Bulgarian Academy of Sciences

Department of Parallel Algorithms
Institute of Information and Communication Technologies

4.6. Computer Science

Post-Learning Strategy and Evolutionary Architecture in Neural Networks

Kristina Georgieva Kapanova

1. Reviewer N/A
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences

2. Reviewer N/A
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences

Supervisors Ivan T. Dimov and Jean Michel Sellier

November 15, 2016

Kristina Georgieva Kapanova
Post-Learning Strategy and
Evolutionary
Architecture in Neural Networks
4.6. Computer Science, November 15, 2016
Reviewers: N/A and N/A
Supervisors: Ivan T. Dimov and Jean Michel Sellier

Bulgarian Academy of Sciences
Institute of Information and Communication Technologies
Department of Parallel Algorithms
Acad. G. Bonchev 25 A
1113 and Sofia

Acknowledgement

I would like to thank my thesis advisor Prof. Ivan Dimov, DSc for his patiance, guidance and support, his suggestions and ideas were very essential for the successful completion of this project. One can't forget his role in introducing me to Prof. Jean Michel Sellier who also became my thesis advisor. JM, our conversations, ideas, mutual passion for science were for me of utmost importance. From that, beautiful friendship was born and hopefully a long lasting collaboration. J.M. Sellier provided not only ideas for this thesis, guidance and proofreading of it, but also introduced me to the wonderful (and confusing) world of quantum mechanics.

Additionally, I would like to extend my thankfulness to the wonderful people and scientists from the department of Parallel Algorithms who welcomed me with such open hearts. Pencho, Stefka, Ceco, Rayna, Mixi have always been ready to help with whatever problems I might encounter.

A special mention goes to Nicky Georgieva and Raya Kraleva, without whom my life at the Bulgarian Academy of Sciences would never have been the same. Thank you for your support, encouragement and laughs.

Most of all, I would like to thank my family - my sister Mirka, my mother Katya and grandmother Milka, for standing by me in the weirdest ideas I come up with and for their constant encouragement, positive outlook, support and love.

Support

This research was supported in part by the Bulgarian National Fund, under grant BNSF I02/20 and DN 02/10.

Contents

Introduction

1

> *Intelligence is the art of good guesswork.*
>
> **— H.B. Barlow**
> (The Oxford Companion to the Mind)

In the past 25 years, scientists were focused on developing machines capable of beating the most proficient humans in games such as chess, checkers or Jeopardy!. In 2016 for the first time in our history, a machine by Google was able to claim a decisive victory in a game of Go over the best player in the world [26]. The Go game is over 2500 years old and it is considered one of the most complex games since it requires a degree of intuition. Why is it important for the future of science, technology and human progress?

Some, as the editor of "Six Generation Computing" ($1980s$), Derek Stubbs, believe that the most important leaps in the evolution of life encompass several stages. First is the ability to reproduce, the beginning of mutation and genetic variability (sexual reproduction), as well as the creation of multi-cell organisms. The second stage involves the development of specialized nerve cells and the nervous system, from which point, according to Stubbs, no more discernible leaps occur until the invention of computers. The final stage, and the most fundamental leap in our evolution, he concurs is the invention of **artificial neural networks**.

The history of artificial neural networks(ANN) has naturally progressed in parallel with the development of von Neumann-architecture computers. Artificial neural networks have been part of the field of Artificial Intelligence(AI), devoted to development of agents, displaying intelligent behavior [58]. Many philosophers and researchers in the past 80 years have tried to understand the principles of intelligent behavior and **how** organisms behave intelligently. Important consideration has been given to the internal functioning of the system. In this sense, the field of AI has been driven forward by two important research tracks - symbolic AI and neural networks [58].

In the subfield of symbolic AI, the path to achieve intelligence is viewed in terms of the manipulation of symbols according to formal rules. The artificial neural network subfield also known as connectionism, on the other hand, regards the possibility of creating an intelligent system through the development of a simplified neuron model by using four key elements of a biological neuron - dendrites, synapses, cell body and axon [75]. Through the simulation of a biological neural network, the ANN design aims for utmost computational and algorithmic simplicity, while the self-organizing feature contributes to the high adaptability to a broad range of problems of this network. Those features provide a distinct advantage over traditional computational paradigms, due to the lack of strict programming which sends instructions to the hardware.

Artificial neural networks incorporate a network of simple processing units, interconnected and some cases running in parallel. Each unit contains an activation value, which communicates with other units along connections of different strengths. Those connections determine how input to the processors is transformed into output. Significantly, the knowledge in a connectionist structures is encoded in the strength of the connections between the units and not in the symbolic structure of the machine. The parameters of the connections are found adaptively through a learning strategy until an acceptable solution is found. The particular learning paradigm, known as supervised learning, introduces to the system a training set of an input-output mapping. In the course of the training, the difference between actual output and the desired output (predicted by the model) is calculated and the connections between nodes are adjusted according to predefined principle in order to minimize the error between desired and actual output. The procedure is repeated multiple times, often randomizing the order of the training samples presented to the network until no considerable changes in the synaptic connections are observed. Thus the connections between the neurons determine the functioning of the artificial neural network. This adaptability of the neurons is essential characteristic of ANNs to adapt to various environments - including the ability to perform in real-time conditions with continuous change of information.

In contrast to von Neumann machines, which store information in a formal set of instructions in the memory, neural networks store the received information in the synaptic weights instead (in other words, the knowledge resides in the weights of the connection). Moreover, the information in neural networks can be transmitted not only to the immediate neighbors (unlike cellular automata), but also to the most distant nodes due to the hierarchical multi-layered structure of the network. As such, representation of the information is not found in a concrete programmed structures, but is instead distributed - the input pattern is represented over the whole network. This contextual knowledge may be affected by the activity of all other neurons in it. As a consequence there is no central processing unit in ANNs, rather each neuron operates independently, as well as concurrently with other neurons in the network. Consequently their parallel distributed structure provides effectual computing power. In terms of hardware implementation of a neural network, when hardware failure occurs, this inherent distributed robustness of the network allows it to adapt to the changing environment. The big advantage in the interconnectedness lies in the capability of neural networks to continue functioning even if damage in the network occurs, with performance drop in direct proportion to the amount of damage to the network. Contrasting to programmable units, where even a small change of the programmed instructions leads to a failure. In this respect, artificial neural networks emulate loosely the biological nervous system, where thousands of neurons die out annually, while the brain continues its work largely unaffected (unless a medical condition has not occurred) because of the high parallelism of the biological neural networks.

The benefits provided from the structure and functioning of artificial neural networks have been studied by cognitive scientists, neurobiologists, physicists, computer scientists and engineers throughout the years. The first mathematical investigation of the behaviors of neurons and networks of neurons (although less known) belongs to Rashevsky, who in $1930s$ published several papers developing the mathematical theory of nerves conduction based on electrochemical gradients [2]. The idea of

the Russian physicist was to utilize two linear differential equations along with a nonlinear threshold [133], which would provide underlying principles of the behavior of single neurons. Rashevsky was the first to argue that the neuron model could be connected in a network of neurons in order to produce complex behavior. Importantly, part of Rashevsky's research group was a logician - Walter Pitts, who in 1943 along with Warren McCulloch, a neuroscientist, would go to formulate the first theory for modeling artificial neurons. They described a neuron mathematically as a logical gate with two possible states - the neuron either fires or does not fire, thus performing a computational task [112]. Their model also established that the outputs of some neurons serve as inputs to others. The state of the neuron is then calculated as the linear summation of those inputs and weights, later compared to a preselected threshold. Should the summed signal be larger than the threshold, the neuron will fire and propagate the signal, otherwise it will not. After McCulloch and Pitts, Donald Hebb contribution to the field encompassed the introduction of a learning strategy by which biological neurons could actually learn on the basis of classical conditioning (Pavlov discovery) - in other words synaptic connections can be reinforced by the simultaneous and corresponding activity levels between neurons.

In 1958 Frank Rosenblatt introduced one of the first training algorithms for learning in neural networks in [138] and proving the convergence of the perceptron rule. Rosenblatt suggested the innovation of numerical weights, the computing units which serve as threshold elements and an interconnection pattern. In his model, the learning is achieved by adaption of the weights of the network through a predefined algorithm. The initial experimentation with the Rosenblatt's model brought an important question about the possible limitations that might be observed in various pattern recognition tasks and the ability of the network to solve them effectively. Some early experiments with the Rosenblatt's perceptron produced unrealistic expectations in the connectionism field.

Marvin Minsky and S. Papert had developed a simplified perceptron to scrutinize the computational capabilities of the perceptron network [119]. In "Perceprons" ($1960s$) the authors have demonstrated that a single-layer neural network is unable to solve a class of problems that are not linearly separable. Furthermore they held a rather pessimistic stance about the possibility that training multilayer networks can successfully solve those problems. This outline is believed to have had a crucial negative effect on the redistribution of strategic funding in the AI field and therefore the departure of many capable scientists. Nevertheless, scientific work continued on neural networks and in the $1980s$ the discovery of the error back-propagation learning algorithm provided a direct answer to the criticism raised in "Perceptrons" about the training of multilayer networks. This algorithm has been independently discovered by several researchers (David Rumelhart, James McClelland, Werbos, Parker) and brought renewed energy to the field. Considerable contribution to the revival of the field can be recognized in the invention of the cognitron by Fuskushima (1975) and Kohonen's investigation of neural network through the use of topological feature maps (1984). Additionally Hopfield introduced statistical mechanics to the field through the formalization of the Hopfield network, which aims for explanation of the behavior of a class of recurrent neural networks, used as an associative memory.

Many aspects of artificial neural networks have been studied since their inception, with over 40 different paradigms proposed and used for various task related problems. Nevertheless, the field faces many problems such as the inherent instability of the system depending on the chosen model. One of the most fundamental problems in the field is the existence of local minima and the possibility of the network to reach it during training without a potential escape from it. To deal with those inherent problems, various combinatorial algorithms have been proposed. Unfortunately not always a substantial improvement have been introduced due to slow convergence problems and the presence of multiple saddle points. The growth of network parameters provides additional burden to the convergence speed of algorithms and increase of the computational complexity of the system. The network parameters are generally based on manual design. The decision about the number of input and output neurons depends on the specific problem at hand. A developer, knowledgeable about the specific task, selects the number of hidden layers, number of neurons in every hidden layer, the training algorithm, as well as the type of activation function. Those components are crucial for the computational performance, efficiency and accuracy of the network but are often established through a trial and error basis and on the personal experience of the researchers.

One of the most important questions in the field of artificial neural networks and AI in general is the path for future development in the field. Moreover, the limited understanding of the function of the human brain artificial neural networks are considered to provided important insights into those functions. Therefore the field of artificial neural networks represent a multidisciplinary field, incorporating knowledge from neuroscience, mathematics, statistics, physics, computer science, engineering, etc. Artificial neural networks have been implemented in various other fields - process modeling and control, data analytics, robotics, environmental control, intelligent agents, where ANNs are a powerful tool for dealing with multidimensional data.

Objectives and Tasks

The main objectives of the current PhD thesis are from scientific and application perspective. They can be summarized as follows:

1. The development of a novel post-learning algorithm, which once implemented allows for the network to escape from possible local minima or saddle points reached during the optimization process.

2. The implementation of sensitivity analysis indicators to study the perturbation of the weights in the network, affected by noise in the system. The study of sensitivity analysis could provide beneficial to the area of hardware based neural networks, where noise from various sources influences the network's performance.

3. Development and implementation of a novel automatic evolutionary architecture of a neural network through a hybrid genetic algorithm. As such, trial and error approaches to network design are discarded for an automatic discovery

of network topology appropriate for specific tasks. The aim is to assist research with the decision making possibilities and computational complexity involved in designing neural networks.

To address these objectives the following tasks were constructed:

1. Development of a new optimization algorithm initialized once the training process of a network is completed.

2. Selection and practical implementation of 3 sensitivity analysis benchmarks to investigate levels of noise in the system in real time.

3. Development of a hybrid evolutionary algorithm, employed towards the automatic selection of a neural network architecture dependent on specific task.

4. Program implementation in C a neural network architecture, including the developed algorithms.

The current PhD thesis consists of 6 chapters. The thesis is 120 pages and includes 29 figures, 5 tables, 180 references. The framework of the thesis was developed on the basis of 3 articles, published in international journals with impact factors. The numerical experiments have been part of 2 scientific projects.

Chapter 2

Chapter 2 provides a critical review and analysis of the basic components of an artificial neural network, specific topologies and their real-life applications.

Chapter 3

Chapter 3 introduces the concept of the supervised training in neural networks. The training of ANNs is examined in terms of an optimization task. Several optimization (training) algorithms are presented - based either on deterministic or stochastic principles. Main advantages and drawbacks are described. The problems of local minima and saddle points, which restrain the algorithm to find the best possible configuration of the weights, are discussed. This chapter introduces a novel and effective post-learning strategy designed to escape such local minimas or saddle points at a relatively small computation cost. This novel method is based on analogy with naturally occurring quantum effects. Several numerical experiments are presented to validate the approach. The algorithm and results have been published in **K.G. Kapanova, I.T. Dimov, J.M. Sellier, On randomization of neural networks as a form of post-learning strategy, Soft Computing (**2015**).** $doi : 10.1007/s00500 - 015 - 1949 - 1$**, (IF.**$1.63$**)**.

Chapter 4

Chapter 4 presents several sensitivity analysis indicators to understand the influence of noise in the weights on the performance of the network. Those measurements provide a better understanding about the network's behavior and the sensitivity of the outputs from the perturbation of weights during the training process. By means of sensitivity analysis tools the acceptable level of noise which provides optimal

solution to the random fluctuations without sacrificing the behavior of the network is obtained. Three different indicators are utilized to observe the impact of the noise on the system and how it can be used in an advantageous way from an engineering perspective. The description of the sensitivity analysis method and the performed numerical analysis has been published in **K.G. Kapanova, I.T. Dimov, J.M. Sellier, A Neural Network Sensitivity Analysis in the Presence of Random Fluctuations, Neurocomputing (**2016**),** $doi : 10.1016/j.neucom.2016.10.060$**, (IF.**2.392**)**

Chapter 5

Chapter 5 describes a novel hybrid genetic algorithm for the automatic architecture of a neural network given a specific task. The algorithm provides several degrees of freedom on the parameters of the network: number of possible neurons, number of hidden layers, types of connections, types of transfer functions used and type of an optimization algorithm. The problem of expanding the space of possible solution with the increase of free parameters is discussed. It is shown that despite the multidimensional space of possible structures, the proposed method is capable of achieving sufficient performance at affordable computational cost. To validate the model, several numerical experiments are presented. The algorithm, as well as some of the results from the numerical analysis has been published in **K.G. Kapanova, I.T. Dimov, J.M. Sellier, A genetic approach to automatic neural network architecture optimization, Neural Computing and Applications (**2016**),** $doi : 10.1007/s00521 - 016 - 2510 - 6$**, (IF.**1.492**)**.

Chapter 6

The final chapter concludes with a discussion of future directions of study, as well as overall description of the PhD Thesis.

2 The Theory of Neural Networks

> *By far the greatest danger of Artificial Intelligence is that people conclude too early that they understand it.*
>
> **— Eliezer Yudkowsky**
> (Co-founder of Singularity Institute for Artificial Intelligence)

Artificial neural networks are computational units that emulate in a very restrictive way the biological nervous system. This chapter provides description of the particular computational paradigm, its underlying structure, as well as the information processing capabilities and limitations of the most widely applied architectures. The chapter begins with a brief explanation of the integral functions of biological neurons from the field of neuroscience. Next, we provide description of the intrinsic characteristics of artificial neural networks. We conclude with real-life implementation examples.

A biological neural network consists of nerve cells(neurons), coupled in a highly interconnected structure. It is determined that the human brain is composed of approximately 10^{11} neurons, responsible for the neural computational task, connected in a network structure through more than 10^{14} synapses [154]. For information to progress from one neuron to another, an electrochemical current travels between them through the neuron's axon, which can be described as a long fiber. This transportation of a current goes through the neuron, along the internecine connections called synapses (meaning electrochemical junctions located on the cell branches known as dendrites). Then the signal continues further through a very narrow synaptic space to the dendrites and/or soma of the next neuron at an average rate of $3m/sec$ (see fig. 2.1). By electrical current one considers the electro-chemical process of a voltage-gated ion exchange that travels through the axon. Each neuron can be connected to up to 10000 other neurons, transferring signals to one another via 10^{14} synaptic connections. In a similar manner, neurons may receive multi-fold inputs from many other neurons through their multiple dendrites. As such, our neural system is highly interconnected. Despite this interconnectivity, not all connections are equal - some have a higher transmission priority than others. Likewise, neurons can be either excitatory, and thus promote impulse generation, or inhibitory, preventing the neuron from firing.

Significant part of our initial neural structure is defined at birth, although throughout a person's life certain new connections are developed, while others die out. In peoples' lifetime we achieve strengthening or weakening of the neural junction by the modification of the synaptic strengths (also known as learning). The high connectivity, the generation of impulses and the ability of neurons to combine many diverse signals simultaneously provides a high level of parallelization in the brain.

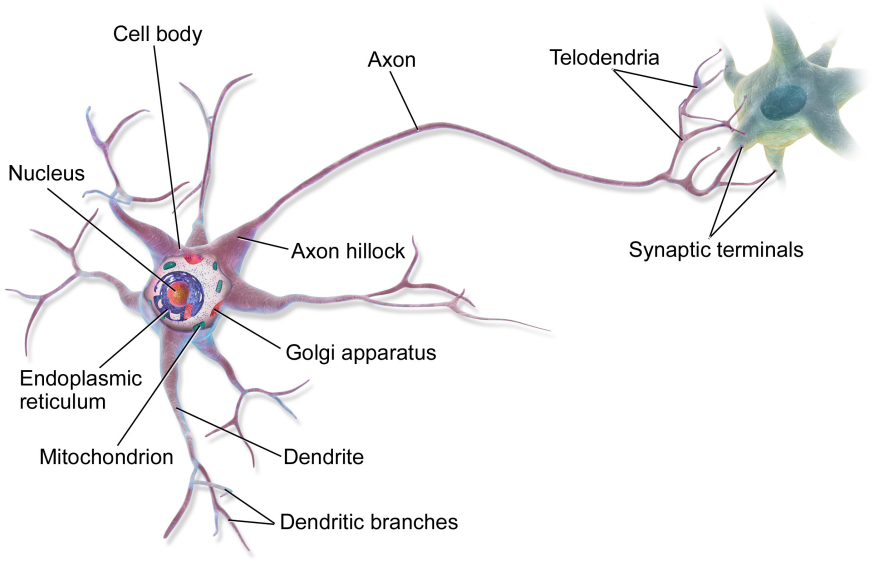


Fig. 2.1: Structure of biological neuron

Currently, artificial neural networks are incapable of achieving the complexity of the biological brain structure. Nevertheless, key structures of biological neurons are implemented, through which high interconnectivity is possible. The artificial neurons simulate the biological neural cells in a general manner. The synapses are depicted by a weight which is linearly combined with the neuron input and then sent to the neuron in order to provide an activation pattern. Similar to biological neurons, certain activation level needs to be reached, exceeding a certain threshold limit in order for a neuron to fire. This connectionist approach and various improvements in the field of artificial neural network has attracted considerable attention in the recent years. Researchers have focused on the development of an increasing number of network architectures new mathematical optimization models which can be implemented in various applications. It is important to remember that ANNs are very simplified models of the brain and therefore conclusions about the function of ANNs and its properties should not be translated as general conclusions about biological neurons. Yet, the universal approximation capabilities of neural networks are one of the reasons for their applicability to various scientific and engineering domains.

2.1 Mathematical Model of Artificial Neural Networks

In $1940s$ W.S. McCulloch, a neuroscientist, and W. Pitts, a logician, encouraged by Rashevsky, proposed in a paper [112] a mathematical description of an extremely simplified model of a neuron called "threshold logic unit" for complex pattern recognition task. Their neuron either produces an output or not. For a neuron to fire, more than 1 synaptic weight needs to be excited in a certain time interval. If instead an inhibitory synapse is active, it prevents the firing of the neuron at specific time period. Later, the set of processing nodes (neurons) that are connected to one

another has been described as an artificial neural network. The McCulloch-Pitts model was capable of converting a certain logical proposition in terms of an artificial neural network.

In this work, by network it is understood the reference to a system of artificial neurons. It can be something as simple as a single neuron or a large collection of nodes in which each one is connected to every other in the network according to a predefined set of conditions. The architecture of the network is defined as the type of connections between the neurons, the specific arrangement of neurons in the network, as well as the type of training algorithms and activation functions that are implemented. A feedforward neural network represents layered network architecture, where the neurons in the input layer are directly connected to the neurons in the output layer, and no feedback mechanism is provided (there is no backward connection). The predominant neural network architecture consists of several layers of neurons, which can belong to one of the following types of layers:

- **Input Layer** - the initial layer in the network, where the neurons receive the external input from external sources (such as data sets, sensors, real-time data).

- **Hidden Layers** - the neurons in the hidden layers generally interact with the neurons from the layer just below the current layer and the one just above it.

- **Output Layer** - the neurons in this layer produce the output of the network.

Connections. Each connection is defined by a certain weight value, acting on the connection strength between any two connected neurons. The connections between the neurons are not equally weighted. Some receive a higher priority than others, acting in a excitatory or inhibitory way, closely resembling biological neurons. From a computational perspective, the synaptic weights of artificial neurons can include a range of both positive and negative values. Since the type of connection between neurons determines the nature of the network, the specific connections between neurons are of fundamental importance. The type of connections in the network can be seen as another way to characterize the networks as cyclic or acyclic. Cyclic networks are the feedback and recurrent networks, while the feedforward and the perceptron networks (Rosenblatt, 1958), radial basis function networks, Kohonen maps and Hopfield network can be classified in the second group.

In a similar manner, the neural network architecture can be represented as a graph. The network is described in terms of vertices (nodes, units) and edges (connections). A graph/network (GNN) can be represented as a pair (N, E), where N is a set of neurons/nodes and E is a set of edges/connections between neurons from E. While classical graph theory deals mostly with undirected (random) graphs [73], [31], neural networks contain directed edges.

The variability of architectures can be broadly classified according to several important attributes. First, the architecture of the network relies on the specific type of task it will be used for. For example, one might implement an ANN for clustering task (data mining) where the network explores the similarity between patterns;

for a classification/pattern recognition tasks, where an input pattern is assigned a predefined class; or for function approximation used to ascertain the value of an unknown function $f = f(x)$ (applied mostly in the engineering and scientific field). Neural networks can be utilized for forecasting purposes in a dynamic system through time-sequenced data (especially for decision support systems). Second, the type of architecture selected by the developer- single or multilayer topology, recurrent network or some sort of self-organized architecture. Another essential attribute is the type of connections between neurons. Finally, one may classify neural networks on the type of learning method used in the design - whether supervised, unsupervised. In this chapter we provide a description of the fundamental elements of an artificial neural network.

Activation function. The power of an artificial neural network and its general non-linear approximation capabilities are derived by the activation function. The activation function provides not only the ability of the system to solve multidimensional problems, but also to determine the output value of each neuron. In this sense, the activation function is ultimately responsible whether a neuron will fire or remain inactive. Many discriminant functions have been implemented, although one must consider the importance of the nonlinear characteristic of an activation function so a network is capable of separate complex relationships in the feature space and therefore provide the appropriate classification boundary. Each neuron in the network needs an activation function, which is determined based on the specific position of the neuron in the network. The selection of such a function directly affects the performance and complexity of the network. In the field of Artificial neural networks, the conventional activation functions include the sign function, the sigmoid, the hyperbolic tangent and the Gaussian ones. Those functions depend on the discriminant function. For example, we could implement a discriminant function that computes **the dot product** between an input pattern x and weight vector w, or the discriminant function calculating the **the measured distance** of the weight vector w and the input pattern x.

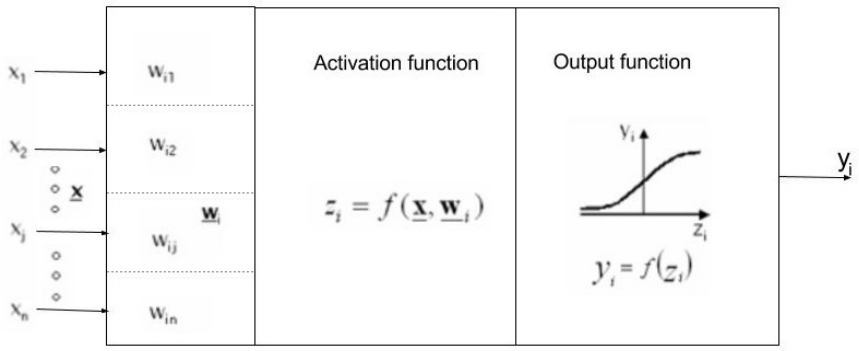


Fig. 2.2: Basic elements of a formal, static neuron.

The activation function can be represented in the following manner. Let x be the number of input variables of neuron i, the weight w_{ji} represents the information transmission between the neurons j and i, and z_i is the output of neuron i. The threshold value of neuron i is represented by θ, while σ stands for the activation function. The **bias term** is described as w_0. The Euclidean length of a vector x is

denoted by $\|x\|$ and the magnitude of the scalar is $|x|$. Finally, the slope parameter of the network is denoted by β, the matrix as C, and C_{ij}, where ij describes the ij-th element of matrix C (see fig. 2.2). Therefore,

$$z_i(w_j x) = \underline{w}_i^T \cdot x = \sum_{j=1}^{n} w_{ij} \cdot x_j = \left\|\underline{w}_j\right\| \cdot \|\underline{x}\| \cdot cos\gamma \tag{2.1}$$

which can be interpreted considering an equation such as

$$\sum_{j=1}^{n} w_{ij} \cdot x_j = 0. \tag{2.2}$$

Then the activation function $z_i = f(x_i, w_i) = 0$, if an input x is located on a hyperplane, and is thus traversing through weights w_i and w_j. Since each neuron i has a threshold value θ_i, one can describe the function as

$$\sum_{j=1}^{n} (w_{ij} \cdot x_j) - \theta_i = 0. \tag{2.3}$$

In the second case where the activation function is based on the measured distance, the activation of a neuron is computed as the distance of the weight vector (w_i) from the input value (x). Among possible choices for the computation of the distance, one can implement the Euclidean distance, where the input spaces are symmetrical, or the Mahalanobis distance - used for asymmetric input spaces in the covariance matrix C_i (see fig. 2.2), where

$$z_i(\underline{w}_j, \underline{x}, \underline{C}_i) = d_m(\underline{x}, \underline{w}_j, \underline{C}_i)\underline{w}_i = \bar{\underline{x}} = \frac{1}{N} \sum_{p=1}^{n} \underline{x}^p. \tag{2.4}$$

The activation function when maximum distance is considered is determined by the modulus of the components of the different vector $x - w_i$ where

$$m_i(w_j, x) = \max_{1 \le j \le n} |x_j - w_{ij}|, \tag{2.5}$$

and where the minimum distance corresponds to

$$t_i(w_i, x) = \min_{1 \le j \le n} |x_j - w_{ij}|, \tag{2.6}$$

while the Manhattan distance is determined by the linear combination of the absolute values of the vectors x and w_i where

$$b_i(w_j, x) = \sum_{j-1}^{n} |x_j - w_{ij}|. \tag{2.7}$$

One type of activation function where no threshold value is applied is the **The Identity Function**. The result is an output z, that is actually equal to the weighted sum of the inputs $z(x_i, w_i)$ and is see as actually representing the basic interpolation function [102].

The Threshold Function has been introduced by McCulloch and Pitts as all-or-nothing neuron activation procedure and is described in the following way:

$$f(z) = \begin{cases} 1 & : z \geq 0 \\ 0 & : z < 0 \end{cases} \tag{2.8}$$

The threshold function is used in a linearly separable problems, and generally applied to the output neuron of the network.

The Sigmoid Function on the other hand is best described as an increasing function balancing between linear and non-linear behavior [75], such as

$$y(z) = \frac{1}{1 + e^{\beta z}} \tag{2.9}$$

where β (as defined in the beginning of the section) represents the slope parameter of the sigmoid function.

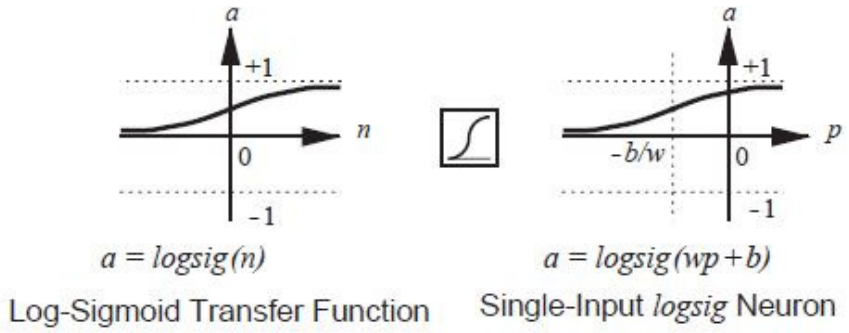


Fig. 2.3: Sigmoid activation function

One of the main drawbacks of the sigmoid activation function is that the error produced tends to be flat near the origin and the farthest from the origin point. In such case weight initialization with very small numbers is prohibitive. To alleviate such a problem, the addition of another linear term is recommended to avoid such regions. Another disadvantage of the sigmoid function is its limited range $[0, 1]$. Therefore, the hyperbolic tangent can be used in order to utilize a bigger range $[-1, 1]$:

$$f(z) = \begin{cases} 1 & : z > 0 \\ 0 & : z = 0 \\ -1 & : z < 0 \end{cases} \tag{2.10}$$

Cost Function. The goal of the learning algorithm is to attain a weight parameters of the network which will lead to sufficient solution to a certain predefined task. In order to achieve this, most training algorithms use certain cost function (i.e. error function) to establish the difference between desired and actual output. The maximization or minimization of this cost function is accomplished through a predetermined optimization (learning) algorithm. Therefore, the cost function is regarded as a parameter to assist the network in the generation of output signal that as close as possible to the desired one [153].

2.2 Neural Network Architectures

In this section we briefly introduce the most widely implemented neural network architectures and their mathematical descriptions.

One of the first network architectures is introduced in $1958s$ by F. Rosenblatt, when he incorporated a perceptron in a simple neural network to classify linearly separable patterns [138]. The Rosenblatt perceptron consists of a single neuron, with adjustable weights and bias (w_0). Rosenblatt has introduced for the first time also a novel algorithm applied to the free parameters of the system to adjust their values. Major drawback of this model is its convergence only when the system is presented with patterns from two linearly separable classes (known as perceptron convergence theorem). The Rosenblatt's perceptron is one of the fundamental parts of every neural network. While it provides an oversimplification to the biological neuron, it adheres to the following association of the input/output parameters. Let us define the number of inputs as $(x_1, x_2, x_3, \ldots, x_i)$, and the weight as w_i and the connection from neuron to neuron as w_{ij}. The output (Out) of a neuron is seen as

$$Out = \sum_{i=1}^{N} w_i x_i \tag{2.11}$$

constituting the weighted sum of the incoming inputs and weights. If a predefined threshold (θ) is exceeded, the neuron produces an output Out. As part of the training algorithm is the introduction of bias (w_0), which can be represented in the network as

$$Out = w_0 + \sum_{i=1}^{N} w_i x_i \tag{2.12}$$

In computational terms the bias represents the degree of variance between the desired and actual output of the network, which is averaged over the number of training data samples ($(x_1, y_1), \ldots, (x_n, y_n)$). During the initialization and subsequent training iterations, the bias level is acceptably large since there is a big difference between the desired and actual output. The advancement of the training algorithm should provide a constant decrease of the bias since the training influence of the data should be noticeable for regression analysis in ANN. A very well known trade-off in the field of machine learning and neural networks is specifically the **bias-variance dilemma** which is the minimization of the bias and variance parameters and therefore limiting the ability of the training to generalize beyond the training set [62]. Those parameters are actualized as minimization of the system to learn the underlying noise in the training data (i.e. overlearning). An efficient way to prevent such overlearning and increase in the bias term is a predefined stopping criteria. In this regard, the activation function is presented as

$$z = f(d) \tag{2.13}$$

where the operator z can be one of the previously described activation functions.

