Neuro-GOMEA
Using Modern Evolutionary Algorithms to Train Neural Networks

Author:
Luc Everse
4470397

Supervisor:
Prof.dr. Peter A.N. Bosman

Committee member:
Dr. Marco Loog

Daily supervisors:
Dr. Marco Virgolin
Arkadiy Dushatskiy, M.Sc.

Algorithmics research group
Department of Software Technology
MSc Computer Science
Faculty of Electrical Engineering, Mathematics & Computer Science
Delft University of Technology

August 4, 2022

Abstract

Neural networks (NNs) have, in recent years, become a major part of modern pattern recognition, and both theoretical and applied research evolve at an astounding pace. NNs are usually trained via gradient descent (GD), but research has shown that GD is not always capable of training very small networks. As a result, networks trained via GD are often significantly larger than necessary, demanding more computing power and energy to evaluate, and thereby hampering adoption on lower-power devices. This thesis investigates whether evolutionary algorithms (EAs) can successfully train NNs and if these networks can be smaller than those required by GD. Four algorithms, namely the GD-based Adam, Adam with cold restarts, and the EAs GOMEA and BIPOP-CMA-ES, were used to train various configurations of multilayer perceptrons (MLPs) with one hidden layer for the Exclusive-OR (XOR) problem. Their relative performance was gauged by comparing the rates at which each algorithm attained an acceptable loss for a given number of hidden nodes. The main findings are that EAs could find smaller ReLU-activated networks than GD could, while, oppositely, the GD could generally find smaller networks for sigmoid-activated networks. However, GOMEA in particular was able to successfully train XOR networks using the highly discrete Heaviside activation function, whereas GD could not due gradient erasure. Problem-specific knowledge in the form of a soft symmetry breaking constraint was found to be effective to increase success rates in a limited number of cases. These findings indicate that using EAs is a viable strategy for training NNs, yielding smaller, and therefore more efficient networks than those trained via GD, provided that the network topology is suitable for the chosen EA. This opens up future research into, including but not limited to, the many ways EAs can be scaled up to training larger NNs, hybridization with GD, and niche network topologies.

Preface

If anyone in 2015 told me I'd eventually graduate in *machine learning* I would look at them very funny. My future was in hardware! If any programming were involved it would be very low-level, think VHDL and—if (un)lucky—C++. But nothing ever fully goes as planned. Deciding to completely pivot your future adds spice to life (which is why I've done it—twice).

That's not to say that any of this was my Plan B. I've loved nearly every second I spent on the subject of computer science and even machine learning. It certainly made me appreciate the latter.

Especially the concept of evolutionary algorithms resonates with me very well. The course on EAs taught by Peter cemented my appreciation of them. I was overjoyed when Peter could fit me in his already busy schedule, and on top of that proposed a project that would need ”someone crazy enough to try it”. I though that description fit me to a T (which I would prove later on) and that is how this thesis came to be.

The road toward the document that you are reading right now was long and far from smooth. I'd like to thank Peter, Marco and Arkadiy for their patience and seemingly unending willingness to put up with my problems throughout the project. Without their help this project would have been the academic equivalent of a wild dumpster fire. The same goes for my friends and family, some of whom had to put up with me on a near-daily basis at times, and also (or *still*) gave me excellent feedback and advice.

With this thesis behind me I know for sure that academic research is not for me, but I am nevertheless glad to have had the opportunity to do this work with my advisors, CWI and the TU Delft. I am proud of this thesis and the work behind it and I hope that you, dear reader, will like it too.

Luc Everse
Delft
September 28, 2022

Contents

Abstract **i**

Preface **ii**

1 Introduction **1**

2 Background **3**

2.1 Neural networks . . . 3

2.1.1 Activation functions . . . 5

Sigmoid . . . 5

Rectified linear unit (ReLU) . . . 6

Heaviside step function . . . 6

2.2 Training with gradient descent . . . 7

2.2.1 Optimizers . . . 9

Adam . . . 9

2.2.2 Cold restarts . . . 9

2.3 Evolutionary algorithms . . . 10

2.3.1 Basic evolution . . . 10

Initialization . . . 10

Selection . . . 11

Mutation . . . 11

Crossover . . . 11

Rinse and repeat . . . 12

2.3.2 CMA-ES . . . 12

Covariance matrices . . . 12

Covariance matrix adaptation . . . 13

BIPOP-CMA-ES . . . 14

2.3.3 GOMEA . . . 14

Linkage . . . 14

Genepool Optimal Mixing . . . 15

Interleaved Multi-Start . . . 16

Anticipated Mean Shift . . . 16

2.4 The Exclusive-OR problem . . . 16

2.4.1 More than two inputs . . . 17

2.4.2 Formulated as a network . . . 17

2.5 Prior research . . . 18

3 Solving XOR with minimal neural network sizes **20**

3.1 Experiment setup . . . 20

3.1.1 Network initialization . . . 20

3.1.2 Hyperparameters . . . 21

3.1.3 Repeats, iterations and limits . . . 22

3.2 Experiment 1: ReLU activation at the hidden layer only . . . 22

3.2.1 Results . . . 23
3.2.2 Discussion . . . 23
3.3 Experiment 2: Sigmoid activation at the hidden and output layers . . . 24
3.3.1 Results . . . 25
3.3.2 Discussion . . . 25
3.4 Experiment 3: Sigmoid with wide initialization . . . 26
3.4.1 Results . . . 27
3.4.2 Discussion . . . 27
3.5 Experiment 4: Heaviside activation . . . 27
3.5.1 Results . . . 27
3.5.2 Discussion . . . 30
3.6 Experiment 5: ReLU with output activation and sigmoid-like without 30
3.6.1 Results . . . 30
3.6.2 Discussion . . . 31
3.7 General discussion . . . 32
3.8 Conclusion . . . 33

4 Applying problem-specific knowledge: asymmetry loss 34
4.1 Experiment setup . . . 35
4.2 ReLU activation at the hidden layer . . . 36
4.2.1 Discussion . . . 37
4.3 Sigmoid activation at the hidden and output layers . . . 37
4.3.1 Discussion . . . 37
4.4 Other networks . . . 38
4.4.1 Heaviside with and without output activation . . . 38
4.4.2 ReLU with output activation and sigmoid without . . . 39
4.5 General discussion . . . 40

5 Conclusion and future work 41
5.1 Discussion . . . 41
5.2 Future work . . . 41
5.3 Conclusion . . . 42

A Partial evaluations and linkage 43
A.1 Partial evaluations of neural networks . . . 43
A.1.1 Nonlinearity . . . 43
Storing intermediate results . . . 44
Training layer-by-layer . . . 45
A.1.2 Conclusion . . . 46
A.2 Problem-specific linkage models . . . 46
A.2.1 Linkage spanning two layers . . . 47
Linkage spanning more than two layers . . . 47

B Experiment platforms 49

C Linkage examples 50

D Implementation details 54
D.1 libGOMEA . . . 54
D.2 mininn . . . 56
D.3 NNProblem . . . 57

E Found GD and GDCR hyperparameters **58**

Bibliography **73**

List of Figures

2.1 A node in an NN. 3
2.2 A feed-forward network with four input nodes, two output nodes, and two hidden layers of five and three nodes, respectively. Activation functions and weights on the edges are not shown. 4
2.3 The sigmoid activation function. 5
2.4 The ReLU activation function. 6
2.5 The Heaviside activation function. 7
2.6 Three iterations of GD with varying learning rates applied to the example parabola. The orange "double-headed arrow" is from GD moving back and forth between two points. The other arrows are moving towards the minimum, with $\eta = \frac{1}{3}$ approaching it much faster than $\eta = \frac{1}{9}$. 8
2.7 The "lumpy staircase" with GD getting stuck at $x \approx 4.31$. 9
2.8 Various distributions and their covariances and covariance matrices. . 13
2.9 The values of $\mathrm{DT}(\mathbf{x})$ for $|\mathbf{x}| = 6$. 15
2.10 Schematic (IEEE) representation of an XOR gate. 17
2.11 A chain of 2-input XOR gates making a 4-input XOR gate. 17
2.12 A tree of 2-input XOR gates making a 4-input XOR gate. 17
2.13 A perfect solution to the 2-input, 2-hidden XOR problem. The top hidden node outputs 1 iff $x_1 = x_2 = 0$, while the bottom hidden node outputs 1 iff $x_1 = x_2 = 1$. The output nodes implement a logical OR and NOR, respectively. 18

3.2 The first five graphs show success rates for n-input ReLU-based networks without output activation. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 95 % success rate. 24
3.3 The first five graphs show success rates for n-input sigmoid-based networks with output activation. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 95 % success rate. 25
3.4 GOMEA final loss value distributions for five- and six-input networks. For five-input networks, as the hidden layer shrinks, losses tend to approach $\frac{1}{4}$. For six-input networks this happens across the board. . . 26
3.5 Each row of plots compares the success rates for five- and six-input sigmoid-based networks for a narrow versus wide initialization, for GD, GOMEA and BIPOP-CMA-ES, respectively. Previous (narrowly initialized) results for each algorithm are shown as broken lines, and results for narrow GD are included as a baseline result to be improved on. Columns are aligned; values along the x-axis are in the same position on each graph within a column. 28

3.6 The first five graphs show success rates for n-input Heaviside-based networks with output activation. GD and (for more than two inputs) GDCR appear in the bottom-right corner. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 50 % success rate. GD does not appear here as it never attained a 50 % success rate. 29
3.7 Minimum hidden layer sizes for various networks. The first row shows the minimum number of nodes in the hidden layer required for success rates of 50 % and 95 %, respectively, for ReLU-based networks with activation both at the hidden and output layers. The second row displays this information for sigmoid networks *without* activation at the output layer, and the third for Heaviside networks without output activation. In the last row GOMEA is close to, but not quite reaching the required threshold for smaller networks (those with 8 hidden nodes, for example), leading to large error bars. 31

4.1 Two distinct, but both perfect solutions to the 2-input, 2-hidden XOR problem. The top network is the same as shown in fig. 2.13, whereas in the bottom network the two hidden nodes and their associated weights have been exchanged; now the top node outputs 1 iff $x_1 = x_2 = 0$ and the bottom node outputs 1 iff $x_1 = x_2 = 1$. 35
4.2 The first five graphs show success rates for n-input ReLU-based networks without output activation. Previous results (those without asymmetry loss applied) are shown as broken lines. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 95 % success rate, also with previous results included. . . . 36
4.3 The first three graphs show success rates for n-input sigmoid-based networks with output activation. Previous results (those without asymmetry loss applied) are shown as broken lines. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 95 % success rate, also with previous results included. 37
4.4 Minimum hidden layer sizes for Heaviside problems. The first row shows results for Heaviside-based networks without output activation, with previous results (where no asymmetry loss was applied) as broken lines. GD and GDCR without asymmetry loss are included as a baseline result, but are (mostly) absent due to their inability to train networks with Heaviside activation as seen in fig. 3.6. BIPOP-CMA-ES is similarly absent from most of the graphs. The second row shows results for Heaviside networks without output activation in the same way. 38
4.5 Minimum hidden layer sizes for various problems. The first row shows results for sigmoid-based networks without output activation, with previous results (where no asymmetry loss was applied) as broken lines. GD and GDCR without asymmetry loss are included as a baseline result. The second row shows results for ReLU networks with output activation in the same way. 39

A.1 Two modes of evaluation of a network. 44
A.2 Training a network layer-by-layer. 46
A.3 A linkage set covering two layers, anchored around a bias. 47

C.1 A linkage set covering the first set of input weights on the first layer. . 50
C.2 A linkage set covering the second set of input weights and associated biases on the first layer. 51
C.3 A linkage set covering the third set of output weights on the first layer. 51
C.4 A linkage set covering the fourth set of output weights and the associated bias on the first layer. 52
C.5 A linkage set covering all weights of the second layer. 52
C.6 A linkage set covering all biases of the second layer. 53
C.7 A linkage set covering the entire third layer. 53

Chapter 1

Introduction

In the past decade, neural networks (NNs) have taken a center spot in the field of pattern recognition [1]. Vaguely resembling our understanding of biological neural networks such as brains, neural networks (NNs) can be "trained" by, among other methods, evaluating them against a previously labeled dataset and continuously correcting their behavior. Once trained, the network can be put to use, obtaining predictions for new data for which the label is not known. Their main applications range from image labeling, which not only cover automatically categorizing pet pictures but can also aid in diagnosing diseases such as skin cancer [2], to a near-human grasp on natural language with far-reaching consequences [3].

However, with such expressive abilities comes a great need for computational power. A language model that one can have a conversation with needs billions of parameters, each of which takes part in the complex calculations mapping the conversation to a new response and needs to be carefully trained for the network to function correctly. While advancements in computer engineering may soon allow us to train and apply these networks on small, portable, low-power devices such as smartphones and internet-of-things appliances, right now they need a relatively expensive hardware, in terms of both money and energy. Furthermore, even if we had such an amount of computational power, it would be wasteful to keep running these enormous models if smaller, and therefore more efficient, networks were available to us.

The currently most popular method for training NNs, namely gradient descents (GDs), often requires networks to be over-parameterized. An over-parameterized NN contains more parameters than theoretically should be necessary to solve the problem. In 2018, Frankle and Carbin posited the "lottery ticket hypothesis", namely that GD relies on "winning tickets": a set of parameters embedded in the randomly initialized network necessary for the network to be trained successfully [4]. Under their hypothesis, over-parameterization simply gives the initialization stage a greater chance of generating a winning ticket.

Much of the research by Frankle and Carbin focused on the Exclusive-OR (XOR) problem, which is a binary logic problem that is simple to formulate, trivial to verify, and theoretically requires very small networks to solve. However, GD was found unable to train networks this small. This thesis investigates whether the training of these NNs can be improved upon in terms of XOR network size by an entirely different class of optimization algorithms: evolutionary algorithms (EAs).

Outside NN research, both EAs and GD are widely applied techniques for optimization. GD iteratively takes small steps following the gradient of a function to a minimum; when applied to NNs, the network and its parameters make up the function to optimize. The vast majority of NNs are trained using GD or a variation thereof, such as the very popular Adam optimizer [5], and therefore "training a neural network" often implies the use of GD.

EAs, on the other hand, locate a function's minimum through maintaining a population of solutions that are evaluated, ranked, mutated and recombined in order to synthesize new solutions. There are many different forms of EAs, of which the genetic algorithms (GAs) and evolutionary strategies (ES) are of particular interest in this work, namely GOMEA and BIPOP-CMA-ES. Applications of EAs are (compared to GD) similarly wide, with examples including antenna development [6, 7], architectural layouts [8] and medical treatments.

Theoretically, there are distinct advantages to using such an evolutionary approach: most notably, population-based algorithms have a more complete picture of the solution space and can therefore more easily avoid local minima; do not require gradients, thereby removing the need for differentiable activation functions (the basic non-linearity implemented in an NN's neurons); and allow for the easy and efficient implementation of multi-objective problems.

It is important to remark that EAs have rarely been applied to train neural networks due to the great computational and memory costs of maintaining such a population [9]. In this thesis, we do not consider this a problem as we are investigating the optimization potential of EAs when applied to small NNs. Often, EAs are instead used to find a network structure or architecture, sometimes in addition to determining network parameters. Such an approach usually precludes using hand-made architectures that encode key traits of the problem to solve. This thesis places the focus on only the training procedure, in order to gauge the performance of EAs when training the NNs normally trained using GD.

Research objectives

The work performed here aims to answer the question: are EAs capable of successfully solving networks that are smaller relative to those required by GD? As part of this, it answers the following sub-questions:

1. Can EAs consistently train smaller NNs for the XOR problem compared to GD?
2. How do different activation functions impact the algorithms' performance?
3. Can search be improved through the application of problem-specific knowledge?

Outline

To that end, this thesis first discusses the necessary background information and then investigates the performance when training various small XOR networks with several evolutionary algorithms versus gradient descent, followed by a look into applying problem-specific knowledge to improve the training of NNs. This is finalized by a discussion of the findings and recommendations for future research.

Chapter 2

Background

This chapter covers the background information necessary to have a full picture of the work performed here. First, an overview of NNs, focusing on the construction of simple feed-forward networks and, in particular, the various activation functions that are available. After that, the application of GD to train these networks is discussed, including a short review of some optimizers at our disposal. Next, an introduction of EAs describes their fundamental ideas and delves deeper into two algorithms that improve on these, namely GOMEA and CMA-ES. Following that, a formulation of the XOR problem is given, including a description of the NN architecture that is used in the subsequent chapters. Last, a brief overview of prior research on the topic of training NNs through evolution.

2.1 Neural networks

NNs have been a staple of artificial intelligence research for more than half a century. Their emulation of biological NNs, like those in our own brains, allows for modeling extremely powerful and expressive functions from only a few simple primitives.

Since research on NNs started, a wide variety of layer types and networks have been devised. This thesis puts the focus on networks of one of the oldest and simplest kinds: multilayer perceptrons (MLPs). The central element in an MLP is the *node* and is commonly said to be analogous to a neuron. Each node, as part of a directional graph, has a set of incoming and outgoing weighted edges, a bias, and a nonlinear activation function. A diagrammatic representation of a node with three inputs, four outputs, a bias b and a sigmoid-shape activation function is shown in fig. 2.1.

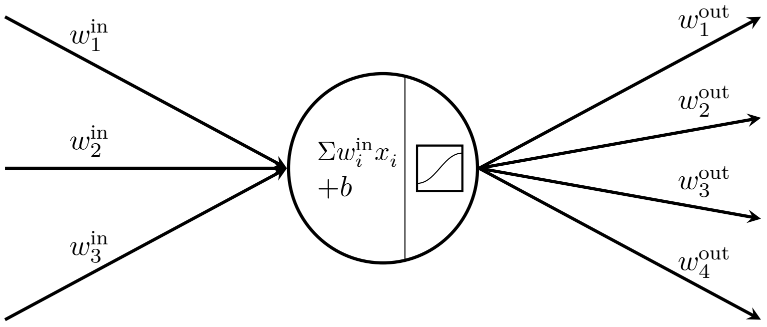


FIGURE 2.1: A node in an NN.

Each node takes the incoming values multiplied by their respective weights, calculates their sum and adds the node's bias. Then, an activation function is applied and the result is distributed to the next nodes in the graph or taken as an output value.

Nodes are arranged in groups called *layers*, and the sequence of each layer feeding the next is what makes up the feedforward network. Layers are named: a feedforward network has one input layer, where the nodes receive the inputs to the network, any number of hidden layers, and an output layer, where the outputs of the node are the outputs of the network:

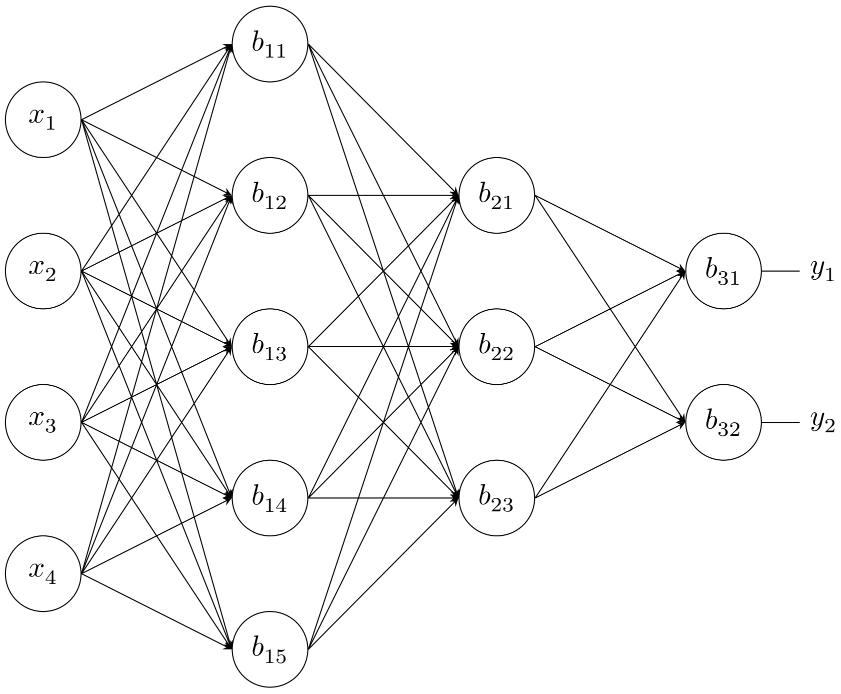


FIGURE 2.2: A feed-forward network with four input nodes, two output nodes, and two hidden layers of five and three nodes, respectively. Activation functions and weights on the edges are not shown.

Every node in each layer depends on all values generated by the previous layer. There are no cycles in the network and there are no connections within a layer; each node has a dependency on only the nodes in preceding layers.

Each set of nodes within a layer and their associated incoming weights, biases and activation function form a transformation of an input vector. Given a layer with n inputs and m nodes, the weights of the incoming connections can be expressed as an $m \times n$ matrix W and the bias as a m-element vector $\mathbf{b}$:

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \tag{2.1}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{2.2}$$

Using this, the sum plus the bias at each of the nodes for an input $\mathbf{x}_i$ can be calculated using a simple matrix product and vector sum, and finally activated using an activation function f_a:

$$\mathbf{x}_{i+1} = f_a\left(W\mathbf{x}_i + \mathbf{b}\right) \tag{2.3}$$

Here, it is important to note that $Wx_i + \mathbf{b}$ is a linear transformation, being a series of multiplications and additions in vector space. The *non*linear activation function is necessary to achieve any nonlinear behavior by the network, as otherwise the entire feed-forward network would be no more than a highly parameterized linear map.

2.1.1 Activation functions

Multiple different kinds of activation functions are used in NNs. Here we discuss some of the most relevant functions.

Sigmoid

The logistic function, or colloquially *sigmoid*, after its characteristic S-like-shape, maps values from $\mathbb{R}$ to $(0, 1)$. The full definition of the sigmoid function is:

$$S^*(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{2.4}$$

The most commonly used "standard" variant takes $L = 1, k = 1, x_0 = 0$ and simplifies to:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

This produces a "swish"-style curve symmetric at $x = 0$, where $S(0) = \frac{1}{2}$, and quickly but asymptotically approaches 0 and 1 at for negative and positive x, respectively. As an example of how quick the asymptotes are reached, it is already at $x = 6$ that $S(6) \approx \frac{998}{1000}$.

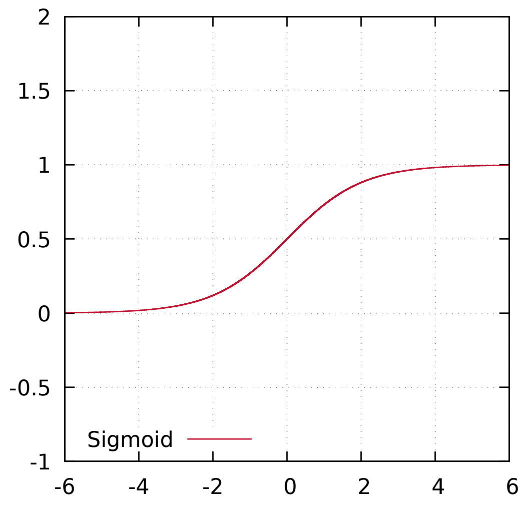


FIGURE 2.3: The sigmoid activation function.

This activation function was a common choice early on in NN research because of these properties [10]: no matter the input values and their weights, the output was guaranteed to be within a reasonable range, while remaining injective. The quick approach to either 0 or 1 has a major drawback, however: the gradient reaches zero just as quickly. From the derivative it is clear that the gradient diminishes very fast:

$$\frac{dS}{dx} = S(x)\,(1 - S(x)) \tag{2.6}$$

At $x = 0$, the derivative is $\frac{1}{4}$, while at $x = 6$ the derivative has already reached approximately $\frac{2}{1000}$. This is known as the vanishing gradients problem [11], where the gradient diminishes so far that GD is unable to perform meaningful updates.

Rectified linear unit (ReLU)

The ReLU activation function rectifies its input by clamping negative values to 0, while passing nonnegative values as-is:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.7}$$

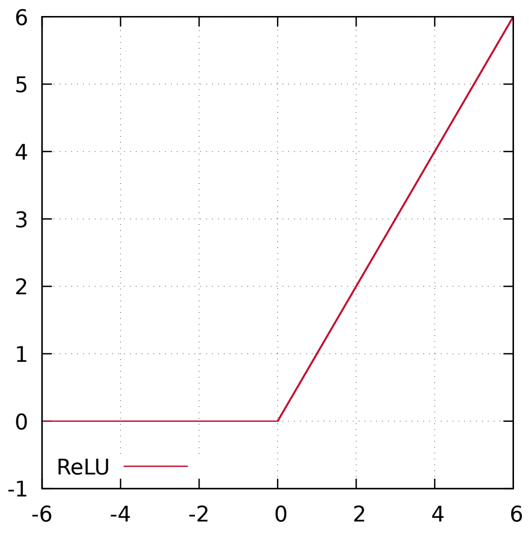


FIGURE 2.4: The ReLU activation function.

This function is a popular activation function due to its computational speed, as only a comparison and, sometimes, a value write are necessary [12]. Its derivative of 1 for nonnegative values of x also allows perfect propagation of gradients (as explained in section 2.2), though the gradient of 0 at other values can complicate the training process for GD, similar to the problems seen with the sigmoid function.

Heaviside step function

The Heaviside step function outputs 1 for positive inputs and 0 for negative inputs; the behavior at zero is often application-defined. The piece-wise formulation for the definition used here is as follows:

$$H(x) = \begin{cases} 1 & \text{if } x > 0 \\ \frac{1}{2} & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.8}$$

The step function can be attained through “squeezing” the generalized sigmoid function S^* in the x-direction by taking the limit of k to infinity:

$$H(x) = \lim_{k \to \infty} \frac{1}{1 + e^{-kx}} \tag{2.9}$$

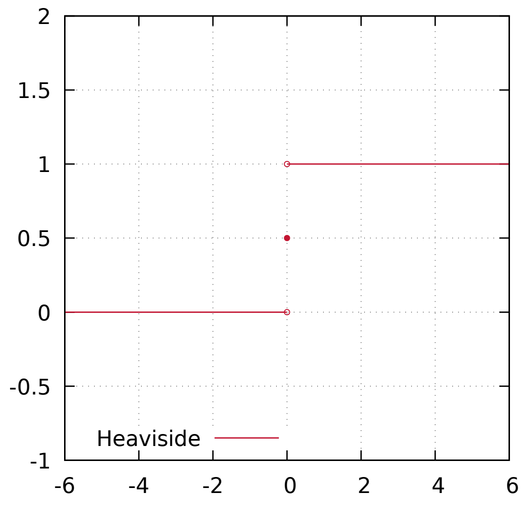


FIGURE 2.5: The Heaviside activation function.

From this definition, taking $x = 0$ makes it easy to see the reasoning behind deciding on $H(0) = \frac{1}{2}$. In practice, however, this choice will not matter much because of the infinitesimal chance of getting a value of exactly 0 at a node before activation.

A form of this activation function was used as part of the perceptron, a very early single-layer NN also used for classification [13].

For GD this function is an exceptionally poor choice. Considering how sigmoid poses a problem due to the very small gradients at its extents, the Heaviside function will erase the gradient *everywhere* and prevent its propagation along the network. Effectively, GD will be unable to touch any parameter occurring before this function. However, it is reasonable to expect that an EA will be more resilient to this problem, which makes this function interesting for experimentation.

ll

2.2 Training with gradient descent

GD is an iterative optimization method for finding the minimum of a function. It works by continuously examining the function's derivative at the current point, which forms the gradient, and taking small steps in the opposite direction of this gradient. The process stops when a minimum of the desired quality is found, the iteration limit is reached, or any other desired stopping criterion is met.

GD works for any number of parameters. The generalized update step for parameters $\mathbf{x}$, differentiable multivariate function f and learning rate η is:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \nabla f(\mathbf{x}_n) \tag{2.10}$$

As an example, one may want to find the minimum of the following parabola using GD:

$$f(x) = x^2 + x$$

$$f'(x) = \frac{df}{dx} = 2x + 1$$

Starting at $x_0 = 1$, we find $f(x_0) = 2$ and $f'(x_0) = 3$, strongly pointing towards a minimum in the negative x-direction. Taking a step at the full magnitude of the gradient might be too far (certainly in this case), so we pick a learning rate $\eta = \frac{1}{3}$, with which we calculate the next point: $x_1 = x_0 - \eta f'(x_0) = 0$. We repeat the process:

$$f(x_1) = 0$$

$$f'(x_1) = 1$$

$$x_2 = x_1 - \eta f'(x_1) = -\frac{1}{3}$$

Once more:

$$f(x_2) = -\frac{2}{9}$$

$$f'(x_2) = \frac{1}{3}$$

$$x_3 = x_2 - \eta f'(x_2) = -\frac{4}{9}$$

Arbitrarily, we decide to stop here. Our solution, x_3, is fairly close to the true minimum at $x = -\frac{1}{2}$. Continuing this process would have encroached upon this minimum even further.

The choice of η is important, as illustrated by fig. 2.6. If it is too high, search will keep jumping around the minimum. In the example, choosing $\eta = 1$ caused the algorithm to jump side-to-side in the parabola without end. In less extreme cases, where η is only *slightly* too high, the search ends up moving from side to side, like a marble circling a funnel. Meanwhile, setting η too low makes the search process unnecessarily slow and more prone to getting stuck in a local minimum.

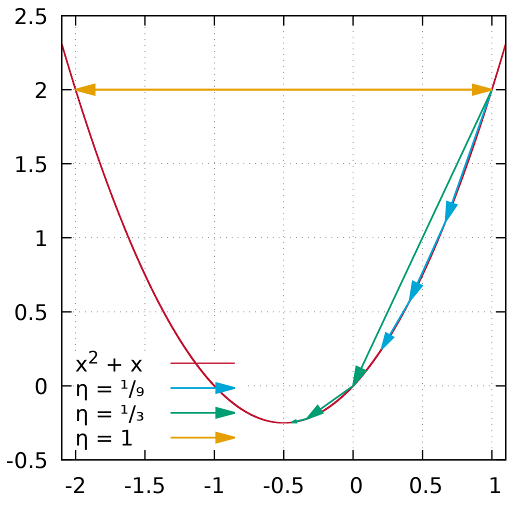


FIGURE 2.6: Three iterations of GD with varying learning rates applied to the example parabola. The orange "double-headed arrow" is from GD moving back and forth between two points. The other arrows are moving towards the minimum, with $\eta = \frac{1}{3}$ approaching it much faster than $\eta = \frac{1}{9}$.

