An Empirical Investigation of Neural Networks, Evolution Strategies, and Evolutionary-trained Neural Networks and their Application to Some Chemical Engineering Problems.

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik

von
Martin Mandischer

Dortmund
May 2000

Tag der mündlichen Prüfung:

Dekan:
Prof. Dr. B. Reusch

Gutachter:
Prof. Dr. H.-P. Schwefel
Prof. Dr. C. Moraga

Acknowledgements

My thanks go to all the people who helped make this work possible.

I thank Frank Kursawe, Joachim Sprave, Robert E. Keller, and Jens Niehaus who provided me with helpful comments on earlier versions of some chapters, Thomas Bäck for his support on the chapter on evolution strategies, and Ralf Garionis for his motivation and fruitful discussions.

Special thanks go to Hannes Geyer for being a productive collaborator on the chemical-engineering problems.

The English was greatly improved due to Robert E. Keller's excellent and thorough proof reading of various draft versions. Thomas Rölleke and Monia Lamas contributed a final proof reading.

I thank all other folks at the chair of systems analysis for discussions, help on several technical issues and for the collegial and friendly atmosphere they created. Finally I thank my partner Angela Wrangel for her patience and motivation whenever doubt or desperation started sneaking up on me.

Last not least I thank Prof. Hans-Paul Schwefel for his interest and the trust he put into my work and Prof. Claudio Moraga for commenting on the thesis with valuable suggestions.

Contents

Chapter 1

Computational Intelligence

1.1 Outline of the Field

Computational Intelligence (CI) comprises three biologically inspired paradigms, neural networks (NNs), fuzzy logic (FL), and evolutionary algorithms (EAs), that have their origin in the Forties and early Sixties. Research on these methods proceeded almost independently until the late Eighties and early Nineties when the respective scientific communities started to employ each other's methods to solve difficult problems in their own domains. In those years research started in the area of evolutionary neural network design [68, 1], fuzzy logic and neural network combinations [12] as well as hybrid methods with classical artificial intelligence (AI) methods like rule-based expert systems [89] and logical reasoning [91]. Although classical AI methods are based on symbolic rather than sub-symbolic, numeric computation like CI methods, they should not be neglected, since hybridization can yield synergetic effects regardless of paradigm-specific demarcation lines.

Since 1994 when the first World Congress on Computational Intelligence (Orlando, Florida) took place CI became a popular term covering those three paradigms. Among the huge number of scientific publications and books available today, the handbooks of Neural Computation [17], Evolutionary Computation [4] and Fuzzy Computation [70] document the state-of-the-art in each field.

1.2 Scope and Structure of the Thesis

Evolutionary algorithms and neural networks have been successfully used to solve difficult problems in various domains. Researchers and practitioners have applied them as single paradigms or in combination with each other.

Contributions of the thesis are:

- A compact introduction to neural networks and evolutionary algorithms.
- An investigation of evolution strategies (ESs, a subclass of EAs) as an alternative to gradient-based neural network training.

  Based on an empirical comparison of population- and gradient-based search, we will derive hints for parameterization and draw conclusions about the usefulness of evolution strategies for this purpose. We will see that ESs can only compete with gradient-based search in case of small problems and that ESs are a good method for training neural networks with a non-continuously differentiable activation function.
- From the viewpoint of ESs, we will gain insights in how evolution strategies behave in search spaces generated by neural networks. Here, we will see that for this class of objective functions, the dimensionality of the problem is critical. With increasing numbers of decision variables, the learning becomes more and more difficult for ESs, and the "efficient" parameterization becomes crucial.
- The thesis shows how to solve difficult real-world problems from the field of chemical engineering with the aid of neural networks and evolutionary algorithms. The problems originate from the Collaborative Research Center "Computational Intelligence" (SFB-531 "Design und Management komplexer Prozesse und Systeme mit Methoden der Computational Intelligence").

  In this context various neural, evolutionary and thermodynamics-based approaches will be compared with respect to their performance and usefulness for the given problems.

This thesis is structured as follows. Chapters 2 and 3 give a brief introduction into neural networks and evolutionary algorithms, respectively. The fundamentals of both paradigms are reviewed and advanced techniques introduced. Chapter 4 investigates evolution strategies as an alternative to gradient-based neural network training while chapter 5 is concerned with the application of neural networks and evolutionary algorithms in chemical engineering. The thesis concludes with a summary and outlook in chapter 6. The appendix A gathers subsidiary experiments and comprises results (charts, tables) from experiments reported in chapters 4 and 5.

Chapter 2

Neural Networks

The human brain consists of assemblies of neurons which are highly structured and interconnected by synapses. Artificial neural networks are gleaned from nature but are far less complex and powerful compared to their natural paragon.

We will look at supervised neural networks and introduce advanced techniques to improve learning which will be used throughout this thesis. Section A.2 will be concerned with the complexity of learning and choosing topologies.

2.1 Basic Definitions

This section introduces the notation for neural networks and some basic definitions that will be used throughout this thesis. The focus will be on supervised learning in feed-forward networks.

2.1.1 Notation

From the computer science view the structure of an artificial neural network can be defined as a graph.

Definition 2.1.1 (Network structure, order)
The structure of a neural network *is given by a graph $G = (U, W)$ with U as a set of units (neurons) and $W \subseteq U \times U$ as a set of weighted edges (weights). The total number of units in the network $|U|$ is called network* order.

For two units $u_j, u_i \in U$ let $w_{i,j} \in \mathbb{R}$ be called the weight from unit u_j to unit u_i.

This definition does not fully describe a neural network. The following sections deal with other functional parts of networks such as unit types and training algorithms.

2.1.1.1 Units

A network unit can be described by the following components: An activation range, an activation function, a transfer function and a bound function. Additionally, there is a distinction between deterministic and stochastic activation schemes.

Definition 2.1.2 (Activation range, transfer function)
Let $A \subset \mathbb{R}$ *be the* activation range, U *a set of units,* $g : \mathbb{R}^n \to \mathbb{R}$ a transfer function *and* $h : \mathbb{R} \to A$ *a bound function. The* activation function $f : \mathbb{R}^n \to A$ *can then be defined as* $f : h \circ g$. *Here,* n *denotes the number of incoming weights, also called* fan-in *of the unit.*

The threshold behavior of biological neurons can be modeled with a dedicated bias value and a bounded activation range.

Definition 2.1.3 (Threshold, activation, bias, state)
We define that every unit $u_i \in U$ *should have* a threshold $s_i \in \mathbb{R}$, *and at time* t, *an* activation $a_i(t) \in A$. *Let* $m := |U|$, *then the vector* $Z_t := (a_1(t), \cdots, a_m(t))$ *is called* state *of the network at time* t.
Throughout this thesis the transfer function ($net_i : \mathbb{R}^n \to \mathbb{R}$*) of a unit is called* netinput *and defined by:*

$$net_i := b_i + \sum_{j=1}^{n} w_{i,j} \cdot a_j \tag{2.1}$$

with b_i *as* bias *for unit* i.

McCulloch and Pitts [54] first modeled the threshold behavior of biological neurons. This leads to the following definitions.

Definition 2.1.4 (Threshold unit)
A unit is called threshold unit *or* McCulloch-Pitts unit, *if the activation range is given by the set* $A := \{-1, +1\}$ *(sometimes* A *is also chosen to be* $\{0, 1\}$*) and the activation of the unit is:*

$$a_i = f(net_i) = \begin{cases} +1 & \text{if } net_i \geq s_i, \\ -1 & \text{if } net_i < s_i \end{cases} \tag{2.2}$$

Such a unit with n inputs calculates a binary function $f : \mathbb{B}^n \rightarrow \mathbb{B}$. Functions that can be calculated by a threshold unit[1] are called *linear separable*.

Units that approximate an S-shape are called *sigmoid units*. A typical sigmoid activation function is the logistic function with a usual activation range of $A = [0, 1]$.

Definition 2.1.5 (Logistic unit)
The logistic unit *is defined as:*

$$f(net_i) = \frac{1}{1 + e^{-\gamma \cdot net_i}} + S_i \tag{2.3}$$

where γ is the slope (or $\frac{1}{\gamma}$ the temperature) of the activation function and S_i the threshold which can be used to shift the function.

The γ value controls the degree of non-linearity of that unit. Large γ modulate the logistic function to a step function like the McCulloch-Pitts units while small γ produce an almost linear behavior. In figures 2.1 and 2.2 the effect of varying γ on the logistic function and its first derivative is illustrated.

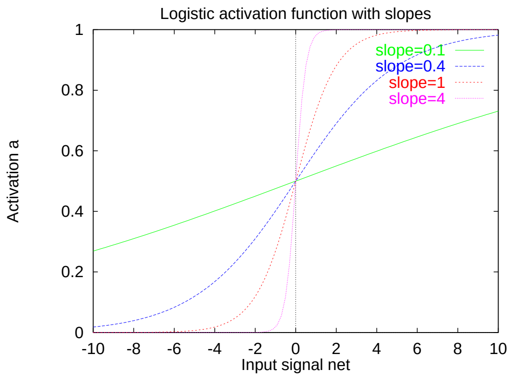


Figure 2.1: Logistic function with different slopes (γ).

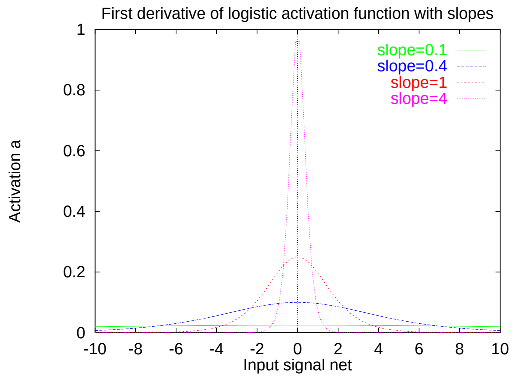


Figure 2.2: Derivative of logistic function with different slopes (γ).

The influence of the activation function slope on the network training should not be underestimated. For some problems the right choice can make a significant difference in learning speed. This can be observed in two experiments that are documented in the appendix A.1.2.

[1]Threshold units have been extensively investigated. If the fan-in of a unit is finite, which is always the case in computer simulations, then one can ask what precision the weights must have to warrant linear separability? In the early sixties Muroga, Toda and Takasu showed that $n \cdot \log n$ bits for each weight of the threshold unit are sufficient. For a long time it remained unclear whether this many bits were needed. In 1992 Hastad constructed a function which needed exactly $n \cdot \log n$ bits. A detailed overview of this work can be found in [71].

2.1.1.2 Update Schemes

A network changes its internal states in response to input values. The successive network state Z_{t+1} is determined by the activation functions of a subset V of units. This subset is chosen according to a given update scheme. Following [90] we can distinguish parallel, serial, synchronous and deterministic and non-deterministic schemes.

Definition 2.1.6 (Update schemes)
Let $V \subset U$ be the set of units to be updated at time t. An update scheme is called:

- serial update *scheme* $\Leftrightarrow |V| = 1$*: at time t only one unit changes its activation.*
- parallel update *scheme* $\Leftrightarrow |V| > 1$*: more than one unit changes its activation at a time.*
- synchronous update *scheme* $\Leftrightarrow V = U$*: the activities of all units change at the same time.*

An update scheme is called deterministic *if the subset selection of V is ruled by a deterministic process, otherwise the scheme is called* non-deterministic.

2.1.1.3 Topology of a Network

A network topology can be defined in terms of graph properties. The main distinction can be made between layered versus non-layered and undirected versus directed network graphs.

Definition 2.1.7 (Layered network graphs)
A network with structure (U, W) is called layered*, if each unit $u_i \in U$ can be assigned a unique number $layer(i) \in \mathbb{N}$ such that:*

$$layer(i) = \begin{cases} 0 & if\ \neg\exists(u_i, u_j) \in W \\ max\{layer(j) \mid \forall(u_i, u_j) \in W\} + 1 & else \end{cases} \tag{2.4}$$

The number of layers in a network is given by $k = max\{layer(i) \mid 1 \leq i \leq |U|\}$.

If a network graph consists solely of undirected edges (weights), then the network is called *undirected or symmetric*, otherwise it is called *directed*. Other properties of a network graph are its size (order of the network) and the number of edges (weights) and the interconnection structure.

There are basically two different operation modes in neural networks: feed-forward and relaxation.

Definition 2.1.8 (Feed-forward, relaxation)
An operation mode is called feed-forward *if:*

- *the network is layered and*
- *the input values are propagated from the input layer through the hidden units to the output layer.*

The activation of the units is done according to the layered structure of the network and the activations of the units in the output layer are considered as response or result of the network.
An operation mode is called relaxation *if:*

- *starting with an initial state Z_0, the network iterates through several states $Z_1, \ldots, Z_{n-1}$ according to its update scheme and*
- *the network settles in a stable state, that is, the network does not change its state after updating, or it oscillates between a (small) set of states. The final state (or set of states) is considered the result.*

A more detailed definition is given for the architecture of a network.

