TWO NOVEL ENSEMBLE APPROACHES FOR IMPROVING CLASSIFICATION OF NEURAL NETWORKS

KHOBAIB M. ZAAMOUT
Bachelor of Science, University of Lethbridge, Lethbridge, Alberta, 2010

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

To my parents and my wife, may God bless them.

Abstract

The task of pattern recognition is one of the most recurrent tasks that we encounter in our lives. Therefore, there has been a significant interest of automating this task for many decades. Many techniques have been developed to this end, such as neural networks. Neural networks are excellent pattern classifiers with very robust means of learning and a relatively high classification power. Naturally, there has been an increasing interest in further improving neural networks' classification for complex problems. Many methods have been proposed.

In this thesis, we propose two novel ensemble approaches to further improving neural networks' classification power, namely paralleling neural networks and chaining neural networks. The first seeks to improve a neural network's classification by combining the outputs of a set of neural networks together via another neural network. The second improves a neural network's accuracy by feeding the outputs of a neural network into another and continually doing so in a chaining fashion until the error is reduced sufficiently. The effectiveness of both approaches has been demonstrated through a series of experiments.

Acknowledgments

First and foremost, I would like to thank Dr. John Zhang for his support and guidance. This thesis could not have been written without him. He not only served as my supervisor but also encouraged and challenged me throughout my M.Sc. program accepting nothing less than my best efforts. Many thanks to my committee members Dr. Yllias Chali and Dr. Gongbing Shan for helping me in directing my work and for reading my thesis and giving me feedbacks.

I would like to thank Mr. Ben Burnett for setting up Condor (High Throughput Computing) and providing me with tremendous help and information that allowed me to make use of it. Without condor, it would have taken me months to finish my experiments. I also would like to thank Mr. Shah Mostafa Khaled for the help he provided to me throughout my degree. His patience in reading and correcting my scribbles as well as lending me his ear for any problem I encountered has given me much encouragement. I also would like to thank my brother Sa'ad Zaamout for his support on the home front, cooking, cleaning, and attending to me without a complaint. For that I will always be in debt.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Undeniably computers have transformed our lives. With computers, we possess the power to perform millions of computations in fractions of a second. The processing power and the consistency of such power make it possible to execute an enormous number of instructions with minimal errors, a task that is typically detested by humans. Some may wonder if such a tremendous ability of computers would constitute something we call *intelligence*.

1.1 Intelligence

Intelligence is defined by Merriam-Webster as "the ability to learn or understand or to deal with new or trying situations" [1]. This definition, of course, does not encompass the entire concept of intelligence since some animals, such as monkeys, dolphins, parrots, etc., are able to learn, understand and reason in new situations but they are viewed as less intelligent than humans. However, we shall consider this definition as a framework of the measurement of intelligence. In other words, an intelligent agent would be capable of:

1. learning: the ability to encode and utilize new knowledge toward a general target.

2. understanding: the ability to make necessary adjustments in behavior in accordance to the meaning implied in the knowledge.

Dealing with a new situation calls for having the ability to learn ideas and be able to understand and predict the outcomes of different situations. This ability allows forecasting and decision making. By this framework, computers are not intelligent agents since they lack learning and understanding. Therefore, computers lack decision making capabilities and are incapable of adjusting to new situations or new information. Therefore,

given the computational power of computers and the consistency of this power, the challenge becomes how we could formulate a given problem in such a way that all it requires is straightforward calculations to reach the desired outcome. *Artificial intelligence* is concerned with exactly this [50].

1.2 Artificial Intelligence

Many people have attempted defining *artificial intelligence*. The majority of the definitions consider intelligent agents to be those that think like humans and act like humans [50]. Evidently, all these definitions are human-centric. This is quite natural since our ultimate example of intelligence is the human being. The field of artificial intelligence is concerned with the development of intelligent agents capable of learning, understanding and therefore, making decisions and predictions [50].

Of course, the task in artificial intelligence is not always to develop intelligent agents. Rather, in most cases it is to formulate a problem in a manner that makes attaining a solution a matter of processing. Scientifically speaking, it is to recognize a pattern of solutions and then reformulate the problem in terms of the pattern. For example, a map navigation problem can be formulated as a search problem as follows: present the map as a matrix with passable and non-passable squares and specify an initial location in the matrix and a destination location. Then, the solution can be achieved by employing a search algorithm, such as *breadth-first graph search algorithm* [50].

Artificial intelligence has many subareas that deal with different types of problems or deal with a problem from different viewpoints. For example, *logical artificial intelligence* deals with representing knowledge of an environment in which an agent is supposed to act [40]. It creates the basis of the agent's world such that it will be able to make decisions and infer behaviors. *Search artificial intelligence* deals with formulating a problem in such

a manner that the solution can be achieved using a search technique just as in the map navigation problem described above.

Due to our limitations to understand, formulate, and model very complex problems, not all problems can be formulated and solved deterministically. With some problems we have no intuition regarding underlying relationships among various variables involved. Therefore, we have to be able to find solutions, or approximate solutions, while operating under uncertainty.

Consider a situation where we need to find relationships among various variables of a problem in order to formalize them. The problem can be translated to finding patterns among large number of variables. For example, the analysis of weather data can yield information on some relationship between rain and humidity, which then allows us to draw conclusions on the weather type. Therefore, we will be able to make classifications on the weather types, given the humidity, or predict the humidity level for a day, given the type of weather. This task essentially acquires knowledge by learning from data.

In the above example, data are any logical, numerical, or textual variables that can be processed in order to derive knowledge regarding the relationship between rain and humidity. Information in this example is the patterns, associations, or relationships among all the variables which can be understood and thus provide knowledge. The underlying field behind this learning and understanding is called *pattern recognition*.

1.3 Pattern Recognition

The task of *pattern recognition* is perhaps one of the most important and recurrent tasks that we encounter in our daily activities. Unlike computers, humans have the ability to perform many complex pattern recognition tasks effortlessly. For example, consider a task of text reading without expression or understanding. A person with qualifications of a third grader

can perform this task with utmost flawlessness. This task is purely a pattern recognition task. The reading process we undergo consists of vaguely two stages. Firstly, one has to be able to recognize various shapes in a given text as alphabets and properly name them. Secondly, the person must be able to recognize the different groups of these alphabets as words and pronounce these words in a manner that is comprehended by others. Perhaps the task of recognizing printed alphabetical shapes is not a very difficult task for a computer. A simple template matching technique would succeed. But if we vary the problem slightly, by using hand written text rather than printed one, then the template matching technique would fail easily.

As humans, we have this "given" talent of being able to see patterns that are nearly impossible to formulate. The task of pattern recognition is loosely defined by Fukunaga [21] as the task of determining to which category or class a given input instance belongs. In other words, it is the task of assigning a class label to a given input instance or generally being able to differentiate an instance from others by assigning a unique class to it.

Pattern recognition consists of two stages: *feature extraction* and *classification* [9]. Feature extraction is the task of providing measurements on a given input pattern. These measurements constitute properties of an input pattern that define the framework of its classification. In other words, these measurements are variables that define a problem domain. Feature extraction is an important task in pattern recognition. Without reliable features, the task of classification is impossible.

Classification is the task of mapping the features to specific predefined *classes* [12]. When examining the relationship between inputs and classes, we see that there are two different types of relationships that require different tools to handle them. In the next two sections, we shall consider two examples that demonstrate these different types.

Table 1.1: The inputs and outputs of the logical AND.

x	y	$x \wedge y$
1	1	1
1	0	0
0	1	0
0	0	0

1.3.1 Linearly Separable Problems

Consider the logical AND problem. In this problem the task is to determine the class of the output of the function $f(x,y) = c$, given all the possible combinations of the values of the variables (i.e., features) x and y, as shown in Table 1.1.

The features or variables of this problem are x and y. Each of the variables, including the output, can be either zero or one. The relationship between the input variables and the classes (i.e., outputs) is determined to be linear. That is, this problem is *linearly separable*. The reason is that the relationship can be modeled using a general linear form, as in Equation 1.1. We can rearrange this equation and change it to Equation 1.2.

$$Ax + By + C = 0 \tag{1.1}$$

$$-(A/B)x - C/B = y \tag{1.2}$$

The problem then becomes finding the proper ratio between A and B such that for all the inputs x and y there is a line that is perpendicular to the weight vector (A,B) that divides the output into two sets of distinct classes, as demonstrated in Figure 1.1. The three solid dots in the figure represent the elements of one class and the other empty dot represents the only element of the other class. In other words, the correct values for A and B must allow the equation to produce values larger than zero for the inputs of one class and smaller than

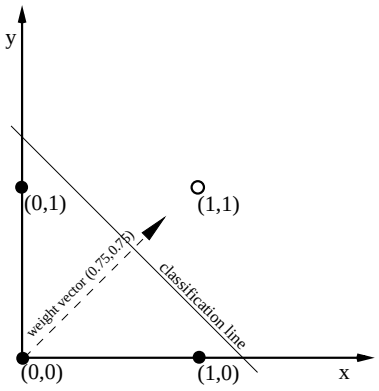


Figure 1.1: Logical AND is linearly separable.

or equal to zero for the inputs of the other class. We can examine the classification line for the logical AND problem using the values $A = 0.75$, $B = 0.75$ and $C = -0.75$, as is shown in Table 1.2. From this table we can clearly see a defining point in the outputs. All the instances of one class have been scored values less than or equal to zero while the instance of the other class have been scored a value larger than zero.

When dealing with linearly separable pattern recognition problems, many techniques can be employed. In the case of classification, *multivariate linear regression* [24], *k-nearest neighbors* [24], or *decision trees* [24], to name a few, can be used. Multivariate linear egression is also known as *multiple linear regression*. It is the technique we used in classifying the logical AND problem shown before. It attempts to find a dividing line between the classes of a problem by adjusting the weight vector. The *k*-nearest neighbors is a technique that classifies each record in a dataset based on *k* similar instances' classes in the dataset. The new instance will belong to the same class that the *k* similar instances belong to. The similarity is established by many ways, for instance, the Euclidean distance or Manhattan distance [24]. A decision tree is a tree-like structure that performs regression and classification. Given an input, the tree returns a decision. The tree is typically geared towards a specific target decision. In the case of pattern recognition, the decision would be whether a given input belongs to a specific class or not.

Table 1.2: Outputs of the linear form for the logical AND problem.

x	y	$Ax+By+C$
1	1	0.75
1	0	0
0	1	0
0	0	-0.75

Table 1.3: The inputs and outputs of the logical XOR.

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

1.3.2 Nonlinearly Separable Problems

Now, let us consider another example that is similar to the first in terms of the input variables but substantially different in terms of the classification.

In this problem, similar to the previous one, the task is to determine the class of the output of $f(x,y) = c$, given all the possible combinations of the values of the variables x and y, as shown in Table 1.3.

The relationship between the input variables and the classes (i.e., outputs) is determined to be nonlinear. The reason is that it cannot be modeled using a general linear form, as in Equation 1.1. That is, this problem is nonlinearly separable. Finding the proper ratio between A and B such that, for all the inputs x and y, there is a line perpendicular to the vector (A,B) that divides the output into two sets of distinct groups, is impossible, as shown in Figure 1.2.

From this Figure, the impossibility of finding a straight line to divide the two sets is evident. In other words, it is not possible to find values for A and B that allows the equation to produce values larger than zero for the inputs of one class and smaller than or equal to zero for the inputs of the other class. Therefore, a linear classifier will not be able to solve

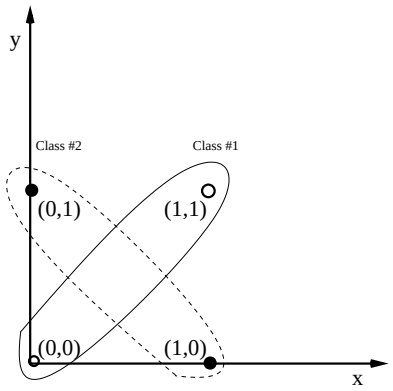


Figure 1.2: Logical XOR is nonlinearly separable.

this problem and a nonlinear approach is needed.

When the relationship between the inputs and the output classes is nonlinear (nonlinearly separable) then the task becomes more difficult. Moreover, it is not always the case that all the variables can be captured and the captured variables are not always well measured. For example, consider a real-world problem of predicting the weather in an upcoming day, given some variables, such as the past days' temperatures, humidity, pressure, etc. This problem is not well-defined since we do not know if the given variables are sufficient for predicting the weather. Furthermore, it is very common to have missing values, glitches, or noise in each of the variables, due to, for instance, sensor malfunctions or interference. In such cases, a model that is error tolerant, robust, and capable of learning is required.

For such tasks, many models present themselves, such as *Bayesian networks*. Bayesian networks are graph-like probabilistic models that provide the probabilities of a given input belonging to a specific class given previous data [24]. They learn posterior probabilities of events, or patterns, directly from the data such that they can make generalizations on them. This approach is well-known for its robust learning and fault-tolerance since they depend on general trends of data and the underlying distributions rather than specific values. Furthermore, it allows for incorporating prior knowledge and assumptions of a problem

domain by encoding them as distributions or relations into the learning process. However, this approach suffers from over simplification of problems since the distributions learned or encoded by experts can overfit the data and therefore can make nonrealistic predictions. Also, Bayesian networks have no robust means to estimate its structure, although some methods have been proposed. Therefore, they require a great deal of expertise and trial-and-error tests. Moreover, Bayesian networks require large amount of data in order to properly estimate the conditional probabilities.

Another model presents itself in dealing with the same situation. *Neural networks* are nonlinear predictive models that learn through training, as shall be seen in the following section.

1.4 Artificial Neural Networks

Artificial neural networks, also named *neural networks*, *connectionist models*, *parallel distributed processing models*, *perceptron*, *adaline*, *learning matrix*, or *neuromorphic systems*, are mathematical models based on our present understanding of biological nervous systems. Though the name strongly suggests that they are an imitation of the biological neural networks, the fact is that neural networks are a simplification of the biological model in that they only imitate the structure of the biological model through a collection of simple computational units interlinked by a system of connections.

1.4.1 Basics

A neural network consists of intricately interconnected components called *neurons*. The neurons are simple computational elements. The task of each neuron is to sum up the values of all other neurons connected to it. If the summation value exceeds some pre-

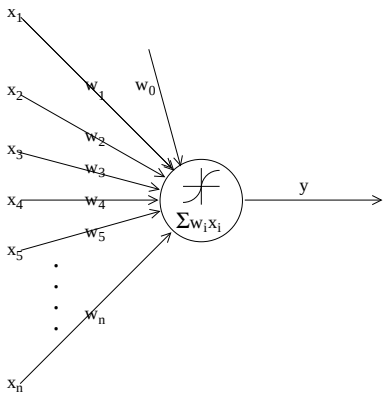


Figure 1.3: A single neuron or perceptron.

set threshold value, the neuron produces a value; otherwise, it produces another value. A single-unit neuron, named *perceptron*, as graphically illustrated in Figure 1.3 and mathematically defined in Equation 1.3, resembles multiple linear regression model in which the expected response y is related to the inputs $X = (x_1, x_2, ..., x_n)$. Neural networks consist of many neurons that resemble multiple linear regression models and thus each neuron acts as a tuning filter for a particular pattern in the data. This fact shows that neural networks simultaneously explore many hypotheses on different patterns in a given problem.

$$y = f(w_0 + \sum_{i=1}^{N} w_i x_i) \tag{1.3}$$

As shown in Equation 1.3, a neuron sums N weighted inputs and a bias and then passes the result of its summation through some nonlinear thresholding function, such as *hard limiters*, *sigmoid function*, *tanh function*, etc. Therefore, the produced value of a neuron becomes nonlinear. Moreover, the fact that the neuron produces different values depending on the threshold limit contributes to the nonlinearity of the outcome. Of course, there are more complex neurons which perform operations other than summation, such as *temporal integration*, but their use is limited to special cases.

