LU-TP 20-24
June 2020

Self-adaptive random walk with pesudo-gradients for genetic evolution of an artificial neural network

Robin Emanuelsson

Department of Astronomy and Theoretical Physics, Lund University

Master thesis supervised by Patrik Edén

LUND
UNIVERSITY

Life belongs to the living,
and he who lives must be
prepared for changes.

Goethe

Abstract

To optimize the weights in an artificial neural network most methods rely gradients, which are not always obtainable or desirable. Evolutionary algorithms are instead based on Darwinian evolution where no derivative is needed. These algorithms have a set of strategy parameters that can be dynamically updated during the search to increase performance. Two ways of updating the parameters are the so called "1/5th-rule", which uses the offspring survival rate to self adapt, and random mutation which uses inheritance and mutation to evolve the strategy parameters as well.

We present an algorithm that combines the aspects of these two self adaptation methods by changing strategy parameters differently for new offspring and older survivors. We also introduce a pseudo-gradient by adding a memory of the previous step taken in the search space and let the new mutation be shifted by this remembered step. In this investigation these two methods failed to improve the performance over the "1/5th-rule" but performed better than the random mutation. The new algorithms showed promising results regarding combining the aspects of the "1/5th-rule" and random mutation.

Popular scientific description

Ever since the invention of the computer, there has been a demand for faster computers both for commercial and scientific use. One such scientific use is data classification, and it has turned out to have a mind boggling amount of applications; from self-driving cars to fraud detection.

Machine learning is a way to classify data. To make this process more effective scientists have turned towards nature, and especially evolution, for inspiration. In evolution the survivors pass on their characteristics to successive generations, but there is always a chance of small mutations that can benefit the individual or be a disadvantage. Since a mutation that benefits the individual increases its fitness for survival and gives it a higher chance to reproduce; we have a natural selection algorithm. This way of thinking can be applied in machine learning. A candidate solution to a problem is seen as an individual and a group of these candidate solutions are called a population. The population is spread out and their positions can be seen as slight differences in genetic stock. Depending on their position, the individuals have different fitness with respect to the problem that needs be optimized. Now the best individuals, or solutions if you will, will survive and reproduce in some manner and hopefully their offspring will be even better individuals.

These computer algorithms are called evolutionary algorithms. The individuals are reproducing by taking a random step away from the surviving individual. Imagine how a tree sends out pollen in the spring. The pollen spreads in all directions from the tree, and hopefully for the tree some direction was good so a new tree can start to grow. This reproduction procedure is not the most effective way of finding new and better individuals, due to its random nature. In this project we propose a new algorithm.

Imagine yourself and a group of friends are on a quest for the perfect four-leaf clover. Once you all are at the starting point you are faced with a decision; what direction should you go to find this four-leaf clover? You all head out in different directions. You strike luck and find some regular clover. You think for yourself that this was a good direction and keep going in this general direction in hopes of finding the four-leaf clover. Some of your friends weren't so lucky and they decide to go to your position where they know there at least is some clover. They aren't so venturous now when they have already failed once so they decide to start searching much closer to your previous position. Some are starting to get their hopes up again and regain their courage and start searching in a direction that they seem to have a higher concentration of clovers while other give up completely. We hope to improve the training of computers with an algorithm that is based on this type of logic.

Contents

List of acronyms

ANN Artificial Neural Network
EA Evolutionary Algorithm
EP Evolutionary Programming
ES Evolution Strategy
DW Dynamic Walk
GA Genetic Algorithm
GM Genetic Memory
RW Random Walk

List of Tables

List of Figures

1 Introduction

Artificial neural networks (ANN) are machine learning methods that try to model the networks of neurons in animal brains, which has become increasingly popular since its formulation in the 1980s [1]. An overwhelming majority of studies using an ANN used a gradient descent method [2]. The ANN usually forms a highly non-linear search space which may contain many local minima which can cause convergence problems for a purely gradient based search algorithm. To address these problems, there are modifications to the gradient descent method such as Adam [3]. For problems where it is difficult to recover a clear analytical solution and/or where a gradient is not obtainable one can instead train the ANN with so called evolutionary algorithms (EA) [5]. EA are a group of algorithms based on the Darwinian concept of survival of the fittest, where a population of solutions to the original problem reproduce and evolve depending on some fitness [4]. There are three main classes of EA:

- Evolution Strategy (ES)
- Genetic Algorithms (GA)
- Evolutionary Programming (EP)

Even though these different algorithms were easy to distinguish when they were first conceived, today it's difficult to build disjunct sets [10].

The success of an EA is usually dependent on a set of well chosen strategy parameters. These parameters do not take part in the calculation of the fitness, but are passed on to the offspring. Examples of strategy parameters are mutation strength, probability to mutate and probability for crossover between two individuals. In this project we will focus on mutation strength and a genetic memory in the form of a remembered previous step in the search space.

How well the strategy parameters perform can change drastically from problem to problem, and even during the search itself. This requires a set of self adaptive parameters. One such method is the so called "1/5th-rule". This method increases or decreases the parameter depending on the offspring survival rate. However, this method changes the parameter of all the individuals by the same amount [9]. This need not be the best option since one individual could be in a dramatically different environment compared to other individuals, and therefore could be needing a different set of parameters.
Another method of self adaptation is to let the offspring inherit a randomly modified version of the parent's parameter [11]. In this project we introduce a dynamical self adaption that treat new-born and older individuals differently, to try to combine the advantage of having a random mutation with a global offspring survival based parameter adaptation.

Directed mutation is another self adaptation method. Solutions in the population are usually mutated with a spherically symmetric distribution, such as a normal distribution.

By introducing a spherically asymmetrical distribution you can direct the mutation. This direction is a form of self adaption in that the better direction should propagate through the search. Using this, one can imitate a pseudo-gradient by letting it mutate in a preferred direction [6]. This method introduces another set of directional parameters and the need to construct a complex asymmetrical probability distribution. In an attempt to simplify we introduce a step direction in the form of a genetic memory (GM) that gets passed down from the parent to the offspring, which could act as a pseudo-gradient.