Definition 2.1.9 (Architecture)
A network architecture *can be described by a 5-tuple $A = (U, H, I, O, DE)$ with*

- *U as set of all units,*
- *H as set of h units: $H := \{u_1, u_2, .., u_h\} \subseteq U$,*
- *I as set of n input units: $I := U - H$,*
- *O as set of output units: $O \subseteq H$ and*
- *DE as set of directed edges: $DE \subseteq \{(u_i, u_j) | u_j \in H, u_i \in U, j < i\}$.*

If not specified the units of the networks are logistic units as defined in definition 2.1.5.

2.1.2 Learning in Networks

Learning in neural networks is accomplished through weight changes. We can distinguish between methods with an explicit learning goal and methods without an explicit learning goal. Methods of the first type are called supervised learning algorithms and methods of the latter type which usually have their learning goal incorporated into the learning algorithm are called unsupervised.

Definition 2.1.10 (Supervised and unsupervised learning)
A learning algorithm is called:

- supervised *if the quality of the network's output can be measured by an explicit learning goal and the learning goal is used for learning.*
- unsupervised *if structural properties, such as distributions or regularities of the input space are used by the learning algorithm to adapt the weights.*

Unsupervised learning algorithms are often used to find statistical properties of the input space, whereas supervised algorithms are mainly used for classification and prediction.

2.1.3 Backpropagation

Because of the relative ease of use and many successful applications, feed-forward networks are the most common type of networks used today. These networks are usually trained with the error-backpropagation training algorithm or variants of it. The following section introduces the backpropagation principle which was first proposed by Webros [92] and made popular by Rummelhart and McClelland in the mid-Eighties [69].

The basis for the backpropagation algorithm is the delta rule. It is based on the assumption that the correct output of every output unit is given for all training patterns. One can define the quality of the network's output by an error measure. The most common error function is the sum of squared errors.

Definition 2.1.11 (Squared error and mean sum of squared error)
Let θ be the actual target vector with n elements for the output layer with size n and a_i the activation of unit u_i then the squared error (se) *for a given pattern k can be defined as:*

$$se_k := \sum_{i=1}^{n} (\theta_i - a_i)^2 \tag{2.5}$$

For the backpropagation algorithm the squared error is often multiplied by 1/2 in order to simplify the derivative. The mean sum of squared errors (msse) *for all patterns can then be defined as:*

$$msse := \frac{\sum_{k=1}^{N_{pat}} se_k}{N_{pat}} \tag{2.6}$$

with N_{pat} as the number of patterns. To allow for an easier comparison the error is often normalized and reported as squared error percentage (sep)*:*

$$sep := 100 \cdot \frac{O_{max} - O_{min}}{n} \cdot msse \tag{2.7}$$

where O_{max} and O_{min} are the minimum and maximum values of the output units.

2.1.3.1 The Delta Rule

The basis to all backpropagation-like algorithms is the *delta rule* [69]. The error measure *msse* (2.6) shall be minimized through appropriate weight changes. We therefore need a weight change $\Delta w_{i,j}$ at unit a_i that is proportional to the error of the unit $-\frac{\delta E}{\delta w_{i,j}}$ (negative gradient)

$$w_{i,j}(t+1) = w_{i,j}(t) + \Delta w_{i,j}(t) \tag{2.8}$$

If the netinput net_i, the activation a_i and the target activation θ_i are known and the activation function f is differentiable, then the weight change is realized through:

$$\Delta w_{i,j} = -\frac{\delta E}{\delta w_{i,j}} \tag{2.9}$$

$$\frac{\delta E}{\delta w_{i,j}} = -(\theta_i - a_i) \cdot f'(net_i) \cdot a_j \tag{2.10}$$

$$= -\delta_i \cdot f'(net_i) \cdot a_j \tag{2.11}$$

If f is a linear function then the derivative $f'(net_i)$ is a constant and the result is the so-called *delta rule*.

Definition 2.1.12 (Delta rule, learning rate)
According to the delta rule *a weight change is accomplished through:*

$$\Delta w_{i,j} = \eta \cdot a_j \cdot \delta_i \tag{2.12}$$

with $\eta \in \mathbb{R}$ as learning rate *to control the size of the weight change.*

Normally θ is known only for the output units but not for inner (hidden) units of a network. The problem of assigning an error to such units is called *credit assignment problem* and cannot be solved by the delta rule. A possible solution to this problem is the *generalized delta rule* and the *backpropagation algorithm*. This algorithm delivers a way of assigning errors to hidden units and therefore overcomes the limitations of simple one-layered perceptrons. If the activation function f is differentiable then the generalized delta rule can be written as:

$$\Delta w_{i,j} = \eta \cdot \delta_i \cdot f'(net_i) \cdot a_j \tag{2.13}$$

with $\eta \in I\!R$ as learning rate. Thus, the backpropagation algorithm can be defined as:

Definition 2.1.13 (Backpropagation)
Let $o \in U$ be a unit of the output layer and θ_o the desired target value for o then, according to the delta rule, the following holds:

$$\Delta w_{o,i} = \eta \cdot \delta_o \cdot a_i \tag{2.14}$$

$$\delta_o = (\theta_o - a_o) \cdot f'(net_o) \tag{2.15}$$

Let I be the set of input units, H the set of inner (hidden) units, O the set of output units and $IH = I \cup H$ be the set of units that receive input from a unit $i \in IH$. The internal error for this unit i is given by:

$$\sum_{k \in IH \cup O} \delta_k \cdot w_{i,k} \tag{2.16}$$

Now we can calculate the internal error for all units of the same layer

$$\delta_i = \sum_{k \in IH \cup O} \delta_k \cdot w_{i,k} \cdot f'(net_i) \tag{2.17}$$

and their weight change

$$\Delta w_{i,j} = \eta \cdot \delta_i \cdot a_j \tag{2.18}$$

$$= \eta \sum_{k \in IH \cup O} \delta_k \cdot w_{i,k} \cdot f'(net_i) \cdot a_j \tag{2.19}$$

The backpropagation algorithm can be considered an iterative method for discrete gradient descent which minimizes the error between the desired target values and the actual output of a network (Eq. 2.5). The algorithm can be outlined as:

1. Initialization of weights $w_{i,j}$ with uniformly distributed random values[2].
2. Presentation of an input vector at the input units.

[2] Other initialization procedures such as Gaussian random numbers and weight initialization that depends on the number of weights connected to a unit can be used to find better starting points in the search space.

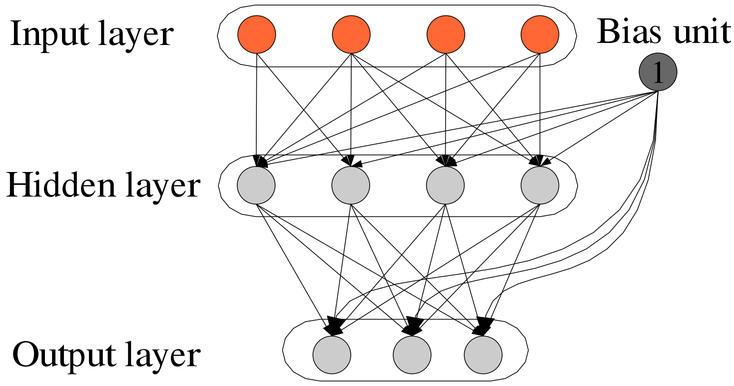


Figure 2.3: Architecture of a feed-forward network with a single hidden layer. Activation of the input units is usually given by presenting an input vector to the units while the activation of hidden and output units are computed through their activation functions. A forward activation is top-down while the error backpropagation is bottom-up.

3. Forward propagation of the signals, through all layers until the output layer is reached.
4. Presentation of the desired target vector and calculation of error signals.
5. Backward propagation of the error signals through all layers according to the generalized delta rule.
6. Iterate steps 2 - 5 until a desired stopping criterion is met[3].

The network architecture is illustrated in figure 2.3. Input units do not perform any computation on the input, they are merely used for presenting the training patterns to the network. Hidden and output units normally are "functional" units with sigmoid or other types of activation functions.

A presentation of all training patterns (target vectors θ) and the weight change is called *epoch*. We have two choices for changing weights. Adapting the weights after each single pattern presentation is called *online learning* which is the normal case. The second choice is the accumulation of error signals until all patterns are presented followed by a single weight update according to the accumulated error. This is called *batch learning*.

In chapter 4 we will see that the weights of a network can also be adapted by means of an evolutionary algorithm. In this case no backpropagation of error signals is needed and the necessity of differentiable activation functions can also be relaxed.

[3]A common stopping criterion is the *msse* error (Eq. 2.6). More advanced stopping criteria will be introduced in section 2.3.1.

2.1.3.2 Variants of Backpropagation

Since its invention the popular backpropagation algorithm has undergone major improvements. All of the improvements are used to speed up the learning process. Modifications vary from the introduction of a simple momentum term to more sophisticated changes of the algorithm like the very successful resilient propagation and the cascade correlation algorithm which also constructs the network structure during learning.

The momentum term introduces some sort of history into the gradient descent algorithm. It improves learning in flat areas of the search space and sometimes stabilizes the learning process.

Definition 2.1.14 (Momentum term)
The delta rule with momentum term *is defined as:*

$$\Delta w_{i,j}(t) := \eta \cdot \delta_i \cdot f'(net_i) \cdot a_j + \alpha \cdot \Delta w_{i,j}(t-1) \tag{2.20}$$

with α as new momentum parameter to control the influence the last gradient has on the actual weight change.

The backpropagation algorithm is very sensitive to variation of the two learning parameters, so it seems natural to either eliminate them or to control them through a proper heuristic. Amongst the possible improvements are the use of a hyperbolic function to change the learning rate or dynamic momentum strategies [33], [15].

2.1.3.3 Quickpropagation

Fahlman proposed a method to eliminate the momentum term and reduce the influence of the learning rate. His quickpropagation algorithm is based upon the second derivative of the activation function f and the assumption that the local error surface is hyperbolic [16].

Definition 2.1.15 (Quickpropagation update)
The quick propagation update *is defined as:*

$$\Delta w_{i,j}(t) := \frac{S(t)}{S(t-1) - S(t)} \cdot \Delta w_{i,j}(t-1) \tag{2.21}$$

$$S(t) := -\frac{\delta E}{\delta w_{i,j}(t)} \tag{2.22}$$

with $S(t)$ and $S(t-1)$ as first derivatives of the activation function.

According to this update strategy Δw will grow with successive weight changes in the same direction and decrease with alternating directions. This bears the danger of a too large Δw that might miss the minimum. The introduction of a maximum weight change parameter helps.

2.1.3.4 Resilient Propagation

Another important variant is the Resilient Propagation (RProp) batch learning algorithm. It introduces individual step sizes (learning rates) for each weight and a new mechanism to adapt them. One problem with standard backpropagation is that the step size control is directly based on the partial derivative of the units activation function. If a unit is operating in its saturated area, that is, close to the boundaries of its activation range, the derivative is small and the learning progress is very slow. One solution to this problem is to use only the sign of the derivative as gradient direction and determine the step size by other means.

The update rule for RProp [67, 65, 66] can be written as:

$$\Delta w_{i,j}(t) := -\Delta_{i,j}(t) \cdot sign(\frac{\delta E(t)}{\delta w_{i,j}}) \text{ with } sign(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{2.23}$$

The mechanism for step size control is:

$$\Delta_{i,j}(t) := \begin{cases} \eta^+ \cdot \Delta_{i,j}(t-1) & \text{if } \frac{\delta E(t-1)}{\delta w_{i,j}} \cdot \frac{\delta E(t)}{\delta w_{i,j}} > 0 \\ \eta^- \cdot \Delta_{i,j}(t-1) & \text{if } \frac{\delta E(t-1)}{\delta w_{i,j}} \cdot \frac{\delta E(t)}{\delta w_{i,j}} < 0 \\ \Delta_{i,j}(t-1) & \text{else} \end{cases} \tag{2.24}$$

with $\eta^+ > 1$ and $0 < \eta^- < 1$.

The update rule can be described as follows: Whenever the last step was too large and a local minimum was missed, the partial derivative of a weight $w_{i,j}$ changes its sign and the step size $\Delta_{i,j}$ is decreased by a factor η^- and the weight adaptation in the following learning step is omitted. This can be achieved by setting $\frac{\delta E(t-1)}{\delta w_{i,j}} := 0$. [4] If the direction remains the same, the step size is slightly increased by the factor η^+. Usually η^+ is fixed at 1.2 and η^- is fixed at 0.5. As a starting step size $\Delta_0 := 0.1$ is introduced and because of the potentially exponential growth of the step size it makes sense to introduce a maximum value $\Delta_{max} := 50.0$ for the step size as well.

[4] The original RProp algorithm performed a weight backtracking whenever a change in sign occurred. Recent work of Igel [32] showed that a backtracking mechanism that is based on the global instead of the local error is more successful. Omitting backtracking completely is better than local backtracking and almost as good as the one based upon the global error.