Figure 1.4 shows a complete example of a neural network. The basic structure of a

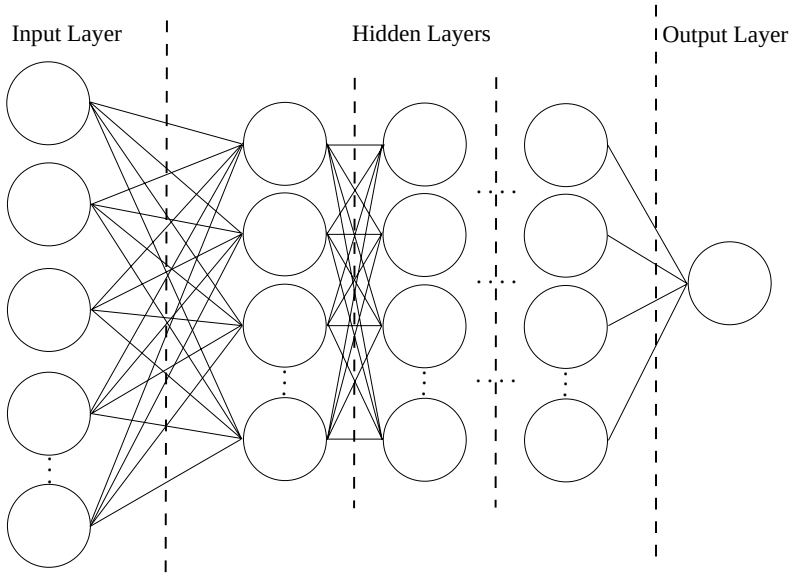


Figure 1.4: General structure of a neural network.

neural network consists of an *input layer* which contains the *input nodes* that represent and take in values of problem domain variables. Each of the input nodes connects to all neurons in the next layer via weighted links. The neurons of this layer are called *hidden neurons* and the layer is accordingly named *hidden layer*. A neural network can have many hidden layers with various numbers of hidden neurons. The number of hidden layers and neurons is typically decided based on a trial-and-error process such that the chosen number makes the network perform the best classification while keeping the computations feasible. The neurons of each hidden layer are connected to all neurons from the proceeding layer via weighted links. The final layer in a neural network is called the *output layer*. This layer determines the class for each input instance.

The logical XOR problem we described previously, for example, can be solved using a simple multilayer neural network, such as the one shown in Figure 1.5. With a threshold value of zero (0) for the neurons of the hidden layer, this network will yield the value zero (0) for instances of one class (where x and y are equal) and the value one (1) for the instances of the other class (where x and y are not equal).

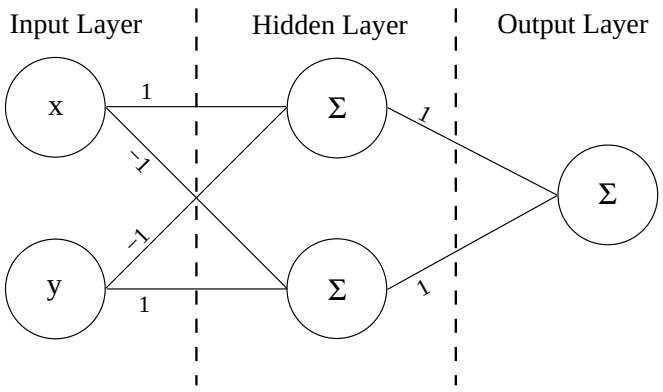


Figure 1.5: The XOR neural network structure.

Neural networks provide a great degree of robustness and fault tolerance. Their robustness is due to the training process via feedback loops which, together with nonlinearities of the neurons, make them very adaptive to changes and therefore can handle complicated dynamic real-world problems. The fault tolerance of neural networks is due to the relatively large number of hidden neurons. Since each of them is strongly connected to all neurons or input nodes, damage to a few nodes, neurons, or links will not significantly impair the overall performance [38].

Most neural network learning algorithms adapt connections' weights to improve performance based on the error of classification. Adaptation or learning is a major advantage of neural networks. The ability to adapt and continue learning is essential in problems, such as *hand writing recognition* or *speech recognition* where new inputs are continuously encountered. Moreover, since neural networks are non-parametric models, they make no assumptions regarding underlying distributions and learn directly from data. This property, combined with the adaptation capability, provides a high degree of robustness which allows for accommodating variabilities in inputs [38].

There are many types of neural networks developed with intriguing capabilities. In the following section, we present a few of them that we find to be interesting.

1.4.2 Different Types of Neural Networks

Feedforward Neural Networks

This type of neural networks is the simplest and most commonly used. It is the first network that learns and has an accommodating structure. In this network, inputs propagate strictly in one direction. That is, through the input nodes, the input gets weighted, summed, and projected by some thresholding function in the hidden neurons of the hidden layer(s), if any exist. The results are then further weighted, summed and projected onto the output layer. Feedforward neural networks contain no cycles in their structure and are typically constructed from two different types of neurons, binary and continuous. Binary neurons use hard limiters as their thresholding functions, i.e., they output either zero or one. Continuous neurons use a continuous function, such as sigmoid, tanh, or *radial basis* functions [38, 16, 34, 36, 60, 63].

Feedforward networks use back-propagation of *delta errors* as the means of learning. That is, the training data consists of pairs of an input and a target. With this approach an input is presented to the network and the output is compared with the target. If they differ, the weights of the network are altered slightly to reduce the error in the future output. This is repeated as necessary until the network produces the desired output.

Recurrent Neural Networks

Recurrent neural networks are bi-directional feed networks. Their basic structure was created in the 1980s. The network consists of a number of neurons connected to each other via directed links. The neurons transport information backward through the network as well as forward. This way, the network can memorize the previous states of the hidden layers with

each input presented [45].

Cascading Neural Networks

Cascade correlation is a neural network architecture and a supervised learning algorithm developed by Scott Fahlman and Christian Lebiere [18]. Instead of just adjusting the weights in a network of fixed topology, cascade correlation begins with a minimal network, and then automatically trains and adds new hidden neurons one by one, creating a multi-layer structure. Once a new hidden neuron has been added to the network, its input-side weights are frozen. This unit then becomes a permanent feature-detector in the network, available for producing outputs or for creating other more complex feature detectors.

The cascade correlation architecture has several advantages over other learning algorithms. The network learns very quickly and can determine the size and topology of its network structure. Moreover, it retains the structure it has built even if the training set changes, and requires no back-propagation of errors through the connections of the network.

1.5 Our Contributions

Our contributions in this thesis target improving the classification of neural networks. We propose two novel ensemble techniques, *paralleling neural network ensemble* and *chaining neural network ensemble*.

The paralleling ensemble technique uses a neural network to combine the predictions of a collection of neural networks with the intention to improve the classification. Detailed explanations and discussions can be found in Chapter 4. The chaining ensemble technique aims at improving classification by feeding the predictions of one neural network into an-

other and repeatedly doing so such that each newly produced network will contain the predictions of all or some previous networks. This technique is shown to be very effective and is discussed in detail in Chapter 5.

1.6 Thesis Outline

This thesis consists of six chapters. In Chapter 2 we will give a brief history of neural networks followed by an overview of back-propagation neural networks and their learning. Moreover, we will discuss some of their advantages and disadvantages and overview some of the structure estimation techniques available. In Chapter 3 we will describe the datasets we used to validate our approaches. Information such as the sources of the datasets, how the data was gathered, attribute descriptions, and some descriptive statistics will be provided in this chapter. In Chapter 4 we will describe our paralleling ensemble approach and narrate the process we undertook for performing our experiments as well as discuss in detail our experiments results. In Chapter 5 we will describe our chaining ensemble approach, detail the process we undertook, and discuss experiment results. Chapter 6 concludes the thesis with some remarks on our future work.

Chapter 2

Background

2.1 The History of Neural Networks

The study of the human brain and its cognitive process dates back thousands of years. However, it was not until the last two centuries that formulations and scientific explanations of the internal works of the human brain have been given. The main questions that needed to be addressed were those of learning, information representation and storage, and information recall. In 1873, Bain [7] recognized that the human brain consists of intricately connected components called *neurons*, each of which connects to others via *synapses*. He theorized that the memory in the brain is formed as a reaction to a stimulus from an environment. He further theorized that the memory consists of a set of nerve currents weaker than that produced by the causing stimulus. Moreover, he suggested that the ability to recall a specific memory requires that an association first be made with another memory, sensation, or motor action via some kind of neural growth. These two ideas gave birth to the first threshold neural structure and laid the basis to what came to be known as Bain's *adaptive rule* [7, 43].

In 1890, an American psychologist James [29] restated Bain's adaptive rule and attributed all thoughts and motor actions to the flow of neural electrical currents named *engrams*. James proposed that the flow of electrical current is directional. That is, the current flows from charge-saturated regions to regions lacking electrical charge. He proposed that the intensities of thoughts and actions are in direct relation to the current flow rate [29, 43]. James's proposal defines learning as adapting or forming new paths between neurons that are repeatedly active together [55]. Although his work contains no mathematical formulation of learning, it strongly resembles a key local neural learning mechanism, such as

Hebbian learning which will be discussed later in this chapter.

It was not until 1938, when electronics and computers had become available, that the first attempt was made to model human thinking process. Rashevsky [43] devised an electronic circuit similar in structure to that in Figure 1.5, showing how a binary logic XOR operation could be implemented using addition and subtraction operations. In 1943, a neuro-physiologist Warren McCulloch and a mathematician Walter Pitts formulated the *McCulloch-Pitts Theory of Formal Neural Networks* [26], which was implemented using simple electric circuits. Their implementation came to be the first known *artificial neural network*. However, their proposed network lacked learning and adaptation. In 1949, Donald Hebb [26] laid the basis of the concept of neural learning by describing how neural connections can be strengthened or weakened each time they were used. His work became the essence of *Hebbian learning*.

In 1958, Frank Rosenblatt developed the *perceptron* [47], as shown in Figure 1.3. The perceptron was the first learning and adaptable artificial neural network that employs supervised learning to learn its weights. The robust and adaptive nature of this model answered significant questions, such as *in which form information is stored* and *how stored information influences recognition*, etc.

The perceptron is the simplest form of a neural network since it is only capable of classifying linearly separable patterns which is a major downfall. Its structure consists of input nodes, a single output neuron, adjustable weights, and a bias. Rosenblatt proved that the perceptron learning algorithm converges when training samples are sampled from two linearly separable classes [6]. The perceptron ignited research into neural networks and ushered in a new age for artificial intelligence [9, 43].

In the 1970's *Self-Organizing Maps (SOM)* were developed by Teuvo Kohonen [35]. Inspired by the studies of the visual cortex region in the brain, SOM are unsupervised neural networks that learn to map a set of artificial neuron inputs to outputs in lower dimensional

space. That is, SOM reduce the dimensionality of the input data while performing unsupervised training and classification. In other words, they provide a way of transforming multidimensional data to data in much lower dimensionality. This reduction of dimensionality employed by SOM is in essence a data compression technique known as *vector quantization* [9, 43].

In 1982, Hopfield [9, 43] introduced a new type of neural networks that became known as the *Hopfield model*. The structure of this model is similar to the one of a perceptron but is different in that it has bi-directional connections between neurons. Moreover, the network has no special input nodes and the neurons are fully connected and symmetrically weighted. That is, the weight of a connection from one neuron to another is the same in both directions. Hopfield networks learn using a Hebbian-variation rule which maintains the locality of Hebbian learning rule, i.e., connections between neurons become stronger or weaker depending on their associations, but differs from the Hebbian rule in that the network changes its connections' weights in a manner such that fewer paths will be active.

In 1986, back-propagation networks were introduced by a number of researchers. These networks are excellent classifiers and are applicable to large number of problems. We use this type of networks throughout our work. In Section 2.3, we shall discuss them in more detail. In the following section, we shall discuss key learning techniques used in neural networks.

2.2 Learning in Neural Networks

Learning in the context of neural networks is defined by [26] as a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by a manner in which the parameter changes take place. In other words, the learning process of

a neural network is determined by a manner in which the *tuning features* of the network are to be changed. The tuning features are weights, threshold values, etc. The stimuli that actuate the tuning process differ and are determined by the environment. In some cases, *delta error* in prediction is used while in others *spatial distance* is used. The differences in the stimuli and in the manner the tuning features are changed determine the type of learning we are performing.

There are three main categories of learning, *supervised learning*, *unsupervised learning*, and *hybrid learning* [28]. Supervised learning is also known as learning with a teacher. In this type of learning, a neural network is presented with input patterns along with their corresponding expected outputs. When the network's output is not equal to the expected output, the difference is used as the actuator of change of the weights of the network. Unsupervised learning, as the name suggests, does not require the target value or expected output. The network learns the underlying structure in the data and becomes able to organize the patterns. Hybrid learning uses both the supervised and the unsupervised techniques on different patterns such that the weights of some parts of the patterns are determined with the supervised learning while the others are determined using the unsupervised learning. Sometimes both techniques are applied to all the patterns. A brief overview of the most common and effective methods of neural learning is presented below.

As mentioned above, neural networks have a large number of learning mechanisms, such as *Hebbian learning*, *memory-based learning*, *competitive learning*, *boltzmann learning*, etc. Hebbian learning is a supervised local learning technique that is derived from a theoretical neural learning mechanism proposed by Donald Hebb in 1949. The locality of learning means that changes to any given connections' weights are only related to the two neurons connected through it. Some criticism toward Hebbian learning is in that the Hebbian weights tend to grow exponentially [9, 26].

Memory-based learning is a group of learning algorithms that depend on memorizing

the training data and performing classification explicitly on the stored instances. In this type of learning, the past training instances are stored in memory and then recalled when classification of a new input is required. There are many algorithms under memory-based learning. For example, *nearest neighbor* is a memory-based learning algorithm that defines the locality of a new instance as the nearest instance to it, and therefore the class of the new instance is the one of its nearest. On the other hand, *k-nearest neighbors* algorithm defines the locality as the closest k instances to the new instance where k is an integer greater than zero. According to this algorithm, the class of a new instance becomes the one of the majority of its neighbors [26].

Competitive learning is a learning rule based on the idea of "winner takes all". In this type of learning weights are adjusted such that only one neuron in a layer at a specific iteration can fire at a time. Competitive learning is useful for classification of categorical-class data and is especially useful for discovering statistically significant features [26]. The mechanism of allowing one neuron to output at a time forces neurons to become feature detectors specialized on different classes of input patterns. The winner neuron learns by removing some of the weights of all its inactive inputs and distributing that weight equally amongst the active inputs. No learning occurs for the other neurons.

Boltzmann Learning is developed by Geoffrey Hinton and Terry Sejnowski [3]. Boltzmann learning is a statistical stochastic learning algorithm that introduces a new neural structure [26]. Neural networks built around this rule are named *Boltzmann machines*. These are undirected graphical models that are symmetrically connected with neuron-like units that make stochastic decisions about whether to be on or off. They provide a powerful tool for representing a dependency structure among random variables. The learning process is governed by an energy function in which the learning task attempts to minimize [26, 3].

The learning strategy used in back-propagation is, abstractly speaking, learning from

mistakes. Similar to the perceptron network, the error of a network is determined by the difference between the network's prediction and the actual class value. The error is then propagated backwards to contribute to the weight adjustment for each connection in the network in each layer. Since our work uses this learning mechanism, it is discussed more below [9].

2.3 Back-propagation Neural Networks

2.3.1 Basics

Back-propagation networks are multi-layered networks that use *back-propagation learning*. Back-propagation learning is a supervised learning that is an extension of the *Widrow-Hoff delta rule* to multiple layers proposed in 1986 [26]. The name of this learning mechanism stems from the way that the error of a network under training propagates backwards to adjust the connections' weights. The networks built around this learning technique are very robust, error tolerant, and adaptable.

The power of back-propagation neural networks is rooted in their structure and learning mechanism. The basic structure is shown in Figure 1.4. It consists of as few layers as three or as many layers as a computer can handle. The same can be said about neurons. The basic neuron of the network is similar to a perceptron and is shown in Figure 1.3. The typical structure of a back-propagation neural network consists of three layers, an *input layer*, at least one *hidden layer*, and an *output layer*.

