y_i = f(x_i)$ of the data point for the given dataset D. It is expressed as a component in the bias-variance decomposition in terms of the mean squared error (MSE), given as:

$$E_D[Generalisation] = (Bias)^2 + (Variance).$$
$$E_D[(h(x_i) - f(x_i))^2] = (E_D[h(x_i)] - f(x_i))^2 + E_D[(h(x_i) - E_D[h(x_i)])^2]$$

To achieve the best generalization, the bias and variance components need to be minimized, such that the learning algorithm has little variations in the predictions between the models it produces on different samples of the problem. In other words, the difference between each model's prediction and the average prediction of the whole lot of the models should be minimal. This has a grouping effect which ensures some consistency between the models in their predictions. However, it is not enough that the models have a tight cluster, they also have to be accurate. Being accurate requires that learning algorithm considers the dataset in making its hypothesis about the underlying function of the problem. Having a preconceived notion of the hypothesis is a bias, and is quantified by the bias component.

In the context of meta-learning, the bias is regarded in its more linguistic meaning of anything that influences the learning algorithm [90, 75, 102, 103, 100]. This could be the limits on an individual parameter, or inherent tendencies of an initialization method, feature selection method, or the choice of transfer functions for the hidden units.

It can be said that both definitions of bias are essentially referring to the same effect, however while one is of a quantitative nature (i.e. bias-variance decomposition), the other is of a qualitative nature (i.e. meta-learning definition). This is because both describe the two primary properties of bias:

- **Restriction**: Dictating the size of the search space available to the optimization algorithm for any learning algorithm.

- **Inherent Tendency**: This is the inherent tendency of components, such as the tendency of transfer functions towards certain responses due to their inherent design; it is born from the assumption that lead to the design of the components. This tendency could be regarded as an inherent heuristic. (e.g. tendency of the selection phase of learning algorithms to choose solutions with the least errors on patterns).

We shall explain why both definitions are indeed the same. In the case of the bias-variance decomposition, which we refer to as *bias-error*, the effect of restriction is measured with the bias-error. The first extreme of the restriction effect is when the bias for the learning algorithm is not able to adapt to the training set, possibly due to the lack of a viable solution as a result of a bias that is too restrictive. We shall assume that the model produces a constant output, in this case, k.

$$\begin{aligned} h(x_i) &= k \\ E_D[(h(x_i)-f(x_i))^2] &= (E_D[k]-f(x_i))^2+(E_D[(k-E_D[k])^2]) \\ &= (k-f(x_i))^2+(E_D[(k-k)^2]) \\ &= (k-f(x_i))^2+(E_D[0]) \\ &= (k-f(x_i))^2 = (k-f(x_i))^2 \end{aligned}$$

In this case, there will be no variation in predictions of the models produced on the dataset. The expected error will be the bias-error, which is the error between the constant value k and the target output of a pattern y_i, given the pattern x_i, i.e. $y_i = f(x_i)$. The second component of the decomposition, *variance-error* on the other hand will be zero, signifying that the learning algorithm is not receptive of the dataset. In meta-learning, this is described as a *strong* bias [75, 103].

The opposite extreme is when the learning algorithm is very receptive of the dataset such that it always predicts the target output exactly, in which case the bias component will be zero. In other words, the learning algorithm has more free parameters than is probably required; thus, it essentially memorizes the dataset D. Let's assume that the output of the learning algorithms on a pattern x_i is the same as the target y_i for the dataset D, i.e. the bias-error is $(E_D[h(x_i)]-f(x_i))^2 = 0$. This can be expressed as:

$$h(x_i) = f(x_i) = y_i$$

However, the target for the data point x_i in the dataset is likely to have some inherent irreducible noise $\eta_i \in N(0,\sigma)$, which is independent of the dataset [45], and as such y_i is likely not exactly the *true* target value produced by the underlying function $f(x_i)$ portrayed by the dataset D. Thus, the function being portrayed by the dataset can be expressed as:

$$f(x_i) = g(x_i) + \eta_i$$

Therefore, regarding the bias component of the generalization error:

$$\begin{aligned} E_D[(h(x_i) - g(x_i))^2] &= (E_D[h(x_i)] - g(x_i))^2 \\ &= (E_D[g(x_i) + \eta] - g(x_i))^2 \\ &= (E_D[\eta_i])^2 \\ &= (\sigma)^2 \\ &= \sigma^2 \end{aligned}$$

In such a case, the error for the dataset D will be the variance of the noise squared σ^2 in the dataset, since the noise η has zero mean [45]. In this case, the algorithm has overfitted the dataset, as well as its noise. In other words, the algorithm is too receptive. In meta-learning, this is described as a *weak* bias [75, 103]. In a case where the variance of the noise σ is high; the learning algorithm will incur a high degree of variance as a result. In which case, it becomes overly sensitive to noise.

3.6 Transfer Functions Diversity Can Improve Generalization

It can be shown that the transfer function of artificial neural networks contributes towards the generalization performance of neural networks by their bias using state graphs. It can also be shown that the bias of neural networks needs to be self-adaptive according to the problem for them to be effective [45] as well as applicable to a wider range of problems.

We start by defining a state graph representing the hypothesis space H_ℓ of a neural network, ℓ as in Fig. 3.5 generated by the function γ in Eq. 3.1, which returns the search space $H_\ell = \{h_1 ... h_K\}$ of the neural network ℓ based on its set of biases, i.e. $B_\ell = \{b_i ... b_n\}$. The size of the hypothesis space is represented by K , which is dependent on the computational capacity of the neural network.

$$H_\ell \leftarrow \gamma(B_\ell) \tag{3.1}$$

In this case, b_i in the set of biases B could be a range for parameters of the neural network components such as the transfer functions, weights or topology - which can result in the restrictive effect, or inherent tendencies in the components of the neural network. For example, the inclination of a radial basis function unit to respond to values in proportion to how close they are to its center [103, 52]. These also have a combined effect that contributes

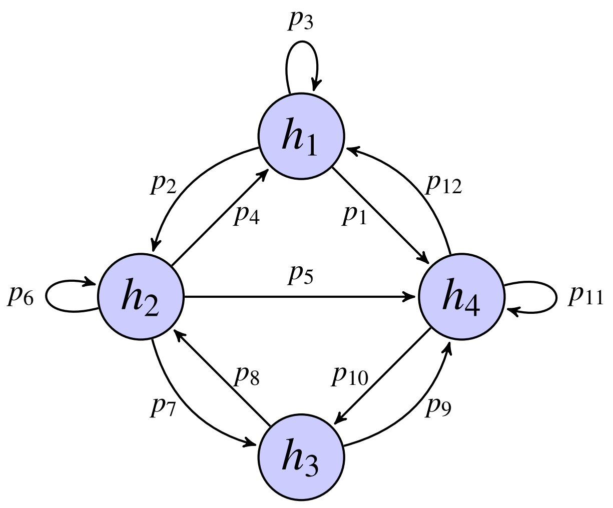


Figure 3.5: State graph defining the hypothesis space H_ℓ for the neural network ℓ

towards determining the probability of the neural network traversing the edges in the state space of our neural network ℓ. Thus, the probability p_{ij} represents the tendency of the neural network towards a hypothesis j, from a hypothesis i given the set of biases B_ℓ. One of the contributing components that influences these probabilities is the error function (or loss function) $\mathscr{L}(.,.)$ defined by the optimization algorithm. For example, the Mean Squared Error (MSE) that approximates the discrepancies between the hypothesis and the underlying function $f(x)$ which also belongs the global space of functions H_G that produces the dataset D. In other words, $f(x) \in H_G$.

$$\mathscr{L}(h_i(x), f(x)) \tag{3.2}$$

and squared error loss function defined as:

$$\mathscr{L}(h_i(x), f(x)) = (h(x) - f(x))^2$$

Another component which contributes to determining the probabilities is the optimization function which directs the search towards finding the hypothesis that minimizes the loss function:

$$argmin_{h_i(x) \in H_\ell} \mathscr{L}(h_i(x), f(x)) \leq e \tag{3.3}$$

the goal is to find the hypotheses $h_i(x)$ from the set of hypotheses H_ℓ which minimizes the loss function below a target error e.

If we consider a single state in the graph, for example h_1, the set of hypothesis or states that it can traverse to from its current configuration, given operators that can change the neural network model such as local learning, mutation of weights, back-propagation, and

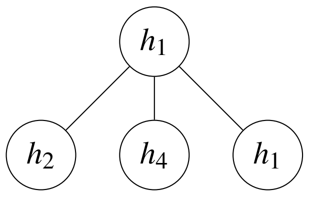


Figure 3.6: Graph from the state of h_1, this also highlights refinements of the states.

others; it can be represented as a tree. The children $\{h_2, h_4\}$ of the parent h_1 node can then be described as the hypotheses that can be adopted after applying an optimization operation or a sequence of them on the neural network model.

In this case (Fig. 3.6), the graph also has a state of choosing to remain with the hypothesis h_1, by operations that result in variations of the same hypothesis. The optimization algorithm guided by the optimization function (Eq. 3.3) steers the neural network around this state graph with its operators to converge on a hypothesis represented as a local minimum in the fitness surface over the hypothesis space H_ℓ, defined by the loss function - as in Eq. 3.2.

It becomes more apparent that the optimization process depends on the set of biases of the neural network B_ℓ in searching through the hypothesis space available to the neural network H_ℓ. This is because the states or hypotheses accessible during the search are implicitly determined by the combined effect of biases of the neural network as defined in Eq. 3.1. In other words, if there is a set of biases B_ℓ such that the hypothesis h_2 cannot be approximated with every available optimization process applied to the neural network, then such a state will not exist in H_ℓ. Likewise, if there was a hypothesis h_i that could be approximated given a set of biases $b_1 ... b_m$, then that hypothesis will exist in the hypothesis space for the neural network H_ℓ. As Vilalta [103] also explained, bias determines the region of the hypothesis space being searched. And as German et. al [45] also concluded, this is one of the reasons why the bias needs to be *purposefully* designed and introduced in a way that it provides access to the a search space with the underlying function being approximated or an approximation of it.

Given the function for state generation (Eq. 3.1), we can show that the set of biases adopted by a neural network B_ℓ influences the likelihood of finding a hypothesis that generalizes well, i.e., the neural network's ability to learn and generalize the problem.

Using the optimization function given in Eq. 3.3, we can define a subset of the hypothesis space for the neural network H'_ℓ which consist of hypotheses that minimize the loss function, such as a mean squared error, to at least a target error θ.

$$H'_\ell \leftarrow argmin_{h_i \in H_\ell} \mathscr{L}(h_i(x), f(x)) \leq \theta$$

As such, convergence less than or equal to θ will be within the set hypotheses described in H'_ℓ. This has the effect of thresholding other sub-optimal local minima (or hypotheses) and considers only those with the desired generalization error.

Eventually, the final hypothesis converged on that minimizes the loss function will be a member of the H'_ℓ set. But, H'_ℓ is a subset of H_ℓ, i.e. $H'_\ell \subset H_\ell$. In other words, the set of biases B_ℓ of the neural network ℓ defines the hypothesis space, thereby influencing whether or not the hypothesis with the most generalization ability is found. If the hypothesis space doesn't have the most appropriate bias for the problem, then it is unlikely that a hypothesis that generalizes well will be found. In other words:

$$\nexists\, h_i(.) \in H_l :\ E_D[(h_i(i_j) - f(i_j))^2] \leq e$$

However, it can be expected that as neural network gains more access to more computational strategies (or hypotheses) from the global set of all possible computational strategies, H_G then one can expect that there will be an increased likelihood of it finding a function that describes the underlying function with the desired error, e. In other words:

$$lim_{|H_l| \to |H_G|} Pr(\exists\, h_i(i_j) \in H_l : E_D[(h_i(i_j) - f(i_j))^2] \leq e) = 1$$

As $|H_l| \to |H_G|$ then there is certainly going to be a computational strategy $h_i(.)$ that best describes the underlying function $f(.)$ with the desired error e. However in practice, as $|H_l| \to |H_G|$, there will be a high degree of *variability*, because of the hypothesis space having various other sub-optimal hypotheses, or local minima. This also heightens the danger of overfitting in which a hypothesis $h_i(x)$ memorizes the examples $\{(x_i, y_i)...(x_n, y_n)\} \in D$. This is because of the increased computational capacity, which typically entails additional free parameters.

In a nutshell, we can conclude that there is a need for the adaptation of the biases $B = \{b_i...b_m\}$ of the neural network according to the problem. This is also supported by various other works from both meta-learning [103, 75, 100, 90] and statistical [45] perspectives. In the case of neural networks, one of the components that plays the important role of forming decision boundaries is the transfer function, which also has inherent biases. The motivations for diversifying transfer functions apart from its biological plausibility is so that the optimization algorithm can optimize the bias of the neural network.

Chapter 4

Ensemble of Neurally Diverse Artificial Neural Networks

In the previous chapter, an explanation of how neural diversity in the form of transfer function diversity was provided in light of the literature that links bias to generalization ability, namely the bias-variance decomposition and the meta-learning concept of the search space.

This chapter experiments with ensembles of neurally diverse artificial neural networks to show that it can produce diversity in the form of diverse computational strategies, which is an essential part of ensembles. This is significant because it shows empirical results that transfer function diversity can result in the ability to exhibit more computational strategies, which is the basis of the argument on how neural diversity in the form of transfer function diversity can improve generalization. It also provides conclusive evidence of how the computational strategies differ such as the ones for Diabetes, XOR, and Iris problems by studying the models produced.

4.1 Motivation

There are several reasons for using ensembles for experimenting with neural diversity; the first being that the whole concept of ensembles is based on diversity of biases in the members of the ensemble. Showing that neural diversity can achieve significant generalization ability without other explicit diversity maintenance such as bagging or boosting will imply that neural diversity is helping the neural networks to exhibit diverse computational strategies. Furthermore, we also provide conclusive evidence that the neural networks are diverse by studying their computational strategies. The second is that it also helps us address the gap in the literature of lack of works that use neural networks with hybrid transfer functions in ensembles as highlighted by Brown et. al [19]. In general, the area of transfer function optimization, of which hybrid transfer function is a part of, is generally under-researched.

Ensembling is an elegant solution towards the restrictive nature of bias in learning. It is a method of learning that typically involves several diverse learners trained on a common problem. There are various works [19, 94, 83, 106, 50, 66, 82] that have shown that neural networks combined as ensembles exhibit an improved generalization ability provided there is diversity in the bias of its members. In other words, if there are differences in the limitations of the learners, then an ensemble of such learners should result in increased learning accuracy. In addition to these promising empirical findings, it is also proven [19] that the error of an ensemble is guaranteed to be the same or better than the average error of its members.

The importance of diversity in ensembles is quite intuitive. If the individual members of an ensemble were all identical, the performance of the ensemble would not differ from any of the members of the ensemble. This is because they are likely to have the same set of biases. Without any form of diversity maintenance, the learners will have similar generalization errors on patterns. However, if all the members of the ensemble were different from each other such that the decision boundaries that they project onto the input-space are varied yet accurate, then one can expect that the averaged decision boundary of the learners is likely to be significantly more accurate, or at the very least the same. The significance also depends on some other factors such as the method used to combine the outputs of the members, the number of members of the ensemble, and the accuracy of the members of the ensemble. The diversity of the members of the ensembles (e.g. different learning algorithms) means that they will have different biases. Essentially, these learners will be searching different regions of the computational strategies search space, as a result of their bias. Thus, they are more likely to adopt different hypothesis with different generalization errors. Combining the classifiers is essentially compensating for the bias-error of each of the members of the ensemble in predicting patterns.

The fact that ensembling depends on the members being diverse and accurate makes it ideal for showing that neural diversity can produce diverse, and accurate computational strategies within one experiment. It also enables us to study the different computational strategies that make up the ensemble of models produced for the problems. In addition to these, it also helps cover a gap in the literature of ensembles that

4.2 Related Works

Most of the promising methods for creating diversity in ensembles can be categorized into three according to their area of focus [19]: data set, model, and training algorithm. In the first case, approaches found in the literature typically use re-sampling and pattern distortion methods to achieve some variations in the training data set thereby implicitly inducing behavioral differences in members of the ensemble. This is due to differences in the input space which essentially affect the error surface and consequently, the final computational strategy adopted by each member. Popular re-sampling methods include bagging and boosting [19]. In bagging, random samples of the data set (with replacement) are used to train each member of the ensemble. Boosting differs because it creates an ensemble in a sequential process. It doesnt assume equal weights but assigns weights to the patterns based on the error of a base learner. Patterns that are predicted correctly get higher weights while the pattern corrected wrongly get lower weights. The main idea is to get each classifier to perform better than its the base classifiers.

Another method used by some studies that was highlighted by Brown et. al. [19] is to re-sample the features of each pattern in the training data set. This also essentially, causes changes in the dimensions of the input space, and as a result indirectly influences the hypotheses adopted by each member. On the other hand, distortion methods used include the addition of Gaussian noise or non-linear transformations of the training patterns in the data set. One of the non-linear transformation approaches found [19, 94] found to be effective was to stimulate a randomly generated neural network with the training pattern and then use its output as the distorted pattern. Gaussian noise was also found to be helpful [19, 94]. Memetic Pareto Evolutionary Artificial Neural Networks (MPANN) [1] used this approach to generate a validation set, which was then used as the second objective function apart from the objective of optimizing the training error. Thus, two objectives needed to be optimized; the training and validation errors. The pareto-front, which consists of a set of models that optimize both the training and validation error better than the rest of other models, was then used for form the ensembles.

In the case of diversity creation methods focusing on models [19], some methods use a mixture of models. The most popular include the use of similar models with varied parameters [19], such as neural networks of different hidden layer sizes as used in Constructive Cooperative Neural Network Ensembles (CNNE) [53]. In the study, the training epoch and hidden unit size were varied between the members of the ensemble or different types of architectures - such as Multilayer Perceptrons and Radial Basis Function networks [19]. Other methods [19] use mixed models, such as decision trees and neural networks within an ensemble. The final category consists of methods which focus on the training algorithm, some of which include the use of different training algorithms [94, 27]. Another is the introduction of an additional term in the objective function [66, 82, 67], such as in neural network ensembles trained by evolutionary algorithms [26, 19].

There are still some aspects of ensembles that have yet to be studied. One such aspect is the topic of model diversification approaches for neural network ensembles. Intuitively, it makes sense that if we are aiming for diversity in the bias within our neural network ensemble, an equally likely approach to the others that could yield significant diversity is an approach that is explicit. By *explicit*, we mean an approach that takes a direct approach, such as the combination of diverse architectural models of neural networks. Though there has been work with different architectures of neural networks, according to our knowledge, there is a lack of experiments with ensembles using hybrid artificial neural networks, specifically architectures implementing a set of diverse transfer functions, which we would expect to increase diversity in ensembles. This was also highlighted in the thorough survey of ensembles by Brown et. al[19] as an aspect that needs studying. The only partially related work done so far was by Partridge who used pure models of Multilayer Perceptron's (MLP) and Radial Basis Functions (RBF) in an ensemble to achieve diversity. However, even that work was suggested to be a preliminary study by [19].

4.3 Contributions

This chapter's main contribution is that it empirically shows that neural diversity in the form of transfer function diversity can improve generalization performance. It does this by demonstrating that it can exhibit diverse and accurate computational strategies, which is one of the bases for the argument of how it can improve generalization ability in artificial neural networks (as presented in the previous, i.e. Chapter 3). This is because the diverse and accurate models in the ensembles complement each other, thus resulting in promising results on five popular benchmarks with an ensemble of relatively low complexity models.

It also addresses the general lack of literature on transfer function optimization methods in general and lack of results from their use in ensembles. Specifically, it presents transfer function diversity as a biologically plausible alternative to diversity maintenance in neural network ensembles. In particular, it is shown that the bias adaptation property of transfer function diversity can produce neural network ensembles with promising generalization ability. Finally, this chapter contributes by proposing measures for artificial neural networks that are then used for unveiling the underlying computational strategies the ensemble members have learned.

4.4 Overview

Firstly, the section presents the application of hybrid neural networks in ensembles and provides a study on some of the effects of transfer function diversity in neural network ensembles. Secondly, it shows that this neural network framework can develop different strategies for a problem that can be used in ensembles without explicit diversity maintenance that can be expensive, such as selection of the Pareto front for use as ensembles, or fitness sharing. It also shows that this approach can evolve a relatively smaller ensemble of small networks that has a competitive performance. Finally, it demonstrates how neural diversity can result in diverse classifiers by analyzing two computational strategies evolved for the diabetes problem. Furthermore, this also illustrates how the strategies contribute to the model's overall performance with their strong discriminatory property using the XOR problem; thus, conclusively showing the role of neural diversity in the generalization performance.

Experiments with hybrid neural network ensembles implementing a diverse transfer function set, also known as Neural Diversity Machine Ensembles (NeuDiME) are conducted. This is unlike other approaches found in the literature which have used a mixture of pure models as reported by Brown et. al [19]. We study the performance of this method in different circumstances; specifically, we test it on popular pattern recognition problems such as the Iris, Sonar, Hepatitis, Diabetes, and the Australian credit card data sets, commonly found in the UCI machine learning repository [13].

4.5 Neural Diversity Machine Ensembles (NeuDiME)

Neural Diversity Machine Ensembles (NeuDiME) is an ensemble of neural networks with transfer function diversity. As explained earlier, the framework of neuronal diversity in the form of transfer function diversity was proposed by Maul [71], where he highlighted their promising results in preliminary experiments. It uses a transfer function set consisting of various activation functions (see Table 4.1) and output functions (see Table 4.2). This enables the optimization algorithm to evolve clever solutions to problems as we shall see in the experiment with NeuDiME, which sometimes involves the use of rare activation functions such as standard deviation, *max* and *min* as filters as we highlighted in [4, 71]. This work builds on the work by using them in ensembles to address: the lack of literature using hybrid transfer functions in ensembles, and to empirically demonstrate that neural diversity can exhibit diverse and accurate computational strategies, which was the basis for the argument in the previous chapter.

Diversity in transfer function set is a form diversity promotion mechanism, as also highlighted in [19]. There has been little work on ensembles with hybrid models, specifically, those using a diverse set of transfer functions for neural networks. A diverse set of computational strategies as a result of this form of diversity should lead to more diverse computational strategies or hypotheses available for the neural network, i.e., the hypothesis space H_ℓ should be expected to consist of a lot more variety. Consequently, this is expected to result in the desired diversity of bias required by ensembles. In terms of the bias-variance decomposition, a more diverse ensemble of classifiers adapting computational different strategies should compensate for the bias-error of the individual classifiers when their output is combined; thus, reducing bias-error. In terms of variance, the other criteria of having accurate members of the ensemble should result in less variance between the models. This could be supplemented with other penalty functions attached to the objective function as in negative correlation learning [53, 66, 67, 20]. However, NeuDiME does not implement any penalty function.

It is intuitive that an ensemble made up of different yet accurate models will likely yield more *useful* diversity. By *useful*, we are referring to the diversity that results in significant improvements in the generalization ability of the ensemble. This work differs from other approaches which have attempted the use of neural networks of different sizes within an ensemble or a diverse selection of classifiers types [19]. While ensembles of neural networks with varying sizes will result in having various regions of the computational strategies search space, the inherent bias of the transfer functions of the neural network remains the same. There is a need for a more explicit approach to creating diversity of bias, and neuronal diversity is one of such approaches which is also biologically inspired.

4.5.1 Neural Diversity

There are only a handful of studies that have proposed using hybrid models implementing neural networks with a variety of transfer functions [47, 55, 61, 3, 31, 30]. While the other works were mostly motivated by the theory of duality of functions for their pool of transfer functions, which shows that any continuous function can be decomposed into a radial basis and projection basis component; this work is primarily motivated by the biological design pattern of neuronal diversity [18, 101].

Neuronal diversity is prevalent in biological neural networks. As already stated, it is believed to be one of the primary reasons behind the efficiency, and computational capacity of biological neural networks [18, 99, 101, 70]. In this work, it is used to achieve better generalization ability and efficiency in artificial neural networks with relatively limited computational resources (i.e. in the form of hidden units, and connection weights).

Table 4.1: List of some input combination and output functions used by Neural Diversity Machines and their visualization color codes.

Indices	Activation Functions	Color Code
1	Inner-Product ($j = \sum_i^k w_i i_i + w_{bias}$)	Red Solid Edge
2	Higher-Order Product ($j = \prod_i^k cw_i * i_i$)	Yellow Solid Edge
3	Higher-Order Subtractive ($j = \sum_{i=1}^k \lvert x_0 - x_i \rvert$)	Yellow Dashed Edge
4	Euclidean Distance ($j = \sqrt{\sum_i^k (w_i - i_i)^2}$)	Magenta Dashed Edge
5	Standard Deviation ($j = stdDev(w_i i_i, w_{i+1} i_{i+1} ... w_k i_k)$)	Blue Solid Edge
6	Min ($j = min(w_i i_i, w_{i+1} i_{i+1} ... w_k i_k)$)	Gray Dashed Edge
7	Max ($j = max(w_i i_i, w_{i+1} i_{i+1} ... w_k i_k)$)	Black Dashed Edge

Table 4.2: List of output functions for Neural Diversity Machines and their visualization color codes. θ - is a threshold that is learned during optimisation.

Indices	Output Functions	Color Codes
1	Linear ($z = \alpha * j$)	Yellow Node Outline
2	Hyperbolic tangent ($z = \frac{1-e^{-\alpha * j}}{1+e^{-\alpha * j}}$)	Cyan Node Outline
3	Sigmoid ($z = \frac{c}{1+e^{-\alpha * j}}$)	Red Node Outline
4	Gaussian ($z = e^{\frac{-(j)^2}{width}}$)	Blue Node Outline
5	Gaussian II ($z = e^{\frac{-(j)^2}{width}} if\, z > \theta\, then\, z = 1$)	Dark-Blue Node Outline

This implementation mimics that property by using a pool of diverse classes of activation and output functions, which can then be combined and adopted by any of the hidden or output units of the neural network. The list of activation functions available to the optimization algorithm are as in Table 4.1, while the list of output functions are as in Table 4.2. There are no restrictions on the type of activation functions that can be combined with output functions, nor are there restrictions on certain combinations of activation functions and output functions occurring more than once in the neural network model. The neural network model is completely free to adopt any form of transfer functions for the hidden and output units. This flexibility has enabled the study of the computational strategies that emerge as solutions. As we shall see, this approach tends to evolve creative strategies for problem-solving.

Regarding the topology, there are also no restrictions on inter-layer connections or intra-layer connections for this implementation. The global stochastic algorithm can optimize the topology by applying any of the evolutionary operators (i.e. cross-over and mutation) or the differential evolution operation on a probabilistic basis.

Table 4.3: Neuron (node) parameters.

Node Parameters	Description
$i(g(x))$	Index of the activation function
$i(f(x))$	Index of the output function
p_1	First output function parameter
p_2	Second output function parameter
p_3	Third output function parameter
b	Bias
b_w	Bias weight

4.5.2 Optimization

The use of evolutionary algorithms for optimizing the transfer functions of neural networks, among other architectural components such as the connectivity and weights, is partly due to the nature of some of the transfer functions, which makes other algorithms such as Backpropagation not applicable due to its reliance on gradients.

Direct encoding is used to represent the encoded components and their parameters using real values on the genetic string. The components encoded include: the weights between neurons, the transfer function choices of each neuron and their functional parameters, and the bias of neurons (See Tables 4.3 and 4.5).

Encoded Components

In terms of neurons, the choice of the activation function $i(g(.))$, is encoded as the index of the activation function in the pool of activation functions (Table 4.1), and the choice of the output function $i(f(.))$, is encoded as the index of the output function in the pool of output functions (Table 4.2). The output function parameters $\{p_1, p_2, p_3\}$ such as variance in the case of Gaussian functions are also encoded onto the genetic string. In addition, the bias typically used for neurons in the literature, i.e. b and its weight b_w are also encoded onto the genetic string for optimizing. The encoding of these parameters with the exception of the neuron bias and its weight are made in sequence such that their context (i.e. of belonging to a certain neuron) is preserved. Table 4.3 shows the list of parameters encoded.

$$\{i(g((.)), i(f(.)), p_1, p_2, p_3\}$$

The bias and its weight for each neuron i was also encoded in sequence as well but at the tail of the genetic string in the form of $\{b_i, b_{w_i}, b_{i+1}, b_{w_{i+1}} ...\}$. In a later chapter, we present an implementation that encoded the network in a more contiguous manner.

Table 4.4: Details of the output function parameters encoded onto the genetic string. Some of the output function parameters, in particular, p_2 and p_3 were not used, but encoded onto the genes for future experiments that might require encoding additional output function parameters.

Output function	Parameters	Description
Hyperbolic Tangent (tanh)	p_1	steepness
	p_2	N/A
	p_3	N/A
Sigmoid	p_1	steepness
	p_2	numerator (c)
	p_3	N/A
Gaussian	p_1	width
	p_2	N/A
	p_3	N/A
Gaussian II	p_1	variance
	p_2	threshold (for wider center)
	p_3	N/A

Table 4.5: Listing of other components encoded onto the genetic string.

Component	Description
C	Connectivity matrix for the neural network model
W	Weight matrix for the neural network model

The set of output functional parameters $\{p_1, p_2, p_3\}$ encoded varied depending on the type of output function. Table 4.4 lists the output functions, and their respective parameters and descriptions.

Regarding topology and weights, the components encoded included the connectivity matrix C and weight matrix W of the neural network model.

Constraints

There were some constraints introduced to regulate the complexity of the search space. These included the acceptable range of parameters such as the output function parameters $\{p_1, p_2, p_3\}$. These are listed in the table 4.6.

Some of the constraints were transfer function specific, in particular; the $minp_i$ and $maxp_i$ constraints were different for different output functions. In other words, there are constraints for the $minp_i$ and $maxp_i$ parameter for a sigmoid output function's parameters p_i, and a separate range of output function parameter for a Gaussian output function's parameter p_i. This was because the p_i played different roles in different functions.

Table 4.6: Constraints on the parameters encoded onto the genetic string.

Constraint	Description
$minb_w$	weight bias minimum
$maxb_w$	weight bias maximum
$minp_i$	output function parameter p_i minimum
$maxp_i$	output function parameter p_i maximum
$minx_i$	minimum allowed range for any other parameter x_i in the genetic string.
$maxx_i$	maximum allowed range for x_i

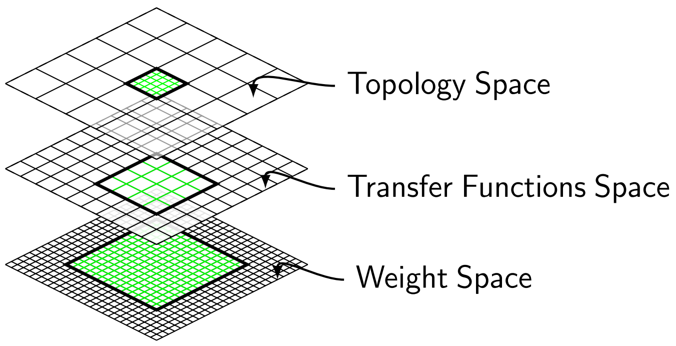


Figure 4.1: The hierarchical nature of the search space.

Representation and Encoding

The method of encoding considers an intuitive knowledge of the hierarchical nature of the search space of artificial neural networks. In artificial neural networks, the size of the neural network determines the size of the connectivity space, weight space and the transfer function space. This is because as the size of the neural network N is grown, so does the number of possible connections and their weights. The same applies to the transfer function space, which in our case consists of the various transfer function choices and their functional parameters. Some of these sub-spaces grow at a larger rate than others, for example, the connectivity space will consist of at most N^2 possible connections between all the neurons in the neural network model. However, the weight space will be composed of all the possible real values that can be used as weights between these connections (i.e. $w^{n*n} \in \mathbb{R}$), which is an infinite search space - if not constrained by precision and range.

The size of the neural network N influences the computational strategies or hypotheses made available H_ℓ by the bias it introduces, and in turn the computational strategies and size of the neural network influence the connectivity space and the fitness of the members of that search space. Finally, there is the weight space which depends on the connectivity space for its dimension and fitness as well. In a later chapter, we shall see how this interdependence can be represented in an artificial neural network, and show its promising generalization ability on real world datasets.

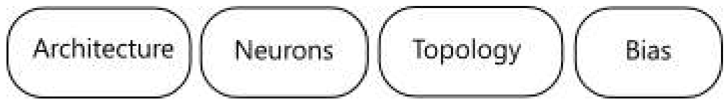


Figure 4.2: An illustration of the sequential encoding of components of the neural network by their types.

As such the neural network search space is hierarchically dependent on the size of the neural network, which determines the size of the connectivity space, and in turn dictates the dimensions of the weight space (see Fig. 4.1). Thus the connectivity space can be considered to be a sub-space above the weight space, in the sense that it influences the dimensions of the weight space. Selecting a single connectivity pattern C_i from the topology space then results in a set of possibilities for weights $\{w_i^{n*n}, ...\}$ that can be given for each of the active connections.

The method of encoding takes these into account and optimizes the dimensionality of the search by only encoding parameters of active nodes onto the genetic string. Information about nodes and connections that are not active are not encoded. This helps to reduce the dimensionality of the search space, as it removes information about nodes and connectivity related to it (i.e. either from, or to it) that does not have any effect on the phenotype yet complicates the search space further.

Another feature of the encoding method is that the genes are encoded according to the context as highlighted earlier in section 4.5.2 (Page 55). In addition, components of similar types are also encoded in close proximity onto the genetic string by keeping them in sequence (see Fig. 4.2). This was intended to help improve the cross-over of learned information between solutions. Cross-overs usually select random cross-over points which has the potential for making incomplete information transfer between solutions, such as the case of a radial basis unit trained to approximate a cluster of points in the input space, but is crossed-over without some of its functional parameters. In this case, some learned information is lost and an offspring of such an operation would need to relearn the functional parameters left out. Though this could be a good thing from the perspective of diversifying the solutions in the neural network, there are studies which argue that cross-overs can be quite detrimental for neural networks [12, 34, 97, 112].

Optimization Algorithm

The global stochastic algorithm (GSO) adopted was a hybrid that consisted of a differential evolution (DE) algorithm with additional operators such as crossover and mutation. The optimization process is as follows (see Algorithm 1).

Algorithm 1 The underlying process of the hybrid optimization that optimizes the members of the ensembles.

```
Require: $Pop \leftarrow initPopulation()$
Require: $Pop \leftarrow evaluate(Pop)$
  for $i : maxGen$ do
    $nextGen \leftarrow diffEvolution(Pop)$ {differential evolution on a subset of $Pop$}
    $nextGen.append(crossOver(Pop))$ {cross-over on a subset of $Pop$}
    $nextGen.append(mutate(Pop))$ {mutation on a subset of $Pop$}
    $nextGen.append(generateRandomCandidates())$
    $Pop \leftarrow evaluate(Pop)$ {evaluate train and validation errors}
    $Pop \leftarrow sort(Pop)$ {sort the population by their training and validation errors}
    $Pop \leftarrow selection(Pop)$ {select top $n$ as ensemble members}
  end for
```

Firstly, the genes representing the choice of the input combination function and output function are randomly generated for each node; this also includes their functional parameters, which are all encoded onto the genetic string. This is followed by encoding the connectivity between the layers and their weights. A population of these genes is generated with varying lengths. The variation in length is as a result of the probabilistic nature of hidden unit creation. Hidden units were created on a probabilistic basis to increase the diversity within the population of neural network architectures. In this case, due to the constraints on cross-overs and differential evolution only solutions of the same length are allowed to interact (e.g. hidden layer size), this also helps in providing some degree of niching or specialization; thus, protecting computational strategies of various complexities. Varying the hidden layer size of neural networks in an ensemble has been used in related works as a diversity maintenance mechanism [19, 53, 94].

The next stage consists of evaluating the population of neural network solutions represented as genetic strings. This involves decoding genes into phenotypes (i.e. the actual neural network models) and then evaluating them on the training set. This is followed by the next step which consists of applying differential evolution to generate offsprings from a subset of the population. Cross-over operations were not biased towards the fitness of the parents. This helps in preserving the diversity of the computational strategies being evolved such that no strategy dominates the population. Also, it encourages some degree of competition, which can help in converging on the neural network species with the most appropriate architecture.

In some of the experiments, two fitness values were assigned to the neural networks. One was the fitness on the original training dataset, while the second was the fitness of the neural network model on another variant of the dataset which has received Gaussian noise treatment, i.e. $(x_i, y_i)...(x_n, y_n) \in D$, then $x'_i = x_i + \eta$. This has been used in a related work [2, 1] as a method of diversity maintenance as well.

4.5.3 Ensemble Member Selection

In this experiment, we use the Top N solutions for its relative simplicity and also because it has been used in related works [80]. This method lacks the relatively higher computational cost of other selection methods, such as selecting solutions from the Pareto-front [1, 2]. In those works, Abbass selects a set of members of the population with the least training and validation errors as the members of the ensemble. Another is the selection of members using a hill climbing approach while maintaining diversity with fitness sharing as used in [26, 60].

The method of selection selects the fittest N solutions from the population after sorting by fitness, where N is the desired size of the ensemble, example for an ensemble of 10 members, $N = 10$. This helps in improving the chances of picking the N solutions with the most generalization ability.

This selection method relies on diversity in the population, and as such, if the top N solutions are selected after sorting, they should still be a diverse set. In the case of NeuDiME, this diversity is introduced by the use of a diverse set of transfer functions, which allows for a variety of computational strategies to be explored by the neural network models during optimization. In addition to that, there is also diversity in terms of the sizes of the neural networks which helps to provide a niching effect for the computational strategies being evolved. A *niche* is essentially a sub-population that has little or no interactions with other sub-populations. In this case, the niching effect occurs as a result of the restriction that only allows solutions of the same size (i.e., hidden units) interact by getting involved in evolutionary events, such as cross-over and differential evolution.

4.6 Experimental Setup

The benchmarks used consisted of some commonly used in the ensemble literature: Iris, Sonar, Australian credit card, Hepatitis and Diabetes datasets retrieved from the machine learning repository [13] (Table 4.7) . To conform to the standard measure of generalization ability found in the literature [47, 63, 50], 10-fold cross-validation was used. The networks used all had feed-forward connectivity at initialization, with two layers of randomly generated weights. However,t hey could mutate their topologies during optimization without restrictions. A feed-forward connectivity was favored as it provides a bias towards an informal standard of topologies that is known to work well.

Table 4.7: The list of benchmarks acquired from the UCI machine learning repository [13].

No.	Benchmarks	No. of Samples	No. Of Attributes
1	Australian Card	690	7
2	Diabetes	768	8
3	Iris	150	4
4	Sonar	208	60
5	Hepatitis	155	19

Table 4.8: Experimental setup for the benchmarks showing the maximum number of hidden units allowed for each ensemble member, the size of the ensemble, and the number of folds used for K-fold cross-validation. These were set to reflect parameters in published works [1, 2, 79] as well as findings from preliminary experiments.

Benchmarks	Max Hidden units	Members(Ensemble)	Folds (K-fold CV)
Iris	5	10	10
Sonar	5	10	10
Diabetes	5	20	10
Hepatitis	5	20	10
Card (Australian)	5	20	12

The optimization parameters for the optimization algorithm are given in Table 4.9.

Table 4.9: The optimization parameters used for the neuroevolution of the ensemble members. These parameters were chosen after some pilot experiments to determine the parameters that worked best.

Optimization Parameter(s)	Value(s)
Max Iterations	100
Population size	30
Percent to eliminate	0.3
Min cost (elimination)	0.66
Min age (elimination)	3
Cross Over	True
Probability of Cross Over	0.2
Differential Evolution (DE) Iterations	3
DE alpha	0.2
Gene range	[-0.9, 0.9]
Probability of Mutation	0.2
Gaussian Mutation (Mean,Std)	(0.0, 0.2)

4.6.1 Measures

In addition to standard measures, two additional measures were developed for analyzing the underlying computational strategies adopted by the neural diversity neural networks (i.e. NeuDiME). These are referred to as signatures and are studied in more detail in the next chapter. These measures include the following:

4.6.1.1 Likelihood of Occurrence

This measures the likelihood of an activation function $g_j(.)$ and an output function $f_i(.)$ being used together as a transfer function $f_i(g_j(.))$ in elite solutions for any of the hidden or output units. This entails some information about the building blocks of the elite computational strategies, as well as some degree of implicit information about the nature of the problem among other valuable details. This is referred to as a likelihood since it represents the probability of the transfer functions based on a sample $s \in S$ - which in our case are the results of the experiments. The true probability of the transfer functions will require all the samples in S. However, for practicality, a sufficiently large sample s, which approximates the true probability should be adequate.

This was calculated by counting the frequency of appearance of transfer functions $f_i(g_j(.))$ in the elite models during evaluation, which is then normalized by the number of possible transfer function combinations. Normalizing by the number of runs usually resulted in small likelihoods that were hard to work with. As such, it could be regarded as a relative likelihood of the transfer function occurring in relation to other possibilities. Thus, it can be expressed as in Eq.4.1.

$$l_{a,b} = \frac{n_{a,b}}{|I| * |J|} \tag{4.1}$$

Here, $a, i \in I$, $j, b \in J$, a is the activation function being considered, b is the output function being considered, and $l_{a,b}$ is the likelihood of a transfer function with an activation function a and output function b. $n_{a,b}$ is the number of times we find the combination of the activation function a and the output function b in a model (i.e $f_i(g_j) : i = a, j = b$) during the sampling process.

4.6.1.2 Associated Error

This measures the error associated with having the occurrence of a transfer function for any of the hidden or output units in elite neural network models. This essentially adopts the computation of the error associated with the immediate hidden units to the output units in back-propagation. It associates error to a transfer function by multiplying the weight

w_{ho} of the connection of the transfer function to the output unit $\delta(o_i)$. Thus giving an error associated with the transfer function. In back-propagation, this associated error is back-propagated again further back into the neural network model to associate an error to each weight according to its contribution. In this case, only the error associated with the transfer function is of interest and as such the error is associated with the transfer function.

The limitation however is that this is likely to have a noisy measurement of the performance of the transfer function in the model, as weights and biases can also contribute to the error. In addition to that, it is not known to what degree all these factors, including the chosen transfer functions and their parameters, contribute to the error. The hope is that this noise will be averaged out over a significant number of runs (20 runs of more is usually recommended for sampling in statistics). The associated error for each transfer function after evaluation represents the accumulated associated error of the transfer functions. Thus, it can be expressed as in Eq.4.2.

$$e'_{a,b} = \sum_i w_{ho}.\delta(o) \tag{4.2}$$

Here, $e'_{a,b}$ is the associated error for the transfer function (i.e., the combination of activation function a, output function b) being considered, w_{ho} is the hidden to output weight, and $\delta(o)$ is the error being back-propagated from the output unit for the pattern i.

4.7 Results

In this section, we present the results of the neural network ensembles on the mentioned data sets.

Results on popular benchmarks

The results produced were competitive with relatively less complexity.

There are a variety of reasons that might explain why this was so for the Australian credit card problem, which includes its relatively higher dimensionality (i.e input space being $\{x\}^{51}$ and sample size being 690). Problems of higher dimensionality in terms of their input space often require relatively complex solutions. This is because as the dimensions of a problem increase, solutions have to account for these new dimensions by making various decisions, including the decision of which subset of dimensions has more relevance to the prediction over others, and form decision boundaries through these dimensions that approximates the class boundaries as accurately as possible. As one can imagine, this problems gets consistently harder with increased dimensionality. In the case of NeuDiME, its

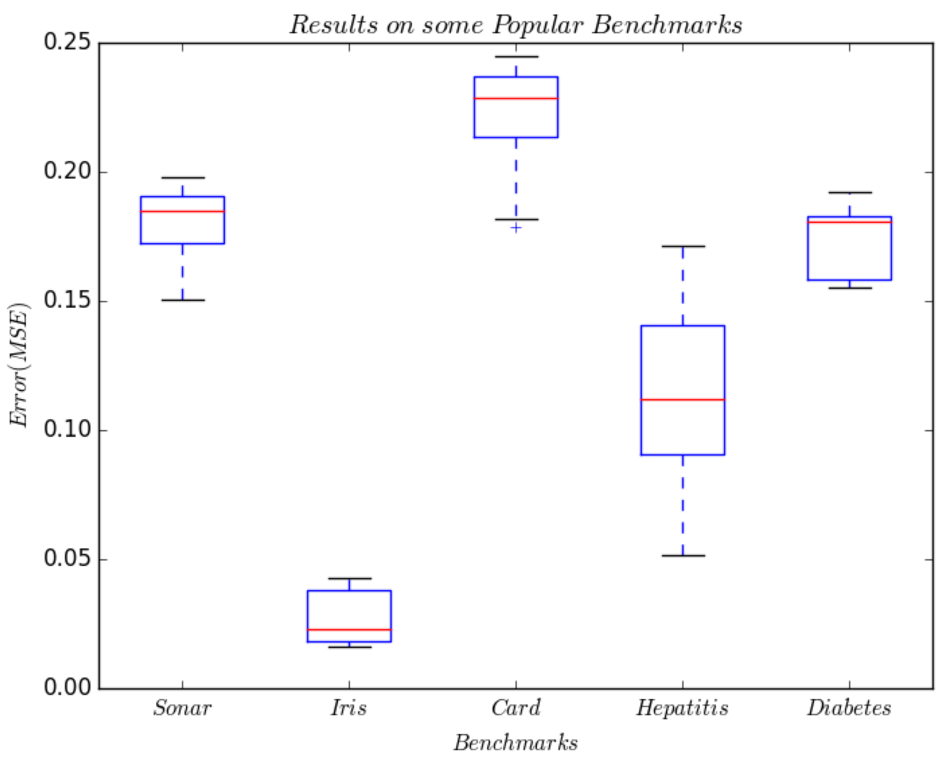


Figure 4.3: Results on the popular data sets for NeuDiME- The testing error was averaged over the 10-fold cross-validation results, except in the case of Australian credit card problem which was set to 12-fold cross-validation as used in the literature.

improved access to a wider variety of solutions comes with both potential for performance gain, such as it's statistical activation functions, which have been found to filter important features. However, at the same time, there is a trade-off. The advantage is that it increases its chances of finding computational strategies with creative hypotheses that describe the problem with simplicity and usually better generalization [55, 30, 31, 47, 71, 6]. The trade-off, if not handled carefully, is that this can present more local minima. In the case of the results for the Australian Credit card and Sonar, the dimensionality of the benchmarks is likely to have played some role in the slight performance difference; while the Card benchmark had 51 inputs, the Sonar had 60. In addition to having relatively more local minima, one can also expect that convergence might also be relatively slower as a result of the increased computational capacity.

4.8 Discussion

Interesting Computational Strategies Evolved for Prediction

In this subsection, we explore the statistics of the transfer functions used for some of the datasets used in the experiments. These include the likelihoods of each of the input combination functions and output functions being combined in the fittest members of the ensemble and the associated error with each combination. In addition to that, we also reveal some of the creative strategies used by the members of the ensemble towards solving the diabetes problem using these statistics.

Diabetes

The diabetes problem showed an emphasis on strategies that relied on combining standard deviation and an output function such as the identity function, or hyperbolic tangent. The least error was associated with the combination of Euclidean distance and Gaussian II, which is essentially a variant of a radial basis function unit with a wider center. This is because of the threshold parameter in Gaussian II which makes the boundaries of the peak response wider, as such a peak response can be given for a wider range of values. This was found to improve performance as presented by Maul [71]. Higher-order products combined with an identity function also showed a relatively lower associated error.

The statistics are presented in Table 4.10 and Table 4.11.

Table 4.10: Likelihood (%) of using combinations of input combination and output functions for the Diabetes problem - The most likely combination was standard deviation and identity, and standard deviation and hyperbolic tangent.

	Identity	Sigmoid	Gaussian	Hyperbolic Tangent	Gaussian II
Inner Product	0	0	0	3.57143	3.57143
Euclidean Distance	0	0	0	0.0	3.57143
Higher Order Product	3.57143	3.57143	0	0.0	3.57143
Higher Order Subtractive	0	0	0	0.0	7.14286
Standard Deviation	**17.85714**	0	0	**14.28572**	3.57143
Min	3.57143	7.14286	0	7.14286	0.0
Max	3.57143	7.14286	3.57143	3.57143	0.0

Table 4.11: Associated error of using combinations of input combination and output functions for the Diabetes problem.

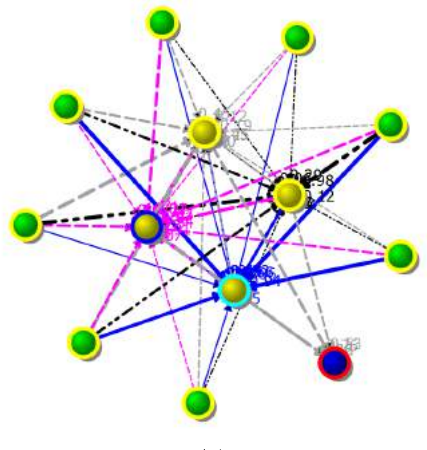
	Identity	Sigmoid	Gaussian	Hyperbolic Tangent	Gaussian II
Inner Product	-	-	-	0.01709	0.01161
Euclidean Distance	-	-	-	-	**0.00125**
Higher Order Product	**0.00210**	0.00644	-	-	0.01319
Higher Order Subtractive	-	-	-	-	0.02434
Standard Deviation	0.03476	-	-	0.05334	0.00405
Min	0.01357	0.00539	-	0.01066	-
Max	0.00500	0.02989	0.02224	0.00543	-

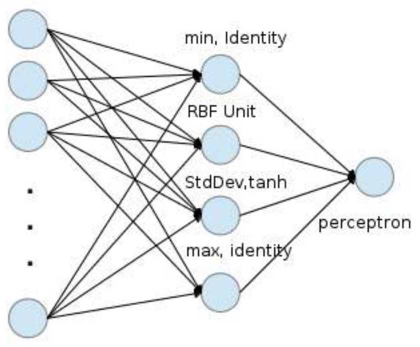
In the following discussion, we will present a study of two diverse and interesting computational strategies evolved by NeuDiME for the diabetes problem. This reveals the ability of NeuDiME to exhibit diverse neural computation strategies that are accurate.

One of the most accurate strategies evolved for the diabetes problem was a fully connected network consisting of four hidden units, implementing the following transfer functions found in Fig. 4.4(a)&(b).

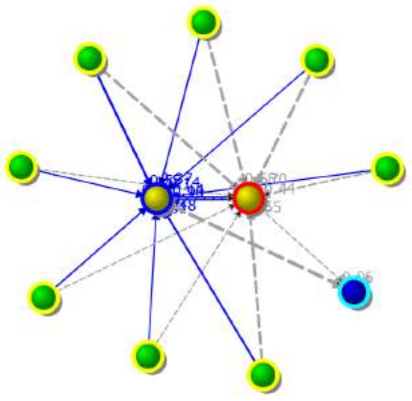
It seems that the range of the values for the features was a vital part of this strategy as it frequently made use of min and max activation functions. It is likely that these were being used as filters in this case to extract certain attributes values from the set of features. In other words, min and max functions were relaying information from an attribute that was probably highly correlated to the class. This might explain why it was being conveyed to the projection unit (i.e. the perceptron output unit), which evolved as the output unit. To understand why this is essential for this strategy, we compared the min and max of the raw data set, and the results were impressive; most of the time, the max value corresponds to the 2nd feature of the data set, i.e. *glucose concentration reading*. While the min value usually corresponded to either the 4th or the 5th feature, i.e. *skin fold thickness* and *serum insulin reading*, respectively. Based on the connection weights of these features to the hidden layers using min and max as a relay; the 4th feature (skin fold thickness) was given more weight as compared to the rest. The 2nd feature (glucose concentration) and 5th feature (serum insulin reading) were both given a medium weight, perhaps to normalize its values with the rest of the features, as they usually have the highest values. The most important feature relayed by these relay units based on their weights to them was *age*.

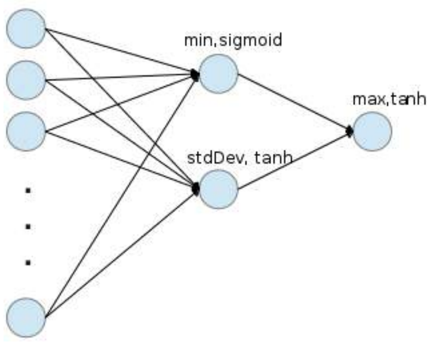
Interestingly, *age* and *skin fold thickness* are regarded as features that are recognized to be highly correlated to diabetes. The American Diabetes Association, for example, regards age as one of the leading contributing factors that increase the risk of a person having type-2 diabetes [10].


(a)



(b)


(c)



(d)

Figure 4.4: Visualizations of two models that evolved different strategies for the Diabetes problem.

In general, this strategy seems to be taking advantage of the variety of input combination function and output functions to extract relevant features using unusual and creative combinations of transfer functions to obtain useful information. These included obtaining the minimum feature value, the weighted variance between features, proximity of the feature vector to the center of the RBF unit, and maximum feature value, and finally using these features for training a simple perceptron in the output layer. In other words, the hidden layer in this case has been mostly used as a feature selection layer. The relatively lower dimensionality of these features is then taken advantage of to train a simple hyperbolic tangent perceptron to learn the problem.

Another strategy evolved was a fully connected network with two hidden units where one adopted a min input combination function and a sigmoid output function and the other adopted a standard deviation output function with a hyperbolic tangent output function (see

Fig. 4.4(c)&(d)). The output unit differed from the other strategy, consisting of a max input combination which has a winner-take-all effect on the hidden nodes. It's hyperbolic tangent output function has another normalizing effect.

Once again, there is a similarity with the earlier strategy in the use of the min input combination function. However, in this case, it is coupled with a sigmoid output function, which has a normalizing effect on the output value - restricting it between 0.0 and 1.0. In addition to that, in this case, it seems to act as a threshold for the other hidden node using the weighted variance. This is because of the choice of using max as the input combination function by the output unit, which would tend to pick the value from the hidden units with the highest output. In other words, the skin-fold thickness parameter was being squashed and used as a threshold for the other hidden unit, which seems to have adopted the standard deviation as an averaging mechanism.

In general, the normalizing effect of the output functions of both hidden nodes allows the hidden node using the min input combination to essentially emulate the role of a bias node that is dependent on the features.

4.8.0.3 Iris

Interestingly, the Iris problem emphasized on the utilization of a different set of input combination and output functions as shown by the probabilities of transfer functions used (see Table 4.12). This demonstrates that neuronal diversity can offer a variation of diverse computational strategies evolved specifically to problems with different characteristics.

On the Iris problem, the main strategies seemed to emphasize the use of a combination of Standard deviation input combination and Gaussian II activation functions. Given the way standard deviation works, this combination can be regarded as a distant relative of the radial basis function unit, which uses the difference between feature vectors and the mean value of the feature vector, instead of the canonical measurement approach of the Euclidean distance which uses the center vector for the radial basis unit.

Table 4.12: Likelihood (%) of using combinations of input combinations and activation functions for the Iris problem.

	Identity	Sigmoid	Gaussian	Hyperbolic Tangent	Gaussian II
Inner Product	0.000	0.000	0.000	**7.407**	3.704
Euclidean Distance	3.704	0.000	0.000	**7.407**	0.000
Higher Order Product	0.000	3.704	**7.407**	3.704	0.000
Higher Order Subtractive	3.704	3.704	3.704	0.000	0.000
Standard Deviation	3.704	0.000	**7.407**	**7.407**	**14.815**
Min	0.000	0.000	0.000	0.000	0.000
Max	**7.407**	3.704	0.000	**7.407**	0.000

Table 4.13: Associated errors of using combinations of input combinations and activation functions for the Iris problem.

	Identity	Sigmoid	Gaussian	Hyperbolic Tangent	Gaussian II
Inner Product	0.000	0.000	0.000	0.003	**0.001**
Euclidean Distance	**0.003**	0.000	0.000	**0.002**	0.000
Higher Order Product	0.000	**0.001**	**0.001**	0.001	0.000
Higher Order Subtractive	0.000	**0.003**	**0.002**	0.000	0.000
Standard Deviation	0.000	0.000	**0.003**	**0.003**	0.008
Min	0.000	0.000	0.000	0.000	0.000
Max	**0.007**		0.000	**0.002**	0.000

In general, the transfer function statistics (i.e. likelihood of occurrence and associated error) of the Iris and Diabetes datasets not only reveal some hint to us regarding which transfer functions are being used and what strategies are being evolved, but they also show that different problems require different strategies, and as such need an adaptable form of bias.

Discriminatory Property of the Transfer functions

Further experiments on the XOR problem, was used to conclusively show that the performance gain was mainly as a result of the diversity of transfer functions. XOR is a problem that is easy to analyze, as such it makes it an ideal problem for studying the computational strategies.

Transfer functions with strong discriminatory property were being evolved. One of the more interesting strategies was the use of Euclidean distance with hyperbolic tangent, which assuming the weights are $[1,1]$, results in an error of 0.26 on the XOR. Another interesting strategy seemed to be exploiting the property of parity to enhance its discriminatory property; this was the combination that used the combination of a max activation function, and a Gaussian output function. The MSE error for the transfer function alone, assuming the weights are $[1,1]$ computes to just 0.17. This is significant because a single transfer function is doing it without the cooperation of other neurons in the neural network. One can expect that the other transfer functions will only need to use a strategy that addresses the blind spots of these transfer functions to achieve outstanding performance.

This particularly highlights the ability of transfer function diversity to evolve diverse strategies for problems and their accuracy, which conclusively shows that neural diversity in the form of transfer function diversity is the main contributor to the performance here.

4.9 Conclusion

In this chapter, the application of hybrid neural networks in the field of ensembles was presented, and it was shown that neural diversity in an artificial neural network can exhibit diverse and accurate computational strategies as a result of neural diversity. This was followed up by further analysis which conclusively showed the strategies associated with the evolved transfer functions had strong discriminatory properties. Neural diversity was also shown to be effective in ensembles without the need of other explicit diversity maintenance measures that tend to be computationally expensive. Specifically, it was shown by revealing the two different strategies for the diabetes problem, where clever functions such as feature filters and feature normalizers were evolved for pre-processing. The filter function was used to extract important features such as age and skin fold thickness, which were found to be correlated to the risk of diabetes. Interestingly, these features are among the factors that increase the risk of diabetes as recognized by the American Diabetes Association (ADA) [10]. It was shown that this approach can evolve relatively small ensembles of compact networks that have a competitive performance, as seen in the case of the diabetes problem which had a small number of ensemble members of no more than 20, each with relatively fewer hidden units of no more than 5 hidden units, relative to those found in the literature which tend to have at least 20 members with 10 hidden units or more [79, 19, 82]. The results suggest that neural diversity not only shows the ability to significantly produce diverse computational strategies with diverse biases; it also shows that it can evolve creative solutions to complex problems. However, it also had limitations which include the increased local minima, as a result of the increased access to the global search space of computational strategies. In addition, relatively slower convergence is also a concern. However, considering the increased probability of generalization gains and efficiency (i.e. in terms of using limited resources), it is arguable that this significantly outweigh the drawbacks.

Chapter 5

Problem Signatures

In the previous chapter, we introduced neuronal diversity in the form of transfer function diversity and how it can produce accurate and diverse computational strategies for problems. We also showed that it was able to evolve computational strategies with strong discriminatory properties, which contributes significantly to the performance of the neural network. While doing so, we also showed how neuronal diversity helps with ensemble diversity.

In this chapter, we explore more details on ways of overcoming the dimensionality increase of the search space as a result of neuronal diversity. This increased dimensionality is significant for many reasons. It has the potential of having both positive and negative impacts, which directly affects generalization ability. As explained in the previous chapter (i.e. Chapter 4), as the size of the computational strategy space (or hypothesis space) for the learner expands, the variance-error increases as well - as a result of the high variation in models. This is mainly due to the increased complexity of the search space, which allows the models to adopt a wider range of hypotheses. Though this can be advantageous in cases where the most ideal hypothesis $h_i \in H_\ell$ is found as a result; the randomness of the weights, and transfer functions at initialization, in addition to the increased sub-optimal solutions (or local minima) makes it difficult for the neural network to be consistent in finding and possibly refining that hypothesis.

The primary objective of this chapter is to explore the feasibility of using measures in the previous chapter (i.e. transfer function likelihood and associated error) as well as others inspired by neuroscientific measures as meta-features. These can then be used for transfer function initialization, and other bias-adjustments before training. In other words, this is a presented as a pre-processing step. The neuroscience-inspired measure includes connection density, coexistence likelihood, and connection strength. Connection density measures the ratio of active connections to total connections to a transfer function (i.e. combination of an activation function $g_j()$ and output function $f_i(g_j()))$, it can indicate the importance of transfer functions in a computational strategy. Coexistence likelihood measures the likelihood of a pair of transfer functions being connected to each other. This measure also captures the direction of the connection, which helps in knowing the chain of computation being done on the inputs of the neural network. Finally, the connection strength measures the accumulated weight used between a pair of transfer functions. This also captures the direction of the connection and helps in knowing the degree to which the transfer function at the receiving ends, considers the outputs of any connected transfer function in its calculation.

The criteria used to qualify the generated results of the measures as meta-features were two: firstly, the ability to produce consistent results for a given problem, i.e. given a particular problem such as the iris, the results of the measures for that problem should be consistent. Secondly, the ability to produce different results for different problems, i.e. given a set of problem, the results of each should differ from the others. These measures were then used in studying the underlying computational strategies learned by the neural network models, to provide further evidence that conclusively shows their ability to produce diverse computational strategies.

The following introductory sections provide the motivation, literature and our contributions.

5.1 Motivation

Transfer function optimization can lead to an increased dimensionality of the search space, thus, exposing the neural network to more potential local minima [48]. In particular, approaches such as increasing transfer function flexibility by increasing its adjustable parameters [29, 41, 54, 38], and transfer function pooling [49, 55, 3, 61, 30, 31] have relatively larger search spaces as a result of the additional possibilities introduced.

One of the effects of the larger search space is that it affects the ability to consistently convergence on promising computational strategies, which affects generalization. The increased size of the search space introduces more local minima due to the increase of sub-optimal solutions. At which point, the starting points of the optimization algorithm are likely to vary more as a result of variability in the transfer functions of the neurons. Subsequently, one can expect there will be more variability in the consistency of convergence. The consistency of convergence will rely on the ability of the optimization algorithm to escape various local minima to converge on a promising computational strategy. The common approach in the literature is the use of evolutionary algorithms to evolve the artificial neural networks [65, 98, 34, 12, 34, 71, 77, 44, 46, 78]. However, there is a lack of literature that addresses ways of initializing other architectural components of neural networks apart from weights, such as transfer function choice, and topology.

In this chapter, we explore ways of exploring the neural network hypotheses (i.e. computational strategies) space before training to get an idea of the predisposed architecture for the neural network, and subsequently the promising computational strategy (h_i) more likely to be a fit for the problem. In particular, we use measures from the previous Chapter (i.e. associated error and transfer function likelihood), in addition to measures commonly used in the study of biological neural networks to artificial neural networks, to enable us to understand how transfer function preference varies between problems (i.e. discrimination). In addition, we also study how it can be consistent for the same problem (i.e. consistency). This also helps us to get a glimpse of both their underlying computational strategies and some insights into the problem. It also helps to lift the lid of complexity that hides the underlying workings of the neurally diverse artificial neural networks.

There is also the lack of literature related to model-based meta-features; the most closely related study used features of a trained decision tree [81] such as depth, to determine properties of the problem and fascinate the choice of the learning algorithm to be used. This work differs because the models are of neural networks for the purpose of directly initializing the neural networks.

5.2 Contributions

The main contribution in this chapter includes the novelty of application of the modified analytic techniques [96, 21, 91] used in neuroscience on artificial neural networks and the insights generated from it. Artificial neural networks tend to be used as black-box learning algorithms, just as their biological counterparts are also veiled in the mystery of their complexity, although recent decades have seen a significant acceleration in research findings [96, 18, 70, 69, 101, 99]. It is intuitive that techniques applicable in the study of biological neural networks be applicable -with modifications- in the study of artificial neural networks as well.

This was done by utilizing graph-theoretic measures for the purpose of further analyzing the results of the neuroscience-inspired techniques. It involved isolating the set of connections to the transfer function with the most connection density (i.e. the highest ratio of active to non-active connections), and visualizing these neural computational paths. Thus, capturing vital information such as the transfer function choice with the highest connection density, in addition to the sets of transfer functions which connect to it.

Additionally, we also propose a set of criteria for problem signatures and show that problem signatures can be consistent for the same problems and discriminatory between different problems as we illustrated in [7], thus making them a viable meta-feature specific to neurally diverse neural networks.

5.3 Chapter Overview

This chapter consists of various sections that explore the various properties of different types of problem signatures, in particular, their consistency and discriminatory ability. In addition to that, it also explores transfer function initialization and complexification, which are some of the applications of problem signatures in artificial neural networks.

The signatures section (section 5.4), defines problem signatures, and then further describes the classification of signatures. Specifically, lower-order and higher-order signatures. This is followed by the explanation of the criteria for signatures. In particular, it describes some of the properties necessary for a potential signature to exhibit to be considered a signature.

The next section (section 5.4.3), describes some of the tools used for analyzing these signatures. This includes the popular threshold function and another analysis tool derived from graph theory.

Afterward, the next section (section 5.6) presents some of the preliminary application results of lower-order signatures in initializing, and gradually growing the set of transfer functions made available for optimization. These were meant to improve convergence and generalization ability by managing the effects of the dimensionality increase as a result of transfer function diversity.

This is followed by sections (section 5.7 and section 5.8) that show that both lower-order and higher-order signatures meet our proposed criteria for problem signatures. Specifically, that the signatures of the different datasets tested have many neural computation paths in common, which makes them similar at a glance, but after a path analysis, their differences become more apparent and conclusive. In addition to that, we also show that both lower-order and higher-order signatures are consistent regardless of the size of the population used for sub-sampling (P), the level of noise (γ), or size of the set of elite models used to sample the signatures (N). We also show that increasing the size of this set (N), introduced new neural computational paths suggesting differences in the strategies of the elite models, which also suggests that transfer function diversity is also a viable method of introducing diversity in ensembles as highlighted in the previous chapter.

In this chapter, we refer to *models* as the neural networks that adopt the *hypothesis* or *computational strategies* from the Hypothesis space or Computational strategies space. We use the term *hypothesis space* and *computational strategy* space interchangeably to define the search space containing the various types of hypotheses and neural computational strategies available. We also use the terms *problem signatures* and *computational signatures* interchangeably.

5.4 Signatures

In this chapter, we refer to data gathered from measures in the previous chapter as well as our neuroscience-inspired measures as signatures. Preliminary results showed that these signatures seemed to be different between problems and show some consistency for a problem as highlighted in the earlier chapter, however these are approximations of the nature of problems. This is because the signatures are extracted from the neural network models trying to learn the problem, and the problem's nature is something that is unknown. Models are not always completely representative of the underlying strategy even after training. This is because a model doesn't always adopt a hypothesis $h_i(.)$ that completely minimizes the loss function with respect to the underlying function of the problem $f(.)$. There is always some error which is measured as the expected bias-error in the bias-variance decomposition i.e $E_D[(h_i(x) - f(x))^2]$. Therefore, the signature identification process is an effort to approximate of the nature of the problem. An analogy can be drawn from probability, where the probability according to a certain set of observations n from an infinite set of observations N ($n \subset N$) is usually described as a likelihood or relative probability. This is because it is an approximation of the probability that gets more accurate as more observations are sampled; thus, making it closer to the *true* probability of the event.

Problem signatures are essentially meta-features of neural network models proposed in this thesis for sampling from the three major components of the neural network, i.e. weights, topology and transfer functions. These helps in analyzing the underlying computational strategies of the sampled models. In our experiments, the sampled models are from a population of randomly generated neural networks consisting of various topologies, weights, transfer functions and hidden layer size.

The process of signature extraction is a sampling process (see 2), which samples only from the set of elite neural network models. Measures classified into higher-order and lower-order (i.e. first and second-order) signatures are used to sample features of the fittest models, which indirectly are hints to the nature of the problem. Lower-order signatures measure the statistics on the components of transfer functions, specifically the pair of activation $g()$ and output functions $f()$. Higher-order signatures measure statistics of a pair of transfer functions.

Algorithm 2 The process of signature extraction.

```
for $i$ : $maxSamples$ do
    $Pop \leftarrow initPopulation()$
    $Pop \leftarrow evaluate(Pop)$ {evaluate train and validation errors}
    $Pop \leftarrow sort(Pop)$ {sort the population by their training and validation errors}
    $sample_i \leftarrow topN(Pop)$ {select top $n$ for signature extraction}
    $signature \leftarrow extract(sample_i)$ {extract both lower and higher-order signatures}
end for
```

In this following sections, lower and higher-order signatures used for the experiments and the methods of sampling them are described (see also Table 5.1). Subsequently, a set of criteria for problem signatures is also described.

Table 5.1: Problem signature types and visual representations.

Order	**Signature**	**Visualization**
Lower-Order	Transfer function Likelihood	see Fig. 5.1
	Associated Error	-
	Connection Density	see Fig. 5.2
Higher-Order	Coexistence Likelihood	see Fig. 5.3
	Connection Strength	see Fig. 5.3

5.4.1 Lower-Order Computational Signatures

In the previous chapter, we introduced two measures used for analyzing the computational strategies evolved using neural diversity. In this section, we classify as well as describe the nature of these signatures and other variations of lower order signatures.

5.4.1.1 Transfer Function Likelihood

The transfer function likelihood (or likelihood of occurrence - as described in the previous chapter) captures the likelihood of each transfer appearing in the best neural network model, either before or after training. A high likelihood of a certain transfer function to exist in the best model can be regarded as an indicator that that transfer function provides access to a region of the search space that might have the most appropriate hypothesis. In other words, the transfer functions might be more suited for the given problem.

This was described as the number of times a transfer function appears in the best model normalized by the total number of possible transfer functions as in Eq. 4.1 on page 62. This gives an relative likelihood of the transfer function appearing in the best models $E[f_a(g_b(.))] = l_{a,b}$. Given the role of transfer functions as the components that play the important role of forming decision boundaries, we will expect that the likelihoods of each of the transfer functions for a certain dataset D, will give us some hints regarding the nature of the problem. In addition to that, it also gives us a hint of the underlying computational strategy ($h_i(x)$) adopted by the best neural network model, which could then be used to refine the choice of transfer functions among other possibilities.