Because too large weights are usually the result of over-training, one might introduce a penalty term on the output weights to improve the generalization behavior. This weight penalty term can be added to the normal error term $msse$ (Eq. 2.6) and written as:

$$E := msse + 10^{-\alpha} \sum w_{i,j}^2 \tag{2.25}$$

with α as parameter to control the ratio of normal error and weight penalty. If $\alpha = 4$ the ratio is $1 : 10^4$.

A recent work of Schraudolph [72] proposed a new technique for adapting the step sizes for weight updates. He compared his method (ELK1) with four other stochastic gradient decent methods, and concludes that his algorithm is superior to others in terms of convergence speed as well as computational effort, while eliminating the momentum term.

Another important issue is the sensibility of the learning parameters. A parameter study of Braun [13] showed that the average number of training epochs needed to solve problems does not vary much over a wide range of potential parameter settings. In other words, RProp is not sensitive to learning parameter changes.

2.1.4 Complexity Issues

This section briefly discusses some complexity issues in neural networks on an informal basis. A more thorough coverage of the matter is given in appendix A.2.

The theorems of Judd [36] [37] and, based on them, the work of Lin and Vitter [47], show that learning as well as the problem of topology determination are NP-complete. By use of his theorems and their proofs Judd showed that the problem of finding a neural network architecture with the common logistic activation function that realizes a given function is NP-complete [34]. Restricting the activation function to subclasses does not help as the theorem even holds for simple linear threshold functions.

The question arises whether other restrictions might relax the problem. Placing constraints on the network architecture is a straightforward approach. Only for a very restricted class of "tree-like" architectures, Judd could show that learning can be accomplished in polynomial time.

His theorems are independent of the used training algorithm. They show that no training algorithm exists which can solve a given learning task in P-time if $P \neq NP$. Note that it also holds for training NNs with evolutionary algorithms.

Lin and Vitter [47] carried on the work of Judd. Their research is concerned with the complexity of learning specific mappings and the complexity of finding an optimal architecture for specific problems. Their work has shown that comparing the functionality of two given networks with more than one hidden unit is not feasible in the general case and that the problem of finding an equivalent or optimal network for a given network is NP-complete as well. Mandischer [51, 52] took this as one motivation for using an evolutionary algorithm as a heuristic search procedure for good architectures.

2.2 Data Preprocessing

The following sections deal with the problem of data preprocessing. The main focus will be on the introduction of scaling techniques that play a crucial role for the success of learning.

2.2.1 Scaling Techniques

An important issue in applying neural networks to a given dataset is the scaling of input and output vectors. For the desired target vectors, it is necessary to reside within the range of the output activation function in order to propagate a correct error signal. For input variables the range does not matter that much, because properly adjusted weights might perform a linear scaling, but input variables with very different input ranges make very different contributions to the error signal and weight change. Therefore it is desirable to rescale all variables to the same range.

Definition 2.2.1 (Scaling techniques)
Let $V := \{v_1, .., v_n\}$ *denote a set of values,* $\overline{v}$ *the mean of the set with standard deviation* σ_V *and* $\tilde{V}_{max}, \tilde{V}_{min}$ *the new range.*

- *Linear scaling*

  This simple scaling method can be used, if the data is uniformly distributed[5] *within the given range. Rescaling a value* $v \in V$ *can be done by:*

$$v' := \frac{v - \min(V)}{(\max(V) - \min(V))} \times (\tilde{V}_{max} - \tilde{V}_{min}) + \tilde{V}_{min} \tag{2.26}$$

- *Logarithmic Scaling*

  This method can be used if $max(V) > \exp(\overline{V})$ *and if larger values which will be squashed do not carry much information.*

$$v' := \frac{\log v - \log \min(V)}{\log(\max(V) - \min(V))} \times (\tilde{V}_{max} - \tilde{V}_{min}) + \tilde{V}_{min} \tag{2.27}$$

[5] A criterion for checking the distribution is to look for outliers that can be specified by the parameter λ. If $\max(V) < \overline{v} + \lambda\sigma_V$ or $\min(V) > \overline{v} - \lambda\sigma_V$ then linear scaling can be applied.

- *Hyperbolic scaling*

  This method can be used when values at the limits of the range can be neglected.

$$v' = 4v(1 - v) \tag{2.28}$$

For a more thorough overview of scaling methods see [84] and [29].

2.2.2 Choosing Target Values

Setting target values at the bound of the activation range sometimes has several drawbacks [45]. During training the network tries to push its output values as close as possible to the desired target values. As this is only possible with very large weights which cause the units to operate in their saturated areas where the sigmoid derivatives are close to zero, there is no weights change anymore and the weights may become stuck. In classification tasks, large weights also tend to force all outputs to the boundaries of the activation function, pretending a confidence that does not allow to differentiate between typical and untypical examples of the class.

As a solution to this problem, one can scale the input values to the range of the "active" part of the sigmoid instead to the boundaries of the activation function. For an activation range of $[0, 1]$ typical values for the output range are $[0.1, 0.9]$.

2.2.3 Dimensionality Reduction

The reduction of the input space is of some relevance for real-world applications, where not much is known about the data. Removing variables with few or no influence on the rest of the data means reducing the network's complexity and training time. High correlations between input values can be used to identify redundant inputs. Information theoretical measures like mutual information between each input variable [8] or statistical methods like principal component analysis [29] might be used.

2.3 Generalization

Minimizing the error of a network on a set of given training patterns does not necessarily mean minimizing the generalization error. When the training of a network starts, the network's error on the training set usually decreases while the weights adapt to fit the data. If the network has too many free parameters

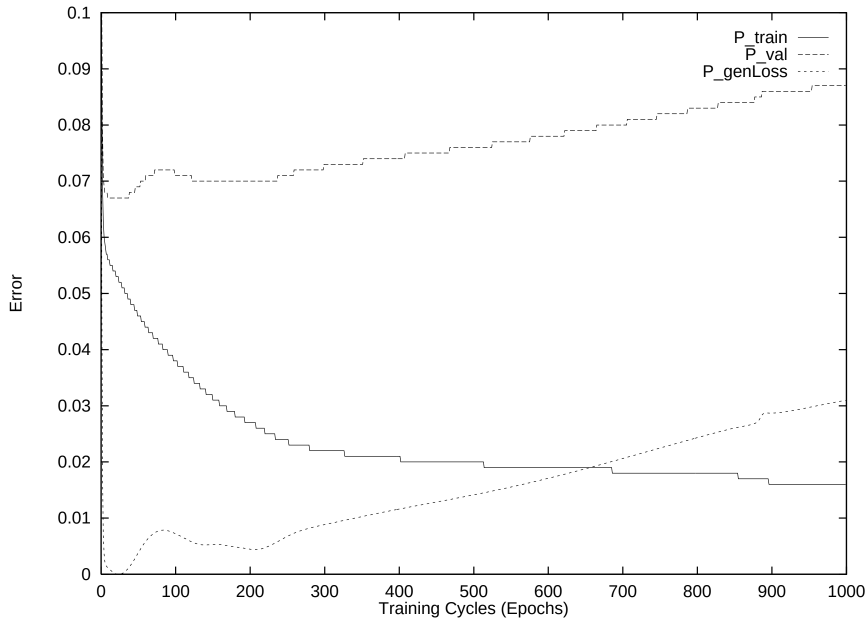


Figure 2.4: Overtraining and generalization loss (P_genLoss) during neural network training. Errors of two disjunct data sets are shown. The error on the training data (P_train) decreases while the error on unseen validation data (P_val) starts to increase after a few epochs.

(weights) and training continues to reduce the error for too long a time, over-fitting occurs. This means the network memorizes the data instead of developing a model of the input/output relation. The network's performance on unseen data, that is generalization ability, decreases. This tradeoff between generalization ability and accuracy can easily be observed in figure 2.4.

Achieving a good generalization ability still remains an open issue in training neural networks. Several techniques have been proposed to tackle this problem. The "correct" number of hidden-units plays a crucial role in this. Too many units make the network prone to over-fitting the data and too few result in a poor fitting of the data. Other methods which are independent of the architecture of a network are cross-validation, early-stopping and regularization.

White, Stinchombe, and Hornik [31] have shown that feed-forward networks are, in principle, capable of arbitrary exact function approximation given an infinite number of weights or units. In the context of this work Stinchombe [83] recapitulates some theoretical aspects of choosing a network architecture. He tries to answer the question how one can actually lean such arbitrary input/output relations and how well such a learning generalizes.

2.3.1 Cross-Validation and Early-Stopping

Cross-validation is a standard tool in statistics which is used to select good model parameterizations from a set of candidate parameterizations. Transfered to the neural network domain it is used as a means of improving the generalization of neural networks through partitioning of the available data, testing the performance on the different partitions, and selecting the best network parameterization, which is, in this context, the best set of weights.

First, the available data is divided into three disjunct and randomly sampled sets, the training, validation and test set. The idea is to use only the training set for the actual learning phase of the network and estimate its final performance by the unseen data from the validation set. The test set is held back and only used to evaluate the final performance of the network.

To clarify the terminology used throughout the experiments a few technical terms have been proposed by [62]. All data we have is simply called data. It will at least be divided into two disjunct sets. The first set, training data, is used for training and validating the network, the second set, the test set, is only used for testing the networks performance. The training data can be divided into a training set and a validation set so that we end up with three distinct data sets.

- **Training set** (50% of all data):
  This set is only for training the network. Weights are adjusted according to the error on this set only.

- **Validation set** (25% of all data):
  The validation set is part of the training data and it is usually not used to adjust the networks weights but to validate the results achieved on the training set. It is used during training to measure the expected generalization error.

- **Test set** (25% of all data):
  The test set is used to estimate the network's error on unseen data. The generalization error is measured. No weights are adapted to fit the test set.

Early-stopping is a so-called "non-convergent method" and is usually used in combination with cross-validation. The training process is stopped before it fluctuates around a minimum. When the error on the validation set starts to increase, the network is overfitting the training data and the final performance on unseen data decreases. The question arises when exactly one should stop the training. Prechelt proposed three classes of stopping criteria and evaluated several stopping criteria from these classes [62, 63].

Let $E_{training}, E_{validation}, and E_{test}$, be the $msse$ errors (Eq. 2.6) on the corresponding data sets, then the lowest validation error $E_{opt}(t)$ up to epoch t is defined as

$E_{opt}(t) := \min_{t' \leq t} E_{validation}(t')$. Thus the loss of generalization during training can be defined as:

$$GL(t) = 100 \cdot \left(\frac{E_{validation}(t)}{E_{opt}(t)} - 1 \right) \tag{2.29}$$

The training can be stopped as soon as the generalization loss GL exceeds a certain threshold α.

In most cases overtraining does not begin before the training error starts decreasing slowly. This leads to a measure which includes progress over a certain number of epochs, called *training strips of length k*.

$$P_k(t) := 100 \cdot \left(\frac{\sum_{t'=t-k+1}^{t} E_{training}(t')}{k \cdot \min_{t'=t-k+1}^{t} E_{training}(t')} - 1 \right) \tag{2.30}$$

The quotient of generalization loss and progress, $\frac{GL(t)}{P_k(t)}$, can be used to stop training if it exceeds the threshold α. Examples of stopping criteria are:

- GL_α: stop after first epoch t with $GL(t) > \alpha$
- PQ_α: stop after first end-of-strip epoch t with $\frac{GL(t)}{P_k(t)} > \alpha$
- UP_s: stop after epoch t if UP_{s-1} stops after epoch $t-k$ and $E_{validation}(t) > E_{validation}(t-k)$
  UP_1: stop after first end-of-strip epoch with $E_{validation}(t) > E_{validation}(t-k)$

A third type of criteria relies only on the sign of the validation error. Here the training stops when the generalization error increased in s successive training strips.

An extensive parameter study[6] of the above stopping criteria allows to draw some conclusion for the best trade-off between training time and resulting network performance [63]. If the expected network performance of a single training run has to be improved, the stopping criteria UP_3, UP_4, and UP_6 are fast and robust. If the best network can be picked from a series of several runs, one can use GL as criterion.

Other techniques for improving the generalization ability are adding noise to the input data or using a regularization term on the weights so that larger weights produce a larger error [84]. This prevents the network from memorizing specific patterns. Both methods are proven to be equivalent under certain assumptions [10]. A way of integrating a regularization term is to define an error which penalizes larger weights more than smaller once.

[6]In this study, 12 different network architectures had to solve 12 different learning tasks.

Such regularization terms do not take into account the cross-correlation between hidden units and the distribution of the data in the input units. Information-theoretic regularization methods may be more appropriate in this case [14].

2.3.2 Choosing the Optimal Number of Units

As we have seen in the previous section, a good generalization ability depends on the training time of the network. Another parameter is the number of hidden units and thus the number of weights. The optimal number of hidden units depends on several factors:

- the amount of available training data,
- the complexity of the task to be learned,
- the number of input and output units,
- the architecture (node functions and structure),
- the training algorithm (training with weight decay, regularization, early-stopping).

