DTU Compute
Department of Applied Mathematics and Computer Science

Implementation and Evaluation of Recent Neuroevolution Algorithms

Master Thesis

Implementation and Evaluation of Recent Neuroevolution Algorithms

Master Thesis
June, 2023

By
Samy Haffoudhi

Cover photo: Vibeke Hempler, 2012
Published by: DTU, Department of Applied Mathematics and Computer Science, Richard Petersens Plads, Building 324, 2800 Kgs. Lyngby Denmark
www.compute.dtu.dk
ISSN: [0000-0000] (electronic version)
ISBN: [000-00-0000-000-0] (electronic version)

ISSN: [0000-0000] (printed version)
ISBN: [000-00-0000-000-0] (printed version)

Approval

This thesis has been prepared over five months at the Department of Applied Mathematics and Computer Science at the Technical University of Denmark, DTU, in partial fulfilment for the degree Master of Computer Science & Engineering. The project has been supervised by Prof. Carsten Witt and corresponds to 30 ECTS points.

It is assumed that the reader has fundamental knowledge of computer science.

Samy Haffoudhi - s222887

.....

Signature

.....

Date

Abstract

Neuroevolution is a method for optimizing the topology or parameters of neural networks by means of evolutionary algorithms. This technique is more general than white-box approaches, such as supervised learning, and can therefore be applied to a wider range of problems. It has been studied in research for decades and has been successfully applied to problems such as artificial life, evolutionary robotics and continuous domains of reinforcement learning. In this thesis, we present a framework that implements neuroevolution algorithms and that is used to evaluate these algorithms on a selection of benchmark problems. Algorithms and benchmarks were collected from the state of the art in applied and theoretical research in the field of neuroevolution. The framework, implemented in Rust, allows for a visualization of key problem characteristics and the evolution process through a graphical user interface. Selected algorithms and benchmarks are presented in detail. Results collected from the conducted experiments are analyzed, discussed and used to provide a series of guidelines for the choice of algorithms and parameters with respect to problem classes.

TODO Discuss results, guidelines...

Acknowledgements

TODO

Contents

Chapter 1

Introduction

Neuroevolution is a subfield of artificial intelligence which consists in the evolution of ANNs (artificial neural networks). ANNs are traditionally trained using gradient-based methods, such as stochastic gradient descent. Over the years, these methods have been successfully applied to a variety of problems, such as image classification, speech recognition and natural language processing. Such problems allow for supervised learning, where ANNs are trained on a dataset of input-output pairs. However, there is a class of problems for which supervised learning is not applicable, where instead of input-output pairs, only a measure of performance is available. In addition, the performance of ANNs is also heavily impacted by their architectures. The design of ANNs architecture is a complex and time-consuming task, which is topically done by hand, based on experience.

Neuroevolution, on the other hand, as a more general approach, can in particular be applied to this other class of problems, as well as to the design of ANNs architectures. This method is based on evolutionary algorithms, which are inspired by the process of natural selection. These algorithms maintain a population of individuals, which are mutated and recombined to evolve towards optimal solutions. They have shown success for black-box problems and have successfully been applied to a wide range of engineering problems.

The field of neuroevolution has been researched for over 40 years, hence many different algorithms, benchmarks and applications have been proposed. As a matter of fact, neuroevolution encapsulates algorithms with different goals, such as the optimization of the weights of a fixed topology, the evolution of a network topology alongside the use of gradient-based methods for optimizing weights, the evolution of both the topology and weights, as well as the evolution of hyperparameters or reinforcement learning policies. In this thesis, we are interested in approaches relying entirely on neuroevolution, without the need for gradient-based methods, for evolving neural network parameters, using evolved or fixed topologies. In addition to these approaches, Various benchmark problems covering different problem classes sch as classification, continuous control or game planning, have also have been proposed in the literature for evaluating and comparing the different algorithms.

The focus of this thesis is the development of a framework that implements a selection of neuroevolution algorithms. The framework is used to evaluate and compare the algorithms on a selection of benchmarks. The framework is implemented in Rust. It allows for the visualization of the problems and the solution process through a graphical user interface. The algorithms can be run and tested through a command line interface, in order to allow for the execution of experiments and the collection of results.