The Rosenblatt's implementation and the novel training algorithm has been shown to be of limited implementation to the types of problems it can approach. In 1965 M. Minsky and S. Papert published "Perceptrons" [119], where they introduce the

inherent limitations of the perceptron model. Minsky and Paperts underscored the ability of the perceptron to solve only linearly separable problems. Furthermore they have established that a singled-layer perceptron is incapable of realizing the **Exclusive OR** logical function, which is composed of the association of the binary patterns $(0;0)$ and $(1;1)$ in one class and patterns $(1;0)$ and $(0;1)$ to another class. For such function, a network and an optimization algorithm, with input values (x_1, x_2), weights represented as w_1, w_2 (i.e., fig. 2.4), and an assigned threshold (θ), and an activation function $(f(z))$ for the output node $z = w_1x_1 + w_2x_2$, where

$$f(z) = \begin{cases} 1 & : z \geq 0 \\ 0 & : z < 0 \end{cases} \tag{2.14}$$

will be unable to select weight values for each input pair values in the XOR function such that a proper output value is achieved. Moreover, Minsky and Papert considered the problem of scaling during the learning process, where the computational time for the training of the network increases rapidly for some problems when the number of inputs increases as well. In this fundamental work by Minsky and Papert using predicate calculus in the their analysis, have not considered the possibility of a multilayer perceptron network tackling the inherent limitations presented by the Rosenblatt's perceptron. In the years since the publication of "Perceptron", funding

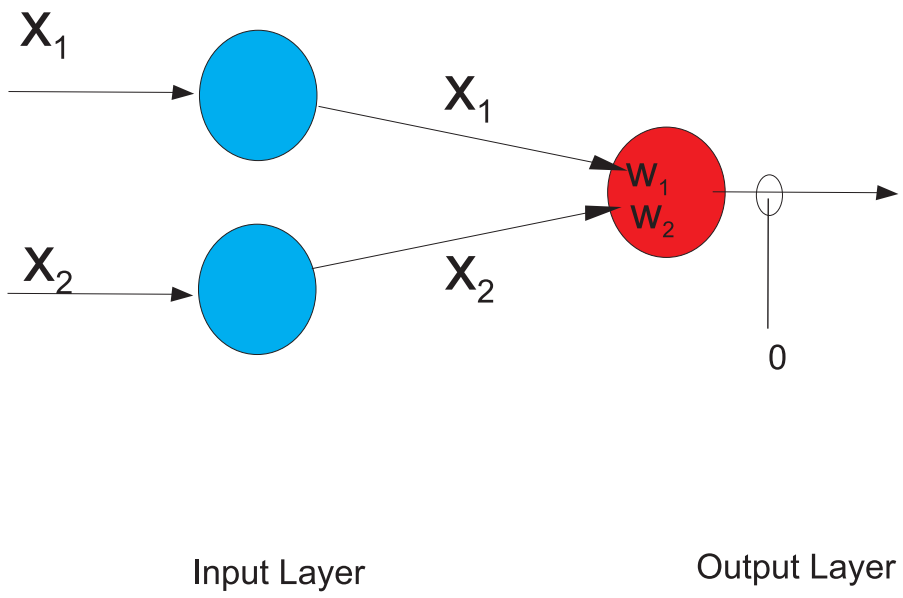


Fig. 2.4: A two-layer network, consisting of two input neurons (x_1 and x_2), which take value 0 or 1.

for the field of Artificial intelligence had been significantly reduced, nevertheless the scientists that remained active in the field focused on the task to overcome the limitations discussed by Minsky and Papert. In the next 50 years numerous network architectures with varying properties have been developed.

A variety of approaches to the design of neural networks have been considered in the literature, with few distinct considerations [129], [79], [80], [94]:

- The selection of the number of hidden layers;

- The number of neurons to be represented in every hidden layer;
- The selection of a globally optimal solution, which avoids local minima;
- Convergence time should be reasonable;
- Validation of the network for under or over fitting;
- Consideration for the interaction of hidden neurons and their interaction on a global level.

The ability to balance the considerations described above will provide a network preserving its function approximation capabilities without decrease of efficiency [61].

Multilayer Perceptron
The Multilayer perceptron network (MLP) is a feedforward network, consisting of many similar, non-linear neurons grouped in multiple layers, carrying many to one mappings from the input to a scalar output. The multilayer network is a modification of perceptron network (introduced by Rosenblatt), and has been proven to be capable of universal function approximation and classification of data that is not linearly separable [75], [21]. The connections between neurons in each layer are realized

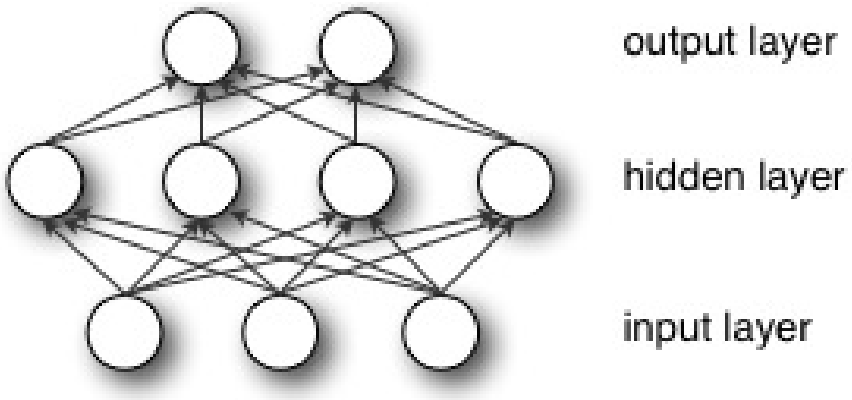


Fig. 2.5: A scheme of the multilayer perceptron network with 1 hidden layer.

in a feedforward manner (represented graphically in fig. 2.5) in a directed graph, where each layer is fully connected to the following one. The computation performed by the MLP network is mathematically described in the following manner. Let x be a vector of inputs, and z be a vector of outputs, with w representing the weights. The matrix of the weights A of the first layers, with θ the bias vector of the first layer. H and $\underline{b}$ represent the weight matrix and the bias vector of the second layer respectively. The function σ represent a nonlinear activation function. Therefore a multilayer network can be represented as

$$z = f(x) = H\sigma(Ax + \theta) + \underline{b}. \tag{2.15}$$

An MLP network with only 1 hidden layer is capable of modeling functions of arbitrary complexity [81]. The number of neurons in the input and output layers

determines the types of data the network can accept and the dimensions of the input feature, while the number of output neurons determine the dimensions of the output labels (especially for a multi-class problem). Therefore the neurons from the hidden layers are of tremendous importance for the final output of the system. A feedforward multilayer network (FNN) is based on the following mathematical model, where l is the number of layers in the neural network, net_j^l represents the $j-th$ neuron in the layer l, where $j = 1, \ldots, n_l$, with sum of the weighted inputs of the neuron. The weights from the $i-th$ neuron of the network in layer $l-1$ to the $j-th$ neuron of the layer l are denoted as $w_{ij}^{l-1,l}$, and the output of the $j-th$ neurons that is in the $l-th$ layer, with $f(net_j^l)$ the $j-th$ neuron's activation function. Therefore,

$$net_j^l = \sum_{i=1}^{n_{l-1}} w_{ij}^{l-1,l} y_i^{l-1}, y_j^l = f(net_j^l) \tag{2.16}$$

The neurons in the hidden layers are a significant design choice since the inclusion of too many neurons will decrease the generalization accuracy of the model and present the problem of memorization. In such case, the network provides too many degrees of freedom which enforce a type of memorization of the data rather than estimating the general features of the data. In a network where the number of hidden nodes is equal to the number of data inputs used for the training, complete memorization of the features occurs. On the other hand too few hidden neurons will be insufficient for the learning of complex decision boundaries. The precise number of neurons in each hidden layer is not established and it highly depends on a trial and error basis and the specific task to be solved by the neural network. An accepted approach to the problem is the application of the Kolmogorov's theorem which states that to compute any arbitrary continuous function, one needs to provide neurons equal to twice the number of input nodes plus one more. A different path is to compute the number of hidden neurons denoted by HM is represented by the number of neurons in the input layer k and from the output layer v, where

$$HM = \frac{1}{2}(k + v). \tag{2.17}$$

On the other hand, the high connectivity of a feedforward network can in certain situations be detrimental of the overall knowledge of the system for theoretical analysis and visualization of the learning process specifically in regards of the hidden layers. Therefore a feasible method to discover the number of the connections in a standard multilayer perceptron network with k input neurons, h hidden neurons and v output neurons can be described as

$$Weights = (1 + v) * h. \tag{2.18}$$

For non-continuous functions and most occurrences two hidden layers might be sufficient for the network to solve specific problems [40], [61]. This argument is supported in the context of a function $R^n \rightarrow R^m$ which can be approximated by a network with only two hidden layers and controlled number of neurons in each of them, assuming the backpropagation algorithm can find a global minimum or a local minimum close to the global one [97]. A network with more than one hidden layer is necessary for tasks where data is modeled with discontinuities [110]. The conventional approach to determine the number of neurons in every network have been adopted by experts in the field, however no precise formulation has

been proposed. The network architecture needs to achieve a satisfying performance on a data input it has never been trained on [141]. The data that the network is trained on, should also be considered, which should be representative enough for the specified task. A good training data size N should satisfy the following condition, where W denotes the overall free parameters in the network, ε - the classification error permitted on the test data, and the $O(\cdot)$ representing the order of quantity enclosed within, where

$$N = O\left(\frac{W}{\varepsilon}\right) \tag{2.19}$$

The selection of neurons in the hidden layers for specific tasks can sometimes introduce two distinct disadvantages. On one hand if there are insufficient number of neurons in the hidden layers, the neurons in the network are incapable of detecting the appropriate underlying causalities in a complex data set. Therefore, underfitting of the data occurs. In the other case, the reliance of too many neurons in the hidden layer could lead to several problems. Firstly, if the network has unused processing capacity as well small training data set, the network is unable to correctly train the neurons in the hidden layers and overfitting occurs. Whereas the training data is large enough, the computational time for the training of the network increases which could prevent the network to adequately learn. Determining the patterns from the input layer to be represented in the hidden layers of the network represents a crucial point for the training algorithm which works in a multidimensional search space of possible solutions [75]. Setting the network weights and activation levels also need to be carefully regarded to minimize the prediction error, which is the main goal of the training algorithms.

Radial basis function network

The Radial basis function network (RBF) is often considered as a subclass of the MLP networks. Whereas in MLP networks, one can add many hidden layers, the RBF network consists of a fixed three layer architecture, where there is 1 input layer, 1 hidden and 1 output layers [177]. The input data is linearly separated in the hidden layer. The design of the RBF network depends on several factors, including the network size, in initial parameters of the network (centers and widths) and the type of training. One can extend a multilayer perceptron network to radial basis network by increasing the input dimensions [174]. Due to its very limited 3 layer architecture, RBF networks are considered to be simpler than the MLPs. Since the network output is determined by specified hidden neurons in local receptive fields, RBF are considered as local approximation networks [175]. In contrast, multilayer perceptron networks are global approximation networks. On one hand because the neurons in RBF networks utilize only certain region from the input space, learning is achieved faster. Conversely, the locality is disadvantageous in highly dimensional spaces, where neurons need to cover increased input space [99], leading to high computational time. The initialization of parameters in the networks is achieved manually, they are not randomly generated. The neurons calculate the Euclidean distance between the input and weights as an and the activation function is an exponential. To estimate the output (z), lets denote the weights as w_i, $v_i(\sigma_i)$ represents the average standard deviation of the $i-th$ Gaussian, where

$$f(z) = \sum_{i=1}^{n} w_i \left(-\frac{1}{2\sigma_i^2}\right) \left\| z - v_i^2 \right\| \tag{2.20}$$

Additional disadvantage of the RBF network is that the centers of the function depend on the distribution of the input data, where the prediction tasks is not considered. Therefore, computational time increases with the amount of training data because the training algorithm covers input space areas even if they are not relevant to the training. A possible solution to this inherent disadvantage is the creation of centers to the training data points, leading to large memory storage spaces and thus to overfitting of the model [113], [11].

The Recurrent Network

While networks like MLP and RBF propagate the inputs only in a feedforward manner from layer to layer, cognitive science has established that our brain incorporates a feedback mechanism (i.e. recurrent signal). This underlying principle is part of the recurrent neural network (RNN), which works by incorporating the output signal as input after an learning iteration has been completed. This recurrent element occurs in discrete time steps of weight adjustment [140]. One of the computational advantages of recurrent neural network is the implementation of backrpopagation with small number of hidden layers. Comparatively, the recurrent network can be seen as employing s number of layers, which corresponds to the s cycles of the recurrent computation [55]. Therefore, the network topology represents a possibility to connect one neuron to any other in the network, even to itself, which allows the network architecture to have many different forms [168], [39], [32], [28], [72]. An MLP network can become recurrent if even one loop is added, thus exploiting the powerful non-linear mapping capabilities of MLP, as well as adding a form of memory to the system. Hence RNNs are best suited for temporal processing and sequence recognition or reproduction [41]. A fully recurrent neural network constitutes an MLP network where the previous set of hidden neurons feeds back into the network along with the inputs [49]. The "time" has to be discretized and the activations need to be updated at every time step. Additional delay neuron is introduced, which holds the activations until they are process at the next time step. Let the network inputs be denoted by $x(t)$, the outputs by $y(t)$, the weights from the input layer by W_{ih}, the weights from the hidden layer by W_{hh} and to the output layer by W_{ho}, the activation function for the hidden neuron by f_h and for the output neuron by f_o. The recurrent network behavior can be described as a dynamical system through a pair of non-linear matrix equations:

$$h(t+1) = f_h(W_{ih}x(t) + W_{hh}h(t))y(t+1) = f_o(W_{ho}h(t+1)) \tag{2.21}$$

The state of the dynamical system, which describes the information about the past behavior of the system in order to provide a unique description of the future behavior, is defined here by the set of hidden neuron activations $h(t)$. To convert a recurrent network to a feedforward one can use unfolding over time (see fig. 2.6).

Deep neural networks

The neural networks described so far present a shallow architecture (many neurons in few layers), and are successfully implemented in many fields. The increase of data availability and due to the fact that those data sets contain multidimensional data, scientists have focused on the development of deep neural networks(DNN), which apply more than two hidden layers to a conventional multilayer network [144]. DNNs are utilized in tasks such as handwritten character recognition, audio

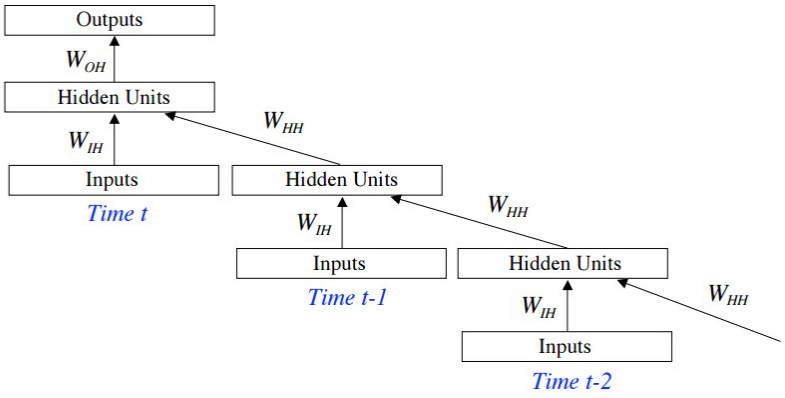


Fig. 2.6: Unfolding of a recurrent neural network to a feedforward one

classification, acoustic modeling, motion recognition, phone recognition, facial expressions and many others [10].

In this thesis we implement a multilayer neural network, and therefore the focus is on its functions and implementation in greater detail.

2.2.1 Evolution of network topologies

The development of neural network architecture today is mostly based on manual design. To determine the network topology, the number and type of connections and number of layers and neurons when developing a neural network, one still needs to rely on general recommendations and arbitrary methods, as well as on the personal experience of the developer. While certain methods exist for the automatic network construction, one needs to consider beforehand a specific topology, which is not always appropriate for specific training data. Since every type of neural network architecture has certain benefits and disadvantages, depending on the specific problem for which it has been applied, a lot of research has been focused on developing evolutionary neural network architecture, through which we can reduce the model complexity of the topologies. In this regard many self-organizing topologies have been proposed and implemented [24], [176]. Yet, such strategies often have to consider the magnitude of the space of possible neural networks, which often is so complex and large that automatic search for the optimal network architecture may present a computationally intractable, or in some cases impractical solution for certain applications [29], [38].

The substantial research in the evolution of neural network topologies can be broadly separated in two categories: the first group is concerned with parametric learning, where the search is focused on the specific weight values. In this case, all weights (whether in binary or real representation) are encoded in a chromosome, and the algorithm searches for the acceptable weights by using a genetic selection. The network topology is defined with the sequential feedforward connections prior to the initialization of the network. After the parameters are defined, an evolutionary algorithm is utilized to perform a search only on the synaptic weights in the topology,

selecting the ones that are with the best performance [98]. In such a case, the size of the network presents computational challenges. If the network is large and densely connected, encoding the weights might be inefficient. Therefore, a hybrid approach where genetic global approach is combines with a backpropagation (local search) which provides fine tuning of the weights has been beneficial [1].

The second group represents structural learning, where the implemented algorithms search for the best topology of neurons and connections. Here, the evolutionary algorithm responsible for the selection of the topology often implements a tactic either to add or prune neurons and connections between them. This method introduces significant computational costs during the network run in two regards. The pruning technology requires the initial creation of a bigger network (more layers and neurons) than is actually necessary. The network is trained and once the training is accomplished the pruning of neurons and connections from the network occurs according to a predefined algorithm if they are not used in an active way [135]. Therefore, this pruning methodology leads to increase in the training time, slow convergence, overfitting problems, especially in methods that remove hidden neurons, thus leading to increase of the error level [6], [14].

The constructive algorithm, computationally less demanding, initializes a small network with a minimal number of neurons, layers and weights. It evolves the topology appending additional neurons and weights during the optimization process. One of the benefits of the algorithm is provided by the fact that most small networks are sufficient for function approximation and therefore, the implementation of the constructive algorithm is requires less computational time since it is initialized with a minimal network construction and small training data sample. A disadvantage of the model is the propensity of the algorithm to fall in structural local optima, unable to identify available parameters [9], [92]. The cascade correlation algorithm is a constructive algorithm, evolving the layers in a sequential manner with heterogeneous neurons, non-restricted network topology and possibility to choose a transfer function [53].

A hybrid approach, combining parametric and structural evolution, like the Topology and Weight evolving artificial neural networks [156] can also be implemented, although depending on the task type and size of the network, the algorithm might be inefficient due to the computational complexity to reach a solution. A possible solution to the computational time is the utilization of a predefined list of possible modifications, as the addition to the evolutionary models only fully connected hidden neurons [156], [157]. The strategy evaluates the developed structures and if they do not pass certain fitness threshold, they are discarded and new topologies are created until a stopping criteria is met. Further symbiotic and adaptive neuro-evolution strategy involve the random co-evolution of populations of neurons and networks that are randomly composed [122], [121].

The implementation of an automatic evolutionary strategy has to account for increased computational time and complexity, depending on the free parameters of the system. Moreover, the general space of network topologies and training algorithms can be so large that is impractical to search efficiently. On the other hand, such

algorithms could reduce the time necessary to develop a network on an *ad hoc* basis, as well as remove the human decision making process [66]

2.2.2 Noise in Neural Networks

Similar to biological neurons, for which a brief description has been given in the beginning of the chapter, artificial neurons experience a presence of noise, defined as a "random or unpredictable fluctuations and disturbances" [54]. The manifestation of noise can interfere with the signal (noise in the input of the network), effectively distorting the information transfer and therefore affecting the information processing capabilities of both biological and artificial neurons [170]. The presence of noise in the biological systems is constant part of the information processing functions and multitude of studies have focused on determining the strategies the nervous system employs to counter and compensate for the existence of noise in the structure. The expression of noise can appear from different sources. Electrical currents produce electrical or channel noise through the random opening and closing of gated ion channels [159]-[167]. The electrical noise leads to membrane-potential fluctuations even in the absence of input signals. Furthermore, membrane fluctuations are affected by the noise in the dendrites and in the soma, which can have significant consequences to the initiation and propagation of the action potentials. The latter has been proved to be driven by a small ionic current situated inside the axon and is naturally noisy. Whereas the presence of noise is inescapable byproduct of the computational capabilities of the brain, affecting it in some way, it is nevertheless capable of functioning reliably under certain noise constraints.

Analogous to biological systems, the field of artificial neural networks needs to understand how noise influences the performance of the network and whether we might be able to exploit it to our advantage and produce better network solutions. An idea to use noise accumulation from the randomness in the cellular system to influence computation and topology interactions was proposed in [27]. A strategy to apply a sensitivity analysis on the level of noise in the system to understand how the performance of the network is affected depending on the level of noise in the data and how one can produce an advantegous results without a destruction to the overall realization of the system [105].

It is possible that the noise can provide certain benefits to information processing in ANNs. The **stochastic resonance** strategy applies noise when the threshold level of the system needs to detect and transmit weak (periodic) signals, thus strengthening the signal by adding noise to them. The capability of a signal to pass through the threshold is reinforced by the addition of noise and as such if the noise level is too modest, few signals pass and loss in communication occurs. On the other hand, amplifying the signal with increased amounts of noise will deluge the system with noisy information and as an incorrect output will be carried through. If the right controllable amount of noise is introduced, the signal reaches the activation without altering significantly the output [19]. In engineering systems, the existence of noise is considered as a negative to the performance of the system and the quality of the output. In [95] it is argued that neural networks operating by absorbing certain levels of noise, will be on average more robust, capable of exploiting more states of

solutions and capable of learning and adapting to changing environments. Exploiting the variability of these random fluctuations and disturbances could lead to certain benefits to the system and thereupon enhancing the information processing tasks [54], [8] [51], [43], [158]. Noise fluctuations have been further utilized during the training process of a neural network [83]. In such cases, noise has been artificially added to the synaptic connections in either noncumulative or cumulative (with additive and multiplicative) process on every time step or per string. The objective is to provide the training algorithm with a possible escape from local minima and consequently improve the convergence rate of the method. Alternative advantage is provided by the acceleration of the learning stage [123]. Positively affected are the fault-tolerance and error-function of the network.

If a combinatorial optimization strategy is selected for the training process, the addition of noise to the algorithm during the random probing of the space of possible solutions could provide a better selection of weights which will decrease the value of the cost function. [123], as well as certain effects on the fault-tolerance and error-function of the network.

In the hardware implementation of neural networks, where enclosure of many electrical devices lead to electrical fluctuations, such variations influence the performance of the neurons and their interaction with the network. In the context of the Memristor devices, developed by Hewlett-Packard and IBM [132], the limited physical space of the device contributes to a higher electrical current activity and the presence of random fluctuations, which affect the performance of the system.

Therefore, for neural networks, implemented either on hardware or software level, the ability to measure in a computationally efficient manner the sensitivity of ANNs in the presence of noise could provide significant advantage for understanding how the presence of noise can be utilized to improve the function of the network. On the other hand those mathematical indicators could also be of immeasurable assistance to determine the detrimental levels of noise in the system and how the performance is affected.

2.3 Practical Applications of Neural Networks

Nowadays, the renewed interest in the field of artificial neural networks and their applicability for many problem solving tasks has led to their pervasive use in many research fields and application domains.

Robotics. Currently neural networks have many practical applications. One is in the field of robotics, where the networks are used as a way to direct manipulators to hold objects, based on some sensory data. Neural networks are applied in robot steering and path planning [46], [103] as well as object manipulation. Successful implementation is the Baxter robot [50], [91] in the manufacturing industry.

Machine Vision. The field of machine vision deals with the ability to retrieve information about the environment through the processing of data from a collection of images. One might further separate the field in low and high level vision. The former deals with image filtering, feature detection, etc. The high level vision is occupied with segmentation and control of data from the low level, which is combined with other knowledge and further analyzed. Deep neural networks like Supervision, consisting of more than 650000 neurons and utilizing more than 60 million parameters during the training and recognize objects in particular categories [44], Neural networks has been essential in the object recognition research, where generating precise and detailed object recognition in real time environment becomes crucial. In such context of extreme importance is not only the correct classification of images, but also the correct estimation of the class and location of objects in the images [93].

Financial Analysis. Neural networks have been employed with considerable success in the financial/banking sector. They are used in daily basis in measuring credit card risk for applicants, for loan evaluation procedure and loan rates prediction, mortgage screening, real estate appraisal, financial market and exchange rate forecasting as well as corporate financial analysis [128], [12]. The ever increasing volume in stock trading provides a furtive ground for the employment of neural networks for forecasting financial time series, something that is computationally prohibitive to implement with traditional statistical models. The reason behind this is that data in dynamic systems forecasting possess certain attributes like non-stationarity (changing over time), nonlinearity (the relationship between economic variables changes), and constant variations in the time series). Although those attributes provide a challenge to traditional statistical methods, robust neural networks are very well suited to work with such data types for both multivariate analyses. Artificial neural networks are applied for the daily forecast of the direction of the Standard & Poor's (S&P 500) index [125]. Recurrent neural networks on the other hand are used to predict the exchange rate of currencies for a certain time period [126] and more.

Manufacturing. Another field where artificial neural networks has found incredible usage in the last 15 years in in manufacturing. Such activities include planning and management, process control, product analysis, machine diagnosis and analysis, quality analysis for different products (from the food, chemical, chip industry, etc) [139]. The current demand for high quality and low cost production has been essential for rise of automated manufacturing systems. The effective monitoring and fault diagnostics in real time allows for the identification of possible failures before they occur and thus reducing the downtime while insuring high productivity. This monitoring information gives insight into the whole manufacturing process and enables managers to perform high-level decision making at low cost. Since it has been established that over 70% of operational costs are for maintenance procedures [163] such monitoring and diagnostic systems provide a valuable tool for cost reduction and high effectively of the manufacturing plant. The implementation of neural networks in the monitoring and diagnostics systems have been valuable tool for identification of "unknown" situations. The neural network ability to learn and adapt to new conditions, as well as its nonlinearity has been proven effective in early fault detection and warning, providing significant less cost for maintenance,

part replacement, decreasing the downtime and providing efficiency in scheduling for work [5], [70].

Medical Research. Currently many medical personnel are employing neural networks for cancer cell analysis, optimization of surgical time slots, EEG and ECG analysis, hospital quality control [67], [161], [36], [7]. In the sub field of diagnostic systems, neural networks are widely employed to detect cancer and heart conditions, the analysis of blood samples, to track glucose levels, detect pathological conditions such as tuberculosis. The need for deploying efficient medical imaging techniques has led to the usage of neural networks in the field. From the point of view of patients, imaging enhances the medical service provided to them, improves the decision making process based on actual evidence. For medical professionals medical imaging with underlying neural network architectures can produce a better performance, as well as efficient and focused treatment of patients. Neural networks have been also implemented in robotic surgeries where incredible sensitive equipment is used [104]. Implementation of deep neural networks in the prediction of bioactivity of small molecules in drug-discovery has been successfully shown [171].

Telecommunication Industry. Although largely unnoticed by the general population, neural networks have been used in automated information services, in real-time translations (Google translate), customer payment processing systems and image and data compression. In 2015 Google introduced for the first time a deep neural network inside their Google Translate application for phones, capable of real-time visual translation of more than 20 languages. While the question of training the network is of great importance, fundamental is the work of the neural network on mobile phones where computational capabilities are limited due to hardware and mobile data limitations. Therefore the development of small neural network is limited on the density of information it can handle. The team from Google has focused on the amount of training data to be used for their network, where rotation of the symbols needs to be the correct amount as not to introduce noise in the sample. The next step is to establish which training data samples are sufficient and why. Thus in only a few minutes, Google can change the training algorithm used in relation with the above considerations [65].

Latest Developments. The ability of artificial neural networks to learn have been of great use in various scheduling activities - vehicle transportation scheduling, and routing system of various kinds. The swiftkey (a keyboard app for Android phones) is also utilizing neural networks to predict and correct language. Nowadays Facebook has been using GPU based neural networks to recognize images, but have also used ANNs to create automatic pictures based on specific understanding of what objects look like, especially in the production of thumbnail images [45]. On the other hand, Google has used neural networks and let them produce visual elements (see fig. 2.7) the network wants to emphasize on [57].

Artificial neural networks have been also deployed in self-driving cars. Recently NVIDIA has proposed that neural networks will accelerate self-driving cars and provide them with self-awareness especially in precision vision where huge amounts of information need to be processed and acted upon in navigational tasks. Those essential needs are provided by deep neural networks to turn the information into

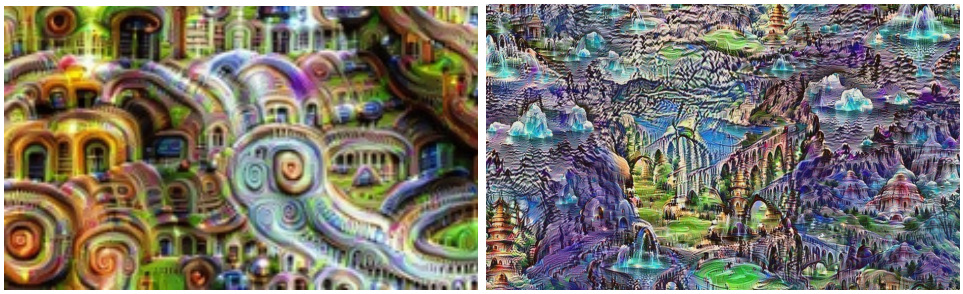


Fig. 2.7: Images courtesy of Engaget.com

three- dimensional images and assist vehicles navigate the world surrounding them [148].

Researchers from Microsoft and other scientific institutions have used neural networks for acoustic analysis, natural language processing and especially in the game industry [146], [117], [118].

Hardware implementation of neural networks, the SyNAPSE project, a collaboration between IBM and Hewlett-Packard, has been one of the leading research initiatives in the field [85]. The project has received over thirty million dollars (27 million Euros) grant from DARPA to create this "cognitive computer" which is essentially made up of thousands of parallel CPUS, each a square micron in size representing the neurons and connection of a neural network. DARPA is not limiting its financial focus only on this project. In fact, the annual budget of the organization is over sixty one million dollars (55 million Euros) for Machine Learning and almost fifty million dollars (or 45 million Euros) for Cognitive Computing [15].

The description of artificial neural network implementation is not an exhaustive list. The variety of ways for the application of neural networks represents also the many limitations of ANNs, especially for their training. The next section of the Thesis will introduce a novel post-learning strategy which can possibly overcome certain training methods limitations.

3 The problem of neural network training and implementation of a post-learning strategy

> *For Reason, in this sense, is nothing but Reckoning (that is, Adding and Subtracting).*
>
> — **Thomas Hobbes**
> Leviathan

3.1 Training of neural networks

Humans differ from other species in variety of ways, but the one of a paramount importance is their ability to learn. Through the process of learning, humans are capable of discerning patterns in the real world, to reach conclusions and develop non-genetically based behaviors to advance their survival. The field of artificial neural networks has attempted to emulate certain learning capabilities through the applicability of symbolic, data-intensive and statistical approaches. Nowadays there are myriad of algorithms and techniques applied in various domains in the field of neural network training (i.e. learning [1]). Any attempt to accomplish an overview of the methods and applications used for training would be incomplete. One may classify the learning algorithms as **supervised and unsupervised learning** (see fig. 3.2), being differentiated through their learning structure. Some consider reinforcement learning as a separate learning category, or as a subcategory of supervised/unsupervised learning [75]. The basis for many neural network learning algorithms is structural inferences focusing on the relationship among the variables and the data structure. We can formulate any learning task for a neural network in terms of a variational problem. Data modeling types like function regression, pattern recognition and time series prediction, could also be presented as a variational problems and are the most common problems encountered in neural networks.

Functional Regression is a traditional learning task for neural networks and can be recognized as a function approximation from provided data ($\hat{f}(x)$ to $f(x)$)[143]. The neural network has to determine the relationship between the input and output set from an input-target training examples, which can be expressed as a training set $T = (x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)$ where x_i ($x = (x_1, x_2, x_3, x_i)$) are the input vectors. In such an instance, the algorithm utilizes those learning patterns and tries to correctly separate them into classes with the help of a supervisor (i.e. teacher).

[1] The reader should note that in this thesis we interchangeably use the words training, learning and optimization having in mind the same meaning.

The main goal of the learning algorithm is to provide good generalization of the data. The topic of function regression suffers from two problems - underfitting and overfitting (e.g. lacking necessary complexity for a good data generalization). The overfitting problem occurs in the increase of complexity of the model and leads to a increase in the cost error value. On the other hand, underfitting lacks data complexity and therefore it needs to be changed, but it is generally very difficult to predict such an outcome.