2 Evolutionary Algorithm and Artificial Neural Network

2.1 Basic Evolutionary Algorithm

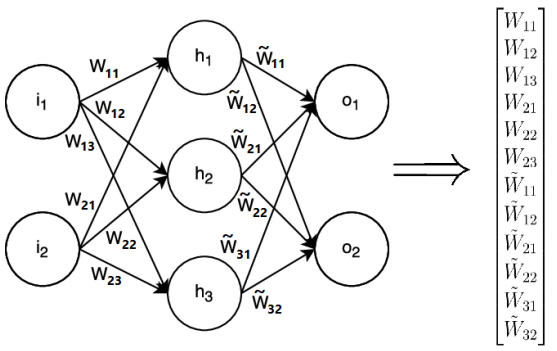


Figure 1: Illustration of how the weights in an ANN gets turned in to a solution vector so that it can be used in an EA. The bias weights/nodes have been omitted, but the procedure is exactly the same.

An EA can be broken down to a few general steps. A solution to the problem must be able to be represented as a vector of numbers. In the ANN case it's done by letting the weights in the network form a solution vector, see Figure 1. A so called individual contains: a solution vector, $\mathbf{x}$; a set of strategy parameters, $\mathbf{s}$, where the parameters governs the evolution of $\mathbf{x}$; and a fitness dependent on the solution vector, $F(\mathbf{x})$. Using a similar notation as in [10], this can be written as

$$I = \{\mathbf{x}, \mathbf{s}, F(\mathbf{x})\}\,. \tag{2.1}$$

A set of such solution strings form the population.

$$P = \{I_k\}\,. \tag{2.2}$$

2.1.1 Reproduction

Crossover recombines and mixes parts of the parents $\mathbf{x_1}, \ldots, \mathbf{x_p}$ to form a new offspring. Usually this is done with two parents, but multiple parents can be used. The typical crossover operator combinations are usually dominant or intermediate. Dominant crossover randomly combines the elements from the parents. That is, with p parents $\mathbf{x_1}, \ldots, \mathbf{x_p} \in \mathbb{R}^d$ produce an offspring $\mathbf{x}' = (x'_1, \ldots, x'_d)$ such that the i-th components is

$$x'_i = (x_i)_j, \quad j \in \text{random}\,(1, \ldots, p). \tag{2.3}$$

For intermediate crossover, x' is the arithmetic mean of the components of all p parents

$$x'_i = \frac{1}{p} \sum_{j=1}^{p} (x_i)_j \tag{2.4}$$

The idea behind crossover is that individuals with good fitness relative to the population have good subvectors. By combining the subvectors of two parents the resulting vector will have components spread over the search space, giving the algorithm a global search property [9].
This project focuses on mutation and for clarity crossover has been omitted.

Mutation adds some noise to the individual, usually Gaussian noise, according to:

$$\mathbf{x}' = \mathbf{x} + \mathbf{z} \tag{2.5}$$

where

$$\mathbf{z} \sim \sigma \cdot \mathcal{N}(0, \mathbb{1}) \tag{2.6}$$

Here σ is called the mutation width and it determines how much noise is added to the solutions. By adding mutation to the algorithm we allow it to reach, in theory, all of the points in the search space. The mutation also introduces a local search and a way to avoid local minima [9].

2.1.2 Selection

After reproduction, the population, both parents and offspring, get evaluated by a loss function to determine the individual's fitness. For the ANN case this is a function that depends on the output of the network and the training data. Selecting which individuals get to survive to the next generation can be done in a number of different ways. Some examples are:

- **Roulette Wheel Selection.** Each individual in the population is given a probability of getting drawn. Better fitted individuals get a higher probability.
- **Tournament Selection.** Randomly sampling two individuals of the whole population. The best of these two gets to reproduce while the other is removed.
- **Elitism Selection.** Choosing the best individuals to reproduce and remove the rest.
- **"Roulette Wheel-Elitism" Selection.** A percentage of of the best ranking individual are certain to reproduce while the same percentage of worst ranking individual are removed. The remaining individuals are picked at random with equal probability to be selected for reproduction. The individuals that were not picked gets removed.

The thought behind different selection methods is to avoid local minima by, often, letting some sub-optimal individuals reproduce [10]. The last of these options is the one we used.

After the selection phase the process repeats itself for a number of generations, or until some criterion is met.

2.2 Self-adaption

2.2.1 The "1/5th-rule"

By changing σ during the search, the algorithm can adjust itself between a global and a more local search. One such method is the "1/5th-rule" [9]. During the optimization process, σ changes according to:

$$\sigma' = \begin{cases} \sigma/c, & \text{if} \quad f > f_0 \\ \sigma \cdot c, & \text{if} \quad f < f_0 \\ \sigma, & \text{if} \quad f = f_0. \end{cases} \tag{2.7}$$

Here f is the ratio between the number of successful mutations and the total number of mutations, and must be reevaluated for each generation. The critical value f_0 is a free parameter and $f_0 = 1/5$ was originally developed to find the optimal progress rate for

two functions, the hypersphere and the rectangular ridge, but has been very effective in maintaining a very high progress rate for other problems [13]. The tuning parameter c is given in the interval $0 < c < 1$. The idea behind this method is that if the fitness of the offspring is too high, that is $f > 1/5$, then it's assumed that we are far from the optimum, since almost all changes lead to better fitness. The method therefore increases σ to span a larger part of the search space. The opposite is assumed for the the other case, when too few offspring survive. Then σ is probably to large and should be reduced. This method is limited by the fact that σ is the same for all of the individuals. One can imagine a scenario where one individual is close to an optimum and should therefore need a small σ, while others are quite far from one and should consequently need a larger σ.

The "1/5th-rule" was developed for a so called (1+1)-ES. The (1+1)-ES is a special case of the more general $(\mu/\rho \stackrel{+}{,} \lambda)$-ES algorithm. In the $(\mu/\rho \stackrel{+}{,} \lambda)$-ES the population consists of μ parents and λ offspring. The parameter ρ determines the number of parents which take part in the procreation of a single individual. The difference between $(\mu/\rho, \lambda)$ and $(\mu/\rho + \lambda)$ is how the individuals participating in the selection are chosen. For the first case, only the offspring are considered for selection and parents die per definition. In the other case, the selection pool consists of both parents and offspring [10].