Algorithms and benchmarks implemented in the framework were selected from the state-of-the

art in the theoretical and applied research in the field of neuroevolution, with a particular focus on recent algorithms and becnhmarks proposed in 2023 and 2024 in the neuroevolution theory literature. In addition to these proposals, NEAT (NeuroEvolution of Augmenting Topologies), a classic algorithm in the field, and the use of evolution strategies with CMA-ES (Covariance Matrix Adaptation Evolution Strategy), achieving state-of-the-art results, were also considered. Regarding the benchmarks, in addition to the simple two dimensional binary classification benchmarks from the considered theory litterature, the classic double pole balancing problem and the CANCER1 classification problem were also implemented.

1.0.1 Overview

The ordering of chapters in this report follows the chronological order of work performed for this project.

Chapter Chapter 2 gives the results of a literature review conducted on neuroevolution algorithms and benchmarks. It presents the state-of-the-art in the field, with a particular focus on recent algorithms and benchmarks proposed in the neuroevolution theory literature from 2023 and 2024. Moreover, it also presents the selection of algorithms and benchmarks that were considered in this thesis and implemented in the framework, motivating the choices and discussing the selected algorithms and benchmarks in further detail.

Furthermore, chapter Chapter 3 describes the process of designing and implementing the framework for running neuroevolution algorithms on benchmark problems, gathering statistics on these runs, and allowing for a visualization of the problems and solution process through a graphical user interface. This is done by first specifying the goals and requirements of the framework, followed by a walkthrough of the main design and implementation points of the framework.

Moreover, chapter **??** lays out the results collected from running experiments using the implemented framework. Results are presented for the selection of algorithms and benchmarks, and are discussed and compared in detail. Based on these results, a collection of observations, hypothesises and guidelines for algorithms and parameters selection, given the problem at hand, are presented. These observations are backed up by statistical tests compiled from additional experiments, which are also summarized in the chapter.

Finally, chapter **??** identifies the limitations of the project and discusses the potential lines of future work for refining and expanding on the results previously presented. The main work and contributions of this project are finally summed up in chapter **??** which concludes the report and reflects on its results.

Chapter 2

Literature Review

...

2.1 Methodology

...

2.2 Neuroevolution algorithms

2.2.1 (1 + 1) NA

The $(1 + 1)$ NA algorithm and its variants were introduced in Fischer, Larsen, and Witt 2023. In this work, the authors consider a simple neuroevolution setting where these algorithms are used to optimize the weights and activation function of a simple artificial neural network.

The artificial neural network topology

Artificial neurons with D inputs and a binary threshold activation function are considered. These neurons have D parameters, the input weights $w_1, \ldots, w_D$ and the threshold t. Let $x = (x_1, \ldots, x_D) \in \mathbb{R}^D$ be the input of the neuron. The neuron outputs 1 if $\sum_{i=1}^{D} w_i x_i \geq t$ and 0 otherwise. This can be interpreted geometrically as the neuron outputting 1 if the input vector x is above or on the hyperplane with normal vector $w = (w_1, \ldots, w_D)$ and bias t. j Furthermore, an alternative representation of the decision hyperplane can be used by considering spherical coordinates. The normal vector to the decision hyperplane is described by by $D - 1$ angles and the bias, where the bias corresponds to the distance from the origin measured in the opposite direction of that of the normal vector. As a matter of fact, for $D = 2$, the normal vector can be represented by its cartesian coordinates (x_1, x_2) or by its polar coordinates (r, θ), where r is the distance from the origin and θ is the angle with the x_1 axis. Similarly, for $D = 3$, the normal vector can be represented by its cartesian coordinates (x_1, x_2, x_3) or by its spherical coordinates (r, θ, ϕ), where r is the distance from the origin, θ is the angle with the x_1 axis and ϕ is the angle with the x_3 axis. It is easy to convert between these two representations. In addition, the spherical representation uses one less parameter than the cartesian representation, and hence, allows for the reduction of the number of inputs to the neurons to $D - 1$.

The ANNs which are considered in the study contain two layers, a hidden layer with $N > 1$ neurons and an output layer with a single neuron. Each of the hidden neurons are connected to the D inputs and output a binary value. The output neuron is connected to the N hidden neurons and computes the Boolean OR function of their outputs. This architecture is motivated by the problems which are considered in the study, described in **??**. Geometrically, these ANNs output the union of a number of N-dimensional hyperplanes.

