UNIVERSIDAD MICHOACANA DE
SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POSGRADO

AUTOMATIC DESIGN OF ARTIFICIAL NEURAL NETWORK CLASSIFIERS BY MEANS OF GENETIC ALGORITHMS

by
Marco Tulio Arreola Fernández

THESIS

Requirement for the degree of
MASTER OF SCIENCES IN ELECTRICAL
ENGINEERING

Advisor:
Juan José Flores Romero

Co-advisor:
Mario Graff Guerrero

July 2012

AUTOMATIC DESIGN OF ARTIFICIAL NEURAL NETWORK CLASSIFIERS BY MEANS OF GENETIC ALGORITHMS

Los Miembros del Jurado de Examen de Grado aprueban
la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica** de ***Marco Tulio Arreola Fernández***

Dr. Félix Calderón Solorio
Presidente

Dr. Juan José Flores Romero
Director de Tesis

Dr. Mario Graff Guerrero
Co-Director de Tesis

M Graff G.

Dr. Jaime Cerda Jacobo
Vocal

Dr. Nelio Pastor Gómez
Revisor Externo
UMSNH Ingeniería Civil

Dr. J. Aurelio Medina Rios
Jefe de la División de Estudios de Posgrado
de la Facultad de Ingeniería Eléctrica.

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Julio 2012

To Carlos and Margarita

Acknowledgements

First and foremost I offer my sincerest gratitude to my advisors, Juan J. Flores and Mario Graff, who have supported me throughout my thesis with his endless patience and knowledge, and without them, this thesis would not have been completed or written.

I would like to thank to my thesis examiners, their feedback have been really helpful and enriching, thanks to Felix Calderón, Jaime Cerda, and Nelio Pastor.

I also thank the teachers and researchers who taught me in the graduate class. They enriched my mind and soul. With them, I learned in directly or indirectly way how to be a better human being.

I am grateful to my many student colleagues for providing a stimulating and fun environment in which to learn and grow. Sorry for not mention any one but it would impossible to write all those names here.

I am forever indebted to the Universidad Michoacana de San Nicolás de Hidalgo and also with the División de Estudios de Posgrado of the Facultad de Ingeniería Eléctrica, from now on, my *alma mater*. This thesis could not been possible without the economic support of CONACyT, thanks.

I wish to thank my entire family for providing a loving environment for me. My brother Nitos, my sisters Celi and Jani, my brother in law Paco, my sister in law Eda, my nephew Carlos the third, my two nieces Karlita and Andy, my aunt Carmen, my uncle Alvaro, my cousins Edson and Bebis, and my stinky pet Pascal.

Finally, and most important, I wish to thank my parents Carlos and Margarita, who have put the voices in my head that always push me on to give my best.

ABSTRACT

Pattern classification is a field that has many applications, including image analysis, speech and audio recognition, biometrics, bioinformatics, data mining, and information retrieval, among others.

Artificial Neural Networks (ANN) have been successfully used as a classifier system in recent years. They have shown an exceptional ability to learn classification functions. However, in order to design an efficient ANN classifier one needs to address the following questions: (1) What neural network architectures should be used? (2) How many neurons are appropriate for the task? (3) Which learning algorithm is the most suitable? (4) How much should be trained one neural network in order to be effective? In addition to these questions, intrinsically every classification problem also involves answering to: (5) What features are relevant to discriminate one object from another?

The task of designing the structure of an ANN, is typically done by an expert in both the problem and the ANN's field. To avoid the inconvenience of relying on a human expert, it is a good idea to obtain the design of the network by an automatic process, this is, (6) automated design of artificial neural network.

This work aims to answer the aforementioned questions; to do so an evolutionary approach is proposed to automatically design ANNs for classification. GEANN, Genetic-Evolutionary Approach for Neural Networks performs feature selection. Designs the ANN's topology and selects a training algorithm between backpropagation, batch backpropagation, resilient backpropagation, and quickprop. We deal with overfitting of the net by doing a K-fold cross validation.

We compared our evolved ANN against previous methods, experimental results show that our approach produces competitive ANNs. Since GAENN automatically designs ANN's topology, it can be used by anyone with no experience in the field of artificial neural networks, and can be applied to a wide variety of classification problem, the only requirement is to have a tagged dataset of *a priori* known objects.

RESUMEN

La clasificación de patrones es un campo que tiene muchas aplicaciones, incluyendo el análisis de imágenes, reconocimiento de voz y audio, datos biométricos, bioinformática, minería de datos y recuperación de información, entre otros.

Las Redes Neuronales Artificiales (RNA) se han utilizado con éxito como sistemas clasificadores en los últimos años. Han demostrado una capacidad excepcional para aprender las funciones de clasificación. Sin embargo, con el fin de diseñar un eficiente clasificador de RNA se necesita hacer frente a las siguientes preguntas: (1) ¿Qué arquitectura de RNA se debe utilizar? (2) ¿Cúantas neuronas son apropiadas para la tarea? (3) ¿Qué algoritmo de aprendizaje es el más adecuado? (4) ¿Cúanto debe ser entrenada una RNA para ser eficaz? Además de estas preguntas, intrínsecamente cualquier problema de clasificación también involucra responder a: (5) ¿Qué características son relevantes para discriminar un objeto de otro?

La tarea de diseñar la estructura de la red, tipicamente es realizada por un experto, tanto en el área del problema como en el campo de las redes neuronáles. Para evitar el inconveniente de depender de un experto humano, es una buena idea obtener el diseño de la red mediante un proceso automático, esto es, (6) diseño automatizado de la red neuronal artificial.

Este trabajo pretende dar respuesta a las preguntas anteriores; para hacerlo se propone un enfoque evolutivo para diseñar automáticamente RNAs para clasificación. EGERN, Enfoque Genetico-Evolutivo para Rededes Neuronales lleva a cabo selección de características, el diseño de la topología de la RNA y selecciona un algoritmo de entrenamiento entre propagación hacia atras, propagación hacia atras por lote, propagación hacia atras resistente y quickprop. Nos ocupamos del sobreentrenamiento de la red haciendo una validación cruzada.

Comparamos nuestra red evolucionada contra métodos previos, los resultados experimentales muestran que nuestro enfoque produce RNAs competitivas. Debido a que EGERN diseña automáticamente la topología de la RNA, puede ser utilizado por cualquier persona sin experiencia en el campo de las redes neuronales artificiales, EGERN puede usarse en una amplia variedad de problemas de clasificación, el único requisito es contar con un conjunto de datos etiquetas de objetos conocidos *a priori*.

Contents

List of Figures

List of Tables

List of Programs

List of Algorithms

List of Symbols and Abbreviations

SYMBOLS

$\varphi(\cdot)$	Activation function
v_j	Activation potential of neuron j
ξ	Average squared error of the sum of squared errors
$\lceil\ \rceil$	Ceiling function
c_i	Class i
y_i	Class type of pattern i
$[a, b]$	Closed interval of variable x signifies that $a \leq x \leq b$
$\setminus$	Complement binary operator
Pc	Crossover probability
D	Dimensionality or number of measurements
x_i	Feature i
Pm_{gene}	Gene mutation probability
$\xi^{(n)}$	Instantaneous value of the sum of squared errors
η	Learning-rate parameter
$\delta_j^{(n)}$	Local gradient of neuron j at time n
Pm	Mutation probability
C	Number of classes
N	Number of samples in TS
M	Number of samples in VS
(a, b)	Open interval of variable x signifies that $a < x < b$
$\mathbf{x_i}$	Pattern i
Δw	Small change applied to weight w
$\in$	Symbol for “belongs to”
w_{kj}	Synaptic weight of synapse j belonging to neuron k
$\mathbf{w}$	Vector of weights

ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
BP	Back Propagation
CV	Cross Validation
DNA	Dioxy Ribo Nucleic Acid
EA	Evolutionary Algorithm
EC	Evolutionary Computation
GA	Genetic Algorithm
GBML	Genetic-Based Machine Learning
GEANN	Genetic-Evolutionary Approach for Neural Networks
iRPROP	Improved Resilient Backpropagation
ML	Machine Learning
NNEE	Neural Network Ensemble Editing
PE	Processing element
RPROP	Resilient Backpropagation
SC	Soft Computing
TS	Training Set
VS	Validation Set

List of Publications

Juan J. Flores, Marco Tulio Arreola, Jean François Mas, Mario Graff, Félix Calderón. *"Evolving artificial neural networks for binary and multi-class classification problems"*. XIX Reunión nacional SELPER-México, Sociedad de percepcion remota y sistemas de información espacial. Morelia, Michoacán 2011.

Mario Graff, Juan J. Flores, Marco Tulio Arreola. *"Lessons learned in Evolving Artificil Neural Networks for Classification and Forecasting.* Reunión de Otoño de Potencia, Electrónica y Computación, ROPEC 2011 Internacional. Morelia, Michoacán noviembre del 2011.

Chapter 1

Introduction

Pattern recognition[1] is the field with the objective to classify objects into different categories and classes [Duda et al., 2001]. This field has many applications, such as image analysis, speech and audio recognition, biometrics, bioinformatics, data mining, and information retrieval.

Human beings are very good at recognizing patterns. Since the beginning humans needed to classify for surviving, this is an ability that has been perfected through evolution. Unconsciously, we do it all the time, for example, we distinguish harmful food, we recognize faces when we see them, voices over a poor telephone line, we diagnose diseases, identify types of cars, classify galaxies by their shape, etc.

In the past, recognition systems were manually developed and maintained by human experts. The traditional approach requires that a human expert insight into the objects to be detected and recognized since it is very difficult to identify a set of features that characterize a complex set of objects.

It is not difficult to see that during the twentieth century automation had free human beings from the heavy physical labor in the industry. However, many tasks, which were thought to be light in physical labor, such as parts inspection in order to reject defective products are still in their primitive human operation stage. Such work involves mainly the acquisition of information through the human sensory organs and then processing this information in the brain so that the worker make a decision. Humans classify all the time, it is a necessary task for surviving; however,

[1] Pattern recognition is also called pattern classification, discriminant function analysis, or decision making.

nowdays there is a prohibitive amount of information and in order to deal with it we need to automatize this task.

The basic function of human intelligence is to ensure survival in nature, not to perform precise calculations. The human brain can process millions of visual, acoustic, olfactory, tactile, and motor data, and it shows astonishing abilities to learn from experience, generalize from learned rules, recognize patterns, and make decisions. Without any doubt the human mental faculties of learning, generalizing, memorizing, and predicting should be the foundation of any intelligent artificial device or smart system. Many algorithmic approaches have been developed trying to mimic human intelligence, yet we are still far away from achieving anything similar to that.

Notwithstanding, we are very bad at calculations or at any kind of computing, there is a long-standing tradition in science that gives more respect to theories that are quantitative, formal, and precise than to those that are qualitative, informal, and approximate. Recently, however, the validity of this tradition has been challenged by the emergence of efficient soft computing techniques to satisfy them.

Many contemporary problems do not lend themselves to precise solutions within the framework of classical hard computing, for instance, recognition problems of all sorts (handwriting, speech, objects, images, etc.), forecasting (weather, financial, or any other time series), and combinatorial problems like the "*traveling salesman*". To be able to deal with such problems, there is often no choice but to accept solutions that are suboptimal. In addition, even when precise solutions can be obtained, their cost is generally much higher than those solutions that are imprecise.

Under this scenario, we can find the automatic design of neural networks. In this problem, one deals with the identification of all the parameters of the neural network such as: number of layers, number of neurons, activation functions, and so on. As expected, one cannot use traditional optimization techniques to tackle this problem given that it is a combinatorial optimization problem. As a consequence, one is forced to look for different techniques, perhaps approximate ones, that allow to determine the parameters and structure of an accurate neural network.

1.1 Motivations

In literature, there exist several classification methods. Some of them are of statistical nature, such as: Bayesian networks, Bayesian discriminative approaches, Bayesian learning methods, Markov chains, kernels methods, discriminant functions, maximum likelihood via Expectation Maximization, mixture models for discrimination, and k-nearest neighbors, among others. Also, there are methodologies developed in the field of Artificial Intelligence (AI) such as: support vector machines, decision trees, fuzzy logic models, ANNs, etc. [Kecman, 2001, Bishop, 2006, Ripley, 2007, Webb and Copsey, 2011] are excellent books where to find more detailed information on those and others approaches.

Given the amount of work done in this field one may be tempted to think that this is a closed problem. However, there are still a number of questions that need to be addressed. In order to apply any of the aforementioned classification techniques in real life problems, one needs to know very well the methodology and have some experience in its usage, besides a relative major knowledge of the problem itself.

ANNs have been successfully used as classifier systems in recent years, they have shown an exceptional ability to learn classification functions. However, in order to design an efficient ANN classifier one needs to address the following questions:

(1) What neural network architectures should be used?

(2) How many neurons are appropriate for the task?

(3) Which learning algorithms are most suitable?

(4) What features are relevant for discriminating one object from another?

(5) How much should be trained one neural network in order to be effective?

In this thesis, we address all the aforementioned questions. Issues raised in questions (1), (2), (3), and (4) are problem dependent, and are answered for each particular problem at a time. Find how large an ANN should be, find the relevant features for classification, and find a proper learning algorithm clearly are optimization problems, we delegate the task to find these answers to a genetic algorithm. Since its origins in mid-70's GA have been proved to be an efective poblational evolutive approach used to solve optimization problems. Once GA have found answers for question (1) to (4), we use a cross-validation technique in order to answers question number (5).

1.2 Objectives

The main objective of this thesis is to develop a methodology capable of automatically design an ANN for classification problems.

Particular Objectives

- To propose an evolutive approach to design ANNs for classification.
- To apply a technique for reducing dimensionality of the classification problem; this problem is known as feature selection.
- Use a parsimony criteria to produce the simplest ANN that performs a proper classification.
- Select a suitable learning algorithm.
- Avoid overfitting.

1.3 State of the Art

Artificial Intelligence is broadly defined as a branch of computer science concerned with intelligent behavior on artifacts. Intelligent behavior, involves perception, reasoning, learning, communicating, and acting in complex environments.

At present, *Soft computing* (SC) is not a closed and clearly defined discipline. It includes an emerging and more or less established family of problem-stating and problem-solving methods that attempt to mimic the intelligence found in nature [Kecman, 2001]. In traditional computing, the prime objectives of computations are precision and certainty. However, in soft computing, the precision and certainty carry a cost. SC considers the integration of computation, reasoning, and decision making as partners in order to provide a framework for the trade-off between precision and uncertainty. This integration of methodologies provides a foundation for the conceptual design and deployment of intelligent systems. In contrast to analytical methods, soft computing methodologies mimic consciousness and cognition.

Mitchell defines *Machine Learning* (ML) as the study of computer algorithms that improve automatically through experience [Mitchell, 1997]. [Kovacs, 2011] manifests that ML is concerned with machines which improve with experience and reason inductively or deductively in order to: optimize, approximate, summarize, generalize from specific examples to general rules, classify, make predictions, find associations, propose explanations, and propose methods to group objects.

There is some redundancy in some concepts and fields used in this thesis. Subjects as learning, ML, SC, and intelligent machine are a blend of different areas. The various fields bound together here used to be separate, and today they are amalgamated in the broad area of Artificial Intelligence. Therefore, each area was developed separately by researchers, scientists, and enthusiasts with different backgrounds.

ANNs and evolutionary computation (EC) have each been proved to be effective in solving certain classes of problems. ANNs are very good at mapping inputs to outputs and evolutionary algorithms are very good at optimization. Therefore, it was natural for scientists to combine the methodologies to develop hybrid computational tools that are even more effective than either methodology by itself.

According to [Kovacs, 2011], Genetics-based Machine Learning (GBML) is the application of Evolutionary Algorithms (EAs) to machine learning. This places evolvable ANNs in GBML. Also in his work, Kovacs classifies GBML systems algorithmically in two approaches, in the Pittsburg approach one chromosome encodes one solution, in the Michigan approach one solution is represented by many chromosomes. Evolving ANNs generally uses the Pittsburgh approach.

An artificial neural network consists of a set of nodes, a set of directed connections between a subset of nodes and a set of weights on the connections. The connections specify inputs and outputs to and from nodes and there are three forms of nodes: input nodes (input to the network from the outside world), output nodes, and hidden nodes, which only connect to other nodes. Nodes are typically arranged in layers: the input layer, hidden layer(s), and output layer. Nodes compute by integrating their inputs using an activation function and passing on their activation as output. Connection weights modulate the activation they pass on and, in the simplest form of learning, weights are modified while all else remains fixed. The most common approach to learning weights is to use a gradient descent-based learning rule such as backpropagation.

The architecture of an ANN refers to the set of nodes, connections, activation functions and the plasticity of nodes; whether they can be updated or not. Most often all nodes use the same activation function and in virtually all cases all nodes can be updated.

Evolution has been applied at three levels: weights, architecture, and learning rules. In terms of architecture, evolution has been used to determine connectivity, select activation functions, and determine plasticity [Kovacs, 2011]. A fourth area, the evolution of inputs (finding the optimal set of inputs), has received considerable amount of attention [Flores et al., 2009, Flores et al., 2010].

Relatively little has been done on the evolution of neural network processing element (PE) transfer functions and even less on evolving topological structure and PE transfer functions simultaneously [Kovacs, 2011].

Three forms of representation have been used to encode ANN into the chromosome of an EA: 1)*direct encoding*: [Yao, 1999, Floreano et al., 2008] in which all details (connections and nodes) are specified, 2)*indirect encoding*: [Yao, 1999, Floreano et al., 2008] in which parameters that specify the network topology are evolved (e.g. number of hidden layers and nodes) and a learning process determines the details, and 3)*developmental encoding*: [Floreano et al., 2008] in which a developmental process is genetically encoded [Gruau, 1995, Szirnyi and Csapodi, 1998]. Indirect and developmental representations are more flexible and tend to be used for evolving architectures while direct representations tend to be used for evolving weights alone.

Since the popularization of the back propagation (BP) algorithm (based on gradient descent) in the mid-1980s [Werbos, 1974, Rumelhart and McClelland, 1986, Hecht-Nielsen, 1989], there has been a significant increase in research and development in the area of applying EC techniques for the purposes of evolving learning rules aspects of AANs. [Schaffer et al., 1992] and [Yao, 1999] are excellent surveys of evolutionary ANNs covering 80-s, 90-s research decades but [Kovacs, 2011] is an up to date state of the art of evolutionary ANNs and GBML in general.

Research has been done on evolution of weights, in which the fitness function penalizes the ANN error and also the network complexity (number of hidden neurons). In terms of learning accuracy there is no clear winer between evolution and gradient descent; however, Yao [Yao, 1999] states that evolving weights and architecture is better than evolving weights alone.

There is not a best learning rule for all architectures or problems. One approach is to evolve only learning rule parameters [Yao, 1999] such as the learning rate and momentum in BP. Castillo [Castillo et al., 2007] found that, evolving the architecture, initial weights and rule parameters together carry out good or better results than, evolving only the first or the second of the three.

Yao proposes a three nested cycles framework for evolving architectures, training rules and weights [Yao, 1999]. Weight evolution is innermost as it occurs at the fastest time scale while either rule or architecture evolution is outermost. The framework can be thought of as a 3-dimensional space of evolutionary neural network, where zero on each axis represents one-shot search and infinity represents exhaustive search.

1.4 Thesis Outline

This thesis is organized as follows. Chapter 2 Artificial Neural Networks as Classifiers, talks about basic terminology about ANNs, what an artificial neuron is and how to arrange them in order to form ANNs; it briefly explains the biological background of ANNs, describes the perceptron and the multilayer perceptron, and explains how ANNs learn.

Chapter 3, Genetic Algorithms, explains what it is and how it works the canonical GA. Covering simple GA's scheme, basic operations with individuals and how result improves over generations.

Chapter 4, Evolving ANNs, describes in depth our approach and how ANNs have been evolved by means of GAs in order to get efficient and effective classificators. Describe the chromosome structure used to represent ANN classifier and how the fitness is computed to achieve a better classification.

Chapter 5 presents the results obtained in experiments conducted, also details the datasets used.

Finally, Chapter 6 discusses conclusions and future work.

Chapter 2

Artificial Neural Networks as Classifiers

For many decades, it has been a goal of engineers and scientists to develop a machine with elements similar to those found in the human brain. An artificial neural network, or simply neural network, is a type of artificial intelligence that attempts to mimic the way human brain processes and stores information [Rojas, 1996].

In general, artificial neural networks[1] are composed of many simple processing elements connected in a particular way, emulating various brain activities. It works by creating connections between mathematical processing elements, called neurons. The network function is determined largely by the connections between these elements.

Neural networks can be trained to perform complex mapping input-output problems. The powerful capability of the network comes from the characteristics of its elements arrangement. There are one or more layers of hidden neurons that are not part of the input or output of the network; a smooth nonlinearity activation function employed at the output end of each neuron; and a high degree of connectivity in the network. These three distinct characteristics enable multilayer networks to learn complex tasks.

