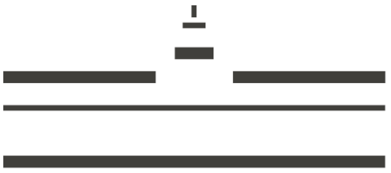


Westfälische
Wilhelms-Universität
Münster

Fach Informatik

Advancing Evolution of Artificial Neural Networks through Behavioral Adaptation

Inaugural-Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften im Fachbereich Mathematik und Informatik
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von
M.Eng. Kristina Davoian
aus Tbilisi, Georgien

2011

Dekan: Prof. Dr. Matthias Löwe, Westfälische Wilhelms-Universität Münster
Erster Gutachter: Prof. Dr. Wolfram-M. Lippe, Westfälische Wilhelms-Universität Münster
Zweiter Gutachter: Prof. Dr. Xiaoyi Jiang, Westfälische Wilhelms-Universität Münster
Tag der mündlichen Prüfung: 20. Januar 2012
Tag der Promotion: 20. Januar 2012

"A woman uses her intelligence to find reasons to support her intuition."

Gilbert K. Chesterton

Abstract

Evolutionary algorithms (EAs) have found wide application in Artificial Neural Networks (ANNs) learning due to their ability to outperform gradient descent methods in terms of adaptability to the dynamic environment defined by a considered task, resistance to local minima trapping, faster convergence to optima and simplicity of implementation. Besides, EAs are more robust, as they do not depend on gradient information, which makes them applicable to problem domains where this information is unavailable. In addition, EAs along with weights training can be used to evolve other parameters of ANNs, i.e., architectures, learning rules, etc.

In spite of advantages, not all types of EAs are efficient in ANNs' learning. Previous studies showed that crossover-based EAs, i.e., genetic algorithms (GAs), do not perform well and often are destructive. This is caused by inability of their primary search operator (crossover) to deal with the permutation problem faced by evolutionary training in ANNs. In contrast to GAs, the mutation-based EAs, i.e., evolutionary programming (EP) and evolution strategies (ES) do not utilize crossover and thus, can reduce the negative impact of the permutation problem.

This thesis is concerned with learning in ANNs and introduces a new mutation-based evolutionary algorithm for evolving ANN parameters, referred to as the network-weight-based evolutionary algorithm (NWEA). The main contribution of this thesis is to involve other mechanisms of nature in computational evolution. The goal of this research is to develop a learning strategy which incorporates knowledge of an individual's position in the search space, its fitness, and ANN topology in the modification mechanism. The key idea behind NWEA is to perform behavioral adaptation alongside with structural adaptation. The involvement of behavioral adaptation provides interconnection between individuals and the environment, which enables to consider information about individual's habitat in the evolution process.

The modification mechanism of NWEA utilizes both genotype and phenotype information while evolving individuals. Genotype information is represented by an individual's error. Phenotype information is included in the component, called the network weight (NW), which describes an ANN's internal structure and depends on a total number of hidden layers and the average number of hidden neurons. The relationship between NW and a corresponding ANN topology is defined by the Fermi-Dirac-like function.

In order to evaluate the proposed algorithm as well as to investigate its features, NWEA was applied to evolve ANNs for several benchmark problems. The experimental studies showed that NWEA produces ANNs of small sizes and good generalization ability, which generally outperform ANNs constructed with the existing approaches.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Prof. Dr. Wolfram-Manfred Lippe, for his unstinting support, encouragement, and fruitful discussions. His insights and suggestions played an invaluable role in my dissertation. He has a great sense of humor and is one of the most positive people I know. It is my honor and fortune to have such a great advisor.

I would like to extend my very special thanks to Alexander Reichel for his invaluable discussions and suggestions on my research work. He gave me many insightful comments in particular, on the physical part of the dissertation.

My sincere thanks to Prof. Dr. Xiaoyi Jiang, who has reviewed my dissertation, and the committee members. Many thanks to Dr. Sven F. Crone, Dr. Robert Stahlbock, and Dr. Jeffrey Wallace for their helpful comments, suggestions and discussions.

I am very grateful to Guida Gentes and Hedi Hoff-Weikart for their assistance and support. I thank members of our research group for friendly and welcoming environment.

The research presented in the dissertation has been completed while I have been working at Eucon GmbH. I would like to thank my management, Maurice M. Oosenbrugh and Marcel R. Oosenbrugh for their continuous support and encouragement provided to complete this work. I also thank my numerous colleagues at Eucon, especially - Ulrich Bertelsmeier, Maria Slavova, Dr. Mirko Dobberstein, Sigrid Gathmann, Marcus Ristau, Thomas Prümer, André Ehsner, Dimitri Kolke and Genia Artarov - for their help and company.

Many thanks to all my dearest friends, especially to Dr. Verena Klamroth, Pavel Leuss, Dr. Harut Harutyunyan, Ioana Florea, Erzsébet Nagy and Hanna Gershuni for being always on my side.

Last but not least, I give my dedicated thanks to my parents and grandmother for their love, support, and understanding. This work would not be possible without their encouragement and help. Many thanks to my Godfather for supporting and believing in me.

Words cannot express my most sincere thanks to my beloved sister, Ruzana, for her unlimited support and encouragement. You are the best sister ever and the best friend of mine.

This dissertation is dedicated to my mom and grandmother.

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BP	Back-Propagation
CEP	Classical Evolutionary Programming
CNWEA	Combined Network Weight-based Evolutionary Algorithm
EA	Evolutionary Algorithm
EANN	Evolutionary Artificial Neural Network
EP	Evolutionary Programming
ES	Evolution Strategy
FEP	Fast Evolutionary Programming
FFANN	Feed-forward Artificial Neural Nwtwork
GA	Genetic Algorithm
GP	Genetic Programming
IFEP	Improved Fast Evolutionary Programming
MEP	Mixed Evolutionary Programming
MNWEA	Mixed Network Weight-based Evolutionary Algorithm
MLP	Multi-layered Perceptron
MSE	Mean Squared Error
NW	Network Weight
NWEA	Network Weight-based Evolutionary Algorithm
PEA	Parallel Evolutionary Algorithm
PEANN	Parallel Evolutionary Artificial Neural Network
PNWEA	Parallel Network Weight-based Evolutionary Algorithm
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network

To my mother and grandmother

Chapter 1

Introduction

"Begin at the beginning and go on till you come to the end: then stop."

Lewis Carroll, ''Alice's Adventures in Wonderland."

Looking around we see populations of life forms that live in symbiosis with each other. The biological species of these populations live, reproduce offspring, interact with the environment and adapt themselves to their habitat. Despite the variety of living organisms (genotypic, phenotypic and behavioral), they all change over generations as a result of the *evolution process*, called *adaptation*, that occurs in nature and aims at producing individuals better suited to the environment.

According to the theory of evolution, individuals with favorable traits determined by genotype are more likely to survive and reproduce ("survival of the fittest") [23]. These traits define individual's ability to adapt to the environment, i.e., an individual's fitness. Adaptation is a continuous process based on the concept that populations of individuals change over time as a result of natural *selection*. Generally, biological adaptation is not evident in everyday life; however, we can observe its result in adaptive traits that can be *structural* and *behavioral*. While structural adaptations are genetically-based physical features, behavioral adaptations represent the ability of species to learn. In practice, behavioral adaptations determine a form of adaptation that occurs in everyday life and represents a process of an individual's adjustment to its habitat. In this process, an individual collects knowledge from the environment and involves it while making decisions and taking actions.

This dissertation is concerned with evolution and learning in *evolutionary artificial neural networks* (EANNs) [16, 92, 153, 155]. The essence of EANNs is the combination of

two artificial intelligence (AI) algorithms, inspired by natural processes, i.e., *evolutionary algorithms* (EAs) [5, 8, 92] and *artificial neural networks* (ANNs) [122, 123], where EAs are employed to design and/or train an optimal network for a given task.

EAs are inspired by the principles of biological evolution and work with a population of individuals, where each individual is a candidate solution for a problem. EAs simulate mechanisms of natural selection to perform an adaptation process and find an optimal solution. However, adaptations performed in EAs represent a simplified form of biological structural adaptations, while behavioral adaptations are not considered at all. This is mainly caused by the lack of clear distinction between *genotype* and *phenotype*. In nature, phenotype is a result of the interaction between genotype and the environment. In EAs, the environment is determined by a fitness function, and an EA often does not make any assumption about underlying fitness landscape. Hence, the evolved adaptive traits in EAs represent structural adaptations, while lack of interconnection between genotype and the environment makes it impossible (and unnecessary) to perform behavioral adaptations.

Evolution in EANNs is another fundamental form of adaptation in addition to *learning*. The main advantage of EANNs is their adaptability to the dynamic environment, i.e., to the environment as well as changes in the environment. Evolution is accomplished by an EA in order to adjust different ANN parameters, such as connection weights, architectures, etc. In contrast to EAs, EANNs have clear distinction between these two notions: a genetic representation of parameters is *genotype*, and an actual ANN with a given topology and a full set of connection weights is *phenotype*. During the evolution process, some phenotypic characteristics determine the optimization criteria in regard to evolving parameters. Hence, the adaptation process becomes more complicated, as the environment is defined not only by the conditions of a solving task, but also by ANN parameters (either fixed or evolved). For example, evolution of connection weights is carried out in the environment of a predefined ANN architecture; that means, a given ANN topology is another optimization criteria in addition to the fitness function. The distinction between genotype and phenotype in EANNs gives an opportunity to perform behavioral adaptation. In other words, it becomes possible to *involve knowledge of the environment defined by the phenotypic characteristics of a network* in the evolution process.

In line with this inspiration, this thesis introduces a novel approach for evolving ANNs, referred to as the *network-weight-based evolutionary algorithm* (NWEA) [26, 27] that involves *both genotype and phenotype information* in the evolution process. NWEA is a mutation-based EA, i.e., relies on mutation as a main search operator. The mutation mechanism of NWEA consists of three components: phenotypic, genotypic and random,

that modify object parameters. The phenotypic and genotypic components represent *individual-level adaptive parameters* responsible for adaptation in NWEA. Phenotype information is encapsulated in the new component, called the *network weight* (NW), which describes an *internal ANN topology*. Another novelty of NWEA is a type of genotype information represented in the modification mechanism. Genotype information is introduced by an individual's error, which shows the worth of the individual regarding the solving task. Thus, these two components contain informative knowledge about an individual's position in the search space and its environment. Their inclusion in the mutation approach enables the algorithm to consider the most perspective regions of the search space in order to obtain solutions of good quality.

1.1 Research Question

The objective of this thesis can be formulated in the following research questions:

- What kind of information should be incorporated in the mutation strategy to adapt mutation strength to the characteristics (e.g., a position regarding an optimum, a network structure) of a particular individual?
- What is the benefit of performing behavioral adaptation alongside with structural adaptation? In other words: how does inclusion of phenotype information affect evolution of ANNs?

The goal of our research is to develop an evolutionary algorithm for ANNs' design and training, that uses both phenotype and genotype information in determining the mutation step size and biasing evolution towards optima. The major challenge we faced when constructing NWEA was to:

- Define the generalized equation that describes the dependency between the phenotypic component and an ANN architecture.

In order to evaluate the efficiency of the proposed algorithm and the quality of NWEA-developed ANNs, analytical and experimental studies in this thesis are provided to:

- Study the generalization ability of the evolved ANNs, i.e., ability to find good solutions on both training and testing sets;
- Establish the algorithm's ability to develop compact architectures;

- Estimate the average rate of successfully improved individuals, i.e., the rate of successful mutations;
- Determine convergence speed, i.e., the average generation at which the optimal ANN is obtained;
- Investigate the role of non-evolved internal ANN parameters, e.g., the type of activation function, in evolution of ANNs;
- Explore the role of internal NWEA-parameters, e.g., the type of genotype information and distribution of random values, in evolution of ANNs;
- Study the effect of parallelization on the generalization in parallel EANNs.

1.2 Scope of the Thesis

Despite the variety of ANNs and their learning paradigms, the research in this dissertation is limited to examining feed-forward ANNs [92] with the proposed NWEA algorithm, and is concerned with the supervised learning they maintain. Unsupervised and reinforcement learning techniques are not considered in this thesis.

Evolution of ANNs with NWEA is performed at following levels: evolution of connection weights in the environment of fixed architectures and, the simultaneous evolution of connection weights and architectures. The evolution of connection weights is applied to the simple task, such as XOR, in order to investigate the basic characteristics of the proposed learning strategy. Although the experiments in this part are made for a limited number of predefined ANNs, the generalized equation (4.10) enables applying NWEA to any ANN topology. Generalization of NWEA-evolved ANNs is studied on the complex benchmark problems, such as classification and prediction tasks [115], through the evolution of both connection weights and architectures.

In addition to EANNs, this thesis examines parallel EANNs (PEANNs) [28, 120, 154] for the purpose of observing the effect of the interconnection between parallel populations on the quality of developed ANNs.

1.3 Contributions

The main contribution of this dissertation is to develop an EANN learning algorithm that performs behavioral adaptation alongside with structural adaptation, i.e., involves both phenotype and genotype information in the evolution process to ensure favorable

adjustments. Natural process of behavioral adaptation is simulated in EANNs through individuals' adjustment to their habitat represented by an ANN topology and conditions of a solving task. The main objective of our research is to investigate the impact of including phenotype information in the modification mechanism of NWEA on the quality of the evolved networks. The key ideas that allow us to achieve this goal are:

- Incorporate informative knowledge about an individual in the adjustment mechanism.
- Perform evolution considering both phenotype and genotype information.

The following points present the main components of our solution:

- NWEA: The proposed solution for EANN learning is a novel algorithm, referred to as the network-weight-based evolutionary algorithm (NWEA) [26, 27]. NWEA represents a mutation-based approach, which relies on mutation as a primary reproduction operator and does not include other genetic operators (such as crossover, local search operators, etc.). One distinct feature of NWEA is that in addition to structural adaptation it performs behavioral adaptation, i.e., considers interaction between an individual and the environment. In contrast to the existing evolutionary learning algorithms, the modification mechanism of NWEA involves both genotype and phenotype information to improve individuals at each stage of evolution. Genotype information is represented by an individual's error, which estimates an individual's position in the search space concerning the optimum. Phenotype information is incorporated in a value, called the network weight (NW), which implicitly describes an ANN topology. During the evolution, an ANN architecture not only represents the optimization parameter, but also determines, alongside with the fitness function, the environment to which an optimal set of connection weights has to be found. Thus, NWEA comprises information of an individual and the environment, and exploits this information in the determination of the mutation strength.
- Phenotypic component: A critical challenge in NWEA is defining the phenotypic component, i.e., establishing essential parameters it is based on and determining a function that describes the dependency of NW values on a particular ANN topology. We tackled this problem by providing numerous tests with different NW values to train ANNs with different predefined topologies, and then studied the relation between a particular ANN structure and a corresponding NW value. Our extensive analysis showed that the phenotypic component NW depends on the internal ANN architecture, i.e., on a total number of hidden layers and the average

number of hidden neurons per each layer, and is distributed by the Fermi-Dirac-like function. The presented generalized equation allows calculating NW values for ANNs of any complexity.

- Genotypic component: Another component of NWEA is the genotypic component, which is fitness-based and describes the worth of an individual. In EANNs, fitness is a value inversely proportional to the network's output error. The error is defined by the error function between actual and desired outputs over all training examples. The relationship between fitness and error is obvious: the higher the error, the lower the fitness and vice versa. The incorporation of the error in NWEA enables the algorithm to adjust the mutation strength depending on the individual's position in the search space regarding the optimum.

- Implementation and analysis: The experimental studies were provided to examine NWEA on various real-world problems as well as to indicate the features of the proposed learning strategy. For this reason several sets of experiments at different evolutionary levels were carried out. The first set of studies was conducted for the XOR problem considering the evolution of connection weights in the environment determined by fixed ANN topologies. The goal of the experiments was to study the internal characteristics of NWEA, such as the average rate of successful mutations, ability to obtain solutions of the high accuracy, as well as to determine NWEA's convergence speed [26, 27, 34] in evolving the optimal network. Our results were compared with those of classical mutation-based approaches. The second set of experiments was performed on the prediction and classification problems of different complexity for the purpose of investigating generalization ability of NWEA-evolved ANNs [33]. The experiments were conducted for the simultaneous evolution of connection weights and architectures. Further, we extended studies by examining the role of initially determined parameters on the performance of NWEA-evolved ANNs. In particular, the impact of ANN activation function type [30], as well as benefits of using different distribution functions [31, 32] with respect to EANNs generalization ability was studied. Finally, we examined generalization in parallel EANNs (PEANNs) [28, 29]. Two parallelization schemes were applied to parallelize NWEA: the well-known migration parallelization scheme, which enables interconnections between different populations, and a novel parallelization scheme, called the *migration-strangers* [25], which in addition to interconnection maintains diversity of population by extending search space during the evolution.

1.4 Thesis Overview

The rest of the thesis is organized as follows. Chapter 2 discusses fundamental concepts. In particular, it describes evolutionary algorithms (EAs), artificial neural networks (ANNs), the application of EAs to evolve ANNs, i.e., evolutionary artificial neural networks (EANNs) and parallelization of EANNs, i.e., parallel evolutionary artificial neural networks (PEANNs).

Chapter 3 presents well-known mutation-based EAs that found wide application in ANN learning, i.e., classical evolutionary programming (CEP) [45, 48] and fast evolutionary programming (FEP) [161, 166]. Following that, it discusses two combinations of CEP and FEP, referred to as the improved fast evolutionary programming (IFEP) [157, 164] and the mixed fast evolutionary programming (MEP) [164].

Chapter 4 introduces NWEA, a novel learning algorithm for evolving ANN weights and architectures. The chapter starts with motivation for incorporating phenotype information in NWEA, discussing the main idea behind NWEA and information necessary to manage the mutation strength. The solution is based on the assumption that alongside with genotype information, phenotype information represents another powerful knowledge that allows an additional form of adaptation during the evolution. Further, we describe the empirical process of estimating the phenotype component called the *network weight* (NW) and derive the equation that determines the dependence of the NW values on ANNs' internal structures.

Chapter 5 examines the proposed NWEA algorithm from different perspectives. The experimental part consists of several sets of experiments. The first set of experiments studies the performance of ANNs evolved in the environment of predefined topologies. In the second set of tests, evolution of EANNs was provided to solve a number of benchmark problems and observed under simultaneous adjustment of both connection weights and architectures. The obtained results of NWEA-evolved ANNs were compared with those of existing evolutionary and non-evolutionary ANN learning algorithms. Further, we investigate the impact of different activation functions and distributions of random values on generalization in NWEA-evolved ANNs. Finally, generalization in PEANNs evolved with the parallelized NWEA is investigated.

Chapter 6 resumes the conclusions of this dissertation and proposes directions for future work.

Chapter 2

Fundamental Concepts

"If you don't know where you are going, you will probably end up somewhere else."

Lawrence J. Peter

This chapter provides a brief overview of fundamental concepts related to this thesis research. More specifically, it introduces theoretical background of AI methodologies necessary to understand this thesis, such as evolutionary algorithms (EAs), artificial neural networks (ANNs) and a special class of ANNs evolved with EAs and referred to as evolutionary artificial neural networks (EANNs). In addition to that, this chapter discusses the effect of parallelization in parallel evolutionary algorithms (PEAs) and parallel evolutionary artificial neural networks (PEANNs).

The rest of this chapter is organized as follows. Section 2.1 presents general concept and framework of EAs and their main classes, i.e., genetic algorithms (GAs, Section 2.1.1), evolution strategies (ES, Section 2.1.2), evolutionary programming (EP, Section 2.1.3) and genetic programming (GP, Section 2.1.4). Section 2.1.6 describes motivation behind parallelization of serial EAs and provides an overview of parallelization schemes. Section 2.2 is devoted to ANNs. It presents the fundamental concepts of the theory of ANNs and discusses issues related to learning in ANNs. Sections 2.3 introduces EANNs, encoding schemes and denotes types of EAs beneficial in evolution of ANNs. The application of PEAs to evolve PEANNs is described in Section 2.3.5.

2.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) [5, 8, 92] are metaheuristic search methods, which are inspired by the principles of natural evolution and simulate the biological processes of

organisms in solving computer-based problems. Two prominent features distinguish them from other search methodologies: population-based search and, interconnections and information exchange between individuals in the population.

EAs work with a set of *individuals*, called the *population*, where each individual represents a candidate solution for a given instance. The initial population is generated at random. Each individual or *chromosome* contains information about parameters of a solving problem, which is stored into a sequence of *genes*. By analogy with the nature, information encoded in a chromosome is called *genotype*, while the individuals' representation in the environment is referred to as *phenotype*. EAs use two schemes for individuals representation, i.e., *binary* and *real-valued* representation. All individuals are evaluated based on their worth by the *fitness function* and each of them receives a value, called *fitness*. Fitness is a measure, that indicates how well a possible candidate fits a solving problem and affects the ability of an individual to survive and reproduce.

The search for the optimum is accomplished by the evolutionary process that takes place in the environment defined by the *objective function*.[1] Evolution is provided by the iterative application of *selection*, *crossover* and *mutation* operators, which simulate biological processes of selection, inheritance and variability.

The *selection* operator chooses individuals for reproduction.[2] EAs perform fitness-based selection, where the probability of an individual to be chosen as a parent depends on its fitness. The main selection mechanism used by EAs is the fitness proportionate selection or *roulette-wheel selection*. This method selects individuals using the "roulette", whose wheel contains one sector for each individual and the sector's size is proportional to the fitness of an individual. The main reason for utilizing fitness proportionate selection is to allow the genotype of the best individuals to propagate to the next generations. It is obvious, that the sectors of individuals with higher fitness are larger that those of individuals with low fitness. Thus, individuals with higher fitness have higher chances to be selected for reproduction than those with low fitness. As a result, offspring inherit the favorable traits of best individuals, while the properties of the worst individuals disappear from the next generations during the evolution process.

In spite of advantages, the fitness proportionate selection often bounds the entire search space with the local regions, where the current fittest individuals in the population are located. This may lead to the convergence to a local optimum. Figure 2.1 demonstrates

[1] Usually, the objective function is the same as the fitness function.

[2] It is worth noting, that selection is performed at different stages of evolution, depending on the type of EAs. For instance, in classical EAs, i.e., genetic algorithms (Section 2.1.1), the fitness-based selection determines individuals for reproduction. The other types of EAs, i.e., evolution strategies and evolutionary programming, apply selection after reproduction, while choosing individuals for the next generation (Sections 2.1.2 and 2.1.3, detailed in Section 3.2).

a problem's landscape with one global (G) and two local (L_1 and L_2) optimal solutions. Apparently, if the current best individuals are located around the point L_1, the algorithm might trap into the local optimum, because individuals with low fitness have less chances to be selected and reproduce. In order to increase diversity in the next generations, the rank and the tournament selection methods have found wide application in EAs. The *rank selection* assigns a numerical rank to each individual based on its fitness, i.e., the individual with the lowest fitness in the population has the lowest rank (1) and the fittest individual has the highest rank. Individuals are chosen according to their rank rather than to the absolute differences in the fitness. The *tournament selection* randomly selects a group of individuals from the population and compares them. The best out of the group becomes a parent. Although the rank and tournament selections are different methods, Goldberg and Deb in [60] mentioned that linear ranking and 2-tournament associate almost the same probabilities with the individuals in the population.

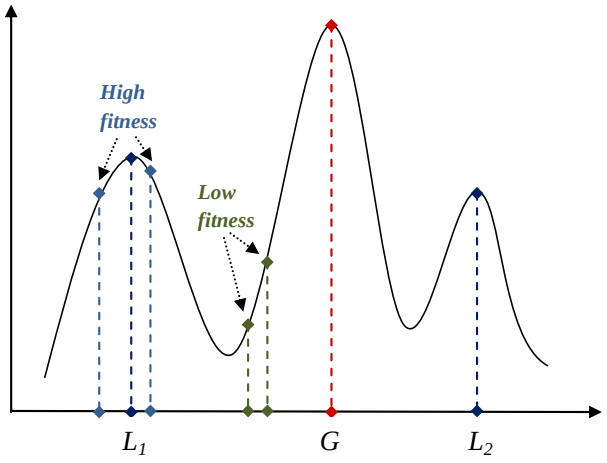


FIGURE 2.1: An example of problem's landscape with one global and two local optima.

At the next stage the selected individuals reproduce offspring. Depending on the type of EAs, selected individuals undergo *crossover* or *mutation* procedures to create new individuals. The *crossover* operator recombines genetic material of two parental individuals to create one or two offspring. Generally, the crossover operator divides the parental chromosomes into two or more segments, swaps them and thus forms new individuals (n-point crossover [75]). However, the type of crossover strongly depends on the solving task. For instance, it is impossible to swap genetic information of parents while solving the traveling salesman problem (TSP) [90], where swapping segments of parental individuals may generate offspring with duplicate cities.[3] In this case alternative crossover operators, e.g., greedy crossover [61], are used.

[3]For TSP, EAs traditionally use the real-valued chromosome representation, where each gene in a chromosome is the number of a city and each chromosome represents one possible tour.

In contrast to crossover, which produces offspring by exchanging genetic material of parental individuals, *mutation* creates new individuals by changing genotype of a particular parental chromosome. More specifically, the mutation operator randomly modifies genes in a chromosome and thus, creates new genetic material to be examined at the next generation. The performance of the mutation operator depends on the individuals' representation. For binary representation, it is conducted simply by changing the current value of a gene to its opposite. For real-valued representation advanced mutation techniques, such as Gaussian [45] and Cauchy [161] mutations, are applied.

The processes of selection, crossover and mutation are repeated until the offspring population of the same size as the parental one is formed. The creation of the new population indicates one evolutionary cycle or *generation*. The evolution process iterates until some stopping criteria, e.g., a predefined number of generations or computational time limit are reached. Figure 2.2 outlines a typical cycle of an EA.

```
Generate an initial population at random;
t = 0; // Iteration number
REPEAT {
    Evaluate fitness of each individual in the population;
    Select individuals for reproduction;
    Modify selected individuals by means of evolutionary
    operators;
    Create new (offspring) population;
    t = t + 1;
    }
UNTIL Stopping criteria are satisfied
```

FIGURE 2.2: A typical cycle of an evolutionary algorithm.

Despite being simplified models of natural processes, EAs are effective search methods because they do not make any assumption about the fitness landscape and are able to explore perspective regions of the search space. However, they do not guarantee the global optimum finding but instead obtain potentially good solutions. There exist four main methodologies of EAs: genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. Each methodology has numerous modifications, which employ different genetic operators and parameter settings. These modifications are problem dependent, as there exists no universal algorithm that can solve all problems.

2.1.1 Genetic Algorithms

Genetic algorithms (GAs), developed by John Holland [75, 76], represent the most popular type of EAs. Similar to the other evolutionary methods, they maintain basic mechanisms of nature, i.e., inheritance and variability, and follow the principle "survival of the fittest", described by Charles Darwin in his work *On the Origin of Species* [23].

GAs are distinguished from other classes of EAs by emphasizing genetic evolution and considering recombination as the primary reproduction operator and mutation as a background operator [57–59]. More correctly, mutation is used not as a searching operator, but rather as a mechanism adding a small randomness. Analogously to nature, mutation in GAs occurs with low probability.[4]

Traditional GAs use binary stings to represent individuals and apply classical crossover and mutation operators. However, modern GAs tend to utilize real-valued representations, since it reduces the length of the genetic string, which is especially important for solving complex problems with many parameters. In contrast to the other EAs, GAs apply all genetic operators in the classical order: first, they select individuals for reproduction, then produce new individuals by applying crossover and finally, modify offspring by means of the mutation operator. After these steps offspring individuals are included in the new population. A typical cycle is similar to that shown in Figure 2.2.