Pattern Recognition (or classification) is also a traditional learning problem for neural networks One can recognize pattern classification as a process where the system receives a pattern, which can be described by a set of features, and is assigned to one of a predefined number of classes. In such instance, a neural network has to determine the relationship between the input and output set from an input-target training examples. The input/output pattern can be described as a set of patterns $T = (x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)$ and the goal is to model the ensuing probabilities of class membership, predicated from the input variables. For a pattern recognition problem to be completed, one needs to have the input space separated into regions, each representing a specific class. The decision boundary represents the border between any two regions (see fig. 3.1. The objective is to obtain a function $\hat{f}(x)$ as an approximation of the $f(x)$ [21]. The pattern recognition task also experiences problems of underfitting and overfitting. In the current situations underfitting represents too simple decision boundary which classifies poorly the training data. Overfitting, on the other hand, produces too complex decision boundary. It can produce a good training data separation, but poor generalization capabilities (the test data). In unsupervised learning, on the other hand, there is no teacher present

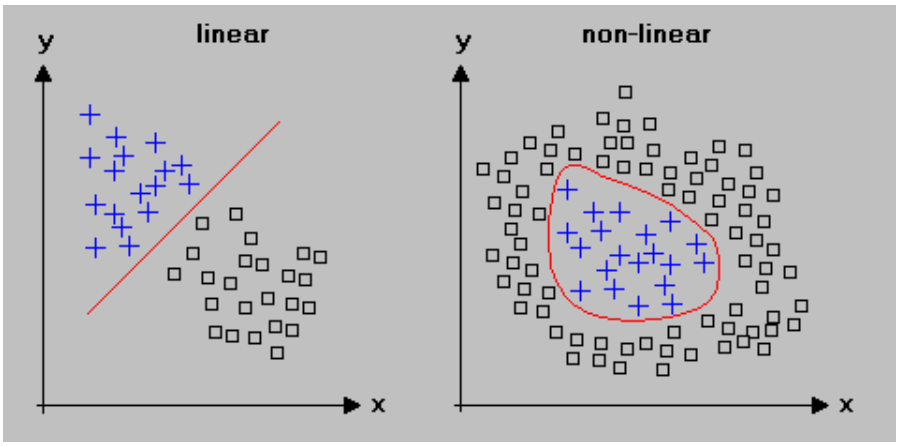


Fig. 3.1: Example of linearly and non-linearly separable classification tasks

(desired response), thus impossible to provide an explicit error information which acts as a function to improve the output of the network. Learning is accomplished through observations of previous responses to inputs for which the network has limited knowledge - the network identifies the pattern class through heuristics, with no known output vector required. Reinforcement learning is the implementation of learning through trial and error through the network interaction with the environment. This method was developed by Donald Hebb and is very well known as Hebbian learning or Pavlov training.

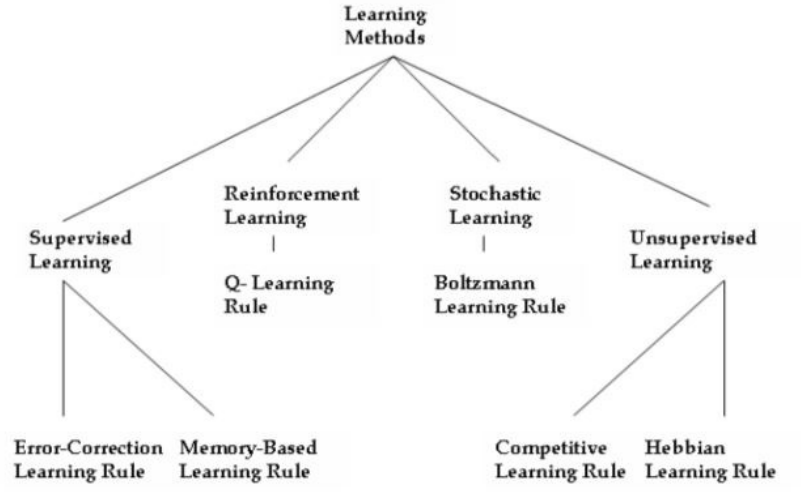


Fig. 1. Learning Rules Of ANN

Fig. 3.2: A simplified graph on the most common learning modes

There are many variations of training algorithms. Some have been inspired by natural effects, some are deterministic or stochastic (see fig. 3.3). In instances where simple non-convex functions are considered with well separated local minima and consisting of several dimensions, deterministic approaches like steepest descent and quasi-Newton can be successfully applied. Some of the drawbacks of such algorithms is the possibility of the algorithm to be trapped in a local minimum. Therefore, stochastic algorithms are often implemented. They are less likely to fall into local minima and are capable of reaching an acceptable solution near the global minimum. This is achieved with lower computational complexity than the one for deterministic models. Most widely implemented stochastic algorithms used in the optimization of neural networks have been genetic algorithms [77], evolutionary algorithms [160], simulated annealing [90], and others. Whence many problems are known in the field of artificial neural networks, several disadvantages during the learning process include the error surface falling into local minima, saddle points and plateaus, as well as long narrow ravines.

3.2 The training of Neural networks as an optimization problem

For the training of a network to find the optimal set of weights and therefore produce the desired output we need to define the technique through which this is achieved. Therefore the training of artificial neural networks may be represented as an optimization task, i.e. the selection of efficient strategy to discover the minimum or the maximum values of a function, which may consists of many independent variables. In the field of neural networks such a function is considered to be the cost function (e.g. energy or objective function). It is dependent on the actual

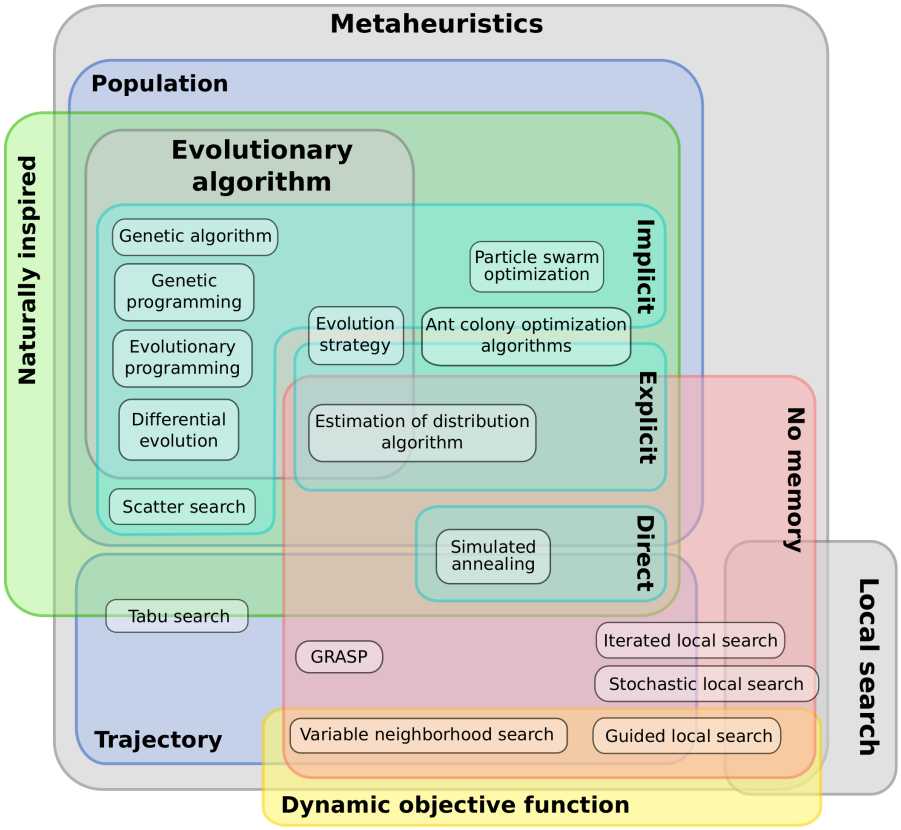


Fig. 3.3: Classification of metaheuristics methods. The representation is provided by Johann "nojhan" Dre under $CCBY - SA3.0$ license

configuration of the various parts of the systems. The cost function represented the actual measurement of the appropriate solution in a complex system. The optimization algorithm works to reach the best state of the system according to an objective function [142]. The whole process is repeated until no further improvement is discovered or the system reaches the maximum amount of iterations. In supervised learning, the operational cost generally searches until the error signal is close to or equal to zero. The problem is defined as one of function minimization [21]. The probabilistic function is used for function modeling and pattern classification [75]. One should consider the error calculation as a computationally complex task for the high dimensionality of the weights space in the presence of the objective function consisting of multiple local minima and broad flat regions with narrow steep ones [75], [140].

The optimization process is initialized by the random setting of the weights according to predefined strategy. The next step is the introduction of the training patterns (x_i, y_i), with x_i representing the input, and y_i the desired/correct output to the input sample. The network computes the output z, which will initially differ from the desired output defined by y_i. The cost function here is implemented to measure the difference between the expected y_i and desired output z, During the training of the network, the weights are changed in order to minimize the error.

The selection of appropriate error function depends on the particular optimization algorithm, as well as on the topology of the neural network (see table 3.2 for other cost functions [75], [21]). Some of the frequently utilized cost functions include::
Linear
Quadratic
Sum of squares (The least squares)
Smooth nonlinear
Nonsmooth
Additional constraints to the objective functions can be defined as **bound, linear, general smooth, discrete** or unconstrained. If, for instance, the cost function follows a Gaussian distribution, the most frequently used cost function will be the **Quadratic cost** (e.g. mean squared error, maximum likelihood, and sum squared error) (L_2 in table 3.2). The squared error function can be considered as a continuous differentiable function. One can define it as the sum of the squared error for each separate sample, over the samples from the input-output data set. Let us define the number of samples in the data set by T, and the number of output variables by m, where

$$E\left[y\left(x;\underline{\alpha}\right)\right]=\sum_{i=1}^{T}\left(E^{(i)}\left[y\left(x;\underline{\alpha}\right)\right]\right)^{2}=\sum_{i=1}^{T}\left(\sum_{k=1}^{m}y_{k}\left(x^{(i)};\underline{\alpha}\right)-t_{k}^{(i)}\right)^{2} \tag{3.1}$$

Variants of the sum squared error (SSE) are the mean squared error (MSE) and the root mean squared error (RMSE). Both have the same properties as the SSE. In this case the value of the error does not increase with the size of the data size. The mean squared error can be expressed in terms of the sum squared error, as

$$MSE=\frac{1}{T}SSE, \tag{3.2}$$

while the root mean squared is given by

$$RMSE=\sqrt{\frac{1}{T}SSE}. \tag{3.3}$$

Cross-entropy cost. Also known as Bernoulli negative log-likelihood. In a single neuron, let us define the number of samples in the training data by T, the inputs and desired outputs by x and y respectively, and let the output of the neuron be $z=f(x)$ with $C>0$ where

$$C=-\frac{1}{T}\sum_{x}\left[ylmz+(1-y)\,ln\,(1-z)\right] \tag{3.4}$$

The cross-entropy is valuable to address the learning slowdown problem in back-propagation.

Exponential cost Let us construe a cost function to be of the form $C(W,B,T^{i},E^{r})$. The exponential cost can be defined with the weights of the network represented by W, the biases by B, the input of a single training sample by T^{r}, and the desired output of that training sample to be E^{r}. If we define a neuron j in layer i, then the function is also dependent on the input and outputs y_{j}^{i} and z_{j}^{i} respectively, for those values are dependent on W, B, and S^{r}. We furthermore choose a parameter τ for

extracting a desired behavior. This cost function needs certain trial and error testing until the appropriate behavior is found, where:

$$C_{EXP} = \tau \ \exp\left(\frac{1}{\tau}\sum_{j}\left(a^{L_j} - E^{r_j}\right)^2\right) \tag{3.5}$$

The calculation of the gradient in regards to an output and a training sample r is

$$\frac{2}{\tau}\left(a^{L} - E^{r}\right) C_{EXP}\left(W, B, T^{r}, E^{r}\right) \tag{3.6}$$

Name	Cost Function
L_2	e^2
L_1	$\lvert e \rvert$
L_p	$\frac{1}{p} \cdot \lvert e \rvert^p$
$Logistic$	$\frac{1}{\alpha} \cdot log\left(cosh\left(\alpha \cdot e\right)\right)$

Most Used cost functions. α controls the robustness of the outliers, the desired signal is denoted by d, the actual output by z and the error signal by e, with $e = d - z$

The supervised learning algorithms applied to the training of neural networks are focused on the minimization of the cost function in order to solve a particular task. The algorithms need to possess several important characteristics [75]:

- Need to be able to solve a task with minimum computational burden.
- The algorithm should be robust in the presence of noise.
- The algorithm should be able to produce similar outcomes independent of the initial conditions used during the initialization.
- The generalization capabilities should provide adequate outputs when the neural network is provided with data different from the training set.
- The computational burden of the algorithm should not be strongly dependent on the dimensionality and size of the problem and training data. It should be scalable. This problem is more pronounced currently in big data.
- Convergence requirements.
- Setting appropriate stopping criteria in order to find the best possible outcome.

Even though there are many variations of neural network architectures and learning algorithms employed, every algorithm changes the network parameters according to certain rule in order to accommodate the network's characteristics to its desired pattern.

Currently, there are many training algorithms that use various optimization techniques [130], with gradient search algorithms like backpropagation being often the preferred ones. The gradient descent algorithms are local search ones, achieving the best solution in the regions where the search has initiated from. Therefore the initial choices by the developer influence the ability of the algorithm to reach a global solution.

3.2.1 Backpropagation

The backpropagation (BP) training algorithm is one of the most frequently implemented optimization methods in multilayer perceptron networks. It is based on the gradient descent (GD) method. The objective is to minimize the chosen error function through the reduction of the gradient of the error curve [75]. In the course of the training, the difference between actual output and the desired output (predicted by the model) are propagated back through the architectural configuration of the network. The selected cost function C_x needs to satisfy the following for individual training examples x

$$C = \frac{1}{n} \sum_{x} C_x. \tag{3.7}$$

The algorithm computes the gradient, which is itself dependent on the weights and biases for every single training data point. Important characteristic of BP is that the computation starts from the output (final) layer. The cost function C_x is not dependent on a particular activation value but on the output values. If the cost function relies on on any other layer different than the output one, the backpropagation technique is invalid since no accurate tracking backwards model is present (see Algorithm 1 for the steps of the algorithm). In order for a neurons j to produce an output $0 \leq z_j^L \leq 1$, the algorithm needs an activation function. We define the cost function within the range $\sqrt{z_j^L}$, with $z_j^L \geq 0$.

Algorithm 1 Backpropagation learning algorithm

for d in data **do**
  Starting from the input layer, do a forward pass trough the network, computing the activities of the neurons at each layer.
  Compute the derivatives of the error function with respect to the output layer activities
  for layer in layers **do**
    Compute the derivatives of the error function with respect to the inputs of the upper layer neurons
    Compute the derivatives of the error function with respect to the weights between the outer layer and the layer below
    Compute the derivatives of the error function with respect to the activities of the layer below
  end for
  Updates the weights.
end for

If one defines the objective function for the backpropagation algorithm to implement the squared error cost function:

$$J(w) = \frac{1}{2}\sum_i (y^i - out(x^i))^2 \tag{3.8}$$

we then use the gradient descent to train the weights (w):

$$w := w - \alpha \frac{\partial J(w)}{\partial w} \tag{3.9}$$

and with the backpropagation algorithm we can compute $\frac{\partial J(w)}{\partial w}$. Even though the gradient descent method is useful for representation of nonlinear boundaries and is commonly implemented for small feed-forward architectures, there is no guarantee it will reach a global optimum. In terms of computational cost, the computational burden for the gradient descent algorithm increases with the addition of hidden layers and neurons in the architecture. One calculates the computational cost with the necessity of $O(n)$ operations, where n is the number of weights in the network. To alleviate this problem, a stochastic gradient descent has been implemented where the weights are updated after each propagation (epoch). In batch learning, on the other hand, a predefined amount of propagations occur before the weights are updated in a single training step. In this process, the whole set of training data is introduced at each step to establish the weights. Stochastic backpropagation is advantageous in dynamic environments where the algorithm goes through the data in a random order in order to reduce the local minima problem. In this case, the algorithm updates the weights at each iteration after examining a random training sample. For the stochastic gradient descent, the backpropagation for a particular training sample can be viewed in the following manner. Let us define the activation of a node j in layer i by z_j, and the weights by w. If we consider the sigmoid activation function of the neuron to be $g(z)$, when

$$\frac{\partial J(w)}{\partial w} = (y - out(x)) \left[\frac{\partial (x)}{\partial w}\right] \tag{3.10}$$

then

$$a_j = \sum_i w_{ji} z_i z_j = g(a_j). \tag{3.11}$$

Using the chain rule, and defining $\delta_j \equiv \frac{\partial out}{\partial a_j}$ we have

$$\frac{\partial out}{\partial w_{ji}} = \frac{\partial out}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i. \tag{3.12}$$

Then we calculate the activation of neuron z_j with a forward steps, meaning we apply the current input x with the current weights w to the network and we compute all the values for the neurons in the hidden layers, starting from the bottom up. We then calculate the δ_j which can be done from top to bottom (or so called backpropagation). If the chain rules for the partial derivatives is:

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial w} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial w} \tag{3.13}$$

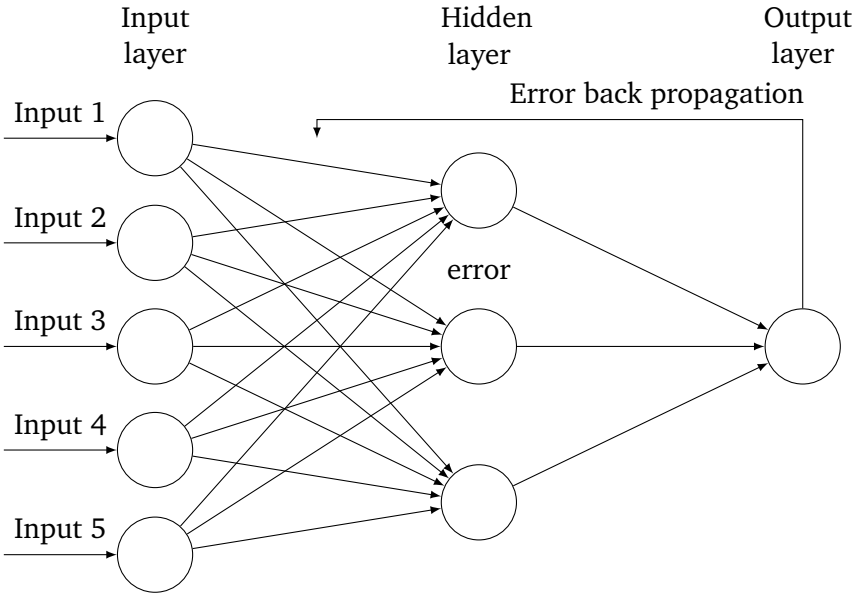
which is verifiable with $f(x, y) = xy; x = 2w; y = 3w$, with x and w being vectors. Then:

$$\delta_j = \frac{\partial out}{\partial a_j} = \sum_k \frac{\partial out}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k w_{kj}\frac{\partial z_j}{\partial a_j} = g'(a_j)\sum_k \delta_k w_{kj} \tag{3.14}$$

where $g' = (z) = g(z)(1 - g(z))$ for the sigmoid activation function. We compute The initialization of the recursion by

$$\delta_{out} = \frac{\partial out}{\partial a_{out}} = g'(a_{out}). \tag{3.15}$$

Whence the backpropagation algorithm is a preferred training method for neural networks, it is often trapped in a poor local optima in either batch-mode or in stochastic gradient descent when it starts at random initial points. The problem increases severely with the expansion of the network architecture, often falling into local optima, of being trapped in local minima [3, 100] and suffering from slow convergence and instability, depending on thousands of iterations [92],[124]. The backpropagation algorithm is known to overshoot the minimum of the error surface [84] especially in a complex weight search space, with the presence of multiple local minima. The specific network architecture is known to influence the convergence of the algorithm- in particular a direct consequence to the processing capabilities of the network is the amount of hidden layers and neurons, the learning speed, as well as the preconfigured initial condition and the size of the training sample [16],



[4]. In the situation where the error surface is irregular, then the backpropagation algorithm, as well as its variations like QuickProp [52] and RProp [136] or Silva-Almeida implementation [150] are in all likelihood to reach a local minima. The Gradient descent (and the hill climbing) strategy is efficient in situations where the search space consists of only one minimum and whenever this is not realized, the technique falls in local minima. Moreover, the random initialization of the parameters and the dependence on the initial conditions could produce an outcome where the algorithm is positioned between local maximum and local minima, or in some

kind of saddle point, thus going in a downward direction and sticking to the local minimum [165].

Despite the many advantages of the backpropagation algorithm, it has also shown significant drawbacks [75, 21]:
Neuron Saturation. In the case of a feedforward multilayer neural network which has been trained with backpropagation, the gradient descent method in many cases fails to converge entirely for many reasons and various improvements have been proposed [107, 52]. This failure of convergence may be caused by improper network architecture (not enough/too many layers and/or hidden neurons). A different reason for this failure of convergence can be due to saturation of some neurons before others in the network. Saturation is observed when the output is found near extremes before convergence is achieved. In this situation the derivatives are so small, they are incapable of achieving further weight changes. This in turn leads to the algorithm settling in a local minimum. Saturations may be detected due to many reasons - initialization of weight is too big, the weight mapping is inappropriately set, the learning rate is too big or the training set is not normalized. Further reason for saturation is the presence of overtraining and therefore overfitting of the data [71].
Weight initialization. In backpropagation the initialization of the weights is of significant role, since depending on it the algorithm might fall into a local minima. The role of weight initialization can be described in terms of the convergence speed (initial distance to the minimum). The nature of the algorithm means that it will find the closest minimum to the initial point. If the weight initialization was not efficiently calculated then algorithm is trapped into a local minimum (initial distance problem). Finally if the weight initialization is too large, neuron saturation may be observed.
Plateaus. Since the weight update is proportional to the error derivative, if the latter approaches zero (or is a very low value), weights are hardly updated and the algorithm falls into a flat region incapable of finding an escape solution.
Learning rate. Depending on the chosen learning rate, the algorithm can becom instable if too high a rate is chosen. On the other hand, too small value leads to slower convergence speed.
Stopping Criteria. Various strategies are utilized as a stopping criteria - fixed number of iterations, predefined error threshold, discovering plateaus in the objective function. One observes that up to a certain epoch, the error decreases and a good generalization of the data is achieved. After that point, the error increases again and overfitting of the patterns occurs. Therefore the training should be stopped. For this reason a sufficient number of training patterns needs to be achieved.
Architecture of the network. The training algorithm is susceptible to the precise network architecture, the number of layers and neurons.

In the last decades many numerical optimization solvers have been utilized to improve the efficiency of the backpropagation algorithm or to overcome its weaknesses by including global search techniques. Such algorithms can be considered to be the ant colony technique, genetic algorithms, evolutionary algorithms (EA), simulated annealing (see fig. 3.3. In [25] the presence of a trade-off when one considers the efficiency of metaheuristic training as compared to gradient descent methods. While in some cases it has been shown that evolutionary training is more efficient than backpropagation [120], in other cases no significant discrepancy has been seen in

the training methods and in fact according to some analysis the efficiency of the training algorithm really depends on the particular problem [22]. Furthermore, local optimization algorithms, such as backpropagation the selection if initial weights often leads to falling into a poor local optimum. On the other hand metaheuristic methods are global algorithms and they are less sensitive to those initial weight settings and do not rely on the initial weights neighborhood solution.

3.2.2 Stochastic

Stochastic algorithms are capable of handling multiple solution search spaces and able to provide a solution in such an environment. While this is a certain benefit, since they rarely fall in local minima, they are often computationally demanding and can not always offer a fast convergence since the large space of possible solutions to be searched.

Genetic Algorithm With supervised learning, one may use the genetic algorithms (GA), which has proven to be an efficient and advanced heuristic optimization techniques. It is derived from biological evolutionary processes such as mutation, inheritance, crossover or selection. Introduced by Holland in the $1960 - s$ [76], the genetic algorithm is suitable for solutions with high degree of complexity, which are often large, non-linear and discrete. There is no inherent guarantee that the algorithm will be able to find a solution close to the global optimum. The algorithm is initialized with the random generation of a large number of populations, which are crossbreeded in a similar manner to biological evolutionary process. The algorithm samples the search space, directed toward an area of the best solutions to this moment. An objective function - usually minimization of the sum of squared errors (or the absolute errors) is chosen for the optimization task. The algorithm measures each population fitness function according to an evaluation test for each individual. This can be done when applied to a training set, measuring the actual and desired output and thus reach the fitness of the individual. Once the fitness is calculated, the individuals are "mated". Those individuals are selected stochastically, where the chance to be chosen is equivalent to the relevant fitness value. One needs to calculate the total fitness in the simulation, which is provided by the linear summation of all individuals fitness values from the population. Once done, the algorithm selects two individuals and recalculate again until a predetermined number of individuals has been chosen for mating or some different criteria has been met. We proceed with pairing and mating the individuals which can be accomplished in several possible ways. We can have several cross-points, which actually breaks an individual string to interchange parts of it with some of its mate. Before they are reinserted in the population, we mutate them. Not all new individuals are subject to mutation. We select an offspring for mutation according to some probability calculated to establish whether a new trait could be an asset in the fitness value. The mutation is achieved through choosing a value (or a number of them), which is randomized (or changed). After the mutation, the new individual is reinserted into the population. Following is a restart of the entire process, which continues until a certain rule is satisfied. One of the best advantages of genetic algorithms compared to backpropagation learning is that GA rarely gets stuck in local minimum. Provided enough iterations and high population diversity, GA will always provide an optimal solution. In the case where

the population is stuck in local minimum solution from the search space, there is the availability of a fitness penalty to any individual when his fitness comes close to that of another individual. Therefore diversity is achieved/maintained.

In the implementation of local optimization methods like backpropagation, which are hill climbing algorithms, we depend on the ability to find a suitable starting point [87]. To find a suitable starting point is therefore a daunting task considering the various degrees of freedom in a network and therefore to increase of the number of evaluations. This makes local optimizer impractical. Although genetic algorithms are influenced by the degree of freedom in the network, their essential global optimization structure gives them the ability to scale rather efficiently to multi-dimensional problems. A distinct advantage is their ability to work good even with noisy evaluation functions. Furthermore, GA can be easily adjusted depending on the problem at hand, although genetic algorithms are known to suffer from premature convergence to a local extrema which is obtained by incompatible initial configuration.

Yet, certain optimization problems are not suitable for genetic algorithms since the fitness function is not known or not enough information is available for it. In such instance, the fitness function provides bad chromosome blocks, even in the presence of good block cross-over. Moreover, there is no absolute guarantee that GA will reach a global optimum, especially when populations generate a lot of subjects for evaluation. Similar to other methods, computational time is crucial and combinatorial optimization techniques suffer from inability to provide constant optimization response times. Compared to traditional gradient descent methods GAs shortest optimization time is much larger. Combinatorial optimizations are with limited implementation in real-life applications so far due to the demand of high computational time to generate random solutions. Moreover, certain convergence problems may arise even in the presence of population improvement, if the individuals lack the ability in the sample to improve. Genetic algorithms require much more time for function evaluation than linear methods. There is also the inherent need for discretization of the parameter space. The inability to generate all possible permutations for populations, leads to overfitting of the training data. Low usability of results can be due to lack of enough iterations (generation permutation), leading to long optimization times.

Algorithm 2 GA selection algorithm

```
Choose an initial random population
Evaluate each candidate solution
while Termination condition is not true do
    SELECT individuals for the next generation
    RECOMBINE pairs of parents
    MUTATE the resulting offspring
    EVALUATE each candidate solution
end whileend
```

Particle Swarm Optimization In 1995 a new method for parameter optimization - particle swarm optimization (PSO), was introduced in [48]. The technique is based on the principle of dynamic systems such as bird flocks and fish swarms or swarms of bees, which search for food resources where the perfect location of the food in unknown in the beginning. The main goal of the algorithm (based on stochastic global optimization technique)) is by virtue of swarm of particles (the potential solution) *flying* through the multimodal search space in the pursuit of the best location (global optimum) through communication with one another [88], [89]. The PSO algorithm initializes with the generation of a group of random particles. Each particle maintains an individual memory of the best position it has attained so far (known as *pbest* value). Furthermore, each particle tracks the best solution obtained by any other particle of the swarm (called *lbest* or global memory). Hence, the particles exchange information between themselves and therefore influence each others' movements in order to reach a global best (*gbest*) value. At each time step, the optimization consists of changing the velocity of each particle to reach the pbest or lbest solution. One achieves particle acceleration by the execution of a random number generator. Once the particle finds the best values, the velocity and positions of the particles are updated. The particles can algorithmically be allowed to fly fast and far from the best discovered positions in order to explore unknown areas (the global search) or it can fly slow and close to a specific position (the fine tuning) in search for a better result. A definition can be given in the following manner. Consider that v_i and x_i are velocity and position vectors of particle i respectively. The best local position found by particle i can be denoted by P_i, while the best global position discovered by the whole population is described by P_g. The learning factors are parameters c_1 and c_2 which are positive constants, where c_1 represents the particle moving to its best position, while c_2 denotes the same but for the global position. Then, the algorithm can be described as:

$$v_i \leftarrow wv_i + c_1 r_1 (P_i - x_i) + c_2 r_2 (P_g - x_i), \tag{3.16}$$

and

$$x_i \leftarrow x_i + v_i \tag{3.17}$$

Initially, all particles are updated according to the described equations. Once a new generation of particles are created, then they are used for the search for best position in the search space [116]. In the case of multilayer neural network training, the algorithm combines all the weights of the network in a vector, which is considered as the solution in the search space. The swarm and its particles represent solution candidates. According to a predefined criterion, usually L_2 cost function represented between patterns and outputs of the network, the particles converge to a position that represents the best found solution. Significant advantages of the PSO algorithm are its implementation simplicity, as well as the ability to reach a reasonably satisfactory solution even in the present of multiple local optima. Since its first implementation, many improvements and variations have been introduced [115]. In practice, it has been shown that some of the parameters of the PSO algorithm do in fact affect its performance and ability to converge [149], [20]. As a stochastic method, PSO's performance deteriorates with the increase of the dimensionality of the search space. In a situation where the swarm converges to a local minimum, there could be a possibility to escape with a momentum that is build into the algorithm through the inertial term. However, with time, the particle momentum will decrease until the swarm reaches a state of stagnation with no escape. Since the velocities are

Algorithm 3 PSO selection algorithm

initialize population (position and velocity of particles)e REPEAT
evaluate all particles
for all particle i do
IF current position of particle i, x_i , produces the best fitness in its history then
$P_i \leftarrow x_i$
if fitness of x_i is the best fitness in global then
$P_g \leftarrow x_i$
end if
end if
end for
update velocity and position of particles according to the equations 3.16 and 3.17 until termination criteria met

representing small values due to the decrease rate, one could allow for further exploration of a portion of the search space. Such a decision could produce removal of the nearest solution from the search space, regardless the amount of iterations that pass. This limitation of the original algorithm is overcome by providing varying velocities through velocity update equation for the best particle and guarantee convergence to a local minima [20].
Nevertheless, the problem remains that particles converge to a local minimum before they are able to reach a global one. Thus an improved multi-start PSO algorithm was developed to automatically restart the algorithm in the presence of stagnation [166]. By restarting, one understands the start of a new search with a new sequence of generated random numbers to avoid the same initial positions from the previous searches. In this instance the particles lose their memories. Once the independent search is accomplished, the obtained values are compared to the ones reached from the previous searches. Once the predefined number of restarts has been reached, the best values from all the possible global best from all the searches is selected.

The algorithm is known to be susceptible of falling into local optima even in the presence of a fast search speed. To accelerate the convergence rate, a new PSO algorithm is proposed in [172] employing dynamic Cauchy mutation of the best particle and using opposition based learning for each particle. Even in the presence of research to understand the effect of the parameters of the algorithm on its performance and convergence speed and ability, the limitations of the PSO technique and its trapping into local minima still exists.