ANNs exploit parallel local processing and distributed representation properties that are believed to exist in the brain. The behavior of the trained ANN depends on the weights, which are also referred to as strengths of the connections

[1] Artificial neural networks are also known as neurocomputer, connectionist networks, or parallel distributed processors.

between the PEs. ANNs learn from experience and generalize from previous samples. They modify their behavior in response to the environment, and are ideal in cases where the required mapping function is not known and tolerance to faulty input information is required.

ANNs offer certain advantages over conventional processing techniques. These advantages are the generalization capability, parallelism, distributed memory, redundancy, adaptivity, fault tolerance, and learning.

In recent years, ANNs have been applied successfully to a variety of problems such as signal processing [Lapedes and Farber, 1987, Herault and Jutten, 1987, Amari and Cichocki, 1998, Kechriotis and Manolakos, 1993], image compression [Dony and Haykin, 1995], transportation [Kurokawa and Takeshita, 2004], forecasting [Flores et al., 2009, Flores et al., 2010], robotics [Pomerleau, 1996], medical diagnosis [Brause, 2001], manufacturing [Roseiro et al., 2005], adaptive control [Chen and Narendra, 2001, Johnson and Calise, 2001, Idan et al., 2001], and machine vision [Rowley et al., 1996].

2.1 Biological Background

In order to understand some basic functions and architectural building blocks of the human brain from the engineering and mathematical perspectives, this section briefly introduces such topics of biological neural systems as the morphology of biological neurons and neural signal processing.

The human nervous system is a very complex network of cells. The brain is the central element of the human nervous system, consisting of near 10^{10} biological neurons, coupled to receptors and effectors [Haykin, 2007].

The human brain is a highly complex, nonlinear, and parallel information-processing system. It has the capability to organize its structural constituents known as neurons, as well as to perform certain computations (e.g. pattern recognition, perception, and motor control) faster than the fastest electronic computer developed so far. The brain's plasticity permits the nervous system to adapt to its surrounding environment.

Adaptation implies that the element can change in a systematic manner and in doing so alter the transformation between input and output. In the brain, transmission within the neural system involves coded nerve impulses and other physical chemical processes that form reflections of sensory stimuli and incipient motor behavior.

A biological neuron is a cell whose main function is the collection, processing, and dissemination of electrical signals. The brain's information processing capacity is thought to emerge primarily from networks of such neurons.

Neurons have particular morphologies, depending on their role and position in the nervous system, despite there is no such thing as a typical neuron [Arbib, 2002], we can summarize properties shared by many neurons. The "basic prototype neuron" is composed of a body, an axon and a multitude of dendrites.

The *Soma* (body), is the metabolic center of the neuron; *dendrites*, are the receptive area of the neuron; the *axon*, is the neuronal conducting unit that conveys the information to other cells; *presynaptic terminals* of the axon, are the transmitting elements of the neuron. Through these presynaptic terminals, one neuron contacts and transmits information to the receptive surfaces of another neuron. This point of contact is known as the *synapse* [Rabuñal and Dorrado, 2006].

The synapse is the basic input-output unit for transmission of information between neurons. The neuron acts as a multi-input/single-output unit. A single neuron can have several neighbors connected to it and bring in electrical signals across the synapses and through the dendrites while it can be connected to other neuron via the axon. Within the brain, neurons are connected to each other in some way, and form a huge network.

The cell body of a neuron sums the incoming signals from dendrites as well as the signals from numerous synapses on its surface. A particular neuron will send an impulse to its axon if sufficient input signals are received to stimulate the neuron to its threshold level. However, if the inputs do not reach the required threshold, the input will quickly decay and will not generate any action. The majority of neurons encode their outputs as a series of brief voltage pulses, known as spikes. A synapse is a simple connection that can impose excitation or inhibition, but not both to the receptive neuron. Figure 2.1 shows a biological neuron's composition. As explained below its dendrites acts as input signal channels then its body processes those signals so that finally the result is transmitted through a axon to others neurons.

This basic biological description of neuronal morphology will provide some inspiration for the development of new neural structures for engineering and science

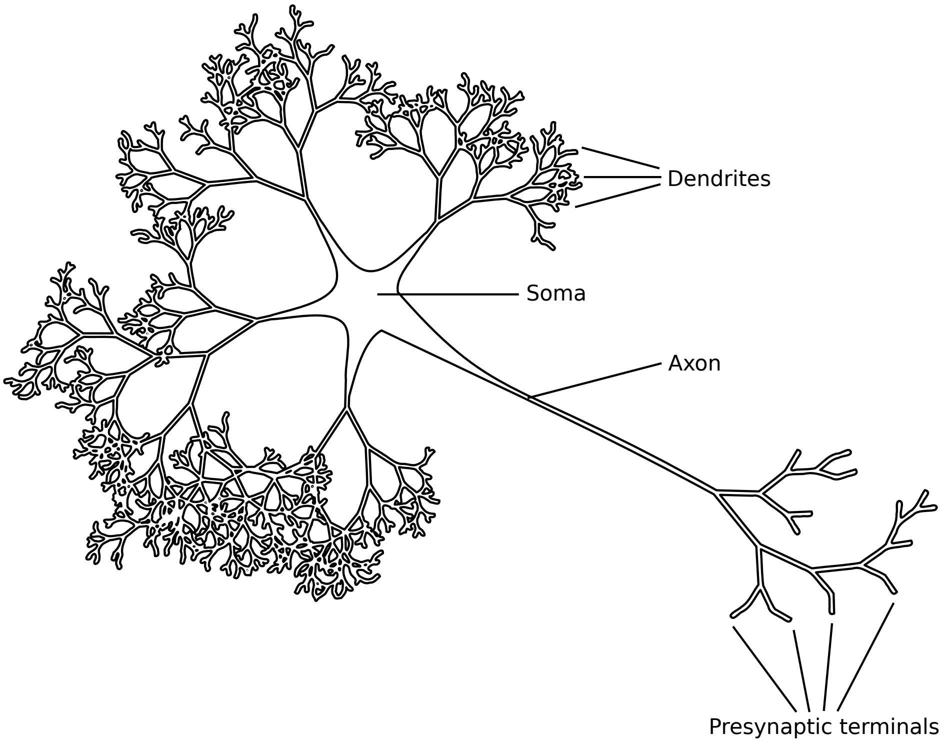


Figure 2.1: Biological neuron.

applications. In advance, we can notice that the processing of information within a neuron involves two main mathematical operations:

1) *Synaptic operation.* The synaptic operation assigns a relative significance (weight) to each incoming signal according to the past experience (knowledge) stored in the synapse.

2) *Somatic operation.* The somatic operation provides various mathematical operations such as aggregation, thresholding, nonlinear activation, and dynamic processing to the synaptic inputs. If the weighted aggregation of the neural inputs exceeds a certain threshold, the soma will produce an output signal to its axon.

2.2 Artificial Neural Network

Artificial neural networks, as models of specific biological computational structures, consist of distributed information processing units. Each computing unit in the network is based on the concept of an idealized artificial neuron. An ideal neuron is assumed to respond optimally to the applied inputs. Artificial neurons are connected through synaptic connections characterized by weight coefficients and every single neuron makes its contribution towards the computational properties of the whole system.

As an information processor, an individual neuron performs an aggregation on its weighted inputs and yields an output through a nonlinear activation function with a threshold. Neural units are the basic building blocks for complex neural network architectures. The study of neurons suggested that a single neuron operates like a linear classifier, and that a combination of many neurons may produce a complex, piecewise linear boundary.

2.2.1 Artificial Neuron

In brain theory, the complexities of real neurons are abstracted in many ways to aid in understanding different aspects of neural network development, learning, or function, therefore there is no such thing as a "typical" neuron. The artificial neurons are designed as variations on the abstractions of brain theory and are implemented in software. However, we can specify characteristics of particular neuron model.

The first formal model of the neuron was proposed as early as 1943 by McCulloch and Pitts [McCulloch and Pitts, 1943], this model is formed by: a set of two-valued inputs, $x_1, x_2, \ldots, x_n \in \{0, 1\}$; one single two-value output, $y \in \{0, 1\}$; a set of two-valued weights, $w_1, w_2, \ldots, w_n \in \{-1, 1\}$; and a threshold, $w_0 \in \mathbb{R}$. Each weight w_i is associated with a particular input x_i. The output of the neuron is calculated by function $g\left(\sum_{i=1}^{n} w_i\, x_i - w_o\right)$, where $g(\cdot)$ is a step function [Gupta et al., 2003]. This neuron model follows an "all-or-none" law.

More recently, the development of adaptive methods offers an opportunity for emulating the learning function of biological neural processes. Some of such neural models were developed in the 1960s by Widrow and Hoff (Adaline), and Rosenblatt (Perceptron). Adaline or Adaptive Linear Element consists of an adaptive linear combiner cascaded with a hard-limiting quantizer, which is used to produce a binary output. The threshold parameter, or bias weight w_0 is connected to a constant input $x_0 = 1$ and controls the threshold level of the quantizer. The difference between the perceptron and the Adaline lies in the training procedure and not in the organization of components of the neuron.

The neuron model used in this thesis is quite similar to the Perceptron and Adaline neuron model. It consists of four basic components that include (1) weights, (2) aggregation function, (3) bias and an (4) activation function. Figure 2.2 shows neuron model described in this section.

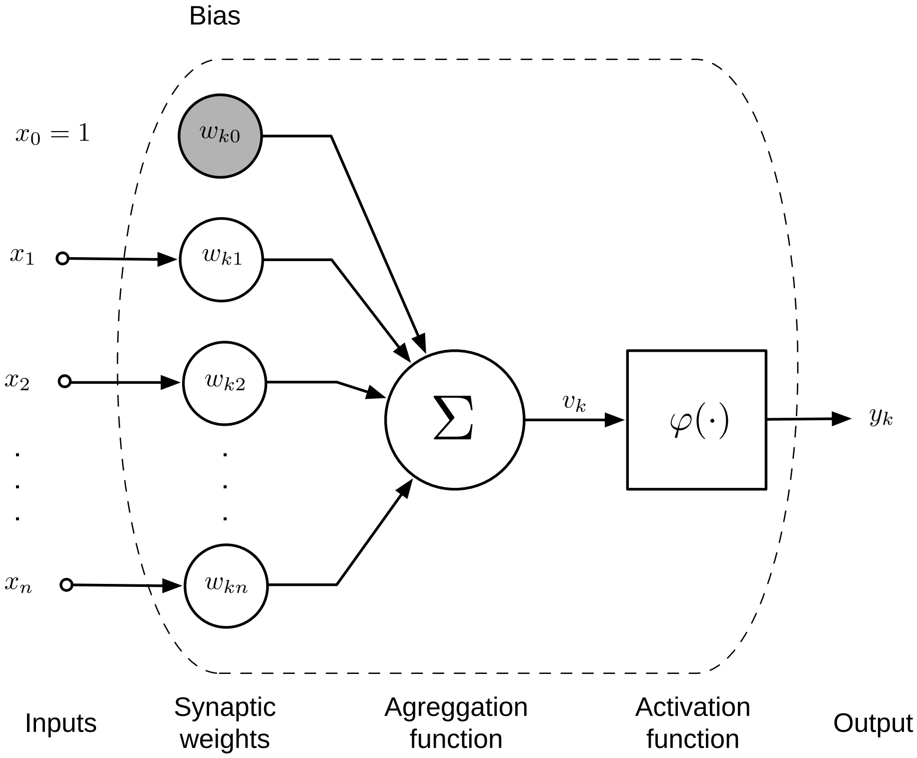


Figure 2.2: Artificial neuron.

(1) A set of synapses, each of which is characterized by a weight or strength of its own. A measure x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. The first subscript refers to the neuron in question while the second refers to the input end of the synapse to which the weight refers. Unlike biological synapse, the synaptic weight may lie in a range that includes negative as well as positive values, if the weight is positive, commonly excites the node output; whereas, for negative weights, tends to inhibit the node output. $x_j \in \mathbb{R}$, $w_{kj} \in \mathbb{R}$, and $j > 0$.

(2) Aggregation function, usually is an adder for summing the input signals, weighted by the respective synapses of the neuron; the operation described here constitute a *linear combiner.*

(3) The bias w_{k0} has the effect of increasing or lowering the neuron input of the activation function. $w_{k0} \in \mathbb{R}$, and $x_0 = 1$.

(4) An activation function is used for limiting the amplitude of the output of a neuron. Typically, the normalized amplitude range of the output neuron is written as the closed unit interval $[0, 1]$ or alternatively $[-1, 1]$.

In mathematical terms, a neuron k is represented as the following equation:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{2.1}$$

where $x_1, x_2, x_3, \ldots, x_m$ are the inputs features; $w_{k1}, w_{k2}, w_{k3}, \ldots, w_{km}$ are the synaptics weights of the neuron k; w_{k0} is the bias; v_k is the linear combiner output due to the input signals and bias, also v_k is the input of activation function. Finally, the output of the neuron is computed as:

$$y_k = \varphi(v_k) \tag{2.2}$$

where $\varphi(\cdot)$ is the activation function.

Having described all the pieces that compose the artificial neuron model, it is sometimes more convenient to express it in just one equation as:

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right) \tag{2.3}$$

The bias is an external parameter of artificial neuron k and has the effect of applying an affine transformation to the output $\sum_{j=1}^{m} w_{kj} x_j$ of the linear combiner.

2.2.2 Types of activation function

The activation function denoted by $\varphi(\cdot)$, defines the output of a neuron in terms of the induced local field v. Due to the learning algorithm used to adapt weights, the activation functions needs to be continuous and differentiable. However, after the training phase one can change the activation function for a discrete version for instance to get all the outputs between $[-1, 1]$. The most common activation functions found on the literature are:

1) *Threshold or step function.* Here we have:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \tag{2.4}$$

The output of the neuron takes the value of 1 if the induced local field of that neuron is positive, and 0 otherwise.

2) *Linear piece function.* In this one we have:

$$\varphi(v) = \begin{cases} 1, & v \geq 1 \\ v, & 0 < v < 1 \\ 0, & v \leq 0 \end{cases} \tag{2.5}$$

This form of activation function may be viewed as an approximation to a nonlinear amplifier.

3) *Sigmoid function.* The sigmoid function, whose graph is a s-shaped, is by far the most common form of activation function used in ANNs. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. The sigmoid function is defined by:

$$\varphi(v) = \frac{1}{1 + \mathrm{e}^{-2vs}} \tag{2.6}$$

where s is the slope of the sigmoid function. Whereas, a threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1.

The activation functions described in Equations 2.4, 2.5 and 2.6 range from 0 to 1. It is sometimes desirable to have an activation function range from -1 to 1, in which case the activation function assumes a symmetric form with respect to the origin.

4) *Symmetric threshold function* is defined as:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases} \tag{2.7}$$

5) *Linear pice symmetric function*:

$$\varphi(v) = \begin{cases} 1, & v \geq 1 \\ v, & -1 < v < 1 \\ -1, & v \leq -1 \end{cases} \tag{2.8}$$

6) *Sigmoid symmetric function*:

$$\varphi(v) = \frac{2}{1 + \mathrm{e}^{-2vs}} - 1 \tag{2.9}$$

2.2.3 Network topologies

Network topology refers to the particular way artificial neurons are arranged in order to produce desired outputs, this is also called network architecture. There exist three fundamentally different classes of network architectures:

1) *Single-Layer Feedforward Networks.* This is the simplest form of a layered network, we have a vector of inputs that are projected onto an output layer of neurons, but not vice versa. This networks is strictly a feedforward type and we do not count the inputs as an independent layer because no computation is performed there [Haykin, 2007].

2) *Multilayer Feedforward Networks.* The second class of feedforward neural network distinguishes itself by the presence of one or more hidden layers. The function of neurons in hidden layers is to intervene between the external input and the network output in some useful manner. The source nodes in the input layer supply elements of the activation pattern, which constitute the input signals applied to the neurons in the second layer. The output signals of the second layer are used as inputs to the third layer, and so on for the rest of the network. The signals emerged from output neurons constitute the overall response of the network to the input pattern supplied in the source nodes in the input layer.

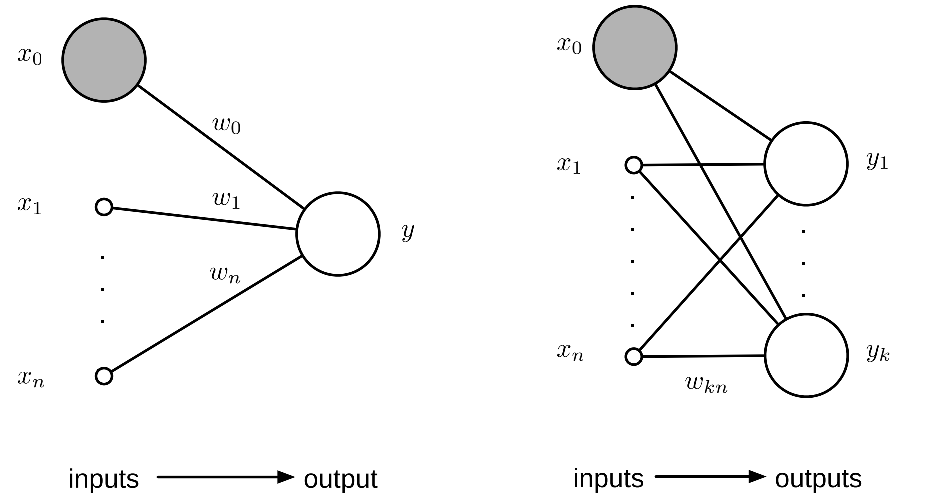
3) *Recurrent networks.* A recurrent neural network distinguishes itself from a feedforward neural network in that it has at least one feedback loop. The presence of feedback loops, have an impact on the learning capability of the network and on its performance. Feedback loops involve the use of particular

branches composed by unit-delay elements, which result in a nonlinear dynamic behavior.

2.2.3.1 Single Layer Feedforward Network

A single layer feed-forward consists of one or more output neurons, each one is connected with weighting factors to all inputs. The network having only just one neuron is the simplest form of neural network used for the classification of two linear separable classes. The single layer network represents a *linear discriminant function*.

By increasing the number of neurons in the output layer, we may classify with more than two classes. However, classes still need to be linearly separable for the ANN to work properly [Bishop, 2006, Theodoridis and Koutroumbas, 2008].



(a) One neuron, the simplest feedforward neural network.

(b) Single layer feedforward network of k neurons.

Figure 2.3: Single layer feedforward networks.

Figure 2.3 shows a graphical representation of two single layer feedforward networks, in Figure 2.3a we can see the simplest form of a feedforward neural network, while Figure 2.3b depicts one single layer neural network with k neurons. It is important to mention that we only count the output layer since inputs does not perform any operations.

2.2.3.2 Multi-Layer Feedforward Network

To solve more complicate and diverse classification problems, we have to go to networks with one or more hidden layers between the inputs and outputs of the system.

For example, a two-layer network (one hidden layer) with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units [Bishop, 2006].

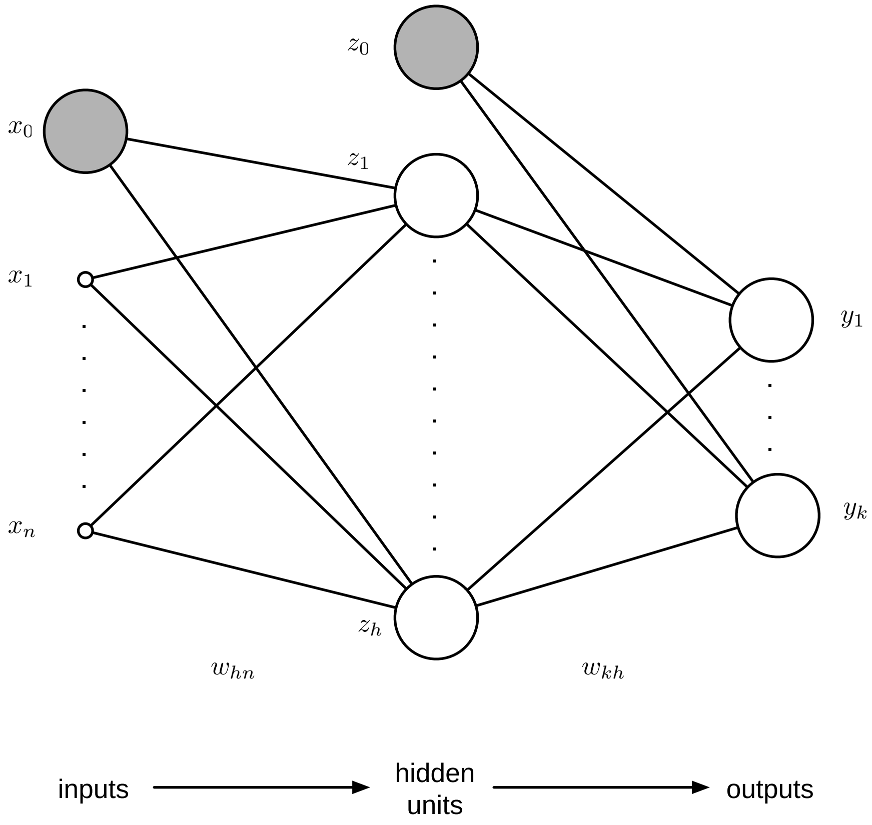


Figure 2.4: One hidden layer feedforward neural network.