Although GAs ensure the finding of appropriate solutions, they are less resistant from the local optimum trapping, compared to the other types of EAs, particularly when an optimization problem has a complex landscape. To cope with this shortcoming, multiple runs of GAs are executed and then the best solution over all runs is accepted as a solving solution. An alternative method is to parallelize GAs (see Section 2.1.6), and then select the best individual over all parallel populations.

2.1.2 Evolution Strategies

Evolution strategies (ES), introduced by Ingo Rechenberg [118, 119], belong to a class of evolutionary algorithms, where mutation is considered as the key search operator. They traditionally use the real-valued representation scheme and evolve new populations by iterative application of the selection and mutation operators.

Initially, ES worked with a population of size one, i.e., population of one individual. The individual was modified by adding a Gaussian random value to each component of a real-valued vector. Further, offspring was competed with the parent and the best

[4] The ratio of mutation is usually not higher than 5%.

one was selected as a winner to continue the evolution. Later, Hans-Paul Schwefel [133–135] proposed using populations with more than one individual, as well as utilize recombination as a secondary operator.

At present, all existing ES methods can be classified on two main types: not elitist and elitist [7, 14]. To the first group belong (μ, λ)-ES, where μ is the population size and λ is a number of the produced offspring, $\lambda \geq \mu$. This strategy selects μ best individuals *from* λ offspring for the next generation. The elitist ES, referred to as ($\mu + \lambda$)-ES, allow the best μ individuals *from both* parental and offspring populations to survive.

2.1.3 Evolutionary Programming

Evolutionary Programming (EP) was developed by Lawrence J. Fogel *et al.* [51, 53] as an optimization method for artificial intelligence generation, and later completed by David B. Fogel [45, 48]. In comparison to GAs, EP puts emphasis on the phenotypic likeness between parental and offspring individuals rather than focuses on the genotype exchange between them. Therefore, EP does not differentiate between genotype and phenotype [47].

Initially, EP represented individuals in form of the universal finite-state machines [51, 53], which changed their state depending on incoming from the environment signals. New individuals were created by random changes in the state transition table. Nowadays EP is a method of evolutionary computation, which utilizes different types of representation, tailored to the problem domain [46–49, 52]. The mutation operator depends on the representation type and is generally self-adaptive (see Section 3.1). EP for the real-valued representation, referred to as the *classical evolutionary programming* (CEP), is discussed in Section 3.2.

Alike ES, EP considers mutation as the primary reproduction operator. After initialization, all individuals in the population are mutated in order to produce offspring. New offspring solutions are evaluated and the best individuals are selected from both parents and offspring by means of the fitness-based selection function. The solutions with low fitness are eliminated from the population. Such an approach makes EP an elitist method, because only individuals with high fitness are likely to survive for next generations. Similar to the classical ES, EP does not perform recombination, as the advanced mutation strategies are able to produce perturbations similar to recombination [139, 140].

2.1.4 Genetic Programming

Genetic Programming (GP), proposed by John Koza [84–87], is a machine learning method, used to qualify a computer to solve problems autonomously. The goal of GP is to generate on the basis of the training data a computer program that solves a given task.

As common with EAs, GP works with the population of individuals, where each individual represents a possible program for the solving task. The fitness of a chromosome is evaluated by a program's ability to perform a given task. Unlike other EAs, GP uses *tree structures* for individuals' representation. The crossover operator generates new solutions by swapping nodes (nodes with whole branch) between two parental ones. Similar to GAs, mutation is performed with the low probability; it replaces a node in offspring with a new randomly generated one. After evaluation, best offspring are selected for the next generation.

2.1.5 Global Convergence and Computational Complexity of EAs

The global convergence of EAs can be described as

$$\lim_{n\to\infty} P\{X_n \in S^*\} = 1$$
$$S^* = \{X | X \in S,\ F_X \leq F_Y\ \forall Y \in S\},$$

where P is the probability, X_n is the solution at time n, F_X is the fitness of X, S is the whole search space, and S^* is the set of global optima [154].

Previous studies showed that the global convergence of EAs can be established under some conditions [24, 38, 125, 126, 142]. As cited by [154], the global convergence depends on whether an EA is elitist or not. For elitist algorithms, it can be analyzed using Markov chains [125, 126]. The analysis of non-elitist EAs is more complicated. It has been shown that non-elitist EAs, e.g., the classical GAs (also called the canonical GAs) without elitism, "cannot converge to global optima regardless of the objective function and crossover operators used" [125, 154]. However, non-elitist algorithms may still converge to the global optima under certain conditions [126].

Although the global convergence is an important issue for EAs, it has less significance in practice, especially in designing of new EAs. From this point of view, more interesting is the computational complexity of EAs for a particular problem [111, 154]. Computational complexity is one of the most important issues in the analysis of algorithms. Unfortunately, there exist few studies devoted to this topic. The theoretical and experimental studies on the computational complexity of a GA in [70] indicated, that "GA which does

not specify the class of functions being optimized can make few claims regarding the efficiency of the genetic algorithm for an arbitrary fitness function".[5] In other words, it does not make sense to discuss computational complexity of an EA without pointing out the problem the algorithm is applied for.

2.1.6 Parallelization of EAs: Parallel Evolutionary Algorithms

As stated in Section 2.1, although EAs provide global search, they do not always guarantee global optima finding, mainly in complex problems. This issue is mainly caused by the homogeneity of populations at the latest stages of evolution. In other words, if all individuals in the population surround the local optimum, EAs face difficulties to produce offspring located in the region of the global optimum. Less resistent in this case are GAs; however, the elitist ES and EP also poorly maintain the diversity of individuals. One way to overcome this shortcoming is to execute EAs repetitively, and then select the best result out of the obtained solutions [20, 141]. Another way to cope with this problem is to evolve multiple initial populations simultaneously or, *parallelize* EAs. The parallelism is the key aspect of the parallel evolutionary algorithms (PEAs) [20].

The idea of evolving many populations at the same time gives a number of benefits. The main advantage of PEAs is their ability to consider different portions of search space, which makes the algorithm resistant to premature convergence to the local optimum and increases the probability of finding the global optimum. Additionally, PEAs enable to perform interconnections between evolved populations, i.e., exchange of genetic material and thus, maintain diversity of individuals in the populations. Finally, PEAs are advantageous in terms of computational speed, as they often requires less time to obtain good solutions than serial EAs.

The simplest parallelization method is to start several random populations at the same time, and consequently, execute multiple copies of the same EA. Each parallel process works with its own initial population, evolves it independently from the others and terminates optimization process when certain criteria are reached. The best solution is selected from the final populations of all concurrent processes. This method is called the *independent* parallelization scheme [20, 25, 141].

The independent approach does not perform any interconnection between parallel populations and is equal to choosing the best solution after multiple executions of the serial EA with different initial populations. However, later studies indicated that the *interconnection* between parallel populations is a decisive factor in increasing search space and

[5] In [70], an algorithm is considered to be efficient if it converges in polynomial time.

thus, probability of the global optimum finding [20, 108, 113, 147]. For this reason alternative parallelization strategies that enable exchange of the genetic material between concurrent populations have been proposed [20, 25, 108, 113, 141, 147].

In order to evolve parallel ANNs (PEANNs), in this thesis we apply two parallelization schemes with interconnections between parallel populations, which were shown to be efficient [25]: the *migration* strategy [147] and the new technique, referred to as the *migration-strangers* strategy [25]. Both schemes are described in detail in Section 5.3.1.

2.2 Artificial Neural Networks

An *artificial neural network* (ANN), first described by Warren McCulloch and Walter Pitts [99], is a mathematical model that consists of a set of interconnected processing elements, called *neurons* or *nodes*. ANNs were inspired by biological nervous systems, which propagate information through many nerve cells connected to each other with nerve fibres. Analogously to natural nervous systems, neurons in ANNs are connected to each other into a processing network, where each neuron receives, modifies and transfers signals. The links between neurons are called *connections*. Each connection has a corresponding value, called a *connection weight*, which is used to modify an incoming signal. Connection weights play the same role as synapses in biological systems and determine the strength of outgoing signals. In mathematical terms, an ANN is a directed graph with N number of neurons and C number of connections that link the neurons.

The outgoing signal of each neuron is calculated by means of the *activation function* (also referred to as the *transfer function*) expressed by the following equation:

$$y_i = f_i \left(\sum_{j=1}^{n} x_j w_{ij} - \theta_i \right), \tag{2.1}$$

where y_i is the output of a neuron i, x_j is the j-th input signal to a neuron i, w_{ij} is the connection weight between neurons i and j, θ_i is an internal threshold value[6] of a neuron i, i.e., value, which weighted sum of incoming signals needs to achieve to activate a neuron. Mathematically, the threshold value shows the position of the highest increment of the monotony increasing activation function.

The function f_i is usually chosen such that it adds nonlinearity in the work of a network, but does not change the result significantly. Most popular are S-shape (sigmoid) functions, such as logistic, hyperbolic tangent, Heaviside, etc. However, some linear functions, e.g., piecewise linear function, are also applicable to ANNs with non-gradient

[6] Threshold values (or biases) are usually implemented as connection weights with fixed input -1.

learning methods [30]. Figure 2.3 shows the simplified model of neuron, where $\bar{x} = (x_1, x_2, \ldots, x_n)$ is a vector of incoming signals, $\bar{w} = (w_1, w_2, \ldots, w_n)$ is a vector of connection weights, and f_{act} is the activation function, which calculates the outgoing signal y_i according to Eq. (2.1).

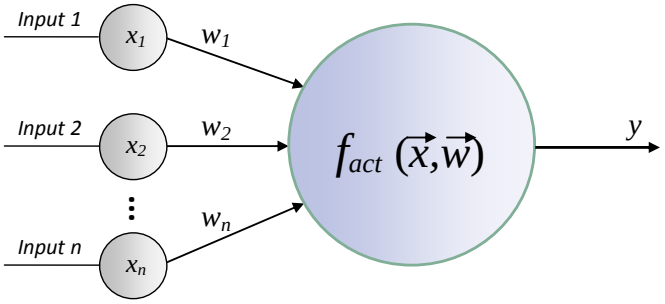


FIGURE 2.3: A model of an artificial neuron.

Neurons in an ANN are located in layers, which are classified into *input*, *hidden* and *output*. The input layer contains neurons that process input parameters of a solving problem. Neurons located in the output layer summarize the work of an ANN and output the results. The number of input and output neurons is determined by the considering task. The hidden layer contains neurons, which modify and process information from the input layer towards the output layer. An ANN can contain one or more hidden layers with various numbers of neurons in them. Hidden layers with hidden neurons build an ANN's internal structure, which is usually user-defined. Figure 2.4 shows an example of an ANN with one hidden layer. The network contains two input, three hidden and one output neurons.

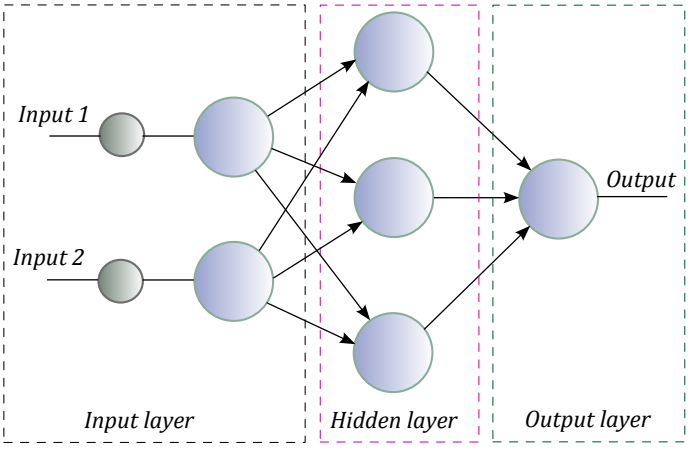


FIGURE 2.4: An ANN with two input neurons, one hidden layer with three neurons in it, and one output neuron.

All connections, directed to a particular neuron in an ANN have corresponding connection weights. The connectivity structure of the ANN, or its *topology*, can be schematically presented in form of the *connectivity matrix* $M = (w_{ij})_{N \times N}$, where rows and columns are identified by N neurons, and the matrix elements denote the connection weight values (see Section 2.3.4).

2.2.1 Classification of ANNs

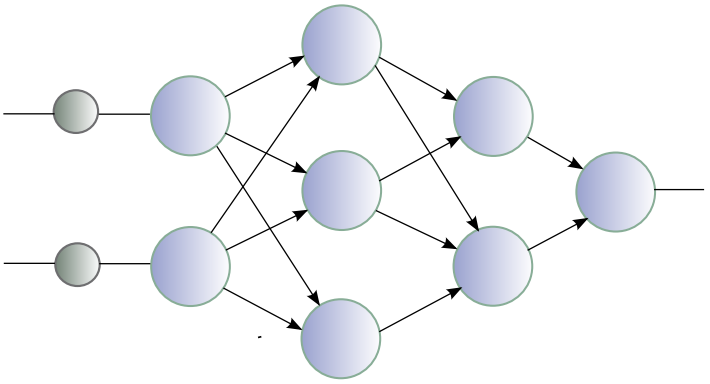
Despite the variety of neural network models, they are generally classified by the type of neuron interconnections and learning methods [92]. According to their connectivity, ANNs are divided into the feed-forward and recurrent. An ANN is *feed-forward*, if it transfers signals in *one direction*, i.e., all connections in the network are directed *from* input neurons *towards* output neurons and there is no connection between neurons located in the same layer. To this class of ANNs belong single- and multi-layer perceptrons [122, 123]. An ANN is *recurrent*, if it allows information propagation in *both directions*. That means, recurrent ANNs enable information propagation to the previous layers as well as connections between neurons located in the same layer. An example for a recurrent ANN is the Hopfield network [77]. An alternative definition for feed-forward and recurrent networks is given in [155]: "An ANN is feed-forward, if there exists a method which numbers all the nodes in the network such that there is no connection from a node with a large number to a node with a smaller number. All the connections are from nodes with small numbers to nodes with larger numbers. An ANN is recurrent if such a numbering method does not exist." Figure 2.5 shows examples of the feed-forward and recurrent ANNs; red colored connections in Figure 2.5(b) are recurrent connections. The classification of ANNs by learning methods is described in the next section.

2.2.2 Learning in ANNs

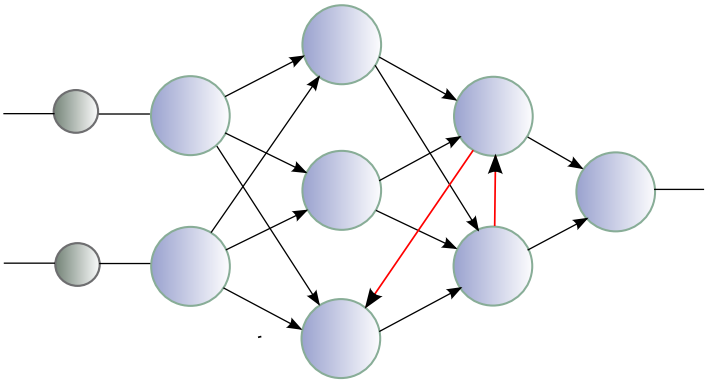
The central issue in the theory of ANNs is *learning* [71], also known as *training*, which is accomplished using examples, or the *training data*. Learning is a process of finding an optimal set of connection weights according to specific optimality criteria. It is performed by the iterative adjustment of the connection weights in an ANN so that a trained network can solve a given task. The essence of a learning algorithm is the *learning rule*, which determines how connection weights are changed.

Learning in ANNs is classified into supervised, unsupervised and reinforcement learning. All learning paradigms aim at optimizing the *objective function*,[7] which characterizes

[7]The objective function is also referred in literature to as the *cost function*.



(a) Feed-forward ANN: connections are directed towards the output neuron.



(b) Recurrent ANN: connections are directed in both directions.

FIGURE 2.5: Classification of ANNs: feed-forward and recurrent networks.

how well an ANN with a given set of connection weights performs a solving problem. *Supervised* learning [127] is provided by the direct comparison between the actual output of ANN and the expected correct output. Obtained *training error* is calculated by the *error function*.[8] Generally, supervised learning is formulated as a minimization of the error function, such as the mean squared error (MSE) or the root mean squared error (RMSE) between the expected and the actual outputs over all training examples.

Reinforcement learning [148] represents a special case of supervised learning, where the exact desired output is unknown. The network' actual output is evaluated based on the external information of whether it is correct or incorrect.

In contrast to supervised and reinforcement learning methods, the *unsupervised* learning (also referred to as *self-organized learning* [83]) does not have any information about the correct output. In this case learning is provided based on the correlations among input

[8]The error function is a type of the objective function, usually referred to a supervised learning.

data. The learning algorithm tries to derive likely features in the training examples and classify input data according to those features.

2.2.3 Overview of Supervised Learning Algorithms

The most popular ANN training algorithm is *back-propagation* (BP) [73, 127, 149]. It is a gradient descent method, that aims at minimizing the total MSE between actual and desired outputs of a network by employing gradient information. BP imposes limitations on a type of the ANN activation function; it requires the activation function to be differentiable. The BP algorithm works as follows: it calculates MSE and then, propagates the obtained error backwards in order to calculate weights' updates adjusted to a gradient-descent direction, i.e., backwards to gradient. The error calculation and weights' update processes are provided iteratively for all training examples.

Although BP has been successfully applied to various problems (as cited in [153, 155]), it has a number of drawbacks due to its gradient descent nature. Among shortcomings of BP are slow learning process, tendency to converge to the local minimum, and difficulties in training large ANNs, i.e., ANNs with large number of layers and neurons.

In order to overcome the drawbacks of gradient descent methods, EAs have been proposed for ANNs' training, which resulted in the development of a special class of ANNs, referred to as *evolutionary artificial neural networks* (EANNs, see Section 2.3). Besides their ability to perform the global search in complex surface, EAs are more robust, as they do not depend on gradient information of the error function, which makes them applicable to the problem domains where this information is unavailable or very costly to obtain. Unlike gradient descent algorithms, EAs do not need to calculate derivatives of the error function and thus, can work with non-differentiable or even non-continuous functions. EAs can be applied to train networks regardless of whether they are feed-forward, recurrent, etc. Moreover, EAs can be used not only to obtain the optimal set of connection weights, but also to optimize other ANN parameters, such as topology, learning rules, etc. The research in the field indicated that learning performed by EAs can be significantly faster than gradient training [65, 114, 137]. In addition to that, the evolutionary learning is proven to have better scalability in comparison to the BP training.

Alongside with evolutionary learning, hybrid learning algorithms that combine EAs and BP found wide application. The main idea of hybrid approaches is to apply an EA first for the purpose of finding the most efficient surface in the search space and then a gradient descent method in order find an optimum within that surface. The studies

on hybrid learning algorithms indicated high efficiency of hybrid learning in terms of generalization [9, 81, 91, 143, 153, 155].

2.2.4 Generalization in ANNs

The main goal of learning is *generalization*, i.e., ability of a network to report correct results on data that are not used in the training process. These data are called the *testing data*. When an ANN shows high accuracy on the training data, but performs poor on the testing data, it is said that a network is *overtrained* or *overfitted*. Thus, the aim of learning is not to learn the training data, i.e., achieve a minimal error on the training data, but instead to generalize training examples and discover underlined trends behind them. In other words, learning should avoid overfitting on a training data set.

2.3 Evolutionary Artificial Neural Networks

As described above, evolutionary artificial neural networks (EANNs) refer to a class of ANNs, where evolutionary search procedures, such as EAs, are used to evolve parameters of ANNs [15, 16, 79, 103, 153, 155, 160]. The key feature that distinguishes EAs from other learning methodologies is their adaptability to the dynamic environment as well as to changes in that environment.

Evolution of ANNs with EAs can be performed at three different levels: evolution of connection weights, evolution of ANNs topologies (architectures), and evolution of learning rules. We review the first two types of evolution; the evolution of learning rules is not discussed, since it is out of scope of this thesis.

2.3.1 Genotype and Phenotype

This section introduces the central notions of biology, such as genotype and phenotype, necessary to understand evolution in ANNs. Natural organisms possess genetic information stored at the molecular-genetic level in DNA. This information, called *genotype*, determines an individual's characteristics and features, as well as its manifestation in the environment i.e., *phenotype*. The distinction between genotype and phenotype is that only genotype can be inherited, while phenotype is a result of interaction between genotype and the environment. In analogy to biology, the theory of EANN operates with the same terminology. In biological organisms the nervous system is a part of phenotype; similarly, an EANN with a given topology and a set of weights is considered

as phenotype. Information specified in a chromosome and evolved by means of EAs represents an EANN's genotype.

Evolution in EANNs strongly depends on distinctions and relations between genotype and phenotype, i.e., *genotype-to-phenotype* mapping. Indeed, the genetic information located in the chromosomes is modified through the evolution process and inherited from parents to offspring. At the same time, the phenotype affects the selective reproduction, as it plays a decisive role in the estimation of the individual's worth, i.e., in the fitness evaluation.

2.3.2 Evolution of Connection Weights in EANNs

Evolution of connection weights [107, 109, 110, 155, 156] in ANNs is a learning process performed by an EA, which aims at obtaining an optimal set of weights so that a trained ANN can perform a given task. It takes place in the environment determined by a fixed ANN topology and conditions of a solving problem. The evolutionary approach to connection weights' training consists of two major stages. At the first stage, the form of connection weights' representation, i.e., binary or real-valued, is decided. At the second stage, the evolution process is conducted, where the type of search operators (crossover and mutation) depends on the individuals' representation. It is worth noting that different representation schemes and search operators can lead to different training performance [154, 155].

The evolution process starts with the generation of an initial population of individuals, where each individual represents a possible set of connection weights for a given ANN topology. After initialization, each individual is evaluated according to its worth regarding to the solving problem. The role of the fitness function in EANNs plays the error function, i.e., MSE or RMSE; thus, the lower the error of an individual the higher its fitness. Following that, typical steps of the classical EA (see Figure 2.2) are performed: first, individuals are selected for reproduction based on their fitness; then new individuals are formed by means of genetic operators; and finally, new population is created. The evolution process continues until some termination criteria are reached. A typical cycle of the evolution of connection weights is as follows:

- Generate a population of size k, where each individual represents a set of n connection weights;
- Evaluate fitness of each individual in the population according to the error between actual and desired outputs over the training data;
- Select parents for reproduction based on their fitness;

- Create offspring by applying genetic operators to the parents;
- Form the next generation of k offspring individuals.

In the binary representation scheme, each connection weight is represented by a certain number of bits, where the number of bits used to encode a weight depends on the maximal numerical value a connection weight might receive. Each individual (a set of weights for a given ANN) is encoded by concatenation of all connection weights of a network, whereby the connection weights to the same hidden/output node are put together. Considering that hidden nodes in ANNs are in essence feature detectors, "Separating inputs to the same hidden node far apart in the binary representation would increase the difficulty of constructing useful feature detectors because they might be destroyed by crossover operators" [155]. Figure 2.6 (left) demonstrates an example of an ANN with the predefined topology and its binary representation. Each weight is represented by three bits, which means that the maximal numerical value of a connection weight is 7. The ANN has six connection weights; thus, the given ANN is represented by 18 bits in a chromosome.

The main advantage of the binary representation for connection weights is the possibility to apply simple genetic operators, such as n-point or uniform crossover and bit-switching mutation. The weakness of such a representation lies in the chromosome length, which depends on the number of bits used to represent a connection weight, and thus is defined by the complexity of an ANN topology. It is obvious that if too many bits represent each connection weight and an ANN is large, the chromosomes' length becomes extremely long. This negatively affects the evolution and makes it inefficient.

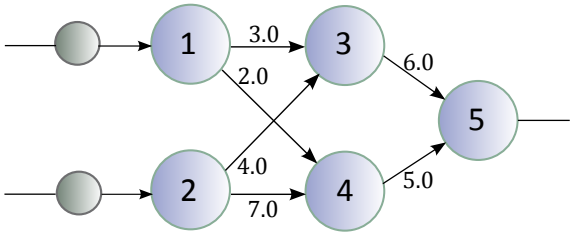


FIGURE 2.6: ANN encoding: binary (left) and real-valued (right) representations of ANN connection weights.

In order to overcome drawbacks faced by the binary representation of connection weights, real-valued representation has been proposed. The individuals with the real-valued representation are encoded directly, i.e., one real number for each connection; thus each

individual is represented by a real vector and the number of genes in the chromosome is equal to the total number of connections between neurons. Figure 2.6 (right) shows the real-valued encoding of connection weights for the given ANN.

The real-valued representation makes impossible to use traditional crossover and mutation operators, associated with the binary chromosome representation. Instead, the evolution of individuals is provided by EP and ES, since they work with the populations of real vectors and are "particularly well-suited for treating continuous optimization" [155].

2.3.3 Difficulties by Evolutionary Learning

The main problem that appears by evolutionary training is the permutation problem, also called competing convention problem. It refers to the case when different genotypic representations encode ANNs of the same functionality. The permutation problem is caused by many-to-one mapping from the encoded ANN representation to the decoded actual network, when many genotypes have different order of hidden nodes in the representation, i.e., have different chromosomes, but are functionally equivalent. Figure 2.7 illustrates the permutation problem; two chromosomes contain different genotype information, but represent ANNs of the same functionality.

The permutation problem makes the recombination operator, i.e., crossover, less efficient, as it produces new individuals by exchanging the blocks of genotype information between parents. Generally, in the evolution of ANNs any permutation of hidden nodes creates ANNs of the same functionality with different genotypic representations. In order to reduce the negative impact of the permutation problem in EANNs, it is beneficial to evolve ANN by means of mutation-based EAs, i.e., EP and ES, which rely on mutation and do not utilize crossover.

2.3.4 Evolution of Architectures in EANNs

The weights training described in the previous section assumed that an ANN architecture remains invariable and fixed during the evolution process. However, as indicated in [155], "architecture design is crucial in the successful application of ANN's because the architecture has significant impact on a network's information processing capabilities". Indeed, if an ANN has a simple topology with few connections it might perform task poorly due to its limited capabilities, while an ANN with a large structure and too many connections might overfit noise in the training data. Generally, designing ANNs is an expert job and depends on his experience, as there is no systematic way to determine

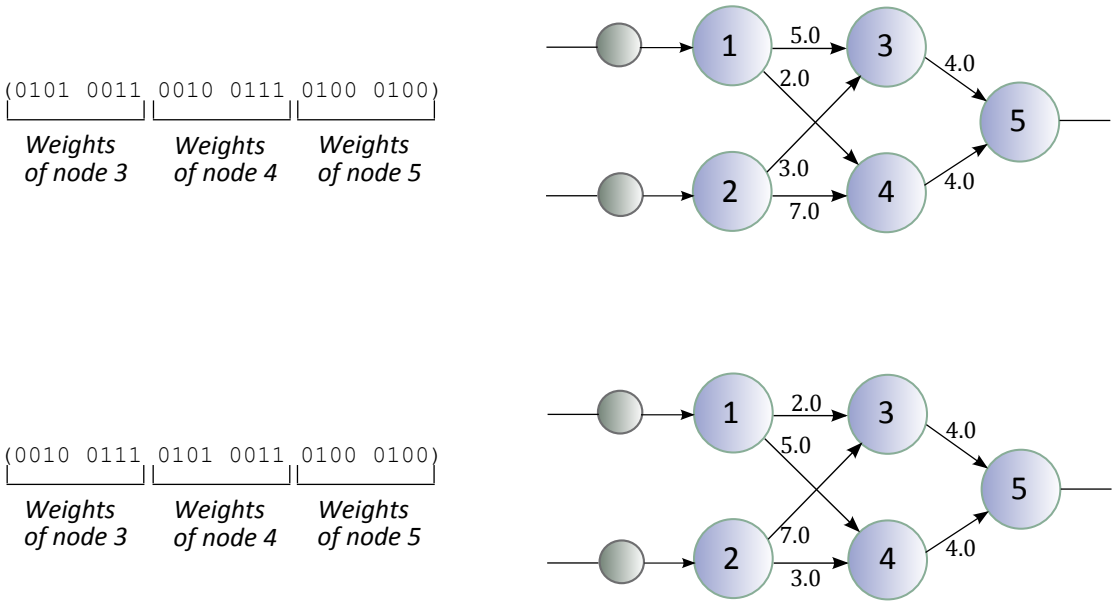


FIGURE 2.7: The permutation problem: two ANNs are functionally equal but have different chromosome representations, since they order hidden neurons differently.

the optimal architecture automatically. Nevertheless, there are some attempts to design ANN topologies automatically by means of constructive and destructive algorithms [40, 54, 74, 124, 138]. A *constructive algorithm* starts with minimal topology, i.e., topology with a minimal number of hidden layers, nodes and connections, and then adds new hidden layers, nodes and connection if necessary during the training process. In opposite to constructive algorithms, a *destructive algorithm* starts with a maximal architecture and removes unnecessary hidden layers, nodes and connections during training. However, as indicated in [4], "Such structural hill climbing methods are susceptible to becoming trapped at structural local optima" and "only investigate restricted topological subsets rather than the complete class of network architectures".