2.2.2 Random Mutation

An alternative approach to make self adaptation for the more general case of (1+λ) is to change the mutation width multiplicatively by a random number ξ according to

$$\tilde{\sigma}_{k'} = \xi \sigma_{k'}, \quad k' = 1, \ldots, \lambda \tag{2.8}$$

and with

$$E[\xi] \approx 1. \tag{2.9}$$

One of the best known types of this kind of self adaption is the so called log-normal operator

$$\xi = \exp(\tau \mathcal{N}(0, 1)) \tag{2.10}$$

where τ is the learning parameter and is usually set to $0 < \tau \lessapprox 1$ [11]. The idea behind this is that good parents, that is an individual with good fitness and mutation width, should create good offspring that inherit this good mutation width. But a good mutation width is, as we have said before, not static, so noise needs to be introduced. Since $E[\xi] \approx 1$, there is no systematic drift in σ, and with a small τ, the good mutation width should on average only be slightly changed for its offspring. The best of these offspring will in turn pass on their mutated mutation width to the next generations.

2.2.3 Dynamic Walk

For a $(\mu + \lambda)$-ES based scenario, we propose a self adaption where the offspring inherits the parent's mutation width σ.

$$\mathbf{x}_{k'} = \mathbf{x}_k + \sigma_k \mathbf{z} \quad \mathbf{z} \sim \mathcal{N}(0, \mathbb{1}) \tag{2.11}$$

where k and k' are indices for the parent and its offspring. The mutation widths are updated according to:

$$\begin{cases} \sigma_{k'} = \gamma B \sigma_k, & \text{if offspring} \\ \sigma_k = \gamma \sigma_k & \text{if parent} \end{cases} \tag{2.12}$$

where $0 < \gamma < 1$ and $B > 1$.
If γ and B are chosen such that $\gamma B > 1$, the mutation width σ will increase for the surviving offspring. This is the property we wanted to adapt from the "1/5th-rule" but instead affecting the individual. If the offspring survives many generations the offspring's σ can become quite large while searching a big part of the search space, given the global search property. If only the parents survive, σ will decrease over the generations, giving the local search property. The potential advantage over the "1/5th-rule" is given by illustration of an imaginary scenario in Figure 2.

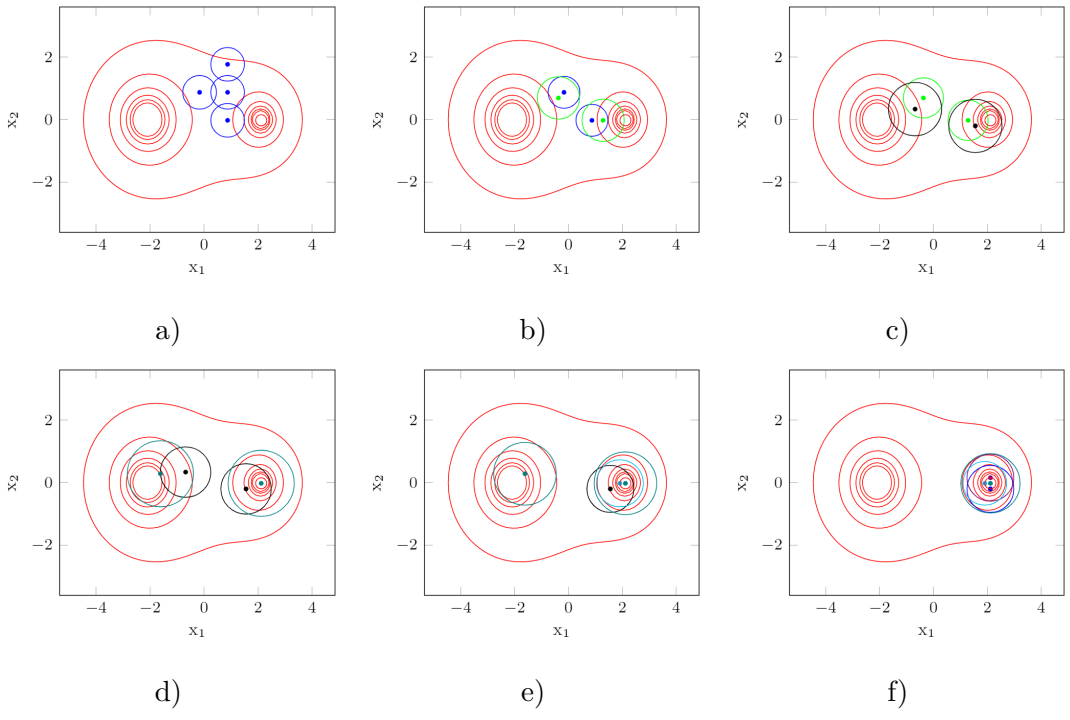


Figure 2: Illustration of a possible scenario using a dynamic walk algorithm. Each individual, represented by a point in the 2D search space has some mutation width, represented by a circle, that changes during the search. A new colour represents a new generation that survives the selection phase.

In Figure 2 in the top left corner shows four individuals, denoted by points, in some 2D search space. Each individual has initially the same mutation width noted by a circle around each corresponding point; the changing colours indicate new generations of individuals. The contour lines symbolise some fitness landscape with two minima; one is the global while the other is local. In the next panel to the right the individuals have reproduced and gone trough the selection phase. We see that the offspring have gotten a larger mutation width while the parent's mutation width has gotten slightly smaller. The bottom panels shows how the offspring get to the minima and their mutation width get smaller, and the algorithm turns more towards a local search. Finally, the bottom right panel shows how individuals gather around the global minimum. By having an individual-based self adaption, each individual can search the space independently of the performance of other individuals, and still retain a self adaption that enables a family line to go from a global search to a local search.

One disadvantage is that this method requires two parameters, γ and B, instead of one, c or τ, for the "1/5th-rule" and random mutation, respectively.

An approximate formula for the average mutation width can be reasoned to be in the form of

$$\langle\sigma\rangle_{g+1} = \gamma\left(f \cdot B\langle\sigma\rangle_g + (1-f)\langle\sigma\rangle_g\right) \tag{2.13}$$

where f is the fraction of offspring that survived the selection phase. $\langle\sigma\rangle_g$ is the average mutation with of the whole population for the generation g. In words this formula reads: the average mutation width in the next generation $g+1$, will be the average mutation width of the offspring in the current generation g multiplied by a factor γB; the average mutation width of the parents is multiplied by a factor γ.