We consider two conceivable forms of capturing this information; the first approach was to obtain the likelihood of the activation function or in other another case the output function irrespective of their combination. In other words, the likelihood of an activation function $g(.)$ like the Euclidean distance will be captured from the neural network model without regard for the output function $f(.)$ it is combined with. We refer to this as a *disjoint* signature extraction, denoted as *first-order* problem signature of the transfer function likelihood. This is because the measure is essentially capturing statistics of the decoupled components of the transfer function.

The second approach was to capture the likelihood of transfer functions as a unit, in other words the likelihood of all the possible combinations of activation and output functions. We referred to this as *joint* signature extraction, and is classified as *second-order* problem signature of the transfer function likelihood . This is recorded in a matrix with the x-axis representing the output functions, while the y-axis represent the activation functions. As such the value for the likelihood of a transfer function of a certain activation function

$g(.)$, and a certain output function $f(.)$, is given by the value found at the index of activation function $i(g(.))$, and output function $i(f(.))$, i.e $(i(g(.)), i(f(.)))$ (see Fig. 5.1).

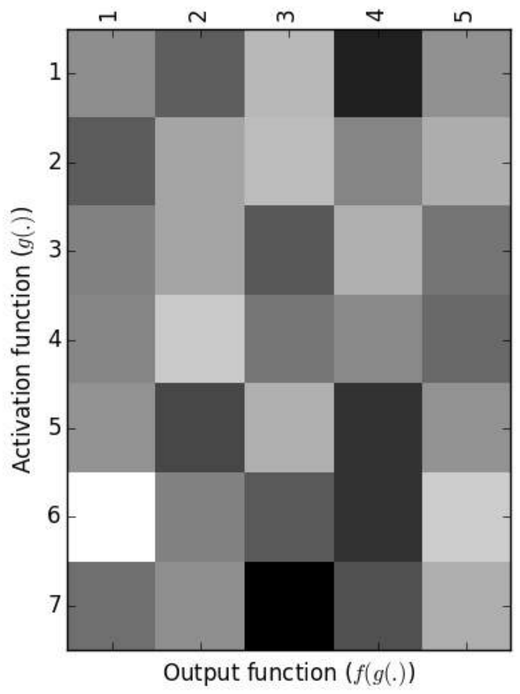


Figure 5.1: The transfer function likelihood visualization for the Iris problem showing the relative expectations of the transfer functions to be used in elite models. The intensity represents the degree of usage with dark and light signifying heavy and light usage, respectively.

5.4.1.2 Associated Error

While the transfer function likelihood captures the likelihood of transfer functions being used, the associated error captures a different aspect. The associated error measures the error associated with a transfer function. It does so by back-propagating the error of the output unit on a pattern x_i to each hidden unit. The associated error e' for each hidden unit is then calculated as in Eq. 4.2 on page 63, and this is associated with the transfer function of the hidden unit. This would have required a different approach in the case of a neural network with multiple layers of hidden units since the error of the neurons from the deeper layers depends on differentiating and then propagating the error backward. However; in our case, the neural network had one hidden layer, i.e. two layers of weights.

The associated error is averaged over several runs (samples) to get an approximate error that should be somewhat representative of the expected error for the transfer function.

The associated error also has disjoint first-order, as well as joint second-order problem signatures.

5.4.1.3 Connection density

Connection density is one of the measures used in neuroscience for studying biological neural networks [96, 91, 21], and was adopted for studying neural diversity machines. It is calculated for each neuron in the hidden and output layers as the sum of the active (i.e. turned on and takes part computation) connections to the neuron, normalized by the number of all the connections (both active and inactive). This can be expressed as in Eq. 5.1.

$$d_i = p_i / q_i \tag{5.1}$$

Where q_i and p_i, are the total number of connections, and the total number of active connections for neuron i, respectively. This is accumulated over the number of sampling runs for signature extraction. Figure 5.2 shows the example of an illustration of the connection density for the Iris dataset.

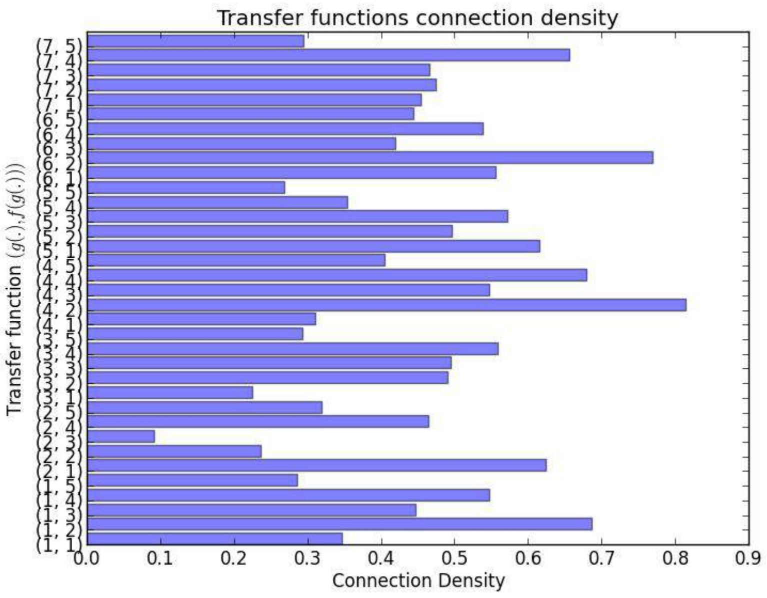


Figure 5.2: An illustration of the connection density for the Iris dataset.

5.4.2 Higher-Order Computational Signatures

Another analysis tool used in neuroscience to understand biological neural networks is graph theory [96, 91, 21]. Regions of interest in the brain are defined, and their structural or functional connectivity are represented in a two-dimensional connectivity matrix.

Each intersection in this matrix could then represent the presence of structural or functional connectivity.

We adopt this graph theoretical analysis for discovering the signatures of problems in our neural network models. This produces two sorts of matrices: coexist-on-path matrix (or the coexistence likelihood), and connection strength matrix. These were classified as higher-order signatures (HOS).

Higher-order computational signatures capture information of a higher-level representing statistics, such as the likelihood of transfer functions either being connected to each other (on a path) or coexisting in the same model. Examples of these signatures include the *coexistence likelihood*, and *connection strength*. In our experiments, the coexistence likelihood captures the likelihood of transfer functions being connected to each other on a path, while connection strength captures the cumulative weights of those connections (see Fig. 5.3). The direction of the connections is also captured, where the y-axis represents the origin of the path, and the x-axis represents the receiving end of the path. As expressed earlier, the transfer function is represented as a tuple (a,b) where, a and b are indices of the activation function and the output function, respectively (see Table 4.1 and Table 4.2). Thus, a connection from $(7,2)$ on the y-axis and $(1,2)$ on the x-axis represents information about a connection from a transfer function consisting of a *max* and *sigmoid* connecting to another consisting of an *inner-product* and *sigmoid*. As with the transfer function likelihood, darker regions indicate higher values, while lighter regions indicate lower values.

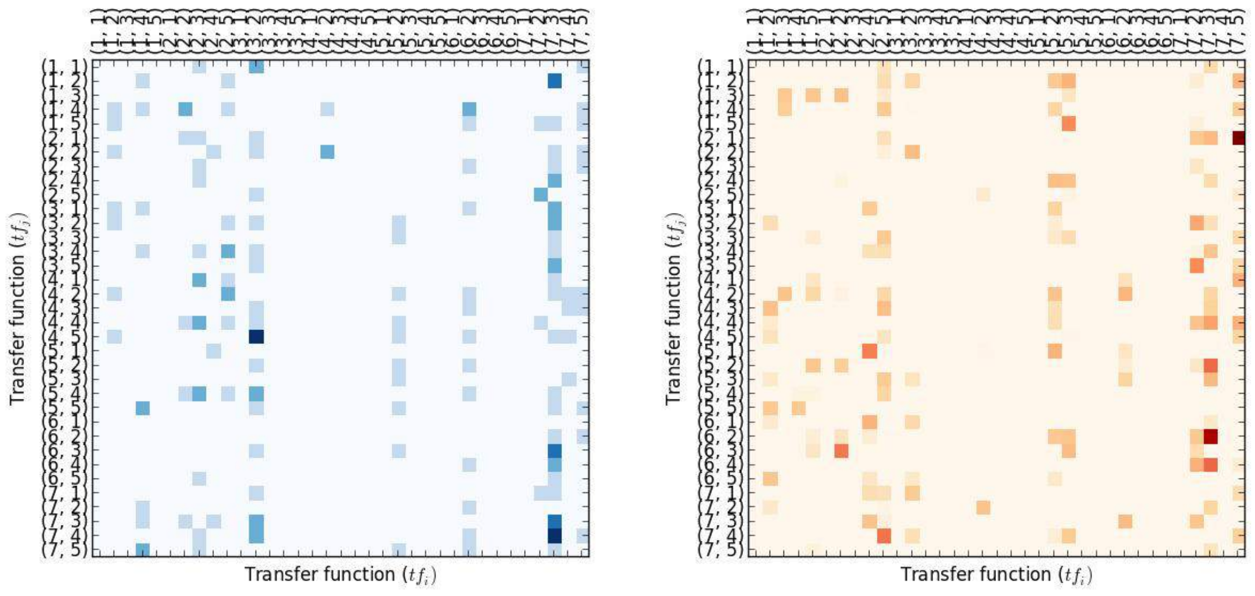
The coexistence likelihood can be expressed as in Eq. 5.2.

$$E[p_{x,y}] = \frac{n_{x,y}}{N} \tag{5.2}$$

Here x,y are both transfer functions in consideration, $n_{x,y}$ is the number of times we encounter the transfer functions x and y, and N is the total possible transfer functions. $E[p_{x,y}]$ is then the expectation of the coexistence likelihood of transfer functions x and y on a path $p_{x,y}$. In the case of connection strength (see Eq. 5.3), we are mainly accumulating the connection weights between the transfer functions.

$$c_{x,y} = \sum w_{x,y} \tag{5.3}$$

Here, $w_{x,y}$ represents the connection weight between the transfer functions x and $y.c_{x,y}$ is then the connection strength between the transfer functions in consideration. This was not normalized in practice as it yielded very small values that were hard to work with.



(a) **(b)**

Figure 5.3: Higher-order problem signatures on the Iris dataset : (a) coexistence likelihood, (b) connection strength.

5.4.3 Signature Criteria

Upon inspecting the visualization of the coexist-on-path and connection strength matrices (see Fig. 5.3(a) and Fig. 5.3(b)), there are signs of a consistent pattern that emerges in the form of some strongly shaded vertical regions. These show some of the neural computation paths most likely to be found in the fittest sub-sample of neural network models. The question is if these were conclusively consistent and if they differed between unrelated problems. This pattern of consistency was also seen in the other forms of signatures described.

In summary, the criteria for computational signatures to be feasible as a reliable description of the computational strategy for problems are as follows:

- Consistency of signatures belonging to the same problem.
- Difference between signatures belonging to unrelated problems, i.e. discriminative ability.
- Similarity between signatures of problems that are related.

In our work, we only address the first two criteria, i.e. the consistency and discriminatory properties of signatures. This is because the third criterion of proving that signatures of similar problems should be similar requires a reliable measure of similarity between problems in the problem space. However, this extends beyond the scope of this thesis.

5.5 Signature Analysis Visualization and Techniques

In this section, we highlight the tools and techniques used for showing the consistency and discriminatory property of signatures. These include thresholding, neural computation path analysis, and correlation measures (i.e. Pearson and Spearman correlation coefficients).

5.5.1 Thresholding

Thresholding is one of the key tools that was used in the analysis of the computational signatures. It is a standard technique used in various analyses including the study of data from graph theoretical analyses of the brain [96]. It is valuable in revealing relatively notable features by filtering out some that are below the threshold.

In our work thresholding was done to filter out the values that were below the mean value to reveal information that could be of interest. The mean (μ) was chosen because it is relative to the range of values being considered. Thus, it makes it unlikely to filter out some information that might be important or not filter out signatures that are not useful.

We also used another adjustment parameter (α), which can be used to increase the threshold expressed as in Eq. 5.4.

$$T = \mu + \alpha \tag{5.4}$$

Here T is the threshold, μ is the mean of the given data (e.g. coexistence likelihood matrix), and α is the adjustment parameter.

5.5.2 Neural Computation Path Analysis

Though computational signatures present a holistic view of the underlying strategies being used for a problem by various models using their meta-features (i.e. likely neural computation paths and their connection strengths), it is hard to examine the most prevalent building blocks common to these models for specific problems. There is a need for a method that combines the results of some of these signatures to present a simplistic and informative representation of the signatures. As such, we use the neural computational paths found in the higher-order signature to reconstruct portions of these strategies. We define neural computational paths as the connections between neurons (i.e. transfer functions) in the neural network, which transforms data using these paths. Thus, a path between two transfer functions f_x and f_y could be expressed as (f_x, f_y).

This approach involves the use of a graph-based analysis of the neural computational paths from the resulting higher-order signatures (i.e. coexistence likelihood and connection strength). The analysis involved using the neuron (transfer function) with the most connection density from the signatures as a reference for recreating the most likely portion of the strategy by visualizing the transfer functions that were most likely to connect to it. This was reproduced in the form of a graph, with the edges representing the connections, and the intensity of the edges representing the connection strength of the edges. Only connections with the most likelihood to connect to the reference neuron (i.e. with the most connection density) relative to the other transfer functions connecting to it were used in the recreation. This was done by using the average (μ) as the threshold, which allows for a more adaptive form of thresholding; thus, showing only those that were above average.

The steps of the neural computational path analysis were as follows:

- Find the transfer function $f_{terminal}$ that had the highest connection density.
- Add it to a directed graph (di-graph) G.
- Trace back the transfer functions connected to it $T_{in} = \{f_i ... f_n\}$, where T_{in} refers to set of transfer functions sending inputs to the terminal transfer function (i.e. T_{in}).
- Find the mean connection strength for all the transfer functions from the set $m_{connStrength}$.
- Remove transfer functions from the set T_{in} with connection strengths lower than $m_{connStrength}$
- Add the transfer functions to the graph G and connect them with the $f_{terminal}$ node/vertex.

The resulting graph is a reconstruction of the most likely neural computational paths connecting to the neuron with the highest connection density, which gives an empirical expectation of the primary building blocks for the neural computation strategy with the fittest score for the problem. These building blocks further reveal hints of the underlying computational strategy predisposed to the problem.

In the context of showing that the signatures are consistent for problems, and indicating the differences between signatures for different problems, path analysis is especially useful. This is because demonstrating that the primary building blocks for the signatures are consistent regardless of changes to factors such as the size of the population used for signature extraction, or the number of independent samples collected (i.e. runs R), conclusively shows that they are consistent under those conditions. Similarly, if different problems exhibit different building blocks, then they can be conclusively shown to be discriminatory for problems.

5.5.3 Measuring Differences between Signatures

Measuring the similarity between signatures was done using correlation measures such as the Pearson correlation coefficient, and the Spearman correlation coefficient built into the Python SciPy library [57]. This is especially useful for evaluating the differences in correlation between signatures and was used to conclusively verified with results from the neural computational path analyses. This has also been used to verify the findings of the discriminatory ability of higher-order problem signatures in conjunction with thresholding.

The Pearson correlation coefficient is expressed as follows:

$$r = \frac{\sum_i \sum_j (\mathrm{A_{ij}} - \mathrm{E[A]})(\mathrm{B_{ij}} - \mathrm{E[B]})}{\left[\sum_i \sum_j (\mathrm{A_{ij}} - \mathrm{E[A]})^2\right] \left[\sum_i \sum_j (\mathrm{B_{ij}} - \mathrm{E[B]})^2\right]} \tag{5.5}$$

Where A and B are $m \times n$ matrices representing higher-order signatures or second-order signatures such as the transfer function likelihood. Expectation of A ($E[A]$) and B ($E[B]$) are defined as:

$$\mathrm{E[A]} = \begin{bmatrix} \mathrm{E[A_{1,1}]} & \mathrm{E}[A_{1,2}] & \mathrm{E}[A_{1,3}] & \dots & \mathrm{E}[A_{1,n}] \\ \mathrm{E}[A_{2,1}] & \mathrm{E}[A_{2,2}] & \mathrm{E}[A_{2,3}] & \dots & \mathrm{E}[A_{2,n}] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[A_{m,1}] & \mathrm{E}[A_{m,2}] & \mathrm{E}[A_{m,3}] & \dots & \mathrm{E}[A_{m,n}] \end{bmatrix} \tag{5.6}$$

and

$$\mathrm{E[B]} = \begin{bmatrix} \mathrm{E[B_{1,1}]} & \mathrm{E[B_{1,2}]} & \mathrm{E[B_{1,3}]} & \dots & \mathrm{E[B}_{1,n}] \\ \mathrm{E[B_{2,2}]} & \mathrm{E[B_{2,2}]} & \mathrm{E[B_{2,3}]} & \dots & \mathrm{E[B}_{2,n}] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathrm{E[B}_{m,1}] & \mathrm{E[B}_{m,2}] & \mathrm{E[B}_{m,3}] & \dots & \mathrm{E[B}_{m,n}] \end{bmatrix} \tag{5.7}$$

$E[A_{ij}]$ - the expectation of a neural computation path ij, where i and j are both transfer functions, and $E[A]$ is a simple average/expectation of the elements of A.

5.6 Transfer Function Initialization and Complexification

5.6.1 Motivation

Artificial neural networks capture information about the nature of the problems within their structure; including weights, connectivity patterns and the functional parameters of the transfer functions. These can be harvested as meta-features and help in not only deciding the choice of transfer functions that might be more appropriate for any given problem but also to reveal their underlying computational strategies. In the previous section, we described some of these measures which were described as problem signatures.

The most related work regarding capturing information from the features of models is from the meta-learning domain [84, 81, 102, 100]. In particular, the most similar work is one which uses the features of decision trees (e.g. depth) trained on a problem to determine its characteristics. The features were then used to identify the classifier that is most suitable [81].

This work contributes to the field by exploring the use of signatures as meta-features of neural network models to enable us to understand the underlying hypothesis that they learn, and identify possibilities of more suitable starting positions in the search space, i.e. initialization. However, initialization does not always guarantee the most appropriate starting position due to limitations of the signatures, especially concerning the size of the sample, as well as noise as a result of parameters with high dimensionality such as weights. It may give rise to a restrictive bias since the set of transfer functions is determined by the signatures (or meta-features), and is a subset of the transfer function set used for our neurally diverse networks. As such, there is also a need for a method of gradually expanding the search space by introducing other transfer functions into the set of transfer functions previously not included. This is referred to as *transfer function complexification*.

The decision of the choice of transfer functions for the neurons of artificial neural networks is not apparent. Earlier works [47] have raised the concern of the lack of a standard method of using radial and projection functions in neural networks within the context of the theory of duality of functions. There have also been reports of techniques that use k-means clustering to identify the number of clusters as the number of radial basis functions units. This is followed by initializing the rest of the hidden units with projection basis functions [47]. Though this can be applicable in the context of neural networks dealing with only radial basis functions and projection basis functions alone; it does not completely apply to the problem of neural diversity. This is because of the nature of other transfer functions included in the set such as Higher-order transfer functions.

Transfer function complexification is one approach which could be argued to be biologically plausible. Complexification is one of the biological design patterns found in biological neural networks, especially during the development stage [24]. The idea is to regulate the complexity of the search space by introducing the transfer functions in stages. One of the advantages of this approach is that the optimization algorithm will be exposed to a fraction of the local minima since it's focusing on a relatively small region of the search space in the early stages of optimization. Subsequently, after giving some time for the neural network to convergence on a local minimum, a new set of activation and output functions is introduced into the transfer function set. This has the effect of introducing more species of computational strategies to be considered, and this is the second advantage which might help in escaping local minima.

There are various possibilities for initializing transfer functions. In this section, we explore three possibilities of detecting the meta-features of neural network models (i.e. DSI, JSI, RBFI, and All), which can be used for identifying and possibly initializing the transfer functions of artificial neural networks having a diverse transfer function set. We also explore the effects of two complexification techniques on all the three techniques (i.e. DSI, JSI, RBFI, and All). The first method involved the random injection of transfer functions into the pool of transfers not included in the set used for optimization. This is likely to have the effect of randomly introducing sub-spaces into the search space of the neural network. This concept was explored in a later chapter where it is shown to result in dramatic performance improvements. The second was the fitness-based introduction of transfer functions whereby the associated error signature of the problem with respect to the transfer function set, was used to choose which transfer functions ought to be introduced before the others. In other words, the transfer functions are ranked by their associated error and introduced in sequence. The injection of transfer functions was done on an interval basis for both techniques.

In relation to the literature, the related works include NEAT [97] which explored complexification in an evolutionary approach by adding hidden units to grow a neural network and adapt its topology. However, in this section, the approach is different. It finds promising transfer functions for the neural network model using either of the three techniques, uses the promising subset for initialization, and gradually introduces the other members of the promising transfer function set into the pool of available transfer functions for the neural network models in either a fitness-based or randomly manner. This enables us to verify if there might be any advantages in complexification if it is based on some prior knowledge, i.e. fitness in this case. This differs from work from the literature by introducing an approach towards increasing the complexity of artificial neural networks while retaining its size. This approach also doubles as a method of controlling the complexity of the search space by systematically expanding the search space in search of strategies that might be more promising. The resulting effect is the ability to escape local minima.

5.6.2 Experimental Setup

In this subsection, we present the experimental setup. This includes the datasets used, and the respective maximum number of hidden nodes stipulated in addition to other details.

The experiment consisted of two stages; the first stage was to form the set of initial transfer functions used for the neural network before the optimization process. This involved sampling the signatures and using the first three most likely set of transfer functions as the initial transfer function set. The second process was the training process where the initial set of transfer functions was then used to optimize the neural networks, and depending on the experiment, additional transfer functions are introduced at intervals during the optimization process.

The general idea of sampling signatures is not new and Duch [37] has suggested a similar method as a way of selecting the most suitable transfer functions for different problems. It is basically to generate a vast and diverse population of neural network models using all the transfer functions, then to evaluate them without training on the problem and to select the fittest model. This can be repeated for many times until an apparent pattern in the use of transfer functions and associated errors is noticed. In the case of this experiment, the results were averaged over 30 trials and a population size of 1000 neural network models, of which the top solution is selected, and its meta-features are sampled.

In terms of the optimization, all the experiments were allowed to run for 100 generations, even though there was a stopping criteria that terminated a trial when the target error of 0.0 was met. The setup for the datasets was as follows:

Table 5.2: Experimental setup for the benchmarks showing the maximum number of hidden units allowed for each ensemble member, and the number of folds used for K-fold cross-validation.

Benchmarks	Max Hidden units	Members(Ensemble)
Iris	5	10
Sonar	5	10
Diabetes	5	10
Double Spiral	15	N/A
XOR	5	N/A

Double Spiral In this case, the networks were allowed to adopt a larger number of hidden nodes. The networks could adopt anywhere between 1 and 15 hidden nodes for this problem. This was setup to match the number of hidden nodes used by other works for easier comparison. The double spiral did not use cross-validation, instead, it was trained using 300 data points uniformly sampled from the 2D image of the double spiral. To get a reliable measure of generalization ability, another 300 data points were randomly sampled from the same 2D image to serve as the test set. The results were averaged over 30 runs.

XOR Finally, in the case of the XOR dataset, the maximum number of hidden nodes stipulated was the same as that of the Iris, Diabetes, and Sonar datasets. However, because this dataset was relatively smaller. The training set and test set were both the same, and the results were average over 30 runs as was the case for the double spiral. In other words, this did not produce a test error.

The following are descriptions of experimental conditions relating to means of initializing the set of transfer functions before optimization.

All (Baseline) This condition was used as the baseline for the rest of the cases in the experiments. It was implemented to use all the available activation and output functions of the neurally diverse neural networks throughout the optimization process.

Joint signature set Initialization (JSI) This case consists of using joint problem signature detection to determine the transfer functions predisposed to the given problem and uses a subset of them as the initial set of transfer functions usable. This set is gradually grown by selecting a random activation function and output function from the transfer function set at intervals.

Disjoint signature Initialization (DSI) Like the previous case, this case uses problem signatures to determine the predisposed transfer functions to the given problem and uses a subset of those with the highest likelihood of occurrence as the initial set of transfer functions usable. However, this case uses the disjoint method of problem signature detection. Likewise, the usable set of transfer functions is grown in the same manner as in the previous case.

Radial Basis Functions Initialization (RBFI) In this case, Radial Basis Function network's transfer functions are used as the initial transfer function set. This set is then grown in the same way as the previous cases (i.e. DSI and JSI).

The following are descriptions of the experimental conditions relating to the gradual introduction of transfer functions during optimization, i.e. transfer function complexification.

Joint signature Initialization with Random Complexification (JSI-R) This case is an extension of JSI which starts with an initial set that has only one transfer function (i.e. one activation function, and one output function). The set is grown by selecting a random activation function, and an output function at intervals during the optimization process. It differs from JSI because it uses a smaller subset and has a relatively smaller interval.

Disjoint signature Initialization with Random Complexification (DSI-R) This extends the DSI by starting with an initial set that has only one transfer function. The set is grown in the same way as the previous case (i.e. JSI-R). Likewise, this also differs from DSI by its use of a smaller subset at initialization and has a relatively smaller interval.

Joint Signature Initialization with Fitness-Based Complexification (JSI-FB) This case is also another extension of the JSI and starts with an initial set that has only one transfer function. However, the set is grown by selecting the next activation function and output function based on its likelihood of occurrence in elite solutions from the joint problem signature detection stage.

Disjoint Signature Initialization with Fitness-Based Complexification (DSI-FB) Likewise, this case also extends the DSI and also starts with an initial set that has only one transfer function. Like the previous case (i.e. JSI-FB) it grows the usable transfer function set based on associated fitness values.

5.6.2.1 Measures

The measures were as follows:

- Mean squared Error: This is the usual error measure used in most of the literature on machine learning, sometimes the mean absolute error is also used. The mean squared error is expressed as in Eq. 5.8.

$$MSE = \frac{\sum_i^N (y_i - t_i)^2}{N} \tag{5.8}$$

- Cross-validation: It is used as a more reliable measure of generalization ability. In this work, we used k-fold cross validation. It works by shuffling, then dividing the dataset into k portions defined by the number of folds (k). Each portion i is then used as a testing set in one of the rounds $r_i \in \{1..K\}$ during the evaluation process. This helps to avoid cases of a biased assessment as a result of uneven class distributions favoring a particular model. The mean squared errors for the all the rounds $\{r_i \ldots r_k\}$ is then averaged to get an approximation of the expected generalization error.

5.6.3 Results

Transfer Function Initialization

The results that follow show the performance of the cases: All, JSI, DSI, and RBFI on the given datasets.

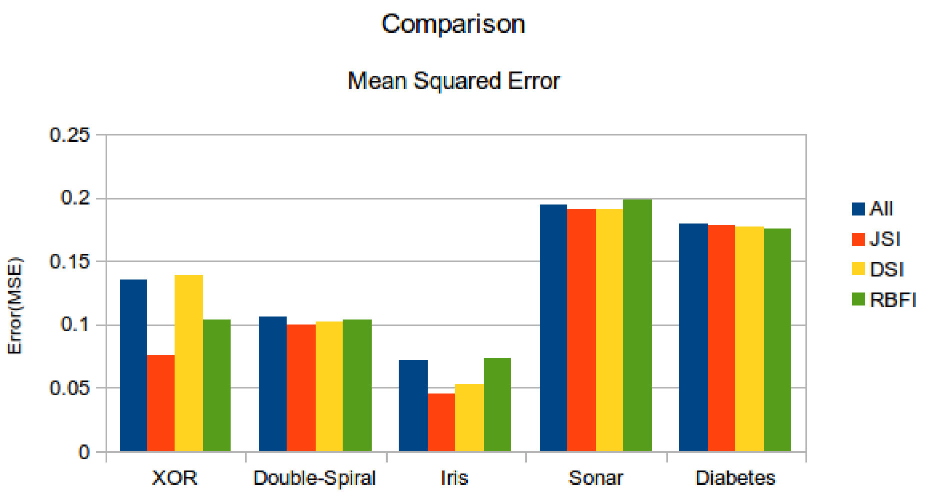


Figure 5.4: Bar chart of the mean squared error for the various initialization methods.

The results of the Iris and Sonar datasets indicate that the JSI and DSI were not significantly different (see Fig. 5.6 and 5.6), and this was regardless of the complexification methods used. Both show improvements over All and RBFI. The exception was in the case of the XOR dataset where the JSI performed better than the DSI both regarding error and convergence (see Fig 5.5 and 5.6).

For the Diabetes and Double spiral datasets, there was no statistically notable improvement (as suggested by a t-test) for the other cases (i.e. JSI, DSI, RBFI) with regards to the All condition in terms of cross-validation error (see Table 5.3 and 5.4). However, regarding convergence ratios, while JSI and RBFI were better off for the Diabetes dataset, the DSI condition was better off for the Double spiral dataset.

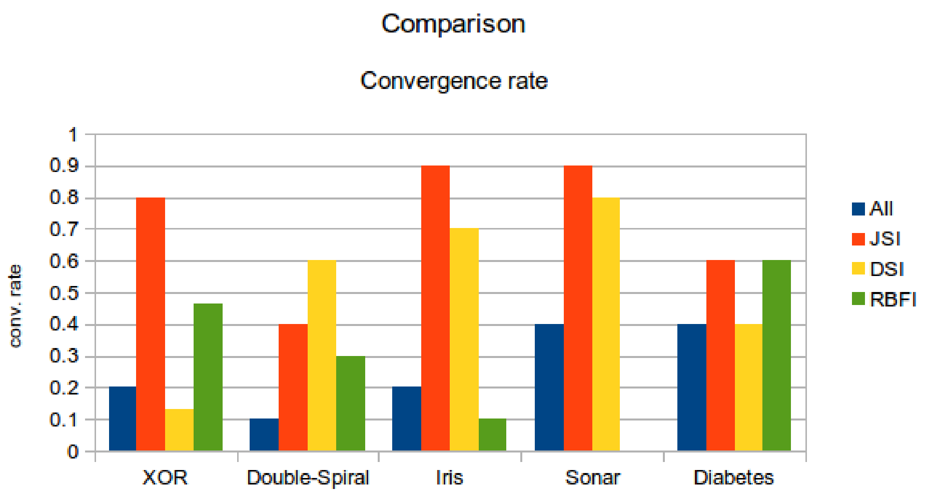


Figure 5.5: Bar chart of the convergence rate for the various initialization methods.

Table 5.3: Results comparing some of the cases.

	XOR	Spiral	Iris	Sonar	Diabetes
All	0.135	0.105	0.071	0.195	0.178
JSI	**0.076**	**0.100**	**0.045**	**0.191**	0.179
DSI	0.139	0.102	0.053	**0.191**	**0.176**
RBFI	0.103	0.103	0.073	0.199	**0.176**

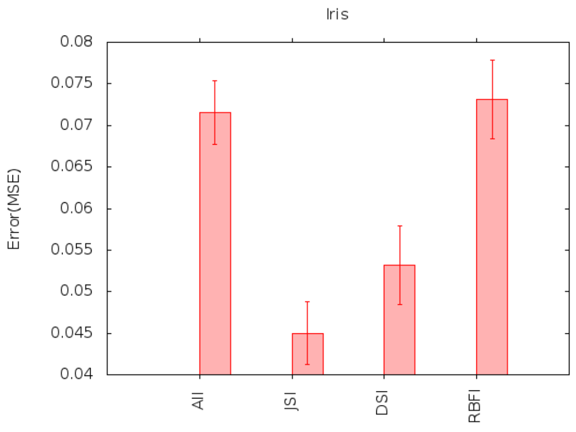
In general, JSI performed better than the other approaches (see Fig. 5.5 and 5.6) - especially in terms of convergence. Sometimes this was significant such as in the case of the Iris, XOR, and Sonar, and other times not - such as for the Diabetes and Double spiral datasets.

Transfer Function Complexification

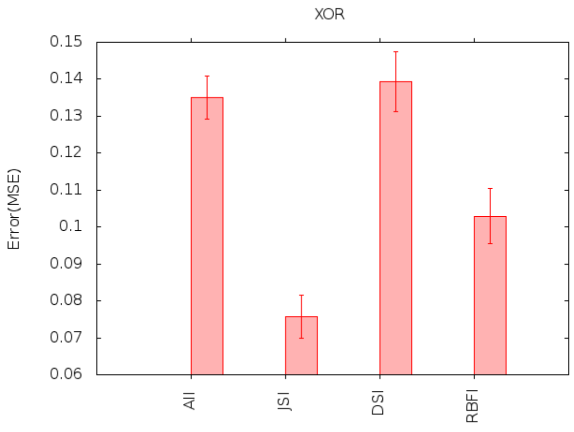
In the case of the XOR problem, the DSI performed better than the JSI with random complexification applied on both. This was the same for the fitness-based complexification case which also showed that they were not statistically different from each other for the given dataset.

However, although this was true for the Diabetes dataset regardless of the complexification scenario it was different in the case of the Double-spiral (see Table 5.4).

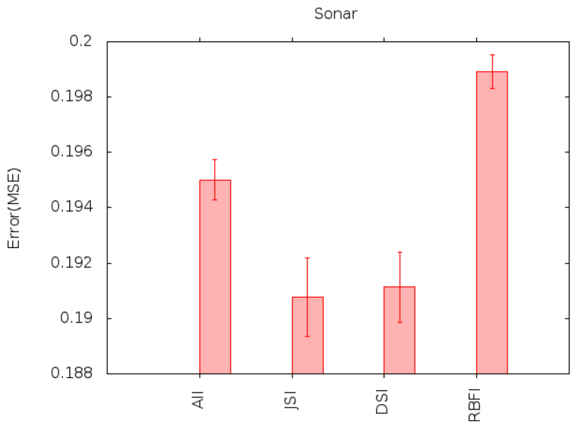
Interestingly, in the case of the double spiral dataset; the JSI was better than the DSI with random complexification (i.e. JSI-R and DSI-R) with a statistically significant difference. However, and interestingly this was not the case with the other complexification method, i.e. fitness based: in this case, the JSI-FB and DSI-FB were similar.



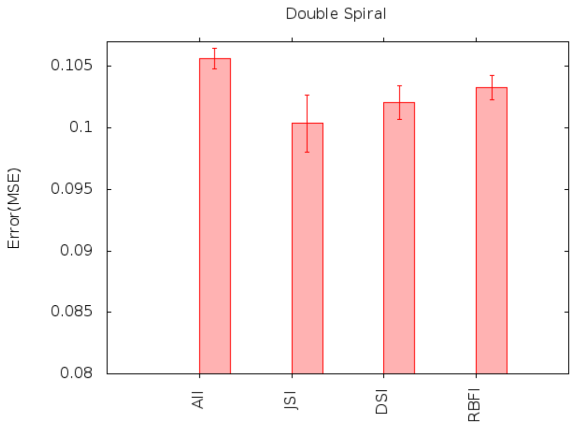
(a) Iris



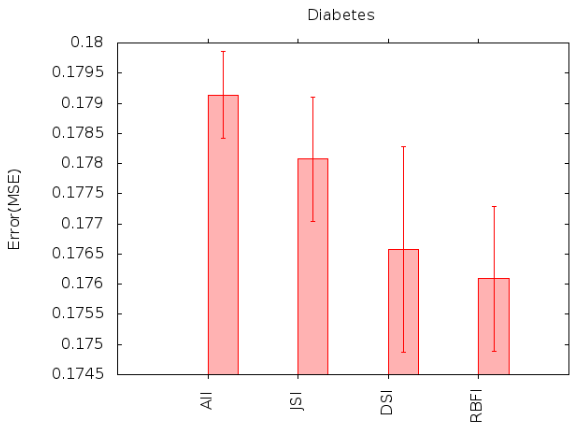
(b) XOR



(c) Sonar



(d) Double Spiral



(e) Diabetes

Figure 5.6: Mean squared error of the various approaches tested on the datasets.

In general, the complexification approaches did not show any statistically significant difference. Though there we two cases (i.e. XOR and double spiral) which showed slight

Table 5.4: Results of the complexification approaches applied with the problem signature detection methods.

	XOR	Spiral	Iris	Sonar	Diabetes
JSI-R	0.145	**0.105**	**0.057**	**0.191**	**0.177**
JSI-FB	0.128	0.105	0.068	0.197	0.179
DSI-R	**0.123**	0.109	0.064	**0.191**	**0.177**
DSI-FB	0.127	0.105	0.069	0.196	0.180

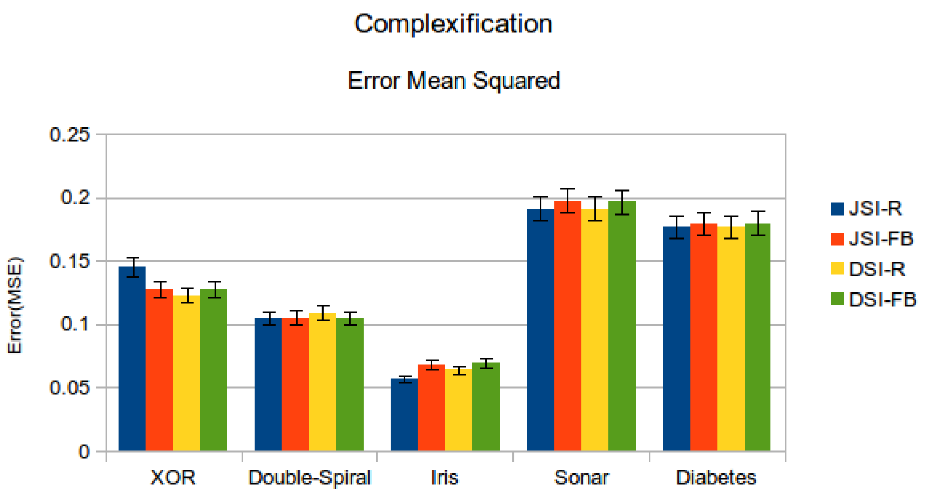


Figure 5.7: Bar chart of the mean squared error for the various methods of transfer function complexification.

differences between the methodologies. This could have been as a result of the interval based introduction of transfer functions which might have been too fast. In such a case, there will be no time for the neural network to converge on a hypothesis (local minimum in that subspace).

5.6.4 Discussion

Transfer Function Initialization

The Joint signatures and disjoint signatures, in general, don't seem to be statistically different from each other in terms of improving generalization error; however, JSI does help in improving the convergence rate for the neural network models. One explanation for the reason behind this is that both disjoint and joint signature information could be argued to both capture similar information. They are both essentially capturing the likelihood of transfer functions occurring in the elite models of the neural network, which is expected to represent the most optimal local minima within the set of hypothesis available to the learner ℓ. Capturing the input combination and output function's joint likelihood is similar

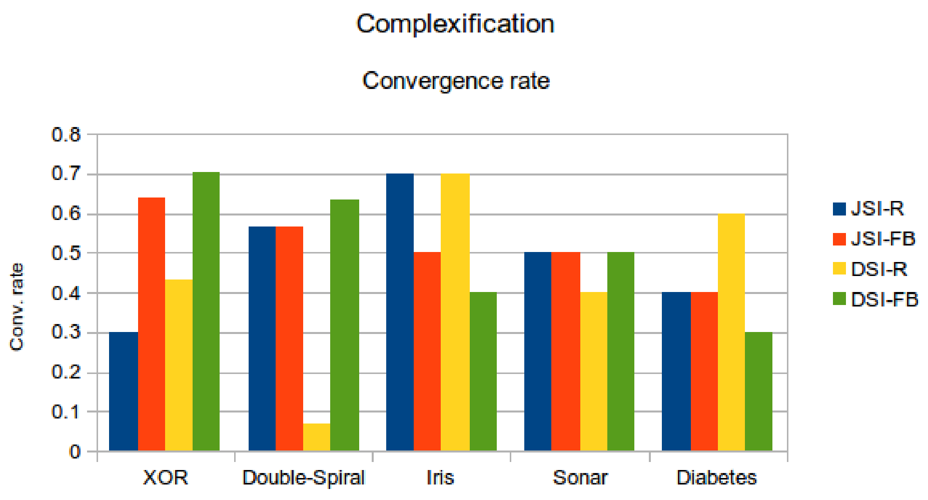


Figure 5.8: Bar chart of the convergence rate of the various methods of transfer function complexification.

to obtaining the likelihood of each of them separately since they are being sampled from the same source (i.e. the best model). This is because the increase in the joint likelihood of capturing a certain input combination function with another output combination function also translates as an increase in the likelihood of that input combination. In other words, they are somewhat proportionate.

However, it is important also to point out that they are not completely the same, as the convergence rate results illustrate: the additional information captured by the joint probabilities seems to help in improving the convergence. This is arguably a consequence of the signature providing more information about the starting point, which gives a head start to the optimization process.

Transfer Function Complexification

The results seem to suggest that regardless of introducing transfer functions according to their associated fitness (i.e. some prior knowledge), or just doing it randomly; there is no significant difference. This is quite profound in the sense that it is not intuitive. There are a variety of reasons that could be argued for the performance of the complexification methods being the same. We believe that one contributing factor could be that intervals need to be adaptive to other factors in the optimization process such as convergence, and stagnation. Complexification could be more useful when there has been stagnation in the training error over a long number of generations. It is likely to be less so when the neural network models are converging on a hypothesis and are making small adjustments to approximate it. This is because diversity in the species of solutions, or global spread of the search space is found to act usually as an antagonistic pair to convergence. This has been the motivation for works such as some hybrid algorithms that use both global and local search. We believe that in the case that the interval for introducing the transfer function is not optimized according to other variables such as convergence, stagnation, and spread over the search space, it's unlikely to yield different results regardless of if it's fitness-based or just random. This is because if the interval was faster relative to the convergence rate, such that new transfer functions push towards more exploration, the advantages of the relatively smaller search space in the early stages will be lost very quickly.

5.6.5 Conclusion

In terms of the initialization results, we have shown that disjoint or joint signatures are not statistically different with regards to improving the generalization of the neural networks. This because they are both measures that sample the same information in different ways and were bound to produce proportionate results. It was pointed out that with regards to convergence, their results were not similar and that the additional information captured by the joint signatures is useful in this regard. This was explained to be a result of the increased information regarding the starting position of the neural network, which might give it a head start in optimization.

We have also explored the complexification of an initial set of promising transfer functions using two methods; one using prior information such as fitness, and the other based on randomness to introduce transfer functions gradually into the search space of artificial neural networks. It has been found that both methods didn't result in significant differences except in the case of the double spiral where JSI-R was better than DSI-R in these experimental conditions. It has been suggested that these results were likely due to the interval not being adaptive to the stage of the search, and possibly being too fast relative to convergence ratios as a result.

5.7 Consistent and Discriminatory Properties of Lower-Order Signatures

In this section we explore the consistent properties of lower-order signatures, in particular: connection density and transfer function likelihood.

5.7.1 Motivation

In this section, we perform analysis meant to reveal if consistency is part of the properties of lower-order signatures; in particular, connection density and transfer function likelihood. There was the concern that because lower-order signatures captured information on a finer scale, and might be unable to capture any pattern, which could be explained with regards to the under-sampling common to other domains such as image processing. This section specifically seeks to explore the ability of lower-order signatures to be consistent for a problem, and discriminatory between different problems.

Another motivation is that there have been relatively few works in the meta-learning domain [81, 84, 15] in developing meta-features for learners in general, and even fewer for works that derive meta-features from models [81]. Unlike other works, this work presents a meta-feature for artificial neural networks that use neural diversity as their transfer function optimization approach.

5.7.2 Experimental Setup

The datasets used for the experiments in this section consisted of the Iris, Sonar, Diabetes and XOR datasets. The Iris, Diabetes, and Sonar datasets were retrieved from the UCI machine learning repository [13].

The signature extraction process remained the same as described earlier. The computational signatures were extracted from a subset of the population, in particular, the top N solutions after sorting. Independent samples of these signatures were accumulated over a number of runs R. For various conditions of the experiments, these parameters (i.e. R and N) were varied to examine their effects on the consistency of the signatures. In this section, we are mainly interested in lower-order problem signatures. The Mean squared error was used to evaluate the neural networks. The populations consisted of not only neural networks with diverse transfer functions but also of varying complexities (i.e. hidden layer size), which provided a broader set of regions from the search space to be sampled.

In summary, this section seeks to answer if the following affected the signature consistency:

- Size of the of fittest N solutions collected for signature extraction.
- Noise level injected into the training examples (γ).

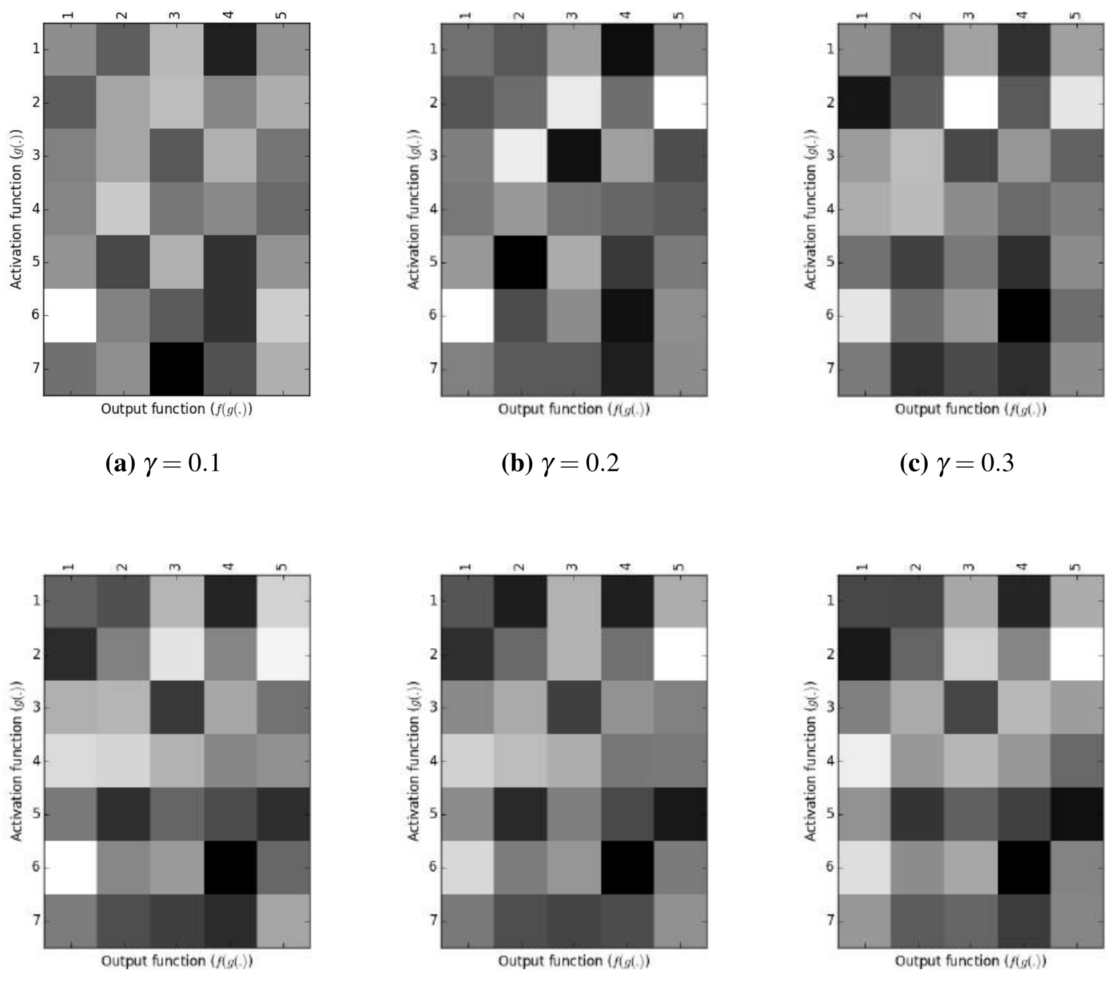
As such, the conditions for the experiments were as in Table 5.5.

Table 5.5: Conditions under which the lower-signatures were tested.

Condition	Description
Size of fittest N	Tests for various sizes of $N \in \{1...4\}$ solutions to be selected after sorting
Noise levels γ	Tests for some range of runs $\gamma \in \{0.1...0.9\}$.

5.7.3 Results

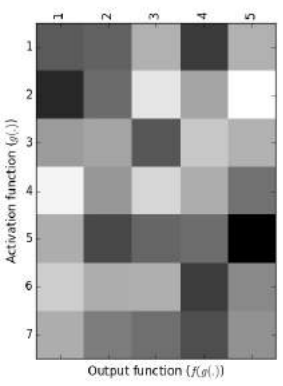
In this section, we present the results for the Iris, Sonar, and XOR. These included the lower-order signatures with varying levels of noise and sizes of N. The results of the connection densities can be found in the appendix (see Appendix A.1.1); likewise, the results of the effects of the sizes of N has been moved to the appendix (see Appendix A.1.2).



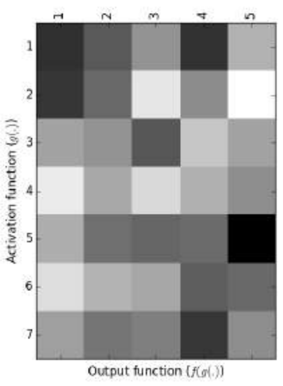
(a) $\gamma = 0.1$ **(b)** $\gamma = 0.2$ **(c)** $\gamma = 0.3$

(d) $\gamma = 0.4$ **(e)** $\gamma = 0.5$ **(f)** $\gamma = 0.6$

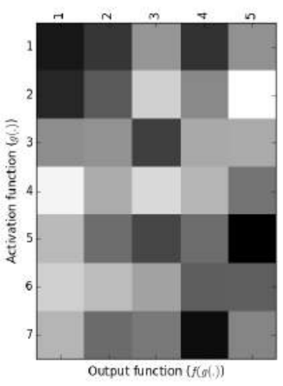
Figure 5.9: An illustration showing the average transfer function likelihood for the noise levels within the range of $\gamma \in \{0.1..0.6\}$ on the Iris dataset.



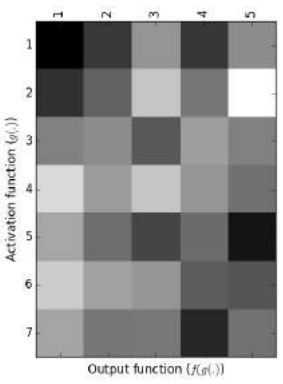
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$



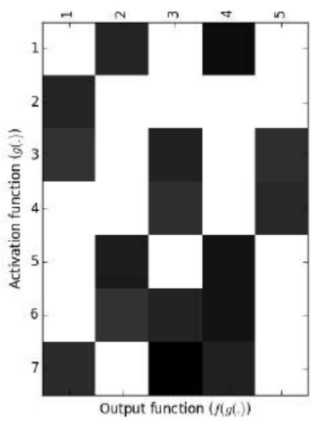
(c) $\gamma = 0.9$



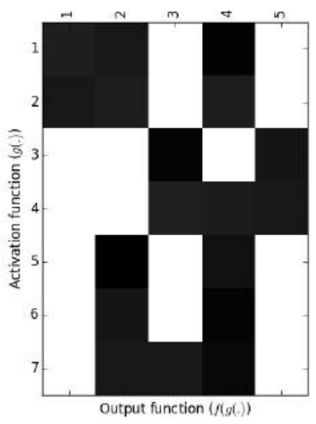
(d) $\gamma = 1.0$

Figure 5.10: An illustration showing the average transfer function likelihood for the noise levels within the range of $\gamma \in \{0.7..1.0\}$ on the Iris dataset.

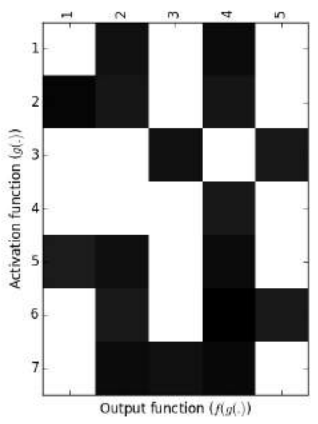
Fig. 5.9 and Fig. 5.10 shows the average transfer function likelihood for the levels of noise $\gamma \in \{0.1...1.0\}$. It was apparent that some of the transfer functions were more likely in comparison to other possibilities. In particular, the combination of the $(7,4)$ - which according to the indices of the transfer functions refers to the combination of a *max* activation function (7), and a hyperbolic tangent output function (4). This has been found to work as both a filter mechanism that extracts and normalizes a given feature of the problem as discussed in the earlier chapter.



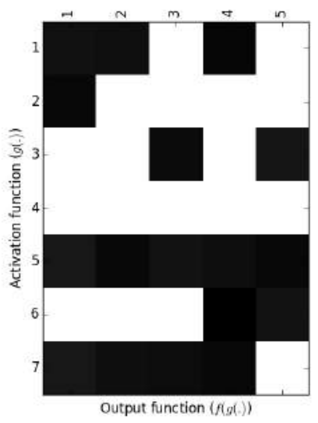
(a) $\gamma = 0.1$



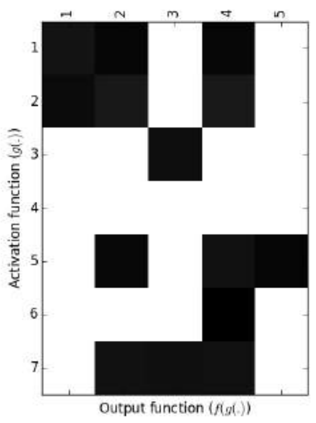
(b) $\gamma = 0.2$



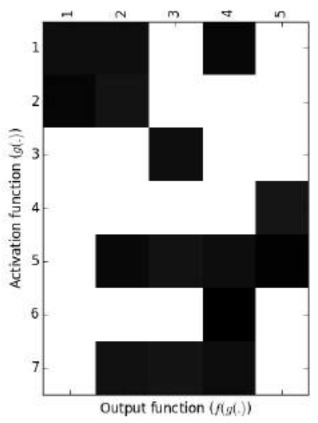
(c) $\gamma = 0.3$



(d) $\gamma = 0.4$

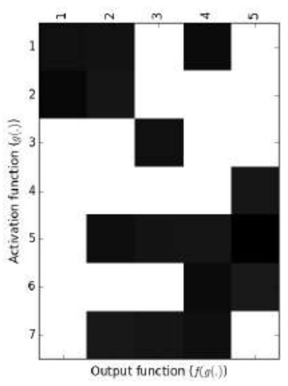


(e) $\gamma = 0.5$

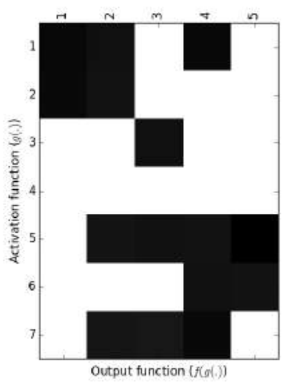


(f) $\gamma = 0.6$

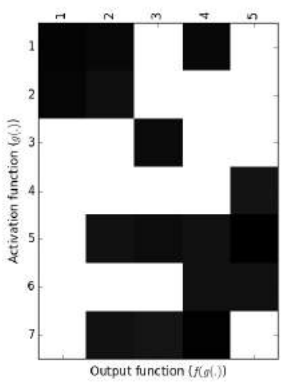
Figure 5.11: The average transfer function likelihood after thresholding for the noise levels within the range of $\gamma \in \{0.1..0.6\}$ on the Iris dataset.



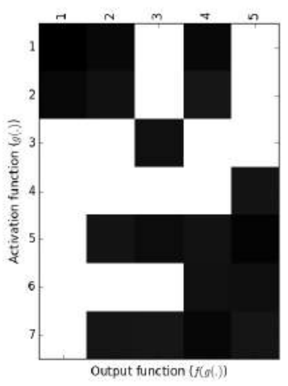
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$



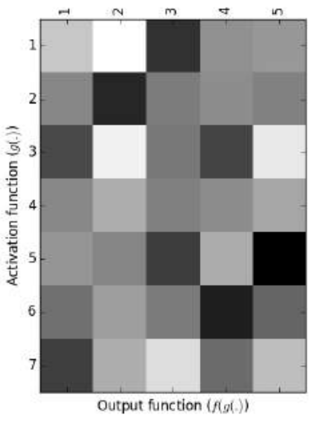
(c) $\gamma = 0.9$



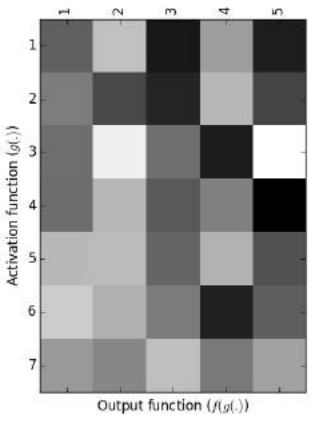
(d) $\gamma = 1.0$

Figure 5.12: The average transfer function likelihood after thresholding for the noise levels within the range of $\gamma \in \{0.7..1.0\}$ on the Iris dataset.

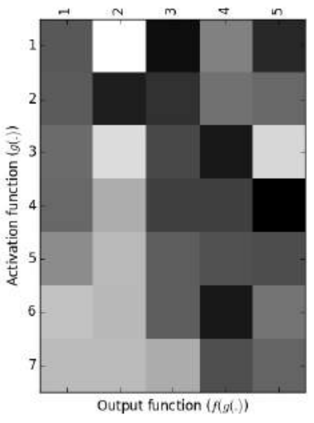
Fig. 5.11 and Fig. 5.12 shows the results after thresholding was applied to the signatures in Fig. 5.9 and Fig. 5.10. After thresholding, more consistent patterns of the transfer function likelihoods were apparent.



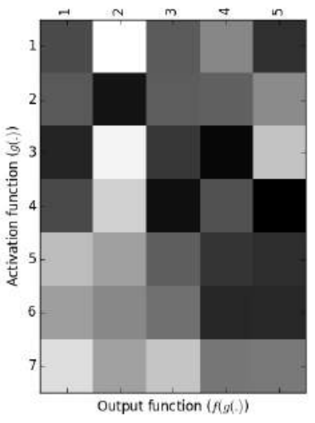
(a) $\gamma = 0.1$



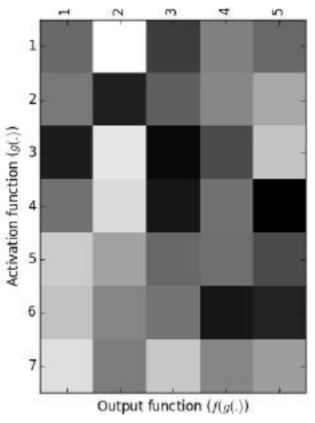
(b) $\gamma = 0.2$



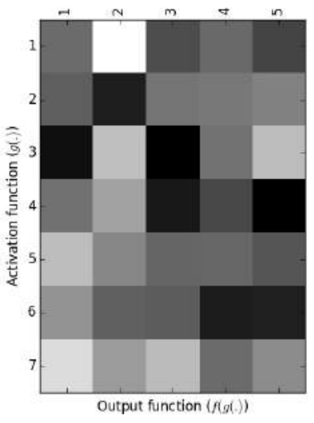
(c) $\gamma = 0.3$



(d) $\gamma = 0.3$

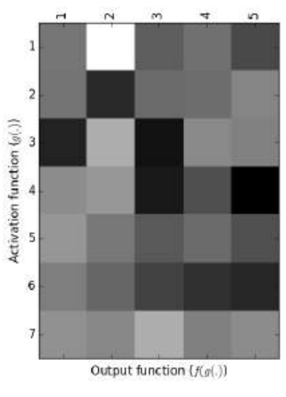


(e) $\gamma = 0.5$

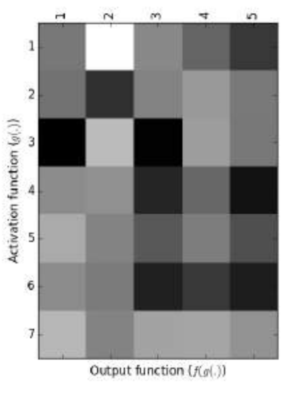


(f) $\gamma = 0.6$

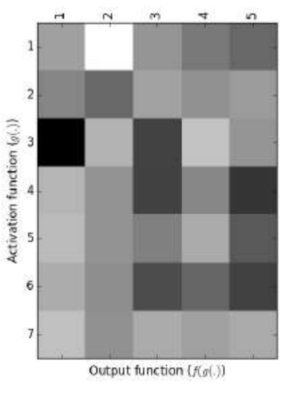
Figure 5.13: The average transfer function likelihood for the noise levels within the range of $\gamma \in \{0.1..0.6\}$ on the Sonar dataset.



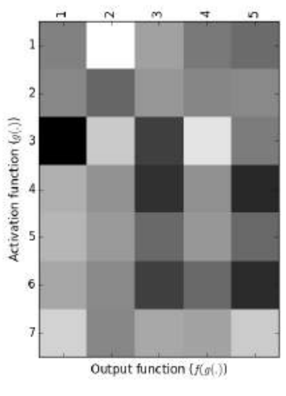
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$



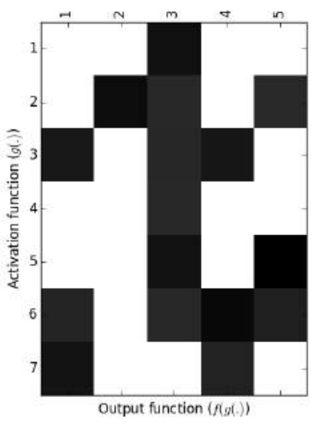
(c) $\gamma = 0.9$



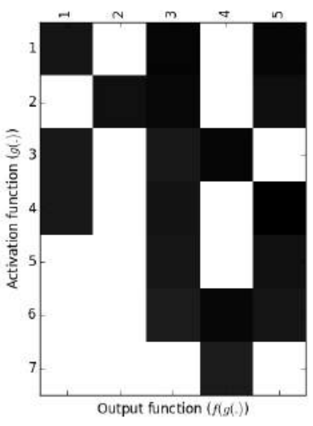
(d) $\gamma = 1.0$

Figure 5.14: The average transfer function likelihood for the noise levels within the range of $\gamma \in \{0.7..1.0\}$ on the Sonar dataset.

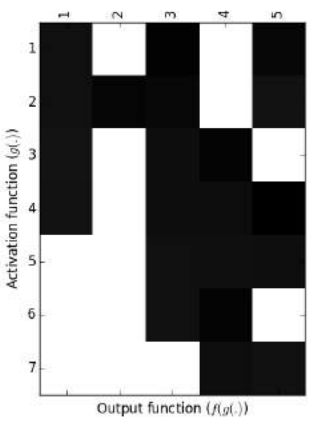
Similarly, the transfer functions likelihoods for the Sonar dataset shown in Fig.5.13 and Fig. 5.14 have some apparent patterns, however not completely obvious.



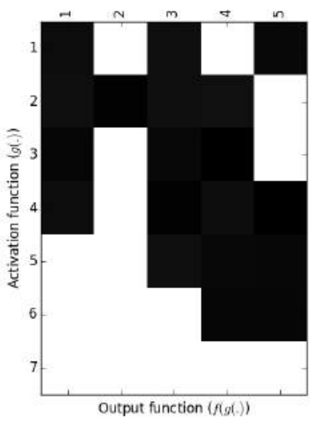
(a) $\gamma = 0.1$



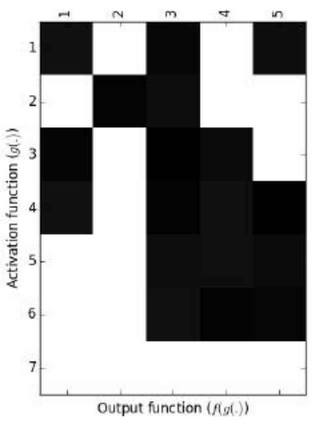
(b) $\gamma = 0.2$



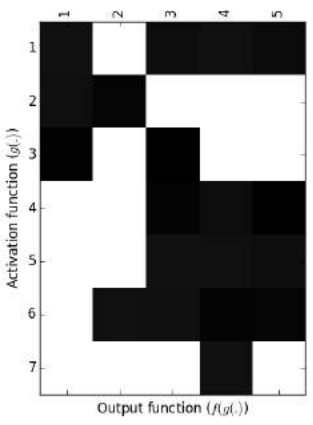
(c) $\gamma = 0.3$



(d) $\gamma = 0.4$

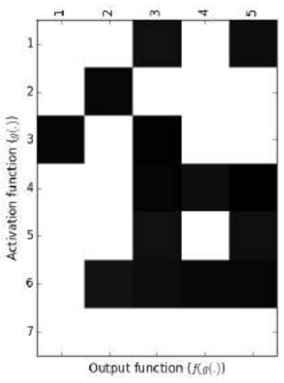


(e) $\gamma = 0.5$

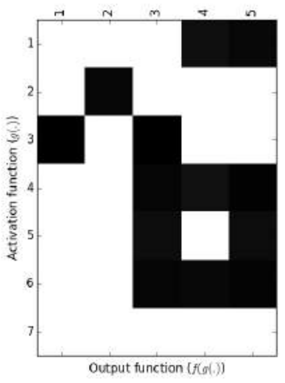


(f) $\gamma = 0.6$

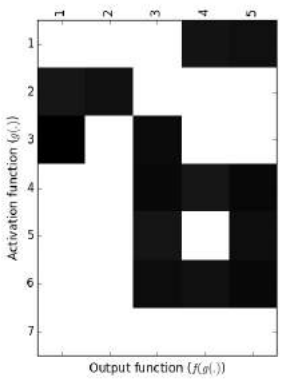
Figure 5.15: The average transfer function likelihood after thresholding for the noise levels within the range of $\gamma \in \{0.1..0.6\}$ on the Sonar dataset.



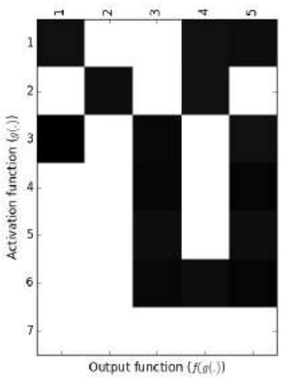
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$



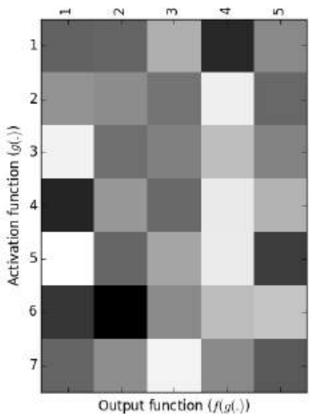
(c) $\gamma = 0.9$



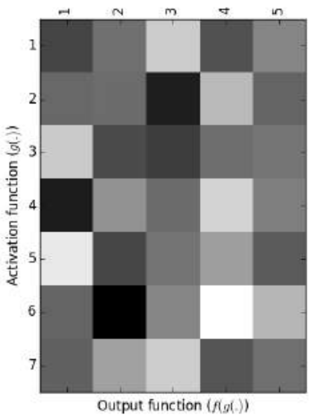
(d) $\gamma = 1.0$

Figure 5.16: The average transfer function likelihood after thresholding for the noise levels within the range of $\gamma \in \{0.6..1.0\}$ on the Sonar dataset.

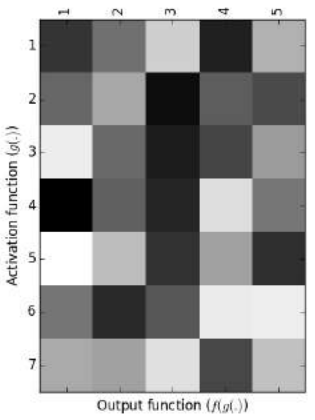
Interestingly, as shown by the more obvious and consistent patterns of likely transfer functions for the Sonar with increasing levels of noise (see Fig. 5.15 & Fig. 5.16); one of the likely transfer functions is the combination of a *standard deviation* activation function and a *hyperbolic tangent* output function. This is one of the transfer function that has been found to be used as a mechanism for relaying a normalized average of features as highlighted in the earlier chapter. This was considered to be particularly useful for the diabetes problem as well.



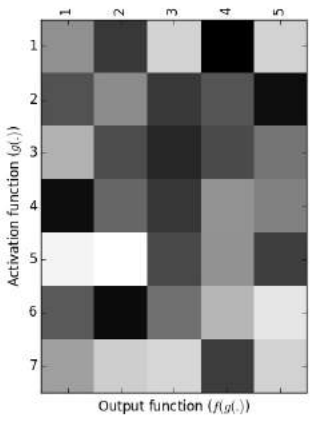
(a) $\gamma = 0.1$



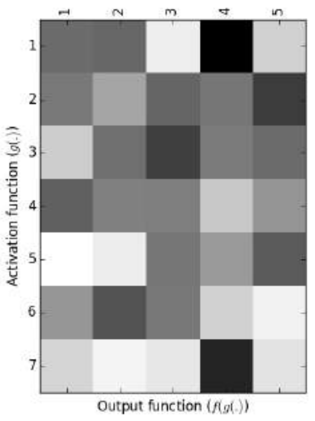
(b) $\gamma = 0.2$



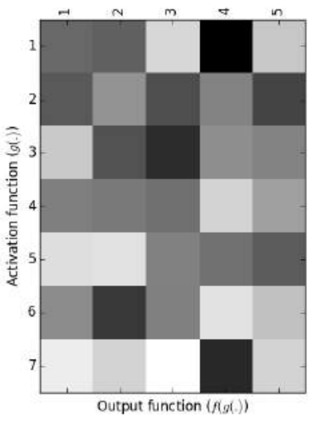
(c) $\gamma = 0.3$



(d) $\gamma = 0.4$

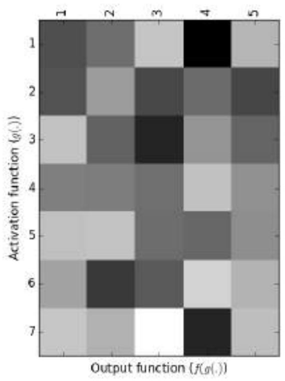


(e) $\gamma = 0.5$

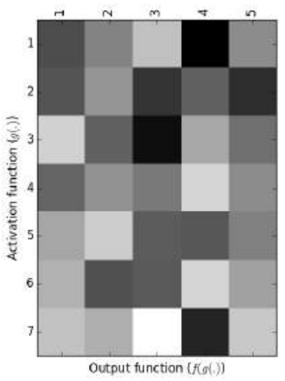


(f) $\gamma = 0.6$

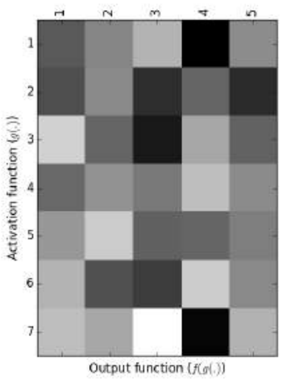
Figure 5.17: The average transfer function likelihood for the noise levels within the range of $\gamma \in \{0.1..0.6\}$ on the XOR dataset.