An alternative way to design an optimal ANN architecture is to evolve it by EAs, i.e., formulate an optimization process as a problem of finding the optimal ANN topology in the space, where each point represents an architecture. The performance level of all architectures forms a discrete surface in the space, where evolution process satisfies optimality criteria, e.g., the lowest training error, the lowest ANN complexity, etc. [155]. Miller *et al.* in [105] determined characteristics of such a surface that make application of EAs for the optimum searching advantageous over constructive and destructive algorithms:

- The surface is infinitely large since the number of possible nodes and connections is unbounded;

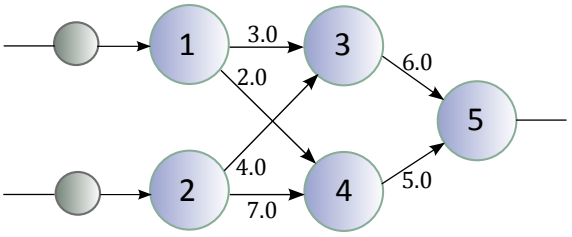
- The surface is non-differentiable since changes in the number of nodes or connections are discrete and can have a discontinuous effect on EANNs performance;
- The surface is complex and noisy since the mapping from an architecture to its performance is indirect, strongly epistatic, and dependent on the evaluation method used;
- The surface is deceptive since similar architectures may have quite different performance;
- The surface is multimodal since different architectures may have similar performance.

Similar to the evolution of connection weights, the evolution of architectures consists of two major phases in which the representation scheme and the type of EA that provides the optimization process must be specified. At the first stage, the key point is to decide how much information about an ANN architecture should be presented in the chromosome. Depending on the information amount represented in the chromosome, two approaches to encode topologies are distinguished. One of them is the *direct encoding scheme* [1, 105, 128, 150, 152], which specifies all the details about an ANN topology, i.e., all nodes and connections, in the chromosome. Another approach, termed the *indirect encoding scheme* [63, 64, 68, 69, 82, 106, 145, 146] represents only the most important characteristics of an ANN structure, such as the number of hidden layers and hidden neurons in each layer. After deciding the representation scheme and the type of EA, evolution of architectures is performed as follows:

- Generate a population of size k, where each individual represents a possible ANN architecture;
- Train each ANN with the decoded architecture by a predefined learning rule. For each ANN, the initial set of connection weights is generated randomly;
- Evaluate fitness of each individual in the population according to the training result and other performance criteria, such as the complexity of the network;
- Select parents for reproduction based on their fitness;
- Create offspring by applying genetic operators to the parents;
- Form the next generation of k offspring individuals.

The direct encoding scheme uses the binary representation to specify connections of an ANN in the *connectivity matrix* $M = (w_{ij})_{N \times N}$, where N is a number of nodes, $w_{ij} =$

1 indicates a connection between nodes i and j, and $w_{ij} = 0$ indicates no connection. The connectivity matrix M has a direct one-to-one mapping to the corresponding ANN topology. Each chromosome representing an architecture is a binary string obtained by the concatenation of rows (or columns) of the connectivity matrix. In fact, connections w_{ij} in the matrix M can also be represented by their real-valued connection weights, which enables evolution of both architectures and connection weights at the same time [4, 94, 97, 100, 116, 120, 159]. The simultaneous evolution of both ANN architectures and weights is advantageous in avoiding noisy fitness evaluation. The evolution of architectures uses phenotype's fitness, i.e., fitness of a given ANN with a full set of weights, to evaluate genotype's fitness, i.e., fitness of an ANN without weights. This makes fitness evaluation noisy and inaccurate, and can deceive evolution. In contrast to evolution of architectures, the simultaneous evolution of topologies and weights represents both architectures and connection weights in the chromosome and therefore, makes no difference between phenotype's and genotype's fitness. The fitness evaluation becomes more accurate, since such a representation reduces noise in evaluation of individuals. Figure 2.8 shows an example of an ANN and two corresponding connectivity matrices that encode an architecture and both an architecture and weights.



$$M_{5x5} = \begin{vmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{vmatrix} \qquad M_{5x5} = \begin{vmatrix} 0.0 & 0.0 & 3.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 4.0 & 7.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 6.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 5.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{vmatrix}$$

FIGURE 2.8: The connectivity matrices of an ANN architecture (left) and an ANN architecture and weights (right).

The direct encoding scheme is suitable for representing both feed-forward and recurrent ANNs, as it can specify constraints on architectures in the connectivity matrix. However, a connectivity matrix of a feed-forward ANN has non-zero elements only in the upper-right triangle, while a recurrent ANN has non-zero elements in the whole matrix. These characteristics lead to the different chromosome representation depending on a type of an ANN. Obviously, it is possible to reduce the length of a chromosome that represents a feed-forward ANN, since we need to encode information located only in the upper-right triangle of a connectivity matrix. For the representation of a recurrent ANN, the

concatenation of complete rows (or columns) must be done. Figure 2.9 demonstrates the chromosome representation with the direct encoding scheme for feed-forward and recurrent ANNs. The ANN given in Figure 2.9 has no recurrent connections; however, it is assumed, that such connections are permitted for the chromosome that represents a recurrent ANN and can appear during the evolution.

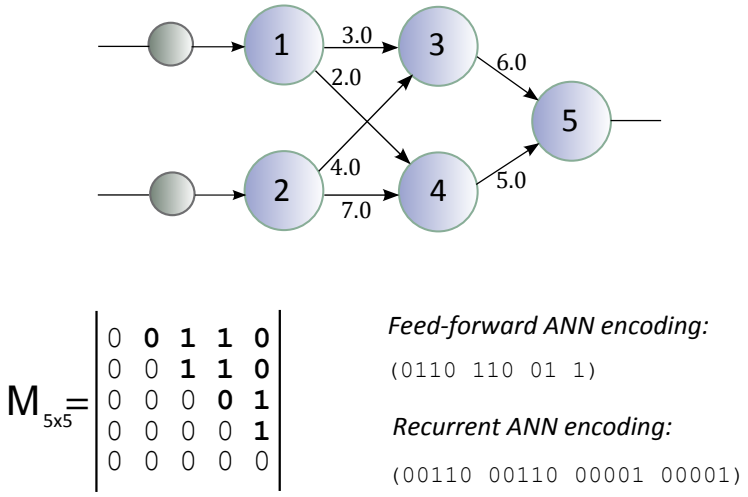


FIGURE 2.9: The direct encoding of an ANN architecture: the chromosome representation for feed-forward and recurrent ANNs.

Apparently, the length of the chromosome augments as a number of neurons in ANN increases. In order to reduce the length of the genotypical representation of ANN architectures, the indirect encoding scheme has been proposed. In contrast to the direct encoding scheme, it represents only partially an ANN. The details of each connection in an ANN are either predefined according to prior knowledge or incorporated in a set of deterministic developmental rules [155]. There are two main methods for indirect encoding, referred to as parametric representation and developmental rule representation.[9] The *parametric representation* incorporates information about some important parameters of ANN structure, e.g., the total number of hidden layers, the total number of neurons in hidden layers, the total number of connections between two layers, etc. in the chromosome [64, 68, 69]. Although this method enables compact genotypic representation of ANN architectures, EAs can not provide the global search due to limited information represented in the chromosome and may fail at finding a compact ANN topology with good generalization ability. The *developmental rule representation* encodes developmental rules used to assemble an ANN topology in the chromosome [63, 82, 106, 145, 146]. The developmental rule is determined by a recursive equation [106] or a generation rule "similar to a production rule in a production system with a left-hand side (LHS) and

[9]This paragraph provides a brief overview of the indirect encoding methods, since they are out of scope of this thesis. Detailed information about the indirect encoding methods is presented in [153, 155].

a right-hand side (RHS)" [155]. Similar to the parametric representation, the developmental rule representation also reduces the length of the chromosome. However, this representation method has a number of limitations and shortcomings, e.g., might develop large ANN topologies (as the compact genotypic representation does not guarantee the compact phenotypic representation), has issues with evolving detailed connectivity patterns among individual nodes, does not evolve architectures and connection weights at the same time, etc.

2.3.5 Modification of EANNs: Parallel Evolutionary Artificial Neural Networks

Alongside with EAs, PEAs have found wide application in ANNs evolution due to their advantages, described in Section 2.1.6, i.e., resistance to premature convergence to the local optimum and ability to consider different portions of the search space. Recent work in this field showed that PEAs often outperform EAs in terms of the quality of the obtained networks, as well as reduce convergence speed [28, 29, 55, 56, 120]. Several tests on different parallel EANNs (PEANNs) indicated their higher accuracy in comparison to EANNs. In addition, due to the global search PEAs are likely to evolve more compact ANN architectures with good generalization ability. However, PEAs require large computational resources; therefore their application is suggested for tasks, where accuracy of the evolved ANNs is of the highest importance [28, 120].

Chapter 3

Mutation-based Evolutionary Algorithms

"It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change."

Charles Darwin

Among variety of the learning methodologies used to train ANNs, EAs stand out from the rest, as they enable not only to find an optimal set of connection weights, but also to develop an optimal ANN topology during the evolution [153, 155]. Moreover, EAs often outperform the originally proposed gradient-descent learning approaches in terms of computational speed, simplicity and the quality of the obtained ANNs. However, not all types of EA are beneficial in ANNs training. The studies in the field showed that mutation-based EAs, i.e., EP and ES, are more efficient in ANNs learning than GAs, which due to their primary search operator (crossover) often face the permutation problem [153, 155].

The key aspects that EP and ES concentrate on are the self-adaptive methods for changing the object parameters and the distribution used in mutation. The classical EP and ES algorithms use the standard normal distribution and similar self-adaptation strategy. Both algorithms adjust the mutation strength by means of strategy parameters; such an approach allows not deviating much in the search space from already existing in the population good solutions. However, trying to be close to the current good solution, the classical EP (CEP) performs small step sizes, and thus, insignificantly improves individuals at each stage of evolution, which leads to the slow convergence to optima.

Later works established that the distribution in the mutation strategy is crucial in the determination of the mutation step size [157]. Yao *et al.* proposed an alternative mutation approach, referred to as the fast evolutionary programming (FEP) [161, 166], which adopts the self-adaptation strategy of CEP, but uses the Cauchy distribution instead of the Gaussian one. The main motivation for applying the Cauchy distribution is that it allows the mutation strategy to perform long step sizes.

As CEP and FEP are self-adaptive methods, this chapter starts with an overview of adaptation and self-adaptation in EP and ES [6, 8, 11–14, 101, 140]. Following that, two classical mutation-based algorithms, i.e., CEP and FEP, applicable to evolve ANNs are described in Sections 3.2 and 3.3. Additionally, combined techniques based on CEP and FEP approaches, i.e., improved fast evolutionary programming (IFEP) and mixed evolutionary programming (MEP) are presented in Sections 3.4.2 and 3.4.1, respectively.

3.1 Adaptation and Self-Adaptation in EP and ES

The idea of self-adaptation is widely used in ES [117, 131] and EP [50, 51, 53] and rarely used in GA [35–37, 72, 104]. According to the definition, given in [101], "self-adaptation in its purest meaning is a state-of-the-art method to adjust the setting of control parameters",[1] where the algorithm manages a set of control parameters itself, by incorporating them in an individual's genotype and evolving them alongside with the object parameters. Self-adaptation aims at biasing the population's distribution towards perspective regions of the search space, keeping up the diversity in the population in order to enable its further evolvability [2, 3, 11, 101].

The principle of self-adaptation was originally introduced by Rechenberg [118] and Schwefel [132–134] for ES and by Fogel [45] for meta-EP. Concerning ES, Rechenberg [118, 119] proposed the idea of adapting the mutation strength during the evolutionary process. He provided analysis in order to study relationship between the rate of successful mutations (i.e., offspring chromosomes with better fitness than parental ones) and the convergence rate, investigating (1+1)-ES on two simple models. As a result, Rechenberg proposed heuristic for controlling the mutation step size, known as the *1/5-success rule*. It can be formulated as follows: the ratio of successful mutations to all mutations should be 1/5; if it is greater than 1/5 the variance of the mutation operator should be increased, otherwise the variance should be decreased [118]. In other words, the optimal convergence rate can be obtained when the 1/5 of all offspring are superior to their parents. However, as shown in [18], the 1/5-success rule may perform less than optimal on many benchmark functions that are non-liner or convex.

[1] Control parameters are also referred to as strategy parameters.

In addition to the 1/5-success rule, Rechenberg proposed the idea of explicit self-adaptation, i.e., embedding the evolution of the strategy parameters with that of the object parameters. During the optimization process the strategy parameters were randomly changed. Compared to ES using the 1/5-success rule, ES with self-adaptation is more universally usable technique, as it leads to faster convergence and is applicable to problems where it is improper to use the 1/5-success rule [101].

Schwefel [132–134] introduced a technique for changing the strategy parameters in ES, which is nowadays associated with the term self-adaptation. According to this approach, each individual in the population is mapped with a corresponding strategy parameter, responsible for the adaptation of the mutation step size. This value corresponds to the standard deviation σ in the mutation operator used to create a new individual. The main step in the self-adaptation consists of a mutation of the mutation strategy parameters themselves. The resulting mutation parameters are then applied in the variation of the object parameters. More specifically, each time an individual undergoes mutation, first, the strategy parameter is slightly changed, and then it is applied to modify the corresponding object parameter. It is assumed, that the strategy parameter shifts mutation step size towards a favorable value. The mutation is performed by the following scheme:

$$\sigma_i{}' = \sigma_i \exp(\tau' N(0,1) + \tau N_i(0,1))$$
$$x_i{}' = x_i + \sigma_i{}' N(0,1),$$

where x_i and $x_i{}'$ are values of the object parameters before and after mutation, respectively, σ_i and σ_i' are values of the strategy parameters before and after mutation, respectively, $N(0,1)$ and $N_i(0,1)$ are normally distributed random values, τ and τ' are the *learning rates*, $\tau \propto 1/\sqrt{2N}$, $\tau' \propto 1/\sqrt{2\sqrt{N}}$. A similar self-adaptation technique was proposed by Fogel for meta-EP [45, 46]:

$$\sigma_i{}' = \sigma_i(1 + \alpha \cdot N(0,1))$$
$$x_i{}' = x_i + \sigma' N(0,1).$$

Both operators lead to similar results, provided that τ and α are sufficiently small.

Adaptive evolutionary computations can be distinguished by the type of adaptation, i.e., how parameters are changed, and by the level of adaptation, i.e., where changes occur [2, 101].[2] Considering the type of adaptation, adaptive evolutionary computations are divided into two distinct types, i.e., algorithms with absolute and empirical update

[2]The classification of adaptive evolutionary algorithms is defined as by Angeline [2]. Angeline's classification was later broadened in [39].

rules. *Absolute update rules* compute a statistic over a number of generations and use the obtained result to decide when and how to modify the adaptive parameters. A well-known example of the class of absolute update rules is the 1/5-success rule.

In contrast to the absolute update rules, algorithms with the *empirical update rules* control the strategy parameters themselves by allowing the evolution process to determine their appropriate values. The strategy parameters are embedded into individuals' genome, i.e., represent a part of genome, and are modified by a separate mutation function. If the strategy parameter leads to an individual with a sufficiently good fitness, it is considered as appropriate. Individuals with the appropriate strategy parameters have usually higher chances to survive than those with badly tunes parameters and, correspondingly, lower fitness. Examples of this class of algorithms are ES and EP with Schwefel's and Fogel's self-adaptation strategies.

Concerning the representational level the adaptive parameters operate on, adaptive evolutionary computations can be divided into algorithms with population-, individual- and component-level adaptive parameters [2]. *Population-level* adaptive parameters dynamically tune parameters that are global to the whole population. Examples are the mutation strength and the covariance matrix adaptation in certain evolution strategies [66, 67, 112]. *Individual-level* adaptive methods use separate strategy parameters for each individual that determine *which of an individual's representational components* is to be modified. For instance, the probability of crossover in [129] is adapted at the level of individuals. *Component-level* techniques perform modification in each component of an individual. Similarly to the individual-level methods, component-level adaptive methods also associate strategy parameters with each individual in the population, but instead determine *how each representational component* of the individual is changed. To this type of adaptation belongs self-adaptations in ES and EP.

3.2 Classical Evolutionary Programming

The classical evolutionary algorithm based on the idea of adaptive mutation is the *classical evolutionary programming* (CEP) [8, 45, 48, 52]. CEP is a modern approach developed from the Fogel's standard-EP [44, 51, 53], which is distinguished by the type of individuals representation and complemented by the self-adaptation strategy. In contrast to the Fogel's EP, which evolves population of finite-state machines to generate AI (see Section 2.1.3), CEP evolves real-valued n-dimensional vectors and thus, can be regarded as a population-based variation of the classical generate-and-test algorithm. The self-adaptation strategy of CEP is similar to that of ES [7, 8, 132, 133].

A cornerstone of CEP is a self-adaptive mutation based on the Gaussian distribution, often referred to as *Gaussian mutation*. Previous studies on CEP with and without adaptive mutation indicated benefits of self-adaptation in the algorithm's performance [6, 8, 45, 48]. The Gaussian mutation operator is applied to all individuals in the population to produce offspring.

Optimization by CEP is stated as a global minimization problem which can be formalized as a pair of (S, f), where $S \subseteq R^n$ is a bounded set on R^n and $f : S \mapsto R$ is an n-dimensional real-valued function. The goal is to find a point $x_{min} \in S$ such that $f(x_{min})$ is a global minimum on S:

$$\forall x \in S : f(x_{min}) \leq f(x),$$

where f does not need to be continuous but it must be bounded. The evolution with CEP is implemented as follows:

1. Generate an initial population of μ chromosomes randomly. Each chromosome is represented by a pair of real-valued vectors (x_i, η_i), $\forall i \in 1, \ldots, \mu$, where x_i is a vector of object parameters, and η_i is vector of standard deviations (often referred to as vector of strategy parameters).

2. Evaluate the fitness value of each individual (x_i, η_i), $\forall i \in \{1, \ldots, \mu\}$ in the parental population based on the objective function.

3. Create new individuals by applying the self-adaptive Gaussian mutation to each parental individual. Each parent (x_i, η_i) produces a single offspring (x_i', η_i') by the following scheme: for $j = 1, \ldots, n$,

$$\eta_i'(j) = \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1)) \tag{3.1}$$

$$x_i'(j) = x_i(j) + \eta_i'(j) N(0,1), \tag{3.2}$$

   where $x_i(j)$, $x_i'(j)$, $\eta_i(j)$, $\eta_i'(j)$ denote the j-th component of the vectors x_i, x_i', η_i and η_i', respectively. $N(0, 1)$ is a normally distributed random value with mean 0 and variance 1 and is generated once for every population member. $N_j(0, 1)$ is a similar random number, but is generated anew for each value of j. The operator-set parameters τ and τ' [46, 48] are commonly set to:

$$\tau \propto \left(\sqrt{2\sqrt{n}}\right)^{-1}$$
$$\tau' \propto \left(\sqrt{2n}\right)^{-1}.$$

The variance vector η_i is a self-adaptive parameter, which is used to control the mutation step size and adapt it to each mutated individual. More specifically, it decides how much the new value of the mutated gene must be deviated from the current value. Such scheme enables significant improvement of individuals with low fitness, while well contributing individuals are not changed much.

4. Evaluate the fitness of each offspring individual (x_i', η_i'), $\forall i \in \{1,\ldots, \mu\}$.

5. Provide pairwise comparison over the union of parents (x_i, η_i) and offspring (x_i', η_i'), $\forall i \in \{1,\ldots, \mu\}$. Each individual of this union is compared with a fixed number of other individuals, randomly selected from both parental and offspring populations. For each comparison the individual with the highest fitness is marked as a "winner".

6. Form new population by selecting μ individuals out of (x_i, η_i) and (x_i', η_i'), $\forall i \in \{1,\ldots, \mu\}$, that have the most "winners" marks for the next generation. The creation of new population completes one evolution cycle, called *iteration* or *epoch*.

7. Repeat this process until some termination criteria are satisfied; each new cycle begins from the Step 3.

3.3 Fast Evolutionary Programming

The analysis of CEP on high-dimensional functions indicated slow convergence to optimum caused by small step sizes performed by the Gaussian mutation [161, 166]. In order to investigate the impact of the Cauchy mutation operator on EP, an alternative mutation approach, referred to as the *fast evolutionary programming* (FEP), has been proposed by Yao *et al.* [161, 163]. The main idea behind FEP is to utilize random values based on the Cauchy instead of Gaussian distribution in the mutation of the object parameters. The one-dimensional Cauchy density function is defined as follows:

$$f(x) = \frac{\gamma}{\pi\,(\gamma^2 + x^2)}, \qquad -\infty < x < \infty,$$

where $\gamma > 0$ is a scale parameter. The corresponding distribution function is

$$F(x) = \frac{1}{\pi} \arctan\left(\frac{x}{\gamma}\right) + \frac{1}{2}.$$

The use of random numbers which follow the Cauchy distribution instead of Gaussian one is motivated by several advantages. As shown in Figure 3.1, the shape of Cauchy density function resembles that of the Gaussian one but approaches the axis so slowly

that an expectation does not exist. As a result, the variance of the Cauchy distribution is infinite. The analysis provided in [43] indicated the benefit of increasing the variance. Specifically, the increased variance improves the efficiency of the global search due to the increased probability of escaping from a local optimum.

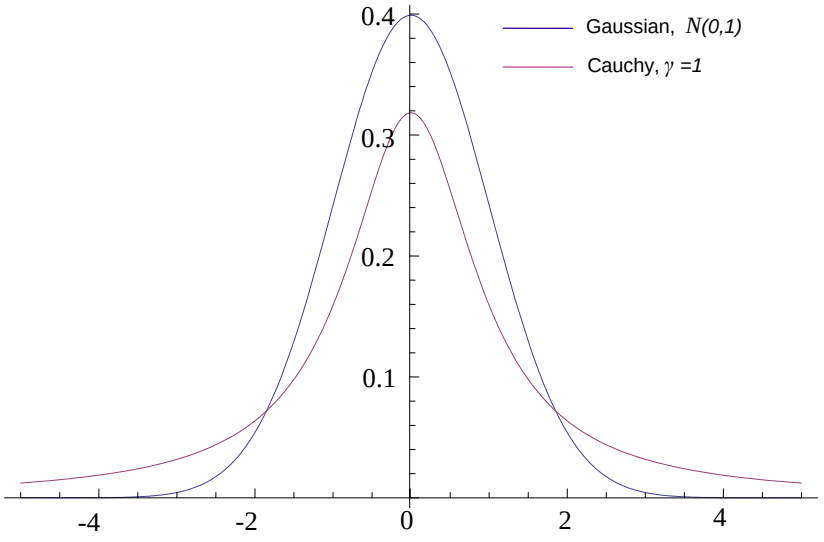


FIGURE 3.1: Gaussian and Cauchy density functions.

Moreover, Figure 3.1 demonstrates that due to long fat tails, the Cauchy mutation is more likely to perform long jumps and to produce offspring that is further away from its parent than Gaussian mutation. Besides, it has higher probability of escaping from a local optimum or moving away from the plateau [166]. However, the smaller hill around the center in Figure 3.1 "indicates that Cauchy mutation spends less time in exploiting the local neighborhood and thus has a weaker fine tuning ability than Gaussian mutation small to mid-range regions"[157].

The FEP algorithm was developed in such a way to keep the modifications of CEP to a minimum. The evolution with FEP repeats all steps of CEP as described in the previous section and adopts the self-adaptation strategy (Eq. 3.1). The object parameters in FEP are modified the same way as in CEP with the only difference that the mutation operator in Eq. (3.2) uses the Cauchy random numbers instead of Gaussian ones:

$$x_i'(j) = x_i(j) + \eta_i'(j)\delta_j, \tag{3.3}$$

where δ_j is a Cauchy random variable with the scale parameter $\gamma = 1$, which is generated anew for each value of j. The vector η in FEP plays the role of the scale parameter γ not the variance in the Cauchy distribution.

As stated in [161], there are two reasons for leaving Eq. (3.1) unchanged. Firstly, this self-adaptation mechanism was constructed for Gaussian and not Cauchy mutation operator. And secondly, the main goal of FEP was to investigate the impact of Cauchy random values on the performance of EP.

3.4 Combined Approaches

The following section presents two mutation-based approaches designed by combining CEP and FEP algorithms. It is assumed that mixing different search biases of Gaussian and Cauchy mutations at the individual and component levels show their beneficial traits and thus, increase the algorithms' functionality.

3.4.1 Improved Fast Evolutionary Programming

The comparative study of CEP and FEP algorithms [161, 163] provided for function optimization problems showed that FEP performs better in solving multimodal functions with many local minima, while CEP is superior in solving multimodal functions with only a few local minima. For other testing functions the results of FEP were comparable to those of CEP.

In order to achieve higher efficiency of an optimization algorithm, Yao *et al.* proposed the modification of FEP, called the *improved fast evolutionary programming* (IFEP) [157, 164], which mixes search biases of Gaussian and Cauchy mutations at the individual level. The main idea of IFEP is to form two offspring chromosomes from the same parent, applying CEP and FEP mutation operators to the first and second offspring's creation, respectively:

$$x_i'(j) = x_i(j) + \eta_i'(j)N(0,1)$$
$$x_i'(j) = x_i(j) + \eta_i'(j)\delta_j.$$

After comparing their fitness, an individual with the least error is chosen as offspring.

3.4.2 Mixed Evolutionary Programming

As mentioned, the Gaussian and Cauchy mutations are combined in IFEP at the individual (chromosome) level. This means that both mutations are applied to all genes of a mutated individual. An alternative mutation approach, called the *mixed evolutionary programming* (MEP) [164], combines Gaussian and Cauchy mutations at the *component* (gene) level. That means, some genes of each parental individual are mutated based on

CEP, while others are modifies by FEP. Given that the probability of Gaussian mutation is P_G and the probability of Cauchy mutation is $P_C = 1 - P_G$, each offspring is created by the following equation:

$$x_i'(j) = \begin{cases} x_i(j) + \eta_i'(j)N(0,1), & \text{with } P_G \\ x_i(j) + \eta_i'(j)\delta_j, & \text{with } P_C \end{cases},$$

where δ_j is a Cauchy random variable with the scale parameter $\gamma = 1$ and $N(0, 1)$ is a normally distributed random value with mean 0 and standard deviation 1. Both γ and $N(0, 1)$ are generated anew for each value of j. The values P_G and P_C are suggested to be set to 0.75 and 0.25, respectively [164].