Figure 2.4 depicts an ANN with one hidden layer. The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and fixed variables $x_0 = 1$ and $z_0 = 1$. Arrows denote the direction of information flow through the network during forward propagation.

There is some confusion in the literature regarding the terminology for counting the number of layers in an ANN. Thus the network in Figure 2.4 may be described as a 2-layer network (which counts the number of layers of units, and does not count the inputs as units) or may be described as a single-hidden-layer network (which counts the number of layers of hidden units). We decided to use the later terminology that is Figure 2.4 is called a one hidden-layer network.

2.3 Learning in Artificial Neural Network

The fundamental difference between a system that learns and one that merely memorizes is that a learning system generalizes to unseen samples. Much of our work is with supervised learning, getting a network to behave in a way that successfully approximates some specified pattern of behavior or input-output relationship.

The structure of the ANN as a classifier system is given along with a number of known parameters $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), \ldots, (\mathbf{x}_N, y_N)$. The input vector $\mathbf{x}$ is fed, one sample at a time, in sequence. A teacher stands beside the machine, observing the input $\mathbf{x}_i$, the expected output y_i and the classificator's output $\hat{y}_i$. When a discrepancy is observed between the expected output and the obtained output, the teacher notifies the machine, and the machine changes the parameters according to a predesigned algorithm. Historically, the earliest forms of supervised learning involved changing synaptic weights to oppose the error in a neuron with a binary output, or to minimize the sum of squares of errors of output neurons in a network with real-valued outputs.

When all samples in the training set have been used to modify the parameters then an iteration of learning is completed, this is called *epoch*. This learning process is repeated until some stop criteria is reached. This could be that the classificator has reached a good classification accuracy or the maximum number of epochs has been reached.

2.3.1 Delta rule

If we consider the case of a neuron k constituting the only node in the output layer of a feedforward neural network, such neuron is driven by a signal vector $\mathbf{x}^{(n)}$ produced by one or more layers of hidden neurons of the neural net. The argument n denotes discrete time or the time step of an iterative process involved in adjusting the synaptic weights of neuron k. The output is denoted by $y_k^{(n)}$. This neuron output, representing the only output of the neural network, is compared to a desired output, denoted by $d_k^{(n)}$. Consequently, an error, denoted by $e_k^{(n)}$ is calculate by

$$e_k^{(n)} = d_k^{(n)} - y_k^{(n)} \tag{2.10}$$

The error $e_k^{(n)}$ actuates as control mechanism, which purpose is to apply a sequence of corrective adjustments to the synaptic weights of neuron k. The corrective adjustments are designed to make the output signal $y_k^{(n)}$ come closer to the desired response $d_k^{(n)}$ in a step-by-step manner. This objective is achieved by minimizing a cost function $\xi^{(n)}$, defined in terms of the error output $e_k^{(n)}$ as:

$$\xi^{(n)} = \frac{1}{2} \left(e_k^{(n)}\right)^2 \tag{2.11}$$

That is, $\xi^{(n)}$ is the instantaneous value of error. The step-by-step adjustment to the synaptic weights of neuron k are continued until the network reaches a steady state. At that point the learning process is terminated.

Minimization of the cost function $\xi^{(n)}$ leads to a learning rule known as the *delta rule* or Widrow-Hoff rule, named in honor to his originators. In mathematical form delta rule describing adjustment to be applied to the synaptic weight is stated by:

$$\Delta w_{kj}^{(n)} = \eta \, e_k^{(n)} \, x_j^{(n)} \tag{2.12}$$

where $w_{kj}^{(n)}$ denotes the value of synaptic weight w_{kj} of neuron k excited by element $x_j^{(n)}$ of the measurements vector $\mathbf{x}^{(n)}$ at time step n. And η is a positive constant that determinates the rate of learning as we process from one step in the learning process to another.

Just in words delta rule can be stated as: *The adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse in question* [Haykin, 2007].

2.3.2 Back Propagation Algorithm

BP applies a delta rule modified version with the backpropagation of the error through previous layers of neurons. The BP algorithm consists of two pass through the different layers of the network: a forward pass and a backward pass. In the forward pass, an activity pattern is applied to the input nodes of the network, and its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. As expected during the forward pass the synaptic weights are all fixed whereas during the backward pass, the synaptic weights are all adjusted in accordance with an error-correction rule. Specifically, the actual response of the network is subtracted from desired output to produce an error signal, this error is propagated backward through the network. The synaptic weights are adjusted to make the actual response of the network move closer to the desired output.

In particular BP algorithm, works in the following way: the error signal of the output neuron j at iteration n (i.e. presentation of the nth training sample) is defined by

$$e_j^{(n)} = d_j^{(n)} - y_j^{(n)} \tag{2.13}$$

where $d_j^{(n)}$ is the desired response for neuron j and $y_j^{(n)}$ is the output obtained in neuron j at iteration n.

$$\xi^{(n)} = \frac{1}{2} \sum_{j \in C} \left(e_j^{(n)} \right)^2 \tag{2.14}$$

is the mean square error over all neurons in the output layer at iteration n, the set C includes all neurons in the output layer of the network. Hence,

$$\xi = \frac{1}{N} \sum_{n=1}^{N} \xi^{(n)} \tag{2.15}$$

is the average square error over all N samples. This average error is a function of all the free parameters (i.e. synaptic weights and bias) of the network. For a given **TS**, ξ represents the cost function as a measure of learning performance. The objective is to adjust the free parameters of the network that minimize ξ.

The minimization problem is tackled using a variant of *gradient descent*, where the solution moves down in the weight space according to the calculated slope of the error function with respect to each weight. Using calculus the slope is defined as the partial derivative of the error with respect to the weight vector $\partial \xi^{(n)} / \partial w_{ji}^{(n)}$. BP applies a correction $\Delta w_{ji}^{(n)}$ to the synaptic weight $w_{ji}^{(n)}$ which is proportional to partial derivative. This gradient represents a sensitivity factor, determining the direction of search in weight space for the synaptic weight w_{ji}. But when we only apply this rule, the weights from input to hidden units are never changed, so in order to adapt the weights from inputs to hidden units, we again want to apply the delta rule. In this case, however, we do not have a value for Δw for the hidden units.

In order to update the weights in the hidden neurons, one need to apply the chain rule. The chain rule does the following: distribute the error of an output unit e_i to all the hidden units that is connected to, weighted by this connection.

Former in words is, a hidden unit j receives a delta from each output unit i, this delta is weighted according to the connection weight between those units.

According to the chain rule, $\partial \xi^{(n)} / \partial w_{ji}^{(n)}$ is expressed as:

$$\frac{\partial \xi^{(n)}}{\partial w_{ji}^{(n)}} = -\, e_j^{(n)}\ \varphi_j^{'}\left(v_j^{(n)}\right)\ y_i^{(n)} \tag{2.16}$$

where $\varphi_j^{'}(\cdot)$ is the derivative of the activation function. The use of the minus sign accounts for gradient descent in the weight space. The local gradient $\delta^{(n)}$ is defined by

$$\delta_j^{(n)} = e_j^{(n)}\ \varphi_j^{'}\left(v_j^{(n)}\right) \tag{2.17}$$

Then, correction $\Delta w_{ji}^{(n)}$ applied to w_{ji} in BP algorithm is defined by the *delta rule* as:

$$\Delta w_{ji}^{(n)} = \eta\ \delta_j^{(n)}\ y_i^{(n)} \tag{2.18}$$

where η is the learning-rate parameter of the BP algorithm. From Equation 2.17 we can state that, the local gradient for an output neuron j is equal to the corresponding error $e_j^{(n)}$ for that neuron and the derivative $\varphi_j^{'}\left(v_j^{(n)}\right)$ of the associated activation function.

We can apply delta rule in two cases: when j is an output node or when j it is a hidden node.

Neuron j is an output node. When neuron j is located in the output layer, it is supplied with a desired response of its own. We can use Equation 2.13 to compute error signal associated with this neuron; having determined $e_j^{(n)}$, it is straightforward compute gradient $\delta_j^{(n)}$ using Equation 2.17 and compute $\Delta w_{ji}^{(n)}$ with Equation 2.18.

Neuron j is a hidden node. When neuron j is located in a hidden layer, there is no specified desired response for that neuron. Error for a hidden neuron needs to be determined recursively in terms of the error of all the neurons to which that hidden neuron is directly connected. In this case the local gradient $\delta_j^{(n)}$ for hidden neuron is defined as:

$$\delta_j^{(n)} = \varphi_j^{'}\left(v_j^{(n)}\right) \sum_k \delta_k^{(n)}\ w_{kj}^{(n)}. \tag{2.19}$$

The factor $\varphi_j' \left(v_j^{(n)} \right)$ depends solely on the activation function associated with hidden neuron j. The remaining factor, namely the summation over k, depends on two sets of terms. The first set of terms, the $\delta_k^{(n)}$, requires knowledge of the errors $e_k^{(n)}$ for all neurons that lie in the layer to the immediate right of hidden neuron j, and that are directly connected to neuron j. The second set of terms, the $w_{kj}^{(n)}$, consists of the synaptic weights associated with these connections.

To summarize, BP algorithm specifies that the correction applied to the synaptic weight connecting neuron i to neuron j is defined by Equation 2.18, where η is the learning rate parameter, $\delta_j^{(n)}$ is the local gradient and, $y_i^{(n)}$ is the input signal of neuron j. The local gradient depends on whether neuron j is an output node (use Equation 2.17 to compute it) or a hidden node (use Equation 2.19).

2.3.2.1 Momentum

The smaller we make the learning-rate parameter η, the smaller is the change of the network weights from one iteration to the next. On the other hand, if η is too large, in order to speed up the rate of learning, then it is possible to obtain an unstable network. In order to accelerate the learning rate reducing the risk of having an unstable network, Rumelhart proposed to include a *momentum* term to the delta rule. This is

$$\Delta w_{ji}^{(n)} = \alpha \, \Delta w_{ji}^{(n-1)} + \eta \, \delta_j^{(n)} \, y_i^{(n)} \tag{2.20}$$

where α is called *momentum constant*. The idea is to make the new change of weights large if it is in the direction of the previous change of weights., while if it is in a different direction make it smaller. Clearly the momentum must be between 0 and 1. Equation 2.20 is called *generalized delta rule.*

2.3.2.2 Sequential and Batch modes of training

For a given training set, BP may be applied in one of two basic ways:

1) *Sequential mode.* This is also known as on-line, pattern, stochastic or incremental mode. In this mode the update of the weights is performed after the presentation of each training sample. Usually, In this mode the patterns are presented to the network randomly.

2) *Batch mode.* In this mode, weight updating is performed after all the training samples have been seen by the net. It is common to use Equation 2.15 as cost function.

2.3.2.3 Stopping criteria

In general, BP algorithm does not have well defined criteria for stopping its operation [Haykin, 2007]. Nevertheless, there are some popular techniques that have been used to stop iterative learning while BP is applied:

- When the rate of change in the average squared error is small.
- The mean square error is sufficiently small.
- When the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.
- After accomplish a prefixed number of epochs.
- When the generalization performance is adequate.

2.3.2.4 Weights Initialization

It is difficult to choose the initial value of the weights so that they are as close as possible to the global minimum in the weight space. Since *a priori* knowledge about the global minimum is limited, the initial weights must be estimated. It is common practice to initialize randomly the weights with small values, for example, between -0.5 and 0.5 [Gupta et al., 2003].

2.3.3 General Sequential BP Algorithm

The well know BP algorithm for sequential training is listed in Algorithm 2.1, as explained before BP is an iterative learning algorithm where training is accomplished executing two phases, forward and backward; in the algorithm steps 4 and 5 form the forward phase and steps 6 to 9 form the backward phase; the two initial steps are initialization requirements.

Algorithm 2.1 Sequential back propagation algorithm.

```
Initialize weights randomly taking values from the interval $[-0.5, 0.5]$
Set learning rate $\eta \in (0,1)$ and momentum $\alpha \in (0,1)$
repeat
    From TS choose a tagged input pattern $\mathbf{x}_i$ to apply it to the input layer
    Propagate pattern forward in order to calculate ANN's output $\mathbf{y}_i$
    Calculate error between desired output and obtained; $e_j = d_j - y_j$
    Calculate delta on the output layer; $\delta_j = e_j \; \varphi_j \left(v_j\right)$
    Calculate delta for the hidden layer; $\delta_i = \varphi_i^{'} \left(v_i\right) \sum_j \delta_j \; w_{ji}$
    Update all weights according to $\Delta w_{ji}^{(n)} = \alpha \; \Delta w_{ji}^{(n-1)} + \eta \; \delta_j \; y_i$
until Stop criteria is reached
```

2.3.4 BP derivatives

Since BP is based on gradient descent and this is inherently a slow convergence method to find the minimum of the error function, many researchers have developed improvements and extensions to the basic BP algorithm.

2.3.4.1 Resilient Backpropagation

By 1990's Riedmiller[Riedmiller and Braun, 1992, Riedmiller and Braun, 1993] developed RPROP, where the direction of each weight update is based on the sign of the partial derivative $\partial\xi^{(n)}/\partial w_{ji}^{(n)}$. The idea is to eliminate the harmful influence of size of the partial derivative on the weight step. Only the sign of the derivative is considered to indicate the direction of the weight update. The size of the weight change is determined by a *update-value* $\Delta_{ji}^{(n)}$:

$$\Delta w_{ji}^{(n)} = \begin{cases} -\Delta_{ji}^{(n)}, & \text{if } \frac{\partial\xi^{(n)}}{\partial w_{ji}^{(n)}} > 0 \\ +\Delta_{ji}^{(n)}, & \text{if } \frac{\partial\xi^{(n)}}{\partial w_{ji}^{(n)}} < 0 \\ 0, & \text{otherwise} \end{cases}$$

And the update-value is determined on a sign independ adaptation process [Riedmiller, 1994]:

$$\Delta_{ji}^{(n)} = \begin{cases} \eta^{+} \cdot \Delta_{ji}^{(n-1)}, & \text{if } \frac{\partial\xi^{(n-1)}}{\partial w_{ji}^{(n)}} \frac{\partial\xi^{(n)}}{\partial w_{ji}^{(n)}} > 0 \\ \eta^{-} \cdot \Delta_{ji}^{(n-1)}, & \text{if } \frac{\partial\xi^{(n-1)}}{\partial w_{ji}^{(n)}} \frac{\partial\xi^{(n)}}{\partial w_{ji}^{(n)}} < 0 \\ \Delta_{ji}^{(n-1)}, & \text{otherwise} \end{cases}$$

where $0 < \eta^{-} < 1 < \eta^{+}$.

2.3.4.2 Improved RPROP

Despite RPROP is one of the best performing first-order learning methods for neural networks, in 2000 year Igel proposed iRPROP [Igel and Hüsken, 2000] as a modified version of original algorithm RPROP.

Two versions of iRPROP was stated, $iRPROP^{+}$ and $iRPROP^{-}$; the first performs weight-backtracking in the cases where overall error increased and always sets the derivative $\partial\xi^{(n)}/\partial w_{ji}^{(n)}$ to zero; the second performs the same as $iRPROP^{+}$ without weight-backtracking [Igel and Hüsken, 2003].

2.3.4.3 QuickProp

Fahlman [Fahlman, 1988] developed a BP based heuristic which takes into account the curvature of the error surface at any point by defining:

$$\Delta w_{ji}^{(n)} = \begin{cases} \alpha_{ji}^{(n)} \cdot \Delta w_{ji}^{(n-1)}, & \text{if } \Delta w_{ji}^{(n-1)} \neq 0 \\ \eta \cdot \frac{\partial\xi^{(n)}}{\partial w_{ji}^{(n)}}, & \text{if } \Delta w_{ji}^{(n-1)} = 0 \end{cases}$$

where

$$\alpha_{ji}^{(n)} = \mathbf{min}\left(\frac{\frac{\partial\xi^{(n)}}{\partial w_{ji}^{(n)}}}{\frac{\partial\xi^{(n-1)}}{\partial w_{ji}^{(n)}} - \frac{\partial\xi^{(n)}}{\partial w_{ji}^{(n)}}}, \alpha_{max} \right)$$

Quickprop update weights once after each presentation of the entire set (batch training) and requires the storage of four values for every weight: the weight itself, the slope accumulated for the current training epoch, the previous delta, and the previous slope. If the goal is to use the minimum number of memory bits during training, a simple gradient descent algorithm may be superior to quickprop [Hoehfeld and Fahlman, 1992].

2.4 Generalization, Accuracy, and OverFitting

ANN generalize well it appropriately classifies items that are not included in the training set. Generalization ability is measured by the accuracy of these classifications. In order to understand more easily generalization we can make an analogy with a curve fitting. That is, we try to approximate a set of points with a straight line or a low-degree polynomial curve. At this point, we are confident that we have captured some underlying relationship in the data if the fit to the data is very good and if there is a lot of data. The fitted curve can then be used to estimate (with reasonable reliability) values for new data points not used earlier in the fitting process. If a straight line does not fit the data well, then perhaps we might try a second degree curve, and so on.

A similar story can be told for ANN. ANN computes a complex, nonlinear function of its inputs. If the classification of these inputs is in fact some function that is close to some member of the set of functions implementable by the network, and if the fit by a trained network on the data is very good for a large number of inputs, then it is likely that the training process has captured the underlying relationship between inputs and outputs. In this case, generalization to new inputs should be good [Nilsson, 1998].

A classificator accuracy usually is measured in both training set and validation set. A common measure is the number of samples correctly classified in those sets. For instance, if the training set has 500 members and the validation set only 100, and our classification system correctly classifies 490 and 95 respectively, then our classificator has an accuracy of 98% in the training set, and accuracy of 95% in the validation set.

Even if a training set error is low, generalization might not be good. This error is called in statistics the out-of-sample-set error rate. If a classifier present low error rate in TS but high error rate in VS means that the classificator has learnt very well the training samples but generalization has been lost. This is called *overfitting*, and scientists have been developed some technics to avoid this situation.

Perhaps the simplest technique is to divide the input vectors available for training into two disjoint sets and use one of these for training. After training is finished, we use the other set for estimating the out-of-sample error rate. When we do not have enough data, one is forced to use another technique to fight overfitting. One of these techniques is *cross validation*. In this method we divide the vectors available for training into k disjoint subsets, called *folds*. We select one of these folds as a validation set and use the other $k - 1$ as a training set. We do this k times, each time selecting a different fold as a validation set and its complement as the training set. We compute the error rate for each validation set (after training on its complement) and take the average of these error rates as the estimate of the out-of-sample error. Experimental results suggest that taking k from 5 to 10 gives reasonable estimates of generalization. For the special case of $k = m$ where m is the number of labeled vectors available, we have what is called *leave-one-out* cross validation.

2.5 Concluding Remarks

The essence of BP learning algorithm is to encode an input-output mapping into the synaptic weights and thresholds of a multilayer perceptron. The hope is that the network becomes well trained so that it learns enough about the past to generalize to the future events.

Chapter 3

Genetic Algorithms

Charles Darwin proposed the theory of natural evolution in the origin of species [Darwin, 1909]. Over several generations, biological organisms have evolved to reach certain remarkable features based on the principle of natural selection, i.e., survival of the fittest. Darwinian evolution is intrinsically a search and optimization mechanism. Evolved organisms demonstrate optimized complex behavior at each level: the cell, the organ, the individual and the population. Biological species have solved the problems of chaos, chance, nonlinear relationships, and temporality. These problems proved to be in equivalence with the classic methods of optimization [Sivanandam and Deepa, 2008]. The evolutionary concept can be applied to problems where traditional optimizations techniques have been unable to produce solutions or these are unsatisfactory.

The theory of natural selection proposes that the plants and animals that exist today are the result of millions of years of adaptation to the demands of the environment. At any given time, a number of different organisms may coexist and compete for the same resource in an ecosystem. Due to this, poorly performing individuals have less chance to survive, and the most adapted or *fit* individuals produce a relatively large number of offsprings. It can also be noted that during reproduction, a recombination of the good characteristics of each ancestor can produce *better fit* offspring. After some generations, species evolve to become more and more adapted to their environment.

3.1 Biological Background

The science that deals with the mechanisms responsible for similarities and differences in species is called Genetics. The word *genetics* is derived from the Greek word *genesis* meaning origin. The science of genetics helps us to differentiate between heredity and variations, genetics seek to account for the resemblances and differences. The concepts and mechanisms implemented in GA are directly derived from natural evolution. The main terminologies involved in the biological background of species are described in genetics as follows:

Every human/animal *cell* is a complex of many "micro" factories that work together. The center of all this is the cell nucleus. The genetic information is contained in the cell nucleus.

All the genetic information gets stored in well defined structures know as *chromosomes*. Each chromosome is build of Dioxy Ribo Nucleic Acid (DNA). In humans, a chromosome exists in the form of pairs (23 pairs found). The chromosomes are divided into several parts called genes. *Genes* code the properties of species i.e., the characteristics of an individual. The set of all the genes of a specific species is called *genome*. Each and every gene has an unique position on the genome called *locus*, *allele* is one of alternative forms of a gene. Most living organisms store their genome on two or more chromosomes copies (diploidism and polyploidism respectively), but in the GA, all the genome information usually are stored on the same chromosome (haploidism). Thus chromosomes and genomes are synonyms with one other in GA.