Figure 1.4 shows a template of neural networks. The input layer contains the *input nodes* that represent and take in the values of the variables of a problem domain. Each input node connects to all the neurons in the next layer via weighted connections. Input nodes are the representation of an environment in which the network is trying to learn. The

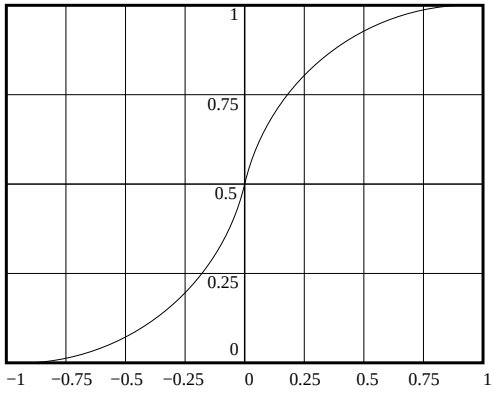


Figure 2.1: The sigmoid function.

neurons of the hidden layer(s) are named *hidden neurons*. A neural network can have many hidden layers with various numbers of hidden neurons. The number of hidden layers and neurons is determined using many techniques as will be discussed in Section 2.6. Each neuron in a hidden layer is connected to all neurons in the proceeding layer via weighted connections.

The final layer in a back-propagation neural network is named the *output layer*. This layer contains a number of *output neurons*. These neurons determine the class for each input instance. The output neurons, as well as the hidden neurons, are more intricate than the input nodes, as demonstrated in Figure 1.3. They are computational units which sum up the weighted inputs and project them into some thresholding function. The value of the thresholding function gets weighted and propagated to the next layer, given that the summation value exceeds the threshold limit, as formulated in Equation 1.3. The most common thresholding function used in back-propagation neural networks is the *sigmoid function* formulated in Equation 2.1 and illustrated in Figure 2.1.

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

The sigmoid function is a type of logistic functions, also named *thresholding* or *transfer*

function, that resembles an "S" shape and produces values within the range [0, 1]. The reason of popularity of the sigmoid function is due to the fact that it is differentiable which eases the weight adjustment process [50].

2.3.2 Learning

Back-propagation learning, also named *delta rule, error-correction, or widrow-hoff*, is a form of supervised learning that trains a given network based on expected outputs. That is, the network is trained on making predictions by repeatedly adjusting its weights such that its predicted output matches the expected output or comes subjectively as close as possible.

This process can be formalized as follows. For a given untrained network, the weights w of the connections are initially assigned through some mechanism, such as random assignment or constant value assignment. The back-propagation learning algorithm adjusts the weights of the untrained network with each training iteration i in proportion to the error of the prediction $e(i)$ produced by the network (i.e., the difference between the network's output $y(i)$ and the actual desired output $d(i)$ associated with the specific iteration $e(i)$), as in Equation 2.2 [9, 26, 28].

$$e(i) = d(i) - y(i) \tag{2.2}$$

Equation 2.2 constitutes the stimulus that triggers the weights' adjustment process. In order to make the weights' adjustment proportional to the error, we need to define a cost function based on the error produced at each training iteration i. This cost function must not exhibit the same oscillation behavior of the error function $e(i)$. Otherwise, this would make automating the error reduction a much more difficult task. The learning algorithm observes and attempts to reduce the values produced by a cost function defined based on

the error of Equation 2.2. This cost function is defined in Equation 2.3.

$$\varepsilon(i) = \frac{1}{2}e^2(i) \tag{2.3}$$

The task of learning becomes an optimization task of minimizing the cost function $\varepsilon(n)$ and thus can be solved using some methods, such as *Gradient Descent*. The mechanism by which the weights are adjusted is the merit of the learning. In this learning mechanism, the adjustment for each connection's weight w_{kj}, connecting neurons k and j, for a specific training iteration i is defined in Equation 2.4:

$$\Delta w_{kj}(i) = \eta\varepsilon^{`}(i)x_j(i) + \beta(w_{kj}(i-1) - w_{kj}(i-2))$$

$$\Delta w_{kj}(i) = \eta e_k(i)x_j(i) + \beta(w_{kj}(i-1) - w_{kj}(i-2)) \tag{2.4}$$

where η is the *learning rate*. Learning rate is a neural network's control or tuning parameter. It is used by some learning algorithms to control the amount of weights that need to be adjusted. Typically, the value of the learning rate ranges between [0, 1]. A high learning rate causes the network to make radical changes to the weights, making the previously learned weights almost overruled at each iteration. This may cause the network to diverse. On the other hand, a low learning rate allows for small weight changes, which causes the network to take longer to converge. Another control parameter is *momentum*, denoted as β. It is a term associated with each connection that varies within range [0, 1]. Its main purpose is to help reduce the oscillation of the weight changes and to prevent the system from converging to a local minima or *saddle point*. Therefore, it helps the objective function to converge faster and ultimately speeds up the learning process. Setting a momentum too high can create a risk of missing the minimum while a momentum that is too low cannot

reliably avoid local minima [53].

Now that we have determined the weight adjustment value Δw_j, we can formulate the update value for a specific weight at the next iteration $w_j(i+1)$, as shown in Equation 2.5.

$$w_{kj}(i+1) = w_{kj}(i) + \Delta w_{kj}(i) \tag{2.5}$$

2.4 Advantages of Neural Networks

Neural networks' advantages are due to their components, such as the thresholding functions, their structures, and learning mechanisms. They are excellent classifiers as they are able to classify both linearly and nonlinearly separable problems. This is due to the nonlinear transformation they perform on the learned data which allows them to fit linearly separable problems as well as more complex nonlinearly separable problems [57].

Many learning algorithms and neural structures have emerged, giving neural networks a selection of methods to improve performance. Neural networks are also error tolerant. This is largely due to the relatively large number of neurons they contain. Errors in the form of missing data, noise or glitches get averaged out over the entire network. This effectively reduces their impact to minimal. Neural networks are also very robust in that for given a dataset, neural networks can adjust themselves to fit the given data automatically via some learning algorithm [26].

The true power of neural networks is demonstrated when they are applied to complex multivariate nonlinear problems. Neural networks require no prior assumptions or knowledge regarding the underlying relationships between variables of a given problem, since they learn directly from the data in a robust manner.

Neural networks express many statistical techniques, i.e., *regression models* from *simple linear regression* to *projection pursuit regression*, *nonparametric regression*, *general-*

ized additive models, *logistic regression*, *Fisher's linear discriminant function*, *classification trees*, etc. [16]. This furthered interest in neural networks in the fields of data analysis.

Although neural networks are effective, there are still many ways to improve their classification accuracy. Many techniques, such as the *input preprocessing*, *modular approach* [39] and the *ensemble technique* [25, 44, 64], etc., can be used toward this end. These techniques will be discussed in Chapter 2.7.

2.5 Disadvantages of Neural Networks

A major disadvantage of neural networks is in the difficulty to interpret the meaning of its structure. That is, given a trained network, it is difficult to derive meanings from the weights of the network to understand the underlying relationships between the inputs and the outputs. Although the network is excellent at detecting significant features and relationships, it is difficult to understand them [57].

Neural networks require a large number of training instances to be able to generalize well on a given problem. Moreover, they require knowing, prior to training the network, what features of the data are more indicative to the class since neural networks do not learn such information. Attribute selection and preprocessing, such as normalization, discretization, etc., are often required [57], as to be discussed shortly in Chapter 3. Moreover, it is difficult to determine the best neural network structure and learning time for a given problem domain. Although many techniques are presented to deal with this problem, no state-of-the-art algorithm is able to determine the best neural structure.

2.6 Structure Estimation

The numbers of hidden layers and neurons are typically decided based on a trial-and-error process such that the chosen number makes the network perform the best classification while keeping the computation feasible. Some techniques have been proposed to estimate the structure of neural networks. They fall into two broad categories: *constructive algorithms* and *pruning algorithms* [56].

Constructive algorithms start with an initially small network and iteratively add new hidden neurons and their weights until a desired error is achieved [46]. Some of these algorithms that demonstrated good potential include *tower algorithm* and its derivative *pyramid algorithm* [22], *upstart algorithm* [20], *cascade-correlation learning architecture* [18] and its extension *perceptron cascade algorithm* [13], and *tailing algorithm* [41].

On the other hand, pruning algorithms start by an initially large network and train it until a desired error is achieved. Then the algorithm eliminates neurons that are no longer in use [46]. Pruning algorithms can be categorized into two categories: *sensitivity* methods and *penalty-term* methods.

The sensitivity methods provide a measure on the sensitivity of the error function of a given network [46]. Some examples are *skeletonization* [42], *shadow arrays* [31], and *optimal brain damage* [17]. The penalty-term methods operate by rewarding or penalizing the network through some terms for choosing efficient solutions [46]. Some penalty-term algorithms are cost function penalty [15] and error function modification [30].

2.7 Improving Classification in Neural Networks

While neural networks are effective in many application domains, there are still continuing efforts to improve their classification accuracy and general performance. There are three

areas in which neural network models can be improved. Since the classification task is a process relying on inputs, classification model, and output utilization and boosting, it is in these parts that work can be done to enhance the accuracy and efficiency of the task of classification.

Inputs or features are variables of a problem domain. They differ in their descriptiveness of a target class. Models are any type of classifiers which can be tuned via some control properties of the models themselves. Output utilization and boosting are manipulations of the results of a model such that the output is enhanced. The general categories of these tasks are preprocessing, model manipulation, and ensembles and similar techniques.

Preprocessing is a general term for any operation performed on the input data prior to feeding it to the classifier. Since neural networks do not select or rank attributes based on their benefit for classifying the class, like in logistic regression for instance [57], preprocessing is sometimes needed. Model manipulation is the changes or adaptations performed on the structure of a given model such that it improves its classification. In backpropagation neural networks, the parameters, such as learning rate, momentum, number of hidden layers and neurons, and choice of thresholding function, are all aspects of the structure of the network in which improvements can be made. *Ensemble* and *modularity* are techniques developed to utilize the outputs of a number of neural networks in order to arrive at better classification.

2.7.1 Preprocessing

Preprocessing means to perform some work on the raw data in order to enhance or extract specific features to improve accuracy of classification [51]. In our work, we make use of two main preprocessing methods, namely, *data normalization and cleaning*, and *attribute selection*.

Data normalization refers to the task of reducing a variable's range to a predetermined range, typically [0, 1]. The need for data normalization comes of the use of thresholding function. The sigmoid function, for example, requires the data to be normalized to [0, 1]. Sola and Sevilla [54] have demonstrated that normalization has reduced network training iteration and final error.

Data cleaning is a general term that targets many problems, such as *Fourier transformation* and *low-pass filters* for signal noise removals, *discretization* and *modulation* for binning continuous signals, etc. However, in our case, we use it to refer to outlier removal process. This task is concerned with removing contradictions from the training data. Rosin et al. [48] have shown that removing outliers helps reduce overfitting. They demonstrate the impact of outliers removal on neural network learning by applying a *k-nearest neighbors filter* to the input data.

Attribute selection, *feature selection*, *dimensionality reduction*, etc. are preprocessing techniques that select a subset of features from a given data set such that the selected subset will increase classification accuracy while reducing dimensionality and complexity of the data. This technique is commonly used in machine learning. Typically, attribute selection methods are performed to target specific characteristics in the data, such as correlation between attributes and classes, distinguish-ability of the instances, etc.

There are many methods that can be applied to perform this task. Each method available has its advantages and disadvantages. In order to select a suitable method, one must have a good understanding of the nature of the input data and the classifier that will be used. For the purpose of our work we employ *Correlation-based Feature Selection (CFS)*, *ReliefF*, and *Principal Component Analysis (PCA)*, which will be discussed later in Chapter 3. These methods are used to target the following characteristics in our data:

1. Correlation among attributes (excluding class). We employ PCA to reduce the di-

mensionality of the data and derive new attributes that are completely independent of each other, so called *orthogonal* attributes.

2. Dissimilarity of an attribute's values belonging to different classes and the similarity of these values belonging to the same class. ReliefF aims for this purpose. Ideally, an attribute will have a set of values belonging to each class category and the sets for different classes do not intersect.

3. Correlation between attributes and class. CFS selects a subset of attributes that are most correlated with the class and least correlated with each other. Also using ReliefF could exclude the attributes that are highly correlated among different classes.

There are no guarantees that the produced subset is the optimal one. If such a subset is needed, one has to perform a comprehensive search of the problem domain attributes such that all possible subsets are evaluated using standardized neural network and thus the best subset chosen would most certainly be the one that produces the lowest error. However, this method is not computationally feasible.

2.7.2 Model Manipulation

Model manipulation refers to tuning the various control properties of a neural network, such as learning rate, momentum, the target function, the learning rule, the number of hidden layers and neurons. We have discussed the different learning rules, the different structures, and the different control properties of neural networks in Section 2.2. In this section we overview few methods for improving neural network efficiency and prediction power.

A significant amount of literature has discussed the benefits of dynamic learning rate on the learning performance of back-propagation neural networks. When examining a given network's error in response to learning iterations, it becomes evident that the error has a

trend where it increases, decreases, fluctuates, or is a combination of them. In a situation where the error of two consecutive iterations is increasing or decreasing, it would more efficient to increase the learning rate since the weights of the network need to be changed in one direction. In other situations, the error would be fluctuating and therefore the learning rate needs to be reduced such that the weight changes are not radical. This means that a fixed learning rate is inefficient and that the network must be able to decide when to learn faster or slower based on the error produced. A number of techniques have been suggested for tackling this problem [19, 49, 61].

The thresholding function used in neural networks is another area where improvement can be made. As mentioned previously, the sigmoid function is a very common function in back-propagation networks. However, it suffers from the problem of getting stuck in local minima. One solution to this problem is in the concept of momentum, which was discussed previously. Another solution is in using *Radial Basis Function (RBF)* [9, 60, 10, 11].

2.7.3 Ensemble and Modularity

Ensemble

The *ensemble* techniques [25, 44, 64] have been proposed as a mechanism to improve classification power. They aim at combining the outputs of different classifiers members, denoted *CR*, trained on the same task, through some gating function to increase the overall classification accuracy [58, 50]. Some of the combining mechanisms are, *plurality*, where the correct classification is the one that was agreed upon by the largest number of ensemble members, *majority voting*, where the correct classification is the one voted on by more than half of the ensemble members, etc. [25]. The ensemble members may be trained using different subsets of a given data set, using different classification techniques, or using

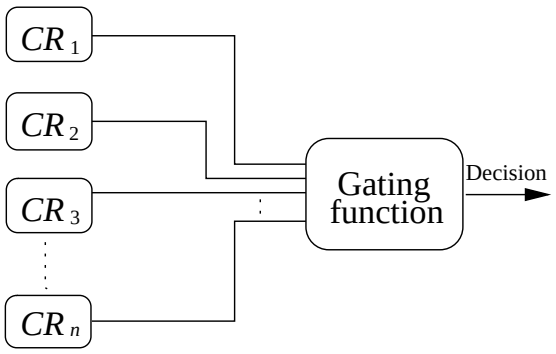


Figure 2.2: An ensemble architecture.

the same technique but varying parameters, structures, or learning mechanisms. All the ensemble networks, however, are trained to predict the same target class, as illustrated in Figure 2.2.

Neural networks can be used as ensemble members since they could be varied in many ways. Those trained on the same dataset can still perform differently due to facts, such as the randomization of training and testing sets, different initial weight values, etc. This means that each network could result in different classifications and errors. Therefore, a decision combining technique that is able to detect the correct classifications from a collection of individual neural networks could avoid the different errors and produce even more accurate classifications. Therefore, the collective decision of the ensemble is less likely to produce errors than any single neural network does, if their outputs are handled correctly.

Typically, the creation of a neural network requires a trial-and-error process involving the network's structure, e.g., input attributes, the number of hidden layers, the number of hidden neurons in hidden layers, etc. and the network's logical properties, e.g., neurons' thresholds, weights, learning rate, learning iterations, etc., to arrive at one that performs the best in terms of classification accuracy. Since the varying neural networks will make errors on different areas of the input space [25], it is intuitively obvious that a good combining

technique would yield more accurate classifications.

Modular

The modular approach is concerned mainly with breaking down a problem into smaller sub problems, in which separate networks (experts) will be trained and their outputs are then combined [39, 58, 8, 52]. This approach differs from the ensemble in that the networks are trained on different targets. For example, each network could be trained to predict a different class or is trained on different data, etc.