Ant Colony Optimization The implementation of the backpropagation algorithm has been shown to be slow process which often reaches local minima [173]. In an attempt to overcome those disadvantages, heuristic algorithms like the genetic algorithm, PSO and others have been used. In this section we give a brief introduction to the Ant Colony optimization method, which have also been utilized in an attempt to train neural networks more efficiently [22]. The Ant Colony Optimization (ACO) is a meta-heuristic approach which emulates the ability of ant colonies to find the shortest path between the ant's nest and the food resource [47]. The idea of the algorithm described in fig. 3.4 is that ants initially wander in a random manner

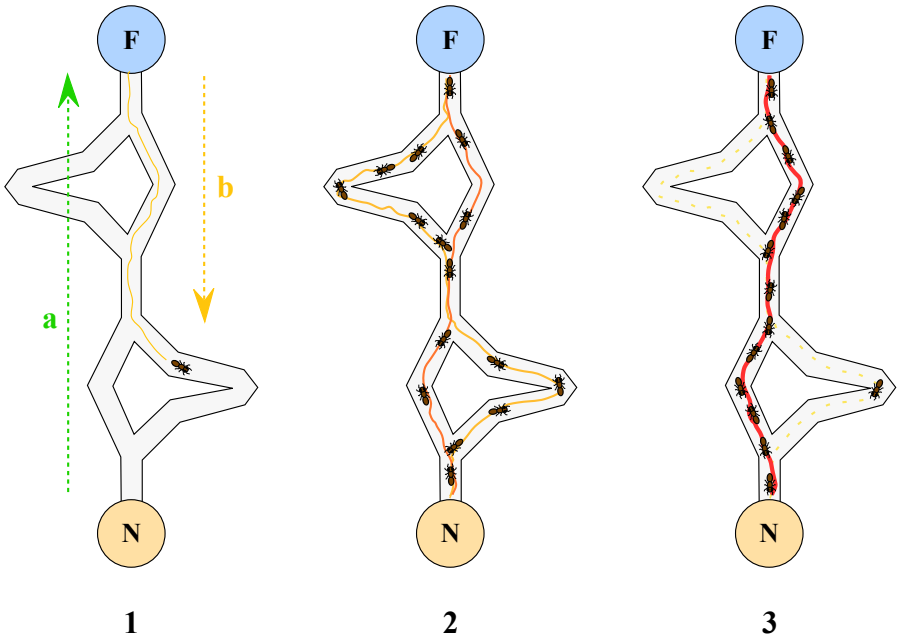


Fig. 3.4: Shortest path find by an ant colony. 1) the first ant find a food source (F), using some path (a), then it comes back to the nest (N), laying a pheromone trail. 2) the ants follow one of the 4 possible paths, but the reinforcement of the trail make the shortest path more appealing. 3) the ants follow the shortest path, the pheromone trail of the longest ones evaporates. Author: Johann Dreo under $CCBY - SA3.0$ license

until a food resource is discovered, on the way back to the nest the ants deposit pheromones, which evaporates with time (fig. 3.4 step 1). The ants follow the pheromone trail with the strongest concentration of the property (fig. 3.4 step 2). Over certain time period, as more and more ants follow a trail with the highest pheromone level, it becomes the shortest path to the food resource (fig. 3.4 step 3). Therefore, every ant will follow the shortest path, which will be with the highest pheromone level respectively. The reader should note that there are two main components in the ACO algorithm, which can be seen as a forward and backward modes:

- Pheromone update - the pheromone is an important part of the algorithm since it provides a dual purpose.Firstly, it leads the ants to adopt an appropriate path. Secondly, it serves as an information exchange tool among the ants. After the construction of paths is accomplished at each time step, the pheromone information is updated. The update is a two step process. Initially the pheromone is added in accordance to the path length. On the other hand to avoid rapid convergence and to allow the ants to search for new areas, a pheromone evaporation component is used in the update as well.

- Path construction - to construct possible solutions, one establishes a certain probability of path construction in accordance to pheromone information. The bigger the amount of pheromone, the higher the probability of a path being chosen.

The algorithm alternates between pheromone update and path construction until a stopping criteria is met. Generally, the two steps are iterated in order to solve some optimization problem. One constructs a possible solutions in a probabilistic manner with probability distribution over the multimodal space. Secondly, those candidate solutions are used in the probability distribution modification through an approach to bias the solution construction over time to regions toward high quality solutions.

The initial implementation of the optimization method has been in the application of discrete optimization problems. Recently, it has been used for continuous optimization problems such as neural network training [152], [114]. During the ACO training of a neural network, one needs to establish the optimal combination of synaptic weight values. To calculate the number of weights l in a neural network we denote the number of input, hidden and output neurons by n_i, n_h, and n_o respectively with additional values representing the bias inputs of the neurons [111].

$$l = n_h \left(n_i + 1\right) + n_o \left(n_h + 1\right) \tag{3.18}$$

The probabilistic solution construction, every ant selects only one discrete point for each synaptic weight. The memory of the ant preserves the previously visited points of the weights. The number of weights in the network l determines as well the dimensionality of the ant's memory according to Eq. 3.18. To choose a value for a synaptic weight (w_{ij}, which connects neuron i to neuron j) with a probability $1 - q$ where q is parameter of the decision rule ($0 \leq q \leq 1$), where p_{ijh} is the probability of selecting f_{ijh} for weight w_{ij}, d denotes the number of discrete points, ι_{ijh} is the existing pheromone trail assigned with f_{ijh} , then

$$P_{ijh} = \frac{\iota_{ijh}}{\sum_{k=1}^{d} \iota_{ijh}}. \tag{3.19}$$

Once the ant determines a value for each synaptic weight and bias, the pheromone update procedure is initialized. In this case only the best ant is able to retrace its values for each weight and therefore provides a pheromone, where PH^{best} is the "**the combination of discrete points selected by the best-so-far ant". The amount of pheromone to be left is $\Delta\iota_{ijh}^{best}$, which can be described with E^{best} being the network error (i.e.L_2) from the T^{best} combination.**

$$\Delta\iota_{ijh}^{best} = \frac{1}{E^{best}} \tag{3.20}$$

Therefore it follows that the pheromone updating is described as:

$$\iota_{ijh} \leftarrow \iota_{ijh} + \Delta\iota_{ijh}^{best}, \forall f_{ijh} \in T^{best}. \tag{3.21}$$

The representation of the pheromone evaporation is accomplished with a constant $p \in (0, 1)$ evaporation rate, following:

$$\iota_{ijh} \leftarrow (1 - p)\iota_{ijh}, \forall f_{ijh}. \tag{3.22}$$

Certain disadvantage of the ant algorithm is that the analysis of the training is difficult to establish. Furthermore, the algorithm changes the probability distribution by

every iteration and thus the time to convergence is uncertain (although convergence is accomplished).

Simulated Annealing First introduced by Kirkpatrick in 1983 (and independently by Černỳ (1985) [30]), the simulated annealing optimization algorithm draws on similarities between combinatorial optimization and statistical mechanics. Kirckpatrick's initial algorithm was successfully implemented for optimization in electronics design [90]. The concept of the algorithm are inspired by an analogy to the physical annealing process of solids. In condensed matter physics, annealing is the thermal process for obtaining low energy states of a solid in a heat bath. This process follows two steps [90]. Initial increase of the effective temperature of the heat bath to a maximum level, at which melting of the solid occurs. Secondly, a careful decrease of the effective temperature until the particles arrange themselves in the ground state of the solid. In the liquid phase, all the particles are arranged randomly. During the solid state the particles are arranged in a structured lattice, with minimal corresponding energy. When the maximum effective temperature is high enough and the cooling is produced in a sufficiently slow manner, a ground state of the solid is obtained. Otherwise from an optimization perspective the solid falls into equilibrium state rather than a real ground state. This ground state is perceived as a global minimum in terms of thermodynamic energy. From this perspective, the simulated annealing algorithm attempts to discover a global optimum of an objective function, which is perceived as the energy function of the solid. The cooling of the temperature is applied stochastically, which provides a way to the system to escape from local minima, with the thought that near the end of the process, the algorithm reaches neighborhood basin of the global minimum. Due to the inherent statistical nature of the algorithm, it is perceived as a way for the algorithm to hop over local minima easier than the gradient descent methods. The goal is to find a solution where the error/energy is minimum. The process begins with an random initialization guess. he simulated annealing algorithm explores the entire surface of possible solutions, moving both uphill and downhill, independent of the starting values, in direct contrast to the backpropagation algorithm. The uphill and downhill movements provide an escape tool from falling into local optima in the path to reach a global one. While the temperature is still high, often worse solutions are accepted to provide a way for the algorithm to escape local optimums if it has fell into such early in the algorithm execution. The probability of a move for a point is given by

$$Pr[accept] = e^{\frac{-\Delta E}{T}}, \tag{3.23}$$

where ΔE is the difference between the actual energy and the energy before the move, and T is the effective temperature of the system. Therefore a move is accepted if the generated random number $[0, 1] \ni R < Pr[accept]$. With decrease of the effective temperature fewer instances are accepted of solutions with higher temperature. The algorithm focuses on a certain area of downhill improvement in the search space, hoping it is close to the optimal solution (see algorithm 3.5. Simulated annealing is best suited for problems where search of possible solutions consists of multiple local optimas, saddle points or plateaus. Nevertheless, for the implementation of the simulated annealing optimization algorithm one needs to consider the suitable evaluation function, appropriate cooling schedule and initial state should be identified.

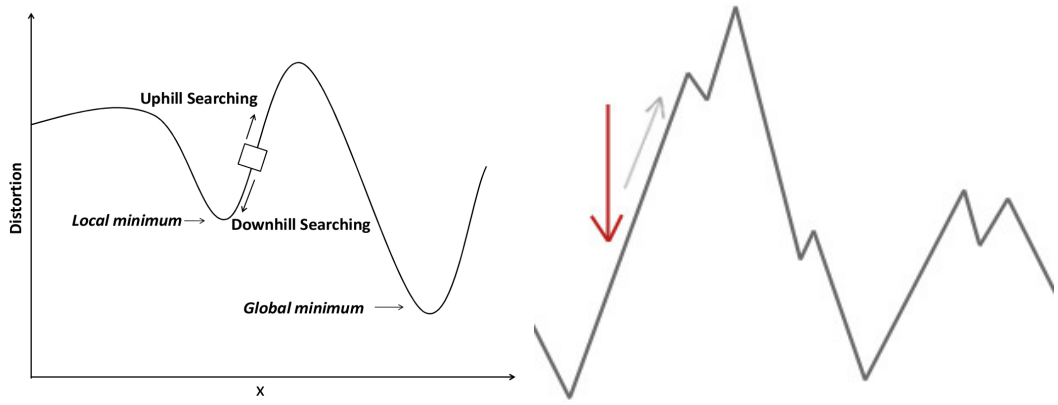


Fig. 3.5: Example for the hill climbing algorithm and the simple process of movement for the simulated annealing algorithm.

The implementation of a cooling schedule should consist of four elements: starting temperature, final temperature, temperature decrease and iterations at each temperature. The first component - staring temperature must be set in the implementation high enough so to allow the algorithm to move to almost most neighborhood states. Failure to achieve this means that the end solution will be similar to or the same to the starting one. If, on the other hand, the temperature value is set to be too high, one might transform the search (early stages) into a random search since the algorithm can move to any neighbor. The outcome will be that the search will be random until the cooling is low enough to provide the algorithm to act as a simulated annealing method. Various strategies have been proposed for the starting temperature. One possible could be to start from a very high temperature and to provide rapid cooling up to about 60% worst solutions are accepted, which effectively forms the actual starting temperature and from then the algorithm should implemented slower cooling strategy [134].

To set the temperature to decrease to zero could potentially lead to longer running time in case of the implementation of geometric cooling. In practice the algorithm could be stopped when the temperature approaches zero considering the fact that the probability of a worse solution to be accepted will be equal to the temperature being zero. Once the temperature reaches an inability to find better or worse solution, the algorithm is stopped, since the system is considered "frozen".

The step to decrease the temperature is needed so we can eventually reach a stopping criteria. This is of critical success for the algorithm. Although theoretically it is recommended to allow for enough iterations to pass until the system stabilizes at a certain temperature, it could also mean that the number of iterations at each temperature can increase exponentially to the problem size. Computationally, we can decide to provide the algorithm with the option for many iterations at few temperatures or few iteration at many temperatures. A third option is a hybrid approach between the two.

Algorithm 4 Simulated Annealing algorithm

Generate a random solution
Calculate its cost according to predefined cost function
Generate random neighboring solution
Calculate its energy
Compare the old and new solution
IF enew < eold - move to new solution
IF enew > eold - possible move to new solution
ENDIF
Repeat until stopping criteria is met
ENDWHILE end

The purpose of simulated annealing as a combinatorial optimization solution has been adapted also for continuous function optimization problems. Provided that a long enough cooling period is allowed for, the system will converge to a global optimum. The disadvantage is that to reach a global optima the algorithm needs higher computational time, as well as the required number of training data samples for some problem to reach optimum convergence may be more than the whole space of solutions. If not enough data is provided, overfitting might occur. Furthermore, a possible strategy can be the restarting of the cooling schedule through the use of the best found solution. The temperature of the algorithm can also influence the size of the solutions, starting with broad neighborhood and narrowing it with the iterations of the algorithm.

Quantum Simulated Annealing In 1998 the quantum annealing [86] method was proposed to solve combinatorial optimization problems. Similar to other optimization algorithms, it searches for the minimum of a cost function, achieving this by quantum fluctuations. Here, the cost function consists of an Ising model of statistical mechanics, where the energy function (the Hamiltonian) is chosen such that the lowest energy state represents the solution to the problem. A quantum mechanical fluctuation term is introduced in order to produce a quantum transition between states. The Hamiltonian of the Ising models can be represented as

$$H = -\frac{t}{T}\sum J_{ij}\sigma_i^z\sigma_z^i - \Gamma(t)\sum\sigma_i^x \tag{3.24}$$

where σ is the Pauli matrix. The first term on the right side is the Ising model and the second terms - the transverse-field term, causes quantum fluctuations between the classical states. The $\Gamma(t)$ is initialized with a very large number and is decreased toward zero to the end of the annealing process, following the time evolution (Schrödinger equation). To initialize the process, a sufficiently large coefficient for this quantum term is introduced in order to produce a uniform probability of existence of states in all states. Then the value of the coefficient is gradually decreased with the state of the system following the time-dependent Schrödinger equation. When the coefficient reaches zero, only the classical Hamiltonian remains. Then it is considered that the expected probability is the largest value for the right solution to the optimization problem at the end of the annealing process.

Although it has been used for combinatorial optimization problems in machine learning and natural language processing, it still remains to be proven that quantum annealing achieves exponential speedup over conventional optimization methods. The presence of superposition and tunneling suggest the possibility that some energy landscapes are more efficiently explored by the quantum annealing compared to the classical one. On the other hand quantum annealing needs exponentially long time to find an exact solution to difficult problems.

In 2011 D-wave systems, a Canadian company, introduced **D-Wave**, is a physical representation of quantum annealer with certain confirmation on the presence of quantum tunneling in the device [169]. D-wave represents a $2D$ array of quantum bits, developed by superconducting loops that are carrying electric currents. Since each qubits can point up, down, or it can be at the same time up and down, and the interaction of the qubits is achieved through the lowering of their energy by their directionality, to find the ground state of the annealing process, the states of these qubits are changed up-down slowly turning the interactions. Of great importance is that D-wave exploits a quantum tunneling effect [137]. To assess the capabilities of the machine, it was tested against a von Neuman architecture computer using simulated annealing and discovered that currently the quantum annealer (used in D-wave) does not produce a quantum speedup [34], with many researcher actively questioning whether quantum annealer could produce a speed-up over classical approaches.

3.2.3 Summary

Artificial neural networks utilizing various learning algorithms have proved very successful in distinctive fields (i.e. in predicting construction crew productivity [127]). The consideration of a neural network training as an optimization problem to minimize a cost function has introduced the implementation of many heuristic algorithms in the ANN field. Although they are known to escape local minima problems, in other instances the algorithm falls into a a saddle point, leading to network paralysis.

Additional problems in function fitting (e.g. curve fitting of polynomials) exist in the type of training data used. The inability to set the exact amount of training necessary for neural networks, often leads to overlearning - meaning the network learns the the noise in the measured data as well, unable to discern the underlying pattern structure. This process is also known as **overfitting**. If the set of training points represents a smooth function with some levels of noise to the outputs, then utilizing just a few data points will be inefficient for the successful learning of the network. On the other hand, too many data points will lead to the remembering of the network of the underlying input noise, with poor results. Therefore, the training sample, as discussed in the previous chapter, should be carefully considered in order to procure the best predictions. Yet, the generalization capabilities of a network improve with the increase of a training set. When the task is to fit a polynomial of a certain degree to a set of points, the function can be overfitted (learns the training points perfectly but interpolates unknown points incorrectly) - see fig. 3.6. The function, therefore, oscillates in order to fit all the training points with no error, but

it is rather smooth in the unknown function. The introduction of more points does not reduce the error as long as the concept is represented by polynomials of a certain degree. If we need to keep more training points, we need to reduce the search space size through the limitation of the degree of the polynomial approximations. Training algorithms such as backpropagation provide variability in the solutions in the part of local minima in the cost function. Although simulated annealing improves the problem of local minima, it is not a definitive answer to the problem.

To alleviate such algorithm shortcomings, in the next section we propose a novel post-learning strategy and its practical implementation. Several numerical experiments concerning the problem of approximating a known function are performed in order in order to validate this novel approach.

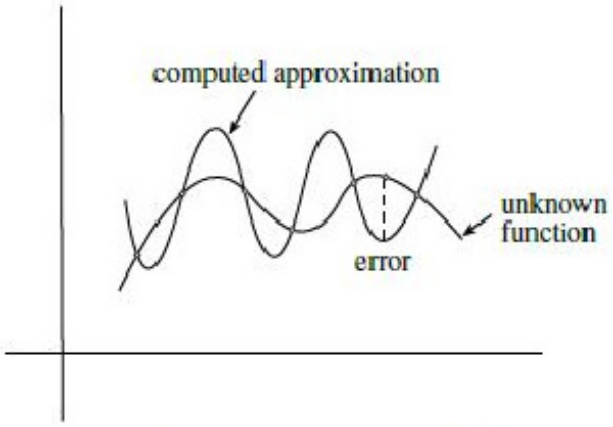


Fig. 3.6: overfitting a pol. approximation

3.3 Post-learning Strategy

The field of artificial intelligence has been a place where arguments have taken place as to whether our world is relying on deterministic casual events, where one uses probabilities to calculate missing knowledge. The dissenting opinion states that our world is essentially indeterministic with we use probabilities to measure the random events. As described in the previous chapter, optimization problems (training algorithms) are broadly classified as deterministic and stochastic. In this classical approach all weights transformation is carried through during the optimization process. A step further, one can say that the training methods from the previous chapter are constituted in terms of classical (deterministic) physics. But if one sees the world as essentially indeterministic, an interesting possibility to exploit is quantum mechanical effects in a ANNs. The idea has been suggested for the first time (to the best of the limited information available) in [17] where the authors propose that biological expressions of quantum phenomena are related to the activation points in the brain activity. All of this is described as a nerve impulse firing achieved

by the movement of a quantum particle in the vicinity of a potential energetic barrier. The availability of an action potential is proposed to play a significant role in the information processing functions of the brain since there effects such as tunneling arises. On the other hand Penrose [131] suggests the concept of the existence of micro-tubules, capable of maintaining a macroscopic coherent superposition.

The proposed post-learning technique [1]is particularly inspired by the work described in [17]. The method mimics quantum effects and thus is capable to provide an improvement to the set of weights in a network in the *post-learning* step. The main goal is to introduce a method for supplement the level of accuracy of the network even in the of training failure - whether for missing data, insufficient architecture or the optimization falling into a saddle point or being stuck in local minima. This new strategy is achieved at a relatively low computational burden.

In this section the post-learning strategy is implemented on a multilayer feedforward network, consisting of several neurons, each of them fully connected to every neuron in adjacent forward layers. The post-learning strategy is not limited to this particular implementations and can be used in various layouts. For the current network, the activation function for every neuron $(\sigma_i(x))$ is the function $tanh(x)$ with output range $[-1, 1]$.

3.3.1 Description of the Method

The possibility of biological neurons to exploit quantum effects provides an encouragement of introducing randomness in a neural network implementation with the goal of achieving certain computational advantages [17]. The proposal for the implementation of quantum mechanical laws inside every neuron correlates to the condition of the numerical simulation of the time-dependent Schrödinger's equation (or any other equivalent formalism like Feynman, Wigner). This is necessary to quantitatively determine the eventual tunnelling effects. Therefore, the network needs to numerically simulate the following time-dependent partial differential equation [64] for every neuron:

$$i\hbar\frac{\partial\Phi}{\partial t}(\mathbf{r},t) = \left(-\frac{\hbar^2\nabla^2}{2m} + V(\mathbf{r})\right)\Phi(\mathbf{r},t), \tag{3.25}$$

where i is the imaginary unit, $\Phi(\mathbf{r},t)$ is the wave function defined over space and time, $\hbar$ is the reduced Planck's constant, r is the position of the particle, t is the time, m is the mass of the particle, ∇^2 is the Laplacian operator, and $V(\mathbf{r},t)$ is the potential energy acting on the particle. In the case of relatively small neural networks, consisting of just one hidden layer and limited amount of neurons, this simulation will be computationally affordable. In a real-life application of neural networks such as the ones used for acoustic analysis or speech recognition, where hundreds or thousands of neurons are used, this represents a daunting and computationally prohibitive task. Therefore, a computationally convenient technique is proposed that is capable of miming the existence of randomness, fundamental in a quantum

[2]The words quantum, random and noise are used interchangably, having in mind the same meaning. The same applies for the terms classical and deterministic.

system, without the high computational cost associated with available and accurate quantum simulations.

The method is modeled by analogy of a biological neuron as a semiconductor heterostructure that consists of an energetic barrier (e.g. AlGaAs) that is situated between two energetically lower areas (e.g. GaAs). [17] gives as the possibility to consider the activation function of an artificial neuron as one or more particles interacting with the barrier (see Fig. 3.7) once they enter the heterostructure. The probability of finding the particle in some point of the device at time t, is described by the modulus of the wave-function $|\Phi(\mathbf{r}, t)|^2$. In such a way, a randomness is introduced to the process (Born rule). If the probability of a tunnelling is lower than the probability of back scattering, the activation function is considered to be inhibitory.

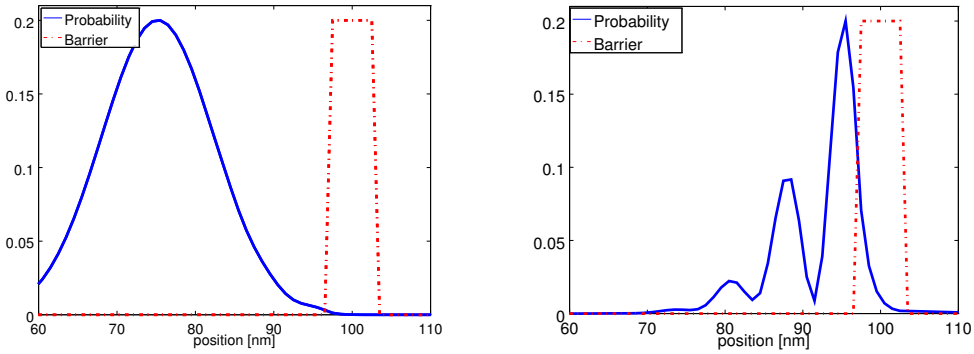


Fig. 3.7: Left plot: a Gaussian wave-packet, (blue) continuous line, is travelling against an energetic potential barrier, (red) dashed line. Right plot: after a certain time, the wave-packet is interacting with the barrier. Part of the packet is scattering back while the rest is tunnelling.

Certain similarities in this method exists between the proposed method and the main structure of an Action Potential. The latter one is described as a distinct voltage-gated ion channels in a cell's membrane. When the potential increases to a defined threshold value, the membrane potential of the cell opens. This electric potential difference (voltage), maintained by the membrane potential, once triggered, is activated.

In practical implementation, this can be achieved by adding to the network a new function called ***addHiddenNoise*** which, adds noise to the already computed weights after the training process. This is done only for the neurons in the hidden layers of the network

$$hiddenLayer.neurons[i].weights[j] += randomDouble() * weightNoise \quad (3.26)$$

which mathematically corresponds to the expression

$$f(w_1x_1 + w_2x_2 + w_3x_3... + w_nx_n) \quad (3.27)$$

RandomDouble returns a random double (generated by a Mersenne Twister) and it is implemented with the idea to bound the noise to a percentage of the given range, depending on the actual problem. The function ***backupState*** is used to copy the weights for restoration if the network with the added Noise produces worse mean squared error than the one without noise.

The following section describe the implementation of the proposed method to three numerical experiments in order to validate the approach.

3.4 Numerical Validation

In this section a numerical validation of the suggested post-learning strategy is presented. The implementation of the neural network works to fit two known functions - a polynomial of second degree $(f(x) = x^2)$ and the square root of a polynomial $(f(x) = \sqrt{x})$, with three training data points provided. In both experiments the network architecture is identical - with one input node, 4 hidden neurons and one output neuron. Furthermore, additional constraints to the weights search space have been utilized. In the first scenario, the search space is limited in the range $-12(min)$ - $+12(max)$. The weight space for the second function are distributed between $-1(min)$ and $+1(max)$. In this work, the training algorithm is based on the simulated annealing method for both functions.

During the numerical experiments, we have deliberately hindered the network by stopping the algorithm at an arbitrary local minimum. The task is accomplished by means of the temperature rate of the algorithm, which we decrease in a non-optimal fashion. Through the strategy, the goal is to clearly show that our technique is able to provide a further solution to improve the training even after the optimization process is completed. One of the advantages in this situation is the importance of finding the best set of weights for an ANN among the increase of available training data and growing complexity.

The training algorithm utilizes three equidistant data points, in the range $[0, 1]$, although we have excluded the extrema. In the first case (see Fig. 3.9, upper left side), the algorithm relies on $(0.1, 0.01)$, $(0.5, 0.25)$ and $(0.9, 0.81)$ training data points. The example for the square root of polynomial exploits for the training the following data points:$(0.15, 0.38)$, $(0.6, 0.77)$, $(0.85, 0.92)$.

In order to better understand the influence of noise in terms of the network's output and error decrease, 5 different scenarios are investigated. The initial run is accomplished without any noise addition and as such serves as a benchmarking tests for the quantum part of the network compared to the classical one (see Fig. 3.9 and Fig. 3.11 upper left plots). The error from the classical part of the network is depicted as a (blue) x, while the error from the quantum side of the network is shown as a (red) square. The upper left plot illustrates the network running with no noise for validation purposes. Upper right plot illustrates the network's error in scenario with 0.5% applied. The middle left and right plots constitute network error in simulations with 1% and 2% noise. The left lowest plot represents error with 4% noise respectively.

The next three subsections attempt to explain how the level of noise is affecting the two functions as well as the sphere function.

3.4.1 Polynomial of second degree

The first numerical experiment depicts the fitting of a polynomial of second degree. An initial test is performed without addition of noise to ascertain the performance of the network and as such to give a benchmarking assessment for the precise functioning of the network.

To examine the influence of the noise on the network's output, we initialized a situation where we include negligible noise to the system (0.5%).In Fig. 3.9 the (red) star symbolizes the desired network's output, the (blue) x indicates the output from the classical mode, and the quantum mode is denoted by a square. The upper left side plot exhibits a validation test for the network, running in both classical and quantum mode, with no noise applied. The upper right plot represents the output when 0.5% noise is applied. The middle left and right plot, display the network's output when 1% and 2% noise is assigned respectively. The final plots consists of the output when the network is executed with 4% noise.

Consequently, the effect on the quantum behavior are negligible (see Fig. 3.9 upper right plot), influencing the first few outputs. One can corroborate those results comparing them to the error of the network (as seen in Fig. 3.10 upper right plot). Conceivably, the quantum part of the network contributes to slight improvement of the first few data points. In order to accomplish this, the algorithm at every point compares the error from the classical to the quantum part, chooses the better option, while discarding the other. Should a better quantum solution is calculated, the network accepts it and continues forward.

The reader might observe that indicative to the amount of noise applied is the dispersement of both the quantum output and error throughout the plot in settings where one successively increases the amplitude of the noise. As such, a moderate increase of noise to 1% leads to a situation where the network output in the middle of the solutions decreases its accuracy, while still retaining better results in the upper and lower bound of the curve.

The application of 2% noise advances the deterioration of the network precision. The network is able to provide only few reliable outputs close to the fitting curve. Therefore, the increase of noise contributes to increase of output outliers in the outcome (shown in Fig. 3.9 middle right plot).

With the addition of more noise to the system, the quantum network outperforms the classical one in few cases. The increase from 1% to to 2% and 4% noise, leads to an overall gradual decrease in error (Fig. 3.10, middle right and lower left plots).

The current numerical experiment shows that the performance of the network is stable when 0.5% and 1% noise is added. We observe the quantum part's ability to

produce smaller error in the beginning of the validation, with the option of almost instantaneous decrease of the error.

Nevertheless, even with higher levels of noise, the network still produces small improvements, but at far lesser scale and depth.

3.4.2 Square root of a Polynomial

We have performed a second test in fitting a square root of a polynomial, taking into account the assumption of the capacity of the novel technique to potentiality operate in a distinctive mode when computing different functions. In the first scenario, where the noise is established at 0.5%, the results between the quantum part approximate the classical ones (Fig. 3.11 upper right plot). The reduction of the error appears to be stepwise, close to the example from the first function. The current setup provides only few improved outcomes, irrespective of the steep error descent from point to point.

Expanding the analysis by adding 1% noise, we see the outcome close to the optimal (Fig. 3.11 middle left plot). On fig. 3.11 the (red) star symbolizes the desired network's output, the (blue) x indicates the output from the classical mode, and the quantum mode is denoted by a square. The upper left and right plots represent network's output when 0% and 0.5% noise is executed. The middle left and right plots exhibit the output affected from 1% and 2% noise respectively. The lowest left plot represent results from executing the network with 4% noise.

Similar to the case from the 0.5% noise, the error reduction continues in a stepwise manner. While one might ascertain that in accordance to the previous numerical validation test, the increase of noise should contribute to the decrease in network performance, the current experiment illustrates the opposite conclusion.

The efficiency of the quantum network is in point of fact increased in a situation where 2% noise is added. The network produces as much as twice as many quantum outcomes, as when imposing 0.5% or 1% noise. Moreover, we observe the presence of lower errors (see Fig. 3.10 middle right plot). There is significant divergence from the initial experiment after the early steep error reduction from the quantum part. Here, the clustering of the quantum error points is found near the classical ones.

In the last example, at 4% noise, the error decreases consecutively. In this scenario (fig. 3.11 lower left plot), the network provides as much as twice as many quantum outcomes, as when only 0.5% is applied. Throughout the various noise scenarios, the outputs from the quantum network are analogous to the classical one.

One might interpret the divergent network performance for the two numerical experiment in terms of variation of the weight space range for the two functions.

Illustration of the divergence of error reduction for the two known function is available from Fig. 3.8. The left plot confirms the perception of increased network

performance on noise levels at 0.5% or 1%. The behavior of the network is opposite for the second numerical experiment. Running the network with gradual increase in the noise actually contributes to the tapering off of the error.

3.4.3 Sphere Function

To establish how advantageous the post-learning algorithm will be when applied to different functions, we have applied it to the Sphere model function $f(x) = \sum_{i=1}^{D} x_i^2$. It is symmetric and is one of the most widely used benchmarking functions. For the current work, we have applied 2% and 4% noise to test the algorithm.

In Fig. 3.13 one can the (blue) circle symbolizes the desired network's output, the (blue) $*$ indicates the output from the classical mode, and the quantum mode is denoted by a $+$ sign in black. The first plot exhibits the validation test of the network, where no noise is applied, while running in both quantum and classical mode. The middle plot represents the output when 2% noise is assigned and the last plot shows the output with 4% noise(Fig. 3.13. As seen in the previous examples, the more noise applied, the more deterioration of the system one observes. Despite this, there are certain points, where the network, even with the sphere function provides a quality output. While the network is able to produce from the quantum part only few better outcomes, it is still able to minimize the error substantially (see Fig. 3.14). The error from the classical part of the network in fig. 3.14 is depicted as a (blue) x, while the error from the quantum side of the network is shown as a (red) square. The upper plot represents the error with 2% noise. The left lowest plot represents error with 4% noise. In the case where 4% noise is applied the first outcome is better for the quantum network, as well as close to the last few errors.

3.5 Summary and Contribution of the Chapter

In this chapter, we introduced several optimization strategies frequently utilized for the training of neural networks. Some of the algorithms shortcomings involve reaching saddle points, local or global minima with inability to escape. Furthermore, we proposed a novel **post-learning strategy** that is implemented as a auxiliary reinforcement to the classical learning process of neural networks. The main purpose of this novel technique is to provide a method that is computationally reasonable in the scenario of a network trying to circumvent a local minimum during the training process. In order to achieve it, we suggested an approach based on the generation of random numbers at the core of artificial neurons, which attempts to mimic the presence of quantum randomness. By performing several numerical experiments, we validated the method against the problem of fitting a known function, given a certain number of training points, and we have shown how our technique provides certain improvements in the system, without relevant additional computational costs. In the next chapter we introduce a sensitivity analysis to understand the necessary amount of noise which has to be introduced in a network in order to achieve real improvements.

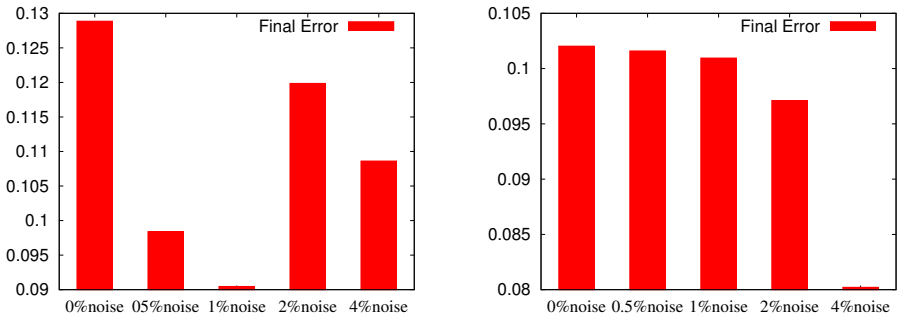


Fig. 3.8: Final error after the network calculates for 32 output points. The left side plot illustrates the error for the function of polynomial of second degree. The right side plot depicts the error for the function of square root of a polynomial. The initial error for the left plot for all levels of noise is 0.12886. The initial error for the right plot for every noise level is 0.10203.