3.5 Conclusions

This chapter discussed two mutation-based evolutionary methodologies widely used to evolve EANNs. Both CEP and FEP employ the classical component-level self-adaptation mechanism [132, 133] and are differentiated only by a type of distribution used to generate random values, i.e., CEP is based on the Gaussian distribution, while FEP relies on Cauchy random values.

The main variation operator in CEP and FEP is mutation, whose primary searching mechanism is self-adaptation. Self-adaptation guides optimization towards the most perspective regions of the search space by adjusting mutation strength during evolution. Nevertheless, it is hard to underestimate the role of distribution. As shown in [166], the distribution type determines the step size performed by an algorithm. The analysis in [161, 166] showed, that CEP is characterized by small jumps, while FEP performs long step sizes.

Concerning the evolution in ANNs, CEP and FEP evolve population of real-valued vectors, where each vector represents ANN parameters, i.e., either connection weights for a given topology or both connection weights and an ANN architecture. All parental individuals undergo mutation and produce offspring; then, the best individuals from both parental an offspring population form next generation.

CEP and FEP modify parental individuals by involving basically genotype information, and only partially phenotype information. Besides the object parameters (connection weights) that can be considered as both genotype and phenotype information, Eq. 3.1 and 3.3 utilize a total number of the evolving parameters in the components τ and τ', i.e., information of ANN connectivity, which can be regarded as phenotype information. Detailed information about an ANN structure, such as a number of hidden layers and

hidden nodes, is not represented in a modification mechanism. This is explained by the fact that CEP and FEP as EAs methods do not distinguish between genotype and phenotype. Moreover, CEP and FEP are developed as independent search algorithms rather than techniques for ANNs construction and learning.

Chapter 4

Including Phenotype Information in Mutation

"Nam et Ipsa Scientia Potestas Est." ("Knowledge is power.")

Sir Francis Bacon

The objective of this dissertation is to explore alternative mutation mechanisms for ANN design and training, which increase adaptability in the adjustment approach and thus, enable a more efficient improvement of object parameters during evolution. The research in this field was resulted in the development of a novel ANN learning strategy called the *network weight-based evolutionary algorithm* (NWEA). Similar to CEP and FEP, NWEA considers mutation as a primary search operator and does not utilize crossover at all.

The key idea behind NWEA is to involve additional mechanisms of natural evolution in computational evolution in order to improve adaptability of individuals. The goal is to incorporate informative knowledge of an individual and its environment in the mutation mechanism. The question that arises here is: what kind of information is important and should be employed to control the mutation step size?

The central object of evolution is an individual's genotype, which determines its characteristics and manifestation in the environment. However, the information encoded in a chromosome does not show an individual's worth regarding the solving problem. The worth of an individual is estimated by its fitness, which is inversely proportional to the output error. Obviously, the error represents informative knowledge as it shows the individual's position in the search space with respect to the optimum (the higher the

error, the farther the individual is located from the optimum and vice versa, the lower the error the closer is the individual's position to the optimum).

The goal of evolution in EANNs is to minimize an error of an individual. This process takes place in the environment determined by the fitness function as well as some phenotypic characteristics, e.g., an ANN topology. In other words, an ANN topology is another optimization criterion alongside with the conditions of a solving task. Let us examine the learning process in detail. Learning in EANNs is introduced by the evolution of connection weights or by the evolution of both connection weights and architectures. In case of evolution of connection weights, evolution aims at finding the optimal set of weights for a predefined architecture. Hence, a topology represents not only phenotype information, but also a criterion that bounds the search space. The evolution of both connection weights and architectures evolves architectures together with the connection weights. In this case, an ANN topology is an optimization parameter that can be modified; however, the mutation of architectures is performed when the mutation of connection weights of a current topology fails at producing an offspring with the higher fitness. That means, an ANN topology is an implicit optimization criterion and determines the environment.

It is clear that the learning algorithm based on genotype information performs structural adaptations, as it evolves "physical" features of individuals, such as connection weights and architectures. We have assumed that an efficient improvement of individuals may depend not only on genotype, but also on phenotype information. The incorporation of phenotype information allows another form of adaptation, i.e., behavioral adaptation, which enables adjustment of individuals to their habitat. The motivation of including phenotype information is straightforward and has its origin in nature: according to the theory of evolution, individuals with favorable traits are likely to survive and reproduce, and the fitness of individuals is determined by their ability to adapt to the environment. Abstracting from nature, where species have different abilities to learn and adapt to the habitat, we introduce the mutation approach that defines the same behavioral adaptations for all individuals in the population. The adaptation mechanism of NWEA is controlled by two components that represent genotype and phenotype information: the error that shows the worth and position of the individual in the search space (genotype information), and the component, referred to as the *network weight*, which involves knowledge of the individual's habitat represented by information about its ANN topology (phenotype information). The NW component represents an ANN's internal structure, i.e., contains information about the number of hidden layers and neurons located in them. Thus, the adaptation in NWEA is carried out at the individual's level, as both the error and the ANN architecture describe a particular individual. In addition

to genotype and phenotype information, NWEA incorporates random values based on the uniform distribution. Figure 4.1 illustrates information represented in NWEA.

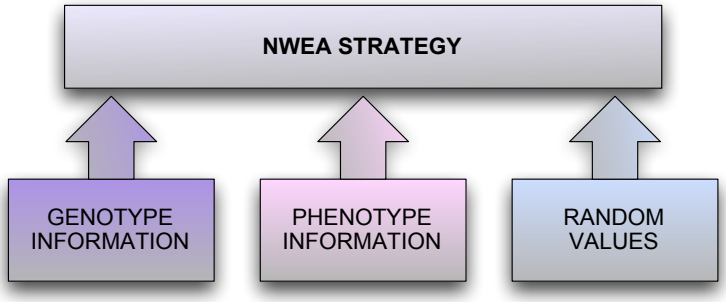


FIGURE 4.1: Information represented in NWEA.

The modification of individuals with NWEA is performed according to the following equation:

$$x_i{}'(j) = x_i(j)\left(1 + N_W^i(l,\bar{n}) \cdot N_E^i \cdot N_{Rand}^{D_U}\right), \tag{4.1}$$

where $x_i(j)$ is a mutated gene (connection weight) of an individual x_i, $N_W^i(l,\bar{n})$ is a NW value, that describes an internal ANN structure of an individual x_i, N_E^i is an error of an individual x_i, and $N_{Rand}^{D_U}$ is a uniformly distributed random value, which is generated anew for each mutated gene.

The structure of gene modification in the chromosome expressed by the Eq. (4.1) differs from that of CEP (see Eq. (3.2)) in terms of the main operation performed over the previous value of the connection weight: while CEP adds the step size to the mutated gene, NWEA multiplies it with the previous value of the connection weight.

By controlling the mutation step size according to information derived from genotype and phenotype, NWEA enables adaptation of mutation strength to the characteristics of individuals and carries out suitable adjustments. This might increase the average percentage of successful mutations and accelerate the evolution process, and also improve the quality of the obtained solutions. Furthermore, NWEA does not contain *a priori* knowledge of ANN topology, i.e., the number of input and output neurons, which makes the algorithm widely applicable.

4.1 Genotype Information in NWEA

The optimization with EAs assumes periodical increment of the average fitness of a population in each new generation. This is achieved by searching the effective regions in the search space by means of systematic selection of the best individuals from both

parental and offspring individuals. In evolution of EANNs, the worth of each potential solution is estimated by its error between the actual and the desired outputs. Thus, the error is a measure that explicitly describes how well or bad the genetic material stored in a chromosome is.

Unlike the vector of strategy parameters in CEP, which controls mutation strength by standard deviations corresponding to the step sizes of a zero mean multivariate Gaussian random variable, NWEA uses the error of a particular individual in determining mutation step size. In contrast to CEP and FEP, which modify object parameters by applying the strategy parameter corresponding to a *particular gene of each individual*, NWEA improves genes by considering a position of a *particular individual* in the search space.

Obviously, the mutation step size is proportional to the individual's error, i.e., NWEA performs long jumps for individuals placed far from the optimum and small step sizes for those in the optimum's neighborhood. Thus, NWEA regulates the mutation strength according to the location of an individual in the search space rather than relies on the length of jumps defined by the distribution a mutation approach is based on.

4.2 Deriving Phenotype Information

The innovation of the genotypic component N_E^i is the idea of using the error of an individual in the mutation mechanism to determine the mutation strength (the existing approaches consider the error only to determine the worth of an individual). In contrast to the genotypic component, the phenotypic component $N_W^i(l, \bar{n})$ is entirely new and thus, its distribution is undefined. In order to investigate the dependency of NW values on ANN architectures, we provided empirical and analytical studies. At the empirical stage, we derived the $N_W(l, \bar{n})$ values for a number of predefined ANN architectures. Following that, at the analytical stage, we explored the distribution of NW values and obtained the generalized equation that describes the dependency of the NW component on the internal structure of any ANN topology.

4.2.1 Empirical Study

The key hypothesis of our approach is that each ANN topology is associated with a value, referred to as the network weight, which characterizes its properties. The properties of a topology are determined by its internal structure, i.e., hidden layers and hidden neurons, as the number of input and output neurons is defined by a solving task. Assuming that

each ANN has a particular network weight, our goal at this stage was to find the NW values in the trial-and-error way, that generally improve the evolution process and quality of the obtained solutions.

The evolution process was observed in the environment determined by a number of ANNs with predefined fixed topologies. We examined ANN architectures with 1 to 5 hidden layers and 2 to 6 hidden neurons in each layer, and the minimal allowable number of hidden neurons in each layer was 2. The values in range [0.01..100] were assigned to the component N_W in Eq. (4.1) and remained unchanged during the optimization process. In order to reduce stochastic nature of random initial populations, a set of 50 different initial populations of size 100 were considered for each of the studied architectures. That means, each value in range [0.01..100] was used in NWEA and tested on 50 initial populations. The worth of individuals was estimated by MSE between the actual and the desired outputs over all testing examples.

The NWEA algorithm with different values of N_W was applied to train ANNs with regard to constructing meta-models [17, 80, 88, 167] for function approximation. Eight global optimization functions were used as test problems [8, 44, 135, 144]: high-dimensional unimodal f_1, f_2 and multimodal f_3, f_4 functions (dimension 30); and low-dimensional functions f_5 (dimension 4), f_6, f_7 and f_8 (dimension 2) with only a few local minima. The functions are listed in Table 4.1. The reason for considering eight functions of different complexity (the most difficult are multimodal functions where the number of local minima increases exponentially with the problem dimension) was to obtain the generalized NW values which increase the average improvement and do not contain any *a priori* information about a solving problem. Besides, if a number of test problems is small, it is difficult to make a generalized conclusion.

Function	n	S	f_{min}
$f_1(x)=\sum_{i=1}^{n} x_i^2$	30	$[-100,100]^n$	0
$f_2(x)=\sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	30	$[-10,10]^n$	0
$f_3(x)=\sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	30	$[-5.12,5.12]^n$	0
$f_4(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + +20 + e$	30	$[-32,32]^n$	0
$f_5(x)=\sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2+b_i x_2)}{b_i^2+b_i x_3+x_4}\right]^2$	4	$[-5,5]^n$	0.0003075
$f_6(x)=4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5,5]^n$	-1.0316285
$f_7(x)=\left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1-\frac{1}{8\pi}\right)\cos x_1 + 10$	2	[-5, 10]×[0, 15]	0.398
$f_8(x)=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+ +3x_2^2)] \times [30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2- -36x_1x_2+27x_2^3)]$	2	$[-2,2]^n$	3

TABLE 4.1: Test functions used to obtain the NW-values. n is the dimension of the function, f_{min} is the minimum value of the function, and $S \subseteq R^n$.

While constructing meta-models, off-line learning was used [39, 151], i.e., ANNs were trained on data generated before optimization. The data sets consisted of 1000 data points for high-dimensional functions and 500 points for low-dimensional functions were divided into two subsets: a half of data was used as the training data; the remaining part was used as the validation data. For each NW value of a particular ANN architecture, the best NWEA-evolved ANN over 50 runs, i.e., the ANN with the smallest error on the validation set, was selected as a meta-model.

The constructed meta-models supported FEP in finding optima for the test functions. The evolved ANNs were used together with the original function in the fitness evaluation according to the generation-based evolution control [151]. Specifically, for a given number of consecutive generations c, called a "cycle", offspring individuals of g generations, $g < c$, are evaluated by the original fitness function, and individuals of the remaining $c - g$ generations are evaluated by the meta-model. For the optimization with FEP, we used $c = 20$, $g = 10$, i.e., the original objective function and the meta-model were applied periodically for 10 generations each. The initial setup for FEP was similar to that described in [161]. For each test function 30 runs of FEP were performed.

The experiments at the empirical stage included preliminary and primary studies, which are differentiated by the complexity of the examined ANN topologies. In the preliminary step, simple ANN topologies with varying number of hidden layers and the same number of neurons in each hidden layer were examined. For a given number of hidden layers, the total number of ANNs tested at the preliminary stage was 5; respectively, the total number of the studied topologies was 25. In the primary step, tests were carried out for complex ANN topologies with varying number of hidden layers and varying number of neurons in them. Obviously, primary studies are the extension of preliminary analysis, since ANN architectures considered in the former case were also examined in the latter analysis. This gave an opportunity to verify results on the simple architectures from the preliminary step with those obtained in the primary step. For a given number of hidden layers, the total number of ANN topologies $\hat{Q}(l, n_{max})$ tested in the primary stage was:

$$\hat{Q}(l, n_{max}) = (n_{max} - 1)^l,$$

where n_{max} is the maximal possible number of neurons in each hidden layer, and l is a given number of hidden layers. For instance, an ANN with 3 hidden layers was represented with $(6 - 1)^3 = 125$ different topologies in the primary step, while in the preliminary analysis the number of examined topologies was 5. The total number of ANN architectures studied in the primary step was $\sum_{l=1}^{5} \hat{Q}(l, n_{max}) = \sum_{l=1}^{5} (n_{max} - 1)^l = 3905$.

4.2.1.1 Results

As a result of the empirical analysis, we selected the best NW values which directed the learning process optimally in terms of convergence speed and the quality of the obtained ANNs. The convergence speed was measured by the average number of generations needed to train ANNs, and the quality of evolved ANNs was estimated by the approximation accuracy in the testing phase (i.e., while calculating the fitness of individuals in FEP). The number of selected values was equal to the total number of studied ANN topologies, i.e., 3905.

The obtained results maintained the hypothesis of phenotypic information influence on the evolution process. For a given ANN topology, NWEA with the NW values within very small range (minimal ± 0.02, maximal ± 0.08) obtained the best generalization results for all test functions over 50 runs. For those architectures considered in both preliminary and primary steps (simple topologies) the maximal difference between NW values in the preliminary and primary stages was ± 0.04. Thus, the results empirically proved that *for a given ANN architecture, there exists a corresponding to it the NW value that depends on the architecture's internal properties, i.e., the number of hidden layers and the number of neurons located in them.* This value is not related with genotype information nor with *a priori* knowledge of a solving task.

An interesting observation was made from the results. The best NW values for ANNs with the same number of hidden layers and the same total number of hidden neurons were in the same range and had insignificant distinctions. For example, for ANNs with the hidden structures 2-3 and 3-2, i.e., ANNs with 2 hidden layers and 5 total hidden neurons, the best NW values were 4.856 and 4.862, respectively. This led to an assumption that for the fixed number of hidden layers it is not the number of neurons located in each hidden layer, but instead the total (or average) number of hidden neurons that characterizes the phenotypic component. This assumption was verified in the next step.

4.2.2 Analytical Study

The ultimate goal of the analytical study was to investigate the dependency of the obtained NW values on the ANN architectures and deduce the corresponding function. The main challenge here was to define the approximation function which is close to the function of NW values. The key point of our approximation is that we consider ANNs as physical particles in statistical mechanics [62]. Assuming that ANN-particles are identical, we studied a set of ANN architectures as a system of particles in quantum statistics.

Quantum statistics is a branch of statistical mechanics that describes the behavior of n-particle quantum systems, i.e., systems of identical (indistinguishable) particles of a particular type that follow the rules of quantum mechanics. The state of an n-particle system is determined by a set of quantum numbers.

Let us consider an ANN topology as a particle and a set of ANN-particles as a system of m[1] identical particles that have negligible mutual interactions. This allows a system to be described in terms of single-particle energy states. A state of a system is determined by a set of quantum numbers (l, n_1, n_2, ..., n_l, 0, ..., 0), where l is the number of hidden layers, n_i is the number of neurons in i-th hidden layer, $i = 1, \ldots, l$, and 0 defines the empty hidden layers, i.e., hypothetical hidden layers with 0 neurons. Assume ε_k is the energy of a single-particle state k, which corresponds to the functionality of an ANN with a given quantum numbers, and m_k is the number of particles in this state (the occupancy number). Hence, the state of a system is defined by specifying occupancy numbers m_k of different quantum states. In other words, the statistical distribution of particles over quantum states describes the many-particle system.

In statistical mechanics, the statistical distribution of particles over single-particle energy states is described either by Maxwell-Boltzmann, Fermi-Dirac or Bose-Einstein statistics, depending on the type of particles. Maxwell-Boltzmann statistics describes the statistical distribution of particles in classical mechanics (where particles are considered distinguishable) and assumes that the average occupancy numbers are small ($\ll 1$). Fermi-Dirac and Bose-Einstein statistics are quantum statistics, which determine the statistical distribution of fermions and bosons, respectively. However, both Fermi-Dirac and Bose-Einstein statistics approach Maxwell-Boltzmann statistics at high temperatures or low concentrations.

In our case, to each ANN with the given quantum numbers corresponds one quantum state, i.e., one schematic ANN topology. Hence, a system of ANN-particles is similar to a system of fermions which obey the *Pauli exclusion principle*, that is, no two identical fermions may occupy the same quantum state simultaneously. Consequently, the distribution of ANN-particles over quantum states is expected to be Fermi-Dirac-like.

As described above, in a particular case, Fermi-Dirac statistics becomes Maxwell-Boltzmann statistics; thus, it is possible to use Maxwell-Boltzmann statistics as an approximation to Fermi-Dirac statistics. Follow that, we approximated the NW values by the Boltzmann distribution.

The results of the empirical studies showed that the NW values are approximately equal for ANNs with the same number of hidden layers and the same average number of

[1] In the literature, a number of particles is denoted by n. We denote this number by m, as the variable n refers to the neurons in ANNs.

neurons in them. This means, more than one different ANN-particles can have the same energy. In terms of quantum mechanics, these quantum states are degenerate, as they are at the same energy level. The degeneracy of the energy levels explains why ANNs with different topologies may have the same functionality.

The presence of degenerate states reduces the total number of topologies of different functionality ($\hat{P}$) for ANNs with a given number of hidden layers; this number depends on a number of hidden layers l and a maximal allowable number of neurons n_{max} in each layer:

$$\hat{P}(l, n_{max}) = l(n_{max} - 2) + 1. \tag{4.2}$$

It is easy to notice that for ANN architectures with 1 to 5 hidden layers and 6 maximal number of neurons in each hidden layer, considered at the empirical stage, $\hat{P}$ were 5, 9, 13, 17 and 21, respectively.

Further, while deriving the distribution of the NW values, the quantum state of ANN-particle was described by two quantum numbers (l, $\bar{n}$), where $\bar{n}$ is called a *effective quantum number* and represents an average number of neurons in l layers. This replacement does not change the theoretical approach and is done for the purpose of simplifying the analysis model.

4.2.2.1 Results

As a result of the approximation by the average number of neurons $\bar{n}$ in each layer, $n = 2, \ldots, 6$, we obtained five functions of the type

$$N_w(l, \bar{n}) = \frac{C_1}{\exp(C_2\bar{n}) + 1}, \tag{4.3}$$

for each number of hidden layers l, $l = 1, \ldots, 5$, where C_1 and C_2 are approximation coefficients, yet to be found.

Having in mind the dependency on the number of hidden layers, this expression can be rewritten

$$N_w(l, \bar{n}) = C_1^0 + \frac{C_1^1}{\exp(C_2\bar{n}) + 1}, \tag{4.4}$$

where C_1^0 and C_1^1 represent linear and constant parts of the dependency. Usage of simple linear regression leads to

$$C_1^0 = 3.0 + \frac{l}{2}, \tag{4.5}$$

$$C_1^1 = 2.0 - \frac{l}{2}. \tag{4.6}$$

Using coefficients $A_1 = 3.0$, $B_1 = 2.0$ instead of constants in (4.5) and (4.6), and substituting them in (4.4), we obtained:

$$N_w(l, \bar{n}) = A_1 + \frac{l}{2} + \frac{B_1 - \frac{l}{2}}{\exp(C_2 \bar{n}) + 1}. \tag{4.7}$$

The expression (4.7) can be further rewritten while taking into account an analogy with thermodynamics

$$N_w(l, \bar{n}) = A_1 + \frac{l}{2} + \frac{B_1 - \frac{l}{2}}{\exp\left(\frac{\bar{n} - C_2^0}{T_1}\right) + 1}, \tag{4.8}$$

where the constant T_1 has the same physical meaning as temperature and the term C_2^0 is similar to the chemical potential in thermodynamics. Although this factorization does not reveal the analytical dependency of the remaining component C_2^0, it reduced the function (4.4) to the Fermi-Dirac-like function. Let us also mention that in Fermi-Dirac statistics the chemical potential μ is determined from the condition that the total number of particles in gas is equal to a given number N:

$$\sum_k \frac{1}{\exp\left(\frac{\varepsilon_k - \mu}{kT}\right) + 1} = N.$$

The approximation by the number of hidden layers let us express C_2^0 as a function similar to (4.3)

$$C_2^0(l) = A_2 + \frac{B_2}{\exp\left(\frac{l - B_2}{T_2}\right) + 1}, \tag{4.9}$$

where $A_2 = 1.2$, $B_2 = 3.2$ and $T_2 = 0.6$. C_2^0 was further denoted as μ by analogy with the chemical potential in the Fermi-Dirac statistics.

Thus, the studies showed that the phenotypic component N_W in the equation (4.1) depends on the total number of hidden layers and the average number of neurons in hidden layers, i.e., is related with the internal structure of ANNs. This relationship is defined by the Fermi-Dirac-like function:

$$N_W(l, \bar{n}) = A_1 + \frac{l}{2} + \frac{B_1 - \frac{l}{2}}{\exp\left(\frac{\bar{n} - \mu}{T_1}\right) + 1}, \tag{4.10}$$

where μ is determined by the following equation:

$$\mu = A_2 + \frac{B_2}{\exp\left(\frac{l - B_2}{T_2}\right) + 1}, \tag{4.11}$$

and the coefficients $A_1 = 3.0$, $B_1 = 2.0$, $T_1 = 0.4$, $A_2 = 1.2$, $B_2 = 3.2$ and $T_2 = 0.6$. Figure 4.2 demonstrates the NW values for the considered ANN topologies.

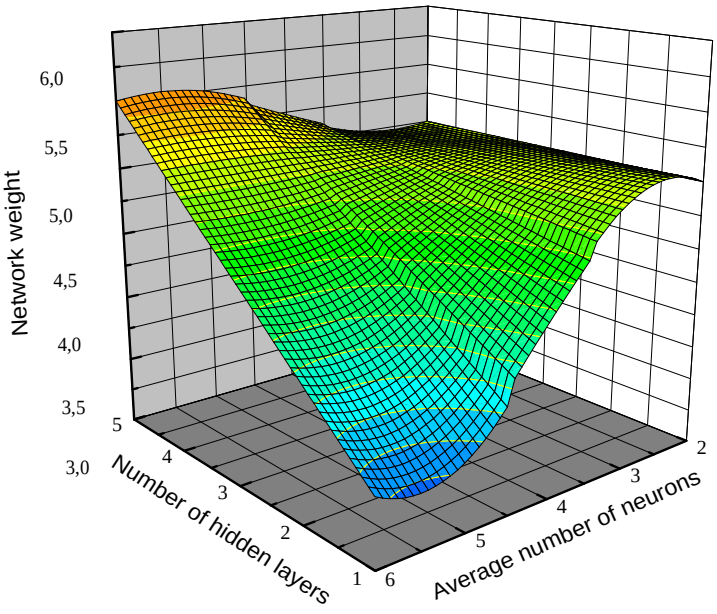


FIGURE 4.2: The network weight values.

The value μ has the same physical meaning as the chemical potential in thermodynamics, i.e., shows the amount by which the energy of a system would change if one particle were added (other parameters are fixed). As mentioned, we assumed that the energy ε_k of a single-particle state k corresponds to the functionality of a topology with a given number of hidden layers and neurons. At the same time, it was shown that the functionality of ANNs is defined by a given number of hidden layers, see Eq. (4.2). Obviously, in our case the energy of the system will change if the number of hidden layers is varied.

Let us study the behavior of the derived functions (4.10) and (4.11). If the average number of neurons increases, the function (4.10) tends to:

$$\lim_{\bar{n}\to+\infty} N_w(l,\bar{n}) = A_1 + \frac{l}{2},$$

as

$$\lim_{\bar{n}\to+\infty} \exp\left(\frac{\bar{n}-\mu}{T_1}\right) = +\infty.$$

If the number of hidden layers increases, the function (4.11) approaches A_2:

$$\lim_{l\to+\infty} \mu = A_2,$$

as

$$\lim_{l\to+\infty} \exp\left(\frac{l-B_2}{T_2}\right) = +\infty.$$

Hence, the function (4.10) tends to infinity as $l \to +\infty$:

$$\lim_{l \to +\infty} N_w(l, \bar{n}) = +\infty.$$

The function (4.10) tends to infinity as both the number of hidden layers and neurons in them increase:

$$\lim_{l \to +\infty, \bar{n} \to +\infty} N_w(l, \bar{n}) = +\infty.$$

The right hand fractional expression of the function (4.10) depends on the number of hidden layers:

$$\frac{B_1 - \frac{l}{2}}{1 + \exp\left(\frac{\bar{n} - \mu}{T_1}\right)} \begin{cases} > 0, & if \quad l < 4 \\ = 0, & if \quad l = 4 \\ < 0, & if \quad l > 4 \end{cases},$$

which demonstrates how the network weight is related with the size of an ANN topology and difficulties in its learning. As known, ANNs with small number of connections may not be able to learn good due to their small capability, while the large networks may overfit the noise in the training data and thus, have poor generalization ability. For small ANNs ($l < 4$), the network weight decreases as the average number of neurons increases, which means that NW tends to reduce the mutation strength when the number of connections is growing. For large ANNs ($l > 4$), the network weights increases as the average number of neurons in hidden layers increases. In this case, NW encourages long mutation step sizes and thus, long jumps in the search space, in order to prevent overtraining in large networks. The ANN with 4 hidden layers can be considered as a transitional network between small and large topologies; its functionality does not depend on a number of hidden neurons and the corresponding network weight value is 5.0.

4.3 Network Weight-based Evolutionary Algorithm

This section describes the evolution of ANNs with the NWEA algorithm. The main steps of evolution differ depending on whether the evolution of connection weights or the evolution of both connection weights and architectures is provided. The evolution of connection weights with NWEA is implemented as follows:

1. Generate an initial population of m randomly generated individuals. Each individual x_i, $\forall i \in \{1, ..., m\}$, represents one possible set of connection weights, $x_i(j) \in [-1.0; 1.0]$, $j \in \{1, ..., k\}$, where k is the total number of connections between neurons.

2. Evaluate the fitness of each individual in the population; the fitness is inversely proportional to the output error over all training examples.