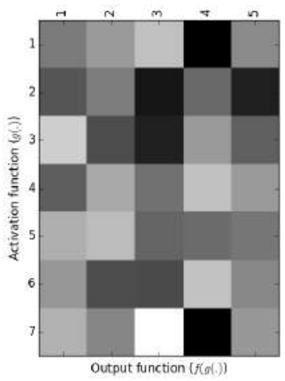
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$



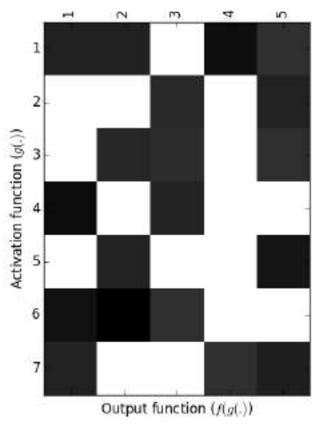
(c) $\gamma = 0.9$



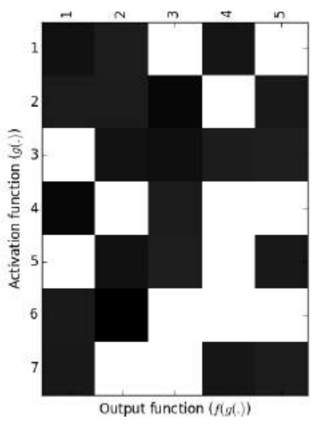
(d) $\gamma = 1.0$

Figure 5.18: The average transfer function likelihood for the noise levels within the range of $\gamma \in \{0.6..1.0\}$ on the XOR dataset.

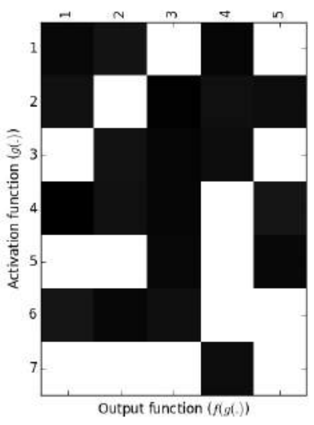
Regarding the XOR dataset, Fig. 5.17 & Fig. 5.18 show its result of transfer function likelihood as the level of noise is increased. It also shows consistent patterns as with the later results of the other datasets.



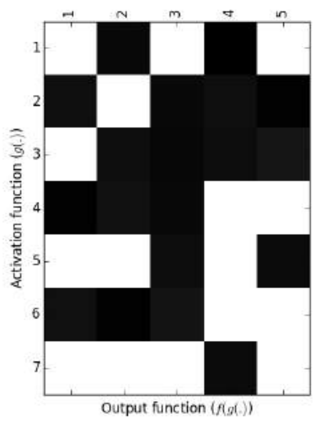
(a) $\gamma = 0.1$



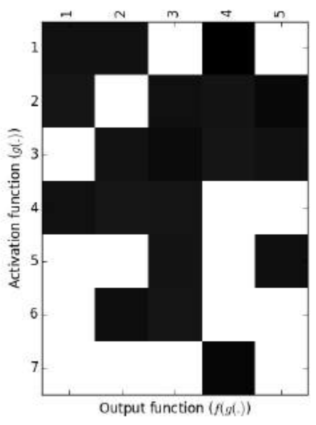
(b) $\gamma = 0.2$



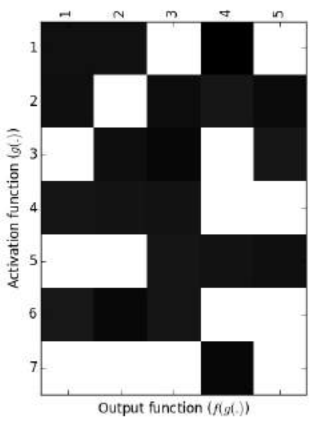
(c) $\gamma = 0.3$



(d) $\gamma = 0.4$

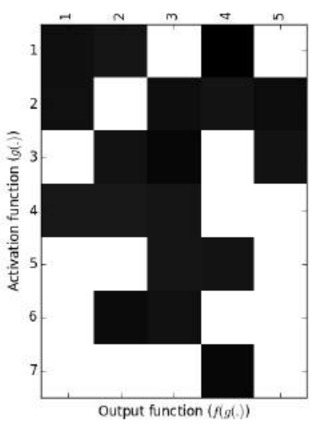


(e) $\gamma = 0.5$

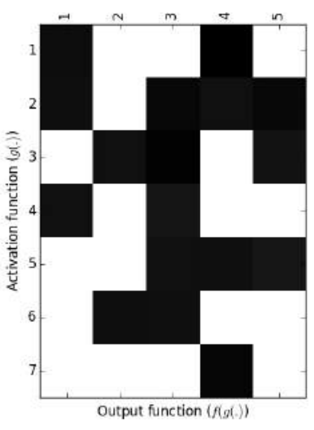


(f) $\gamma = 0.6$

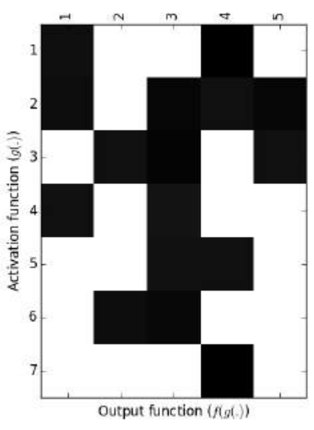
Figure 5.19: The average transfer function likelihood after thresholding for the noise levels within the range of $\gamma \in \{0.1..0.9\}$ on the XOR dataset.



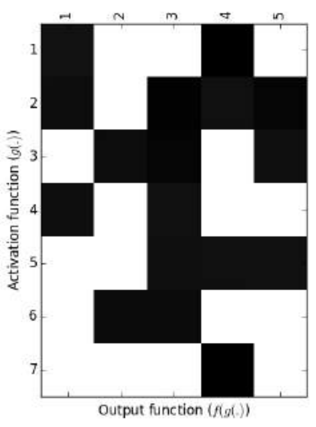
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$



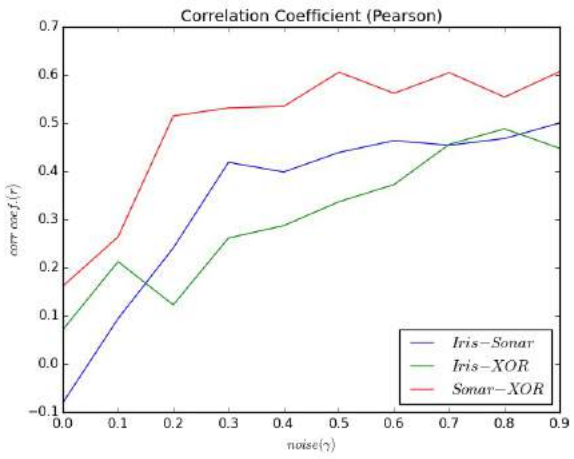
(c) $\gamma = 0.9$



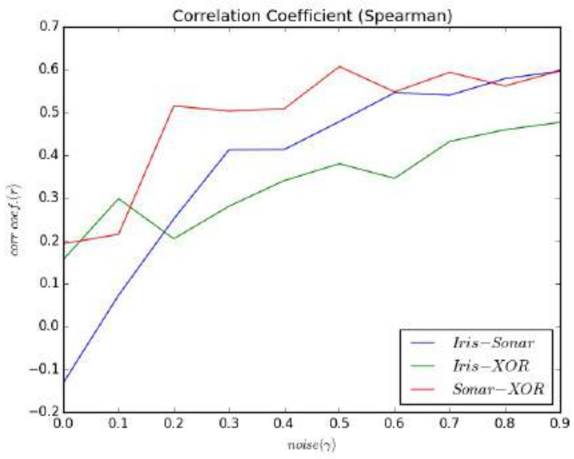
(d) $\gamma = 1.0$

Figure 5.20: The average transfer function likelihood after thresholding for the noise levels within the range of $\gamma \in \{0.6..1.0\}$ on the XOR dataset.

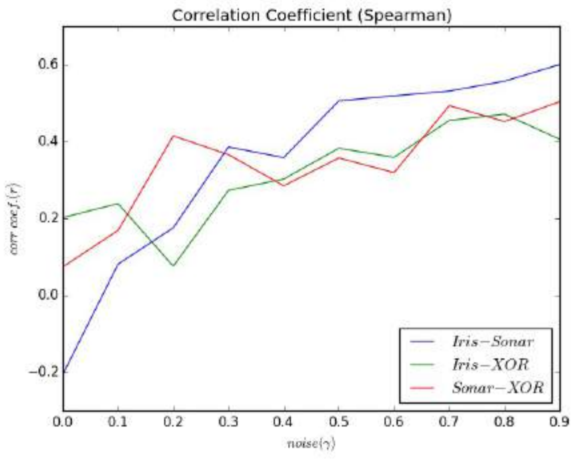
After thresholding, the figures (i.e. Fig. 5.17 & Fig. 5.18), the resulting transfer function likelihoods are as in Fig. 5.19 & Fig. 5.20. This made the most likely transfer functions more obvious, and also showed the consistency of the signatures as the level of noise was increased. One observation, as with the later datasets is that some transfer functions likelihoods faded out from the resulting transfer function likelihoods post-thresholding as the noise level was increased. One example is $(4,3)$, which appeared at noise levels of $\gamma \in \{0.2, 0.3, 0.4, 0.5\}$. Another observation is the that like the other datasets, there was a consistency of a significant number of the most likely transfer functions as the noise level was increased. Additionally, another interesting the most likely transfer functions for the different datasets appeared to be unique. This was one of the motivations for studying the consistency and discriminatory ability of problem signatures.



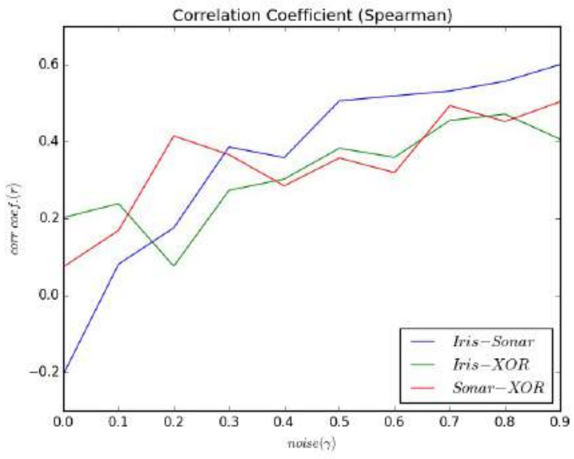
(a)



(b)



(c)

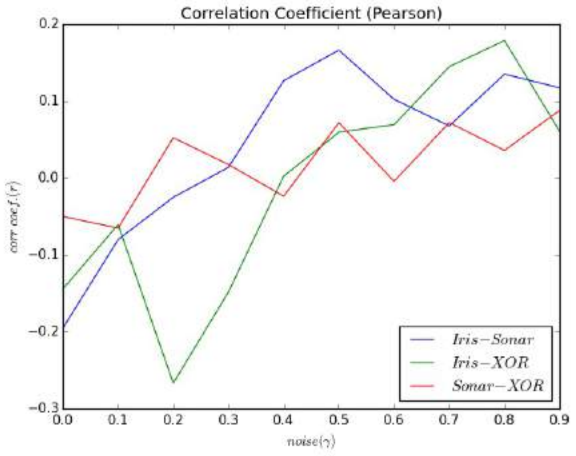


(d)

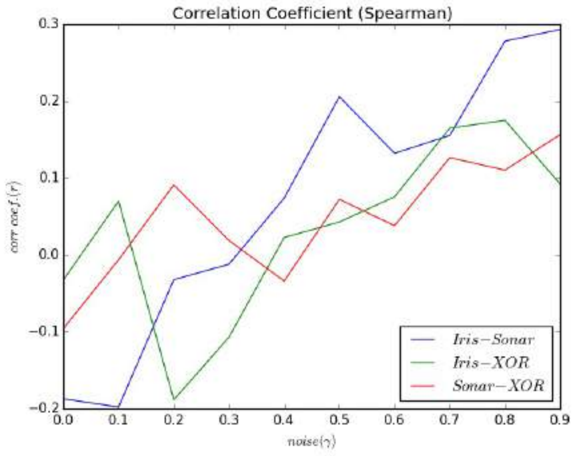
Figure 5.21: The correlation between the connection densities of the problems with and without thresholding. (a) and (b) represent results without thresholding, while (c) and (d) are results with thresholding.

The results of the correlation between the various types of signatures presented were used to answer the question of whether lower-order problem signatures were discriminatory by nature.

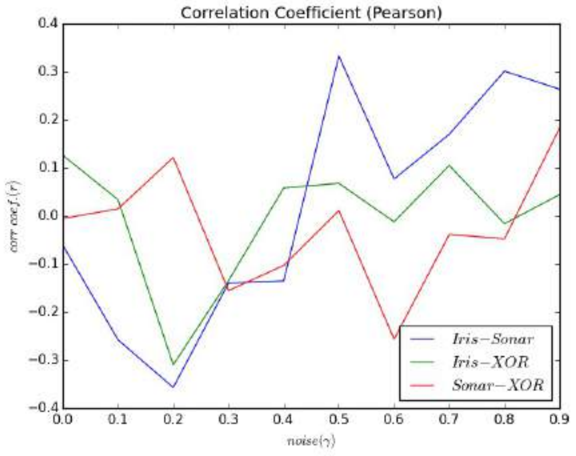
In terms of the correlation between the problems, it was apparent that the comparison of the connection densities between problems showed weak correlations for both the correlation measures used. The results in Fig. 5.21 (a) and (b) illustrates the correlation of the problems connection densities without thresholding as the level of noise was increased, i.e. $\gamma \in \{0.1 ... 0.9\}$. While the Figures Fig. 5.21 (c) and Fig. 5.21 (d) presents the results after thresholding as the noise was increased over the same range. This was intended to explore the possibilities of the correlations differing in both cases. In general, the results show a weak correlation between problems in terms of their connection densities.



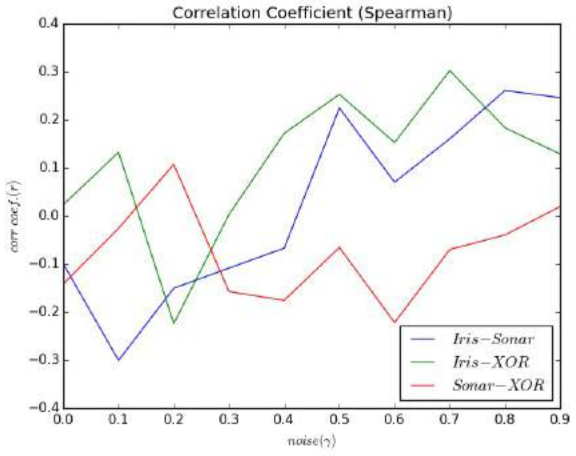
(a)



(b)



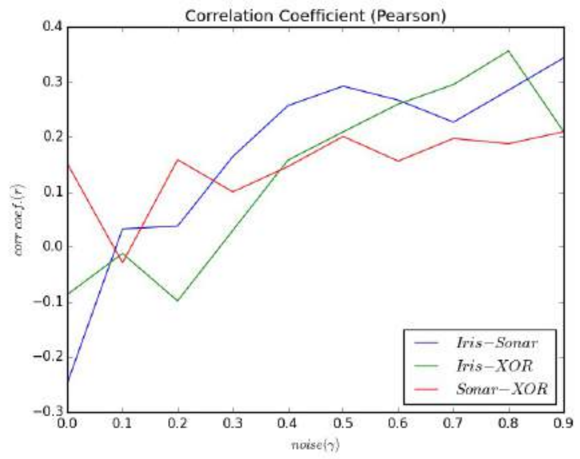
(c)



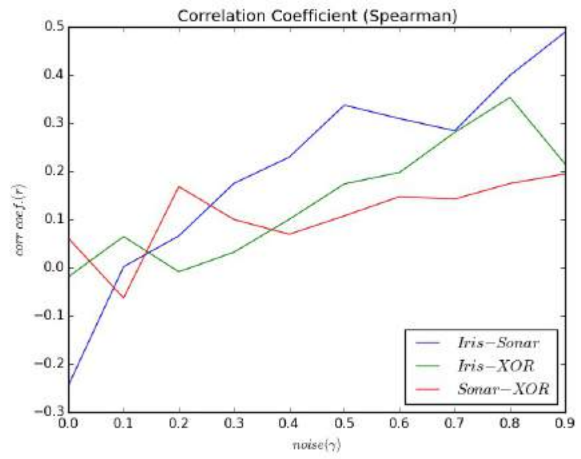
(d)

Figure 5.22: Correlation between the transfer function likelihoods of the problems both with and without thresholding. (a) and (b) are Pearson correlations coefficients, (d) and (c) are Spearman correlation coefficients.

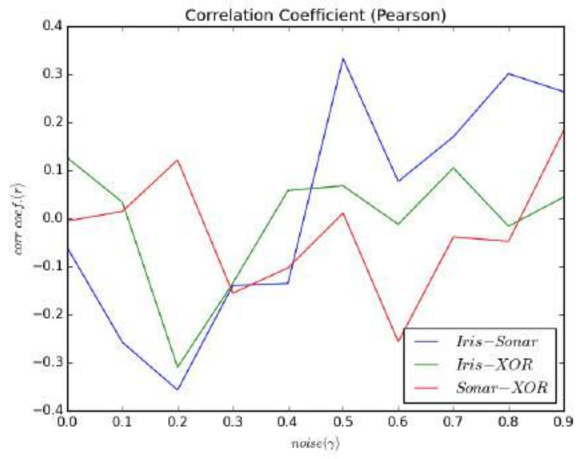
The results of the correlation between the transfer function likelihoods of the problem both with and without thresholding are highlighted in Fig. 5.22. Interestingly, these showed even weaker correlations of between 0.1 and 0.3, in general. Another observation is that the correlations results for the connection densities seemed to be relatively smoother as the level of noise was increased, compared to the transfer function likelihood.



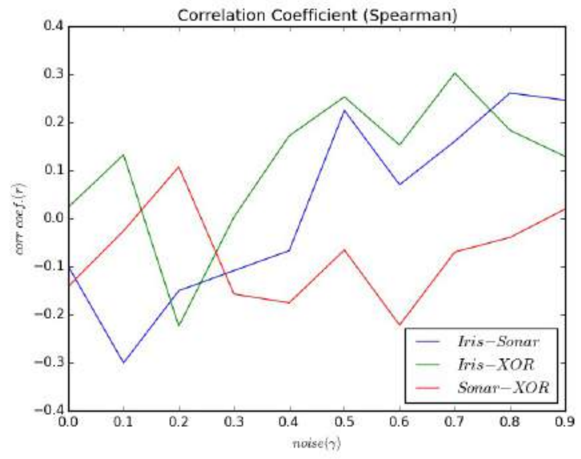
(a)



(b)



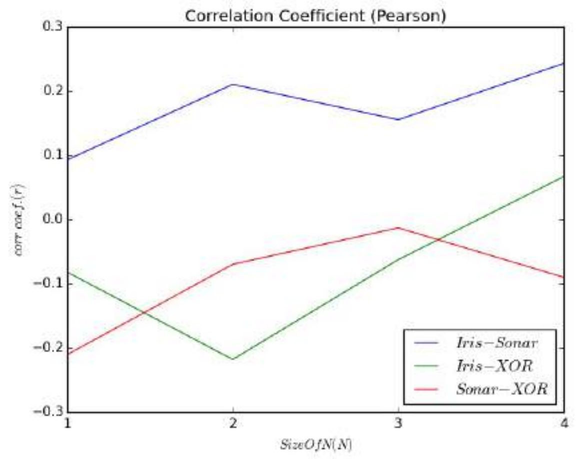
(c)



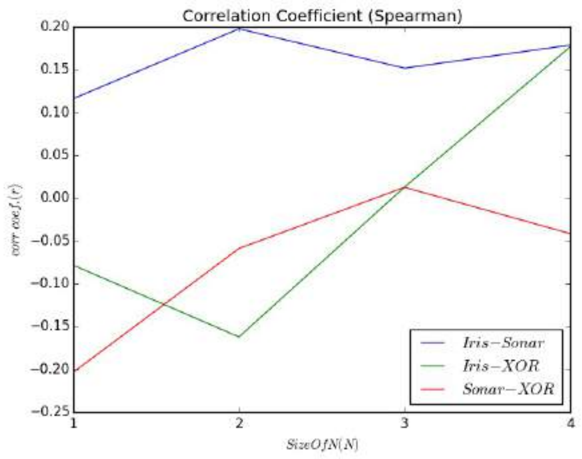
(d)

Figure 5.23: Correlation between the association associated error of the problems both with and without thresholding. (a) and (b) are Pearson correlations coefficients, (d) and (c) are Spearman correlation coefficients.

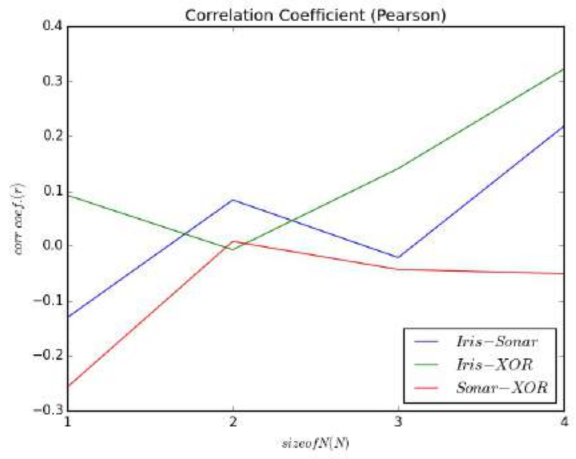
Similar results were also observed for the correlation of the associated error of the transfer functions between the problems. It also showed what was within the range that indicates weak or no correlation between the problems associated error. Thus, further suggesting that the lower-order signatures were distinct.



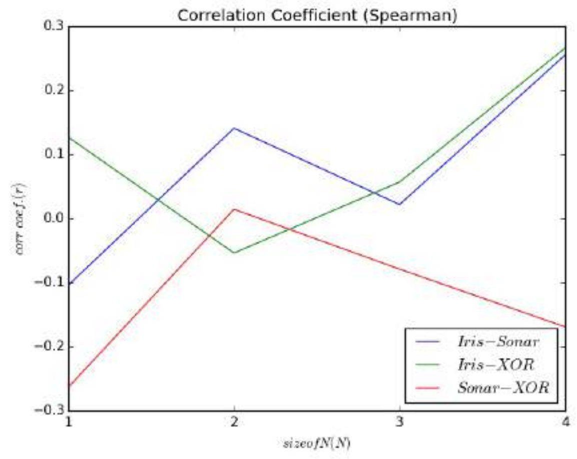
(a)



(b)



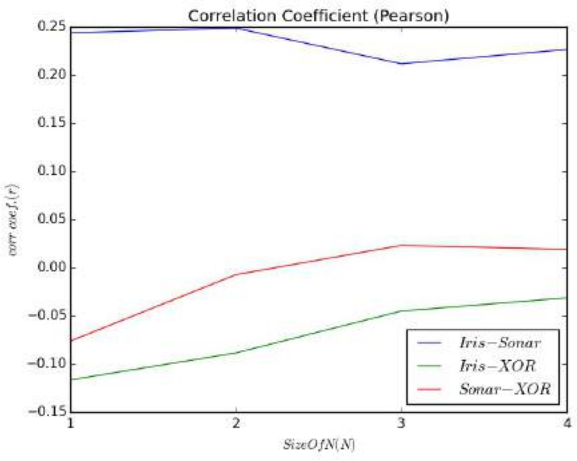
(c)



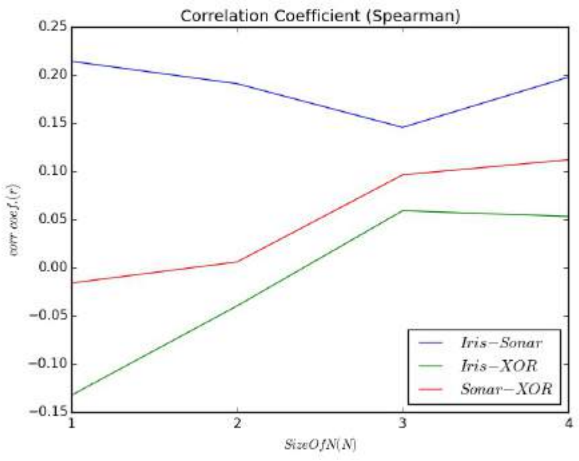
(d)

Figure 5.24: The correlation between the connection densities of the problems both with and without thresholding as the size of N is increased $\{1..4\}$.

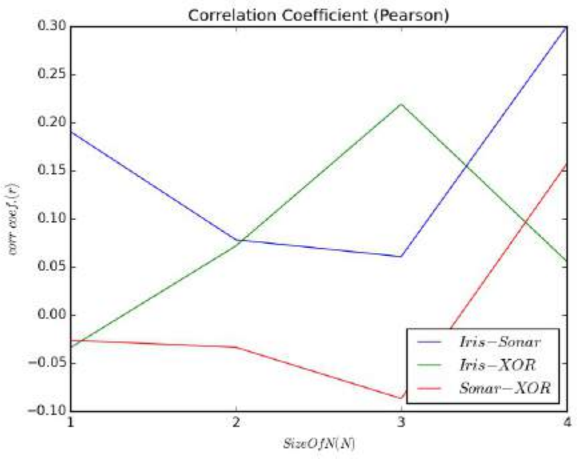
The results in Fig. 5.24 shows the correlation between the connection densities of the problem the size of N was increased. As expressed earlier, N represents the number of elite models to be sampled from during the extraction process. Fig. 5.24 (a) and (b) show the Spearman and Pearson correlations of the connection densities prior to thresholding. While Fig. 5.24 (c) and (d) show the Spearman and Pearson correlations of the connection densities after thresholding. Interestingly, relative to the results of increasing the level of noise γ, the results of the correlations showed much weaker correlations. Specifically, while the results of the condition of increasing noise seemed to increase gradually to a correlation of 0.5, which is still regarded as a weak correlation, the results of increasing the size of N had a maximum correlation of 0.3.



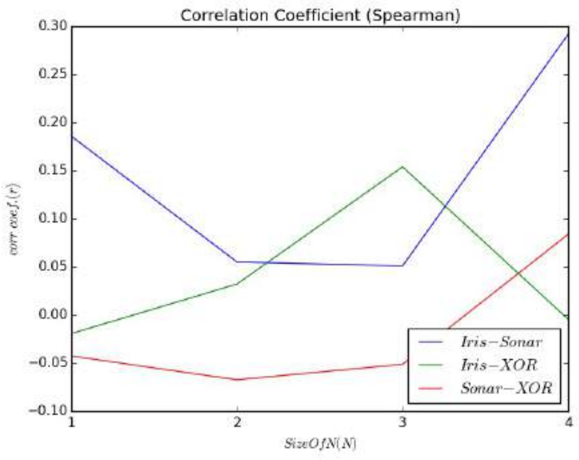
(a)



(b)



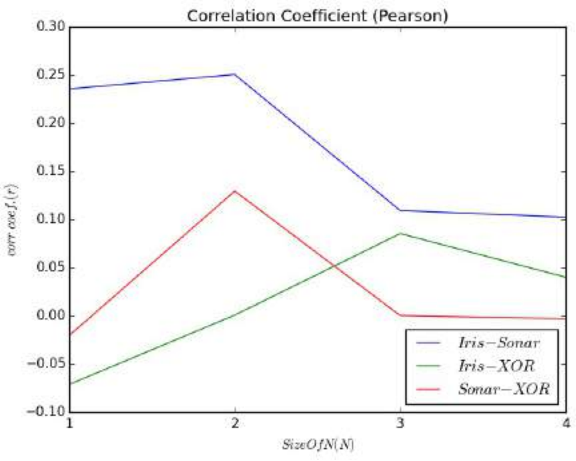
(c)



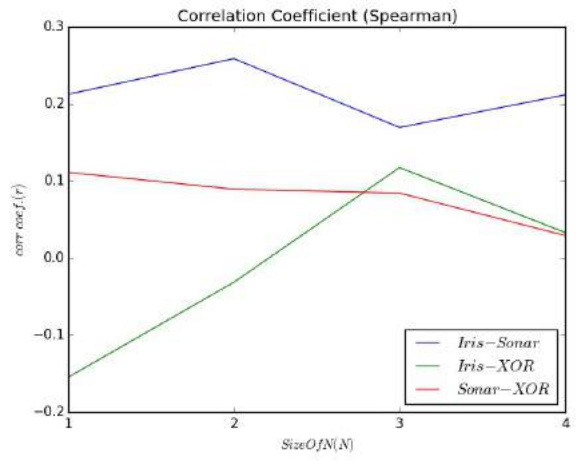
(d)

Figure 5.25: Correlation between the transfer function likelihoods of the problems both with and without thresholding as the size of *N* is increased $\{1..4\}$. (a) and (b) are Pearson correlations coefficients, (d) and (c) are Spearman correlation coefficients.

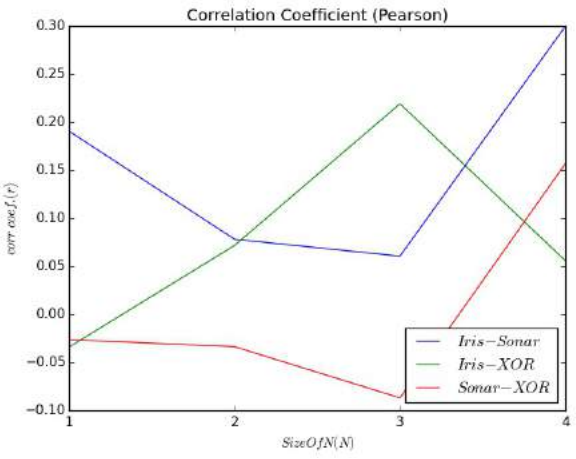
In terms of the correlation of the transfer function likelihoods, the correlations results between the problems are highlighted in Fig. 5.25. Similarly, the results also portrayed suggest that there are either no correlations or very weak correlations between the problems.



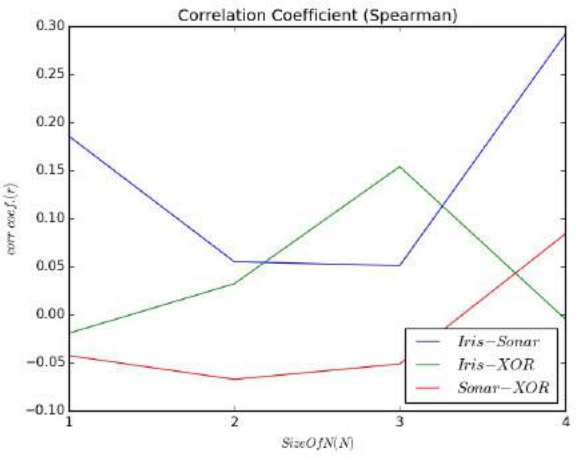
(a)



(b)



(c)



(d)

Figure 5.26: Correlation between the transfer function association of the problems both with and without thresholding as the size of N increases $\{1..4\}$. (a) and (b) are Pearson correlations coefficients, (d) and (c) are Spearman correlation coefficients.

Similarly, the results of the correlations for the associated error of the transfer functions between the problems as shown in Fig. 5.25, also suggested that they were very weakly correlated.

5.7.4 Discussion

In this section, we discuss the consistency of firs and second order signatures on the various circumstances including noise levels (γ), population size (P), and size of the top (N).

5.7.4.1 Consistency with Noise Levels γ

In general, lower-order signatures showed consistency for the different values of noise γ as seen in the figures (see Fig. 5.9, 5.13, and 5.17), and this was even clearer after a threshold function was applied (see Fig. 5.11, 5.15, and 5.19). However, it was apparent that there was some degree of randomness in lower-order signatures. This was likely due to the lower-order signatures representing a lower-level of information which is seems to be more susceptible to noise than higher level information, as we shall see in the next section.

Some of the transfer functions that make up the consistent pattern for the transfer function likelihood include the following:

- $(6,2)$ - Min and Sigmoid
- $(4,1)$ - Higher-Order subtractive and Identity
- $(1,4)$ - Inner product and Hyperbolic tangent
- $(5,5)$ - Standard Deviation and Gaussian II
- $(7,4)$ - Max and Hyperbolic tangent

Interestingly, these were also correlated with the connection densities of the transfer functions. However, there was some degree of randomness in some of the correlations where it showed a correlation in some but not in others, for example, $(1,4)$.

These results highlight some of the advantages of the higher-order problem signatures over lower-order; while lower-order signatures generally seem to have a lot more randomness or sensitivity to the whole sampling process, higher-order signatures appeared to be more resilient - as shown in the next section. In addition to that, lower-order provide little information about the underlying computation which might help in reconstructing it. This is because the information is not in the context of other computations done by other neurons, as it is very specific. In other terms, it could be said to be of a low granularity. However, regardless of that, it is also a significant finding to show that there were consistencies in first and second order signatures.

5.7.4.2 Consistency with Size of N

In terms of consistency with the size of N (i.e. the number of elite solutions selected for sampling the lower-order signatures), there was also a consistent pattern in general. However, like the results of the tests with the different levels of noise; it also seems to have some randomness that is also visible within the signatures as with the other signatures.

In terms of the Iris dataset, the results for the range of $N \in \{1,2,3,4\}$ shows consistency in the patterns as N was increased. The transfer functions that made up the consistent patterns in the signatures included the following:

- $(7,4)$ - Max and Hyperbolic tangent
- $(5,5)$ - Standard deviation and Gaussian II
- $(5,4)$ - Standard deviation and Hyperbolic Tangent
- $(5,1)$ - Standard deviation and Identity
- $(4,2)$ - Higher-Order Subtractive and Sigmoid

Interestingly, as we shall see in the next section, this was consistent with the higher-order problem signatures.

In terms of the Sonar problem, the results for the same range of $N \in \{1,2,3,4\}$, some patterns also remained consistent as the size of N was increased. The transfer functions that made up the consistent patterns, in this case, included the following:

- $(7,5)$ - Max and Hyperbolic Tangent
- $(5,3)$ - Standard Deviation and Gaussian
- $(4,5)$ - Higher-Order subtractive and Gaussian II
- $(4,3)$ - Higher-Order subtractive and Gaussian
- $(1,1)$ - Identity and Inner product

It was interesting to see that these were different from those presented for the Iris dataset. Thus, suggesting that lower-order signatures also have a discriminatory property between different problems. Intuitively, consistency on an individual problem should also help result in a discriminatory property for signatures.

In terms of the XOR dataset, the results also showed a different pattern from the rest of the problems. In addition, it was also consistent for the range of values for $N \in \{1,2,3,4\}$.

In particular, some of the consistent transfer functions for the elite N models included the following:

- $(1,1)$ - Inner-product and Identity
- $(1,4)$ - Inner-product and Hyperbolic Tangent
- $(5,3)$ - Standard deviation and Gaussian
- $(4,2)$ - Higher-Order subtractive and Sigmoid
- $(5,4)$ - Standard Deviation and Hyperbolic tangent

Interestingly, this was consistent with the pattern of transfer function likelihood for the higher-order problem signatures, as we shall also see in the next section.

5.7.4.3 Discriminatory Property between Problems

In terms of the connection densities between the three problems (i.e. Iris, Sonar, and XOR); the correlation seems to be on an increasing trajectory with a slow rate of increase in the range of noise levels $\gamma = \{0.1..0.9\}$. This was likely as a result of distortion from the noise generated from a normal distribution. The figure (see Fig. 5.24) shows the results for the Pearson correlation, which measures for linear correlations. A Spearman correlation measure was also used to determine if there were monotonic relationships between the signatures of the different problems. In this results, the range of correlations seemed to be within [-0.1, 0.5] for both the Spearman and Pearson (see Fig. 5.24). This is usually considered to be somewhere between no correlation and a weak correlation. In other words, there is likely no linear or monotonic correlation between the signatures for different problems. The implication of this is that the signatures of different problems are unlikely to correlate under the condition given (i.e. noise levels). This suggests that the connection density signatures were discriminatory between the problems, but also somewhat sensitive to noise.

In terms of the correlation measures for the transfer function likelihood, relatively lower correlations were observed (see Fig. 5.22). Especially when increasing the level of noise, there was particularly a very weak correlation and sometimes no correlation between the signatures of problems. The results post-thresholding also showed also showed no or very weak correlations. It was interesting that these were less correlated between problems as compared to connection densities. This is likely as a result of the order of the signatures, connection densities are first-order signatures which tend to capture information without context while second-order signatures such as the transfer function likelihoods capture information with some degree of context. Additional information in signatures seems to help improve their discriminatory property. This is because additional information increases specificity, and increased specificity makes it easier to discriminate between signatures. This could be one reason for the connection densities showing more correlation as compared to both the transfer function likelihood and their associated errors.

5.7.5 Conclusion

In a nutshell, we have shown the results the signatures of different problems; both the first-order (connection density) and second-order signatures (transfer function likelihood). These were for two conditions, the levels of noise, and the sizes of N. The signatures, in general, showed no to weak correlation for the signatures between problems. Visually, the signatures were also apparently consistent. However, there was some disturbance. Specifically, some patterns that emerge in the signatures with thresholding applied to them seemed to disappear as the different levels of the condition was changed, e.g. as in the case where the noise levels were increased. We have also noticed the potential for better discrimination of second-order signatures in particular. This was hypothesized to be because of the additional information in second-order signatures which helps in increasing specificity, which helps in improving the discriminatory property of signatures. Though first-order signatures also seemed discriminatory after thresholding, the correlations between problems was a lot higher than that of second-order signatures. Thus, suggesting that it had weaker discriminatory property. These results were consistent with the results of the higher-order problem signatures, which are also discriminatory after thresholding. In general, the signatures showed potential of being consistent and discriminatory between problems. This can be explained in terms of the differences in relationship to input space of problems, the error surface over the hypothesis space of artificial neural networks, and the law of central tendency. The input space and the nature of problems define the error surface of the hypothesis space, with the most optimal solution being the global minima in that error surface and various other alternative solutions that form local minima. As features of the architecture of the fittest models are sampled, by the law of central limit this will eventually converge on an expected pattern for the fittest models. The fittest models, in this case, are likely approximating a hypothesis of a local minimum, and as such that expectation of the features will be of the features of the model that approximates that local minimum. Indirectly, this also tells us something about the hypothesis being approximated, or in other words, the computational signature. In terms of the literature, there has been very few work with regards to the meta-analysis of neural networks. This provides a contribution towards understanding neural networks by understanding what computational strategies are learned after learning. In turn, this presents an avenue for other applications such as the initialization of neural network topologies and transfer functions based on the problem. In other words, architectures that adapt to the problem. This presents a significant and important contribution.

5.8 Consistent and Discriminatory Properties Of Higher-Order Signatures

In this section, we show that higher-order problem signatures are consistent for the conditions experimented with; specifically, the varying levels of noise, sizes of the population and size of the set of elite solutions selected for sampling the signatures.

5.8.1 Motivation

As explained earlier, there was a need for verifying the consistency of higher-order signatures on problems. This was mainly to explore the feasibility of higher-order problem signatures being used as meta-features. The implications include the possibility of initializing the architectural components of artificial neural networks such as topology and set of transfer functions, as illustrated in the earlier section. Higher-order problem signatures will need to satisfy the criteria explained in section 5.4.3 to be considered feasible meta-features for artificial neural networks. We hypothesize that sampling the elite models of artificial neural networks without training should result in convergence on promising regions of the search space.

Another property that was tested for was the discriminatory property of higher-order computational signatures. Showing that signatures are discriminative is one step closer to showing that they can be used as meta-features for artificial neural networks. As expressed earlier, this has significant implications: it shows that higher-order properties of problems can be sampled. In terms of applications, these can be used to determine which neural network architecture might work best for problems based on higher-order signatures.

In relation to the literature, the closest work is from the domain of meta-learning which uses the features of a decision tree, such as the depth trained on a problem, as meta-features [81]. These model properties are then used to decide which learning algorithm would be more suitable for learning the problem.

As of the time writing, there is yet to be a general framework of neural network meta-features that could be used to both understand the underlying strategy, and also help initialize the architectural components of neural networks. Thus, showing that higher-order problem signatures are feasible to be used as meta-features presents a significant contribution with potential applications that include neural network architecture initialization and a deeper understanding of neural computational strategies.

The experiments that follow show that higher-order problem signatures are also consistent. In the previous section, lower-order signatures were shown to have consistent and discriminatory properties. In addition, it was demonstrated that they could be used to improve convergence in neurally diverse artificial neural networks. In this section, we show that higher-order problem signatures are also consistent. We also show that higher-order problem signatures are relatively more consistent than lower-order signatures, and could be used to gain valuable insight into the neural computation strategies evolved by the neurally diverse artificial neural networks.

5.8.2 Experimental Setup

The datasets used for the experiments in this section consisted of the Iris, Sonar and XOR datasets. These datasets were retrieved from the UCI machine learning repository [13].

The signature extraction process remained the same as in the previous chapter. The computational signatures were extracted from a subset of the population, in particular, the top N solutions after sorting. Independent samples of these signatures were accumulated over a number of runs R, from a population of neural networks with size P. For various conditions of the experiments, these parameters (i.e. P, R, and N) were varied to examine their effects on the consistency of the signatures. In this section, we are mainly interested in higher-order problem signatures. The Mean squared error was used to evaluate the neural networks which had to be recreated from their genetic string representation. The neural diversity machines used were consistent with the description given in the earlier chapter and the original framework for neural diversity machines [71]. This means the populations consisted of not only neural networks with diverse transfer functions but also of varying complexities (i.e. hidden layer size). This has the potential of increasing the spread of the regions from the model space being sampled.

Also, the average of the signatures for each of the experiments was used to generate a heat map. This helped to reveal some of the most used neural computation paths from the fittest neural network models.

In summary, this section seeks to answer if the following affected the signature consistency:

- Size of the of fittest N solutions collected for signature extraction.

- Population size (P).
- Noise level injected into the training examples (γ).

As such, the conditions for the experiments were as in Table 5.6.

Table 5.6: Conditions under which the higher-order signatures were tested.

Condition	Description
Size of fittest N	Tests for various sizes of $N \in \{1...4\}$ solutions to be selected after sorting
Population size P	Tests for various sizes of $P \in \{100...1500\}$ used for signature extraction
Noise levels γ	Tests for some range of runs $\gamma \in \{0.1...0.9\}$.

5.8.3 Results

In this section, we present the results of extracting the computational signatures for the Iris, Sonar and XOR datasets under various conditions. Specifically, varying the levels of noise, sizes of the top N models selected for signature extraction, and finally the population size P.

The figures (Fig. 5.27, 5.28 and 5.29) show the signatures for the Iris, Sonar , and XOR datasets, respectively. There was an apparent pattern of heavy usage of some transfer functions as illustrated in figures (i.e. Fig. 5.27, 5.28 and 5.29) . Fig. 5.27 presents the results of the iris dataset with various population sizes, from 100 to 1500. It also presents the results after filtering with the mean as the threshold (i.e. $\alpha = 0.0$), with filtering with an adjustment value ($\alpha = 1.5$). The α value of 1.5 was found to be within the upper limit, beyond which there was barely any usable information left. The figures Fig. 5.28 and 5.29 present similar results for the Sonar and XOR datasets, respectively.

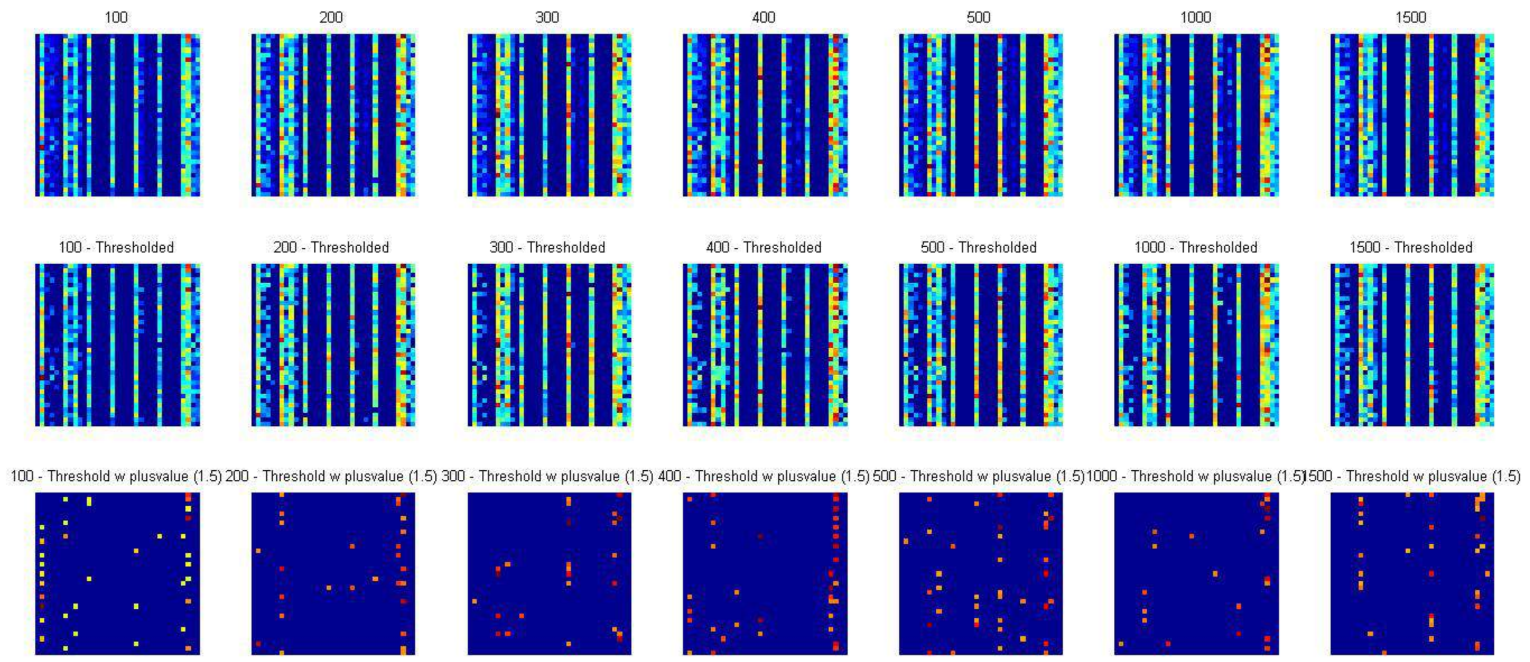


Figure 5.27: Iris heat-map of signatures showing how the features get more pronounced as the population size used for the signature sampling is increased.

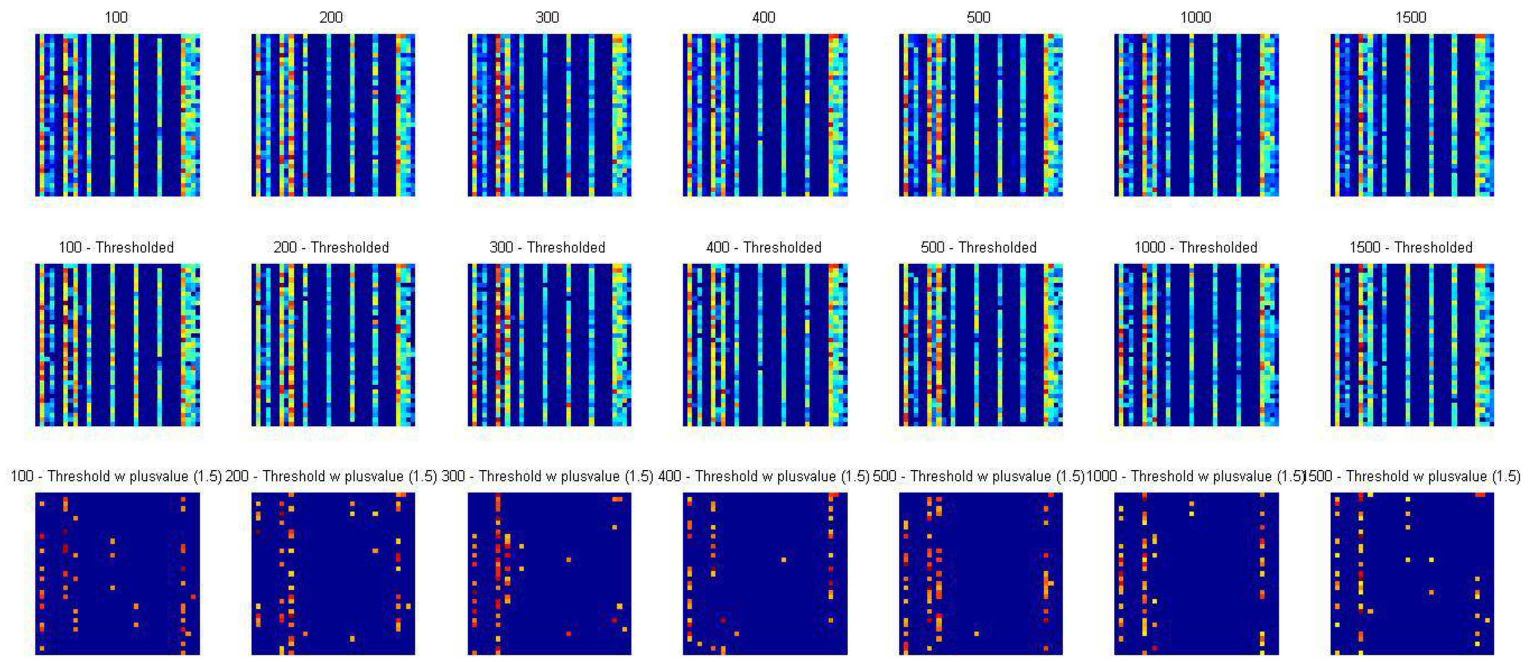


Figure 5.28: Sonar heat-map of signatures showing how the features get more pronounced as the population size used for the signature sampling is increased.

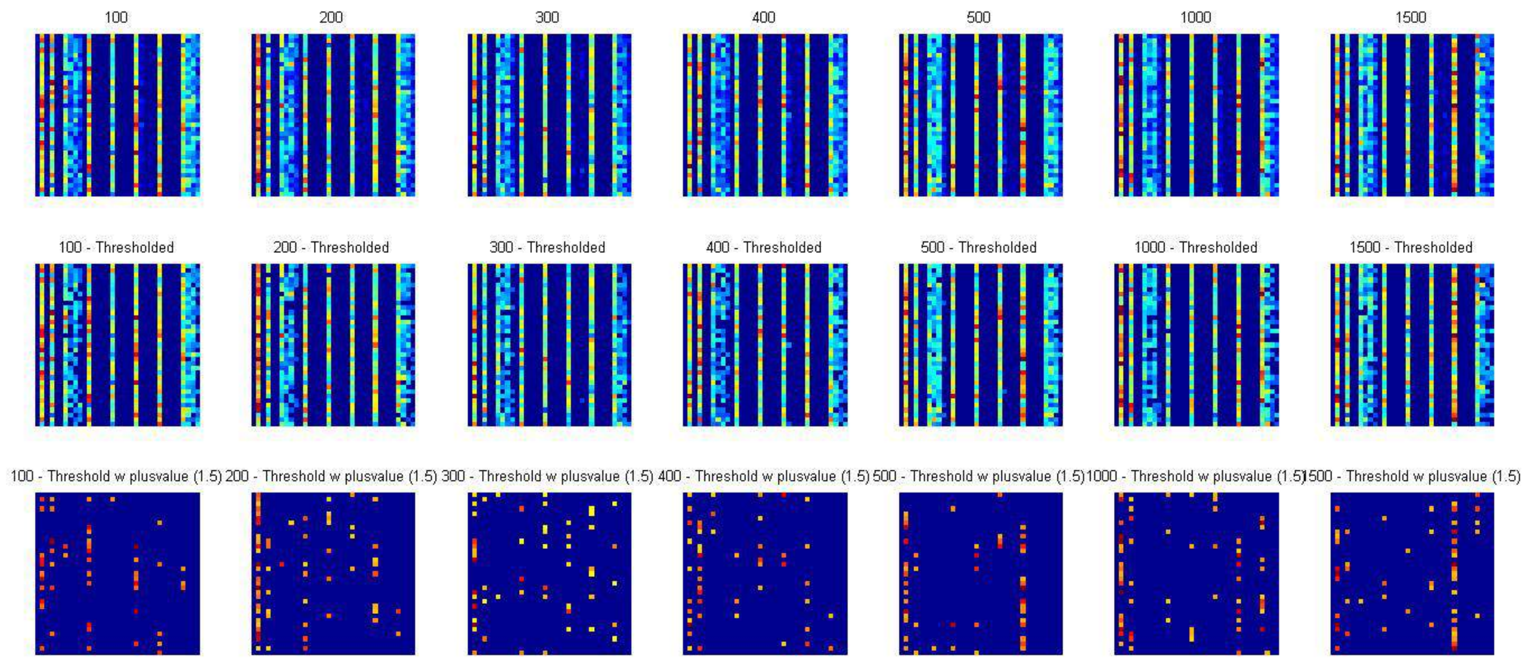
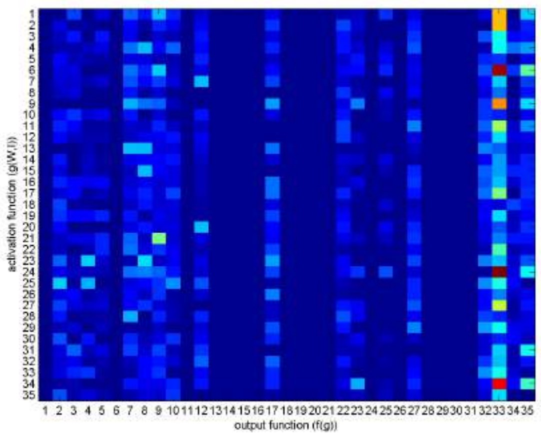
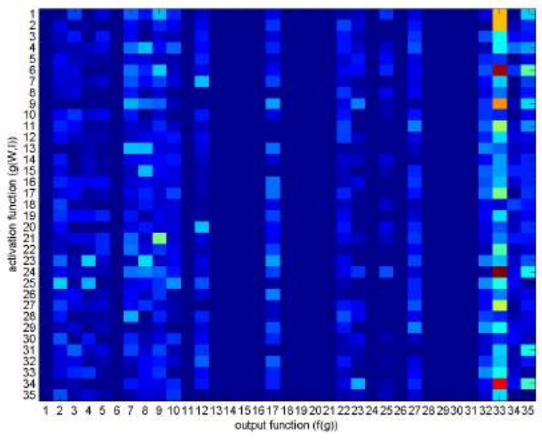


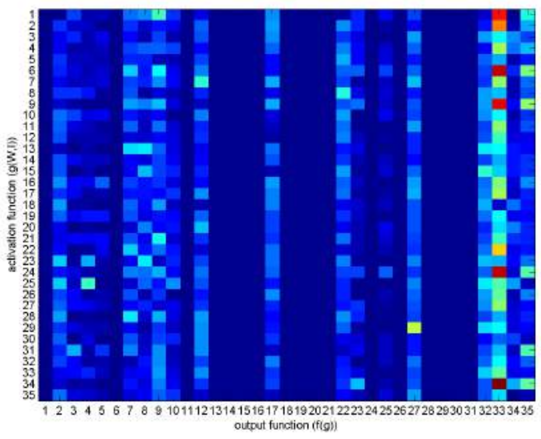
Figure 5.29: XOR heat-map of signatures showing how the features get more pronounced as the population size used for the signature sampling is increased.



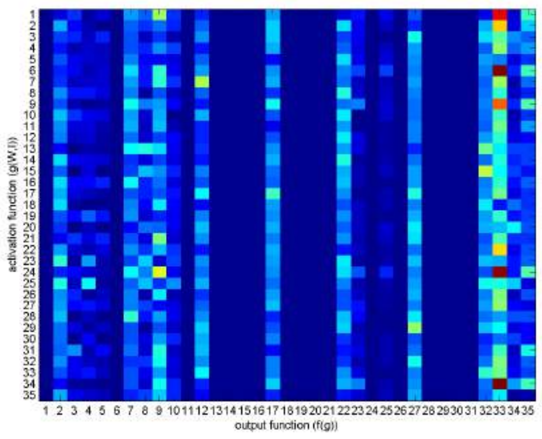
(a) $N = 1$



(b) $N = 2$



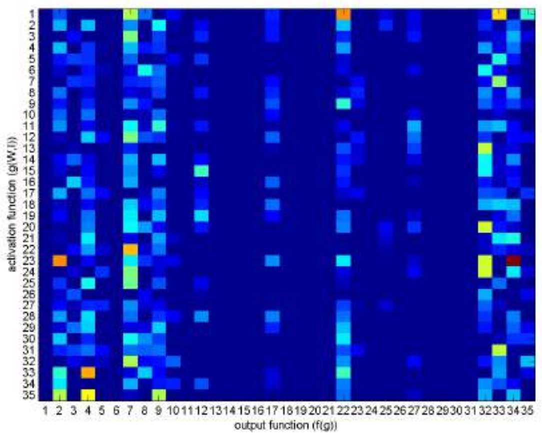
(c) $N = 3$



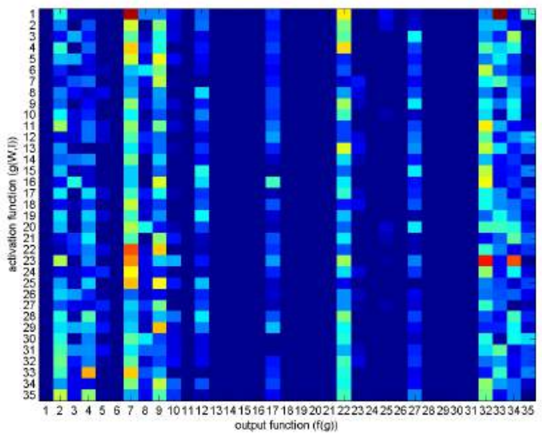
(d) $N = 4$

Figure 5.30: Heat maps showing the connection strengths as the size of N increases for the Iris dataset.

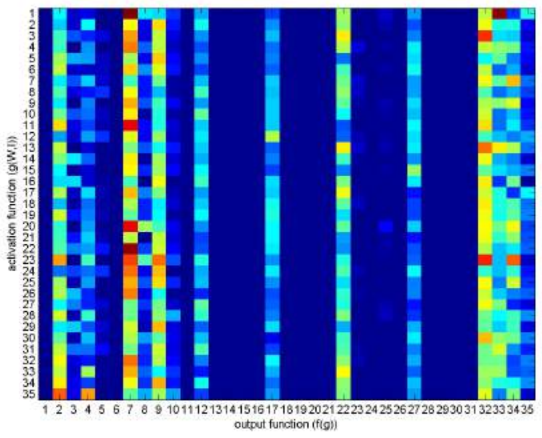
Fig. 5.30 shows the heat map of the connection strength between transfer functions for the Iris dataset. It presents the results for increasing sizes of $N \in \{1..4\}$, which is the size of the set of elite models used for sampling. It was apparent that just as the heat maps for increasing population sizes P, there were consistencies with regards to the connection strengths between transfer functions.



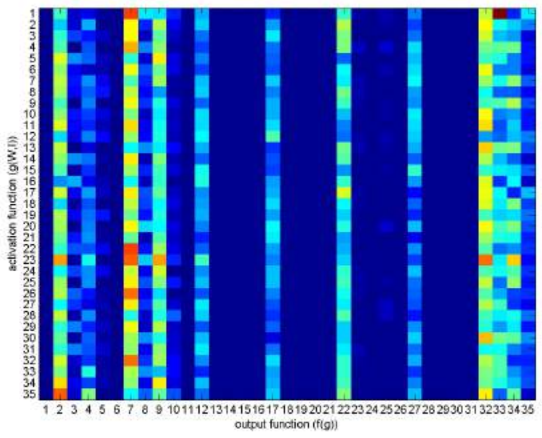
(a) $N = 1$



(b) $N = 2$



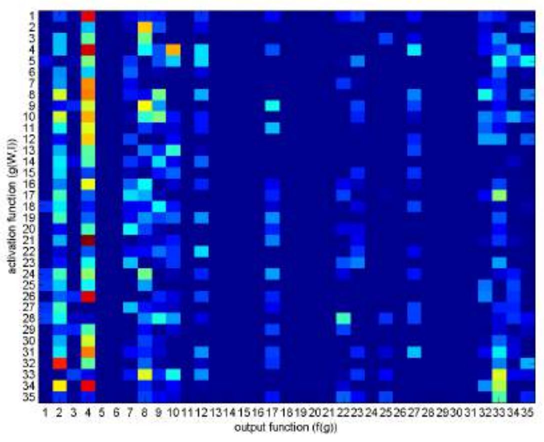
(c) $N = 3$



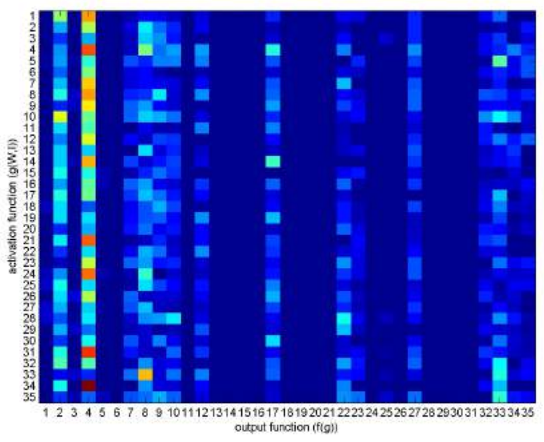
(d) $N = 4$

Figure 5.31: Heat maps showing the connection strengths as the size of N increases for the Sonar dataset.

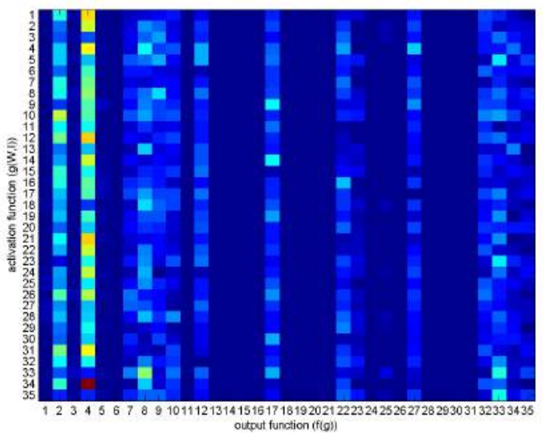
Fig. 5.31 shows the results of the heat maps of the connection strength between transfer functions as the size of *N* increases. Like the results of the Iris dataset, there were also visibly consistent patterns shown in the heat maps. Another observation is the apparent thickening of some of the faded patterns as the size of N is increased. This is because as the sample size increases, more dormant signatures of relatively less fit models from the top *N* models are also sampled. Hence, suggesting that the top *N* models of the population use different sets of transfer functions, and as a result different computational strategies.



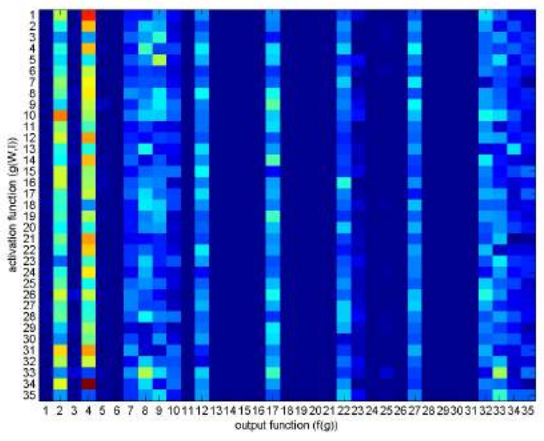
(a) $N = 1$



(b) $N = 2$



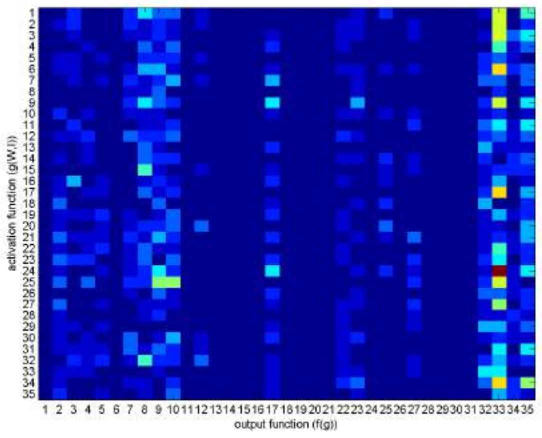
(c) $N = 3$



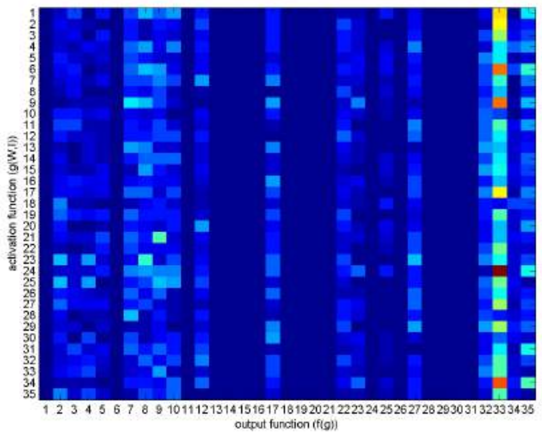
(d) $N = 4$

Figure 5.32: Heat maps showing the connection strengths as the size of N increases for the XOR dataset.

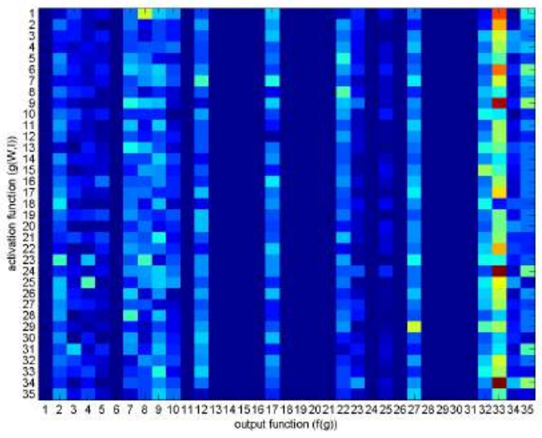
Similarly, Fig. 5.32 depicts the heat maps of the connection strengths for the XOR dataset. As the results of the previous datasets, there were visibly consistent patterns shown by the heat maps as well as the apparent increase in the use of connection strengths between other transfer functions.



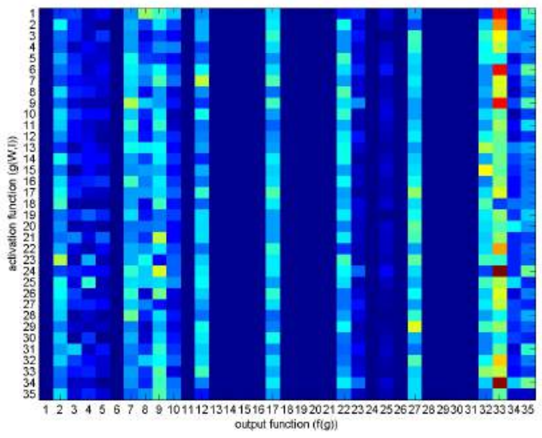
(a) $N = 1$



(b) $N = 2$



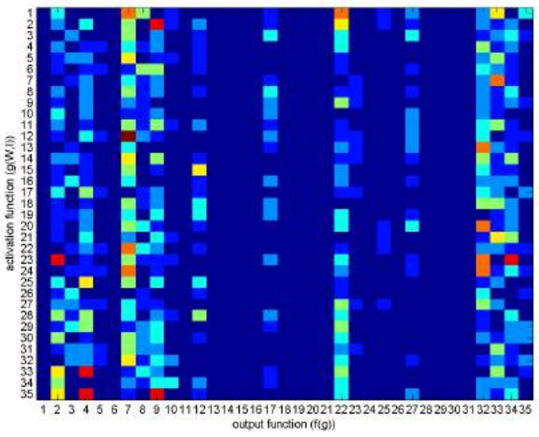
(c) $N = 3$



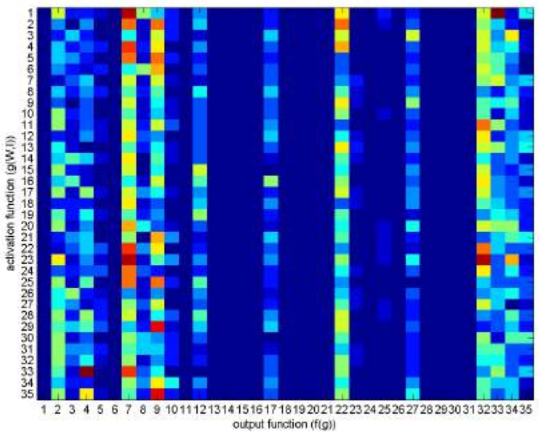
(d) $N = 4$

Figure 5.33: The coexistence likelihood for the Iris dataset for increasing sizes of N.

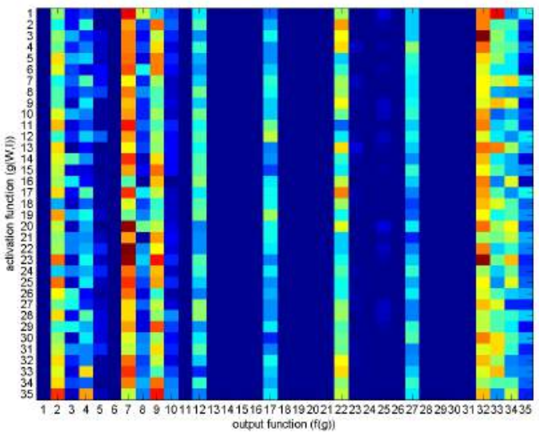
Fig. 5.33, 5.34 and 5.35 shows the results of the coexistence likelihoods of transfer functions with increasing size of N for the Iris, Sonar and XOR datasets, respectively. Fig. 5.33 visibly shows similar patterns of the likelihood of transfer functions connecting to each other with the patterns observed for the connection strengths.