The formula can easily be rewritten as

$$\frac{\langle\sigma\rangle_{g+1}}{\langle\sigma\rangle_g} = \gamma\left(f \cdot B + (1-f)\right). \tag{2.14}$$

The critical value for the the left hand side is when it's equal to one, which is the dividing line for when the average mutation width increases or decreases. In Figure 3 the left hand side has been plotted against f. We see that our self adaption can mimic the properties of the "1/5th-rule", making the dividing lines between increasing or decreasing average mutation width for the two algorithms intersect at $f = 1/5$. Our algorithm has the property that the increasing of the average mutation width is, in theory, smoother than the fixed value for the "1/5th-rule".

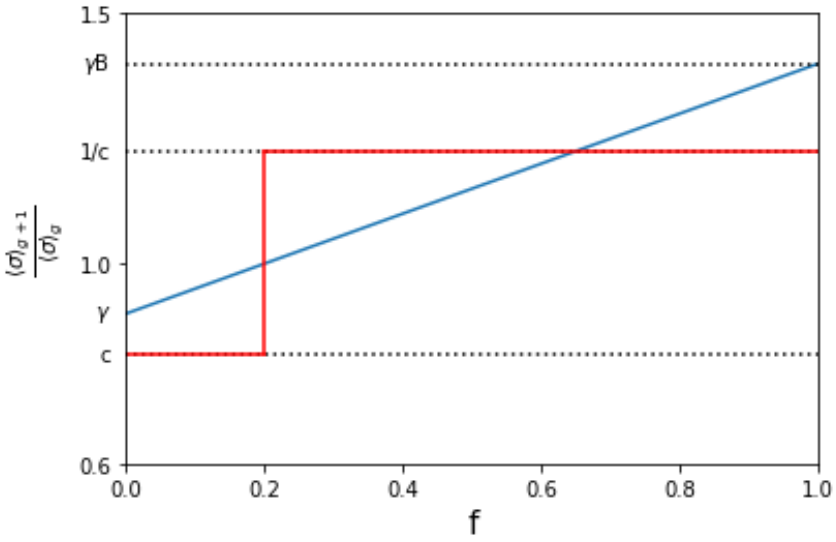
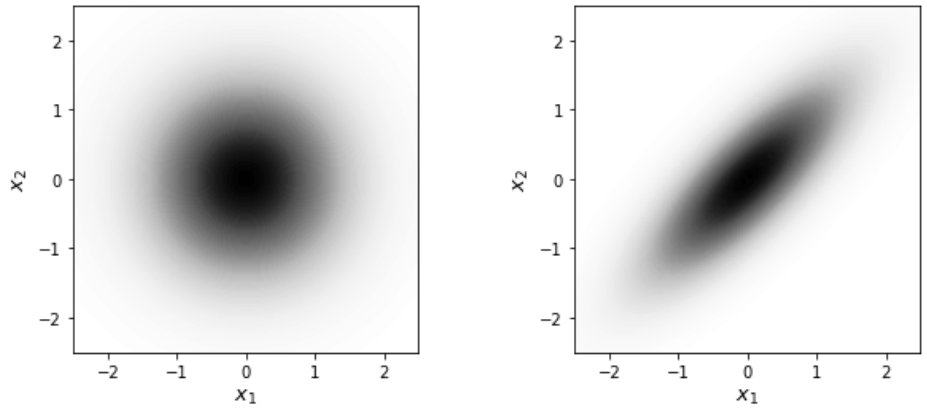


Figure 3: Plot showing the ratio of average mutation witdh between two adjacent generations as a function of the percentage of offspring that survives the selection phase. The red line is for the "1/5th-rule" and the blue line is equation 2.14.

2.3 Genetic Memory



(a) (b)

Figure 4: Illustration of different mutation width distributions in 2D. In a) we have a spherically symmetric Gaussian distribution. In b) we have an arbitrary orientated elliptic distribution.

For a DW algorithm the mutation width changes in a random Gaussian manner, as seen in Figure 4 a). One could imagine that a directed mutation could increase convergence

performance by directing the mutation towards a potential minimum. This can be done by introducing a spherically asymmetrical distribution for the mutation width [6][7] as in Figure 4 b).

We propose to introduce a "genetic memory" (GM), where the offspring remembers the step it previously took and uses it as a pseudo-gradient. The update rules becomes:

$$\mathbf{x}_{k'} = \mathbf{x}_k + \boldsymbol{\delta}_k + \sigma_k \mathbf{z} \quad \mathbf{z} \sim \mathcal{N}(0, \mathbb{1}) \tag{2.15}$$

$$\boldsymbol{\delta}_{k'} = \mathbf{x}_{k'} - \mathbf{x}_k \tag{2.16}$$

The changes in mutation widths, σ remains the same

$$\begin{cases} \sigma_{k'} = \gamma B \sigma_k, & \text{if newborn} \\ \sigma_k = \gamma \sigma_k & \text{if old} \end{cases} \tag{2.17}$$

where $\gamma < 1$ and $B > 1$ and the initial "step" $\delta^{(0)}$ is set to zero.

Since a parent's offspring is already searching a given direction δ, the direction could be considered less important to that parent. One can therefore introduce a decay to the direction step and let the parent start searching closer to its vicinity. The decay is given by

$$\boldsymbol{\delta}_k = \lambda \boldsymbol{\delta}_k \quad \text{if parent} \tag{2.18}$$

where $\lambda < 1$.