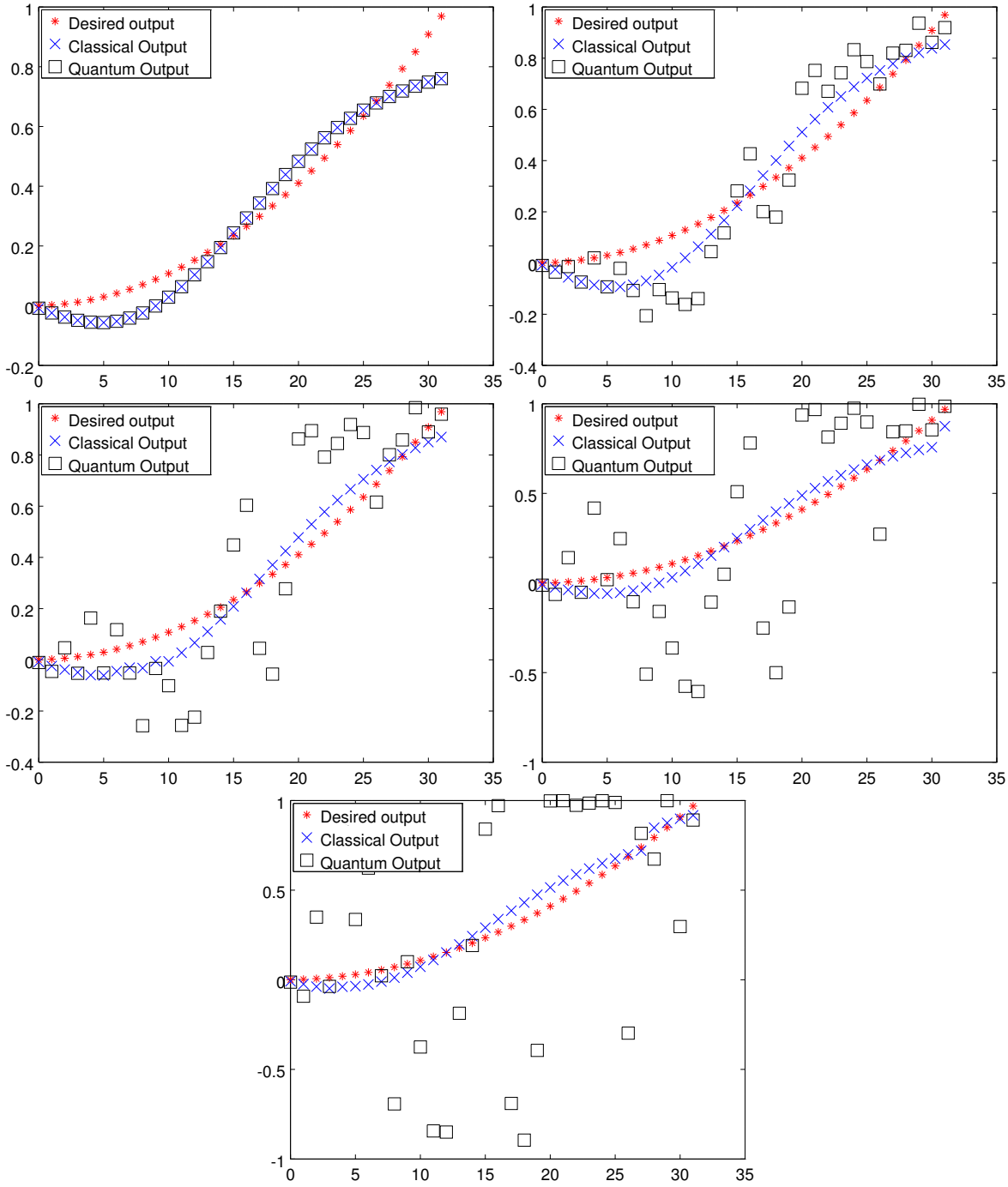


Fig. 3.9: The plots feature the output of our neural network for the function of polynomial of second degree.

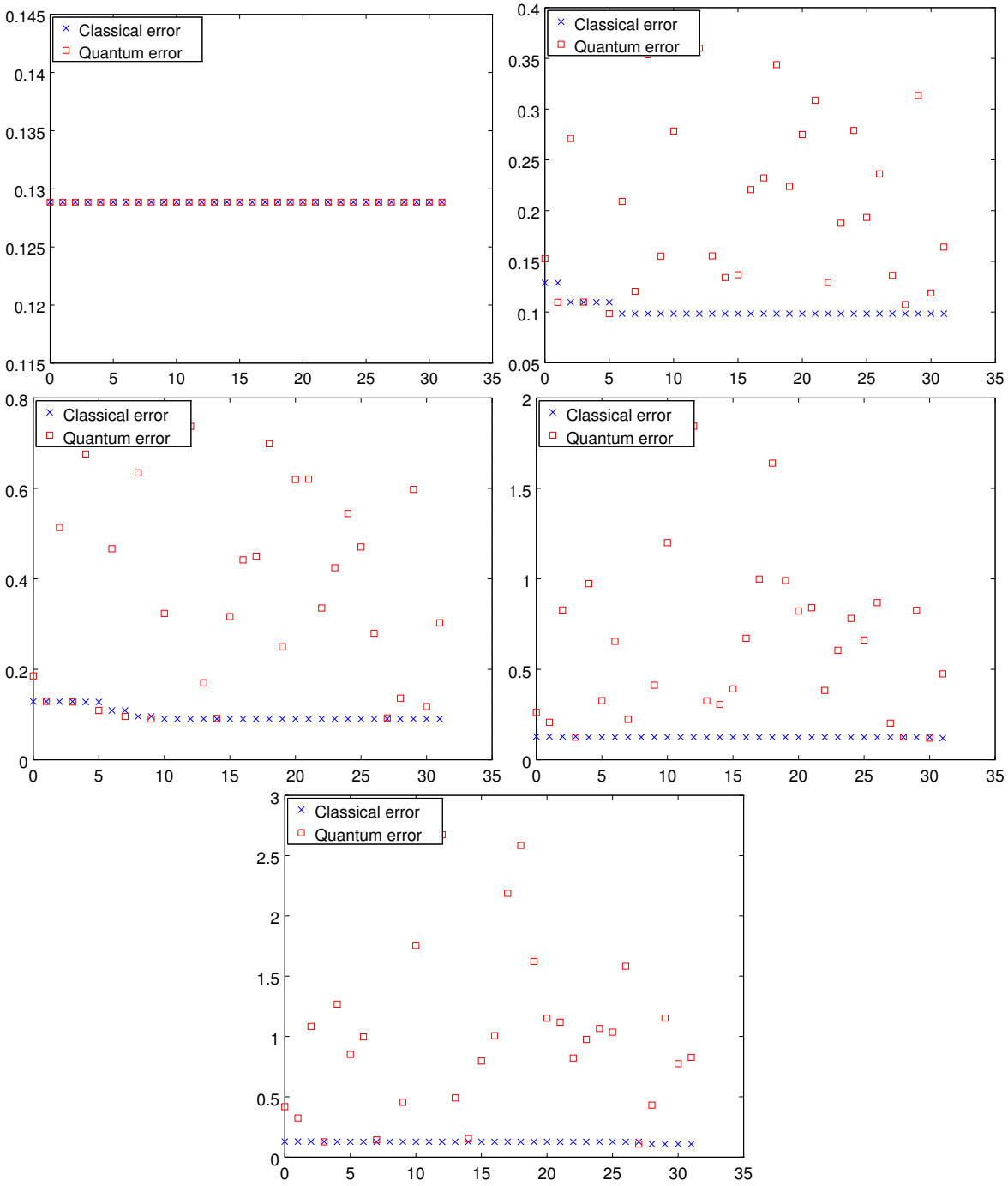


Fig. 3.10: Error reduction from the neural network in the case of polynomial of second degree.

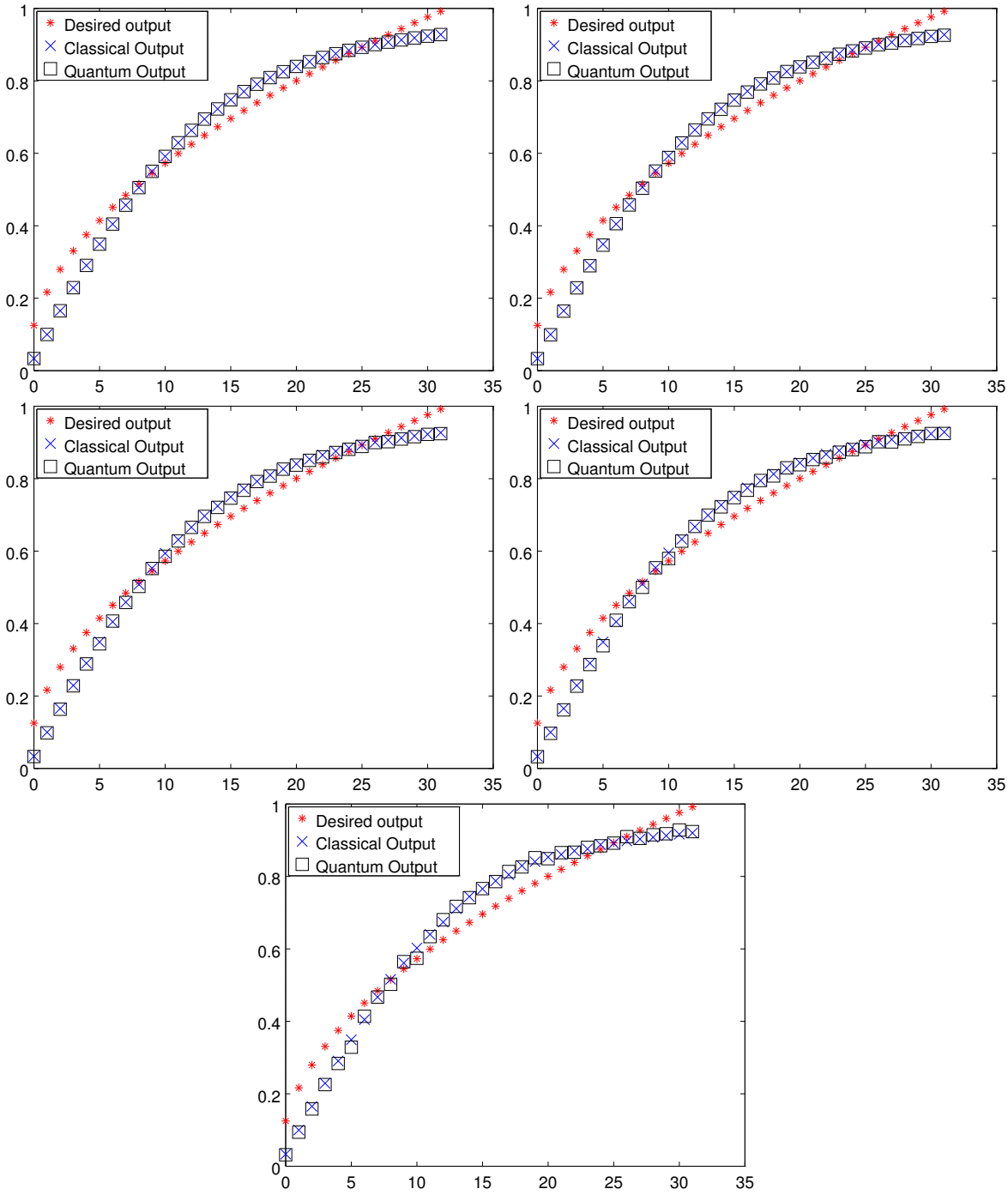


Fig. 3.11: Neural network outputs in the case of function of a square root of a polynomial.

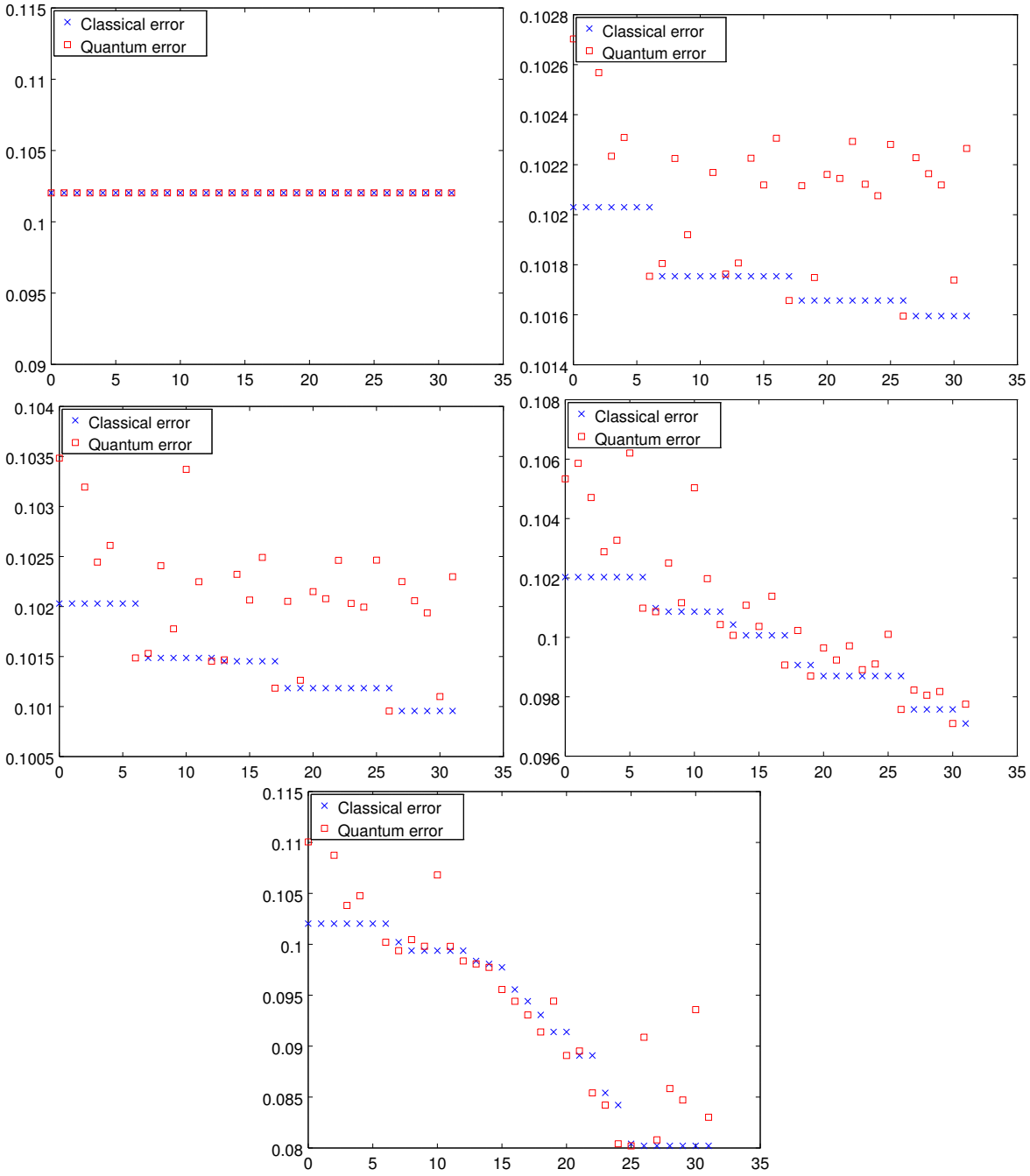


Fig. 3.12: Error decline from the neural network in the simulation of square root of a polynomial.

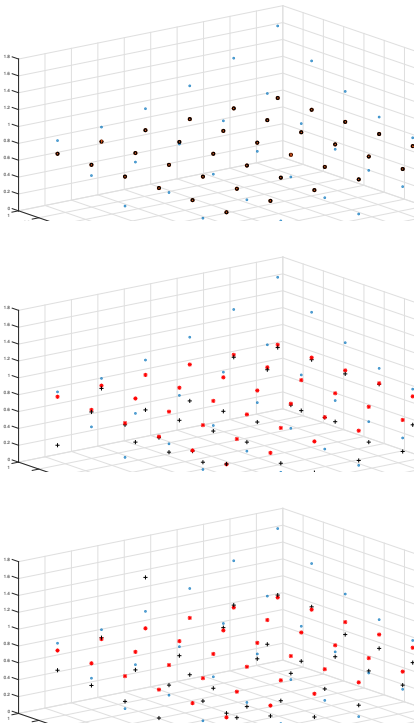


Fig. 3.13: The plots feature the output of our neural network for the sphere with no noise, 2% and 4% noise respectively.

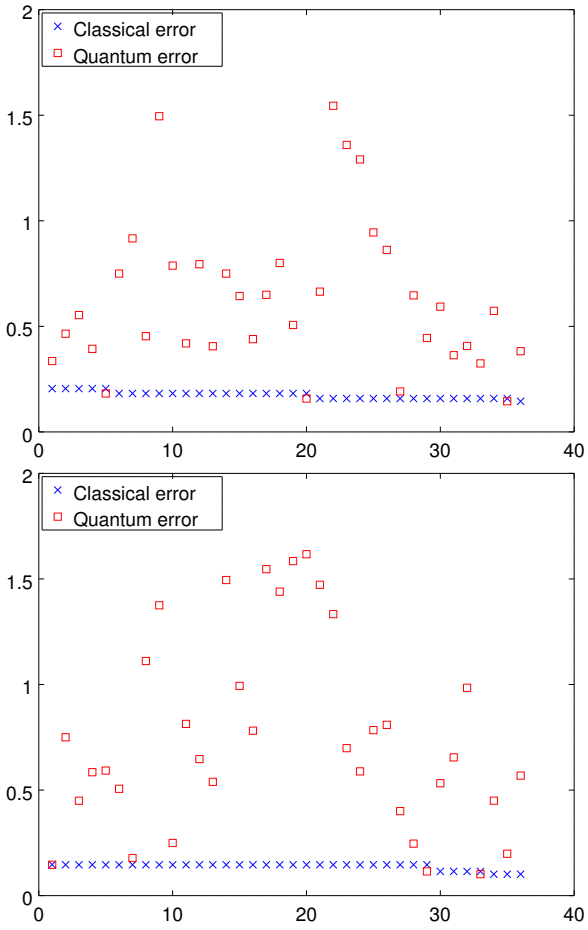


Fig. 3.14: Error reduction from the neural network in the case of the sphere function.

Sensitivity Analysis in Neural Networks

4

> *Users do not care about what is inside the box, as long as the box does what they need done.*
>
> — **Jef Raskin**
> about Human Computer Interfaces

In this chapter we introduce a sensitivity analysis tool to establish the level of noise in the network. Other research have been carried out on the "random or unpredictable fluctuations and disturbances" [54] and the possible benefits noise in neural systems can introduce [51]. For instance in engineering, the noise is identified as a detrimental to the system and the quality of the output. In biological neural systems, on the other hand, noise is naturally existing, often providing certain benefits to information processing [54, 51, 43, 158]. Currently, we believe no definitive explanation has been reached about how naturally occurring fluctuations in neural systems could be exploited, why they do in fact occur and what could be the potential advantages. While adding certain amounts of noise could provide benefits from the network oscillations, in other cases this oscillation could be detrimental [8].

Currently, there are certain experiments that add noise to the weights during the training process of a recurrent neural network [83] to improve the optimization convergence even in the absence of an input. To achieve this one needs to provide noise in order to generate opening and closing of ion channels to reach a good generalization [54]. One can apply noise either in a noncumulative and cumulative (with additive and multiplicative) manner in every time step or for every string. One may perceive the addition of noise to the synaptic weights as a beneficial in situations where the optimization algorithm is stuck in local minima and therefore through the addition of noise, we can provide a method to escape from it. Exploring the fault tolerance of a network, the possibility of learning acceleration and the effects on the error function where noise is inserted in the weights during raining of a multilayer perceptron network has already been examined in [123].

One may choose an alternative approach to the optimization problem with the algorithm randomly probing the space of weights with augmentation of all the weights though the addition of certain level of noise. Therefore we execute the network, calculate the error (with the selected weights) the perform additional search for a second batch of weights. The next step involved the comparison of the two option and the selection of the one which produces smaller error. Similar approach has been utilized in [146], establishing the network performance in the presence of noise in the training of deep neural network applied to a speech recognition system. There the noise is present only in the input layer (in the signal) and could be utilized either through representing the noise in the system through model adaptation of environmental information or by removing the noise in order to reduce variability.

Similarly, one could examine the presence of noise in biological system and in the hardware representation of neural network. In the latter case, it is reasonable to consider that the enclosement of many electrical devices contributes to electrical fluctuations that influence the performance of the network and the interaction of the neurons. Such instance is observed in experiments of the neural network developed with Memristors by Hewlett-Packard and IBM [132]. As a result of the memristor devices created in a limited space for an ANN, one detects increased presence of electrical current activity as well as random fluctuations that could affect the performance of the system in various ways.

In this chapter we present a sensitivity analysis methodology of a neural network performance in the presence of noise introduced in all the weights of the network. The goal of those sensitivity analysis techniques is to provide us with an accurate mathematical approach to determine the noise levels in a network. Our aim is with those tools to establish whether the noise can be utilized for improving network performance and when it is detrimental and acts as a source of fluctuations. Furthermore, with the sensitivity analysis tools we could achieve a better understanding of the noise variation on the network's behavior and the sensitivity of the output in relation to the noise fluctuation. The utilized sensitivity coefficients indicate what is the possible variance of the outputs of the network due to fluctuations in the weight parameters of the system. To the best of our knowledge, no study has been accomplished to understand the level of influence of the noise in the network and what type of consequences such noise present for the network's performance. The functionality of a network could be considerably altered with the modification of noise in the weights and therefore there would be considerable sensitivity to the system. Vice versa, with negligible sensitivity, the performance of the network should not be substantially altered.

Those sensitivity tools provide a quantitative representation of the fact that moderate amount of noise do in fact improve the network's performance. Such a conclusion, while counterintuitive, is clearly supported by the data. It can be shown with the selection of several indicators the exploration of the sensitivity of a neural network over the introduction of noise in the weight space. For this sensitivity analysis in this chapter, we have implemented three measurements to establish the scope of the solutions: the Euclidean distance (L_2), the maximum norm (L_∞) and the cosine-similarity (L_{cos}). In a situation of two weight vectors, selected from the weight-space, the cosine similarity provides information about their orientation, the Euclidean distance produces the possible directions and magnitudes of the selected weights. Finally the L_∞ indicator presents the parameter of their maximum departure. The reader should note that every instance from the performed numerical analysis is initiated with the same starting conditions to provide duplicability and comparability of the obtained results.

With this sensitivity analysis we pursue a better understanding of a network's behavior with the noise introduction in the weight space. Furthermore, we try to establish what is the sensitivity of the output when perturbation of the weights is introduced. Important consideration is that the perturbation is seen as a noise fluctuation and not always the addition of noise will provide the weight solution. Moreover, in certain settings the level of noise provides a variance in the whole weight space composition.

4.1 Sensitivity Analysis Tools

For the current sensitivity analysis we have developed a fully connected feedforward neural network (see Fig. 4.1). The input layer contains one node, the hidden has four neurons and the output layer is comprised of one neuron as well. The selected activation function is $tanh(x)$ with a range $[-1, 1]$. For our particular case, we have decided to use the simulated annealing algorithm as an optimization strategy for the weights. This particular design decision is in no way limiting to this particular choice. Sensitivity studies have already been performed in relation to

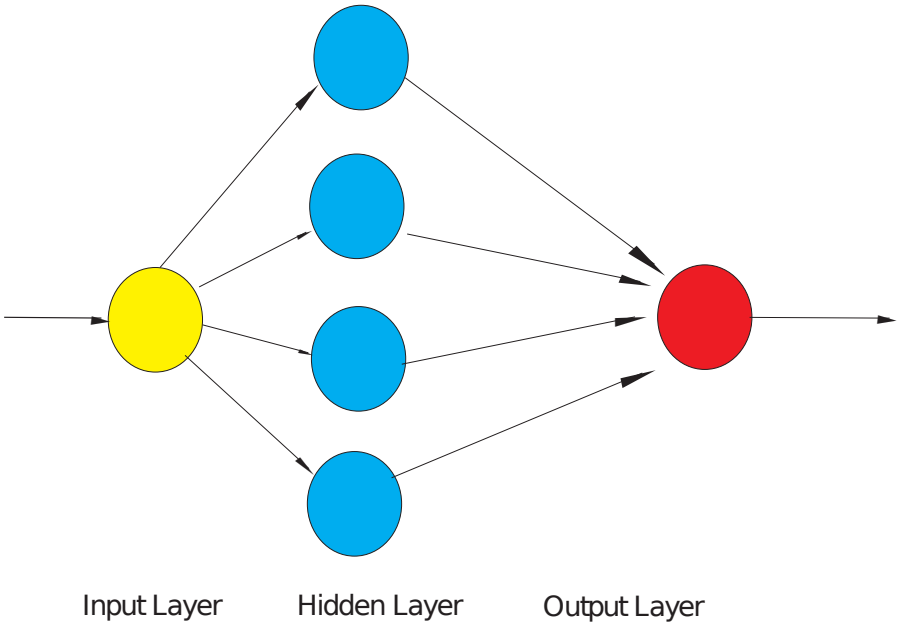


Fig. 4.1: Current design of the neural network implemented in this paper. Neurons are represented as nodes, the connections between the neurons are illustrated as directed edges.

neural networks, although the focus has been only on general feedforward networks, while imposing various restrictive expectations. In one instance the sensitivity analysis is characterized by a partial derivative of the network's output to the input of the network [74]. In other work the focus is on variance calculation from the network's output error under some stochastic assumptions [180, 35]. Various implementations have applied sensitivity analysis for input perturbation for a single neuron taken from a multi-layer feedforward network, where the analysis is on the error from the output layer, which sequentially computes the sensitivity neuron by neuron from the first to the last layer [180].

In this chapter, we have implemented a different sensitivity analysis approach to investigate the introduction of noise in the weights. The next paragraph provides a brief description of the indicators. Here, we presume that the underlying mathematical model is achieved in terms of a function model [151]. Provided a n-dimensional vector of input parameters $x = (x_1, x_2, \ldots, x_n)$, which are chosen in a space $U_n = [a_1; b_1] \times [a_2; b_2] \times \cdots \times [a_n; b_n]$ (a_i, b_i in this case depicted as real numbers and $a_i < b_i$ for $i = 1, \ldots, n$) with a joint probability density function

$p(x) = p(x_1, x_2, \ldots, x_n)$, allowing the model to provide an m-dimensional output vector $u = (u_1, u_2, \ldots, u_m)$, where

$$u = f(x). \tag{4.1}$$

We consider the output vector u to be a random vector, inasmuch as being depicted in terms of the random input vector x. Whence models are defined generally by multiple input parameters, in this particular case, we are examining only the noise level parameter. Hence, the space of input parameters U_n decreases to a a $1D$ space $[a; b]$ with the (real) values a and b, defined in the following section (with $a < b$). Although the work defines an n-dimensional input vector and an m-dimensional output vector, we are only assessing the method through the assignment of only one input and one output neurons in the network. One should consider that this is not an inherent restraint of the model. In this particular sensitivity study, the network is described as dependent on the weights, and therefore we consider it as an assembly of functions with $f = f(x)$.

The introduction of three different indicators presents us with an exhaustive interpretation of the dependence of f on the weight perturbation. In this case, each indicator provides a divergent perspective regarding the problem and therefore assists with the analysis of the results. Particularly, we use the concepts of cosine similarity, euclidean norm (also known as the $2 - norm$), as well as $\infty - norm$ or supermum norm (e.g. maximum norm). The analysis and definition of these indicators rests in the exploration of two experimental cases, in which we have applied noise to all available weights in the network.

In this work, the perturbation of the weights is denoted by u, while v depicts the weight configuration from the weight space. As such, those indicators express the configuration of certain weights in the network's output. Furthermore, we can represent u and v as two weight vectors with N_x components.

We introduce the first indicator, based on the cosine similarity, as

$$l_{\cos}(w'') = l_{\cos}[u(w''), v] = \frac{u \cdot v}{|u||v|}, \tag{4.2}$$

with the operation $\cdot$ representing the scalar product and $|u|$ describing the modulus of the vector u. To be exact, the indicator can't be a metric one, since the non-negativity axiom is violated. Particularly, L_{cos} can represent a negative value for vectors, that are positioned in the opposite directions in the N_x - dimensional space of output vectors. Yet, it is a good indicator of similarity between two vectors. In point of fact, in situation where two vectors point in the same direction and also have equivalent moduli, the value of L_{cos} is close to 1. In cases when they are orthogonal then $L_{cos} = 0$, while $L_{cos} = -1$ represents their opposite position. In situation when L_{cos} is close to 1, one may conclude the perturbation of the weights is not too sensitive (provided a certain range).

The second indicator, utilized in this work, is based on the Euclidean norm. One can describe it in terms of

$$I_2(w'') = I_2[u(w''), v] = |u(w''), v|_2 = \sqrt{\sum_{i=1}^{N_x} (u_i - v_i)^2}, \tag{4.3}$$

with u_i and v_i being the i_{th} component of the vectors u and v respectively. The current indicator is in fact a metric one and therefore can be construed as the distance between the initial weights (u) and the perturbed ones (v). This provides a concise indication about the quality of the computed solution. Therefore, The lower the value of L_2, the better the quality of the solution.

The final metric indicator utilizes the $\infty - norm$ where

$$I_\infty(w'') = I_\infty[u(w''), v] = |u(w''), v|_\infty = max|u_i - v_i| \tag{4.4}$$

In particular, the value $\infty - norm$ is exploited as an expression of the maximum distance the perturbation of weights can have from the weight solution in a point of the weight space. Although we have access to the analytical derivatives of the indicators, the current work focuses only on their numerical derivatives. This is achieved for the sake of generality.

4.2 Numerical Experiments

The current sensitivity analysis implements two numerical experiments - for convex and concave non decreasing functions with the goal of avoidance of biased results. Respectively, the two functions presented are polynomial of second degree ($f(x) = x^2$) and the square root of a polynomial. For both cases the implemented neural network consists of one input node, 4 hidden neurons and one output neuron. Again (as in the previous chapter) for the experiments, we provide only three training data points. There is recognition that in real-life applications many more points are used during the training process. The ultimate goal through this, in the current work, is to depict that even when limited training data is provided for general function approximation [21], the performance of the system can indeed be preserved regulated by the different levels of noise in the system. Furthermore we show how the SA indicators can be employed in such cases.

Moreover, we present an additional constraints to the weight space for both functions. In the first case, the weight space is confined in the range $[-12, 12]$. For the second function the weight distribution is inhibited in the $[-1, 1]$ range.

Figures 4.2-4.3 provide the results of the investigation of the sensitivity analysis of neural networks over the introduction of three indicators. The (red) stars in the upper left plot illustrate the Euclidean distance (L_2). The (black) diamonds in the middle left plot represent the outcomes for the L_∞ indicator. The (blue) triangles depict the cosine similarities in absolute value.

The representation of peaks on every plot exhibits the optimal solution of the function, which is a function of the noise applied. Likewise, irrespective the type of indicator that has been used, one recognizes that the peaks for the solutions are represented in the same position (see fig. 4.4 for magnified illustration).

The upper left plot on fig. 4.2 represents the Euclidean distance parameter which shows that $45 - 47\%$ noise improves the solution significantly. Additionally, an observation in terms of the outputs from the L_2 and L_∞ show that although they diverge quantitatively, they are qualitatively (their behavior is) similar. In this particular numerical experiment, one expects the parameters to ascertain the same information. Although small amount of noise represent a sufficient solution to the optimization problem, the cosine similarity indicator does not provide the same conclusion. In fact, one observes in this case the presence of oscillation of the solution. Considering that in relation to this particular indicator, the closer the output to 1 is the better, the depiction on the lower left plot on fig. 4.2 represent deterioration of the solution (moving farther from 1).

The reader should note, that although we have used the cosine similarity indicator, we in particular have utilized a variant of it, using the absolute value of $L_{cos} = \left|\frac{u \cdot v}{|u||v|}\right|$. In this sense, regardless the similar peaks from the three indicators, we are capable to observe different information. Importantly, whereas the first two indicators provide sufficient noise levels results for the behavior on the network, the third one reveals opposite conclusion.