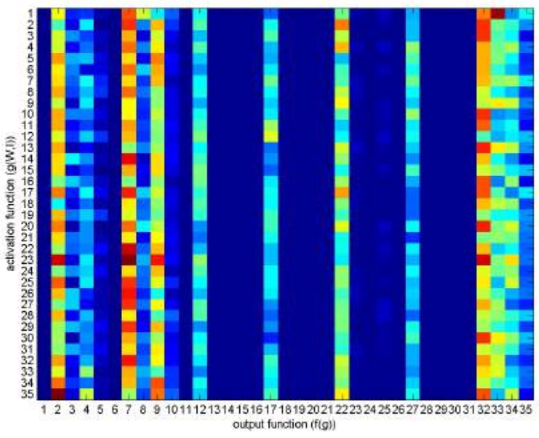
(a) $N = 1$



(b) $N = 2$



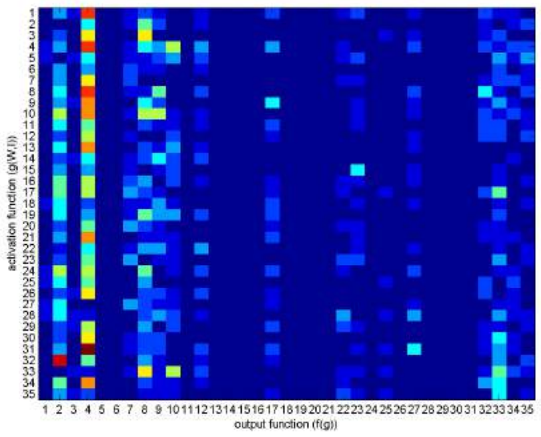
(c) $N = 3$



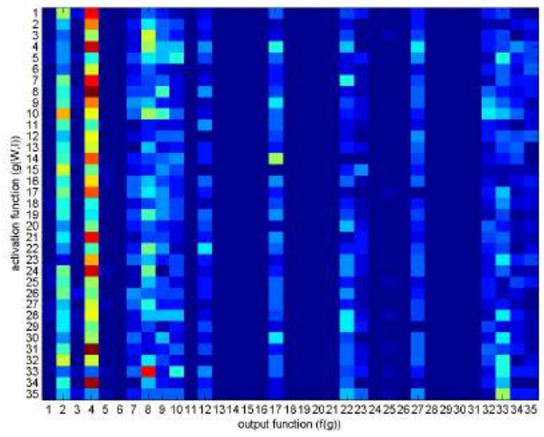
(d) $N = 4$

Figure 5.34: The coexistence likelihood for the Sonar dataset for increasing sizes of N.

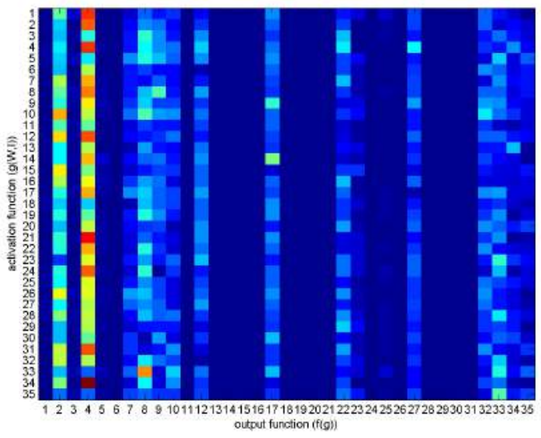
Interestingly, the coexistence likelihoods of the transfer functions for increasing sizes of N as shown in the heat maps of Fig. 5.34 differ from the ones observed for the Iris dataset. This suggests that different computational strategies are being used. Additionally, it also suggests that the coexistence likelihoods of transfer functions are discriminatory by its nature.



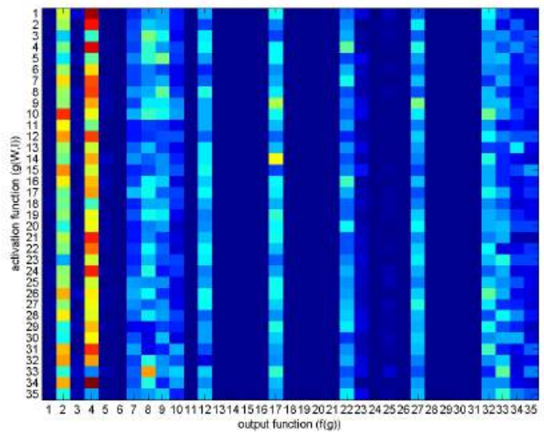
(a) $N = 1$



(b) $N = 2$



(c) $N = 3$



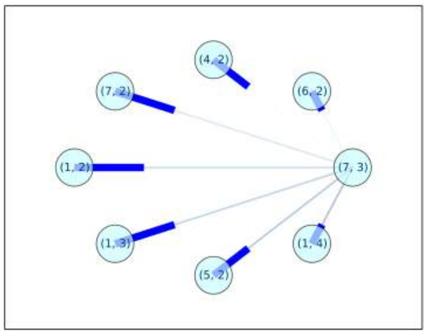
(d) $N = 4$

Figure 5.35: The coexistence likelihood for the XOR dataset for increasing sizes of N.

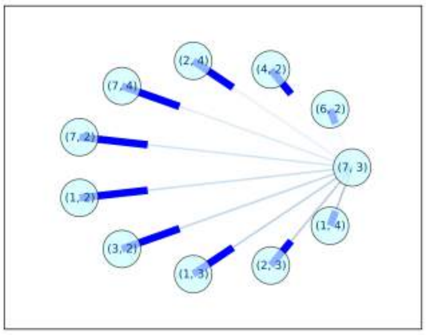
Similar observations were also made for the XOR dataset in Fig. 5.35, where the coexistence likelihoods differed from those of the other datasets. Additionally, it is also visibly apparent that the coexistence likelihood was consistent with increases size of N.

5.8.3.1 Results Of The Signatures Path Analysis

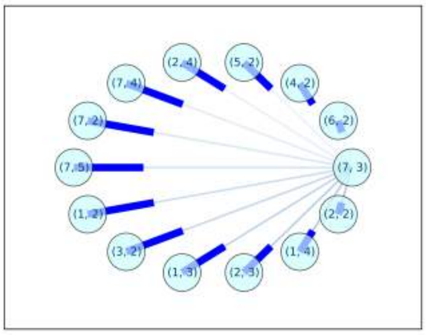
Path analysis was then performed on both the coexistence likelihood and connection strength signature types. This allows us to reconstruct the neural computational path that should be most likely to be found in the fittest neural network for that particular problem. As explained in the experimental setup, the terminal transfer function represents the neuron with the highest connection density, and other transfer functions connected to it are those most likely to connect to it after filtering with the mean as the threshold. The figures below show the results as the size of N is increased for Iris, Sonar, and XOR datasets.



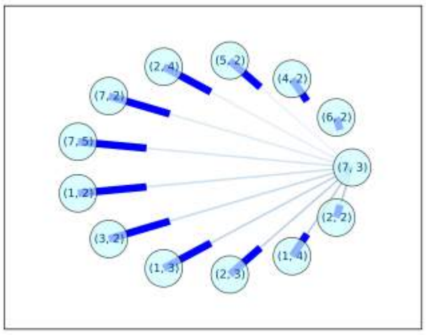
(a) $N = 1$



(b) $N = 2$



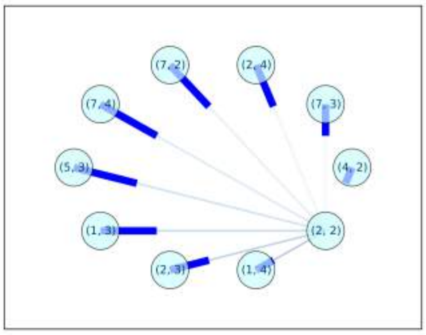
(c) $N = 3$



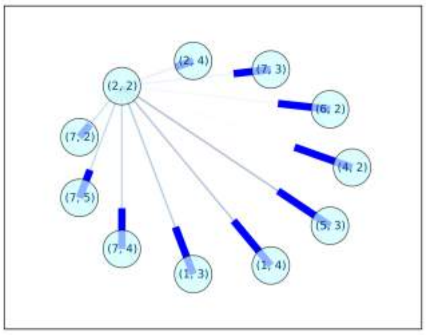
(d) $N = 4$

Figure 5.36: Path analysis for the Iris as the size of N is increased.

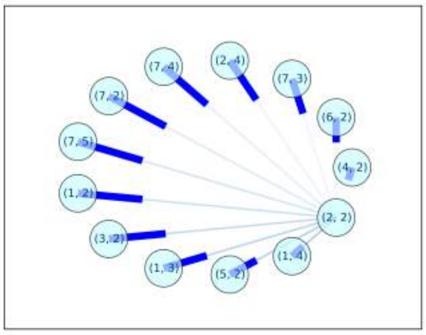
The path analysis of the Iris dataset reconstructed as Fig. 5.36, shows an interesting pattern that confirms the earlier observation of more transfer functions being more likely to co-exist on a connection path as the size of N increases. Compared to (a), the results in (b), (c) and (d) all have some additional path to the most densely connected transfer function.



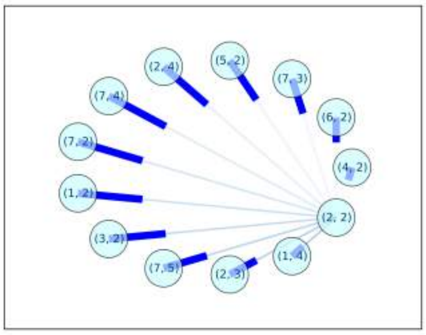
(a) $N = 1$



(b) $N = 2$



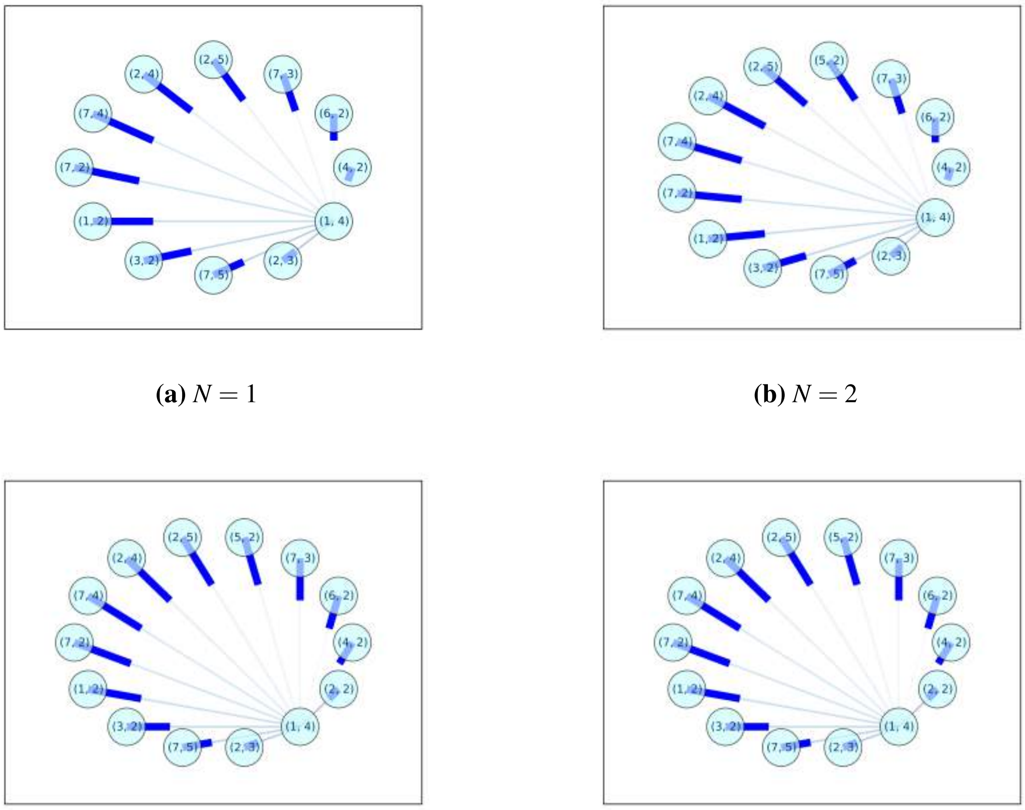
(c) $N = 3$



(d) $N = 4$

Figure 5.37: Path analysis for the Sonar problem as the size of N is increased.

Similarly, the same can be said for the Sonar dataset reconstruction of the paths in Fig. 5.37 as the size of N increases. Furthermore, it was also observed that the neuron most densely connected to for the Sonar dataset was different to the one for the Iris dataset, which also further suggests that the problems used different computational strategies.



(a) $N = 1$

(b) $N = 2$

(c) $N = 3$

(d) $N = 4$

Figure 5.38: Path analysis results for the XOR problem as the size of N is increased.

Likewise, similar results were also observed for XOR reconstructed paths in Fig. 5.38 for an increasing size of N. It also showed the pattern of new transfer functions being included in the signature of coexistence likelihood of transfer functions on a path. Additionally, the paths shown were also different from those of the other two problems, which also further suggests that these problems have different computational strategies.

In addition to the other experimental conditions, another that was tested for was the increasing levels of noise $\gamma \in \{0.1, 0.2, 0.3...1.0\}$ on both the correlation of the higher-order problem signatures (i.e. coexistence likelihood and connection strength) and the consistency of the signatures.

The noise levels were simulated by increasing the probability of Gaussian noise being added to the input pattern $x_i \in D$; as the level the probability of noise being introduced into the input patterns of the dataset increases, so does the noise levels in the dataset D. The results also showed similar consistencies with the results of increasing the population of neural network models P, and the size of the top N solutions sampled (see Appendix

A.2.2).

In addition to these results, the correlation between connection strength and coexistence likelihood was noticeable in most results, so a Pearson correlation was used to measure the degree of correlation between the two signatures. As expected, there was a significant correlation between the coexistence likelihoods and connection strengths in general as shown in Fig. 5.39.

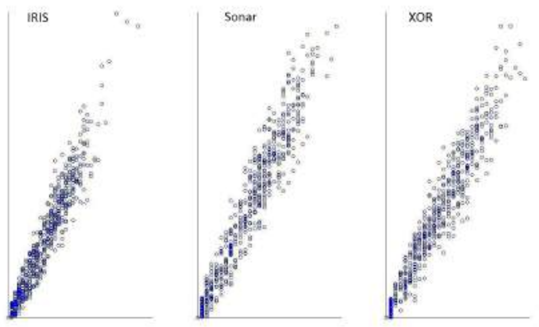


Figure 5.39: Generally, there was a correlation between coexistence likelihood and connection strength as illustrated by the scatter plot annotated with the Pearson correlation coefficient (r).

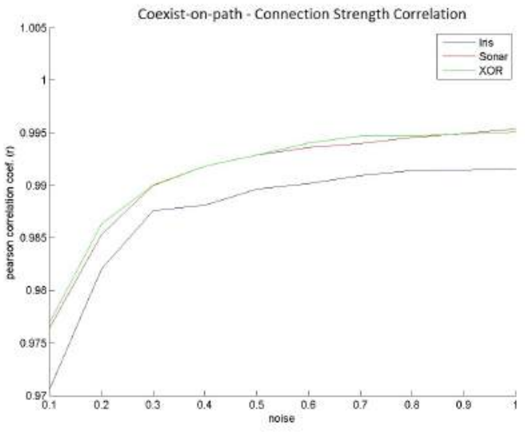


Figure 5.40: Correlation between coexistence likelihood and connection strength for increasing level of noise γ.

Another set of intriguing results is the correlation of higher-order problem signatures between problems. While they seem to have a lot of neural computational paths in common, their underlying strategies are different in terms of the set of most likely neural computational paths for the problem. The Figures 5.41, 5.43, & 5.42 illustrate that as the threshold for filtering the signatures was raised using the adjustment parameter (α) to reveal the set of most likely neural computational paths, the Pearson correlation between the problems' higher-order signatures reduced exponentially.

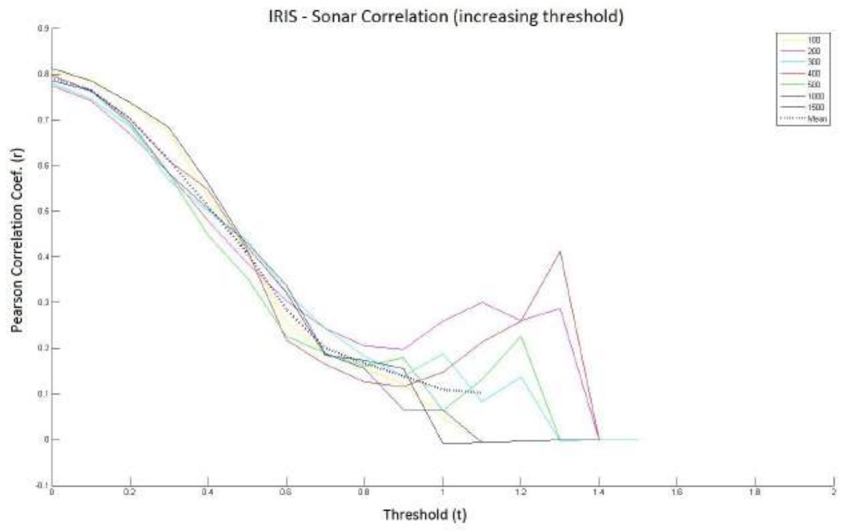


Figure 5.41: Correlations between problems as the population size and alpha are increased.

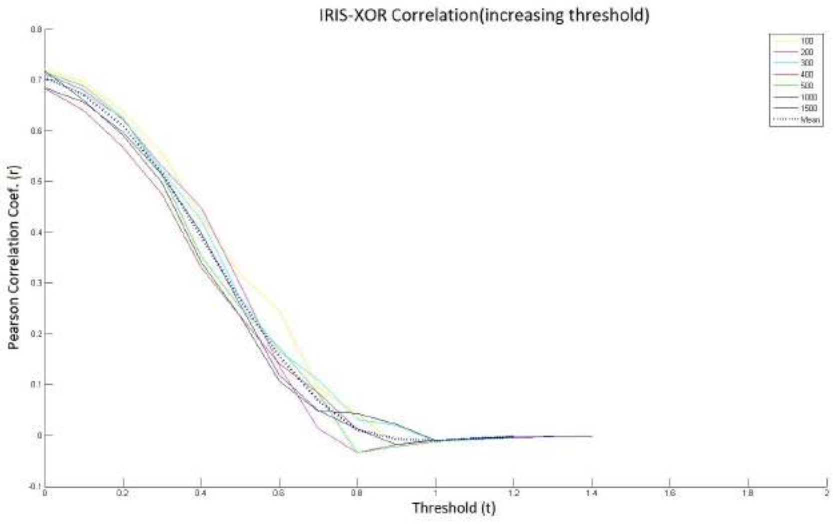


Figure 5.42: Correlations between the Sonar and XOR as the population size and alpha are increased.

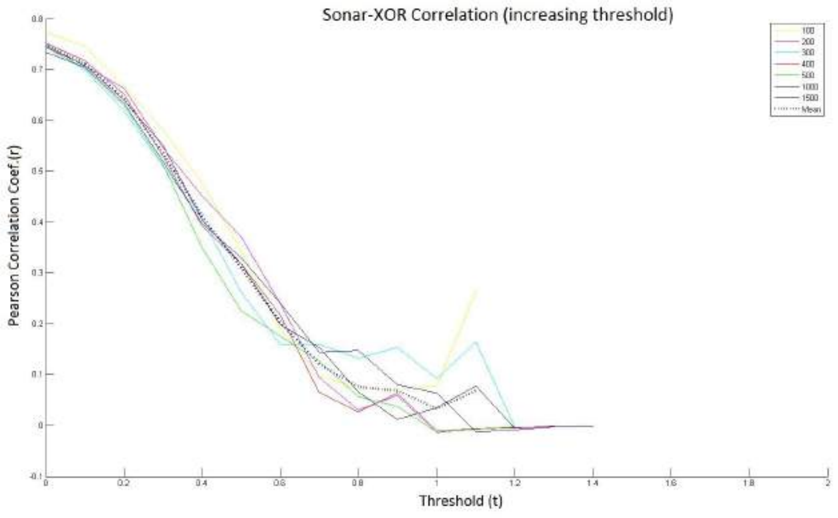


Figure 5.43: Connection strengths correlations between problems as the population size and alpha are increased.

5.8.4 Discussion

In this section, we discuss some of the major findings of the results presented. These include: the correlation between the connection strengths and coexistence likelihood matrices, the primarily neural computational paths for the Iris and XOR datasets, and the consistent and discriminatory nature of the signatures under various conditions.

5.8.4.1 Correlation between Coexistence Likelihood and Connection Strengths

As expected, there was a significant correlation of $r > 0.9$ found between the coexistence signatures and connection weights for all the datasets. Intuitively, there is likely to be a correlation because while the coexistence signatures captures the expectation of a neural computational path $E_D[A_{i,j}]$ occurring in the fittest models, the connection strength captures the cumulative weights between those connections. As such for signatures that are connected more often, their cumulative weights over the independent sampling of signatures will start to correlate. In this case, the correlation was linear for the given range of tests, as the Pearson correlation coefficient captures linear relationships (see Fig. 5.39).

5.8.4.2 Primary Neural Computational Paths

It is interesting to see a few transfer functions being preferred as was the case in the earlier results in the previous chapter. In particular, for the XOR problem, transfer functions combining an *Euclidean distance* with a *Hyperbolic Tangent*, is shown to be quite a clever solution to the XOR problem. Assuming the weights connecting to the neuron with such a transfer function is $[1,1]$ then the discrepancies between the output and target values measures to a Mean Squared Error of 0.26.

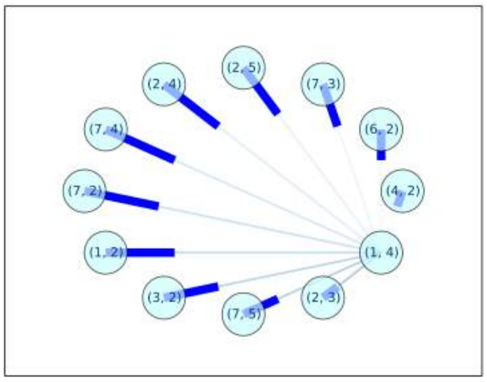


Figure 5.44: An illustration of the path analysis done on a XOR computational signature showing its neural computational paths for when only the single most elite model is used for sampling the signatures, i.e. $N = 1$.

XOR

Euclidean distance with Hyperbolic tangent:

```
weights: [ 1.  1.]
inputs: [ 0.  0.]
activation: 2.0
output: 0.964027580076
target: 0.0
error: 0.929349175147

weights: [ 1.  1.]
inputs: [ 0.  1.]
activation: 1.0
output: 0.761594155956
target: 1.0
error: 0.0568373464744

weights: [ 1.  1.]
inputs: [ 1.  0.]
activation: 1.0
```

```
output: 0.761594155956
target: 1.0
error: 0.0568373464744

weights: [ 1.  1.]
inputs: [ 1.  1.]
activation: 0.0
output: 0.0
target: 0.0
error: 0.0

*** Performance ***
- MSE:0.26
```

As illustrated above, the combination of Euclidean distance and Hyperbolic tangent had strong discriminatory property in the case of the XOR. Another example is the most densely connected to neuron that adopted the *inner product* and *hyperbolic tangent*, which is a sigmoidal unit (see Fig. 5.44). As illustrated below, the sigmoid unit also had strong discriminatory property in the case of the XOR as well.

Inner-Product with Hyperbolic tangent:

```
weights: [ 1.  1.]
inputs: [ 0.  0.]
activation: 0.0
output: 0.0
target: 0.0
error: 0.0

weights: [ 1.  1.]
inputs: [ 0.  1.]
activation: 1.0
output: 0.761594155956
target: 1.0
error: 0.0568373464744

weights: [ 1.  1.]
inputs: [ 1.  0.]
```

```
activation: 1.0
output: 0.761594155956
target: 1.0
error: 0.0568373464744

weights: [ 1.  1.]
inputs: [ 1.  1.]
activation: 2.0
output: 0.964027580076
target: 0.0
error: 0.929349175147

*** Performance ***
- MSE:0.26
```

Furthermore, The transfer functions that connected to the Perceptron the most was the combination of a *max* and *Gaussian function* (See Fig. 5.44). The behavior of the transfer function was such that the output is only weak when both values from the inputs $[x_1, x_2]$ are 1 (see below). It is possible this could be used as part of a computational strategy, in which this edge in the neural computation path successfully isolates one of the members of the class zero (as illustrated in the input-output values for the neural computational path above). Another part of this strategy could also isolate the other pattern in class zero, thus resulting in an accurate classification.

Max with Gaussian:

```
weights: [ 1.  1.]
inputs: [ 0.  0.]
activation: 0.0
output: 0.0
target: 0.0
error: 0.0

weights: [ 1.  1.]
inputs: [ 0.  1.]
activation: 1.0
output: 0.761594155956
target: 1.0
```

```
error: 0.0568373464744

weights: [ 1.  1.]
inputs: [ 1.  0.]
activation: 1.0
output: 0.761594155956
target: 1.0
error: 0.0568373464744

weights: [ 1.  1.]
inputs: [ 1.  1.]
activation: 1.0
output: 0.761594155956
target: 0.0
error: 0.580025658386

*** Performance ***
- MSE:0.17
```

Another transfer function from the subset of transfer functions that are likely to connect to it is the combination of an *Euclidean distance* with a *Gaussian* function or sigmoid output function, i.e either $(2,5)$, *or* $(2,4)$ as illustrated in Fig. 5.44. The unusual combination of *Euclidean distance* and *sigmoid/tanh* functions is usually used to generate more features from the datasets which are then normalized between [0,1] or [-1,1] depending on whether the *sigmoid* or *hyperbolic tangent* is being considered. In the case of the XOR problem, the Euclidean distance could help generate really vital information that is discriminative. If it is assumed that both the weights for the hidden unit are [1.0, 1.0], or [0.0, 0.0], then the Euclidean distance of the inputs (1,1) and (0,0) would both produce a number of even parity. The other two patterns of the dataset, on the other hand, would both produce a number of odd parity. This is one of many clever feature pre-processing techniques evolved with neural diversity; in this case, it has evolved a method of pre-processing features that has discriminative information. The Perceptron receiving such information can learn the problem on a simplified dimension of the problem. This is demonstrated by the outputs of the Euclidean distance below.

Euclidean Distance:

```
weights: [ 1.  1.]
```

```
inputs: [ 0.  0.]
activation: 2.0
parity: 0.0
target: 0.0
error: 0.0

weights: [ 1.  1.]
inputs: [ 0.  1.]
activation: 1.0
parity: 1.0
target: 1.0
error: 0.0

weights: [ 1.  1.]
inputs: [ 1.  0.]
activation: 1.0
parity: 1.0
target: 1.0
error: 0.0

weights: [ 1.  1.]
inputs: [ 1.  1.]
activation: 0.0
parity: 0.0
target: 0.0
error: 0.0

*** Performance ***
- MSE:0.07
```

Apart from the parity information, it is also easily noticeable that the activation after calculating the Euclidean distance, once passed to the hyperbolic tangent is also discriminative, and actually classifies the problem somewhat accurately.

Iris

In the case of the Iris dataset, the transfer function with the most connection density utilized the combination of max input combination and Gaussian output function. This has been found to have filter-like properties which utilize the property of un-normalized features to filter out a particular attribute that seems to contribute towards the final classification - as presented in our previous work [4]. Specifically, this has been found to filter out the age of people for diabetes prediction. Interestingly, aging is one of the factors associated with increased risk of type-2 diabetes [10]. In this case, its is likely that the transfer function is also filtering out a particular activation of interest and learning the distribution of the attribute using the Gaussian function. This could be very useful if the attribute has a high correlation with the final classification.

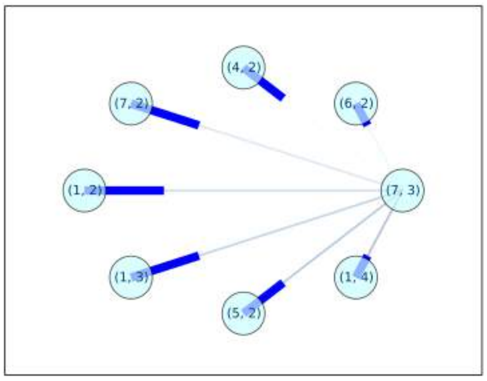


Figure 5.45: An illustration of the neural computational paths for the Iris dataset, extracted from the signature with path analysis where the size of the top $N = 1$.

5.8.4.3 Consistency with population size *P*

Regarding consistency, the patterns portrayed in the signatures did not change significantly when the number of populations used for signature extraction was significantly modified for the range between [100 .. 1500]. Each member of the population represents a point in the search space, and as such this represents the number of points spread around the search space H_ℓ for sampling using the learning algorithm ℓ. Due to the diversity of transfer functions of the artificial neural networks used, these points are more than likely to be relatively more spread out as compared to that of a canonical neural network using only a single class of functions (e.g. inner-product and sigmoids) as their transfer functions. The patterns illustrated in Fig 5.27, 5.28, and 5.29 show consistency in the pattern of neural computational paths as shown by the heat map.

Visual inspection of the graphs also revealed other interesting observations: the expectations of neural computational paths seem to slightly differ between each other as the threshold is increased (see Fig. 5.27, 5.28, and 5.29). Increasing *P* basically increases the points in the search space considered during the sampling, which gives a wider range of computational strategy variants to be considered. As such, as long as the error of the local minima from the subspaces is not optimal, it gets more likely that a local minimum that is more optimal is found as the size of P is increased. This is also partially as a result of the randomness of weights and the transfer function initialization, which means the sub-spaces vary with each run. In addition to the law of central tendency, there is likely going to be an expected neural computational strategy for each problem depending on their biases (i.e., the range of weights, transfer functions, and the input space).

Another observation that can be made is that the features of the signatures become more pronounced (i.e. thickened) as the size of the population P used for sub-sampling is increased (see Fig.5.27, 5.28, and 5.29). However, this is to be expected when the sampling points for the signatures is increased as the expectation for each of the neural computational paths is likely to gain more density (i.e. thickness).

Furthermore, another interesting observation is that the pattern of expected co-existence of connectivity for some transfer functions - which produces the vertical lines on the heat map - consistently appears for all the cases of P.

5.8.4.4 Consistency with size of N

With regards to sampling from a wider range of fitness values, i.e. considering the top N solutions, where $N \in \{1..4\}$ is the number of solutions to be selected from the population after sorting (by fitness), the core signatures remained the same but additional neural computational paths were introduced each time N was increased (see Fig. 5.36, 5.37, and 5.38). This suggests that the various models of ℓ not only have varying fitness values, there are differences in the computational strategies as well, however the primary neural computational paths (or building blocks) remained the same (see computational signatures: Figures 5.33, 5.34 and 5.35). This is also an indication that the error surface over the computational strategy space is made up of diverse computational strategies. This could be desirable for a number of reasons including a wider coverage of the computational strategy search space.

5.8.4.5 Consistency with Noise levels γ

In terms of the result with various noise levels, there were also similarly interesting findings: firstly, some of the datasets seemed to adopt a computational strategy that was different at lower noise levels, and then appears to converge as the levels of noise in the dataset was increased (See Appendix A.2.2). Secondly, another interesting finding is the effect of increasing the correlation between the signatures as the level of noise was increased (see Fig. 5.40). It is likely that the noise generated from a normal distribution distorted the input space, thus making the problem-space less distinct.

In the case of the Iris dataset, the neuron most densely connected to seemed to assume a transfer function that was not consistent with the earlier results; instead it assumed $(2,2)$, which is the combination of an *Euclidean distance* and a *sigmoid* output function. As the noise level was increased, this progressed to adopt a transfer function $(7,2)$, which combines the usage of a *max* and *sigmoid*. Finally, it seemed to converge on the transfer function discussed in the earlier sections as the neuron with the most input connections density among the neural computational paths; i.e. $(7,3)$, which is the combination of a *max* and *Gaussian* function.

A similar result was also observed in the case of the XOR dataset, which had inconsistencies in the first two levels of noise $\{0.1,0.2\}$, which was then more consistent as with the earlier results as the noise level increased (See Appendix A.2.2). Specifically, it converged on the neuron with the most input connection density neuron from the neural computational path as $(1,2)$, which is a *sigmoidal* unit. Interestingly, this was from the same family of transfer functions as the one in the earlier section (projection-basis units), which used an inner product and hyperbolic tangent $(1,4)$.

It is unclear how lower noise levels cause the perceived discrepancies observed in the results. One possibility is that the noise might have distorted the input space just enough to make the learners less sensitive to the patterns of the dataset, thus making them less prone to over-fitting.

Another interesting observation pertained to differences in connection strengths of the neural networks which lead to a slightly different arrangement of the neural computational paths as compared to those observed in the results of other consistency experiments (see Appendix A.2.2). In the case of the XOR dataset, the neural computational paths that contributed to this density in the previous section were mostly present in the neural computational paths presented by these results (See Appendix A.2.2). This was also the same for the Sonar and Iris datasets. However, the connection strengths for the Iris neural computational paths seemed to have remained consistent in terms of their connection strength as in the earlier results (see Appendix A.2.2). Variations in weights reflected by the connection strengths should be expected; elements such as weights which have a high-dimensional space compared to the rest such as choice of transfer functions or topological pre-disposition, are bound to exhibit lots of variation. In contrast, the finding that the connection strengths of two different conditions closely matches is a significant one, due to the complexity involved (i.e., the search space) and the sensitivity of neural networks. It makes randomness an unlikely explanation.

In a nutshell, it seems to be the case that there is a small range of noise that interferes with the consistency of the neural computational paths. However, in general, the signatures remained consistent with the ones from the other tests which presents a significant finding in itself. It seems like the Law of large numbers has a role to play in the consistency of signatures. Specifically, as more randomly generated models are sampled, the expectation of the neural computation strategy appropriate for that problem - given the constraints (i.e. learner, the range of weights and other biases) - converges to an average expectation which might represent a local minimum. This is given a robust sampling method which samples diverse computational strategies, i.e. a sampling method that is not just sampling similar strategies.

5.8.4.6 Discriminatory Property between Problems

In general, the Pearson correlation values for the Iris, Sonar, and XOR datasets consistently decreased as the threshold was increased (i.e. α). One might argue that this was as a result of the variance in the models - as was the case for consistency with regards to the population size being increased (See Fig. 5.27, 5.28 and 5.29). However, there were differences in the primary neural computational paths used for all the problems. While the Iris dataset primarily relied heavily on neural computational paths involving the *max* activation function and *sigmoid* output function; the sonar differed by using a combination of an *Euclidean distance* activation function and a *sigmoid* output function. Likewise, the XOR problem also differed by using a combination of an *inner-product* activation function and a *hyperbolic tangent* output function, in other words, a simple sigmoidal unit. Additionally, they also differed generally from the set of neural computational paths ending with these neurons.

Transfer functions and neural computational paths are bound to vary between problems as long as the input spaces of the problems are different. This is because of the bias induced by an input space, which influences the fitness values of the set of hypothesis H_ℓ adoptable by the learner. In other words, it defines the error surface for the transfer functions, the connectivity between them and their weights. In this case, we have explored the neural computational paths of problems, which are the building blocks that help project hypersurfaces that could hypothetically match the underlying function of the problem $f(.)$.

Among the neural computational paths with most coexistence likelihood and connection strength for the problems are the following.

Sonar

- (1,4) - inner product and hyperbolic tangent (projection-basis unit)
- (2,3) - Euclidean distance and Gaussian (Radial-bais unit)

- (1,3) - Inner product and Gaussian

Iris

- (1,4) - inner product and hyperbolic tangent (Projection-basis Unit)
- (5,2) - Standard deviation and Sigmoid
- (1,3) - Inner product and Gaussian

XOR

- (2,3) - Euclidean distance and Gaussian (Radial-basis unit)
- (7,5) - Max and Gaussian II
- (3,2) - Higher order product and Sigmoid (Higher Order Unit)

In the case of the Sonar dataset, it was interesting to see that the traditional projection-basis and radial-basis units were among the set of neurons contributing most according to their connectivity weight. There have been various works [55, 30, 31, 49] that have explored using both of these transfer functions motivated by the work by Donoho (Donoho, 1992) on the theory of duality of functions, which showed that any continuous function could be decomposed into a projection basis and a radial-basis component. It is possible that this might be exploited here to learn the underlying function. Apart from that, there was the unusual combination of an *inner-product* and a *Gaussian* output function.

In the case of the Iris dataset, there was also the familiar projection-basis function among the subset of the top three selected from the set of likely neural computational paths. However, there was the unpopular combination of standard deviation and the *sigmoid* output unit. The standard deviation has been found to be evolved as an input combination method that behaves in a manner that averages the inputs, as reported earlier in the previous chapter.

Finally, in the case of the XOR dataset, there was on transfer function apart from the familiar radial-basis unit, namely *max-Gaussian*, which uses the max function to isolate the input feature with the highest activation value. This is then passed to a *Gaussian* function. In the previous section, this was found to produce discriminatory information about the problem, which had a notable correlation to the problem's classes.

5.8.5 Conclusion

In conclusion, we have shown the resilience of higher-order computational signatures for a wide spectrum of conditions considered; specifically, changing noise levels, the number of sampling points, and the number of models sampled. Given that the models in the population are randomly initialized, and not trained before sampling, intuitively, one might expect randomness. However, the signatures were shown to be consistent. This is significant due to the complexity involved, particularly, the dimensionality of the search space. However, by the central limit theorem, as more samples are taken, a system will tend to converge on a normal distribution, with a mean and variance. In our case, the mean accounts for the consistency in the signatures, while the variance accounts for the randomness.

Furthermore, it was also shown that higher-order problem signatures of the different problems shared neural computational paths between problems; however, the primary neural computational paths used for each problem differed. Additionally, this was supported by the results shown by increasing the threshold value to filter out more neural computational paths that were less likely to appear in the fittest models for the problem. In other words, given the dataset D, the neural computational paths filtered were those that satisfied the given criteria: $p((f_x, f_y)|D) < \mu + \alpha$. This enabled us to visualize the more primary neural computational path used for each problem. Furthermore, path analysis was applied to these signatures, which enabled us to confirm that their primary building blocks were different. They had different sets of neural computational paths, which meant it was unlikely that they were doing similar computations. Contrary to that, they were adapting their hypotheses to the nature of the problem's hyperspace. In conclusion, it could be said with a high degree of confidence that higher-order problem signatures are discriminative. The significance of this finding, as highlighted earlier is that this presents a significant contribution in terms of meta-features specifically for artificial neural networks utilizing a pool of transfer functions. This could be applied to help address the dimensionality problem that arises from transfer function optimization. It could also be extended to topology selection as well.

Chapter 6

Co-evolution of Neurally Diverse Neurons and their Topology

In the earlier chapter, we explored various ways of analyzing the underlying portions of the computational strategy of problems. This was used to analyze the computational strategy for the Iris, Sonar, and XOR datasets.

This chapter presents a holistic approach to optimizing neurally diverse artificial neural networks with their topology and weights by co-evolution. As explained chapter relating neural diversity to generalization, a neural network can be robust and achieve good generalization, if it is able to adapt its bias to the nature of the problem. This was the conclusion of one the works that explained bias-variance decomposition in relation to neural networks [45]. We present a coevolutionary optimization approach as a means of evolving more adaptable neurally diverse artificial neural networks.

This chapter presents a component-wise representation that enables the adaptation of the three major components responsible for learning in artificial neural networks. These are the transfer functions, weights and topology. These components are interdependent of each other, and each influences the error of the other components. This chapter presents a more holistic approach to optimizing transfer functions which considers other components. This differs from the work on NeuDiME primarily from the representation aspect which incorporates the interdependence of components explicitly.

The following sections present the motivation, problem, related works, contribution, and overview of this chapter.

6.1 Motivation

There are various motivations for proposing a component-wise representation to optimizing neurally diverse transfer functions.

First and foremost, it presents a holistic approach to optimizing the transfer functions of the artificial neural network, which considers both the role of other components, as well as their interdependence to each other.

Secondly, the component-wise representation of the proposed co-evolutionary approach reduces the risk of eliminating a solution with a portion of the solution that is promising. In traditional encoding, due to the encoding of the components onto a single genetic string, portions of the solutions evolved in solutions with poor fitness values might be eliminated or dominated. This is because the fitness value of a single-genetic string solution overly generalizes the fitness of the individual components encoded onto the genetic string and as such, is not truly representative of their individual fitness values. A co-evolutionary optimization approach reduces this bias, thus preserving these portions of the solution.

Additionally, cross-overs are somewhat tricky and have been regarded by some as detrimental to the evolution of artificial neural networks [74, 98, 34, 12]. This is because operations such as removing nodes in particular represent significant changes to an artificial neural network's computational strategy, which results in a significant effect on performance. However, we believe crossovers still remain one of events that underpins the central theme of evolutionary algorithms; information transfer. This view is also shared by other neuroevolution researchers such as Xin Yao [109, 107, 108] among others [46, 78, 44], who use cross-overs in their work. It can be said that cross-overs needs to be used with appropriate representations of problems. In the case of traditional encoding (used in NeuDiME), which does not differentiate components, it is more intuitive that randomly selecting points of cross-over will not yield the most ideal results. This is so for a number of reasons; firstly, because each component of the genetic string does not have a fitness, there is no way of telling which component encoded onto the genetic string has useful information. Secondly, while the cross-over might have the chance to transfer some information that is useful, it might also package this with other information from other components that are causing issues with learning when selecting the cross-over point. This is because of the random nature of cross-over point selection. However, a component-wise representation explicitly represents fitness of a candidate component, and its information. Thus, making cross-overs more effective as they are more informed.

In addition, the component-wise representation of the neural network also paves the way for future works such as sampling of signatures, architecture initialization and complexification without the need for the computational overhead of finding the genes of a particular component as in single-genetic string representations.

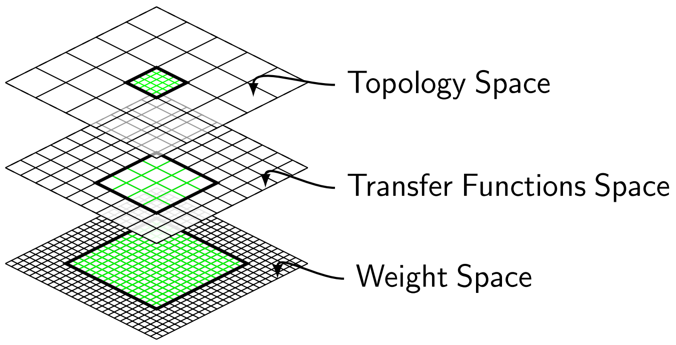


Figure 6.1: In some cases of fixed network size and topology, bias can accumulate.

6.2 Compounding Bias and Information transfer

In the previous section, we explained some of the consequences of encoding individual components of an artificial neural network onto a single genetic string, which includes overly generalizing the fitness values of the individual components. In this section, we explain with more detail other consequences such as its bias compounding effect and effects on information transfer capabilities.

Artificial neural networks are made of components that depend on each other to approximate a hypothesis of the underlying function of a problem. These dependencies result in the bias of the components compounding to form a bias that forms the overall bias of the learner as explained in the earlier chapter (Chapter 3). This can be a problem if there is no efficient method of transferring learned information between members of the population. Evolutionary operators that facilitate information transfer between solutions include cross-over and other variants, which factor in the fitness of the solutions to determine the probability of crossing over individual genes [40]. The canonical cross-over typically involves picking random cross-over points, which are then used as cut-off points to determine which genes are transferred to the offspring. In the case of a single genetic string representing the neural network, cross-over operations can do more harm than good. In fact there are some that discourage the use of canonical cross-over operations due to their damaging effects [12, 34]. These effects occur when learned information such as the connections between neurons are pruned. This leads to a change in the neural computation strategy, which could possibly result in degraded performance [12, 34].

However, this is not purely a cross-over issue, it could be regarded to be a problem of representation. In particular, the representation of neural networks onto a single genetic string might not be the most optimal in terms of information transfer. Firstly, as the size of a neural network grows, so do the genetic strings, which leads to an increased dimensionality of the problem search space. Subsequently, we also speculate that this makes it harder to transfer learned information more efficiently. A view which has also been shared in

[78, 44]. This is because if there is an optimal cross-over point for transferring a particular chunk of learned information, the chances of making this optimal cross-over of learned information from the parents to their offspring gets more unlikely as the length of the genetic string is increased. In other words, the representation might not be scalable. An ideal representation of a complex system such as a neural network might be to encode components onto corresponding genetic strings that represent their parameters in such a way that the cross-over operation is more invariant to the length of the genetic string, and consequently, the complexity of the neural network. In addition to that, context is also maintained in the representation of the neural network.

Co-operative co-evolution of sub-components by Porter presents a representation [85, 86] that sheds light on a co-evolutionary optimization framework which co-evolves components of a complex system. In particular, each of the components such as the weights, neurons, and topologies of the neural network have a dedicated population of candidates which represents their respective candidate parameters. This is likely to provide some degree of efficiency in cross-over operations which enables them to cross-over information about a component without changing other components. In particular, this can be particularly suitable for neural networks due to their sensitive and complex nature - as Porter also suggested [86, 85].

6.3 Related Works

In the terms of co-evolution of neural network sub-components, some of the related works include SANE [78], COVNET [44], and CoSyNe [46]. Another which uses a slightly different approach, i.e. Evolutionary Programming, is EPNet [111].

COVNET [44] is made up of a population of sub-networks that the authors described as *nodules*. A nodule is composed of a set of artificial neurons that can freely connect to each other, in addition to the input and output layer, thus creating sub-networks in the hidden layer. A population of neural networks then adopts any number of these nodules to form a neural network model. However, there are restrictions on inter-nodule connections. In [44], a co-evolutionary approach was used cooperatively to evolve the sub-populations of networks and nodules until a suitable model was learned. Evolutionary operations such as mutation introduced the variations needed to traverse the search space. In particular, mutation operations included the addition or pruning of artificial neurons or connections in nodules. Additional mutation on parameters such as the weights or functional parameters of the transfer functions used simulated annealing to tune these parameters locally. The results showed that COVNET produced neural networks that were relatively more compact with competitive results.

Another related work is SANE [78], which is an acronym for Symbiotic Adaptive neural evolution, which co-evolves a dedicated population of candidate neurons for each node position in the hidden layer [78] - as opposed to sub-networks as in COVENET [78]. Individual neurons are co-evolved with other hidden neurons to form complete neural networks. A genetic algorithm is used to optimize these populations. Each sub-population in SANE is a partial solution to the problem. Individual neurons from each sub-population have to maintain a symbiotic relationship with neurons from other sub-populations to achieve higher fitness. The fitness of an individual neuron is the average fitness of the individual in $n : n \in \{1,2,3...\}$ random subsets of complete solutions it participated in. Interestingly, the partial solutions were found to speciate and specialize towards different aspects of the problem. This was also found to promote diversity, as well as prevent premature convergence. Evaluation starts with selecting random neurons from each species(sub-population) to create a network model, which is then evaluated. Once the neurons have participated in a sufficient number of networks, their fitness is evaluated and assigned to them.

Cross-over is then performed with the fittest neurons to create variations. Afterward, the neurons are ranked based on their fitness for selection. Selection by rank is then used to ensure bias towards best performing neurons. This was tested on the pole-cart problem, with a network that had two-layers of weights. The results suggested that symbiotic evolution was efficient at genetic search without reliance on high mutation rates. Thus, further indicating more efficient traversal of the model space. Also, it also achieved fast, and efficient learning.

EPNet [111] used a hybrid algorithm for training the neural networks, which consisted of a modified back-propagation (MBP) and an evolutionary programming algorithm. The modified BP had an adjustable learning rate which worked in a hill-climbing approach using the networks error as a reference. EPNet emphasized on maintaining the link between parents and offspring to make evolution more effective. This was done by introducing evolutionary operators such as node splitting, which grows the hidden layer by making a slightly modified copy of a selected neuron. This was intended to cause as little disturbance as possible to the computational strategy being converged upon. Preference was given to connection and node deletion to bias towards architecturally simpler models. The networks were trained on the train set using the MBP, evaluated using the validation set to prevent over-fitting, and tested for generalization ability on the test set. Like the other related works, it was also found to evolve relatively small networks with sometimes novel solutions. However, the author mentioned concerns over the user specified parameters being excessive in number.

Finally, a work that revolved around co-evolving weights for a fixed topology in a predetermined neural network model was CoSyNE [46]. Each connection in the topology had a sub-population of potential weights, which were initialized uniformly for a given range, $[w_{min}, w_{max}]$. One of the distinct contributions of this work was to probabilistically permutate the weights which are likely to result in a different neural network model, thus increasing the spread of the search, as well as addressing the issue of premature convergence.

6.4 Contribution

Though the works in the literature such as SANE, COVNET and CoSyNE contribute to artificial neural networks by presenting different representations and optimization approaches to co-evolution of artificial neural networks, there is yet to be work on co-evolution which also optimizes the transfer functions of the artificial neural network. In this chapter, we present experiments on a holistic approach to transfer function optimization by co-evolution of neurally diverse neurons, and their weights and topologies, i.e. the three major components of artificial neural networks.

This work is motivated by neuronal diversity and as such is a biologically inspired approach. This diversity was achieved by forming a set of of different classes of functions, namely: projection basis functions, radial basis functions, higher-order functions, and statistical functions. However, there were no restrictions on the transfer functions that could be exhibited since any activation function (input combination function) could be paired with any output function to produce a transfer function from 35 different possible combinations for any of the neurons in the hidden or output layer. As such there are a variety of transfer functions that are exhibited during the course of the cooperative co-evolution.

Another distinguishing feature of our work lies in how it represents the hypotheses of a model space specifically for artificial neural networks, which consists of three interdependent sub-spaces: weight space, topology space, and neuron space. This was represented as three sub-populations, each consisting of smaller sub-populations within them for specific components of the network. In other words, in a sub-population of potential weights W, there is a niche for the sub-population of weights for the input-to-hidden layer w_{ih}, hidden-to-hidden layer w_{hh}, and hidden-to-output layer w_{ho}. Finally, there are other differences in the methodology of optimization (see section 6.6.2). In particular, the method of fitness assignment for each of member of the sub-populations used was as proposed in [85, 88].

Additionally, cross-validation was used for selecting the sub-components of a neural network model. This was done by assigning a tuple of errors to the components (i.e. neurons, weights and topologies) that represented their performance on both the training and validation set, and selecting using tournament selection based on this fitness tuple. This helps in selecting components associated with good generalization ability.

6.5 Chapter Overview

The rest of this chapter is organized as follows; firstly, the next section introduces co-operative co-evolutionary neurally diverse networks. This consists of subsections that describe the relation to generalization, and how networks are represented and optimized. This is followed by the experimental sections which highlight, and discuss the results of the co-evolutionary neurally diverse networks. This also includes comparisons to NeuDiME, and other neural network models such as the classic Multilayer Perceptron and Radial Basis Function Network implemented by pyBrain [93]. Various types of support vector machines (SVM) implemented in mlpy [8] were also included in the comparison. Finally, the results also include comparisons to some of the state of the art learning algorithms in the industry, specifically those made available by Microsoft on their machine learning platform, *Microsoft Azure*.

The is followed by a section that presents the significant generalization effects of injecting random transfer functions to sub-populations of the potential neurons.

6.6 Co-operative Co-evolution of Neural Network Sub-spaces

Neural networks work towards approximating the underlying function that describes a problem by making their components work together towards finding a model in the model space that has a hypothesis, which closely describes the nature of the problem being considered. As described in Chapter 3, the hypothesis space accessible to a neural network is limited by its bias - which represents the bias of all of its components as well as its training routines and constraints. In this chapter, we describe a model space of artificial neural networks consisting of three major components; i.e. weights, topology and transfer functions. There are various works that have described the model space of learning algorithms [75, 103, 102] from a more theoretical perspective. This chapter contributes by attempting to consolidate the discussion in chapter 3 by showing the practical implications of reducing the bias of artificial neural networks with diverse transfer functions. This was done by defining a representation of the model space representing the three major components:

weights, topology and transfer functions which are interdependent of each other due to their influence on each other's error-surface, and subsequently their fitness. In other words, the fitness values of these major components are interdependent. These three sub-spaces (topology space, weight space, and neuron space) indirectly form another search space; the model space M (Figure 6.2), which contain instances of the elements of the abstract hypothesis space H. We further adopt the general notion in theoretical works which have established that the model space has a many-to-one relationship with the hypothesis space [75, 103, 102]. This is because a variety of models with slightly different weights - which constitutes a set of different model instances $m_1(X)...m_n(X)$ - could actually all be an approximation of the same hypothesis $f_i(X) : f_i(X) \in H$ in the hypothesis space. In addition, the set of models M_l accessible to the neural network l is determined by the pressures of the accumulated bias of the neural network, optimization algorithm and their constraints on the model space as expressed earlier. In other words, depending on the pressures of the bias, not all elements of the global model space M , might be contained in the set of accessible models for the neural network M_l. Therefore, it is just a subset of the global model space, i.e. $M_l \subset M$.

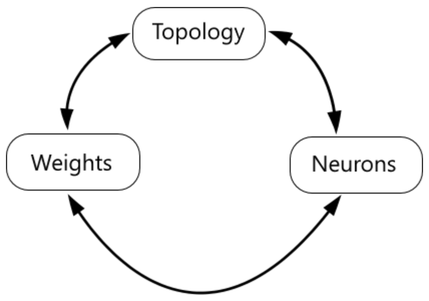


Figure 6.2: The interdependence of the three sub-spaces (i.e. weights, topology and transfer function) that make up the neural network's *model space*.

Representing the model space with the defined components (i.e. weights, topology and transfer function) should help to increase the ability of the neural networks to explore more promising regions of the search space as a result of the component-wise representation, and subsequently increase the likelihood of converging on promising regions of the hypothesis space. This is because the representation assigns to each candidate component it's respective fitness value. This enables candidate components that cooperate towards better generalization to be preserved. In addition, the representation can also provide more consistent performance with operations such as cross-over as the complexity of the neural network increases. This is particularly a problem for single-genetic string representations, where the length of the genetic string is affected by the complexity of the model, and could indirectly affect the cross-over performance. However, in a component-wise representation

Table 6.1: List of the activation function set. This includes additional activation functions such as the Manhattan distance (also known as Taxicab distance) and Max distance (also known as Chebyshev distance). In addition, there was also the mean activation function.

Activation Functions
Inner-Product ($j = \sum_i^k w_i i_i + w_{bias}$)
Higher-Order Product ($j = \prod_i^k cw_i * i_i$)
Higher-Order Subtractive ($j = \sum_{i=1}^k \lvert x_0 - x_i \rvert$)
Euclidean Distance ($j = \sqrt{\sum_i^k (w_i - i_i)^2}$)
Manhattan Distance ($j = \sum_i^k (w_i - i_i)$)
Max Distance ($j = max(\lvert w_i - i_i \rvert, \lvert w_{i+1} - i_{i+1} \rvert ... \lvert w_k - i_k \rvert)$)
Standard Deviation ($j = stdDev(w_i i_i, w_{i+1} i_{i+1} ... w_k i_k)$)
Mean ($j = mean(w_i i_i, w_{i+1} i_{i+1} ... w_k i_k)$)
Min ($j = min(w_i i_i, w_{i+1} i_{i+1} ... w_k i_k)$)
Max ($j = max(w_i i_i, w_{i+1} i_{i+1} ... w_k i_k)$)

Table 6.2: List of the output function set. The additional output function here was the Arc tangent, which is an inverse of the hyperbolic tangent.

Output functions
Linear ($z = \alpha * j$)
Hyperbolic tangent ($z = \frac{1-e^{-\alpha * j}}{1+e^{-\alpha * j}}$)
Arc tangent ($z = tanh^{-1}(\alpha * j)$)
Sigmoid ($z = \frac{c}{1+e^{-\alpha * j}}$)
Gaussian ($z = e^{\frac{-(j)^2}{width}}$)
Gaussian II ($z = e^{\frac{-(j)^2}{width}} if\, z > \theta\, then\, z = 1.0$)

of the genetic strings, the length of the genetic string is invariant to the complexity of the neural network, as it is traded-off for additional sub-populations.

6.6.1 Representation

The representation offered by the cooperative co-evolutionary framework, also known as Cooperative Co-evolutionary Genetic Algorithm (i.e. CCGA) [86, 87], provides an explicit representation of the elements highlighted above. Specifically, it makes it possible to co-operatively co-evolve the sub-spaces of the three components, i.e., neurons, topology, and weights. In our work, the three components of the neural network - neurons, weights, and topology - were cooperatively co-evolved using the framework presented in [86, 87]. An evolutionary algorithms library in python, DEAP [43], was used to implement this framework.

Once the size of the hidden layer, $H \in \{1,2,3...\}$, is decided before optimization, a sub-population of candidate neurons is created, where the number of sub-populations is equal to the size of the hidden layer. Each sub-population contains potential neurons that could be used for a particular neuron position of the hidden layer. Likewise, for each of the neurons in the output layer, a sub-population of candidate neurons is also created, where the size of the output layer is $O \in \{1,2,3...\}$. Fig. 6.3 shows an illustration of this.

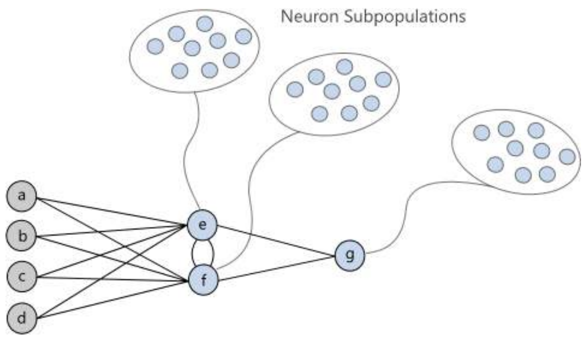


Figure 6.3: The sub-population of diverse candidate neurons created for each hidden and output unit.

At initialization, the neurons are allowed to select any combination of input combination functions (or activation functions - Table 6.1) and output functions (see Table 6.2). This enables a variety of transfer functions to be exhibited during the course of the optimization, some of which were found to act as filter-like functions that relay important information. The information encoded onto genetic strings representing the neurons includes the parameters of the transfer functions (i.e. the functional parameters). In addition, it also encodes the bias value of the neuron (which is by default set to 1) and it's bias weight as illustrated in Fig. 6.4.

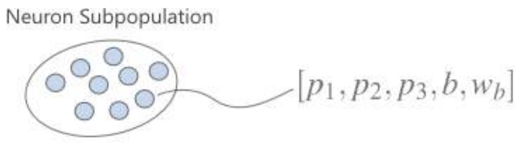


Figure 6.4: The information encoded for candidate neurons.

Likewise, there are sub-populations initialized for the candidate topologies and weights between the layers, i.e. input-to-hidden, hidden-to-hidden, and hidden-to-output layers. In other words, each layer has a population of candidate weights and topologies as shown in Fig. 6.5 and Fig.6.6.

6.6.2 Optimization

The optimization starts off by initializing the members of each sub-population by generating random candidates to populate each sub-population. Afterward, a representative is selected from each sub-population randomly as the population's representative. The set

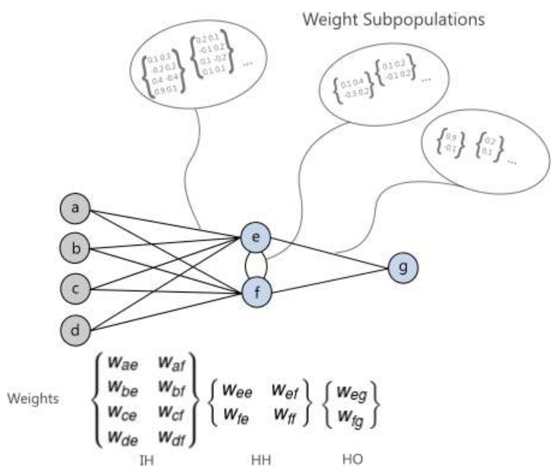


Figure 6.5: The sub-population of weights created for each layer in the neural network model, i.e. input-to-hidden (IH), hidden-to-hidden (HH), hidden-to-output (HO).

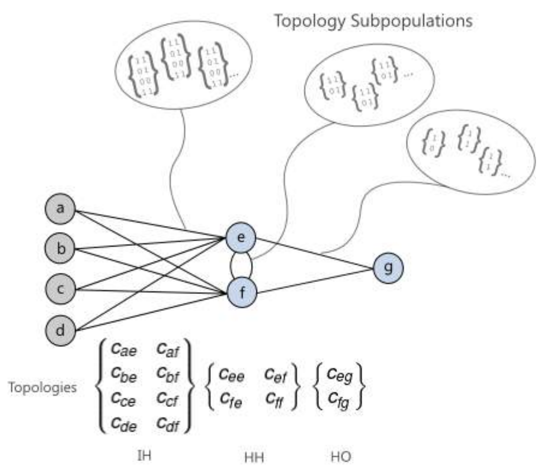


Figure 6.6: The sub-population of topologies created for each layer in the neural network model, i.e. input-to-hidden (IH), hidden-to-hidden (HH), hidden-to-output (HO).

of representatives R from each sub-population - which constitutes all the components for building a complete neural network model- is then used for evaluation. To evaluate any member m_i (i.e. candidate component), the member substitutes its respective representative $r_i \in R$. The new set of representatives R' are then used to build a model, which is evaluated on the dataset. The fitness of the model is then assigned to the member m_i. This is done for all the members $m_j \in c_i$ of the sub-populations $c_i \in C$.

For each a sub-population $c_i \in C$, this process is repeated until each member of the sub-population has been assigned a fitness. This is followed by evolutionary operators such as mutation and cross-over - as illustrated in Algorithm 3. The offsprings resulting from the operators are also evaluated. This is followed by a tournament selection which then selects the best member of the sub-population as the new representative.

Algorithm 3 The underlying process of the cooperative co-evolutionary algorithm. This was proposed by Potter and De Jong in [87] as a co-evolutionary framework, which they explained could be useful for learning algorithms such as neural networks.

```
Require: $C = initComponentCandidates()$
Require: $R = pickRandomComponents(C)$
  for $i$ : $maxGen$ do
    for sub-population/component $c_i \in C$ do
      for candidate $m_j \in c_i$ do
        $R[r_i] \leftarrow m_j$ {replace representative $r_i$ with candidate $m_j$}
        $(f_t, f_v) \leftarrow evaluate(R)$ {evaluate train and validation errors}
        $x_i.fitness \leftarrow (f_t, f_v)$ {assign train and validation errors to component $m_j$}
      end for
      $nextGen \leftarrow crossOver(c_i)$
      $nextGen.extend(Mutate(c_i))$
      $nextGen.extend(generateRandomCandidates())$ {injects randomly gen. members to candidates}
      $nextGen \leftarrow evaluate(nextGen)$ {evaluate component candidates}
      $c_i \leftarrow tournamentSelect(nextGen)$
      $sort(c_i)$ {sort the components by their training and validation errors}
      $R'[r_i] \leftarrow c_i[0]$ {Pick the best component candidate as new representative}
    end for
    $R \leftarrow R'$ {replace old representatives with new representatives}
  end for
```

The sub-population of weights representing the layers are mutated by Gaussian mutation. Likewise, neurons were mutated using Gaussian mutation, which generates a random real value from a Gaussian distribution with mean μ and variance σ to add to a gene value. The topologies on the other hand are mutated using flip-bit mutation, which switches on connections by flipping on/off the connections between the layers, represented as connectivity matrices. These mutations were carried out based on an independent probability, which determines the probability of mutating the independent genes of the parents. There was no explicit exchange of information between the sub-populations such as a cross-over. These were restricted to only intra-population two-point cross-over. The restriction was intended to help preserve the learned information's context within its respective sub-populations. Specifically, because each subpopulation represents the three components of the neural network defined earlier, i.e. weights, topology and transfer functions; it is essential to maintain the context of the information by not subjecting members of different sub-populations to cross-overs.

In the case of neurons, a two-point cross-over was carried out on the string of real numbers representing the functional parameters of the transfer functions, the bias value of the neuron and its bias weight. This provided a mechanism for information sharing between neurons of the same sub-population. However, these were also restricted to cross-over with candidate neurons of the same neuron subpopulation and as such candidate neurons meant for a particular hidden unit in the hidden layer did not cross-over with another sub-population of neurons meant for another hidden unit. Topologies were also crossed-over with other candidates within their respective sub-populations, and this was the same for the sub-population of weights. This was meant to preserve learned information within the sub-population, and enable the sub-populations to specialize towards converging on their respective part of the solution. Like the flip-bit mutation operation, cross-over was also carried out based on a probability of crossover set prior to optimization.

The next-generation of members for each subpopulation was selected based on the training error and validation error of the members after sorting. This is about the same as selecting the members of the Pareto-front, which have relatively better fitness on both training and validation error. The members of the populations were selected from both the parents and offspring of the population. The advantage of selecting based on both the training and validation error of the components is that it reduces the chances of keeping elements of the neural network model that might contribute toward over-fitting or under-fitting the problem. This is because members that contribute towards overfitting or under-fitting the problem will tend to have a weaker validation error. It is also meant to purely keep the candidate components that cooperate towards a model that adopts a strategy with good generalization ability which is characterized by strong training and validation errors. Tournament selection (see Table 6.4) was used to achieve this without risking premature convergence as a result of the selection pressure (i.e. bias towards solutions with better training and validation error). Tournament selection relieves this pressure by incorporating some randomness in the selection process. It works by selecting n individuals at random, where n is the tournament size. This is then followed by selecting the best individual from the pool/tournament of n individuals with a probability p. This is repeated as required, until the desired number of individuals is accumulated.

6.7 Performance on PROBEN1 Benchmark

In this section, we highlight the results of CoevoNDMs on 22 popular benchmark datasets and compare them to variants of SVM and Neural Network algorithms.

6.7.1 Experimental Setup

Benchmarks for the experiments were obtained from the UCI repository [13]. However, some of the benchmarks - though also found in the UCI repository- were acquired from the

PROBEN1 set of benchmarks [89], which are formatted and split up into a train, validation and test set. The PROBEN1 Benchmarks included: diabetes, card, cancer, and hepatitis (see Table 6.3).

Table 6.3: The list of benchmarks acquired from the UCI machine learning repository [13].

No.	Benchmark	No. of Examples	No. Of Attributes
1	Australian Card	690	7
2	Diabetes	768	8
3	Brest Cancer (Cancer)	699	10
4	Lung Cancer	32	56
5	Echocardiogram	132	12
6	Parkinsons	197	23
7	Lenses	24	4
8	Iris	150	4
9	Sonar	208	60
10	Bankruptcy	250	7
11	Seeds	210	7
12	Abalone	4177	8
13	Monks1	432	7
14	Monks2	432	7
15	Monks3	432	7
16	Vertebral2C	310	6
17	Vertebral3C	310	6
18	Heart	920	35
19	SPECT Heart	267	22
20	Ionoshphere	351	34
21	Hepatitis	155	19
22	Acute Inflammations	120	6

Most of the benchmarks were evaluated using 10 fold cross-validation. However, some of the smaller benchmarks were assessed with smaller folds. In particular, these were: Lenses, and Lung Cancer datasets. It was 3 folds for 10 runs in the case of the Lenses dataset, and 2 folds for 10 runs in the case of Lung Cancer dataset. The error function used for evaluation was the Mean Squared Error (MSE).

Each of the benchmarks was split up into two parts, a train set and a test set, except for the PROBEN1 benchmarks which are already split into three parts. The train set was used for cross-validation while the test set was used for testing. As explained earlier, there was a need for a validation set for selecting the candidate components during the optimization process, where the candidates of each sub-population for the next generation were selected using a tuple of their training and validation error.

Table 6.4: Network and optimization parameters

Parameter	Value
Hidden nodes	2
Sup-population size	15
Max. Iterations	100
Cross-over	Yes
Mutation	Yes
Prob. Cross-over	0.6
Prob. Mutation	0.6
Gauss. Mutation	($\mu = 0.0$, $\sigma = 0.2$)
Topology at init.	Fully-connected Recurrent(Elman)
Tournament Selection (rounds)	3

As such, there had to be an unseen dataset to evaluate the test error. In the case of the benchmarks from the PROBEN1 dataset, the first part of the pre-partitioned dataset was used for cross-validation, while the third part was used for testing.

The network models were allowed to have a maximum of two hidden units in the hidden layer. As expressed earlier, this work is motivated by the neuronal diversity which is associated with efficiency (i.e. relative fewer neurons required) [101, 18, 99, 69, 70] and computational capacity, as such there was a need to replicate the limited number of neurons.

Networks were fully connected at initialization. There were no restrictions on the connectivity between the layers; input-to-hidden, hidden-to-hidden, and hidden-to-output connections were allowed. There was a context layer (see Fig. 6.7) which stored the output values of the hidden layer each time (t_i) to be injected with the activation values of the next pattern for the next pattern at (t_{i+1}). Thus, the networks being evolved are recurrent, which gives them temporal depth, which gives context to the patterns. This opens more possibilities for the computational strategies that can be evolved.

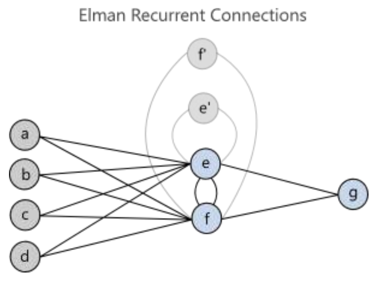


Figure 6.7: Elman context layer consisting of the context units for each hidden unit.

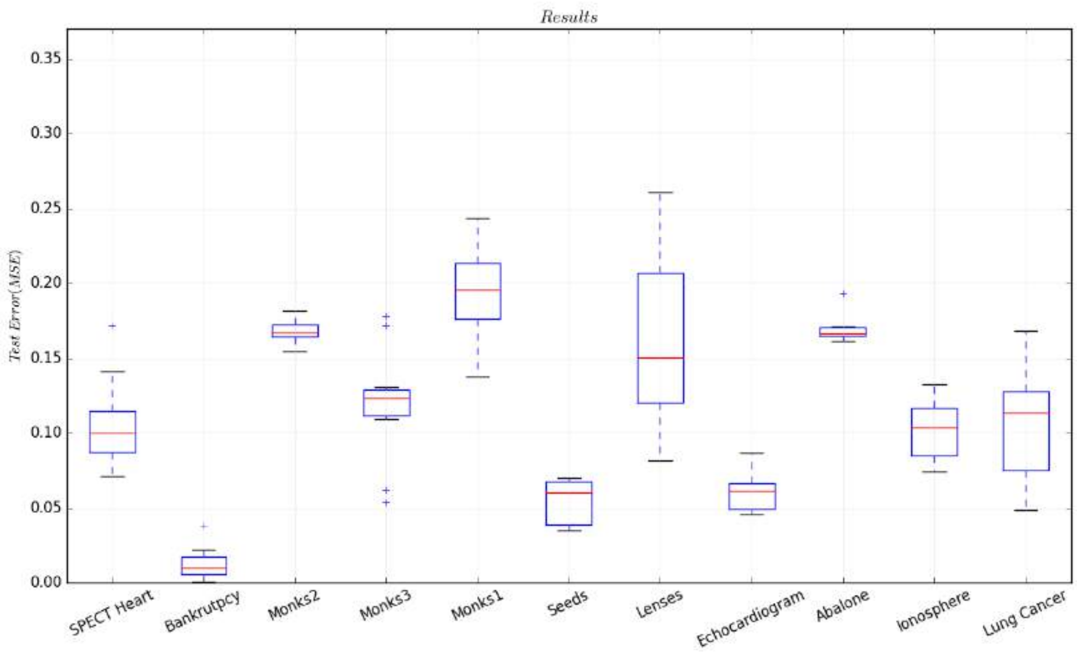


Figure 6.8: Box plot of some of the benchmark results. In general the error seems to suggest good generalization properties. However, the lenses dataset showed high error variance.