3 Method

3.1 The Algorithm

We investigated evolution where each individual creates one offspring each generation. The algorithm becomes:

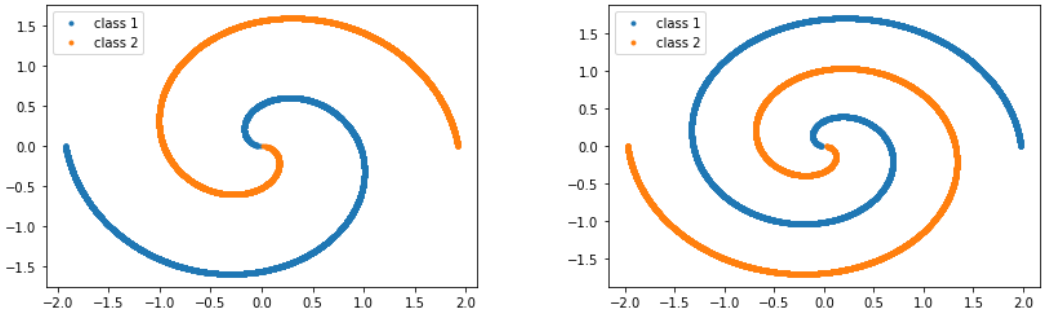
Begin;
g=0;
Assign value to $\mathbf{B}$, γ and λ;
$\boldsymbol{\delta}_k = 0$;
initialize$\Big(P^{(g)} = \{I_k = (\mathbf{x}_k, \sigma_k, \boldsymbol{\delta}_k, F(\mathbf{x}_k)), \quad k = 1, \ldots, p\}\Big)$;
while $g < g_{final}$ **do**
  forall k **do**
    $\tilde{\mathbf{x}}_{k'} = \text{mutation}(\mathbf{x}_k, \sigma_k, \boldsymbol{\delta}_k)$;
    $\tilde{\sigma}_{k'} = B \cdot \sigma_k$;
    $\tilde{\boldsymbol{\delta}}_{k'} = \text{construct direction } (\tilde{\mathbf{x}}_{k'}, \mathbf{x}_k)$;
    $\boldsymbol{\delta}_k = \lambda \boldsymbol{\delta}_k$;
  end
  $\tilde{P}^{(g)} = \Big\{I_{k'} = \Big(\tilde{\mathbf{x}}_{k'}, \tilde{\sigma}_{k'}, \tilde{\boldsymbol{\delta}}_k, F(\tilde{\mathbf{x}}_{k'})\Big), \quad k' = 1, \ldots, p\Big\}$;
  $P^{(g)} = P^{(g)} \sqcup \tilde{P}^{(g)}$;
  $P^{(g+1)} = \text{selection}\Big(P^{(g)}\Big)$;
  $g = g + 1$;
  forall k **do**
    $\sigma_k = \gamma \sigma_k$;
  end
end

Here g is the generation, P is the population, which contains p amount of individuals, I. These individuals contains the weights, $\mathbf{x}$, for the network, the individual's mutation width, σ, the direction, $\boldsymbol{\delta}$, and its fitness which is some function F of the weights $\mathbf{x}$. In the case where we have a GM, the 'construct direction' operator returns $\tilde{\mathbf{x}}_{k'} - \mathbf{x}_k$, else it returns zero.

3.2 The Data Set

The data set was a binary classification problem in the form of a spiral, seen in Figure 5. All the runs used 1000 data points and no noise. Since we are excluding noise there is no risk of overtraining. We still include a L2 regularization term hoping that the decision boundary will become smoother. With Keras, which is a model-level library for developing deep learning models [8], we found that 0.0001 was a good value for the regularization

parameter.
The number of turns in the spiral is a tunable parameter to make the problem harder/easier.



(a) (b)

Figure 5: Illustration of the data set used. In a) the data set has one turn of the spiral while for b) it has 1.5 of a turn.

3.3 The Network and Hyper-Parameters

The weights for the network were drawn from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution, where n is the number of inputs and the bias weights are drawn from a $\mathcal{N}(0, 1)$ distribution.
The hyper-parameters are the mutation width σ, a percentage of certain survivors, the decay factor γ, and in the case of a GM the direction decay λ. These were optimized by keeping all but one fixed at a time. In the first optimization run the values was set to $\gamma = 0.95$, a percentage of certain survivors was set to 100%, and in case of a GM, $\lambda = 0$. The width σ was the first parameter to be varied, then it went in the following order: γ, λ and lastly percentage of certain survivors. These optimized parameters were later used in a different set of runs to obtain the results.

To calculate the boost factor B we use equation 2.13, with $f = 0.2$ (same as the 1/5-rule) and the ratio between the current generation's average mutation width and the next generation's average mutation set to one, which is to say that on average the mutation stays the same. For the "1/5th-rule" the factor c in equation 2.7 was set to $c = 0.817$ which is the calculated value for the sphere model. However the choice of c is relatively uncritical [12]. For random mutation τ was set to $\tau = 0.95$. For all the runs an initial population of 50 individuals was used.
The network consisted of two hidden layers with 20 nodes and with corresponding bias nodes. We found that this network could solve the problem using Keras and a gradient descent method. The activation function for the non-output layer was the ReLu and the output activation function was the sigmoid function. The cost function was the binary cross-entropy function.

4 Results

4.1 Log Loss Evolution

The hyper-parameters for all the runs can be found in Appendix A.

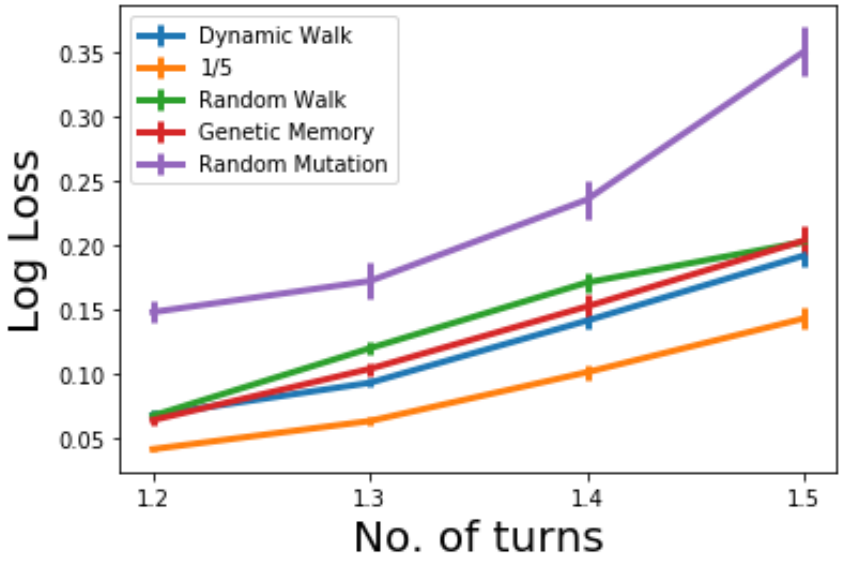


Figure 6: The best log loss for the "1/5th-rule", DW, RW, GM and random mutation algorithm for different number of turns in the spiral data set. The error bars show the standard error of the mean.