Considering the Euclidean distance parameter (fig. 4.2 upper left plot), we observe the presence of a distinctive peak around 40% noise, which indisputably indicates improvement of the network. As such the quantity of noise is improving the solution drastically. In contrast, the third parameter provides a snapshot of a deterioration of the network's behavior when more noise is applied. Moreover, in terms of L_2 and L_∞, their decreasing amounts represent a good solution to the problem, in L_{cos} we observe oscillations and notice that near 1 the network provides a sufficient solution, and away from it there is no such available good solution.

Acknowledging the fact that while the plots provide valuable information, the dynamic nature of the results necessitates the computation of the numerical derivative of the three indicators. The right side plots of fig. 4.2 describe the derivative values for the SA indicators. The (red) stars in the upper right plot provide the derivative for the Euclidean distance (L_2). The (black) diamonds in the middle right plot are the derivative outcome for the L_∞ indicator. The (blue) triangles depict the derivative for the cosine similarities in absolute value.

We have used a second order forward finite differential approximation in order to utilize the derivative values for the SA indicators. Specifically,

$$y''(x_i) = \frac{y(x_{i+2}) - 2y(x_{i+1}) + y(x_i)}{\Delta x^2}. \tag{4.5}$$

Considering that the peaks from the three indicators behave qualitatively similarly, the utilization of the derivatives allows us to conclude that the same remains true

in the derivatives case as well. Here, the peaks similarly correspond to the optimal solution. Consequently, the derivatives of the parameters contribute for the better localization of the peaks.

There is some uncertainty in the situation with the initial noise levels, where there is oscillation of the optimal solutions. In terms of the the cosine similarity this is the case precisely relating to the calculation of the second derivative in order to better understand the underlying processes.

From the lower right plot on fig. 4.2 of the derivative for the L_{cos} parameter, we observes that better solutions can be found near the 50% noise level, as well as in the smaller amounts where the noise is only about 15%.

The second experiment (see fig. 4.3) for the the square root of a polynomial ($f(x) = \sqrt{x}$) has also been calculated with the three parameters and their respective derivative outputs and contributes to similar observations. While the first two parameters are qualitatively similar, they are in fact quantitatively different. The presence of the peaks for the optimal solutions for the L_2 and L_∞ again present similar information from the sensitivity analysis. Whereas there are few better solutions produced for less than 10% of noise, surprisingly one can observe there are several better solutions in the $70th$ percentile. Irrespectively, there is a better chance for an optimal solution when smaller amounts of noise are added to the system, which improves the network's behavior (see fig. 4.3, upper and middle left plots). In contrast to the previous numerical experiment, presently, we can illustrate that there are several smaller hills pointing to the solution with less rapid oscillation from point to point.

Expectedly, the cosine similarity provides a divergent results when compared to the information from the previous indicators. Fig. 4.3 lower left plot illustrates the L_{cos} parameter in its absolute value. For better understanding of the indicators, we have once again used the derivative of the indicators, as explained above. In the upper and middle right plots from fig. 4.3 one may observe that the indicators gravitate towards zero regardless the amount of noise, although better solutions are found in the $5 - 10\%$ noise application part, as well as when 40% is applied and a little bit when over 60% noise is considered.

4.3 Summary and Contribution of the Chapter

In this chapter, we have introduced a sensitivity analysis of the perturbation of the weights on a three layer fully connected network. For training algorithm we have utilized the simulated annealing technique. With those sensitivity analysis tools we have provided a method for calculating the noise in the network. To achieve this we applied three indicators - Euclidean distance (L_2), cosine similarity (L_{cos}), and L_∞. Each one contributes important knowledge to the optimal solution for the acceptable noise applied to the network. Furthermore, with those tools we can establish the amount of noise which won't negatively influence the behavior of the network or

having a negative effect on the output. Moreover, those indicators give us the opportunity to determine accurately the level of noise which provides improvement to the optimization process or instead are acting as a source of fluctuation.

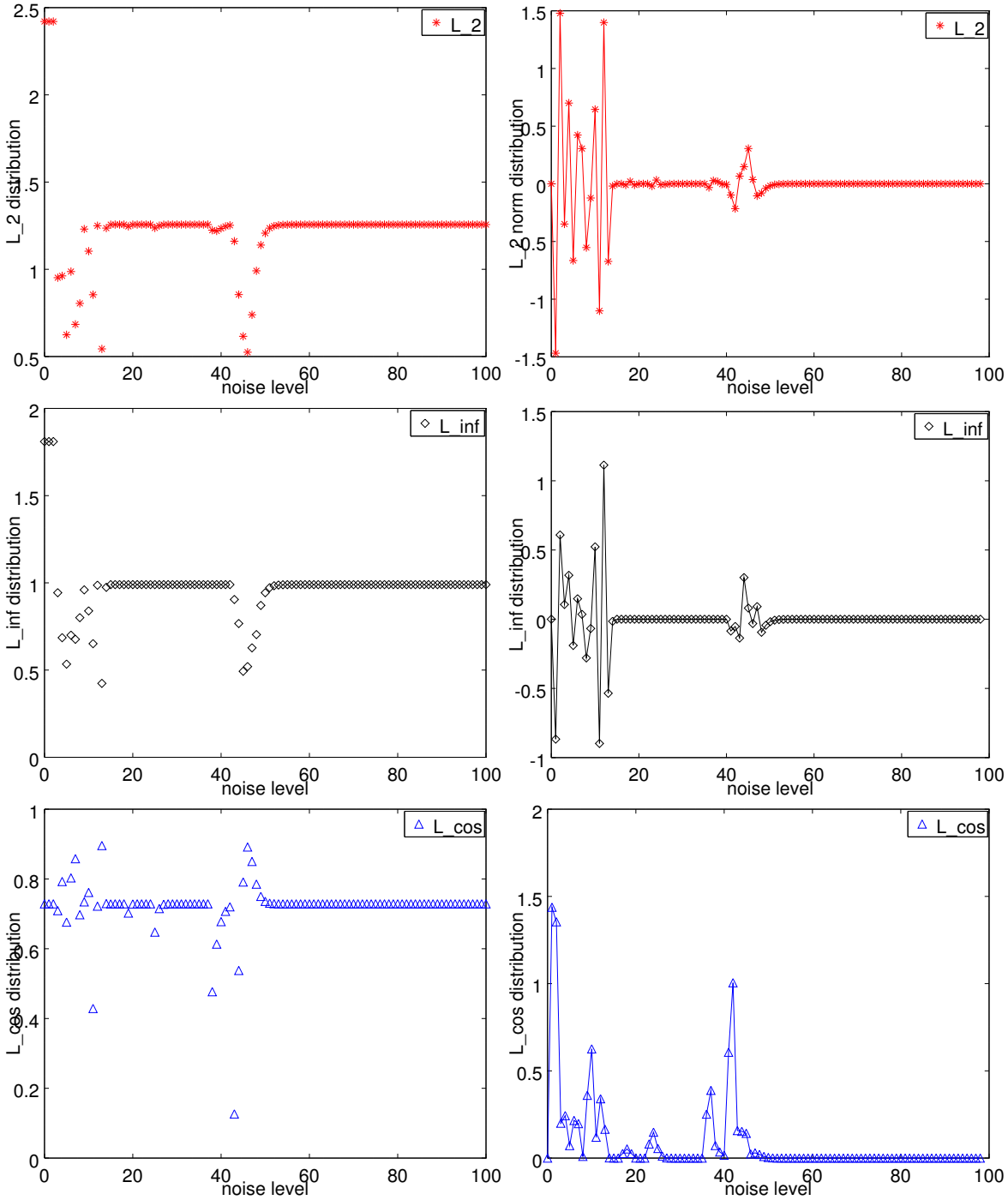


Fig. 4.2: The plots on the left side represent the three sensitivity analysis indicators applied for a polynomial of second degree ($f(x) = x^2$). .

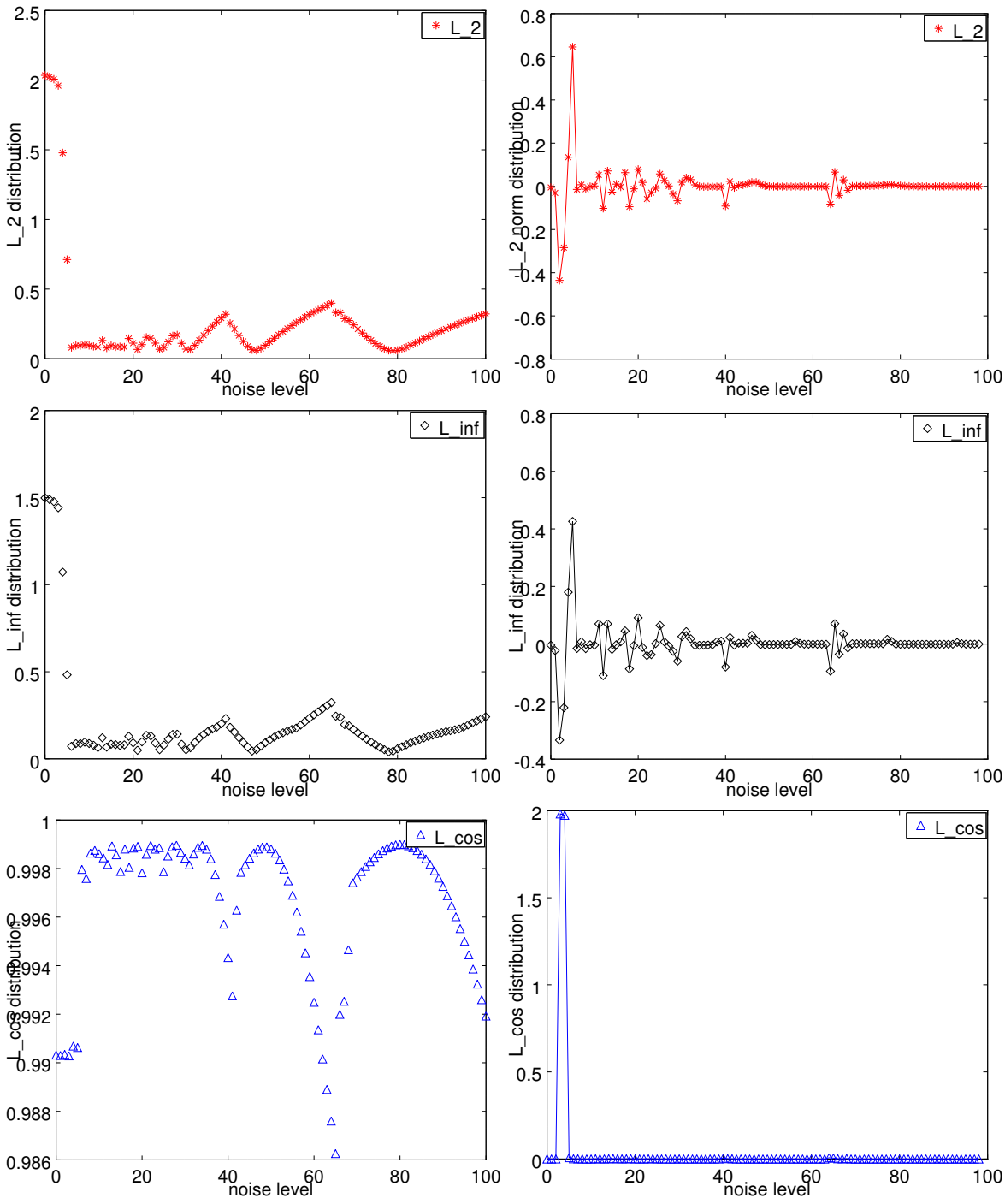


Fig. 4.3: The figure represents the sensitivity analysis indicators for the square root of a polynomial ($f(x) = \sqrt{x}$).

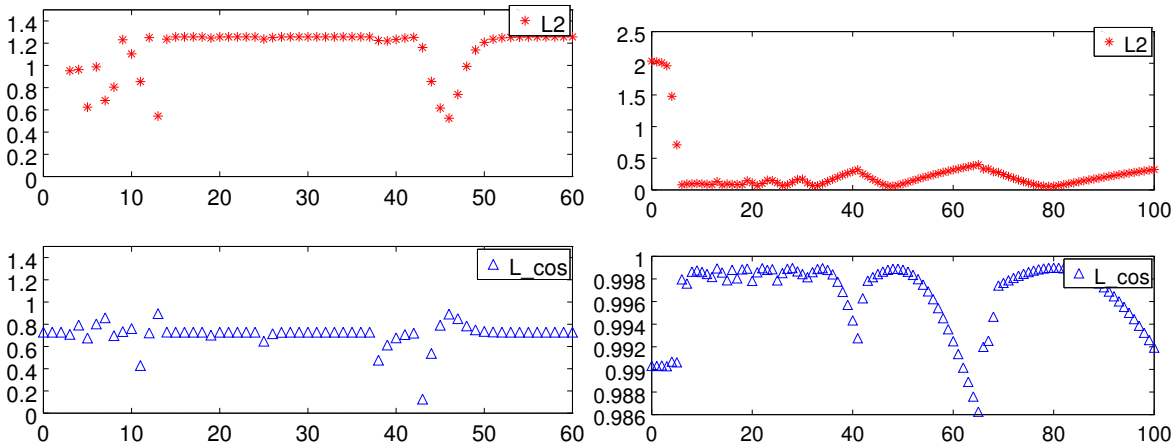


Fig. 4.4: The left side plots describe the comparison between the Euclidean distance (L_2) indicator and the cosine similarities in absolute value for the case of a polynomial of second degree ($f(x) = x^2$). The right side plots illustrate the comparison between the Euclidean distance (L_2) indicator and the cosine similarities in absolute value for the case of the square root of a polynomial ($f(x) = \sqrt{x}$)

Evolutionary ANN Architecture

5

> *The Moment of truth is running a program.*
>
> **— Herbert Simon**
> Artificial Intelligence: an Empirical Sciences

In Chapter 2 we have presented some of the fields where artificial neural networks are used and have proved to be a successful and efficient approach to various problem solving tasks. The ability of neural networks to utilize learning techniques to adapt their connections when new information is provided in direct contrast with conventional software techniques which needs to be programmed, has been one of the reasons why various paradigms and network architectures have been proposed and studied in the last 80 [75], [21]. The connectionist field and the field of artificial intelligence have addressed multiple problems pertaining to neural network structures and the influence of various parameters have on the network's performance [69]. T he possible number of layers, the amount of neurons in the network and per layer, the type of connections, the type of activation functions, weight initialization, training algorithms, error functions and pre- and post processing of data have been some of the parameters for the successful implementation of a neural network. Those components are essential to the computational performance, efficiency and accuracy of every network [92], [6]. It has been established that the topology of a neural network in fact influences the performance of the network with examples in the field of computer vision, where researchers have found that convolutional networks provide the best performance in this particular case [98]. The exponential number of parameters one can choose from depending on the problem and the knowledge of the parameters often presents a complex combinatorial optimization problem. Therefore various self-organizing topologies have been proposed and implemented [98], [176], [24], [29].

We introduce a novel approach to the evolution of the neural network architecture[1]in order to provide an automatic and computationally feasible self-organization of an artificial neural network layout to solve a given problem. A hybrid stochastic optimization and genetic algorithm, allowing the network to choose almost every possible structure in the implementation of the network in a given space of possible architectures is discussed. Through this evolving architecture, we obtain a method which is easily parallelized and is capable of escaping to overfitting of the output data when the minimal amount of training data is available.

For th automatic search/optimization of neural network architectures, as well as their parameters an Evolutionary Algorithm (EA) is commonly utilized. In such scenarios an evolutionary search is employed for some or all of the parameters [37]. Two possible utilization areas: parametric learning, where one searches for weight values. The second one is structural learning, where one searches for the best topology of the neurons and connections. Traditionally, the topology is defined before

the initialization of the network in certain manner. The simplest implementation of the structural type of evolving topologies is through the addition or removal of neurons, which limits the architecture, without exploring the whole space of possible architectures [14], [108]. Different approach is through the definition of a predefined list of modifications such as adding only fully connected hidden neurons, etc [156], [157]. In this instance in an event where the network topology is deemed insufficient, the architecture is discarded and a new one is developed. Disadvantage of this method is the algorithm and therefore the developed architectures often fall in structural local optima, incapable of identifying further available parameters [92]. Evolutionary algorithms (EAs) have been implemented in both the optimization (training) of networks as well as the automatic design of neural networks [122], [121], [9], [178], [106]. Evolutionary algorithms as heuristic methods are efficient in the exploration of the possible space of solution for the optimization problem. In the automatic search for the various parts of a topology one can implement a genetic algorithms, which can operate in a sequential layer search process, where the information for every layer is found in a specific genetic algorithm. Disadvantage to this method is the prohibitive computational costs and high processing time [109].

Consideration on the trade-off between high computational costs and the implementation of automatic evolution of topologies and training of the network and the possible advantages of such a method need to be accounted for. This chapter, therefore, introduces novel approach to the evolution of the neural network architecture in order to provide an automatic and computationally feasible self-organization of an artificial neural network layout to solve a given problem. A hybrid stochastic optimization and genetic algorithm, allowing the network to choose almost every possible structure in the implementation of the network in a given space of possible architectures is discussed. Through this evolving architecture, we obtain a method which is easily parallelized.

5.1 Methodology and Development

Our algorithm allows for the randomization of the number of neurons, number of hidden layers, types of synaptic connections, use of transfer functions and type of training algorithm, when selecting the best possible topology. The recombination of all those components establishes a multidimensional space of possible structures, while capable of achieving sufficient performance at affordable computational resources. During the numerical experiments, we have provided a degree of freedom for the evolutionary process for the following different network parts:

- number of possible hidden layers,
- number of possible neurons in every hidden layer,
- number of connections per neurons,
- the connection between neurons,
- the type of activation function.

The following subsections provide more details about those options.

5.1.1 Layers, Neurons and Connections

The number of layers, the amount of neurons per hidden layer and the available connections are the first three criteria used in the neural network architecture method developed in this work. The model operates first by restricting the available choices to a minimum and maximum value allowed for layers and neurons for each run of the network. There is no specific implementation in the code for the addition or removal of neurons during the process.

The network is not limited to a type of connections between neurons. The evolutionary algorithm can choose to develop a sequential neuron connection (in a feedforward manner), or non-sequentially - connecting neurons irrespective of their position between layers. The evolutionary algorithm provides the best possible solution by comparison between elements of various generations. One could restrict the number of inputs per neuron allowed in the architectures depending on the specific task that needs to be solved (i.e. to further control dimensions of the space of solutions).

Once those possible choices are considered, the next step to account for is the activation functions.

5.1.2 Activation Functions

The fourth criterion used in our proposed evolutionary network architecture provides several distinct activation functions to be utilized in the network. There are at least two possibilities during the evolutionary search: one may request all neurons in the hidden layer to use the same activation function, or to provide the option for the algorithm to choose the best one for every neuron in the hidden layers. This work incorporates several types of activation functions:

- identity Function - where $f(x) = x$. It is the simplest activation function where the activation is passed as the output of the neuron,
- exponential activation, where $f(x) = exp(x)$,
- hyperbolic Tangent (tanh) - a trigonometric function with output in the range $[-1, +1]$. One of the most used activation functions,
- function based on orthogonal polynomials - Laguerre, Legendre or Fourrier.

In this particular work, for each hidden layer, we leave a complete freedom to the network to choose the type of activation function.

5.1.3 The Training Process - optimization Strategy

The training process is based on the simulated annealing method [90]. By analogy with physical systems, we initially increase an effective temperature to a maximum value and then we start to decrease it until the particles reach an equilibrium (which represents a solution to the task). One describes the probability of a point to move by

$$Pr[accept] = e^{\frac{-\Delta E}{T}}, \tag{5.1}$$

where ΔE is the difference between the actual energy and the energy before the move, and T is the effective temperature of the system. A move is accepted if the generated random number $[0, 1] \ni R < Pr[accept]$.

5.1.4 Evolutionary Strategy

The genetic algorithm has been one of the most widely used approaches in the studies of evolution of neural network architectures [92], [176] [90], [156], [109]. In this work, we introduce a new hybrid genetic approach through the application of an additional stochastic layer, running simultaneously to the genetic one. The latter combines mutation and crossover approach, through which the algorithm evolves the potential topological combinations. Then we select the offspring with the best fitness.

Fitness and Selection Strategy. The algorithm initialization assigns fitness scores to each individual in the population, indicating the the quality of the specific network topology. In this work, we have utilized the $L_2 - norm$ fitness function (although we are not limited to only this function). Each time step, the algorithm accepts for further evolution an individual with the best fitness. The process is similar to the Tournament selection method [162].

Training Process. At each time step, every individual from the population is trained by the simulated annealing method described above. In this regard, one might consider the combination of the architecture selection and the training process as a two nested optimization problem. Therefore, one might speculate the possibility of both strategies influencing one another.

Initialization of the Technique. The algorithm is provided with the ability to choose from four specified parameters described in the previous subsection: number of possible hidden layers, number of possible neurons in every hidden layer, number of connections per neurons, and type of activation function when choosing topology parameters.

The algorithm involves two synchronous steps. We utilize integer value encoding, with each chromosome representing a string of the parameters of the network. In the integer value encoding, the crossover operator is applied in the same manner as that in a binary encoding. This is needed since the system has the option to select between several different activation functions or training algorithms, which in the code are notated in an integer form.

Type	Value
Max Number generations	32
Population	50 in genetic and 50 in stochastic layer
Rate of change	0.25
Selection Method	Tournament
Crossover	single-point uniform
fitness function	$L_2 - norm$

Tab. 5.1: Main values for selection in the hybrid genetic algorithm

```
// activation functions
#define TANH     0
#define EXP      1
#define ID       2
#define POL1     3
#define POL2     4
// discriminant functions
#define LINEAR   0
#define LEGENDRE 1
#define LAGUERRE 2
#define FOURIER  3
// error/cost/target functions
#define L2   0
#define LINF 1
#define COS  2
Code implementation for value encoding
```

The genetic algorithm is initialized randomly and creates 50 populations of individual architectures $C_1, C_2, C_3, \ldots C_n$ from the space of possible configurations. Concurrently, the stochastic layer generates randomly another 50 population of individual architectures $S_1, S_2, S_3, \ldots S_n$. The stochastic layer functions entirely randomly in every iteration. The genetic layer utilizes a combination of mutation and crossover to evolve the solutions in order to find the best one for each generation. The mutation randomly selects a chromosome. In mutation according to *ratechange* the algorithm change the value of the specific chromosome by a certain value at random to be mutated. In terms of the crossover operator we utilize a single-point crossover chosen at uniform random form (using the *ratechange* parameter) either from the first parent or from the second one in the following manner:

```
Parent #1: 0111010101
Parent #2: 1001110011
Offspring #1: 0101010111
Offspring #2: 1001110101
Example of a crossover
```

While the stochastic layer on the other hand performs random search to generate new individuals for each generation. At every step, the algorithm evaluates the

fitness level of each individual from both batches. It then selects one individual from the the genetic and stochastic layer. From there, the individual with the best fitness is further selected. Follows a stochastically evolved individual, applying the controlled settings described above. We create a new set of populations of individual network topologies for the genetic and stochastic layers of the algorithm. The entire run of the genetic algorithm utilizes both crossover, as well as mutation strategies.

```
int maxgen =   32;    /* number of generations */
int npopulation = 50;  /* number of individuals per generation */
int ratechange = 0.25;/* rate of change between one generation and the parent */
int eps  =   1.e-5;  /* numerical accuracy */
crossover = uniform;
Code implementation for genetic layer
```

One might consider the stochastic evolution of the best individual to allow us several advantages. Importantly, it provides a way for the algorithm to escape from local optima, especially if the overall solution has various populations. We achieve this through the adaptation according to minimum design standards in the changing environment. Moreover, the stochastic part allows the program to develop a new architecture specific to a problem in an affordable computational time even in the presence of multidimensional search space.

Termination To be terminated the algorithm needs to meet one of two conditions. It needs to reach the maximum assigned number of generations or the fitness error to fall below the assigned threshold value after r consecutive generations.

5.2 Results

This section presents the results from a numerical experiment of fitting a known function $f(x) = x^2$ in order to corroborate the ability of the proposed method. Several examples are provided, presenting the method's outcomes with restrictions in the number of hidden layers and neurons in them. This work presents architectures from 4 scenarios - implementing architectures with $3, 4, 5,$ and 6 hidden layers and different number of neurons allowed for each hidden layer $2, 3, 4, 5, 6$. For the current numerical validation, the amount of input connections per neuron in limited to the maximum- 2. Those restrictions are only executed in the current experiment and are not in any way intrinsic to the effectual work of the proposed hybrid evolutionary technique.

Similar to the training data utilized in the previous chapters, the training data is limited to three points - $(0.1, 0.1)$, $(0.5, 0.5)$ and $(0.9, 0.9)$. Since it has been established [21] that in order to guarantee a good function approximation ANNs require large enough data samples, by limiting the available training data one should expect the presence of overfitting and inability of the network to learn complex relationships [155]. Accordingly, one of the aims of the experiments is to present whether the hybrid evolutionary architecture could overcome those limitations

5.2.1 Three layers from 2 to 6 neurons

For the first experiment, the possible layers are limited to 3 - one input layer, one hidden layer and one output layer. Further constrictions (applicable to all numerical experiments) are provided on the maximum amount of neurons in every hidden layer to $3, 4, 5$ or 6. The results in table 5.2 indicate that the cases[2]($3 \times 2 \times 2$ and $3 \times 4 \times 2$) where the smaller error is observed are in fact the situations where the algorithm utilizes only 1 neuron in the hidden layer. In table 5.2 one observes 5 cases in this scenario - with the maximum of $2, 3, 4, 5$ and 6 neurons per hidden layer. The fourth column represents the total amount of neurons in the network (including the neurons in the input and output layer), while the last column represents the final error from the evolutionary process.

In figure 5.1 presents the output of the network (left column) and the respective architectures used in each case (right column). In the first case $3 \times 3 \times 2$ with the (red) line is outlines the function we try to fit, the (blue) squares describes the neural network output and the (red) stars represents the three data points provided for the training. On the right side of the figures, the plots outline the respective architecture for every case. In this regard the (yellow) node depicts the input node in the input layer, the (red) node shows the output node in the output layer, and the (blue) nodes represents the neurons in the hidden layers respectively. The synaptic connections between the neurons are represented as edges between the nodes. The type of activation function is symbolized according to a number, which is explained in the legend on the lower right corner of each plot. The middle left and right plots represent the output and network architecture in the case for $3 \times 4 \times 2$. The lower left and right plots represent the network's output and architecture in the case for $3 \times 6 \times 2$.

Significantly, in the last case (where 6 neurons can be used in the hidden layer), one observes that the algorithm seldom utilizes all allowed neurons. In this scenario, the associated limitations of the number of hidden layers impose the model to generate feedforward networks. In the cases of $3 \times 3 \times 2$ and $3 \times 4 \times 2$ the method contributes to an output with close to perfect function approximation (see fig. 5.1 upper left and middle left plots respectively).

The first scenario represents network topologies in the form of a feedforward neural network, where the $3 \times 3 \times 2$ and $3 \times 4 \times 2$ networks provide outputs in the range of desired accuracy (see fig. 5.1 left comlumn, top and middle rows respectively). Notwithstanding the architecture, in the case of $3 \times 6 \times 2$, one might consider the situation where the selection of an activation function actively impacting the desired network performance. Adversely the selection of neurons in the hidden layer appears to contribute to an over-fitting outcome for the results. Such a conclusion is supported by [155].

[2]The reader should note that the notation $n_l \times n_{max} \times c_{max}$ connotes information about the number of layers in the architecture (in this case n_l), the maximum number of neurons in every hidden layer (in this case n_{max}) as well as the maximum amount of connections from neuron to neuron (in this case c_{max})

Case	Num. layers	# neurons in hidden layer	# Neurons in the Network	Error
3x2x2	3	1	3	0,017309
3x3x2	3	2	4	0,018815
3x4x2	3	1	3	0,017909
3x5x2	3	1	3	0,0185091
3x6x2	3	3	5	0,0477594

Tab. 5.2: Scenario 1: network architecture is limited to only one hidden layer.

5.2.2 Four layers from 2 to 6 neurons

The second scenario represent an increase of the space of possible network topologies due to adjustment of available hidden layers to 2, while the possible cases represent the maximum amount of neurons per hidden layer with $2, 3, 4, 5$ and 6. Table 5.3 represents those 5 cases in this scenario. The fourth column represents the total amount of neurons in the network (including the neurons in the input and output layer). The last column in the table provides the final error from the hybrid evolutionary process.

One may recognize the accrual of more hidden layers and neurons in them, conditions the hybrid algorithm to develop networks with the same number of neurons in every hidden layer (see table 5.3). In the first case of $4 \times 2 \times 2$, the best architectural solution represents 2 hidden layers, each with 1 neuron. The topology is a type of feedforward network, although the algorithm has utilized two connections from each neuron to the next one. Along with those connections, the selected activation function for the neurons is the exponential one. The final topology provides the smallest error (0.006946, in the $L_2 - norm$), even in the presence of an overfitting (see fig. 5.2 first row, left and right column). Possible inference is the ability of the cost function decrease to influence the irregularity of the output pattern.

In fig. 5.2 one observes in the first case $4 \times 2 \times 2$ that with the (red) line is represented the function to be fitted, the (blue) squares depict the neural network output while the (red) stars represent the three data points provided for the training. Respectively, the right side plots illustrate the respective architecture for every case. The (yellow) node depicts the input node in the input layer, the (red) node shows the output node in the output layer, and the (blue) nodes represents the neurons in the hidden layers. As in the previous scenario, the numbers near every neurons represent the type of activation function utilized by each neuron (described by the legend on each plot). The middle left and right plots represent the output and network architecture in the case for $4 \times 4 \times 2$. The lower left and right plots represent the network's output and architecture in the case for $4 \times 5 \times 2$.

Incremental increase of the maximum number of neurons per hidden layer to 4 and with maximum of hidden layers to 2, contributes to an outcome where the hybrid algorithm produces network topology with the same amount of neurons (3) for every hidden layer. Accordingly, the method selects three types of activation are

Case	Max. # layers	# neurons in hidden layer	# Neurons in the Network	Error
4x2x2	4	hidden #1 - 1 neuron hidden #2 - 1 neuron	4	0,006946
4x3x2	3	hidden #1 - 2 neurons	4	0,017937
4x4x2	4	hidden #1 - 3 neurons hidden #2 - 3 neurons	8	0,019676
4x5x2	4	hidden #1 - 2 neurons hidden #2 - 2 neurons	6	0,017426
4x6x2	4	1 hidden - 1 neuron	4	0,034233

Tab. 5.3: Scenario 2: network architecture is limited to only two hidden layers.

utilized - logistic activation function in 2 neurons and the identity and second order polynomial as activation functions for the other neurons.

Significantly, the hybrid evolutionary method has automatically pruned certain neurons from connecting any further (see fig. 5.2,right column, second right row). Comparing the final error from the other cases shown in table 5.3, one might observe the final error being larger. This directly contrasts with the actual output of the network (see fig. 5.2 left column, middle row), which is the desired range accuracy.

Fig. 5.2 lower right plot (case $4 \times 5 \times 2$) depicts the ability of the hybrid algorithm to utilize the neuron connection freedom. The construction of the topology consists of 4 layers - 1 input, two hidden and 1 output layer. Interestingly, the input layer is connected to every other available neuron through the network, inclusive of the output neuron. Along with this connection, the output neuron receives a connection from the second neuron in the first hidden layer. As a consequence, the hybrid method discards the implementation of the second hidden layer by pruning the output connections from those neurons. Evident from table 5.3 the error converges to the accepted minimum. The output of the network respectively exhibits the intended function approximation. Probable explanation of the small error and the desired output can be given by means of the presented results in [40] (important observation in this case is the removal of neurons through their connection). As a results, one might see our hybrid evolutionary method as capable of building a representation of a model, where the option of discarding input connections is viable and seemingly beneficial to the particular method.

5.2.3 Five layers from 2 to 6 neurons

One should evaluate the performance of the hybrid algorithm in a situation where there is an expansion of the multidimensional space of possible structures. As a matter of course, we have examined scenarios with 3 hidden layers. 5 cases in this scenario were developed - with implementation of maximum $2, 3, 4, 5$ and 6 neurons per hidden layer. Byproduct of such expansion should be higher computational costs, possible performance decrease, as well as divergent output in relation to the desired

range of solutions. The latter should be considered as a probable aftereffect as the network is unable to learn complex patterns when provided with limited training data while there is an increase of the number of neurons in the network [155].

In the first architecture from this scenario - $5 \times 3 \times 2$, each of the 3 hidden layers utilizes 2 neurons. Further adjustment of the network's architecture is achieved through the reduction of the number of connected neurons (see fig. 5.3 second column first row). In fig. 5.3 with the (red) line is depicted the function to be fit, the (blue) squares describe the neural network output while the (red) stars represent the three data points provided for the training. The right side plots characterize the respective architecture for every case. The (yellow) node shows the input node in the input layer, the (red) node represents the output node in the output layer, with the (blue) nodes - the neurons in the hidden layers respectively. The edges from the nodes account for the synaptic connections between nodes. Once again the numbers near each neuron represent the type of activation function selected by the algorithm, with the legend on every plot describing each function.