6.7.2 Results on PROBEN1

As illustrated in Fig. 6.8 and Fig. 6.9, the results of the CoevoNDM exhibits impressive accuracy for a network of 2 hidden units.

The Lenses dataset had a relatively higher variance. However, it was also interesting to see other benchmarks having relatively low variance and a low median; which is indicative of a low mean test error. In particular, these included the Vertebral Column (3 Class), Iris, Cancer, Diabetes, Bankruptcy, Abalone and Monks2 datasets. Some of the benchmarks had some outliers, particularly for the SPECT Heart, Bankruptcy, Monks3, and Abalone benchmarks.

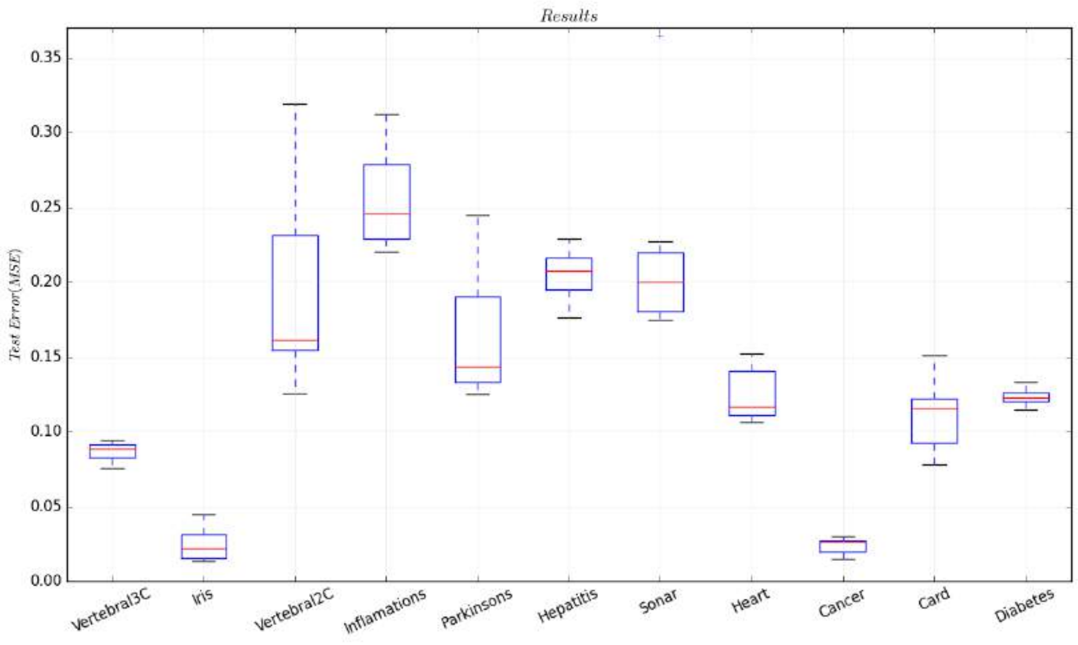


Figure 6.9: Box plot of the rest of the benchmark results. In general the error seems to suggest good generalization properties. Large datasets such as the Diabetes dataset seem to have closely clustered errors, while relatively smaller benchmarks suggest a relatively higher variance.

6.7.3 Comparison with other algorithms

In this section, we compare co-evolutionary neural diversity machines to variants of Support Vector Machines, and Neural Networks. The set of algorithms is made up of support vector machines of various kernels, specifically: radial basis function, polynomial and linear kernels. The comparison also includes neural networks with different architectures, specifically: Multilayer Perceptron (MLP), Radial Basis Function Neural Network (RBF), and a hybrid Neural Network (MLP-RBF) with two layers: the first consisting of radial basis functions, and the second consisting projection basis functions. This is somewhat similar to the architecture proposed by [64].

The MLP and MLP-RBF hybrid neural networks were configured to have two hidden layers consisting of 10 hidden units each. As for the RBF, the Neural Network was configured to have 20 hidden units in a single layer. In the case of the MLP, the projection units used were Hyperbolic tangents, which offer a wider output range as compared to Sigmoid units. The Radial basis function used for the RBF was a Gaussian function. The same choice of projection and radial basis functions were used for the MLP-RBF hybrid. Implementation of the Neural Network was achieved with the help of pyBrain [93] - a python package for neural networks. Back-propagation was used to train the weights of the neural networks for 100 iterations. All the learning algorithms were evaluated using cross-validation on the PROBEN1 benchmark [89], which provides a standardized dataset that makes comparison reproducible and fair. The Support Vector Machine implementation was one offered as a package by mlpy [8] - a machine learning package for the Python programming language.

In addition to these learning algorithms, others from Microsoft's Azure Machine Learning platform were also compared to the results of CoevoNDM. The learning algorithms evaluated on the datasets include Azure's Boosted Decision Tree, and its Neural Network regressor. The Neural Network regressor was allowed to have a maximum of 20 hidden units, and allowed to run for a maximum of 100 iterations. 10 fold cross-validation was used to train the model on the training set. The boosted decision tree on the other hand was allowed to have 20 decision trees in its ensemble - which is within the range of ensemble size in literature with good results [79, 26, 19]. Finally, there is also a comparison with the results of NeuDiME as presented in Chapter 4.

The comparison results for the pyBrain and mlpy learning algorithms are as shown in Fig. 6.10 - 6.29. Table 6.5 shows the performance results of the algorithms on the benchmarks consisting of their mean squared error and standard error. The Appendix has additional data of two-tailed t-test results for all the algorithms performance results on the given benchmarks.

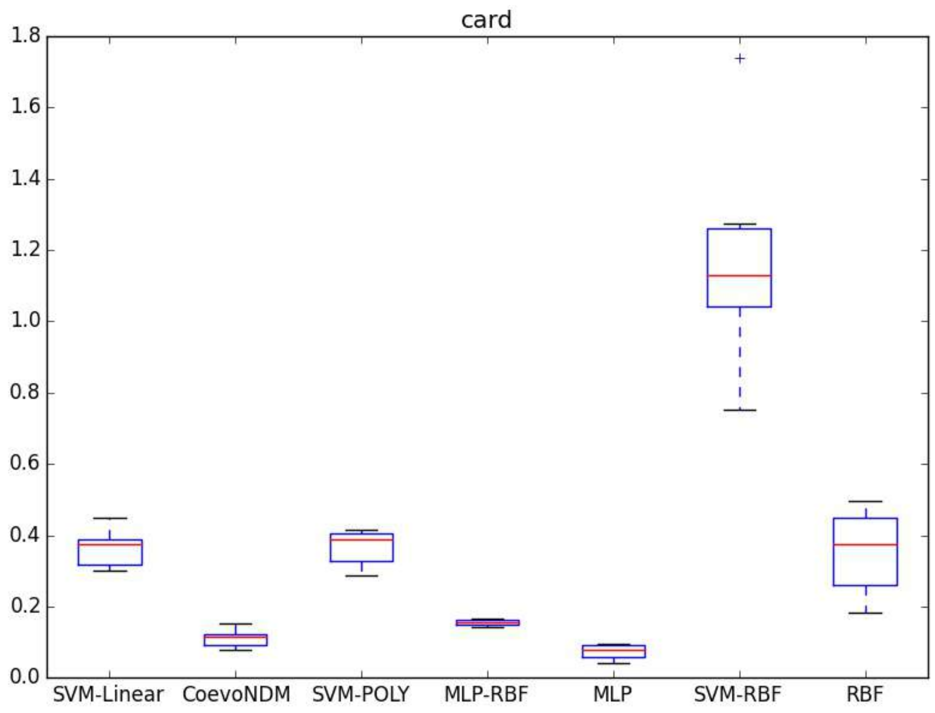


Figure 6.10: Australian credit card benchmark box-plots showing competitive results of CoevoNDM.

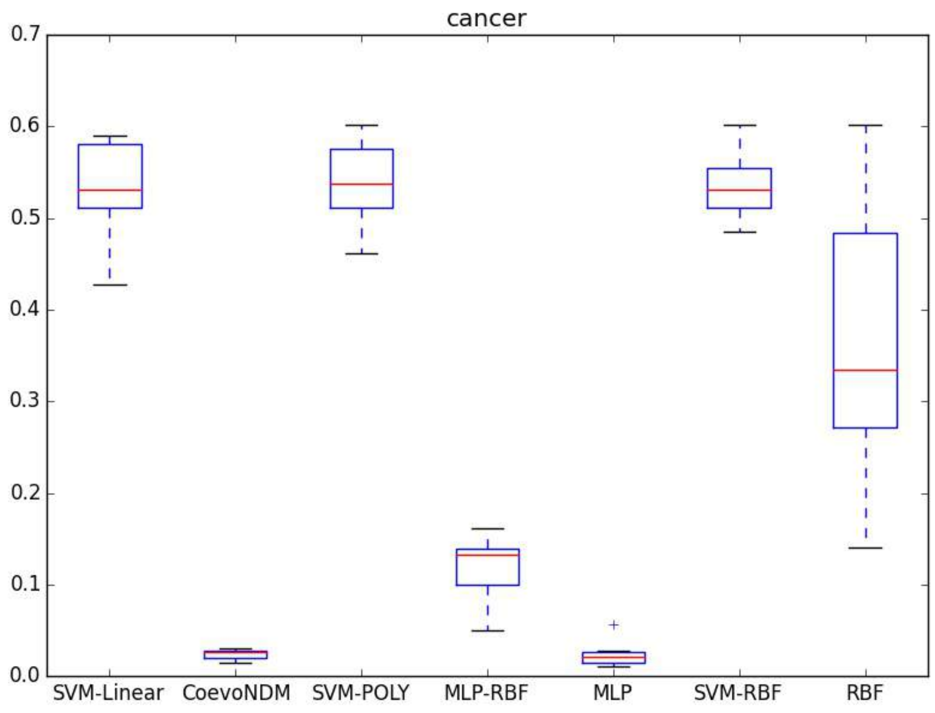


Figure 6.11: Breast Cancer (Prima) benchmark box-plots showing similar results.

Table 6.5: The results showing the mean squared error and standard error of the algorithms on the benchmarks.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
Abalone	0.1819(+/-)0.009	0.3964(+/-)0.022	0.1833(+/-)0.005	**0.1692(+/-)0.003**	0.3926(+/-)0.042	0.3779(+/-)0.038	0.3881(+/-)0.045
Bankruptcy	**0.0125(+/-)0.005**	0.0669(+/-)0.023	0.0625(+/-)0.012	0.0131(+/-)0.003	0.3233(+/-)0.037	0.3167(+/-)0.027	0.3233(+/-)0.033
cancer	0.0231(+/-)0.004	0.3549(+/-)0.052	**0.1205(+/-)0.012**	0.0243(+/-)0.002	0.5341(+/-)0.016	0.5341(+/-)0.011	0.5375(+/-)0.015
card	0.0733(+/-)0.006	0.3603(+/-)0.035	0.1539(+/-)0.003	**0.111(+/-)0.007**	0.3659(+/-)0.016	1.1942(+/-)0.103	0.3659(+/-)0.015
diabetes	**0.1107(+/-)0.005**	0.3267(+/-)0.038	0.1467(+/-)0.002	0.1227(+/-)0.002	0.29(+/-)0.007	0.2879(+/-)0.008	0.2879(+/-)0.013
Echocardiogram	0.1473(+/-)0.034	0.3705(+/-)0.057	0.1973(+/-)0.011	**0.0601(+/-)0.004**	0.297(+/-)0.034	0.3105(+/-)0.057	0.297(+/-)0.056
heart	**0.0931(+/-)0.008**	0.3881(+/-)0.038	0.1402(+/-)0.006	0.1244(+/-)0.006	0.3674(+/-)0.007	0.3674(+/-)0.019	0.3674(+/-)0.014
Hepatitis	0.0883(+/-)0.034	0.1134(+/-)0.065	**0.0755(+/-)0.031**	0.2045(+/-)0.005	0.7452(+/-)0.036	0.7452(+/-)0.026	0.7452(+/-)0.026
Ionosphere	0.1338(+/-)0.014	0.341(+/-)0.047	0.1739(+/-)0.017	**0.1028(+/-)0.006**	0.41(+/-)0.026	0.4147(+/-)0.021	0.41(+/-)0.032
IRIS	0.0366(+/-)0.014	0.3262(+/-)0.029	0.0989(+/-)0.011	**0.025(+/-)0.004**	0.3742(+/-)0.034	0.3742(+/-)0.035	0.3742(+/-)0.019
Lenses	**0.1569(+/-)0.038**	0.2149(+/-)0.046	0.2814(+/-)0.148	0.1741(+/-)0.015	0.3367(+/-)0.098	0.29(+/-)0.02	0.3367(+/-)0.067
Lung_Cancer	**0.0911(+/-)0.019**	0.3904(+/-)0.102	0.1394(+/-)0.033	0.11(+/-)0.008	0.3904(+/-)0.078	0.3904(+/-)0.059	0.3904(+/-)0.033
Monks1	**0.1576(+/-)0.017**	0.4938(+/-)0.057	0.208(+/-)0.006	0.192(+/-)0.01	0.405(+/-)0.02	0.3983(+/-)0.027	0.3983(+/-)0.041
Monks2	0.1837(+/-)0.007	0.3599(+/-)0.048	0.1928(+/-)0.006	**0.1677(+/-)0.002**	0.3088(+/-)0.041	0.3189(+/-)0.027	0.3038(+/-)0.024
Monks3	0.1419(+/-)0.013	0.361(+/-)0.033	0.2105(+/-)0.004	**0.1196(+/-)0.013**	0.3998(+/-)0.048	0.3998(+/-)0.043	0.4248(+/-)0.034
Parkinsons	**0.1434(+/-)0.015**	0.4251(+/-)0.075	0.2343(+/-)0.058	0.1639(+/-)0.014	0.603(+/-)0.023	0.603(+/-)0.033	0.621(+/-)0.034
SONAR	**0.1274(+/-)0.007**	0.3865(+/-)0.066	0.1597(+/-)0.009	0.2144(+/-)0.018	0.4214(+/-)0.03	0.41(+/-)0.057	0.4214(+/-)0.052
Spect_Heart	0.1709(+/-)0.017	0.3662(+/-)0.052	0.1574(+/-)0.013	**0.1073(+/-)0.009**	0.41(+/-)0.042	0.41(+/-)0.058	0.41(+/-)0.049
VertebralCol2C	0.1623(+/-)0.003	0.434(+/-)0.041	**0.1619(+/-)0.003**	0.1942(+/-)0.023	0.418(+/-)0.028	0.41(+/-)0.036	0.402(+/-)0.035
VertebralCol3C	0.1331(+/-)0.012	0.3649(+/-)0.036	0.1248(+/-)0.011	**0.0872(+/-)0.002**	0.4166(+/-)0.031	0.402(+/-)0.041	0.4165(+/-)0.051

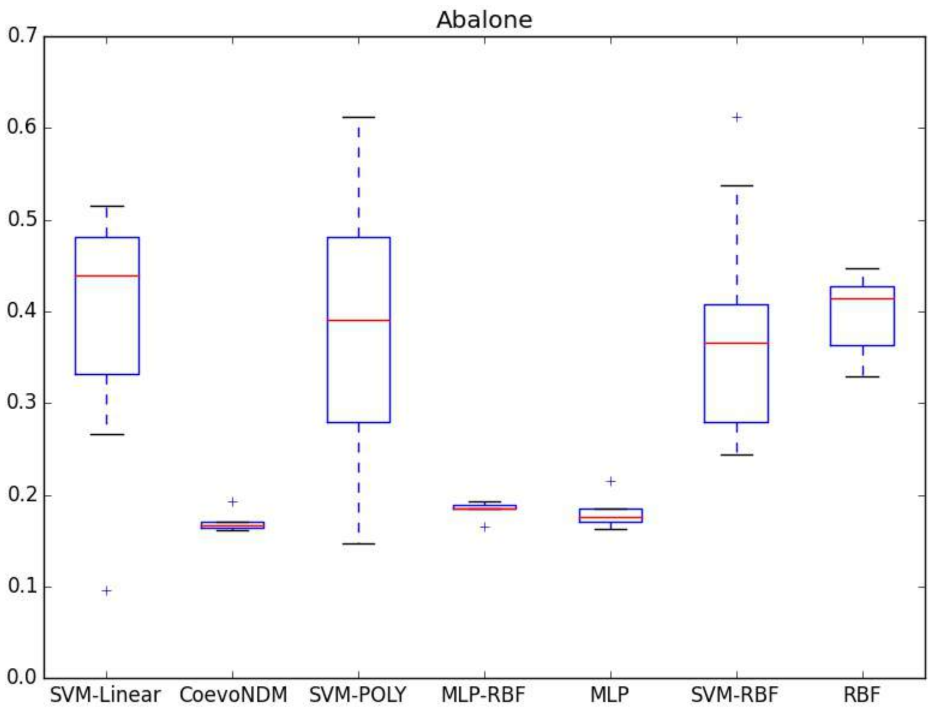


Figure 6.12: Abalone benchmark box-plots for the algorithms.

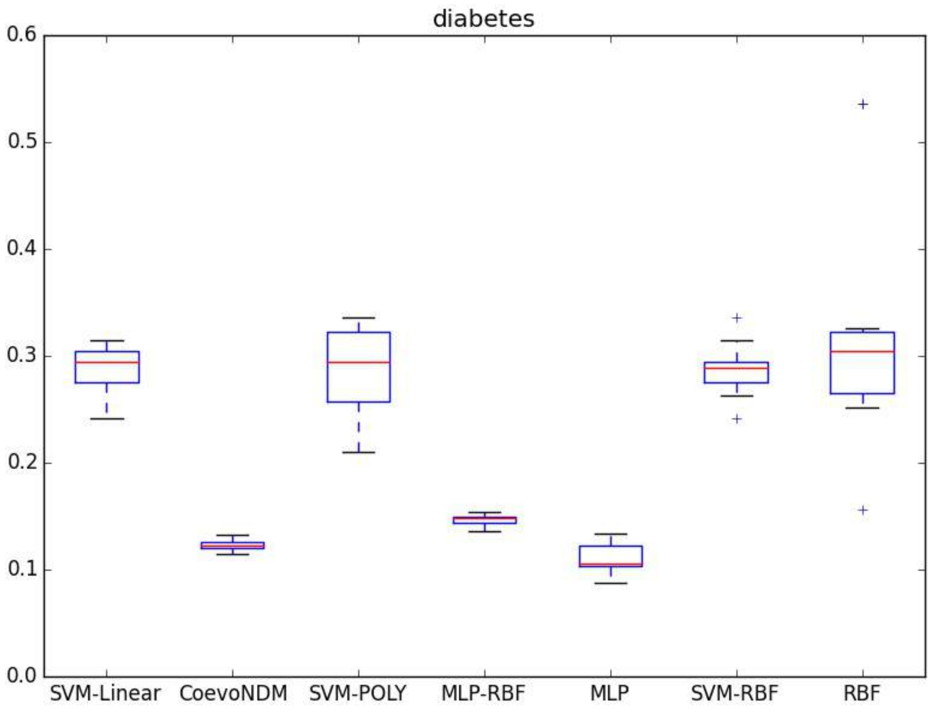


Figure 6.13: Diabetes benchmark box-plots.

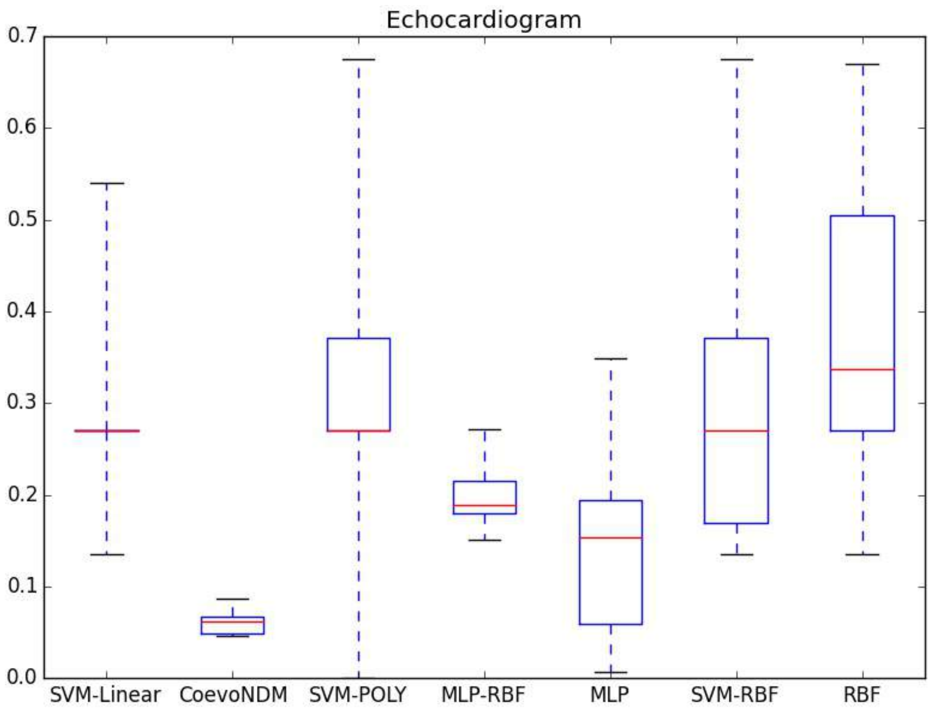


Figure 6.14: Echocardiogram benchmark box-plots.

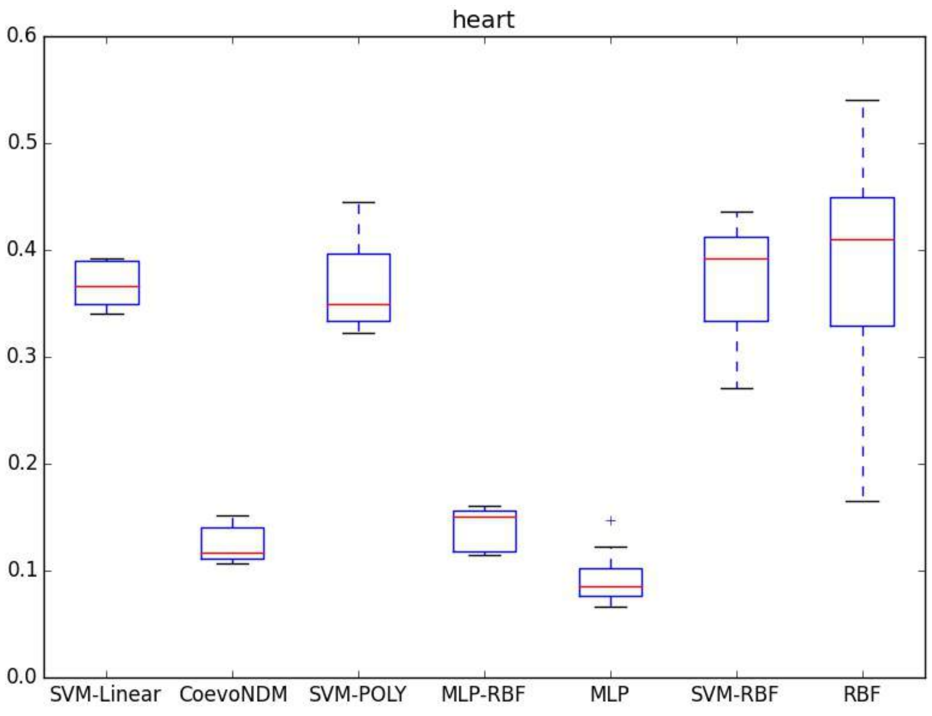


Figure 6.15: Heart benchmark box-plots for the algorithms.

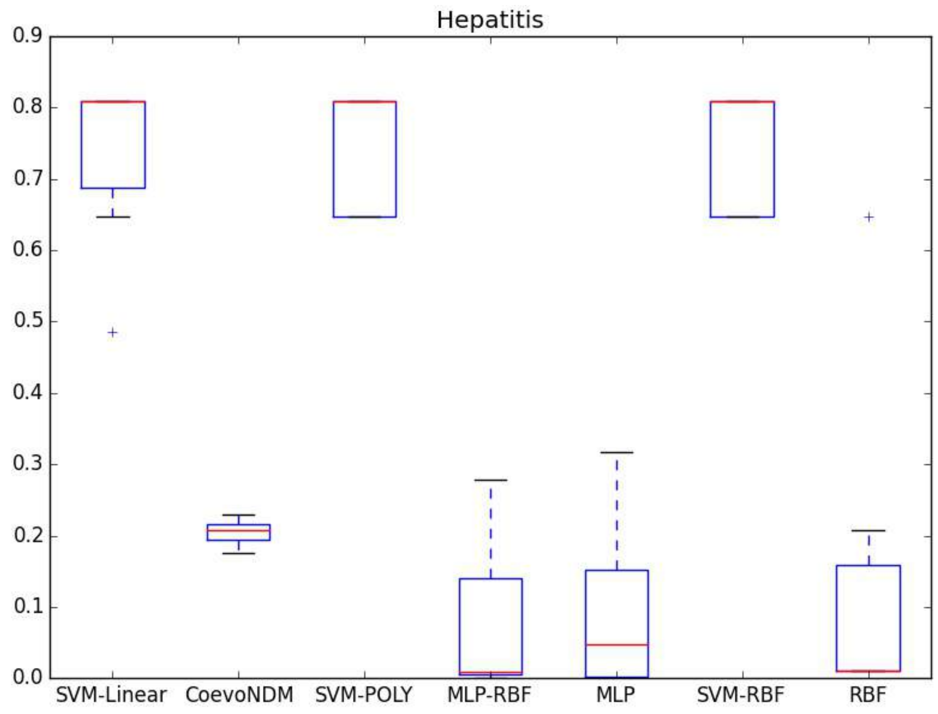


Figure 6.16: Hepatitis benchmark illustrated in box-plot.

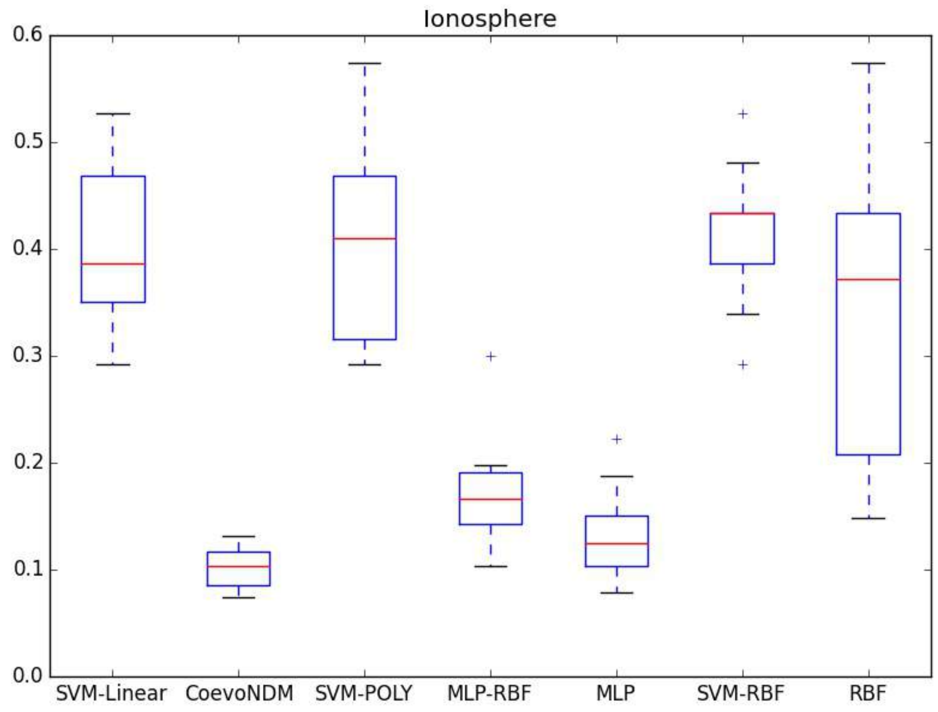


Figure 6.17: Ionosphere benchmark results for the algorithms in box-plots.

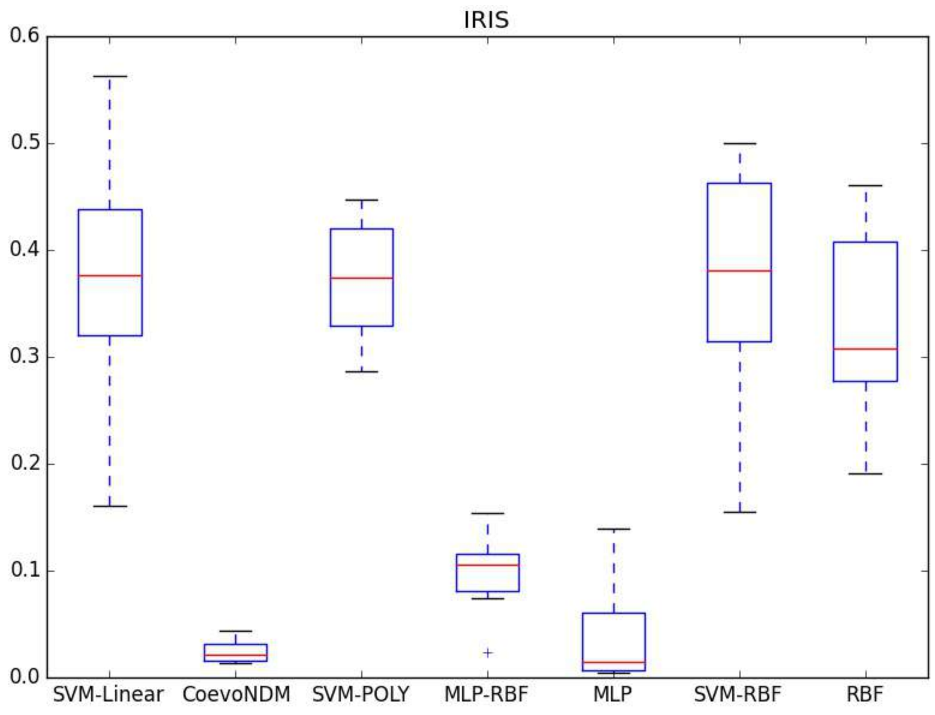


Figure 6.18: Iris benchmark results in box-plots.

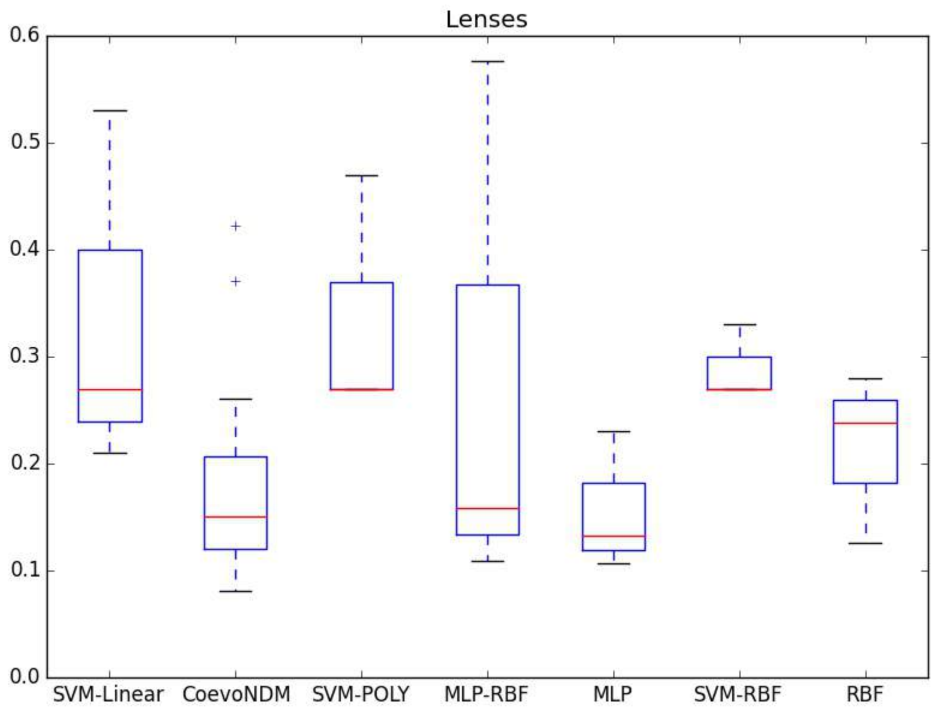


Figure 6.19: Lenses benchmark results in box-plots. The Lenses dataset is relatively much smaller compared to the rest of the benchmarks having only 24 examples.

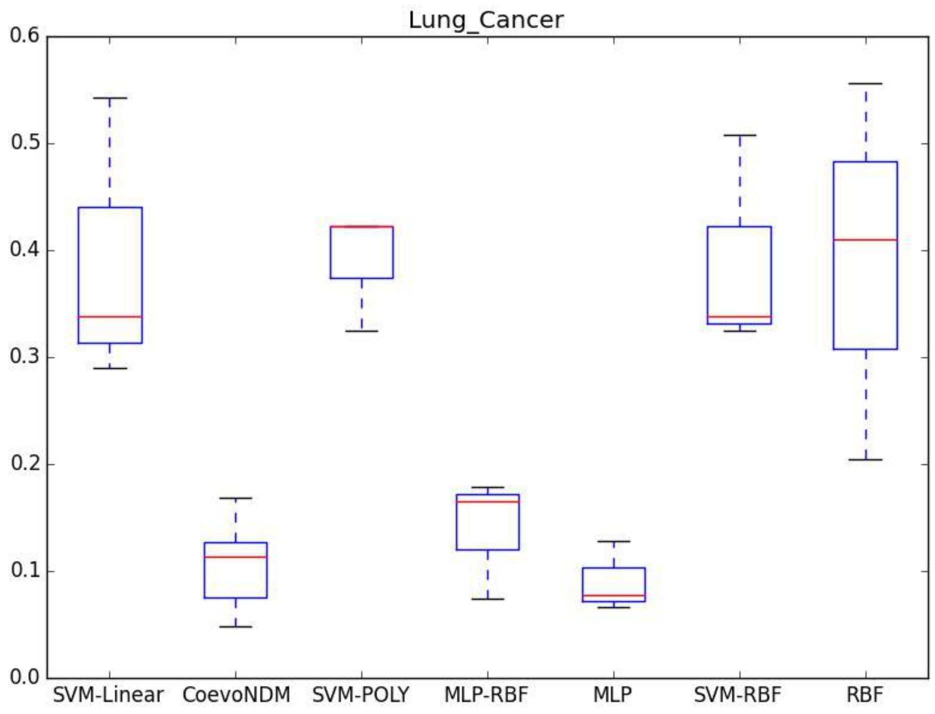


Figure 6.20: Box plot of some of the Lung Cancer benchmark results for the algorithms.

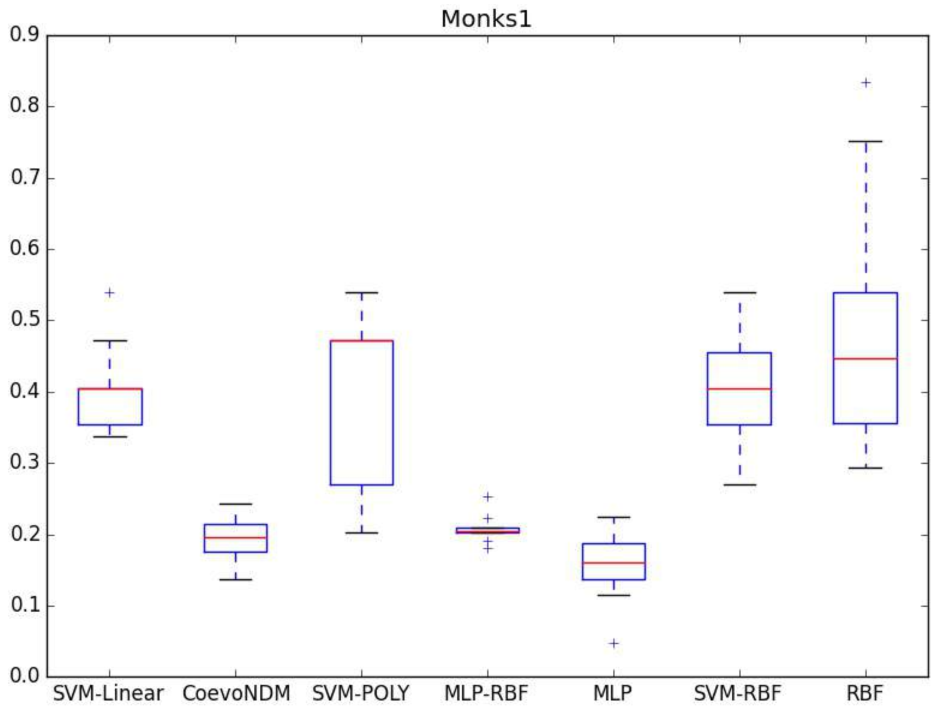


Figure 6.21: Box plot of the Monks1 benchmark results for the algorithms.

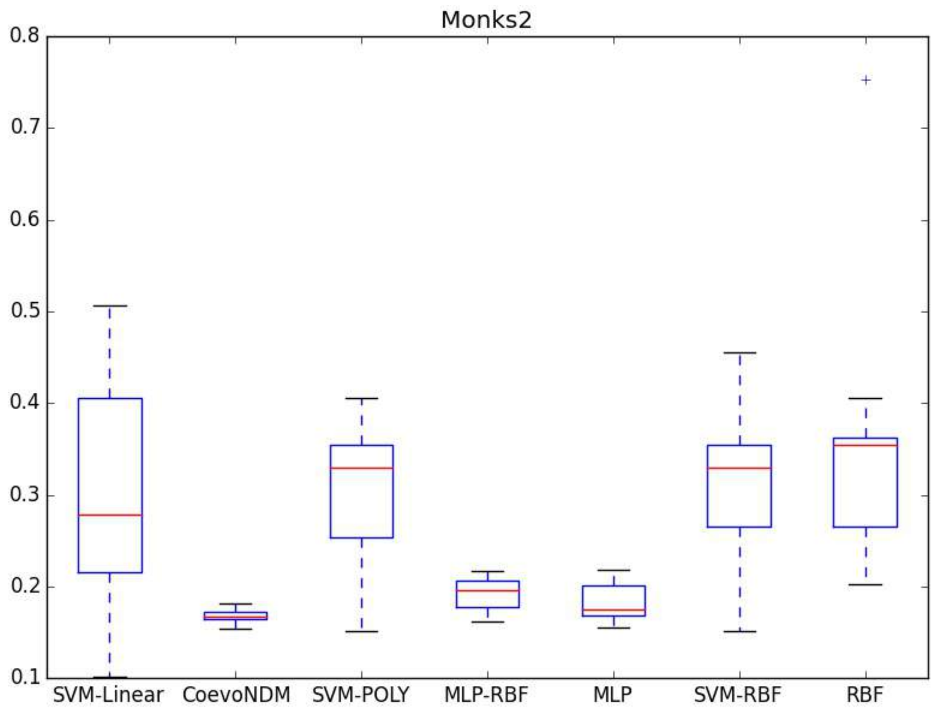


Figure 6.22: Box plot of some of the Monks2 benchmark results for the algorithms.

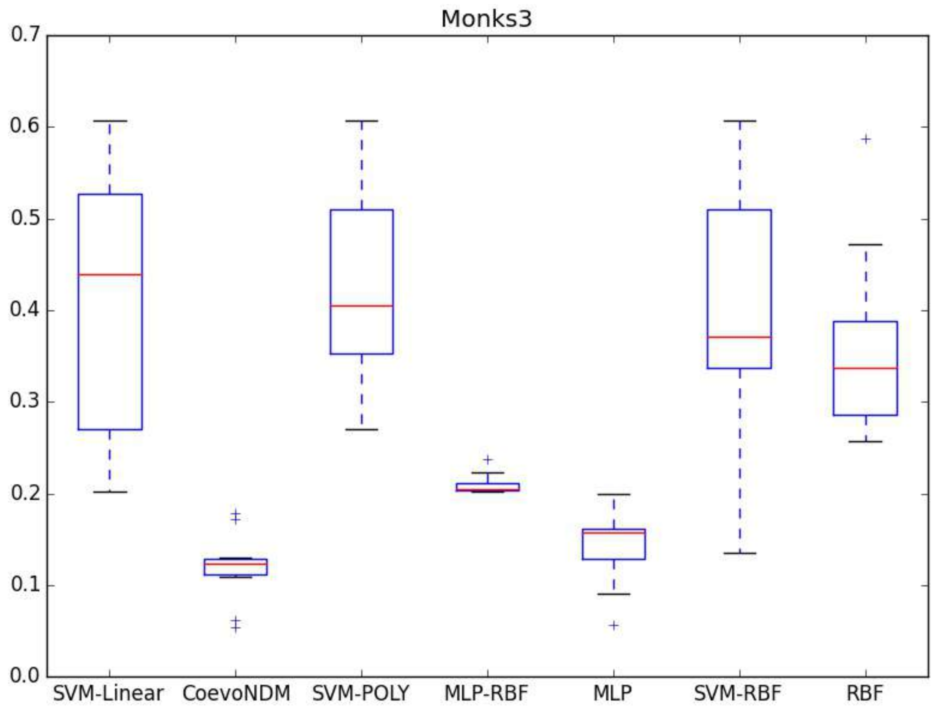


Figure 6.23: Box plot of some of the Monks3 benchmark results for the algorithms.

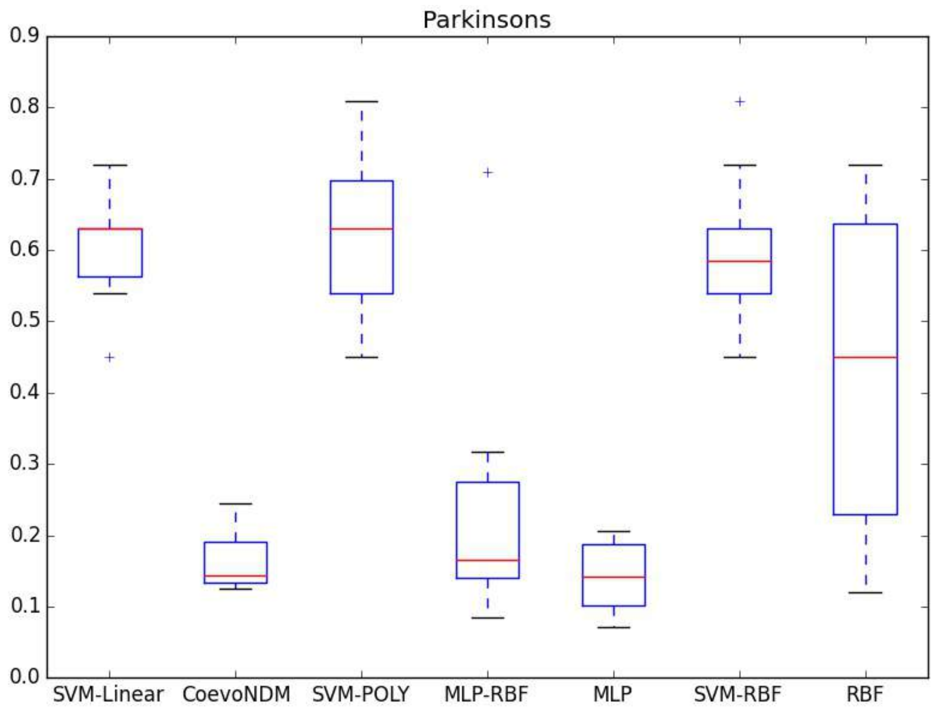


Figure 6.24: Box plot of some of the Parkinson's disease benchmark results for the algorithms.

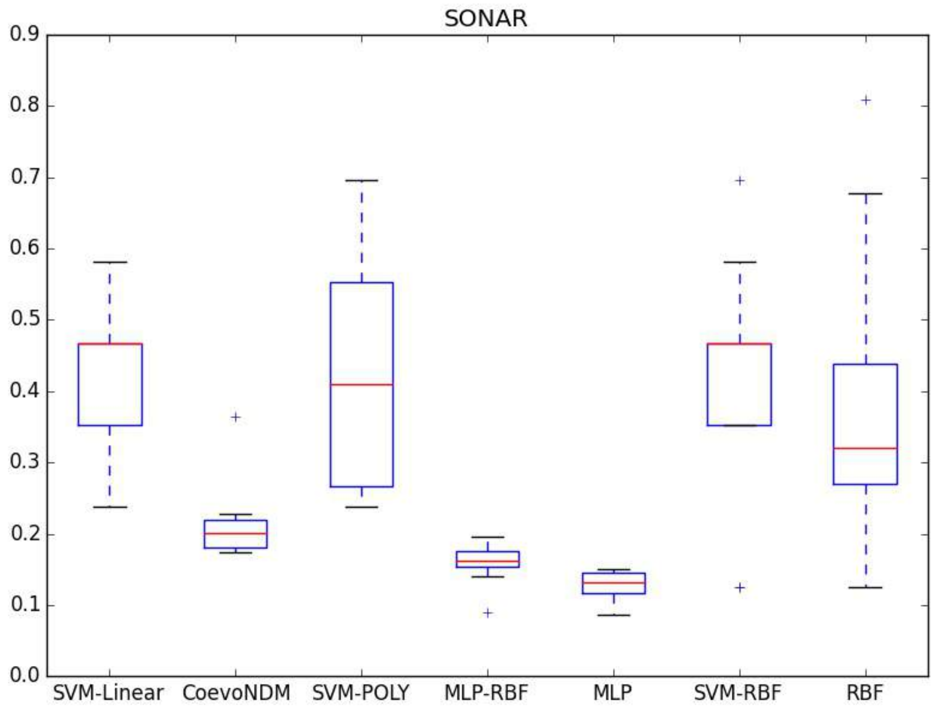


Figure 6.25: Box plot of some of the Sonar benchmark results for the algorithms.

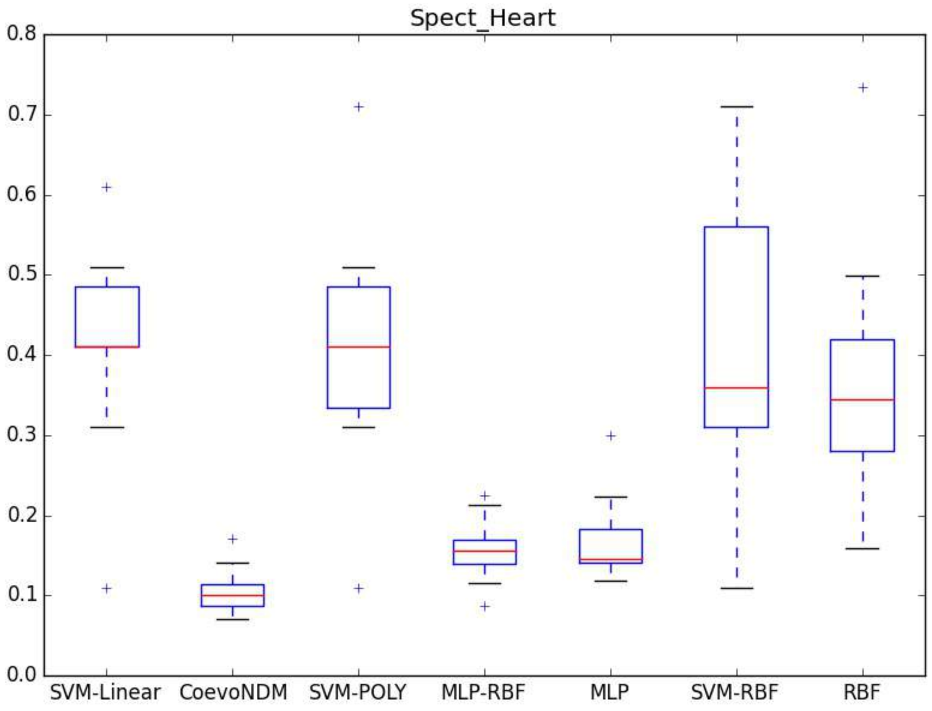


Figure 6.26: Box plot of some of the SPECT Heart benchmark results for the algorithms.

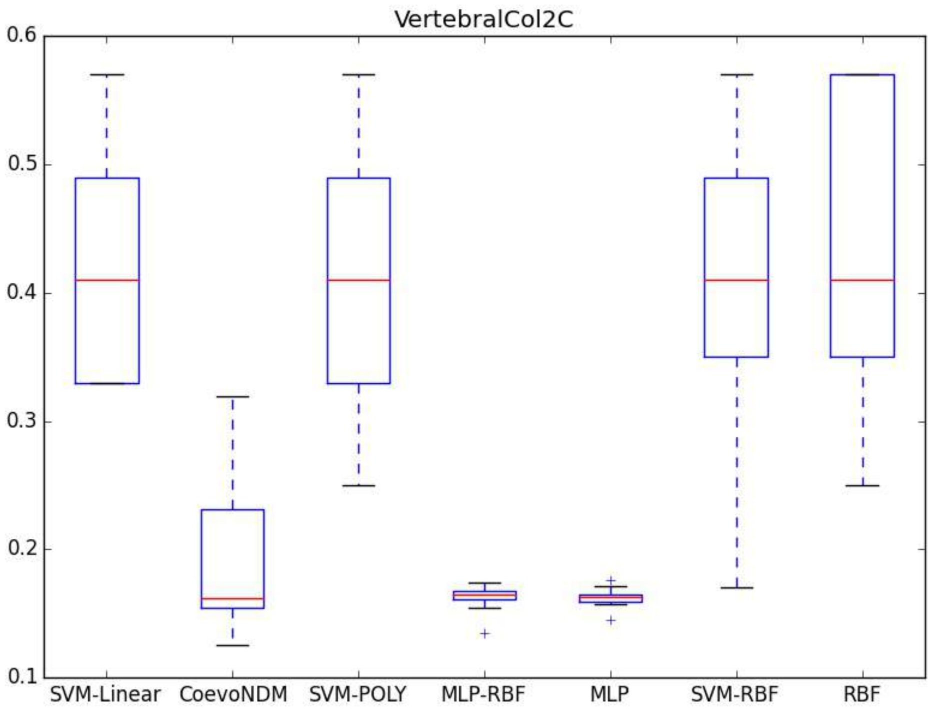


Figure 6.27: Box plot of some of the Vertebral Column (2 class) benchmark results for the algorithms.

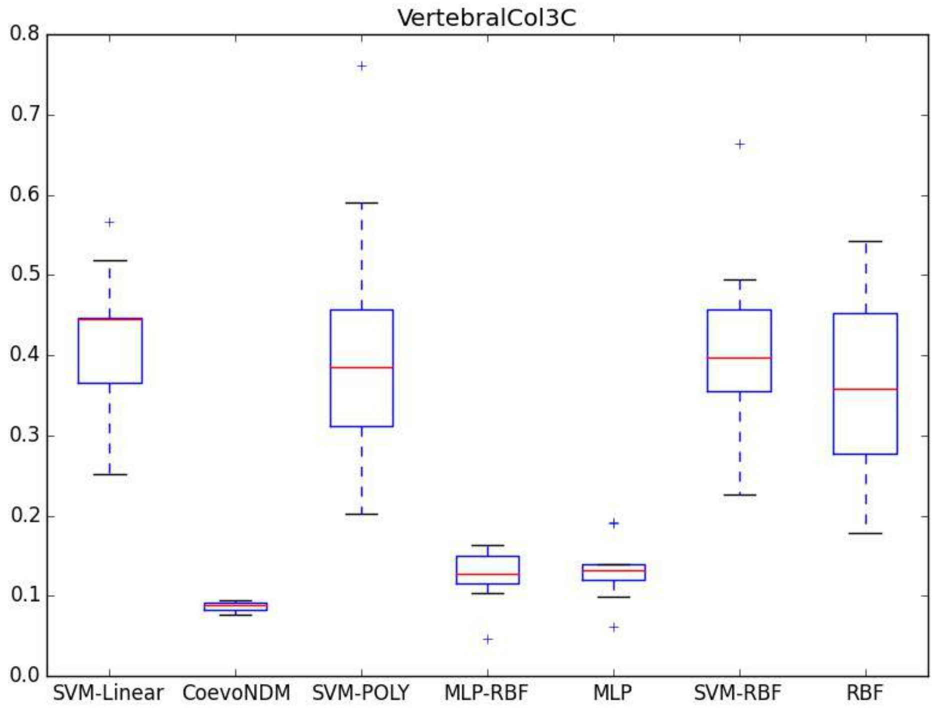


Figure 6.28: Box plot of some of the Vertebral Column (3 class) benchmark results for the algorithms.

The MLP and MLP-RBF Hybrids were significantly better than the SVM variants in terms of their consistency of errors, and showed signs of good generalization ability. This is because there are variations in the test errors, which suggest that different computational strategies are being tried out. In addition, because these are tightly clustered on the relatively lower mean squared errors, it suggests that the collective bias of the neural network is guiding the neural network towards the region of the search space with variations of hypotheses that have promising estimated generalization errors.

Interestingly, both MLP and MLP-RBF produce somewhat similar results in general, even though the RBF produces results that are relatively less competitive. However, in one case, such as for the Bankruptcy dataset (see Fig. 6.29), the neural network variants (i.e. MLP-RBF and RBF) had relatively similar errors.

The algorithms show a lot of variance for the Lenses benchmark, just as co-evolutionary neural diversity machines. This could be because of its relatively small size, which might be making the learning algorithms vary their extrapolation of the problem when different folds of the dataset are being learned. Interestingly, there was a lot more variation in the errors of the vertebral column with two classes dataset (i.e. VerteberalCol2C) compared to that with three classes (i.e. VertebralCol3C) for the SVM variants as well as the RBF-NN.

In general, though, MLP seems to be the best competition to the co-evolutionary neural

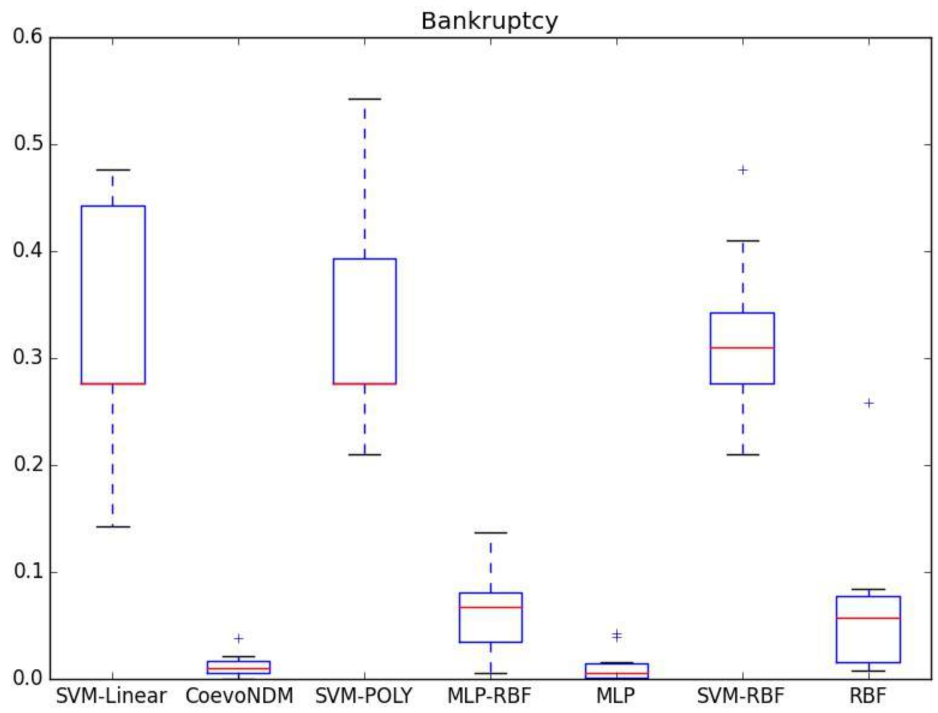


Figure 6.29: Box plot of some of the Bankruptcy benchmark results for the algorithms.

diversity machines even though all the MLP related networks were configured to have the same number of hidden units, and training algorithm (i.e. a gradient descent algorithm, specifically, backpropagation).

In terms of the comparison of the results obtained by these algorithms to the co-evolutionary neural diversity machine (CoevoNDM), it was interesting to observe that CoevoNDM achieved a competitive performance with the other algorithms with much fewer hidden units (i.e. 2) suggesting it is working more efficiently. Thus, suggesting that replicating neural diversity in the form of transfer function diversity produces similarities in the performance properties of biological neural networks.

In terms of the t-test results (see Appendix), CoevoNDM exhibited statistically significant results for some of the datasets, while having non-statistically significant results for the others. Specifically, it had statistically significant differences in results for the card, heart, Monks 2, Sonar, SPECT Heart, and the vertebral column. In other cases, the CoevoNDM had similar results to the MLP as also illustrated by the box-plots. However, in general the CoevoNDM had relatively more consistency in terms of the errors which resulted in less variance of the errors as illustrated by the box-plots.

Relative to the other algorithms, co-evolutionary neural diversity machines seem to have a more tightly clustered estimated generalization error that is very competitive, which suggest signs of generalization ability as well as an ability for more consistent convergence.

In general, the neural network algorithms have a more tightly clustered estimated generalization ability as illustrated by the box-plots in comparison to the support vector machines. This suggests that the support vector machines are likely manifesting a higher degree of variance.

It was interesting to observe that CoevoNDMs seemed to be relatively more competitive as compared to the other hybrid, i.e the MLP-RBF hybrid. This could be for a variety of reasons, one of which could be the architectural configuration. While the CoevoNDM is able to evolve layers of connections in its hidden units, it is also flexible in making the connections as well as choosing the transfer functions of the hidden units in the layers. This enables it to evolve some clever computational strategies for solving problems, hence one of the reasons it was so competitive. The MLP-RBF hybrid on the other hand is only trained using the canonical back-propagation on a predetermined architecture that has a radial basis layer, fully connected to a projection basis layer. The rigidity in the design as well as the gradient descent's proneness to getting stuck at local minima could have a compounding effect which might severely affect its performance potential. In other words, this could build up some restrictive bias for the hybrid.

Interestingly, while flexibility of evolving the hidden layers' connectivity patterns and the transfer functions of hidden units comes with the challenge of a weaker bias, and significantly larger search space, which could both have negative effects on generalization ability, the CoevoNDMs showed no signs of such effects as there was both consistency of convergence and signs of competitive generalization ability. One could speculate that the small size of the CoevoNDMs of just two hidden units has a role to play. A smaller hidden layer size means the CoevoNDMs will help keep the size of the computational strategies search space small, while also driving the optimization algorithm to evolve clever solutions with less computational complexity. In general, less computational complexity has been also associated with good generalization ability [33, 47, 98, 55, 53].

In terms of the comparison to the Microsoft's Azure learning algorithms - specifically, the Neural Network Regressor (Fig. 6.30), and Boosted Decision Trees (Fig. 6.31) - The results revealed that CoevoNDM was competitive in terms of generalization ability, and was also efficient with regards to its relative complexity. This is because the Neural Network Regressor was allowed to have a maximum of 20 hidden units, and was optimized for a maximum of 100 iterations. The boosted decision tree on the other hand was allowed to have 20 decision trees in its ensemble - which is within the range of ensemble size in literature with good results [79, 26].

In terms of its comparison to Boosted decision tree's results (Fig. 6.31), a t-test performed with $\alpha = 0.05$ suggested that CoevoNDM was significantly different from the

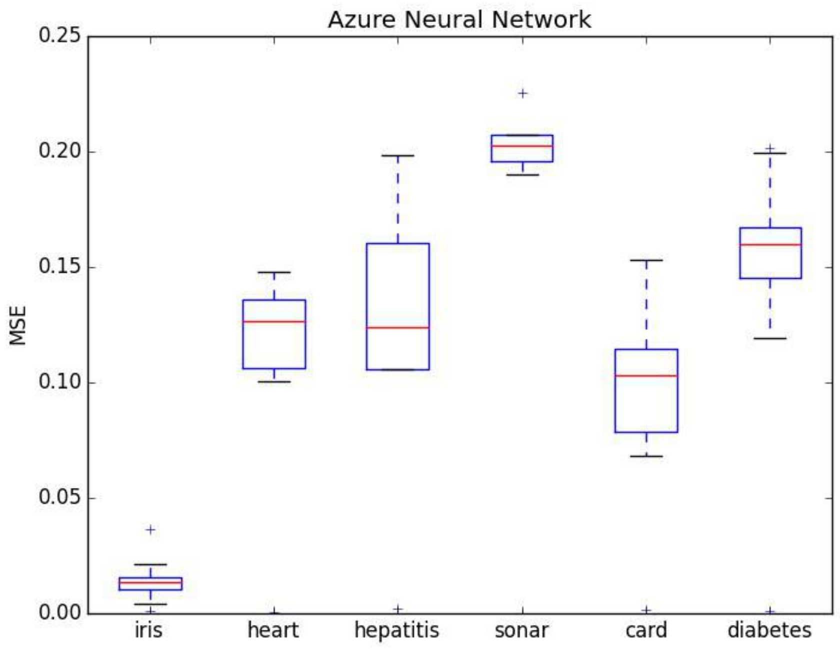


Figure 6.30: Results of the Azure Neural Network Regressor on the datasets.

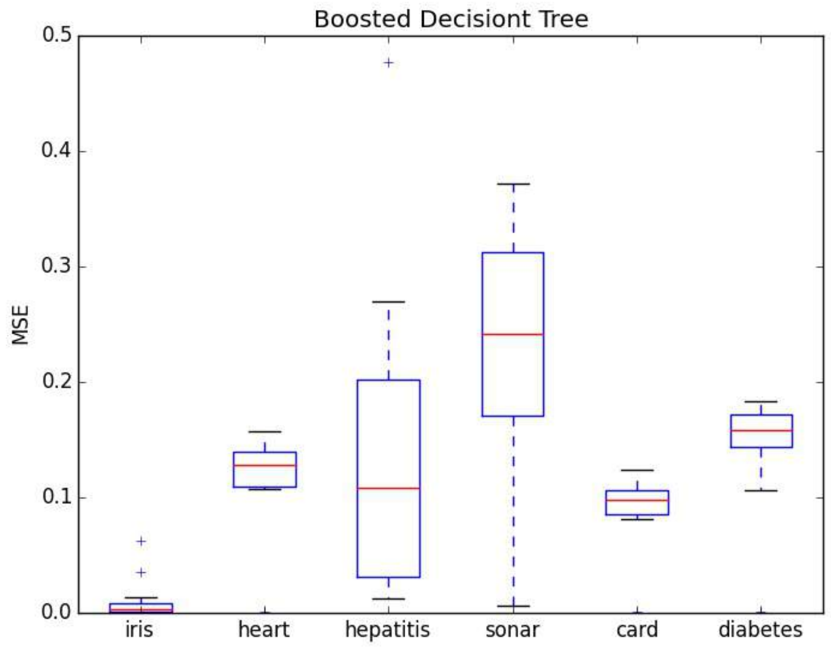


Figure 6.31: Results of the Azure boosted decision tree on the datasets.

boosted decision trees on all the six datasets. Likewise, it was also statistically different from Azure Neural Network regressor. However, the CoevoNDM was in general able to achieve competitive generalization error with significantly less complexity, for example

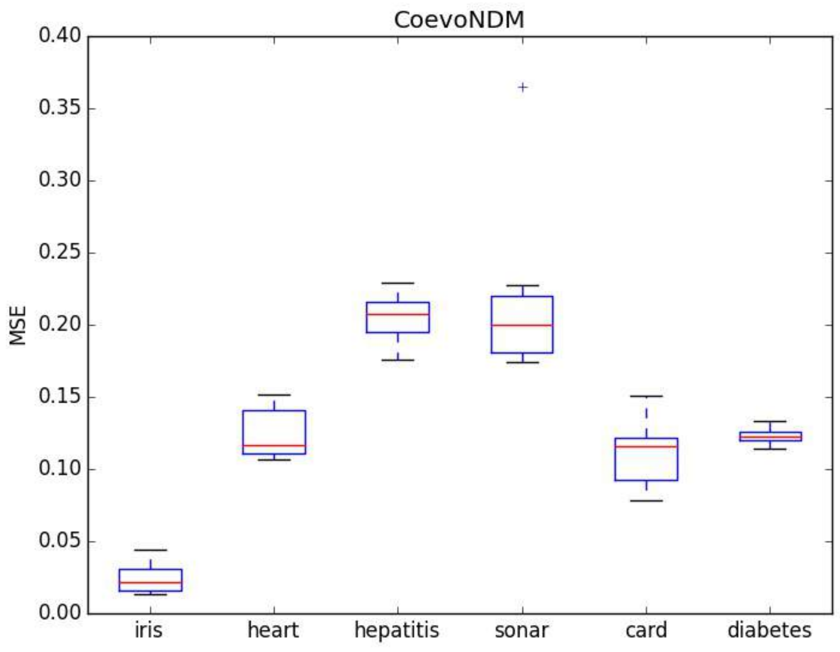


Figure 6.32: Excerpts from the results of the CoevoNDM on the same datasets.

the CoevoNDM had particularly promising results for the Diabetes dataset. Essentially, this further suggests that competitive generalization ability can be achieved with significantly less hidden units using neuronal diversity.

In terms of its comparison to NeuDiME results as presented in Chapter 4, the results show that the CoevoNDM were also really interesting as it's results were better for 3 out of 5 of the benchmarks. One has to keep in mind that the NeuDiME was an ensemble that typically had 10 neural networks, each with the capacity to have 5 hidden units. That puts the total capacity attainable at 50 hidden units. By comparison, the CoevoNDM was just allowed to have two hidden units, since one of the main points of this work was to show that neural diversity can result in efficiency in learning. By our definition, that means having relatively little resources (i.e. hidden units) to produce competitive generalization performance. As such, by that measure the CoevoNDM has succeeded in showing that conclusively both in comparison to state of the art algorithms from Microsoft Azure, and implementation of popular algorithms like the SVM, MLP and RBF in open source machine learning toolkits.

The results specifically show that CoevoNDM had more consistency in terms of convergence as shown in Fig. 6.32, where the grouping of the errors was more tightly clustered and is a sign of good generalization ability. This was more so for the diabetes dataset, where the CoevoNDM even had best results. By comparison, the box plot of NeuDiME seemed to show signs of being stuck at local minima, as indicated by the high variance in errors as shown in Fig. 4.3.

In a nutshell, in the context of the complexity of the networks, and consistency in convergence as shown by the box plots of both the CoevoNDM and NeuDiME; the CoevoNDM is clearly the better algorithm for the given set of problems.

6.7.4 Discussion

In general, the performance of co-evolving diverse neurons, their weights, and topologies shows signs of promising generalization performance. Firstly, the estimated generalization error measured by the mean squared error for all the problems averages at 0.30 for all the 22 benchmarks, which suggest both adaptability and generalization ability. There are a variety of factors that have contributed towards this from the design of the neural networks.

First of all the diverse set of neurons provides access to a variety of adoptable hypotheses, enabling the neural network models to explore and evolve creative solutions to problems. Secondly, the selection approach for choosing the components that cooperate well with others towards good generalization- i.e. selection based on both training and validation error - helps to prevent premature convergence on local minima as well and apply pressure towards converging on components that cooperate better towards good generalization performance. In the next section, we show that the injection of neurons with diverse transfer functions into the neuron subpopulations has a significant positive effect on both generalization and convergence of the neural networks.

In terms of the comparison to the other learning algorithms, the pyBrains's MLP was also very competitive, but so was the CoevoNDM which was in most cases on par and sometimes even better than pyBrain's MLP. And in a few cases, the MLP achieved better results. However, these were statistically not significant as shown by the t-test results (see Appendix). It seemed to be the case that neuronal diversity could be used to scale down the hidden layer size of neural networks without a significant effect on performance. Additionally, in comparison to the industry standard algorithms from the Azure machine learning platform, CoevoNDM was shown to have competitive performance in terms of generalization. It also further shows the efficiency that can be achieved with neural diversity, in addition to the clever solutions that can be evolved. This was the main motivation of the use of neuronal diversity in this research and the results suggest that the biological design pattern is repeating itself in the artificial neural networks domain as well. Neuronal diversity is one of the reasons why biological neural networks are smaller [18, 99, 101], making them more efficient and even having an ability to exhibit more computational strategies.

The results of this experiment show that neuronal diversity results in both competitive performance with far less hidden units, suggesting efficiency; but also supports the argument of CoevoNDM's ability to exhibit creative and clever computational strategies for problem solving.

Porter explained that there is a limitation in the evaluation approach proposed for the cooperative coevolutionary framework for complex interdependent components [87]. Fitness assignment for cooperative co-evolution is particularly not as straight forward as in traditional genetic algorithms. The fitness values of the sub-components are usually not disjoint, on the contrary the fitness of one sub-component warps the fitness surface (or error surface) of the other sub-component, and the assigned fitness depends on the other components. The components of neural networks are interdependent as they affect each others error surfaces, and as such makes them susceptible to the limitations of the evaluation method (i.e. fitness assignment). Therefore, there is still potential for improving co-evolutionary neural diversity machines by exploring other evaluation methods.

6.7.5 Conclusion

In conclusion, it has been shown that co-evolutionary neural diversity machines are efficient and show signs of more representational power as well as an ability to converge on promising solutions with consistency. This is supported by the fact that CoevoNDMs achieved a competitive performance with relatively fewer hidden units without a significant effect on generalization ability. On the contrary, as shown in earlier chapters; neuronal diversity has the ability to exhibit a wide variety of creative and clever computational strategies. This was particularly apparent in the analysis of the neural networks which showed the computational signatures produced by neural networks with neuronal diversity were also diverse due to the differences in their neuronal computational paths. In addition, CoevoNDMs show an ability to converge with relatively more consistency on promising computational strategies, as indicated by the relatively smaller spread of errors in the box plots figures (see Fig. 6.8 and 6.9) for a majority of the problems. Finally, it has also been suggested that a point of improvement for CoevoNDMs could be the exploration of more fitness assignment methods for the components, due to the interdependent nature of neural network components. This is because of the interdependent nature of the components of neural networks which as Porter suggests [87], may have an effect on the performance potential of the optimization algorithm.