3. Create p offspring individuals, $p = m$, by modifying each parental individual according to Eq. (4.1):

$$x_i{}'(j) = x_i(j)\left(1 + N_W^i(l, \bar{n}) \cdot N_E^i \cdot N_{Rand}^{D_U}\right),$$

where $x_i(j)$ is a j-th gene of a chromosome x_i (connection weight), $N_W^i(l, \bar{n})$ is a NW value that corresponds to a given ANN topology, N_E^i is the error of x_i, determined by some error function (e.g., MSE, RMSE), and $N_{Rand}^{D_U}$ is a uniformly distributed random value.

The value $N_W^i(l, \bar{n})$ is defined according to the Fermi-Dirac-like function (4.10), which depends on the number of hidden layers l and the average number of neurons in hidden layers $\bar{n}$ in an ANN

$$N_w(l, \bar{n}) = A_1 + \frac{l}{2} + \frac{B_1 - \frac{l}{2}}{\exp\left(\frac{\bar{n}-\mu}{T_1}\right) + 1},$$

where μ depends on the number of hidden layers and is calculated as follows:

$$\mu = A_2 + \frac{B_2}{\exp\left(\frac{l-B_2}{T_2}\right) + 1},$$

and $A_1 = 3.0$, $B_1 = 2.0$, $T_1 = 0.4$, $A_2 = 1.2$, $B_2 = 3.2$, $T_2 = 0.6$. In case of the evolution of connection weights, the component $N_W(l, \bar{n})$ is calculated once in the beginning of the optimization process, as the ANN topology is the same for all individuals and remains fixed during the evolution.

4. Evaluate the fitness of each offspring $x_i^{'}$, $\forall i \in \{1, ..., p\}$.

5. Determine the most perspective individuals by applying (μ+λ)-ES, which considers both parental and offspring individuals as candidates to be parents in the next generation. Conduct pairwise comparison over the populations of m parental and p offspring individuals by performing the tournament selection. For each individual, a group of q opponents, $q = 4$, is selected randomly from both parental and offspring populations with an equal probability; if the individual's fitness is higher (the error is lower) than the opponent's, it is marked as a "winner".

6. Select the best m individuals from both parental and offspring populations, that have the most wins for the next generation.

7. Repeat the process from step (3) until some stopping criteria are satisfied.

The simultaneous evolution of connection weights and architectures with NWEA is implemented as follows:

1. Generate an initial population of m randomly generated individuals, where each individual represents a feed-forward ANN with the random number of hidden layers and neurons within a certain range, defined by a user. The initial networks are fully connected. The initial connection weights are assigned randomly within a range [-1.0; 1.0].

2. Evaluate the fitness of each individual in the population; the fitness is inversely proportional to the output error over all training examples.

3. Partially train each network in the population. Select each individual of the population one at a time and initialize a sub-population of size $m/2$ consisting of copies of that individual. Evolve the sub-population during a given number of generations in the environment defined by a given ANN topology, i.e., perform the evolution of connection weights in the environment of the fixed topology (steps (3) - (7) of the evolution of connection weights). After a certain number of generations the evolution process is stopped and the best individual in the sub-population replaces the initial individual in the main population.

4. Create p offspring individuals, $p = m$. For each parental individual in the population:

   (a) Create an offspring by modifying a parental individual according to Eq. (4.1) (see step (3) of the evolution of connection weights). If the fitness of a new individual is higher than that of a parental one, mark a new individual as an "offspring"; otherwise, mark it as an "offspring candidate" and move to step (4b).

   (b) Remove a certain number of hidden nodes from the parental individual. The number of mutated hidden nodes is user-specified. The node deletion process is described in Section 5.2.2. Partially train a new network performing the evolution of connection weights as described in step (3). After a given number of generations stop the training process. If the fitness of a new individual is higher than that of a parental one, mark a new individual as an "offspring"; otherwise, mark it as an "offspring candidate" and move to step (4c).

   (c) Remove a certain number of hidden connections from the parental individual. The number of mutated hidden connections is user-specified. The connection deletion process is described in Section 5.2.2. Connections close to zero are removed first. Partially train a new network performing the evolution of connection weights as described in step (3). After a given number of generations

stop the training process. If the fitness of a new individual is higher than that of a parental one, mark a new individual as an "offspring"; otherwise, mark it as an "offspring candidate" and move to step (4d).

(d) Add a random number of connections to the parental individual according to their importance; this process is described in Section 5.2.2. Partially train a new network performing the evolution of connection weights as described in step (3). After a given number of generations stop the training process. Denote the obtained individual as an "offspring 1". Add a random number of hidden nodes to the parental individual. The nodes are added by splitting connections of the already existing node; this process is described in Section 5.2.2. Partially train a new network performing the evolution of connection weights as described in step (3). After a given number of generations stop the training process. Denote the obtained individual as an "offspring 2". Compare the fitness of the "offspring 1" and "offspring 2" individuals, then compare the best of two offspring to the parent. If the fitness of a new individual is higher than that of a parental one, mark a new individual as an "offspring"; otherwise, mark it as an "offspring candidate" and move to step (4e).

(e) Compare the fitness of all "offspring candidates"; mark the best individual as an "offspring".

5. Determine the most perspective individuals by applying (μ+λ)-ES, which considers both parental and offspring individuals as candidates to be parents in the next generation. Conduct pairwise comparison over the populations of m parental and p offspring individuals by performing the tournament selection. For each individual, a group of q opponents, $q = 4$, is selected randomly from both parental and offspring populations with an equal probability; if the individual's fitness is higher (the error is lower) than the opponent's, it is marked as a "winner".

6. Select the best m individuals from both parental and offspring populations, that have the most wins for the next generation.

7. Repeat the process from step (4) until some stopping criteria are satisfied.

4.4 Conclusions

In this chapter the ANN learning algorithm, called NWEA was presented. It is based on the novel mutation mechanism, which utilizes both phenotype and genotype information of individuals to determine the mutation strength and bias the evolution towards

an optimum. More specifically, NWEA performs not only structural adaptations, i.e., evolves the genetic characteristics of individuals, but also behavioral adaptations, i.e., conducts interactions between individuals and their habitat, and uses knowledge of the environment in the evolution process. The presented algorithm can be used to evolve either the connection weights or both connection weights and topologies.

The main challenge of NWEA was to derive the function of the phenotypic component NW and establish its dependency on an ANN topology. The empirical studies showed that the phenotypic component is related to an ANN's hidden structure, i.e., the total number of hidden layers and the average number of neurons in them. This dependency is justified analytically and is determined by the derived Fermi-Dirac-like function.

Chapter 5

Experimental Studies

"However beautiful the strategy, you should occasionally look at the results."

Sir Winston Churchill

This chapter presents experimental studies provided to investigate features of the proposed approach and examine performance of NWEA-evolved ANNs by comparing the obtained results with those of other algorithms. The chapter documents three major parts of empirical studies. The first part analyzes characteristics of NWEA by evolving connection weights in the environment determined by predefined fixed ANN topologies (Section 5.1). The second part studies NWEA-evolved ANNs on complex tasks and investigates their performance from different perspectives: 1) compares the generalization of NWEA-evolved ANNs with that of the existing approaches; 2) examines the role of activation function type on the evolution process; 3) examines evolution under different step sizes determined by different distributions. This part considers simultaneous evolution of both connection weights and architectures (Section 5.2). Finally, the impact of parallelization on the generalization ability in ANNs is investigated in Section 5.3.

5.1 Evolving Connection Weights

This section examines evolution of connection weights with NWEA, conducted in the environment determined by the predefined fixed ANN topologies. In particular, the issues bounded up with the NWEA's internal characteristics, such as the efficiency of representing genotype information by an individual's particular error, the algorithms's convergence speed, the quantity of successful mutations, and the precision of the obtained solutions are questioned.

In order to evaluate the performance of NWEA, networks trained with NWEA for the XOR problem were compared to those trained with CEP [46, 48], FEP [161, 163] and IFEP [157, 164] algorithms. The choice of XOR as a case application was motivated by its independence of ANN topologies, i.e., solving XOR does not set limitations on a ANN structure and does not require finding an optimal topology, as even a simple ANN with one hidden neuron can successfully solve this function [92]. The evolution of connection weights was observed under the same initial conditions for all compared algorithms, which reduced the effect of randomness caused by the stochastic nature of EAs on the evolution process and enabled fair comparison.

5.1.1 Experimental Setup

All experiments were conducted using the same initial conditions, i.e., the same initial populations and ANN topologies. The tests were carried out for 25 different feed-forward ANNs with 1-5 hidden layers and 2-6 numbers of neurons in each hidden layer, so that the simplest ANN had 1 hidden layer with 2 neurons and the most complex ANN contained 5 hidden layers with 6 neurons in each layer. The initial populations of individuals was generated at random and consisted of 50 individuals for NWEA, CEP and FEP algorithms, where each individual represented a real-valued vector of connection weights for a considered ANN architecture. As suggested in [157], the population for IFEP consisted of 25 individuals, because each individual in IFEP generates two offspring. The individuals were evaluated according to their error defined by the mean squared error (MSE) between the actual and the desired outputs over all training examples. Each algorithm (with 1000 initial populations) was run 1000 times for each ANN. The evolution process terminated when a set of connection weights with the output error 1.0e-3.0 was obtained.

5.1.2 Investigating the Impact of a Particular Error in NWEA

The first set of experiments were provided to investigate the efficiency of genotype information represented by an individual's *particular* error in NWEA. For this purpose, three types of NWEA with the same phenotype information and different genotype information were applied to train ANNs. Following genotype information in the component N_E of Eq. 3 was studied: 1) the *minimal* error of the current population, i.e., the error of the best individual in the population; 2) the *average* error of the current population, i.e., the average error of all individuals in the population and, 3) the *particular* error of each individual in the current population. The algorithms were evaluated by the average number of iterations required to find the optimal set of connection weights.

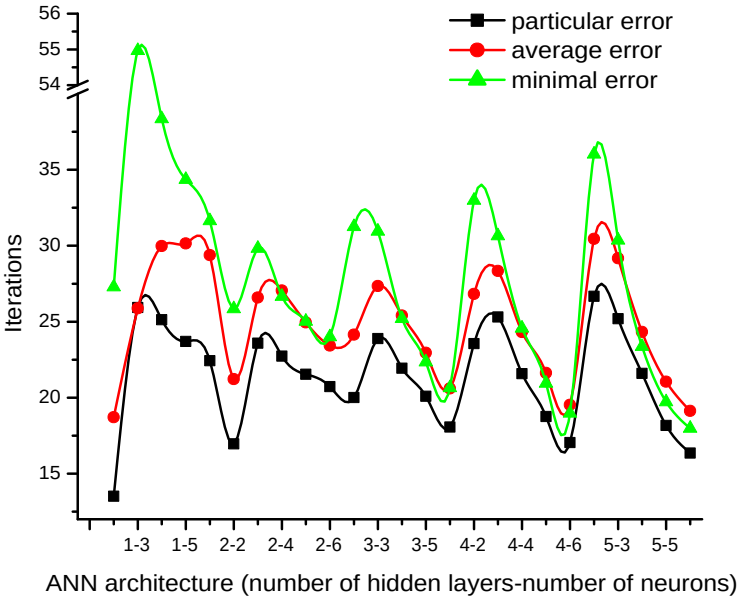


FIGURE 5.1: Convergence of NWEA with particular, average and minimal errors in N_E: average number of iterations.

Figure 5.1 presents the average results over 1000 runs of each algorithm for each ANN topology described in Section 5.1.1. As shown, NWEA with the particular error in mutation outperformed those with the minimal and the average errors in terms of convergence. The results were predictable, as such a performance of algorithms is explained by the comprehension of the improvement strategies. Since the learning was provided in the environment of the predefined fixed ANN architectures, the phenotypic component was the same for all algorithms and remained unchanged during the evolution. That means, the mutation strength was controlled by the genotypic component. Apparently, the error describes a position of an individual regarding the optimum. Hence, mutation with the minimal error performs too small step sizes for the individuals with a greater error, as it contains partial information about their positions in the search space; mutation with the average fitness makes near-optimal adaptation for majority of the population, as it incorporates the average distance of individuals to the optimum. In contrast, mutation with the particular error contains explicit information about an individual and carries out appropriate adjustments. This leads the algorithm to fast convergence.

5.1.3 Convergence Speed: Iterations and Time

This and following sections provide a comparative analysis of NWEA, CEP, FEP, and IFEP algorithms used to train ANNs for the XOR function. The study presented in this section estimates the algorithms' convergence speed measured by the average number of

iterations and the average time needed to obtain the optimal solution. The experimental setup was the same as described in Section 5.1.1. Figure 5.2 and Table 5.1 provide the average results over 1000 runs for each of four algorithms.

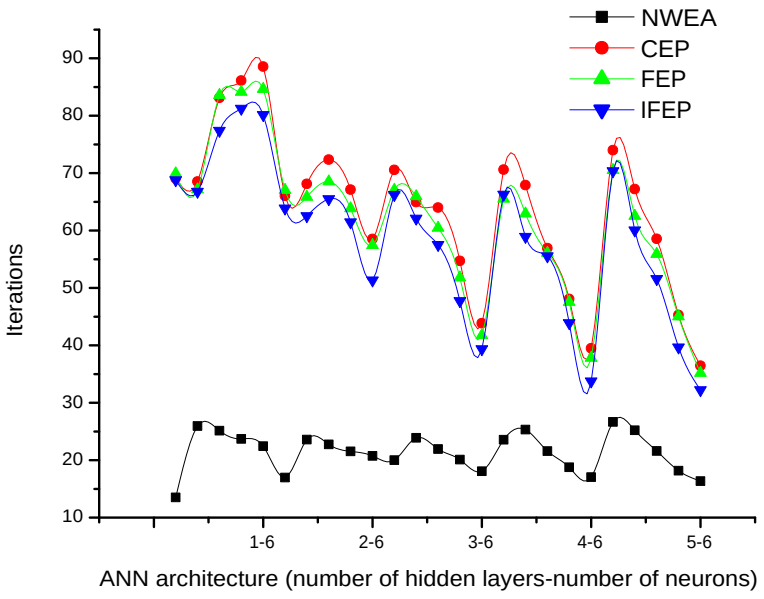


FIGURE 5.2: Convergence of NWEA, CEP, FEP, and IFEP: average number of iterations.

The outcomes indicated better performance of NWEA in comparison to CEP, FEP, and IFEP. For all considered ANN topologies, the convergence speed of NWEA was at least two times faster than that of CEP, FEP, and IFEP. Such a performance of NWEA is conditioned by the efficiency of the mutation mechanism that makes efficient adjustments depending on the features of a particular individual. In contrast to CEP, FEP, and IFEP, NWEA improves individuals based not only on their genotypic characteristics, but also on phenotypic ones. By involving phenotype information, NWEA "knows" more about the environment and thus, is likely to provide more suitable step sizes and reach the optimum faster. Besides, as shown in Figure 5.2 NWEA has the ability to find internal resemblances within different ANN topologies. The curve for NWEA in Figure 5.2 is monotonic, which implies that for varied number of hidden layers and the same number of neurons in them, the number of average iterations to find the optimum is almost the same. Hereby the increment of computational time is caused by the length of chromosomes and thus, by the increment of time needed to modify them. At the same time we can notice small differences in the performance of CEP, FEP, and IFEP strategies. These similarities are conditioned by the relationship between Gaussian and Cauchy distributions and the simplicity of the solving task.

ANN Architecture		**Average time,** *ms*			
Hidden layers	Hidden Neurons, per layer	**CEP**	**FEP**	**IFEP**	**NWEA**
1	2	8.8	9.0	8.8	1.7
1	3	12.6	12.4	12.3	4.8
1	4	19.4	19.5	18.1	5.9
1	5	29.1	28.4	27.4	8.0
1	6	40.9	39.1	37.0	10.4
2	2	10.3	10.4	10.0	2.6
2	3	17.8	17.3	16.4	6.2
2	4	24.7	23.4	22.4	7.8
2	5	34.3	32.6	31.4	11.0
2	6	39.1	38.4	34.3	13.8
3	2	13.5	12.8	12.7	3.8
3	3	22.0	22.3	21.0	8.1
3	4	28.2	26.6	25.4	9.7
3	5	36.0	34.1	31.4	13.2
3	6	40.6	38.7	36.5	16.7
4	2	15.0	13.9	14.1	5.0
4	3	27.2	25.2	23.6	10.1
4	4	30.8	30.3	30.0	11.7
4	5	40.6	40.1	37.1	15.8
4	6	45.7	43.7	39.0	19.7
5	2	18.5	17.6	17.5	6.6
5	3	33.1	30.8	29.6	12.4
5	4	36.8	35.1	32.4	13.5
5	5	45.7	45.4	40.0	18.3
5	6	50.9	49.1	45.0	22.8

TABLE 5.1: Convergence of NWEA, CEP, FEP, and IFEP: average convergence time.

5.1.4 Percentage of Successful Improvements

Alongside with the optimal step size, the rate of the successful improvements at each generation accelerates the evolution. The study presented in this section quantifies the average percentage of successful mutations of the NWEA, CEP, FEP, and IFEP algorithms, i.e., the percentage of the offspring individuals with higher fitness than that of their parents. For this purpose, we measured the rate of offspring that had lower error than their parents and not the offspring that reached the next generation. A fair comparison between the NWEA, CEP, FEP, and IFEP algorithms was possible as all they follow the same strategy to select individuals for the next generation, i.e., form the next generation according to $(\mu + \lambda)$ strategy by choosing best individuals from both parental and offspring populations.

Figure 5.3 demonstrates the average results of this analysis. The results show, that the average rate of successful mutations in CEP, FEP, and IFEP varies within the range of 6-14%, while NWEA improves approximately 17-24% of all individuals undergo mutation. The high rate of successful mutations increases the probability of offspring to be selected

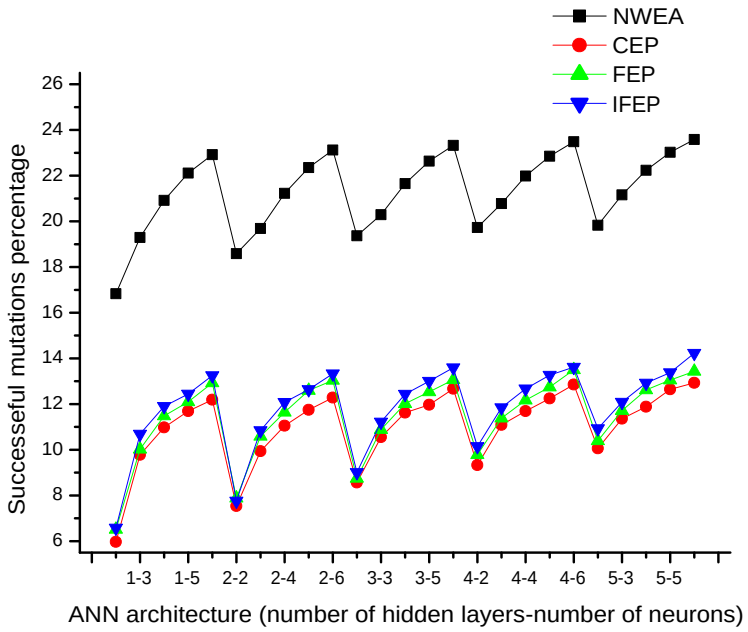


FIGURE 5.3: Average rate of successfully improved individuals.

for the next generation. This is advantageous, as the high rate of offspring in the new population not only promotes the increment of the average fitness in the next generation, but also enables examining new genetic material in the next generation.

5.1.5 Increasing Accuracy of the Evolved Solutions

In the experiments, described in Sections 5.1.3 and 5.1.4, the evolution process terminated when an individual with the error 1.0e-3.0 was found, i.e., a set of connection weights that could solve XOR with a given accuracy was evolved. In order to explore the ability of NWEA, CEP, FEP, and IFEP to find solutions with higher accuracy, we provided experiments, where the termination criteria was defined by the machine accuracy. The rest of the initial setup was the same as described in Section 5.1.1.

Figure 5.4 reports the average results over 1000 runs of each algorithm for each ANN topology. The results show that NWEA was the only algorithm that found solutions of higher accuracy, while CEP, FEP, and IFEP were unable to evolve the solution with the error equal to the machine accuracy. Note that NWEA needed incomparable less computational time and iterations to reach the maximal possible accuracy of the other algorithms. From the results can be concluded that NWEA is more resistant to local optima trapping and can be applied to evolve ANNs for the problems where the accuracy is of the highest importance.

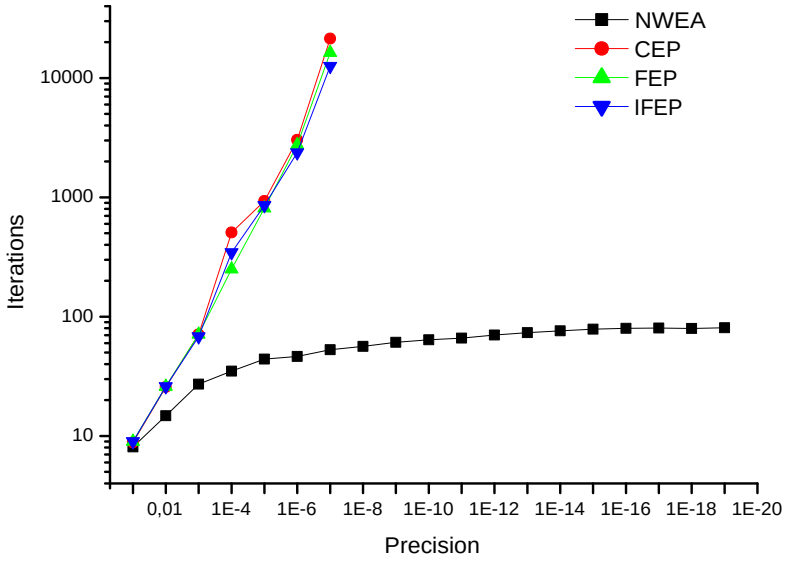


FIGURE 5.4: Convergence of NWEA, CEP, FEP, and IFEP: ability to achieve machine precision.

5.2 Evolving Connection Weights and Architectures

The evolution of connection weights is attractive in cases when an optimal ANN topology is known or the ANN size is irrelevant regarding the solving task. However, for the majority of the real-world problems optimal ANN architectures are unknown. One possible way is to let an expert choose an appropriate structure; however this is a long process (usually done by trial-and-error) and does not guarantee the positive outcome. An alternative way is to incorporate an ANN structure alongside with weights into a chromosome and thus, evolve both weights and architectures.

This section is concerned with evolving both architectures and connection weights for solving real-world problems. Four medical diagnosis problems and one forecasting task have been used to evaluate NWEA-evolved ANNs. The goal of experiments is to investigate NWEA's ability to evolve compact ANNs with good generalization ability.

Further, this section continues studies on NWEA properties and explores the effect of different types of activation function on the generalization in ANNs. Besides that, it introduces two modification of NWEA which combine Gaussian, Cauchy, and uniform distributions at the individual and the component levels.

5.2.1 Encoding Scheme for ANN Topologies and Connection Weigths

In order to represent connection weights (including biases) and architectures, we used the direct encoding scheme. Similar to the existing algorithms [162], we specified an ANN into one vector of size N and two matrices of equal size M. The vector is used to encode hidden nodes, i.e., contains entries that can be either one (a node exists) or zero (a node does not exist). The size of the hidden node vector is determined by a user-defied limit N, which is the maximal number of allowable hidden nodes. The first matrix is the connectivity matrix of an ANN that has entries either one (a connection exists) or zero (a connection does not exist). The other is the corresponding weight matrix that contains real-valued entries and specifies weights and biases. The dimension of matrices is $M \times M$, $M = N_{in} + N + N_{out}$, where N_{in} is a number of input nodes, N_{out} is a number of output nodes, and N is the maximal number of hidden nodes allowable in the ANN. Since we consider feed-forward ANNs, information in upper-right triangle of both matrices is encoded in the chromosome.

The described genotype representation scheme facilitates mutation of architectures. For instance, node deletion/addition changes a bit in the hidden node vector on its opposite value. These changes in the hidden node vector affect two matrices, e.g., node deletion disables all connections from and to the node in the connectivity matrix and sets to zero all corresponding connection weight values in the weight matrix. Connection deletion and addition involve changes in the connectivity and weight matrices, e.g., connection deletion sets to zero the connection in the connectivity matrix and disables a corresponding weight entry in the weights matrix. The mutation of connection weights is provided according to Eq. 3 as described in Section 4.3.

5.2.2 Architecture Mutation During Evolution

In NWEA, the mutation of architectures is performed only when the algorithm fails to reduce the error of the network. It is conducted in following ways: hidden node deletion, connection deletion, connection addition or node addition. Usually, connection or node addition is attempted only after node or connection deletion fails to produce good individuals [162]. This order was proposed for the purpose of supporting parsimony of already evolved ANNs.

Node Deletion

Certain hidden nodes are deleted randomly from a current ANN. The number of hidden nodes that can be deleted is specified by a user. It can be a predefined fixed number

or a random number within a given range. When a node is removed, all associated connections are also removed. Then, in order to reduce the behavioral change caused by the node deletion, a new ANN is partially trained with NWEA in the environment determined by a new topology. If the trained ANN is better than the worst ANN in both parental and offspring populations, it replaces the worst individual; otherwise connection weight deletion is performed. The minimal number of nodes that must be retained in the network is user-specified.

Connection Deletion

Certain connections are deleted probabilistically according to their importance. The number of connections that can be selected for deletion is user-specified. It can be a predefined fixed number or a random number within a given range. The importance of connection weight is defined by the significance test for the weight's deviation from zero in the weight update process [42]. We employed a nonconvergent method, proposed in [42], which defines the strength of the significance test as follows:

$$test(w_{ij}) = \frac{\sum_{p=1}^{P} w_{ij}'^{p}}{\sqrt{\sum_{p=1}^{P} (w_{ij}'^{p} - \bar{w}_{ij}')^2}}, \tag{5.1}$$

where $w_{ij}'^{p}$ is the updated value of the connection weight w_{ij}, and $\bar{w}_{ij}'$ denotes the average over the set $w_{ij}'^{p}$, $p = 1, 2, ..., P$. Higher value of $test(w_{ij})$ indicates higher importance of a connection weight. The advantage of the non-convergent method is that it does not need convergence of the training process in order to test weights. Besides, this method can also be applied for connections with zero weights and when deciding on connections to be added in the connection addition phase.

Similar to the case of node deletion, the new ANN is partially trained by NWEA. If the trained ANN is better than the worst ANN in both parental and offspring populations, it replaces the worst individual; otherwise node/connection addition is performed.

Connection Addition

Certain connections are added into the network probabilistically according to Eq. 5.1. They are chosen from those connections with zero weights and assigned with small random weights in range [-0.3; 0.3]. The new ANN is then partially trained by NWEA.

Node Addition

Node addition is implemented through a process called "cell division", which splits an already existing hidden node [110]. The main advantage of this method is that the splitting of existing nodes maintains behavioral links between parent and offspring better than the addition of a random node. The nodes are selected to be split uniformly at random among all hidden nodes. Two nodes obtained in the result of splitting have identical connections as the original node. The connection weights of the new nodes have following values:

$$\begin{aligned} w^1_{ij} &= w^2_{ij} = w_{ij}, \quad && i \geq j, \\ w^1_{ki} &= (1+\alpha) w_{ki}, && i < k, \\ w^2_{ki} &= -\alpha w_{ki}, && i < k, \end{aligned}$$

where w is the weight vector of the existing node i, w^1 and w^2 are the weights vectors of new nodes, and α is a mutation parameter which is either a fixed or random value.

The new ANN is partially trained by NWEA. As the node splitting implies that behavioral links between parents and their offspring are preserved, the offspring need little adjustment of inherited weights. Then offspring is compared to that produced by connection addition. As a result, the individual with smaller error (higher fitness) replaces the worst individual in the population.

5.2.3 Data Sets and Experimental Setup

This section introduces the benchmark problems used to evaluate the performance of NWEA-evolved ANNs and the experimental setup for these tasks.