The modular approach is dependent on two main aspects: the *gating function* and the *division of data*. The gating function is the function that determines the correct output from the outcomes of the expert networks. Many gating functions, such as majority voting and plurality, can be used. The division of data is an essential part. Given a data set, the experts will be built and trained on disjoint sets of the dataset. One example would be in the case of movie classification. A modular approach to this problem would have a number of expert models each trained on a separate aspect of the movie, such as audio, subtitles text, graphics, etc., as shown in Figure 2.3. Another example is that of [5]. For a given dataset with k classes, a modular approach to classifying this dataset is by dividing the set into k 2-class problems such that for each subset, the classification becomes whether a given instance in that subset belongs to a class or not.

In our work, we will present two novel ensemble approaches that are shown to be very effective in further increasing neural networks' classification accuracy. Moreover, we present a new mechanism of neural chaining in which is also demonstrated to be effective at improving accuracy.

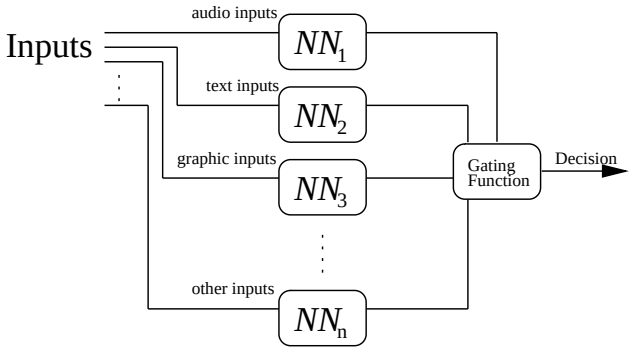


Figure 2.3: Modular neural architecture.

Chapter 3

Data Preparation

Four datasets were used to perform the validation experiments of our proposed approaches, as to be presented in Chapters 4 and 5: *Agassiz Tomato Yield*, *Synthetic*, *Steel Plates Faults*, and *Restaurants Reviews*. Their summaries are shown in Table 3.1.

3.1 The Datasets

3.1.1 Agassiz - Tomato Yield Dataset

The first set, *Agassiz*, is denoted as D_{AGA}. This dataset was obtained from Dr. David Ehret, Pacific Agri-Food Research Centre. The data in the set were collected from four independent greenhouse compartments with a growing area of 65 m^2 each. A subset of 12 plants was used to measure growth and yield for each compartment. The growth of all 12 plants is averaged and recorded.

CropAssist, an agriculture application [27], was used to collect the data routinely at one minute time interval. D_{AGA} is a record of the growth and yield of greenhouse tomato plants under controlled temperature, light, humidity, and feed conditions. The data consists of 17 input variables and a single continuous class variable, as shown in Table 3.1. Each measured variable is either averaged or summed over a one-week interval.

The input variables consist of seven (7) temperature-related variables (minimum temperature, maximum temperature, average temperature during day, average temperature during night, average temperature over 24 hours period, growing degree day GDD, and accumulated GDD), three (3) humidity-related variables (Average humidity during day, night, and over 24 hours period), four (4) light-related variables (average light during day, night

Table 3.1: Summary of datasets.

Dataset Name	# Attributes	# Instances	Class Type (#)	Missing Values?	Data Characteristics
D_{AGA}	17	72	Real	No	Time Series-Real
D_{SPF}	27	1941	Categorical(7)	No	Integer-Continuous
D_{SYN}	17	7280	Real	No	Times Series-Real
D_{RER}	21	405	Categorical(2)	Yes	Integer

and over 24 hours period as well as the added light), one (1) growth variable which is a measure of the growth of all above-ground tomato plant, including stems, leaves, and fruit in centimeter, one (1) water use variable which is the amount of water retained in the plant in milliliter, and one (1) harvest variable which is the total weight of all harvested tomatoes in a specific day in kilogram.

3.1.2 Synthetic Dataset

The second dataset is called *Synthetic* dataset and is denoted as D_{SYN}. It is a synthetic dataset that was extrapolated from the D_{AGA} dataset. For the purpose of our experiments, we randomly increase or decrease each attribute's value in D_{AGA} by a random amount, creating a new value from the original. This process was repeated a hundred times for each instance in D_{AGA}, resulting in a total of 7280 instances while maintaining the original number of attributes, as shown in Table 3.1.

In more detail, for each variable in D_{AGA}, the minimum value was subtracted from the maximum value and then divided by 10, to produce a value n. Then, for each instance of the variable, a random value between -10 and 10 is selected and multiplied by the value n and then added to the instance of the variable to derive a synthetic data point. This process is repeated for each variable and each instance in the dataset to derive the synthetic dataset. It was repeated a hundred times for each instance in the original dataset, producing a total

Class values	# of instances
1	158
2	190
3	391
4	72
5	55
6	402
7	673

Table 3.2: D_{SPF} instances distribution over class values.

of 7280 instances while maintaining the original number of attributes.

It should be noted that some variables adhere to certain value constraints which were observed after having produced the synthetic data. D_{SYN} was created to verify the efficiency of our approach and to be used as a control dataset to demonstrate how our approaches handle random data.

3.1.3 Steel Plates Faults Dataset

The third set is called the *Steel Plates Faults* dataset and is denoted as D_{SPF}. It was obtained from UCI machine learning repository[1] which was donated by Semeion Research Center of Sciences of Communication [14]. It consists of 27 independent attributes and 1941 instances. It records various aspects of steel plates, such as type of steel, thickness, luminosity, etc., which allow predicting various faults in steel plates. The class attribute of this dataset consists of 7 classes representing different types of faults, such as pastry, stains, dirtiness, bumps, etc. The distribution of number of instances for each class label is very uneven. Table 3.2 provides a summary of D_{SPF}.

[1] http://archive.ics.uci.edu/ml/.

3.1.4 *Restaurant Reviews Dataset*

The fourth dataset is called *Restaurant Reviews* dataset and is denoted as D_{RER}. It was collected from a website[2]. It is intended to determine from customer reviews whether a costumer will return to a reviewed restaurant or not. The data consists of 40 categorical attributes, including a binary class and 405 instances.

The data was collected using a software program that parsed through the Restaurantica website[3]. Five attributes are directly rated by customers and are accordingly named *rating* variables (food quality, service quality, atmosphere quality, price range, and party size). Variables, such as presentation, portion, consistency, variety, taste, etc. are *calculated variables* that were derived from the text reviews of the customers. These calculated variables suffered a major problem of having many missing values. Therefore, many of the variables were combined, such as presentation and portion, consistency and variety, etc. This has solved the majority of the missing-values problem and reduced the number of attributes to 21. The logical OR was used to combine the attributes. That is, the combined value of a number of attributes is the result of OR between all the attributes.

The distribution of number of instance for each class is uneven. Class 1 has 133 instances while class 2 has 272 instances. Class 1 represents a positive intention to return to the reviewed restaurant in the future and class 2 is the opposite.

3.2 Preparation

In order to prepare the data for experiments, we have performed data cleaning and filtering as well as attribute selection.

For each dataset, we have applied some preprocessing filters to better prepare the data

[2]This dataset was gathered by Taha Azizi, from University of Lethbridge.
[3]http://www.restaurantica.com.

for the classification task. Two filters were applied: nominal to binary filter and normalization and centering filter. The nominal to binary filter converts all nominal variables into binary variables. If a nominal variable consists of k values, and if the class is also nominal, this filter will transform the variable into k binary attributes.

Data normalization algorithms transform a given variable's range to another given range with minimal distortion to the data and its properties. The normalization filter we applied normalizes all variables' values in a given dataset, including the class, to values in the range [-1, +1]. As mentioned previously, the need for data normalization and centering stems from the use of sigmoid thresholding function in neural networks.

We have performed attribute selection and dimensionality reduction on our datasets to diversify them. We employ *Correlation-based Feature Selection (CFS)*, *ReliefF*, and *Principal Component Analysis (PCA)*, which will be discussed in the following sections.

3.2.1 Correlation-based Feature Selection

Correlation-based Feature Selection (CFS) evaluates variable subsets rather than single variables [23]. The intuition behind this method is that a prediction task of some target class is more accurate if it is based on a variety of characteristics rather than a single characteristic. The traits that this method utilizes are the correlation with the target class and the inter-correlations of variables.

A selected subset of variables must be highly correlated with the target and least correlated with each other. A variable that is highly correlated with a class is strongly indicative of it, while a variable that is highly correlated with another variable is redundant. CFS uses a heuristic for evaluating the usefulness a subset of variables. This heuristic encodes the correlation of a subset of variables with a class label along with the level of inter-correlation among the variables of the subset. CFS uses this heuristic to guide its search for a good

variable subset.

CFS selects a subset of variables based on global measures which does not take into account *locally predictive* variables [23]. Applying CFS to a dataset with variables of such predictive characteristics will certainly diminish the prediction task of this dataset.

When CFS was applied to D_{AGA} the number of attributes was reduced from 17 to 8 and reduced the number of attributes of D_{SPF} from 27 to 12. Moreover, when CFS was applied to D_{RER} the number of attributes was reduced from 21 to 13 and when applied to D_{SYN} the number of attributes was reduced from 17 to 9.

3.2.2 Principal Component Analysis

Principal Component Analysis (PCA) is a multivariate analysis technique that utilizes orthogonal transformation to transform a set of inter-correlated attributes into a new smaller dataset of independent (i.e., orthogonal) attributes (i.e., *principal components*) that retain most of the original dataset properties [2]. The principal components are defined as a linear combination of properly weighted attributes. The first principal component accounts for the largest amount of variance in the data while the succeeding components have progressively less variance and are all uncorrelated with the preceding components.

The PCA process starts by creating a covariance matrix of the given data and deriving the eigenvectors and eigenvalues of this matrix. After this step, the eigenvectors are sorted based on their corresponding eigenvalues in descending order. Then, a number of eigenvectors are chosen such that the accumulated sum of their corresponding eigenvalues amounts to some predetermined percentage of the total sum. The number of principal components is determined by the number of eigenvectors chosen. The data is then projected into eigenspace (sometimes it is said projected into PCA space) which produces transformed non-correlated principal components.

When PCA was applied to D_{AGA} the new transformed data had 17 principal components which match the number of the attributes in the original set and when applied to D_{SPF} the number of principal components became 14 attributes.

Some strange results were seen when PCA was applied to D_{RER} and D_{SYN}. In the first, the number of principal components became 56 as opposed to 21 attributes in the original dataset and in the latter the number of principal components became 31 as opposed to 17 in the original dataset. We speculate that this increase in the number of principal components is due to the rather close covariance between the different attributes in the two datasets which causes the produced eigenvalues to be relatively small and close in value.

3.2.3 ReliefF

ReliefF is a statistical weight-based variable selection algorithm which was inspired by instance-based learning [4]. For a given dataset D composed of I instances, a dimensionality m, and a threshold of relevance $(0 \leq \tau \leq 1)$, ReliefF selects variables that are relevant to the target class [32, 33, 37]. The intuition behind this method is that a selected variable should contain instances that are distinguishable from others of a different class while those of the same class should have relatively close values.

The way ReliefF works is by picking a random sample composed of m triplets (a triplet for each variable): an instance x_i, its *near-hit* instance and *near-miss* instance. An instance is *near-hit* if it belongs to the same class as x_i and is in close proximity distance to x_i. An instance is *near-miss* if it does not belong to the same class as x_i but is in close proximity distance to x_i. The distances between instances are typically determined using p-dimensional Euclid distance. For each triplet k in a given sample x_i, ReliefF updates a weight vector that is associated with it then uses the average weight of each variable to determined the *relevance* level of the variable. If the relevance level exceeds the given

threshold level τ, the variable is selected.

ReliefF is applicable for all types of classification problems (binary class, multiple class [32, 33] and for continuous class [37]) and is relatively fast. It suffers from being unable to detect and remove redundant variables. A variable is selected as long as its relevance level is large enough. Moreover, insufficient number of training instances causes ReliefF to make mistakes regarding the acceptable proximity distance for a certain class and therefore may mistakenly detect near-misses and near-hits [33, 59].

When ReliefF was applied to D_{AGA} the number of attributes was reduced from 17 to 7 and when applied to D_{SPF} the number of attributes was reduced from 27 to 22. Also, when ReliefF was applied to D_{RER} the number of attributes was reduced from 21 to 13 and when applied to D_{SYN} the number of attributes was reduced from 17 to 4.

Chapter 4

Paralleling Neural Network Ensemble[1]

The ensemble approach is a method used to improve classification for a given task by combining different classifiers via some mechanisms. This approach was discussed in Chapter 2.

The typical ensemble approach uses various heterogeneous classifiers and a probabilistic gating function (i.e., a function that combines different classifiers). In this chapter, we propose to combine structurally different homogeneous classifiers (i.e., neural networks) via a neural network gating function, as will be shown in Section 4.1. It is possible to create an ensemble for a specific dataset by varying the structures of the neural networks involved in the ensemble, i.e., creating them on different parts of the dataset. It was shown in [25, 44], among others, that the best ensemble should contain members that make their errors on different parts of an input space.

4.1 Approach Formulation

In our approach, we propose to use a neural network, called *ensemble network*, to combine a set of individual neural networks, called *ensemble members*, taking advantage of the classification power of each individual network. This approach has two variations, *pure paralleling neural network ensemble*, as shown in Figure 4.1 (a) and *enhanced paralleling neural network ensemble*, as shown in Figure 4.1 (b).

The first variation seeks to improve the classification of neural networks by recording the predictions of structurally differing neural networks for each instance of a given dataset to produce a new dataset. After this, a new neural network is trained on this new dataset

[1] A preliminary version of this chapter is in [62].

in the hope of improving classification accuracy. The second variation is similar to the first but with the addition of the attributes of the original dataset along with the predictions from individual neural networks. Both approaches will be demonstrated and discussed in the following sections.

In our approach, a given dataset $D_{original}$, that consists of a set of attributes $X_{original}$ and a set of class labels Y, undergoes attribute selection, targeting specific characteristics of the data and producing J subsets. For each of the J subsets, a neural network is constructed. A neural network NN_j is trained on the j^{th} subset using an n-fold cross-validation process producing a set of predictions, P_j, for instances of the subset. After attaining all predictions, a new dataset, $D_{ensemble}$, is created by combining the predictions of all J neural networks. That is, $D_{ensemble}$ consists of $X_{ensemble} = Y \cup_j P_j$. For the enhanced approach, a simple change is made on the new dataset by combining the predictions of all J networks as well as the attributes of the original data $X_{original}$. That is, $X_{ensemble} = Y \cup X_{original} \cup_j P_j$.

The intuition behind our approach is that individual neural networks constructed using a subset of the input dataset, which shares different properties from others, represent different views on the data from various perspectives. Therefore, combining them would hopefully yield better classification performance. In addition, we propose to include the original data as part of the input to the ensemble network to further enhance classification accuracy. As will be shown in Section 4.3, such inclusion will further increase the ensemble's classification accuracy.

4.2 Experiment Setup

Neural networks can be varied structurally or logically. Two networks are structurally different when they have different number of inputs or different number of hidden layers and neurons. Two networks are logically different when they have the same structure but use

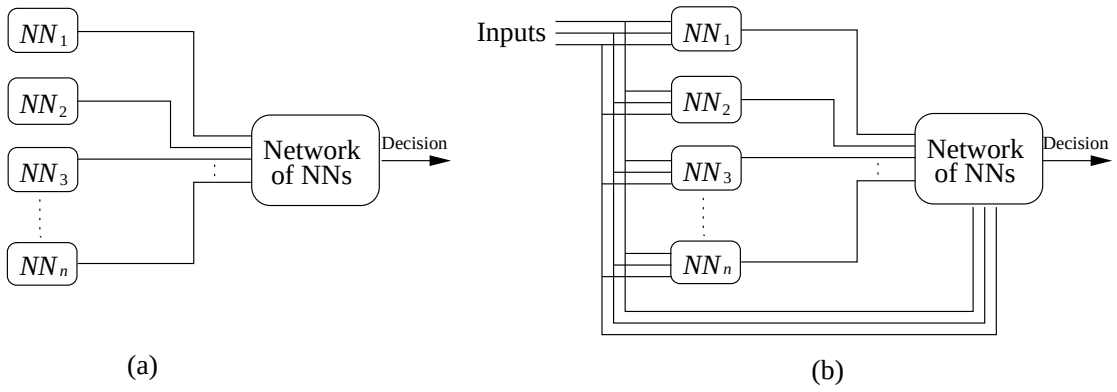


Figure 4.1: The pure paralleling approach vs. enhanced paralleling approach.

different thresholding functions, or are trained using different learning rates, momenta, etc. As for the purpose of this chapter, we create ensemble members that are structurally different in terms of their number of inputs. There are many ways in which the data can be varied in order to produce structurally different neural networks, such as *sampling*, *disjoint training sets*, *preprocessing*, etc. [51]. In our work, we focus more on preprocessing techniques, namely PCA, CFS, and ReliefF, as discussed in Chapter 3, to create subsets of the original datasets that consider four different characteristics, i.e., correlation between attributes, correlation between attributes and classes, distinguishability of an attribute's values across different classes, and similarity of an attribute's values within the same class. For each of the new subsets and the original datasets, a member network is created. A trial-and-error process is conducted using different number of hidden layers, hidden neurons, and using different learning rates, epochs, and momentum in order to determine the "best" network.