The strategy of neuron pruning has been utilized by various evolutionary methods in previous cases. Particularly in this case, one recognizes rather large final error (see table 5.4, where the fourth column represents the total amount of neurons in the network (including the neurons in the input and output layer), while the last column represents the final error from the evolutionary process). We furthermore observe a sharp overfitting of data. One may reasonably infer that the resemblance of the network to a traditional feedforward topology, in this situation presented with limited data for learning could be a limitation the ability of the method to provide a sufficient function generalization.

In the $5 \times 5 \times 2$ experiment (represented in fig. 5.3 lower left and right plots), similar to the previous case, the hybrid algorithm selects 3 hidden layers, consisting of 2 neurons each. Comparable to th previous $4 \times 5 \times 2$ network architecture, the method develops connections from the neuron in the input layer to every other neuron in the structure, including the one in the output layer (see fig. 5.3 second column, second row). Once again one may observe that the obtained topology has discarded almost all connections from the rest of the neurons, leaving neuron 1 from hidden layer 2 to connect to the output layer, along with the connection between the input and the output neuron. As such, the output reaches near optimal approximation (see fig. 5.3 first column, second row).

5.2.4 Six layers from 2 to 6 neurons

The last experiment introduces further expansion in the space of possible architectures, allowing for the algorithm to select up to 4 hidden layers. 5 cases in this scenario were developed - with implementation of maximum $2, 3, 4, 5$ and 6 neurons per hidden layer.

In the $6 \times 3 \times 2$ case, depicted in fig. 5.4 upper right plot, the algorithm produces an architecture with 4 hidden layers. Each layer has 1 neuron, which in turn is sequentially connected by 2 synaptic connections. Three possible activation functions

Case	Max. # layers	# neurons in hidden layer	# Neurons in the Network	Error
5x2x2	3	hidden #1 - 1 neuron hidden #1 - 2 neurons	3	0,019079
5x3x2	5	hidden #2 - 2 neurons hidden #3 - 2 neurons	8	0,016791
5x4x2	5	hidden #1 - 1 neuron hidden #2 - 1 neuron hidden #3 - 1 neuron	5	0,006132
5x5x2	5	hidden #1 - 2 neurons hidden #2 - 2 neurons hidden #3 - 2 neurons	8	0,015428
5x6x2	4	1 hidden - 1 neuron	10	0,008473

Tab. 5.4: Scenario 3: network architecture is limited to only three hidden layers.

are assigned to the neurons. In fig. 5.4 with the (red) line we represent the function we try to fit, the (blue) squares relates to the neural network output and the (red) stars represent the three data points provided for the training. On the right side of fig. 5.4 are showed the corresponding architectures for each case. There the yellow node characterizes the input node in the input layer, the (red) node shows the output node in the output layer, with the (blue) nodes - the neurons in the hidden layers respectively. Each edge in the network represents the synaptic connections between nodes. The numbers attached to each neuron depict the type of activation function used by every neuron, while the legend in the plots explains the types of activation functions.

In correlation to the $4 \times 2 \times 2$ scenario, the implementation of a feedforward topology generated bigger error (see table 5.5 as well as a presence of output overfitting. In table 5.5 the fourth column depicts the total amount of neurons in the network (including the neurons in the input and output layer), while the last column represents the final error from the hybrid genetic process. Notwithstanding, considering in this particular case, the availability of a free parameters - number of layers, neurons and the types of activation functions, constituting a multidimensional space of possible architectures, the method still provides a sufficient data fitting outcome at affordable computational cost.

The composition of the topology in the last case - $6 \times 6 \times 2$ (see fig. 5.4, lower left and right plots) utilizes 4 hidden layers, with 2 neurons each.

Compared to the $6 \times 3 \times 2$ case, one might observe a variation in the architecture consisting of a lack of sequential connection between neurons. Since the method utilizes a randomized synaptic connection method, the input neuron is connected to every other available neuron with one or two input connections. As a consequence, it seems the strategy allows for the removal of nearly 90% of the neurons in the network. Barring the input neuron connection to the output neuron, the only other connection is established with the second neuron in hidden layer 2. Similarly to the $4 \times 5 \times 2$ and $5 \times 5 \times 2$ cases, the implementation of randomized connections leads

Case	Max. # layers	# neurons in hidden layer	# Neurons in the Network	Error
6x2x2	6	hidden #1 - 1 neuron hidden #2 - 1 neuron hidden #3 - 1 neuron hidden #4 - 1 neuron	6	0,010450
6x3x2	6	hidden #1 - 1 neuron hidden #2 - 1 neuron hidden #3 - 1 neuron hidden #4 - 1 neuron	6	0,027712
6x4x2	6	hidden #1 - 2 neurons hidden #2 - 2 neurons hidden #3 - 2 neurons hidden #4 - 2 neurons	10	0,039770
6x5x2	3	hidden #1 - 1 neuron hidden #1 - 2 neurons	3	0,018634
6x6x2	6	hidden #2 - 2 neurons hidden #3 - 2 neurons hidden #4 - 2 neurons	10	0,0271704

Tab. 5.5: Scenario 4: network architecture is limited to only four hidden layers.

to a negligible overfitting of the output in the upper $3-4$ data points. Nonetheless, the network appears capable of developing an inner model, as well as providing a good data fit, irrespective of the training limitations imposed and the larger error (see table 5.5).

In accordance to the discussed numerical experiments, one may conclude that in certain situations a sequentially connected network might not be superior for certain tasks. Moreover, in situations where the proposed hybrid evolutionary technique utilizes more than one hidden layer with several neurons, the algorithm frequently performs a model of a neuron removal. As such it discontinues redundant calculations. In those instances, the network accomplishes sufficient results according to a user defined cost function. Conjointly, the neurons connected to the output layer consistently use the logistic activation function. Therefore, inherent consequence in bigger architectures, which allow for expanded search space of possible solutions, the hybrid algorithm reduces the architecture by providing several layers, though leavig only one neuron from them is connected to the output layer.

5.3 Summary and Contribution of the Chapter

In this chapter we have introduced a novel method for the automatic search of an optimal neural network architecture, given a specific problem (in our case - the fitting of a function). Through the implemented approach we provide several

degrees of freedom for parameter selection for the hybrid approach to search for a network structure, which can eventually become quite complex. According to the specific task, we believe the evolutionary strategy in our method reaches the optimal network topologies - whether through the automatic removal of unnecessary connections, or through the frequent usage of better activation function. Therefore, we obtain a reliable network topology not based on the experience of the researcher, but on well defined automatic evolutionary strategy. Furthermore, our approach is capable to produce close approximation results in the presence of a limited training data set, with little or no overfitting.

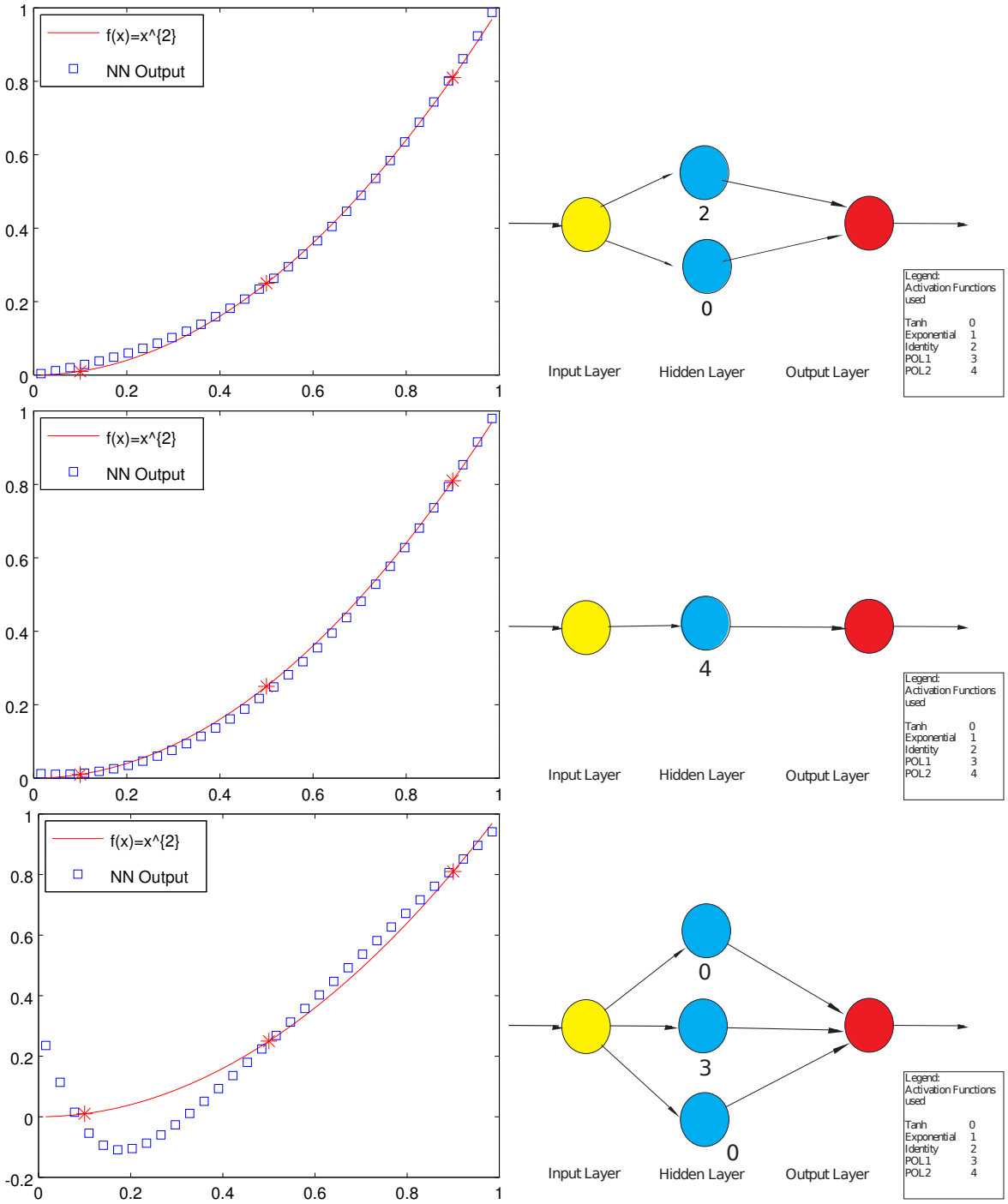


Fig. 5.1: Scenario with 3 layers,with $3, 4, 5$ or 6 maximum neurons per layer. The plots represent the output of the network (left column) and the respective architectures used in each case (right column).

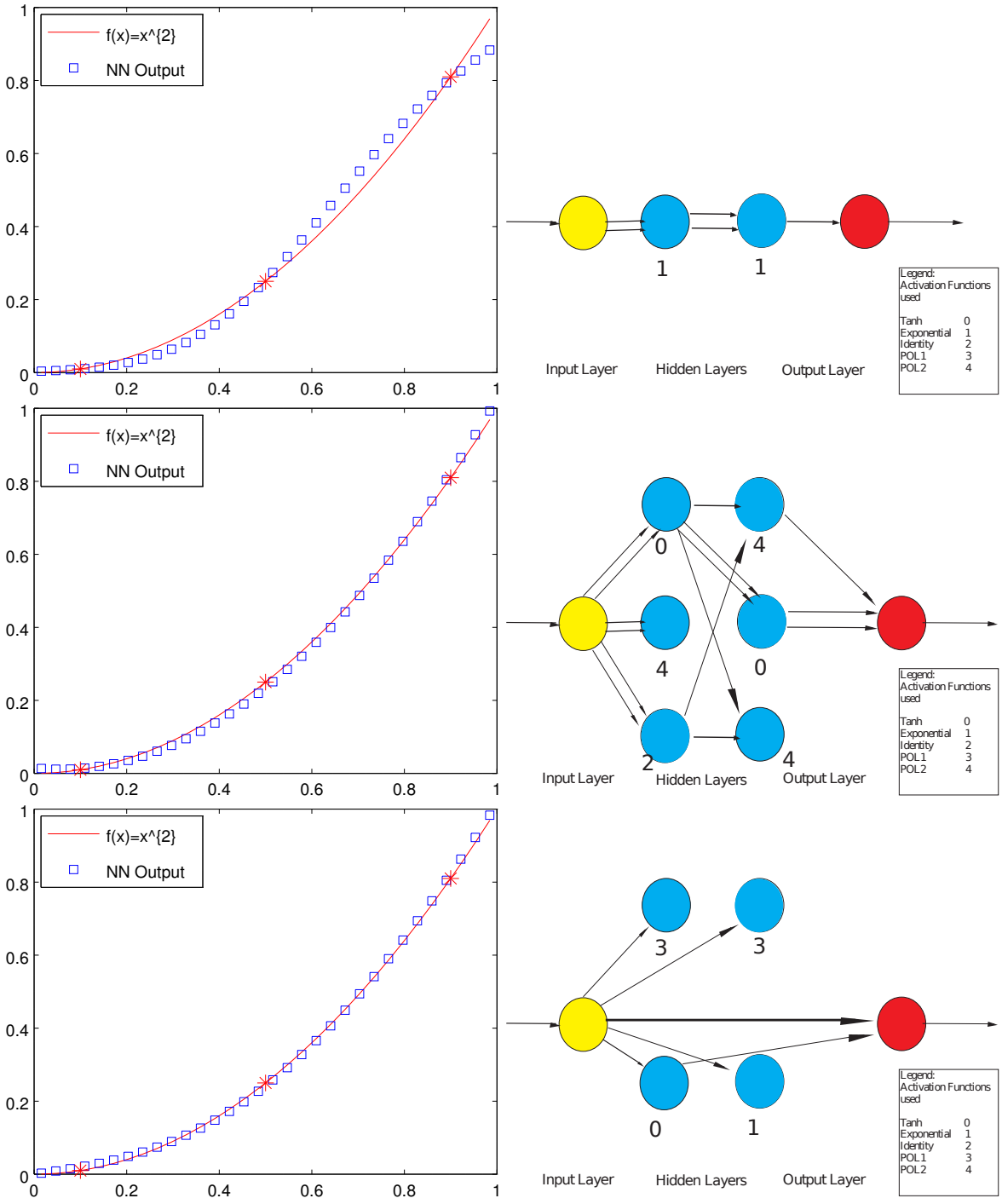


Fig. 5.2: Scenario with 4 layers,with $3, 4, 5$ or 6 maximum neurons per layer. The plots represent the output of the network (left column) and the respective architectures used in each case (right side plots).

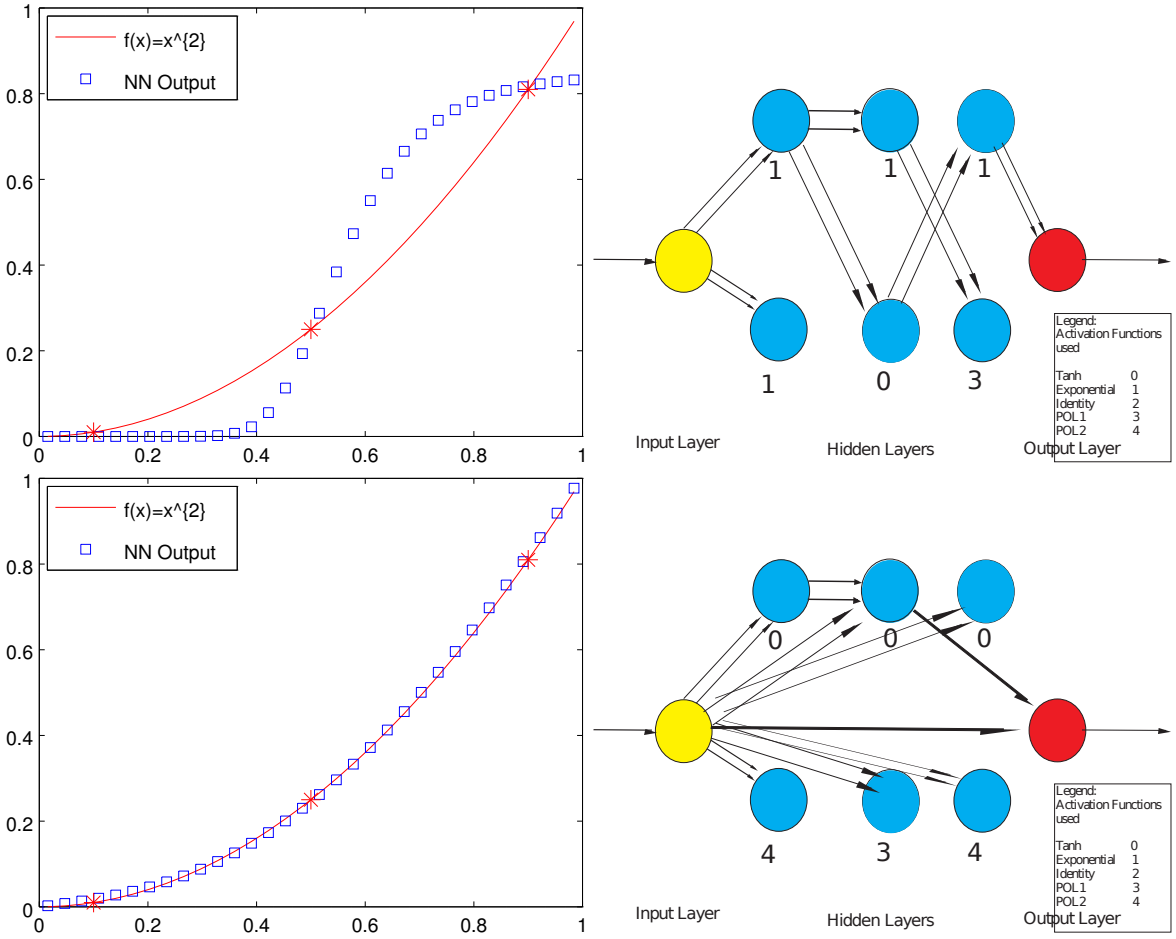


Fig. 5.3: Scenario with 5 layers,with $3, 4, 5$ or 6 maximum neurons per layer. The plots represent the output of the network (left column) and the respective architectures used in each case (right column).

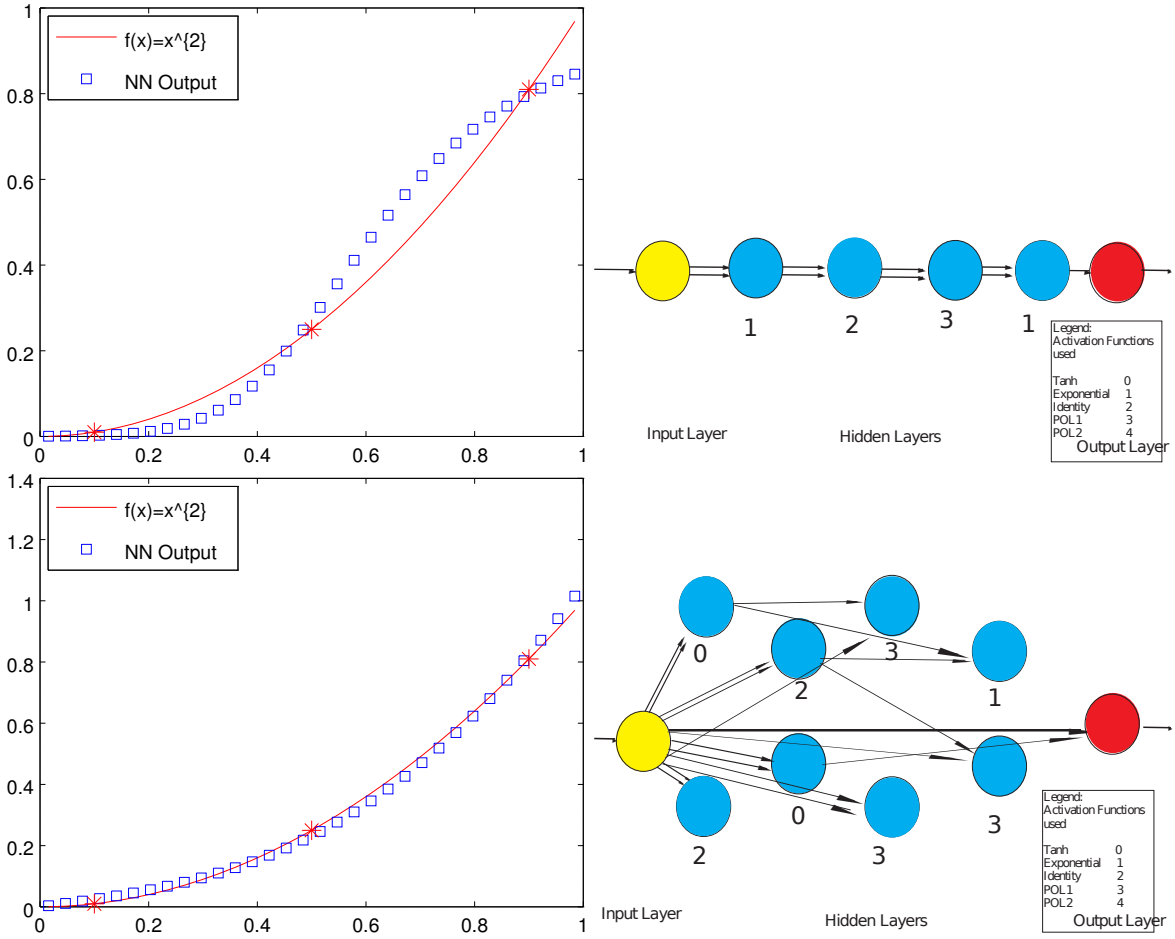


Fig. 5.4: Scenario with 5 layers, with $3, 4, 5$ or 6 maximum neurons per layer. The plots represent the output of the network (left column) with 6 layers and the respective architectures used in each case (right side plots).

Conclusion and Future work

6

> *Users do not care about what is inside the box, as long as the box does what they need done.*
>
> — **Jef Raskin**
> about Human Computer Interfaces

This dissertation has been structured with 6 chapters. The obtained results are presented in 29 figures, 5 tables, and 180 cited resources. The main part of the research has been submitted to international journals and has been already published.

Scientific Contributions

- This dissertation has introduced a new post-learning technique to assist the supervised learning algorithm of an artificial neural network to escape a local minima or a saddle point if such has been reached during the optimization process.
- We have implemented a mathematical tool to quantitatively measure the influence of noise in the perturbed weights on the performance of the network. We demonstrated numerical results indicating the effects of noise on the network and its performance sensitivity to such process.
- We have introduced a hybrid genetic algorithm for the automatic evolution of neural network architectures. This method, although providing a search in a multi-modal search space for all possible solutions is capable with minimal training data to produce an output with small or no overfitting of the output data, which is a common problem in the field of neural networks.

The neural network, the proposed post-learning technique, as well as the evolutionary algorithm for the automatic network architecture has been developed in the C language. The software allows for the specific definition of several network characteristics and more specifically:

1. Minimum and maximum number of neurons.
2. Minimum and maximum number of layers.
3. The specificity of the connections between neurons.
4. Specification of several cost function/activation functions.

5. Specification of three training algorithms - backpropagation (as well as definition whether the backpropagation should be batch or online), genetic algorithm, simulated annealing.

The author hopes that this research have provided useful tools for the further development of the field of neural networks. The described results were influenced by the world of quantum mechanics. Hopefully, this is only a small part of the future implementation of such interdisciplinary approach, as well as for the practical applications in those methods in real-life situations. Further research is focused on exploring quantum effects in deep neural network architectures, as well as improvement of the hybrid automatic architecture for deep architectures.

List of Publications

1. K.G. Kapanova, I.T. Dimov, J.M. Sellier, On randomization of neural networks as a form of post-learning strategy, Soft Computing (2015). doi : $10.1007/s00500-015-1949-1$, (IF.1.63).

2. K.G. Kapanova, I.T. Dimov, J.M. Sellier, A Neural Network Sensitivity Analysis in the Presence of Random Fluctuations, Neurocomputing (2016), doi : $10.1016/j.neucom.2016.10.060$, (IF.2.392)

3. K.G. Kapanova, I.T. Dimov, J.M. Sellier, A genetic approach to automatic neural network architecture optimization, Neural Computing and Applications (2016), doi : $10.1007/s00521-016-2510-6$, (IF.1.492)

Dissemination of results The results have been presented in two seminars at the IICT, BAS in department "Parallel Algorithms". The novel algorithms and the numerical experiments, presented in the thesis, have been reported at:

- Large Scale Scientific Computing, Sozopol, 2015.
- MCM, Linz, 2015.
- LinuxCon, Dublin, 2015.
- PhD Forum, BAS, 2016.

The presentation on the hybrid evolutionary algorithm and the numerical experiments have been awarded for best presentation during the PhD Forum, BAS, 2016.

The results have been published in 3 scientific articles, all of them in international journals with Impact Factor.

Bibliography

[1] Ajith Abraham. „Optimization of evolutionary neural networks using hybrid learning algorithms“. In: *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*. Vol. 3. IEEE. 2002, pp. 2797–2802 (cit. on p. 20).

[2] Tara H Abraham. „Nicolas Rashevsky's mathematical biophysics“. In: *Journal of the History of Biology* 37.2 (2004), pp. 333–385 (cit. on p. 2).

[3] Waleed A Maguid Ahmed, El M Saad, and ESA Aziz. „Modified back propagation algorithm for learning artificial neural networks“. In: *Radio Science Conference, 2001. NRSC 2001. Proceedings of the Eighteenth National*. Vol. 1. IEEE. 2001, pp. 345–352 (cit. on p. 35).

[4] S Akaho and S Amari. „On the capacity of three-layer networks“. In: *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*. IEEE. 1990, pp. 1–6 (cit. on p. 35).

[5] Israel E Alguindigue, Anna Loskiewicz-Buczak, and Robert E Uhrig. „Monitoring and diagnosis of rolling element bearings using artificial neural networks“. In: *Industrial Electronics, IEEE Transactions on* 40.2 (1993), pp. 209–217 (cit. on p. 24).

[6] Leandro M Almeida and Teresa B Ludermir. „A multi-objective memetic and hybrid methodology for optimizing the parameters and performance of artificial neural networks“. In: *Neurocomputing* 73.7 (2010), pp. 1438–1450 (cit. on pp. 20, 71).

[7] Filippo Amato, Alberto Lopez, Eladia María Peña-Méndez, et al. „Artificial neural networks in medical diagnosis“. In: *Journal of applied biomedicine* 11.2 (2013), pp. 47–58 (cit. on p. 24).

[8] Costas A Anastassiou, Rodrigo Perin, Henry Markram, and Christof Koch. „Ephaptic coupling of cortical neurons“. In: *Nature neuroscience* 14.2 (2011), pp. 217–223 (cit. on pp. 22, 61).

[9] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. „An evolutionary algorithm that constructs recurrent neural networks“. In: *Neural Networks, IEEE Transactions on* 5.1 (1994), pp. 54–65 (cit. on pp. 20, 72).

[10] Itamar Arel, Derek C Rose, and Thomas P Karnowski. „Deep machine learning-a new frontier in artificial intelligence research [research frontier]“. In: *Computational Intelligence Magazine, IEEE* 5.4 (2010), pp. 13–18 (cit. on p. 19).

[11]Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. „Locally weighted learning for control“. In: *Lazy learning*. Springer, 1997, pp. 75–113 (cit. on p. 18).

[12]Arash Bahrammirzaee. „A comparative survey of artificial intelligence applications in finance: artificial neural networks, expert system and hybrid intelligent systems“. In: *Neural Computing and Applications* 19.8 (2010), pp. 1165–1195 (cit. on p. 23).

[13]Janet M Baker, Li Deng, James Glass, et al. „Developments and directions in speech recognition and understanding, Part 1 [DSP Education]“. In: *Signal Processing Magazine, IEEE* 26.3 (2009), pp. 75–80.

[14]Sandy D Balkin and J Keith Ord. „Automatic neural network modeling for univariate time series“. In: *International Journal of Forecasting* 16.4 (2000), pp. 509–515 (cit. on pp. 20, 72).

[15]James Barrat. *Our final invention: Artificial intelligence and the end of the human era*. Macmillan, 2013 (cit. on p. 25).

[16]Eric B Baum. „On the capabilities of multilayer perceptrons“. In: *Journal of complexity* 4.3 (1988), pp. 193–215 (cit. on p. 35).

[17]Friedrich Beck and John C Eccles. „Quantum aspects of brain activity and the role of consciousness“. In: *How the SELF Controls Its BRAIN*. Springer, 1994, pp. 145–165 (cit. on pp. 47–49).

[18]Yoshua Bengio. „Learning deep architectures for AI“. In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[19]Roberto Benzi, Alfonso Sutera, and Angelo Vulpiani. „The mechanism of stochastic resonance“. In: *Journal of Physics A: mathematical and general* 14.11 (1981), p. L453 (cit. on p. 22).

[20]Frans Van den Bergh and Andries P Engelbrecht. „Effects of swarm size on cooperative particle swarm optimisers“. In: (2001) (cit. on pp. 39, 40).

[21]Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995 (cit. on pp. 15, 28, 30, 31, 36, 65, 71, 76).

[22]Christian Blum and Krzysztof Socha. „Training feed-forward neural networks with ant colony optimization: An application to pattern classification“. In: *Hybrid Intelligent Systems, 2005. HIS'05. Fifth International Conference on*. IEEE. 2005, 6–pp (cit. on pp. 37, 40).

[23]Léon Bottou and Yann L Cun. „Large Scale Online Learning“. In: *Advances in Neural Information Processing Systems*. 2003, None.

[24]Jürgen Branke. „Evolutionary algorithms for neural network design and training“. In: *In Proceedings of the First Nordic Workshop on Genetic Algorithms and its Applications*. Citeseer. 1995 (cit. on pp. 19, 71).

[25] John A Bullinaria. „Evolving neural networks: Is it really worth the effort?“ In: *ESANN*. 2005, pp. 267–272 (cit. on p. 36).

[26] Sam Byford. *Google's AlphaGo AI beats Lee Se-dol again to win Go series 4-1*. 2016. URL: http://www.theverge.com/2016/3/15/11213518/alphago-deepmind-go-match-5-result (cit. on p. 1).

[27] William H Calvin and CHARLES F Stevens. „Synaptic noise and other sources of randomness in motoneuron interspike intervals“. In: *J Neurophysiol* 31.4 (1968), pp. 574–587 (cit. on p. 21).

[28] Jinde Cao. „Global asymptotic stability of delayed bi-directional associative memory neural networks“. In: *Applied Mathematics and Computation* 142.2 (2003), pp. 333–339 (cit. on p. 18).

[29] Adenilson R Carvalho, Fernando M Ramos, and Antonio A Chaves. „Metaheuristics for the feedforward artificial neural network (ANN) architecture optimization problem“. In: *Neural Computing and Applications* 20.8 (2011), pp. 1273–1284 (cit. on pp. 19, 71).

[30] Vladimír Černỳ. „Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm“. In: *Journal of optimization theory and applications* 45.1 (1985), pp. 41–51 (cit. on p. 43).

[31] Gary Chartrand, Linda Lesniak, and Ping Zhang. *Graphs & digraphs*. CRC Press, 2010 (cit. on p. 9).

[32] Guy Chéron, Matthieu Duvinage, Thierry Castermans, et al. „Toward an integrative dynamic recurrent neural network for sensorimotor coordination dynamics“. In: *Recurrent Neural Networks for Temporal Data Processing* 5 (2011), pp. 67–80 (cit. on p. 18).

[33] Soumith Chintala. *FAIR open sources deep-learning modules for Torch*. 2015. URL: https://research.facebook.com/blog/879898285375829/fair-open-sources-deep-learning-modules-for-torch/.

[34] Adrian Cho. „Quantum or not, controversial computer yields no speedup“. In: *Science* 344.6190 (2014), pp. 1330–1331 (cit. on p. 46).

[35] Jin Young Choi and Chong-Ho Choi. „Sensitivity analysis of multilayer perceptron with differentiable activation functions“. In: *Neural Networks, IEEE Transactions on* 3.1 (1992), pp. 101–107 (cit. on p. 63).

[36] Dan C Cireşan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. „Mitosis detection in breast cancer histology images with deep neural networks“. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2013*. Springer, 2013, pp. 411–418 (cit. on p. 24).

[37] Carlos A Coello Coello, David A Van Veldhuizen, and Gary B Lamont. *Evolutionary algorithms for solving multi-objective problems*. Vol. 242. Springer, 2002 (cit. on p. 71).

[38] Carlos Coello Coello, Gary B Lamont, and David A Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007 (cit. on p. 19).

[39] Holk Cruse. „Neural Networks as Cybernetic Systems 3 rd and revised edition". In: *Neural Networks* 2 (2007), p. 2006 (cit. on p. 18).