5.2.3.1 The Mackey-Glass Chaotic Time Series Problem

The Mackey-Glass times series prediction problem [96] is a task with continuous output, and is generated by the following differential equation:

$$\frac{dx}{dt} = \beta x(t) + \frac{\alpha x(t-\tau)}{1 + x^{10}(t-\tau)},$$

where α= 0.2, β = –0.1, τ= 17 [41, 96]. As mentioned in [98], "$x(t)$ is quasiperiodic and chaotic with a fractal attraction dimension 2.1 for the above parameters". The system shows chaotic behavior when $\tau > 16.8$. The goal is to predict the output $x(t+6)$ having four past data points $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$ as the input. In order to make multiple step prediction during testing, tests with large time span $\Delta t = 90$ were provided. Iterative predictions of $x(t+6)$, $x(t+12)$, ..., $x(t+90)$ were made, where

the true value of $x(t + 6)$ was used as a target value during the training process. Such experimental setup is the same as that used by Martinetz *et al.* [98] and Yao *et al.* [162].

The fourth-order Runge-Kutta method with initial conditions $x(0) = 1.2$, $x(t - \tau) = 0$ for $0 \leq t < \tau$ and the one step at a time was used to generate data for Mackey-Glass time series. 500 patterns (of point 118 to 617) were considered as the training data, the following 500 samples were used as the testing data. No validation data were used. The values of training and testing errors were rescaled linearly to between 0.1 and 0.9.

The normalized root mean squared error (RMSE) was used to evaluate the performance of NWEA. The RMSE is determined by the absolute prediction error for $\Delta t = 6$ divided by the standard deviation of $x(t)$ [41, 98]:

$$E = \frac{\left\langle [x_{pred}(t, \Delta t) - x(t + \Delta t)]^2 \right\rangle^{\frac{1}{2}}}{\langle (x - \langle x \rangle)^2 \rangle^{\frac{1}{2}}}, \tag{5.2}$$

where $x_{pred}(t, \Delta t)$ is the prediction of $x(t + \Delta t)$ from the current state $x(t)$ and $\langle x \rangle$ represents the expectation of x. As stated in [41], the prediction is perfect if $E = 0$; if $E = 1$, the prediction is not better than a constant predictor $x_{pred}(t, \Delta t) = \langle x \rangle$.

The following NWEA parameters were used in Mackey-Glass time series prediction experiments: the population size 30, the maximum number of generations 200, the number of hidden nodes for each individual in the initial population was generated uniformly at random between 8 and 16, the number of mutated hidden nodes 1, and the number of generations for the partial learning of ANNs with the mutated architecture 5.

5.2.3.2 The Breast Cancer Data Set

The breast cancer data set was originally obtained from Dr. William H. Wolberg at the University of Wisconsin Hospitals, Madison. The data set consists of 699 examples of which 458 (65.5%) are benign examples and 241 (34.5%) are malignant examples. Each example contains 9 attributes: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitoses. The goal of the data set is to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination.

In the experiments, the whole data set was divided into three subsets, as suggested by Prechelt [115]: a training set, a validation set, and a testing set. The first set was used to train ANNs. The validation set was explored as a pseudo-testing set in order to evaluate the fitness of networks during evolution. This prevents overtraining of the network and usually improves its generalization ability. During this process ANN's learning is

carried out until the minimal error on the validation set (note, not on the training set) is achieved. Finally, the testing data were considered to evaluate the performance of the evolved ANNs. 349 examples of the given breast cancer data set were used as the training set, the following 175 examples as the validation set, and the final 175 patterns as the testing set.

The error of each individual was calculated by the equation, proposed by Prechelt [115],[1] over a validation set containing P patterns:

$$E = 100 \cdot \frac{o_{\max} - o_{\min}}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2, \tag{5.3}$$

where $o_{\min}$ and $o_{\max}$ are the minimum and maximum values of output coefficients in the problem representation. N is the number of output nodes, o_{pi} and t_{pi} are the actual and desired outputs of node i for pattern p. The fitness of each individual was determined by the inverse of the error.

Following parameters were used for the experiments: the population size 30, the maximum number of generations 200, the initial node connection density 1.0, the number of mutated hidden nodes 1, the number of mutated connections 1 to 3, the number of generations for the partial learning of ANNs with the mutated architecture 5. The number of initial hidden nodes was generated uniformly at random between 1 and 3. The output attributes were encoded by 1-of-m output representation for m classes. We used the winner-takes-all method, where the output with the highest activation designates the class.

5.2.3.3 The Heart Disease Data Set

The heart disease data set was obtained from Cleveland Clinic Foundation and was supplied by Robert Detrano of the V.A. Medical Center, Long Beach, CA. The data set consists of 270 examples. The heart disease data set originally consisted of 303 examples, but 6 of them contained missing class values and were excluded from the database. Other 27 examples of the remained data were eliminated as they retained in case of dispute.

Each example in the database contains 13 attributes, which present results of medical tests provided on patients: age, sex, chest pain type, resting blood pressure, cholesterol, fasting blood sugar < 120 (true or false), resting electrocardiogram (norm, abnormal or hyper), max heart rate, exercise induced angina, oldpeak, slope, number of vessels

[1] The error function was proposed in order to decrease the dependence of the error measure on the size of the validation set and the number of outputs.

colored and thal (normal, fixed, rever). These attributes have been extracted from a larger set of 75. The goal of diagnosis is to recognize the presence or absence of heart disease given the attributes. Initially, the data set considered four different degrees of the heart disease to classify the output results. The later modification in the problem definition suggested reducing the number of predicted values on two and categorizing results into two classes: presence or absence of illness.

For the heart disease diagnosis, the first 134 examples of the entire data set were used for the training set, the following 68 examples for the validation set, and the final 68 examples for the testing set. The input attributes were linearly rescaled to between 0.0 and 1.0. The other experimental parameters, as well as the error function were the same as for the breast cancer diagnosis, except for the initial number of hidden nodes, which was generated uniformly at random between 3 to 5.

5.2.3.4 The Diabetes Data Set

The diabetes data set was constructed by Vincent Sigillito from Johns Hopkins University. He collected the data set by constrained selection of data from a large database held by the National University of Diabetes and Digestive and Kidney Disease. The selected data set represents the test results of female patients of at least 21 years old and of Pima Indian heritage living near Phoenix, AZ.

Each example contains eight input attributes: number of times pregnant, plasma glucose concentration in an oral glucose tolerance test, diastolic blood pressure, triceps skin hold thickness, 2-hour serum insulin, body mass index, diabetes pedigree function and age. The goal is to predict whether a patient would diagnose positive for diabetes according to World Health Organization criteria given a number of physiological measurements and medical test results. The classification to be made between two classes, where the class value one is considered as "tested positive for diabetes" and class value two as "tested negative for diabetes". There are 500 examples of class one and 268 patterns of class two in the data set. The classification of the diabetes data set is a challenging problem, as the so-called "class" value is a binarised form of another attribute, which is itself "highly indicative of certain types of diabetes but does not have a one to one correspondence with the medical condition of being diabetic" [162].

For the diabetes diagnosis, the first 384 examples of diabetes data set were used for training set, the following 192 examples for the validation set and the final 192 patterns for the testing set. The input attributes of the diabetes data set, similar to the heart disease data set, was rescaled to between 0.0 and 1.0 by a linear function. The other experimental parameters, as well as the error function were the same as for the breast

cancer diagnosis, except for the initial number of hidden nodes, which was generated uniformly at random between 2 to 8.

5.2.3.5 The Thyroid Data Set

The thyroid data set is the "ann" version of the "thyroid disease" data set from the UCI machine learning repository. This data set consists of two files. The first file, "ann-train.data" contains 3772 training examples. Another one, "ann-test.data" contains 3428 testing examples. Each example has 21 attributes (15 binary and 6 continuous). The goal of this data set is to determine whether a patient is hypothyroid. Therefore three classes are built: normal (not hypothyroid), hyperfunction and subnormal functioning. The problem is very challenging for any classifier, as 92% of the patients are not hyperthyroid. This means that a good classifier must be significantly better than 92%.

The data set from "ann-train.data" was divided into two subsets: the first 2514 examples were used for the training set, the rest 1258 examples for the validation set. The whole data from "ann-test.data" were used for the testing set. The experimental parameters, as well as the error function were the same as for the breast cancer diagnosis, except for the initial number of hidden nodes, which was generated uniformly at random between 6 to 15.

5.2.4 Evolving ANNs with NWEA: Results and Comparative Analysis

This section presents the experimental results on the studied benchmark problems and compares them to those obtained by other algorithms. Since the direct comparison with other evolutionary approaches to evolving ANNs is very difficult [162], we compared the results of NWEA-evolved ANNs to those available in the literature, regardless of a type of the training algorithm, i.e., whether it was evolutionary, gradient-descent, hybrid, etc.

Mackey-Glass Chaotic Time Series Prediction

Tables 5.2 and 5.3 present the results of the NWEA-evolved ANNs over 30 runs for the Mackey-Glass chaotic time series problem. Tables 5.4 and 5.5 report the comparative results of different learning strategies for this benchmark problem.[2] The data are taken from [19, 21, 22, 78, 89, 95, 98, 121, 162]. The symbol "–" in the column "Connections" refers to the values that are not available.

[2] As common in the literature, Table 5.5 presents the comparative results for the time span $\Delta t = 84$.

The following observations can be made from the obtained results. The results for a small time span $\Delta t = 6$ demonstrated very competitive generalization ability of networks evolved by NWEA. The best testing error of the NWEA-evolved ANN was 0.01220 and the average was 0.01426, which means that the evolved ANNs leant very good the underlined function. This is also corroborated by the insignificant distinctions between the training and testing errors (0.01356 and 0.01426, respectively). The comparison to other learning strategies demonstrated very competitive results of NWEA-evolved ANNs; however, some of the existing strategies showed smaller testing error than NWEA (see Table 5.4). It is worth noting that some of these algorithms were developed with respect to the time series prediction problem. For instance, the FNT algorithm [19] uses adapted to this problem exponential function instead of sigmoid to transform neurons incoming signals, which increases precision of prediction, but takes long computation time. The classical RBF method [21] showed the comparable results, but the size of the evolved ANNs was much larger than of those evolved by NWEA (23 and 10.97 hidden neurons, respectively). The modified RBF, i.e., PG-RBF [121] showed better results than NWEA; however, it was achieved by an ANN with 12 neurons.

For a large time span $\Delta t = 84$, NWEA-evolved ANNs also provided favorable results comparing to those of EPNet [162], BP [89] and CC [22] learning methods (Table 5.5).[3] The average prediction error of NWEA-evolved ANNs on 500 training data points over 30 runs was 0.03765, while the average accuracy of EPNet and BP networks was 0.06 and 0.05, respectively. NWEA-evolved ANNs also outperformed the results of the "neural-gas" networks [98]. The "neural-gas" network needed 1000 training data points to achieve the error of 0.05, while for the training set of 500 examples its smallest prediction error was 0.06. However, NCNNs [95] with the average accuracy 0.03 (0.0326) performed better than NWEA-evolved ANNs, though the best results for both networks were the same, 0.03 (0.02836 for NWEA-evolved ANNs and 0.0279 for NCNNs).

Besides evolving networks with good generalization ability, NWEA produced more compact networks than other approaches. The average number of connections of NWEA-based ANNs was 98.21, while the average ANNs evolved by EPNet had 103.33 connections (the smallest ANNs had 64 and 66 connections, respectively). The ANNs evolved by BP and CC learning methods were extremely large (540 and 693 connections respectively), the average size of "neural-gas" networks had about 500 connections (using the training data set of 1000 points) and 1800 connections to achieve the smallest error for the training data set of 500 examples.

[3]The other algorithms from Table 5.4 do not provide results for the multiple step prediction.

	min	max	mean	SD
Connections	64	142	98.21	20.3
Hidden nodes	8	14	10.97	1.7
Generations	86	186	122	23.6

TABLE 5.2: ANN architectures for the Mackey-Glass time series problem.

	min	max	mean	SD
Training error	0.01217	0.01682	0.01356	0.00162
Testing error, $\Delta t = 6$	0.01220	0.01874	0.01426	0.00177
Testing error, $\Delta t = 84$	0.02836	0.06832	0.03765	0.00743
Testing error, $\Delta t = 90$	0.03215	0.07373	0.04382	0.00883

TABLE 5.3: Prediction accuracy for the Mackey-Glass time series problem.

Algorithm	**Connections**	**Testing error**
NWEA	98	0.01 (0.01426)
EPNet	103	0.02 (0.0205)
BP	540	0.02
CC Learning	693	0.06
NCNN	-	0.01 (0.100)
FNT model (Case 1)	-	0.0069
Autoregressive model	-	0.19
Sixth-order polynomial	-	0.04
Linear prediction method	-	0.55
ANFIS and Fuzzy System	-	0.007
Product T-norm	-	0.0907
Classical RBF (with 23 neurons)	-	0.0114
PG-RBF network (with 12 neurons)	-	0.00287
Genetic algorithm and fuzzy system	-	0.049

TABLE 5.4: Comparative results of the prediction accuracy for the Mackey-Glass time series problem with a time span $\Delta t = 6$.

Algorithm	**Connections**	**Testing error**
NWEA	98	0.04
EPNet	103	0.06
BP	540	0.05
CC Learning	693	0.32
NCNN	-	0.03

TABLE 5.5: Comparative results of the prediction accuracy for the Mackey-Glass time series problem with a time span $\Delta t = 84$.

Breast Cancer Diagnosis

Tables 5.6 and 5.7 report the results of NWEA-evolved ANNs over 30 runs for the breast cancer diagnosis problem. The error in Table 5.7 is defined by Eq. 5.3 and the error rate indicates the percentage of wrong classified examples. Table 5.8 presents the comparative analysis between NWEA-evolved ANNs, EPNet [158, 165], an ANN constructive algorithm, called FNNCA [136] and the hand-designed ANN (referred to

as HDANNS) [115] obtained by the trial-and-error method. The average results of NWEA-evolved ANNs were compared to those of EPNet and the best results produced by FNNCA (in 50 runs) and HDANNS.

The results showed that NWEA evolved compact ANNs with good generalization ability. Although the classification accuracy of NWEA-evolved ANNs is comparable to that of EPNet and the best results of other algorithms (though the minimal testing error rate of NWEA-evolved ANNs was 0.0%), it is clear from Table 5.8 that NWEA designed more compact ANNs. The average number of hidden nodes in NWEA-evolved ANNs was 1.3, which is almost twice as small as the average topology of EPNet and the best network of FNNCA and more than four times smaller than the best HDANNS. The average number of connections in NWEA-evolves ANNs was 36, while EPNet had 41 connections (the number of FNNCA and HDANNS connections is not available). One can argue that the size of the network is inessential once its generalization accuracy is high; indeed, this assertion is correct for large or complex problems, where good generalization itself is a challenge. However, the breast cancer data set, studied in this thesis, is a comparatively simple problem due to the small number of attributes [162]. From this point of view, the ability of an algorithm to evolve compact networks that generalize well is advantageous.

	min	max	mean	SD
Connections	14	78	36	12.3
Hidden nodes	0	4	1.3	0.8
Generations	101	192	139.1	32.8

Table 5.6: ANN architectures for breast cancer diagnosis.

	min	max	mean	SD
Training error	1.418	3.650	2.722	0.464
Training error rate	0.01698	0.04672	0.03921	0.00637
Validation error	0.052	1.018	0.557	0.178
Validation error rate	0.00000	0.01072	0.00547	0.00145
Testing error	0.178	3.546	1.413	0.708
Testing error rate	0.00000	0.03397	0.01384	0.00942

Table 5.7: Classification accuracy for breast cancer diagnosis.

Algorithm	**Hidden nodes**	**Testing error rate**
NWEA	1.3	0.01384
EPNet	2.0	0.01376
FNNCA (best result)	2.0	0.0145
HDANNS (best result)	6.0	0.01149

Table 5.8: Comparative results of the classification accuracy for breast cancer diagnosis.

Heart Disease Diagnosis

The competitive results of NWEA-evolved ANNs on the breast cancer problem encouraged further studies of the proposed algorithm on more difficult tasks. In comparison to the breast cancer problem, the heart disease problem has larger number of attributes (13 attributes, see Section 5.2.3.3) and thus, is a more complex task [162]. This makes more difficult to learn the underlined function that specifies dependencies between the attributes and the output.

The results of NWEA-evolved ANNs over 30 runs for the heart disease diagnosis are presented in Tables 5.9 and 5.10. Similar to the breast cancer problem, the error rate in Table 5.10 shows the rate of wrong classifications. Table 5.11 presents the generalization results comparison among NWEA-evolved ANNs, EPNet [162], GM-constructed ANNs, MSM1, MSM algorithms, BP-trained ANNs [10] and manually designed HDANNS [115].

The smallest rate of wrong classifications on the testing set was 0.13195 (13.2%), while the average error rate was 0.15165 (15.17%). The average classification error, reported by EPNet and MSM1 are 16.77% (0.16765) and 16.53% (0.1653) respectively, which are worse than the average results of the NWEA-produced ANNs. The RBF networks constructed with GM algorithm had 18.18% testing error rate, the MSM method - 25.92% and BP reported about 25% of wrong classifications. These results are worse than the maximal testing error rate of NWEA-evolved ANNs. The best HDANNS achieved the classification rate of 14.78%, which is worse than that of the NWEA-based ANN. At the same time smallest ANNs evolved by NWEA had 1 hidden node and 28 connections, while in average ANNs had 4.0 hidden nodes and 88.4 connections. These results are comparable with those of EPNet (4.1 and 1 hidden nodes, respectively) and outperform the best result of HDANNS (4 hidden nodes). However, the average number of connections in NWEA-evolved ANNs was 88.4, while in average EPNet had 92.6 connections.

	min	max	mean	SD
Connections	28	202	88.4	37.6
Hidden nodes	1	8	4.0	1.9
Generations	127	200	172.2	53.2

Table 5.9: ANN architectures for heart disease diagnosis.

Diabetes Diagnosis

The diabetes diagnosis is recognized as one of the most challenging problems in ANN and machine learning due to its relatively small data set and high noise level [162].

	min	max	mean	SD
Training error	7.489	12.166	11.007	0.693
Training error rate	0.07879	0.15184	0.12477	0.01465
Validation error	11.746	14.301	12.450	0.468
Validation error rate	0.12124	0.19706	0.15935	0.01736
Testing error	10.126	13.842	12.266	0.701
Testing error rate	0.13195	0.17997	0.15165	0.01772

TABLE 5.10: Classification accuracy for heart disease diagnosis.

Algorithm	**Hidden nodes**	**Testing error rate**
NWEA	4.0	0.15165
EPNet	4.1	0.16765
GM	-	0.1818
MSM1	-	0.1653
MSM	-	0.2592
BP	-	0.25
HDANNS (best result)	4.0	0.1478

TABLE 5.11: Comparative results of the classification accuracy for heart disease diagnosis.

Generally, the medical data are very costly to obtain; thus it is inefficient to increase the training data to improve generalization of a classification approach. This implies that a good algorithm should not rely on the large training set.

Tables 5.12 and 5.13 report the results of NWEA-evolved ANNs over 30 runs for the diabetes diagnosis. Similar to the previous classification tasks, the error is defined by Eq. (5.3) and the error rate shows the rate of incorrect classifications. Table 5.14 compares the results of NWEA-evolved ANNs with those produced by other algorithms [102, 115, 162]. All considered algorithms except for EPNet [162] and HDANNS [115] represent the best methods out of 23 techniques tested in [102]. The results of these algorithms were obtained by 12-fold cross-validation [102].

As shown in Table 5.14, the NWEA-evolved networks outperformed the compared algorithms in terms of generalization. The average rate of wrong classifications of NWEA-evolved ANNs was 0.18074 (18.07%), while the testing error rate of the other methods were higher than 22%. The best accuracy reported by EPNet and HDANNS were 19.27% (0.19271) and 21.35%, respectively, while the best error achieved by NWEA-evolved ANNs was 16.71% (0.16711).

In terms of the network size, NWEA also showed favorable results. The average ANN, produced by NWEA had 2.7 hidden nodes, the largest had 5 nodes and the smallest just 1 node. The smallest NWEA-evolved ANN is comparable to that of EPNet (1 hidden node), the average and the largest ANNs evolved by EPNet had 3.4 and 6 nodes,

respectively. At the same time the highest classification accuracy of HDANNS was achieved by the network with 8 hidden neurons.

	min	max	mean	SD
Connections	22	69	42.9	12.7
Hidden nodes	1	5	2.7	1.1
Generations	83	173	102.4	42.2

TABLE 5.12: ANN architectures for diabetes diagnosis.

	min	max	mean	SD
Training error	14.723	18.177	16.069	0.272
Training error rate	0.14352	0.23934	0.19827	0.00010
Validation error	12.304	14.129	13.026	0.405
Validation error rate	0.13882	0.19911	0.16288	0.00006
Testing error	13.137	14.625	13.788	0.243
Testing error rate	0.16711	0.23418	0.18074	0.00084

TABLE 5.13: Classification accuracy for diabetes diagnosis.

Algorithm	**Hidden nodes**	**Testing error rate**
NWEA	2.7	0.18074
EPNet	3.4	0.22379
Logdisc	-	0.223
DIPOL92	-	0.224
Discrim	-	0.225
SMART	-	0.232
RBF	-	0.243
ITrule	-	0.245
BP	-	0.248
Cal5	-	0.250
CART	-	0.255
CASTLE	-	0.258
Quadisc	-	0.262
HDANNS (best result)	8	0.2135

TABLE 5.14: Comparative results of the classification accuracy for diabetes diagnosis.

Thyroid Diagnosis

Tables 5.15 and 5.16 present the results of NWEA-evolved ANNs over 30 runs for the thyroid diagnosis. As stated in Section 5.2.3.5, a good classifier for the thyroid problem must have an accuracy over 92%. The minimal classification error rate achieved by NWEA-evolved ANNs was 0.01167 (1.17%) and the average error rate was 0.01628 (1.63%), which indicates a superior performance of NWEA in developing networks of high classification accuracy.

Table 5.17 reports the comparison results between NWEA-evolved ANNs, EPNet [162], ANNs trained with a modified GA [130] and HDANNS [115]. It is obvious that NWEA-evolved ANNs performed better than other algorithms. The average classification error

showed by EPNet and the ANNs trained by the modified GA were 2.12% (0.02115) and 2.5%, respectively. The minimal rate of wrong classification reported by EPNet was 1.634% and 1.278% by HDANNs, which is worse than the best result of NWEA-evolved ANNs.

The results on ANN architectures evolved by NWEA also compare favorable to those produced by other algorithms. The NWEA-evolved ANNs had on average 4.9 hidden nodes and 181.6 connections, while ANNs designed by EPNet had 5.9 hidden nodes and 219.6 connections, and the ANN evolved with the modified GA had 50 hidden nodes and 278 connections. The highest accuracy of hand-designed HDANNS was shown by the ANN with 12 hidden neurons.

	min	max	mean	SD
Connections	115	378	181.6	64.9
Hidden nodes	3	10	4.9	1.7
Generations	15	92	52.6	16.2

TABLE 5.15: ANN architectures for thyroid diagnosis.

	min	max	mean	SD
Training error	0.178	0.685	0.389	0.073
Training error rate	0.00326	0.01231	0.00719	0.00117
Validation error	0.361	0.893	0.570	0.086
Validation error rate	0.00497	0.01244	0.07341	0.00197
Testing error	0.677	1.162	0.896	0.093
Testing error rate	0.01167	0.01912	0.01628	0.00155

TABLE 5.16: Classification accuracy for thyroid diagnosis.

Algorithm	**Hidden nodes**	**Testing error rate**
NWEA	4.9	0.01628
EPNet	5.9	0.02115
Modified GA	50	0.025
HDANNS (best result)	12	0.01278

TABLE 5.17: Comparative results of the classification accuracy for thyroid diagnosis.

5.2.5 Exploring the Impact of Activation Function Type

The following sections investigate the evolution process and the quality of the obtained networks under different characteristics of a learning algorithm and an ANN architecture. In this section the impact of a particular type of neuron activation function on the performance of NWEA-evolved ANNs was studied. More specifically, we investigated the opportunity of utilizing a linear function as a node transfer function in ANNs evolved by the evolutionary methodology.

The node activation function has been shown to be an important part of an architecture, as it has significant impact on ANNs' performance [93]. As described in Section 2.2.2, the essence of a learning algorithm is the learning rule, which determines how connection weights are changed. The gradient descent methods, e.g., back-propagation, employ gradient descent to minimize the error function, and put strong limitation on a type of the neuron activation function, that requires it to be differentiable (the derivative must be continuous). The perfect choice here are sigmoid functions, such as logistic, hyperbolic tangent, etc., which besides being differentiable, add small nonlinearity in the network so that it do not change its result significantly.

In contrast to back-propagation, EAs involve principles and mechanisms of natural evolution in the optimization process and do not require gradient information, which makes them less dependent on a type of the activation function. On the other hand, the signal processing is controlled by the threshold values, which are evolved and adapted alongside with connection weights during the evolution. Thus, the use of sigmoid activation function is not strongly motivated for EANNs. That means, it is possible to use linear functions, which besides their simplicity might reduce computational time of each cycle of evolution (however, this does not guarantee reduction of the total time needed to find an optimal solution and moreover, better quality of the obtained networks). Additionally, the shape of some linear functions, e.g., piecewise linear function, resembles that of the sigmoid functions [92], which implies that the outcome of the piecewise linear function should not deviate much from that of the logistic function.

In order to examine the role of an activation function in EANNs, we compared performances of NWEA-evolved ANNs with the logistic, hyperbolic tangent (both sigmoid functions), and piecewise linear functions. The piecewise linear function was defined by:

$$f_i\left(\sum_{j=1}^{n} x_j w_{ij} - \theta_i\right) = \begin{cases} 0 : \left(\sum_{j=1}^{n} x_j w_{ij} - \theta_i\right) < -1 \\ \left(\sum_{j=1}^{n} x_j w_{ij} - \theta_i\right) : -1 \leq \left(\sum_{j=1}^{n} x_j w_{ij} - \theta_i\right) < 1 \\ 1 : \left(\sum_{j=1}^{n} x_j w_{ij} - \theta_i\right) \geq 1 \end{cases} .$$

The empirical analysis was carried out for the Mackey-Glass chaotic time series (with the time span $\Delta t = 6$), the breast cancer and the heart disease problems. Since NWEA-evolved ANNs with the sigmoid transfer function were studied in Section 5.2.4, we applied NWEA to evolve ANNs with hyperbolic tangent and linear activation functions. The experimental setup for the considered problems was similar to that in the previous section, including the same initial populations. For a certain type of ANN, 30 runs of NWEA were provided. The results were evaluated in terms of size of the evolved ANNs and their generalization ability.

	tahn(x)	**piecewise linear**
Connections, min	64	64
Connections, max	146	138
Connections, mean	102.1	98.79
Connections, SD	22.1	20.7
Hidden nodes, min	8	8
Hidden nodes, max	16	14
Hidden nodes, mean	11.27	10.73
Hidden nodes, SD	1.9	1.7
Generations, min	94	85
Generations, max	188	181
Generations, mean	148.5	117.2
Generations, SD	20.4	21.8

TABLE 5.18: Utilizing different activation functions: ANN architectures for the Mackey-Glass time series problem.

	tanh(x)	**piecewise linear**
Training error, min	0.01228	0.01239
Training error, max	0.01816	0.01721
Training error, mean	0.01381	0.01401
Training error, SD	0.00196	0.00185
Testing error, min	0.01230	0.01231
Testing error, max	0.01900	0.01814
Testing error, mean	0.01418	0.01443
Testing error, SD	0.00182	0.00187

TABLE 5.19: Utilizing different activation functions: prediction accuracy for the Mackey-Glass time series problem.

	logistic	**tanh(x)**	**piecewise linear**
Connections, min	64	64	64
Connections, mean	98.21	102.1	98.79
Hidden nodes, min	8	8	8
Hidden nodes, mean	10.97	11.27	10.73
Testing error, min	0.01220	0.01230	0.01231
Testing error, mean	0.01426	0.01418	0.01443

TABLE 5.20: Comparative results of ANNs with logistic, hyperbolic tangent and piecewise linear activation functions for the Mackey-Glass time series problem.