For the pure paralleling neural network ensemble in Figure 4.1 (a), once the set of "best" member networks are determined, their classifications will be used as inputs, along with the classes to be predicted, to the ensemble network. The enhanced paralleling neural network ensemble differs in that, in addition to the inputs mentioned before, some subsets of the original data, i.e., the ones derived from PCA, CFS, ReliefF or the entire original dataset, will also be used as additional inputs. This situation is shown in Figure 4.1 (b).

Table 4.1: Results summary (MAE (epoch, learning rate, momentum)).

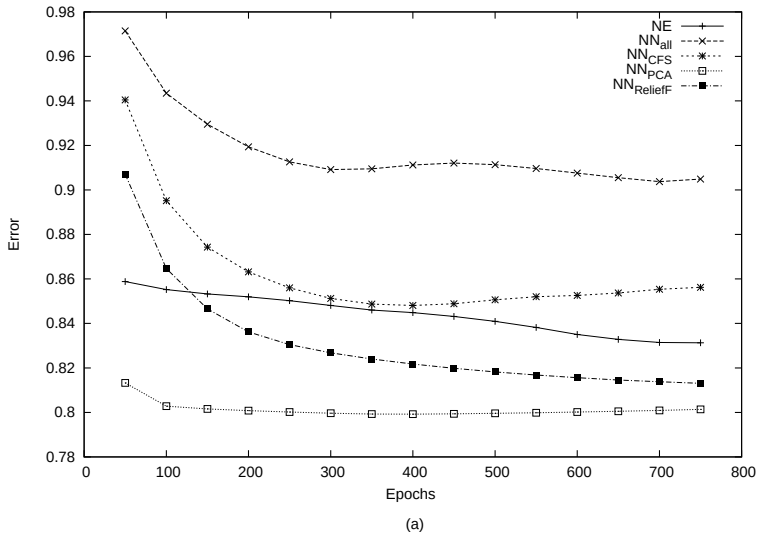
	D_{AGA}	D_{SPF}	D_{SYN}	D_{RER}
NN_{all}	.9037 (700, .05, .2)	**.0829** (750, .20, .1)	4.4498 (50, .05, .1)	.0657 (100, .2, .1)
NN_{cfs}	.8481 (400, .05, .3)	.0996 (750, .20, .3)	4.4416 (50, .05, .1)	.0715 (50, .2, .3)
NN_{pca}	.7992 (400, .05, .1)	.0943 (750, .15, .3)	4.4680 (50, .05, .1)	.0866 (100, .1, .3)
$NN_{reliefF}$	.8131 (750, .05, .3)	.0848 (750, .15, .3)	4.4382 (200, .05, .1)	.0600 (100, .2, .3)
NE	.8313 (750, .05, .3)	.0937 (750, .15, .3)	4.5488 (750, .05, .1)	.0921 (750, .2, .3)
NE_{all}	.7171 (750, .05, .2)	.0852 (550, .10, .2)	4.4479 (50, .05, .1)	**.0581** (400, .2, .1)
NE_{cfs}	.8493 (550, .05, .1)	.0830 (700, .10, .2)	4.4408 (50, .05, .1)	.0630 (150, .15, .2)
NE_{pca}	**.6384** (750, .05, .1)	.0850 (700, .10, .2)	4.4512 (50, .05, .1)	.0689 (600, .15, .3)
$NE_{reliefF}$	.7022 (750, .05, .1)	.0867 (700, .10, .3)	**4.4336** (50, .05, .1)	.0602 (750, .05, .3)

Section 4.3 will show a detailed comparative study of the performances of these neural ensembles.

As aforementioned, in order to generate the best performing network for each of the neural networks, a trial-and-error process is conducted to determine the structures of the networks that reduce the error and that are the most computationally efficient. Table 4.1 shows the properties of the best performing networks for each dataset, where NN_* is a neural network constructed on any of the subsets, i.e., All attributes, CFS attributes, PCA attributes, or ReliefF attributes, of a given dataset, NE is the pure paralleling neural network ensemble, and NE_* is the enhanced paralleling neural network ensemble that includes any of the subsets of a given dataset, i.e., ALL, CFS, PCA, or ReliefF.

In our experiments, we have found that a structure of a single hidden layer with (the number of classes + the number of attributes) neurons performed the best, while the learning rate, momentum, and learning iterations (epochs) were dataset specific. Therefore, a comprehensive set of experiments were performed. A 10-fold cross validation was used to evaluate the performance. Our proposed approach was implemented on top of Weka[2]. All the experiments were run on a network of computers controlled by Condor High Through-

[2]http://www.cs.waikato.ac.nz/ml/weka/.



(a)

Figure 4.2: Classification on D_{AGA}'s subsets using typical neural networks vs. pure paralleling approach.

put Computing[3].

4.3 Results

We ran our experiments using all four datasets and their subsets for both variations of our paralleling ensemble, pure and enhanced. In our experiments, we record the errors as the number of epochs increase. This establishes a comparison between the various networks we explored.

Figures 4.2, 4.4, 4.6, and 4.8 demonstrate the performances of NE, i.e., without original dataset, and the individual ensemble members' performance as the number of epochs increases. Figures 4.3, 4.5, 4.7, and 4.9 demonstrate the performances of NE and the ensemble networks, NE_{all}, NE_{PCA}, NE_{CFS}, and $NE_{ReliefF}$.

In general, from Figures 4.2, 4.4, 4.6, and 4.8 we see that NE produces an average error in comparison with the other ensemble members, while Figures 4.3, 4.5, 4.7, and 4.9

[3]For more information, see http://www.cs.wisc.edu/condor/.

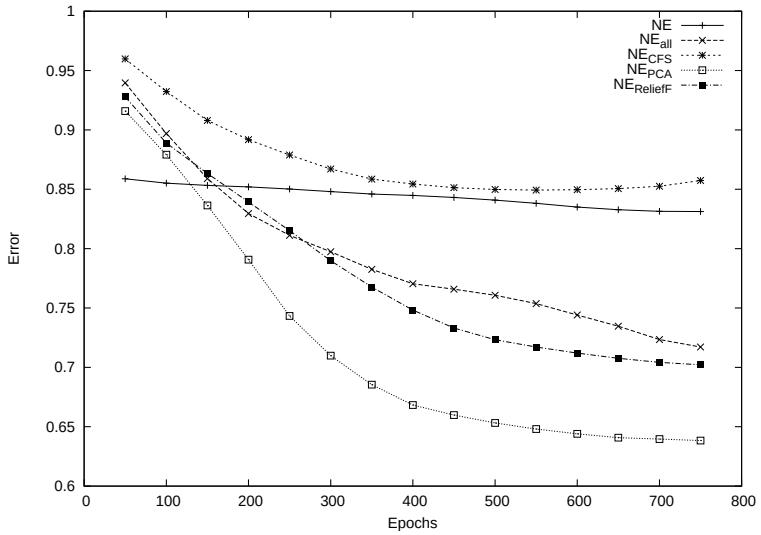


Figure 4.3: Classification on D_{AGA}'s subsets using enhanced paralleling approaches vs. pure paralleling approach.

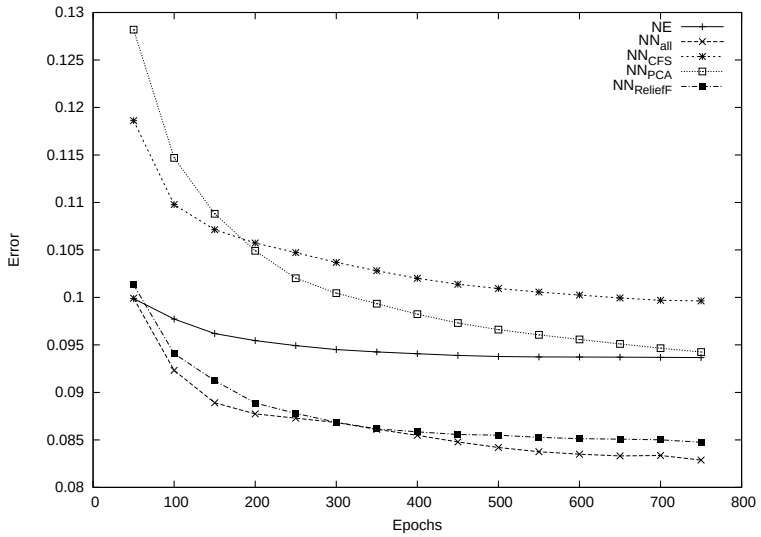


Figure 4.4: Classification on D_{SPF}'s subsets using typical neural networks vs. pure paralleling approach.

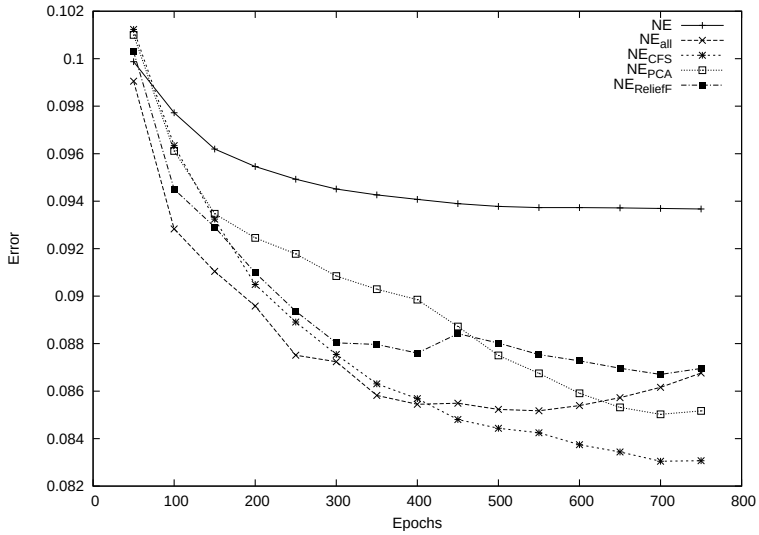


Figure 4.5: Classification on D_{SPF}'s subsets using enhanced paralleling approaches vs. pure paralleling approach.

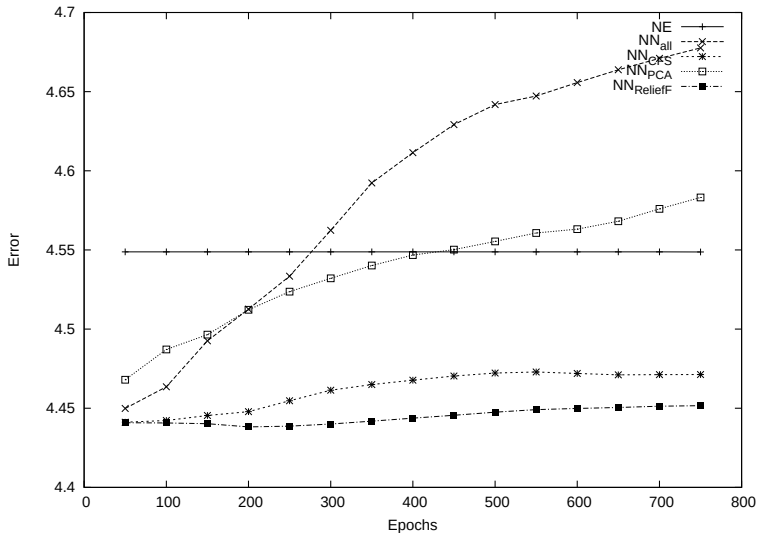


Figure 4.6: Classification on D_{SYN}'s subsets using typical neural networks vs. pure paralleling approach.

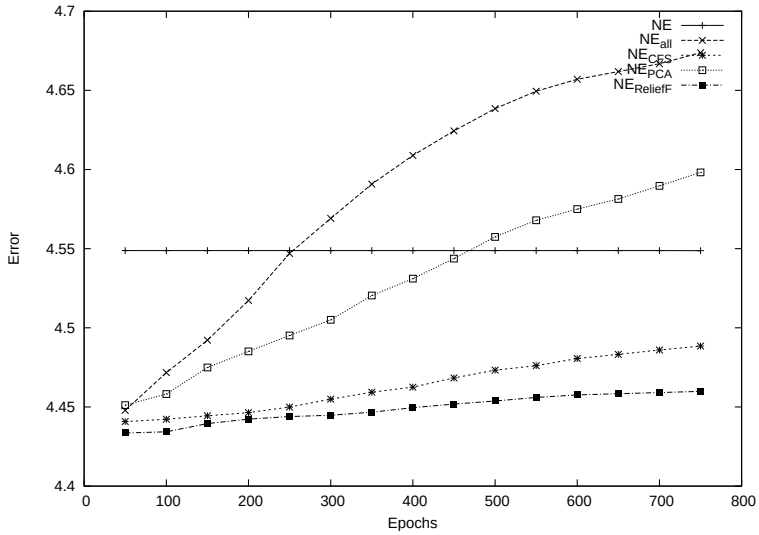


Figure 4.7: Classification on D_{SYN}'s subsets using enhanced paralleling approaches vs. pure paralleling approach.

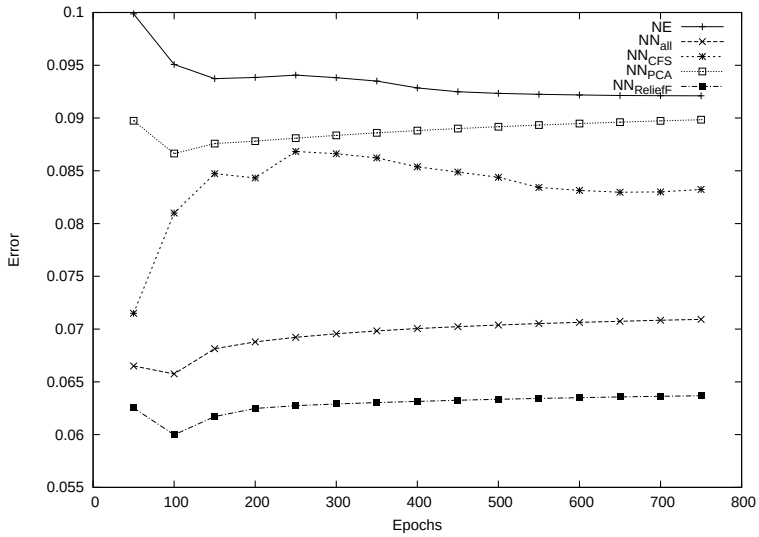


Figure 4.8: Classification on D_{RER}'s subsets using typical neural networks vs. pure paralleling approach.