[40] G. Cybenko. *Continuous valued neural networks with two hidden layers are sufficient*. 1988 (cit. on pp. 16, 79).

[41] Wim De Mulder, Steven Bethard, and Marie-Francine Moens. „A survey on the application of recurrent neural networks to statistical language modeling". In: *Computer Speech & Language* 30.1 (2015), pp. 61–98 (cit. on p. 18).

[42] Thomas Dean, Mark Ruzon, Michael Segal, et al. „Fast, accurate detection of 100,000 object classes on a single machine". In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 1814–1821.

[43] Gustavo Deco, Viktor Jirsa, AR McIntosh, Olaf Sporns, and Rolf Kötter. „Key role of coupling, delay, and noise in resting brain fluctuations". In: *Proceedings of the National Academy of Sciences* 106.25 (2009), pp. 10302–10307 (cit. on pp. 22, 61).

[44] Jia Deng, Wei Dong, Richard Socher, et al. „Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255 (cit. on p. 23).

[45] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. „Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks". In: *arXiv preprint arXiv:1506.05751* (2015) (cit. on p. 24).

[46] Travis Dierks, Bryan Brenner, and Sarangapani Jagannathan. „Neural network-based optimal control of mobile robot formations with reduced information exchange". In: *Control Systems Technology, IEEE Transactions on* 21.4 (2013), pp. 1407–1415 (cit. on p. 23).

[47] Marco Dorigo and Mauro Birattari. „Ant colony optimization". In: *Encyclopedia of machine learning*. Springer, 2010, pp. 36–39 (cit. on p. 40).

[48] Russ C Eberhart and James Kennedy. „A new optimizer using particle swarm theory". In: *Proceedings of the sixth international symposium on micro machine and human science*. Vol. 1. New York, NY. 1995, pp. 39–43 (cit. on p. 39).

[49] Jeffrey L Elman. „Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211 (cit. on p. 18).

[50] Evan Ackerman Erico Guizzo. *How Rethink Robotics Built Its New Baxter Robot Worker*. 2012. URL: `http://spectrum.ieee.org/robotics/industrial-robots/rethink-robotics-baxter-robot-factory-worker` (cit. on p. 23).

[51] G Bard Ermentrout, Roberto F Galán, and Nathaniel N Urban. „Reliability, synchrony and noise“. In: *Trends in neurosciences* 31.8 (2008), pp. 428–434 (cit. on pp. 22, 61).

[52] Scott E Fahlman. „An empirical study of learning speed in back-propagation networks“. In: (1988) (cit. on pp. 35, 36).

[53] Scott E Fahlman and Christian Lebiere. „The cascade-correlation learning architecture“. In: (1989) (cit. on p. 20).

[54] A Aldo Faisal, Luc PJ Selen, and Daniel M Wolpert. „Noise in the nervous system“. In: *Nature Reviews Neuroscience* 9.4 (2008), pp. 292–303 (cit. on pp. 21, 22, 61).

[55] Laurene Fausett. *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc., 1994 (cit. on p. 18).

[56] Emile Fiesler. „Comparative bibliography of ontogenic neural networks“. In: *ICANN'94*. Springer, 1994, pp. 793–796.

[57] Jon fingas. *Facebook and Google get neural networks to create art*. 2015. URL: `http://www.engadget.com/2015/06/20/facebook-and-google-ai-image-creation/` (cit. on p. 24).

[58] Keith Frankish and William M Ramsey. *The Cambridge Handbook of Artificial Intelligence*. Cambridge University Press, 2014 (cit. on p. 1).

[59] Stan Franklin. *Artificial minds*. MIT press, 1997.

[60] Osamu Fujita. „A method for designing the internal representation of neural networks“. In: *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*. IEEE. 1990, pp. 149–154.

[61] Ken-Ichi Funahashi. „On the approximate realization of continuous mappings by neural networks“. In: *Neural networks* 2.3 (1989), pp. 183–192 (cit. on pp. 15, 16).

[62] Stuart Geman, Elie Bienenstock, and René Doursat. „Neural networks and the bias/variance dilemma“. In: *Neural computation* 4.1 (1992), pp. 1–58 (cit. on p. 13).

[63] Bertrand Giraud, Lon Chang Liu, Christophe Bernard, and Herbert Axelrad. „Optimal approximation of square integrable functions by a flexible one-hidden-layer neural network of excitatory and inhibitory neuron pairs“. In: *Neural networks* 4.6 (1991), pp. 803–815.

[64] Abraham Goldberg, Harry M Schey, and Judah L Schwartz. „Computer-generated motion pictures of one-dimensional quantum-mechanical transmission and reflection phenomena“. In: *American Journal of Physics* 35.3 (1967) (cit. on p. 48).

[65]Otavio Good. *How Google Translate squeezes deep learning onto a phone*. 2015. URL: http://googleresearch.blogspot.bg/2015/07/how-google-translate-squeezes-deep.html (cit. on p. 24).

[66]Fr ed eric Gruau, Darrell Whitley, and Larry Pyeatt. „A Comparison between Cellular Encoding and Direct Encoding for Genetic Neural Networks“. In: () (cit. on p. 21).

[67]Ling Guo, Daniel Rivero, Julián Dorado, Juan R Rabunal, and Alejandro Pazos. „Automatic epileptic seizure detection in EEGs based on line length feature and artificial neural networks“. In: *Journal of neuroscience methods* 191.1 (2010), pp. 101–109 (cit. on p. 24).

[68]Mario Gutierrez, Jennifer Wang, and Robert Grondin. „Estimating hidden unit number for two-layer perceptrons“. In: *Neural Networks, 1989. IJCNN., International Joint Conference on*. IEEE. 1989, pp. 677–681.

[69]Martin T Hagan, Howard B Demuth, Mark H Beale, et al. *Neural network design*. Pws Pub. Boston, 1996 (cit. on p. 71).

[70]Mayssa Hajar, Amani Raad, and Mohamad Khalil. „Bearing and Gear Fault Detection Using Artificial Neural Networks“. In: () (cit. on p. 24).

[71]Norhamreeza Abdul Hamid, Nazri Mohd Nawi, Rozaida Ghazali, and Mohd Najib Mohd Salleh. „Improvements of Back Propagation Algorithm Performance by Adaptively Changing Gain, Momentum and Learning Rate“. In: *International Journal of New Computer Architectures and their Applications (IJNCAA)* 1.4 (2011), pp. 866–878 (cit. on p. 36).

[72]Barbara Hammer. *Learning with recurrent neural networks*. Vol. 254. Springer, 2007 (cit. on p. 18).

[73]Frank Harary. *Graph theory. 1969* (cit. on p. 9).

[74]Sherif Hashem. „Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions“. In: *Neural Networks, 1992. IJCNN., International Joint Conference on*. Vol. 1. IEEE. 1992, pp. 419–424 (cit. on p. 63).

[75]Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*. Vol. 3. Pearson Education Upper Saddle River, 2009 (cit. on pp. 1, 12, 15, 17, 27, 30–33, 36, 71).

[76]John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975 (cit. on p. 37).

[77]John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992 (cit. on p. 29).

[78] John J Hopfield. „Neural networks and physical systems with emergent collective computational abilities“. In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[79] Kurt Hornik. „Approximation capabilities of multilayer feedforward networks“. In: *Neural networks* 4.2 (1991), pp. 251–257 (cit. on p. 14).

[80] Kurt Hornik. „Some new results on neural network approximation“. In: *Neural Networks* 6.8 (1993), pp. 1069–1072 (cit. on p. 14).

[81] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. „Multilayer feedforward networks are universal approximators“. In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on p. 15).

[82] Rajat Monga Jeff Dean. *TensorFlow - Google's latest machine learning system, open sourced for everyone*. 2015. URL: `http://googleresearch.blogspot.bg/2015/11/tensorflow-googles-latest-machine_9.html`.

[83] Kam-Chuen Jim, C Lee Giles, and Bill G Horne. „An analysis of noise in recurrent neural networks: convergence and generalization“. In: *Neural Networks, IEEE Transactions on* 7.6 (1996), pp. 1424–1438 (cit. on pp. 22, 61).

[84] Wen Jin, Zhao Jia Li, Luo Si Wei, and Han Zhen. „The improvements of BP neural network learning algorithm“. In: *Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on*. Vol. 3. IEEE. 2000, pp. 1647–1649 (cit. on p. 35).

[85] Sung Hyun Jo, Ting Chang, Idongesit Ebong, et al. „Nanoscale memristor device as synapse in neuromorphic systems“. In: *Nano letters* 10.4 (2010), pp. 1297–1301 (cit. on p. 25).

[86] Tadashi Kadowaki and Hidetoshi Nishimori. „Quantum annealing in the transverse Ising model“. In: *Physical Review E* 58.5 (1998), p. 5355 (cit. on p. 45).

[87] Carl T Kelley. *Iterative methods for optimization*. Vol. 18. Siam, 1999 (cit. on p. 38).

[88] James Kennedy. „Particle swarm optimization“. In: *Encyclopedia of Machine Learning*. Springer, 2010, pp. 760–766 (cit. on p. 39).

[89] James Kennedy, James F Kennedy, Russell C Eberhart, and Yuhui Shi. *Swarm intelligence*. Morgan Kaufmann, 2001 (cit. on p. 39).

[90] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. „Optimization by simmulated annealing“. In: *science* 220.4598 (1983), pp. 671–680 (cit. on pp. 29, 43, 74).

[91] Will Knight. *This Robot Could Transform Manufacturing*. 2012. URL: `http://www.technologyreview.com/news/429248/this-robot-could-transform-manufacturing/` (cit. on p. 23).

[92] Pavel Kordík, Jan Koutník, Jan Drchal, et al. „Meta-learning approach to neural network optimization“. In: *Neural Networks* 23.4 (2010), pp. 568–582 (cit. on pp. 20, 35, 71, 72, 74).

[93] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on p. 23).

[94] Věra Krková. „Kolmogorov's theorem and multilayer neural networks“. In: *Neural networks* 5.3 (1992), pp. 501–506 (cit. on p. 14).

[95] Anders Krogh and John A Hertz. „Generalization in a linear perceptron in the presence of noise“. In: *Journal of Physics A: Mathematical and General* 25.5 (1992), p. 1135 (cit. on p. 22).

[96] SY Kung and JN Hwang. „An algebraic projection analysis for optimal hidden units size and learning rates in back-propagation learning“. In: *Neural Networks, 1988., IEEE International Conference on*. IEEE. 1988, pp. 363–370.

[97] Alan Lapedes and Robert Farber. *How neural nets work*. World Scientific, 1988 (cit. on p. 16).

[98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on pp. 20, 71).

[99] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. „Efficient backprop“. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48 (cit. on p. 17).

[100] William Leigh, Ross Hightower, and Naval Modani. „Forecasting the New York stock exchange composite index with past price and interest rate on condition of volume spike“. In: *Expert Systems with Applications* 28.1 (2005), pp. 1–8 (cit. on p. 35).

[101] Deng Li and D Yu. „Deep Learning: Methods and Applications“. In: *Foundations and Trends in Signal Processing, Now Publishers* (2014).

[102] Hong-Xing Li and ES Lee. „Interpolation functions of feedforward neural networks“. In: *Computers & Mathematics with Applications* 46.12 (2003), pp. 1861–1874 (cit. on p. 11).

[103] Shuai Li, Sanfeng Chen, Bo Liu, Yangming Li, and Yongsheng Liang. „Decentralized kinematic control of a class of collaborative redundant manipulators via recurrent neural networks“. In: *Neurocomputing* 91 (2012), pp. 1–10 (cit. on p. 23).

[104] Zachary C Lipton. „A Critical Review of Recurrent Neural Networks for Sequence Learning“. In: *arXiv preprint arXiv:1506.00019* (2015) (cit. on p. 24).

[105]William A Little. „The existence of persistent states in the brain“. In: *From High-Temperature Superconductivity to Microminiature Refrigeration*. Springer, 1996, pp. 145–164 (cit. on p. 21).

[106]Bo Liu, Ling Wang, Yihui Jin, and Dexian Huang. „Designing neural networks using PSO-based memetic algorithm“. In: *Advances in Neural Networks–ISNN 2007*. Springer, 2007, pp. 219–224 (cit. on p. 72).

[107]Carl G Looney. „Stabilization and speedup of convergence in training feedforward neural networks“. In: *Neurocomputing* 10.1 (1996), pp. 7–31 (cit. on p. 36).

[108]Liying Ma and Khashayar Khorasani. „A new strategy for adaptively constructing multilayer feedforward neural networks“. In: *Neurocomputing* 51 (2003), pp. 361–385 (cit. on p. 72).

[109]Vittorio Maniezzo. „Genetic evolution of the topology and weight distribution of neural networks“. In: *Neural Networks, IEEE Transactions on* 5.1 (1994), pp. 39–53 (cit. on pp. 72, 74).

[110]T Masters. „Practical Neural Network Recipes in C++“. In: (1995) (cit. on p. 16).

[111]Michalis Mavrovouniotis and Shengxiang Yang. „Training neural networks with ant colony optimization algorithms for pattern classification“. In: *Soft Computing* 19.6 (2014), pp. 1511–1522 (cit. on p. 42).

[112]Warren S McCulloch and Walter Pitts. „A logical calculus of the ideas immanent in nervous activity“. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cit. on pp. 3, 8).

[113]Kenneth J McGarry, Stefan Wermter, and John MacIntyre. „Knowledge extraction from radial basis function networks and multilayer perceptrons“. In: *Neural Networks, 1999. IJCNN'99. International Joint Conference on*. Vol. 4. IEEE. 1999, pp. 2494–2497 (cit. on p. 18).

[114]H Mei and Yong Wang. „Ant Colony Optimization for Neural Network“. In: *Key Engineering Materials*. Vol. 392. Trans Tech Publ. 2009, pp. 677–681 (cit. on p. 42).

[115]Michael Meissner, Michael Schmuker, and Gisbert Schneider. „Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training“. In: *BMC bioinformatics* 7.1 (2006), p. 125 (cit. on p. 39).

[116]Rui Mendes, Paulo Cortez, Miguel Rocha, and José Neves. „Particle swarms for feedforward neural network training“. In: *learning* 6.1 (2002) (cit. on p. 39).

[117]Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černockỳ, and Sanjeev Khudanpur. „Extensions of recurrent neural network language model“. In: *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE. 2011, pp. 5528–5531 (cit. on p. 25).

[118]Ian Millington and John Funge. *Artificial intelligence for games*. CRC Press, 2012 (cit. on p. 25).

[119]Marvin L Minsky and Seymour A Papert. *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*. MIT press Boston, MA: 1987 (cit. on pp. 3, 13).

[120]David J Montana and Lawrence Davis. „Training Feedforward Neural Networks Using Genetic Algorithms." In: *IJCAI*. Vol. 89. 1989, pp. 762–767 (cit. on p. 36).

[121]David E Moriarty and Risto Miikkulainen. „Forming neural networks through efficient and adaptive coevolution". In: *Evolutionary Computation* 5.4 (1997), pp. 373–399 (cit. on pp. 20, 72).

[122]David E Moriarty and Risto Mikkulainen. „Efficient reinforcement learning through symbiotic evolution". In: *Machine learning* 22.1-3 (1996), pp. 11–32 (cit. on pp. 20, 72).

[123]Alan F Murray and Peter J Edwards. „Synaptic weight noise during MLP learning enhances fault-tolerance, generalization and learning trajectory". In: *Advances in Neural Information Processing Systems* (1993), pp. 491–491 (cit. on pp. 22, 61).

[124]Nazri Mohd Nawi, Abdullah Khan, and Mohammad Zubair Rehman. „A new back-propagation neural network optimized with cuckoo search algorithm". In: *Computational Science and Its Applications–ICCSA 2013*. Springer, 2013, pp. 413–426 (cit. on p. 35).

[125]Seyed Taghi Akhavan Niaki and Saeid Hoseinzade. „Forecasting S&P 500 index using artificial neural networks and design of experiments". In: *Journal of Industrial Engineering International* 9.1 (2013), pp. 1–9 (cit. on p. 23).

[126]Bogdan Oancea and ŞTefan Cristian Ciucu. „Time series forecasting using neural networks". In: *arXiv preprint arXiv:1401.1333* (2014) (cit. on p. 23).

[127]Mustafa Oral, Emel Laptali Oral, and Ahmet Aydın. „Supervised vs. unsupervised learning for construction crew productivity prediction". In: *Automation in Construction* 22 (2012), pp. 271–276 (cit. on p. 46).

[128]Stjepan Oreski, Dijana Oreski, and Goran Oreski. „Hybrid system with genetic algorithm and artificial neural networks and its application to retail credit risk assessment". In: *Expert systems with applications* 39.16 (2012), pp. 12605–12617 (cit. on p. 23).

[129]Gaurang Panchal, Amit Ganatra, YP Kosta, and Devyani Panchal. „Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers". In: *International Journal of Computer Theory and Engineering* 3.2 (2011), pp. 332–337 (cit. on p. 14).

[130]HM Peixoto, RMS Cruz, AMG Guerreiro, AD Dória Neto, and AG D'Assunção. „A comparison of multilayer perceptrons training algorithms for the optimization of frequency selective surfaces". In: *Proceedings of 8th International Information and Telecommunication Technologies Symposium*. 2009 (cit. on p. 33).

[131]Roger Penrose. *The emperor's new mind: Concerning computers, minds, and the laws of physics*. Oxford University Press, 1999 (cit. on p. 48).

[132]Mirko Prezioso, Farnood Merrikh-Bayat, BD Hoskins, et al. „Training and operation of an integrated neuromorphic network based on metal-oxide memristors". In: *Nature* 521.7550 (2015), pp. 61–64 (cit. on pp. 22, 62).

[133]Nicolas Rashevsky. „Outline of a physico-mathematical theory of excitation and inhibition". In: *Protoplasma* 20.1 (1933), pp. 42–56 (cit. on p. 3).

[134]VJ Rayward-Smith, IH Osman, CR Reeves, and GD Smith. *Modern heuristic search methods. 1996* (cit. on p. 44).

[135]Russell Reed. „Pruning algorithms-a survey". In: *Neural Networks, IEEE Transactions on* 4.5 (1993), pp. 740–747 (cit. on p. 20).

[136]Martin Riedmiller and Heinrich Braun. „A direct adaptive method for faster backpropagation learning: The RPROP algorithm". In: *Neural Networks, 1993., IEEE International Conference on*. IEEE. 1993, pp. 586–591 (cit. on p. 35).

[137]Troels F Rønnow, Zhihui Wang, Joshua Job, et al. „Defining and detecting quantum speedup". In: *Science* 345.6195 (2014), pp. 420–424 (cit. on p. 46).

[138]Frank Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386 (cit. on pp. 3, 13).

[139]George A Rovithakis and Manolis A Christodoulou. *Adaptive control with recurrent high-order neural networks: theory and industrial applications*. Springer Science & Business Media, 2012 (cit. on p. 23).

[140]David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985 (cit. on pp. 18, 30).

[141]David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. „Learning representations by back-propagating errors". In: *Cognitive modeling* 5 (1988), p. 3 (cit. on p. 17).

[142]Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall, 2009 (cit. on p. 30).

[143]R Sathya and Annamma Abraham. „Comparison of supervised and unsupervised learning algorithms for pattern classification". In: *Int J Adv Res Artificial Intell* 2.2 (2013), pp. 34–38 (cit. on p. 27).

[144]Jürgen Schmidhuber. „Deep learning in neural networks: An overview“. In: *Neural Networks* 61 (2015), pp. 85–117 (cit. on p. 19).

[145]Michele Sebag. „A tour of Machine Learning: an AI perspective“. In: *AI Communications* 27.1 (2014), pp. 11–23.

[146]Michael L Seltzer, Dong Yu, and Yongqiang Wang. „An investigation of deep neural networks for noise robust speech recognition“. In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 7398–7402 (cit. on pp. 25, 61).

[147]Randall S Sexton, Robert E Dorsey, and John D Johnson. „Toward global optimization of neural networks: a comparison of the genetic algorithm and backpropagation“. In: *Decision Support Systems* 22.2 (1998), pp. 171–185.

[148]Danny Shapiro. *here is how deep learning will accelerate self-driving cars*. 2015. URL: `http://blogs.nvidia.com/blog/2015/02/24/deep-learning-drive/` (cit. on p. 25).

[149]Yuhui Shi and Russell C Eberhart. „Parameter selection in particle swarm optimization“. In: *Evolutionary programming VII*. Springer. 1998, pp. 591–600 (cit. on p. 39).

[150]Fernando M Silva and Luis B Almeida. „Acceleration techniques for the backpropagation algorithm“. In: *Neural Networks*. Springer, 1990, pp. 110–119 (cit. on p. 35).

[151]Il'ya Meerovich Sobol'. „On sensitivity estimation for nonlinear mathematical models“. In: *Matematicheskoe Modelirovanie* 2.1 (1990), pp. 112–118 (cit. on p. 63).

[152]Krzysztof Socha and Christian Blum. „An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training“. In: *Neural Computing and Applications* 16.3 (2007), pp. 235–247 (cit. on p. 42).

[153]Emilio Soria, Jose David Martin, and Paulo JG Lisboa. „Classical training methods“. In: *Metaheuristic Procedures for Training Neutral Networks*. Springer, 2006, pp. 7–36 (cit. on p. 12).

[154]Larry Squire, Darwin Berg, Floyd E Bloom, et al. *Fundamental neuroscience*. Academic Press, 2012 (cit. on p. 7).

[155]Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. „Dropout: A simple way to prevent neural networks from overfitting“. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on pp. 76, 77, 80).

[156] Kenneth O Stanley and Risto Miikkulainen. „Efficient evolution of neural network topologies“. In: *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*. Vol. 2. IEEE. 2002, pp. 1757–1762 (cit. on pp. 20, 72, 74).

[157] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. „Evolving adaptive neural networks with and without adaptive synapses“. In: *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*. Vol. 4. IEEE. 2003, pp. 2557–2564 (cit. on pp. 20, 72).

[158] Richard B Stein, E Roderich Gossen, and Kelvin E Jones. „Neuronal variability: noise or part of the signal?“ In: *Nature Reviews Neuroscience* 6.5 (2005), pp. 389–397 (cit. on pp. 22, 61).

[159] Peter N Steinmetz, Amit Manwani, Christof Koch, Michael London, and Idan Segev. „Subthreshold voltage noise due to channel fluctuations in active neuronal membranes“. In: *Journal of computational neuroscience* 9.2 (2000), pp. 133–148 (cit. on p. 21).

[160] Rainer Storn and Kenneth Price. „Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces“. In: *Journal of global optimization* 11.4 (1997), pp. 341–359 (cit. on p. 29).

[161] Fatma Taher and Rachid Sammouda. „Lung cancer detection by using artificial neural network and fuzzy clustering methods“. In: *GCC Conference and Exhibition (GCC), 2011 IEEE*. IEEE. 2011, pp. 295–298 (cit. on p. 24).

[162] Dirk Thierens and David Goldberg. „Convergence models of genetic algorithm selection schemes“. In: *Parallel Problem Solving from Nature — PPSN III: International Conference on Evolutionary Computation The Third Conference on Parallel Problem Solving from Nature Jerusalem, Israel, October 9–14, 1994 Proceedings*. Ed. by Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 119–129 (cit. on p. 74).

[163] Peter Tse and DD Wang. „A hybrid neural networks based machine condition forecaster and classifier by using multiple vibration parameters“. In: *Neural Networks, 1996., IEEE International Conference on*. Vol. 4. IEEE. 1996, pp. 2096–2100 (cit. on p. 23).

[164] Lefteri H Tsoukalas and Robert E Uhrig. *Fuzzy and neural approaches in engineering*. John Wiley & Sons, Inc., 1996.

[165] K URAHAMA. „Stability of neural networks and convergence of their sensitivity computation algorithms“. In: *Systems and computers in Japan* 21.8 (1990), pp. 82–88 (cit. on p. 36).

[166] Frans Van Den Bergh. „An analysis of particle swarm optimizers“. PhD thesis. University of Pretoria, 2006 (cit. on p. 40).

[167]MCW Van Rossum, Brendan J O'Brien, and Robert G Smith. „Effects of noise on the spike timing precision of retinal ganglion cells". In: *Journal of neurophysiology* 89.5 (2003), pp. 2406–2419 (cit. on p. 21).

[168]Ir LPJ Veelenturf and Ir SH Gerez. „Analysis of Recurrent Neural Networks with Application to Speaker Independent Phoneme Recognition". In: (1999) (cit. on p. 18).

[169]Davide Venturelli, Salvatore Mandrà, Sergey Knysh, et al. „Quantum optimization of fully-connected spin glasses". In: *arXiv preprint arXiv:1406.7553* (2014) (cit. on p. 46).

[170]John Von Neumann. „Probabilistic logics and the synthesis of reliable organisms from unreliable components". In: *Automata studies* 34 (1956), pp. 43–98 (cit. on p. 21).

[171]Izhar Wallach, Michael Dzamba, and Abraham Heifets. „AtomNet: A Deep Convolutional Neural Network for Bioactivity Prediction in Structure-based Drug Discovery". In: *arXiv preprint arXiv:1510.02855* (2015) (cit. on p. 24).

[172]Hui Wang, Hui Li, Yong Liu, Hui Li, and Sanyou Zeng. „Opposition-based particle swarm algorithm with Cauchy mutation". In: *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*. IEEE. 2007, pp. 4750–4756 (cit. on p. 40).

[173]Philip D Wasserman. *Advanced methods in neural computing*. John Wiley & Sons, Inc., 1993 (cit. on p. 40).

[174]Bogdan M Wilamowski and Richard C Jaeger. „Implementation of RBF type networks by MLP networks". In: *Neural Networks, 1996., IEEE International Conference on*. Vol. 3. IEEE. 1996, pp. 1670–1675 (cit. on p. 17).

[175]Tiantian Xie, Hao Yu, and Bogdan Wilamowski. „Comparison between traditional neural networks and radial basis function networks". In: *IEEE International Symposium on Industrial Electronics*. 2011 (cit. on p. 17).

[176]Xin Yao and Yong Liu. „A new evolutionary system for evolving artificial neural networks". In: *Neural Networks, IEEE Transactions on* 8.3 (1997), pp. 694–713 (cit. on pp. 19, 71, 74).

[177]Bing Yu and Xingshi He. „Training radial basis function networks with differential evolution." In: *GrC*. Citeseer. 2006, pp. 369–372 (cit. on p. 17).

[178]Jianbo Yu, Shijin Wang, and Lifeng Xi. „Evolving artificial neural networks using an improved PSO and DPSO". In: *Neurocomputing* 71.4 (2008), pp. 1054–1060 (cit. on p. 72).

[179]Wojciech Zaremba and Ilya Sutskever. „Reinforcement Learning Neural Turing Machines". In: *arXiv preprint arXiv:1505.00521* (2015).

[180]Xiaoqin Zeng, Daniel S Yeung, and Xuequan Sun. „Sensitivity analysis of multilayer perceptron to input perturbation“. In: *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*. Vol. 4. IEEE. 2000, pp. 2509–2514 (cit. on p. 63).

List of Figures

List of Tables

Colophon

This thesis was created in 2015 at the Institute of Information and Communication Technologies at the Bulgarian Academy of Sciences. The author is Kristina G. Kapanova (kkapanova@gmail.com, kapanova@parallel.bas.bg). The consultants were J.M. Sellier and I.T. Dimov

Declaration

I declare that the current PhD thesis contains original results obtained during a research conducted by myself (with the support and guidance of my scientific coordinators).

The results which were obtained, described and/or published by other scientists are duly cited in detail in the bibliography.

This thesis is not used towards the obtaining of a degree at another university or research Institute.

Sofia, November 15, 2016

Kristina Georgieva Kapanova

Contribution

The central contributions of this dissertation are from scientific and application perspective.

From a scientific aspect, the work had focused on the development of novel optimization algorithms. They have been based either on a quantum or evolutionary approach with the main objective to improve the function of the algorithm in situations where it reaches local minima, saddle points or a plateau and is unable to escape during its runtime.

- A novel post-learning algorithm was developed to assist the network to reach an improved weigths solution, once the training algorithm has completed. The running time of the algorithm was established at 32 iterations, achieving extremely low computational time and requiring minimal additional computational resources.

- A mathematical tool to quantitatively measure the influence of noise in the perturbed weights of the network was implemented.

- Introduced a novel hybrid genetic algorithm for the automatic evolution of neural network architectures. Several degrees of freedom for the design of the network and the algorithm are achieved at acceptable computational costs.

From application perspective a neural network programming environment was created in the C language. The software is utilized to select from several degrees of freedom in terms of number of layers, number of neurons per layer and as a whole, type of activation function per neuron, input/output files, number of iterations.

Appendix

The algorithms proposed here, were implemented and tested by a custom build software. The aim is to develop an artificial neural network able to fit a function, utilizing the proposed methods. The network was developed in the C programming language, with no graphical user interface. The code allows for the following options:

- Maximum number of neurons in the entire network, defined as define TOTAL_NEURONS 12
- Maximum number of layers in the entire network, defined as define TOTAL_LAYERS 4
- Maximum number of training samples, defined as define N_SAMPLES 3
- Maximum number of inputs per neuron, defined as define INPUTS 8
- Maximum number of iterations the network is running for, defined as define ITERATIONS 32

Furthermore, each neuron in every layer could be assigned different activation function. Currently, there are two implemented functions, defined in the following manner:

```
double activationFunction(int class,double x){
 if(class==TANH)      return(tahn(x));
 else if(class==EXP)  return(1-2/(exp(2*-x)+1);
 else (class==IDEN)   return(x);
}
```

One can select which function the network to calculate in the script by choosing the following:

```
// function menu options
char* functionMenu[] =
```

```
{
"    1. Square root",
"    2. Sine",
"    3. Power",
"    4. Parabola",
NULL
};

// calculates the currently selected function
double calculateFunction(double value1, double value2)
{
if(selectedFunction == 1) return sqrt(value1);
if(selectedFunction == 2) return sin(value1);
if(selectedFunction == 3) return pow(value1, value2);
if(selectedFunction == 4) return value1 * value1;
return 0;
}
```

We then create the network as following:

```
// network configuration
int nInputNeurons = 1;
char* filename = NULL;
int nSamples = 0;

// select the current function
selectedFunction = getMenu(functionMenu);
if(selectedFunction == 1) filename = "samples_sqrt.txt";
if(selectedFunction == 2) filename = "samples_sin.txt";
if(selectedFunction == 3) filename = "samples_pow.txt";
if(selectedFunction == 4) filename = "samples_parabola.txt";
if(selectedFunction == 3) nInputNeurons = 2; // for the Power function

// load the training samples
if(!filename) continue;
nSamples = loadSamples(filename, nInputNeurons);
if(nSamples < 1) continue;

// create the network
if(network) freeNetwork(network);
network = createNetwork(nInputNeurons, nHiddenNeurons, nSamples, samples);

// update the status
if(selectedFunction == 1) networkStatus = "Created Square root";
if(selectedFunction == 2) networkStatus = "Created Sine";
if(selectedFunction == 3) networkStatus = "Created Power";
if(selectedFunction == 4) networkStatus = "Created Parabola";
```

```
// report the initial iteration and MSE
printf("iteration = %d   MSE = %f\n", network->currentIteration, calculateMSE(netwo
}
```

The training of the network follows, by using Simulated Anealing as the training algorithm. Once the training is completed, one can choose to continue with the post-learning strategy:

```
// Post-learning selection of one with or without noise
for(k = 0; k < nIterations; k++)
{
addHiddenNoise(network, biasNoise, weightNoise);
}
```

In the event the algorithm has not found a better outcome, the swap function allows it to continue with the previous best function in the following manner:

```
// swaps two values
void swapValues(double* value1, double* value2)
{
double tmp = *value1;
*value1 = *value2;
*value2 = tmp;
}

// swapping of biases and weights with their copies (to restore the previous versio
void swapState(Network* network)
{
int i, j;
// swap the hidden layer
for(i = 0; i < network->hiddenLayer.nNeurons; i++)
{
// swap the bias with its copy
swapValues(&network->hiddenLayer.neurons[i].bias, &network->hiddenLayer.neurons[i].

// swap the weights with their copies
for(j = 0; j < network->inputLayer.nNeurons; j++)
{
swapValues(&network->hiddenLayer.neurons[i].weights[j], &network->hiddenLayer.neuro
}
}
```

To add noise in the hidden layer, we call the following function:

```
void addHiddenNoise(Network* network, double biasNoise, double weightNoise)
{
int i, j;

// add some noise to the hidden layer
for(i = 0; i < network->hiddenLayer.nNeurons; i++)
{
// bias noise
if(biasNoise != 0) network->hiddenLayer.neurons[i].bias += randomDouble() * biasNoise;

// weight noise
if(weightNoise != 0)
{
for(j = 0; j < network->inputLayer.nNeurons; j++)
{
network->hiddenLayer.neurons[i].weights[j] += randomDouble() * weightNoise;
}
}
}
}
```