5.2.5.1 Results and Discussions

Tables 5.18, 5.19, 5.21, 5.22, 5.24, 5.25 report the results of the networks with hyperbolic tangent and piecewise linear functions for the studied problems. Tables 5.20, 5.23, 5.26 present the comparative results between ANNs with different activation functions.

Statistical analysis of data reported in Tables 5.20, 5.23, 5.26 with t-test indicated no significant distinctions between the best and the average results of NWEA-evolves ANNs with the studied activation functions. All networks, independent of an activation function's type, demonstrated good generalization ability and high prediction/classification

	tanh(x)	**piecewise linear**
Connections, min	15	15
Connections, max	82	84
Connections, mean	41	38
Connections, SD	13.7	12.7
Hidden nodes, min	1	1
Hidden nodes, max	5	4
Hidden nodes, mean	1.6	1.4
Hidden nodes, SD	1.1	0.9
Generations, min	93	89
Generations, max	187	185
Generations, mean	142.7	126.2
Generations, SD	27.2	34.5

TABLE 5.21: Utilizing different activation functions: ANN architectures for breast cancer diagnosis.

	tanh(x)				**piecewise linear**			
	min	max	mean	SD	min	max	mean	SD
Training error	1.428	3.769	3.214	0.376	1.422	3.677	2.637	0.411
Training error rate	0.01720	0.04818	0.04212	0.00607	0.01712	0.04823	0.02886	0.00529
Validation error	0.053	1.034	0.691	0.174	0.048	1.026	0.544	0.169
Validation error rate	0.00000	0.01156	0.00602	0.00236	0.00000	0.01069	0.00526	0.00338
Testing error	0.172	3.718	1.502	0.713	0.169	3.550	1.427	0.706
Testing error rate	0.00000	0.03875	0.01363	0.01108	0.00000	0.03617	0.01391	0.01043

TABLE 5.22: Utilizing different activation functions: classification accuracy for breast cancer diagnosis.

	logistic	**tanh(x)**	**piecewise linear**
Connections, min	14	15	15
Connections, mean	36	41	38
Hidden nodes, min	0	1	1
Hidden nodes, mean	1.3	1.6	1.4
Testing error, min	0.178	0.172	0.169
Testing error, mean	1.413	1.502	1.427
Testing error rate, min	0.00000	0.00000	0.00000
Testing error rate, mean	0.01384	0.01363	0.01391

TABLE 5.23: Comparative results of ANNs with logistic, hyperbolic tangent and piecewise linear activation functions for breast cancer diagnosis.

accuracy in solving the benchmark problems. Besides, for all types of ANNs NWEA needed approximately the same number of generations to find the optimal solution. However, NWEA needed in average 4% less time to evolve ANNs with the piecewise linear function.

Several conclusions can be drawn from the obtained results. Firstly, results demonstrated that the functionality of an EANN does not strongly depend on the activation function's type.[4] Indeed, the search for the optimum is provided by the evolutionary

[4]Although this conclusion is made for NWEA, it is likely to be true for all evolutionary methodologies.

	tanh(x)	**piecewise linear**
Connections, min	20	28
Connections, max	212	196
Connections, mean	92.8	87.6
Connections, SD	42.6	40.7
Hidden nodes, min	1	1
Hidden nodes, max	10	8
Hidden nodes, mean	5.1	3.8
Hidden nodes, SD	2.4	1.6
Generations, min	134	118
Generations, max	235	248
Generations, mean	175.9	191.4
Generations, SD	49.6	57.4

TABLE 5.24: Utilizing different activation functions: ANN architectures for heart disease diagnosis.

	tanh(x)				**piecewise linear**			
	min	max	mean	SD	min	max	mean	SD
Training error	8.288	12.345	11.282	0.733	7.842	11.967	10.412	0.725
Training error rate	0.09355	0.16192	0.13384	0.01461	0.07936	0.14652	0.11987	0.01478
Validation error	12.326	14.518	12.794	0.524	12.268	15.062	13.456	0.538
Validation error rate	0.13137	0.20403	0.17641	0.01915	0.12711	0.20064	0.16138	0.01847
Testing error	11.014	14.141	12.720	0.725	10.592	13.812	12.488	0.713
Testing error rate	0.13157	0.18246	0.15220	0.01874	0.12793	0.18217	0.15183	0.01819

TABLE 5.25: Utilizing different activation functions: classification accuracy for heart disease diagnosis.

	logistic	**tanh(x)**	**piecewise linear**
Connections, min	28	20	28
Connections, mean	88.4	92.8	87.6
Hidden nodes, min	1	1	1
Hidden nodes, mean	4.3	5.1	3.8
Testing error, min	10.126	11.014	10.592
Testing error, mean	12.266	12.720	12.488
Testing error rate, min	0.13195	0.13157	0.12793
Testing error rate, mean	0.15165	0.15220	0.15183

TABLE 5.26: Comparative results of ANNs with logistic, hyperbolic tangent and piecewise linear activation functions for heart disease diagnosis.

learning algorithm, which places less emphasis on a type of the activation function. During evolution, the activation function is a part of the error function, needed to determine the fitness of individuals, and not a part of the improvement mechanism (like in gradient descent methods). Secondly, although the best and average time needed to produce the optimal networks is not reported, it is obvious that for the same number of iterations, an algorithm that evolves networks with piecewise linear function converges faster than ANNs with the sigmoid function, given that other parameters are the same. It is conditioned by the fact that the linear function does not require computational time to modify the signal. Finally, it is reasonable to conclude that the type of the

node transfer function should be decided depending on the requirements and conditions of a solving task. For instance, utilization of sigmoid functions might be beneficial for the problems that require high accuracy of prediction, while the piecewise linear transfer function would be preferred for the problems, which require finding of acceptable solutions within a short period of time.

5.2.6 Mixing Different Search Biases in NWEA

In this section we investigated the impact of mixing search biases of NWEA modifications based on three different distributions, on the generalization of the evolved ANNs. The idea of using different distributions in the evolutionary algorithm was studied by Yao *et al.* in [164]. The analysis of the mixed EP algorithms, i.e., IFEP and MEP (see Section 3.4) indicated their consistent performance, excellent scalability and robustness in comparison with CEP and FEP.

Inspired by the results of IFEP and MEP, we observed the evolution under different step sizes determined by mixed distributions. Two modifications of NWEA, which combine Gaussian, Cauchy and uniform distributions at the chromosome and gene levels have been examined with respect to the ANNs' generalization ability. The first modification, referred to as combined NWEA (CNWEA) produces *three offspring* from each parent by applying three mutation mechanisms based on different distributions. CNWEA performs mutation at the chromosome level and modifies all genes in the chromosome by NWEA with either Gaussian, Cauchy or uniform distribution. The second modification of NWEA, called mixed NWEA (MNWEA), applies three mutation operators with different distributions, each with a certain probability, to generate *one offspring*, i.e., provides mutation at the gene level.

The utilization of combined search strategies in the ANNs training implies that different step sizes determined by mixed distributions will efficiently direct the evolution towards good solutions. In order to examine it, the generalization ability of CNWEA- and MNWEA-evolved ANNs has been tested on the breast cancer and heart disease diagnosis problems. For each problem 30 runs of each of studied algorithms were provided. The experimental setup was the same as described in Sections 5.2.3.2 and 5.2.3.3, except the population size of 10 for CNWEA (since CNWEA produces three offspring from each parent). The results were compared with those produced by NWEA (see Section 5.2.4).

5.2.6.1 Length of Gaussian and Cauchy Jumps

As discussed in Chapter 3, differences between the performances of CEP and FEP on various function optimization problems with dimension 30 are conditioned by different step sizes determined by Gaussian and Cauchy distributions. The expected length of Gaussian (with $\mu = 0$ and $\sigma^2 = 1$) and Cauchy (with $\gamma = 1$) jumps can be calculated by integrating their probability density functions:

$$\begin{aligned} E_G(x) &= 2\int_0^{+\infty} x\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}dx = \frac{2}{\sqrt{2\pi}} = 0.80 \\ E_C(x) &= 2\int_0^{+\infty} x\frac{1}{\pi\,(1+x^2)}dx = +\infty. \end{aligned}$$

Apparently, the Cauchy distribution enables longer jumps than the Gaussian one. At first sight it seems that longer jumps in the search space induce faster convergence, and so the Cauchy distribution is preferable in the searching strategy. However, this assumption is wrong. The analytical studies in [157] concluded that long jumps are beneficial only when the global optimum is far away from the current search point. In other words, long jumps are effective when the distance between the global optimum and the current point is larger than the mutation's step size. On the other side, the Cauchy distribution is no longer advantageous when the distance between the neighborhood of the global optimum and the current point is smaller than the step size of the mutation. This implies that the use of small jumps is more effective near the neighborhood of the global optimum. Hence, the Gaussian distribution increases the probability of finding the optimum when the distance between the current point and the neighborhood of the global optimum is small.

5.2.6.2 Combined NWEA

The main idea behind combined NWEA (CNWEA) is to mix different search biases of mutations that utilize Gaussian, Cauchy and uniform random numbers at the individual level. The implementation of CNWEA is simple and differs from NWEA only in point 3 of the algorithm described in Section 4.3. Each parental individual is modified three times and thus, produces three offspring by applying differently distributed random values N^D_{Rand} in Eq. 3. The improving mechanisms described by equations below are combined at the chromosome level, as each of mutations is applied to *all* genes (components) of an individual.

The first offspring is created by using normally distributed values $N_{Rand}^{D_G}$ with mean $\mu = 0$ and variance $\sigma^2 = 1$, i.e.,

$$x_i^{(j)'} = x_i^{(j)} \left(1.0 + N_W(l, \bar{n}) \cdot N_F \cdot N_{Rand}^{D_G}\right), \tag{5.4}$$

the second offspring is defined by using Cauchy random numbers $N_{Rand}^{D_C}$ with a scale parameter $\gamma = 1$, i.e.,

$$x_i^{(j)'} = x_i^{(j)} \left(1.0 + N_W(l, \bar{n}) \cdot N_F \cdot N_{Rand}^{D_C}\right), \tag{5.5}$$

and the third offspring is created by utilizing uniformly distributed random values, $N_{Rand}^{D_U} \in$ [-1.0, 1.0]:

$$x_i^{(j)'} = x_i^{(j)} \left(1.0 + N_W(l, \bar{n}) \cdot N_F \cdot N_{Rand}^{D_U}\right). \tag{5.6}$$

The rest of the algorithm is exactly the same as NWEA (see Section 4.3 from point 4).

5.2.6.3 Mixed NWEA

An alternative way of combining different biases is to apply mutation operators based on Gaussian, Cauchy and uniform distributions in order to create one offspring individual, i.e., to mix them at the component level. In this modification of NWEA, called mixed NWEA (MNWEA) certain probabilities P_G, P_C and P_U are defined to apply mutation with Gaussian, Cauchy or uniform random numbers to change a particular gene in the parental individual. The genes in the chromosome are modified as follows:

$$x_i^{(j)'} = \begin{cases} x_i^{(j)} \left(1.0 + N_W(l, \bar{n}) \cdot N_F \cdot N_{Rand}^{D_G}\right), & \text{with } P_G \\ x_i^{(j)} \left(1.0 + N_W(l, \bar{n}) \cdot N_F \cdot N_{Rand}^{D_C}\right), & \text{with } P_C \\ x_i^{(j)} \left(1.0 + N_W(l, \bar{n}) \cdot N_F \cdot N_{Rand}^{D_U}\right), & \text{with } P_U \end{cases}.$$

Evidently that $P_G + P_C + P_U = 1$. The probabilities P_G, P_C and P_U were set to 0.4, 0.4 and 0.2, respectively. The lowest probability was given to the mutation based on the uniform distribution, as the goal of the experiments was to explore the impact of small and large jumps provided by Gaussian and Cauchy distributions on the evolution.

5.2.6.4 Results and Discussion

Tables 5.27, 5.28, 5.30 and 5.31 present the results of ANNs evolved with the combined approaches for the breast cancer and the heart disease problems. Tables 5.29 and

5.32 report the comparative results among NWEA-, CNWEA- and MNWEA-evolved ANNs. The results show that the differences in evolved ANN architectures with CNWEA and MNWEA are insignificant compared to NWEA-evolved topologies, though the smallest CNWEA- and MNWEA-evolved ANNs had lower number of connections for the heart disease problem (26 connections) and the average topologies were generally more compact than those evolved by NWEA (see Tables 5.27 and 5.30). However, both CNWEA- and MNWEA-evolved ANNs demonstrated higher classification accuracy compared to NWEA-evolved ANNs. Statistical analysis with the t-test between NWEA-, CNWEA- and MNWEA-evolved ANNs indicated excellent performance of the combined algorithms. The results for the breast cancer diagnosis showed that the differences in the classification accuracy are statistically significant for ANNs evolved with CNWEA and not significant for ANNs evolved with MNWEA (Table 5.28), and for the heart disease problem showed extremely significance for both CNWEA- and MNWEA-evolved ANNs (Table 5.31). Besides, the combined approaches demonstrated higher convergence speed compared to NWEA as measured by the number of iterations before an optimal network is obtained.

As can be seen from the obtained results, CNWEA that mixed different distributions at the individual level performed favourable than MNWEA that combines the search biases at the component level. However, we do not attribute the differences in the performance to the method of biasing distributions, but instead credit the multiple offspring generation from one parent in CNWEA. In contrast to NWEA and MNWEA, CNWEA creates *three* offspring from each parent and marks the best individual out of them as a "child", while the mutation mechanism of NWEA and MNWEA produces only *one* offspring from each parent.

	CNWEA	**MNWEA**
Connections, min	14	14
Connections, max	82	86
Connections, mean	29	36
Connections, SD	14.2	12.7
Hidden nodes, min	0	0
Hidden nodes, max	4	4
Hidden nodes, mean	1.3	1.4
Hidden nodes, SD	0.6	0.7
Generations, min	87	104
Generations, max	177	185
Generations, mean	118	123.7
Generations, SD	38.1	31.9

TABLE 5.27: Mixing search biases in NWEA: ANN architectures for breast cancer diagnosis.

	CNWEA				MNWEA			
	min	max	mean	SD	min	max	mean	SD
Training error	1.183	2.659	2.329	0.238	1.377	3.247	2.798	0.523
Training error rate	0.00744	0.02451	0.02246	0.00422	0.00954	0.03134	0.03422	0.00741
Validation error	0.037	0.637	0.349	0.122	0.034	0.924	0.503	0.135
Validation error rate	0.00000	0.00902	0.00562	0.00141	0.00000	0.01091	0.00536	0.00152
Testing error	0.054	2.899	1.217	0.455	0.117	3.487	1.406	0.812
Testing error rate	0.00000	0.02687	0.00867	0.00465	0.00000	0.03455	0.01457	0.00748

TABLE 5.28: Mixing search biases in NWEA: classification accuracy for breast cancer diagnosis.

	NWEA	CNWEA	MNWEA
Connections, min	14	14	14
Connections, mean	36	29	36
Hidden nodes, min	0	0	0
Hidden nodes, mean	1.3	1.3	1.4
Testing error, min	0.178	0.054	0.117
Testing error, mean	1.413	1.217	1.406
Testing error rate, min	0.00000	0.00000	0.00000
Testing error rate, mean	0.01384	0.00867	0.01457

TABLE 5.29: Comparative results of NWEA-, CNWEA- and MNWEA-evolved ANNs for breast cancer diagnosis.

	CNWEA	MNWEA
Connections, min	26	26
Connections, max	192	200
Connections, mean	78.3	82,6
Connections, SD	35.2	38.4
Hidden nodes, min	1	1
Hidden nodes, max	8	8
Hidden nodes, mean	3.7	4.1
Hidden nodes, SD	1.7	2.1
Generations, min	106	113
Generations, max	194	188
Generations, mean	157.9	169.6
Generations, SD	41.7	37.5

TABLE 5.30: Mixing search biases in NWEA: ANN architectures for heart disease diagnosis.

5.3 Parallelizing NWEA: Investigating Generalization in PEANNs

As pointed in Section 2.3.5, parallel EAs (PEAs) have found increasing consideration in evolution of ANNs due to advantages derived from both parallelization and interconnection. Several works on PEANNs indicated very promising results in solving optimization problems [28, 29, 55, 56, 120]. They often outperform EANNs in terms of generalization and the size of the evolved architectures.

	CNWEA				MNWEA			
	min	max	mean	SD	min	max	mean	SD
Training error	5.763	12.005	8.963	0.443	5.893	12.137	7.774	0.561
Training error rate	0.04831	0.12069	0.09446	0.01127	0.05271	0.15040	0.10645	0.01042
Validation error	9.271	12.158	9.677	0.237	9.814	12.816	10.240	0.311
Validation error rate	0.08113	0.13812	0.10543	0.01053	0.09276	0.13762	0.12418	0.01433
Testing error	7.009	12.932	10.633	0.674	7.112	13.004	10.458	0.694
Testing error rate	0.09385	0.14889	0.11676	0.01474	0.11243	0.17002	0.12276	0.01651

TABLE 5.31: Mixing search biases in NWEA: classification accuracy for heart disease diagnosis.

	NWEA	CNWEA	MNWEA
Connections, min	28	26	26
Connections, mean	88.4	78.3	82,6
Hidden nodes, min	1	1	1
Hidden nodes, mean	4.0	3.7	4.1
Testing error, min	10.126	7.009	7.112
Testing error, mean	12.266	10.633	10.458
Testing error rate, min	0.13195	0.09385	0.11243
Testing error rate, mean	0.15165	0.11676	0.12276

TABLE 5.32: Comparative results of NWEA-, CNWEA- and MNWEA-evolved ANNs for heart disease diagnosis.

This section addresses the question of examining generalization in PEANNs evolved by the parallelized NWEA (PNWEA). Two efficient parallelization schemes that perform genetic information exchange between parallel populations, have been exploited to parallelize NWEA. The PNWEA approaches were applied to evolve ANNs for the Mackey-Glass chaotic time series.

5.3.1 Parallelization strategies

The simplest parallelization approach, i.e., *independent* parallelization strategy, described in Section 2.1.6 was found not effective, as it is similar to multiple runs of a serial EA and does not carry out any interconnection between simultaneously evolved populations. The benefit of interconnection lies in the exchange of genetic material between populations, which enables to maintain diversity of individuals. As a result, the populations become more resistant to local minima trapping, since they contain genetic material from different portions of search space.

In order to add interconnection in the evolution process, an alternative parallelization scheme, called *migration* [147], has been proposed. The migration approach augments the independent approach with the periodic individuals exchange between different populations. A PEA with the migration strategy starts the evolution similarly to a PEA with the independent strategy. After each evolution stage or a predefined number of

evolution stages each concurrent process sends some number of the best offspring individuals to a shared storage and gets the same number of individuals from other parallel populations.

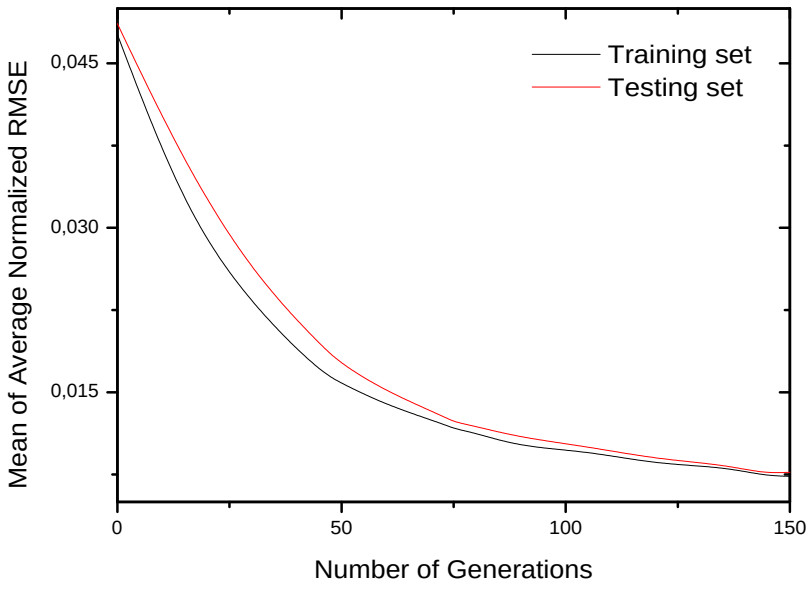
The parallelization approach, proposed in this thesis, is the following improvement of the migration scheme: after migration, some individuals with the low fitness in each parallel population are replaced by randomly generated individuals called *strangers* [25, 28]. The main goal of inserting strangers is to extend the search space. It is likely that after a number of generations the best individuals in the parallel populations might have the same fitness; thus, the exchange of the best individuals between the concurrent processes will not introduce new genotype. By involving strangers in the population we maintain diversity of individuals and allow the process considering the earlier unexplored areas of the search space. As a result, the probability of an algorithm to find the global minimum increases. The experiments made in [25] showed, that the utilization of 4-6% of strangers is optimal. The low rate of strangers enables the algorithm to explore different portion of the search space but does not change the average fitness of a population considerably, while the high rate reduces the average fitness of a population and slows down the evolution process.

5.3.2 Experimental Setup

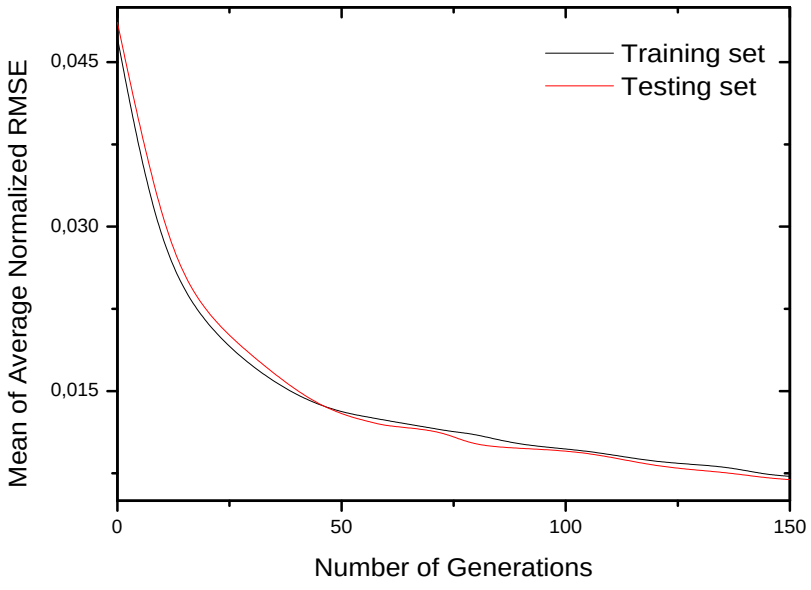
The PEANNs have been evolved for the Mackey-Glass chaotic time series in order to predict the output for small $\Delta t = 6$ and large $\Delta t = 90$ time spans. Following initial conditions have been used in experiments: the population size 120, which was divided into 4 parallel processes (30 individuals for each parallel approach), the migration rate 10%, the strangers rate 4%, the maximum number of generations 150. The migration and the strangers' insertion were carried out every 10th generation. Both PNWEAs started their evolution with the same initial populations; each algorithm with 30 randomly generated populations was run 30 times. Parallel processes were implemented on one machine (Intel Core 2 Duo Processor T7100, 1.80 GHz, 2.00 GB/Go DDR2 SDRAM) using multiple threads. The other parameters as well as error function were the same as determined in Section 5.2.3.1.

5.3.3 Results and Discussions

Tables 5.33 and 5.34 report the results of PEANNs over 30 runs for the Mackey-Glass chaotic time series. Figure 5.5 shows the evolution of PEANNs' performance. The results in Table 5.34 indicated better generalization of PNWEA-evolved ANNs in comparison to that of NWEA-designed ANNs. For a small time span ($\Delta t = 6$), the minimal errors



(a) PNWEA with migration strategy



(b) PNWEA with migration-strangers strategy

FIGURE 5.5: PNWEAs' convergence: evolution of PEANNs' performance.

	Migration	**Mig.-Strangers**
Connections, min	64	60
Connections, max	126	118
Connections, mean	90.0	84.9
Connections, SD	21.1	19.7
Hidden nodes, min	8	8
Hidden nodes, max	13	12
Hidden nodes, mean	10.1	9.6
Hidden nodes, SD	1.72	1.68
Generations, min	67	46
Generations, max	98	90
Generations, mean	73.4	68.1
Generations, SD	17.6	18.4

TABLE 5.33: PEANN architectures for the Mackey-Glass time series problem.

	Migration				**Mig.-Strangers**			
	min	max	mean	SD	min	max	mean	SD
Training error	0.0067	0.0139	0.0078	0.0017	0.0064	0.0126	0.0068	0.0016
Testing error, $\Delta t = 6$	0.0068	0.0163	0.0103	0.0015	0.0065	0.0145	0.0087	0.0015
Testing error, $\Delta t = 90$	0.0121	0.0615	0.0362	0.0057	0.0118	0.0517	0.0227	0.0051

TABLE 5.34: Prediction accuracy of PEANNs for the Mackey-Glass time series problem.

of the PNWEA-evolved ANNs with migration and migration-strangers strategies were 0.0068 and 0.0065, respectively. These results are almost twice lower than the best result of EANNs evolved with the serial NWEA (0.0122, see Table 5.3). The average prediction accuracy of PEANNs also compare favorable to that of EANNs; the average testing errors of the PNWEA-evolved ANNs with migration and migration-strangers strategies were 0.0103 and 0.0087, while the corresponding result of NWEA-evolved ANNs was 0.0123. Note, that there are no huge differences between the lowest and the average results, which means that parallelized algorithm often evolves good generalized ANNs.

Similar results can be observed for the multiple step prediction. The minimal errors indicated for a large time span ($\Delta t = 90$) were 0.0121 for PEANNs evolved with migration and 0.0118 for PEANNs produced with migration-strangers, while the average results were 0.0362 and 0.0227, respectively. The best and the average results of the NWEA-evolved ANN are 0.0395 and 0.0538, respectively.

Table 5.33 demonstrates that PNWEAs evolved more compact architectures. The smallest networks produced by PNWEA with migration and PNWEA with migration-strangers schemes had 64 for and 60 connections, respectively, while the smallest EANN had 64 connections. The average number of connections in ANNs evolved by PNWEA with the migration strategy was 90.0 and in ANNs produced by PNWEA with the

migration-strangers approach was 84.9 (NWEA-evolved ANNs had in average 98.21 connections).

The prediction results of PEANNs evolved with the migration-strangers strategy were better than those of PEANNs developed with the migration scheme (the t-test analysis indicated significant distinctions between them). Specifically, starting the evolution process with the same initial populations, PNWEA with migration-strangers strategy produced more compact networks with smaller testing errors. The results clearly demonstrated the advantage of the strangers' insertion. In contrast to the migration approach, which evolved individuals in the environment restricted with the genotype of parallel populations and strongly relied on quality of the improvement mechanism, the migration-strangers strategy periodically inserted new solutions in the populations. This extended the search space and allowed the algorithm to consider early unexplored areas, i.e., led to the global searching and improved the quality of the evolved solutions.

Chapter 6

Conclusions

"Each problem that I solved became a rule, which served afterwards to solve other problems."