2.2.1 Optimizers

Local minima are a problem for GD in general: imagine a “lumpy staircase” function like $g(x) = \left|\sin^3(x) + x\right|$, shown in fig. 2.7, which has a global minimum at $x = 0$ and for positive x generally trends upward, but also contains local minima. The gradient periodically moves between approximately -0.15 and 2.15. GD could happen to hit a negative gradient and get stuck in a local minimum.

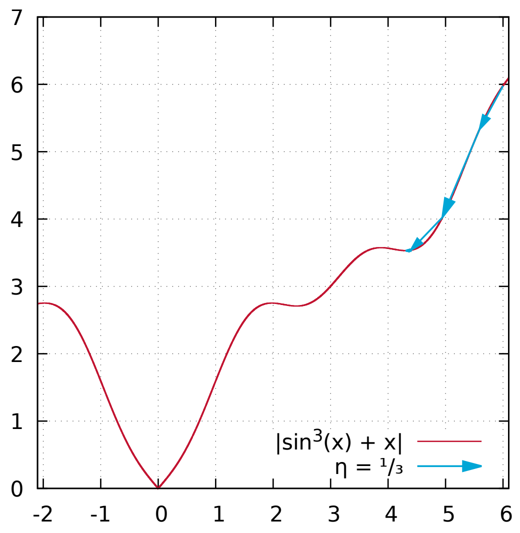


FIGURE 2.7: The “lumpy staircase” with GD getting stuck at $x \approx 4.31$.

Adam

Multiple improvements upon basic GD have been developed, often to make it more resilient to local minima and accelerate convergence. For example, Adam [5] is the combination of two improvements over classic GD, namely RMSProp [14] and Momentum [15], and has seen overwhelming popularity since its inception. It redefines the update step as such:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{\eta}{\sqrt{\hat{\mathbf{g}}_{n+1}} + \varepsilon}\hat{\mathbf{v}}_{n+1}$$

Adam also performs bias correction on the moving averages to ensure a smooth start of the training process: $\hat{\mathbf{v}}_n$ is the bias-corrected moving average first-order gradient and $\hat{\mathbf{g}}_n$ is similar but for the second-order gradient. The rates at which the averages decay are controlled by the hyperparameters β_1 and β_2, respectively. The higher their values, the stronger the averaging is. The values are recommended to be $\beta_1 = 0.9$ and $\beta_2 = 0.999$, but can be set freely between 0 and 1, as the problem requires. The ε term prevents division by zero and is usually set to a very low value such as 10^-8.

2.2.2 Cold restarts

Another way to overcome local minima is by restarting from a different point. Such a *cold restart* can be triggered if the solution does not improve (satisfactorily) after a certain number of iterations or if the gradient falls below a certain threshold and the target value has not been reached.

In this work, gradient descent with cold restarts (GDCR) is considered, and implemented as an Adam optimizer with an additional parameter r that specifies the minimum L^2 norm of the gradient reshaped as a vector. Once this norm falls below r, the problem is re-initialized, without using any information that can be gained from the current solution, hence the term "cold".

Choosing a value of r can be difficult, because it is highly dependent on the network architecture: activation functions attenuate gradients differently and larger or deeper networks tend to have lower gradient values, as each weight or bias has a lower contribution to the output value, and so its contribution to the error will lower as well. An r that is too high causes restarts almost immediately, turning GDCR into random search within the initialization range. Too low and restarts are never triggered, making it equivalent to GD. Therefore, r should be tweaked per network. Here, $r = 10^{-8}$ was chosen as a fairly safe baseline because this was found to be a good setting from preliminary experiments.

2.3 Evolutionary algorithms

GD is just one kind of optimization algorithm; there are many others. One is the class of EAs. Inspired by biological evolution, EAs work by taking a population of solutions and evolving it according to the fitness of each individual in the population.

Individuals are represented by their *genome*, which encodes a solution to the problem to be solved. For the algorithms discussed here, genomes are a series of real-valued numbers. The genome can be evaluated into a fitness value that gauges the quality of the solution. Like GD, search stops when a solution with a suitable fitness is found or some other constraint is reached, such as a maximum number of generations.

2.3.1 Basic evolution

A simple EA works as follows. For illustration purposes, we will use a basic objective function $f(x, y, z) = x^2 + y^2 + z^2$. It is easy to see that the minimum lies at $f(0, 0, 0) = 0$ and that there are no other minima—as such it should also be fairly easy for our EA to find a good solution.

Initialization

A population X of n genomes with $|\mathbf{x}|$ parameters are sampled from an initial distribution. What kind (uniform, normal, and so on) and the range of this initial distribution depends on the problem to solve.

For our problem, we choose a fairly default distribution $U(\pm 1)$ (uniform between -1 and 1). Our initial population of four genomes looks as follows:

#	**x**	**y**	**z**	$f(x, y, z)$
1	-0.5	0.6	-0.6	0.97
2	0.2	-0.4	0.8	0.84
3	-0.1	0.1	0.6	0.38
4	-0.2	1.0	-0.1	1.05

Selection

Each generation, the genomes are ranked using the fitness function that translates each genome into a single, real-valued objective to be minimized. Then, k genomes are selected. This can simply be the k top-performing genomes (named *trunctation selection*), but more complex approaches also exist, including weighted choice or *roulette wheel selection* (where a more fit genome is more likely, but not guaranteed to be picked), tournaments (where sub-populations are chosen at random and the best-performing "wins"), and combinations thereof.

For the example problem, we choose top-2 selection. #2 and #3 have the lowest objective value so these will be retained:

#	**x**	**y**	**z**	$f(x,y,z)$
2	0.2	-0.4	0.8	0.84
3	-0.1	0.1	0.6	0.38

Mutation

New genomes are generated by taking a selected genome and modifying one or more of its constituent values. A very simple mutation method could have, for each value, a (small) probability p of resampling it from the initialization distribution.

In our case, we create offspring for each of the selected genomes and sample one randomly-chosen variable from $U(\pm 1)$ again:

#	**x**	**y**	**z**	$f(x,y,z)$
1	0.2	-0.4	**0.1**	**0.21**
2	0.2	-0.4	0.8	0.84
3	-0.1	0.1	0.6	0.38
4	-0.1	0.1	**0.7**	**0.51**

Genome #1 is an offspring of #2 and genome #4 is an offspring of #3. Mutated variables are indicated in boldface. The first genome improved significantly on its parent. The fourth, on the other hand, had a detrimental mutation that harmed its fitness. That is an unfortunate but hard-to-avoid event in an EA. However, even if both mutations were harmful to the performance of the genome, keeping the selected solutions ensures that we do not regress in overall performance. There will be another attempt in the next generation.

Crossover

Using just mutation is very close to random search from multiple starting points. While this approach is likely to eventually generate good solutions, it would be beneficial if information could be exchanged horizontally—that is, between potentially unrelated genomes.

Crossover is such an operator between genomes. Two genomes in the selection are chosen and parts of them are exchanged. What this achieves is effectively the transformation of the genome relative to the other's in the solution space.

For the example problem, we use "one-point crossover" between the new offspring. One-point crossover works by selecting a random point within the genome, and exchanging either the portions left or right between the genomes. As the point we (randomly) choose the point "between" x and y: both genomes retain their value for x while they exchange y and z:

#	x	y	z	$f(x, y, z)$
1	0.2	**0.1**	**0.7**	**0.54**
2	0.2	-0.4	0.8	0.84
3	-0.1	0.1	0.6	0.38
4	-0.1	**-0.4**	**0.1**	**0.18**

As with mutation, random crossover can be beneficial or detrimental. Now, #4 is the best solution found yet, while #1 has regressed significantly. Note that if only z had been exchanged, an even better solution would have been found, but alas; such is the nature of a simple algorithm like this.

Rinse and repeat

The offspring and their parents are then regarded as the new population. Often, a portion of the previous generation is brought over to the new one as-is. If this portion consists of the best of the previous generation (as it usually is), this is called *elitism*. While this may make the algorithm miss out on opportunities to find even better solutions, it stabilizes the search and allows for faster convergence: if there are multiple minima, elitism will increase the similarity of the solutions and crossovers will be less likely to be detrimental, at a cost of a fuller picture of the solution space.

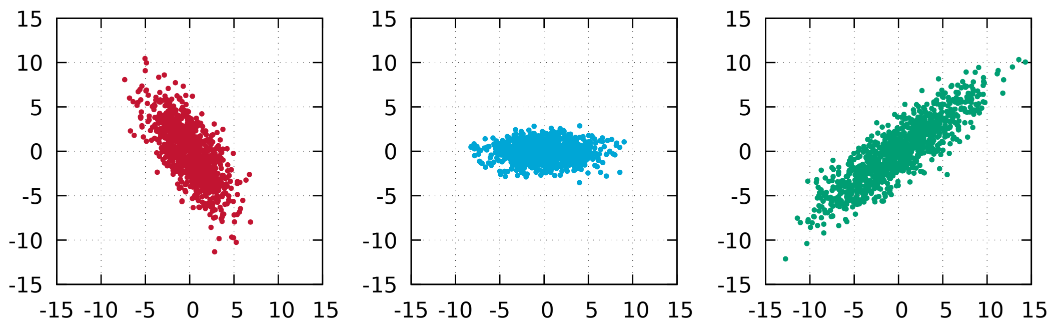
It should be easy to see that, given enough time, this algorithm will eventually find an optimal solution to our example problem. Our simple EA will have more trouble on more complex solution spaces and we have already seen the algorithm make "questionable" modifications to our solution genomes. To alleviate these problems and guide the search to better choices, other algorithms have been developed.

2.3.2 CMA-ES

The covariance matrix adaptation evolutionary strategy (CMA-ES) is a particular instance of an EA belonging to the class of ES. ES algorithms sample new values from normal distributions that are estimated from the previous iteration: not only do the solutions evolve, so does the search itself. Usually the additional parameters are the individual variance for the problem parameters followed by the covariance matrix.

Covariance matrices

A covariance matrix for n variables is an $n \times n$ mapping that, for each pair of variables x_i, x_k, produces the covariance $\text{cov}(x_i, x_k)$. If this covariance is positive ($\text{cov}(x_i, x_k) > 0$), then if x_i increases, then it is likely that x_k also increases, proportional to the covariance. On the other hand, if $\text{cov}(x_i, x_k) < 0$, then an increase in x_i means a likely decrease in x_k. If the covariance is zero, then the variables are unrelated. Figure 2.8 shows example distributions of negative, neutral and positive covariances, respectively.



(A) $\mathrm{cov}(X, Y) \approx -1$
$C \approx \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}$

(B) $\mathrm{cov}(X, Y) \approx 0$
$C \approx \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$

(C) $\mathrm{cov}(X, Y) \approx 2$
$C \approx \begin{bmatrix} 4 & 2 \\ 2 & 3 \end{bmatrix}$

FIGURE 2.8: Various distributions and their covariances and covariance matrices.

By sampling new variables using a covariance matrix, related variables are more likely to jointly "move" into the desired direction. A covariance matrix can be easily estimated from a set of measurements:

$$\begin{aligned} \bar{x} &= \frac{1}{n}\sum_{i=1}^{n} x_i \\ C &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T \end{aligned} \tag{2.11}$$

Covariance matrix adaptation

CMA-ES estimates a covariance matrix each generation. The estimate is formed by finding the matrix that leads to the maximum likelihood of generating the current population *relative to* the previous ones from earlier generations, akin to the use of momentum in gradient descent. This is in contrast to estimating it from just the current generation, which lacks any directional information of the search.

Selection is usually performed by truncation, after which the solutions contribute to the mean in a weighted fashion, where the more successful solutions contribute more estimated compared to worse-performing ones. Therefore, the covariance matrix estimation step of CMA-ES for a selection of λ individuals at generation $g + 1$ is:

$$\begin{aligned} \bar{x}_g &= \sum_{i=1}^{\lambda} \frac{f(x_i)}{\sum\limits_{k=1}^{\lambda} f(x_k)} x_i \\ C_{g+1} &= \frac{1}{n-1}\sum_{i=1}^{n}(x_{g+1,i} - \bar{x}_g)(x_{g+1,i} - \bar{x}_g)^T \end{aligned} \tag{2.12}$$

Note that a major difference between equations 2.11 and 2.12 is that the variant used by CMA-ES estimates the covariance of values *relative to* the mean of the previous generation, as opposed to the mean of the current generation.

CMA-ES is often seen as a baseline EA, owing to its low number of hyperparameters that need tuning (making application easy) and proven good performance in previous problems, both theoretical and practical.

BIPOP-CMA-ES

For many of the hyperparameters, the default values work fine for many problems. Often, only two need to be tweaked per problem: the initialization range and the population size. Of these two, the population size is more difficult to find.

The bi-population CMA-ES (BIPOP-CMA-ES) variant performs both local and global search by maintaining two populations: one small that is analogous to local search and close to the original CMA-ES, and one large for more global search. The algorithm switches between the two populations depending on which one has a lower total number of evaluations.

A restart occurs if the algorithm finished running with one of the two populations following the same termination conditions as used for "vanilla" CMA-ES.

2.3.3 GOMEA

The gene-pool optimal mixing evolutionary algorithm (GOMEA) [16, 17] is similar to CMA-ES in that it learns a multivariate distribution from the population, but it further includes parameter linkage, a multi-start scheme (IMS), mean shift anticipation (AMS) and other improvements.

Linkage

GOMEA can exploit explicit linkage within a problem's parameters. Linkage is a dependency between two or more parameters, such that it is likely beneficial for the algorithm to cross over the parameters at the same time. Doing so ensures that no essential combination within this group is broken up.

GOMEA represents linkage as a set of sets of locations in the genome: a family of subsets (FOS) $\mathcal{F}$ of the problem's parameters S. It is important that the union of all subsets fully covers the parameters: $\bigcup_{s \in \mathcal{F}} s = S$ must hold. Otherwise, the parameters not covered will not be considered by the algorithm.

An extreme example of a problem where linkage information is effective is the discrete concatenated deceptive trap problem. m groups of k binary values are randomly scattered in a vector $\mathbf{x}$ of length $\ell = mk$. A bijective function $g(v = ik + n)$ returns the position of the nth value in the ith group in vector $\mathbf{x}$.

The fitness of a group is maximal if all bits are 1. However, it is *minimal* if all but one are 1 and increases the more bits are 0:

$$f_{\text{trap}}(x_1, \dots, x_\ell) = \sum_{i=0}^{m} \text{DT}(x_{g(ik)}, \dots, x_{g(ik+k-1)}) \tag{2.13}$$

$$\text{DT}(\mathbf{x}) = \begin{cases} k & \text{if } \sum \mathbf{x} = k \\ k - 1 - \sum \mathbf{x} & \text{if } \sum \mathbf{x} \neq k \end{cases} \tag{2.14}$$

In this problem, variables that are part of the same group are strongly linked: their contributed fitness is maximal or near maximal if all values are the same. At the same time, there is no dependency between variables in different groups.

If an algorithm performs one-point crossover, some related values are changed while others remain fixed. If this happens between two genomes where one group is

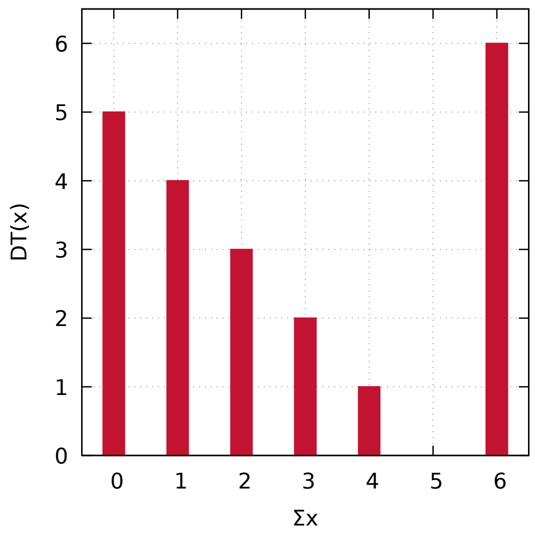


FIGURE 2.9: The values of DT($\mathbf{x}$) for $|\mathbf{x}| = 6$.

all zeroes (near optimal) and another where that same group is all ones (optimal), this will generate a new genome for which that group is mixed ones and zeroes and therefore further from the optimum than either of its parents.

An algorithm informed by linkage can decide to exchange the two groups all at once, preserving (near-)optimality for the groups in the offspring genomes.

Of course, real-world problems are unlikely to have such a degree of interdependence between some variables and *in*dependence between others. Still, dependencies do exist (as otherwise covariance-based strategies would not work) and more complex linkage models are possible, such as linkage trees, forests and graphs.

If no linkage information is available or desired, a univariate model can be constructed by placing each variable in its own group, and a fully covariant model can be constructed by placing all variables in the same one. Both, however, can become slow as the former has to perform n evaluations for n parameters, and the latter needs $O(n^3)$ time to estimate a covariance matrix.

Genepool Optimal Mixing

Instead of taking two individuals and performing crossover over some part of the genome, GOMEA modifies an existing solution by resampling linked variables as indicated by the FOS and only retaining the modification if it leads to an improvement. Sometimes, by default 5 % of all cases, a worse solution is kept anyway. This perturbs the population and may help with escaping local minima.

This resampling is done for every set in the linkage model, in random order. Re-evaluating the entire solution even if only a small number of variables changed may be very costly. If the problem supports it, GOMEA can use the previous result and modify it according to the new values without evaluating the entire solution again; in other words perform a *partial evaluation.* For example, taking the example problem $f(x, y, z) = x^2 + y^2 + z^2$, if a mixing step modifies only z, we can calculate the new value from a previous result v without needing x or y and involving fewer operations:

$$f_{\text{partial}}(z_{\text{old}}, z_{\text{new}}) = v - z_{\text{old}}^2 + z_{\text{new}}^2 \tag{2.15}$$

This does require that there is a linear or otherwise reversible correspondence between the modified variable and the objective value. For variables where this is not the case, a full evaluation must be performed anyway.

Interleaved Multi-Start

The interleaved multi-start (IMS) scheme [18], inspired by the parameter-less GA by Harik and Lobo [19], removes the need to find an optimal population size by starting multiple populations of different sizes. It is similar to the BIPOP scheme for CMA-ES, but it can support more than two populations. The reasoning behind it is the same however.

An initial population size is selected, along with a maximum number of populations. Each next population is double the size of the previous one. Larger populations start only after a few generations of the newest population have run, usually eight. This way, if only a small population is needed, it is likely to finish before or shortly after the last population is started. On the other hand, if a small population converges but does not reach the desired objective value, it can be terminated an computational resources can be focused on the larger populations.

By default, the smallest population contains 10 genomes and the population is grown at most 8 times, for a largest population of 1280.

Anticipated Mean Shift

Anticipated mean shift (AMS) can accelerate search by moving part of the population in the direction it moved in since the previous generation. It is similar to CMA-ES's prediction of search steps, but AMS is applied through the addition of the shift to only a portion of the new samples, whereas CMA-ES models the shift into the covariance matrix and thereby always incorporates it into the search.

Not only does AMS accelerate search, it can also overcome local minima such as in the case of the lumpy staircase. In that way it is similar to momentum, because some genomes keep moving in the direction they moved in previously.

2.4 The Exclusive-OR problem

The XOR is formulated for two inputs as follows: given two inputs $x_1, x_2 \in \{0, 1\}$ (i.e., binary digits), output $y \in \{0, 1\}$:

$$y = 1 \iff (x_1 = 0 \land x_2 = 1) \lor (x_1 = 1 \land x_2 = 0) \tag{2.16}$$

In essence, output 1 iff (*if and only if*) exactly one of the two inputs is 1. Otherwise, output 0. An alternative formulation is:

$$y = 1 \iff \Sigma_i x_i = 1 \pmod 2 \tag{2.17}$$

Which states that the output is 1 iff the binary sum of the inputs is odd.

The inverse of this problem is called the Excusive-NOR or XNOR for short. A solution can be constructed by inverting the output y of an XOR solution as $\bar{y}$. An instance of a solution in real life can be seen in a multiway switching arrangement where two light switches, usually on different floors, control the same light bulb: the bulb is lit iff both switches engage the energized wire; in other words, the light is on if and only if the switches are both up or both down.

XOR gates are common in electronics. Figure 2.10 shows its schematic representation.

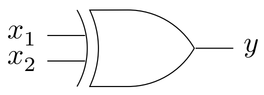


FIGURE 2.10: Schematic (IEEE) representation of an XOR gate.

2.4.1 More than two inputs

The two different formulations for two inputs pose a problem for a larger number of inputs, as both can be expanded to an arbitrary number of binary inputs. This work uses the latter definition, also known as the Parity or "oddness" problem. This formulation is much harder to solve than the former (1-Hot), as Parity's output exhibits extreme instability due to a change in only one input flipping the output: for n inputs, the 1-Hot definition only has n states, out of 2^n, where the output is 1, whereas Parity equally divides the input states into two 2^{n-1} in size each.

Larger XOR gates can be constructed from smaller XOR gates, as shown in figures 2.11 and 2.12.

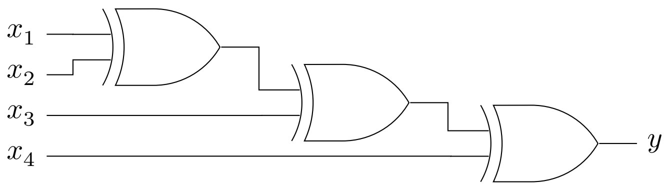


FIGURE 2.11: A chain of 2-input XOR gates making a 4-input XOR gate.

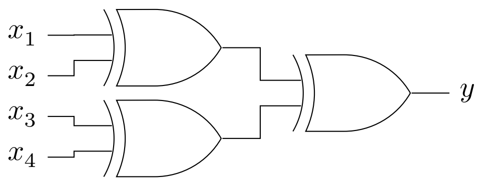


FIGURE 2.12: A tree of 2-input XOR gates making a 4-input XOR gate.

2.4.2 Formulated as a network

Due to the above properties, the XOR problem is an interesting benchmark for gauging the performance of NN optimizers. A very small NN with one hidden layer and only one position with a ReLU activation function is easy to construct:

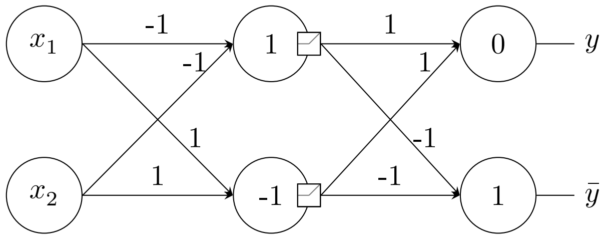


FIGURE 2.13: A perfect solution to the 2-input, 2-hidden XOR problem. The top hidden node outputs 1 iff $x_1 = x_2 = 0$, while the bottom hidden node outputs 1 iff $x_1 = x_2 = 1$. The output nodes implement a logical OR and NOR, respectively.

The first hidden node outputs 1 iff both inputs are 0. It does so by having a bias of 1 and input weights of -1: equivalent to the expression $(1 - x_0 - x_1) > 0$ or the logical NOR gate. The second hidden node produces 1 iff both inputs are 1: $(-1 + x_0 + x_1) > 0$, the AND gate. The output node $\bar{y}$ computes the logical OR. Because the outputs of both hidden nodes are mutually exclusive, a simple sum $h_0 + h_1$ suffices; no bias or nonlinearity are necessary. Finally, output y is the logical negation and therefore computes $1 - h_0 - h_1$ (equiv. to h_0 NOR h_1), again without activation.

It is possible to introduce any number of additional nodes here. Simply set their outgoing weights to 0 and any influence on the output will be zero as well. Alternatively, the level of contribution to the output can be divided between nodes such that the final sum will always be either 0 or 1, according to the inputs.

Scaling up the problem to n inputs, $n > 2$, is more difficult, however. The naïve approach, a chain of XOR gates like in fig. 2.11, requires $2n - 2$ hidden layers: each gate can be replaced by a 2-input network. A tree of nodes as in fig. 2.12 requires $2 \log n - 2$ hidden layers. Parallel gates can be isolated by setting connecting weights to zero. However, even a network with just one hidden layer is expressive enough to encode an XOR gate with any number of inputs, though it is significantly more difficult to train [20]. This style of network will be used in the rest of the thesis.

Furthermore, other activation functions other than ReLU also work. The outputs of the hidden layers can remain the same, though weights and biases will need to change to adapt to the new transformation.

2.5 Prior research

While limited, there has been research into this approach to neuro-evolution. Already in 1989 Montana and Davis designed a functional GA for evolving small MLPs [21]. Their algorithm improved upon GD when training a network classifying sonar recordings. The next year, Fogel et al. designed a similar algorithm but applied it more generally, including to the XOR problem [22]. They also named GD with both cold and warm restarts as possible alternative approaches.

That same year, Whitley et al. published a review of evolutionary approaches to both training and topology generation [9]. Their algorithm, GENITOR, was different from other algorithms by combining a binary encoding of 8-bit weights and biases in $[-127, 127]$ and two-point crossover. Interestingly, it performed comparably to the algorithm used by Fogel et al.

Whitley et al. concluded by anticipating neuro-evolution that generate suitable topologies instead of or in addition to finding parameters. This was a very accurate prediction, as subsequent research has mostly focused on this instead of just training, such as NEAT (NeuroEvolution of Augmenting Topologies) [23–25]. Knowledge from methods such as these often does not transfer well to fixed-topology training, because the networks quickly evolve nonstandard connections (such as those skipping one or more layers [26]) and poor parameter training can be worked around by modifying the topology. This last aspect is similar to the lottery ticket hypothesis for GD.

Still, even as recently as 2016 research on training-only neuro-evolution continued, with Morse finding competitive performance with LEEA [27]. This algorithm performs evaluations on limited numbers of samples as opposed to the entire dataset (known as mini-batching) and implements a fitness inheritance scheme to help the algorithm select individuals that were successful on previous batches.

Other attempts employ additional techniques to reduce the size of the genome, usually through a compression scheme [28] or embryogeny, where the genome describes how to construct the solution instead of directly encoding it [29].

Chapter 3

Solving XOR with minimal neural network sizes

In order to determine which algorithm is the best at finding minimal networks, we choose a network architecture and gauge how well the algorithms are at training instances of this architecture. Finding how small a network each algorithm is capable of training is interesting not only because smaller networks are more efficient, but also in the context of the lottery ticket hypothesis: it is reasonable to assume that EAs are not subject to the same limitations as GD, allowing them to find smaller solutions.

Multiple architectures are examined to get a better picture of each algorithm's strengths and weaknesses. All networks have one hidden layer, but which activations are used and where they are applied is varied between experiments.

3.1 Experiment setup

All experiments here share a common setup. To make sure that these are repeatable, fixed RNG seeds were always used and implementations were written to be fully deterministic where possible. A list of hardware and software used is available in appendix B.

We choose an activation function and where it is applied, vary the size of the hidden layer, and count how many of the randomly initialized networks are successfully trained. A network is considered successfully trained if the total mean squared error (MSE) over all possible inputs is less than or equal to 10^{-3}.

Success rate determination for networks with n inputs was performed by starting each algorithm on a network with a hidden layer size of $3\,|n|$. If the algorithm could not successfully solve all instances, the hidden layer was grown to at most $4\,|n| + 1$ nodes. After that, the network was shrunk until the algorithm could solve only less than 1% of all instances.

3.1.1 Network initialization

Networks for GD and GOMEA are initialized using the method that Pytorch defaults to [30]. That is, for each layer with $m \times n$ weight matrix W and m-element bias $\mathbf{b}$:

$$\mathbf{b}_j, W_{ij} \leftarrow U\left(\pm\sqrt{\frac{g}{n}}\right) \tag{3.1}$$

The initialization gain g defaults to 1, but larger values are chosen for some experiments (so-called "wide initialization").

Since BIPOP-CMA-ES requires a normal distribution and performs best when all parameters are within the same range (because the variance is shared between parameters), it uses the following initialization and transformation:

$$\begin{aligned} \mathbf{b}'_j, W'_{ij} &\leftarrow N\left(\mu = 0, \sigma = \tfrac{1}{4}\right) \\ \mathbf{b}, W &= \mathbf{b}', W' \times \left(\tfrac{1}{4}\sqrt{\tfrac{g}{n}}\right) \end{aligned} \tag{3.2}$$

The σ of $\frac{1}{4}$ was found by starting from guidelines [31], followed by trial-and-error on small- and medium-sized networks of two to four inputs. This yields a more narrow initialization than in section 3.1.1 (considering the 95 % interval), but without this modification the algorithm performed significantly worse.

3.1.2 Hyperparameters

All examined algorithms have hyperparameters. For GOMEA and BIPOP-CMA-ES, most were left at their default values.

For GOMEA, a stopping condition based on the population's objective variance was set at 10^{-11}. The linkage model was univariate or, for two experiments, fully covariant.

GD requires more extensive hyperparameter tuning: the Adam optimizer accepts a learning rate η and two betas β_1, β_2, as described in section 2.2.1. GDCR also has the restart threshold r.

Hyperparameter search was performed with a particle-swarm optimization (PSO) algorithm in lieu of more traditional grid search. PSO was chosen because it enables a larger and finer hyperparameter search space that, over time, concentrates search on the minima of this space. To speed up this search even more, since the training of NNs is quite slow, results were memoized and hyperparameters were transformed and rounded as follows:

$$\begin{aligned} \eta &= 10^{\mathcal{O}(x_\eta, 2)} \\ \beta &= \begin{cases} 1 - 10^{\lceil x_\beta \rfloor} & \text{if } x_\beta < -1 \\ \frac{9}{10}\mathcal{O}(-x_\beta, 10) & \text{if } x_\beta \geq -1 \end{cases} \\ r &= 10^{\lceil x_r \rfloor} \end{aligned}$$

Where $\mathcal{O}$ is a rounding operation:

$$\mathcal{O}(x, y) = \frac{\lceil xy \rfloor}{y} \tag{3.3}$$

(A) Hyperparameter transformation rules

$$\begin{array}{rcll} -3 \leq & x_\eta & \leq -1 & \\ -3 \leq & x_{\beta_1} & \leq 0 & \\ -3 \leq & x_{\beta_2} & \leq 0 & \\ -8 \leq & x_r & \leq 0 & \text{(GDCR only)} \end{array} \tag{3.4}$$

(B) Hyperparameter search ranges

The PSO swarm size was set at 100, and search was stopped after 100 iterations, if the objective value changed less than 10^{-12} between iterations, or if the success rate reached 100 %.