With too few units one gets high training and high generalization errors, and with too many hidden units and without early stopping one may get a low training error but high generalization error. There is no reliable way to determine how many hidden units are needed without trial and error. There are some rules of thumb on how to choose the number of hidden units to achieve a good generalization. Rules that derive the number of hidden units solely from the number of inputs and outputs can easily be disproved by counterexamples[7].

Other rules are derived from theoretical considerations of lower and upper bounds of the number of patterns needed for good generalizations in a given architecture. These rules do not take into account the complexity of the task to be learned.

The complex interdependencies between the factors do not allow simple rules to be applied. An interesting aspect is that it must not be the goal to achieve a minimal number of units in the network but to create a network that finally gives a good generalization regardless of the number of units used. If good generalization can be achieved by means of early stopping, then there is no objection against a large number of hidden units.

[7] The rule: “Never choose h to be more than twice the number of input units.” [84] p. 55, can be proven wrong by the two-spirals benchmark (see section 4.3.5 for a network that needs more than 20 hidden units while having 2 inputs and 1 output.)

Chapter 3

Evolutionary Algorithms

In this chapter an outline of evolutionary algorithms is given. The focus is on techniques which are later on employed to evolve neural network weights. A valuable source of information and the state-of-the-art is the "Handbook of Evolutionary Computation" [4].

The basis of artificial evolution is a set of individuals which establishes a population. The better an individual is adapted to its environment, the greater is its chance to survive and produce offspring. Each individual can be seen as a point in the search space which is, in our case, the space of possible networks or the weight space of a single network. Each individual is evaluated and assigned a fitness value which reflects its ability to survive in the given environment. An evolutionary algorithm can be seen as an iterative search procedure, where each iteration is called generation. During each iteration the three biologically inspired principles of *selection*, *mutation*, and *reproduction* are applied to the population. In general the selection mechanism determines which individuals are considered for the next generation but the role of selection is somewhat different for different evolutionary algorithms.

Another difference between EAs lies in their representation. In nature the properties of an individual are determined by its genes and ontogenesis, that is, the mechanism that uses their contents to build up an individual. Genetic information is coded in chromosomes and defined by sequences of four nucleotide bases: adenine (A), cytosine (C), guanine (G), and thymine (T). The carrier of this information is a molecular structure called DNA (deoxyribonucleic acid). The bases can be seen as nature's alphabet for coding the building plan of an organism. The genes are short strands of triplets of bases (A,C,G,T) that code amino acids. The DNA of an organism is a long linear molecule structured as a double helix that is composed of sequences of nucleotide bases and contains all information to build an individual. In evolutionary algorithms the alphabets are much simpler and the DNA is often reduced to linear sequences.

While the "traditional" versions of evolutionary algorithms like evolution strategies, genetic algorithms, and genetic programming [7] were usually associated with their own native representations[1], real-valued, binary or expressions in a programming language, nowadays strict boundaries or fixed problem domains vanish. Problems are solved using techniques and representations from all types of EAs, and a cross-fertilization between the formerly disjunct algorithms can be observed.

A fairly general and informal algorithmic view is given by the following pseudo code:

Algorithm 1 (Outline of an evolutionary algorithm)

$t := 0$
initialize $P(0)$
evaluate $P(0)$
while $(terminate(P(t)) \neq \textbf{true})$ **do**
    recombine: $P'(t) := rec(P(t))$
    mutate: $P''(t) := mut(P'(t))$
    evaluate $P''(t)$
    select: $P(t+1) := sel(P''(t) \cup Q)$
    $t := t + 1;$
end

- with t as generation counter,
- $P(t) = \{\vec{a}_1, \ldots, \vec{a}_\mu\} \in I^\mu$ as an initial population of μ individuals $\vec{a}_i \in I$, where the structure of the individual space I might range from simple boolean spaces $\mathbb{B}^n$ to complex data structures,
- mut and rec as mutation and recombination operators that are defined on populations,
- $Q \subseteq \{\emptyset \cup P(t)\}$ as an additional set of individuals which is used to model individuals surviving more than one generation,
- sel as operator that selects individuals for the next generation.

The recombination operator often comprises another selection mechanism, the *mating-selection*, which determines which individuals are chosen to produce offspring together. The mating-selection can be either seen as an integral part of the recombination operator or explicitly modeled as an operator. The evolution cycle can then be characterized by figure 3.1.

[1] In the past such associations were often done to allow for theoretical investigations. The ES for example was first defined and used for discrete problems.

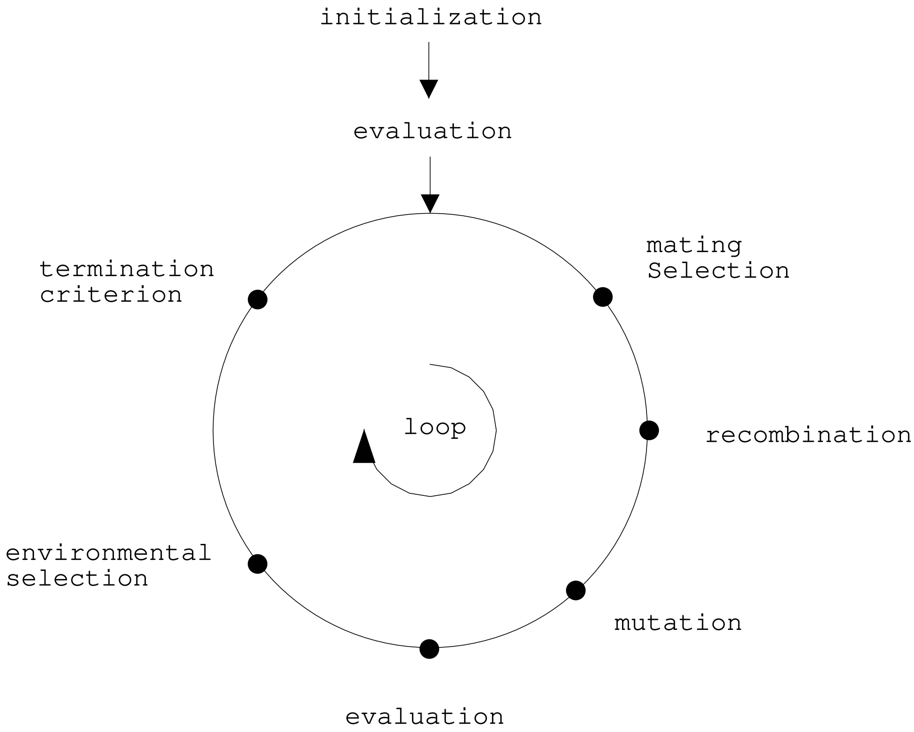


Figure 3.1: Iteration scheme of a general EA (from [77]).

In the next sections evolution strategies (3.1) and genetic algorithms (3.3) will be introduced using a general and informal notation that is gleaned from Bäck [2].

3.1 Evolution Strategies

The old notation for evolution strategies (ESs) introduced by Schwefel in the early Seventies is the $(\mu \overset{+}{,} \lambda)$–ES notation which means that μ parents create λ descendants by recombination and mutation. The μ parents for the next generation are either chosen from a union of parents and descendants in case of the $(\mu{+}\lambda)$–ES with $\lambda \geq 1$ or only from the descendants in case of the $(\mu{,}\lambda)$–ES with $\lambda > \mu$.

Newer variants of evolution strategies relax the strict distinction between "+" and "," selection by introducing the concept of aging [79, 78]. This $(\mu, \kappa, \lambda, \rho)$-ES introduces two new parameters. The parameter κ controls the maximum number of reproduction cycles an individual is allowed to live. With a value of $\kappa = 1$ one gets the classical $(\mu{,}\lambda)$–ES while a value of $\kappa = \infty$ yield the $(\mu{+}\lambda)$-strategy.

With the parameter ρ one can control the number of parents that contribute to the creation of one offspring.

The operators of an ES work on entire populations. It is left to the operator how many individuals are chosen for processing. In general it can be distinguished between asexual, sexual and global operators. Asexual operators work on a single individual, while sexual ones pick two individuals to produce offspring in contrast to the global operators that might use any number of individuals[2].

3.1.1 Representation

For the class of real-valued or discrete parameter optimization problems, evolution strategies work directly on their representation, and in this case no mapping function between genotype and phenotype is needed, or it is included in the fitness function. Search points in ESs are, for example, n-dimensional vectors $\vec{x} \in \mathbb{R}^n$ of object variables. Hence, the fitness function is, in principle, identical to a given objective function $f : \mathbb{R}^n \to \mathbb{R}$. If the optimization problem is not naturally defined on $\mathbb{R}^n$, which is often the case in structure optimization where one might face a mixture of binary, integer and real-valued vectors of varying sizes, mapping functions or problem specific representations and operators are needed. To allow for a better adaptation to the objective functions topology, the object variables are accompanied by a set of so-called *strategy parameters*.

Definition 3.1.1 (ES-individual)
An individual $\vec{a} = (\vec{x}, \vec{\sigma}, \vec{\alpha}) \in I$ of an ES *consists of three components, the object variables $\vec{x}$ and up to n different standard deviations[3] σ_i to control the step sizes and up to $n \cdot (n-1)/2$ rotation angles $\alpha_{j,i} \in [-\pi, \pi]$, where*

$$
\begin{aligned}
I &= \mathbb{R}^n \times A && (3.1)\\
A &= \mathbb{R}_+^{n_\sigma} \times [-\pi, \pi]^{n_\alpha} && (3.2)\\
n_\sigma &\in \{1, \ldots, n\} && (3.3)\\
n_\alpha &\in \{0,\ (2n - n_\sigma)(n_\sigma - 1)/2\}. && (3.4)
\end{aligned}
$$

In case we have $1 < n_\sigma < n$, the standard deviations $\sigma_1, \ldots, \sigma_{n_\sigma - 1}$ are used for the corresponding object variables $x_1, \ldots, x_{n_\sigma - 1}$, and the last σ_{n_σ} is used for the remaining variables $x_{n_\sigma}, \ldots, x_n$.

[2] Sometimes global recombination is also called panmictic recombination [2] which might lead to confusion with panmictic population structures.

[3] The number of strategy parameters can be up to $n \cdot (n+1)/2$.

3.1.2 Mutation

Mutation is an asexual operator that involves only a single individual in contrast to recombination where up to μ individuals can be used to form a new individual. The mutation operator $mut : I \rightarrow I$ is applied to a triple $(\vec{x}, \vec{\sigma}, \vec{\alpha}) \in I$ and yields a new triple $(\vec{x}', \vec{\sigma}', \vec{\alpha}') \in I$. Object variables as well as the strategy parameters of an individual undergo mutation.

The notation $N(0,1)$ denotes a realization of a standard normally distributed one-dimensional random variable having expectation zero and standard deviation one[4]. Instead of a normal distribution the more slowly decaying Cauchy distribution might be used [38].

Definition 3.1.2 (ES-mutation)
The mutation operator is defined as follows: $\forall i \in \{1, \ldots, n\}$, $\forall j \in \{1, \ldots, n \cdot (n-1)/2\}$

$$\sigma_i' = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)) \tag{3.5}$$

$$\alpha_j' = \alpha_j + \beta \cdot N_j(0,1) \tag{3.6}$$

$$\vec{x}' = \vec{x} + \vec{N}(\vec{0}, \mathbf{C}(\vec{\sigma}', \vec{\alpha}')) \quad . \tag{3.7}$$

- *with* $\exp(\tau' \cdot N(0,1))$ *as a global factor which allows an overall change of the mutability,*
- $\exp(\tau \cdot N_i(0,1))$ *allowing for individual changes of the "mean step sizes"* σ_i *and*
- $\beta \approx 0.0873$ *which equals* $5°$ *rotation angle.*

Before the object variables are changed, the standard deviations are mutated using a multiplicative, logarithmic normally distributed process. The rotation angles are then varied by an additive, normally distributed process. Finally, the resulting vectors $\vec{\sigma}'$ and $\vec{\alpha}'$ are used to create a random vector for mutation of the object variable vector $\vec{x}$.