For a particular individual, the entire combination of genes is called *genotype*. The *phenotype* describes the physical aspect, the characteristics of the individual are interpreted through the decoding of genes. One interesting point of evolution is that selection is always done on the phenotype whereas the reproduction recombines genotype. *Morphogenesis* means form of living organisms, it is the procedure to transform individual's genotype to a certain characteristics, this mapping between genotype and fenotype depends directly of the encoding used and the meaning of each gene. Morphogenesis plays a key role between selection and reproduction.

Reproduction of species via genetic information is carried out by, mitosis and meiosis. In *Mitosis* the same genetic information is copied to new offspring. There is no exchange of information. This is a normal way of growing of multi cell structures, like organs. *Meiosis* forms the basis of sexual reproduction. When meiotic division takes place two gametes appear in the process. When reproduction occurs, these two gametes conjugate to a zygote which becomes the new individual (*offspring*).

The origin of species is based on preservation of favorable variations and rejection of unfavorable ones. The variation refers to the differences shown by the individual of a specie and also between the parents and its offspring. Individuals better adapted in living environment have a greater chance of survive (better fit). Qualify individuals in the population takes importance because better individuals usually have better chances to heritage genetic information to their offsprings, as a result, *natural selection* plays a major role in this survival process.

3.2 Overview of GA

Evolutionary computation (EC) techniques abstract biological evolutionary principles into algorithms that may be used to search for solutions to a problem. In EC, there are four historical paradigms that have served as the basis of much of the activity in the field: genetic algorithms [Holland, 1975], genetic programming [Koza, 1992], evolutionary strategies [Rechenberg, 1973], and evolutionary programming [Fogel et al., 1966]. The most popular technique in evolutionary computation research has been the genetic algorithm.

In the mid 70's Holland developed natural evolution's ideas in his book "Adaptation in natural and artificial systems". He described how to apply the principles of natural evolution to optimization problems and built the first Genetic Algorithms. In this book the term *adaptation*, the ability to adapt, plays a central role and is defined as a process of progressive modification of structures, which leads to improve performance of the system interacting with its environment.

Genetic algorithms glean ideas from natural mechanisms of reproduction, the four essential principles manifest in nature are the core ingredients of genetic algorithm [Jacob, 2001] : (1) the *dualism* principle of separating genetic information of the genotype the expressed phenotype, (2) a *discrete encoding* of genotypical structures, (3) *recombination effects* resulting from sexual reproduction, and (4) *elementary buildings blocks*, which are combined according to specific templates, help in the composition of complex interacting systems, representing a core precondition of modular design and construction.

Dualism: In biological systems, the genetic information encoded in DNA is used in two ways, as genetic information, which is replicated, and as instructions, which have to be executed. In GA, the genotypical structures, modified by an evolutionary algorithm through recombination and mutation, are clearly separated from the phenotypical structures. The evolution is not performed in the space of the parameters to be optimized but in an encoding, genotypical structure space. The principal separation of phenotype and genotype in nature is explicitly implemented by genetic algorithms.

Discrete encoding: In DNA strands, genetic information is encoded by a four letter alphabet. For GA structures a binary string representation is used, but any other encoding scheme over a discrete, finite alphabet may be chosen as well.

Recombination effects: The cells of sexually reproducing individuals consist of a double set of homologous[1] chromosomes, each half from the offspring's mother and father. Paired chromosomes are mutually exchanged by crossover, the resulting descendant cells contain a slightly mutated chromosome set, merged from the genetic information of both parental cells. GA replicate these recombination effects and integrate them into evolutionary simulation models and optimization algorithms.

Elementary building blocks: Complex adaptive systems, such as the interactions of genome structures, are hierarchically composed of simple elementary units. This modular organization principle plays a decisive role in the encoding structures of genetic algorithms as well. The genotypical GA structures are also built from elementary buildings blocks: short binary strings, command sequences, variables, constants, or symbolic expressions.

3.3 Terminologies in GA

3.3.1 Individuals

An individual is a single possible solution. Following nature fenotype/genotype dualism, GA implements in individual two forms of solutions:

- The chromosome (genotype), which is the genetic information that the GA deals with.

[1] Chromosomes having the same length and same gene coding but possibly different alleles

- The phenotype, which is the representation of the chromosome in terms of the model.

Figure 3.1 represents the nature dualism implemented in GA, where bottom plane are the genotypical space and upper plane is the phenotypical space, every individual has one genotype and one phenotype.

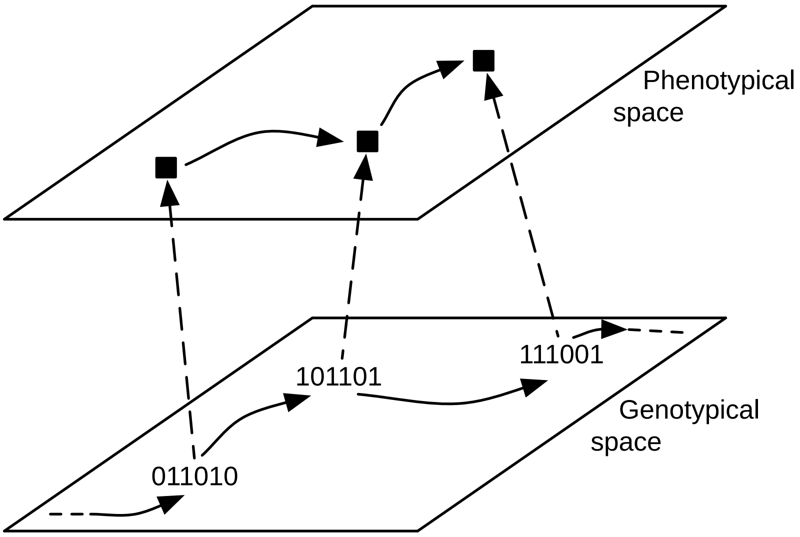


Figure 3.1: Genotype and fenotype dualism as a nature principle in GA.

3.3.2 Chromosome (Genotype)

A chromosome is a sequence of encoded genes. A chromosome contains in some way, information about the solution that it represents. The morphogenesis function associates each genotype with its phenotype. It simply means that each chromosome must define one unique solution. Figure 3.2 shows a typical chromosome used in GA, also at bottom part in Figure 3.1 (genotypical space) a set of chromosomes are represented as binary strings.

1	0	0	1	0	1	0	0	1	1

Figure 3.2: GA chromosome.

3.3.3 Phenotype

Phenotype is the form of the chromosome in the environment, the modifications are performed in the genotypical space and qualifying/selection take place in phenotypical space. Genotype and phenotype form a dualism and a GA evolves individuals by applying operators in both levels. Upper part of Figure 3.1 shows phenotypical space, each filled square represent one phenotype, bottom part represents genotypical space.

3.3.4 Genes

A chromosome is subdivided into genes. A Gene describes a part of a possible solution to a problem, without actually being the complete solution. The structure of each gene is defined in a record of phenotyping parameters. The phenotype parameters are instructions that map from genotype to phenotype. It can also be said as encoding a solution set into a chromosome and decoding a chromosome to a solution set. In Figure 3.2, we can see a binary chromosome form by ten genes, each square (gene) contains one of two possible alleles, 0 or 1.

3.3.5 Morphogenesis

The mapping between genotype and phenotype is necessary to convert solution sets from the model to a structure that a GA can work with, and new individuals from the GA to a form that the model can evaluate. This is called Morphogenesis. In Figure 3.1 dashed arrows represent morphogenesis of each individual.

3.3.6 Fitness

The fitness of an individual in a GA is the value assigned by measuring the aptitude of how good a phenotype in the environment. For computing fitness, the chromosome is first decoded and then the fitness function assigns a number according to their characteristics. In a nutshell the fitness indicates how good the solution is.

3.3.7 Population

A population is a collection of individuals. The two important characteristics of a GA population are: the initial population and the population size. It is common to use a random initial population, but there may be instances where the initial population is built with some known good solutions.

In Figure 3.3 a random population of n individuals is represented as binary strings of ten genes.

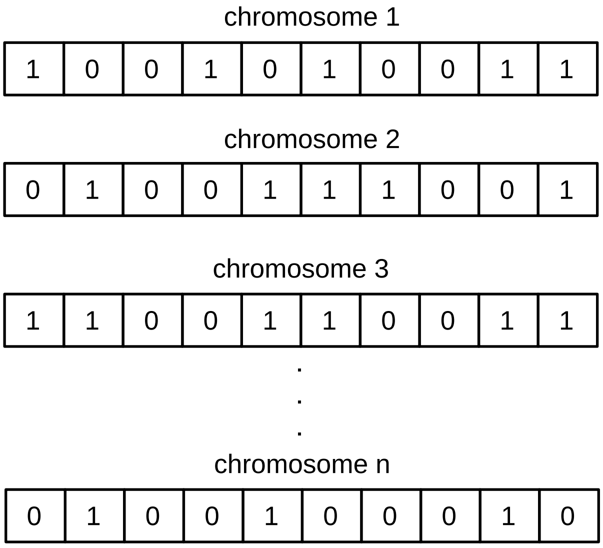


Figure 3.3: Genetic algorithm population.

3.3.8 Encoding

Encoding is a process of representing individual genes into the chromosome. The process can be performed using bits, numbers, trees, arrays, lists or any other object. The encoding depends mainly on the kind of problem. For example, in a continuous problem would make more sense to encode a real number whereas in a discrete problem integers or a binary string would be more appropriate.

3.4 Genetic Operators

For GA to find an optimum solution, it is necessary to perform certain operations over the individuals. This section discusses the basic operators used in GA to achieve a acceptable solutions.

The breeding process is the heart of the genetic algorithm. It is in this process, where new and hopefully fitter individuals are created. The breeding cycle consists of four steps:

I. Selecting parents for crossover.

II. Crossover the parents to create new individuals.

III. Mutating offsprings.

IV. Replacing old individuals in the population with the new ones.

3.4.1 Selection

Selection is the process of choosing parents from the population for crossover. The selection criteria used is a critical factor that determines the success of failure of an evolutionary algorithm given that only those individuals are the ones that pass their genetic information from one generation to the next. GA follows a criterion which is closely related to selection among natural organisms.

As it is in nature, the purpose of selection is to emphasize fitter individuals in the population with the hope that their offsprings would have better fitness. Selection is a method that picks chromosomes out of the population according to their evaluation function. The better the fitness function, the more chance an individual can be selected.

The most common methods of selection are: fitness proportionate selection, rank based selection and tournament selection. Proportionate selection picks out individuals based on their fitness values, the better individuals have greater probability of being selected. Rank based selection schemes selects individuals base on their rank within the population. In tournament selection, n individuals are selected randomly from the population in order to compete each other, the winner is the one with the best fitness and this is selected.

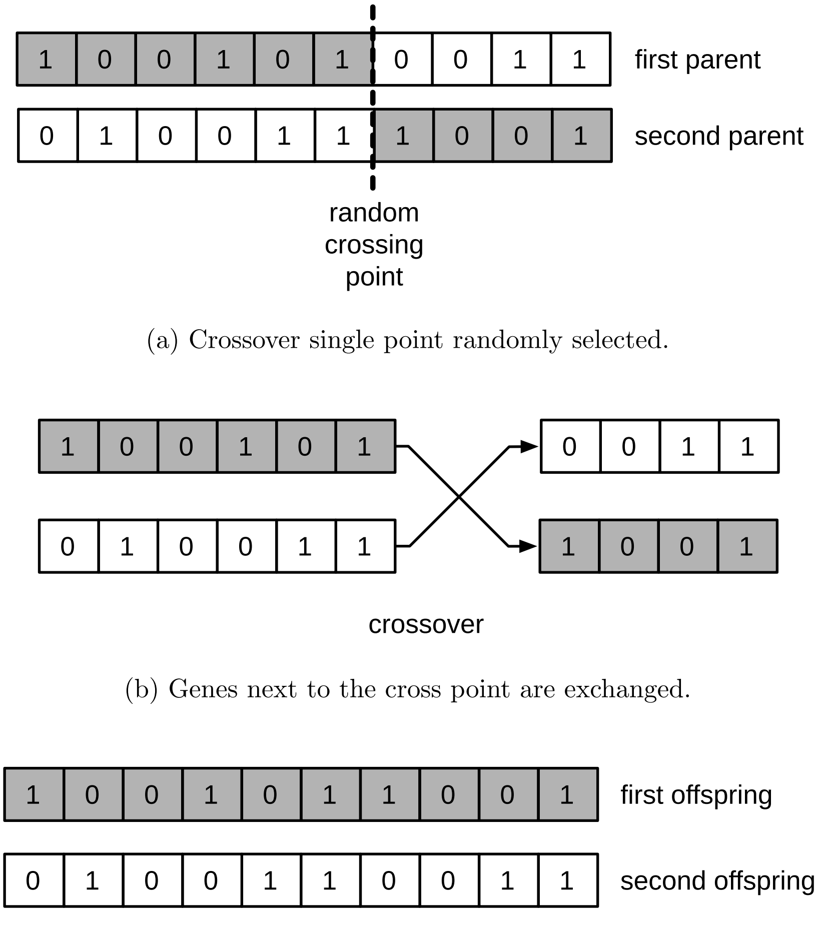
3.4.2 Crossover (Recombination)

Crossover is the process of taking some parents to produce from them some offsprings. The basic parameter in crossover is the *crossover probability* (Pc). This parameter describes how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome.

There are many ways to perform crossover, the simplest one is called single point crossover, this involve two parents and one single point of crossover; others common crossover approaches are: two points, multipoint, uniform and three parents.

Single Point: In this crossover approach one crossover point is selected randomly along the length of the mated chromosomes and genes next to the cross-site are exchanged for getting two offsprings.

Figure 3.4 shows a single point crossover in a pool of two mating binary chromosomes. One random crossover point is selected (3.4a), then the genes next to the cross point are exchanged (3.4b) and finally, two new individuals (offsprings) are generated (3.4c).



(a) Crossover single point randomly selected.

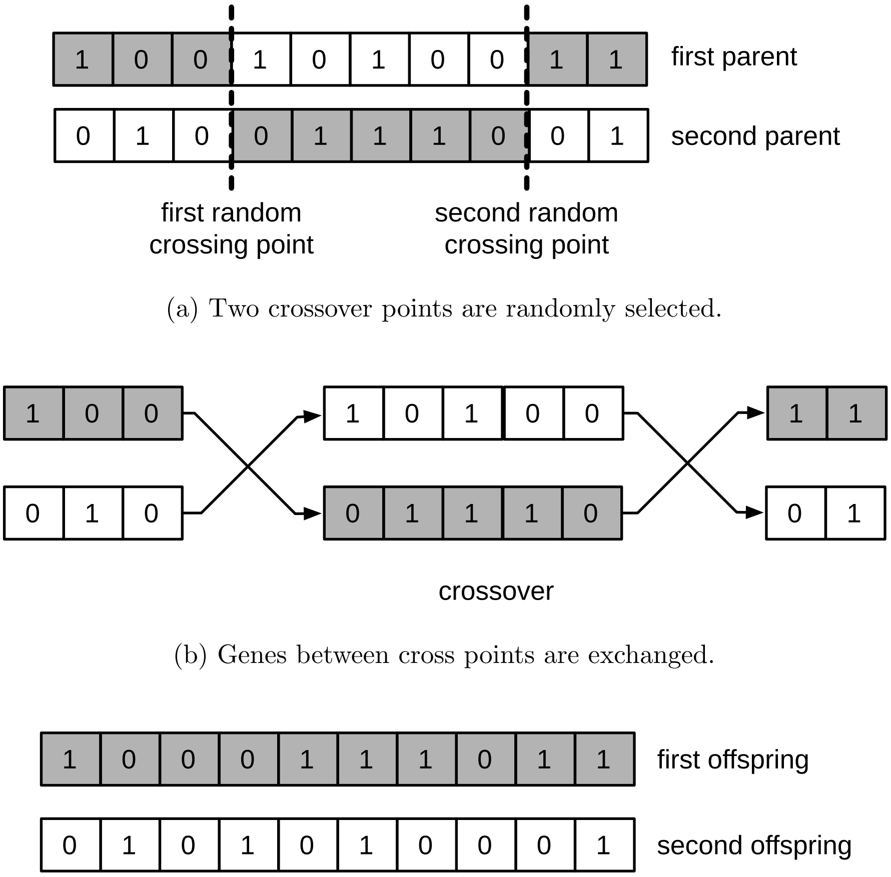
(b) Genes next to the cross point are exchanged.

(c) Two resulting offsprings.

Figure 3.4: Single point binary chromosome crossover.

Two points: In two-point crossover, two crossover points are chosen and the contents between these points are exchanged between two mated parents.

In Figure 3.5 the dotted lines indicate the crossover points (3.5a). The content between these points are exchanged among the parents (3.5b) to produce new children (3.5c) for mating in the next generation.



(a) Two crossover points are randomly selected.

(b) Genes between cross points are exchanged.

(c) Two resulting offsprings.

Figure 3.5: Two points binary chromosome crossover.

Multipoint: This approach can be accomplish in two ways. One is even number of cross-sites and the other odd number of cross-sites. In the case of even number of cross-sites, cross-sites are selected randomly around a circle and information is exchanged. In the case of odd number of cross-sites, a different cross-point is always assumed at the string beginning.

Uniform: Each gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a random generated binary crossover mask of the same length as the chromosomes. Offsprings, therefore contain a mixture of genes from each parent.

Three parents: Three parents are randomly chosen. Each bit of the first parent is compared with the bit of the second parent. If both are equal, the bit is taken otherwise; the bit from the third parent is taken, for the offspring.

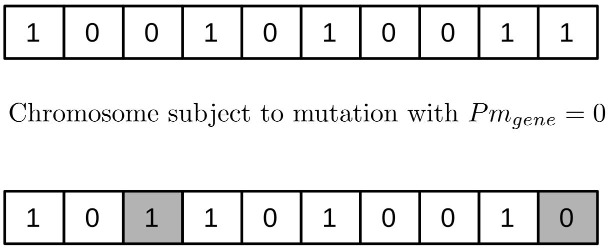
3.4.3 Mutation

After crossover, the chromosomes are set to mutation. Mutation plays the role of recovering the lost genetic materials as well as for randomly disturbing genetic information. It is an insurance policy against the irreversible loss of genetic material [Sivanandam and Deepa, 2008]. If crossover is supposed to *exploit* the current solution to find better ones, mutation is supposed to help for the *exploration* of the whole search space.

There are many different forms of mutation for the different kinds of representation. For binary representation, a simple mutation can consist in inverting the value of each gene with a small probability.

Mutation has two parameters: the mutation probability (Pm) and the gene mutation probability (Pm_{gene}). The mutation probability decides how often chromosome will be mutated. Once this has been determined, Pm_{gene} is used as mutation point probability or mutation gene probability, for deciding if a single gene should be mutated or not.

Figure 3.6 depicts the mutation of a binary chromosome, where the bits are slipped in the locus 3 and 10.



(a) Chromosome subject to mutation with $Pm_{gene} = 0.2$.

(b) Resulting chromosome after mutation.

Figure 3.6: Binary chromosome mutation.

3.4.4 Replacing individuals

Replacement is the last stage of any breeding cycle. Two parents are drawn from a fixed size population, traditionally they breed two children, but not all four can return to the population, so two must be replaced in order to keep the population size constant. Basically, there are two kinds of methods for updating the population; generational updates and steady state updates.

The generational scheme consists in producing μ children from a population of size μ to form the population at the next time step (generation), and this new population of children completely replaces the parent selection. Clearly this kind of update implies that an individual can only reproduce with individuals from the same generation. We called this method comma and use a symbol "," to reference it.

Of course, there exists slightly different versions of generational update, for instance, from a parent population of size μ, λ offsprings are generated. Then the μ best individuals from either the offspring and parent form the next generation. We called this method plus and use a symbol $+$ to reference it.

The insertion of a new individual usually requires the replacement of another population member. The individual to be replaced can be chosen as the worst member of the population. (it leads to a very strong selection pressure), or as the oldest member of the population, but those method are quite radical.

Generally, steady state updates use traditional selection for both creation of the mating pool and the replacement, usually a tournament method. Tournament replacement is exactly analogous to tournament selection except the less fit solutions are picked as the winner in the tournament.

3.5 Genetic Algorithms

GA evolution starts with the creation of the initial population of μ individuals. After interpretation (decoding) and evaluation of the individuals, the population enters a selection-crossover-mutation cycle, which is performed until a stop criteria is reached. Operators applied in this cycle are constrained to manipulate the individuals in a manner consistent with the structural interpretation of genes.

The standard GA procedure is presented in Algorithm 3.1:

Algorithm 3.1 Genetic algorithm.

```
Initialize the population
Calculate fitness for each individual in the population.
repeat
    Select individuals for mating
    Perform crossover to form offsprings
    Mutate the offsprings
    Calculate the fitness of new individuals
    Decide which individuals will form the new generation
until Stop criteria reached
```

3.6 Concluding Remarks

In this chapter, we have presented the overall process followed by genetic algorithm, as well as reviewing its main operators.