To help unsuccessful search converge faster, it was guided towards towards an artificial minimum based on the hyperparameters. Without this, the particle swarm would continue to move in random directions and tuning would take unnecessarily long. The defaults set for Adam were considered "safe" values, because these are proven to be sufficient to train the majority of networks [32]. The objective value for PSO to minimize therefore is, given a 100-run success rate v:

$$-v+10^{-8}\begin{cases}\left(\log_{10}\eta+3\right)^2+(\beta_1-0.9)^2+(\beta_2-0.999)^2+\left(\log_{10}r+8\right)^2 & \text{if } v<80\% \\ 0 & \text{if } v\geq 80\%\end{cases}. \tag{3.5}$$

For GD without cold restarts, the value r was ignored. A default target value of 10^{-8} was chosen to ensure that failure was not due to excessive restarts, i.e. degeneration into random search.

Once PSO terminated, an additional 900 runs for a total of 1000 samples were performed to find the final success rate of GD and GDCR.

PSO-based hyperparameter search was found to be effective for a CIFAR-10 classifier [33]. While rounding the values erases local variations and precludes precise finetuning, the general structure of the fitness space remains and so this approach should still be successful at finding good values.

The hyperparameters found by PSO are listed in table E.1.

3.1.3 Repeats, iterations and limits

Each experiment for GOMEA and BIPOP-CMA-ES was repeated up to 1009 times with different RNG seeds. Due to time constraints, some networks could not be run the full number of times.

GOMEA was allowed to run for up to $20\times\max(n-1,1)$ minutes, with n equal to the number of inputs to the network, or until the fitness variance limit was reached. Each process was allowed $\min\left(2^{\max(n-5,0)},20\right)$ CPU threads. Across all experiments, no more than 20 threads could be in use at the same time. All GOMEA processes were run in a shared memory pool of 16 GiB, though this limit was never reached.

GD was allowed up to 10 000 training iterations, but could stop early if the loss threshold was reached before that. Each experiment could use up to three CPU threads simultaneously, albeit at the lowest priority to allow room for other processes on the system.

Because the systems where experiments were run were different and the load on the systems varied throughout the period in which these experiments were run, runtimes are not directly comparable.

3.2 Experiment 1: ReLU activation at the hidden layer only

The aforementioned classic network for solving the XOR problem is a network of an input layer, a hidden layer and an output layer, all fully connected. The activation function at the hidden layer is ReLU, and there is no activation function at the output layer.

This network architecture was also used as the minimal example for the lottery ticket hypothesis: the network with two hidden nodes could only be solved by GD for approximately 35% of the initializations. The authors of [4] also found that only at 5 hidden nodes, GD managed to solve around 95% of all instances.

3.2.1 Results

Figure 3.2 shows success rates for GOMEA, BIPOP-CMA-ES, GD and GDCR.

Error bars for these and all following similar graphs are calculated according to the Wald method with expected proportion $p = 0.5$ and a 95 % confidence interval $Z = 1.96$:

$$\hat{p} \pm 1.96\sqrt{\frac{0.25}{n}} \tag{3.6}$$

Summaries are also included. These show the smallest hidden layer size needed for the algorithm to solve 95 % of cases. Error bars indicate best- and worst-case performance considering the 95 % confidence interval of the other graphs. ·

Initially, univariate GOMEA, GDCR and BIPOP-CMA-ES perform perfectly with 100% success across the board, much better than GD. The results for GD unsurprisingly match those seen by Frankle and Carbin in 2018. For 3 and 4 inputs GOMEA is unable to keep up when the hidden layers are very small, with BIPOP-CMA-ES and GDCR presenting better results and, in a few cases, even GD outperforming GOMEA. However, at 5 inputs univariate GOMEA beats all other algorithms, except for a few close calls at the small hidden layer range. All algorithms have significant trouble training networks with many inputs and a relatively small hidden layer. Fully covariant GOMEA performs the worst of all in these cases.

3.2.2 Discussion

It is no surprise that the population-based algorithms GOMEA, BIPOP-CMA-ES and to a lesser extent GDCR have little trouble training very small networks such as the 2-input, 2-hidden XOR network: these networks have only 12 parameters and so the solution space is quickly explored. The performance of GDCR, which degenerates into random search, is a testament to this fact.

The next two cases, where there are three and four inputs, are much more interesting, because univariate GOMEA is suddenly beat by GDCR and BIPOP-CMA-ES at the lower end. Looking at the number of generations, most runs are far outside the IMS range of 64 generations (8 populations, and a new population is started after 8 sub-generations). It therefore seems that GOMEA performs best in the IMS regime. This—too—is not surprising, considering that GDCR and BIPOP-CMA-ES operate in a similar fashion for a considerable amount of time, namely via the generation of new individuals as opposed to the improvement of existing solutions. A potential solution to the poor performance of GOMEA is to simply increase the population size, but the largest population of 1280 individuals is already very expensive to maintain, with GOMEA requiring, on average, more than 100 generations and 16 minutes to train a 6-input, 6-hidden network. Therefore this is not viable for even larger problems.

Fully covariant GOMEA failing to keep up with the other algorithms, let alone univariate GOMEA, is surprising considering that it in theory should behave somewhat similarly to BIPOP-CMA-ES. There are many factors at play here, so additional work is necessary to determine what causes this poor performance.

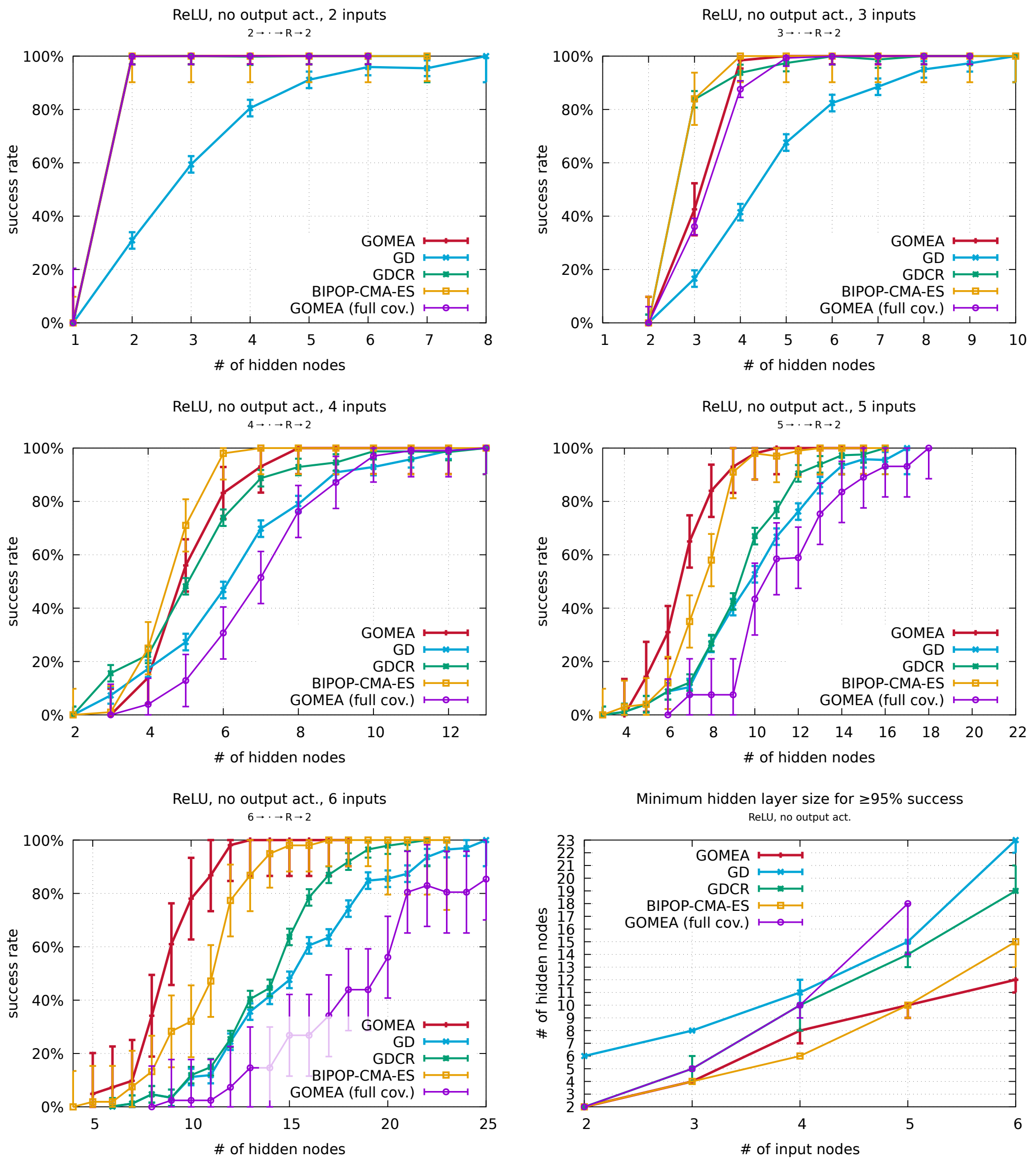


FIGURE 3.2: The first five graphs show success rates for n-input ReLU-based networks without output activation. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 95 % success rate.

3.3 Experiment 2: Sigmoid activation at the hidden and output layers

Another interesting architecture is one using the sigmoid activation function at both the hidden layer and the output layer. Since the sigmoid function symmetrically maps any input values into the range $[0, 1]$, it is especially interesting for binary applications such as XOR. Thus, it would be a fair bet to expect that algorithms will perform better with sigmoid activation rather than ReLU, although vanishing gradients poses a major problem for GD and GDCR.

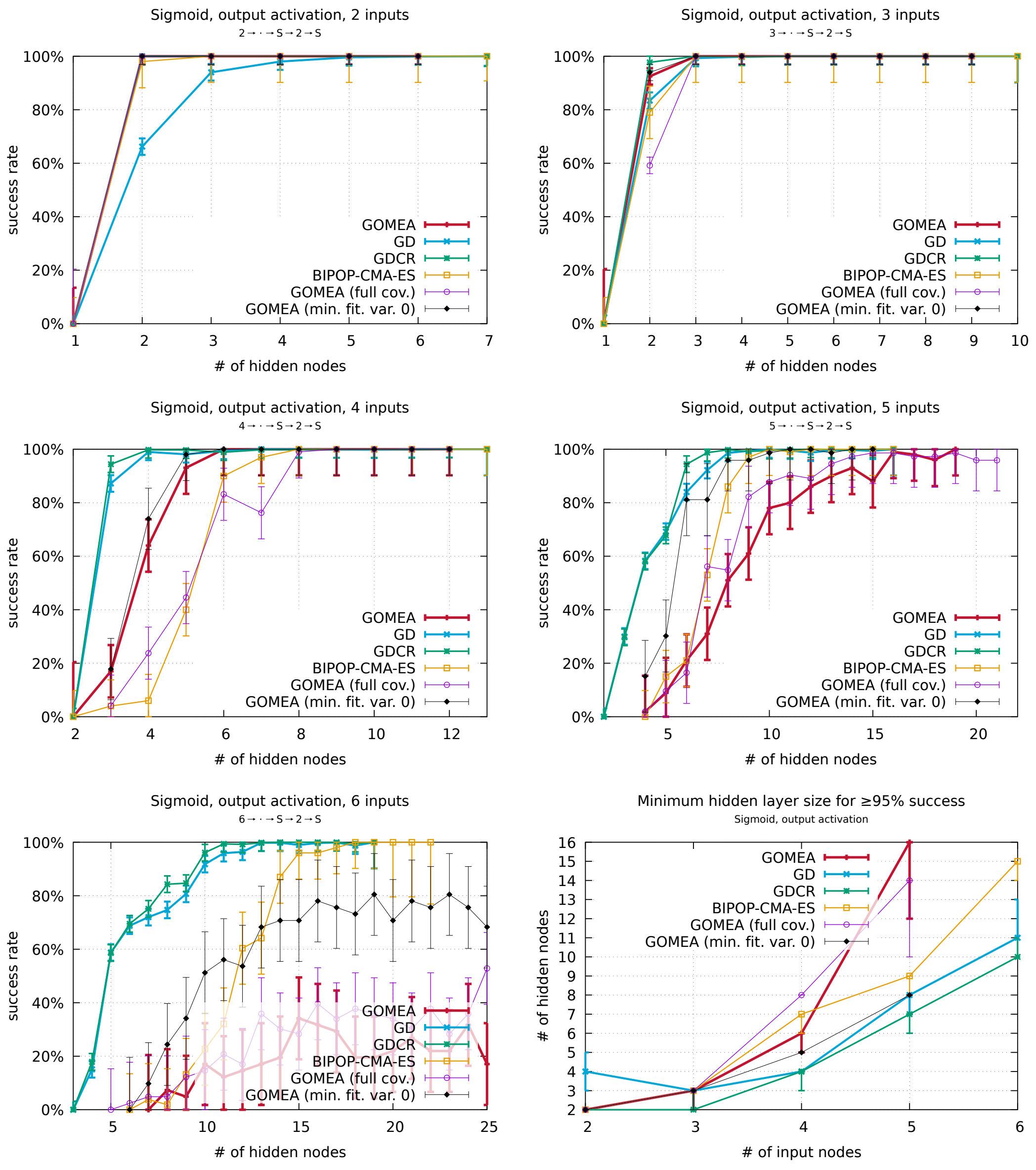


FIGURE 3.3: The first five graphs show success rates for n-input sigmoid-based networks with output activation. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 95 % success rate.

3.3.1 Results

Figure 3.3 shows that, for small input counts (up to four) all algorithms perform fairly well. It is at four and greater numbers of inputs that BIPOP-CMA-ES and GOMEA start to struggle, and after five inputs GOMEA falls off the chart completely.

Figure 3.4 illustrates the distributions of loss values found for 5- and 6-input network by GOMEA. As networks shrink, losses tend to approach 0.25.

3.3.2 Discussion

As expected, GD and GDCR are capable of training this kind of network to success. What is surprising is how unsuccessful GOMEA is here—if the problem is easier on GD, why is it *harder* for GOMEA? Part of the problem is that most randomly

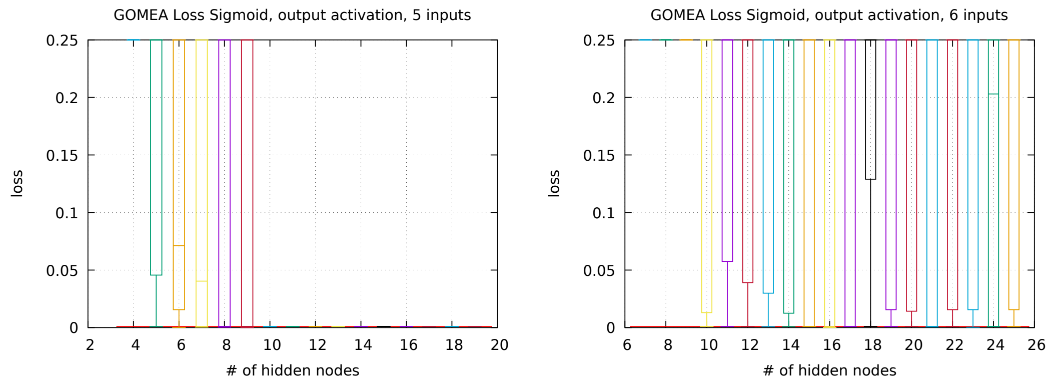


FIGURE 3.4: GOMEA final loss value distributions for five- and six-input networks. For five-input networks, as the hidden layer shrinks, losses tend to approach $\frac{1}{4}$. For six-input networks this happens across the board.

initialized networks will yield a loss value of $\frac{1}{4}$: these networks output a random vector in $[0,1]^2$ for each of the 2^n inputs. Assuming random inputs to the last activation layer, the expected output vector will be $\left(\frac{1}{2}\ \frac{1}{2}\right)$. Since half the expected values are $(1\ 0)$ and the other half $(0\ 1)$, the expected MSE will be $\frac{1}{4}$. This eliminates a major opportunity for GOMEA to explore the solution space, as every solution looks the same, small changes to the first half of the network accomplish little, and so it is quick to trigger the fitness variance stopping condition. The final populations found by GOMEA reflect this, as shown in fig. 3.4.

Working in a tight range around 0 is difficult for GOMEA and BIPOP-CMA-ES because no direct, intra-network gradient information is available: both need to evaluate changes in the network to find the direction to search in, but the small "view" of the network's behavior limit the algorithms' ability to explore. BIPOP-CMA-ES fares slightly better because of the integrated search direction, which GOMEA does not apply to all individuals.

There is not much difference between univariate and fully covariant GOMEA here: both perform poorly. This could give hints as to why the algorithm fails to properly train sigmoid-based networks and fully covariant GOMEA does not work well for ReLU-based networks, but, as before, more work needs to be done to draw conclusions. For the next experiments, fully covariant GOMEA is excluded; all GOMEA experiments used the univariate model.

To ensure that GOMEA's failure to fully train these networks was not fully due to the nonzero minimum fitness variance decided upon in section 3.1.2, experiments were re-run with it set to zero. GOMEA performed better, but still failed to reach success rates larger than 80 %.

3.4 Experiment 3: Sigmoid with wide initialization

One way of forcing GOMEA out of this saddle point, where the gradient of the solution space is zero, but is not a minimum, is to force the values at the nodes into the extreme parts of the sigmoid curve. This can be achieved by increasing the initialization range so that randomly generated networks are less likely to sit right on top of the saddle point.

Also, the large range allows GOMEA to quickly pivot from one side of the curve to another, which can yield significantly different results and keep the evolution process going. As BIPOP-CMA-ES operates on similar principles, it will likely benefit from this increased range as well.

Meanwhile, increasing the initialization range for GD can be expected to lower performance as GD will start to suffer from rapidly vanishing gradients: at $x \geq 6$, the gradient of the activation function quickly approaches zero.

For this experiment, an initialization gain $g = 120$ was chosen by trial and error. Because in the initialization range g appears under the square root, the effective range increase in both directions is $\sqrt{120} \approx 11$ times.

3.4.1 Results

To prevent a deluge of crowded graphs, fig. 3.5 shows only those for 5- and 6-input networks, i.e. the larger problems, and these graphs are split based on their algorithm. First, GD and GDCR performance is presented to establish a baseline. Then, for both GOMEA and BIPOP-CMA-ES, their performance is plotted against their and GD/GDCR's performance in the narrow scenario.

Expectedly, the figure shows that GD's performance suffers under the wide initialization compared to the previous experiment. However, GOMEA and BIPOP-CMA-ES perform much better, in the same ballpark as GD did originally.

3.4.2 Discussion

The "preferences" for the training methods are different here: GOMEA works best in the flat parts of the sigmoid curve, while GD needs values around 0 to function properly. This is not very surprising, because the extremes of the sigmoid curve map perfectly to the set of binary digits, but the vanishing gradients hamper GD's search performance here.

3.5 Experiment 4: Heaviside activation

Now that we know that GOMEA and BIPOP-CMA-ES prefer to work with the flat parts of the sigmoid curve, we will look at how far we can push it: can they still successfully train networks activated using only the flat regime, i.e. the Heaviside activation function?

The derivative of the Heaviside step function is the Dirac delta; the gradient is zero everywhere except at $x = 0$, where it is infinite. Therefore, the question will be easy to answer for GD and GDCR: if output activation is used, no gradients are available. GD will not be able to update any parameters, while GDCR can only restart every iteration (i.e., perform random search). Not having any output activation at least allows the second linear transformation to be trained, but the first will remain fixed. Still, it will be interesting to see these algorithms' performance, if only to gauge the training ability of random search versus EAs.

3.5.1 Results

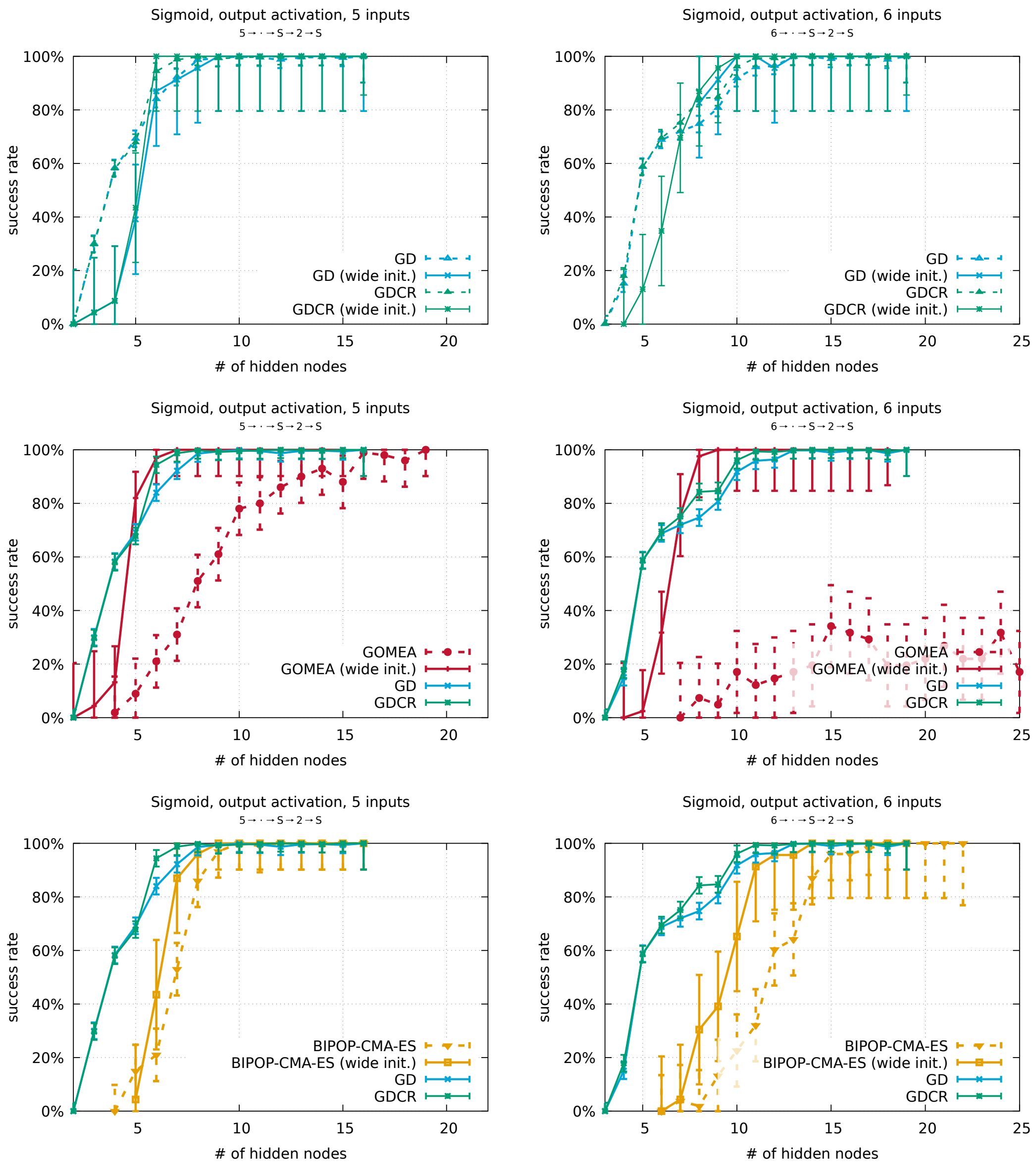


FIGURE 3.5: Each row of plots compares the success rates for five- and six-input sigmoid-based networks for a narrow versus wide initialization, for GD, GOMEA and BIPOP-CMA-ES, respectively. Previous (narrowly initialized) results for each algorithm are shown as broken lines, and results for narrow GD are included as a baseline result to be improved on. Columns are aligned; values along the x-axis are in the same position on each graph within a column.

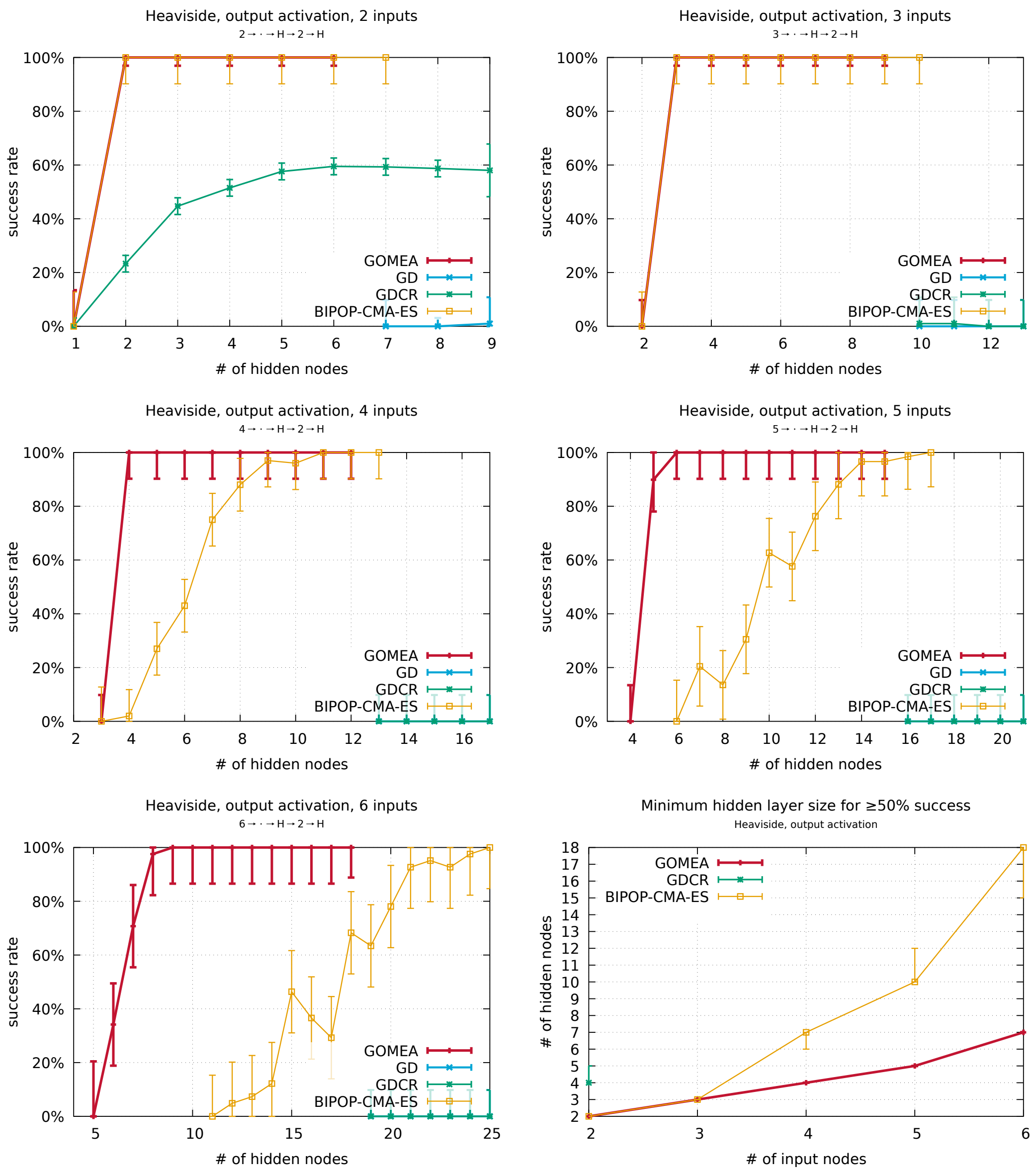


Figure 3.6: The first five graphs show success rates for n-input Heaviside-based networks with output activation. GD and (for more than two inputs) GDCR appear in the bottom-right corner. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 50 % success rate. GD does not appear here as it never attained a 50 % success rate.

3.5.2 Discussion

First, GD is unable to find any successful network. Its only, barely visible point on the graph overlaps the legend, meaning that even the "easy" large networks are not trainable in the slightest. GDCR fares better, but only because its gradient norm reaches 0 immediately and it keeps randomly creating networks until a suitable one is found. Here, the discreteness of the activation function helps, because a network is producing either incorrect or perfect, 0-loss results.

On the other hand, GOMEA and BIPOP-CMA-ES are performing very well, with GOMEA having a significant edge over BIPOP-CMA-ES here. This is in contrast to sigmoid activation—there, BIPOP-CMA-ES could much more effectively search the solution space than GOMEA could. Thus, GOMEA is very good at searching for a correct, discretized network, while BIPOP-CMA-ES is relatively better at searching a gradual, but heavily graded space.

3.6 Experiment 5: ReLU with output activation and sigmoid-like without

For completeness' sake, we will also "take the cross-product" and consider ReLU networks with output activation and sigmoid(-like) networks without.

Adding output activation to already trained ReLU-based networks without can only improve the performance, because since the output is expected to be in $\{0, 1\}$, folding all negative values to 0 makes it adhere better to the output range. Well-performing networks will therefore likely not behave differently at all after introducing output activation.

This does not mean that training such output-activated ReLU networks from scratch is easier. The activation function erases any information about how far the 0 outputs are from correct 1 values; the gradient in the output activation is 0 here. ReLU here will probably also not help GOMEA in this case, though it can handle the erasure better than GD as we've seen before.

Meanwhile, sigmoid-like networks without output activation require more careful training, as sums at each node must be near 0 or 1, instead of some arbitrary small or large value, respectively. GD may perform better here because it was already training mostly in the small-value range where gradients are preserved better. GOMEA will likely have an easier time for similar reasons.

Heaviside activation is also considered; removing output activation gives GD a fighting chance as it can access the gradient up to the hidden layer, but is still forced to keep the values before it the same.

3.6.1 Results

Due to the large number of graphs required for displaying these results, summaries are presented instead. Two graphs are shown, one containing the minimum number of nodes in the hidden layer that each algorithm can solve 50 % of times, and the other 95 % like before.