Rotation angles are not used in this thesis so that an individual consists of only the object parameters and the standard deviations. The space of individuals for the ES in this thesis is reduced to $I = \mathbb{R}^n \times \mathbb{R}^n$ and the above equations can be reduced to:

[4]The notation $N_i(0,1)$ indicates that the random variable is sampled anew for each i. The notation $\vec{N}(\vec{0}, \mathbf{C})$ is used to denote a realization of a random vector distributed according to the generalized n-dimensional normal distribution with expectation $\vec{0}$ and covariance matrix $\mathbf{C}^{-1} = \mathbf{C}^{-1}(\vec{\sigma}, \vec{\alpha})$. For further details see [2], [75].

$$\sigma_i' = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)) \tag{3.8}$$

$$x_i' = x_i + \sigma_i' \cdot N_i(0,1) \tag{3.9}$$

The parameters τ' and τ can also be interpreted in the sense of "global learning rates" as in neural networks. From a somewhat different perspective one can see the standard deviations σ_i as "local learning rates" when they are coupled to the object parameters during recombination. Often they are set as follows (see [73]):

$$\tau' \sim \frac{\varphi^*}{\sqrt{2n}} \tag{3.10}$$

$$\tau \sim \frac{\varphi^*}{\sqrt{2\sqrt{n}}} \tag{3.11}$$

with φ^* as expected rate of convergence. As the convergence rate is only known for theoretical or empirical well researched objective functions like the sphere function, it is usually set to 1 for unknown functions. If we consider the case of only one step size $n_\sigma = 1$, the global and the individual factor for the modification of the σ_i merge into one common factor τ' where $\tau' \sim (\sqrt{n})^{-1}$.

$$\sigma' = \sigma \cdot \exp(\tau' \cdot N(0,1)) \tag{3.12}$$

$$x_i' = x_i + \sigma' \cdot N_i(0,1) \tag{3.13}$$

Recent parameter studies of Kursawe [43] showed that there is a complex interdependence between the recombination operators, the objective function and the selection pressure and that no single "correct" parameter setting of an ES exists, that holds for all problems.

The logarithmic normal distribution for the variations of standard deviations σ_i can be motivated as follows (see Schwefel [73]):

- The multiplicative modification of σ_i ensures positive values of standard deviations.
- To guarantee the neutrality (on average) of the process without selective pressure, a multiplication by a certain value must occur with the same probability as a multiplication by its reciprocal value.
- Smaller modifications must occur more often than larger ones.

Practically it is possible for the standard deviations to become almost zero by the multiplicative process. Therefore a minimum value must be maintained to guarantee a measurable change in the object variables.

An alternative mechanism for the self-adaption of strategy parameters has been proposed by Ostermeier et al. [60]. They employed a derandomized scheme for the mutation of step sizes that uses information accumulated over previous generations. An overview of self-adaption methods is given by Bäck [3].

3.1.3 Recombination

Recombination in ESs can be either *sexual*, where only two parents are involved in the creation of an offspring, or *global*, where up to the whole population contributes to a new offspring. Sexual recombination of just two individuals is often called *local* while the contribution of all individuals is called *global* recombination. Traditional recombination operators are *discrete recombination*, *intermediate recombination*[5], and *geometric recombination*, all existing in a sexual and global form. Another form of recombination, the *global average*, where the components' mean over the entire population is calculated, is also called global intermediate recombination [64, 9].

Beside the global and local version of the operators, one has to decide if object variables and strategy parameters are coupled to each other. Traditionally, both parameter sets are treated independently of each other, and recombination samples genes from distinct pools of object variables and strategy variables so that an offspring might get its x_i vector component from parent P_1 and its σ_i component from a different parent P_2. Nevertheless, many implementations of ESs let object and strategy variables travel together, which will then be called *coupled* recombination.

Definition 3.1.3 (ES-recombination)
The recombination operator creates an individual $\vec{a}' = (\vec{x}', \vec{\sigma}')$ from a population $P(t) \in I^\mu$, i.e. $rec : I^\mu \to I$. In case of the sexual form we have $rec : I^2 \to I$. For the object variables and strategy parameters ($\forall i \in \{1, \ldots, n\}$):

$$x'_i = \begin{cases} x_{S,i} & \textit{no recombination} & (n) \\ x_{S,i} \text{ or } x_{T,i} & \textit{discrete} & (d) \\ x_{S_i,i} & \textit{global discrete} & (D) \\ (x_{T,i} + x_{S,i})/2 & \textit{intermediate} & (i) \\ (x_{T_i,i} + x_{S_i,i})/2 & \textit{global intermediate} & (I) \\ (\sum_{K=1}^{\mu} x_{K,i})/\mu & \textit{global average/intermediate} & (A) \end{cases}$$

[5] Schwefel [78, 79] proposed to generalize intermediate recombination by introducing weight factors from the interval $[0, 1]$ instead of the value 0.5 and furthermore by choosing the weight factor anew for each component in case of global intermediate recombination.

$$\sigma'_i = \begin{cases} \sigma_{U,i} & \textit{no recombination} & (n) \\ \sigma_{U,i} \text{ or } \sigma_{Q,i} & \textit{discrete} & (d) \\ \sigma_{U_i,i} & \textit{global discrete} & (D) \\ (\sigma_{Q,i} + \sigma_{U,i})/2 & \textit{intermediate} & (i) \\ (\sigma_{Q_i,i} + \sigma_{U_i,i})/2 & \textit{global intermediate} & (I) \\ \sqrt{(\sigma_{Q,i} \cdot \sigma_{U,i})} & \textit{geometric} & (g) \\ \sqrt{(\sigma_{Q_i,i} \cdot \sigma_{U_i,i})} & \textit{global geometric} & (G) \\ (\sum_{K=1}^{\mu} \sigma_{K,i})/\mu & \textit{global average/(intermediate)} & (A) \end{cases}$$

with S *and* T *denoting two randomly selected individuals from the parent population. In case of a* coupled *recombination we claim* $S = U$, $S_i = U_i$, $T_i = Q_i$. *Otherwise, it may or may not be assured that* $S \neq U$, $S_i \neq U_i$, $T_i \neq Q_i$.

The index i in T_i, S_i, U_i and Q_i indicates that they are newly sampled for each component i, so that a child that is created via global recombination gets its information from two newly chosen parents for each component. In Bäck [2] we find a somewhat different notation where S is used instead of S_i, which is equivalent to holding one parent fixed and choosing only the second parent anew for each component.

Throughout the thesis a recombination operator will be denoted by $(rec_{\vec{x}} rec_{\vec{\sigma}})$, where $rec_{\vec{x}} \in \{n, d, D, i, I, A\}$ and $rec_{\vec{\sigma}} \in \{n, d, D, i, I, g, G, A\}$ (according to the notation in the last column of 3.14) indicate recombination mechanisms used on the two components of individuals. If not mentioned otherwise the sexual and coupled recombination is used. Table 3.1 summarizes the different recombination operators.

3.1.4 Selection

The environmental selection operator employed in evolution strategies is completely deterministic - and there exist two basic selection methods, the $(\mu{+}\lambda)$-selection and the (μ,λ)-selection - while the mating selection operator is completely (uniformly) random.

Definition 3.1.4 ($(\mu{+}\lambda)$-selection)
The selection operator $sel_{(\mu+\lambda)} : I^{\mu+\lambda} \to I^{\mu}$ *forms the next generation by selecting the* μ *best individuals from the union of* λ *parents and* μ *offspring. Here,* $\lambda \geq 1$ *and independent of* μ.

Definition 3.1.5 ((μ,λ)-selection)
The selection operator $sel_{(\mu,\lambda)} : I^{\lambda} \to I^{\mu}$ *forms the next generation by selecting the* μ *best individuals from the* λ *offspring. Thus,* $\lambda > \mu$ *is a necessary condition.*

	sexual (local) recombination	global recombination
discrete	Select two arbitrary parents, choose each component of an offspring at random from one of the two parents.	For each component of the offspring randomly select a parent from the whole population.
intermediate	Select two arbitrary parents, for each component of an offspring calculate the arithmetic mean of both parents.	For each component of the offspring randomly select two parents from the whole population, calculate the arithmetic mean of both parents.
geometric (only allowed on σ which are always positive)	Select two arbitrary parents, for each component of an offspring calculate the geometric mean of both parents.	For each component of the offspring randomly select two parents from the whole population, calculate the geometric mean of both parents.
global average global intermediate	Each component of an offspring is built by calculating the arithmetic mean of the entire population for that component.	

Table 3.1: Recombination types in ESs.

The $(\mu+\lambda)$-selection guarantees the survival of the fittest individual and is therefore a so-called *elitist selection*: once a (local) optimum is found, it will be kept until a better optimum is found. This selection mechanism has several drawbacks in comparison to the (μ,λ)-selection that restricts the lifetime of every individual to one generation. In changing environments it keeps outdated solutions and cannot follow a moving optimum. In case of multimodal objective functions, the strategy may get practically stuck[6] in local optima. Also, the self-adaptation mechanism for the strategy parameters might be hindered, because poorly adapted strategy parameters may survive when they cause a fitness improvement by chance.

According to [2], the conditions for a successful self-adaptation of strategy parameters can be summarized as follows:

- Use a (μ,λ)-strategy in order to prevent survival of individuals with poorly adapted internal parameters.

[6]The logarithmic normal distribution allows arbitrarily large or small changes such that the strategy can in theory escape from local optima and alway find the global optimum for $t \to \infty$ as long as $\sigma_t > 0$ and $\sum_{t=0}^{\infty} \sigma_t \to \infty$.

- The selective pressure should not become too strong, which means that the number of parents μ is required to be significantly larger than one. A ratio of $\mu/\lambda \approx 1/7$ is recommended.
- Recombination on strategy parameters is necessary and intermediate recombination usually gives best results.

The ratio λ/μ is called *selection pressure* and allows for the characterization of the search process. A small μ emphasizes a more local search (path-oriented), while a large μ leads to a more global search (volume-oriented) [2].

Another rather old selection mechanism is the $(\mu{+}1)$-selection [64, 5] or, in GA terminology, steady-state selection [93]. It is often discarded because is does not offer an appropriate method for self-adapting the strategy variables [76, 81]. However, it might be useful when the evaluation of a fitness function is done in parallel and the computational effort for a single fitness evaluation varies a lot. In such a case a generational approach wastes time while waiting for the fitness value of the slowest individual.

Algorithm 2 ((μ+1) or steady-state evolutionary algorithm)

$t := 0$
initialize $P(0) + \{\vec{a}_1, \ldots, \vec{a}_\mu\} \in I^\mu$
evaluate $P(0)$
while $(terminate(P(t)) \neq$ **true**$)$ **do**
    recombine: $\vec{a}' := rec(P(t))$
    mutate: $\vec{a}'' := mut(\vec{a}')$
    evaluate $\vec{a}''$
    select: $P(t+1) := sel(P(t) \cup \vec{a}'')$
    $t := t + 1;$
end

Other selection methods like *tournament selection* [11] and *rank-based selection* [26] are not covered here. A comparison of different selection methods can be found in [28].

3.1.5 Convergence Measures

When doing parameter studies the question of how to evaluate a single run or a set of runs and when to stop because of premature stagnation arises. Close to a minimum, either the global or some other local one, optimization procedures tend to reduce their step size towards the minimum to approach it even further. In a very flat area, which is similar to a constant objective function, intermediate recombination of the step sizes causes them to grow. In both cases the search practically stagnates.

In the neural network domain, the technique of early-stopping exists (see section 2.3.1) that is used to prevent over-adaptation of parameters . An ES run can, for example, be stopped if a certain difference between the best and the average or the worst individual becomes smaller then a given ε or when the step sizes begin to vanish below a very small value.

The following two convergence measures can be used to evaluate the speed of convergence.

$$C_1 = \frac{n}{\#\text{generations}} \cdot \frac{1}{2}(\log f_{start} - log f_{best}) \tag{3.14}$$

This measure is influenced by the number of generations and favors quick runs, even if they get stuck at local optima. The next measure is independent of the number of generations:

$$C_2 = \log \frac{f_{start}}{f_{best}} \quad . \tag{3.15}$$

3.2 Encapsulated Evolution Strategies

Encapsulated evolution strategies (EESs) [64] belong to the class of meta strategies. They are successfully applied to non-linear regression problems in chemical engineering [42, 87]. Within the scope of this thesis (see chapter 5) they are used to fit physical models to experimental data in order to yield good models of the underlying process [53].

The main characteristic of an EES is its sequential and isolated search strategy on several (usually two) levels. The isolation leads to a higher locality of the search which turned out to be useful in complex search spaces.

EESs use the standard $(\mu\,,\,\lambda)$- and $(\mu + \lambda)$-ES [75] with step size adaptation and correlated mutations, if necessary, on both search levels. All mutation and recombination operators are identical to those described earlier. Accordant with the above notation, Geyer [21] proposed an extended notation for EESs which is based on Rechenberg's notation ([64], page 92 and 158):

$$\left[rec_{\vec{x},1} rec_{\vec{\sigma},1} rec_{\vec{\alpha},1}\ \mu_1 \overset{+}{,} \lambda_1\ \ (rec_{\vec{x},2} rec_{\vec{\sigma},2} rec_{\vec{\alpha},2}\ \mu_2 \overset{+}{,} \lambda_2)^{\gamma_2}\right]^{\gamma_1} - \text{ ES} \tag{3.16}$$

with γ_2 as number of generations for the subpopulations and γ_1 as number of generations on the meta level. The triplet code $rec_{\vec{x},i} rec_{\vec{\sigma},i} rec_{\vec{\alpha},i}$ of the notation denominates the recombination type used on each of the search levels. Recombination is defined for

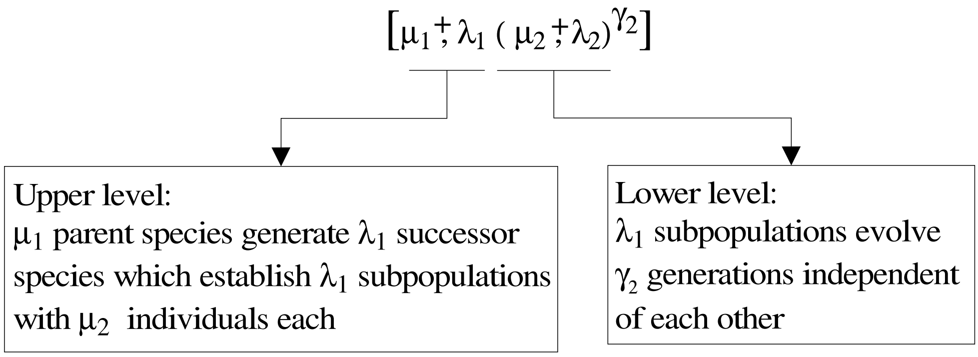


Figure 3.2: Search levels of an encapsulated evolution strategy.