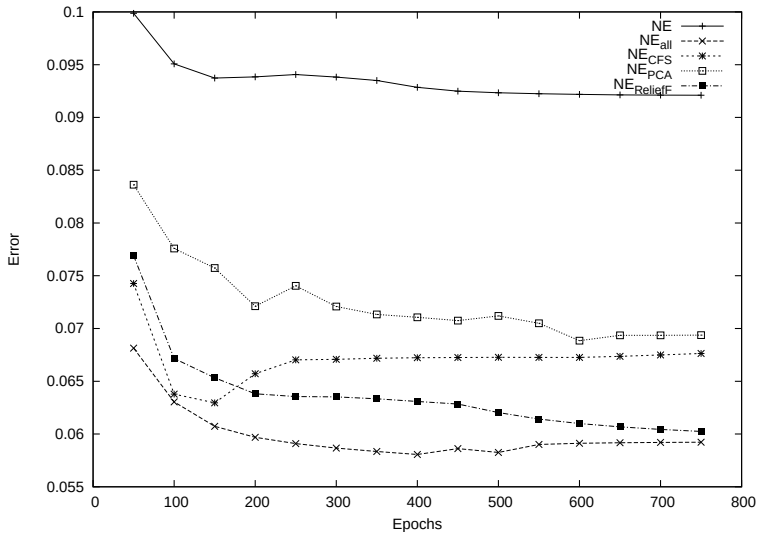


Figure 4.9: Classification on D_{RER}'s subsets using enhanced paralleling approaches vs. pure paralleling approach.

show that all the ensemble networks with the additional dataset inputs, namely NE_* have outperformed NE. We will discuss this in detail in the following section.

4.4 Analysis and Discussions

In Figure 4.2 we see that $NN_{ReliefF}$ and NN_{PCA} have produced the smallest errors among the others. Figure 4.3 shows that when each of these two subsets was included as inputs into the ensemble network, each has produced the best ensemble network.

A surprising finding is in the behavior of NN_{all} in Figure 4.2. We can see that NN_{all} produce by far the largest error among the other networks. However, when inspecting Figure 4.3 we see that NE_{all} has produced a small error almost identical to that of $NE_{ReliefF}$. This could mean that choosing ensemble members based on error may not be the best way or it could be an exceptional behavior. Further experimentation and analysis is required to investigate this observation.

Interestingly, in Figures 4.4 and 4.5 we see the same behavior as that for NN_{all} in D_{AGA}.

In these figures, we see that NN_{CFS} and NN_{PCA} performed the worst but NE_{CFS} and NE_{PCA} performed the best in an ensemble setting. This could be due to the training parameters of the neural networks. It may also be the case that more epochs are needed or different learning rate or momentum values.

In Figure 4.8 we observe the same situation as for D_{AGA}, where the best two ensemble member networks NN_{all} and $NN_{ReliefF}$ have performed the best, while, as shown in Figure 4.9, it also has the same situation where they were used as inputs in $NE_{ReliefF}$ and NE_{all}.

In Figure 4.6 we see that NN_{PCA}, NN_{all}, NN_{CFS}, and $NN_{ReliefF}$ are ranked from the lowest classification error to the highest and the same order is also shown in Figure 4.7 when they are used as inputs to the ensemble network, i.e., NE_{PCA}, NE_{all}, NE_{CFS}, and $NE_{ReliefF}$.

When examining Figures 4.3, 4.5, 4.7, 4.9, we see that NE is inferior to at least one of NE_{all}, $NE_{ReliefF}$, NE_{PCA}, or NE_{CFS}. This situation is especially true for the comparison between them on the two datasets D_{SPF} and D_{RER}, where the performance of NE is the worst.

We also notice that, on the D_{SYN} dataset in Figure 4.7, all the ensemble networks performed worse when the number of epochs increased. We attribute this to the randomness we introduced when we created the dataset.

In D_{AGA} dataset, as shown in Figure 4.3 we observe an exception in the performance of NE. In this figure, NE_{CFS} has been slightly outperformed by NE. This observation might be due to the limited number of instances in the dataset, as shown in Table 3.1. We expect to collect more instances for that dataset and would like to see whether the same observation exists.

As for the comparison between NE and the ensemble members, to our surprise, in most of our experiments, NE resulted in an average error. However, Figure 4.8 shows that NE

performed the worst than all other ensemble members. This could possibly be due to the relative small number of neural ensemble members used by NE.

In addition, when comparing NE and NE_*, it can be found that for most situations, NE_* need a small number of epochs for the ensemble network to converge, as evident in Table 4.1. The results in the table show that including the original data as inputs to the ensemble network will be computationally feasible. However, we need to investigate more along this direction as whether this is true in general.

When exploring more in Figures 4.2 and 4.3, we can clearly see that there are interesting observations. At the start of the training process in Figure 4.2, almost all the ensemble members produce relatively large errors and then the errors reduce with the increase of the number of epochs to a certain point until they fluctuate within the acceptable ranges. But we observe that NE's error reduces slightly with the increase of the number of epochs then remains constant or slightly fluctuates during the majority of the training epochs.

On the other hand, as in Figure 4.3 all NE_* show drastic error drops after some initial epochs, a healthy trend in the training process. This situation is not observed for the individual ensemble members, as contrasted in Figure 4.2. This demonstrates that our NE_* clearly outperform individual ensemble members in most cases in terms of errors. To be more specific, as shown in Figure 4.3, NE_{PCA} produces the smallest error.

In Figures 4.8 and 4.9 it becomes evident that all the networks, including NN_* and NE_*, are fairly accurate and the errors do not reduce during the training. Rather, NN_* in Figure 4.8 become overfit when large numbers of epochs are used. But NE_* do not have this overfitting problem [50], as in Figure 4.9. Although there is some gap in the final errors (approximately 0.036), they all exhibit the same behavior. That is, all of them produce stable and acceptable errors with the increase of the number of epochs.

In Figure 4.8, NE performes the worst. However, this is not the case when examining Figures 4.2, 4.6, and 4.4. While it is surprising to see this, we conjecture that it may be due

to the binary class labels in the dataset. This could mean that our proposed paralleling neural ensemble approach is better to handle classifications of classes with continuous values or a large set of class labels. However, whether this is true or not in general needs more investigation.

Furthermore, in Figure 4.9 we find that *NE* has the largest error amongst the ensemble networks. This is expected, since with additional inputs we hope that the classification accuracy of the ensemble network would be further increased.

4.5 Summary

We have proposed a novel approach that aims to improve classification by combining structurally different neural networks via another neural network. This approach has two variations, namely pure paralleling neural network ensemble and enhanced paralleling neural network ensemble. The first variation combines only the predictions of neural networks while the second variation includes the original data or one of its subsets as well. This approach was demonstrated to improve classification in most of the datasets with a very few exceptions.

Chapter 5

Chaining Neural Network Ensemble

In this chapter, we propose our second ensemble approach, namely *chaining neural network ensemble*. Chaining neural network ensemble, or *neural chaining* for short, attempts to improve a neural network's prediction by including the predictions of previously trained network(s) into the training process of new network(s), forming a chain-like ensemble. This approach has two variations, *single-link chaining (SLC)* and *multi-link chaining (MLC)*. The SLC approach, as shown in Figure 5.1, trains a neural network with a given dataset and then uses the predictions of this network as input to the next network along with the original data. Each network in the "chain" is trained on the original dataset and on the predictions of the network that immediately precedes it. The chaining process continues until an acceptable error is achieved. The MLC approach, as shown in Figure 5.2, is similar to the SLC approach. They differ in that each neural network in MLC is trained on the original dataset and on the predictions of all networks that precede it.

Both variations undergo the same two-step process, chain link selection and training. In the first step, the process of training and appending predictions takes place using restricted number of parameters, such as learning rates and and number of epochs. This is done such that the computations remain feasible. However, considering the relevant small amount of computational power we possess, restricting the parameters seems reasonable. Moreover, it was noticed that when the parameters' ranges were expanded, lower errors where achieved. However, the patterns of error reductions were almost identical. After determining the best performing number of chain link, the produced dataset was used to construct a neural network using large ranges for the parameters. In Section 5.1, more detail will be given on this process.

5.1 Approach Formulation

For a given dataset D_0 that consists of a set of attributes X_0 and a set of class labels Y. A neural network NN_0 is trained on D_0 and validated using an n-fold cross-validation process producing a set of predictions, P_0. This network is a typical neural network trained using D_0, i.e., no network predictions used. Then, a new dataset is created by appending the predictions P_0 with X_0 to produce a new dataset D_1. A new neural network NN_1 is trained and evaluated on D_1. The difference between SLC approach and the MLC approach occurs after the predictions of NN_1 are attained.

In the SLC approach, the predictions of NN_1 replace the predictions of NN_0 in D_1. This creates a new dataset D_2 in which a new neural network NN_2 is constructed. Repeating this process produces a chain of neural networks, as shown in Figure 5.1. This approach increases the number of attributes in the original dataset by only one, keeping computational cost of creating a new network feasible. On the other hand, the MLC approach, shown in Figure 5.2, appends the predictions of NN_1 to D_1 alongside the predictions of NN_0, producing a new dataset D_2 with one more attribute than D_1. The process is then repeated. The predictions of each trained network are appended to the dataset, increasing its number of attributes by one, along with each added set of network predictions (i.e., the number of chain links) in the chain. This increases the number of attributes of the dataset, causing the creation of new neural networks to become computationally expensive.

One concern arises in the SLC and MLC approaches regarding the chain links. The concern is to determine which chain link or group of chain links is best at reducing the overall error. It is difficult to form an intuition regarding this problem. We believe that it is an optimization problem and requires further investigation. Therefore, we have decided to use 20 chain links for our experiments.

The intuition behind both of our approaches is that a neural network's predictions can

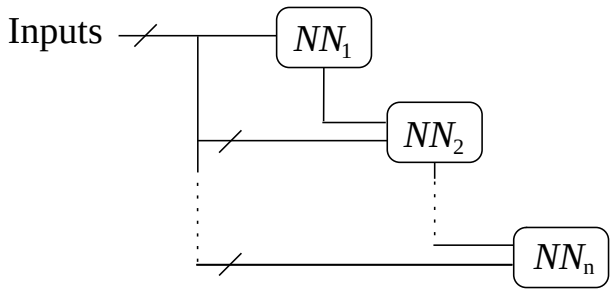


Figure 5.1: Single-Link Chaining.

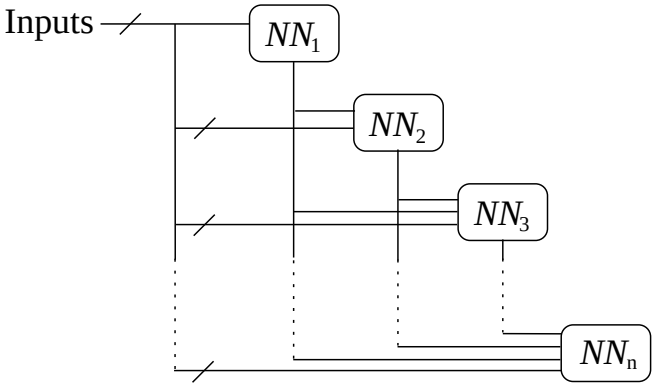


Figure 5.2: Multi-Link Chaining.

be used to correct the predictions of upcoming networks. Given a trained network, its predictions are highly correlated with and strongly indicative of the target classes, since the attributes of the dataset are indicative of the target class and are used to produce these predictions. Therefore, using these predictions is expected to further improve the predictability of the data.

In particular, the intuition behind the SLC approach is that a network in the chain may not need the predictions of all preceding networks in order to correct its classification. A network trained on the predictions of a previous network can produce predictions influenced by that information. Therefore, it seems reasonable that the predictions of the new network should replace that of the previous network in the dataset, thus, avoiding an unnecessary increase in calculations. The intuition behind the MLC approach is that it may be necessary for a network to have access to all the predictions of previous networks. This

way, it is left up to the training procedure to learn what it finds beneficial. These hypotheses will be verified in the following experiments.

5.2 Experiment Setup

As mentioned in Chapter 3, we derived from each of the four datasets: D_{AGA}, D_{SPF}, D_{SYN}, and D_{RER}, three more datasets that differ from the original using the methods discussed previously in Chapter 3, i.e., CFS, PCA, and ReliefF. These methods target some specific properties, such as inter-correlations, correlation with the target class, distinguishability of attributes, etc. These produced datasets along with the original ones will assist us in understanding and interpreting the performance of our chaining neural network ensemble approach.

In our experiments, we use the same neural network structure, computing software, and computing architecture as those in Chapter 4. For each of the new datasets and the original dataset, two chains of neural networks are created, an SLC-style chain, denoted SLC_s, and an MLC-style chain, denoted MLC_s. Due to the high computational expense of determining the best parameters and structures for each added neural network, the structure and parameters determination was left to be determined automatically by Weka, thus eliminating the need of performing parameter sweeps for chain link selection.

After the chaining process is conducted and the best number of links in the chain are determined, a large scale training process was performed, for each chain and typical neural network, over different number of hidden layers and hidden neurons, using different learning rates and momenta, establishing a comparison between them, as will be demonstrated shortly.

5.3 Results

Figures 5.3 and 5.4 illustrate the impact of added chain links on the D_{AGA} chain's error. That is, these figures give a graphical representation of the performance of each of the chains and, therefore, serve as a guide to determine when to stop the chaining process, i.e., selecting the best number of chain links. Demonstrating the behavior of the chain links selection process is not essential to our results and discussions. Therefore, the figures of the other datasets have been omitted. After the best numbers of chain links have been determined, the entire chain gets trained and its performance is compared against a typical individual neural network.

Figures 5.5, 5.6, 5.7, 5.8, 5.9, 5.10 5.11, and 5.12 illustrate the impact of the number epochs on the errors of SLC_s, MLC_s, and typical individual neural networks. These figures show the performance of the networks after their structures have been determined.

Tables 5.1, 5.2, 5.3, and 5.4 establish a comparison between SLC_s and MLC_s performance for each dataset and its subsets. It shows the chain link that has reduced error by the most and highlights the lowest error value achieved in comparison between SLC_s and MLC_s.

5.4 Analysis and Discussions

When examining Figure 5.3 of SLC_s chaining results, it becomes immediately evident the strong oscillation behavior of the chains, although an error reduction occurs. On the other hand, MLC_s chaining results in Figure 5.4 show slightly less oscillation behavior and shows error reduction. This oscillation behavior, although unanticipated and not preferred, has no significant impact on the chain selection since our goal from these experiments is to select the best number of chaining links for SLC_s and MLC_s, regardless of computational expense.

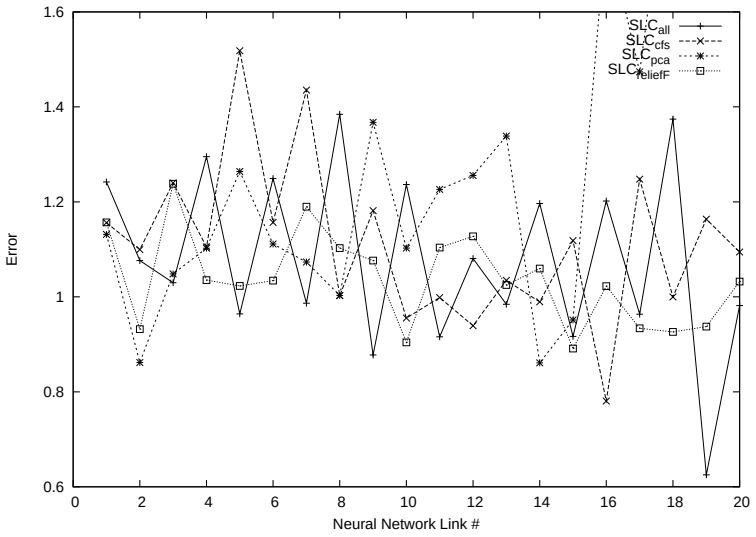


Figure 5.3: Classification on D_{AGA} and its derived subsets PCA, CFS and ReliefF using SLC.

However, further analysis is required to understand the reasons of this observation.

5.4.1 D_{AGA} Analysis

When inspecting Figures 5.5 and 5.6 it is easy to notice that both SLC_s and MLC_s significantly outperformed their corresponding typical neural networks and have done so early in the learning process. Moreover, most of the SLC_s chains and all of the MLC_s chains outperformed all typical neural networks.