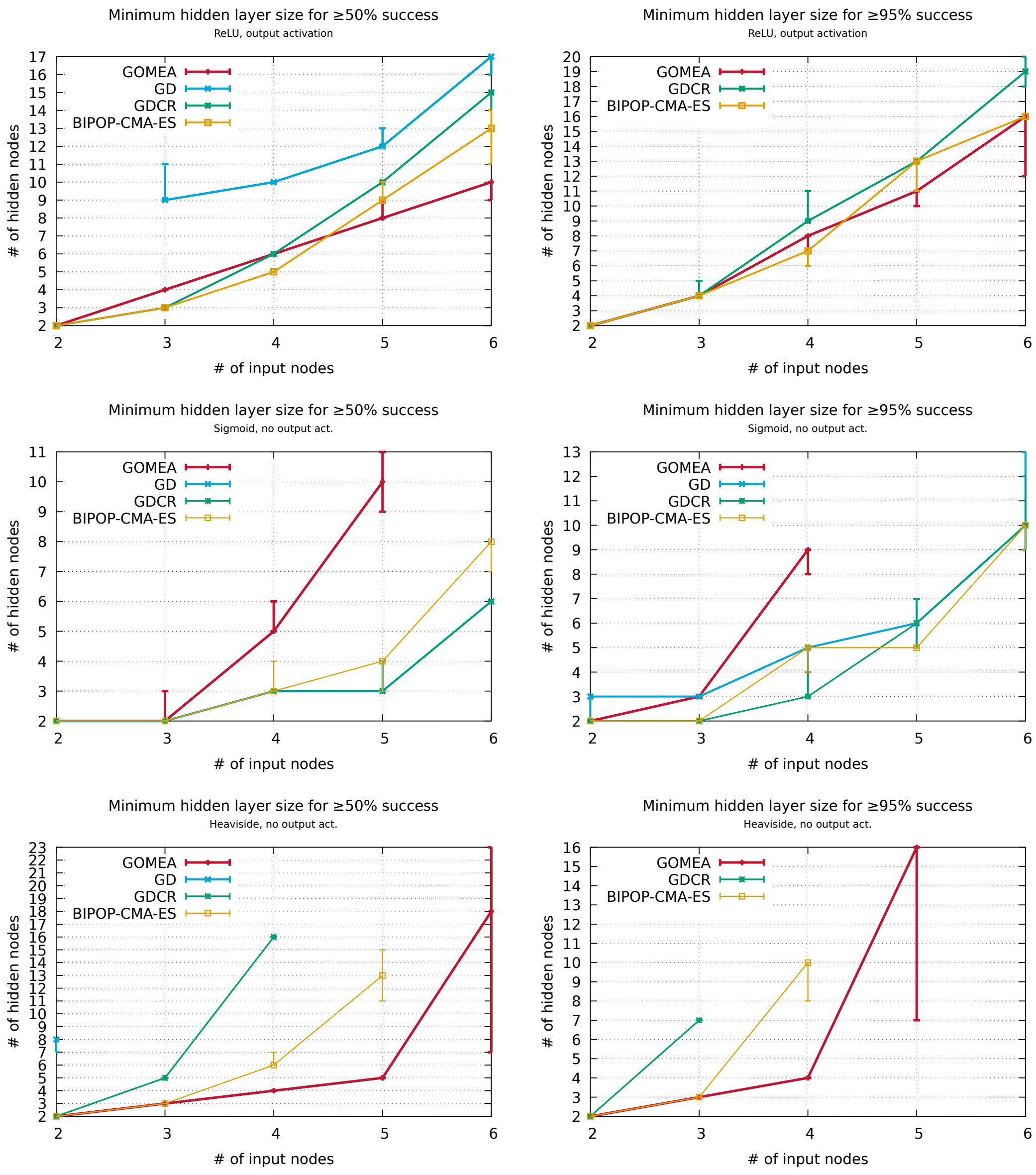


FIGURE 3.7: Minimum hidden layer sizes for various networks. The first row shows the minimum number of nodes in the hidden layer required for success rates of 50 % and 95 %, respectively, for ReLU-based networks with activation both at the hidden and output layers. The second row displays this information for sigmoid networks *without* activation at the output layer, and the third for Heaviside networks without output activation. In the last row GOMEA is close to, but not quite reaching the required threshold for smaller networks (those with 8 hidden nodes, for example), leading to large error bars.

3.6.2 Discussion

By now, the results should not be very surprising: networks with ReLU output activation hamper GD's performance due to unnecessary gradient erasure for half of all outputs, but the three other algorithms can navigate around this limitation. GDCR does this by simple restarts, which happens quickly as the gradient diminishes fast. GOMEA and BIPOP-CMA-ES both beat GDCR for larger networks, with GOMEA having an extra edge.

Sigmoid-based networks with a narrow initialization procedure remain challenging for GOMEA, while BIPOP-CMA-ES fares somewhat better. Both are performing worse than GD and GDCR, which effectively utilize the activation function to train relatively small networks.

Even without any output-layer activation, the gradient-less Heaviside activation function poses an extreme barrier for GD and GDCR. Between GOMEA and BIPOP-CMA-ES, GOMEA performs ever so slightly better. Interestingly, BIPOP-CMA-ES performs about the same as when given Heaviside networks with output activation, while GOMEA is having much more trouble with this configuration; GOMEA is not as good as fine-tuning network weights while it excels at searching the discretely-activated networks.

3.7 General discussion

Not only can GOMEA and BIPOP-CMA-ES successfully train these XOR networks, in many cases they also perform better than both classical GD and GD with cold restarts. When ReLU-based networks without output activation are considered, GOMEA and BIPOP-CMA-ES are well ahead of GD and GDCR.

This partially extends past ReLU-based networks: also sigmoid networks where the output is activated, and other possible configurations. Interestingly, GOMEA requires a wide initialization range for sigmoid-based networks such that the activations tend to be in the flatter parts of the function. This is in stark contrast to GD, which suffers from vanishing gradients in this configuration. What this indicates is that GOMEA trains networks in a different manner compared to GD, “sees” the problems differently, as is obviously to be expected when comparing an EA versus a gradient-based method.

Meanwhile BIPOP-CMA-ES maintains better performance in the narrowly-initialized cases, though it also achieves lower success rates than both GD and GDCR. It does not benefit from increasing the initialization range—in fact, it performs worse, like GD and GDCR. BIPOP-CMA-ES’s step-based search approaches a gradient-based method [34, 35], explaining this behavior.

Of the “conventional” architectures, one based on ReLU without output activation allows GOMEA and BIPOP-CMA-ES to train the smallest networks. For GD and GDCR the best-fit architecture is sigmoid-based. GOMEA performs best overall when training Heaviside networks with output activation.

The fact that GOMEA achieves better results in the relatively flat ranges of the sigmoid curve and GD performs best in the near-linear portion is supported by the excellent performance of GOMEA for networks using the Heaviside step function. As expected, GD is incapable of training the Heaviside network with output activation, due to immediate gradient erasure, and greatly struggles with the network without as it cannot train the hidden layer and its inputs.

For networks with many inputs and small hidden layers, all algorithms perform poorly—but what is surprising is how even GD manages to find a better solution than GOMEA in some cases. Part of this could simply be due to pure luck: only a limited number of runs were performed and so the results are within the margin of error. With networks this small, the probability of randomly generating a “winning ticket”, so to say, is greater. But that would also work in GOMEA’s advantage since it generates many different networks at the start, so there may be something else going on as well. The population fitness variance threshold may simply be set too high, terminating small populations too soon and preventing refinement while larger populations are only starting and can not reach the refinement stage before time runs out.

There may also be a more fundamental cause to this behavior. The losses generated by unsuccessful training attempts are often those typical of saddle points. It seems that, even with AMS forcing search in a given direction, GOMEA is unable to handle the enormous number of minima these XOR networks have. GD does not suffer from this, as it has only one perspective on the solution space, and GDCR simply throws away a network and re-rolls if it seems stuck—no knowledge is passed on. BIPOP-CMA-ES's "momentum" can help it overcome these minima, though it still performs worse in the end.

3.8 Conclusion

We can therefore already partially answer the first and second research questions: yes, EAs are able to train small XOR networks and improve upon GD doing so; and yes, for these networks the activation functions used does impact the performance, but in a different way than they do for GD. A key insight is that best practices for training via GD do not fully transfer to EA-based training procedures, nor can they necessarily be carried over between different EAs. Even basic choices such as which activation functions to use, where to apply them and how to initialize networks need to be reconsidered; networks trained by EAs can be fundamentally different compared to those by GD.

Chapter 4

Applying problem-specific knowledge: asymmetry loss

The high number of minima of XOR networks is partially caused by the extreme degree of symmetry within the hidden layers of MLPs. Because these layers are fully connected to the previous and the next layer, each node within the layer can be "rearranged" at random, yielding $n!$ permutations with the exact same loss value. The larger the hidden layer, the more difficult it will be to compare two random networks.

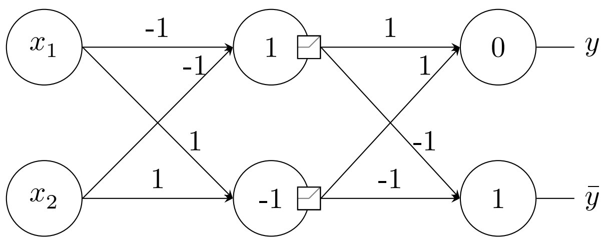
High degrees of symmetry hamper the training procedure as the distributions become highly-modal; each permutation can potentially require its own, distinct parameter distribution. Preferring a single arrangement reduces the number of possible solutions greatly and thereby simplify the parameter distributions.

This effect is strongest for very small networks. Consider the ways a small network can have its hidden layer extended as described in section 2.4.2. If a single node is added, then it can "share" the duty of another node. Because their contributions are summed, the distribution scales linearly—even outside $[0, 1]$, as long as it sums to 1. This can be extended to an arbitrary number of nodes. Doing so heavily skews the distribution of the associated parameters, as most will tend to zero to compensate for other nodes contributing the same information. While the problem is easier, estimating parameter distributions is less useful. The smaller the hidden layer, the fewer of these opportunities arise, concentrating the "correct" distributions for parameter values. It is thus here that asymmetry loss helps with further reducing the number of valid solutions.

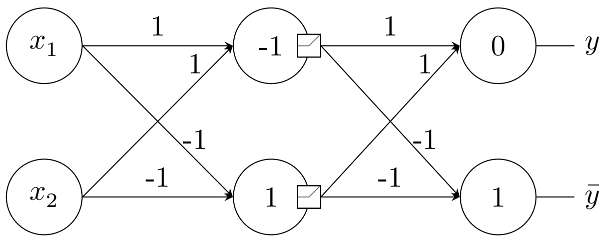
BIPOP-CMA-ES works differently. Since it encodes the expected search direction, as opposed to the distribution itself, it is more directly traversing the gradient of the function (averaged over the population) and strong individual differences are diminished.

A symmetry-breaking soft-constraint can be used to penalize networks not following an expected layout.

One such penalty term for guiding the networks towards a fixed layout of the hidden layer is to require that the bias of each node is not greater than that of any following node: the nodes have to be sorted by the bias in ascending order. For each pair of nodes where the former has a higher bias than the latter, the difference plus one is added to the asymmetry loss. This penalty term is then multiplied by an arbitrary factor of 1000 and added to the original loss. The asymmetry loss is scaled because bias values (and thus also their differences) tend to be insignificant compared to the output MSE loss.



(A)



(B)

FIGURE 4.1: Two distinct, but both perfect solutions to the 2-input, 2-hidden XOR problem. The top network is the same as shown in fig. 2.13, whereas in the bottom network the two hidden nodes and their associated weights have been exchanged; now the top node outputs 1 iff $x_1 = x_2 = 0$ and the bottom node outputs 1 iff $x_1 = x_2 = 1$.

Thus, for a network with m hidden nodes, an additional operation on the loss ℓ takes place:

$$\ell \leftarrow \ell + 1000 \sum_{i=1}^{m} \sum_{k=1}^{i} \begin{cases} b_k - b_i + 1 & \text{if } b_i < b_k \\ 0 & \text{if } b_i \geq b_k \end{cases} \tag{4.1}$$

This effectively "tilts" the solution space towards the solutions with ascending biases in the hidden layer. The biases of the output layer are ignored, because the output order depends on the problem and nodes cannot be permuted at random.

If we were to say that the top hidden node in each network of fig. 4.1 is node 1 (and the other 2), then using this method fig. 4.1a has an asymmetry loss of 3000, while fig. 4.1b yields 0, meaning that configuration (B) is the preferred one—despite both being perfect solutions to the XOR problem.

Using this loss for GD does not make much sense as GD does not need to cross over between multiple solutions. It is not required in order to get a functioning XOR network; just to help an EA select solutions that are more directly comparable by focusing on a smaller part of the solution space. If applied to GD, it would only make the problem more difficult to solve.

4.1 Experiment setup

The experiments performed here are the same as for the previous chapter: for networks with 2 to 6 inputs, shrink the hidden layer until less than 1 % of the attempts successfully trained a network. Except for the introduction of the asymmetry loss described above, everything else remains the same.

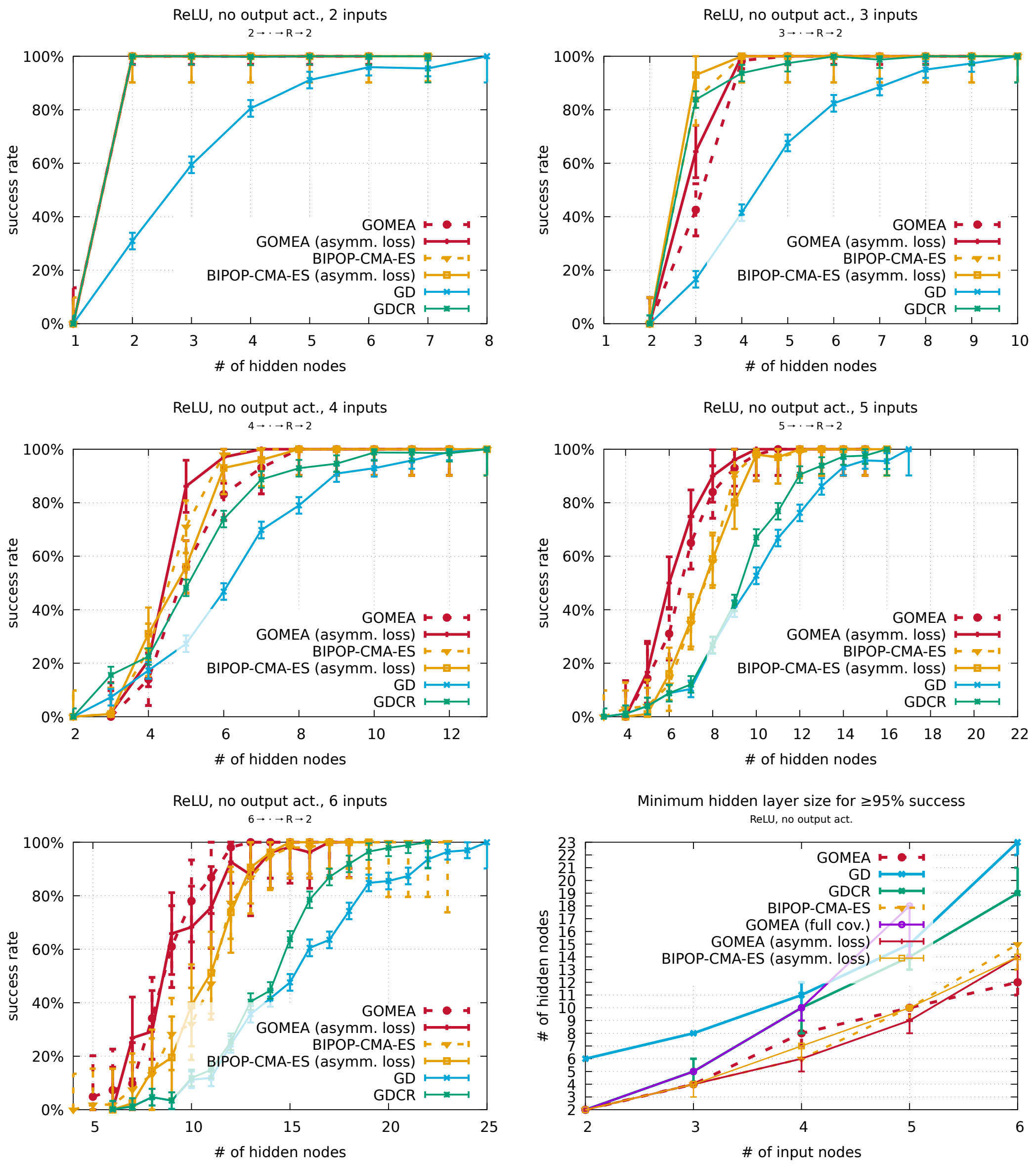


FIGURE 4.2: The first five graphs show success rates for n-input ReLU-based networks without output activation. Previous results (those without asymmetry loss applied) are shown as broken lines. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 95 % success rate, also with previous results included.

4.2 ReLU activation at the hidden layer

GOMEA benefits from asymmetry loss: for networks with three to five inputs it performs slightly better and even gains significantly at the small hidden layer range. For six inputs, however, asymmetry loss does not make much of an impact. GOMEA was also already outperforming other algorithms at this point.

BIPOP-CMA-ES on the other hand does not get as strong of a performance boost compared to GOMEA. In a few cases it achieves better results, but most of it is within the margin of error. Strikingly, for networks with five inputs, its performance is more or less equal to the variant without asymmetry loss.

4.2.1 Discussion

Introducing asymmetry loss as a symmetry-breaking constraint is helpful to GOMEA for small networks, where it yields better results for all sizes, but most importantly improves the performance when training very small networks. Now, GOMEA is also competitive at these sizes.

4.3 Sigmoid activation at the hidden and output layers

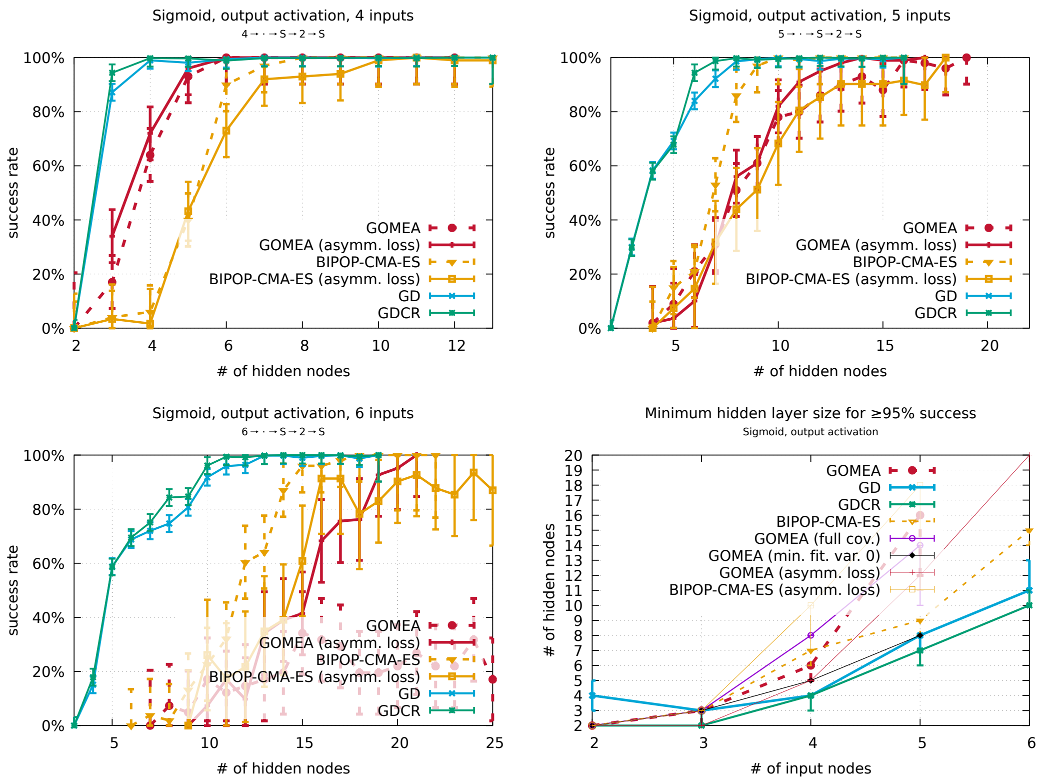


FIGURE 4.3: The first three graphs show success rates for n-input sigmoid-based networks with output activation. Previous results (those without asymmetry loss applied) are shown as broken lines. The last plot is a summary showing the minimum number of nodes in the hidden layer necessary for a 95 % success rate, also with previous results included.

Improvements are not as strong as they were for the non-output-activated ReLU network—except for the case with six inputs, where it helps GOMEA reach much better success rates than before.

As in the previous case, BIPOP-CMA-ES barely changes in performance.

4.3.1 Discussion

This style of network (with a narrow initialization range) is already difficult for GOMEA for reasons other than asymmetry, so adding this constraint is not very effective—except in the case of six inputs, where it suddenly performs much better for larger networks. Even though it is still far behind GD, introducing asymmetry loss suddenly made these networks somewhat viable.

Inhibiting asymmetry would allow search to focus on smaller distributions, usually on one side of the zero: even if individual changes accomplish little, moving search in a one direction (as opposed to keeping the distribution around zero) will produce slightly better results. As shown before, larger networks allow weak activations within the network; thus, even if individual activations are weak (because the parameters are too small), large networks can accomplish working behavior anyway.

However, this is not what appears to be a full explanation. While a few distributions shrink as expected, the vast majority are still centered around 0. Penalizing asymmetry helps, but it is for now not clear exactly why.

4.4 Other networks

All other configurations listed in the previous chapter were also tested with asymmetry loss enabled. Because these are quite numerous, only summaries are shown and discussed.

4.4.1 Heaviside with and without output activation

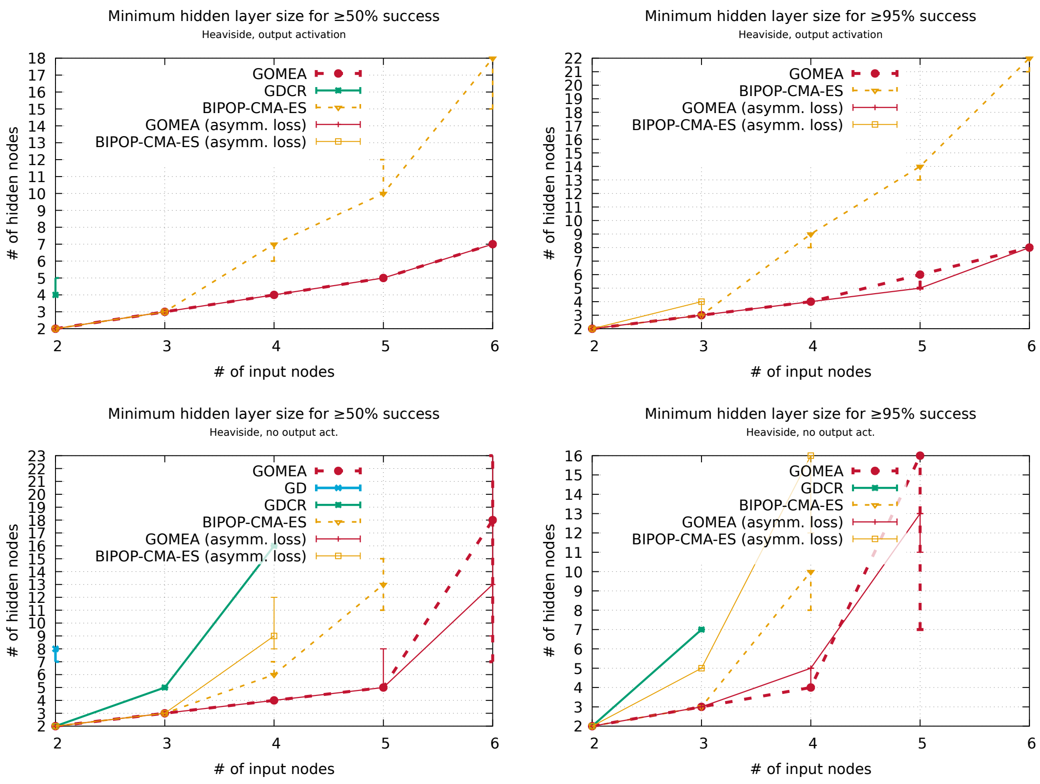


Figure 4.4: Minimum hidden layer sizes for Heaviside problems. The first row shows results for Heaviside-based networks without output activation, with previous results (where no asymmetry loss was applied) as broken lines. GD and GDCR without asymmetry loss are included as a baseline result, but are (mostly) absent due to their inability to train networks with Heaviside activation as seen in fig. 3.6. BIPOP-CMA-ES is similarly absent from most of the graphs. The second row shows results for Heaviside networks without output activation in the same way.

Figure 4.4 shows summaries of training results for these networks.

Asymmetry loss has little to no effect on GOMEA's success with solving output-activated Heaviside networks. There is not much to gain here: GOMEA already found very small networks without any symmetry breaking to begin with.

When the network does not have any output activation, GOMEA benefits slightly from symmetry breaking when given many-input problems.

As before, BIPOP-CMA-ES is performing worse when training with asymmetry loss.

4.4.2 ReLU with output activation and sigmoid without

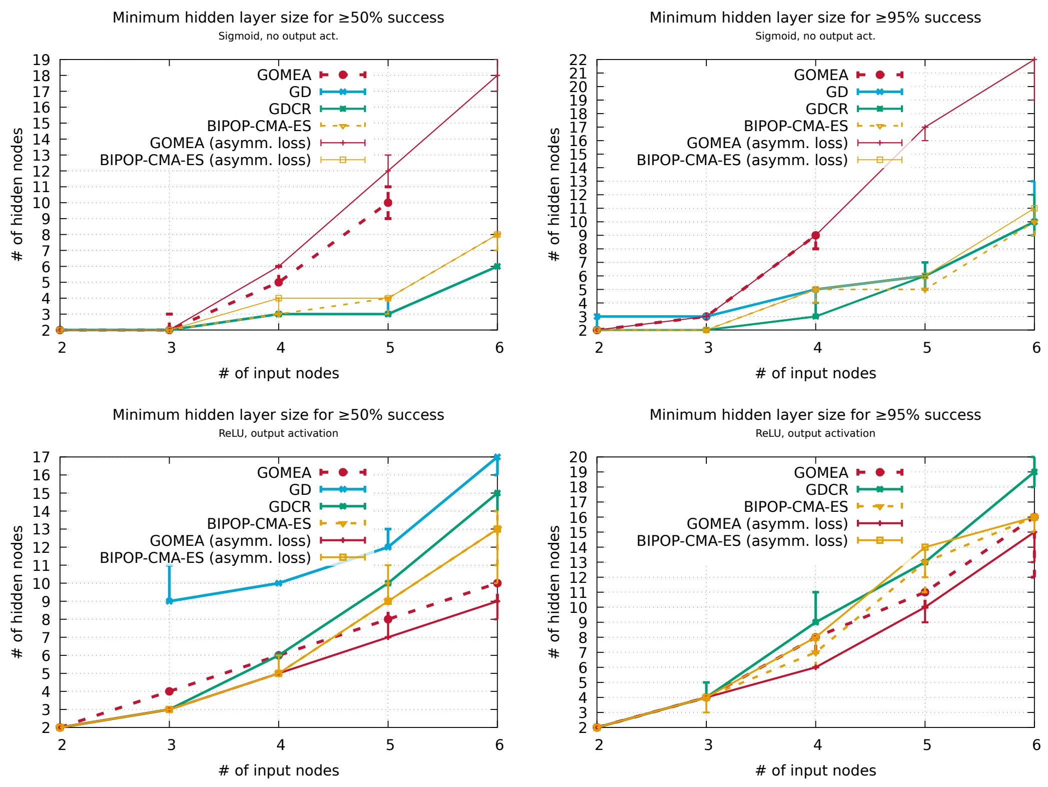


FIGURE 4.5: Minimum hidden layer sizes for various problems. The first row shows results for sigmoid-based networks without output activation, with previous results (where no asymmetry loss was applied) as broken lines. GD and GDCR without asymmetry loss are included as a baseline result. The second row shows results for ReLU networks with output activation in the same way.

As with sigmoid networks with output activation, those without are still difficult for GOMEA and enabling asymmetry loss makes them somewhat more viable: there is at least any result as opposed to none before. This also again shows that symmetry is not the sole cause of training difficulties when GOMEA is given a narrowly-initialized sigmoid-based network.

Similarly, BIPOP-CMA-ES does not benefit from a symmetry-breaking constraint and performs about the same, occasionally worse.

4.5 General discussion

Answering the third research question, imparting problem-based knowledge like asymmetry constraints does help in a number of cases, but it is not a panacea: it only helps with smaller networks where GD was able to produce better solutions, but it has next to no effect on larger network where GOMEA was already performing better. It is also an imperfect solution: it unnecessarily penalizes good solutions that do not fit the mold, which leads to a loss of diversity [36]. These lost (partial) generations can be reduced through an initialization function that automatically generates asymmetric individuals, where the initialization range is divided over each sequence of biases.

This sort of guidance does not even help any EA, as can be seen from the fact that BIPOP-CMA-ES does not benefit at all from asymmetry loss.

Chapter 5

Conclusion and future work

This chapter concludes the work by discussing the most important , as well as possible future work.

5.1 Discussion

Chapter 3 showed that GOMEA and BIPOP-CMA-ES can not only successfully train NNs, they are capable of training smaller networks than GD can. GOMEA also trains networks that are impossible to train via GD with ease.

However, there are cases where established best practices for GD negatively affect EAs, such as how a sigmoid-based network is initialized. An EA like GOMEA trains in a very different manner than GD does, so these differences are not surprising. But, one must take care when transferring GD-based knowledge to an EA-based training method, and even basic decisions may need to be reconsidered.

EAs also face problems that for GD are less pronounced or do not exist, making incorporating problem-specific knowledge beneficial if not necessary. One such trait of feed-forward NNs is the extreme symmetry within hidden layers. Breaking this symmetry appears to be helpful to GOMEA in a number of cases, but it fails to solve other issues and is barely if at all applicable to BIPOP-CMA-ES. Such constraints also need to be implemented carefully to prevent loss of diversity in a population.

GOMEA in particular provides another powerful way of indicating the intrinsic qualities of a network, namely via parameter linkage. Linkage can indicate logical relations within a layer and between consecutive layers, but the nature of feed-forward networks makes it difficult to find any clear links beyond this.

The elephant in the room here is the computational cost of training a network via an EA. Sensible population sizes (for the algorithms discussed here) vary from tens to thousands of networks, each of which needs to be evaluated and mutated. GOMEA performs “partial evaluations”, which can be applied to NNs in some cases, but this does not scale well and overhead may make them more costly than simple full-network evaluations.