6.8 Transfer function Injection

6.8.1 Motivation

Apart from the offspring that result from mutation and cross-over, there was also the injection of randomly generated neurons into each neuron subpopulation after the evolutionary operations. Randomly generated neurons with random transfer functions, and parameters were created and included in the subpopulation prior to the evaluation on the validation set for selection. Although this might be regarded as a standard practice, in this case it differs because of the transfer function diversity factor. Any form of transfer function can be exhibited by pairing any of the activation functions with any output functions. In other words, this actually promotes diversity beyond the generation of random functional parameters alone. It specifically increases behavioral diversity which encourages the search for more computational strategies. It can be regarded as a method of reducing the transfer function bias in the sub-population of neurons by introducing more candidates that might cooperate well with the other sub-components, either by means of having a useful functionality such as feature filtering, or just having a more promising set of functional parameters. In other words, it could be useful in getting out of local minima. In this section, we explore the advantages in performance and convergence of injecting random transfer functions during optimization.

6.8.2 Results

The experimental setup was the same as that used in section 6.7.1 for assessing the performance of the co-evolutionary neural diversity machines (CoevoNDM) on the PROBEN1 benchmarks. The results highlight the estimated generalization performance and the convergence with and without the injection of transfer functions into neuron subpopulations.

It was interesting to see a significant difference in performance between the two conditions, i.e. with random transfer function injection and without it. Specifically, it can observed (in Fig. 6.33) that the testing error with arbitrary transfer functions is significantly better than without it (see Fig. 6.33). This effect was also found in terms of convergence performance (as in Fig. 6.34 - Fig. 6.38) as well as the generalization performance (Fig. 6.33).

One added advantage of random transfer function injection could be its stochastic nature which encourages more exploration, thereby improving convergence. Additional possibilities for exhibiting other computational strategies are introduced when new combinations of transfer functions are injected in the subpopulation of neurons. These combinations might already exist in the pool of transfer functions, but with varying parameters that might be sub-optimal. Thus, it provides variations in the building blocks that could be co-evolved towards finding the most appropriate computational strategy (or hypothesis). This enables the optimization algorithm to escape local minima, by increasing the chances of finding a more optimal computational strategy.

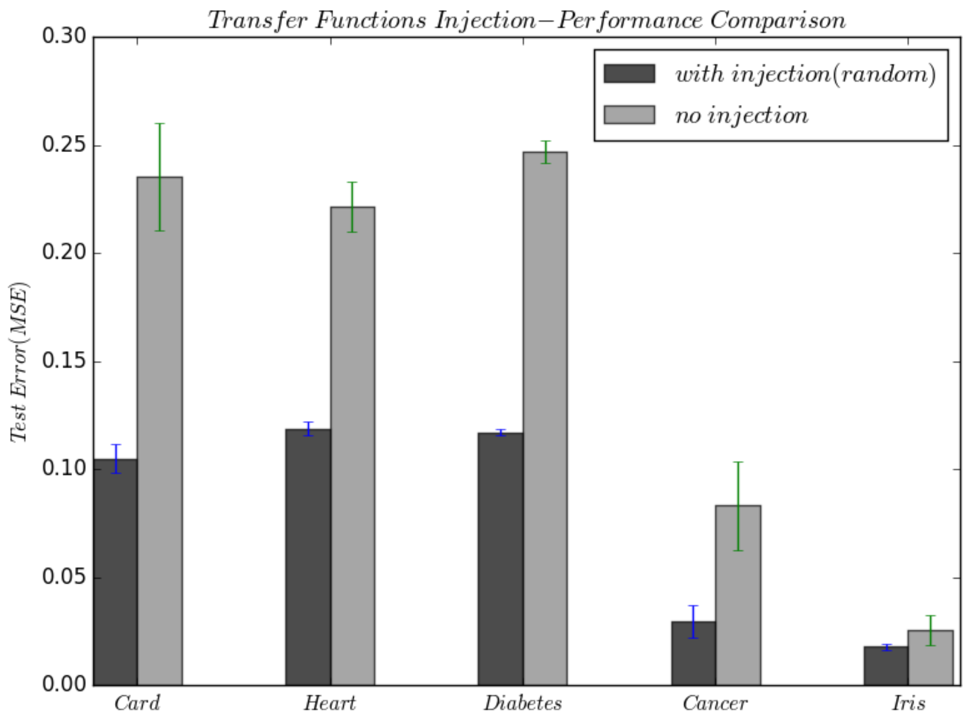


Figure 6.33: Results of the conditions of *with* and *without* transfer function injection on the several benchmarks.

Interestingly, the difference in convergence for the Iris dataset - which is a relatively less complicated problem - was not as significant as seen in the earlier results (see Fig. 6.38).

6.8.3 Discussion

In general, both in term of generalization performance and convergence, the condition with the random injection of transfer functions was shown to be better for the conditions of this experiment and the given datasets, and can be generalized to be an expected trend, especially for more complex problems.

It was apparent that the random injection of transfer functions significantly reduced the transfer function bias, which allowed it to access a wider range of hypotheses by being able to exhibit a wider variety of neural computational strategies. This has the advantage

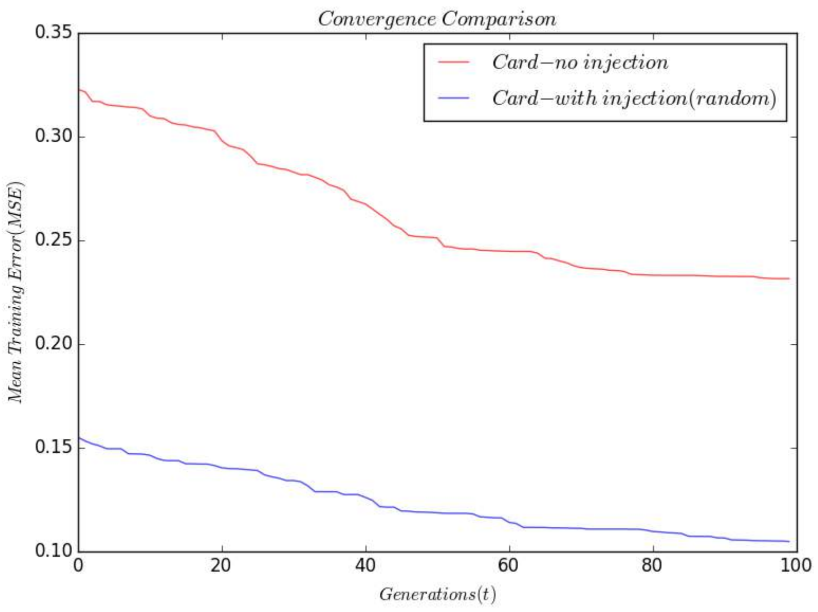


Figure 6.34: In terms of the Card dataset, the results with random injection showed significant improvement than without it; while convergence typically started from a testing error of 0.3 for the condition without injection, the results with random injection of transfer function started from 0.15 on average

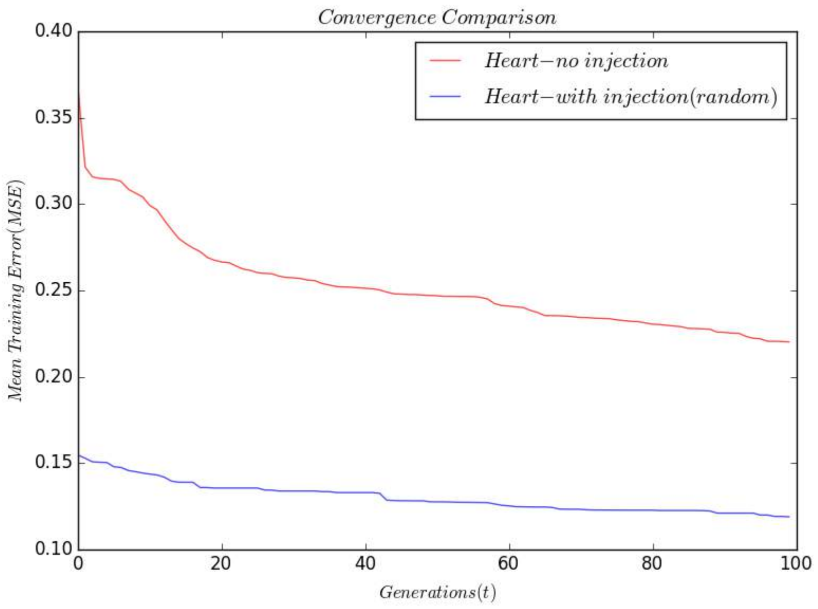


Figure 6.35: The same pattern was also repeated for the Heart dataset which showed significant improvement when random injection was used.

of not only increasing the likelihood of co-evolving a model with a hypothesis that best describes the problem, it also helps in escaping local minima, whereby a more promising fit for the problem is discovered. This is apparent in the convergence difference of the

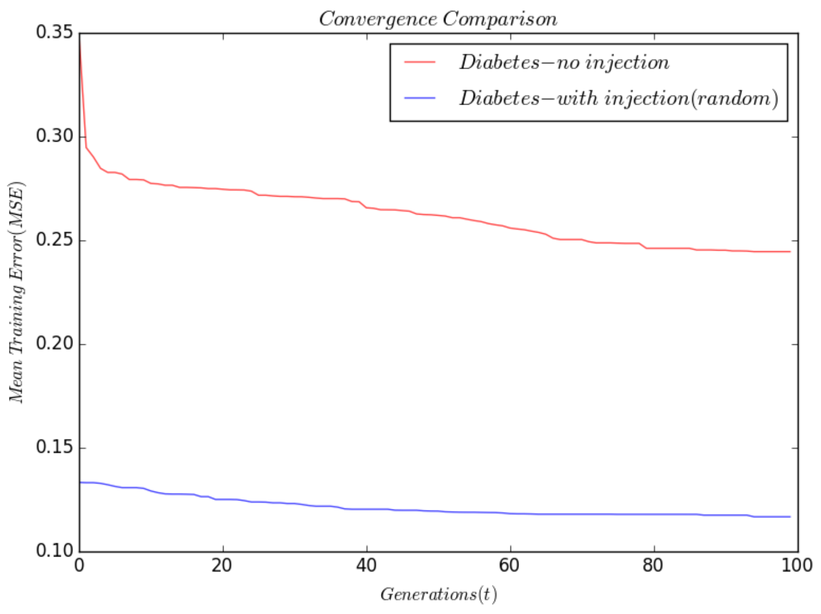


Figure 6.36: Likewise, in the case of the Diabetes dataset the random injection also showed significant improvement in convergence.

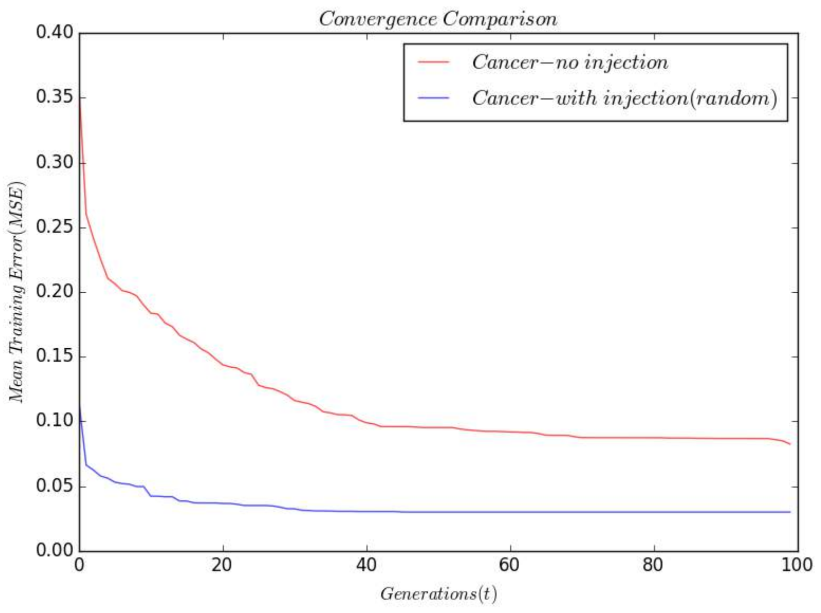


Figure 6.37: The Cancer dataset convergence graph also shows a significant improvement when random injection of transfer function is used.

conditions characterized by presence or absence of random transfer function injection (see Fig. 6.34 - 6.38). Specifically, it can be seen that the rate of convergence in the early stages of optimization were more promising than without it.

It was expected that introducing more variations of neurally diverse transfer functions to each sub-population of neurons might have a slowing effect on convergence. However,

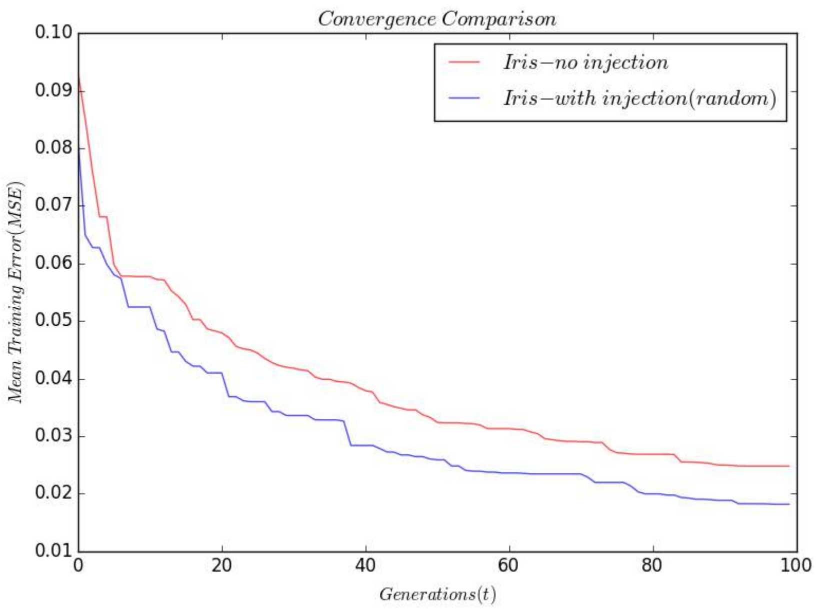


Figure 6.38: Iris showed similar results of the convergence for the condition with random injection.

contrary to that, the injection of random variations of transfer functions accelerated the convergence of CoevoNDM. In addition to convergence, the estimated generalization ability of CoevoNDM with transfer function injection was significantly better than without it. There is the possibility that CoevoNDM might converge on a similar or even better hypothesis, however the uncertainty associated with the convergence makes it somewhat impractical. Specifically, it can be observed that for relatively more complex problems, the difference in convergence between the two conditions were significant as compared (see Fig. 6.37, 6.34, 6.36, and 6.35). This pattern is likely to repeat itself in other real world problems that are much more complex than the current datasets.

However, fast convergence does not necessarily improve generalization, slowing a fast convergence can be useful as it has been shown to improve the generalization performance of co-evolutionary neural networks - as in SANE [78]. This is because a convergence that is too fast, might lead to premature convergence on a local minimum. In the case of CoevoNDM, the faster convergence is kept under control by the fitness assignment. Assigning two fitness values representing how much the candidate component contributes towards better training and validation error is meant to balance the objectives of training error and the estimated generalization performance represented by the validation error. This is an important contributor towards helping prevent a premature convergence. It is likely that the results would have been otherwise without the validation error.

6.8.4 Conclusion

In conclusion, it can be said that inducing diversity of transfer functions by randomly injecting them into the sub-population of neurons being co-evolved has a significant positive effect on both generalization ability and convergence given that the fitness assignment also accounts for the estimated generalization performance. This is because of the decreased bias on the transfer functions, which enables the co-evolutionary algorithm to explore a lot more neural computational strategies, consequently allowing it to explore more hypotheses that might have better potential for describing the underlying nature of the problem. This also improves the chances of finding a more optimal model, thus escaping local minima - as also shown by the improvement in convergence speed. It was also explained how fitness assignment was essential in making sure a premature convergence on a sub-optimal hypothesis was avoided as much as possible. In our case, the fitness values of each candidate component consisted of fitness values representing both training and estimated generalization performance. The multi-objective optimization of both fitness values helps the selection mechanism apply pressure on finding solutions that contribute towards learning the training data, and generalizing to other portions of the problems hyperspace (represented by the validation set).

Chapter 7

Conclusion

This thesis presented contributions across various aspects of efficient and robust learning using the property of neural diversity found in biological neural networks. The major contributions include: (i) establishing the relationship of neural diversity to generalization ability in the context of the bias-variance decomposition and meta-learning; (ii) neural network meta-features (describe as problem signatures) shown to be able to characterize problems, as well as be a valuable tool in analyzing underlying computational strategies evolved by neural networks with diverse transfer functions; (iii) finally, a framework that involves co-evolving neural diversity with their topology and weights to self-adapt the bias of the neural network, and increase ability to perform more efficient information transfer between components using evolutionary operators, such as crossover.

7.1 Established the notion of neural diversity for efficient learning from different perspectives

The thesis reviewed the various related works done in transfer function optimization, classified them according to their motivation, and established a notion of how neural diversity could lead to better generalization in the context of the bias-variance decomposition. There were three major motivations behind works found in the literature [61, 41, 47, 3, 31, 30, 55, 29]: (i) Donoho's theory of Duality of functions; (ii) meta-learning; and (iii) transfer function flexibility. Though these works showed promising results, there are relatively few works on transfer function in general. Also, most of these works are not as extensive relative to research on other components of artificial neural networks. Furthermore, as per the writing of this thesis, there is yet to be a work that specifically and explicitly relates transfer function optimization to generalization, supplemented with significant empirical results. This thesis contributes to the literature by presenting a biologically inspired facet of transfer function optimization, which has been found to have promising results in preliminary experiments [71]. It also explained specifically how the bias of neural network components affect generalization ability in the context of the bias-variance decomposition and meta-learning. It then outlined how neural diversity, represented as transfer function diversity in this study, can enable the neural network to adopt more forms of bias which result in being able to access and search more regions of the hypothesis space. In other words, neural diversity increases the computational capacity of artificial neural networks. Consequently, this increases the probability of finding the most appropriate hypothesis that describes the problem's underlying function.

Experimental results were used to show that transfer function diversity can exhibit diverse computational strategies and as such can be used as a diversity maintenance scheme for neural network ensembles without the need of other explicit diversity maintenance measures that tend to be computationally expensive. This was shown by revealing two different strategies for the diabetes problem, where clever functions such as feature filters were evolved for pre-processing. The evolved filter function was used to extract essential features such as age and skinfold thickness- that the strategies found to be correlated to the risk of diabetes. Interestingly, these features are among the factors that increase the risk of diabetes as recognized by the American Diabetes Association (ADA) [10]. It was also shown that this approach can evolve relatively small ensembles of compact networks that have a competitive performance, as in the case of the diabetes problem which had a small number of ensemble members of no more than 20 networks, each with relatively fewer hidden units of no more than five hidden units. Those found in the literature tend to have 20 members with ten hidden units [19, 79]. The results suggested that neural diversity not only shows signs of ability to significantly produce diverse computational strategies with diverse biases; it also demonstrated that it can evolve creative solutions to complex solutions. However, its limitations included the increased local minima as a result of its greater access to the global search space of computational strategies. Also, relatively slower convergence was also a concern. However, this was addressed in a later chapter.

These findings paved the way for the next contribution which exploited the performance differences of transfer functions to characterize problems, among other tasks.

7.2 Meta-features for problem characterization and analysis of neural network models

The second significant contribution was showing the feasibility of computational signatures being used as meta-features by demonstrating they had both the consistency and the discriminatory properties proposed. Firstly, the chapter explains the idea of computational signatures, which could be regarded as meta-features. Meta-features in the literature have typically been considered to be either a set of features resulting from analyzing a dataset [103, 81], or the performance information of various learning algorithms on a dataset which also provides some degree of insight into the nature of the problem [84, 61, 103, 3]. Though performance information of various learning algorithm could arguably provide some level of information regarding the properties of problems, it was the opinion of the author that more reliable features can be unveiled by studying the models that learn the problem. Related works include the use of features such as the depth of decision trees after training on a dataset as a meta-feature [81]. Unfortunately, just like the study of transfer functions: there seems to have been very little work in this direction, specifically for artificial neural networks. There has been work by Ajith [3] and Kordik [61] on meta-learning neural networks which learn to adopt transfer functions from a pool of the traditional transfer functions consisting of projection and radial basis units.

This work contributes to the literature, by specifically developing various meta-features based on neural network models described as problem signatures. It was shown that these signatures could be used to improve convergence by initializing the transfer functions of artificial neural networks. It also demonstrated that problem signatures exhibit the defined properties that are desirable for meta-features, i.e. consistency in being discriminative between problems that are different, as well a significant degree of invariance for a problem regardless of tweaks to various parameters.

Additionally, it was also shown that higher-order signatures can be analyzed further by using them to generate a digraph that represents the building blocks of the elite models that had promising fitness values for the problem, which in essence inherently represents some information about the nature of the problem. These graphs are then used to verify the feasibility of the neural network computational signatures as meta-features. Furthermore, our graph-based approach which uses the meta-features to form an expectation of the computational strategy found to be most appropriate for the problem also presents a method of not only understanding the neural network models, but also the nature of the problem itself. The results of these have been used to show interesting computational strategies, including one that shows how the neural network creatively used min and max activation functions to filter the most important features of the diabetes problem, namely age, and skin fold thickness. Interestingly, this was found to be established as some of the factors that increase the risk of diabetes by the American Diabetes Association (ADA) [10].

Finally, the thesis then utilizes neuroscience-inspired analysis techniques for the study of artificial neural networks. This was mainly motivated by the tendency of neural networks to be used as black box learning algorithms. There is yet to be a significant work in the direction of techniques for analyzing the hypothesis learned by artificial neural network models. However, Neuroscience has a variety of established methods used for studying biological networks.

7.3 Co-evolution of neural diversity and their topologies for efficient learning

The third contribution of this thesis was the presentation of a holistic approach to optimizing an artificial neural network that involves the co-evolution of neural diversity with their connection weights and topology. In traditional representations of artificial neural networks meant to be optimized using evolutionary algorithms, a single genetic string is usually used. Though there have been very promising results from this approach [112, 12, 34, 97], it has also presented problems such as the cross-over operations being particularly detrimental to performance in some cases. This thesis contributes by particularly co-evolving a representation of the model space with regards to its principal components; i.e. weight, topology and transfer functions, to achieve better information transfer during evolutionary operations. The transfer function diversity, flexibility to evolve any form of topology, and improvement in information transfer are all meant to reduce the bias of the neural network thereby increasing chances of better generalization ability. Though there have been related works that have co-evolved neural networks using various representations, evaluations and optimization techniques such as SANE [78], COVNET[44], CoSyNe [46] and EPNET [111]; this work differs as it co-evolves the three major components of the neural network represented as three interdependent sub-spaces. In particular, the property of neural diversity makes this approach significantly different from the other approaches.

The results showed that co-evolving the components of neural networks with neural diversity can enable it to achieve significant generalization ability with comparatively less complex models consisting of two hidden units [6]. In particular, the results tested on 22 popular benchmarks from the machine learning repository (UCI) [13] and PROBEN1 [89] benchmark show an average mean squared error of 0.30, which is significant for a neural network of that complexity. This suggested signs of more representational power as well as an ability to converge on promising solutions with consistency. This was because the CoevoNDM achieved a competitive performance, particularly of the neural network breed with relatively fewer hidden units without a significant effect on generalization ability. On the contrary, neuronal diversity can exhibit a wide variety of creative and clever computational strategies. This has been particularly evident in the analysis of the neural networks which showed the computational signatures produced by neural networks with neuronal diversity were also diverse due to the differences in their neuronal computational paths. Furthermore, CoevoNDMs show an ability to converge with relatively more consistency on promising computational strategies for a majority of the problems and was on average found to be better than NeuDiME.

It was also shown that injecting more neural diversity during the co-evolutionary process also significantly increases both the generalization ability of the neural network and its convergence rate. This was because of the decreased bias on the transfer functions, which enables the co-evolutionary algorithm to explore a lot more neural computational strategies, consequently allowing it to explore more hypotheses that might have better potential for describing the underlying nature of the problem. This also improved the chances of finding a more optimal model, thus escaping local minima - as also shown by the improvement in convergence speed. It was also explained how fitness assignment was essential in making sure a premature convergence on a sub-optimal hypothesis was avoided as much as possible. In our case, the fitness values of each candidate component consisted of fitness values representing both training and estimated generalization performance. The multi-objective optimization of both fitness values helps the selection mechanism apply pressure on finding solutions that contribute towards learning the training data, and generalizing to other portions of the problems hyperspace (represented by the validation set).

Bibliography

[1] H. Abbass. A memetic pareto evolutionary approach to artificial neural networks. *AI 2001: Advances in Artificial Intelligence*, 2001.

[2] H. H. Abbass. Pareto neuro-evolution: Constructing ensemble of neural networks using multi-objective optimization. In *The 2003 Congress on Evolutionary Computation 2003 CEC 03 (2003)*, volume 3, pages 2074–2080, Cancun, 2003. IEEE.

[3] A. Abraham. Meta learning evolutionary artificial neural networks. *Neurocomputing*, 56(1- 4):1–38, Jan. 2004.

[4] A. Adamu, T. Maul, A. Bargiela, and C. Roadknight. Preliminary experiments with ensembles of neurally diverse artificial neural networks for pattern recognition. In *Recent Advances in Information and Communication Technology 2015*, pages 85–96. Springer, 2015.

[5] A. S. Adamu, T. H. Maul, and A. Bargiela. On Training Neural Networks with Transfer function Diversity. In *International Conference on Computational Intelligence and Information Technology, CIIT '13*, pages 295–304, Mumbai, 2013. Elsevier.

[6] A. S. Adamu, T. H. Maul, and A. Bargiela. Efficient Learning by Coevolution of Neurally Diverse Artificial Neural Networks. In *2014 IEEE Symposium on Computers & Informatics (ISCI 2014)*, Kota Kinabalu, Malaysia, Sept. 2014.

[7] A. S. Adamu, M. Tomas, and A. Bargiela. Assessing the feasibility of approximating higher-order problem signatures in Artificial Neural Networks with hybrid transfer functions. *International Journal of Computer Science Issues*, 11(2):8–18, 2014.

[8] D. Albanese, R. Visintainer, S. Merler, S. Riccadonna, G. Jurman, and C. Furlanello. mlpy: Machine learning python, 2012.

[9] E. Artyomov and O. Yadid-Pecht. Modified high-order neural network for invariant pattern recognition. *Pattern Recognition Letters*, 26(6):843–851, 2005.

[10] A. D. Association. Lower Your Risks: Age, Race, Gender & Family History, 2013.

[11] A. Azzini and A. G. Tettamanzi. Evolutionary anns: A state of the art survey. *Intelligenza Artificiale*, 5(1):19–35, 2011.

[12] A. Azzini, A. G. B. Tettamanzi, and M. Dragoni. SimBa-2 : Improving a Novel Similarity-Based Crossover for the Evolution of Artificial Neural Networks. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 374–379, 2011.

[13] K. Bache and M. Lichman. {UCI} Machine Learning Repository, 2013.

[14] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. Book in preparation for MIT Press, 2015.

[15] H. Bensusan and C. Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. *Principles of Data Mining and Knowledge Discovery*, 2000.

[16] S. A. Billings and G. L. Zheng. Radial basis function network configuration using genetic algorithms. *Neural Networks*, 8(6):877–890, 1995.

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

[18] K. L. Briggman and W. B. Kristan. Multifunctional pattern-generating circuits. *Annual review of neuroscience*, 31:2710294, 2008.

[19] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, Mar. 2005.

[20] G. Brown and X. Yao. On the effectiveness of negative correlation learning. *Proceedings of First UK Workshop on Computational Intelligence*, 2001.

[21] E. Bullmore and O. Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews. Neuroscience*, 10(3):186–98, Mar. 2009.

[22] M. Castellani and H. Rowlands. Evolutionary artificial neural network design and training for wood veneer classification. *Engineering Applications of Artificial Intelligence*, 22(4):732–741, 2009.

[23] B. I. D. M. Center. To learn better, take a nap (and don't forget to dream), April 26 2010.

[24] CERI. *Understanding the Brain: The Birth of a Learning Science*. CERI, 2007.

[25] A. Chandra and X. Yao. Evolutionary Framework for the Construction of Diverse Hybrid Ensembles. In *European Symposium on Artificial Neural Networks (ESANN)*, pages 253–258, 2005.

[26] A. Chandra and X. Yao. Ensemble Learning Using Multi-Objective Evolutionary Algorithms. *Journal of Mathematical Modelling and Algorithms*, 5(4):417–445, Mar. 2006.

[27] A. Chandra and X. Yao. Evolving hybrid ensembles of learning machines for better generalisation. *Neurocomputing*, 69(7-9):686–700, Mar. 2006.

[28] P. Chandra and Y. Singh. A case for the self-adaptation of activation functions in FFANNs. *Neurocomputing*, 56(1-4):447–454, Jan. 2004.

[29] P. Chandra and Y. Singh. An activation function adapting training algorithm for sigmoidal feedforward networks. *Neurocomputing*, 61:429–437, Oct. 2004.

[30] I. Ciocoiu. Hybrid feedforward neural networks for solving classification problems. *Neural Processing Letters*, pages 81–91, 2002.

[31] S. Cohen and N. Intrator. A Hybrid Projection-based and Radial Basis Function Architecture: Initial Values and Global Optimisation. *Pattern Analysis & Applications*, 5(2):113–120, June 2002.

[32] Y. L. Cun, J. S. Denker, and S. A. Solla. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*, volume 2, pages 598–605. Morgan Kaufmann, 1990.

[33] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78, Oct. 2012.

[34] M. Dragoni, A. Azzini, and A. G. Tettamanzi. SimBa: A novel similarity-based crossover for neuro-evolution. *Neurocomputing*, pages 1–15, July 2013.

[35] W. Duch and N. Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2:163–212, 1999.

[36] W. Duch and N. Jankowski. Taxonomy of neural transfer functions. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, 3:477–482, 2000.

[37] W. Duch and N. Jankowski. Transfer functions: hidden possibilities for better neural networks. In *9th European Symposium on Artificial Neural Networks, ESANN '01*, pages 81–94, Bruges, 2001.

[38] W. Duch, N. Jankowski, and W. Duch. Bi-radial transfer functions. *9th European Symposium on Artificial Neural Networks*, 7(1):81–94, 1996.

[39] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby. Hybrid genetic algorithms: A review. *Engineering Letters*, 13(2):124–137, 2006.

[40] A. P. Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[41] F. Fernández-Navarro, C. Hervás-Martínez, P. A. Gutiérrez, and M. Carbonero-Ruz. Evolutionary q-Gaussian radial basis function neural networks for multiclassification. *Neural Networks*, 24(7):779–84, Sept. 2011.

[42] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, Jan. 2008.

[43] C. Gagn. DEAP : Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13(13):2171–2175, 2012.

[44] N. García-Pedrajas. Covnet: a cooperative coevolutionary model for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 14(3):575–596, 2003.

[45] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 1992.

[46] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *Journal of Machine Learning Research*, 9:937–965, 2008.

[47] P. Gutiérrez, C. Hervás, M. Carbonero, and J. Fernández. Combined projection and kernel basis functions for classification in evolutionary neural networks. *Neurocomputing*, 72(13-15):2731–2742, Aug. 2009.

[48] P. Gutiérrez and C. Hervás-Martnez. Hybrid Artificial Neural Networks : Models , Algorithms and Data. *Lecture Notes in Computer Science*, 6692(PART 2):177–184, 2011.

[49] P. P. Gutiérrez, C. Hervás, M. Carbonero, and J. J. Fernández. Combined projection and kernel basis functions for classification in evolutionary neural networks. *Neurocomputing*, 72(13-15):2731–2742, Aug. 2009.

[50] L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[51] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The ”wake-sleep” algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

[52] International Joint Conference on Artificial Intelligence. *Version spaces: A candidate elimination approach to rule learning*, volume 1 of *5*. Morgan Kaufmann Publishers Inc., 1977.

[53] M. M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 14(4):820–34, Jan. 2003.

[54] N. Jankowski. Flexible transfer functions with ontogenic neural networks. *Toru, Poland*, 1(6):1–6, 1999.

[55] N. Jankowski and W. Duch. Optimal transfer function neural networks. In *9th European Symposium on Artificial Neural Networks, ESANN 2001*, pages 101–106, Bruges, 2001.

[56] N. Jankowski and V. Kadirkamanathan. *Artificial Neural Networks — ICANN'97: 7th International Conference Lausanne, Switzerland, October 8–10, 1997 Proceeedings*, chapter Statistical control of RBF-like networks for classification, pages 385–390. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[57] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2016-05-16].

[58] M. T. Jones. *Artificial Intelligence A System Approach*. Laxmi Publications, Ltd., 2008.

[59] M. Kaul, R. L. Hill, and C. Walthall. Artificial neural networks for corn and soybean yield prediction. *Agricultural Systems*, 85:1–18, 2005.

[60] K. Kim and S. Cho. Evolutionary ensemble of diverse artificial neural networks using speciation. *Neurocomputing*, 71(7-9):1604–1618, Mar. 2008.

[61] P. Kordík, J. Koutník, J. Drchal, O. Kovárík, M. Cepek, and M. Snorek. Meta-learning approach to neural network optimization. *Neural networks : the official journal of the International Neural Network Society*, 23(4):568–82, May 2010.

[62] D. Kriesel. A Brief Introduction to Neural Networks. 2007.

[63] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in Neural Information Processing Systems*, 7:231–238, 1995.

[64] M. Lehtokangas and J. Saarinen. Centroid based Multilayer Perceptron Networks. *Neural Processing Letters*, 7(2):101–106, 1998.

[65] Y. Liu and X. Yao. Evolutionary design of artificial neural networks with different nodes. *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 913–917, 1996.

[66] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural networks : the official journal of the International Neural Network Society*, 12(10):1399–1404, Dec. 1999.

[67] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.

[68] H. Lodish, A. Berk, S. L. Zipursky, P. Matsudaira, D. Baltimore, J. Darnell, et al. Overview of neuron structure and function, 2000.

[69] E. Marder. Invertebrate Neurobiology: Polymorphic neural networks. *Current Biology*, 4(8):752–754, Aug. 1994.

[70] E. V. E. Marder. Polymorphic neural networks Recent work on small invertebrate nervous systems. *Current Biology*, 4(8):752–754, 1994.

[71] T. Maul. Early experiments with neural diversity machines. *Neurocomputing*, 113:36–48, Mar. 2013.

[72] T. Maul and S. Baba. Unsupervised learning in second-order neural networks for motion analysis. *Neurocomputing*, 74(6):884–895, 2011.

[73] T. H. Maul, A. Bargiela, U. Malaysia, C. S. Yew, and A. S. Adamu. Towards evolutionary deep neural networks. ECMS, 2014.

[74] R. J. Meuth. Mutation operator evolution for ea-based neural networks. 2005.

[75] T. M. Mitchell. The Need for Biases in Learning Generalizations. *Readings in Machine Learning*, (CBM-TR-117):184–191, 1980.

[76] M. J. Moghaddam and H. Soltanian-Zadeh. *Medical Image Segmentation Using Artificial Neural Networks*. INTECH Open Access Publisher, 2011.

[77] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.

[78] D. E. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

[79] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Aritificial Intelligence Research*, 11:169–198, 2011.

[80] D. Opitz and J. Shavlik. Generating Accurate and Diverse Members of a Neural-Network Ensemble. *Advances in neural information processing Systems*, 8:535–541, 1996.

[81] Y. Peng, P. Flach, P. Brazdil, and C. Soares. Decision tree-based data characterization for meta-learning. In *2nd International Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning, IDDM '02*, number 3, pages 111–122. Helsinki, 2002.

[82] M. Perrone and L. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. *Neural Networks for Speech and Image Processing*, pages 126–142, 1993.

[83] S. Peter and A. Krogh. Learning with ensembles: How over-fitting can be useful. In *Advances in Neural Information Processing Systems*, volume 8, pages 4–10. MIT Press, 1996.

[84] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-Learning by Landmarking Various Learning Algorithms. *Proceedings of the Seventeenth International Conference on Machine Learning ICML2000*, 951(2000):743–750, 2000.

[85] M. Potter and K. D. Jong. A cooperative coevolutionary approach to function optimization. *Parallel Problem Solving from Nature - PPSN III*, pages 249 – 257, 1994.

[86] M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel problem solving from naturePPSN III*, pages 249–257. Springer, 1994.

[87] M. A. Potter and K. A. De Jong. Cooperative coevolution: an architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, Jan. 2000.

[88] M. A. Potter, K. A. De Jong, and K. D. Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, Jan. 2000.

[89] L. Prechelt. Proben1: A set of neural network benchmark problems and benchmarking rules. *Technicak Report*, 21(19/94):94, 1994.

[90] L. Rendell, R. Seshu, and D. Tcheng. Layered concept-learning and dynamically-variable bias management. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI '87.*, pages 308—-314, Milan, 1987.

[91] M. Rubinov and O. Sporns. Complex network measures of brain connectivity: uses and interpretations. *NeuroImage*, 52(3):1059–69, Sept. 2010.

[92] S. Russell, P. Norvig, and A. Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.

[93] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010.

[94] A. Sharkey and N. Sharkey. Combining diverse neural nets. *The Knowledge Engineering Review*, 12(3):231–247, 1997.

[95] Y. Singh and P. Chandra. A class +1 sigmoidal activation functions for ffanns. *Journal of Economic Dynamics and Control*, 28(1):183–187, Oct. 2003.

[96] O. Sporns. Graph theory methods for the analysis of neural connectivity patterns. In *Neuroscience Databases*, pages 169–183. Springer, 2003.

[97] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving adaptive neural networks with and without adaptive synapses. In *Evolutionary Computation, 2003. CEC '03.*, volume 4, pages 2557–2564. IEEE, Dec 2003.

[98] K. O. Stanley and R. Miikkulainen. Efficient Reinforcement Learning Through Evolving Neural Network Topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference , GECCO-2002*, volume 10, pages 569–577, San Francisco, 2002. Morgan Kaufmann.

[99] J.-p. Thivierge. Neural diversity creates a rich repertoire of brain activity. *Communicative & integrative biology*, 1(2):188–189, 2008.

[100] S. Thrun. Learning to learn: Introduction. *In Learning To Learn*, 1996.

[101] S. J. Tripathy, K. Padmanabhan, R. C. Gerkin, and N. N. Urban. Intermediate intrinsic diversity enhances neural population coding. *Proceedings of the National Academy of Sciences of the United States of America*, 110(20):8248–53, 2013.

[102] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[103] R. Vilalta, C. Giraud-Carrier, and P. Brazdil. Meta-learning - concepts and techniques. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 717–731. Springer US, 2010.

[104] K. A. Wesson. From Synapses to LearningUnderstanding Brain Processes, 2012.

[105] L. D. Whitley et al. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *ICGA*, pages 116–123, 1989.

[106] Z. Wu and Y. Chen. Genetic algorithm based selective neural network ensemble. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 201*, 1:797–802, 2001.

[107] X. Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8(4):539–567, 1993.

[108] X. Yao. Evolutionary Artificial Neural Networks. *International Journal of Intelligent Systems*, 4(3), 1993.

[109] X. Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87:1423–1447, 1999.

[110] X. Yao and M. Islam. Evolving artificial neural network ensembles. *IEEE Computational Intelligence Magazine*, 3(1):31–42, Feb. 2008.

[111] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 8(3):694–713, 1997.

[112] X. Yao and Y. Liu. Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, 91(1):83–90, 1998.

[113] L. Yu, K. K. Lai, and S. Wang. Multistage RBF neural network ensemble learning for exchange rates forecasting. *Neurocomputing*, 71(16-18):3295–3302, Oct. 2008.

[114] G. Zhang. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 14(1):35–62, Mar. 1998.

[115] G. Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000.

[116] G. Zhang, B. E. Patuwo, and M. Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.

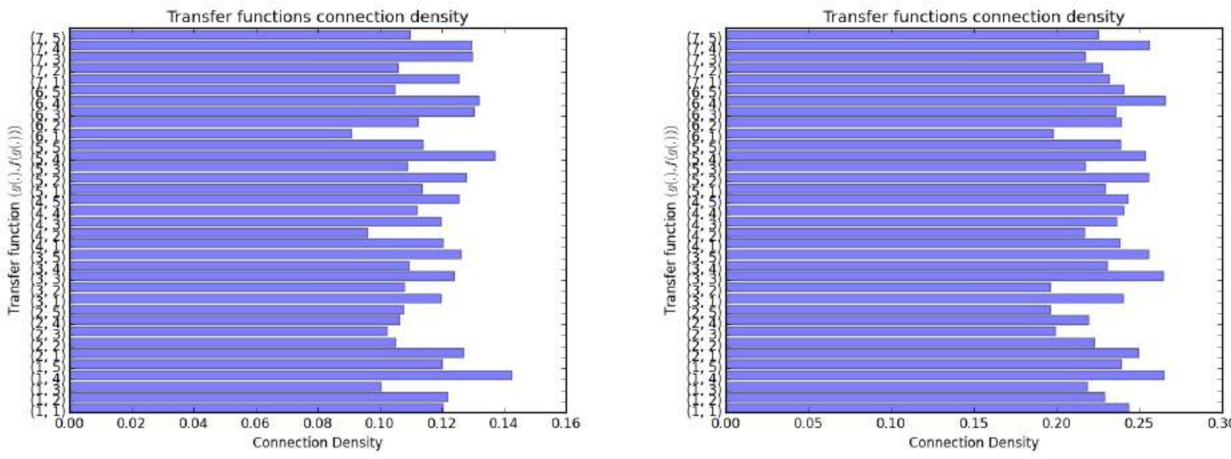
Appendix A

Appendix

A.1 Consistency of Lower-Order Signatures

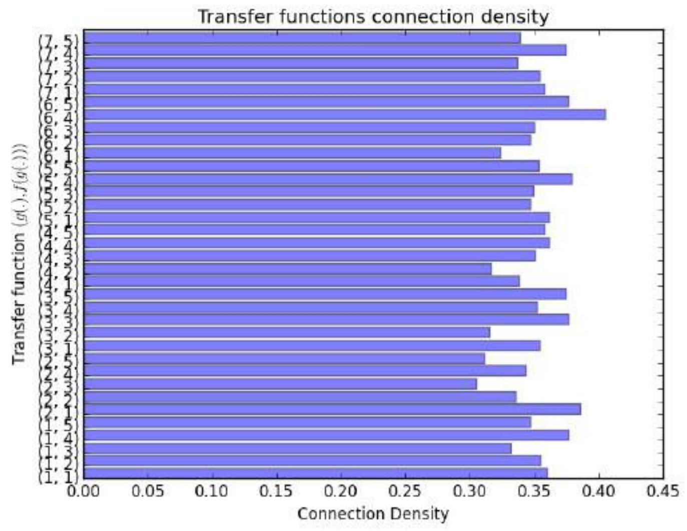
A.1.1 Noise Levels

The figures below show the results of connection density and transfer function likelihood averages for various levels of noise γ. It also presents the results of the connection density and transfer function likelihood averages after thresholding for the various noise level.

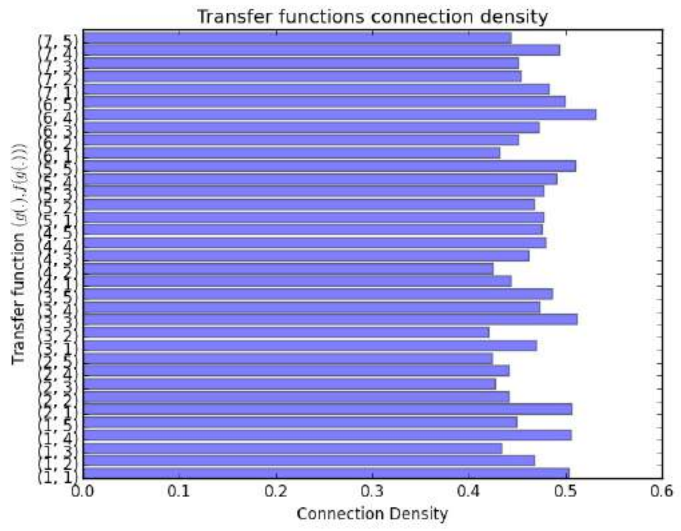


(a) $\gamma = 0.1$ **(b)** $\gamma = 0.2$

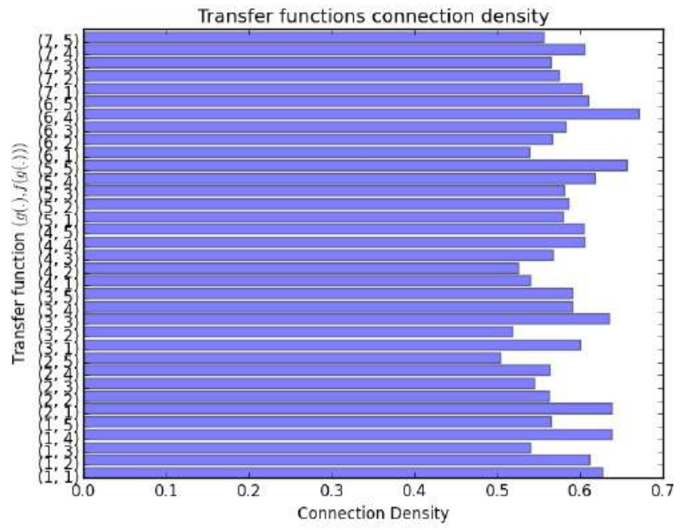
Figure A.1: The average connection density for the range of noise levels $\gamma \in \{0.1, 0.2\}$ on the Iris dataset.



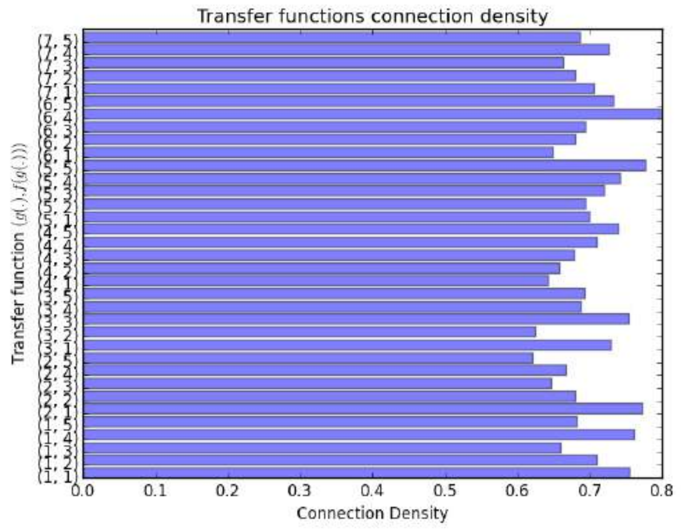
(a) $\gamma = 0.3$



(b) $\gamma = 0.4$

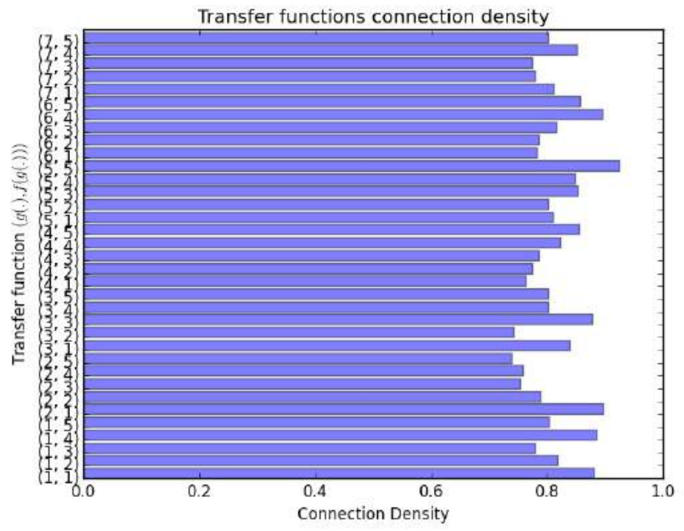


(c) $\gamma = 0.5$

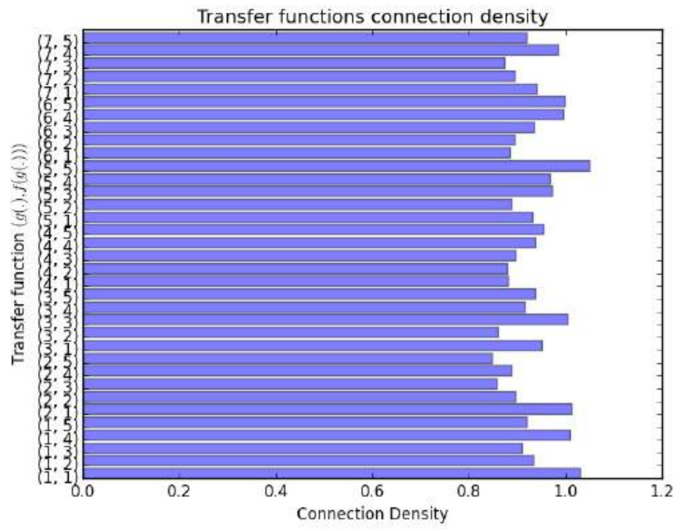


(d) $\gamma = 0.6$

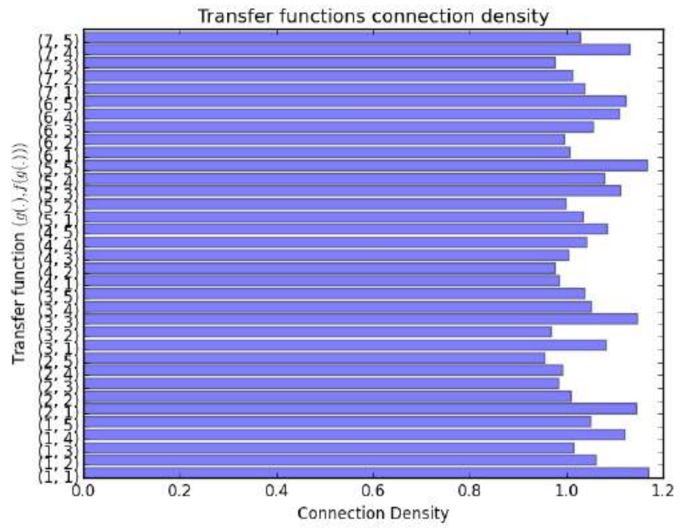
Figure A.2: The average connection density for the range of noise levels $\gamma \in \{0.3..0.6\}$ on the Iris dataset.



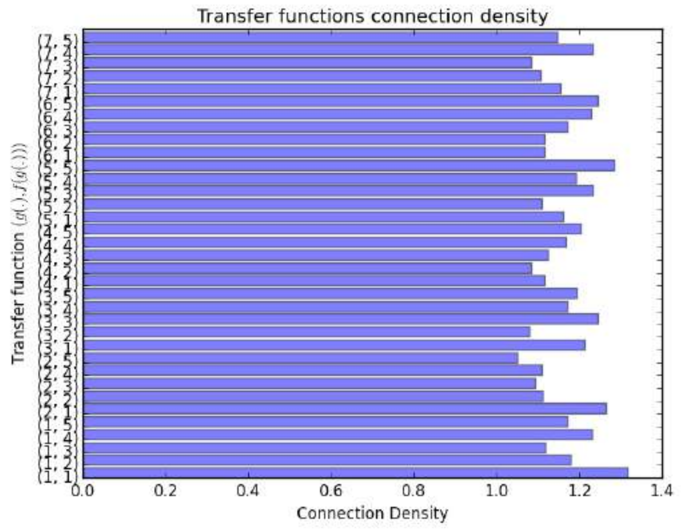
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$

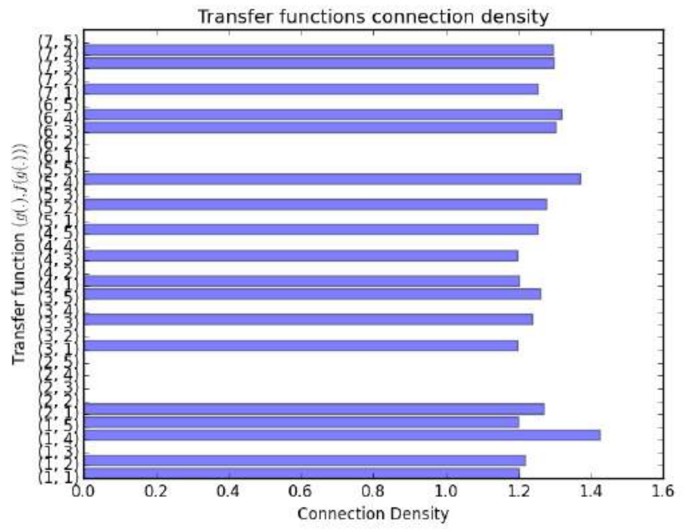


(c) $\gamma = 0.9$

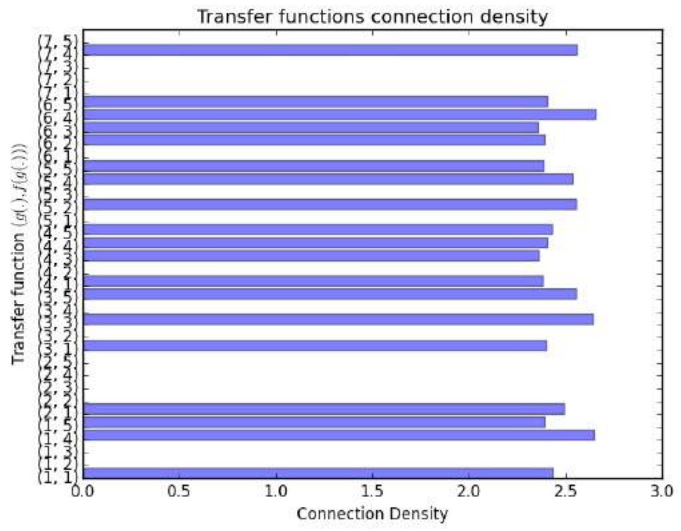


(d) $\gamma = 1.0$

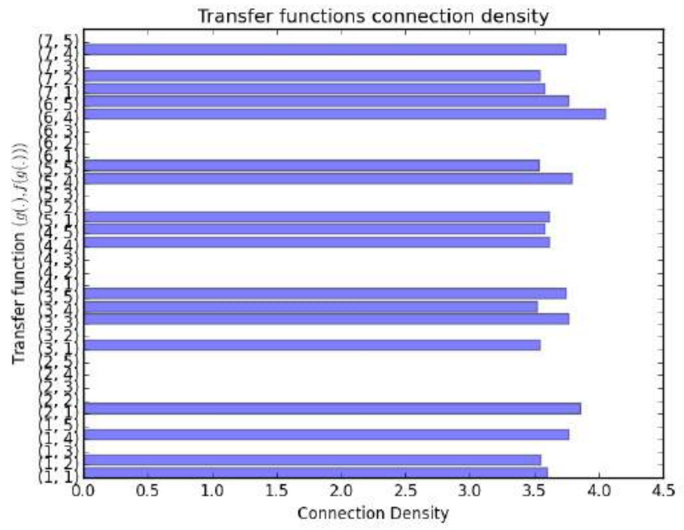
Figure A.3: The average connection density for the range of noise levels $\gamma \in \{0.7..1.0\}$ on the Iris dataset.



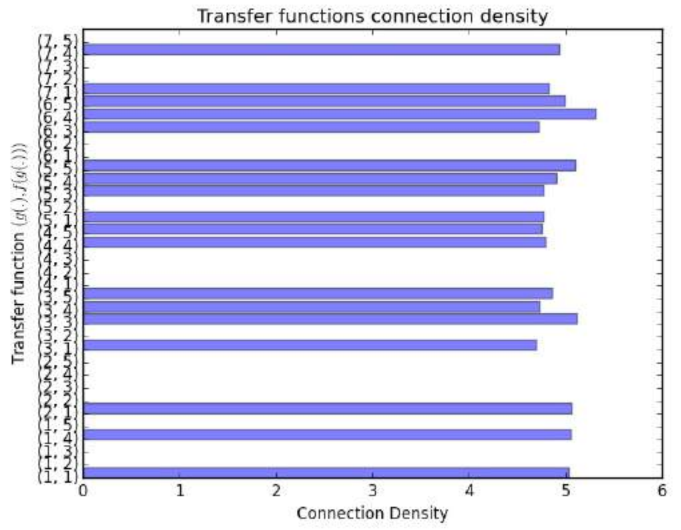
(a) $\gamma = 0.1$



(b) $\gamma = 0.2$

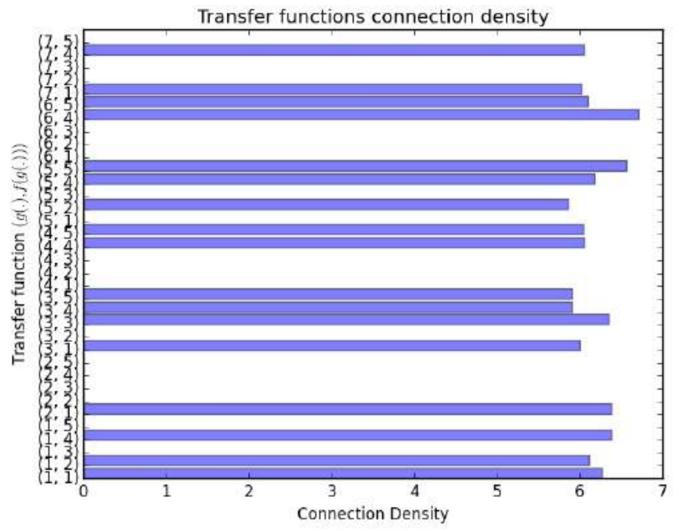


(c) $\gamma = 0.3$

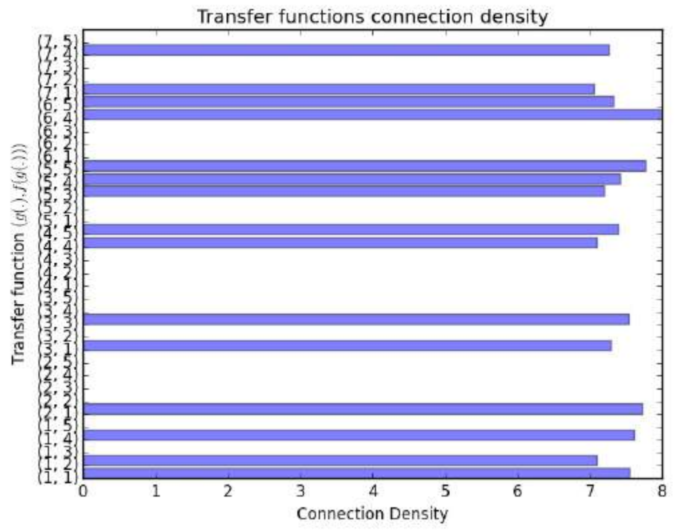


(d) $\gamma = 0.4$

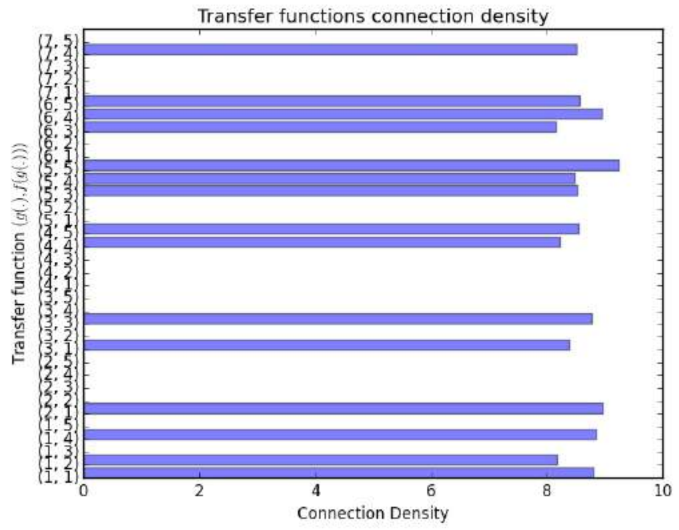
Figure A.4: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.1..0.4\}$ on the Iris dataset.



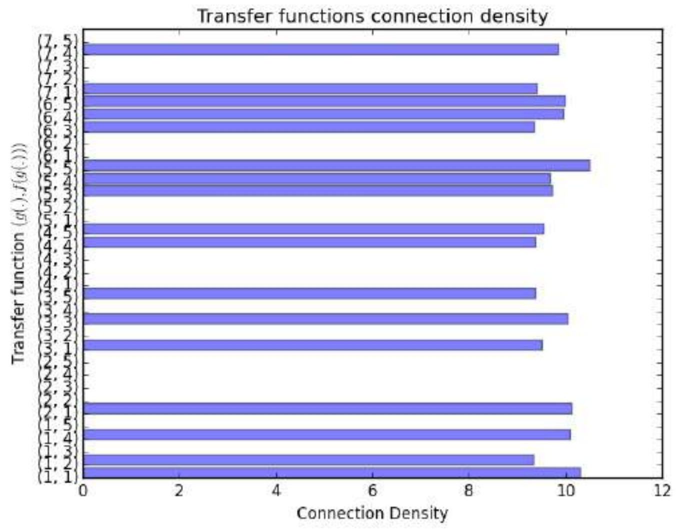
(a) $\gamma = 0.5$



(b) $\gamma = 0.6$

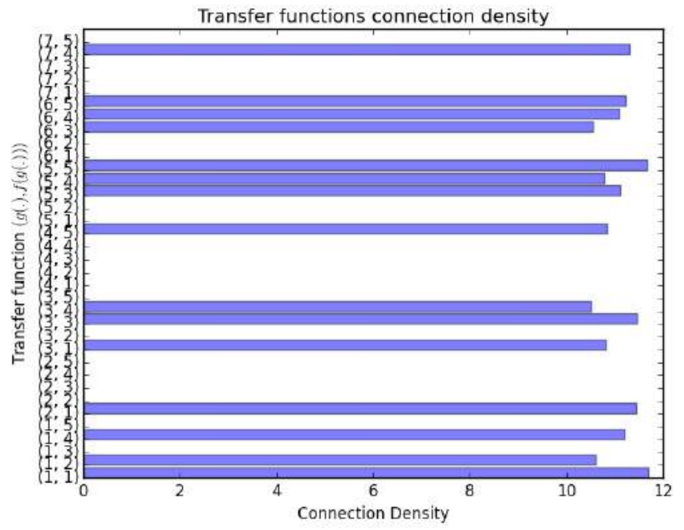


(c) $\gamma = 0.7$

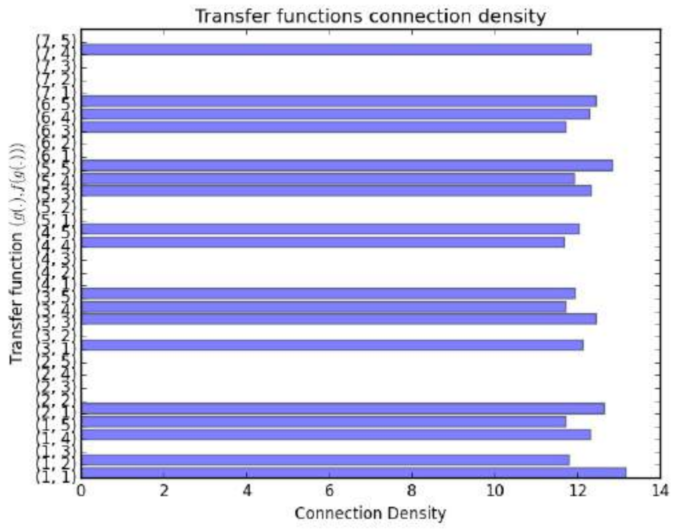


(d) $\gamma = 0.8$

Figure A.5: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.5..0.8\}$ on the Iris dataset.

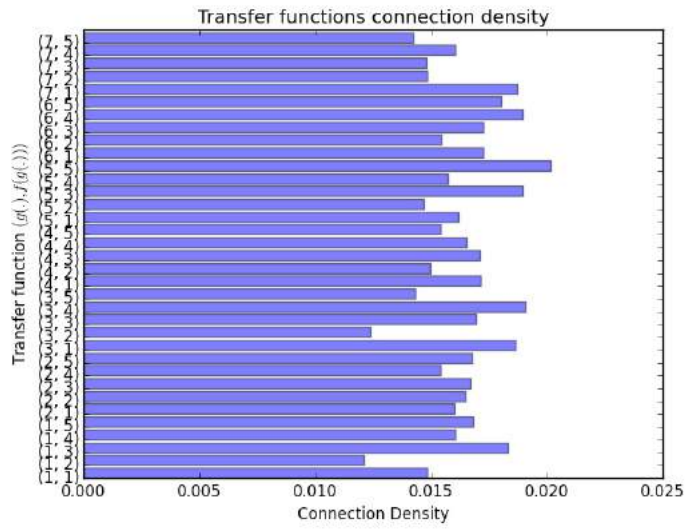


(a) $\gamma = 0.9$

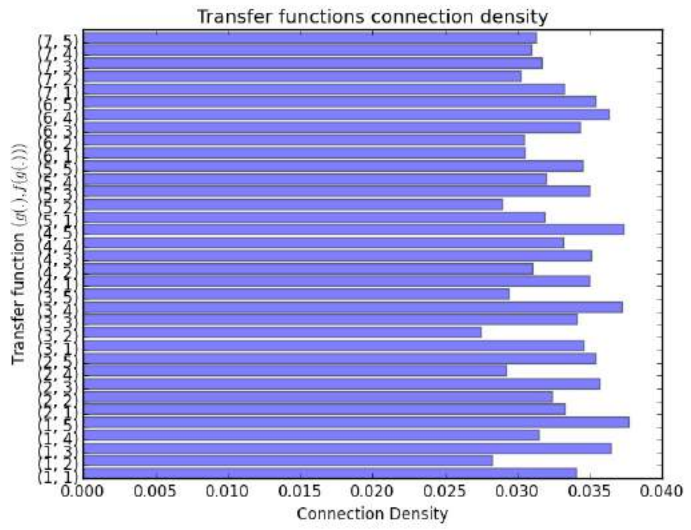


(b) $\gamma = 1.0$

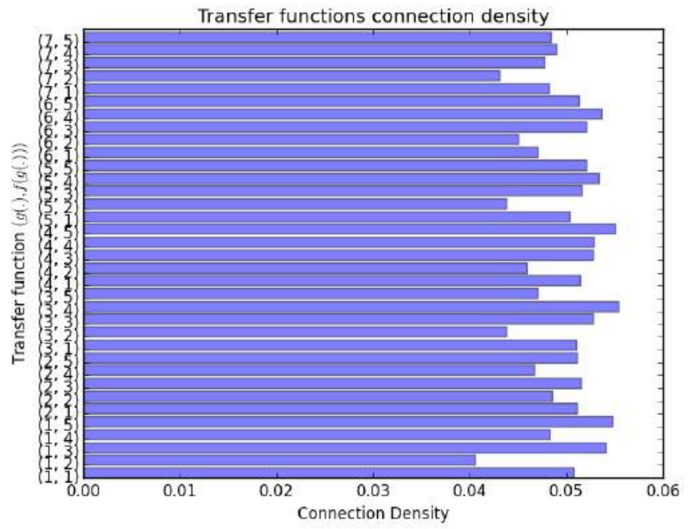
Figure A.6: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.9, 1.0\}$ on the Iris dataset.



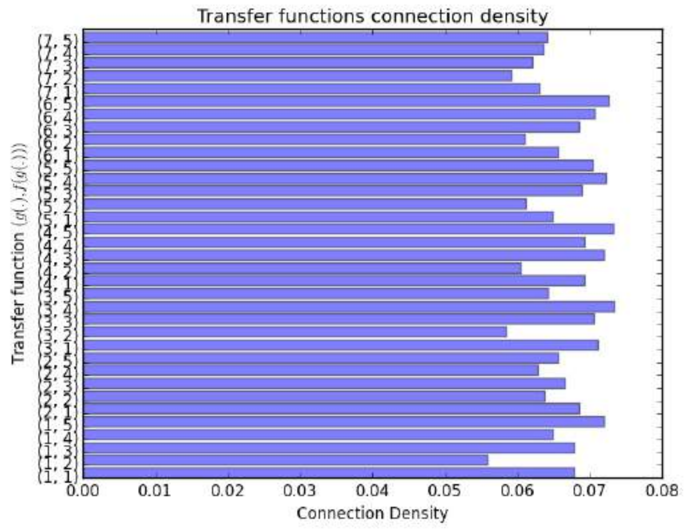
(a) $\gamma = 0.1$



(b) $\gamma = 0.2$

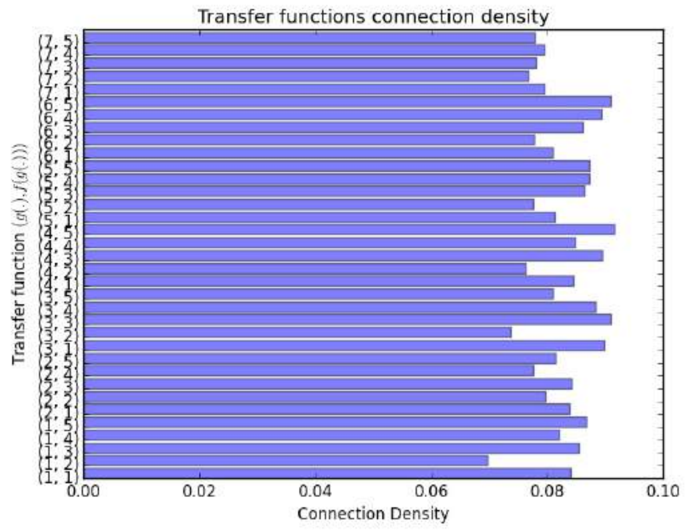


(c) $\gamma = 0.3$

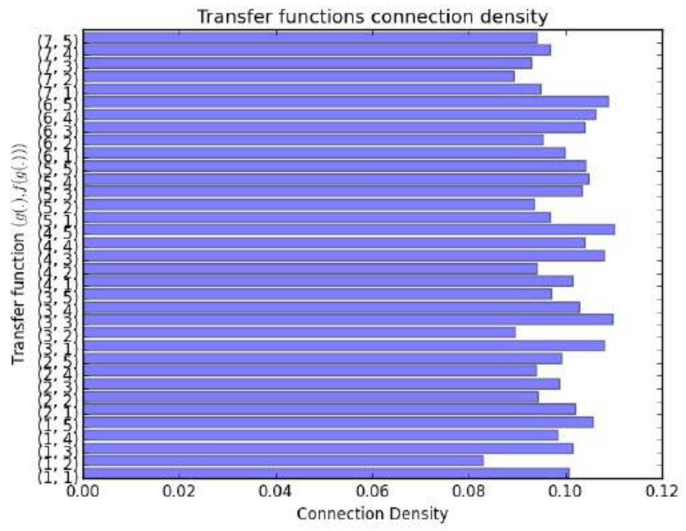


(d) $\gamma = 0.4$

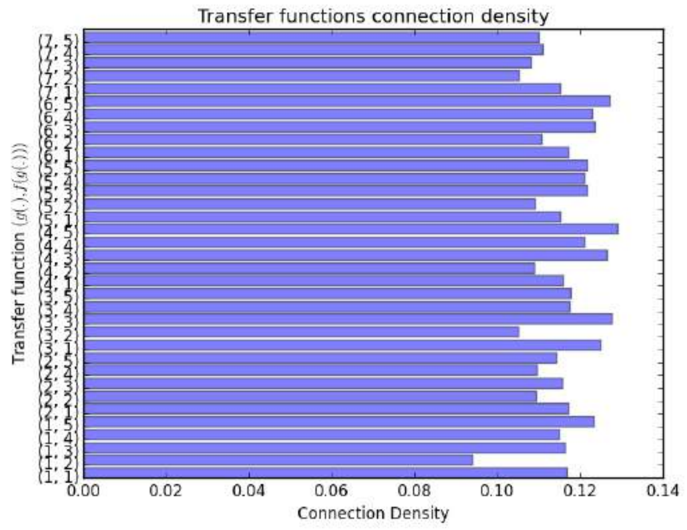
Figure A.7: The average connection density for the range of noise levels $\gamma \in \{0.1..0.4\}$ on the Sonar dataset.