GA is an iterative approach based on Darwin's evolution theory; in where, adaptation of the fittest individual, brings better possibilities of survival to his descendants.

GA works with a population of individuals, each individual is a potential solution. For each generation performs three main operations, selection, crossover and mutation. Selection, picks individuals for crossover according to their fitness. Crossover, mate individuals in order to generate new ones. Finally, mutation, apply small changes to the generated individuals. Those three operations are performed iteratively until the stop criteria is reached.

Chapter 4

Evolving Artificial Neural Networks

Genetic-Evolutionary Approach for Neural Networks, GEANN, is the approach proposed in this thesis. GEANN evolves ANNs in order to design an accurate classifier for a given tagged dataset, GEANN performs one preliminary arithmetic adjustment (data scaling) and carry out three stages to accomplish the final trained classifier ANN.

This evolutionary approach aims to automate ANN's design for classification, it attempts to achieve this by carrying out three simple stages. Figure 4.1 shows the flow chart of 3 stages that form the proposed approach. The procedure begins with a given tagged **TS**, this dataset contains severals samples where $x_i \in \mathbb{R}$. The preprocessing step is an affine transformation, where the observation measurements domain is translated from the original domain $\mathbb{R}$ to the interval $[-1, 1]$.

Once the dataset has been scaled, the first phase uses a GA to design the ANN's topology, to select the features, and to select a training algorithm among: BP, BP batch, QuickPROP, and iRPROP. Relying on a K-fold cross validation technique, the second phase determines the number of epochs that the learning algorithm is applied to the evolved ANN in order to avoid overfitting. The final phase trains the evolved ANN applying the learning algorithm as many epochs as they were determined in the second phase. GEANN returns a fully designed and trained ANN to classify multidimensional samples as those presented in the given dataset.

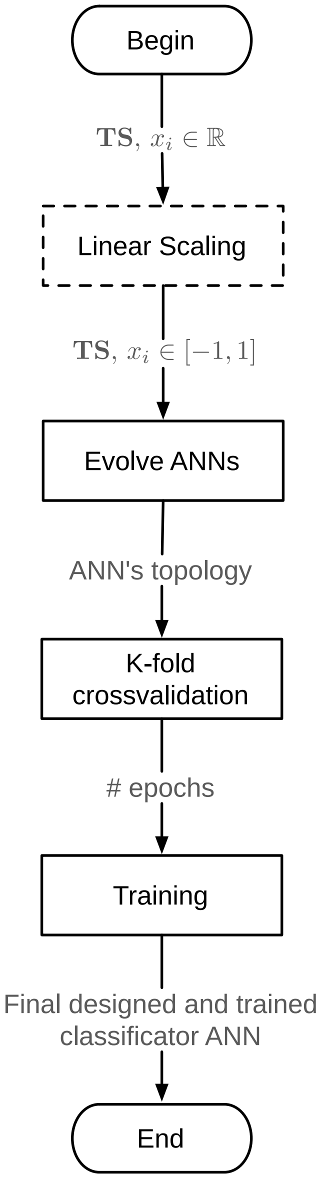

Begin
$\mathbf{TS}$, $x_i \in \mathbb{R}$
Linear Scaling
$\mathbf{TS}$, $x_i \in [-1, 1]$
Evolve ANNs
ANN's topology
K-fold crossvalidation
# epochs
Training
Final designed and trained classificator ANN
End

Figure 4.1: Flowchart of GEANN, the proposed approach for automatic design of artificial neural network classifiers by means of genetic algorithms.

A detailled explanation of the proposed approach is presented along this chapter.

4.1 Data Linear Scaling

Since learning takes place in the form of an iterative process, through the training algorithm, it is a good idea to transform the data before sending it to the GA. This pre-processing of information, is a simple linear rescale. One needs to be careful in rescaling, though, since it may transform a feature from one value to another one, and does not reflect the potential importance of each characteristic [Bishop, 2002].

By applying a linear transformation all inputs are rearranged, and get a respective value between -1 to 1. So, for each characteristic the new minimum value of all samples will be -1 and the new maximum value 1. To do this, we treat each of the input variables independently, and for each feature x_i we recalculate its new value by the application of Algorithm 4.1. First, for each measurement, minimum and maximum values are found with respect to the training set. Then, compute the scale factor as $factor = new\_span/span$, where $span = max - min$, and $new\_span = new\_max - new\_min$, since new interval is $[-1, 1]$ then, $new\_span = 2$. All observations are recalculated with $new\_data = (x_i - min) * factor + new\_min$.

Algorithm 4.1 Data linear scaling algorithm.

```
Require: TS with $N$ samples, $D$ features per sample, and $x_i \in \mathbb{R}$
for $j \leftarrow 1$ to $D$ do
    $min_j, max_j \leftarrow \mathbf{min}(x_j), \mathbf{max}(x_j)$
    $span_j \leftarrow max_j - min_j$
    $factor_j \leftarrow 2/span_j$
    for $i \leftarrow 1$ to $N$ do
        $x_{ij} \leftarrow (x_{ij} - min_j) * factor_j - 1$
    end for
end for
return TS, $x_i \in [-1, 1]$
```

Through the application of Algorithm 4.1 we get one array of factors and one array of minimums, one factor and one minimum for each feature. Using those values, same preprocessing is done for new unseen samples of **VS**.

4.2 ANN Evolution

ANN are evolved using a GA, throughout evolution process many ANN's characteristics are found, such as the number of input neurons, which inputs are relevants, the number of neurons in the hidden layer, the activation function for each layer, and the training algorithm.

There are three main activities to realize when working with genetic algorithms, design chromosome representation, test crossover and mutation operations, and design the fitness function.

4.2.1 Chromosome Description

The chromosomes used in this work are binary strings of fixed length for each problem. The chromosome has been designed in order to faithfully represent one hidden layer ANN and also has been formed according to the canonical GA, so this allows us to use traditional operations without lose of representativity for individuals.

Since the proposed approach is intended to tackle any tagged **TS**, then chromosomes are able to represent a wide variety of networks sizes. GA's individuals will have a variable number of input and hidden neurons, besides, chromosome are able to specify one activation function for each layer from a set, and also specify one training algorithm.

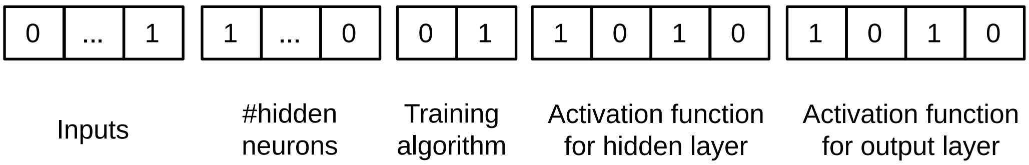

0
...
1
1
...
0
0
1
1
0
1
0
1
0
1
0
Inputs
#hidden neurons
Training algorithm
Activation function for hidden layer
Activation function for output layer

Figure 4.2: Binary chromosome used to represent one hidden layer artificial feedforward neural network.

Figure 4.2 shows the chromosome structure used in this thesis. From left to right, the first section represents inputs; its length is equal to the number of features of the problem, in this section there will be as many genes as observations a sample has. The next section, stores the number of hidden neurons using a binary coded integer number (the length of this section is calculated by $\lceil Log_2\,(2 \cdot (N/D\ log(N)))\rceil$). The third section uses only two genes to represent the training algorithm to be used; Table 4.1 shows the list of possible learning algorithms. The last two sections, contain binary coded number of the activation function for the hidden and the output layer respectively; each of these have a length of 4 genes. Table 4.2 shows the list of possible activation functions for neurons of the hidden and output layers. Appendix B has a plot for each of the functions.

Number of input neurons and feature selection

The goal of feature selection is to find the subset of features that produces the best recognition performance and requires the least computational effort.

Table 4.1: List of training algorithms

	Algorithm	Genes
0	Sequential back propagation	00
1	Batch back propagation	01
2	QuickPROP	10
3	iRPROP	11

Table 4.2: List of activation functions

	Name	Genes
0	Linear	0000
1	Sin	0001
2	Cos	0010
3	Sigmoid	0011
4	Sigmoid stepwise	0100
5	Sigmoid symmetric	0101
6	Sigmoid symmetric stepwise	0110
7	Gaussian	0111
8	Gaussian symmetric	1000
9	Not used	
10	Elliot	1010
11	Elliot symmetric	1011
12	Linear piece	1100
13	Linear piece symmetric	1101
14	Sin symmetric	1110
15	Cos symmetric	1111

Adding more features does not necessarily improve discrimination performance. An important goal is to choose the best set of features from the discriminating features that are available. GA have been used to automatically determine the relative importance of many different features and to select a good subset of features available to the system. We use GA to select a best feature subset, defined as a particular set of features that is the best in discriminating the target from the natural clutter.

GA is used to seek the smallest (or the least expensive) subset of features for which the classifier's performance does not deteriorate below a certain specified level. During the search, each subset can be coded as a D-element bit string (D is the total number of features). The ith element of the bit string assumes 0 if the ith feature is excluded from the subset and 1 if it is present in the subset.

The chromosome's first section represents the inputs, genes that contain a 1 indicates that the respective input variables will be taken into account. On the other hand, genes that contain a 0 indicates that likewise inputs will not be taken into account. This input representation allows GA to perform feature selection at the same time it finds the ANN's topology.

Number of hidden neurons

In [Huang and Babri, 1998] has been rigorously proved that for N arbitrary distinct samples, one ANN with at most N hidden neurons and with any bounded nonlinear activation function which has a limit at one infinity can learn this N distinct samples with zero error. Despite their proof, for our approach this is still a large upper limit for determining the number of hidden neurons. We are not tackling an optimization problem for classification function $f(\mathbf{x}) = y$, we do not intend to learn this function with zero error. But find an ANN that behave tantamount to $f(\mathbf{x}) = y$.

Several researchers have proposed some rules of thumb for determining an optimal number of hidden units for any application. For instance: *A rule of thumb is for the size of this hidden layer to be somewhere between the input layer size and the output layer size ...* [Blum, 1992]. *How large should the hidden layer be? One rule of thumb is that it should never be more than twice as large as the input layer...*, [Berry and Linoff, 1997], and *typically, we specify as many hidden nodes as dimensions needed to capture 70-90 % of the variance of the input data set...* [Boger and Guterman, 1997]. Most of those rules are not applicable to most circumstances as they do not consider the training set size (number of training pairs), or the complexity of the data set to be learnt [Xu and Che, 2008].

Based on Barron's work [Barron, 1994], Shuxiang Xu and Ling Chen propose a novel approach for determining an optimal number of hidden layer neurons for feed forward neural networks [Xu and Che, 2008]. In their work, after conducted a number of experiments they found that for a small or medium-sized dataset (with less than 5,000 training pairs), when N/n is less than or close to 30, the optimal number of hidden neurons most frequently occurs on the number of samples (N); however, when N/D is greater than 30, the optimal number of hidden neurons is close to the value of $\sqrt{\frac{N}{D\, log(N)}}$.

The former approach has been relaxed a little bit in order to allow GA to find the optimum number of hidden neurons; for the case where N/D is less than or close to 30, N is taken as the maximum number of possible hidden neurons;

otherwise is calculated with $2 \cdot \left(\frac{N}{D\ log(N)} \right)$. The above formula is a very relaxed upper limit, large enough that GA will find an adequate number of neurons in the hidden layer.

The former number is coded as part of the chromosome in a binary substring of length $\left\lceil\ Log_2 \left(2 \cdot \left(\frac{N}{D\ log(N)} \right) \right)\ \right\rceil$.

Number of output neurons

Apparently the number of output neurons is missing in chromosome representation, but in the proposed approach, the number of output neurons is not required to be coded in the chromosome. The output produced by the ANN in response to input pattern $\mathbf{x}_j$ lies in output neurons y_1, y_2, y_3, $\ldots$, y_k.

Since the codomain of activation functions used is $[0, 1]$ or $[-1, 1]$, it is expected for a k class type sample, only the y_k output neuron presents the maximum value (1 or near to 1) and the complement set of output neurons present the minimum value (0 or -1 if using symmetric function). This is, the ANN classificator *activates* the output neuron to the corresponding class of input vector $\mathbf{x}_j$.

Using the former scheme of activating only output neuron number k, it is clear to see that the number of output neurons depends on the problem addressed.

This is, in a k-class classification problem, we need a total of k output neurons to represent all possible classification decisions. This is a fixed characteristic, therefore, it is not evolved, and so it is not included in the chromosome.

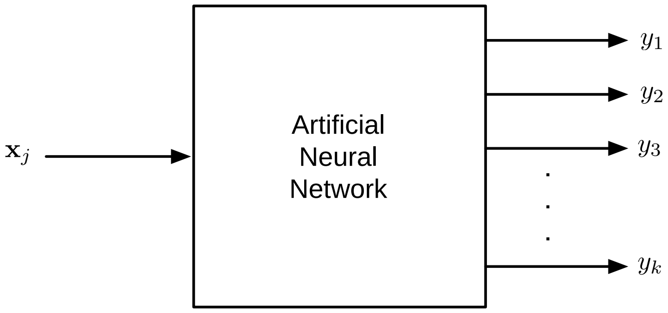


Figure 4.3: Number of output neurons is defined directly by the number of different classes existing in the problem

Figure 4.3 illustrates how the number of output neurons is directly determined by the number of different classes existing in the classification problem. In this figure vector $\mathbf{x}_j$ denotes the jth sample of an D-dimensional input vector to be classified by an ANN.

As an example of chromosome representation used, Figure 4.4 shows one chromosome for a problem with $N = 560$, $D = 5$ and $C = 2$. Chromosome is 21 genes length and encodes 3 input neurons from 5 possible, the first, the second and the fifth; 5 hidden neurons; 2 output neurons; hidden neurons have sigmoid symmetric as activation function, and output layer elliot symmetric; when training in order to calculate fitness, this individual should learn with batch version of back propagation algorithm.

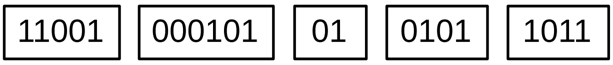


Figure 4.4: Example of chromosome used in this thesis.

When working with binary and same length chromosomes (as used in this thesis), there is no need to check functionality of crossover and mutation operations, because crossing two binary homologous chromosome generates one or more binary chromosome, and when mutating one or more genes will produce an equal length binary chromosome.

4.2.2 Fitness calculation

Fitness is a measure of how good one individual is. In this thesis the fitness value is limited to interval $[0, 1]$ and it is calculated according to the number of correct classifications made by each class. If one individual hits all the samples of one class and half the samples of second one, it will have 0.75 in his fitness function.

We decide to use a learning strategy for training the ANN, instead of evolving its weights. Learning algorithms are generally gradient-based, so one could argue that they would get stuck at local optima.

Given that there exists a set of unknown samples, we do not intend to learn the classification function $f(\mathbf{x}) = \mathbf{y}$ with zero error. But find an ANN that behave tantamount to $f(\mathbf{x}) = \mathbf{y}$. If optimal fitness were found, the generalization ability of the network could be compromised.

Algorithm 4.2 shows the fitness computing used in this thesis. It begins with chromosome decoding, this is done for interpreting phenotype starting from genotype; then, according to structure decoded an ANN is created; next, flags training terminators *time_limit* and *epochs_limit* are initialized; after that, ANN enters to the training loop until one flag reaches stop condition; last, the number of true positives classifications made by class are calculated; finally, the mean value of correct classifications made by class are returned as the fitness of chromosome given.

Algorithm 4.2 Fitness calculation.

```
Require: Chromosome, $time\_limit$, $epochs\_limit$, $\mathbf{TS}$ with $C$ number of classes
Decode chromosome
Create ANN according to chromosome
$timer \leftarrow 0$ seconds
$epochs \leftarrow 0$
while $timer < time\_limit$ and $epochs < epochs\_limit$ do
    Train one epoch ANN($\mathbf{TS}$)
    $epochs \leftarrow epochs + 1$
end while
for $i \leftarrow 1$ to $C$ do
    $class\_accuracy_i \leftarrow \frac{\text{samples correctly classified from class } i}{\text{samples existing in class } i}$
end for
$fitness \leftarrow \mathbf{mean}(class\_accuracy)$
return $fitness$
```

The training loop is limited by time or number of epochs because there is no other stop criteria that ensures reaching the stop condition.

There are two factors that one must take into consideration in order to decide which library to use to train the ANN. First comes the training time, that is, if the algorithm takes a lot of time training the network, then we will be forced to evaluate only a small number of individuals to provide a result in a reasonable time. Secondly, one must consider the flexibility of the algorithm, i.e., the parameters of the net that can be modified. The former restricts the number of parameters that can be evolved. For example, if the chosen algorithm does not accept to modify the number of neurons in the hidden layer, then one would not be able to modify this parameter in the evolutionary process. Here, we decided to use the Fast Artificial Neural Network Library (FANN) [Nissen, 2003], which is efficient and very flexible, with a vast number of parameters that can be modified.

4.2.2.1 Evolution result

Despite some researchers have proposed to use population information in evolutionary ANN for improving generalization in learning systems [Yao et al., 1998, Cho, 1999, Liu et al., 2001, Gabrys and Ruta, 2006], the proposed approach only takes into account the best individual in the final population. The former decision is supported on Kolmogorov's theorem (Section 1.1) and on Huang and Babri's proof [Huang and Babri, 1998], where it is shown that one hidden layer network with enough number of hidden neurons has the capability of learning any multivariable function with zero error.

For the cases where two or more individuals present equal fitnesses, GEANN follows the parsimony principle and prefers a smaller and simpler individual than a complex and larger one; that is, having the minimum possible number of neurons in the input and hidden layers, without detriment to their classifying capabilities.

GA will find an appropriate ANN topology, by performing a feature selection, i.e. it discerns what variables are important for the classification problem and what variables are not. The number of variables determines the number of input neurons in the final classifier. The evolutionary process determines the necessary number of hidden neurons and what training algorithm is the best for the designed topology. Also, GA find what activation function should be used in the hidden layer and the output layer.

Once evolution finishes, we will get information on how many input neurons are needed, which features should be taken into account and which should not, how many hidden neurons are needed, what training algorithm suits better for that topology and **TS**, what activation function should be used in the hidden layer, which activation function should be used in the output layer, and finally we will get one real value as fitness on **TS**.

4.3 K-Fold Cross Validation

In addition to the problem of designing the ANN's topology, there is the problem of knowing how much the ANN must learn. If the ANN learns too little, it will be ineffective to classifying, and if it learns too much it will lose its ability to generalize. There are three stop criteria well known in the application of a learning algorithms: stop the network training when a predefined number of iterations is reached, stop training when a predefined error rate for the training set is reached, and stop training when a minimum error rate is reached for a validation set, this last is known as cross-validation (CV) [Shao et al., 2011].

Since the goal of GEANN is to automate the design of ANN for a given classification problem, we cannot set a predefined number of iterations; this number could work fine for some problems and fail for others. Neither we can set one predefined learning error rate, because not all the problems have the same complexity. Cross-validation has been proved to be an effective stop criteria and an excellent way to avoid overfitting [Ng, 1997, Prechelt, 1998, Shao et al., 2011], so the proposed approach uses cross-validation.

The second phase deals with overfitting. Overfitting is the inability of the evolved ANN to generalize, therefore to perform well the classification task. Generally, this problem is tackled by limiting the learning capabilities of the ANN. To reduce learning, and therefore overfitting in training an ANN, one possible solution is to decrease the number of training epochs. Unfortunately, the potential solution of decreasing the number of epochs raises another question which is how to choose a reasonable value for this parameter. To address this issue, this second phase applies a 5-fold cross-validation technique.

Algorithm 4.3 shows the K-fold cross validation process followed in GEANN. First the original training set **TS** is randomly split into five subsets (folds) of equal size. Then $k-1$ folds are used to form one temporary training set and the remaining set is used as a temporary validation set. This TS and VS formation process is repeated k times, each time leaving out a different fold of the **TS**. Next, we train k networks, each of them trained for 1 to max_epoch, storing accuracy for every iteration. Finally, accuracy average of k nets is computed in order to find the value that provides the best generalization.

Cross-validation's aims to find how many training iterations must be applied to the ANN in order to avoid overfitting, this number of iterations is located at the position where the best average was found.

4.4 Final Training

The third and last phase is the culmination of GA and cross validation results. GA provides ANN's characteristics and cross-validation determines how many iterations of the learning algorithm are necessary; now it is the time to form our

Algorithm 4.3 K-fold cross-validation algorithm.

```
Require: $k, max\_epoch$, TS, ANN's topology
Split TS into $k$ equal sized subsets (folds)
for $i \leftarrow 1$ to $k$ do
    $VS_i \leftarrow fold_i$
    $TS_i \leftarrow \mathbf{TS} \setminus fold_i$
    Create ANN$_i$ according to ANN's topology received
end for
for $j \leftarrow 1$ to $max\_epoch$ do
    for $i \leftarrow 1$ to $k$ do
        Train one epoch ANN$_i(TS_i)$
        $accuracy_i \leftarrow$ Compute accuracy for ANN$_i(VS_i)$
    end for
    $mean\_accuracy_j \leftarrow \mathbf{mean}(accuracy)$
end for
$epochs \leftarrow \mathbf{arg\_max}_x(mean\_accuracy_x)$
return $epochs$
```

final classifier according to the evolved topology and training the network for the number of iterations calculated by cross-validation.