Requiring more resources to train NNs can still be beneficial if the trained networks end up being smaller than what GD can produce: a network is trained once, but applied many times. If an EA takes significantly longer to train a network, but the network is also significantly smaller, then this ends up saving resources in the long run.

5.2 Future work

The success and flexibility of using EAs to train NNs opens up a vast area of research.

A few features of GOMEA that have been unexplored in this thesis are the application of partial evaluations and problem-specific linkage to NNs. Partial evaluations, as explained in section 2.3.3, can reduce the number of computations necessary to evaluate a network where only a subset of the parameters have been modified. Combined with a non-fully-covariant linkage model, this can speed up the training of networks. However, not all network architectures are fully invertible, precluding efficient partial evaluations. Partial evaluations combine well with problem-specific linkage models (see section 2.3.3), which inform GOMEA about relations between parameters intrinsic to the problem. In an MLP, such relations between parameters exist, such as a bias and its associated incoming or outgoing weights. Specifying these could allow GOMEA to perform more effective evolution. Appendix A has a comprehensive overview of possible approaches to both partial evaluations and linkage when applied to NNs.

The wall-clock time spent evaluating and mutating all these networks can be decreased by moving the training process to GPUs, something which is already common for GD. GPUs are excellent at parallel processing and matrix manipulation. With a sufficiently large GPU and a sufficiently small network, an entire population can even be evaluated at the same time. These devices also usually feature a very wide memory bus and large caches, potentially making partial evaluations viable for a larger range of networks.

Linear layers are a common sight in NNs, but they are far from the only ones available. Image recognition networks often feature many convolutional layers, such as in VGG [45] and ResNet [46]. The expressive power of linear layers is often not necessary and their great number of parameters is even for GD quickly too much to handle. Convolutional layers, on the other hand, have a relatively limited number of parameters. The arrangement of these parameters as distinct filters also lends itself very well to explicit intra-layer linkage, and creates more opportunities for effective cross-layer linkage.

Last in this list of ideas (which is far from complete) is having GD and GOMEA cooperate, also known as hybridization. EAs are excellent at exploring the search space, something that GD is less capable of. GDCR works around this by giving up on bad starting positions and trying again, but in bad cases this turns into random search. In particular, it would be very interesting to see if an EA like GOMEA could function as a “winning ticket generator”: the EA selects one or more networks that perform relatively well, and GD trains these to a final network.

5.3 Conclusion

To conclude, EAs perform well when training NNs and often manage to train very small networks, something which is much more difficult for GD. However, there are some cases where GD still performs better, so care must be taken to construct both a network and an EA that work well together, and what works can not always be inferred from previous experience with GD. The application of GOMEA for training NNs opens up an immense world of future research and other opportunities; much is still to be learned.

Appendix A

Partial evaluations and linkage

As described in section 2.3.3, GOMEA is capable of much more advanced linkage and evaluation methods than used in the previous chapters. Here, we give an overview of possible future enhancements to the training procedure that may allow GOMEA to scale to larger networks.

A.1 Partial evaluations of neural networks

The linear transformation that is core to fully-connected feed-forward neural networks is trivial to evaluate partially. From eq. (2.3), this transformation is as follows:

$$\mathbf{y} = W\mathbf{x} + \mathbf{b} \tag{A.1}$$

There are two possible kinds of modification here: to the weights, and to the biases. Evaluating a set of bias changes from b to b' is very simple:

$$\begin{aligned} \mathbf{y}' &= \mathbf{y} - \mathbf{b} + \mathbf{b}' && \text{(A.2)} \\ &= \mathbf{y} + (\mathbf{b}' - \mathbf{b}) && \text{(A.3)} \end{aligned}$$

Changing weights W to W' is slightly more complicated as it needs to multiply with the input vector:

$$\begin{aligned} \mathbf{y}' &= \mathbf{y} - W\mathbf{x} + W'\mathbf{x} && \text{(A.4)} \\ &= \mathbf{y} + (W' - W)\mathbf{x} && \text{(A.5)} \end{aligned}$$

Of course, combined partial evaluations, where both weights and biases have been changed, are also possible.

If only few values are changed than equations A.3 and A.5 involve a sparse vector and matrix, respectively. Sparse matrix multiplication (and sparse vector addition) are cheap to evaluate, and much cheaper than full evaluations if the density is low enough [37].

A.1.1 Nonlinearity

Unfortunately, a layer in a feed-forward network is usually not just a linear transformation; a nonlinear transformation is applied as well. This is not immediately a problem: if the transformation is invertible, then it can be inverted, the linear part evaluated, and finally re-applied. However, many activation functions are not invertible: ReLU and Heaviside are not for example, and ReLU is extremely popular [38]. Many ReLU-like alternatives, such as Swish [39], maintain a non-zero derivative over most of their range, but are still not invertible because of ringing artifacts. Some others like LeakyReLU [40] and potentially PReLU [41] (if parameter $a \neq 0$) are, however.

These alternatives are not drop-in replacements; networks can train and behave very differently using even subtly different activation functions [42].

Storing intermediate results

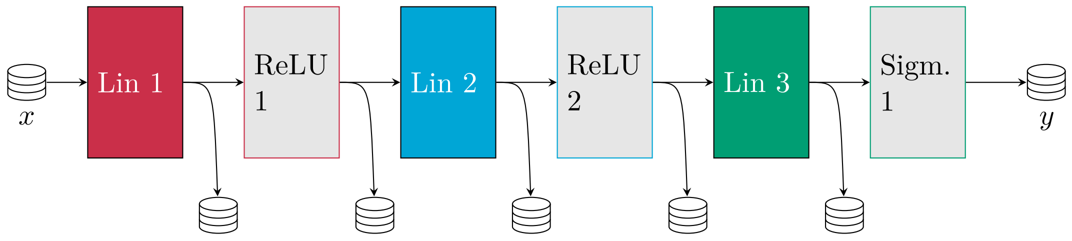
If a non-invertible activation function is used and partial evaluations are desired, then it may be beneficial to store intermediate activations in addition to the final loss value. Once a linear transformation has been computed, the vector is written to memory. If a parameter in that layer changed, the un-activated output vector can be used to compute the new one. Then, the activation function can be applied and the resulting vector can be passed to the next layer (or output).

There are multiple downsides to this. Now, a network under training no longer needs to store its parameters, but also the previous outputs of each layer in the network. Depending on how complex the network is, this is a significant amount of memory, especially if fairy large populations are used by the EA.

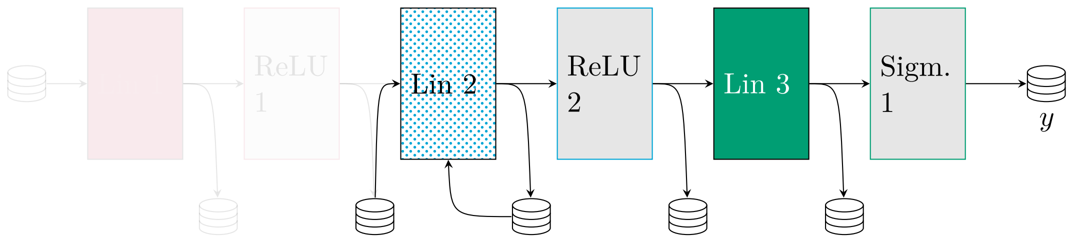
Additionally, there is significant memory bandwidth overhead as these vectors need to be written to and read from memory for each evaluation. Full evaluations can discard any previous vector, but still need to write all vectors completely to memory. Partial evaluations themselves can be lighter on memory bandwidth as their modifications are local, but this does not hold for layers further downstream.

Because each layer in an MLP is fully connected, changing a weight or bias in an earlier layer will propagate the modified value to all nodes in subsequent layers. If the network is especially deep or if the initial layer is large, then partial evaluations will have little advantage over simply evaluating the entire network with the new parameters in place.

All this essentially trades computing power for memory bandwidth—which can be a terrible trade due to the very large and ever-growing processor–memory performance gap [43].



(A) A full evaluation of a network.



(B) A partial evaluation of a network, starting in linear layer #2.

Figure A.1: Two modes of evaluation of a network.

Training layer-by-layer

Many of these problems can be alleviated by training a network not in its entirety, but layer-by-layer. A similar technique has also been successfully applied to GD in the form of local updates [44].

The network is trained starting from the input layer. Because its output shape possibly does not match the expected output of the full network, a “reshaping” layer is appended that transforms the layer-under-training’s output to the expected shape. This layer is discarded when this training stage has finished.

Obviously, this reshaping is not necessary if the penultimate layer is being trained, as the last layer is performing the exact same role.

Once a layer has been trained, it can be evaluated against the full dataset, which can then be used as the input to train the next.

A major downside of training layer-by-layer is that each layer is forced to see the big picture, as it were, whereas networks trained as a whole allow layers to specialize in features of the input data. Laskin et al. put forward different ways to group layers that allow for some inter-layer cooperation, but it will likely always perform worse than if every layer of the network was trained at the same time.

For the general network it is unreasonable to assume that only the first few layers are sufficient to form a network capable of solving the problem it is being trained for; stopping criteria other than a value-to-reach are necessary. Obvious choices include a maximum runtime or evaluation count, though as layers differ in number of parameters, these may need to be set on a per-layer basis.

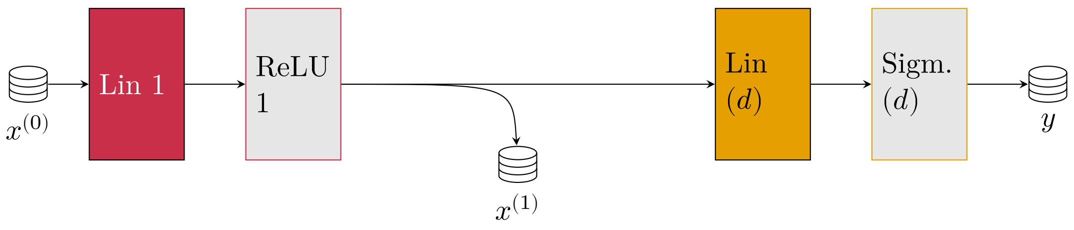
Another possible criterion is the population’s fitness variance; if it looks like search is starting to converge (especially on a value far from the desired value), it may be beneficial to stop with the current layer and already start on the next, as otherwise one may spend too much time fine-tuning the discarded layer, and the next layer could be able to pick up the slack in the previous.

Of course, if search is converging on a good loss, then search can stop at the truncated network with the reshaping layer attached, yielding a network that is different than originally intended, but smaller and possibly more efficient.

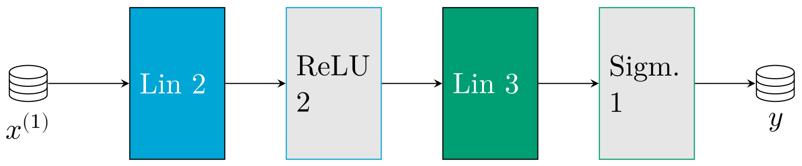
However, well-performing truncated networks may not always be desired for myriad reasons. For example, the reshaping layer could be too large for the intended application. This holds especially if non-fully-connected layers are used in the network (though these are outside the scope of this work), but some kinds of network innately require a certain topology.

One such kind is the *autoencoder*, where a series of progressively smaller layers “compress” their input, and then increasingly larger layers “decompress” the internal encoding back to the original input. For these networks, it is required that that minimum size is reached, and greedily training this network layer-by-layer means applying only one compression step, and immediately decompressing it again using the reshaping layer—vastly simplifying the “problem” from the EA’s point of view, and missing the point of an autoencoder.

Alleviating this is possible through training the compressing layers as if they were one, and then training the decompressing stages one-by-one. Unfortunately this turns half the network into a large monolithic layer, and the effectiveness of partial evaluations diminishes again.



(A) Training the first linear layer. The layers marked "(d)" are scratch layers that are discarded once training has finished.



(B) Training the last two linear layers. The dataset labeled $x^{(x)}$ refers to the same in fig. A.2a. That is, the output of the "ReLU 1" layer for each value in the dataset $x^{(0)}$.

FIGURE A.2: Training a network layer-by-layer.

.

A.1.2 Conclusion

Partial evaluations are a powerful tool when training complex networks with GOMEA, but multiple fundamental aspects of MLPs limit their applicability. Ways around this, such as artificially reducing the depth of the network, can reduce these limiting effects, but they come at their own expenses and cannot be greedily applied to any possible network.

A.2 Problem-specific linkage models

Next to partial evaluations there is another powerful tool at GOMEA's disposal: explicit parameter linkage. One can already identify multiple obvious links between parameters, such as all weights contributing to the same output node, or all weights coming from the same input node.

The following are possible linkage strategies for a typical linear layer with n inputs and m outputs. Not all strategies are meant to be used together—there is significant overlap between them.

Linkage	**Example**	**Figure**
Inputs	$\{w_i, w_{i+1}, \dots, w_{i+n-1}\}$	fig. C.1
Inputs + biases	$\{w_i, w_{i+1}, \dots, w_{i+n-1}\} \cup \{b_1, b_2, \dots, b_m\}$	fig. C.2
Outputs	$\{w_i, w_{i+n}, \dots, w_{i+mn-n}\}$	fig. C.3
Outputs + bias	$\{w_i, w_{i+n}, \dots, w_{i+mn-n}\} \cup \{b_i\}$	fig. C.4
Weights	$\{w_{11}, w_{12}, \dots, w_{nm}\}$	fig. C.5
Biases	$\{b_1, b_2, \dots, b_m\}$	fig. C.6
Full layer	$\{w_{11}, w_{12}, \dots, w_{nm}\} \cup \{b_1, b_2, \dots, b_m\}$	fig. C.7

A.2.1 Linkage spanning two layers

Linkage spanning layers is not as trivial. Because the layers are fully connected, one weight in a first layer influences all outputs of the second layer. Still, linkage between at least two layers can benefit inter-layer cooperation.

Of the intra-layer linkages discussed before, a few can be combined into reasonable sets: the weights and possibly the bias leading to each output node of the first layer have a direct relationship to the weights leading from the corresponding input node on the second layer. A linkage set like this describes, in essence, the relation from every input node on the first layer to every output node on the second layer, controlled by the output node in the middle of this subgraph.

Another way to see this is as the buildup of a feature detector (the incoming weights) and its contribution to the output of these layers working in tandem (the outgoing weights). However, this is a simplistic view and the behavior of complex networks likely cannot be described in such terms.

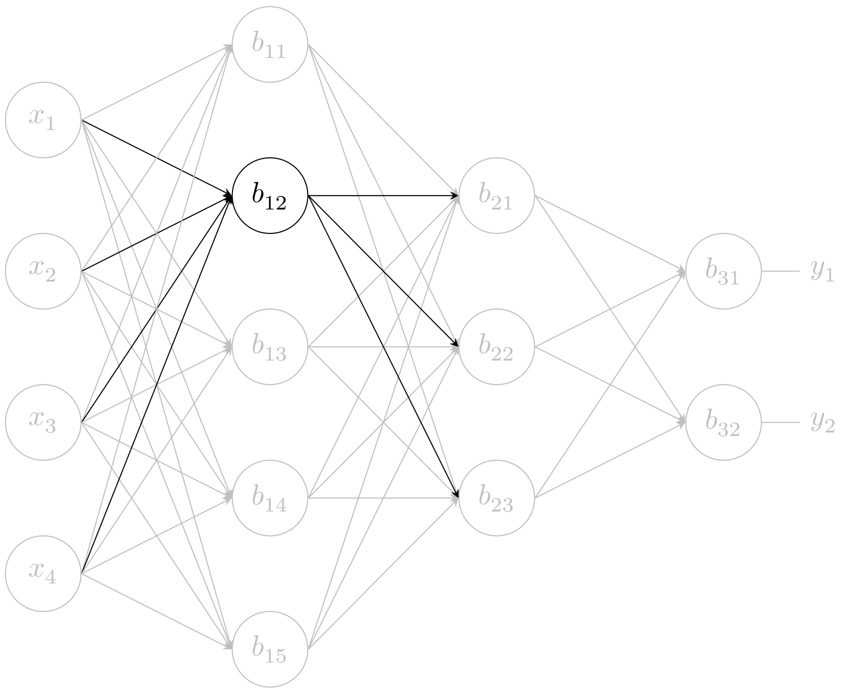


FIGURE A.3: A linkage set covering two layers, anchored around a bias.

Linkage spanning more than two layers

Linkage sets pivoting around a single node like possible in sets of two consecutive layers does not scale to larger networks. In the previous case, linkage described a mapping between all inputs to all outputs of such a set.

Because all layers are fully connected, there is likely little sense in extending linkage sets much further than this. *Some* higher-level linkage is likely to be beneficial, however. Because contributions to the output are fairly distributed throughout the network, one may be tempted to greedily produce power sets of previous linkage sets or taking the union of these. The former case produces an extreme number of subsets and slow down evolution, while the latter starts to approach a fully-covariant strategy instead; neither particularly efficient and probably not very effective.

At this point, because the networks also become very large, a heuristic is necessary. Inspiration can be taken from random linkage graphs. Randomly chosen pairs of lower-level (and therefore smaller) linkage sets can be combined to form cross-layer linkages. These can then be merged again to form even higher-level sets, and the process can be repeated until a single set is formed. For an average network, this makes a more-or-less logarithmic number of subsets, ranging from layer-local to global network linkage, arranged as a mostly balanced tree-like collection of sets.

This graph may still be too large to use effectively, however. A fully covariant linkage set may not be necessary, so the combining step can terminate at some point before, such as a maximum tree height or set size. Similarly, there are many very small sets and these may be too costly to evaluate or of too low impact in such a large network. These “leaves” can be pruned until, again, a desired tree height is attained or only sets of a minimum size remain.

Appendix B

Experiment platforms

Experiments were performed on a variety of platforms (combinations of hardware and software). Experiments of the same type were kept on the same platform to avoid external influences on the performance, most notably where experiments ran under time constraints.

System 1

System 1 ran GOMEA experiments.

CPU Ryzen 9 3900X 12/24 cores (PBO enabled)

RAM 32 GB DDR4-3200 16-18-18-36

OS Debian testing; Linux 5.17

Compiler GCC 10.3.0

System 2

System 2 ran GD experiments.

CPU Ryzen 7 1700 8/16 cores (ECO mode enabled)

RAM 32 GB DDR4-2400 17-17-17-39 ECC

OS Debian 11 (bullseye); Linux 5.10

Python 3.9.2

Pytorch 1.8.1

System 3

System 3 (the only laptop) ran BIPOP-CMA-ES experiments.

CPU Intel i7-4710MQ 4/8 cores

RAM 8 GB DDR3-1600 11-11-11-28, 16 GB DDR3-1600 10-10-10-27

OS Debian unstable; Linux 5.18

Python 3.10.5

Pytorch 1.11.0

Appendix C

Linkage examples

These are examples of single linkage sets according to appendix A.2 generated for the example network in fig. 2.2.

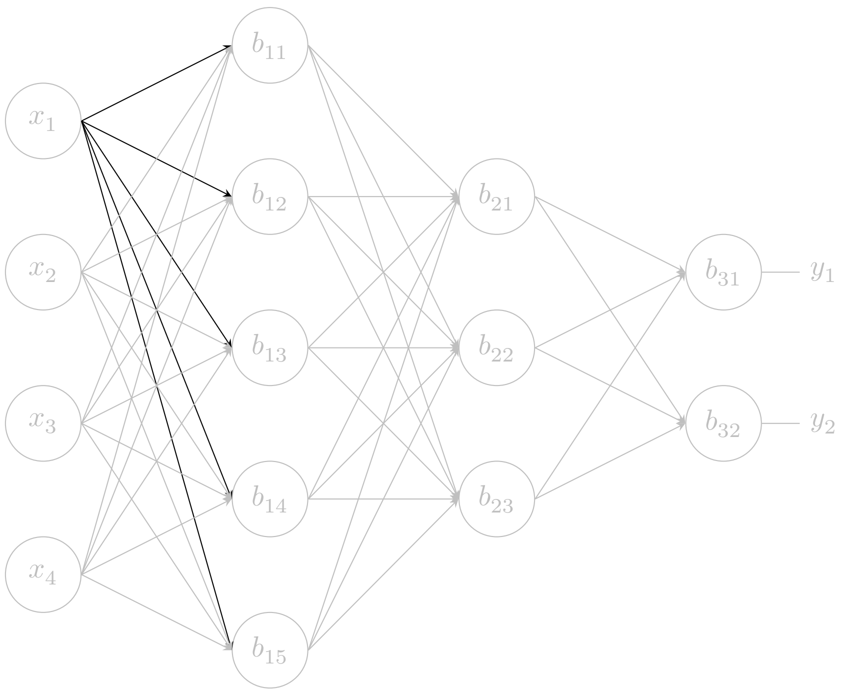


FIGURE C.1: A linkage set covering the first set of input weights on the first layer.

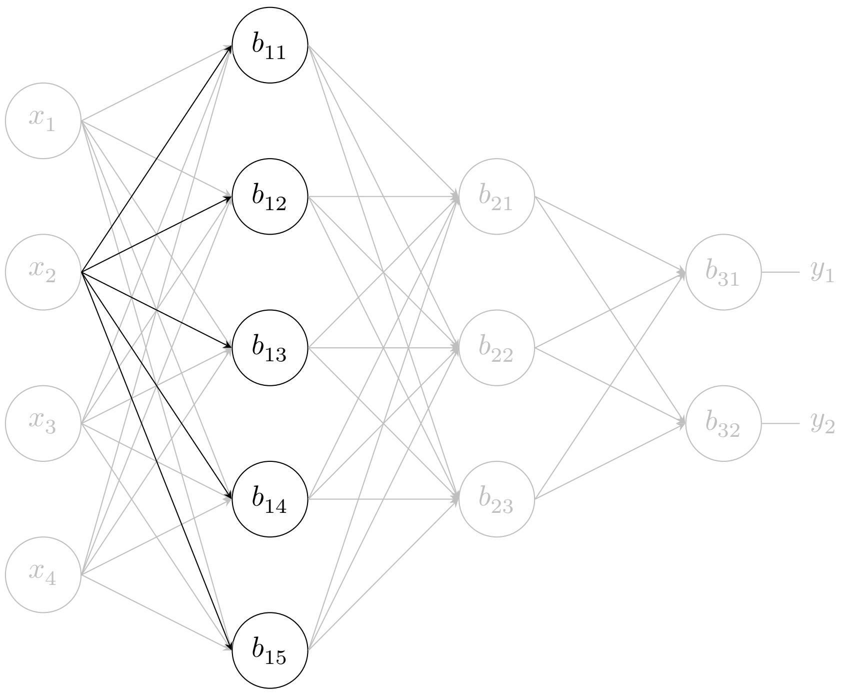


FIGURE C.2: A linkage set covering the second set of input weights and associated biases on the first layer.

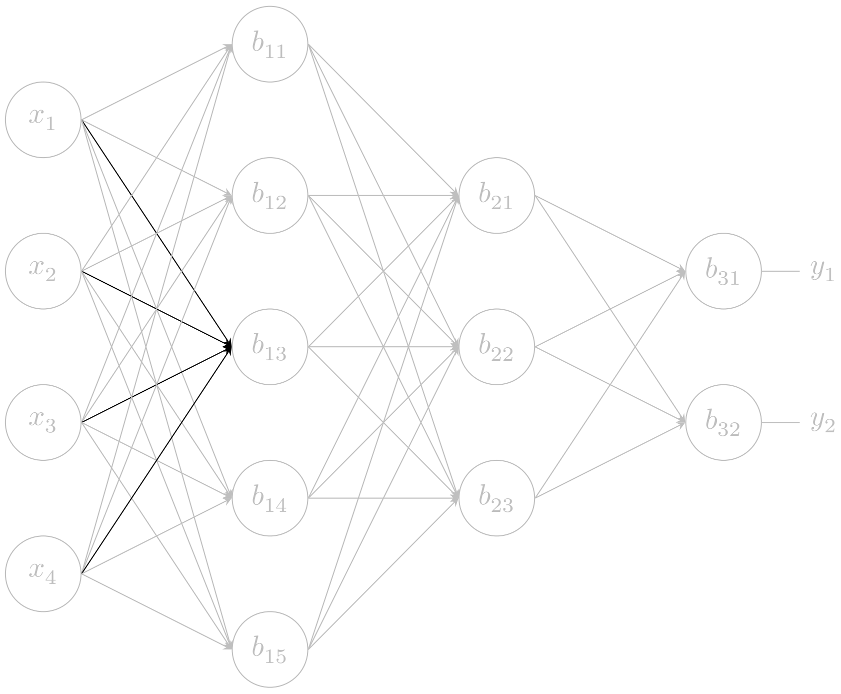


FIGURE C.3: A linkage set covering the third set of output weights on the first layer.

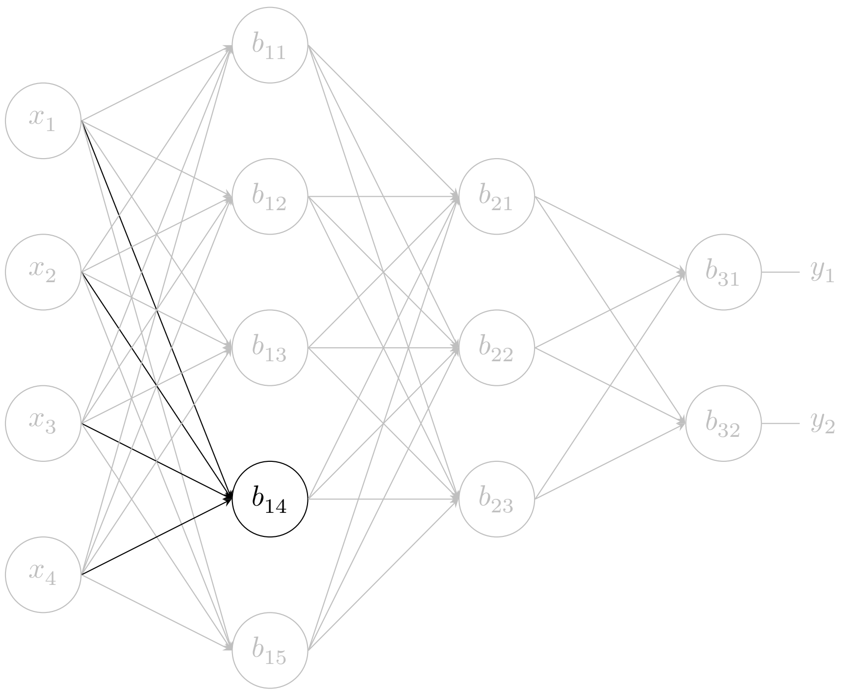


FIGURE C.4: A linkage set covering the fourth set of output weights and the associated bias on the first layer.

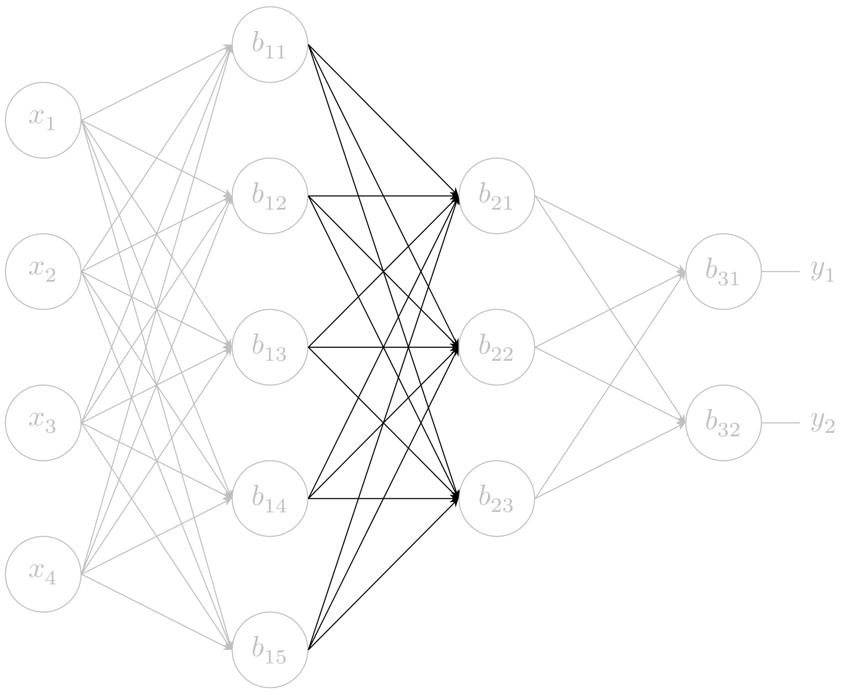


FIGURE C.5: A linkage set covering all weights of the second layer.

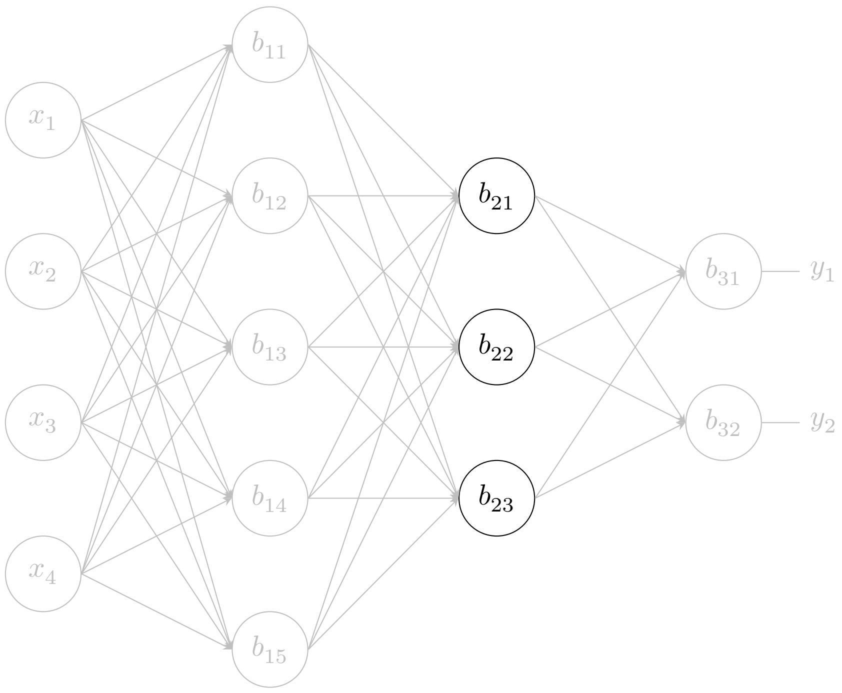


FIGURE C.6: A linkage set covering all biases of the second layer.