- object variables $\vec{x}$,
- step sizes $\vec{\sigma}$ (standard deviations), and
- the rotation angles $\vec{\alpha}$, if correlated mutations are used.

Again, recombination types can be chosen from the standard set of recombination operators $rec_{\vec{x},i}, rec_{\vec{\sigma},i}, rec_{\vec{\alpha},i} \in \{n, d, D, i, I, g, G, A\}$. The working principle of an encapsulated evolution strategy without explicitly given recombination types is illustrated in figure 3.3.

Each of the newly generated λ_1 children on the upper level is the basis for a new subpopulation that evolves for γ_2 generations within γ_1 generations in the outer loop. On the lower level every subpopulation evolves independently of the others. Every child λ_1, which is the founder of a new subpopulation and comes from the upper level, is duplicated μ_2 times and functions as a starting point for the lower level subpopulation. Each subpopulation starts with the same object variable vector $\vec{x}$ as its ancestor but with newly initialized strategy parameters $\vec{\sigma}$ and rotation angles $\vec{\alpha}$.

After γ_2 iterations the object variable vector of the best individual of each subpopulation is returned to the upper (meta) level. This means that every individual on the upper level copies itself into subpopulations which evolve for some time. Each individual on the upper level is then replaced by the best individual of it's subpopulation. Due to this process each individual changes it's position in the search space. After the evolution of the subpopulations and transfer of the best individuals to the upper level, the selection process continues on the upper level as usual.

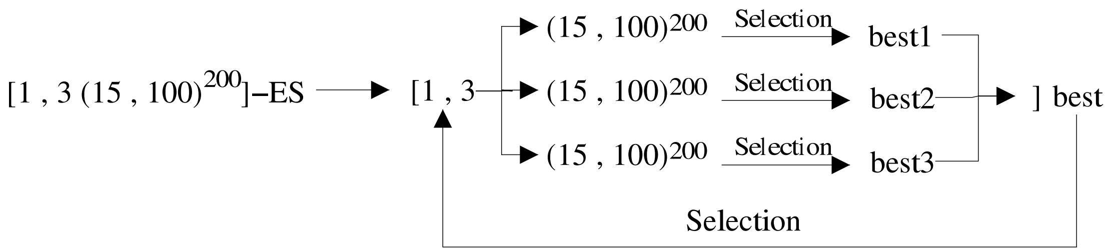


Figure 3.3: Example of an encapsulated evolution strategy.

3.3 Genetic Algorithms

3.3.1 Representations

A classical genetic algorithm [25] works on a binary search space $I = \mathbb{B}^n$, hence individuals are bitstrings of fixed length n. In most cases the objective function is not a boolean function and we have a clear distinction between the genotype and the phenotype of an individual. Hence, a transformation mechanism, a “genotype-phenotype mapping” from the binary coded genotype of an individual to its phenotypic representation, is needed to evaluate an individual. The space of phenotypes shall not be specified. It can range from $\mathbb{R}^n$ as in evolution strategies to arbitrarily complex structures in the case of neural networks.

An individual in GA notation can be defined as follows:

Definition 3.3.1 (GA-individual, search space, population)
Let $\mathbb{B}$ be the alphabet for encoding, then let the search space $I := \mathbb{B}^n$, and we define an individual (or string) as $s := (x_1, \cdots, x_n) \in \mathbb{B}^n$ with length $|s| := n$. A set $P := \{s_1, \cdots, s_p\} \subset \mathbb{B}^n$ is called population.

3.3.2 Operators

Holland described three different genetic operators in 1975: inversion, mutation and crossover [30].

Definition 3.3.2 (GA-inversion)
Let $s := (x_1, \cdots, x_n) \in \mathbb{B}^n$ be a string and $p_1, p_2 \in \mathbb{N}$ with $1 \leq p_1 < p_2 \leq n$ two positions within the string. A mapping $inv : \mathbb{B}^n \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}^n$ with:

$$s' = inv(s, p_1, p_1) :\Leftrightarrow \tag{3.17}$$

$$s'_i = \begin{cases} s_i & , \forall i < p_1 \wedge \forall i > p_2 \\ s_{p_2+p_1-i} & , \forall p_1 \leq i \leq p_2 \end{cases} \tag{3.18}$$

is called inversion operator.

The inversion operator reverses the order between the string positions p_1 and p_2. Figure 3.4 illustrates this.

Figure 3.4: GA-inversion operator.

The role of mutation in GAs is somewhat different from that of mutation in ESs. The rate at which mutations occur is usually very low ≈ 0.001 (on average every 1/1000 bit is flipped) in a GA. The mutation operator is often seen as a "background" operator which does not operate on individual strings but on the whole population.

Definition 3.3.3 (GA-mutation)
Let $P := \{s_1, \ldots, s_p\}$ be a population of strings $s_i \in \mathbb{B}^n$ and $S = s_1 s_2 \ldots s_p$ a concatenation of all strings. A mapping $mut : \mathbb{B}^{n \cdot p} \rightarrow \mathbb{B}^{n \cdot p}$ with

$$S' = mut(S) :\Leftrightarrow \tag{3.19}$$

$$S'_i = \begin{cases} \neg S_i & \text{with probability } p_{mut} \\ S_i & \text{else} \end{cases} \tag{3.20}$$

is called mutation operator with $p_{mut} (0 \leq p_{mut} \leq 1)$ as mutation rate.

For every position in the population, the mutation operator realizes a bit flip with probability p_{mut}. With a low mutation rate, it is the main purpose of the operator to reestablish lost bits in the gene pool. Properties that were lost because of selection can be reintroduced by mutation. In figure 3.5 the mutation operator is illustrated.

Figure 3.5: GA-mutation operator for an individual.

In GAs recombination is thought to be the most crucial mechanism for diversity creation and it is realized via sexual crossover operators. Two parent individuals s and t are selected with probability p each and information from both is transfered to their offspring by crossover. With probability $(1-p)$ the individual remains unchanged. The classical operator is the one-point crossover.

Definition 3.3.4 (GA-one-point crossover)
Let s and t be two strings of length n, and p_1 $(1 < p_1 < n)$ be a crossover point. A mapping $cross_1 : \mathbb{B}^n \times \mathbb{B}^n \times \{1, \cdots, n-1\} \rightarrow \mathbb{B}^n \times \mathbb{B}^n$ with

$$(s', t') \quad = \quad cross_1(s, t, p_1) :\Leftrightarrow \tag{3.21}$$

$$(s'_i, t'_i) \quad = \quad \begin{cases} (s_i, t_i) & \forall i < p_1 \\ (t_i, s_i) & \forall i \geq p_1 \end{cases} \tag{3.22}$$

is called one-point crossover operator with $p_{cross1}(0 \leq p_{cross1} \leq 1)$ as crossover probability for this operator.

The one-point crossover exchanges parts of two strings which lay beyond the given crossover position, which is identical for both strings. Two parents of length n yield two offspring with the same length (see figure 3.6).

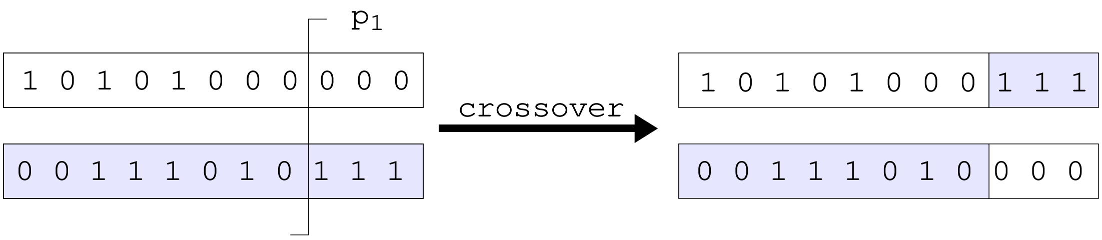


Figure 3.6: GA-one-point crossover operator.

The following two-point crossover is an extension which exchanges information between two crossover points (see figure 3.7).

Definition 3.3.5 (GA-two-point crossover)
Let s and s' be two strings of length n and p_1, p_2 $(1 < p_1 < p_2 < n)$ two crossover points. A mapping $cross_2 : \mathbb{B}^n \times \mathbb{B}^n \times \{1, \cdots, n-2\} \times \{2, \cdots, n-1\} \rightarrow \mathbb{B}^n \times \mathbb{B}^n$ with:

$$(s', t') = cross_1(s, t, p_1, p_2) :\Leftrightarrow \tag{3.23}$$

$$(s'_i, t'_i) = \begin{cases} (s_i, t_i) & if\, \forall i : p_1 < i < p_2 \\ (t_i, s_i) & if\, \forall i : i \leq p_2 \vee i \geq p_1 \end{cases} \tag{3.24}$$

is called two-point crossover operator with $p_{cross2} (0 \leq p_{cross2} \leq 1)$ *as crossover probability for this operator.*

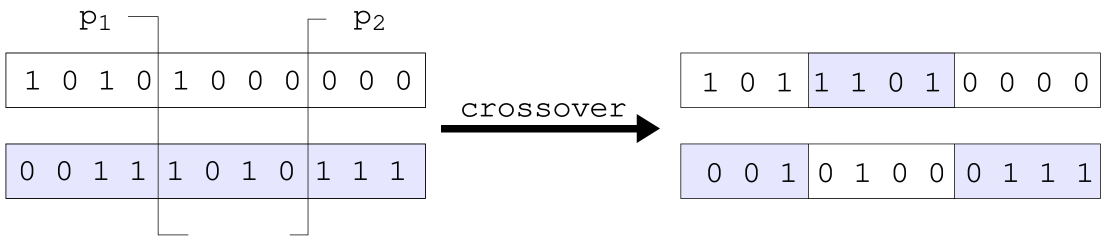


Figure 3.7: GA-two-point operator.

3.3.3 Selection and Fitness Scaling

The selection is an important principle in evolutionary algorithms as it determines which individuals form or contribute to the next generation. In a GA the selection operator is probabilistic and determines the frequency with which an individual is allowed to mate. The number of offspring is based upon its relative fitness, while in ESs a deterministic selection mechanism chooses the best individuals for the next generation. According to [6] two phases of selection can be identified. First, every individual is assigned a fitness value. In the second phase individuals are drawn for mating according to their fitness. The higher the fitness of an individual, the greater is the expected number of its offspring in the next generation. The fitness of an individual in a GA can be determined by the so-called proportional selection method [30].

Definition 3.3.6 (Proportional fitness, expected number of offspring) *Let s and s_i be strings and μ the constant number of individuals in the population, then* proportional fitness *of an individual string is defined for $fit > 0$ as:*

$$prop_{fit}(s) = \frac{fit(s)}{\sum_{i=1}^{\mu} fit(s_i)} \tag{3.25}$$

With this method every individual is evaluated relatively to the sum of fitnesses of the population. This scheme is illustrated by figure 3.8.

fitness	percentage of wheel
657.2	53%
396.8	32%
186.0	15%
1240.0	100%

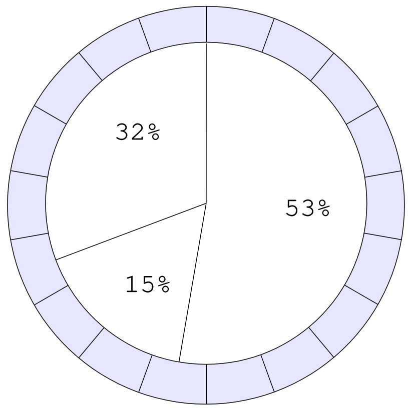


Figure 3.8: Roulette-wheel selection.

Since this measure requires positive fitness values and in order to compensate for possibly low fitness variation at the end of an optimization process, scaling methods are needed. Such methods are *window scaling, linear dynamic scaling, logarithmic scaling, exponential scaling*, and *sigma scaling* [25]. Other methods like linear ranking [6] determine the fitness by the absolute rank of an individual within the population and avoid scaling.