The result of the third stage is the final ANN trained with good enough generalization of **TS**, and ready to classify.

4.5 Concluding Remarks

This chapter presents GEANN, the proposed approach. This evolutionary approach consists of three phases (after the scaling that takes place in the preprocessing).

As a first step, one biologically inspired heuristic was consider for finding ANN's topology: Genetic Algorithm. One chromosome structure was designed to represent one hidden layer ANNs of different sizes and characteristics. One of the main activities to take into account when working with GA is the fitness calculation, here we explained how GEANN computes fitness. GA will determine the ANN's characteristics, but still is unanswered, how many epochs must be used in the training algorithm.

As a second step, one statistical technique was used for avoiding lose of generalization: Cross-validation. ANN needs to learn just enough, without overdoing it. CV will provide us with information on the number of epochs that must be applied to the learning algorithm to avoid lose of generalization.

The final step is the culmination of the last two phases. Here, it only remains to create the ANN found by the GA and train it for the number of iterations suggested by CV.

Chapter 5

Results and Discussion

In order to test the accuracy of the automatically designed ANN, eight classification experiments were performed. The first one corresponds to an artificial dataset, the second one to a real life land cover classification problem and the last six were taken from the UCI Machine Learning Repository [A. Asuncion, 2007], covering different issues such as identification of cancerous tissue, differentiation of flower types, diagnosis of normal or abnormal human hearts, location site of yeast protein, and identification of wines.

For each addressed problem, 30 independent runs were performed. Table 5.1 shows the parameters used in GA for each of the runs. On every evolution process were evaluated 1,500 individuals. It was applied 70% of crossover probability, 70% of mutation probability, and 20% of gene mutation probability. On fitness computation, were used 60 seconds or 10,000 epochs as stop criteria.

Table 5.1: Parameters used in evolution.

Parameter	**Value**
Number of evaluations	1,500
Population	50
Generations	30
Pc	0.7
Pm	0.7
Pm_{gene}	0.2
Fitness time limit	60 seconds
Fitness #epochs limit	10,000 epochs

Next section, presents a detailed description of each of the datasets; finally the results and discussion section shows the average accuracy achieved.

5.1 Data Sets Description

5.1.1 Two moons: Artificial dataset

The two moons (or two bananas) dataset contains two regions representing two classes each region is a half ring with radius $r = 10$, width $w = 8$, one region is upper half and the other is lower half, both regions have a separation distance $d = -31.5$ between them, negative value means overlapping, and each region has 350 two dimensional samples.

Program 5.1: Two moons function

```
TWOmoons[n_: 350, d_: -31.5, r_: 10, w_: 8] :=
  Block[{moon1 = {}, moon2, a, b, c},
    While[Length[moon1] < n,
      pto={a=(2r+w) (Random[]-0.5), b=(r+w/2) Random[], c=Sqrt[a^2+b^2]};
      If[  (r - w/2)  < c < (r + w/2),
        moon1 = Join[moon1, {Part[pto, 1;;2] + (r+w/2)}]
      ]
    ];
    moon2=Partition[Riffle[Part[moon1, All, 1]+r, -Part[moon1, All,
    2]-d], 2];
    moon1=Partition[Flatten[Riffle[moon1, Table[-1, {Length[moon1]}]]], 3];
    moon2=Partition[Flatten[Riffle[moon2, Table[1, {Length[moon2]}]]], 3];
    Join[moon1, moon2]
  ]
```

Listing 5.1 shows the TWOmoons function used to generate 700 points. The function is coded in Mathematica©. This function generates one region centered at $(r + w/2, r + w/2)$; the second region is a mirror image of the first one with a translation of (r, d) respect to the first region's center.

Figure 5.1 shows the two moons dataset generated with the former function. Samples of class one are circles, and samples of class two are squares. This set provides a way to measure an ANN's classification skills. The dataset created was split into two subsets **TS** and **VS**, the first one includes 80% randomly chosen points and the second one the remaining 20%. Figure 5.1 represents **TS** points with lined shapes and **VS** points with filled shapes.

The proposed process, was followed only with the **TS**, the evolved ANN learned from the **TS** only. **VS** was used only for testing purposes once the design of the

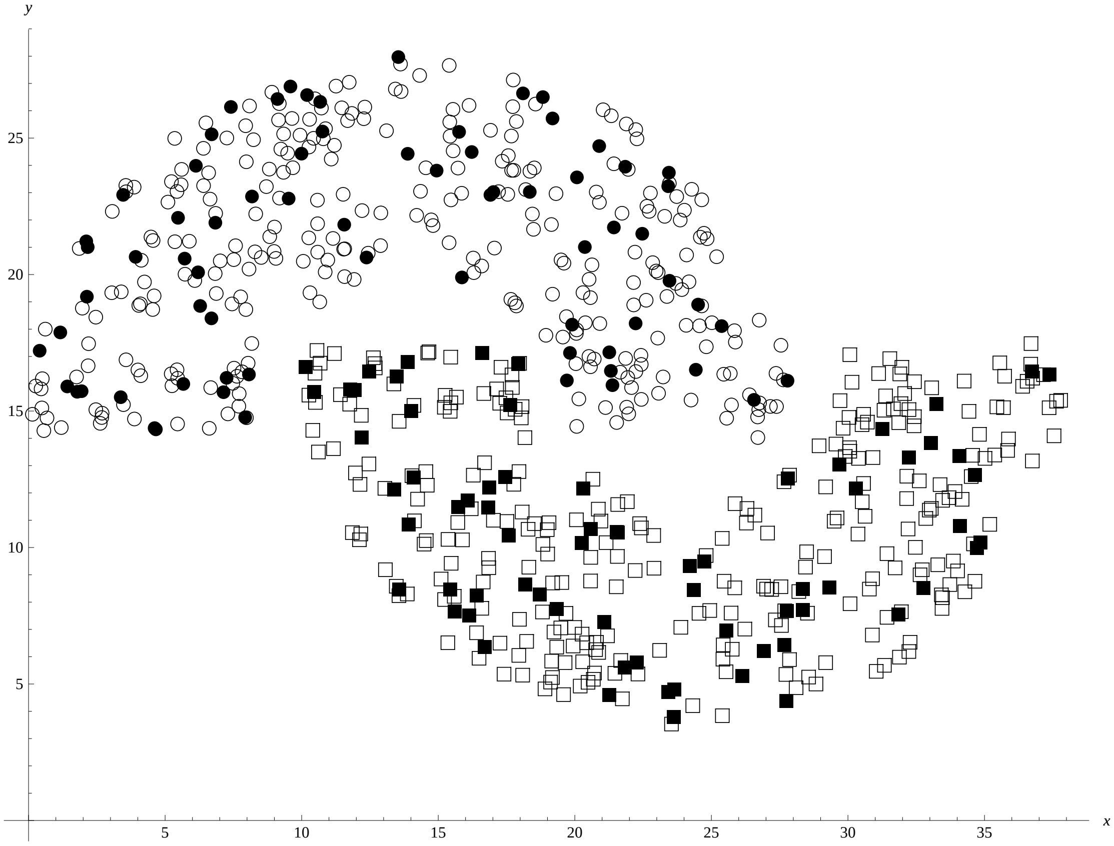


Figure 5.1: Two moons dataset.

ANN's completed. Figure 5.1 shows 560 points of **TS**, circles from class one and squares from class two, also illustrates 140 points from **VS** used for testing after evolution concludes.

5.1.2 Land cover classification: Satellite image from Bolivia

The study area captured in this image is located in the department of Beni, Bolivia; it was selected due to its highly heterogeneous landscapes comprising montane tropical cloud forest (locally know as yungas), lowland tropical forest, and wet savannas.

The classification problem was based on Landsat ETM+ imagery, based upon 8 categories: early-growth forests, old-growth forests, water, bare soil, agriculture, pastures, infrastructure/urban and, savanna areas. Training data selection was based on the data gathered previously in the field after a careful examination of its spectral signatures.

For the mixed version of this dataset, all samples from training set and from validation set were gathered into one single dataset, then this unique dataset were split it again in 80% random samples for the training set and the remaining 20% for the validation set. In the following section a discussion will take effect about this.

Figure 5.2 shows the satellite image concerning to this dataset.

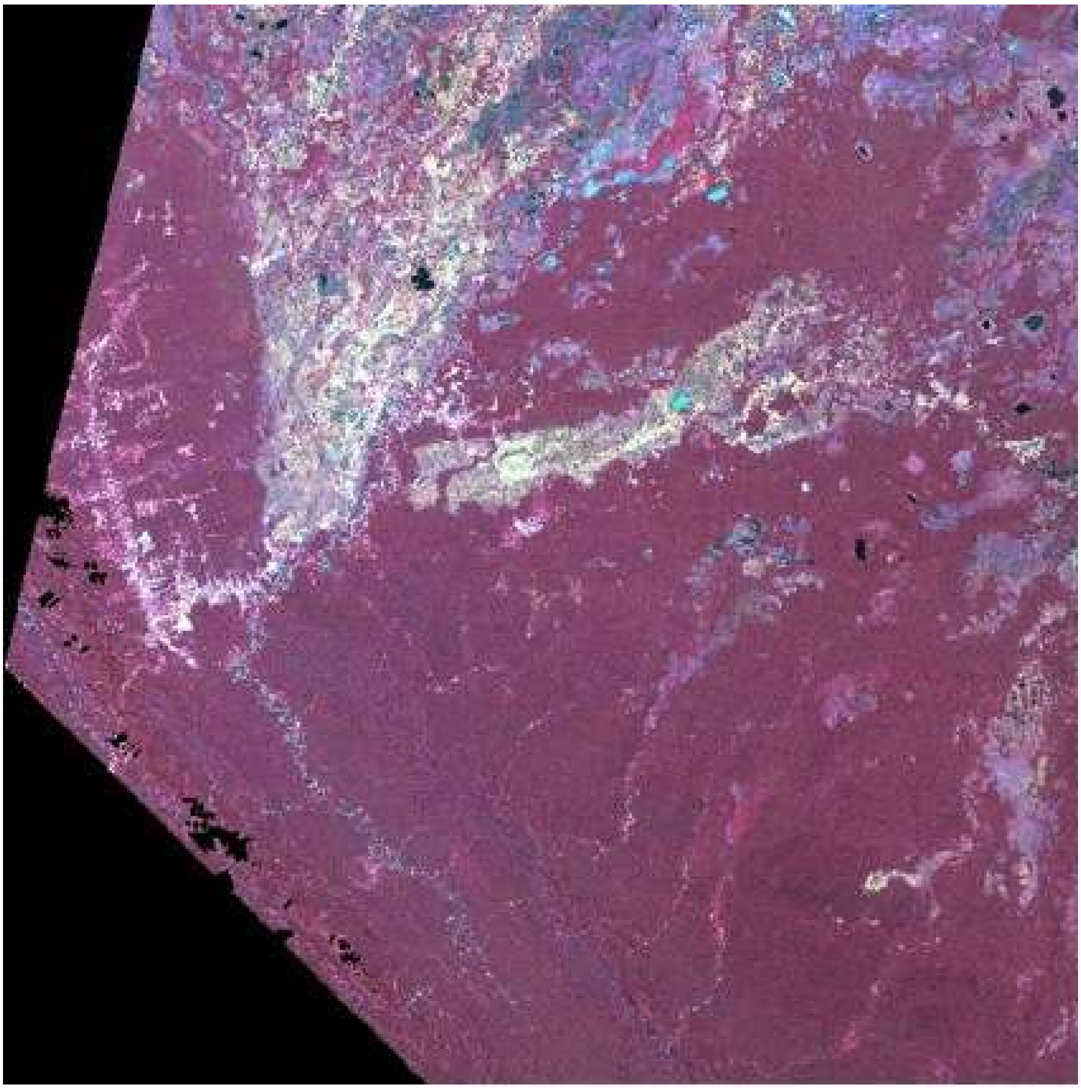


Figure 5.2: Landsat EMT+ imagen of Beni, Bolivia.

5.1.3 Iris: A classic numeric dataset

The iris dataset by R.A. Fisher is arguably the most famous dataset used for classification. It contains 50 samples each of three types of plants: Iris Setosa, Iris

Versicolor, and Iris Virginica. One class is linearly separable from the other 2; the last class is not linearly separable from the other two. There are four attributes in the dataset: sepal length, sepal width, petal length, and petal width (all measured in centimeters).

5.1.4 Yeast: Protein location sites

This database contains information about a set of yeast cells. The task is to determine the location site of each cell. The classes are: 1) CYT, cytosolic or cytoskeletal; 2) NUC, nuclear; 3) MIT, mitochondrial; 4) ME3, membrane protein, no N-terminal signal; 5) ME2, membrane protein, uncleaved signal; 6) ME1, membrane protein, cleaved signal; 7) EXC, extracellular; 8) VAC, vacuolar; 9) POX, peroxisomal; and 10) ERL, endoplasmic reticulum lumen.

5.1.5 WDBC: Breast Cancer Wisconsin (Diagnosis)

This database contains features computed from a digitized image of a fine needle aspirate of a breast mass. To obtain observations investigators follow this process: Once organic material is gotten, it is mounted on a microscope slide and stained to highlight the cellular nuclei. A portion of the slide in which the cells are well-differentiated is then scanned using a digital camera and a frame-grabber board. Then individual nuclei are isolated.

Once all (or most) of the nuclei have been isolated in this way, a specialized software computes the values for each of ten characteristics of each nuclei, measuring size, shape and texture. The mean, standard error and extreme values of these features are computed, resulting in a total of 30 nuclear features for each sample.

Figure 5.3 shows an image of isolated cells, this takes between two and five minutes per slide. The task in this problem is to determine whether a found tumor is benign or malignant.

5.1.6 SPECTF: Heart dataset

This dataset contains information for diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. Each of the patients is classified into

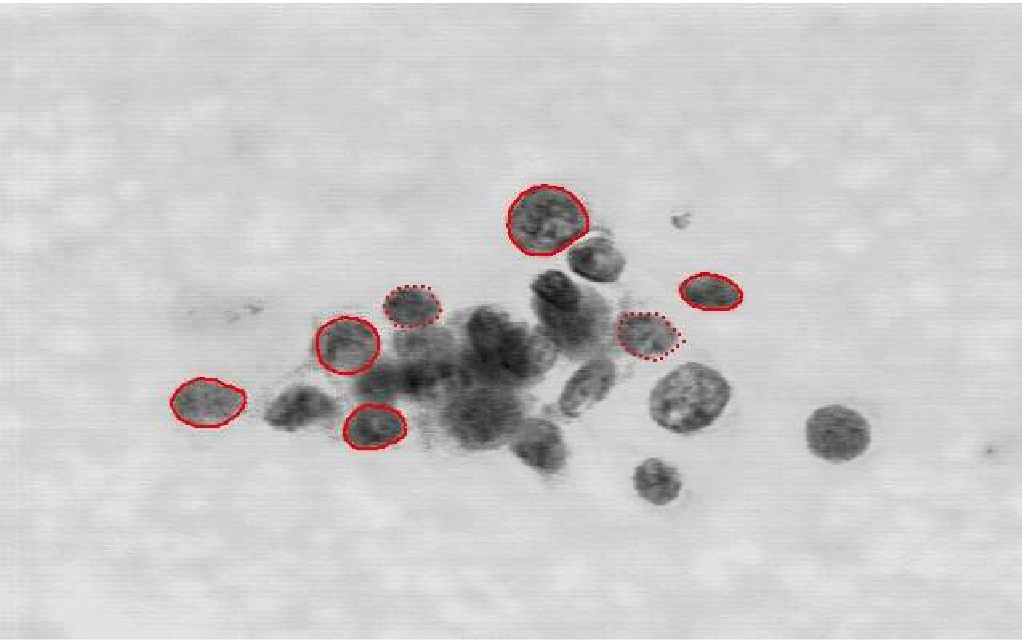


Figure 5.3: Isolating cells from an image taken from breast mass.

two categories: normal and abnormal. The database of 267 SPECT image sets (patients) was processed to extract features that summarize the original SPECT images. As a result, 44 continuous feature patterns were created for each patient. The task in this problem is to determine whether a patient has an abnormal heart condition or not.

5.1.7 SPECT: Heart dataset

This dataset is similar to the previous one, but in this one, the patterns were further processed to obtain 22 binary feature patterns.

5.1.8 Wine: Origin of wines

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different crops. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The attributes are: 1) Alcohol, 2) Malic acid, 3) Ash, 4) Alcalinity of ash, 5) Magnesium, 6) Total phenols, 7) Flavanoids, 8) Nonflavanoid phenols, 9) Proanthocyanins, 10)Color intensity, 11)Hue, 12)OD280/OD315 of diluted wines, and 13)Proline.

Table 5.2 summarizes the datasets used in experiments. From left to right, the first column presents the dataset name of each problem. Second column, shows the number of features. Next two columns are the number of training samples existing in the training and validation set respectively. Finally, last column shows the number of different classes the problem has.

Table 5.2: Datasets used in experiments.

Dataset name	D	**TS** size	**VS** size	*Classes*
Two moons	2	560	140	2
Land cover	12	36,914	800	8
Land cover (mixed)	12	30,172	7,545	8
Iris	4	120	30	3
Yeast	8	1,187	297	10
WDBC	30	455	114	2
SPECTF	44	80	187	2
SPECT	22	80	187	2
Wine	13	143	35	3

5.2 Results and Discussion

The best evolved ANN in each run was optimized using the 5-fold cross-validation to determine the maximum number of epochs. Finally, we trained again each network with the training set using the number of epochs found in the previous step.

Table 5.3 presents the average ratio of correct classification in the cross validation (μCV), training set (μ**TS**), and validation set (μ**VS**).

In order to illustrate the accuracy of our approach, we are going to contrast our results against different techniques that have been used to tackle the different problems presented on Table 5.3.

Two moons

The two moons problem was designed to easy visualize the dataset, due to high dimensionality in classification problems, most of the times it is hard to see how hard one classification problem is. Two moons is a linear separable problem, of low dimension with only two different classes.

Table 5.3: Experimental results.

Dataset name	μCV	μ**TS**	μ**VS**
Two moons	0.9938	0.9984	0.9985
Land cover (Bolivia)	0.8472	0.9210	0.8070
Land cover (Bolivia) mixed	0.9017	0.9401	0.9166
Iris	0.9865	0.9839	0.9865
Yeast	0.5916	0.6599	0.5886
WDBC	0.9367	0.9603	0.9565
SPECTF	0.8191	0.8169	0.7849
SPECT	0.8404	0.8810	0.8374
Wine	0.9447	0.9839	0.9509

Iris

The Iris problem presents an easy to differentiate clouds of points, one class is linearly separable from the other two, but the remaining two are not linearly separable between them. Similarly to the two moons, the iris problem presents low dimensionality and a small number of classes. For those two problems, our approach performed very well, getting almost 100% of correct classifications.

Land cover

If we compare the accuracy reached in this dataset against the accuracy achieved from some other well known approaches used to classify Landsat ETM+ imagery, we realize that land cover classification problem, presents not so good results.

Spectral mixture analysis is one of the traditional techniques used to classify this kind of imagery, in [Song, 2005, Buyantuyev et al., 2007] we can see an overall classification accuracy of 90%, ten percent higher than the accuracy reached by us.

The process of determining a class type for each pixel in a land cover classification problem begins when one technician gathers data *in situ* for some of the training areas. Then, on a desk several pixels are matched to their corresponding class types by its spectral signatures, but most of the times this process does nor accurately reflect reality. The accuracy presented on the mixed version of the land cover dataset, can lead us to conclude that many pixels of the original training set are labeled wrong. That is why, when we mix all samples and reallocate training set and validation set pixels, the proposed approach was able to reach a competitive classification accuracy.

Yeast

Yeast protein location problem is the most difficult addressed in this thesis, it presents a medium sized dimensionality, considerable amount of samples, very different classes and very much overlapping cloud classes. Horton and Nakai defined a model of classification which combines human provided expert knowledge with probabilistic reasoning [Horton and Nakai, 1996], this approach can be viewed either as a probabilistic analog to decision trees or as a restricted form of Bayesian network. Horton and Nakai were able to get 0.55 of correct localisation proteins for the yeast dataset, this result is 0.0386 lower than the localisations got by our approach, and with the drawback than a human expert should hand-tune some factors for optimal prediction accuracy.

WDBC

In [Wolberg and Mangasarian, 1990] Olvi L. Mangasarian *et al.*, took WDBC dataset to test a linear programming-based diagnostic system by a variant of the multi-surface method (MMS) [Mangasarian et al., 1995], they got 0.975 of correct classification. The accuracy reached by GEANN is 0.0185 lower than the one gotten by Mangasarian *et al.* Our solution is fully automated, unlike the one presented by Mangasarian.