Figure 6 shows the performance of the different algorithms, those being the "1/5th-rule", dynamic walk (DW), random walk (RW, which is a special case of a DW where $\gamma, B = 1$), GM and random mutation. The log loss on the y-axis is the best log loss over an average consisting of 25 runs with 300 generations for all the different algorithms. On the x-axis we have the number of turns for the spiral data set. An averaged evolution of the log loss for these 25 runs for each of the algorithms with the number of turns set to 1.5 can be seen in Figure 7.

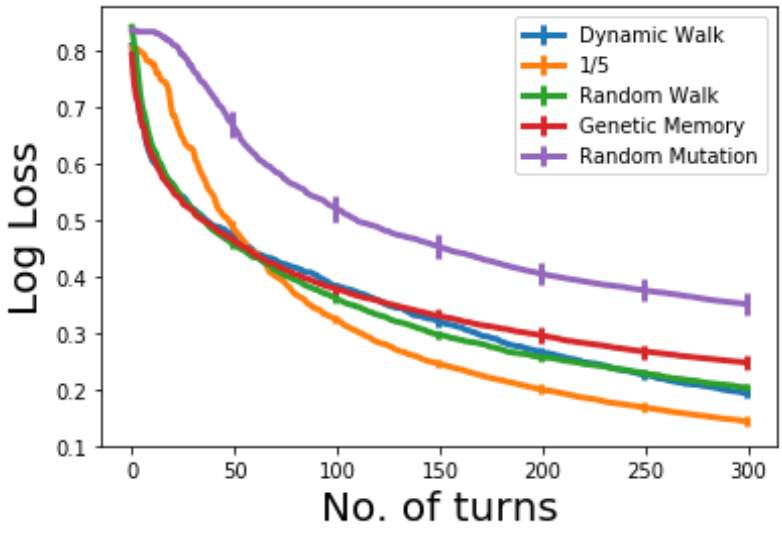
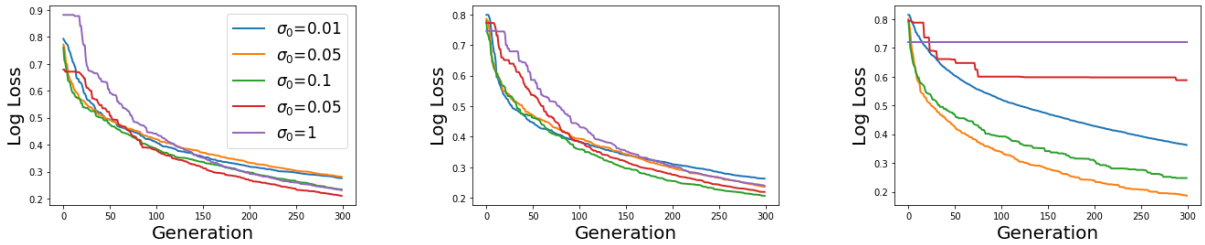


Figure 7: The average log loss evolution of 25 runs for each of the algorithms. The error bars show the standard error of the mean and are only plotted for values where x mod 50 = 0 to make it less cluttered.

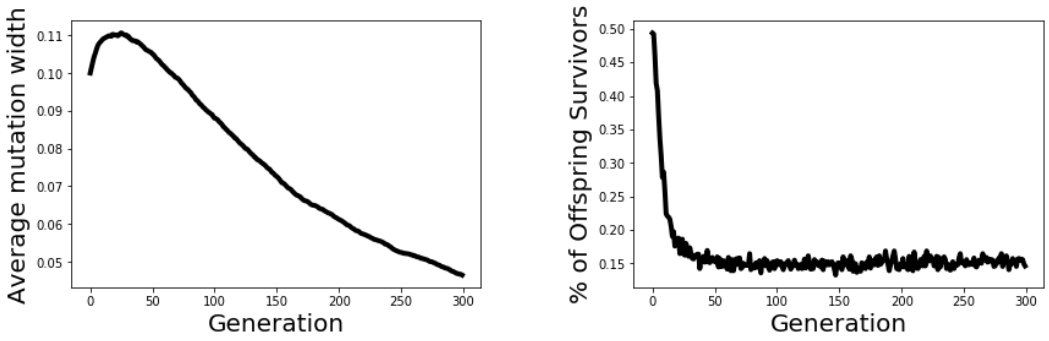


(a) Dynamic Walk (b) Genetic Memory (c) Random Walk

Figure 8: The log loss evolution for different initial mutation widths, shown as different colors. Panel a) shows a DW algorithm. The final values have a spread of 0.027 in terms of the standard deviation. The legend shown in panel a) is the same as for the rest of the panels. Panel b) shows a GM algorithm. The final values have a spread of 0.019 in terms of the standard deviation. Panel c) shows a RW algorithm. The final values have a spread of 0.202 in terms of the standard deviation.

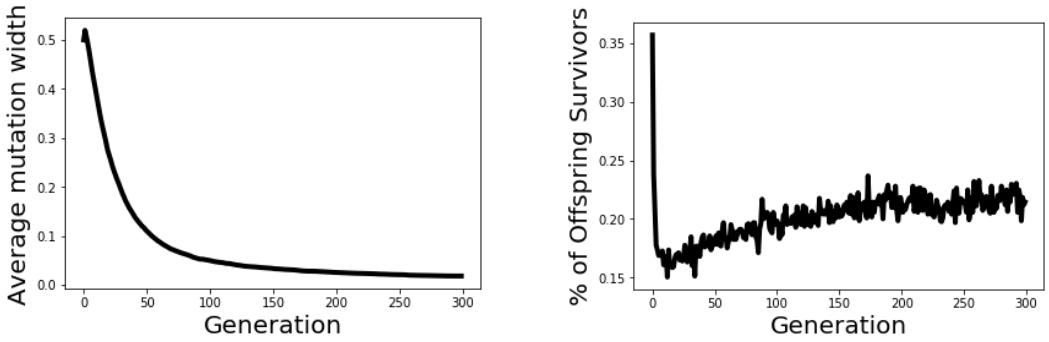
Figure 8 shows the log loss evolution using 1.5 turns over 300 generations for a DW, GM and RW algorithms with different initial values of the mutation width. The different mutation widths are 0.01, 0.05, 0.1, 0.5 and 1. Each curve is the average over 5 different runs. All the runs use $\gamma = 0.95, B = 1.26$ and a fraction of certain survivors set at 80%, to treat them on an equal footing. In the case of a GM, a $\lambda = 0.25$ was used. For DW, the final values, after 300 generations, have a spread of 0.027 in terms of the standard deviationn of, for GM it was 0.019 and for RW it was 0.202.