Definition 3.3.7 (Linear ranking fitness)
Let s be a string and μ the constant number of individuals in the population and $1 \leq max \leq 2$ and $min = 2 - max$, then the linear rank fitness of an individual string is defined as:

$$rank_{fit}(s) = \frac{1}{\mu} \cdot \left(min + (max - min) \cdot \frac{rank(s) - 1}{\mu - 1} \right) \tag{3.26}$$

with rank(s) as absolute position of s in the descendingly ordered population. The parameter max can be used to control the selection pressure.

Both methods assume a constant population size such that the number of expected offspring equals the actual population size ($\sum_{i=1}^{\mu} fit(s_i) = \mu$).
After the individuals have been assigned a fitness value, they can actually be selected for mating by the roulette-wheel selection method. According to their fitness the individuals receive a certain amount of space on the wheel. By spinning the wheel for each needed offspring and selecting the winning individual the mating pool is filled for creating the next generation.

Chapter 4

Evolution of Weights in Neural Networks

The search for training algorithms for NNs is as old as the research field itself. Many methods have been invented and improved to train networks. Most of these methods rely on gradient-based techniques and may therefore stagnate in local minima that are not globally optimal [95]. Methods to cope with such situations are multi-starting, adding noise to the training patterns, perturbing weights, and weight decay to name a few.

In contrast to gradient-based techniques, EAs do not rely on knowledge about first- or second-order derivatives, thus it is assumed that EAs are less affected by the problem of getting stuck in non-global local minima. In recent years EAs have been explored as training methods, and there are strong hopes that they might overcome the limitations of classical gradient-based training, but a thorough comparison of neural versus evolutionary learning is still missing.

This chapter is devoted to an investigation of evolutionary learning (ES) and an empirical comparison with classical neural network learning (backpropagation). Its contribution to the research in neural network and evolutionary algorithms will be:

- From the viewpoint of neural networks, the usefulness of ESs as an alternative non-gradient based training mechanism is investigated. Here, we will see that ESs cannot compete with gradient-based search, except for small problems. We will further see that networks with non-continuously differentiable activation functions can be useful and ESs are good means for training them.

- From the viewpoint of ESs, we will gain insights into the functioning of ESs on a new class of objective functions. Countermeasures to cope with the specific difficulties of the search in weight spaces are proposed and compared. With a growing number of problem dimensions the learning becomes more difficult for an ES and the "correct" parameterization gets crucial.

In section 4.1 a brief overview of the different approaches for evolving weights is given. Section 4.2 poses questions that will be answered throughout this chapter, it discusses and characterizes possible search spaces spanned by neural networks and gives criteria for evaluation and comparison. Each of the subsections in section 4.3 presents results for ESs and backpropagation as learning algorithms on a given learning task. In section 4.4 the scaling properties of ESs and BP are researched while section 4.5 investigates the benefits of non-continuously differentiable activation functions.

4.1 Related Work

There has not been any thorough parameter study that explored EA based learning on a variety of NN learning problems so far. Of the first two who employed an EA to train NNs were Montana and Davis who used a very specialized GA with additional gradient-based operators [55]. There were other GA based approaches as well but our focus is on ES. Some authors have already experimented with ES and concluded that they are superior search strategies for neural networks.

In [94] Wienhold used a non-standard ES. As basic algorithm, he used a fixed $(2, 20)$-ES with averaging recombination for the step sizes, a step size mutation of either 1.5 or 1/1.5 by tossing a coin, no recombination of the object variables x, mutation of $x_{ij}^{new} = x_{ij}^{old} + \frac{\sigma_{ij}^{new}}{\sqrt{n}} \cdot N(0, 1)$. The initialization of the object variables x_i is done with uniform distributed random numbers from the interval $[-0.5, 0.5]$ and the initialization of the step sizes σ_i with $\frac{1}{6}$. In some experiments he used weights and biases while other experiments include the slope of the activation function as well.

His argument that ESs have no parameters to adjust whereas backpropagation strongly depends on the learning rate η and the momentum term α does not hold. He incorporated critical and application specific parameters like recombination and population size into his fixed algorithm and pretended that none of them were free parameters. Nevertheless, to allow for a comparison of results, his learning task, the 6-bit parity network, was investigated.

The results and claims of Goerick and Rodemann [24] could not be confirmed by our experiments. Their first claim that a gradient-based search could not solve a 9-bit parity problem is false. Their feed-forward architecture with backpropagation learning solved the problem in 10 of 10 runs in our experiments (table 4.5). Possible reasons why their backpropagation algorithm could not solve the problem might be a poor choice of learning parameters, the absence of bias units or too few training epochs. Their results with an ES could not be verified as critical parameters (i.e., μ, λ, recombination scheme, and initialization) are not given.

Another approach to use ESs for finding optimal weights is to use an evolutionary algorithm for weight perturbation. Ng and Leung and Luk [57] proposed a method to help backpropagation escape from local minima. If the error of a network did not improve for a given time, its weights underwent an evolution cycle with a population generated by random mutations of the actual network. The best network at the end of the evolution was used for further backpropagation cycles. Fogel [61] compared the performance of backpropagation, simulated annealing and evolutionary programming on some test problems. The EP approach uses weights and biases of the network. Each weight is perturbed by a Gaussian random variable with zero mean and a variance equal to the parental mean square error. They concluded that all three methods give reasonable results on their problem. Kitano [39, 40] proposed an integrated framework of structure evolution and learning. Starting from his earlier work on matrix rewriting rules similar to graph grammars, he initialized weights from a pre-specified matrix. This matrix was used to lookup initial weights from which the gradient-based search started.

4.2 Evolution Strategies for Network Training

The problem of getting trapped in non-global local minima presented itself in the applications of this thesis. In case of the chemical engineering application (chapter 5) we have a failure rate of 20% for backpropagation, the difficult two-spirals problem (section 4.3.5) exhibits 70% failures, and even the simple XOR problem (section 4.3.2) suffers from 4% failed runs, which might be caused by such minima.

This leads to the following questions that will be answered throughout the rest of this chapter:

1. Can evolution strategies be used for the training of neural networks?
2. Can ESs overcome the problem of being trapped in non-global local minima?
3. How do ESs and BP scale with the problem dimension?
4. How do ESs and backpropagation compare in terms of computational effort?
5. What is the best parameterization for an ES as a training algorithm?
6. Which are the advantages and disadvantages of ESs in neural network training?

From the viewpoint of evolution strategies the neural network can be seen as an objective function, that is, the mean sum of squared errors over all training patterns (Eq. 2.6). In principle, the object variables can consist of one of the following neural network parameter sets.

1. Only the weights w_{ij}. This yields $|I| \times |H| + |H| \times |O|$ object variables.
2. The weights plus biases. This yields $|I| \times |H| + |H| \times |O| + |H| + |O|$ object variables.
3. The weights, biases and slopes of the units activation function with $|I| \times |H| + |H| \times |O| + 2|H| + 2|O|$ object variables.

As most NN training algorithms adapt weights and biases they are used here as well. If we assume a standard network architecture with fully connected input, hidden, and output layers then the object variable vector consists of the weights from input to the hidden layer, the weights from the hidden to the output layer, the bias weights for all hidden units, and the bias weights for the output units[1].

$$x := (w_{i,j}, w_{k,i}, b_i, b_k), \quad i \in H, j \in I, k \in O \tag{4.1}$$

In this context the strategie parameters σ can be seen as local "learning rates" for the weights.

4.2.1 Characteristics of the NN Weight Space

To explain the success and failure of backpropagation and different evolution strategies a characterization of the search space, that is the weight space of a network, will be given.

Local but not globally optimal minima can occur in even the simplest network[2] as is can be seen from figure 4.1 and in more complex ones as it can be seen from figure 4.2.

[1]Due to the nature of ES search operators, the ordering of weights and biases in the object variable vector can be chosen arbitrarily.

[2]A network with one input, one sigmoid hidden, and one sigmoid output unit that should learn the identity mapping of $1 \rightarrow 1$ and $0 \rightarrow 0$.

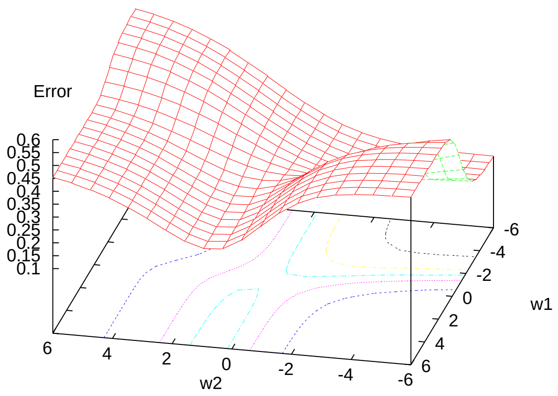


Figure 4.1: The error surface of a simple (1-1-1) network exhibiting a local non-global minimum at $(w_1, w_2) \approx (4, 1)$.

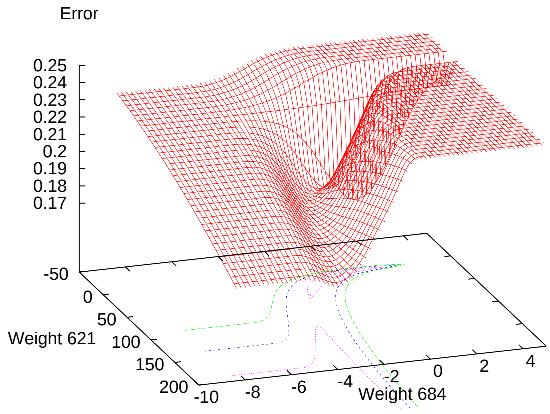


Figure 4.2: The error surface of the 9-bit parity network exhibiting a local non-global minimum at $(w_{621}, w_{684}) \approx (40, -2)$.

The following error landscapes were extracted from the 9-63-1 architecture for solving the 9-bit parity problem. Weights were selected to represent different functional areas, input-hidden, hidden-output, and bias of the network. In figure 4.3 the weight space (two-dimensional subspaces) at the beginning of the search with randomly initialized weights is shown, while figure 4.4 shows the same weights close to an optimum.

To illustrate the different microscopic and macroscopic characteristics of the weight space, each error landscape is given at three different scales. The following observations can be made:

- Different problem subspaces exhibit very different characteristics.
- The microscopic and macroscopic behavior are different from each other.
- In some subspaces the error function is multi-modal.
- In other subspaces it is convex and bowl-shaped.
- Some surfaces are smoothly sigmoid-shaped.
- Large flat planes can be found at all levels.
- Narrow corridors that are not parallel to any axis can be found.