It is interesting to note that the chain that reduced error the most in the chain link selection process did not necessarily reduce error the most after training. In the case of SLC_s, for example, SLC_{all} has reduced the largest error followed by SLC_{cfs}, SLC_{pca}, and $SLC_{reliefF}$, respectively, as demonstrated in Figure 5.3 and Table 5.1. However, when cross-referencing the performance of each of the chains with Figure 5.5, we see that SLC_{cfs} has, by far, reduced the largest error followed by SLC_{pca}, $SLC_{reliefF}$, and SLC_{all}. It is rather interesting that the best performing chain in the chain link selection stage has performed

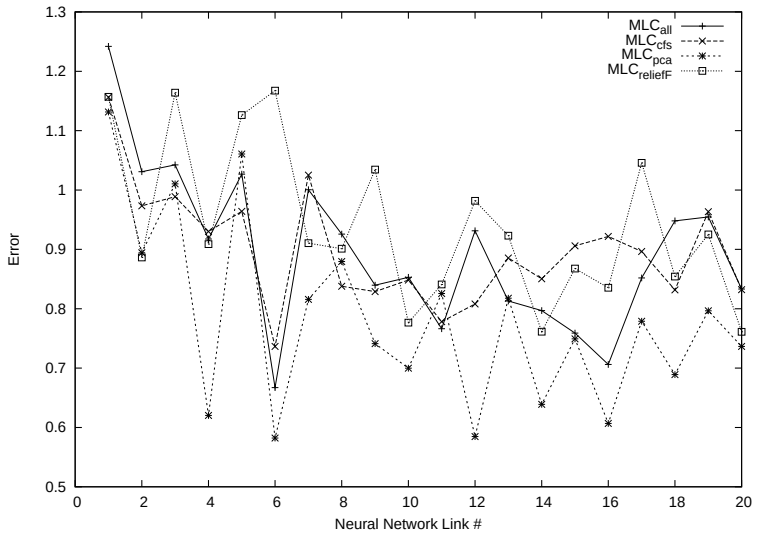


Figure 5.4: Classification on D_{AGA} and its derived subsets PCA, CFS and ReliefF using MLC.

Table 5.1: D_{AGA} chaining results.

D_{AGA}		all	cfs	pca	reliefF
SLC	Chain Link #	19	16	14	15
	MAE	**0.6252**	0.7806	0.8611	0.8911
MLC	Chain Link #	6	6	6	20
	MAE	0.6672	**0.7366**	**0.5821**	**0.7611**

the worst in the training process.

This situation was also evident in MLC_s. The lowest error among the chains was achieved by MLC_{pca} followed by MLC_{all}, MLC_{cfs}, and $MLC_{reliefF}$, as shown in Figure 5.4. However, when each of the chains was trained, the best performing chain was MLC_{pca} followed by MLC_{cfs}, $MLC_{reliefF}$, and MLC_{all}, as shown in Figure 5.6. This could be due to the fact that the parameters used in training these chains were not sufficient to demonstrate the power of these networks.

When inspecting Table 5.5, we see that the MLC approach has outperformed the SLC approach in three of the four chains, namely MLC_{all}, MLC_{pca}, and $MLC_{reliefF}$ with considerable differences while the SLC approach has outperformed the MLC approach in one

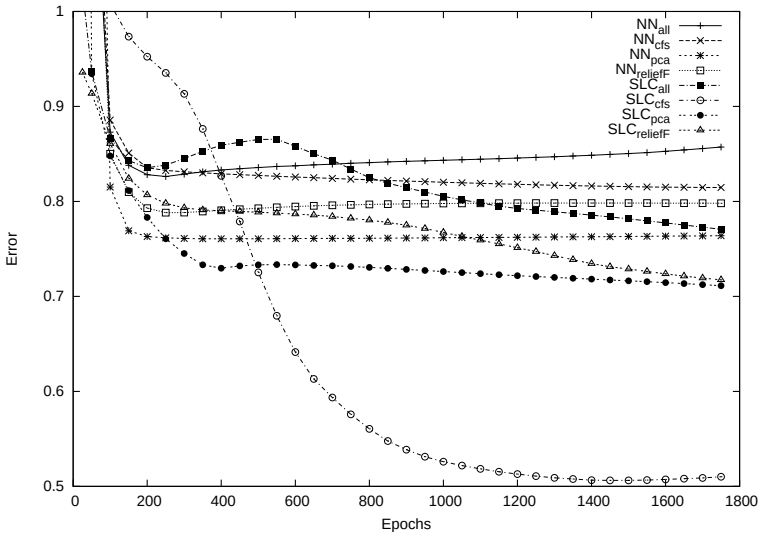


Figure 5.5: Error response to epoch in D_{AGA} SLC networks vs. regular neural networks.

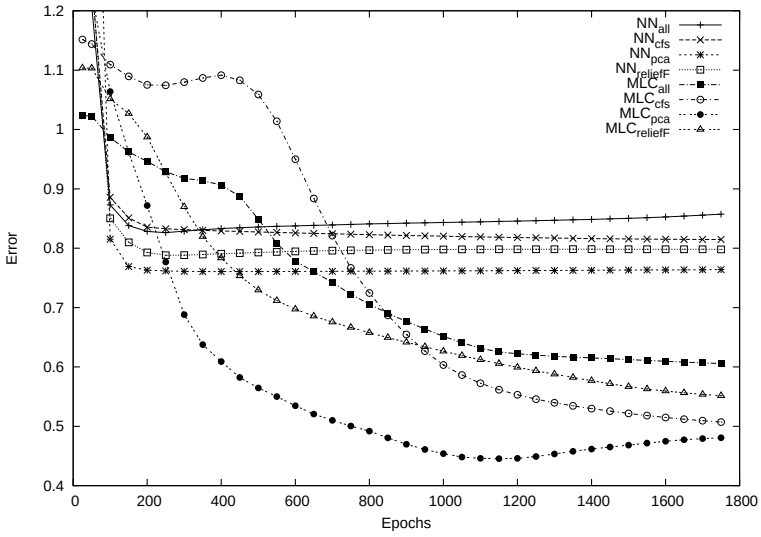


Figure 5.6: Error response to epoch in D_{AGA} MLC networks vs. regular neural networks.

Table 5.2: D_{SPF} chaining results.

D_{SPF}		all	cfs	pca	reliefF
SLC	Chain Link #	10	7	19	4
	MAE	**0.0858**	0.0968	**0.0913**	0.0856
MLC	Chain Link #	7	4	2	5
	MAE	0.0887	**0.0932**	0.0943	**0.0843**

subset, SLC_{cfs}, with a marginal difference.

5.4.2 D_{SPF} Analysis

Inspecting Figures 5.7 and 5.8 shows that the performance of SLC_s and MLC_s on this dataset varies. In Figure 5.7 three of the four SLC_s chains, i.e., SLC_{cfs}, SLC_{pca}, and $SLC_{reliefF}$ have slightly outperformed their corresponding typical neural network. Generally, NN_{all} has outperformed all SLC_s chains. This evident in Figure 5.7 and Table 5.5.

Figure 5.8 shows that two MLC_s chains, MLC_{cfs} and MLC_{pca}, have slightly outperformed their corresponding typical neural networks but were significantly outperformed by NN_{all} and slightly outperformed by $NN_{reliefF}$. NN_{all} outperformed all MLC_s chains.

From Tables 5.2 and 5.5 we see that the order of performance in SLC chain link selection stage was the same as that in the training. This was not the case in MLC since MLC_{all}, MLC_{cfs}, and MLC_{pca} came second, third, and fourth in terms of error reduction but in the training stage, MLC_{pca} became second, followed by MLC_{all} then MLC_{cfs}.

As shown in Table 5.5 MLC performed better than SLC in MLC_{cfs} and MLC_{pca} and worse than SLC in SLC_{all} and $SLC_{reliefF}$.

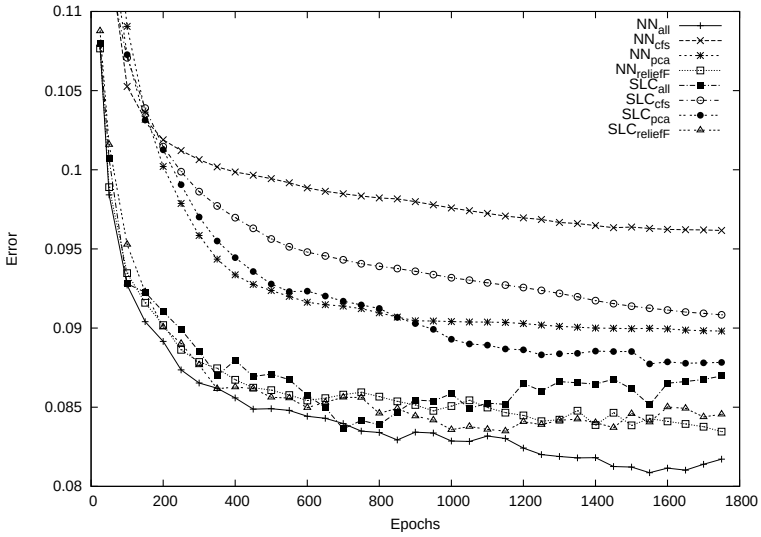


Figure 5.7: Error response to epoch in D_{SPF} SLC networks vs. regular neural networks.

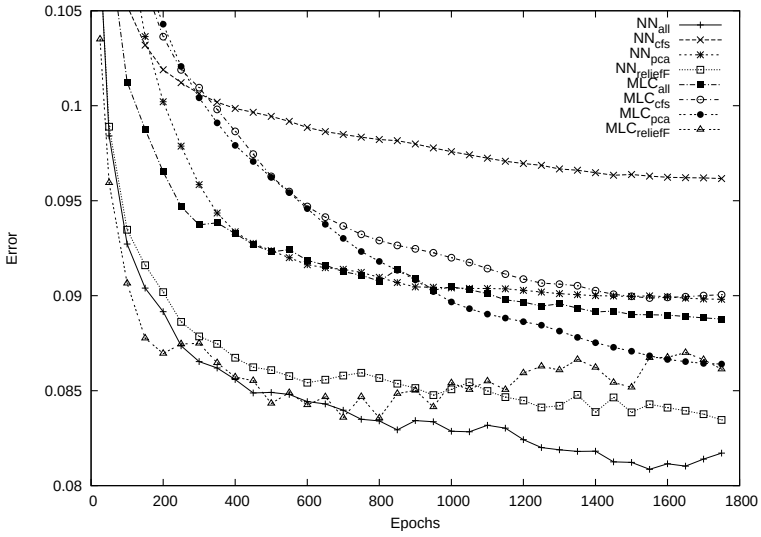


Figure 5.8: Error response to epoch in D_{SPF} MLC networks vs. regular neural networks.

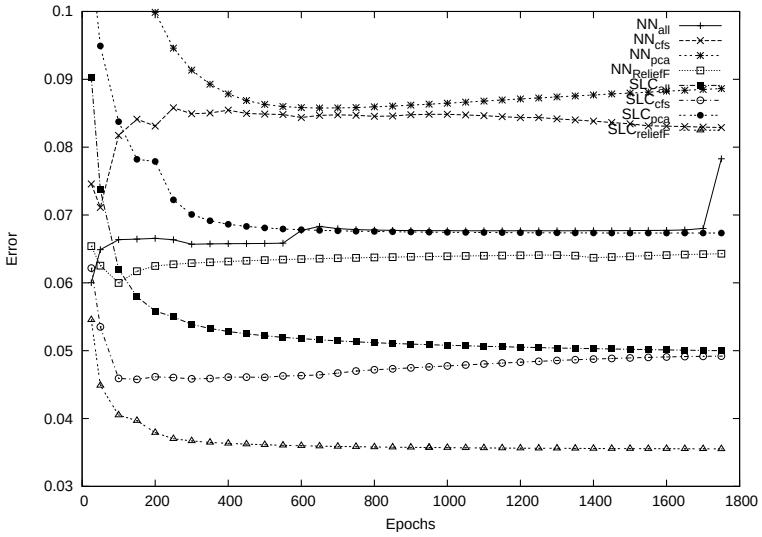


Figure 5.9: Error response to epoch in D_{RER} SLC networks vs. regular neural networks.

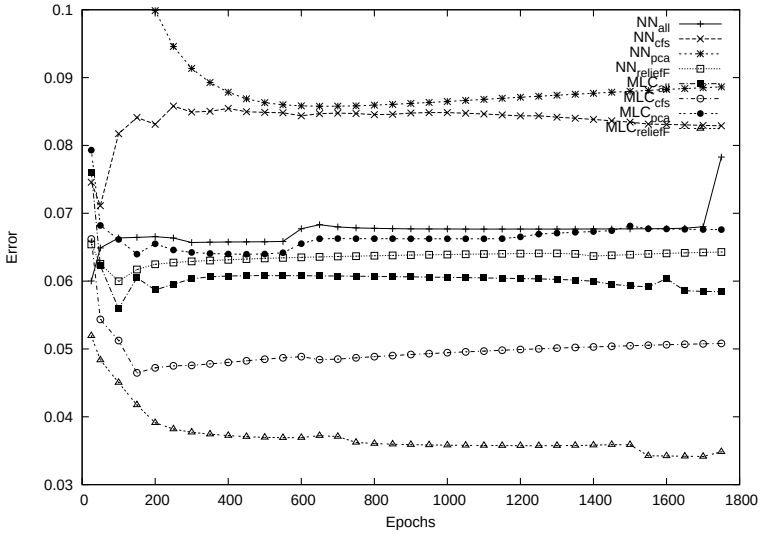


Figure 5.10: Error response to epoch in D_{RER} MLC networks vs. regular neural networks.

Table 5.3: D_{RER} chaining results.

D_{RER}		all	cfs	pca	reliefF
SLC	Chain Link #	19	2	13	16
	MAE	**0.0537**	**0.0461**	0.0716	0.0367
MLC	Chain Link #	10	7	2	10
	MAE	0.0562	0.0469	**0.0640**	**0.0387**

5.4.3 D_{RER} Analysis

When inspecting Figures 5.9 and 5.10, we see that each chain in SLC_s and MLC_s has outperformed its corresponding typical neural network by considerable margins. Moreover, it is evident that all chains have outperformed all typical neural networks except for SLC_{pca} and MLC_{pca} which were outperformed slightly by $NN_{reliefF}$. Table 5.5 further solidifies these conclusions. Moreover, the table shows that MLC_s and SLC_s have performed relatively similar. MLC_s has slightly outperformed SLC_s in two subsets, MLC_{pca} and $MLC_{reliefF}$ while SLC_s has slightly outperformed MLC_s in the other two subsets, SLC_{all} and SLC_{cfs}.

Interestingly, neither MLC_s chains nor SLC_s chains have exhibited the same behavior as those on D_{AGA} when it comes to the order of performance. The best performing chains in the chain link selection step remained as the best performing chains in the training process.

It is noteworthy that NN_{all}, NN_{cfs}, and NN_{pca} have demonstrated overfitting which disappeared in SLC_{all} and SLC_{pca}, and reduced in SLC_{cfs} and MLC_{cfs}.

5.4.4 D_{SYN} Analysis

When applying SLC to D_{SYN}, we expected that the process would fail and produce large errors from the start. This is because D_{SYN} is a synthetic dataset created by randomizing D_{AGA} dataset. By observing Figures 5.11 and 5.12 it is easily noticed that this is exactly the case. Generally, the error increases as the number of epochs increases. Table 5.5 shows that

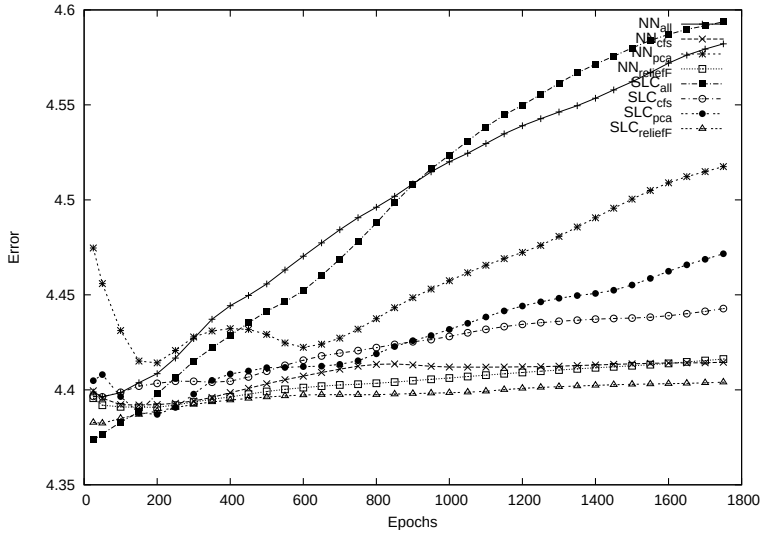


Figure 5.11: Error response to epoch in D_{SYN} SLC networks vs. regular neural networks.