The (1 + 1) NA algorithm
Let's consider an ANN with N neurons in the hidden layer and D inputs, with parameters $(\phi_{1,1}, \ldots, \phi_{1,D-1}, b_1, \ldots, \phi_{N,1}, \ldots, \phi_{N,D-1}, b_N$. In the paper Fischer, Larsen, and Witt 2023, the search space $[0, \ldots, n]^{ND}$ is considered, where r is the resolution of the continuous $[0, 1]$ domain. This discretiisation allows for the values $\{0, 1/r, 2/r, \ldots, 1\}$. Setting the parameters of ANNs is tipically a continous optimization problem, but rigurous runtime analysis is much less developed for continous optimization than for discrete optimization, which motivates this choice. Let $f : \{0, \ldots, r\}^{ND} \to [0, 1]$ be the fitness function which measures the performance of the ANN and is to be maximized.

The (1 + 1) NA algorithm is given in Algorithm 1. It maintains a single individual and mutates all angles and biases independently, based on a global search operator using the harmonic distribution Harm(r) on $\{1, \ldots, r\}$: For $l \sim$ Harm(r),

$$Prob(l = i) = \frac{1}{H_r} \text{ for } i = 1, \ldots, r, \text{ where } H_r = \sum_{i=1}^{r} \frac{1}{i}.$$

Algorithm 1 (1 + 1) NA

$t \leftarrow 0$
Select x_0 uniformly at random from $\{0, \ldots, r\}^{DN}$.
while termination criterion not met **do**
  Let $y = (\varphi_{1,1}, \ldots, \varphi_{1,D-1}, b_1, \ldots, \varphi_{N,1}, \ldots, \varphi_{N,N-1}, b_N) \leftarrow x_t$;
  for all $i \in \{1, \ldots, N\}$ **do**
    Mutate φ_i and b_i with probability $\frac{1}{DN}$, independently of each other and other indices; Mutation chooses $\sigma \in \{-1, 1\}$ uniformly at random and lHarm(r) and adds σl to the selected component, the result is then taken modulo r for angles and modulo $r + 1$ for biases.
    for $i \in \{1, \ldots, N\}$ **do**
      Set bias $2b_i/r - 1$ for neuron i.
      for $j \in \{1, \ldots, D\}$ **do**
        Set the j-th polar angle to $2\pi\varphi_{i,j}/r$ for neuron i.
      end for
    end for
    Evaluate $f(y)$
    if $f(y) \geq f(x_t)$ **then**
      $x_{t+1} \leftarrow y$
    else
      $x_{t+1} \leftarrow x_t$
    end if
  end for
  $t \leftarrow t + 1$
end while

2.2.2 Bias-Invariant (1+1) NA (BNA)

In **bna**, the authors extend upon the analysis in Fischer, Larsen, and Witt 2023 by considering more realistic ANN settings, presenting the Bias-Invariant (1+1) NA (BNA) algorithm. The considered ANNs uses Rectified-Linear-Unit (ReLU) activation functions, commonly used in real-world ANNs. This allows for the construction of bended hyperplanes, resulting in solutions to the problems described in **??** which are invariant to the bias.

The artificial neural network topology
The considered ANNs contain three layers, in which each of the neurons uses a ReLU activation function i.e they output $\max(0, \sum_{i=1}^{k} w_i x_i)$ for k inputs from the previous layer. The weights between the first and second layer and between the second and third layer are fixed. The topology for $D = 2$ in shown in **??**. The use of ReLU activation functions results in piecewise linear output. Hence, as described in **??** for the case $D = 2$, these networks compute a V-shaped area of positive classification. Such topologies are considered as a single neuron, refered to as a **V-neuron**, and which can be part of a standard ANN topology.

Therefore, these V-neurons can be described by $D + 1$ parameters:

- The bias b
- The $D - 1$ angles $\varphi_1, \ldots, \varphi_{D-1}$.
- The bend angle θ.