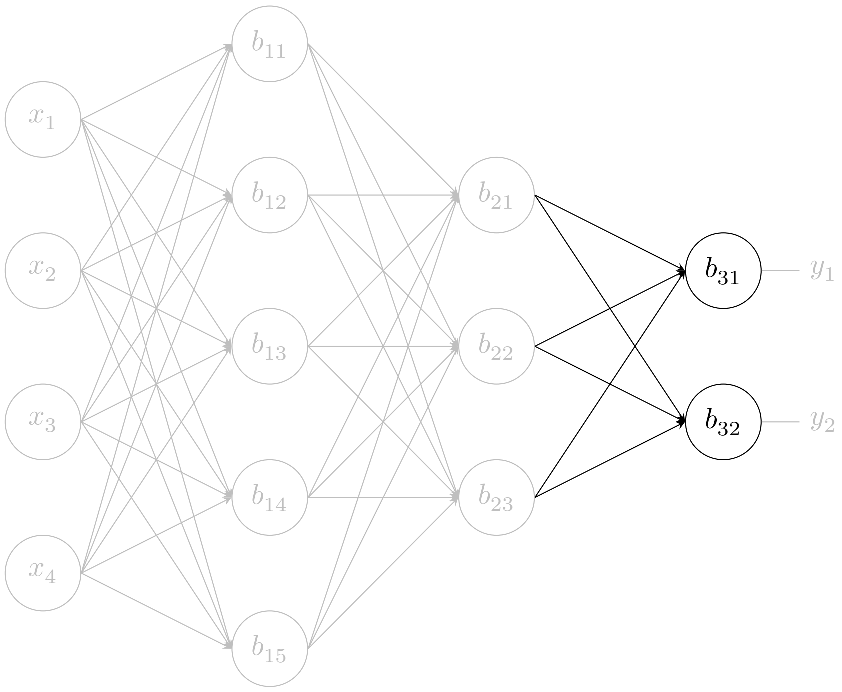


FIGURE C.7: A linkage set covering the entire third layer.

Appendix D

Implementation details

This chapter gives a broad overview of the implementation of “Neuro-GOMEA”. Most classes have been documented using Doxygen in case more detail is desired. Reference implementations are included in the code-base in the form of XorProblem and MnistProblem.

D.1 libGOMEA

The work performed here included a custom neural network implementation that can be hooked into SO- or MO-RV-GOMEA. While these do not contribute scientifically, details on how this was achieved can be helpful in any subsequent work, even if only to verify results.

The codebase is a mix between C11 and C++20. The original RV-GOMEA implementation was almost C89; C99 features were used in a few places. Since this part of the code was fully compatible with C11, the compiler was instructed to compile in C11 mode and some additional code was written in C11.

GOMEA was turned from an executable into a library and problem registration hooks were introduced. In addition to the hardcoded list of problems, libGOMEA also maintains a dynamic list of problems that the user can add problems to (“registration”). These problems are defined as C++20 classes that hold any hyperparameters accepted by GOMEA, but most importantly the number of parameters and initialization ranges, full and partial evaluation functions, and a linkage FOS generator. Many of these are optional, in which case “sensible” values are used. If no partial evaluation function is given, a full evaluation is performed instead every time.

Each problem is solved in its own process; this is due to the implementation’s extensive use of global variables, making parallel executions impossible and state reuse difficult to maintain and therefore risky. The algorithm variant (single- or multi-objective) is selected dynamically depending on the given problem. Currently, only the single-objective variant is guaranteed to work; multi-objective problems require additional modifications to the library.

Further modifications to the GOMEA code include OpenMP support to parallelize genome evaluation, a multithreading-aware RNG (producing deterministic results as long as the hyperparameters and number of threads remain equal), and additional debugging facilities.

```
#include "Problem.hpp"
#include "RuntimeOptions.hpp"
#include "run.hpp"

#include <memory>
#include <utility>
#include <cstdio>

class SphereProblem : public Gomea::Problem {
```

```
private:
    int num_params;

    std::string problem_name = "Sphere";

public:
    explicit SphereProblem(int num_params = 3) : num_params(num_params) {}
    virtual ~SphereProblem() override = default;

    virtual const std::string& get_name() const override {
        return this->problem_name;
    }

    virtual unsigned int get_num_objectives() const override {
        return 1;
    }

    virtual std::pair<double, double> get_param_bounds(unsigned int) const override {
        return std::make_pair<double, double>(-1., 1.);
    }

    virtual Gomea::Hyperparams get_default_hyperparams() const override {
        auto defaults = Problem::get_default_hyperparams();
        defaults.num_params = this->num_params;
        return defaults;
    }

    virtual void evaluate(
        Gomea::Individual&,
        const double* parameters,
        double* objective_value_result,
        double* constraint_value_result,
        unsigned int,
        long
    ) override {
        double loss = 0;
        for (int i = 0; i < this->num_params; ++i) {
            loss += parameters[i] * parameters[i];
        }
        *objective_value_result = loss;
        *constraint_value_result = 0;
    }

    virtual void evaluate_partial(
        Gomea::Individual&,
        const double* parameters,
        unsigned int number_of_touched_parameters,
        const unsigned int* touched_parameters_indices,
        const double* parameters_before,
        const double* objective_values_before,
        double constraint_value_before,
        double* objective_value_result,
        double* constraint_value_result,
        unsigned int,
        long
    ) override {
        double loss = *objective_values_before;
        for (long idx_i = 0; idx_i < number_of_touched_parameters; ++idx_i) {
            loss -= parameters_before[idx_i] * parameters_before[idx_i];
            loss += parameters[touched_parameters_indices[idx_i]] * parameters[touched_parameters_indices[idx_i]];
        }
        *objective_value_result = loss;
        *constraint_value_result = constraint_value_before;
    }
};

int main(void) {
    Gomea::Hyperparams hyperparams;
    Gomea::RuntimeOptions runtime_options;

    hyperparams.linkage_model = 1u;
    runtime_options.verbose = true;
    runtime_options.write_stats_each_gen = true;
    runtime_options.target_value = 1e-300;

    auto problem = std::make_shared<SphereProblem>();
    const auto problem_idx = Gomea::register_problem(problem);
    const auto pop = Gomea::run_problem(problem_idx, *problem, hyperparams, runtime_options);

    const auto& indiv = pop.front();
    const auto& p = indiv.get_parameters();

    for (const auto v : p) {
        printf("%g ", v);
    }
    puts("");
}
```

Source Code D.1: Example implementation of the n-dimensional sphere problem also used in section 2.3.

D.2 mininn

Also included is *mininn*, a small implementation of basic neural network primitives (the linear layer, activation functions, and loss functions), built around the Eigen library. A `Sequential` type is also available, which groups layers into one virtual layer.

Mininn classes are fully templated and therefore extremely flexible: as long as types can be unified, any combination of layers is possible. Theoretically, a sequence of layers could be compacted into a single Eigen expression, though some configurations (such as reshaping) require concrete values and so often intermediate evaluations are performed. Care must be taken when programming with these classes, as small mistakes often generate enormous compiler diagnostics, or—worse—access violations occurring very deep in library code.

The sequential layer is a complex meta-layer that combines the behavior of its constituent layers. It is a convenient way to interact with a complex network, though the typical C++ template caveats (long compilation times, difficult-to-comprehend errors, et cetera) apply.

```
#include "layer/all.hpp"

#include <cassert>

int main(void) {
    Eigen::Matrix2d lin1_weight;
    lin1_weight <<
        -1, -1,
         1,  1;
    Eigen::Vector2d lin1_bias = {1, -1};
    Mininn::Layer::Linear lin1(2, 2);

    Mininn::Layer::ReLU relu;

    Eigen::Matrix2d lin2_weight;
    lin2_weight <<
         1,  1,
        -1, -1;
    Eigen::Vector2d lin2_bias = {0, 1};
    Mininn::Layer::Linear lin2(2, 2);

    Eigen::Vector2d x = {0, 0};
    Eigen::Vector2d yt = {1, 0};
    const auto y = lin2(relu(lin1(x, lin1_weight, lin1_bias)), lin2_weight, lin2_bias);
    assert(y == yt);
}
```

Source Code D.2: Testing the evaluation of a two-input, two-hidden, ReLU-activated XOR network using mininn.

D.3 NNProblem

The synthesis of mininn and libGOMEA is the NNProblem class and its constituent NNSubproblem. An NNProblem is similar to a mininn Sequential—in that it is a sequence of NN layers—that can be "trained".

NNProblem implements the algorithms outlined in appendix A.1.1 (except early stopping) and appendix A.2 (but not two-layer linkage; a bounded random tree can be created instead after intra-layer linkages have been generated). Once training starts, layers that can be trained (i.e., have parameters) are trained, and those that can not (such as flattening or activation functions) are evaluated immediately. This process continues until all layers have been trained, after which the parameters describing the full network are returned.

NNProblem extends a libGOMEA Problem, but it cannot be used as such; it is used to supply default hyperparameters to automatically generated subproblems. Often, values support multiple modes, usually integers and percentages. Percentages scale the number of parameters in the problem. This way, it is possible to always use random linkage where each group contains 2% of the total parameters, regardless of how many parameters there are, for example.

Appendix E

Found GD and GDCR hyperparameters

Table E.1: Hyperparameters found for GD and GDCR.