Rene Descartes

This chapter concludes the dissertation by summarizing its contribution and discussing potential directions for future research.

6.1 Summary

Learning is a central issue in the theory of ANNs. The choice of a learning algorithm is crucial, as it affects the functionality and performance of the network. This thesis proposed a new mutation-based evolutionary learning algorithm, called the *network-weight-based evolutionary algorithm* (NWEA) for evolving ANNs. The main contribution of this dissertation is the development of the algorithm which extends computational evolution by involving other mechanisms of nature. The key idea behind NWEA is to perform behavioral adaptation alongside with structural adaptation, that means, to provide interaction between a population and the environment, collect knowledge of the environment and use it in the optimization process. The modification strategy of NWEA consists of two individual-level adaptive components, which represent genotype and phenotype information, derived from the genetic characteristics of an individual and its local environment. Genotype information is introduced by an individual's error, that determines the worth of an individual with respect to a given problem. Phenotype information is included in the novel component, called the *network weight* (NW), which describes an ANN's internal structure. NWEA does not perform any probabilistic selection for reproduction; all individuals in the population undergo mutation in order to

create offspring. The new generation is formed according to $(\mu+\lambda)$-ES, which selects the best individuals from both parental and offspring populations for the next generation.

The critical challenge of our research was to determine the relationship between NW values and ANN architectures. In order to obtain the function that describes the dependency of NW values on ANN topologies, we provided extensive empirical and analytical studies, where NWEA-evolved ANNs were used as meta-models for fitness approximation. The results showed that the NW component is related to the ANN's internal structure and depends on the total number of hidden layers and the average number of neurons in hidden layers. This dependency is defined by the Fermi-Dirac-like function.

The NWEA-evolved ANNs were tested on a number of benchmark problems of different complexity. The experiments were divided into three parts. In the first part, we investigated the internal features of NWEA, such as convergence speed, the average rate of successful improvements per generation and ability to find solutions of high accuracy. For this purpose, NWEA was applied to evolve connection weights in the environment of 25 fixed ANN architectures for the XOR problem. The results of these experiments were compared with those, obtained by CEP, FEP, IFEP and MEP algorithms. The results showed that NWEA outperformed classical approaches in terms of the average rate of successfully improved individuals per generation, which led to faster convergence to optima. Additionally, NWEA was able to find solutions of higher accuracy. Several tests were also provided to examine the efficiency of the genotypic component represented by an individual's error. These experiments compared the performance of three NWEA variations, which utilized the particular error of each individual, the average error of the population, and the error of the best individual in the population in the evolution process.

The second part of experiments was devoted to examining generalization in ANNs evolved for real-world classification and prediction problems. In order to reduce noise in fitness evaluation and to study the ability of NWEA to evolve compact and good generalized ANNs, the evolution of both connection weights and architectures has been performed. Four benchmark classification problems from the UCI repository (breast cancer, heart disease, diabetes and thyroid problems) and the Mackey-Glass chaotic time series prediction problem were used to evaluate the performance of NWEA-evolved ANNs. The results indicated the NWEA algorithm as a very promising methodology for ANNs design and training; for all five tasks NWEA evolved good generalized ANNs, whose classification/prediction accuracy outperformed many existing approaches. Similar results were obtained regarding the complexity of ANN topology; in contrast to the existing algorithms, NWEA evolved more compact ANN architectures.

Further, the evolution process was investigated under different internal parameters of NWEA and ANNs. In order to study the impact of the activation function type (the internal ANN parameter) on the generalization in ANNs, we compared NWEA-evolved ANNs that utilized logistic, hyperbolic tangent and piecewise linear activation functions while proceeding signals. The results detected no significant differences in performances of the evolved networks. Such a behavior is explained by the fact that evolutionary learning methodologies are less dependent the type of the activation function than non-evolutionary approaches.

Experiments were also provided for the purpose of examining the role of the distribution function's type (the internal NWEA parameter) on the quality of the obtained networks. We studied two modifications of NWEA that combined Gaussian, Cauchy and uniform distributions at the individual- and component-levels. The evolution process was observed under different step sizes determined by mixed distributions. Both modifications outperformed NWEA in terms of the algorithm convergence speed and generalization ability of the evolved networks; the statistical analysis of the obtained results indicated significantly higher accuracy of the combined NWEA approaches (CNWEA and MNWEA) over the standard NWEA.

In the final part of experiments, generalization of PEANNs were studied. Two parallelization schemes, i.e., widely known migration scheme and our *migration-strangers* scheme, were used to evolve PEANNs. As anticipated, parallel NWEA evolved better ANNs in terms of generalization than the serial NWEA. In addition, the results indicated that the migration-strangers scheme is more effective in comparison to the migration approach, as it in addition to migration also uses replacement strategy to maintain population diversity and thus, explores different portions of the search space.

6.2 Future Work

This thesis provided a starting point for investigating the impact of the phenotype information inclusion in the evolution process. Although the research in this dissertation analyzed NWEA algorithm from different perspectives and examined NWEA-evolved ANNs by considering different parameters, there are some potential directions for future work:

- Investigate the combination of NWEA with local search/replacement procedures in order to maintain the global search while solving problems with many local optima. In particular, the application of the replacement mechanisms might be

advantageous in retaining the population's diversity, when the majority of individuals is located around the local optimum.

- A lot more research can be done to investigate the application of NWEA to other types of ANNs, e.g., recurrent neural networks (RNNs).

Bibliography

[1] E. Alba, J. F. A. Montes, and J. M. Troya. Full automatic ANN design: A genetic approach. In *Proceedings of the International Workshop on Artificial Neural Networks: New Trends in Neural Computation*, pages 399–404, London, UK, 1993. Springer-Verlag.

[2] P. J. Angeline. Adaptive and self-adaptive evolutionary computations. In *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[3] P. J. Angeline. The effects of noise on self-adaptive evolutionary optimization. In *Evolutionary Programming*, pages 433–439, 1996.

[4] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65, 1994.

[5] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[6] T. Bäck. An overview of parameter control methods by self-adaption in evolutionary algorithms. *Fundamenta Informaticae*, 35:51–66, August 1998.

[7] T. Bäck, F. Hoffmeister, and H. P. Schwefel. A survey of evolution strategies. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1991.

[8] T. Bäck and H.-P.l Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1:1–23, March 1993.

[9] R. K. Belew, J. Mcinerney, and N. N. Schraudolph. Evolving networks: using the genetic algorithm with connectionist learning. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Proceedings of 2nd Conference on Artificial Life*, pages 511–547. Addison-Wesley, 1991.

[10] K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods Software*, 1:23–34, 1992.

[11] H.-G. Beyer. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, 3:311–347, September 1995.

[12] H.-G. Beyer and K. Deb. On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 5:250–270, 2001.

[13] H.-G. Beyer and S. Meyer-Nieberg. Self-adaptation of evolution strategies under noisy fitness evaluations. *Genetic Programming and Evolvable Machines*, 7(4):295–328, 2006.

[14] H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

[15] J. Branke. Evolutionary algorithms for neural network design and training. In *Proceedings of the 1st Nordic Workshop on Genetic Algorithms and its Applications*, pages 145–163, Vaasa, Finnland, 1995.

[16] B. Carse and J. Oreland. Evolution and learning in neural networks: dynamic correlation, relearning and thresholding. *Adaptive Behavior*, 8:297–311, January 2001.

[17] M. Chambers and C. A., Mount-Campbell. Process optimization via neural network metamodeling. *International Journal of Production Economics*, 79(2):93–100, 2002.

[18] K. Chellapilla and D. B. Fogel. Fitness distributions in evolutionary computation: motivation and examples in the continuous domain. *BioSystems*, 54(1-2):15–29, 1999.

[19] Y. Chen, B. Yang, J. Dong, and A. Abraham. Time-series forecasting using flexible neural tree model. *Information Sciences*, 174:219–235, August 2005.

[20] A. Chipperfield and P. Flemming. Parallel genetic algorithms. *Parallel and Distributed Computing Handbook*, pages 1118–1143, 1996.

[21] K. B. Cho and B.-H. Wang. Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction. *Fuzzy Sets and Systems*, 83:325–339, November 1996.

[22] R. S. Crowder. Predicting the Mackey-Glass time series with cascade-correlation learning. In *Proceedings of the 1990 Connectionist Models Summer School*, pages 117–123, 1990.

[23] C. Darwin. *On the origin of species by means of natural selection: Or, the preservation of favored races in the struggle for life.* D. Appleton, New York, 5th ed., with additions and corrections edition, 1872.

[24] T. E. Davis and J. C. Príncipe. A simulated annealing like convergence theory for the simple genetic algorithm. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 174–181. Morgan Kaufmann, 1991.

[25] K. Davoian. Search space extension and PGAs: a comparative study of parallelization schemes to genetic algorithms using the TSP. In *Proceedings of the 24th IASTED International Conference on Artificial Intelligence and Applications*, pages 26–30. ACTA Press, 2006.

[26] K. Davoian and W.-M. Lippe. Adapting mutation step size to an ANN topology. In *Proceedings of the 2007 International Conference on Artificial Intelligence*, volume 1, pages 233–239. CSREA Press, 2007.

[27] K. Davoian and W.-M. Lippe. Including phenotype information in mutation to evolve artificial neural networks. In *Proceedings of the 2007 International Joint Conference on Neural Networks*, pages 2782–2787. IEEE, 2007.

[28] K. Davoian and W.-M. Lippe. Investigating generalization in parallel evolutionary artificial neural networks. In *Proceedings of the 25th IASTED International Conference on Artificial Intelligence and Applications*, pages 90–95. ACTA Press, 2007.

[29] K. Davoian and W.-M. Lippe. Time series prediction with parallel evolutionary artificial neural networks. In *Proceedings of the 2007 International Conference on Data Mining*, pages 10–15. CSREA Press, 2007.

[30] K. Davoian and W.-M. Lippe. Exploring the role of activation function type in evolutionary artificial neural networks. In *Proceedings of The 2008 International Conference on Data Mining*, volume 2, pages 443–449. CSREA Press, 2008.

[31] K. Davoian and W.-M. Lippe. Greedy learning: Using advantages of distribution functions to improve generalization of ANNs. In *Proceedings of the 2009 International Conference on Data Mining*, pages 361–367. CSREA Press, 2009.

[32] K. Davoian and W.-M. Lippe. Mixing different search biases in evolutionary learning algorithms. In *Proceedings of the 19th International Conference on Artificial Neural Networks*, volume 1, pages 111–120, Berlin, Heidelberg, 2009. Springer-Verlag.

[33] K. Davoian and W.-M. Lippe. Solving classification problems with the NWEA-evolved ANNs. In *Proceedings of the 2010 International Conference on Data Mining*, pages 249–255. CSREA Press, 2010.

[34] K. Davoian, A. Reichel, and W.-M. Lippe. Comparison and analysis of mutation-based evolutionary algorithms for ANN parameters optimization. In *Proceedings of the 2006 International Conference on Data Mining*, pages 51–56. CSREA Press, 2006.

[35] K. Deb and H.-G. Beyer. Self-adaptation in real-parameter genetic algorithms with simulated binary crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 172–179. Morgan Kaufmann, 1999.

[36] K. Deb and H.-G. Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary Computation*, 9:197–221, June 2001.

[37] K. Deb, K. Sindhya, and T. Okabe. Self-adaptive simulated binary crossover for real-parameter optimization. In *Proceedings of the 2007 Genetic and Evolutionary Computation Conference*, pages 1187–1194, New York, NY, USA, 2007. ACM.

[38] A. E. Eiben, E. H. L. Aarts, and K. M. van Hee. Global convergence of genetic algorithms: A markov chain analysis. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, volume 1, pages 4–12, London, UK, 1991. Springer-Verlag.

[39] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

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

[41] J. D. Farmer and John Sidorowich. Predicting chaotic time series. *Physical Review Letters*, 59:845–848, 1987.

[42] W. Finnoff, F. Hergert, and H.-G. Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6(6):771–783, 1993.

[43] G. S. Fishman and V. G. Kulkarni. Improving Monte Carlo efficiency by increasing variance. *Management Science*, 38:1432–1444, October 1992.

[44] D. B. Fogel. *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn Press, 1991.

[45] D. B. Fogel. *Evolving Artificial Intelligence*. PhD thesis, University of California, San Diego, 1992.

[46] D. B. Fogel. Applying evolutionary programming to selected control problems. *Computers and Mathematics with Applications*, 27(11):89–104, 1994.

[47] D. B. Fogel. Phenotypes, genotypes, and operators in evolutionary computation. In *Proceedings of the 2nd IEEE International Conference on Evolutionary Computation*, pages 193–198. IEEE Press, 1995.

[48] D. B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. IEEE Press, John Wiley & Sons, Inc., 2nd edition, 1999.

[49] D. B. Fogel and C. J. Robinson, editors. *Computational Intelligence: The Experts Speak*. IEEE Press, John Wiley & Sons, Inc., New York, NY, USA, 2003.

[50] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[51] L. J. Fogel. *Intelligence through Simulated Evolution: Forty Years of Evolutionary Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[52] L. J. Fogel, P. J. Angeline, and D. B. Fogel. An evolutionary programming approach to self-adaptation on finite state machines. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 355–365. MIT Press, 1995.

[53] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, Inc., New York, NY, USA, 1966.

[54] M. Frean. The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Comput.*, 2:198–209, April 1990.

[55] A. A. Ghorbani and L. Bayat. Accelerated backpropagation learning: Extended dynamic parallel tangent optimization algorithm. In *Proceedings of the 13th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, pages 293–304, London, UK, 2000. Springer-Verlag.

[56] A. A. Ghorbani, A. R. Nezami, and V. C. Bhavsar. An incremental parallel tangent learning algorithm for artificial neural networks. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, pages 301–304. IEEE Press, 1997.

[57] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[58] D. E. Goldberg. Zen and the art of genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 80–85, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[59] D. E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[60] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[61] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 160–168, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

[62] W. Greiner, L. Neise, and H. Stöcker. *Thermodynamics and Statistical Mechanics*. Classical theoretical physics. Springer-Verlag, 1995.

[63] F. Gruau. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In J. D. Schaffer and D. Whitley, editors, *Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 55–74, Los Alamos, CA, USA, 1992. IEEE Computer Society Press.

[64] P. J. B. Hancock and L. S. Smith. Gannet: Genetic design of a neural net for face recognition. In H.-P. Schwefel and R. Männer, editors, *Proceedings of the International Conference on Parallel Problem Solving from nature (PPSN'90)*, volume 496 of *Lecture Notes in Computer Science*, pages 292–296. Springer, 1990.

[65] J. V. Hansen and R. D. Meservy. Learning experiments with genetic optimization of a generalized regression neural network. *Decision Support Systems*, 18:317–325, November 1996.

[66] N. Hansen. Invariance, self-adaptation and correlated mutations and evolution strategies. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 355–364, London, UK, 2000. Springer-Verlag.

[67] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195, June 2001.

[68] S. A. Harp, T. Samad, and A. Guha. Towards the genetic synthesis of neural network. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms and Their Applications*, pages 360–369, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[69] S. A. Harp, T. Samad, and A. Guha. Designing application-specific neural networks using the genetic algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 447–454, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[70] W. E. Hart and R. K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 190–195, San Mateo, CA, USA, 1991. Morgan Kaufmann.

[71] D. O. Hebb. *The Organization of Behavior*. John Wiley & Sons Inc., New York, NY, USA, 1949.

[72] R. Hinterding, Z. Michalewicz, and T. C. Peachey. Self-adaptive genetic algorithm for numeric functions. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 420–429, London, UK, 1996. Springer-Verlag.

[73] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, September 1989.

[74] Y. Hirose, K. Yamashita, and S. Hijiya. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 4:61–66, January 1991.

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

[76] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[77] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science*, 79:2554–2558, 1982.

[78] J. S. R. Jang, C. T. Sun, and E. Mizutani. *Neuro-Fuzzy and Soft Computing: a Computational Approach to Learning and Machine Intelligence*. Prentice Hall, Upper Saddle River, NJ, USA, 1997.

[79] R. Keesing and D. G. Stork. Evolution and learning in neural networks: the number and distribution of learning trials affect the rate of evolution. In *Advances in Neural Information Processing Systems 3*, pages 804–810, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[80] A. Khosravi, S. Nahavandi, and D. Creighton. Developing optimal neural network metamodels based on prediction intervals. In *Proceedings of the 2009 International Joint Conference on Neural Networks*, pages 95–101, Piscataway, NJ, USA, 2009. IEEE Press.

[81] W. Kinnebrock. Accelerating the standard backpropagation method using a genetic approach. *Neurocomputing*, 6(5):583–588, 1994.

[82] H. Kitano. Designing Neural Networks Using Genetic Algorithms with Graph Generation System. *Complex Systems*, 4:461–476, 1990.

[83] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, New York, NY, USA, 3rd edition, 1989.

[84] J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection.* MIT Press, 1st edition, 1992.

[85] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA, 1994.

[86] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, May 1999.

[87] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[88] K. K. Lai, L. Yu, S. Wang, and C. Zhou. Neural-network-based metamodeling for financial time series forecasting. In *Proceedings of the 2006 Joint Conference on Information Sciences*, pages 172–175, Paris, France, 2006. Atlantis Press.

[89] A. Lapedes and R. Farber. Nonlinear signal processing using neural networks: Prediction and system modeling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM, 1987.

[90] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization.* John Wiley & Sons, New York, NY, USA, 1985.

[91] A. Likartsis, I. Vlachavas, and L. H. Tsoukalas. A new hybrid neural-genetic methodology for improving learning. In *Proceedings of the 9th International Conference on Tools with Artificial Intelligence*, pages 32–36, Washington, DC, USA, 1997. IEEE Computer Society.

[92] W.-M. Lippe. *Soft-Computing: mit Neuronalen Netzen, Fuzzy-Logic und Evolutionären Algorithmen.* Springer-Verlag, Berlin, Heidelberg, New York, 2006.

[93] Y. Liu and X. Yao. Evolutionary design of artificial neural networks with different nodes. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 670–675. IEEE Press, 1996.

[94] Y. Liu and X. Yao. A population-based learning algorithm which learns both architectures and weights of neural networks. *Chinese Journal of Advanced Software Research*, 3:54–65, 1996.

[95] Y. Liu and X. Yao. Time series prediction by using negatively correlated neural networks. In *Selected papers from the 2nd Asia-Pacific Conference on Simulated Evolution and Learning*, SEAL'98, pages 333–340, London, UK, 1999. Springer-Verlag.

[96] M. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.

[97] V. Maniezzo. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5(1):39–53, 2002.

[98] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. "Neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks/IEEE Neural Networks Council*, 4(4):558–569, 1993.

[99] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943.

[100] J. R. McDonnell and D. Waagen. Evolving recurrent perceptrons for time-series modeling. *IEEE Transactions on Neural Networks*, 5:24–38, 1994.

[101] S. Meyer-Nieberg and H.-G. Beyer. Self-adaptation in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithm*, pages 47–75, Berlin, Germany, 2007. Springer-Verlag.

[102] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[103] R. Miikkulainen. Evolving neural networks. In *Proceedings of the 2007 Genetic and Evolutionary Computation Conference*, pages 3415–3434, New York, NY, USA, 2007. ACM.

[104] B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4:113–131, June 1996.

[105] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the 3rd International conference on Genetic Algorithms*, pages 379–384, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[106] E. Mjolsness, D. H. Sharp, and B. K. Alpert. Scaling, machine learning, and genetic neural nets. *Advances in Applied Mathematics*, 10:137–163, June 1989.

[107] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, volume 1, pages 762–767, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[108] H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 416–421, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[109] S. Nolfi and D. Parisi. Evolution of artificial neural networks. *Handbook of brain theory and neural networks*, pages 418–421, 2002.

[110] S. V. Odri, D. P. Petrovacki, and G. A. Krstonosic. Evolutional development of a multilevel neural network. *Neural Networks*, 6(4):583–595, 1993.

[111] P. S. Oliveto, J. He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, pages 281–293, 2007.

[112] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation*, 2:369–380, December 1994.

[113] C. B. Pettey, M. R. Leuze, and J. J. Grefenstette. A parallel genetic algorithm. In *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, pages 155–161, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[114] V. W. Porto, D. B. Fogel, and L. J. Fogel. Alternative neural network training methods. *IEEE Expert: Intelligent Systems and Their Applications*, 10:16–22, June 1995.

[115] L. Prechelt. Proben1 - a set of neural network benchmark problems and benchmarking rules. Technical report, Fakultät Für Informatik, Universität Karlsruhe, 1994.

[116] J. C. F. Pujol and R. Poli. Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence*, 8:73–84, January 1998.

[117] I. Rechenberg. Cybernetic solution path of an experimental problem. Technical report, Royal Air Force Establishment, 1965.

[118] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1973.

[119] I. Rechenberg. *Evolutionsstrategie'94.* Frommann-Holzboog, Stuttgart, 1994.

[120] G. Riessen, G. J. Williams, and X. Yao. PEPNet: Parallel evolutionary programming for constructing artificial neural networks. In *Proceedings of the 6th International Conference on Evolutionary Programming*, pages 35–46, London, UK, 1997. Springer-Verlag.

[121] I. Rojas, H. Pomares, J. L. Bernier, J. Ortega, B. Pino, F. J. Pelayo, and A. Prieto. Time series analysis using normalized PG-RBF network with regression weights. *Neurocomputing*, 42:267–285, 2002.

[122] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[123] F. Rosenblatt. *Principles of Neurodynamics.* Spartan Book, New York, NY, USA, 1962.

[124] A. Roy, L. S. Kim, and S. Mukhopadhyay. A polynomial time algorithm for the construction and training of a class of multilayer perceptrons. *Neural Networks*, 6:535–545, 1993.

[125] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5:96–101, 1994.

[126] G. Rudolph. Convergence of non-elitist strategies. In *Proceedings of the 1st IEEE International Conference on Evolutionary Computation*, pages 63–66, Piscataway, NJ, USA, 1994. IEEE Press.

[127] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Neurocomputing: foundations of research*, pages 673–695, 1988.

[128] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman. Using genetic search to exploit the emergent behavior of neural networks. *Journal of Physics D*, 42:244–248, June 1990.

[129] J. D. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, pages 36–40, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[130] W. Schiffmann, M. Joost, and R. Werner. Synthesis and performance analysis of multilayer neural network architectures. Technical report, Institut für Physik, Koblenz, Germany, 1992.

[131] H.-P. Schwefel. Kybernetische Evolution als Strategie der exprimentellen Forschung in der Strömungstechnik. Master's thesis, Technical University of Berlin, 1965.

[132] H.-P Schwefel. Adaptive Mechanismen in der biologischen Evolution und ihr Einflußauf die Evolutionsgeschwindigkeit. Technical report, Technical University of Berlin, Abschlubericht zum DFG-Vorhaben Re 215/2, 1974.

[133] H.-P Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary systems research*. Birkhäuser, Basel/Stuttgart, 1977.

[134] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[135] H.-P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[136] R. Setiono and L. C. K. Hui. Use of a quasi-newton method in a feedforward neural network construction algorithm. *IEEE Transactions on Neural Networks*, 6:273–277, 1995.

[137] R. S. Sexton, R. E. Dorsey, and J. D. Johnson. Toward global optimization of neural networks: a comparison of the genetic algorithm and backpropagation. *Decision Support Systems*, 22:171–185, February 1998.

[138] J. Sietsma and R. J. F. Dow. Creating artificial neural networks that generalize. *Neural Netw.*, 4:67–79, January 1991.

[139] W. M. Spears. Adapting crossover in evolutionary algorithms. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 367–384. MIT Press, 1995.

[140] W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In *Proceedings of the European Conference on Machine Learning*, pages 442–459, London, UK, 1993. Springer-Verlag.

[141] T. Starkweather, L. D. Whitley, and K. E. Mathias. Optimization using distributed genetic algorithms. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 176–185, London, UK, 1991. Springer-Verlag.

[142] J. Suzuki. A markov chain analysis on a genetic algorithm. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 146–154, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[143] A. P. Topchy and O. A. Lebedko. Neural network training by means of cooperative evolutionary search. *Nuclear Instruments and Methods in Physics Research Section A-accelerators Spectrometers Detectors and Associated Equipment*, 389:240–241, 1997.

[144] A. Törn and A. Žilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, Berlin, 1989.

[145] H.-M. Voigt, J. Born, and I. Santibez-Koref. Evolutionary structuring of artificial neural networks. Technical report, Bionics and Evolution Techniques Lab., Technical University Berlin, 1993.

[146] E. Vonk, L.C. Jain, and R. Johnson. Using genetic algorithms with grammar encoding to generate neural networks. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1928–1931. IEEE, 1995.

[147] L. Wang, A. A. Maciejewski, H. J. Siegel, and V. P. Roychowdhury. A comparative study of five parallel genetic algorithms using the traveling salesman problem. In *Proceedings of the 12th International Parallel Processing Symposium on International Parallel Processing Symposium*, pages 345–349, Washington, DC, USA, 1998. IEEE Computer Society.

[148] C. J. C. H. Watkins and P. Dayan. Technical note Q-learning. *Machine Learning*, 8:279–292, 1992.

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

[150] L. D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, 14:347–361, 1990.

[151] L. Willmes, T. Bäck, Y. Jin, and B. Sendhoff. Comparing neural networks and kriging for fitness approximation in evolutionary optimization. In *IEEE Congress on Evolutionary Computation*, pages 663–670, 2003.

[152] S. W. Wilson. Perceptron redux: emergence of structure. *Journal of Physics D*, 42:249–256, June 1990.

[153] X. Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 4:539–567, 1993.

[154] X. Yao. An overview of evolutionary computation. *Chinese Journal of Advanced Software Research*, 3:12–29, 1996.

[155] X. Yao. Evolving artificial neural networks. In *Proceedings of the IEEE*, volume 87, pages 1423–1447. IEEE Press, 1999.

[156] X. Yao. Knowledge extracted from trained neural networks. In *Proceedings of the IEEE*, pages 1423–1447. IEEE Press, 1999.

[157] X. Yao, G. Lin, and Y. Liu. An analysis of evolutionary algorithms based on neighborhood and step sizes. In *Proceedings of the 6th International Conference on Evolutionary Programming VI*, EP '97, pages 297–307, London, UK, 1997. Springer-Verlag.

[158] X. Yao and Y. Liu. Evolving artificial neural networks for medical applications. In *Proceedings of 1995 Australia-Korea Joint Workshop on Evolutionary Computation*, pages 1–16, 1995.

[159] X. Yao and Y. Liu. Ensemble structure of evolutionary artificial neural networks. In *Proceedings of the International Conference on Evolutionary Computation*, pages 659–664, Nagoya, Japan, 1996.

[160] X. Yao and Y. Liu. Evolving artificial neural networks through evolutionary programming. In *Proceedings of the 5th Annual Conference on Evolutionary Programming*, pages 257–266. MIT Press, 1996.

[161] X. Yao and Y. Liu. Fast evolutionary programming. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 451–460. MIT Press, 1996.

[162] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8:694–713, 1996.

[163] X. Yao and Y. Liu. Fast evolution strategies. In *Proceedings of the 6th International Conference on Evolutionary Programming VI*, pages 151–162. Springer-Verlag, 1997.

[164] X. Yao and Y. Liu. Scaling up evolutionary programming algorithms. In *Proceedings of the 7th International Conference on Evolutionary Programming VII*, pages 103–112. Springer-Verlag, 1998.

[165] X. Yao and Y. Liu. Neural networks for breast cancer diagnosis. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1760–1767. IEEE Press, 1999.

[166] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3:82–102, 1996.

[167] L. Yu, S. Wang, and K. K. Lai. A neural-network-based nonlinear metamodeling approach to financial time series forecasting. *Appied Soft Computing*, 9:563–574, 2009.