The area of positive classification is a (multi-dimensional) cone, all points positively classified correspond to points forming an angle smaller than the bend angle θ with the normal vector to the hyperplane given by the bias b and the $D - 1$ angles $\varphi_1, \ldots, \varphi_{D-1}$.

The Bias-Invariant (1+1) NA algorithm
The BNA algorithm is given in Algorithm 2. It is mostly the same as the (1+1) NA algorithm, with the difference that the bend angles are also mutated.

Algorithm 2 Bias-Invariant (1 + 1) NA (BNA)

$t \leftarrow 0$
Select x_0 uniformly at random from $\{0, \ldots, r\}^{DN}$.
while termination criterion not met **do**
  Let $y = (\theta_1, \varphi_{1,1}, \ldots, \varphi_{1,D-1}, b_1, \ldots, \theta_N, \varphi_{N,1}, \ldots, \varphi_{N,N-1}, b_N) \leftarrow x_t$;
  for all $i \in \{1, \ldots, N\}$ **do**
    Mutate φ_i and b_i with probability $\frac{1}{(D+1)N}$, independently of each other and other indices;
    Mutation chooses $\sigma \in \{-1, 1\}$ uniformly at random and $l\text{Harm}(r)$ and adds σl to the selected component, the result is then taken modulo r for angles and modulo $r + 1$ for biases.
    for $i \in \{1, \ldots, N\}$ **do**
      Set bias $2b_i/r - 1$ for neuron i.
      Set bend angle $\pi\theta_i/r$ for neuron i.
      for $j \in \{1, \ldots, D\}$ **do**
        Set the j-th polar angle to $2\pi\varphi_{i,j}/r$ for neuron i.
      end for
    end for
    Evaluate $f(y)$
    if $f(y) \geq f(x_t)$ **then**
      $x_{t+1} \leftarrow y$
    else
      $x_{t+1} \leftarrow x_t$
    end if
  end for
  $t \leftarrow t + 1$
end while

2.2.3 The CMA-ES Evolutionary Strategy

CMA-ES, short for *Covariance Matrix Adaptation Evolution Strategy*, is a kind of evolution strategy (ES). An ES is an optimization technique based on evolution, and belonging to the class of evolutionary algorithms (EA). This kind of blax-box optimization algorithms aim at optimizing a function $f : \mathbb{R}^n \to \mathbb{R}$, for which the analytic form is not known, but for which evaluations of the function are possible. As it ts the case for CMA-ES, These algorithms are typically stochastic and are used for the optimization of non-linear or non-convex continuous optimization problems.

ES algorithms maintain a population of candidate solutions. These candidate solutions are sampled from a multivariate normal distribution. Parameters of the distribution are updated at each generation based on the performance of the candidate solutions. As a matter of fact, a simple greedy ES algorithm could consist in updating the mean of the distribution, and using a fixed standard deviation. The mean is updated to the best solution after the evaluation of the fitness of each of the candidate solutions. The next generation is then sampled around this mean. However, this kind of simple greedy algorithms is particularly prone to getting stuck at local optima because of the lack of exploration.

In order to allow for more exploration, rather than exclusively relying on the single best solution, genetic algorithm maintain a proportion of the best solutions from the current generation, and generate the next one trough recombinations and mutations. However, this approach is also prone to getting stuck at local optima, as in practice, candidates solutions end up converging to a local optimum.

CMA-ES addresses these issues and allows for the adaption of the search space when needed, reducing it when the confidence in current solutions is high, for fine-tuning, or increasing it when the confidence is low, in order to allow for more exploration. This is done by adapting the covariance matrix of the multivariate normal distribution, which stores pairwise dependencies between the parameters for the sample distribution. This makes CMA-ES a powerful and widely used optimization algorithm. The main drawback of this algorithm is its computational cost, induced by the use of the covariance matrix, which makes it less suitable for high-dimensional problems.

...

2.3 Neuroevolution benchmarks

2.3.1 Unit hypersphere sphere classification problems

These problems, which can be thought of as a kind of ONEMAX for the (1 + 1) NA algorithm, were introduced in Fischer, Larsen, and Witt 2023. These problems consist in the binary classification of points in the D-dimensional unit hypersphere.