4.2 Mutation



(a) (b)

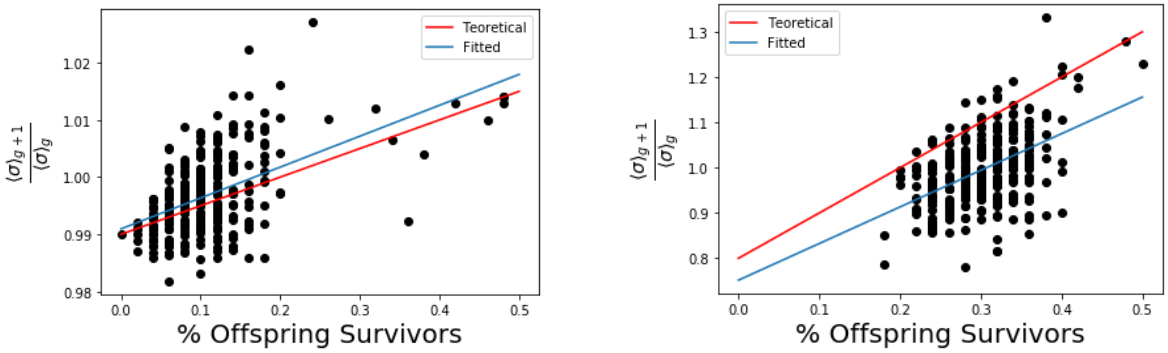
Figure 9: Panel a) shows the average evolution of the average mutation width for the whole population and panel b) shows the evolution of the average percentage of offspring survivors in or 25 runs with a DW algorithm



(a) (b)

Figure 10: Panel a) shows the average evolution of the average mutation width for the whole population and panel b) shows the evolution of the average percentage of offspring survivors in or 25 runs with a GM algorithm.

For both Figure 9 and 10 panel a) shows how the average mutation width, for all the surviving individuals in the population, evolves over the run using a DW algorithm in Figure 9 and GM in Figure 10. Panel b) shows the fraction of offspring that survive the selection phase each generation, which contributes with a boosting factor γB to the average mutation width. This data is taken from the same runs as in Figure 7.



(a) Dynamic Walk (b) Genetic Memory

Figure 11: Equation 2.14, which tells us how the average mutation width changes between two adjacent generation as a function of the offspring that survived the selection phase, is plotted in red with different parameters γ and B for the two panels. The experimental data is plotted as black dots togheter with a fitted blue line.

The agreement between the approximated change in mutation width between generations in equation 2.14 and a linearly fitted line to the experimental data, shown as black dots, is plotted in Figure 11. The experimental data comes from a single run consisting of 300 generations, using 1.5 turns and a selection phase that only picks the best half of the population, to avoid skewing the fraction of offspring survivors. The other parameters used are the ones optimized for 1.5 turns. Panel a) is for a DW algorithm and panel b) is for a GM algorithm.

5 Discussion

The log loss results presented in Figure 6 and 7 show that the "1/5th-rule" is better than both a DW and GM at finding a solution for a spiral problem, but DW and GM were better than the random mutation algorithm. When the problem is too simple, all but the random mutation algorithm can achieve a quite low log loss, but after a slight increase in difficultly there is a clear difference in performance. The random mutation stands out as being significantly worse than the rest for easier problems. Its learning parameter was not optimized, but was still within the vicinity of $\lessapprox 1$, which could contribute to its poor performance to some extent, but probably can't explain the big performance discrepancy.

Both DW and GM are better than a RW algorithm, but a DW performed better than a GM algorithm, suggesting that adding a pseudo-gradient type step is in fact a deterioration. One explanation as to why the "1/5th-rule" performs better than the rest of the algorithms could be that since the search space is very high-dimensional and complex; it could be more favorable to let the individual have the same mutation width for several generations so as to more thoroughly search that "step size" and not force it to a local search if it does not

find any good solution immediately. When looking at the sensitivity to initial mutation widths, Figure 8 shows that DW and GM final log loss values were relatively similar even though the initial values for the mutation width differ by as much as 2 orders of magnitude. For a RW the final values differ wildly. This suggests that DW and GM are insensitive to the initial mutation width, which is otherwise a quite sensitive parameter when using an algorithm without self-adaptation.

Looking at how the average mutation width and fraction of offspring survivors behave, firstly Figure 9 and 10 show that for both algorithms the fraction of offspring survivors is large in the first generations. This is to be expected since the weights for the network are randomly initialized so the following generation should have a similar chance to perform well. In Figure 9 panels a) and b) shows that the average mutation width is increasing from its initial value when we have a large fraction of offspring survivors, which is in line with theory. For both algorithms the fraction of offspring survivors is in the proximity of 20% which supports using equation 2.14 with f=1/5 to calculate the boost factor given a decay factor.

Figure 11 shows that the data points from both algorithms have a general trend line, with some scatter. For a DW the trend seems to follow the theoretical value, while for a GM the trend line appears to be shifted downwards. This could be because a GM algorithm combines a step with some noise to create offspring. This increases the chance of a sizeable change for the offspring compared to the parent, and could give an offspring with a low noise level an advantage in the selection phase. This potential preference could shift the trend downwards.
Since the data seem to be scattered along a trend line it suggests that the algorithms incorporate the survival rate aspect of the "1/5th-rule" and random mutation in that the data points are scattered and therefore the individual behaves independently.

Even though the "1/5th-rule" performed the best, its best log loss value for the problem for 1.5 turns was still quite high. The spiral data set is not particularly suited for evolutionary algorithms simply by the fact that a gradient descent method can be used and is much more efficient both in terms of computational time and results. The results using our setup should not be interpreted in terms of solving the problem contra not solving the problem but only as a problem that can be tuned to see how the different algorithms perform. The problem could have been tunable in different ways such as adding noise and/or making the points in the data set more sparse.