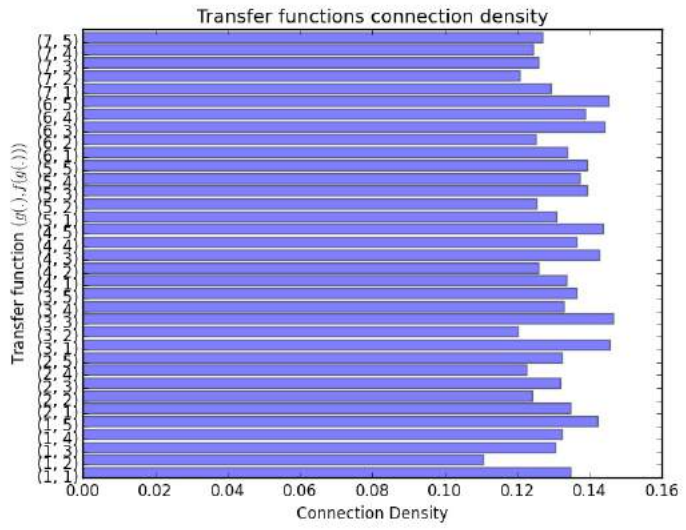
(a) $\gamma = 0.5$



(b) $\gamma = 0.6$

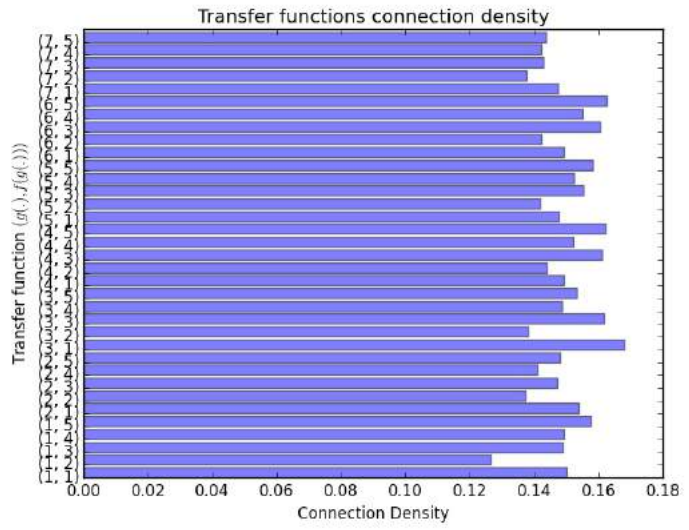


(c) $\gamma = 0.7$

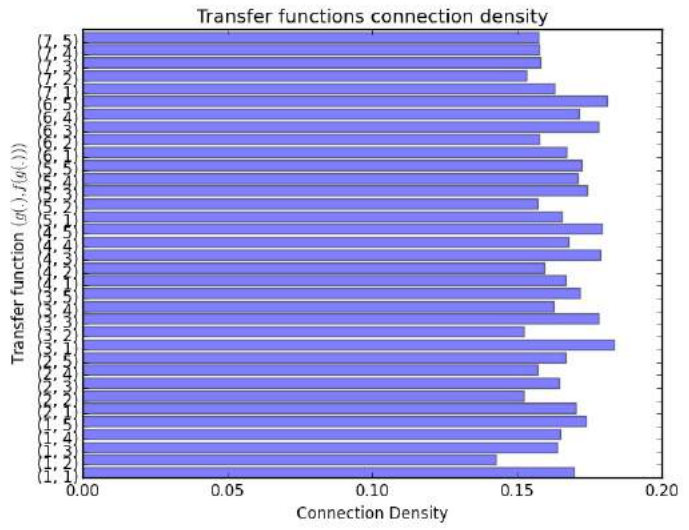


(d) $\gamma = 0.8$

Figure A.8: The average connection density matrices for the range of noise levels $\gamma \in \{0.5..0.8\}$ on the Sonar dataset.

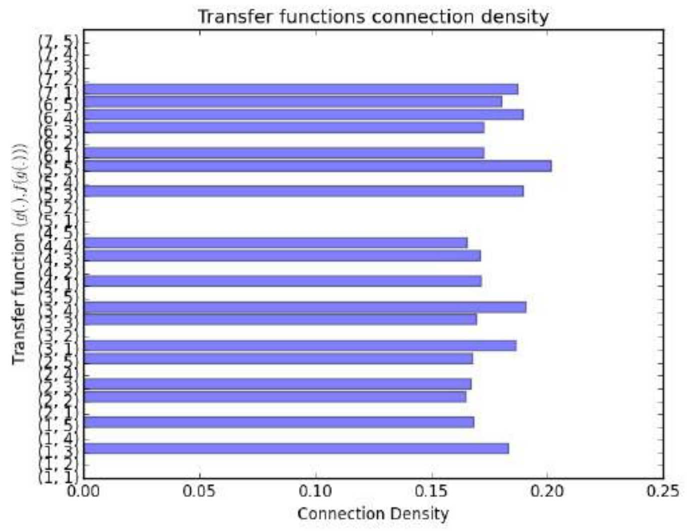


(a) $\gamma = 0.9$

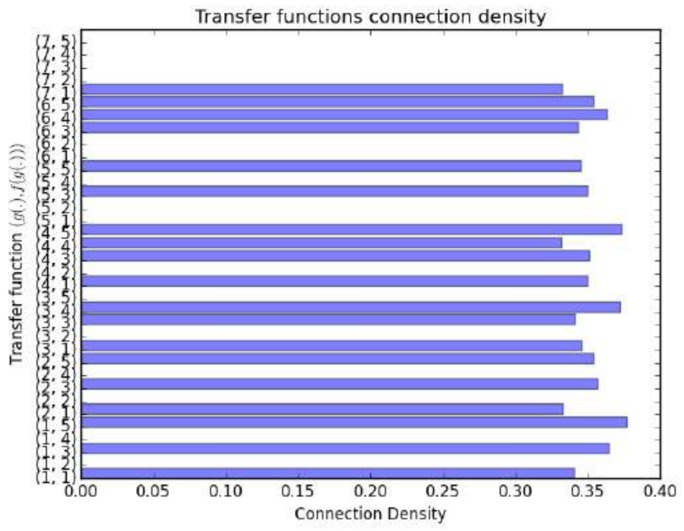


(b) $\gamma = 1.0$

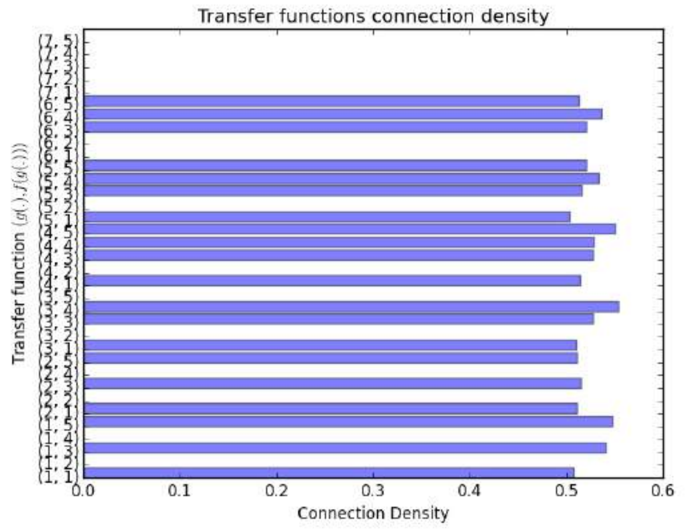
Figure A.9: The average connection density for the range of noise levels $\gamma \in \{0.9, 1.0\}$ on the Sonar dataset.



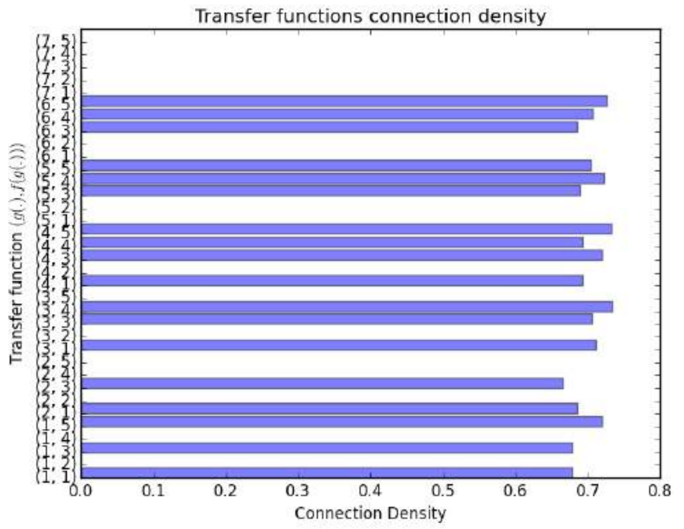
(a) $\gamma = 0.1$



(b) $\gamma = 0.2$

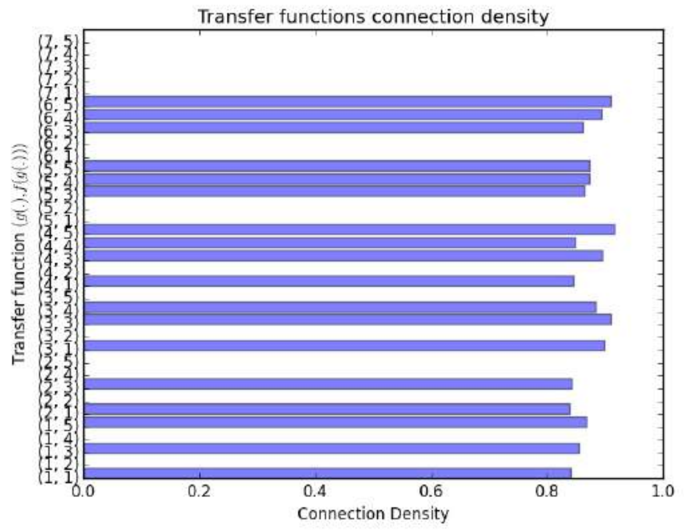


(c) $\gamma = 0.3$

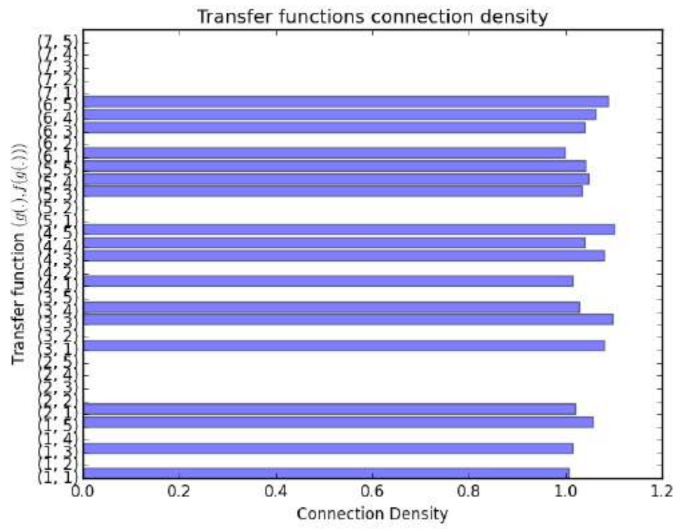


(d) $\gamma = 0.4$

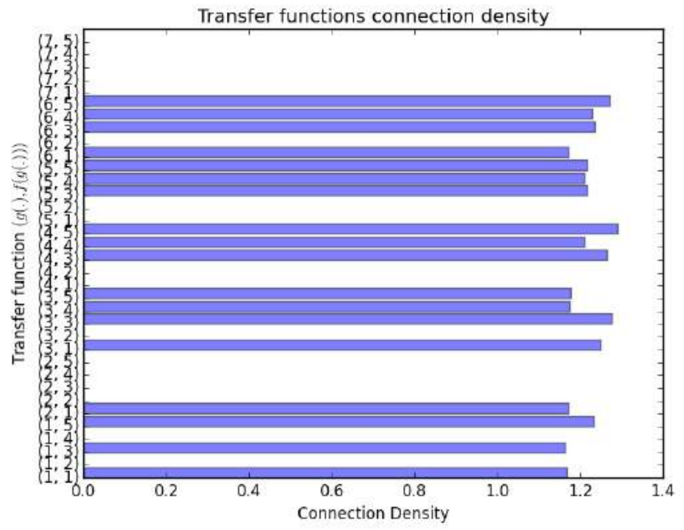
Figure A.10: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.1..0.4\}$ on the Sonar dataset.



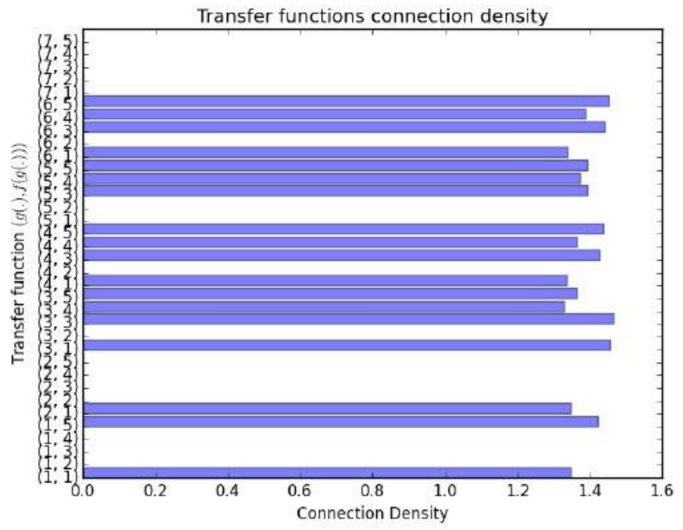
(a) $\gamma = 0.5$



(b) $\gamma = 0.6$

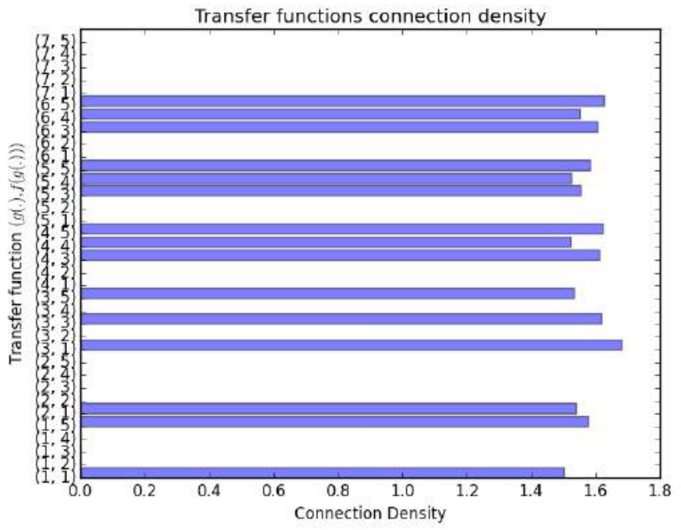


(c) $\gamma = 0.7$

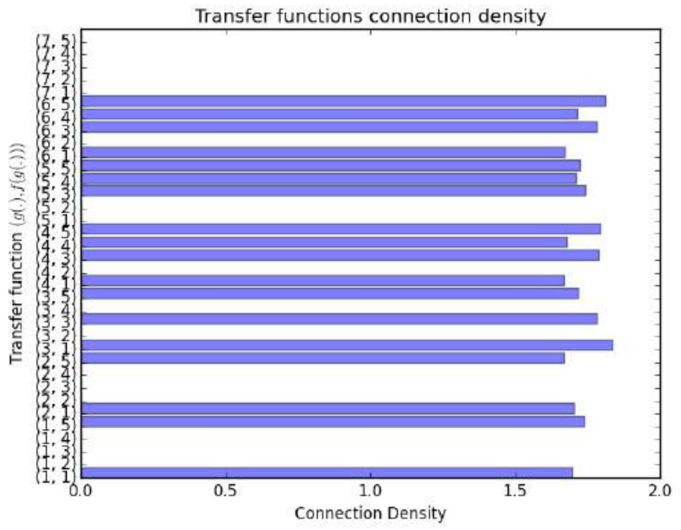


(d) $\gamma = 0.8$

Figure A.11: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.5..0.8\}$ on the Sonar dataset.

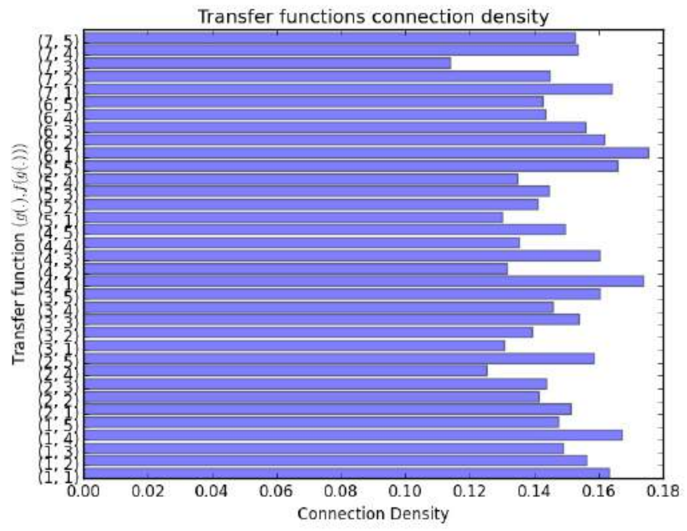


(a) $\gamma = 0.9$

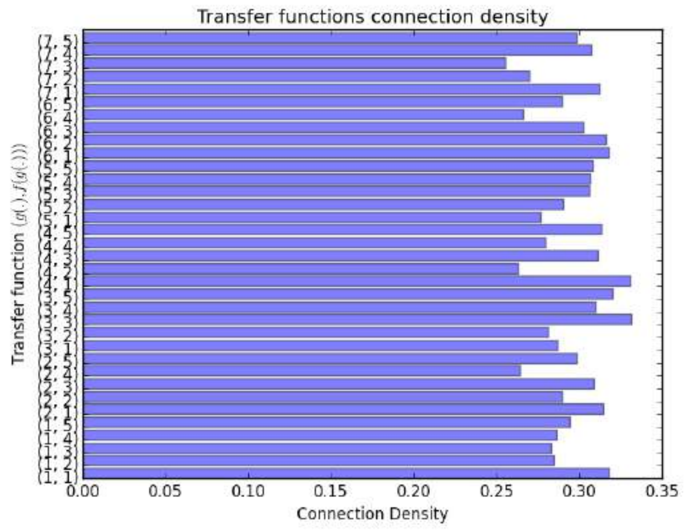


(b) $\gamma = 1.0$

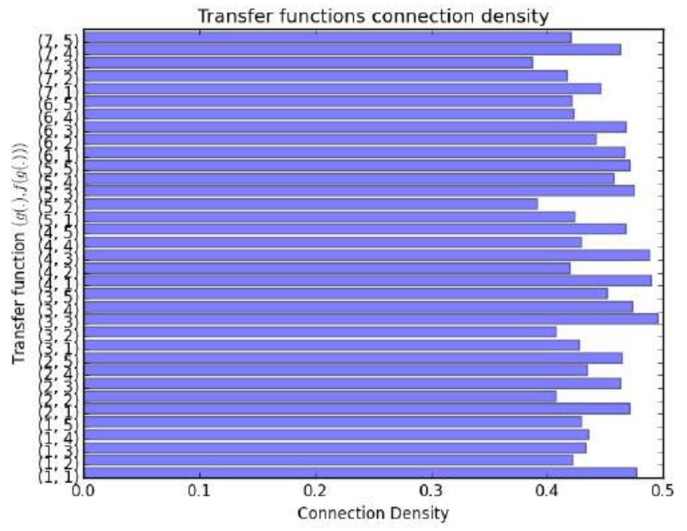
Figure A.12: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.9, 1.0\}$ on the Sonar dataset.



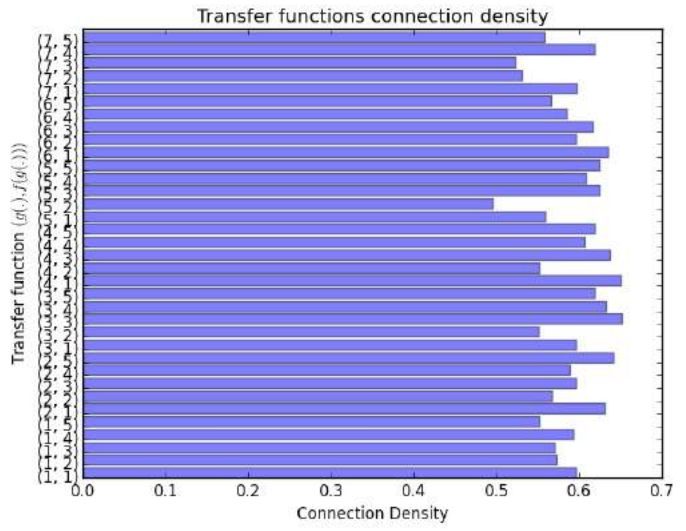
(a) $\gamma = 0.1$



(b) $\gamma = 0.2$

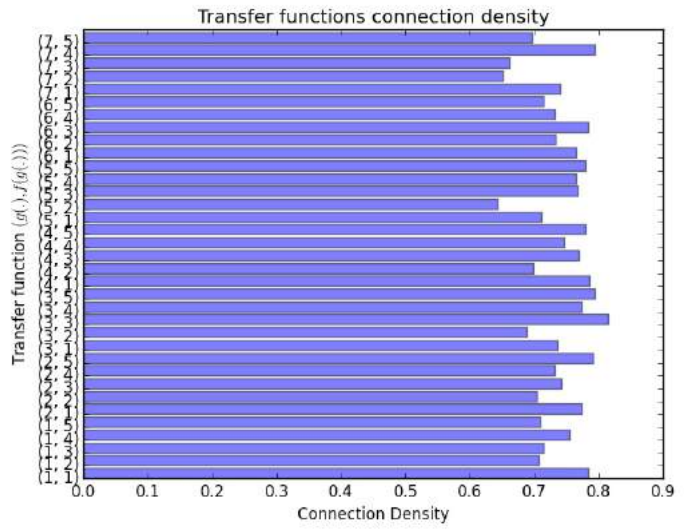


(c) $\gamma = 0.3$

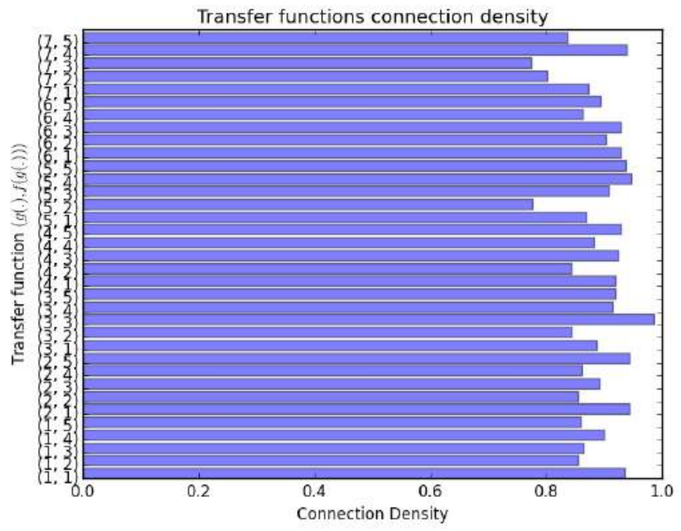


(d) $\gamma = 0.4$

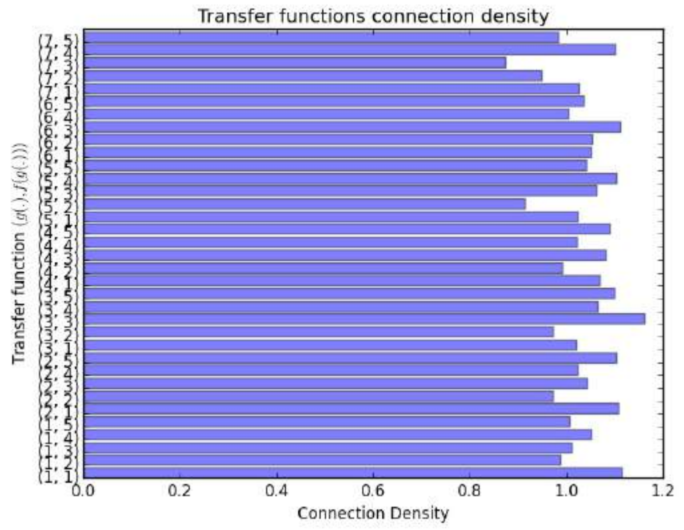
Figure A.13: The average connection density for the range of noise levels $\gamma \in \{0.1..0.4\}$ on the XOR dataset.



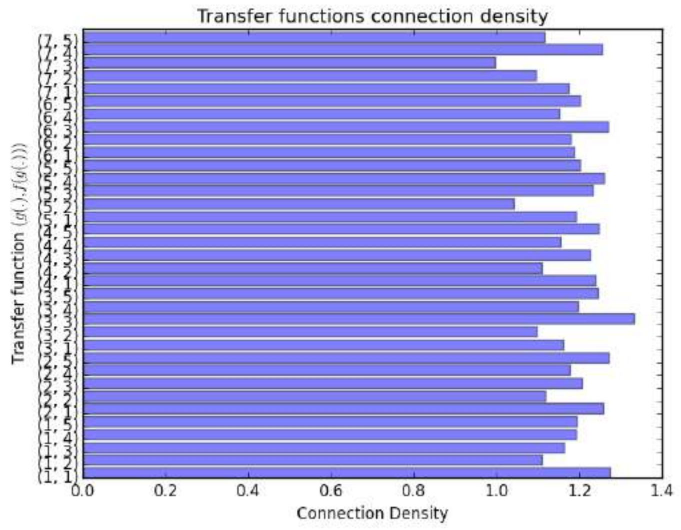
(a) $\gamma = 0.5$



(b) $\gamma = 0.6$

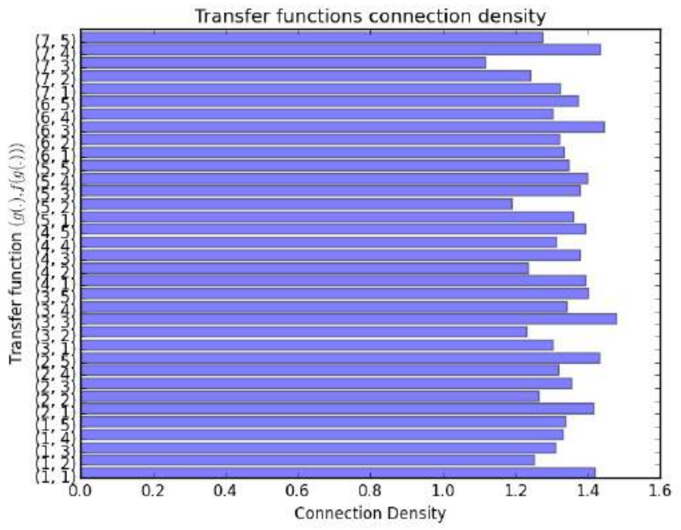


(c) $\gamma = 0.7$

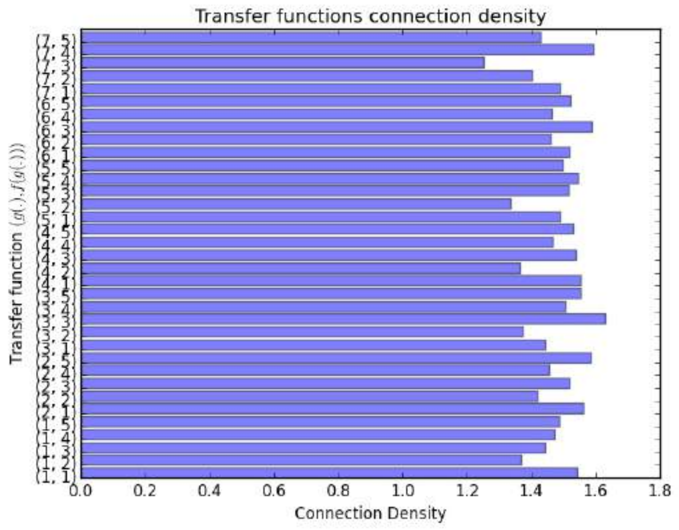


(d) $\gamma = 0.8$

Figure A.14: The average connection density for the range of noise levels $\gamma \in \{0.5..0.8\}$ on the XOR dataset.

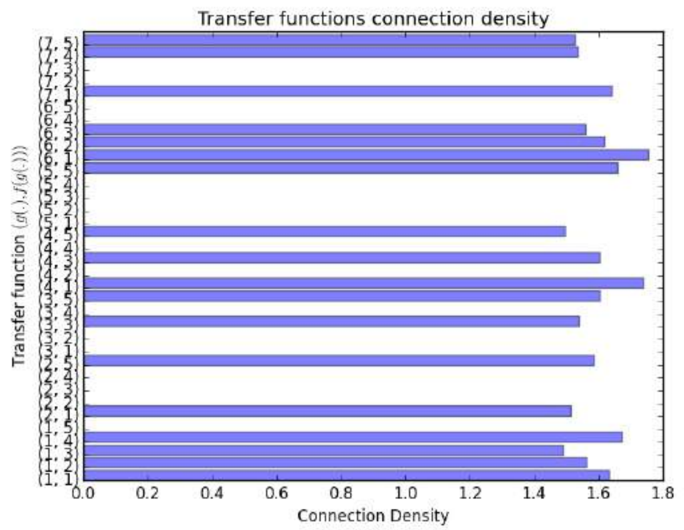


(a) $\gamma = 0.9$

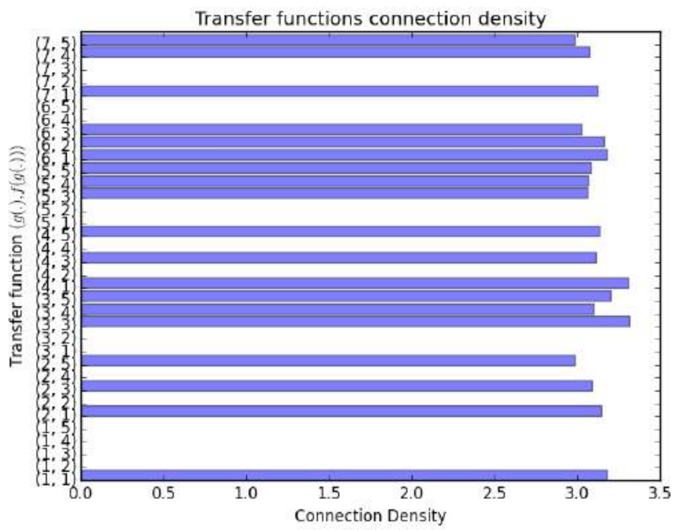


(b) $\gamma = 1.0$

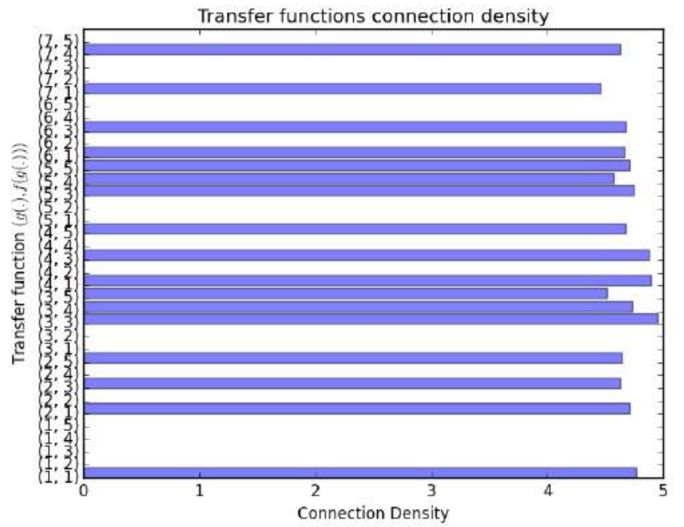
Figure A.15: The average connection density for the range noise levels $\gamma \in \{0.9, 1.0\}$ on the XOR dataset.



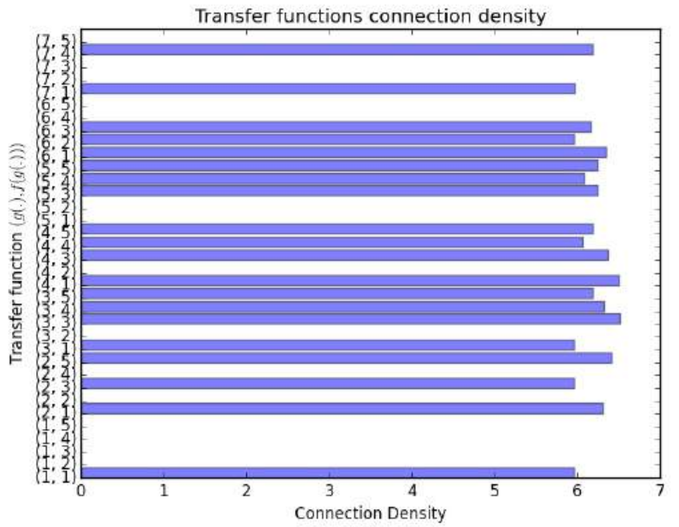
(a) $\gamma = 0.1$



(b) $\gamma = 0.2$

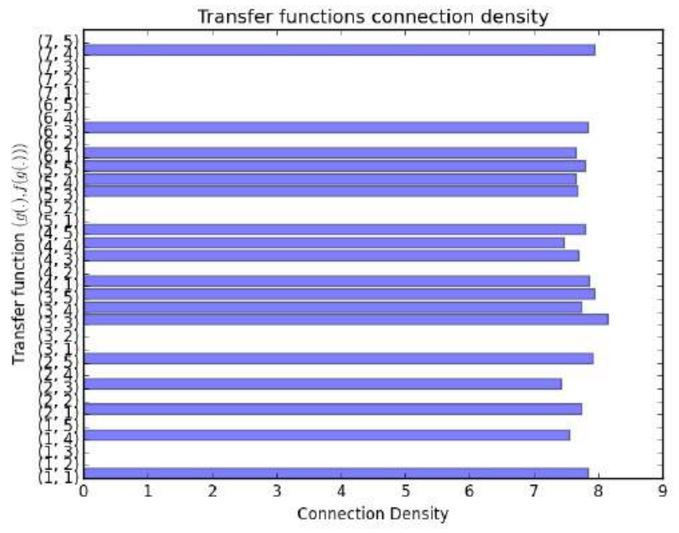


(c) $\gamma = 0.3$

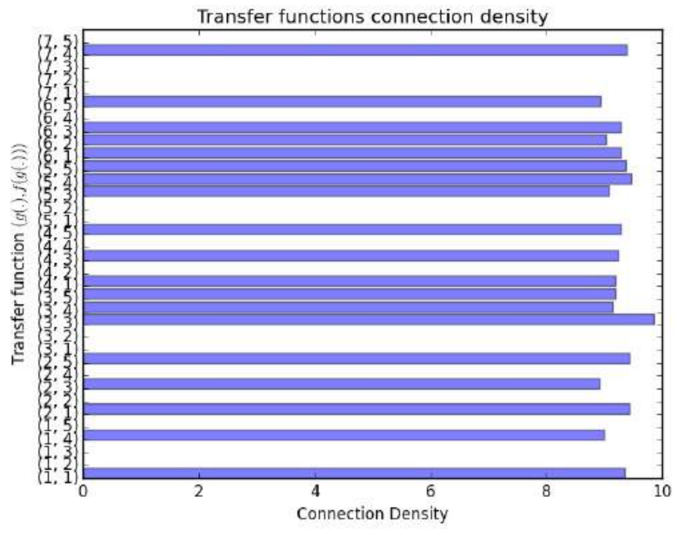


(d) $\gamma = 0.4$

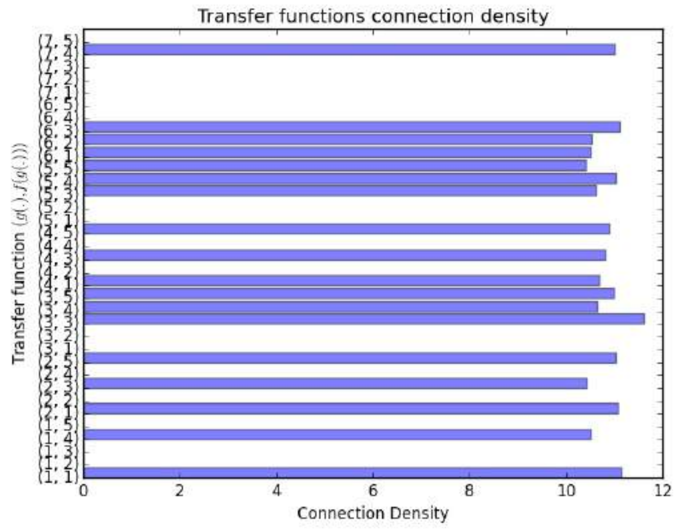
Figure A.16: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.1..0.4\}$ on the XOR dataset.



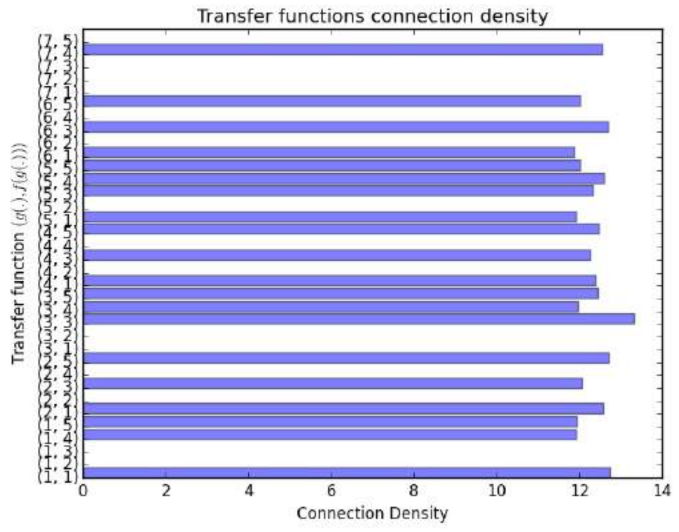
(a) $\gamma = 0.5$



(b) $\gamma = 0.6$

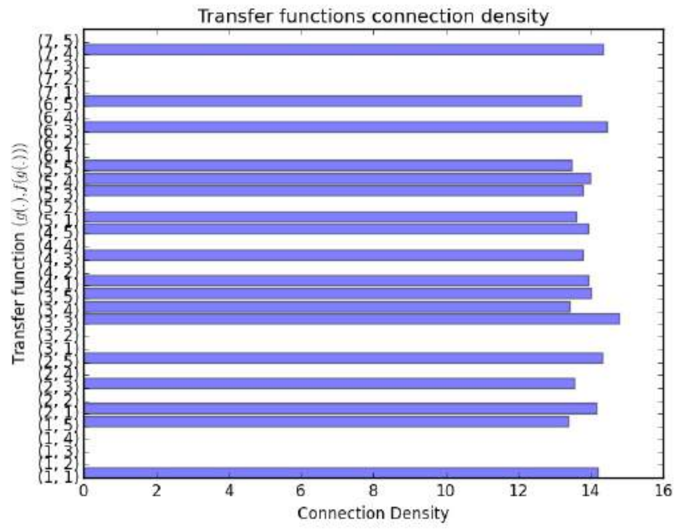


(c) $\gamma = 0.7$

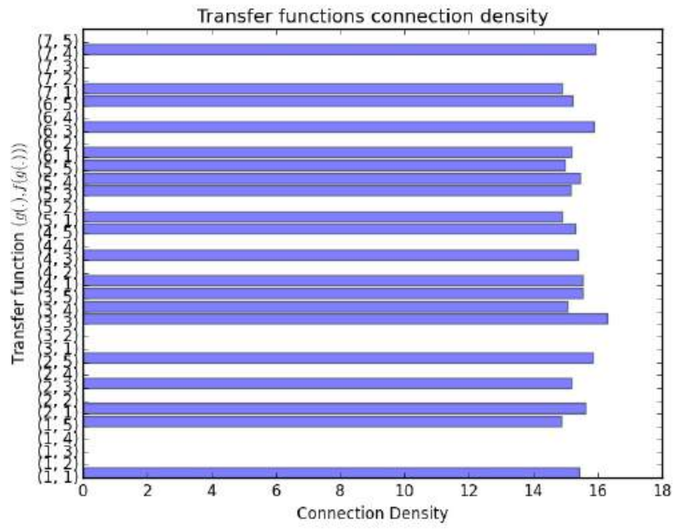


(d) $\gamma = 0.8$

Figure A.17: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.5..0.8\}$ on the XOR dataset.



(a) $\gamma = 0.9$

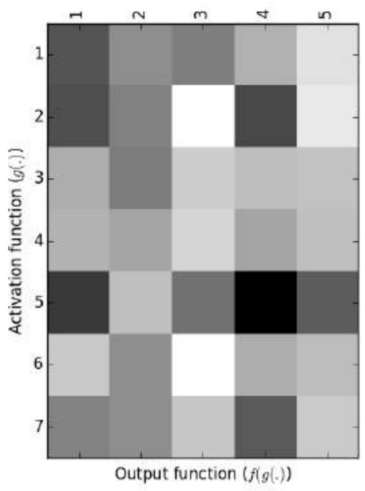


(b) $\gamma = 1.0$

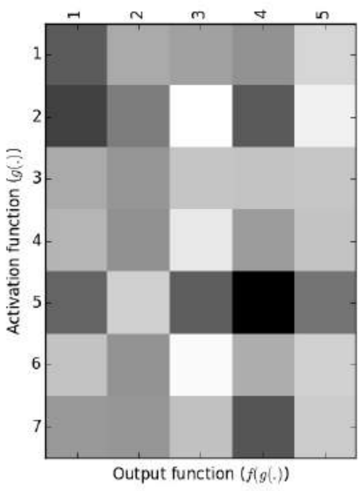
Figure A.18: The average connection density after thresholding for the range of noise levels $\gamma \in \{0.9, 1.0\}$ on the XOR dataset.

A.1.2 Size of N

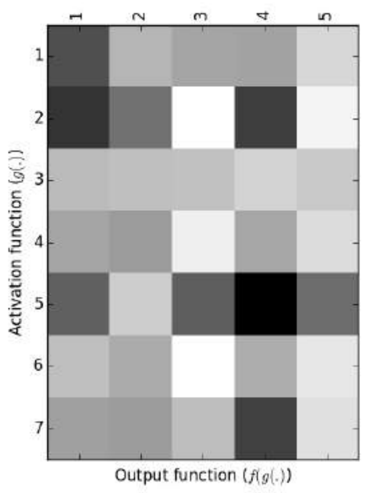
This shows the results for various sizes of N, i.e. the number of solutions top solutions selected for sampling the signatures after sorting.



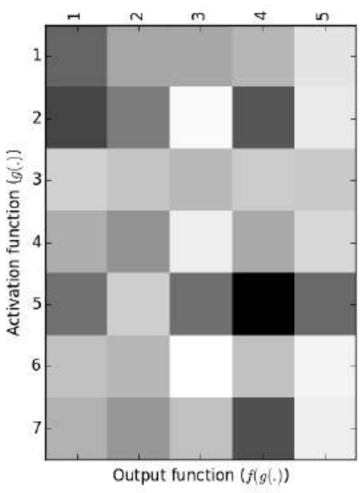
(a) $\gamma = 0.1$



(b) $\gamma = 0.2$

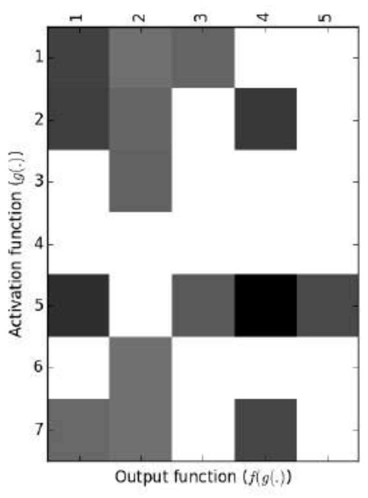


(c) $\gamma = 0.3$

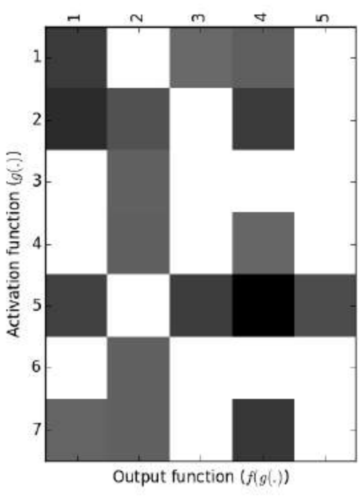


(d) $\gamma = 0.4$

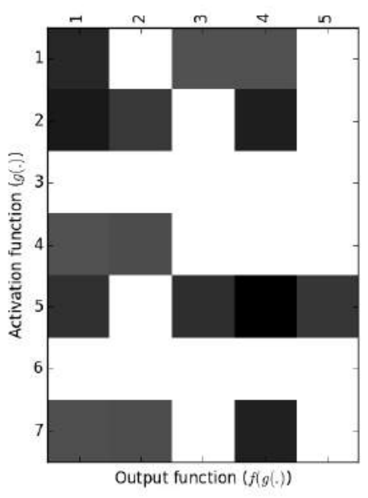
Figure A.19: The average transfer function likelihood for the sizes of $N \in \{1..4\}$ on the Iris dataset.



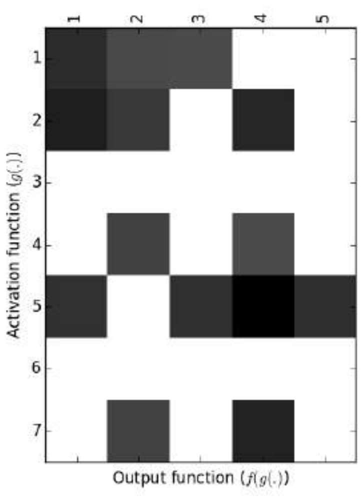
(a) $\gamma = 0.1$



(b) $\gamma = 0.2$

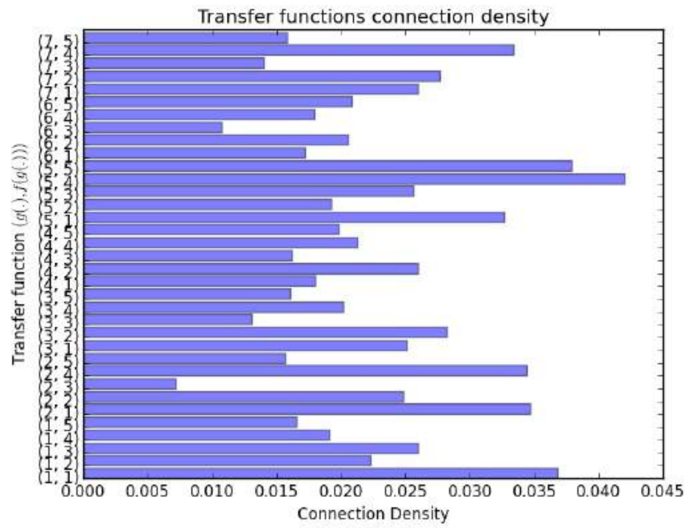


(c) $\gamma = 0.3$

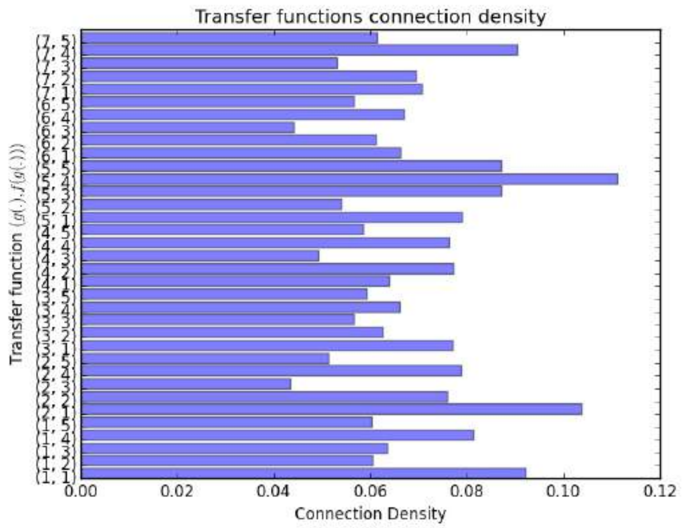


(d) $\gamma = 0.4$

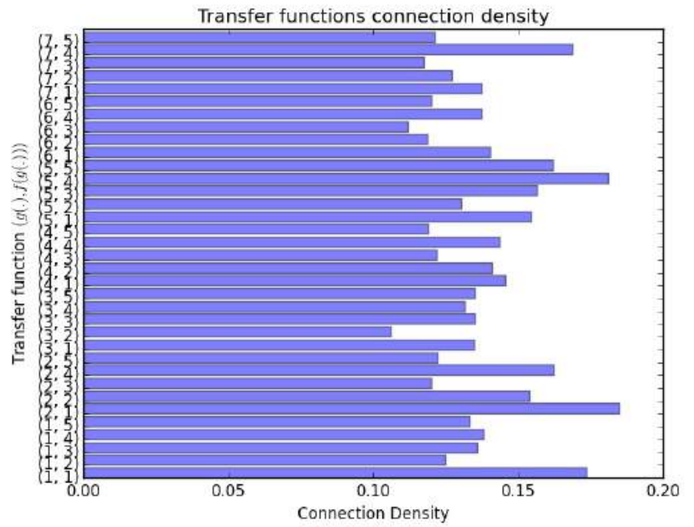
Figure A.20: The average transfer function likelihood after thresholding for the sizes of $N \in \{1..4\}$ on the Iris dataset.



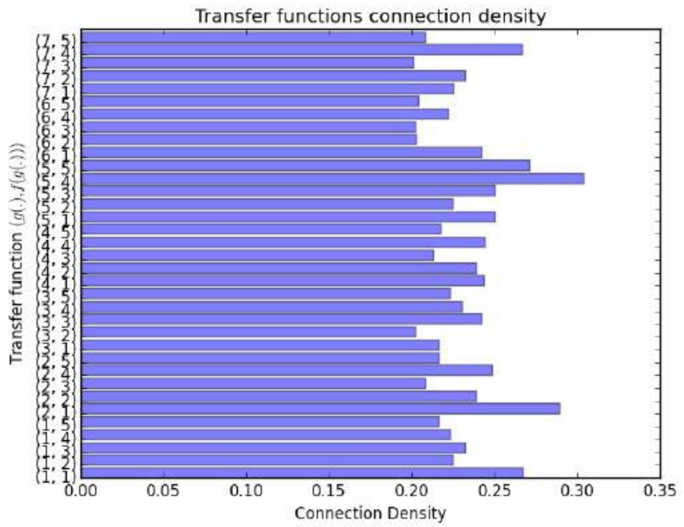
(a) $N = 1$



(b) $N = 2$

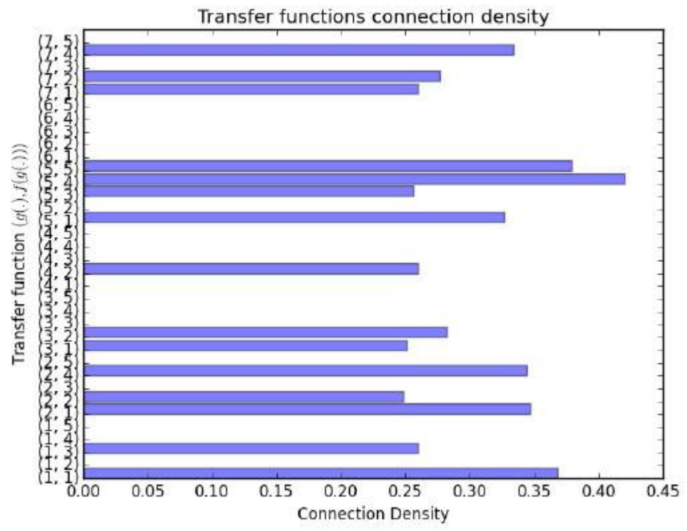


(c) $N = 3$

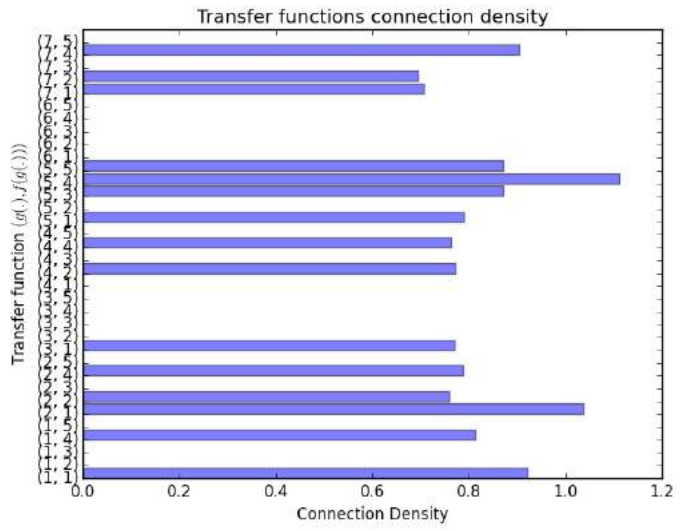


(d) $N = 4$

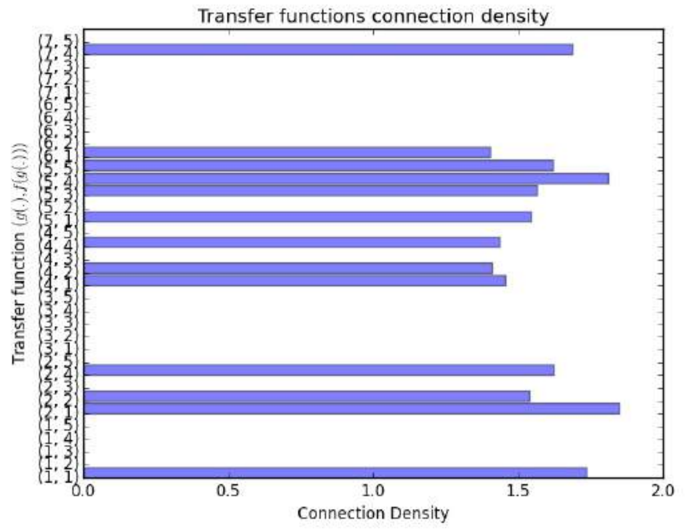
Figure A.21: The average connection density for the sizes of $N \in \{1..4\}$ on the Iris dataset.



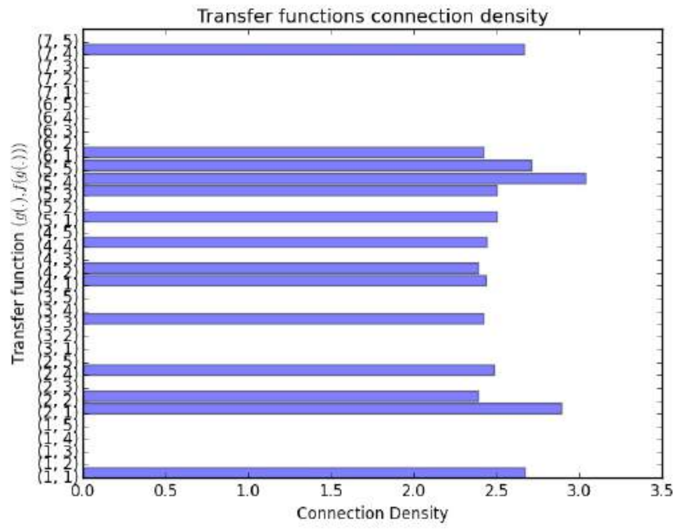
(a) $N = 1$



(b) $N = 2$

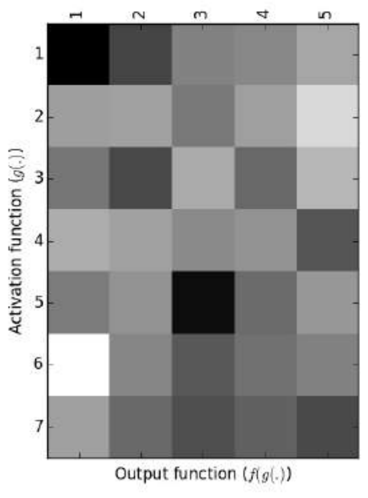


(c) $N = 3$

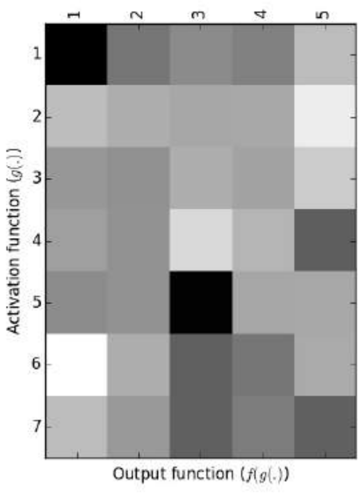


(d) $N = 4$

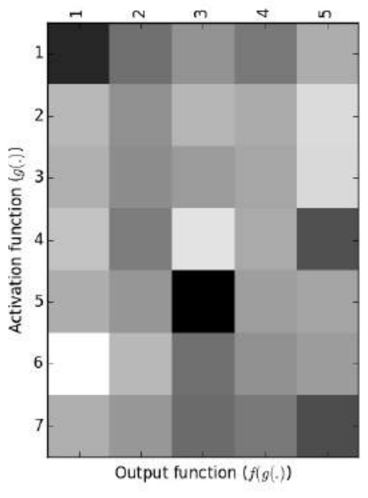
Figure A.22: The average connection density after thresholding for sizes of $N \in \{1..4\}$ on the Iris.



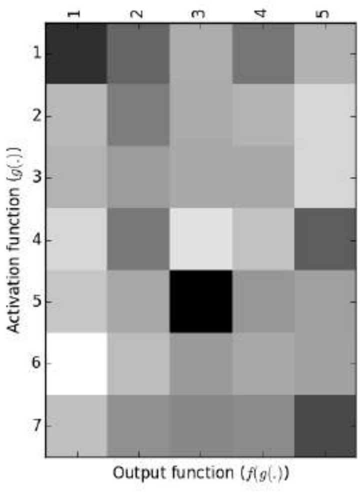
(a) $N = 1$



(b) $N = 2$

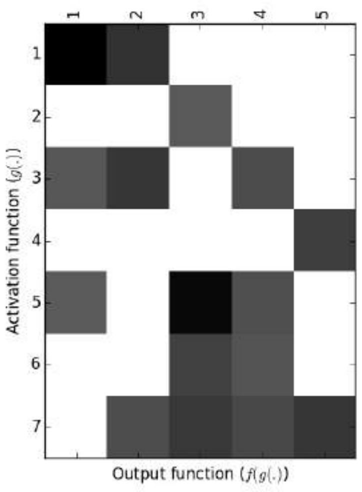


(c) $N = 3$

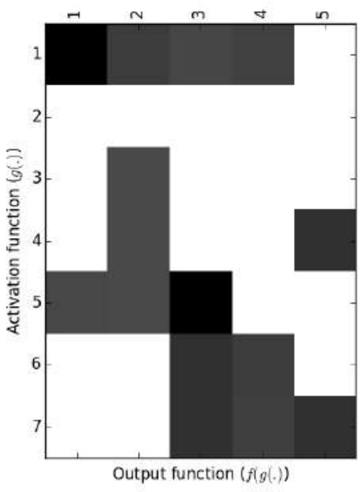


(d) $N = 4$

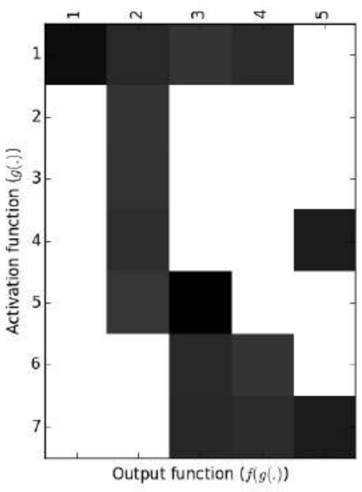
Figure A.23: The average transfer function likelihood for sizes of $N \in \{1..4\}$ on the Sonar dataset.



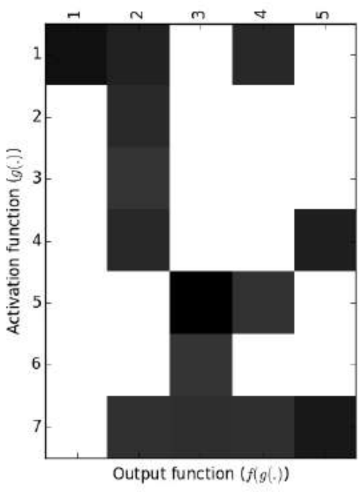
(a) $N = 1$



(b) $N = 2$

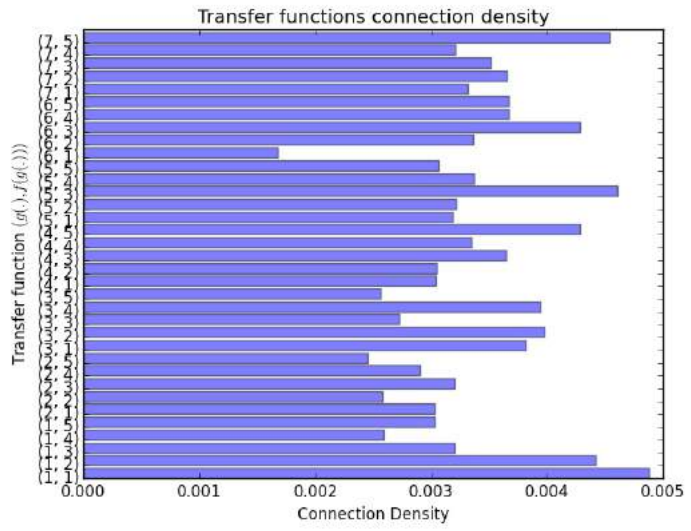


(c) $N = 3$

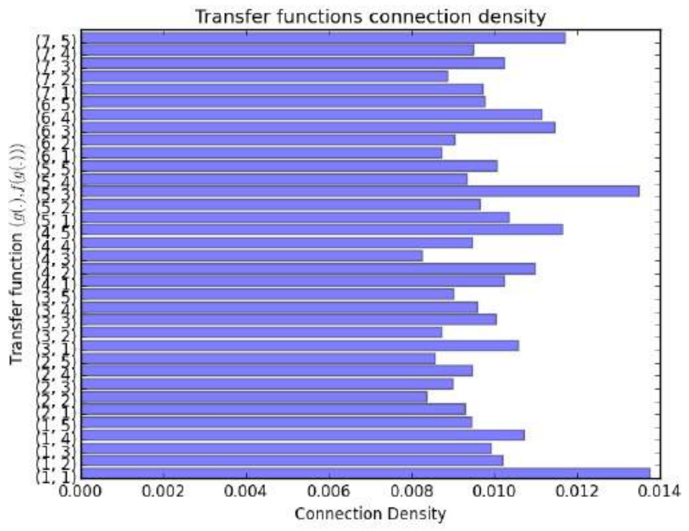


(d) $N = 4$

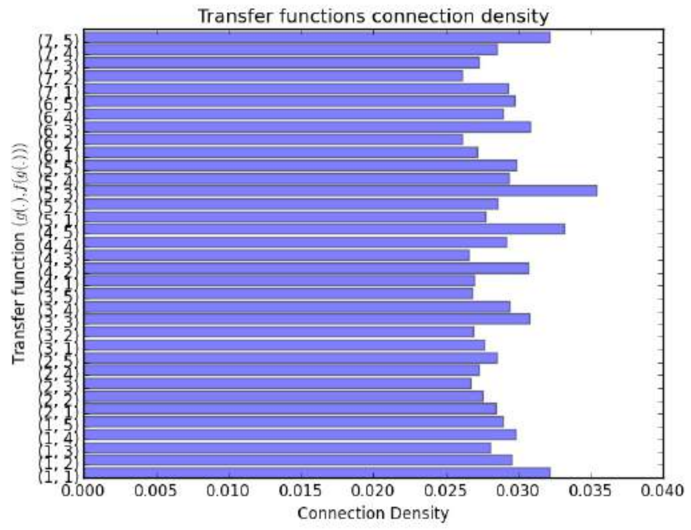
Figure A.24: The average transfer function likelihood after thresholding for sizes $N \in \{1..4\}$ on the Sonar dataset.



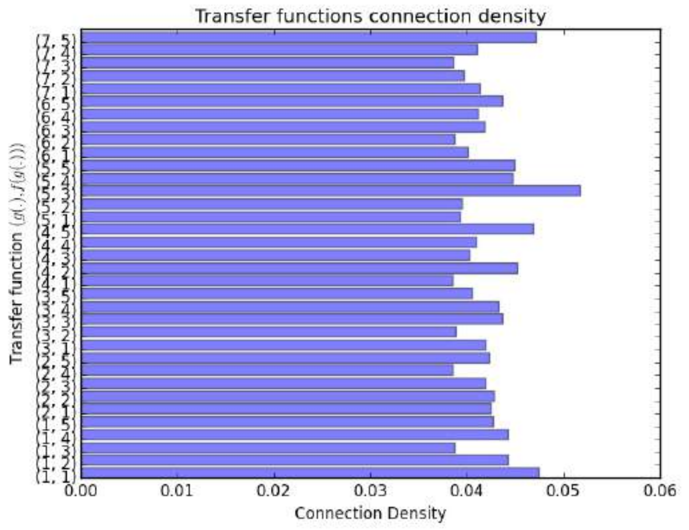
(a) $N = 1$



(b) $N = 2$

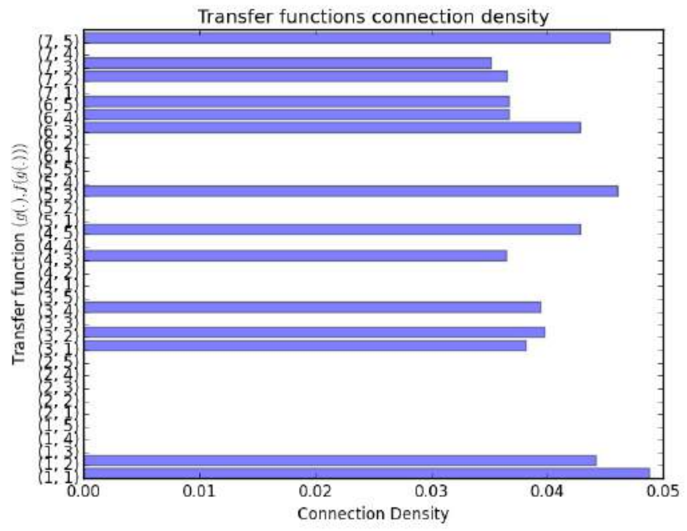


(c) $N = 3$

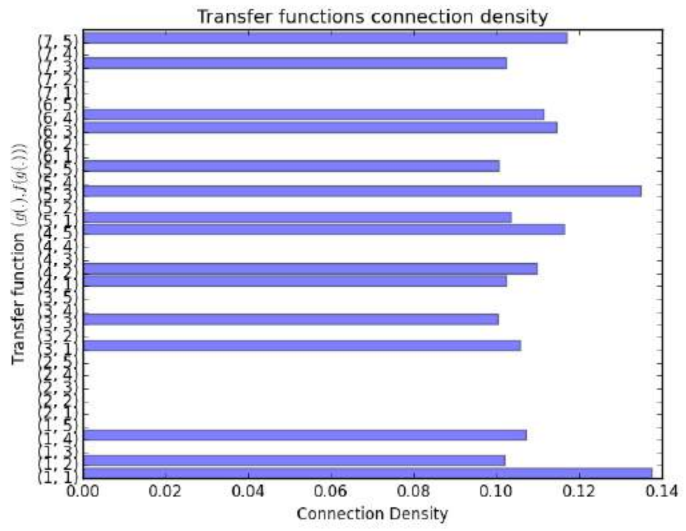


(d) $N = 4$

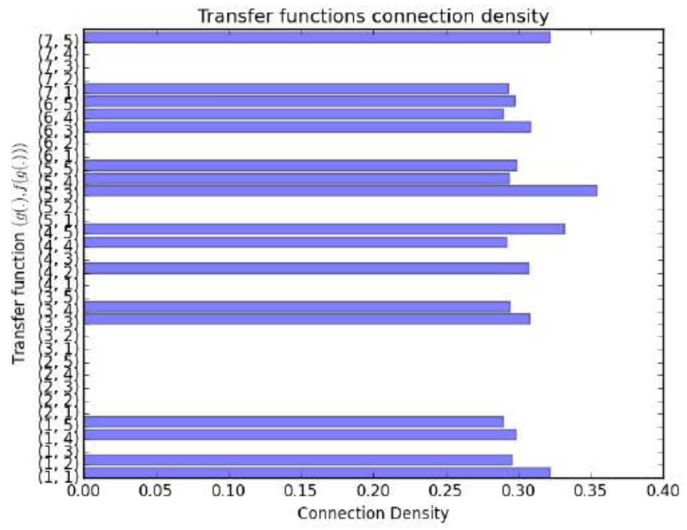
Figure A.25: The average connection density matrices for various sizes of $N \in \{1..4\}$on the Sonar.