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GD	Sigmoid	No	1	2	7	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	No	1	2	6	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	No	1	2	5	0.032	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	1	2	4	0.0032	0	0.81	N/A	99.6 %
GD	Sigmoid	No	1	2	3	0.01	0.09	0.99	N/A	99.6 %
GD	Sigmoid	No	1	2	2	0.1	0.54	0.99	N/A	94.0 %
GD	Sigmoid	No	1	2	1	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	No	1	3	10	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	No	1	3	9	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	No	1	3	8	0.032	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	1	3	7	0.0032	0	0.81	N/A	100.0 %
GD	Sigmoid	No	1	3	6	0.001	0.18	0.9	N/A	100.0 %
GD	Sigmoid	No	1	3	5	0.0032	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	1	3	4	0.032	0.54	0.9	N/A	99.2 %
GD	Sigmoid	No	1	3	3	0.001	0.72	0.99	N/A	100.0 %
GD	Sigmoid	No	1	3	2	0.032	0.81	0.99	N/A	90.1 %
GD	Sigmoid	No	1	3	1	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	No	1	4	13	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	No	1	4	12	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	No	1	4	11	0.032	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	1	4	10	0.032	0.81	0.9	N/A	99.9 %
GD	Sigmoid	No	1	4	9	0.001	0.18	0.9	N/A	98.8 %
GD	Sigmoid	No	1	4	8	0.0032	0.9	0.999	N/A	99.9 %
GD	Sigmoid	No	1	4	7	0.032	0.81	0.99	N/A	98.0 %
GD	Sigmoid	No	1	4	6	0.0032	0.9	0.99	N/A	99.9 %
GD	Sigmoid	No	1	4	5	0.0032	0.99	0.99	N/A	98.7 %
GD	Sigmoid	No	1	4	4	0.0032	0.72	0.99	N/A	93.0 %
GD	Sigmoid	No	1	4	3	0.01	0.72	0.99	N/A	93.3 %
GD	Sigmoid	No	1	4	2	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	No	1	5	16	0.01	0.99	0.999	N/A	100.0 %
GD	Sigmoid	No	1	5	15	0.0032	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	1	5	14	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	1	5	13	0.0032	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	1	5	12	0.032	0.9	0.999	N/A	99.9 %
GD	Sigmoid	No	1	5	11	0.032	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	1	5	10	0.01	0.45	0.999	N/A	98.1 %
GD	Sigmoid	No	1	5	9	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	1	5	8	0.01	0.9	0.999	N/A	99.6 %
GD	Sigmoid	No	1	5	7	0.0032	0.9	0.99	N/A	99.2 %
GD	Sigmoid	No	1	5	6	0.01	0.9	0.99	N/A	96.3 %
GD	Sigmoid	No	1	5	5	0.0032	0.9	0.99	N/A	92.5 %
GD	Sigmoid	No	1	5	4	0.0032	0.9	0.99	N/A	78.9 %
GD	Sigmoid	No	1	5	3	0.0032	0.9	0.99	N/A	52.8 %
GD	Sigmoid	No	1	5	2	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	No	1	6	19	0.032	0.81	0.999	N/A	100.0 %
GD	Sigmoid	No	1	6	18	0.01	0.99	0.999	N/A	98.8 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GD	Sigmoid	No	1	6	17	0.001	0.99	0.99	N/A	99.3 %
GD	Sigmoid	No	1	6	16	0.032	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	1	6	15	0.032	0.9	0.999	N/A	99.4 %
GD	Sigmoid	No	1	6	14	0.032	0.9	0.999	N/A	99.5 %
GD	Sigmoid	No	1	6	13	0.032	0.9	0.999	N/A	98.7 %
GD	Sigmoid	No	1	6	12	0.01	0.9	0.99	N/A	94.9 %
GD	Sigmoid	No	1	6	11	0.032	0.9	0.999	N/A	97.3 %
GD	Sigmoid	No	1	6	10	0.032	0.9	0.999	N/A	96.4 %
GD	Sigmoid	No	1	6	9	0.032	0.9	0.999	N/A	93.6 %
GD	Sigmoid	No	1	6	8	0.032	0.9	0.999	N/A	89.9 %
GD	Sigmoid	No	1	6	7	0.032	0.9	0.999	N/A	78.5 %
GD	Sigmoid	No	1	6	6	0.032	0.9	0.99	N/A	60.0 %
GD	Sigmoid	No	1	6	5	0.01	0.9	0.99	N/A	39.4 %
GD	Sigmoid	No	1	6	4	0.01	0.9	0.99	N/A	20.8 %
GD	Sigmoid	No	1	6	3	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	Yes	1	2	7	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	2	6	0.01	0.99	0.99	N/A	99.9 %
GD	Sigmoid	Yes	1	2	5	0.032	0.45	0.9	N/A	99.6 %
GD	Sigmoid	Yes	1	2	4	0.01	0.54	0.99	N/A	98.0 %
GD	Sigmoid	Yes	1	2	3	0.032	0	0.999	N/A	94.0 %
GD	Sigmoid	Yes	1	2	2	0.032	0.36	0.999	N/A	66.2 %
GD	Sigmoid	Yes	1	2	1	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	Yes	1	3	10	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	3	9	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	3	8	0.032	0.9	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	3	7	0.0032	0	0.81	N/A	100.0 %
GD	Sigmoid	Yes	1	3	6	0.032	0.9	0.9	N/A	100.0 %
GD	Sigmoid	Yes	1	3	5	0.01	0.36	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	3	4	0.0032	0.54	0.9	N/A	99.8 %
GD	Sigmoid	Yes	1	3	3	0.1	0.54	0.99	N/A	99.3 %
GD	Sigmoid	Yes	1	3	2	0.1	0.9	0.99	N/A	83.4 %
GD	Sigmoid	Yes	1	3	1	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	Yes	1	4	13	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	4	12	0.01	0.99	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	4	11	0.032	0.9	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	4	10	0.01	0.09	0.999	N/A	99.8 %
GD	Sigmoid	Yes	1	4	9	0.032	0.9	0.9	N/A	99.9 %
GD	Sigmoid	Yes	1	4	8	0.01	0.36	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	4	7	0.032	0.81	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	4	6	0.032	0.9	0.99	N/A	99.4 %
GD	Sigmoid	Yes	1	4	5	0.032	0.9	0.99	N/A	98.1 %
GD	Sigmoid	Yes	1	4	4	0.1	0.9	0.99	N/A	99.0 %
GD	Sigmoid	Yes	1	4	3	0.1	0	0.99	N/A	87.2 %
GD	Sigmoid	Yes	1	4	2	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	Yes	1	5	16	0.0032	0.63	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	5	15	0.001	0.72	0.27	N/A	99.4 %
GD	Sigmoid	Yes	1	5	14	0.0032	0.99	0.99	N/A	99.7 %
GD	Sigmoid	Yes	1	5	13	0.001	0.63	0.9	N/A	99.6 %
GD	Sigmoid	Yes	1	5	12	0.032	0.9	0.999	N/A	98.7 %
GD	Sigmoid	Yes	1	5	11	0.01	0.09	0.99	N/A	99.5 %
GD	Sigmoid	Yes	1	5	10	0.01	0.72	0.99	N/A	99.9 %
GD	Sigmoid	Yes	1	5	9	0.0032	0.9	0.99	N/A	99.4 %
GD	Sigmoid	Yes	1	5	8	0.0032	0.9	0.99	N/A	98.6 %
GD	Sigmoid	Yes	1	5	7	0.0032	0.72	0.99	N/A	92.2 %
GD	Sigmoid	Yes	1	5	6	0.01	0.72	0.99	N/A	84.0 %
GD	Sigmoid	Yes	1	5	5	0.032	0.81	0.99	N/A	69.2 %
GD	Sigmoid	Yes	1	5	4	0.032	0.81	0.99	N/A	58.3 %
GD	Sigmoid	Yes	1	5	3	0.032	0.9	0.99	N/A	30.0 %
GD	Sigmoid	Yes	1	5	2	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	Yes	1	6	19	0.032	0.54	0.99	N/A	100.0 %
GD	Sigmoid	Yes	1	6	18	0.0032	0.9	0.9	N/A	98.7 %
GD	Sigmoid	Yes	1	6	17	0.032	0.27	0.99	N/A	99.9 %
GD	Sigmoid	Yes	1	6	16	0.032	0.27	0.99	N/A	99.7 %
GD	Sigmoid	Yes	1	6	15	0.032	0.54	0.99	N/A	99.0 %
GD	Sigmoid	Yes	1	6	14	0.032	0.27	0.99	N/A	99.8 %
GD	Sigmoid	Yes	1	6	13	0.032	0.27	0.99	N/A	99.8 %
GD	Sigmoid	Yes	1	6	12	0.032	0.09	0.99	N/A	96.4 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GD	Sigmoid	Yes	1	6	11	0.032	0.27	0.99	N/A	95.9 %
GD	Sigmoid	Yes	1	6	10	0.032	0.45	0.99	N/A	91.8 %
GD	Sigmoid	Yes	1	6	9	0.032	0.54	0.99	N/A	80.7 %
GD	Sigmoid	Yes	1	6	8	0.032	0.54	0.99	N/A	74.7 %
GD	Sigmoid	Yes	1	6	7	0.032	0.9	0.99	N/A	72.0 %
GD	Sigmoid	Yes	1	6	6	0.032	0.9	0.99	N/A	68.8 %
GD	Sigmoid	Yes	1	6	5	0.032	0.9	0.99	N/A	58.8 %
GD	Sigmoid	Yes	1	6	4	0.032	0.81	0.99	N/A	15.1 %
GD	Sigmoid	Yes	1	6	3	0.001	0.99	0.999	N/A	0.0 %
GD	ReLU	No	1	2	8	0.01	0.36	0.99	N/A	100.0 %
GD	ReLU	No	1	2	7	0.0032	0.45	0.9	N/A	95.5 %
GD	ReLU	No	1	2	7	0.032	0.54	0.99	N/A	95.5 %
GD	ReLU	No	1	2	6	0.01	0.18	0.9	N/A	95.9 %
GD	ReLU	No	1	2	5	0.01	0.27	0.9	N/A	91.1 %
GD	ReLU	No	1	2	4	0.01	0.36	0.9	N/A	80.5 %
GD	ReLU	No	1	2	3	0.01	0.63	0.81	N/A	59.4 %
GD	ReLU	No	1	2	2	0.01	0.18	0.81	N/A	30.9 %
GD	ReLU	No	1	2	1	0.001	0.9	0.999	N/A	0.0 %
GD	ReLU	No	1	3	10	0.032	0.18	0.9	N/A	100.0 %
GD	ReLU	No	1	3	9	0.01	0	0.999	N/A	97.3 %
GD	ReLU	No	1	3	8	0.032	0.45	0.99	N/A	95.0 %
GD	ReLU	No	1	3	7	0.032	0.27	0.999	N/A	88.5 %
GD	ReLU	No	1	3	6	0.032	0.27	0.99	N/A	82.4 %
GD	ReLU	No	1	3	5	0.032	0.18	0.9	N/A	67.6 %
GD	ReLU	No	1	3	4	0.032	0.27	0.99	N/A	41.5 %
GD	ReLU	No	1	3	3	0.01	0.09	0.9	N/A	16.6 %
GD	ReLU	No	1	3	2	0.001	0.9	0.999	N/A	0.0 %
GD	ReLU	No	1	4	13	0.032	0.54	0.99	N/A	100.0 %
GD	ReLU	No	1	4	12	0.032	0.18	0.9	N/A	99.0 %
GD	ReLU	No	1	4	11	0.032	0.45	0.54	N/A	95.8 %
GD	ReLU	No	1	4	10	0.032	0.45	0.9	N/A	92.9 %
GD	ReLU	No	1	4	9	0.032	0.36	0.99	N/A	90.9 %
GD	ReLU	No	1	4	8	0.032	0.27	0.81	N/A	79.0 %
GD	ReLU	No	1	4	7	0.032	0.36	0.99	N/A	69.8 %
GD	ReLU	No	1	4	6	0.032	0.36	0.81	N/A	46.8 %
GD	ReLU	No	1	4	5	0.032	0.54	0.9	N/A	27.3 %
GD	ReLU	No	1	4	4	0.032	0.36	0.99	N/A	17.3 %
GD	ReLU	No	1	4	3	0.032	0.45	0.9	N/A	7.3 %
GD	ReLU	No	1	4	2	0.001	0.9	0.999	N/A	0.0 %
GD	ReLU	No	1	5	17	0.032	0.45	0.9	N/A	100.0 %
GD	ReLU	No	1	5	16	0.032	0.36	0.9	N/A	95.5 %
GD	ReLU	No	1	5	16	0.032	0.54	0.99	N/A	95.5 %
GD	ReLU	No	1	5	15	0.032	0.54	0.9	N/A	95.8 %
GD	ReLU	No	1	5	14	0.032	0.45	0.99	N/A	93.3 %
GD	ReLU	No	1	5	13	0.032	0.63	0.99	N/A	86.1 %
GD	ReLU	No	1	5	12	0.032	0.45	0.99	N/A	76.2 %
GD	ReLU	No	1	5	11	0.032	0.63	0.99	N/A	66.8 %
GD	ReLU	No	1	5	10	0.032	0.45	0.99	N/A	52.7 %
GD	ReLU	No	1	5	9	0.032	0.54	0.99	N/A	40.4 %
GD	ReLU	No	1	5	8	0.032	0.54	0.99	N/A	26.6 %
GD	ReLU	No	1	5	7	0.032	0.27	0.9	N/A	10.4 %
GD	ReLU	No	1	5	6	0.032	0.45	0.72	N/A	8.8 %
GD	ReLU	No	1	5	5	0.032	0.54	0.99	N/A	4.0 %
GD	ReLU	No	1	5	4	0.032	0.36	0.9	N/A	1.1 %
GD	ReLU	No	1	5	3	0.001	0.9	0.999	N/A	0.0 %
GD	ReLU	No	1	6	25	0.01	0.54	0.99	N/A	100.0 %
GD	ReLU	No	1	6	24	0.01	0.63	0.99	N/A	97.0 %
GD	ReLU	No	1	6	24	0.01	0.81	0.999	N/A	97.0 %
GD	ReLU	No	1	6	23	0.01	0.72	0.99	N/A	96.5 %
GD	ReLU	No	1	6	23	0.032	0.63	0.99	N/A	96.5 %
GD	ReLU	No	1	6	22	0.01	0.72	0.99	N/A	93.6 %
GD	ReLU	No	1	6	21	0.01	0.63	0.9	N/A	87.4 %
GD	ReLU	No	1	6	20	0.032	0.72	0.99	N/A	85.5 %
GD	ReLU	No	1	6	19	0.01	0.72	0.99	N/A	84.8 %
GD	ReLU	No	1	6	18	0.01	0.72	0.99	N/A	74.3 %
GD	ReLU	No	1	6	17	0.0032	0.72	0.999	N/A	63.5 %
GD	ReLU	No	1	6	16	0.01	0.81	0.999	N/A	60.5 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GD	ReLU	No	1	6	15	0.0032	0.81	0.999	N/A	47.6 %
GD	ReLU	No	1	6	14	0.01	0.81	0.999	N/A	41.6 %
GD	ReLU	No	1	6	13	0.0032	0.9	0.999	N/A	35.7 %
GD	ReLU	No	1	6	12	0.01	0.81	0.99	N/A	24.4 %
GD	ReLU	No	1	6	11	0.0032	0.81	0.999	N/A	11.9 %
GD	ReLU	No	1	6	10	0.0032	0.9	0.999	N/A	11.2 %
GD	ReLU	No	1	6	9	0.01	0.45	0.99	N/A	3.4 %
GD	ReLU	No	1	6	8	0.032	0.54	0.99	N/A	4.7 %
GD	ReLU	No	1	6	7	0.032	0.81	0.99	N/A	1.2 %
GD	ReLU	No	1	6	6	0.0032	0.63	0.81	N/A	0.1 %
GD	ReLU	Yes	1	2	9	0.032	0.54	0.45	N/A	39.0 %
GD	ReLU	Yes	1	2	8	0.1	0.54	0.18	N/A	43.1 %
GD	ReLU	Yes	1	2	7	0.1	0.9	0.9	N/A	39.9 %
GD	ReLU	Yes	1	2	6	0.1	0.81	0.63	N/A	33.7 %
GD	ReLU	Yes	1	2	5	0.1	0.54	0.9	N/A	30.1 %
GD	ReLU	Yes	1	2	4	0.01	0.81	0.36	N/A	28.8 %
GD	ReLU	Yes	1	2	3	0.1	0.27	0.9	N/A	19.0 %
GD	ReLU	Yes	1	2	2	0.032	0.36	0.99	N/A	9.5 %
GD	ReLU	Yes	1	2	1	0.001	0.9	0.999	N/A	0.0 %
GD	ReLU	Yes	1	3	13	0.1	0.54	0.72	N/A	62.0 %
GD	ReLU	Yes	1	3	12	0.032	0.54	0.18	N/A	57.0 %
GD	ReLU	Yes	1	3	12	0.1	0.54	0.9	N/A	57.0 %
GD	ReLU	Yes	1	3	11	0.032	0	0.999	N/A	53.1 %
GD	ReLU	Yes	1	3	11	0.1	0.27	0.54	N/A	53.1 %
GD	ReLU	Yes	1	3	10	0.032	0.27	0.99	N/A	50.7 %
GD	ReLU	Yes	1	3	10	0.032	0.72	0.99	N/A	50.7 %
GD	ReLU	Yes	1	3	9	0.032	0.09	0.99	N/A	50.2 %
GD	ReLU	Yes	1	3	8	0.032	0.81	0.99	N/A	44.9 %
GD	ReLU	Yes	1	3	7	0.032	0.36	0.99	N/A	42.4 %
GD	ReLU	Yes	1	3	6	0.032	0.09	0.99	N/A	38.5 %
GD	ReLU	Yes	1	3	5	0.032	0	0.99	N/A	27.6 %
GD	ReLU	Yes	1	3	4	0.032	0.45	0.99	N/A	17.4 %
GD	ReLU	Yes	1	3	3	0.0032	0.09	0.999	N/A	4.6 %
GD	ReLU	Yes	1	3	2	0.001	0.9	0.999	N/A	0.0 %
GD	ReLU	Yes	1	4	17	0.032	0.45	0.36	N/A	81.0 %
GD	ReLU	Yes	1	4	16	0.032	0.09	0.99	N/A	70.3 %
GD	ReLU	Yes	1	4	15	0.032	0.72	0.99	N/A	67.9 %
GD	ReLU	Yes	1	4	14	0.01	0.27	0.18	N/A	64.2 %
GD	ReLU	Yes	1	4	14	0.01	0.45	0.36	N/A	64.2 %
GD	ReLU	Yes	1	4	13	0.032	0.45	0.81	N/A	65.6 %
GD	ReLU	Yes	1	4	13	0.032	0.63	0.36	N/A	65.6 %
GD	ReLU	Yes	1	4	12	0.032	0.09	0.9	N/A	65.1 %
GD	ReLU	Yes	1	4	11	0.032	0.27	0.99	N/A	61.1 %
GD	ReLU	Yes	1	4	10	0.032	0.36	0.99	N/A	54.8 %
GD	ReLU	Yes	1	4	9	0.032	0.72	0.99	N/A	45.5 %
GD	ReLU	Yes	1	4	8	0.01	0	0.99	N/A	42.0 %
GD	ReLU	Yes	1	4	7	0.032	0.09	0.81	N/A	34.6 %
GD	ReLU	Yes	1	4	6	0.032	0.09	0.63	N/A	23.0 %
GD	ReLU	Yes	1	4	5	0.01	0.18	0.99	N/A	10.6 %
GD	ReLU	Yes	1	4	4	0.0032	0.63	0.54	N/A	4.2 %
GD	ReLU	Yes	1	4	3	0.0032	0.9	0.9	N/A	1.6 %
GD	ReLU	Yes	1	4	2	0.001	0.9	0.999	N/A	0.0 %
GD	ReLU	Yes	1	5	21	0.01	0.36	0.54	N/A	87.0 %
GD	ReLU	Yes	1	5	20	0.01	0.27	0.99	N/A	79.5 %
GD	ReLU	Yes	1	5	19	0.01	0	0.999	N/A	78.6 %
GD	ReLU	Yes	1	5	19	0.01	0.27	0.45	N/A	78.6 %
GD	ReLU	Yes	1	5	18	0.01	0.72	0.72	N/A	72.1 %
GD	ReLU	Yes	1	5	17	0.01	0.18	0.63	N/A	73.9 %
GD	ReLU	Yes	1	5	17	0.01	0.27	0.99	N/A	73.9 %
GD	ReLU	Yes	1	5	16	0.032	0.72	0.99	N/A	72.1 %
GD	ReLU	Yes	1	5	15	0.01	0.09	0.99	N/A	71.2 %
GD	ReLU	Yes	1	5	14	0.01	0.45	0.999	N/A	64.3 %
GD	ReLU	Yes	1	5	13	0.01	0.09	0.99	N/A	58.5 %
GD	ReLU	Yes	1	5	12	0.032	0.54	0.81	N/A	52.8 %
GD	ReLU	Yes	1	5	11	0.01	0.09	0.999	N/A	40.6 %
GD	ReLU	Yes	1	5	10	0.032	0.54	0.99	N/A	31.9 %
GD	ReLU	Yes	1	5	9	0.01	0.36	0.999	N/A	24.3 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GD	ReLU	Yes	1	5	8	0.01	0.45	0.99	N/A	13.6 %
GD	ReLU	Yes	1	5	7	0.01	0.09	0.9	N/A	5.4 %
GD	ReLU	Yes	1	5	6	0.001	0.54	0.63	N/A	2.1 %
GD	ReLU	Yes	1	5	5	0.032	0.36	0.54	N/A	1.7 %
GD	ReLU	Yes	1	5	4	0.0032	0.99	0.999	N/A	0.2 %
GD	ReLU	Yes	1	6	25	0.01	0.63	0.999	N/A	88.0 %
GD	ReLU	Yes	1	6	24	0.01	0.27	0.81	N/A	82.4 %
GD	ReLU	Yes	1	6	23	0.01	0.09	0.99	N/A	76.0 %
GD	ReLU	Yes	1	6	22	0.01	0.54	0.9	N/A	77.9 %
GD	ReLU	Yes	1	6	22	0.01	0.81	0.999	N/A	77.9 %
GD	ReLU	Yes	1	6	21	0.01	0.54	0.99	N/A	77.7 %
GD	ReLU	Yes	1	6	20	0.01	0.72	0.999	N/A	72.9 %
GD	ReLU	Yes	1	6	19	0.01	0.63	0.9	N/A	66.9 %
GD	ReLU	Yes	1	6	18	0.001	0.9	0.999	N/A	58.9 %
GD	ReLU	Yes	1	6	17	0.01	0.45	0.999	N/A	55.8 %
GD	ReLU	Yes	1	6	16	0.0032	0.9	0.99	N/A	47.1 %
GD	ReLU	Yes	1	6	15	0.01	0.81	0.99	N/A	40.8 %
GD	ReLU	Yes	1	6	14	0.01	0.72	0.999	N/A	34.8 %
GD	ReLU	Yes	1	6	13	0.0032	0.81	0.999	N/A	26.1 %
GD	ReLU	Yes	1	6	12	0.0032	0.9	0.999	N/A	20.1 %
GD	ReLU	Yes	1	6	11	0.01	0.72	0.99	N/A	11.5 %
GD	ReLU	Yes	1	6	10	0.01	0.81	0.999	N/A	7.5 %
GD	ReLU	Yes	1	6	9	0.0032	0.81	0.99	N/A	4.1 %
GD	ReLU	Yes	1	6	8	0.0032	0.9	0.999	N/A	2.0 %
GD	ReLU	Yes	1	6	7	0.01	0.9	0.99	N/A	0.6 %
GD	Heaviside	No	1	2	9	0.001	0.9	0.999	N/A	63.0 %
GD	Heaviside	No	1	2	8	0.001	0.81	0.99	N/A	58.4 %
GD	Heaviside	No	1	2	8	0.001	0.9	0.999	N/A	58.4 %
GD	Heaviside	No	1	2	7	0.001	0.9	0.999	N/A	49.3 %
GD	Heaviside	No	1	2	6	0.001	0.9	0.999	N/A	36.9 %
GD	Heaviside	No	1	2	5	0.001	0.9	0.999	N/A	30.4 %
GD	Heaviside	No	1	2	4	0.001	0.9	0.999	N/A	17.7 %
GD	Heaviside	No	1	2	3	0.001	0.9	0.999	N/A	8.0 %
GD	Heaviside	No	1	2	2	0.001	0.9	0.999	N/A	1.5 %
GD	Heaviside	No	1	2	1	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	3	13	0.001	0.9	0.999	N/A	46.0 %
GD	Heaviside	No	1	3	12	0.001	0.9	0.999	N/A	39.7 %
GD	Heaviside	No	1	3	11	0.001	0.81	0.99	N/A	29.2 %
GD	Heaviside	No	1	3	11	0.001	0.9	0.999	N/A	29.2 %
GD	Heaviside	No	1	3	10	0.001	0.9	0.999	N/A	21.4 %
GD	Heaviside	No	1	3	9	0.001	0.9	0.999	N/A	14.8 %
GD	Heaviside	No	1	3	8	0.001	0.9	0.999	N/A	7.9 %
GD	Heaviside	No	1	3	7	0.001	0.9	0.999	N/A	4.1 %
GD	Heaviside	No	1	3	6	0.001	0.9	0.999	N/A	0.9 %
GD	Heaviside	No	1	4	17	0.001	0.9	0.99	N/A	4.0 %
GD	Heaviside	No	1	4	16	0.001	0.9	0.99	N/A	4.0 %
GD	Heaviside	No	1	4	15	0.001	0.9	0.999	N/A	1.2 %
GD	Heaviside	No	1	4	14	0.001	0.81	0.99	N/A	0.5 %
GD	Heaviside	No	1	4	14	0.001	0.9	0.999	N/A	0.5 %
GD	Heaviside	No	1	4	13	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	5	21	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	5	20	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	5	19	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	5	18	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	5	17	0.001	0.81	0.99	N/A	0.0 %
GD	Heaviside	No	1	5	16	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	6	25	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	6	24	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	6	23	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	6	22	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	6	21	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	No	1	6	20	0.001	0.81	0.99	N/A	0.0 %
GD	Heaviside	No	1	6	19	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	2	9	0.001	0.9	0.999	N/A	1.0 %
GD	Heaviside	Yes	1	2	8	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	2	7	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	3	13	0.001	0.9	0.999	N/A	0.0 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GD	Heaviside	Yes	1	3	12	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	3	11	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	3	10	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	4	17	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	4	16	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	4	15	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	4	14	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	4	13	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	5	21	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	5	20	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	5	19	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	5	18	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	5	17	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	5	16	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	6	25	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	6	24	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	6	23	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	6	22	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	6	21	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	6	20	0.001	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	1	6	19	0.001	0.9	0.999	N/A	0.0 %
GD	Sigmoid	No	120.0	2	7	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	2	7	0.01	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	No	120.0	2	6	0.01	0.36	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	2	5	0.032	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	No	120.0	2	4	0.01	0.27	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	2	3	0.032	0.36	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	2	2	0.0032	0.9	0.99	N/A	95.7 %
GD	Sigmoid	No	120.0	2	1	0.01	0.9	0.999	N/A	0.0 %
GD	Sigmoid	No	120.0	3	10	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	3	10	0.01	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	No	120.0	3	9	0.01	0.36	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	3	8	0.032	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	No	120.0	3	7	0.01	0.27	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	3	6	0.032	0.36	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	3	5	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	3	4	0.032	0.9	0.999	N/A	95.7 %
GD	Sigmoid	No	120.0	3	3	0.0032	0.9	0.99	N/A	91.3 %
GD	Sigmoid	No	120.0	3	2	0.032	0.9	0.999	N/A	34.8 %
GD	Sigmoid	No	120.0	3	1	0.0032	0.9	0.999	N/A	0.0 %
GD	Sigmoid	No	120.0	4	13	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	4	13	0.01	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	No	120.0	4	12	0.01	0.36	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	4	11	0.032	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	No	120.0	4	10	0.01	0.27	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	4	9	0.032	0.36	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	4	8	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	4	7	0.032	0.72	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	4	6	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	4	5	0.01	0.81	0.99	N/A	69.6 %
GD	Sigmoid	No	120.0	4	4	0.032	0.81	0.99	N/A	47.8 %
GD	Sigmoid	No	120.0	4	3	0.0032	0.9	0.99	N/A	34.8 %
GD	Sigmoid	No	120.0	4	2	0.032	0.09	0.999	N/A	0.0 %
GD	Sigmoid	No	120.0	5	16	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	5	16	0.032	0.54	0.9999	N/A	100.0 %
GD	Sigmoid	No	120.0	5	15	0.0032	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	5	14	0.01	0.99	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	5	13	0.01	0.45	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	5	12	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	5	11	0.032	0.45	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	5	10	0.032	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	5	9	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	5	8	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	5	7	0.01	0.9	0.99	N/A	91.3 %
GD	Sigmoid	No	120.0	5	6	0.01	0.45	0.99	N/A	65.2 %
GD	Sigmoid	No	120.0	5	5	0.01	0.54	0.99	N/A	43.5 %
GD	Sigmoid	No	120.0	5	4	0.032	0.9	0.99	N/A	34.8 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GD	Sigmoid	No	120.0	5	3	0.032	0.72	0.99	N/A	13.0 %
GD	Sigmoid	No	120.0	5	2	0.01	0.27	0.99	N/A	0.0 %
GD	Sigmoid	No	120.0	6	19	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	6	19	0.032	0.54	0.9999	N/A	100.0 %
GD	Sigmoid	No	120.0	6	18	0.01	0.36	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	6	17	0.032	0.72	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	6	16	0.032	0.72	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	6	15	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	6	14	0.032	0.63	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	6	13	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	No	120.0	6	12	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	No	120.0	6	11	0.032	0.9	0.99	N/A	91.3 %
GD	Sigmoid	No	120.0	6	10	0.01	0.9	0.99	N/A	95.7 %
GD	Sigmoid	No	120.0	6	9	0.01	0.9	0.99	N/A	91.3 %
GD	Sigmoid	No	120.0	6	8	0.032	0.72	0.99	N/A	65.2 %
GD	Sigmoid	No	120.0	6	7	0.01	0.9	0.99	N/A	65.2 %
GD	Sigmoid	No	120.0	6	6	0.01	0.9	0.99	N/A	26.1 %
GD	Sigmoid	No	120.0	6	5	0.01	0.81	0.99	N/A	17.4 %
GD	Sigmoid	No	120.0	6	4	0.0032	0.9	0.99	N/A	4.3 %
GD	Sigmoid	No	120.0	6	3	0.01	0.45	0.9999	N/A	0.0 %
GD	Sigmoid	Yes	120.0	2	7	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	2	6	0.01	0.36	0.99	N/A	93.5 %
GD	Sigmoid	Yes	120.0	2	6	0.01	0.9	0.9999	N/A	93.5 %
GD	Sigmoid	Yes	120.0	2	5	0.01	0.36	0.99	N/A	93.5 %
GD	Sigmoid	Yes	120.0	2	5	0.01	0.9	0.99	N/A	93.5 %
GD	Sigmoid	Yes	120.0	2	4	0.032	0.63	0.9999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	2	3	0.01	0.72	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	2	2	0.01	0.45	0.99	N/A	34.8 %
GD	Sigmoid	Yes	120.0	2	2	0.032	0	0.999	N/A	34.8 %
GD	Sigmoid	Yes	120.0	2	1	0.032	0.45	0.99	N/A	0.0 %
GD	Sigmoid	Yes	120.0	3	10	0.01	0.9	0.999	N/A	97.8 %
GD	Sigmoid	Yes	120.0	3	10	0.01	0.99	0.99	N/A	97.8 %
GD	Sigmoid	Yes	120.0	3	9	0.01	0.63	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	3	8	0.01	0.81	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	3	7	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	3	6	0.032	0.18	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	3	5	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	3	4	0.01	0.9	0.99	N/A	84.8 %
GD	Sigmoid	Yes	120.0	3	4	0.032	0.9	0.99	N/A	84.8 %
GD	Sigmoid	Yes	120.0	3	3	0.032	0.9	0.99	N/A	60.9 %
GD	Sigmoid	Yes	120.0	3	3	0.032	0.9	0.999	N/A	60.9 %
GD	Sigmoid	Yes	120.0	3	2	0.032	0.9	0.99	N/A	19.6 %
GD	Sigmoid	Yes	120.0	3	2	0.032	0.9	0.999	N/A	19.6 %
GD	Sigmoid	Yes	120.0	3	1	0.0032	0.9	0.999	N/A	0.0 %
GD	Sigmoid	Yes	120.0	4	13	0.0032	0.9	0.99	N/A	97.8 %
GD	Sigmoid	Yes	120.0	4	13	0.01	0.9	0.999	N/A	97.8 %
GD	Sigmoid	Yes	120.0	4	12	0.032	0.45	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	4	11	0.01	0.81	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	4	10	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	4	9	0.032	0.18	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	4	8	0.01	0.27	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	4	7	0.01	0.63	0.999	N/A	95.7 %
GD	Sigmoid	Yes	120.0	4	7	0.01	0.9	0.99	N/A	95.7 %
GD	Sigmoid	Yes	120.0	4	6	0.032	0.9	0.99	N/A	87.0 %
GD	Sigmoid	Yes	120.0	4	6	0.032	0.9	0.999	N/A	87.0 %
GD	Sigmoid	Yes	120.0	4	5	0.01	0.99	0.99	N/A	76.1 %
GD	Sigmoid	Yes	120.0	4	5	0.032	0.9	0.99	N/A	76.1 %
GD	Sigmoid	Yes	120.0	4	4	0.032	0.81	0.99	N/A	43.5 %
GD	Sigmoid	Yes	120.0	4	3	0.032	0.81	0.99	N/A	26.1 %
GD	Sigmoid	Yes	120.0	4	2	0.01	0.9	0.99	N/A	0.0 %
GD	Sigmoid	Yes	120.0	5	16	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	5	15	0.01	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	5	14	0.01	0.36	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	5	13	0.0032	0.18	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	5	12	0.01	0.27	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	5	11	0.032	0.36	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	5	10	0.01	0.9	0.99	N/A	100.0 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GD	Sigmoid	Yes	120.0	5	9	0.032	0.72	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	5	8	0.032	0.36	0.999	N/A	95.7 %
GD	Sigmoid	Yes	120.0	5	7	0.032	0.72	0.99	N/A	91.3 %
GD	Sigmoid	Yes	120.0	5	6	0.032	0.9	0.99	N/A	87.0 %
GD	Sigmoid	Yes	120.0	5	5	0.032	0.81	0.99	N/A	39.1 %
GD	Sigmoid	Yes	120.0	5	4	0.032	0.36	0.999	N/A	8.7 %
GD	Sigmoid	Yes	120.0	5	3	0.032	0.36	0.99	N/A	4.3 %
GD	Sigmoid	Yes	120.0	5	2	0.01	0.63	0.999	N/A	0.0 %
GD	Sigmoid	Yes	120.0	6	19	0.01	0.9	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	18	0.01	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	17	0.01	0.36	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	16	0.032	0.9	0.9999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	15	0.01	0.27	0.999	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	14	0.032	0.36	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	13	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	12	0.032	0.72	0.99	N/A	95.7 %
GD	Sigmoid	Yes	120.0	6	11	0.01	0.9	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	10	0.01	0.81	0.99	N/A	100.0 %
GD	Sigmoid	Yes	120.0	6	9	0.032	0.81	0.99	N/A	91.3 %
GD	Sigmoid	Yes	120.0	6	8	0.032	0.9	0.99	N/A	82.6 %
GD	Heaviside	Yes	120.0	2	9	0.01	0.45	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	2	8	0.032	0.72	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	2	7	0.01	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	3	13	0.032	0.63	0.9999	N/A	0.0 %
GD	Heaviside	Yes	120.0	3	12	0.01	0.45	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	3	11	0.032	0.72	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	3	10	0.01	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	4	17	0.0032	0.99	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	4	16	0.032	0.63	0.9999	N/A	0.0 %
GD	Heaviside	Yes	120.0	4	15	0.01	0.45	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	4	14	0.032	0.72	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	4	13	0.01	0.9	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	5	20	0.0032	0.99	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	5	19	0.032	0.63	0.9999	N/A	0.0 %
GD	Heaviside	Yes	120.0	5	18	0.01	0.45	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	5	17	0.032	0.72	0.999	N/A	0.0 %
GD	Heaviside	Yes	120.0	5	16	0.01	0.9	0.999	N/A	0.0 %
GDCR	Sigmoid	No	1	2	7	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	No	1	2	6	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	No	1	2	5	0.01	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	No	1	2	4	0.032	0.81	0.9	-4	100.0 %
GDCR	Sigmoid	No	1	2	3	0.01	0.09	0.99	-5	99.6 %
GDCR	Sigmoid	No	1	2	2	0.0032	0.9	0.99	-4	100.0 %
GDCR	Sigmoid	No	1	2	1	0.001	0.9	0.999	-8	0.0 %
GDCR	Sigmoid	No	1	3	10	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	No	1	3	9	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	No	1	3	8	0.01	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	No	1	3	7	0.032	0.81	0.9	-4	100.0 %
GDCR	Sigmoid	No	1	3	6	0.01	0.09	0.99	-5	100.0 %
GDCR	Sigmoid	No	1	3	5	0.0032	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	No	1	3	4	0.032	0.54	0.9	-8	99.2 %
GDCR	Sigmoid	No	1	3	3	0.0032	0.45	0.99	-7	99.6 %
GDCR	Sigmoid	No	1	3	2	0.032	0.81	0.99	-4	100.0 %
GDCR	Sigmoid	No	1	3	1	0.001	0.9	0.999	-8	0.0 %
GDCR	Sigmoid	No	1	4	13	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	No	1	4	12	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	No	1	4	11	0.01	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	No	1	4	10	0.032	0.81	0.9	-4	99.9 %
GDCR	Sigmoid	No	1	4	9	0.032	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	No	1	4	8	0.0032	0.9	0.99	-4	99.9 %
GDCR	Sigmoid	No	1	4	7	0.01	0.9	0.63	-4	99.4 %
GDCR	Sigmoid	No	1	4	6	0.0032	0.9	0.99	-5	99.9 %
GDCR	Sigmoid	No	1	4	5	0.032	0.9	0.9	-8	98.9 %
GDCR	Sigmoid	No	1	4	4	0.01	0.9	0.99	-4	99.6 %
GDCR	Sigmoid	No	1	4	3	0.01	0.9	0.99	-5	97.1 %
GDCR	Sigmoid	No	1	4	2	0.001	0.9	0.999	-8	0.0 %
GDCR	Sigmoid	No	1	5	16	0.01	0.99	0.999	-4	100.0 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GDCR	Sigmoid	No	1	5	15	0.0032	0.9	0.99	-8	100.0 %
GDCR	Sigmoid	No	1	5	14	0.0032	0.99	0.99	-5	99.9 %
GDCR	Sigmoid	No	1	5	13	0.001	0.63	0.9	-5	99.9 %
GDCR	Sigmoid	No	1	5	12	0.01	0.72	0.99	-6	99.4 %
GDCR	Sigmoid	No	1	5	11	0.01	0.9	0.99	-6	100.0 %
GDCR	Sigmoid	No	1	5	10	0.032	0.9	0.999	-5	99.8 %
GDCR	Sigmoid	No	1	5	9	0.01	0.9	0.99	-6	100.0 %
GDCR	Sigmoid	No	1	5	8	0.01	0.9	0.999	-8	99.6 %
GDCR	Sigmoid	No	1	5	7	0.0032	0.9	0.99	-6	99.2 %
GDCR	Sigmoid	No	1	5	6	0.01	0.9	0.99	-7	96.3 %
GDCR	Sigmoid	No	1	5	5	0.0032	0.9	0.99	-6	93.8 %
GDCR	Sigmoid	No	1	5	4	0.0032	0.9	0.99	-8	78.9 %
GDCR	Sigmoid	No	1	5	3	0.0032	0.9	0.99	-6	58.7 %
GDCR	Sigmoid	No	1	5	2	0.001	0.9	0.999	-8	0.0 %
GDCR	Sigmoid	No	1	6	19	0.032	0.81	0.999	-8	100.0 %
GDCR	Sigmoid	No	1	6	18	0.001	0.99	0.99	-7	99.5 %
GDCR	Sigmoid	No	1	6	17	0.032	0.9	0.999	-7	99.6 %
GDCR	Sigmoid	No	1	6	16	0.032	0.9	0.999	-6	100.0 %
GDCR	Sigmoid	No	1	6	15	0.001	0.99	0.99	-6	96.6 %
GDCR	Sigmoid	No	1	6	14	0.032	0.9	0.999	-6	99.5 %
GDCR	Sigmoid	No	1	6	13	0.032	0.9	0.999	-7	98.7 %
GDCR	Sigmoid	No	1	6	12	0.032	0.9	0.999	-7	98.3 %
GDCR	Sigmoid	No	1	6	11	0.032	0.9	0.999	-8	97.3 %
GDCR	Sigmoid	No	1	6	10	0.032	0.9	0.999	-8	96.4 %
GDCR	Sigmoid	No	1	6	9	0.032	0.9	0.999	-8	93.6 %
GDCR	Sigmoid	No	1	6	8	0.032	0.9	0.999	-8	89.9 %
GDCR	Sigmoid	No	1	6	7	0.032	0.9	0.999	-7	78.5 %
GDCR	Sigmoid	No	1	6	6	0.032	0.9	0.99	-5	56.4 %
GDCR	Sigmoid	No	1	6	5	0.01	0.9	0.99	-8	39.4 %
GDCR	Sigmoid	No	1	6	4	0.01	0.9	0.99	-6	23.2 %
GDCR	Sigmoid	No	1	6	3	0.001	0.9	0.999	-8	0.0 %
GDCR	Sigmoid	Yes	1	2	7	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	2	6	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	2	5	0.01	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	Yes	1	2	4	0.1	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	Yes	1	2	3	0.1	0.99	0.999	-3	100.0 %
GDCR	Sigmoid	Yes	1	2	2	0.032	0.72	0.99	-5	100.0 %
GDCR	Sigmoid	Yes	1	2	1	0.001	0.99	0.999	-8	0.0 %
GDCR	Sigmoid	Yes	1	3	10	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	3	9	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	3	8	0.01	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	Yes	1	3	7	0.032	0.81	0.9	-4	100.0 %
GDCR	Sigmoid	Yes	1	3	6	0.01	0.09	0.99	-5	100.0 %
GDCR	Sigmoid	Yes	1	3	5	0.0032	0.9	0.999	-4	100.0 %
GDCR	Sigmoid	Yes	1	3	4	0.1	0.18	0.9	-8	99.7 %
GDCR	Sigmoid	Yes	1	3	3	0.1	0.54	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	1	3	2	0.032	0.81	0.99	-6	97.7 %
GDCR	Sigmoid	Yes	1	3	1	0.001	0.99	0.999	-8	0.0 %
GDCR	Sigmoid	Yes	1	4	13	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	4	12	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	4	11	0.01	0.9	0.999	-4	99.8 %
GDCR	Sigmoid	Yes	1	4	10	0.032	0.81	0.9	-4	100.0 %
GDCR	Sigmoid	Yes	1	4	9	0.01	0.09	0.99	-5	99.9 %
GDCR	Sigmoid	Yes	1	4	8	0.01	0.9	0.99	-8	99.8 %
GDCR	Sigmoid	Yes	1	4	7	0.01	0.9	0.63	-4	99.8 %
GDCR	Sigmoid	Yes	1	4	6	0.0032	0.9	0.99	-5	98.9 %
GDCR	Sigmoid	Yes	1	4	5	0.032	0.9	0.99	-6	99.7 %
GDCR	Sigmoid	Yes	1	4	4	0.032	0.9	0.99	-5	99.8 %
GDCR	Sigmoid	Yes	1	4	3	0.032	0.72	0.99	-5	94.4 %
GDCR	Sigmoid	Yes	1	4	2	0.001	0.9	0.999	-8	0.0 %
GDCR	Sigmoid	Yes	1	5	16	0.0032	0.9	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	5	15	0.0032	0.36	0.9	-6	100.0 %
GDCR	Sigmoid	Yes	1	5	14	0.0032	0.99	0.99	-7	99.7 %
GDCR	Sigmoid	Yes	1	5	13	0.001	0.9	0.99	-6	99.9 %
GDCR	Sigmoid	Yes	1	5	12	0.01	0.72	0.99	-6	100.0 %
GDCR	Sigmoid	Yes	1	5	11	0.0032	0.36	0.54	-5	99.8 %
GDCR	Sigmoid	Yes	1	5	10	0.032	0.9	0.99	-5	99.5 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GDCR	Sigmoid	Yes	1	5	9	0.032	0.36	0.99	-7	99.2 %
GDCR	Sigmoid	Yes	1	5	8	0.01	0.72	0.81	-4	99.8 %
GDCR	Sigmoid	Yes	1	5	7	0.01	0.72	0.99	-6	98.7 %
GDCR	Sigmoid	Yes	1	5	6	0.1	0.81	0.99	-5	94.4 %
GDCR	Sigmoid	Yes	1	5	5	0.1	0.63	0.99	-5	67.8 %
GDCR	Sigmoid	Yes	1	5	4	0.032	0.81	0.99	-8	58.0 %
GDCR	Sigmoid	Yes	1	5	3	0.032	0.9	0.99	-8	29.7 %
GDCR	Sigmoid	Yes	1	5	2	0.001	0.9	0.99	-8	0.0 %
GDCR	Sigmoid	Yes	1	6	19	0.0032	0.9	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	6	18	0.032	0.9	0.9	-7	99.5 %
GDCR	Sigmoid	Yes	1	6	17	0.01	0.9	0.99	-7	99.9 %
GDCR	Sigmoid	Yes	1	6	16	0.032	0.9	0.99	-8	100.0 %
GDCR	Sigmoid	Yes	1	6	15	0.032	0.27	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	1	6	14	0.032	0.9	0.99	-6	100.0 %
GDCR	Sigmoid	Yes	1	6	13	0.032	0.27	0.99	-7	99.8 %
GDCR	Sigmoid	Yes	1	6	12	0.032	0.9	0.99	-6	99.2 %
GDCR	Sigmoid	Yes	1	6	11	0.032	0.45	0.99	-7	99.4 %
GDCR	Sigmoid	Yes	1	6	10	0.1	0.54	0.99	-5	96.1 %
GDCR	Sigmoid	Yes	1	6	9	0.032	0.81	0.99	-7	84.7 %
GDCR	Sigmoid	Yes	1	6	8	0.1	0.72	0.99	-6	84.3 %
GDCR	Sigmoid	Yes	1	6	7	0.032	0.9	0.99	-8	75.1 %
GDCR	Sigmoid	Yes	1	6	6	0.032	0.9	0.99	-8	69.5 %
GDCR	Sigmoid	Yes	1	6	5	0.032	0.9	0.99	-8	58.6 %
GDCR	Sigmoid	Yes	1	6	4	0.032	0.9	0.99	-8	17.9 %
GDCR	Sigmoid	Yes	1	6	3	0.001	0.9	0.999	-8	0.0 %
GDCR	ReLU	No	1	2	7	0.01	0.99	0.999	-4	100.0 %
GDCR	ReLU	No	1	2	6	0.032	0.54	0.9	-4	100.0 %
GDCR	ReLU	No	1	2	5	0.01	0.9	0.999	-4	100.0 %
GDCR	ReLU	No	1	2	4	0.0032	0.9	0.999	-7	99.8 %
GDCR	ReLU	No	1	2	3	0.032	0.9	0.999	-4	100.0 %
GDCR	ReLU	No	1	2	2	0.01	0.09	0.99	-2	100.0 %
GDCR	ReLU	No	1	2	1	0.001	0.9	0.999	-8	0.0 %
GDCR	ReLU	No	1	3	10	0.0032	0.63	0.99	-4	100.0 %
GDCR	ReLU	No	1	3	9	0.001	0.9	0.9	-3	100.0 %
GDCR	ReLU	No	1	3	8	0.0032	0.72	0.99	-3	100.0 %
GDCR	ReLU	No	1	3	7	0.0032	0.99	0.99	-3	98.7 %
GDCR	ReLU	No	1	3	6	0.0032	0.9	0.99	-4	99.9 %
GDCR	ReLU	No	1	3	5	0.0032	0.45	0.999	-2	97.4 %
GDCR	ReLU	No	1	3	4	0.01	0.63	0.99	-4	93.7 %
GDCR	ReLU	No	1	3	3	0.032	0.54	0.9	-2	83.8 %
GDCR	ReLU	No	1	3	2	0.001	0.99	0.999	-8	0.0 %
GDCR	ReLU	No	1	4	13	0.032	0.54	0.99	-7	100.0 %
GDCR	ReLU	No	1	4	12	0.0032	0.9	0.99	-4	98.5 %
GDCR	ReLU	No	1	4	11	0.01	0.36	0.999	-2	98.7 %
GDCR	ReLU	No	1	4	10	0.032	0.45	0.99	-2	98.8 %
GDCR	ReLU	No	1	4	9	0.032	0.63	0.999	-2	94.6 %
GDCR	ReLU	No	1	4	8	0.032	0.36	0.99	-2	92.9 %
GDCR	ReLU	No	1	4	7	0.032	0.45	0.9	-2	88.7 %
GDCR	ReLU	No	1	4	6	0.032	0.54	0.9	-2	73.9 %
GDCR	ReLU	No	1	4	5	0.01	0.36	0.99	-2	48.2 %
GDCR	ReLU	No	1	4	4	0.01	0.81	0.99	-3	22.5 %
GDCR	ReLU	No	1	4	3	0.01	0.72	0.9	-2	15.6 %
GDCR	ReLU	No	1	4	2	0.001	0.99	0.999	-8	0.0 %
GDCR	ReLU	No	1	5	16	0.01	0.72	0.99	-3	100.0 %
GDCR	ReLU	No	1	5	15	0.0032	0.81	0.999	-2	97.6 %
GDCR	ReLU	No	1	5	14	0.032	0.63	0.9	-2	97.3 %
GDCR	ReLU	No	1	5	13	0.01	0.81	0.999	-2	93.9 %
GDCR	ReLU	No	1	5	12	0.01	0.72	0.99	-2	90.5 %
GDCR	ReLU	No	1	5	11	0.032	0.63	0.99	-2	76.8 %
GDCR	ReLU	No	1	5	10	0.01	0.72	0.99	-2	67.0 %
GDCR	ReLU	No	1	5	9	0.032	0.54	0.9	-2	42.5 %
GDCR	ReLU	No	1	5	8	0.032	0.54	0.99	-3	26.9 %
GDCR	ReLU	No	1	5	7	0.032	0.45	0.81	-5	12.1 %
GDCR	ReLU	No	1	5	6	0.032	0.45	0.72	-7	8.8 %
GDCR	ReLU	No	1	5	5	0.032	0.54	0.99	-8	4.0 %
GDCR	ReLU	No	1	5	4	0.032	0.36	0.9	-8	1.1 %
GDCR	ReLU	No	1	5	3	0.001	0.9	0.999	-8	0.0 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GDCR	ReLU	No	1	6	22	0.0032	0.9	0.99	-3	100.0 %
GDCR	ReLU	No	1	6	21	0.0032	0.9	0.99	-2	98.9 %
GDCR	ReLU	No	1	6	21	0.01	0.72	0.99	-2	98.9 %
GDCR	ReLU	No	1	6	20	0.0032	0.9	0.99	-2	97.9 %
GDCR	ReLU	No	1	6	19	0.0032	0.9	0.99	-3	96.5 %
GDCR	ReLU	No	1	6	18	0.01	0.81	0.99	-2	91.9 %
GDCR	ReLU	No	1	6	17	0.0032	0.9	0.99	-2	87.0 %
GDCR	ReLU	No	1	6	16	0.0032	0.9	0.99	-2	78.5 %
GDCR	ReLU	No	1	6	15	0.0032	0.9	0.99	-2	63.7 %
GDCR	ReLU	No	1	6	14	0.01	0.81	0.999	-3	44.6 %
GDCR	ReLU	No	1	6	13	0.0032	0.9	0.999	-3	40.4 %
GDCR	ReLU	No	1	6	12	0.01	0.81	0.999	-4	25.4 %
GDCR	ReLU	No	1	6	11	0.01	0.81	0.999	-8	14.9 %
GDCR	ReLU	No	1	6	10	0.0032	0.9	0.999	-3	11.9 %
GDCR	ReLU	No	1	6	9	0.01	0.45	0.99	-8	3.4 %
GDCR	ReLU	No	1	6	8	0.032	0.54	0.99	-8	4.7 %
GDCR	ReLU	No	1	6	7	0.032	0.81	0.99	-8	1.2 %
GDCR	ReLU	No	1	6	6	0.0032	0.54	0.999	-8	0.2 %
GDCR	ReLU	Yes	1	2	7	0.01	0.99	0.999	-4	100.0 %
GDCR	ReLU	Yes	1	2	6	0.0032	0.63	0.99	-4	99.9 %
GDCR	ReLU	Yes	1	2	5	0.1	0.9	0.999	-2	100.0 %
GDCR	ReLU	Yes	1	2	4	0.01	0.18	0.999	-3	99.6 %
GDCR	ReLU	Yes	1	2	3	0.01	0.9	0.999	-3	100.0 %
GDCR	ReLU	Yes	1	2	2	0.01	0.72	0.99	-2	100.0 %
GDCR	ReLU	Yes	1	2	1	0.001	0.9	0.999	-8	0.0 %
GDCR	ReLU	Yes	1	3	10	0.0032	0.63	0.99	-4	100.0 %
GDCR	ReLU	Yes	1	3	9	0.0032	0.63	0.99	-4	100.0 %
GDCR	ReLU	Yes	1	3	8	0.0032	0.72	0.99	-4	99.8 %
GDCR	ReLU	Yes	1	3	7	0.01	0.54	0.99	-2	100.0 %
GDCR	ReLU	Yes	1	3	6	0.01	0.45	0.45	-2	99.6 %
GDCR	ReLU	Yes	1	3	5	0.1	0.9	0.999	-2	99.8 %
GDCR	ReLU	Yes	1	3	4	0.1	0.9	0.99	-2	95.9 %
GDCR	ReLU	Yes	1	3	3	0.032	0.54	0.99	-2	72.3 %
GDCR	ReLU	Yes	1	3	2	0.001	0.9	0.999	-8	0.0 %
GDCR	ReLU	Yes	1	4	13	0.0032	0.63	0.99	-4	100.0 %
GDCR	ReLU	Yes	1	4	12	0.032	0.63	0.72	-2	99.8 %
GDCR	ReLU	Yes	1	4	11	0.01	0.18	0.99	-2	98.6 %
GDCR	ReLU	Yes	1	4	10	0.032	0.36	0.99	-2	96.9 %
GDCR	ReLU	Yes	1	4	9	0.01	0.36	0.99	-2	96.9 %
GDCR	ReLU	Yes	1	4	8	0.032	0.72	0.9	-2	91.6 %
GDCR	ReLU	Yes	1	4	7	0.032	0.54	0.81	-2	82.4 %
GDCR	ReLU	Yes	1	4	6	0.032	0.54	0.99	-2	62.2 %
GDCR	ReLU	Yes	1	4	5	0.01	0.54	0.99	-2	33.8 %
GDCR	ReLU	Yes	1	4	4	0.032	0.72	0.63	-2	17.0 %
GDCR	ReLU	Yes	1	4	3	0.032	0.63	0.9	-2	7.2 %
GDCR	ReLU	Yes	1	4	2	0.001	0.9	0.999	-8	0.0 %
GDCR	ReLU	Yes	1	5	16	0.01	0.54	0.999	-2	100.0 %
GDCR	ReLU	Yes	1	5	15	0.01	0.81	0.9	-2	97.4 %
GDCR	ReLU	Yes	1	5	14	0.032	0.63	0.9	-2	97.3 %
GDCR	ReLU	Yes	1	5	13	0.032	0.63	0.9	-2	96.5 %
GDCR	ReLU	Yes	1	5	12	0.01	0.63	0.99	-2	89.6 %
GDCR	ReLU	Yes	1	5	11	0.01	0.72	0.99	-2	81.4 %
GDCR	ReLU	Yes	1	5	10	0.032	0.54	0.9	-2	62.9 %
GDCR	ReLU	Yes	1	5	9	0.01	0.54	0.99	-2	41.8 %
GDCR	ReLU	Yes	1	5	8	0.01	0.81	0.99	-3	20.4 %
GDCR	ReLU	Yes	1	5	7	0.01	0.09	0.9	-8	6.3 %
GDCR	ReLU	Yes	1	5	6	0.01	0.63	0.63	-2	5.8 %
GDCR	ReLU	Yes	1	5	5	0.032	0.36	0.54	-8	1.8 %
GDCR	ReLU	Yes	1	5	4	0.0032	0.99	0.999	-8	0.2 %
GDCR	ReLU	Yes	1	6	21	0.0032	0.9	0.99	-3	100.0 %
GDCR	ReLU	Yes	1	6	20	0.0032	0.9	0.99	-2	98.9 %
GDCR	ReLU	Yes	1	6	20	0.01	0.81	0.99	-2	98.9 %
GDCR	ReLU	Yes	1	6	19	0.01	0.81	0.99	-2	98.0 %
GDCR	ReLU	Yes	1	6	18	0.01	0.81	0.99	-2	93.2 %
GDCR	ReLU	Yes	1	6	17	0.01	0.81	0.99	-2	91.3 %
GDCR	ReLU	Yes	1	6	16	0.01	0.9	0.99	-2	78.1 %
GDCR	ReLU	Yes	1	6	15	0.01	0.81	0.99	-3	58.7 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GDCR	ReLU	Yes	1	6	14	0.01	0.81	0.999	-2	48.4 %
GDCR	ReLU	Yes	1	6	13	0.01	0.9	0.99	-2	36.4 %
GDCR	ReLU	Yes	1	6	12	0.0032	0.81	0.999	-4	20.3 %
GDCR	ReLU	Yes	1	6	11	0.01	0.9	0.99	-2	14.0 %
GDCR	ReLU	Yes	1	6	10	0.01	0.81	0.999	-8	7.9 %
GDCR	ReLU	Yes	1	6	9	0.0032	0.81	0.99	-8	4.3 %
GDCR	ReLU	Yes	1	6	8	0.0032	0.9	0.99	-4	2.3 %
GDCR	ReLU	Yes	1	6	7	0.01	0.9	0.99	-8	0.6 %
GDCR	Heaviside	No	1	2	7	0.0032	0.63	0.99	-4	100.0 %
GDCR	Heaviside	No	1	2	6	0.0032	0.63	0.99	-4	100.0 %
GDCR	Heaviside	No	1	2	5	0.0032	0.72	0.99	-4	99.6 %
GDCR	Heaviside	No	1	2	4	0.01	0.54	0.99	-2	100.0 %
GDCR	Heaviside	No	1	2	3	0.032	0.54	0.99	-3	100.0 %
GDCR	Heaviside	No	1	2	2	0.1	0.45	0.999	-3	99.8 %
GDCR	Heaviside	No	1	2	1	0.001	0.99	0.999	-8	0.0 %
GDCR	Heaviside	No	1	3	10	0.01	0.9	0.99	-4	100.0 %
GDCR	Heaviside	No	1	3	9	0.032	0.9	0.99	-3	99.9 %
GDCR	Heaviside	No	1	3	8	0.1	0.81	0.999	-3	99.9 %
GDCR	Heaviside	No	1	3	7	0.1	0.63	0.99	-3	98.6 %
GDCR	Heaviside	No	1	3	6	0.1	0.72	0.99	-3	86.1 %
GDCR	Heaviside	No	1	3	5	0.1	0.54	0.99	-3	56.8 %
GDCR	Heaviside	No	1	3	4	0.1	0.72	0.9	-2	28.6 %
GDCR	Heaviside	No	1	3	3	0.1	0.63	0.9	-2	9.7 %
GDCR	Heaviside	No	1	3	2	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	No	1	4	17	0.1	0.9	0.999	-3	70.0 %
GDCR	Heaviside	No	1	4	16	0.1	0.9	0.999	-3	59.1 %
GDCR	Heaviside	No	1	4	15	0.1	0.81	0.99	-3	33.6 %
GDCR	Heaviside	No	1	4	14	0.01	0.9	0.99	-3	18.0 %
GDCR	Heaviside	No	1	4	14	0.1	0.81	0.999	-3	18.0 %
GDCR	Heaviside	No	1	4	13	0.1	0.81	0.99	-3	10.0 %
GDCR	Heaviside	No	1	4	12	0.001	0.9	0.999	-8	0.1 %
GDCR	Heaviside	No	1	5	21	0.001	0.54	0.999	-8	0.0 %
GDCR	Heaviside	No	1	5	20	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	No	1	5	19	0.001	0.999	0.9	-8	0.0 %
GDCR	Heaviside	No	1	5	18	0.001	0.9	0.99	-8	0.0 %
GDCR	Heaviside	No	1	5	17	0.001	0.999	0.99	-8	0.0 %
GDCR	Heaviside	No	1	5	16	0.001	0.99	0.999	-8	0.0 %
GDCR	Heaviside	No	1	6	25	0.001	0.63	0.45	-8	0.0 %
GDCR	Heaviside	No	1	6	24	0.001	0.54	0.999	-8	0.0 %
GDCR	Heaviside	No	1	6	23	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	No	1	6	22	0.001	0.999	0.9	-8	0.0 %
GDCR	Heaviside	No	1	6	21	0.001	0.9	0.99	-8	0.0 %
GDCR	Heaviside	No	1	6	20	0.001	0.999	0.99	-8	0.0 %
GDCR	Heaviside	No	1	6	19	0.001	0.99	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	2	9	0.001	0.9	0.999	-8	58.0 %
GDCR	Heaviside	Yes	1	2	8	0.001	0.9	0.999	-8	58.7 %
GDCR	Heaviside	Yes	1	2	7	0.001	0.9	0.999	-8	59.3 %
GDCR	Heaviside	Yes	1	2	6	0.001	0.9	0.999	-8	59.5 %
GDCR	Heaviside	Yes	1	2	5	0.001	0.9	0.999	-8	57.6 %
GDCR	Heaviside	Yes	1	2	4	0.001	0.9	0.999	-8	51.5 %
GDCR	Heaviside	Yes	1	2	3	0.001	0.9	0.999	-8	44.7 %
GDCR	Heaviside	Yes	1	2	2	0.001	0.9	0.999	-8	23.3 %
GDCR	Heaviside	Yes	1	2	1	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	3	13	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	3	12	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	3	11	0.001	0.9	0.999	-8	1.0 %
GDCR	Heaviside	Yes	1	3	10	0.001	0.9	0.999	-8	1.0 %
GDCR	Heaviside	Yes	1	4	17	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	4	16	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	4	15	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	4	14	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	4	13	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	5	21	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	5	20	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	5	19	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	5	18	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	5	17	0.001	0.9	0.999	-8	0.0 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GDCR	Heaviside	Yes	1	5	16	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	6	25	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	6	24	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	6	23	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	6	22	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	6	21	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	6	20	0.001	0.9	0.999	-8	0.0 %
GDCR	Heaviside	Yes	1	6	19	0.001	0.9	0.999	-8	0.0 %
GDCR	Sigmoid	No	120.0	2	7	0.01	0.9	0.999	-7	100.0 %
GDCR	Sigmoid	No	120.0	2	7	0.01	0.9	0.9999	-7	100.0 %
GDCR	Sigmoid	No	120.0	2	6	0.01	0.36	0.99	-4	100.0 %
GDCR	Sigmoid	No	120.0	2	5	0.032	0.9	0.9999	-4	100.0 %
GDCR	Sigmoid	No	120.0	2	4	0.01	0.27	0.999	-7	100.0 %
GDCR	Sigmoid	No	120.0	2	3	0.032	0.36	0.99	-6	100.0 %
GDCR	Sigmoid	Yes	120.0	2	7	0.01	0.9	0.999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	2	7	0.01	0.9	0.9999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	2	6	0.01	0.36	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	2	5	0.032	0.9	0.9999	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	2	4	0.01	0.27	0.999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	2	3	0.032	0.36	0.99	-6	100.0 %
GDCR	Sigmoid	Yes	120.0	2	2	0.032	0.45	0.99	-4	95.7 %
GDCR	Sigmoid	Yes	120.0	2	1	0.01	0.9	0.999	-3	0.0 %
GDCR	Sigmoid	Yes	120.0	3	10	0.01	0.9	0.9999	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	3	9	0.032	0.99	0.999	-8	100.0 %
GDCR	Sigmoid	Yes	120.0	3	8	0.01	0.99	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	3	7	0.0032	0.27	0.999	-5	100.0 %
GDCR	Sigmoid	Yes	120.0	3	6	0.01	0.45	0.999	-5	100.0 %
GDCR	Sigmoid	Yes	120.0	3	5	0.032	0.18	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	3	4	0.01	0.9	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	3	3	0.032	0.36	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	3	2	0.01	0.81	0.99	-5	60.9 %
GDCR	Sigmoid	Yes	120.0	3	1	0.032	0.99	0.99	-3	0.0 %
GDCR	Sigmoid	Yes	120.0	4	13	0.01	0.9	0.9999	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	4	12	0.032	0.99	0.999	-8	100.0 %
GDCR	Sigmoid	Yes	120.0	4	11	0.01	0.99	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	4	10	0.0032	0.9	0.999	-5	100.0 %
GDCR	Sigmoid	Yes	120.0	4	9	0.01	0.9	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	4	8	0.01	0.54	0.9999	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	4	7	0.01	0.99	0.99	-5	100.0 %
GDCR	Sigmoid	Yes	120.0	4	6	0.032	0.18	0.999	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	4	5	0.032	0.9	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	4	4	0.032	0.9	0.99	-8	43.5 %
GDCR	Sigmoid	Yes	120.0	4	3	0.032	0.81	0.99	-8	26.1 %
GDCR	Sigmoid	Yes	120.0	4	2	0.01	0.18	0.9999	-4	0.0 %
GDCR	Sigmoid	Yes	120.0	5	16	0.01	0.9	0.999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	5	16	0.01	0.9	0.9999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	5	15	0.01	0.36	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	5	14	0.032	0.9	0.9999	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	5	13	0.01	0.27	0.999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	5	12	0.032	0.36	0.99	-6	100.0 %
GDCR	Sigmoid	Yes	120.0	5	11	0.01	0.9	0.99	-6	100.0 %
GDCR	Sigmoid	Yes	120.0	5	10	0.032	0.9	0.999	-8	100.0 %
GDCR	Sigmoid	Yes	120.0	5	9	0.01	0.99	0.99	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	5	8	0.01	0.9	0.99	-5	100.0 %
GDCR	Sigmoid	Yes	120.0	5	7	0.032	0.81	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	5	6	0.032	0.63	0.99	-5	100.0 %
GDCR	Sigmoid	Yes	120.0	5	5	0.032	0.36	0.99	-5	43.5 %
GDCR	Sigmoid	Yes	120.0	5	4	0.032	0.9	0.99	-5	8.7 %
GDCR	Sigmoid	Yes	120.0	5	3	0.032	0.54	0.99	-6	4.3 %
GDCR	Sigmoid	Yes	120.0	5	2	0.032	0.9	0.99	-5	0.0 %
GDCR	Sigmoid	Yes	120.0	6	19	0.01	0.9	0.999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	6	19	0.01	0.9	0.9999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	6	18	0.01	0.36	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	6	17	0.032	0.9	0.9999	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	6	16	0.01	0.27	0.999	-7	100.0 %
GDCR	Sigmoid	Yes	120.0	6	15	0.032	0.36	0.99	-6	100.0 %
GDCR	Sigmoid	Yes	120.0	6	14	0.01	0.9	0.99	-6	100.0 %