SPECTF

Lukasz A. Kurgan, *et al.*, developed a six-step knowledge discovery approach to automated cardiac SPECT diagnosis [Kurgan et al., 2001]. In his work, they got 0.77 of correct classifications on the unseen data, as a disadvantage to this method, we can say that, it only works for medical diagnosis. Cuong To and Tuan D. Pham developed a procedure to create decision trees by means of GP [To and Pham, 2009]. For the SPECTF problem they got 0.7907 of accuracy, just 0.0058 of difference against our approach. In this work, several methods were also compared and the results are: Support Vector Machines (SVM) 0.6337, LogitBoost (LB) 0.6860, Logistic Regression (LR) 0.6860, Linear discriminant analysis (LDA) 0.6686, and Linnear regression and least square (LS) 0.6686.

SPECT

Michael G. Madden developed Markov Blanket Bayesian Classifier Algorithm (MBBC) and compare it against other Bayesian approaches in [Madden, 2002]. He showed that for SPECT problem, Naive Bayes (NB) classification got 0.7170 of accuracy, Tree-Augmented NB 0.8125, General Bayesian Network 0.8019 and MBBC 0.8075. These results are slightly lower than the one obtained by our approach. Cios and Kurgan were able to get different results in [Cios and Kurgan, 2002]. They use inductive ML methods for generating hypotheses about a given **TS**. In particular, they use CLIP (Cover Learning using Integer Linear Programming) algorithms, which is a hybrid algorithm that combines rule and decision tree algorithms. They got 0.84 of correctly classified samples from **VS**, this is only 0.0026 better than the average obtained by our approach for the same dataset.

Wine

Since k nearest neighbor (kNN) classifiers are sensitive to outliers and noise contained in the training set, many approaches have been proposed to edit the training data so that the performance of the classifiers can be improved. In [Jiang and hua Zhou, 2004] Yuan Jiang and Zhi-hua Zhou proposes to use a neural network ensemble to edit the training data set for kNN classifiers. They compared 3 different editing approaches against his NNEE (Neural Network Ensemble Editing) and for the wine dataset, they got 0.9494, 0.9494, 0.9605 and 0.9605 of accuracy for the methods Depuration, RelabelOnly, RemoveOnly and MMEE respectively. These results present an evidence of similar classification skills for the one hidden layer ANN evolved under our proposed approach and kNN with training set edited by neural network ensamble. Our results are only 0.0096 lower than the one got it by Yuan Jiang and Zhi-hua Zho for the wine dataset, and also slightly above than the other editing strategies.

5.2.1 Concluding Remarks

It is important to know that the GEANN process we have just presented is competitive with other procedures. Nonetheless, the approach presented is not the clear and unbeatable winner, it produces a competitive ANN, since the aim of this work is to develop an automated procedure to design competitive ANN, we believe that the comparison presented here is enough for this purpose.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis proposes GEANN a new technique to automatically obtain a competitive classification artificial neural network. The proposed approach attempts to solve supervised learning problems, so a tagged training set is needed.

GEANN uses three steps to produce a trained ANN. It first starts with a preprocessing scaling of observations, scaling translates the samples' domain interval from $(-\infty, \infty)$ to $[-1, 1]$. Scaling moves all multidimensional points inside the input space at the same time, so, the distribution of observations in the input space is unchanged.

By reviewing literature, we started from the reliable idea that one hidden layer feedforward neural network is enough to tackle any tagged training set. Also, we can notice that several learning algorithms have been used to improve generalization, but there is no one universal solution that automate learning for any classification problem and at the same time avoids overfitting. For addressing the former inconvenient, the approach proposed mixes biological inspired algorithms like GA (non deterministic), with statistical treatment of given observations in order to limit the amount of knowledge learnt.

The first step is crucial for the final classificator, is where most of decisions will take place, here, the mayor responsibilities such as: find relevant inputs to classify, determinate how many hidden neurons, decide which training algorithm to use, are delegated to a genetic algorithm. GA find a simpler and smaller architecture that will be able to learn better the training samples.

In the second step, proposed approach relies on cross-validation scheme to determine the number of iterations to apply learning algorithm, in order to avoid overfitting. The last step is to train the network using the number of epochs found in the previous step.

Eight problems were tackled in order to measure the accuracy of the proposed method, evolved ANN were compared against a variety of classifications algorithms and classification techniques. We compared experimental evolved ANN against previously presented methods. The experimental results show that our approach produces competitive classifiers. Furthermore, our methodology is completely automated, so it can be used by any researcher with no experience in the field of artificial neural networks.

6.2 Future Work

A deeper study on which parameters affect learning is needed. Identifying what parameters have high importance in learning, will allow us to derive GEANN for taking special treatment of such parameters.

Random initialization of weights as early activity in learning algorithms, causes noise. Given that a network starts to learn at every stage of the proposed approach. Future work could be done avoiding random initialization every time a network begins to learn.

One of the drawbacks of the presented approach is the limit time that every network has available to learn data, some studies have shown that selecting a smaller high representative subset of training pairs by class, allow network to learn in less time. This idea opens the possibility of cross validate while evolving.

The proposed approach evolves just a subset of possibles network arguments, deep studies could be done evolving one bigger subset of such arguments.

Artificial neural networks are a wide study area still open, the proposed approach could be explored in order to change network's main characteristics, such as, topology, recent investigations have demonstrated recurrent networks are able to get excellent accuracy in function approximation problems. Also changing neuron model (and in consequence learning algorithm) some researchers have obtained great accuracies in shorter time than traditional models.

Most of the neural networks implementations consider just one bias by layer, new studies shall be done by implementing one bias by neuron.

Appendix A

Implementation Issues

Since FANN is a neural network library, that implements multilayer artificial neural networks in C language, and Evolvica is a GA library implemented in Mathematica$^{\copyright}$. In order to implement GEANN approach, we need a way to run FANN C coded programs inside Mathematica's notebooks.

A.1 MathLink

MathLink is the proprietary technology of Mathematica for including executable programs coded in C within Mathematica's notebooks. A clean install of Mathematica is enough for been able to use MathLink, but before using it, one need to create few symbolic links to the proper MathLink installed programs and libraries. In particular, one need to reference to *mprep* (mathlink preprocessing) program, *mathlink.h* header file, and *libMLi3.a* static MathLink library file.

Program A.1 creates the above mentioned symbolic links in the respectives directories ∼/bin/mprep, ∼/include/mathlink.h, and ∼/lib/libMLi3.a. Also environment variables $PATH$, $CPATH$, and $LIBRARY\_PATH$ are up to date, so recent created links are loaded in a transparent way.

Program A.1: Shell script for installing MathLink

```
#!/bin/sh

dirbin=$HOME/bin
dirinclude=$HOME/include
dirlib=$HOME/lib

# Function that makes directories:
#    ~/bin   ~/include   ~/lib
# if doesn't exists, otherwise do nothing
create_dirs()
{   if [ ! -d $dirbin ]; then
        mkdir $dirbin
    fi
    if [ ! -d $dirinclude ]; then
        mkdir $dirinclude
    fi
    if [ ! -d $dirlib ]; then
        mkdir $dirlib
    fi
}

# Function that makes symbolic links to corresponding Mathematica files:
#    ~/bin/mprep
#    ~/include/mathlink.h
#    ~/lib/libMLi3.a
# if doesn't exist, otherwise do nothing
create_links()
{   if [ ! -e $dirbin/mprep ]; then
        ln -s ‘find /Applications/Mathematica* -name "mprep"‘ $dirbin/mprep
    fi
   if [ ! -e $dirinclude/mathlink.h ]; then
        ln -s ‘find /Applications/Mathematica* -name "mathlink.h" |
        grep -v framework‘ $dirinclude/mathlink.h
    fi
    if [ ! -e $dirlib/libMLi3.a ]; then
        ln -s ‘find /Applications/Mathematica* -name "libMLi3.a" |
        grep -v MacOSX‘ $dirlib/libMLi3.a
    fi
}

# Function for modifying environment variables:
#    PATH=$PATH:~/bin
#    CPATH=$CPATH:~/include
#    LIBRARY_PATH=$LIBRARY_PATH:~/lib
mod_PATHS()
{   echo $PATH | grep -q -s $dirbin
    if [ $? -eq 1 ] ; then
        echo "\n␣run␣on␣terminal:\n\texport␣PATH=$PATH:$dirbin\n"
        # adding ~/bin to PATH, ~/.profile will be modified
        echo "" >> ~/.profile
        echo "#␣insert␣~/bin␣to␣PATH" >> ~/.profile
        echo "echo␣\$PATH␣|␣grep␣-q␣-s␣\"~/bin\"" >> ~/.profile
        echo "if␣[␣\$?␣-eq␣1␣]␣;␣then" >> ~/.profile
        echo "␣␣␣␣export␣PATH=\$PATH:~/bin" >> ~/.profile
        echo "fi" >> ~/.profile
    fi

    echo $CPATH | grep -q -s $dirinclude
    if [ $? -eq 1 ] ; then
        echo "" >> ~/.profile
        echo "#␣insert␣~/include␣to␣CPATH" >> ~/.profile
        echo "echo␣\$CPATH␣|␣grep␣-q␣-s␣\"~/include\"" >> ~/.profile
        echo "if␣[␣\$?␣-eq␣1␣]␣;␣then" >> ~/.profile
        echo "␣␣␣␣export␣CPATH=\$CPATH:~/include" >> ~/.profile
        echo "fi" >> ~/.profile
    fi
    echo $LIBRARY_PATH | grep -q -s $dirlib
    if [ $? -eq 1 ] ; then
        echo "" >> ~/.profile
        echo "#␣insert␣~/lib␣to␣LIBRARY_PATH" >> ~/.profile
        echo "echo␣\$LIBRARY_PATH␣|␣grep␣-q␣-s␣\"~/lib\"" >> ~/.profile
        echo "if␣[␣\$?␣-eq␣1␣]␣;␣then" >> ~/.profile
        echo "␣␣␣␣export␣LIBRARY_PATH=\$LIBRARY_PATH:~/lib" >> ~/.profile
        echo "fi" >> ~/.profile
    fi
}
#------------------------------------------------------------

create_dirs
create_links
mod_PATHS
```

Once, MathLink is installed it is necessary to meet four requirements in order to run C functions within a Mathematica's notebook.

1) C source file must include *mathlink.h* header file, and program's main function must call MLMain(argc, argv) function. Program A.2 shows the basic template of a C source file program, where function f(int x, int y) is the one planned to run from Mathematica's notebook.

Program A.2: f.c, example of MathLink C program for running functions within Mathematica's notebooks

```
#include "mathlink.h"

int main(int argc, char *argv[]) {
   return MLMain(argc, argv);
}

int f(int x, int y) {
   return x+y;
}
```

2) One must specify in a text file (.tm) a template with the description of the C function that will be executed from the notebook. Listing A.3 shows the template file for the function f(int x, int y).

Program A.3: f.tm, text template file for describing function's parameters

```
:Begin:
:Function:     f
:Pattern:      f [x_Integer, y_Integer]
:Arguments:    {x, y}
:ArgumentTypes: {Integer, Integer}
:ReturnType:      Integer
:End:
```

where; :Begin:, is the beginning mark of the template for a particular function; :Function: is the name of the function in the C program; :Pattern: is the pattern to be defined to call the function; :Arguments: is the list of arguments to the function; :ArgumentTypes: is the list of types of the arguments to the function; :ReturnType: is the type of the value returned by the function; and :End: is the terminal mark in the template for a particular function.

3) Compile C source file and the respective template files in order to generate the running file that will be linked from Mathematica's notebook.

Program A.4 is the shell script implemented for compiling C source files, it links libraries from both FANN and MathLink.

Program A.4: mmcc, my mathlink c compiler, is a shell script for compiling c source programs and use them within Mathematica's notebooks

```
#! /bin/sh

#--------------------------------------------------------------------------
# List of libraries to include in compilation
libs="-lstdc++␣-ldoublefann␣-lMLi3␣-framework␣Foundation"

#--------------------------------------------------------------------------
# Parse the command line options
argc=$#
while [ $argc -ne 0 ]
do
    case "$1" in
        -o)    if [ -z "$2" ] ; then
                   echo "usage:␣$0:␣-o␣output_file"
                   exit 1
               fi
               output_filename=$2
               argc=‘expr $argc - 1‘
               shift
               ;;
        *.tm)
               files_tm="${files_tm}␣$1"
               ;;
        *.c)
               files_c="${files_c}␣$1"
               ;;
        *)
               compile_flags="${compile_flags}␣$1"
               ;;
        esac

        argc=‘expr $argc - 1‘
        shift
done

#--------------------------------------------------------------------------
# Convert the .tm files to one single .c file using mprep
```

```
if [ ! -z "${files_tm}" ] ; then
        file_tm_c=${output_filename}.tm.c
        mprep ${files_tm} -o ${file_tm_c}
        files_c="${files_c}␣${file_tm_c}"
fi

#-----------------------------------------------------------------------------
# Convert the .c files to .o files using gcc -c
for i in $files_c; do
    namefile=‘echo ${i} | sed -e ’s/\(.*\)\..*/\1/’‘
    gcc -c $i -o ${namefile}.o
    files_o="${files_o}␣${namefile}.o"
done

#-----------------------------------------------------------------------------
# Convert the .o files to exec file using gcc
for i in $files_o; do
    gcc -Wall ${libs} ${files_o} -o ${output_filename}
done
#-----------------------------------------------------------------------------
# Delete temporals files
rm *.tm.c
rm *.o
```

The shell script mmcc general usage is:

$ mmcc f.tm f.c -o f

4) Finally, is mandatory to “install” running program from a notebook by means of function Install[ ], once program is installed, one can execute function f[ ] from notebook. At the end of the notebook It is recomendable to uninstall all previously installed programs.

Program A.5

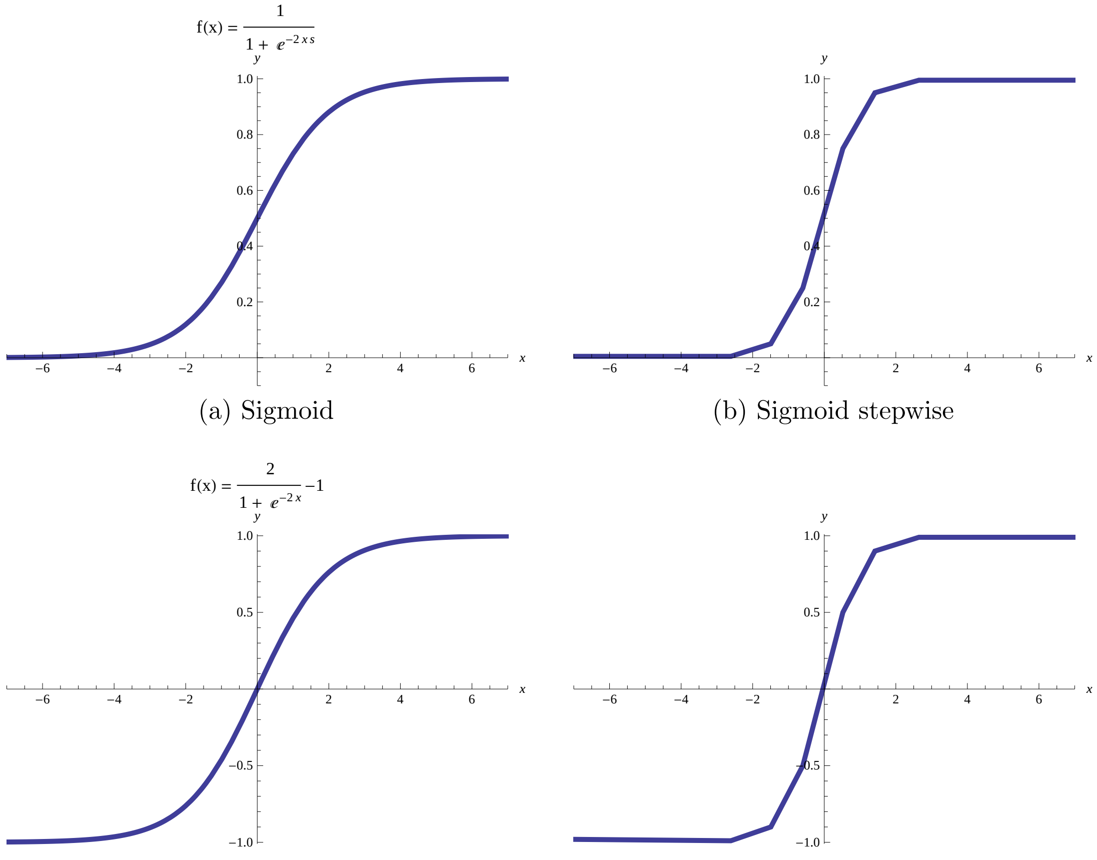
Program A.5: f.nb notebook implementing external functions calls.

```
link = Install[‘‘f’’];
(*Install starts a MathLink-compatible external program and installs
Mathematica definitions to call functions in it *)

f[2343, 4345]
6688

Uninstall[link];
(* Uninstall terminates an external program started by Install, and
removes Mathematica definitions set up by it *)
```

Appendix B

Plots of Activation Functions

This Appendix shows the plots of the 15 activation functions available for training in FANN.



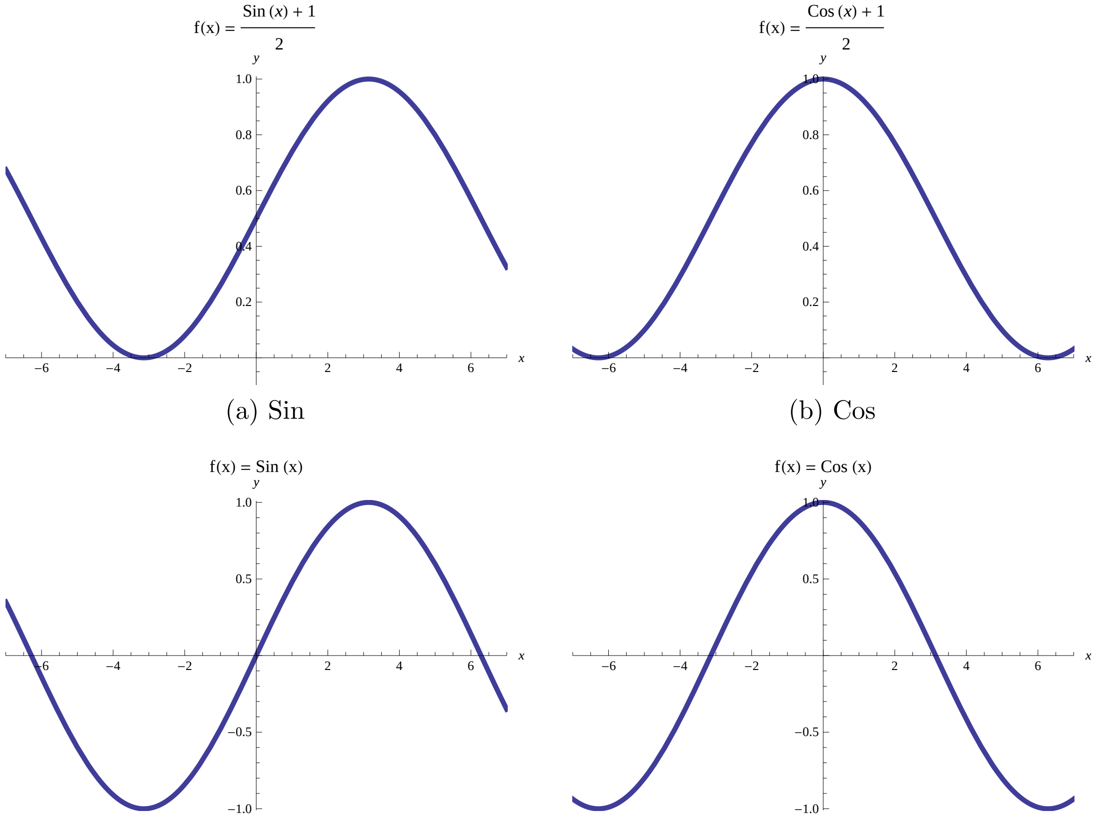
(a) Sigmoid

(b) Sigmoid stepwise

(c) Sigmoid symmetric

(d) Sigmoid symmetric stepwise

Figure B.1: Plots of sigmoidal functions used for training in GEANN.