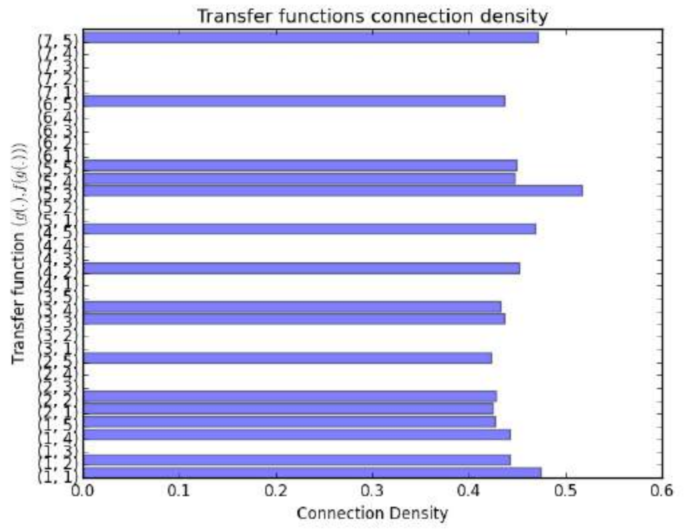
(a) $N = 1$



(b) $N = 2$

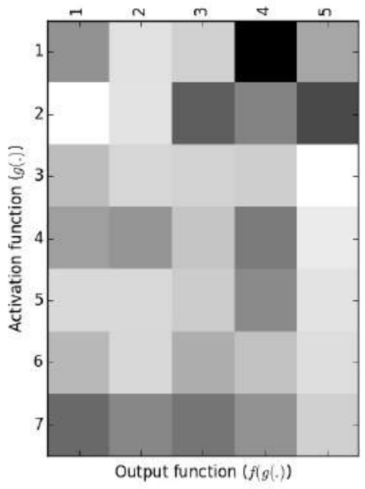


(c) $N = 3$

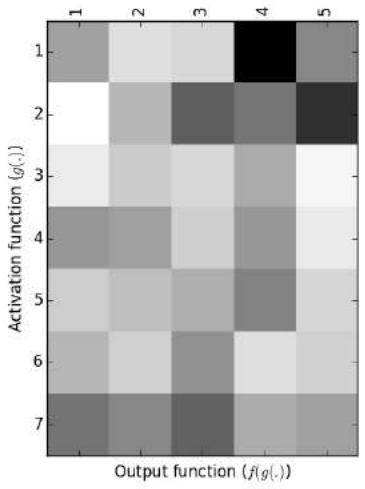


(d) $N = 4$

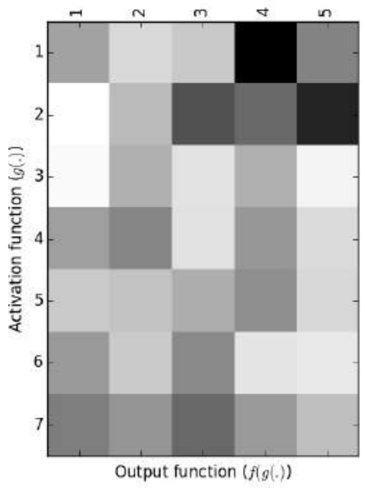
Figure A.26: The average connection density after thresholding for sizes of $N \in \{1..4\}$ on the Sonar dataset.



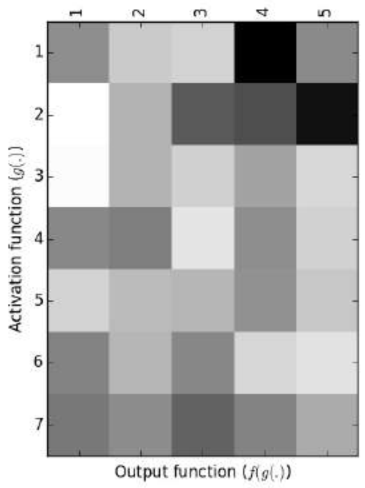
(a) $N = 1$



(b) $N = 2$

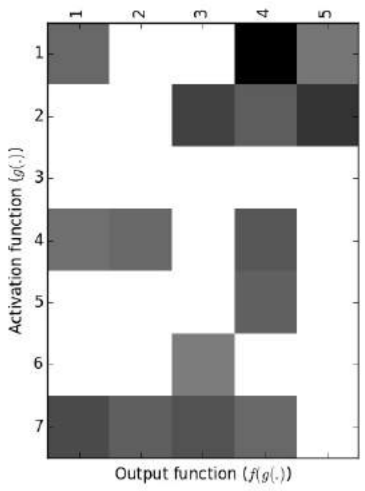


(c) $N = 3$

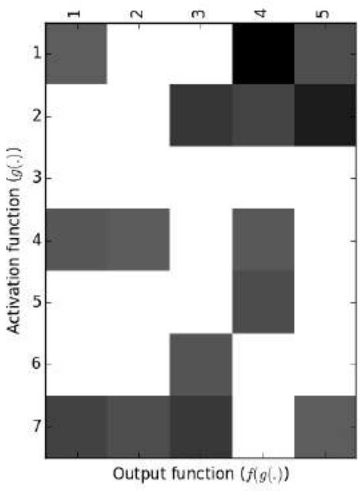


(d) $N = 4$

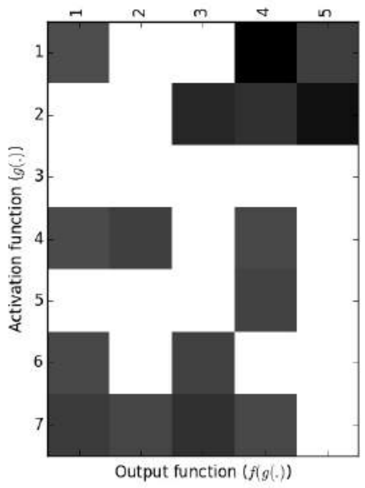
Figure A.27: The average transfer function likelihood for sizes of $N \in \{1..4\}$ on the XOR dataset.



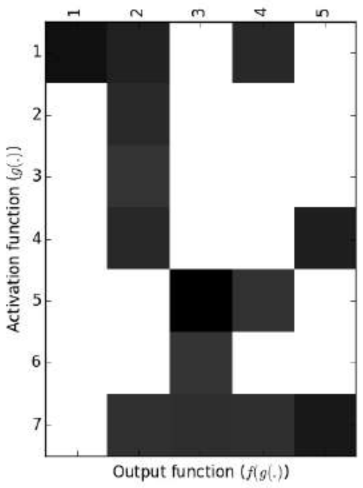
(a) $N = 1$



(b) $N = 2$

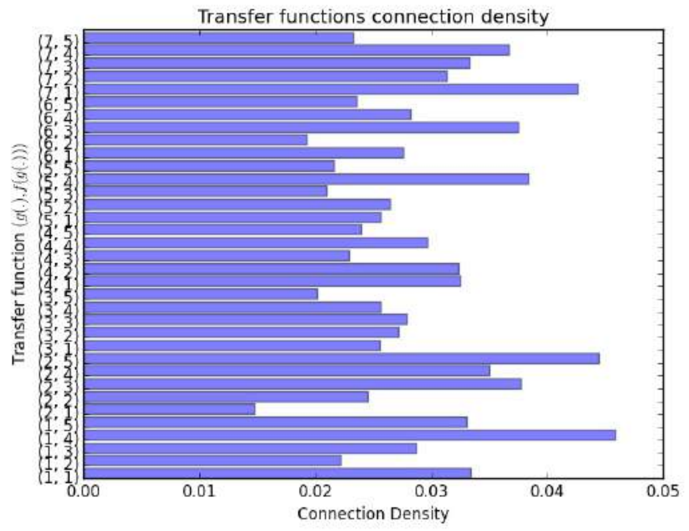


(c) $N = 3$

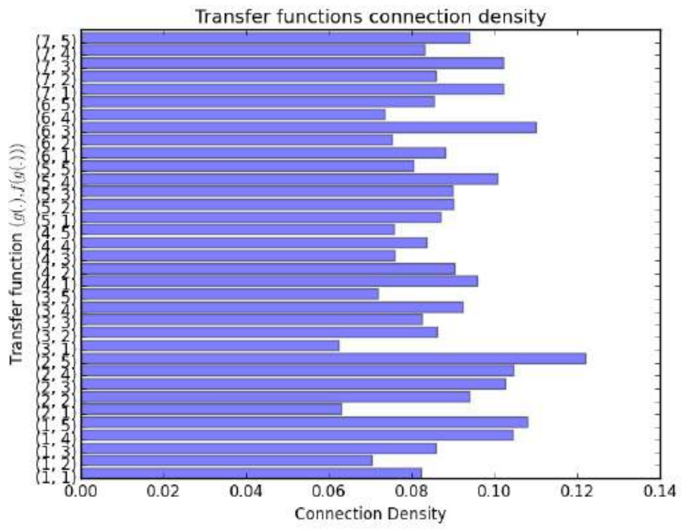


(d) $N = 4$

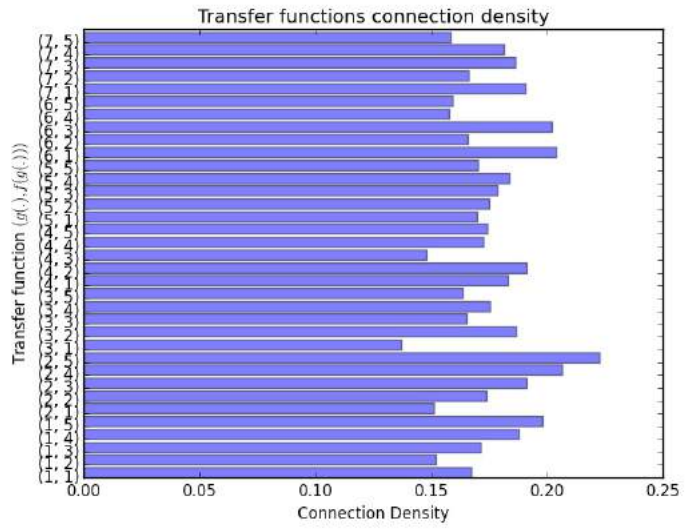
Figure A.28: The average transfer function likelihood for sizes of $N \in \{1..4\}$ on the XOR dataset.



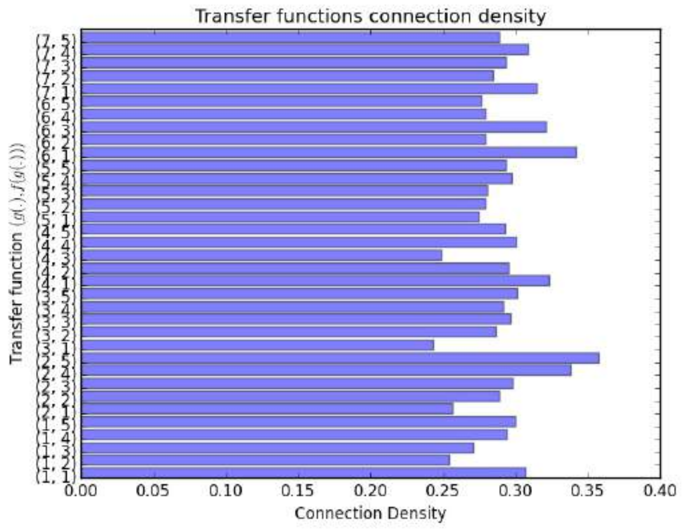
(a) $N = 1$



(b) $N = 2$

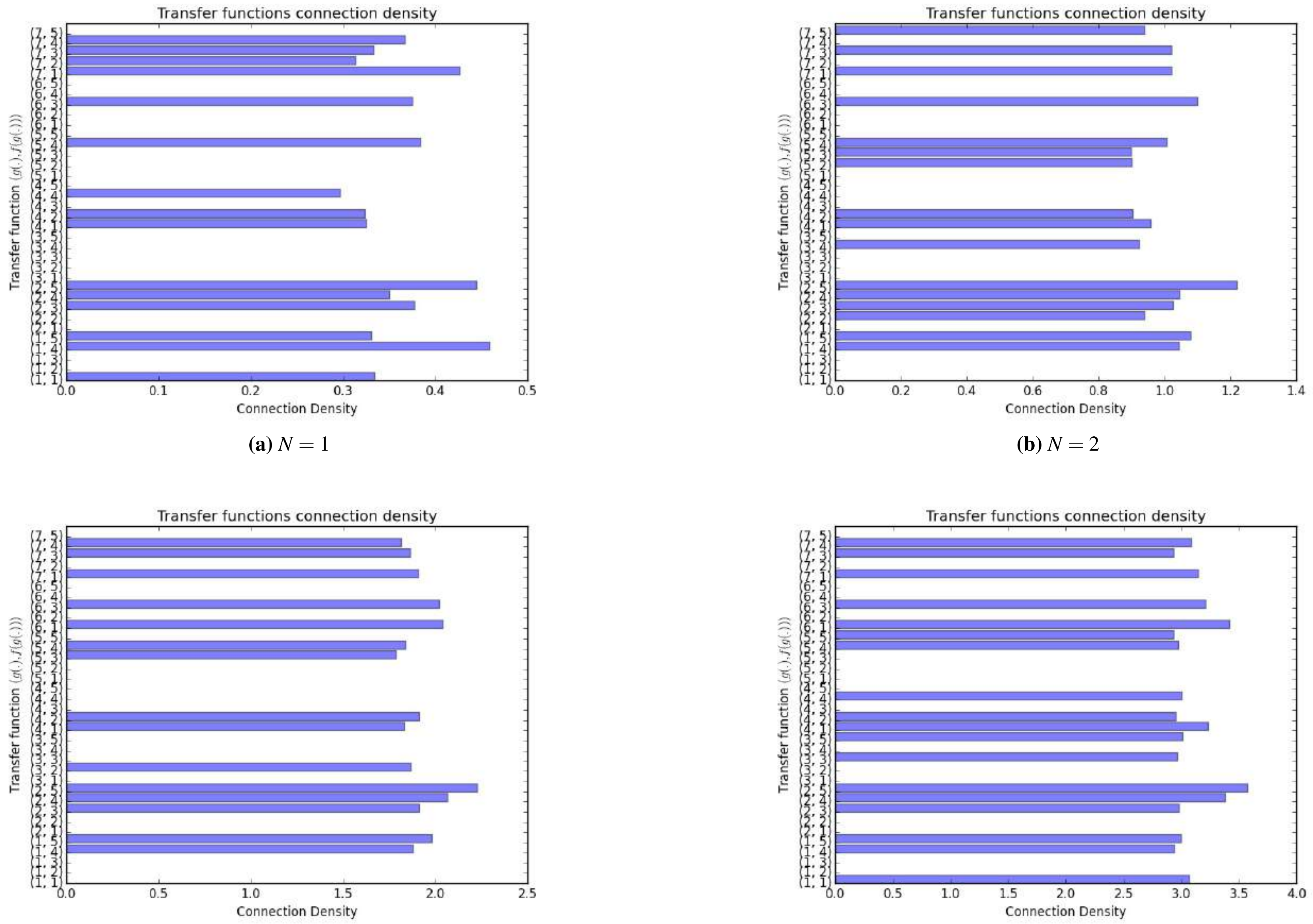


(c) $N = 3$



(d) $N = 4$

Figure A.29: The average connection density for sizes of $N \in \{1..4\}$ on the XOR dataset.



(a) $N = 1$

(b) $N = 2$

(c) $N = 3$

(d) $N = 4$

Figure A.30: The average connection density for various sizes of $N \in \{1..5\}$ on the XOR.

A.2 Consistence Likelihood of Higher-Order Problem Signatures

A.2.1 Coexistence Matrices Heat Map

A.2.1.1 Size Of N

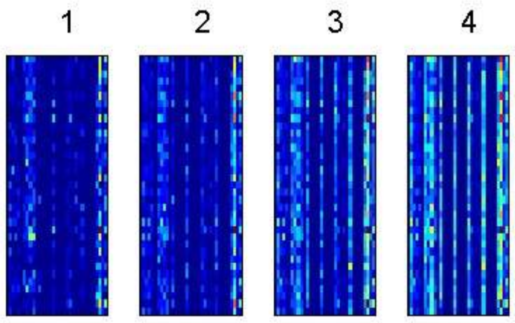


Figure A.31: The heat map of the average coexistence matrix for Iris as the size of N was increased.

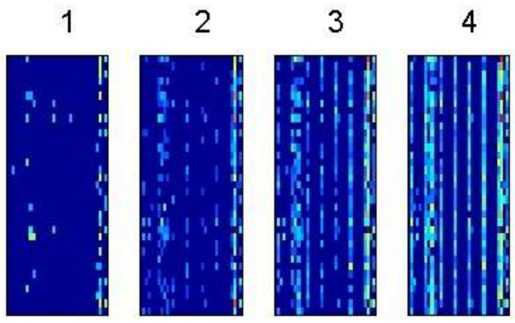


Figure A.32: The heat map of the average coexistence matrix after thresholding for Iris dataset as the size of N was increased.

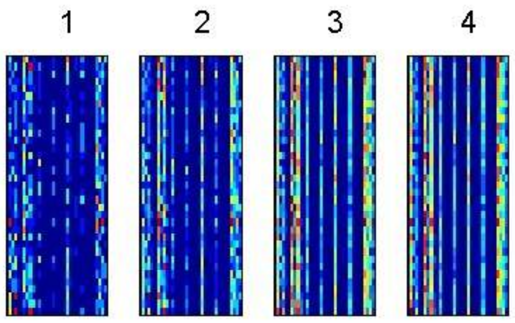


Figure A.33: The heat map of the average coexistence matrix for Sonar dataset as the size of N increases.

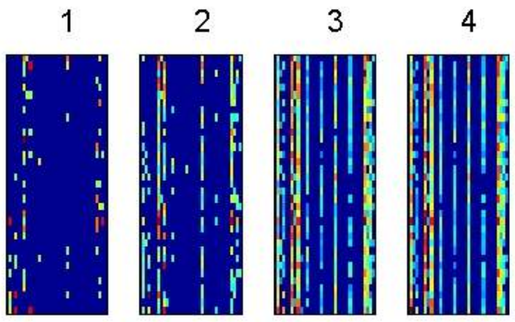


Figure A.34: The heat map of the average coexistence matrix after thresholding for Sonar dataset as the size of N increases.

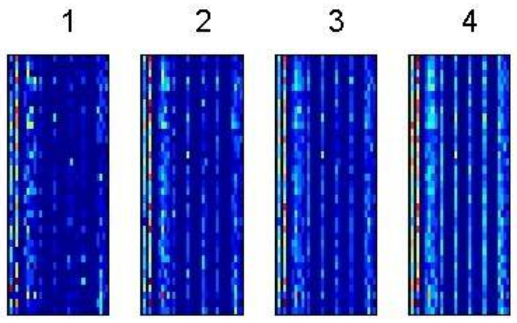


Figure A.35: The heat map of the average coexistence matrix for XOR dataset as the size of N increases.

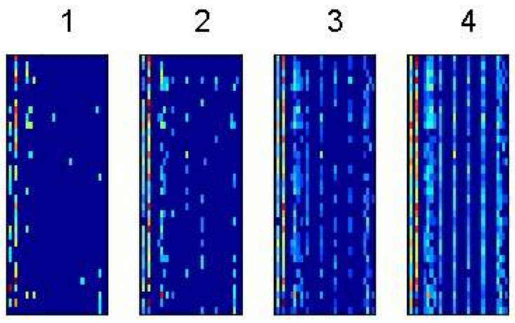
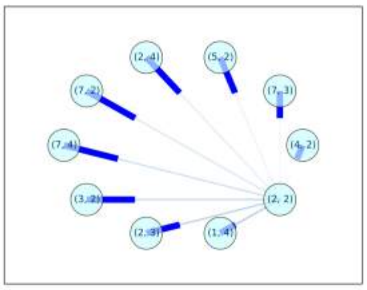
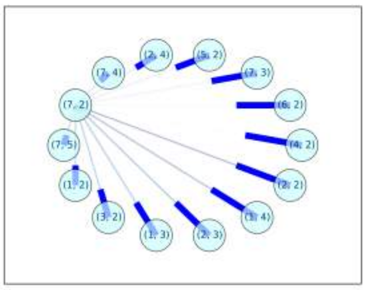


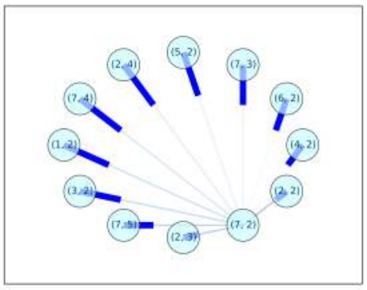
Figure A.36: The heat map of the average coexistence matrix after thresholding for XOR dataset as the size of N increases.



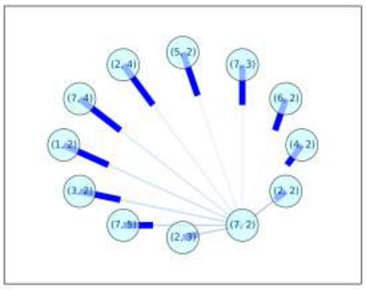
(a) $\gamma = 0.1$



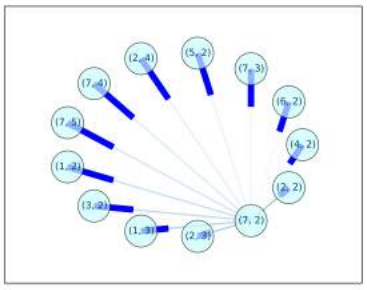
(b) $\gamma = 0.2$



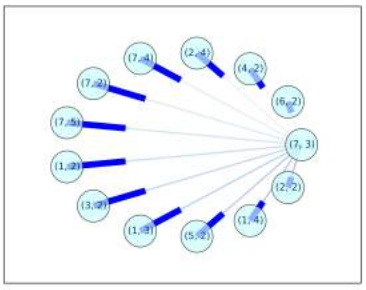
(c) $\gamma = 0.3$



(d) $\gamma = 0.4$



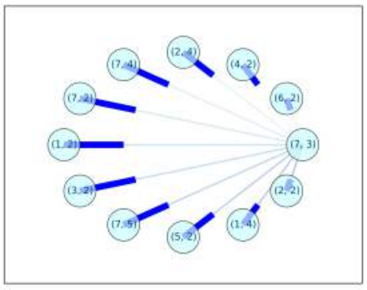
(e) $\gamma = 0.5$



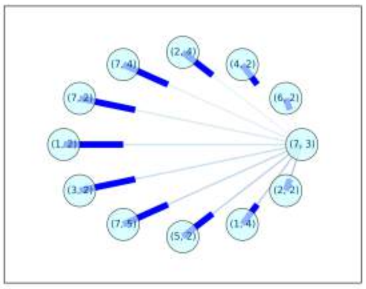
(f) $\gamma = 0.6$

Figure A.37: Path analysis results for the Iris dataset with increasing levels of noise $\gamma = \{0.1..0.6\}$.

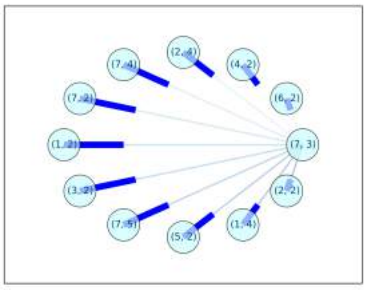
A.2.2 Path Analysis with Increasing level of Noise



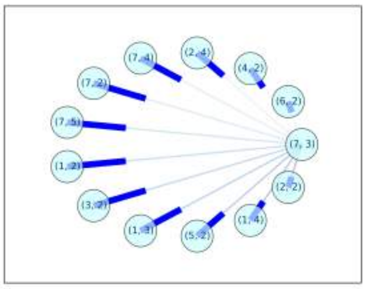
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$

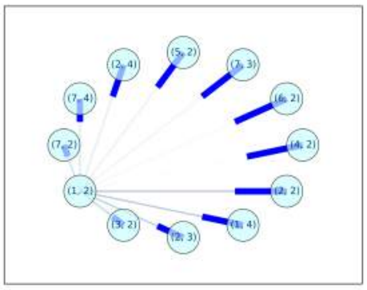


(c) $\gamma = 0.9$

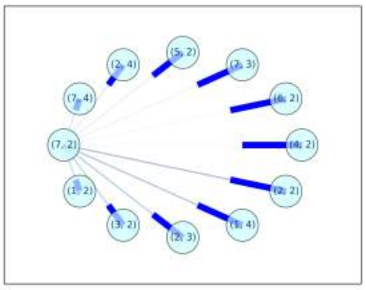


(d) $\gamma = 1.0$

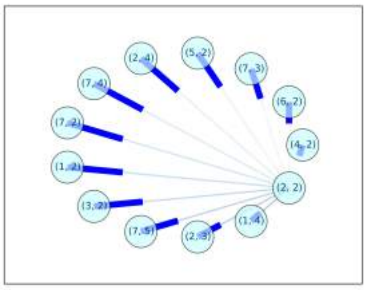
Figure A.38: Path analysis results for the Iris dataset with increasing levels of noise $\gamma = \{0.7..1.0\}$.



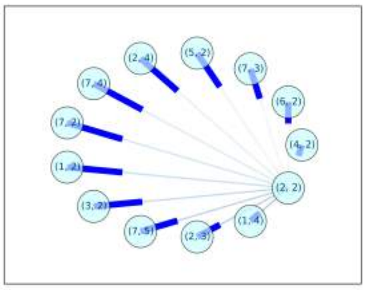
(a) $\gamma = 0.1$



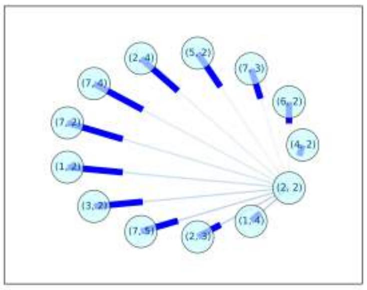
(b) $\gamma = 0.2$



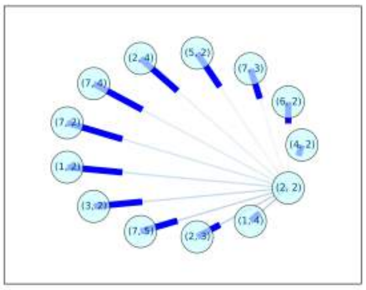
(c) $\gamma = 0.3$



(d) $\gamma = 0.4$

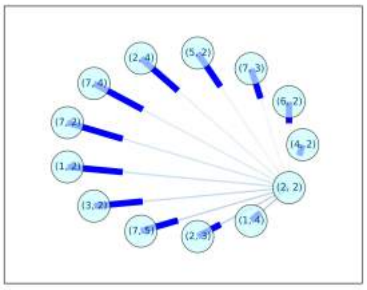


(e) $\gamma = 0.5$

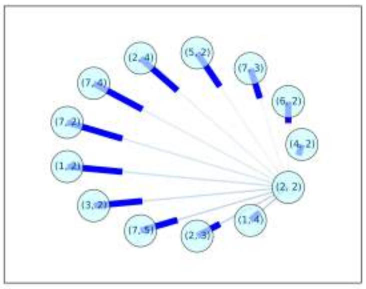


(f) $\gamma = 0.6$

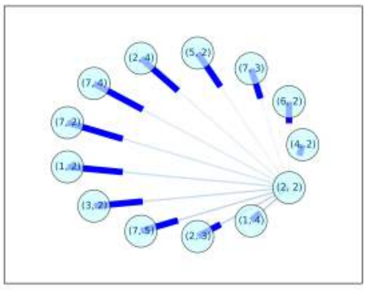
Figure A.39: Path analysis results for the Sonar dataset with increasing levels of noise $\gamma = \{0.1..0.6\}$.



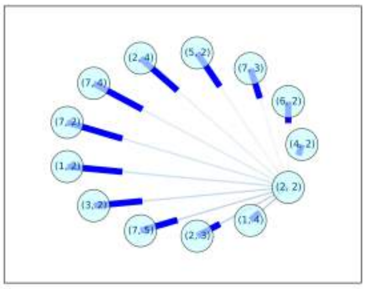
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$

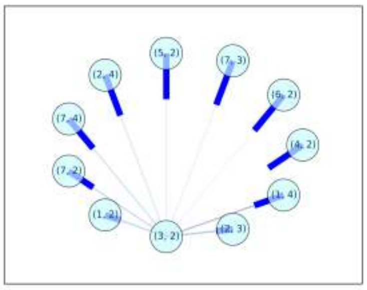


(c) $\gamma = 0.9$

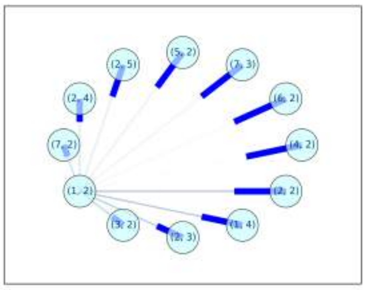


(d) $\gamma = 1.0$

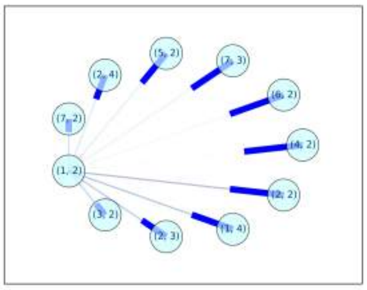
Figure A.40: Path analysis results for the Sonar dataset with increasing levels of noise $\gamma = \{0.7..1.0\}$.



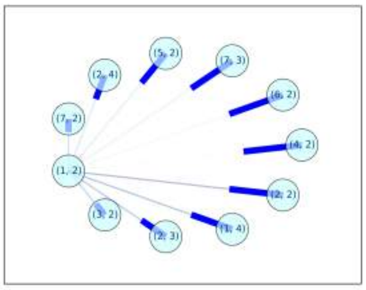
(a) $\gamma = 0.1$



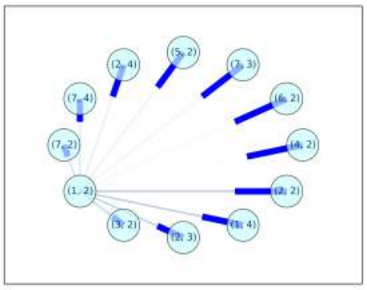
(b) $\gamma = 0.2$



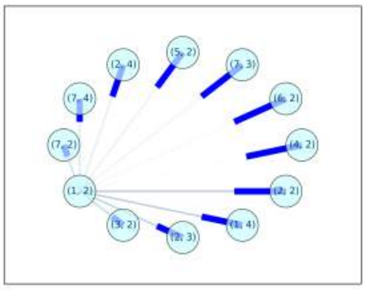
(c) $\gamma = 0.3$



(d) $\gamma = 0.4$

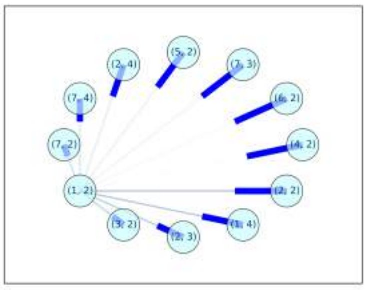


(e) $\gamma = 0.5$

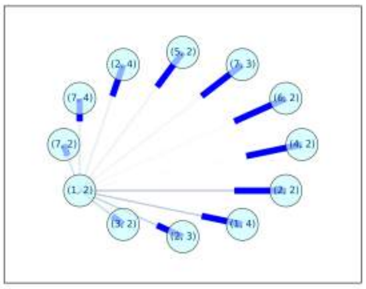


(f) $\gamma = 0.6$

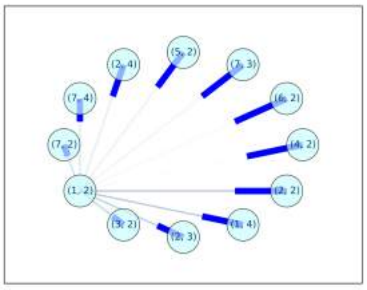
Figure A.41: Path analysis results for the XOR dataset with increasing levels of noise $\gamma = \{0.1..0.6\}$.



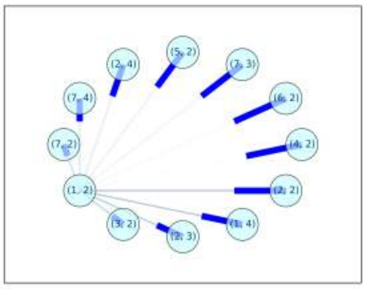
(a) $\gamma = 0.7$



(b) $\gamma = 0.8$



(c) $\gamma = 0.9$



(d) $\gamma = 1.0$

Figure A.42: Path analysis results for the XOR dataset with increasing levels of noise $\gamma = \{0.7..1.0\}$.

A.3 Cooperative Co-evolution of Neurally Diverse Neural Networks

A.3.1 Convergence Graphs

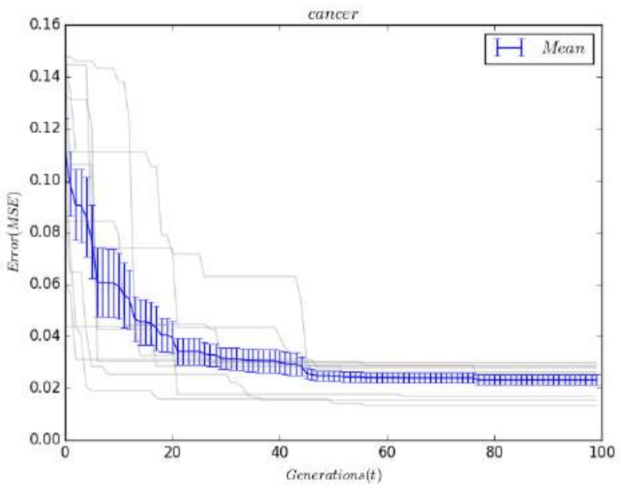


Figure A.43: Cancer convergence graph

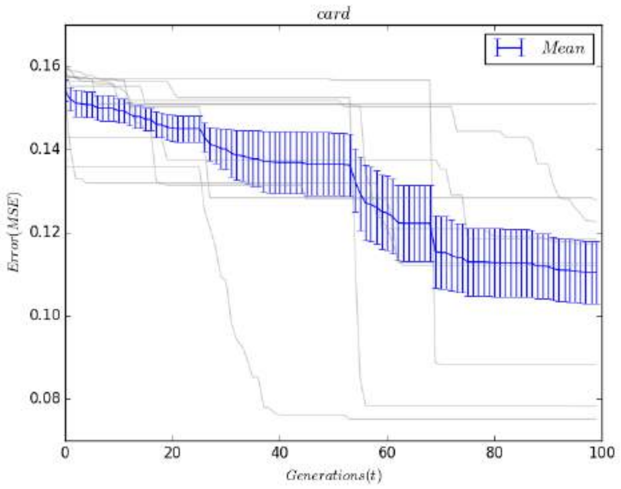


Figure A.44: Card convergence graph

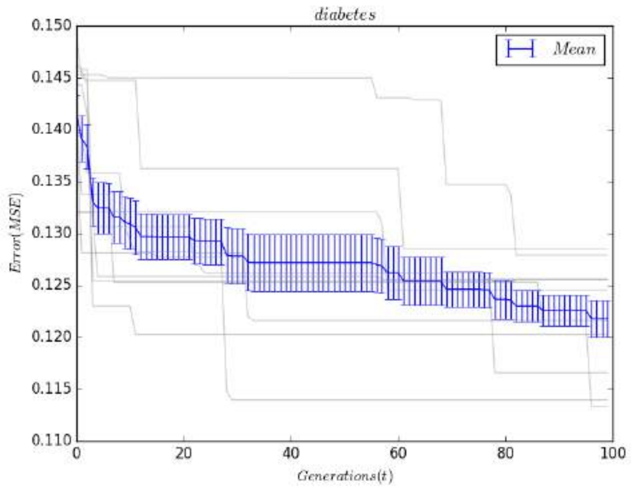


Figure A.45: Diabetes convergence graph

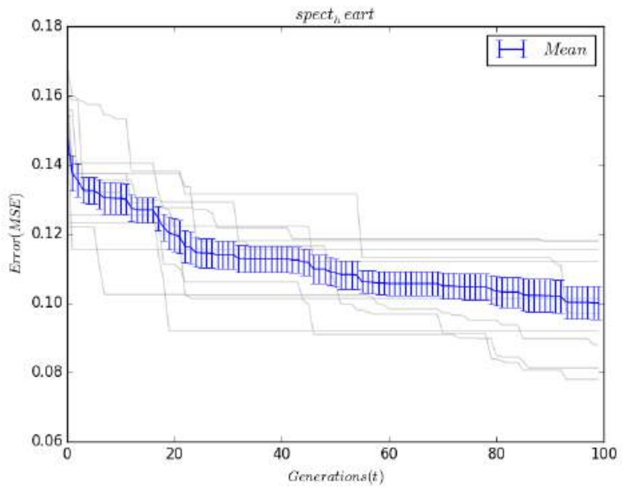


Figure A.46: SPECT Heart convergence graph

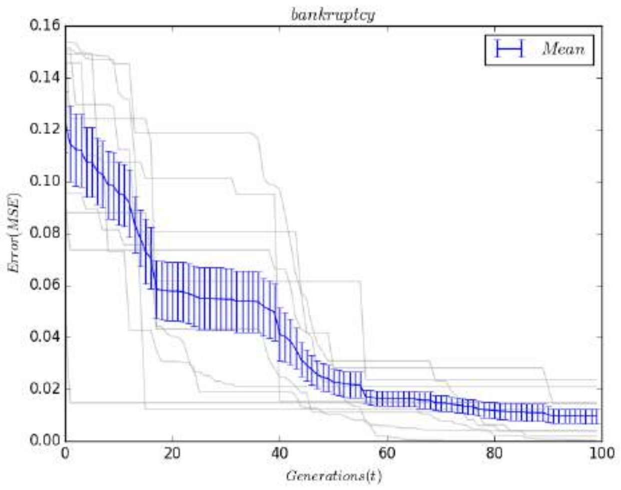


Figure A.47: Bankruptcy convergence graph

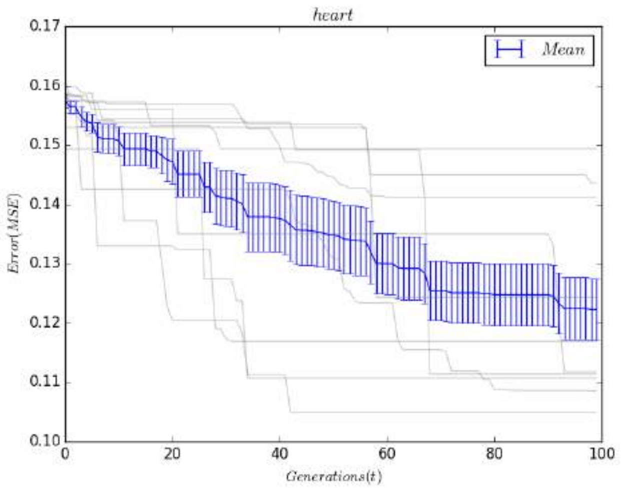


Figure A.48: Heart convergence graph

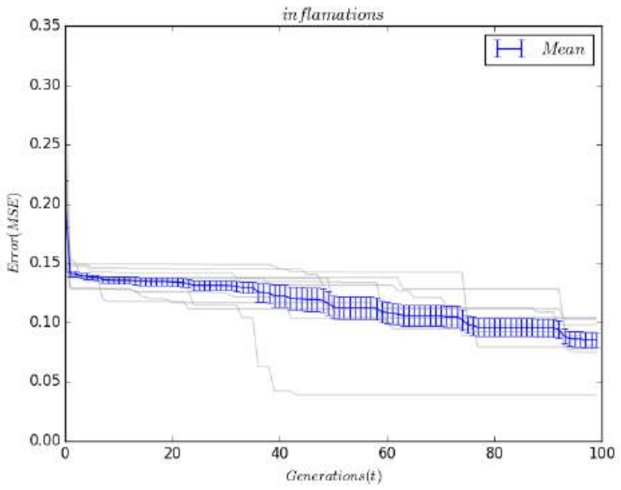


Figure A.49: Inflammations convergence graph

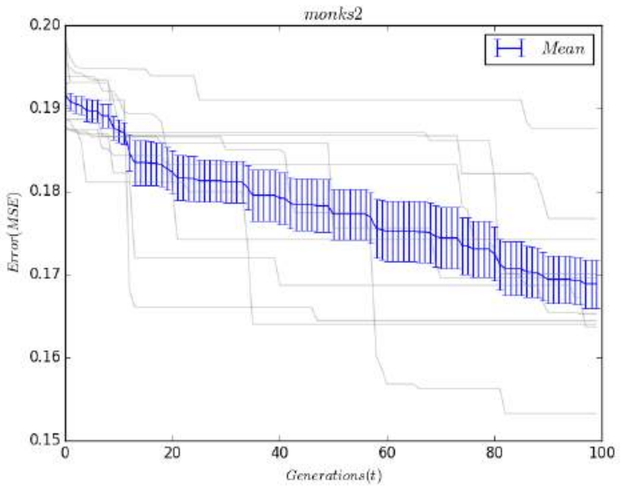


Figure A.50: Monks2 convergence graph

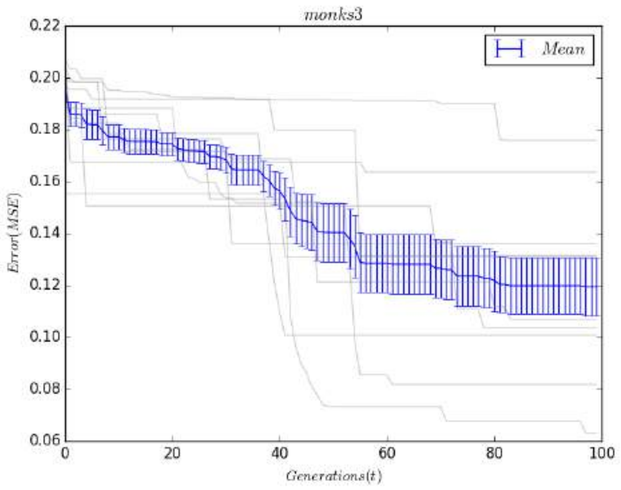


Figure A.51: Monks3 convergence graph

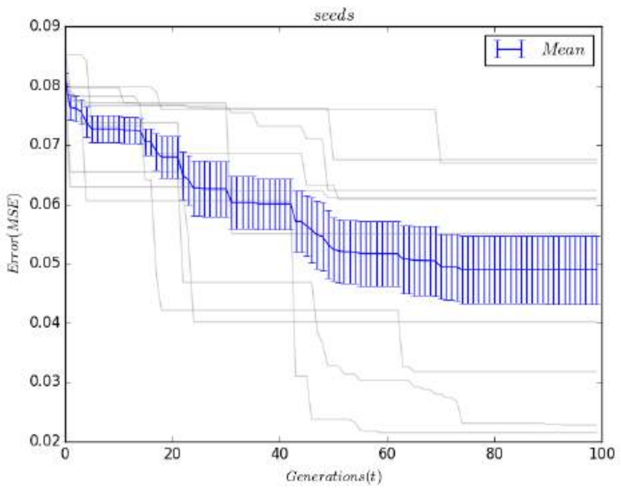


Figure A.52: Seeds convergence graph

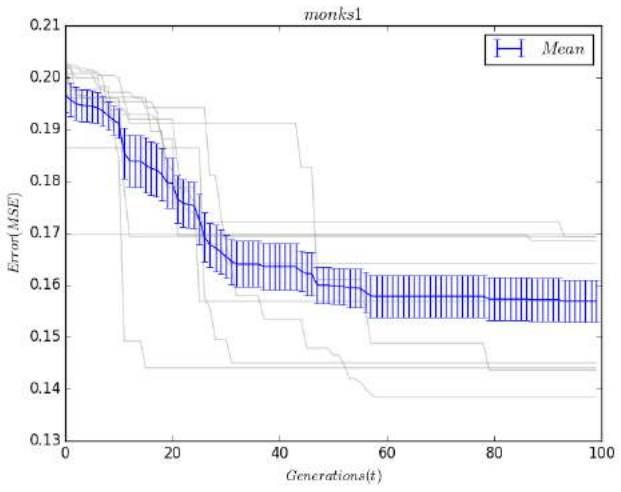


Figure A.53: Monks1 convergence graph

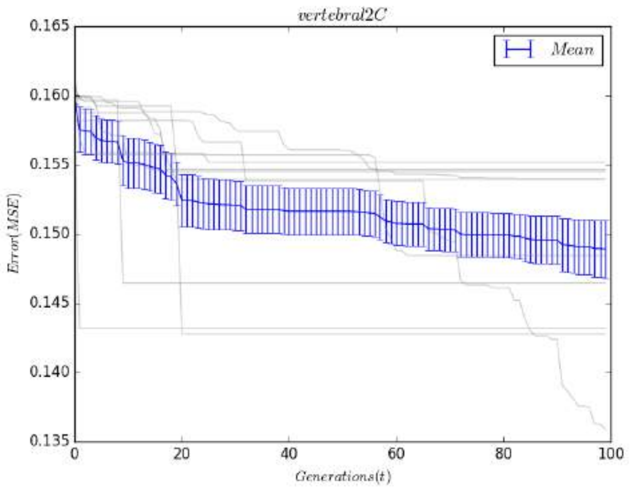


Figure A.54: Vertebral2C convergence graph

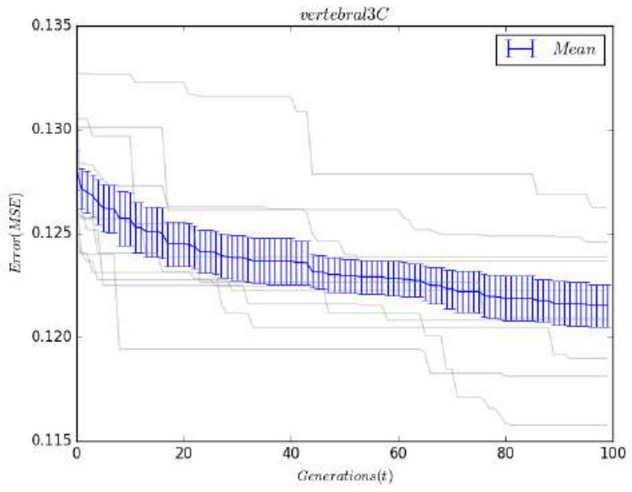


Figure A.55: Vertebral3C convergence graph

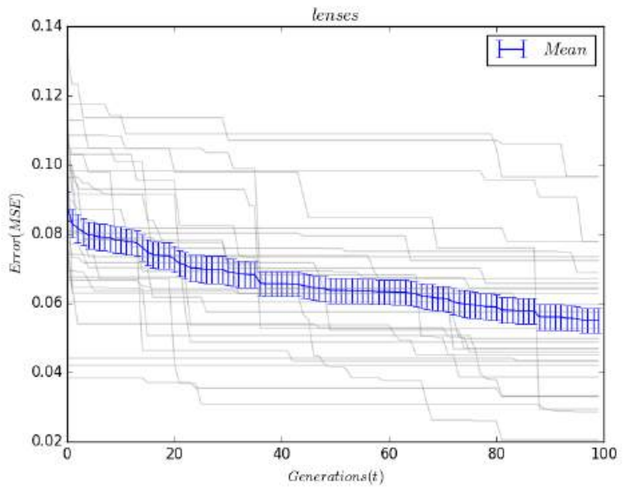


Figure A.56: Lenses convergence graph

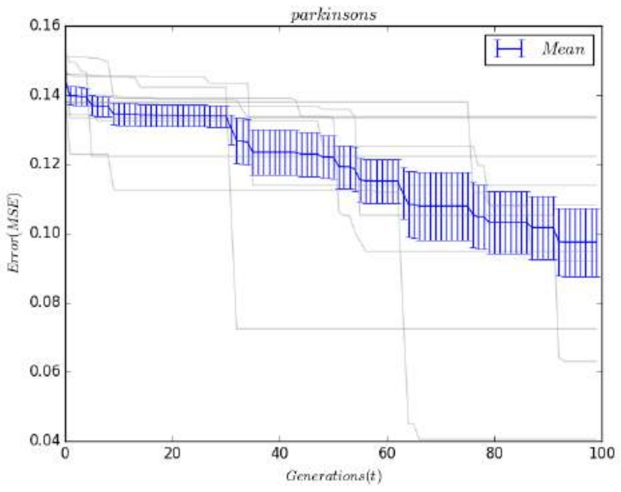


Figure A.57: Parkinsons convergence graph

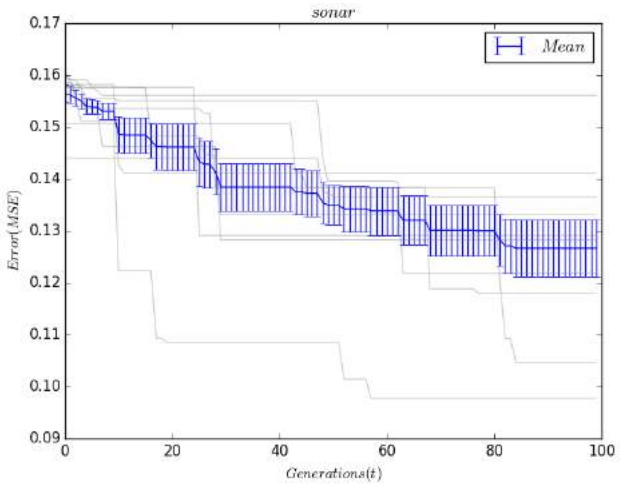


Figure A.58: Sonar convergence graph

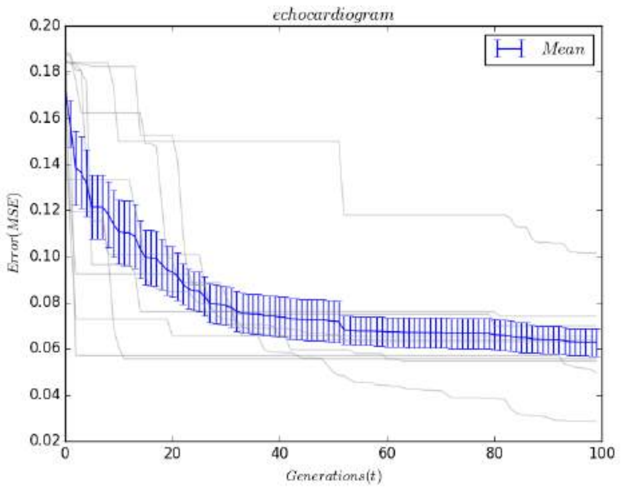


Figure A.59: Echocardiogram convergence graph

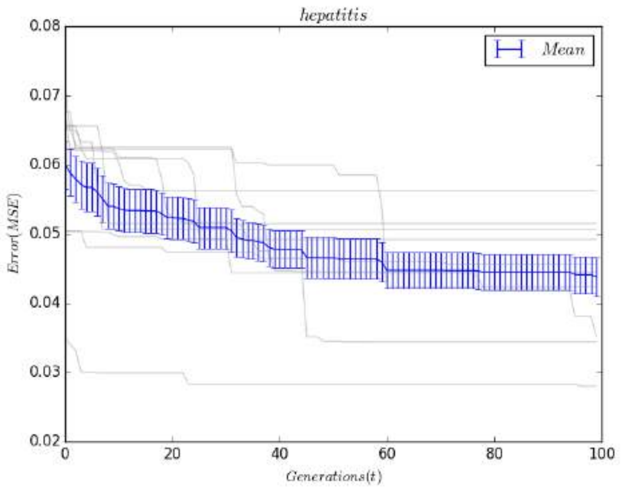


Figure A.60: Hepatitis convergence graph

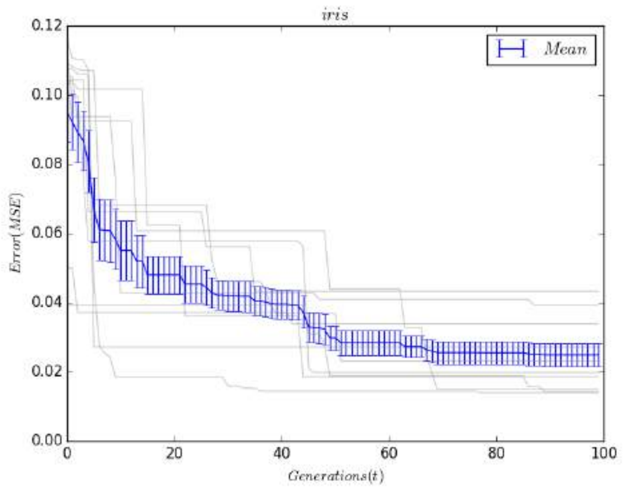


Figure A.61: Iris convergence graph

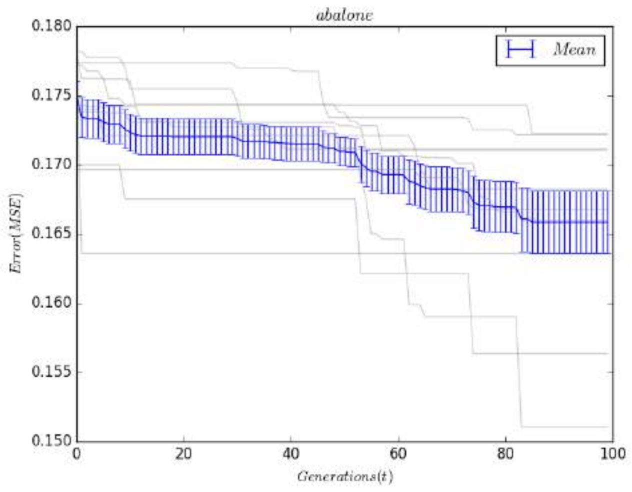


Figure A.62: Abalone convergence graph

Table A.1: T-test results for the Abalone benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.0002056822	0.8914864	0.2366776	0.0006767946	0.000528822	0.001339968
RBF	0.0002056822	1	0.0004501399	0.0004321483	0.9388033	0.68134	0.8726065
MLP-RBF	0.8914864	0.0004501399	1	0.03710981	0.0007559914	0.0006041455	0.001456839
CoevoNDM	0.2366776	0.0004321483	0.03710981	1	0.0004883692	0.0003768437	0.0009454741
SVM-Linear	0.0006767946	0.9388033	0.0007559914	0.0004883692	1	0.7983382	0.9423953
SVM-RBF	0.000528822	0.68134	0.0006041455	0.0003768437	0.7983382	1	0.8655783
SVM-POLY	0.001339968	0.8726065	0.001456839	0.0009454741	0.9423953	0.8655783	1

Table A.2: T-test results for the Bankruptcy benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.04544329	0.002518847	0.9260922	1.361746e-05	7.636066e-07	4.734567e-06
RBF	0.04544329	1	0.8694421	0.0464172	3.175409e-05	1.528435e-06	9.217199e-06
MLP-RBF	0.002518847	0.8694421	1	0.002668012	3.751991e-05	1.158772e-06	1.101121e-05
CoevoNDM	0.9260922	0.0464172	0.002668012	1	1.491298e-05	9.708541e-07	5.394558e-06
SVM-Linear	1.361746e-05	3.175409e-05	3.751991e-05	1.491298e-05	1	0.886083	1
SVM-RBF	7.636066e-07	1.528435e-06	1.158772e-06	9.708541e-07	0.886083	1	0.8770485
SVM-POLY	4.734567e-06	9.217199e-06	1.101121e-05	5.394558e-06	1	0.8770485	1

Table A.3: T-test results for the cancer benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.0001317983	8.117567e-06	0.7966754	1.96952e-11	2.529631e-14	6.52708e-12
RBF	0.0001317983	1	0.001447112	0.0001392162	0.007801099	0.007660817	0.006943597
MLP-RBF	8.117567e-06	0.001447112	1	1.768094e-05	2.915494e-13	1.588914e-15	6.354396e-14
CoevoNDM	0.7966754	0.0001392162	1.768094e-05	1	1.112995e-10	1.982631e-12	5.179853e-11
SVM-Linear	1.96952e-11	0.007801099	2.915494e-13	1.112995e-10	1	1	0.8761552
SVM-RBF	2.529631e-14	0.007660817	1.588914e-15	1.982631e-12	1	1	0.8534691
SVM-POLY	6.52708e-12	0.006943597	6.354396e-14	5.179853e-11	0.8761552	0.8534691	1

Table A.4: T-test results for the Card benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	1.553302e-05	2.723445e-08	0.001062494	1.667411e-09	1.696949e-06	5.668771e-10
RBF	1.553302e-05	1	0.0002389431	4.592306e-05	0.8877294	9.472398e-06	0.8865734
MLP-RBF	2.723445e-08	0.0002389431	1	0.0001881614	2.278229e-07	3.281742e-06	1.1543e-07
CoevoNDM	0.001062494	4.592306e-05	0.0001881614	1	3.285418e-09	2.221534e-06	1.040351e-09
SVM-Linear	1.667411e-09	0.8877294	2.278229e-07	3.285418e-09	1	1.773419e-05	1
SVM-RBF	1.696949e-06	9.472398e-06	3.281742e-06	2.221534e-06	1.773419e-05	1	1.813716e-05
SVM-POLY	5.668771e-10	0.8865734	1.1543e-07	1.040351e-09	1	1.813716e-05	1

Table A.5: T-test results for the Diabetes benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.0002899248	2.240079e-05	0.04031699	7.882946e-13	1.864991e-11	3.712888e-08
RBF	0.0002899248	1	0.001083412	0.0004595944	0.3672978	0.3437817	0.3557652
MLP-RBF	2.240079e-05	0.001083412	1	1.580606e-08	3.080604e-09	1.808054e-08	1.423161e-06
CoevoNDM	0.04031699	0.0004595944	1.580606e-08	1	5.68864e-10	3.625965e-09	3.481459e-07
SVM-Linear	7.882946e-13	0.3672978	3.080604e-09	5.68864e-10	1	0.8507944	0.88917
SVM-RBF	1.864991e-11	0.3437817	1.808054e-08	3.625965e-09	0.8507944	1	1
SVM-POLY	3.712888e-08	0.3557652	1.423161e-06	3.481459e-07	0.88917	1	1

Table A.6: T-test results for the Echocardiogram benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.004595709	0.1907847	0.03106768	0.005865716	0.02711998	0.03803183
RBF	0.004595709	1	0.01478126	0.0004239918	0.287637	0.468388	0.3725348
MLP-RBF	0.1907847	0.01478126	1	1.034458e-07	0.01697179	0.08116817	0.1132933
CoevoNDM	0.03106768	0.0004239918	1.034458e-07	1	5.566491e-05	0.001742653	0.002240105
SVM-Linear	0.005865716	0.287637	0.01697179	5.566491e-05	1	0.8414386	1
SVM-RBF	0.02711998	0.468388	0.08116817	0.001742653	0.8414386	1	0.8680718
SVM-POLY	0.03803183	0.3725348	0.1132933	0.002240105	1	0.8680718	1

Table A.7: T-test results for the Heart (Pima) benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	1.874793e-05	0.0002633357	0.00599868	2.354578e-15	8.823018e-09	6.517217e-11
RBF	1.874793e-05	1	8.748802e-05	5.496341e-05	0.6003074	0.6291987	0.6154511
MLP-RBF	0.0002633357	8.748802e-05	1	0.07481398	5.036916e-15	1.601738e-07	2.979248e-09
CoevoNDM	0.00599868	5.496341e-05	0.07481398	1	1.713644e-15	1.042673e-07	2.246103e-09
SVM-Linear	2.354578e-15	0.6003074	5.036916e-15	1.713644e-15	1	1	1
SVM-RBF	8.823018e-09	0.6291987	1.601738e-07	1.042673e-07	1	1	1
SVM-POLY	6.517217e-11	0.6154511	2.979248e-09	2.246103e-09	1	1	1

Table A.8: T-test results for the Hepatitis benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.7365651	0.784928	0.007561638	9.706807e-11	2.372696e-11	2.372696e-11
RBF	0.7365651	1	0.6072423	0.1946429	6.381772e-07	1.137413e-06	1.137413e-06
MLP-RBF	0.784928	0.6072423	1	0.002401407	4.679518e-11	4.294501e-12	4.294501e-12
CoevoNDM	0.007561638	0.1946429	0.002401407	1	7.487404e-08	3.257779e-09	3.257779e-09
SVM-Linear	9.706807e-11	6.381772e-07	4.679518e-11	7.487404e-08	1	1	1
SVM-RBF	2.372696e-11	1.137413e-06	4.294501e-12	3.257779e-09	1	1	1
SVM-POLY	2.372696e-11	1.137413e-06	4.294501e-12	3.257779e-09	1	1	1

Table A.9: T-test results for the Ionosphere benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.001486834	0.08411555	0.06966857	1.761529e-07	8.965801e-09	2.948767e-06
RBF	0.001486834	1	0.006090493	0.0006157887	0.2164861	0.1758817	0.2394269
MLP-RBF	0.08411555	0.006090493	1	0.001904553	1.045035e-06	8.073483e-08	1.318842e-05
CoevoNDM	0.06966857	0.0006157887	0.001904553	1	3.355383e-07	3.375491e-08	2.969223e-06
SVM-Linear	1.761529e-07	0.2164861	1.045035e-06	3.355383e-07	1	0.889036	1
SVM-RBF	8.965801e-09	0.1758817	8.073483e-08	3.375491e-08	0.889036	1	0.9031828
SVM-POLY	2.948767e-06	0.2394269	1.318842e-05	2.969223e-06	1	0.9031828	1

Table A.10: T-test results for the Iris benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	6.768153e-07	0.003279905	0.4521005	8.983722e-07	1.141578e-06	6.338711e-11
RBF	6.768153e-07	1	1.268056e-05	2.433718e-06	0.3003396	0.3055768	0.1866094
MLP-RBF	0.003279905	1.268056e-05	1	6.156354e-05	1.024052e-05	1.246593e-05	2.391451e-09
CoevoNDM	0.4521005	2.433718e-06	6.156354e-05	1	2.605141e-06	3.074689e-06	7.274236e-09
SVM-Linear	8.983722e-07	0.3003396	1.024052e-05	2.605141e-06	1	1	1
SVM-RBF	1.141578e-06	0.3055768	1.246593e-05	3.074689e-06	1	1	1
SVM-POLY	6.338711e-11	0.1866094	2.391451e-09	7.274236e-09	1	1	1

Table A.11: T-test results for the Lenses benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.3862527	0.4926787	0.704312	0.2007325	0.05208924	0.09612202
RBF	0.3862527	1	0.7042906	0.4715661	0.3474875	0.2391503	0.2157612
MLP-RBF	0.4926787	0.7042906	1	0.54486	0.7734923	0.9589889	0.7576975
CoevoNDM	0.704312	0.4715661	0.54486	1	0.2378151	0.006600261	0.1287478
SVM-Linear	0.2007325	0.3474875	0.7734923	0.2378151	1	0.6841457	1
SVM-RBF	0.05208924	0.2391503	0.9589889	0.006600261	0.6841457	1	0.562282
SVM-POLY	0.09612202	0.2157612	0.7576975	0.1287478	1	0.562282	1

Table A.12: T-test results for the Lung Cancer benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.09416915	0.2836749	0.4330668	0.0540785	0.0269547	0.003151137
RBF	0.09416915	1	0.1220895	0.1096762	1	1	1
MLP-RBF	0.2836749	0.1220895	1	0.4628004	0.06761873	0.0310778	0.005468116
CoevoNDM	0.4330668	0.1096762	0.4628004	1	0.06760089	0.03901552	0.009870658
SVM-Linear	0.0540785	1	0.06761873	0.06760089	1	1	1
SVM-RBF	0.0269547	1	0.0310778	0.03901552	1	1	1
SVM-POLY	0.003151137	1	0.005468116	0.009870658	1	1	1

Table A.13: T-test results for the Monks1 benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.0001641597	0.01608314	0.1017812	2.83793e-08	2.039724e-06	0.0001484066
RBF	0.0001641597	1	0.0006774614	0.0004342213	0.1672986	0.153163	0.1897629
MLP-RBF	0.01608314	0.0006774614	1	0.2076689	1.777814e-06	5.214753e-05	0.001132667
CoevoNDM	0.1017812	0.0004342213	0.2076689	1	2.67865e-07	1.709159e-05	0.0005957815
SVM-Linear	2.83793e-08	0.1672986	1.777814e-06	2.67865e-07	1	0.8452746	0.8843276
SVM-RBF	2.039724e-06	0.153163	5.214753e-05	1.709159e-05	0.8452746	1	1
SVM-POLY	0.0001484066	0.1897629	0.001132667	0.0005957815	0.8843276	1	1

Table A.14: T-test results for the Monks2 benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.005076991	0.3087927	0.04099916	0.01376341	0.000691499	0.0006073902
RBF	0.005076991	1	0.006872563	0.003066997	0.4287188	0.4697917	0.3133528
MLP-RBF	0.3087927	0.006872563	1	0.001743415	0.01981915	0.001153669	0.001096996
CoevoNDM	0.04099916	0.003066997	0.001743415	1	0.007316752	0.0003480613	0.0002831499
SVM-Linear	0.01376341	0.4287188	0.01981915	0.007316752	1	0.8395988	0.916411
SVM-RBF	0.000691499	0.4697917	0.001153669	0.0003480613	0.8395988	1	0.6801364
SVM-POLY	0.0006073902	0.3133528	0.001096996	0.0002831499	0.916411	0.6801364	1

Table A.15: T-test results for the Monks3 benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	4.537459e-05	0.0004594155	0.2347399	0.0003677095	0.0001559052	7.041752e-06
RBF	4.537459e-05	1	0.001216695	1.896791e-05	0.5123163	0.4828827	0.1941625
MLP-RBF	0.0004594155	0.001216695	1	3.05456e-05	0.003352333	0.001763504	0.0001449433
CoevoNDM	0.2347399	1.896791e-05	3.05456e-05	1	0.0001951394	7.997466e-05	3.558434e-06
SVM-Linear	0.0003677095	0.5123163	0.003352333	0.0001951394	1	1	0.6774329
SVM-RBF	0.0001559052	0.4828827	0.001763504	7.997466e-05	1	1	0.6567438
SVM-POLY	7.041752e-06	0.1941625	0.0001449433	3.558434e-06	0.6774329	0.6567438	1

Table A.16: T-test results for the Parkinsons benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.0045507	0.1594837	0.3348894	3.641779e-11	1.600026e-08	1.675067e-08
RBF	0.0045507	1	0.06092211	0.007044202	0.04607867	0.05084547	0.03463008
MLP-RBF	0.1594837	0.06092211	1	0.2655779	7.621309e-05	6.923425e-05	4.30162e-05
CoevoNDM	0.3348894	0.007044202	0.2655779	1	8.774716e-11	3.178495e-08	3.233771e-08
SVM-Linear	3.641779e-11	0.04607867	7.621309e-05	8.774716e-11	1	1	0.6691856
SVM-RBF	1.600026e-08	0.05084547	6.923425e-05	3.178495e-08	1	1	0.7087807
SVM-POLY	1.675067e-08	0.03463008	4.30162e-05	3.233771e-08	0.6691856	0.7087807	1

Table A.17: T-test results for the Sonar benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.003550389	0.0127483	0.0007003109	3.02086e-06	0.0007697649	0.0002802138
RBF	0.003550389	1	0.007614791	0.03049683	0.6404091	0.7914594	0.6830011
MLP-RBF	0.0127483	0.007614791	1	0.01692794	6.111046e-06	0.001704852	0.0006238888
CoevoNDM	0.0007003109	0.03049683	0.01692794	1	3.556531e-05	0.0077242	0.002954234
SVM-Linear	3.02086e-06	0.6404091	6.111046e-06	3.556531e-05	1	0.8624995	1
SVM-RBF	0.0007697649	0.7914594	0.001704852	0.0077242	0.8624995	1	0.8837397
SVM-POLY	0.0002802138	0.6830011	0.0006238888	0.002954234	1	0.8837397	1

Table A.18: T-test results for the Spect-Heart benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.004656614	0.5395858	0.005940688	0.0002087671	0.002367243	0.0007787374
RBF	0.004656614	1	0.003052198	0.0007417554	0.523879	0.5816299	0.5512011
MLP-RBF	0.5395858	0.003052198	1	0.006401815	0.0001483912	0.001674638	0.0005465899
CoevoNDM	0.005940688	0.0007417554	0.006401815	1	3.872753e-05	0.0004936529	0.0001494251
SVM-Linear	0.0002087671	0.523879	0.0001483912	3.872753e-05	1	1	1
SVM-RBF	0.002367243	0.5816299	0.001674638	0.0004936529	1	1	1
SVM-POLY	0.0007787374	0.5512011	0.0005465899	0.0001494251	1	1	1

Table A.19: T-test results for the VertebralCol2C benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	0.0001013794	0.9255249	0.1984193	6.652601e-06	6.669523e-05	6.723699e-05
RBF	0.0001013794	1	9.873279e-05	0.0001694893	0.7526413	0.6662553	0.5610208
MLP-RBF	0.9255249	9.873279e-05	1	0.1945746	6.185585e-06	6.439352e-05	6.480113e-05
CoevoNDM	0.1984193	0.0001694893	0.1945746	1	8.624262e-06	0.0001267464	0.0001399928
SVM-Linear	6.652601e-06	0.7526413	6.185585e-06	8.624262e-06	1	0.8620125	0.7233356
SVM-RBF	6.669523e-05	0.6662553	6.439352e-05	0.0001267464	0.8620125	1	0.8742095
SVM-POLY	6.723699e-05	0.5610208	6.480113e-05	0.0001399928	0.7233356	0.8742095	1

Table A.20: T-test results for the VertebralCol3C benchmark.

	MLP	RBF	MLP-RBF	CoevoNDM	SVM-Linear	SVM-RBF	SVM-POLY
MLP	1	8.2191e-05	0.6193142	0.004430189	2.602154e-06	6.353413e-05	0.0002839421
RBF	8.2191e-05	1	6.587411e-05	3.088676e-05	0.2951901	0.5041227	0.4200545
MLP-RBF	0.6193142	6.587411e-05	1	0.006705445	2.398934e-06	5.313941e-05	0.0002362636
CoevoNDM	0.004430189	3.088676e-05	0.006705445	1	2.225412e-06	2.754169e-05	0.0001116584
SVM-Linear	2.602154e-06	0.2951901	2.398934e-06	2.225412e-06	1	0.7789833	0.9986703
SVM-RBF	6.353413e-05	0.5041227	5.313941e-05	2.754169e-05	0.7789833	1	0.8259267
SVM-POLY	0.0002839421	0.4200545	0.0002362636	0.0001116584	0.9986703	0.8259267	1