Half The HALF problem consists of all points with non-negative x_D coordinate on the unit hypersphere:

$$\text{HALF} = \{x \in \mathbb{R}^D, \|x\|_2 = 1 \text{ and } \varphi_{D-1} \in [0, \pi]\}.$$

Quarter The QUARTER problem consists of all points with non-negative x_{D-1} and x_D coordinate on the unit hypersphere:

$$\text{QUARTER} = \{x \in \mathbb{R}^D, \|x\|_2 = 1 \text{ and } \varphi_{D-1} \in [0, \pi/2]\}.$$

TwoQuarters The TWOQUARTERS problem consists of all points with either both negative or non-negative x_{D-1} and x_D coordinate on the unit hypersphere:

$$\text{TWOQUARTERS} = \{x \in \mathbb{R}^D, \|x\|_2 = 1 \text{ and } \varphi_{D-1} \in [0, \pi/2] \cup [\pi, 3\pi/2]\}.$$

LocalOpt The LOCALOPT problem consists of all points with polar angle φ_{D-1} between 0 and 60, 120 and 180, 240 and 300 degrees:

$$\text{LOCALOPT} = \{x \in \mathbb{R}^D, \|x\|_2 = 1 \text{ and } \varphi_{D-1} \in [0, \pi/3] \cup [2\pi/3, \pi] \cup [4\pi/3, 5\pi/3]\}.$$

Chapter 3

The framework

...

3.1 Requirements

3.1.1 Goals and functional requirements

The overreaching goal of the framework is to provide a tool for the evaluation of neuroevolution algorithms on benchmark problems, based on the selection presented in Chapter 2. Tests are specified through a command line interface, they consist in an algorithm and problem pair, along with a set of additional parameters. The framework collects the results of the tests as well as the list of passed-in parameters, algorithm and problem. These tests can either be run individually or in batch mode, where the framework runs a set of tests in parallel, and collects statistics on these runs.

Furthermore, the framework also allows for the visualization of the problem, solution process and network structure through a graphical user interface. In addition, it generates graphs for visualizing the test results, plotting the performance of the algorithm on the benchmark problem over the generation count.

3.1.2 Non-functional requirements

Non-functional requirements are the requirements that specify the quality of the system, rather than the features it should have. Apart from the functional requirements that specify the features expected of the framework, a number of non-functional requirements have also been identified.

- Usability and user experience: the framework should be easy to use and provide a good user experience.
- Documentation: The framework should be well documented, providing a clear and concise guide on how to use it.
- Error handling: All errors should be handled gracefully as to not result in runtime errors.
- Performance: The framework should allow for the execution of tests in parallel, making use of multiple CPU cores.
- :Extensibility: The framework should be easily extensible, allowing for the addition of algorithms and benchmarks without any major changes to the existing codebase.
- Support: The framework should be able to run on the three major operating systems: Windows, Linux and MacOS.

3.2 Architecture

The framework is implemented in Rust. This general-purpose programming language, originally intended to serve as an alternative for system languages such as C and C++, offers a good balance between performance provided by such low-level languages and the safety and ease of use of higher-level languages such as Python. Furthermore, various libraries (referred to as Rust crates) which could be used when implementing aspects of the framework, such as designing the command-line interfaces and graphical-user interfaces, or running CMA_ES are available in Rust. Lastly, the language can target a range of platforms, including Windows, Linux and MacOS.

The framework is divided into three main components:

- The core: This component is responsible for the execution of the tests, the collection of the results and the generation of the graphs. It contains the algorithms and benchmark implementations.
- The command-line interface: This component allows the user to specify the tests to be run, as well as the parameters for these tests.
- The graphical user interface: This component allows the user to visualize the problem, solution process and network structure, as well as the test results.

The dependency graph of the framework is shown in **??**.

3.2.1 Core

...

3.2.2 Command-line interface

...

3.2.3 Graphical user interface

...

Bibliography

Fischer, Paul, Emil Lundt Larsen, and Carsten Witt (2023). “First Steps Towards a Runtime Analysis of Neuroevolution”. In: *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*”. Association for Computing Machinery”, ”61–72”.

Appendix A

Title

Technical
University of
Denmark

Richard Petersens Plads, Building 324
2800 Kgs. Lyngby
Tlf. 4525 1700

www.compute.dtu.dk