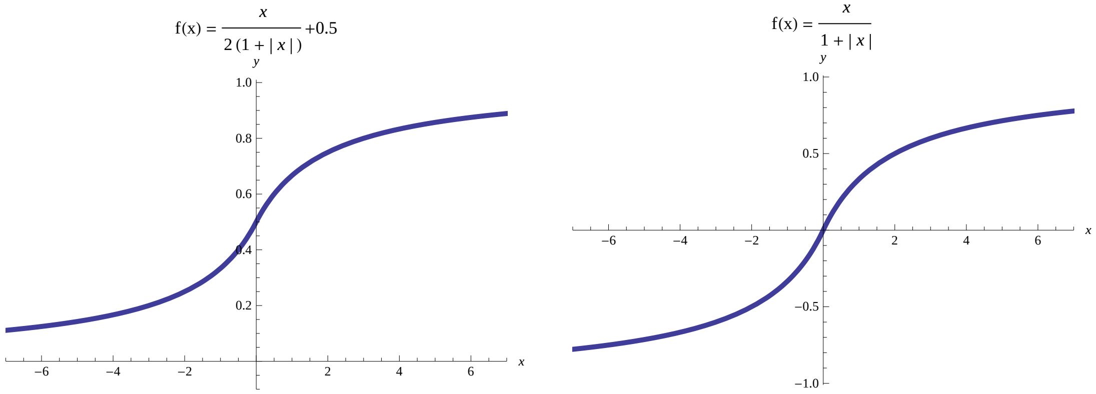
(a) Sin

(b) Cos

(c) Sin symmetric

(d) Cos symmetric

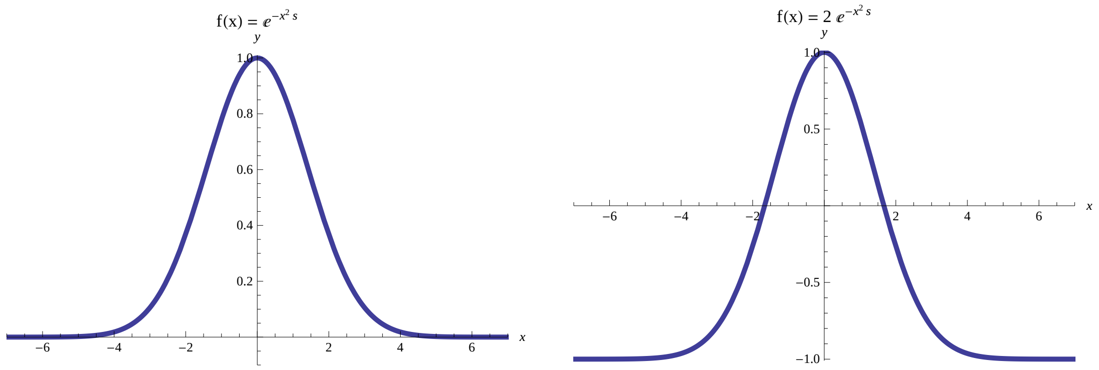
Figure B.2: Plots of sin and cosine functions used for training in GEANN.



(a) Elliot

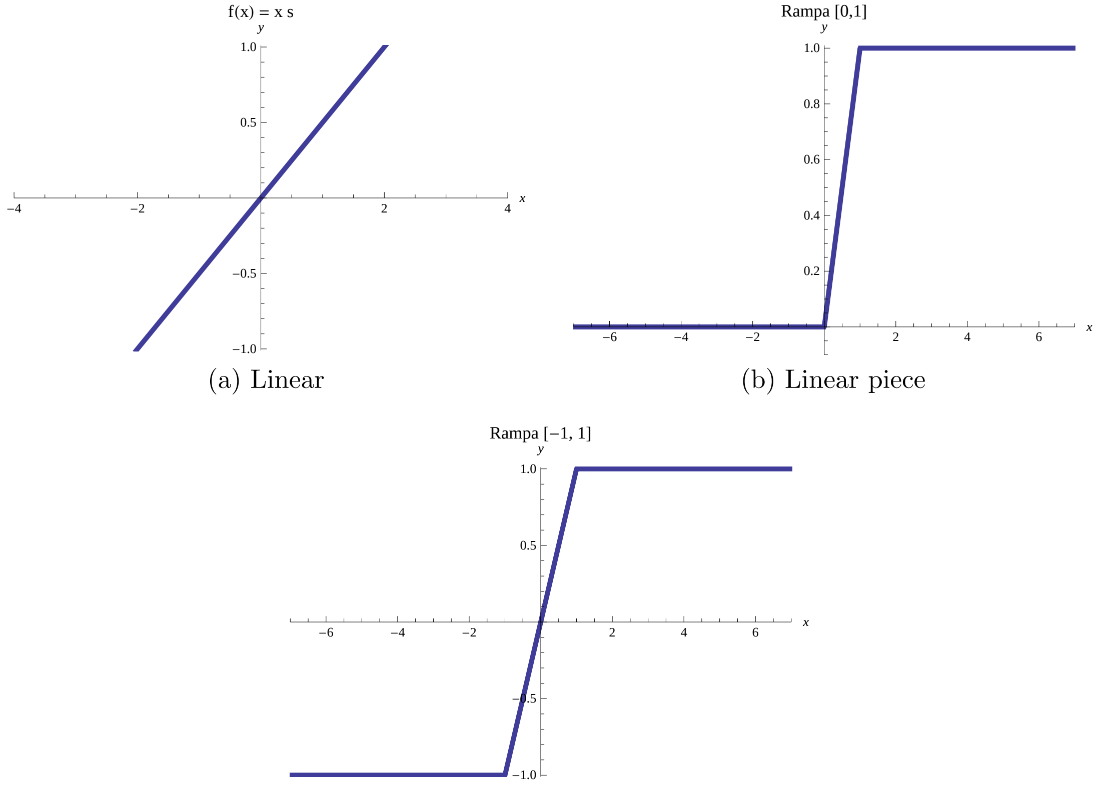
(b) Elliot symmetric

Figure B.3: Plots of Elliot functions used for training in GEANN.



(a) Gaussian (b) Gaussian symmetric

Figure B.4: Plots of Gaussian functions used for training in GEANN.



(a) Linear (b) Linear piece

(c) Linear piece symmetric

Figure B.5: Plots of linear functions used for training in GEANN.

Bibliography

[A. Asuncion, 2007] A. Asuncion, D. N. (2007). UCI machine learning repository.

[Aguirre et al., 2009] Aguirre, A. H., Borja, R. M., and García, C. A. R., editors (2009). *MICAI 2009: Advances in Artificial Intelligence, 8th Mexican International Conference on Artificial Intelligence, Guanajuato, México, November 9-13, 2009. Proceedings*, volume 5845 of *Lecture Notes in Computer Science*. Springer.

[Amari and Cichocki, 1998] Amari, S. and Cichocki, A. (1998). Adaptive blind signal processing-neural network approaches. *Proceedings of the IEEE*, 86(10):2026–2048.

[Arbib, 2002] Arbib, M. A. (2002). *The Handbook of Brain Theory and Neural Networks*. The MIT Press, second edition.

[Aristoklis D. Anastasiadis and Vrahatis, 2003] Aristoklis D. Anastasiadis, G. D. M. and Vrahatis, M. N. (2003). An efficient improvement of the rprop algorithm. In *In Proceedings of the First International Workshop on Artificial Neural Networks in Pattern Recognition ANNPR-03*.

[Barron, 1994] Barron, A. R. (1994). Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14(1):115–133.

[Berry and Linoff, 1997] Berry, M. J. and Linoff, G. (1997). *Data Mining Techniques: For Marketing, Sales, and Customer Support*. John Wiley & Sons, Inc., New York, NY, USA.

[Bishop, 2002] Bishop, C. (2002). *Neural networks for pattern recognition*. Oxford University Press, USA.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Blum, 1992] Blum, A. (1992). *Neural networks in C++: an object-oriented framework for building connectionist systems.* Wiley professional computing. Wiley.

[Boger and Guterman, 1997] Boger, Z. and Guterman, H. (1997). Knowlege Extraction from Artificial Neural Networks Models. *IEEE Systems, Man and Cybernetics Conference*, pages 3030–3035.

[Bow, 2002] Bow, S.-T. (2002). *Pattern Recognition and Image Preprocessing.* Marcel Dekker, Inc., New York, NY, USA, 2nd edition.

[Braun and Griebel, 2007] Braun, J. and Griebel, M. (2007). *On a Constructive Proof of Kolmogorov's Superposition Theorem.* Preprint. SFB 611.

[Brause, 2001] Brause, R. W. (2001). Medical analysis and diagnosis by neural networks. In *Proceedings of the Second International Symposium on Medical Data Analysis*, ISMDA '01, pages 1–13, London, UK, UK. Springer-Verlag.

[Buyantuyev et al., 2007] Buyantuyev, A., Wu, J., and Gries, C. (2007). Estimating vegetation cover in an urban environment based on landsat etm imagery: A case study in phoenix, usa. *Int. J. Remote Sens.*, 28(2):269–291.

[Castillo et al., 2007] Castillo, P., Guervós, J. J. M., Arenas, M. G., and Romero, G. (2007). Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters. *Inf. Sci.*, 177(14):2884–2905.

[Chen and Narendra, 2001] Chen, L. and Narendra, K. S. (2001). Nonlinear adaptive control using neural networks and multiple models. Technical report, Automatica, Special Issue on Neural Network Feedback Control.

[Cho, 1999] Cho, S.-B. (1999). Pattern recognition with neural networks combined by genetic algorithm. *Fuzzy Sets and Systems*, 103(2):339 – 347. ¡ce:title¿Soft Computing for Pattern Recognition¡/ce:title¿.

[Cios and Kurgan, 2002] Cios, K. J. and Kurgan, L. A. (2002). New learning paradigms in soft computing. chapter Hybrid inductive machine learning: an overview of CLIP algorithms, pages 276–321. Physica-Verlag GmbH, Heidelberg, Germany, Germany.

[Darwin, 1909] Darwin, C. (1909). *The origin of species.* Harvard classics. P.F. Collier & son.

[Dony and Haykin, 1995] Dony, R. D. and Haykin, I. S. (1995). Neural network approaches to image compression. In *Proc. IEEE*, pages 288–303.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2nd edition.

[Fahlman, 1988] Fahlman, S. E. (1988). An empirical study of learning speed in back-propagation networks. Technical Report Computer Science Technical Report.

[Floreano et al., 2008] Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62.

[Flores et al., 2009] Flores, J. J., Loaeza, R., Rodríguez, H., and Cadenas, E. (2009). Wind speed forecasting using a hybrid neural-evolutive approach. In [Aguirre et al., 2009], pages 600–609.

[Flores et al., 2010] Flores, J. J., Rodríguez, H., and Graff, M. (2010). Reducing the search space in evolutive design of arima and ann models for time series prediction. In [Sidorov et al., 2010], pages 325–336.

[Fogel et al., 1966] Fogel, L., Owens, A., and Walsh, M. (1966). *Artificial intelligence through simulated evolution*. Wiley.

[Gabrys and Ruta, 2006] Gabrys, B. and Ruta, D. (2006). Genetic algorithms in classifier fusion. *Appl. Soft Comput.*, 6(4):337–347.

[Gruau, 1995] Gruau, F. (1995). Automatic Definition of Modular Neural Networks. *Adaptive Behaviour*, 3(2):151–183.

[Gupta et al., 2003] Gupta, M. M., Homma, N., and Jin, L. (2003). *Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[Haykin, 2007] Haykin, S. (2007). *Neural Networks: A Comprehensive Foundation (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Haykin, 2009] Haykin, S. (2009). *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall.

[Hecht-Nielsen, 1989] Hecht-Nielsen, R. (1989). Theory of the backpropagation neural network. In *, International Joint Conference on Neural Networks, 1989. IJCNN*, pages 593–605 vol.1. IEEE.

[Herault and Jutten, 1987] Herault, J. and Jutten, C. (1987). Space or time adaptive signal processing by neural network models. In *AIP Conference Proceedings 151 on Neural Networks for Computing*, pages 206–211, Woodbury, NY, USA. American Institute of Physics Inc.

[Hoehfeld and Fahlman, 1992] Hoehfeld, M. and Fahlman, S. E. (1992). Learning with limited numerical precision using the cascade-correlation algorithm. *IEEE Transactions on Neural Networks*, 3:602–611.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, USA.

[Horton and Nakai, 1996] Horton, P. and Nakai, K. (1996). A probabilistic classification system for predicting the cellular localization sites of proteins. In *In Proceeding of the Fourth International Conference on Intelligent Systems for Molecular Biology*, pages 109–115.

[Huang and Babri, 1998] Huang, G. and Babri, H. A. (1998). Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Transactions on Neural Networks*, 9(1):224–229.

[Idan et al., 2001] Idan, M., Calise, A. J., and Parekh, D. E. (2001). Adaptive neural network based approach for active flow control. In *In ASME Fluids Engineering Division Summer Meeting, number FEDSM2001-18281.*

[Igel and Hüsken, 2000] Igel, C. and Hüsken, M. (2000). *Improving the Rprop Learning Algorithm*, pages 115–121. Citeseer.

[Igel and Hüsken, 2003] Igel, C. and Hüsken, M. (2003). Empirical evaluation of the improved rprop learning algorithm. *Neurocomputing*, 50:2003.

[Jacob, 2001] Jacob, C. (2001). *Illustrating evolutionary computation with Mathematica.* Evolutionary Computation Series. Morgan Kaufmann Pub.

[Jiang and hua Zhou, 2004] Jiang, Y. and hua Zhou, Z. (2004). Editing training data for knn classifiers with neural network ensemble. In *Lecture Notes in Computer Science, Vol.3173*, pages 356–361. Springer.

[Johnson and Calise, 2001] Johnson, E. N. and Calise, A. J. (2001). Neural network adaptive control of systems with input saturation. In *Input Saturation, Submitted, American Controls Conference*, pages 3527–3532.

[Kechriotis and Manolakos, 1993] Kechriotis, G. I. and Manolakos, E. S. (1993). Using neural networks for nonlinear and chaotic signal processing. In *Proceedings of the 1993 IEEE international conference on Acoustics, speech, and signal processing: plenary, special, audio, underwater acoustics, VLSI, neural networks - Volume I*, ICASSP'93, pages 465–468, Washington, DC, USA. IEEE Computer Society.

[Kecman, 2001] Kecman, V. (2001). *Learning and soft computing: support vector machines, neural networks, and fuzzy logic models.* Complex adaptive systems. MIT Press.

[Kennedy et al., 2001] Kennedy, J. F., Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). *Swarm intelligence.* The Morgan Kaufmann series in evolutionary computation. Morgan Kaufmann Publishers.

[Kovacs, 2011] Kovacs, T. (2011). Genetics-based machine learning. In Rozenberg, G., Bäck, T., and Kok, J., editors, *Handbook of Natural Computing: Theory, Experiments, and Applications.* Springer Verlag.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA.

[Kurgan et al., 2001] Kurgan, L. A., Cios, K. J., Tadeusiewicz, R., Ogiela, M., and Goodenday, L. S. (2001). Knowledge Discovery Approach to Automated Cardiac SPECT Diagnosis. *Artificial Intelligence in Medicine*, 23(2):149–169.

[Kurokawa and Takeshita, 2004] Kurokawa, T. and Takeshita, K. (2004). Air transportation planning using neural networks as an example of the transportation squadron in the japan air self-defense force. *Syst. Comput. Japan*, 35(12):46–56.

[Lapedes and Farber, 1987] Lapedes, A. and Farber, R. (1987). Nonlinear signal processing using neural networks: Prediction and system modeling. *Signal Processing.*

[Larochelle et al., 2009] Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40.

[Liu et al., 2001] Liu, Y., Yao, X., and Zhao, Q. (2001). Evolving a cooperative population of neural networks by minimizing mutual information. In *In Proceedings of the 2001 Congress on Evolutionary Computation*, pages 384–389. IEEE Press.

[Madden, 2002] Madden, M. G. (2002). Evaluation of the performance of the markov blanket bayesian classifier algorithm. *CoRR*, cs.LG/0211003.

[Maiorov and Pinkus, 1999] Maiorov, V. and Pinkus, A. (1999). Lower bounds for approximation by mlp neural networks. *Neurocomputing*, 25:81–91.

[Mangasarian et al., 1995] Mangasarian, O. L., Street, W. N., and Wolberg, W. H. (1995). Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43:570–577.

[McCulloch and Pitts, 1943] McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5:115–133. 10.1007/BF02478259.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning*. McGraw Hill series in computer science. McGraw-Hill.

[Ng, 1997] Ng, A. Y. (1997). Preventing "overfitting" of cross-validation data. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, pages 245–253. Morgan Kaufmann.

[Nilsson, 1998] Nilsson, N. J. (1998). *Artificial intelligence: a new synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Nissen, 2003] Nissen, S. (2003). Implementation of a Fast Artificial Neural Network Library (fann). Technical report, Department of Computer Science University of Copenhagen (DIKU).

[Nolfi and Parisi, 2002] Nolfi, S. and Parisi, D. (2002). Evolution of artificial neural networks. In *In Handbook of brain theory and neural networks*, pages 418–421. MIT Press.

[Pomerleau, 1996] Pomerleau, D. A. (1996). Neural network vision for robot driving. In *The Handbook of Brain Theory and Neural Networks*, pages 161–181. University Press.

[Prechelt, 1998] Prechelt, L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11:761–767.

[Rabuñal and Dorrado, 2006] Rabuñal, J. and Dorrado, J. (2006). *Artificial neural networks in real-life applications*. Idea Group Pub.

[Rechenberg, 1973] Rechenberg, I. (1973). *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipiender biologischen Evolution*. Number 15 in Problemata. Frommann-Holzboog, Stuttgart-Bad Cannstatt.

[Rechenberg, 1994] Rechenberg, I. (1994). *Evolutionsstrategie'94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Friedrich Frommann Verlag (Günther Holzboog KG), Stuttgart.

[Riedmiller, 1994] Riedmiller, M. (1994). Rprop - description and implementation details. Technical report, Proc. of ISCIS VII, Universitat.

[Riedmiller and Braun, 1992] Riedmiller, M. and Braun, H. (1992). Rprop - a fast adaptive learning algorithm. Technical report, Proc. of ISCIS VII, Universitat.

[Riedmiller and Braun, 1993] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE International Conference of Neural Networks*, pages 586–591.

[Ripley, 2007] Ripley, B. D. (2007). *Pattern recognition and neural networks*. Cambridge University Press.

[Rojas, 1996] Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer, 1 edition.

[Roseiro et al., 2005] Roseiro, L., Ramos, U., and Leal, R. (2005). Neural networks in damage detection of composite laminated plates. In *Proceedings of the 6th WSEAS international conference on Neural networks*, NN'05, pages 115–119, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS).

[Rowley et al., 1996] Rowley, H., Baluja, S., and Kanade, T. (1996). Neural network-based face detection. In *Computer Vision and Pattern Recognition '96*.

[Rumelhart and McClelland, 1986] Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. MIT Press, Cambridge, MA, USA.

[Schaffer et al., 1992] Schaffer, J. D., Whitley, D., and Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: a survey of the state of the art. *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*, pages 1–37.

[Schiffmann et al., 1993] Schiffmann, W., Joost, M., and Werner, R. (1993). Comparison of optimized backpropagation algorithms. In *Proc. of ESANN'93, Brussels*, pages 97–104.

[Shao et al., 2011] Shao, Y., Taff, G. N., and Walsh, S. J. (2011). Comparison of early stopping criteria for neural-network-based subpixel classification. *IEEE Geosci. Remote Sensing Lett.*, 8(1):113–117.

[Sidorov et al., 2010] Sidorov, G., Aguirre, A. H., and García, C. A. R., editors (2010). *Advances in Soft Computing - 9th Mexican International Conference on Artificial Intelligence, MICAI 2010, Pachuca, Mexico, November 8-13, 2010, Proceedings, Part II*, volume 6438 of *Lecture Notes in Computer Science*. Springer.

[Sivanandam and Deepa, 2008] Sivanandam, S. N. and Deepa, S. N. (2008). *Introduction to genetic algorithms*. Springer.

[Song, 2005] Song, C. (2005). Spectral mixture analysis for subpixel vegetation fractions in the urban environment: How to incorporate endmember variability? *Remote Sensing of Environment*, 95(2):248–263.

[Szirnyi and Csapodi, 1998] Szirnyi, T. and Csapodi, M. (1998). Texture classification and segmentation by cellular neural networks using genetic learning. *Computer Vision and Image Understanding*, 71(3):255 – 270.

[Theodoridis and Koutroumbas, 2008] Theodoridis, S. and Koutroumbas, K. (2008). *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition.

[To and Pham, 2009] To, C. and Pham, T. D. (2009). *Analysis of Cardiac Imaging Data using Decision Tree based Parallel Genetic Programming*, pages 317–320.

[Webb and Copsey, 2011] Webb, A. and Copsey, K. (2011). *Statistical Pattern Recognition*. John Wiley & Sons.

[Werbos, 1974] Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA.

[Wolberg and Mangasarian, 1990] Wolberg, W. H. and Mangasarian, O. L. (1990). Multisurface Method of Pattern Separation for Medical Diagnosis Applied to Breast Cytology. *Proceedings of the National Academy of Sciences,U.S.A.*, 87:9193–9196.

[Xu and Che, 2008] Xu, S. and Che, L. (2008). A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining. In *5th International Conference on Information Technology and Applications (ICITA 2008)*.

[Yao, 1993] Yao, X. (1993). Evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 4:539–567.

[Yao, 1999] Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

[Yao et al., 1998] Yao, X., (smieee, X. Y., and Liu, Y. (1998). Making use of population information in evolutionary artificial neural networks.

[Zilouchian and Jamshidi, 2000] Zilouchian, A. and Jamshidi, M., editors (2000). *Intelligent Control Systems Using Soft Computing Methodologies*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.