The crossover operator was not included in this project but could potentially be included in future works. If this operator were to be implemented, the "1/5th-rule" has a big advantage over DW and GW. Since all the individuals have the same mutation width, it's trivial to find mutation widths for the offspring. However, for DW and GM, deciding what mutation width the offspring will have is not trivial, since the parents could have different mutation widths and in the case of GM the offspring direction step must also consider both parent's direction steps.

The λ parameter for a GM could be seen as an indirect learning rate for a gradient descent method. For a gradient descent method however, the learning parameter is applied before the update to the weights of the network, in contrast to our method where λ is applied after the parent has produced its offspring. Changing the order when λ is applied to when the offspring is created could potentially increase the performance for a GM by fine tuning the step size at each iteration. The current algorithm could be running the risk of overshooting a potential minimum by inheriting the whole direction of the parent.

The optimization method of keeping all parameters but one constant and finding an optimal value for this parameter and repeating for the rest is not the best way of finding a set of optimal parameters. The rationale for this choice is the question of time. This is also the reason for only using 300 generations. Figure 7 shows that the algorithms do not seem to have reached a final lowest log loss, since the curves have not flattened out completely. With more computational time the curve can be flattened further, but since we are only interested in the relative performance between the algorithms, 300 generations seem to be enough. Here once again the "1/5th-rule" has the advantage over DW and GM in that it has fewer parameters that need to be optimized. However, one could argue that $f_0 = 20\%$ is not at all universal, so that is another parameter that needs optimization to create a sort of "1/Xth-rule". The "1/5th-rule" also has a parameter c that in theory needs to be optimized even if it's relatively inessential to the algorithm's performance. For future studies, one could start with the (1+λ)-ES since it has a more solid theoretical foundation and would probably decrease run times, assuming reasonable λ. Another type of self adaption that could be incorporate in future investigations is to change the noise in equation 2.6 from a isotropic case, where all components are independent, to a dependent case with a correlation matrix.

6 Conclusion

The dynamic walk and genetic memory algorithms performed worse than the "1/5th-rule" for the spiral data set, and the genetic memory seems to be a direct downgrade compared to the dynamic walk algorithm. However, these two algorithms show some interesting properties that seem to combine aspects of "1/5th-rule" and a self adaptation based on random mutation, which could be the subject of further investigation. Even if the focus for this project was an artificial neural network, applications in more traditional optimization problem could be possible.

7 Acknowledgement

Firstly I would like to thank my supervisor, Patrik Edén, for making this project such a pleasant experience. Secondly I would like to thank my fellow student Anton Moberg for productive discussions and direct help with programming and such. I would also like to thank my roommate Martin Montelius for helping me proofread this report.

References

[1] A.P. de Weijer et al., "Using genetic algorithms for an artificial neural network model inversion", Chemometrics and Intelligent Laboratory Systems, 20: pp. 45–55, 1993.

[2] J.N.D. Gupta, R.S. Sexton, "Comparing backpropagation with a genetic algorithm for neural network training", Omega, 27: pp. 679-684, 1999.

[3] D.P. Kingma, J.L. Ba, "Adam: A Method for Stochastic Optimization", International Conference on Learning Representations, San Diego, United States, 2015.

[4] G. David, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, pp.1, 1989.

[5] J. Kalderstam, P. Edén, M Ohlsson, "Ensembles of genetically trained artificial neural networks for survival analysis", 21st European Symposium on Artificial Neural Networks, Bruges, Belgium, p. 333-338, 2013.

[6] L. Hildebrand, B. Reusch, M. Fathi, "Directed mutation-a new self-adaptation for evolution strategies", Proceedings of the 1999 Congress on Evolutionary Computation-CEC99, 2: pp. 1550-1557, 1999.

[7] N. Hansen, A. Ostermeier, "Completely Derandomized Self-Adaptation in Evolution Strategies", Evolutionary Computation, 9: pp. 159-195, 2001.

[8] "Keras backends". keras.io. Retrieved 2020-05-22

[9] O. Kramer, "Machine Learning for Evolution Strategies", Springer International Publishing, pp. 16-18, 2016.

[10] H-G. Beyer, "The Theory of Evolution Strategies", Springer-Verlag Berlin Heidelberg, pp. 2-6, 2001.

[11] H-G. Beyer, "The Theory of Evolution Strategies", pp. 260-262

[12] H-P. Schwefel, "Evolution and Optimum Seeking", Wiley-Interscience, pp. 100, 1995.

[13] H-P. Schwefel, "Evolution and Optimum Seeking", pp. 110

8 Appendix A

Table 1: All the hyper-parameters for the different runs and algorithms. Here 'certain fraction' means the fraction of individuals with a nonrandom selection phase. For example, '80%' means that the top 40% of the individuals are certain to survive the selection phase and the bottom 40% are certain to be removed during the selection phase.

1.5 Turns	Initial Mutation Width	Width Decay	Boost	Certain Fraction	Direction Decay
Dynamic Walk	0.1	0.99	1.05	80	–
1/5th-rule	1	–	–	80	–
Random Walk	0.05	–	–	90	–
Genetic Memory	0.1	0.8	2.25	60	0.5
Random Mutation	1	–	–	90	–
1.4 Turns					
Dynamic Walk	0.5	0.95	1.26	80	–
1/5th-rule	0.5	–	–	80	–
Random Walk	0.05	–	–	80	–
Genetic Memory	0.5	0.95	1.26	80	0.25
Random Mutation	1	–	–	50	–
1.3 Turns					
Dynamic Walk	0.1	0.99	1.05	90	–
1/5th-rule	1	–	–	80	–
Random Walk	0.1	–	–	70	–
Genetic Memory	0.1	0.99	1.05	70	0.5
Random Mutation	1	–	–	100	–
1.2 Turns					
Dynamic Walk	0.5	0.95	1.26	80	–
1/5th-rule	0.5	–	–	80	–
Random Walk	0.1	–	–	70	–
Genetic Memory	0.5	0.95	1.26	80	0.25
Random Mutation	0.5	–	–	90	–