Continued on next page...

Algo.	**Act. fn.**	**O. act.**	**I.g.**	**Inp.**	**Hdn.**	η	β_1	β_2	$\log_{10} r$	**Succ. r.**
GDCR	Sigmoid	Yes	120.0	6	13	0.032	0.9	0.999	-8	100.0 %
GDCR	Sigmoid	Yes	120.0	6	12	0.032	0.81	0.99	-4	100.0 %
GDCR	Sigmoid	Yes	120.0	6	11	0.032	0.54	0.999	-8	100.0 %
GDCR	Sigmoid	Yes	120.0	6	10	0.01	0.36	0.99	-5	100.0 %
GDCR	Sigmoid	Yes	120.0	6	9	0.01	0.63	0.99	-5	95.7 %
GDCR	Sigmoid	Yes	120.0	6	8	0.032	0.9	0.99	-5	87.0 %
GDCR	Sigmoid	Yes	120.0	6	7	0.032	0.72	0.99	-6	69.6 %
GDCR	Sigmoid	Yes	120.0	6	6	0.032	0.81	0.99	-8	34.8 %
GDCR	Sigmoid	Yes	120.0	6	5	0.032	0.9	0.999	-8	13.0 %
GDCR	Sigmoid	Yes	120.0	6	4	0.032	0.18	0.999	-7	0.0 %
GDCR	Heaviside	No	120.0	2	7	0.01	0.36	0.999	-7	100.0 %
GDCR	Heaviside	No	120.0	2	7	0.032	0.99	0.9999	-5	100.0 %
GDCR	Heaviside	No	120.0	2	6	0.01	0.81	0.999	-4	100.0 %
GDCR	Heaviside	Yes	120.0	2	9	0.01	0.45	0.999	-3	56.5 %
GDCR	Heaviside	Yes	120.0	2	9	0.032	0.63	0.9999	-8	56.5 %
GDCR	Heaviside	Yes	120.0	2	8	0.0032	0.99	0.999	-7	60.9 %
GDCR	Heaviside	Yes	120.0	2	8	0.032	0.72	0.999	-1	60.9 %
GDCR	Heaviside	Yes	120.0	2	7	0.01	0.9	0.999	-7	73.9 %
GDCR	Heaviside	Yes	120.0	2	7	0.01	0.99	0.9999	-6	73.9 %
GDCR	Heaviside	Yes	120.0	2	6	0.01	0.99	0.9999	-4	65.2 %
GDCR	Heaviside	Yes	120.0	2	5	0.01	0.09	0.9999	-5	65.2 %

Glossary

AMS anticipated mean shift. 14, 16, 33

BIPOP-CMA-ES bi-population CMA-ES. i, iii, vi, vii, 2, 14, 21–23, 25–28, 30–34, 36–41

CMA-ES covariance matrix adaptation evolutionary strategy. 3, 12–14, 16, 72

EA evolutionary algorithm. i, 1–3, 7, 10–12, 14, 20, 27, 32, 33, 35, 40–42

ES evolutionary strategy. 2, 12

FOS family of subsets. 14, 15

GA genetic algorithm. 2, 18

GD gradient descent. i, v–vii, 1–3, 6–10, 18–25, 27–33, 35, 37–42, 58–71

GDCR gradient descent with cold restarts. v, vii, 10, 21–25, 27, 29–33, 38, 39, 58–71

GOMEA gene-pool optimal mixing evolutionary algorithm. i, vi, vii, 2, 3, 14, 15, 20–23, 25–28, 30–33, 36, 37, 39–42

IMS interleaved multi-start. 14, 16, 23

MLP multilayer perceptron. i, 3, 18, 34, 42

MSE mean squared error. 20, 26, 34

NN neural network. i, vi, 1–3, 5, 7, 17, 21, 41, 42

PSO particle-swarm optimization. 21, 22

ReLU rectified linear unit. i, iii, iv, vi, vii, 6, 17, 18, 22, 24, 26, 30–32, 36, 37, 39, 46, 60–62, 67–69

RNG random number generator. 20, 22

XNOR Excusive-NOR. 16

XOR Exclusive-OR. i, iii, vi, vii, 1–3, 16–18, 20–35

Bibliography

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, e00938, 2018. DOI: 10.1016/j.heliyon.2018.e00938.

[2] T. J. Brinker *et al.*, "Skin cancer classification using convolutional neural networks: Systematic review," *Journal of Medical Internet Research*, vol. 20, no. 10, e11936, Oct. 17, 2018, Company: Journal of Medical Internet Research Distributor: Journal of Medical Internet Research Institution: Journal of Medical Internet Research Label: Journal of Medical Internet Research Publisher: JMIR Publications Inc., Toronto, Canada. DOI: 10.2196/11936. [Online]. Available: https://www.jmir.org/2018/10/e11936.

[3] T. B. Brown *et al.*, "Language models are few-shot learners," *arXiv:2005.14165 [cs]*, Jul. 22, 2020. arXiv: 2005.14165. [Online]. Available: http://arxiv.org/abs/2005.14165.

[4] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv:1803.03635 [cs]*, Mar. 9, 2018, version: 1. arXiv: 1803.03635. [Online]. Available: http://arxiv.org/abs/1803.03635.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980 [cs]*, Jan. 29, 2017. arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980.

[6] G. Hornby, A. Globus, D. Linden, and J. Lohn, "Automated antenna design with evolutionary algorithms," in *Space 2006*, San Jose, California: American Institute of Aeronautics and Astronautics, Sep. 19, 2006, ISBN: 978-1-62410-049-9. DOI: 10.2514/6.2006-7242. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2006-7242.

[7] V. Kuroda, M. Allard, B. Lewis, and M. Lindsay, "Comm for small sats: The lunar atmosphere and dust environment explorer (LADEE) communications subsystem," p. 11, 2014. [Online]. Available: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=3075&context=smallsat.

[8] D. Chia and L. While, "Automated design of architectural layouts using a multi-objective evolutionary algorithm," in *Proceedings of the 10th International Conference on Simulated Evolution and Learning - Volume 8886*, ser. SEAL 2014, Berlin, Heidelberg: Springer-Verlag, Dec. 15, 2014, pp. 760–772, ISBN: 978-3-319-13562-5. DOI: 10.1007/978-3-319-13563-2_64. [Online]. Available: https://doi.org/10.1007/978-3-319-13563-2_64.

[9] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing*, vol. 14, no. 3, pp. 347–361, Aug. 1, 1990, ISSN: 0167-8191. DOI: 10.1016/0167-8191(90)90086-O. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016781919090086O.

[10] D. E. Rumelhart and J. L. McClelland, “Learning internal representations by error propagation,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[11] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, Apr. 1998. DOI: 10.1142/s0218488598000094. [Online]. Available: https://doi.org/10.1142/s0218488598000094.

[12] M. Zeiler *et al.*, “On rectified linear units for speech processing,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 3517–3521. DOI: 10.1109/ICASSP.2013.6638312.

[13] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943. DOI: 10.1007/bf02478259. [Online]. Available: https://doi.org/10.1007/bf02478259.

[14] G. Hinton, N. Srivastava, and K. Swersky, *Neural networks for machine learning*, 2011. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visited on 04/28/2022).

[15] B. T. Polyak, “Some methods of speeding up the convergence of iteration methods,” *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, Jan. 1, 1964, ISSN: 0041-5553. DOI: 10.1016/0041-5553(64)90137-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0041555364901375.

[16] D. Thierens and P. A. Bosman, “Optimal mixing evolutionary algorithms,” in *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO ’11*, Dublin, Ireland: ACM Press, 2011, p. 617, ISBN: 978-1-4503-0557-0. DOI: 10.1145/2001576.2001661. [Online]. Available: http://portal.acm.org/citation.cfm?doid=2001576.2001661.

[17] A. Bouter, T. Alderliesten, C. Witteveen, and P. A. N. Bosman, “Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Berlin Germany: ACM, Jul. 2017, pp. 705–712, ISBN: 978-1-4503-4920-8. DOI: 10.1145/3071178.3071272. [Online]. Available: https://dl.acm.org/doi/10.1145/3071178.3071272.

[18] N. H. Luong, H. La Poutré, and P. A. Bosman, “Multi-objective gene-pool optimal mixing evolutionary algorithm with the interleaved multi-start scheme,” *Swarm and Evolutionary Computation*, vol. 40, pp. 238–254, Jun. 2018, ISSN: 22106502. DOI: 10.1016/j.swevo.2018.02.005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2210650217304765.

[19] G. R. Harik and F. G. Lobo, “A parameter-less genetic algorithm,” in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, ser. GECCO’99, Orlando, Florida: Morgan Kaufmann Publishers Inc., 1999, pp. 258–265, ISBN: 1558606114. DOI: 10.5555/2933923.2933949.

[20] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[21] D. J. Montana and L. Davis, “Training feedforward neural networks using genetic algorithms,” p. 6, 1989.

[22] D. B. Fogel, L. J. Fogel, and V. W. Porto, “Evolving neural networks,” *Biological Cybernetics*, vol. 63, no. 6, pp. 487–493, Oct. 1, 1990, ISSN: 1432-0770. DOI: 10.1007/BF00199581. [Online]. Available: https://doi.org/10.1007/BF00199581.

[23] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, Jun. 2002, ISSN: 1063-6560, 1530-9304. DOI: 10.1162/106365602320169811. [Online]. Available: https://direct.mit.edu/evco/article/10/2/99-127/1123.

[24] K. Stanley, B. Bryant, and R. Miikkulainen, “Real-time neuroevolution in the NERO video game,” *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 653–668, Dec. 2005, ISSN: 1089-778X. DOI: 10.1109/TEVC.2005.856210. [Online]. Available: http://ieeexplore.ieee.org/document/1545941/.

[25] X. Zhou, A. K. Qin, M. Gong, and K. C. Tan, “A survey on evolutionary construction of deep neural networks,” *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 5, pp. 894–912, Oct. 2021, Conference Name: IEEE Transactions on Evolutionary Computation, ISSN: 1941-0026. DOI: 10.1109/TEVC.2021.3079985.

[26] K. Stanley and R. Miikkulainen, “Efficient evolution of neural network topologies,” in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No.02TH8600)*, vol. 2, Honolulu, HI, USA: IEEE, 2002, pp. 1757–1762, ISBN: 978-0-7803-7282-5. DOI: 10.1109/CEC.2002.1004508. [Online]. Available: http://ieeexplore.ieee.org/document/1004508/.

[27] G. Morse and K. O. Stanley, “Simple evolutionary optimization can rival stochastic gradient descent in neural networks,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO ’16, New York, NY, USA: Association for Computing Machinery, Jul. 20, 2016, pp. 477–484, ISBN: 978-1-4503-4206-3. DOI: 10.1145/2908812.2908916. [Online]. Available: https://doi.org/10.1145/2908812.2908916.

[28] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning,” *arXiv:1712.06567 [cs]*, Apr. 20, 2018. arXiv: 1712.06567. [Online]. Available: http://arxiv.org/abs/1712.06567.

[29] K. O. Stanley and R. Miikkulainen, “A taxonomy for artificial embryogeny,” vol. 9, no. 2, p. 39, 2003. DOI: 10.1162/106454603322221487.

[30] “Linear — PyTorch 1.11.0 documentation.” (Mar. 10, 2022), [Online]. Available: https://pytorch.org/docs/1.11/generated/torch.nn.Linear.html#torch.nn.Linear.

[31] “CMA evolution strategy source code.” (Jun. 3, 2011), [Online]. Available: https://cma-es.github.io/cmaes_sourcecode_page.html (visited on 04/26/2022).

[32] R. M. Schmidt, F. Schneider, and P. Hennig, "Descending through a crowded valley - benchmarking deep learning optimizers," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 18–24 Jul 2021, pp. 9367–9376. [Online]. Available: https://proceedings.mlr.press/v139/schmidt21a.html.

[33] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, "Particle swarm optimization for hyper-parameter selection in deep neural networks," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17, New York, NY, USA: Association for Computing Machinery, Jul. 1, 2017, pp. 481–488, ISBN: 978-1-4503-4920-8. DOI: 10.1145/3071178.3071208. [Online]. Available: https://doi.org/10.1145/3071178.3071208.

[34] Y. Akimoto, Y. Nagata, I. Ono, and S. Kobayashi, "Theoretical foundation for CMA-ES from information geometry perspective," en, *Algorithmica*, vol. 64, no. 4, pp. 698–716, Dec. 2012.

[35] Z. Li and Q. Zhang, "What does the evolution path learn in CMA-ES?" In *Parallel Problem Solving from Nature – PPSN XIV*, ser. Lecture notes in computer science, Cham: Springer International Publishing, 2016, pp. 751–760.

[36] A. Prugel-Bennett, "Symmetry breaking in population-based optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 63–79, 2004. DOI: 10.1109/TEVC.2003.819419.

[37] D. Yan, T. Wu, Y. Liu, and Y. Gao, "An efficient sparse-dense matrix multiplication on a multicore system," in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, 2017, pp. 1880–1883. DOI: 10.1109/ICCT.2017.8359956.

[38] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *CoRR*, vol. abs/1811.03378, 2018. arXiv: 1811.03378. [Online]. Available: http://arxiv.org/abs/1811.03378.

[39] P. Ramachandran, B. Zoph, and Q. V. Le, *Searching for activation functions*, 2017. DOI: 10.48550/ARXIV.1710.05941. [Online]. Available: https://arxiv.org/abs/1710.05941.

[40] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013. DOI: 10.1.1.693.1422.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[42] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on MNIST classification task," *CoRR*, vol. abs/1804.02763, 2018. arXiv: 1804.02763. [Online]. Available: http://arxiv.org/abs/1804.02763.

[43] J. D. McCalpin. "Memory bandwidth and system balance in HPC systems." (2016), [Online]. Available: https://repositories.lib.utexas.edu/handle/2152/86843.

[44] M. Laskin *et al.*, "Parallel training of deep networks with local updates," *arXiv:2012.03837 [cs]*, Jun. 15, 2021. arXiv: 2012.03837. [Online]. Available: http://arxiv.org/abs/2012.03837.

[45] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. DOI: 10.48550/ARXIV.1409.1556. [Online]. Available: https://arxiv.org/abs/1409.1556.

[46] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.