Table 5.4: D_{SYN} chaining results.

D_{SYN}		all	cfs	pca	reliefF
SLC	Chain Link #	12	15	16	11
	MAE	**4.9494**	**4.6457**	**4.7848**	4.4919
MLC	Chain Link #	3	2	6	12
	MAE	5.0767	4.8037	4.9475	**4.6738**

our approaches have outperformed the typical neural network in three of the four subsets namely, ALL, PCA and ReliefF. Table 5.4 shows that SLC have outperformed MLC in three of the four subsets, namely SLC_{all}, SLC_{cfs}, and SLC_{pca}.

5.5 Summary

In this chapter, we have proposed another new ensemble approach, namely chaining neural network ensemble, which improves classification by continually feeding predictions of previously trained neural networks into the training process of another network, forming a chain-like ensemble. This approach has two variations, single-link chaining and multi-link chaining. The first variation trains a neural network on the original data of a given dataset

Table 5.5: Summary of errors from typical, SLC_s, and MLC_s networks' error after training.

Typical																
	D_{AGA}				D_{SPF}				D_{SYN}				D_{RER}			
	all	cfs	pca	reliefF	all	cfs	pca	reliefF	all	cfs	pca	reliefF	all	cfs	pca	reliefF
learning rate	0.015	0.015	0.015	0.015	0.2	0.3	0.2	0.2	0.015	0.015	0.015	0.015	0.3	0.2	0.015	0.2
momentum	0.05	0.1	0.1	0.05	0.2	0.3	0.4	0.3	0.05	0.05	0.05	0.05	0.3	0.4	0.05	0.3
epoch	250	1750	400	250	1550	1750	1750	1750	50	150	200	150	25	50	650	100
MAE	0.826	0.815	0.761	0.788	**0.081**	0.096	0.090	0.083	4.396	**4.392**	4.414	4.391	0.060	0.071	0.086	0.060
SLC																
	D_{AGA}				D_{SPF}				D_{SYN}				D_{RER}			
	all	cfs	pca	reliefF	all	cfs	pca	reliefF	all	cfs	pca	reliefF	all	cfs	pca	reliefF
learning rate	0.05	0.1	0.015	0.05	0.3	0.2	0.2	0.2	0.015	0.015	0.15	0.015	0.05	0.3	0.15	0.3
momentum	0.4	0.05	0.3	0.3	0.3	0.1	0.4	0.2	0.05	0.05	0.05	0.05	0.05	0.2	0.2	0.4
epoch	1750	1500	1750	1750	700	1750	1550	1150	25	50	25	50	1750	150	1650	1750
MAE	0.771	**0.506**	0.711	0.717	0.084	0.091	0.088	**0.083**	**4.374**	4.396	**4.370**	4.382	**0.050**	**0.046**	0.067	0.036
MLC																
	D_{AGA}				D_{SPF}				D_{SYN}				D_{RER}			
	all	cfs	pca	reliefF	all	cfs	pca	reliefF	all	cfs	pca	reliefF	all	cfs	pca	reliefF
learning rate	0.1	0.1	0.3	0.1	0.1	0.05	0.1	0.3	0.015	0.015	0.05	0.015	0.3	0.3	0.3	0.2
momentum	0.1	0.05	0.05	0.2	0.2	0.4	0.2	0.2	0.05	0.05	0.4	0.05	0.4	0.3	0.2	0.3
epoch	1750	1750	1150	1750	1750	1550	1750	800	50	150	50	25	100	150	450	1700
MAE	**0.606**	0.507	**0.445**	**0.552**	0.089	**0.090**	**0.086**	0.084	4.385	4.401	4.380	**4.350**	0.056	0.046	**0.064**	**0.034**

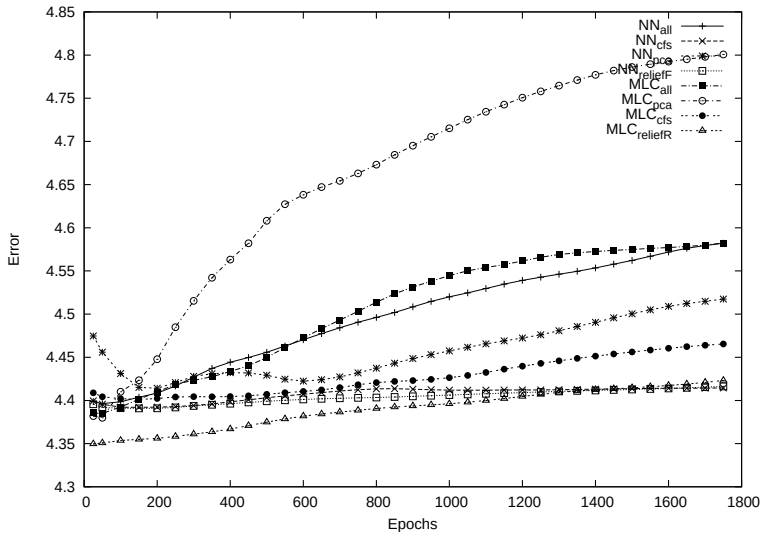


Figure 5.12: Error response to epoch in D_{SYN} MLC networks vs. regular neural networks.

and the predictions of one neural network and repeats this process. The second variation trains a neural network on the original data and the predictions of all previous networks by appending the predictions of each new network to the dataset. Both approaches were demonstrated, through a series of experiments, to improve classification in most of the datasets with a very few exceptions.

Chapter 6

Conclusion

The task of *pattern recognition* is perhaps one of the most recurrent tasks that we encounter in our lives. Therefore, there has been a significant interest of automating this task for many decades. Many techniques have been developed to this end, such as *neural networks* [50].

Neural networks are mathematical models that conduct classifications with a certain degree of confidence. They are excellent classifiers with very robust means of learning. However, neural networks suffer from some disadvantages, such as difficulties in structure estimation and prolonged training times. Therefore, there are still continuing efforts to improve their efficiency and classification accuracy.

Some methods were suggested to improve the effectiveness and efficiency of neural networks, include ensemble and modular approaches. The ensemble approach and the modular approach are among the common methods for improving classification of classifiers. In the ensemble approach, different classifiers (i.e. decision trees, Bayesian networks, neural networks, regression models, etc) are trained on the same data. Then their classification outputs of an instance are considered in a probabilistic manner. A modular neural network is a collection of individually trained neural networks constructed on disjoint subsets of a dataset to predict different target classes. Both approaches use a gating function to combine the outputs of the networks in order to achieve better accuracy.

6.1 Our Contributions

The work presented in this thesis focuses on improving the classification of neural networks. We have proposed two novel approaches toward this end, paralleling neural network ensemble and chaining neural network ensemble. The first uses a neural network to

combine the predictions of structurally different neural networks while the second feeds the predictions of a neural network to another network and repeats this process.

Four datasets were used to verify our approaches, namely Agassiz tomato yield dataset, synthetic dataset, steel plates faults dataset, and restaurant reviews dataset. For each of these datasets, two attribute selection methods, i.e., Correlation-based Feature Selection and ReliefF, and one dimensionality reduction method, i.e., Principal Component Analysis, were applied. There were a total of 16 datasets and subsets with varying properties to verify our approaches.

6.1.1 Paralleling Neural Network Ensemble

In this approach, we proposed using neural networks as a means of combining individual neural networks, with the aim to further improve classification accuracy. Furthermore, we suggest the inclusion of the dataset with the training of the ensemble network along with the predictions of ensemble members.

We have done extensive experiments to show the effectiveness of our approach. We have found that paralleling neural network ensemble, supplemented by additional inputs from the original datasets or one of its subsets, showed promising classification performance.

6.1.2 Chaining Neural Network Ensemble

This approach proposes using a chain of neural networks to further improve classification accuracy. This approach has two variations, single-link chaining, and multi-link chaining. A single-link chain consists of many neural network links. Each of these links is trained on the original dataset and on the predictions of the neural network link that immediately

precedes it, except for the first neural network link which is trained on the original dataset only. A multi-link chain is similar to the first with one restriction removed. In this variation, a neural network is trained on the predictions of all the neural networks that precede it and on the original dataset.

We have done extensive experiments to show the effectiveness of this approach. We have found that this approach, with both variations, shows promising classification performance. They have been shown to outperform each other in different cases and both have outperformed the typical neural network in almost all cases.

6.2 Future Work

While our proposed approaches are very promising, in the future we plan to conduct more experiments and use more varieties of datasets to further investigate their performance. It would be also interesting to further analyze our experiment results.

In our experiments we have noted that when the errors, i.e., MAE, are included in the training and testing process of the paralleling ensemble networks, the classification error of the overall system dropped significantly. Whether this is true in general would be very interesting and we want to explore this further in order to understand it better.

We also plan to provide theoretical justification as to why our approaches work or under what conditions they fail. We also aim to analyze the reasons behind the oscillation behavior in the chaining approach. This could lead us to developing a heuristic for a more efficient mean to selecting chain links. Moreover, we will investigate a method for choosing the number of chain links or for detecting a single good chain link. This will lead to automating and speeding the chaining process.

Bibliography

[1] Merriam webster dictionary, dictionary and thesaurus - merriam-webster.

[2] H. Abdi and L. J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[3] D. Ackley, G. Hinton, and T. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[4] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. In *Machine Learning*, pages 37–66, 1991.

[5] R. Anand, K. Mehrotra, C. K. Mohan, and S. Ranka. Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks*, 6(1):117–124, 1995.

[6] M. Anthony and P. Bartlett. *Neural Network Learning: Theoretical Foundations*. The 1st edition, 1999.

[7] A. Bain. *Mind and Body: The Theories of Their Relations*. D. Appleton and company, the 1st edition, 1873.

[8] W. Baxt. Improving the accuracy of an artificial neural network using multiple differently trained networks. *Neural Computing*, 4(5):772–780, 1992.

[9] R. Beale and T. Jackson. *Neural Computing: an introduction*. The 1st edition, 1994.

[10] M. Bianchini, P. Frasconi, and M. Gori. Learning without local minima in radial basis function networks. *IEEE Transactions on Neural Networks*, 6(3):749–756, 1995.

[11] A. G. Bors and I. Pitas. Robust rbf networks, 2001.

[12] S. Boucheron, O. Bousquet, and G. Lugosi. Theory of classification : A survey of some recent advances. *ESAIM: Probability and Statistics*, 9:323–375, 2005.

[13] N. Burgess. A constructive algorithm that converges for real-valued input patterns. *International Journal of Neural Systems*, 5(1):59–66, 1994.

[14] M. Buscema and S. Terzi W. Tastle. A new meta-classifier. In *North American Fuzzy Information Processing Society*, pages 1–7, 2010.

[15] Y. Chauvin. Advances in neural information processing systems. chapter A back-propagation algorithm with optimal use of hidden units, pages 519–526. 1989.

[16] B. Cheng and D. M. Titterington. Neural networks: A review from a statistical perspective. *Statistical Science*, 9(1):2–30, 1994.

[17] Y. Le Cun, J. Denker, and S. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605, 1990.

[18] S. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, 1990.

[19] S. E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, Carnegie Mellon University, 1988.

[20] M. Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209, 1990.

[21] K. Fukunaga. *Introduction to statistical pattern recognition*. Academic Press, the 2nd edition, 1990.

[22] S. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.

[23] M. A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 359–366, 2000.

[24] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. The 2nd edition, 2006.

[25] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[26] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, the 2nd edition, 1998.

[27] T. Helmer, D. L. Ehret, and S. Bittman. Cropassist, an automated system for direct measurement of greenhouse tomato growth and water use. *Computers and Electronics in Agriculture*, 48(3):198–215, 2005.

[28] A. Jain, J. Mao, and K. Mohiuddin. Artificial neural networks: A tutorial. *IEEE Computer*, 29:31–44, 1996.

[29] W. James. *The principles of psychology*. Macmillan, the 1st edition, 1890.

[30] C. Ji, R. Snapp, and D. Psaltis. Generalizing smoothness constraints from discrete samples. *Neural Computation*, 2(2):188–197, 1990.

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

[32] K. Kira and L. A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *The Association for the Advancement of Artificial Intelligence and The MIT Press*, pages 129–134, 1992.

[33] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 249–256, 1992.

[34] T. Kohonen. An introduction to neural computing. *Neural Networks*, 1(1):3–16, 1988.

[35] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[36] N. Kohzadi, M. S. Boyd, I. Kaastra, B. S. Kermanshahi, and D. Scuse. Neural networks for forecasting: An introduction. *Canadian Journal of Agricultural Economics*, 43:463–474, 1995.

[37] I. Kononenko. Estimating attributes: Analysis and extensions of relief. In *European conference on machine learning*, volume 784, pages 171–182, 1994.

[38] R. P. Lippmann. An introduction to computing with neural nets. *IEEE Acoustics, Speech, and Signal Processing Society Magazine*, 3(4):4–22, 1987.

[39] B. Lu and M. Ito. Task decomposition and module combination based on class relations: a modular neural network for pattern classification. *IEEE Transactions on Neural Networks*, 10(5):1244–1256, 1999.

[40] J. Mccarthy. Logic-based artificial intelligence. chapter Concepts of Logical AI, pages 37–56. 2000.

[41] M. Mezard and J. Nadal. Learning in feedforward layered networks : the tiling algorithm. *Journal of Physics A*, 22(12):2191–2203, 1989.

[42] M. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in Neural Information Processing Systems*, volume 1, pages 107–115, 1989.

[43] D. Olmsted. Neural network history to 1960 @ONLINE, Feb 2012. http://www.neurocomputing.org/NNHistoryTo1960.aspx.

[44] D. W. Opitz and J. W. Shavlik. Actively searching for an effective neural-network ensemble. *Connection Science*, 8(3):337–353, 1996.

[45] B. A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: a survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

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

[47] F. Rosenblatt. The perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, 1957.

[48] P. Rosin and F. Fierens. Improving neural network generalisation. In *International Geoscience and Remote Sensing Symposium*, volume 2, pages 1255–1257, 1995.

[49] S. Roy. Factors influencing the choice of a learning rate for a backpropagation neural network. In *IEEE International Conference on Neural Networks*, volume 1, pages 503–507, 1994.

[50] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. The 2nd edition, 2003.

[51] A. J. C. Sharkey. On combining artificial neural nets. *Connection Science*, 8:299–313, 1996.

[52] A. J. C. Sharkey. *Combining Artificial Neural Nets: Modular Approaches*. Citeseer, 1997.

[53] K. Sheela and S. Deepa. Analysis of computing algorithm using momentum in neural networks. *Journal of Computing*, 2, 2011.

[54] J. Sola and J. Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44(3):1464–1468, 1997.

[55] R. Thompson. The neurobiology of learning and memory: William james in retrospect. *Psychological Science*, 1(3):172–173, 1990.

[56] K. Tin-yau and Y. Dit-Yan. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Transactions on Neural Networks*, 8:630–645, 1997.

[57] J. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231, 1996.

[58] R. Valdovinos and J. Sanchez. Ensembles of multilayer perceptron and modular neural networks for fast and accurate learning. In *Fifth Mexican International Conference on Artificial Intelligence*, pages 229–236, 2006.

[59] M. Robnik Šikonja and I. Kononenko. Theoretical and empirical analysis of relieff and rrelieff. *Machine Learning*, 53(2):23–69, 2003.

[60] H. Yu, T. Xie, S. Paszczynski, and B. M. Wilamowski. Advantages of radial basis function networks for dynamic system design. *IEEE Transactions on Industrial Electronics*, 58(12):5438–5450, 2011.

[61] X. Yu, G. Chen, and S. Cheng. Dynamic learning rate optimization of the backpropagation algorithm. *IEEE Transactions on Neural Networks*, 6(3):669–677, 1995.

[62] K. M. Zaamout and J. Z. Zhang. Improving classification through ensemble neural networks. In *Proceedings of 8th International Conference on Natural Computation*, 2012. To appear.

[63] G. P. Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 30(4):451–462, 2000.

[64] Z. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137:239–263, 2002.