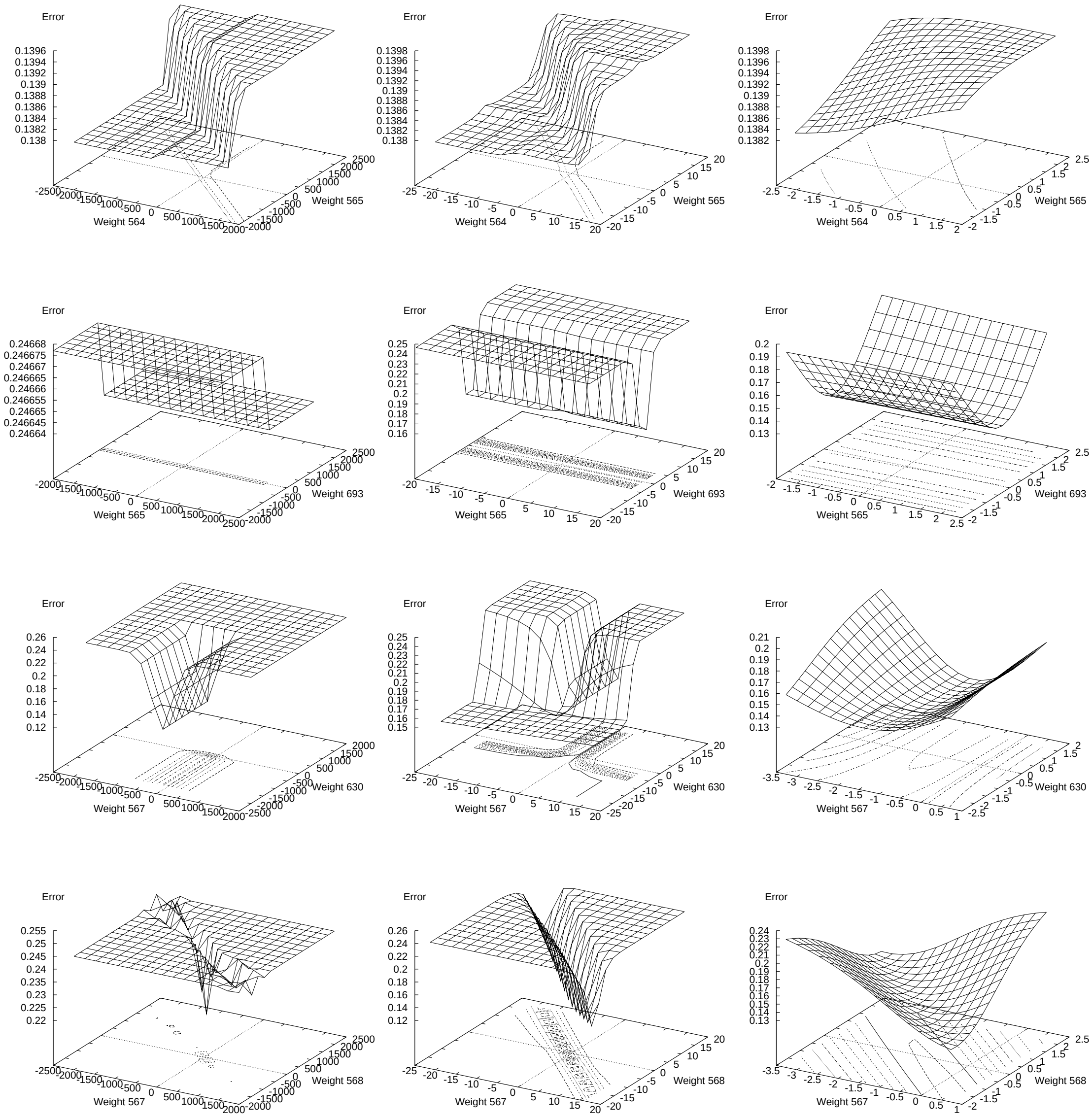


Figure 4.3: Random: The figures show two-dimensional subspaces of a neural network's weight space with random weights. Weights are selected to represent different functional areas, input-hidden, hidden-output, and bias of the network (row 1 weight i_9, h_{62} vs. weight i_9, h_{63}, row 2 weight i_9, h_{62} vs. bias *out*, row 3 weight h_1, out vs. bias h_1, row 4 weight h_1, out vs. weight h_2, out). From the top to the bottom 4 typical error surfaces of a total of 24012 are shown. To illustrate the different microscopic and macroscopic characteristics of the weight space, the scale changes from left $(2 \cdot 10^4)$ to right $(2 \cdot 10^0)$.

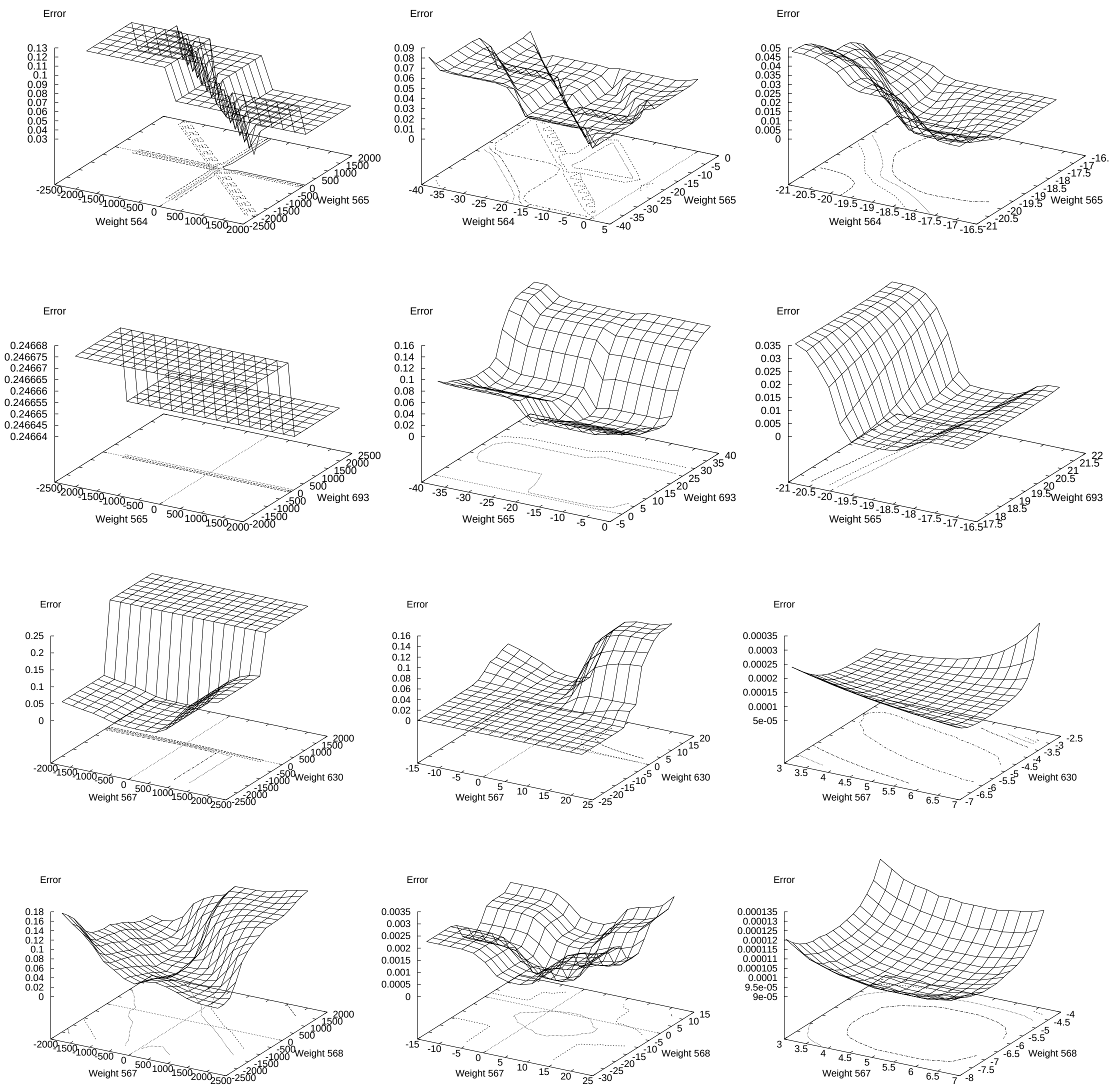


Figure 4.4: Optimum: The figures show two-dimensional subspaces of a neural network's weight space near an optimal solution. Weights are selected to represent different functional areas, input-hidden, hidden-output, and bias of the network (row 1 weight i_9, h_{62} vs. weight i_9, h_{63}, row 2 weight i_9, h_{62} vs. bias out, row 3 weight h_1, out vs. bias h_1, row 4 weight h_1, out vs. weight h_2, out). From the top to the bottom 4 typical error surfaces of a total of 24012 are shown. To illustrate the different microscopic and macroscopic characteristic of the weight space, the scale changes from left $(2 \cdot 10^4)$ to right $(2 \cdot 10^0)$.

4.2.2 Criteria for Evaluation and Comparison

Several benchmark problems with varying dimensions (tables 4.2 and 4.8) are chosen to investigate the capabilities of ESs as training algorithms: Three artificial classification problems, 2-bit, 6-bit and 9-bit parity, the well-known two-spirals approximation problem and a real-world problem from chemical engineering that is thoroughly covered in chapter 5.

The criteria for evaluation and comparison will be:

- the computational effort to reach a certain error limit,
- the convergence reliability, and in some cases,
- the quality of the solution.

The ES and backpropagation are rather different in their working principles, so that one has to establish a relation between generations and training epochs. If we neglect the algorithmic details[3] and assume that the backpropagation phase and the application of ES operators in one generation produce a similar computational overhead, then we only have to account for the forward phase which is the same for both algorithms. The computational effort for one generation cycle and λ epochs are approximately the same which means that the number of function evaluations and the number of epochs can be compared directly.

4.3 Experiments and Results

When conducting experiments with different methods one has to adjust crucial parameters of the methods. The parameter setting is often considered a "black art" or done by "educated guessing". As the focus is on ES rather than BP learning the two crucial backpropagation parameters, learning rate η and momentum term α, are adjusted according to the "educated guess" principle in combination with some preliminary experiments. Here, one can expect better results with optimal settings and even better results with sophisticated training algorithms like Quickpropagation or RProp.

For ES recent work of Kursawe [43] showed that there is no single correct choice of parameters for all problems and a complex interdependence between parameters

[3]The backpropagation algorithm can be divided into two phases. The forward propagation phase in which the input activations are propagated through the network and the error is computed at the output units and the backward phase in which the error is propagated back to the input units and weights changes are performed (see def. 2.1.13 for further details). The forward pass is the same for both algorithms, only that the ES performs it for each individual in the population. Instead of a backward phase with weight adaptation the ES uses recombination, mutation and selection.

exits. An "optimal" parameterization of an ES depends on the objective function and a "suitable" setting of all parameters. For larger problem sizes (i.e. 100 dimensions) he presented simulation results where an ES could be forced by a poor parameter setting to fail on the simple sphere model (see [43], p. 51 and the appendix A.7 p. 151 of this work).

We are left with the problem of exploring the parameterization of an ES with the hope of finding an optimal or at least "good" parameter set for training neural networks.

There are several possible candidate parameters: population size, selection type, recombination scheme, mutation rates, number of strategy variables, and some minor choices like local or global recombination and coupled or decoupled parameters sets. In order to keep the parameter study feasible we kept the population and selection type constant.

Preliminary experiments that are documented in the appendix A.5 suggested to always use n step sizes, coupled and local recombination. For all experiments a (15,100)–ES with n step sizes was used and the following 12 recombination schemes as well as an ES without recombination were investigated.

Recombination matrix		*step sizes* σ			
		discrete	intermediate	average	geometric
obj. vars. x *(weights)*	discrete	x	x	x	x
	intermediate	x	x	x	x
	average	x	x	x	x

Table 4.1: Recombination matrix.

For details on ESs and recombination types see def. 3.14. All other parameters, i.e. the mutation rates τ and τ', were set according to the heuristics in section 3.1.2. Table 4.2 gives the problem dimensions with the corresponding mutation factors.

Problem	XOR	Chem. Eng.	6-bit	Spirals	9-bit
Dimensions	9	25	91	252	694
τ'	0.235	0.141	0.074	0.044	0.026
τ	0.408	0.316	0.228	0.177	0.137

Table 4.2: Problem dimensions and mutation factors for all problems.

If not mentioned otherwise all runs were performed 10 times and only those runs that were able to reach a specified error limit were taken into the average. Weight initialization was done according to a uniform distribution within the interval $[-0.5,\ldots,0.5]$, and as error function the mean sum of squared errors was used (Eq. 2.6). Each parameter setting was run for 10,000 generations (1,000,000 function evaluations). The results of each experiment are summarized by a table and three figures. The table shows the success of all backpropagation and ES runs. It gives the success rate, which allows conclusions about the reliability, the minimal (min), maximal (max), and average[4] (mean) number of function evaluations[5] needed to meet the termination criterion.

The first two figures allow for a comparison of the training of a typical backpropagation run and an ES with the best recombination scheme. The graph of the ES training progress shows the best (Best) and average (Avrg) individual network (weight vector) in the population and the smallest (SigMin) and largest (SigMax) step size σ in the population, thus allowing conclusions about the behavior of ESs. Very large σ are an indicator for too large weights that might cause saturated units, while very small σ account for weights that cannot undergo significant changes. In both cases the learning process is hindered or at least slowed down. The bottom figure summarizes results for each of the 10 ES runs. Depicted are the best individual, the smallest and largest step size in the population. For each run the number of generations needed to reach the error limit (left y-axis) and the smallest (S_min) and largest (S_max) step size (right y-axis) are given. Runs are sorted by the number of generations to detect correlations between running time and step sizes.

4.3.1 Parity Problems

Parity problems are artificial benchmarks that were widely used to explore the capability of neural networks. One of the historical benchmarks is the XOR problem (2-bit parity) that gained its popularity because of the fact that it is non-linear and not learnable by training algorithms known in the Sixties. Artificial benchmarks are purely synthetic and usually have strong regularities in their structure. One can question if the results obtained on them can be transfered to real-world problems. However, the optimal solution of a synthetic problem is known in advance which makes it easy to evaluate the success of a training algorithm. The n-bit parity problem is, for larger n, a difficult to solve learning problem.

Here, three sizes of parity problems are investigated. Other authors reported results on two of the parity problems. In Wienhold's work [94], a network with 2 fully connected hidden layers of 6 units each (6-6-6-1) was used to learn the 6-bit parity task with an ES as training algorithm. The third parity problem for which Goerick and Rodemann [24] reported that it could not be solved by

[4] It should be noted that a large standard deviation (sd) and, in many cases, a too small number of successful runs prohibit a proper interpretation of average values.

[5] Due to the evaluation of the initial population of $\mu = 15$ individuals a constant of 15 must be added to the number of function evaluations for all ES runs.

gradient-based methods is a 9-bit parity problem that had to be learned by a 9-63-1 network.

4.3.2 2-bit parity (XOR)

- Network: 2-2-1, fully connected (9 weights, 4 patterns)
- Algorithm 1 (BP): $\eta = 0.8$, $\alpha = 0$
- Algorithm 2 (ES): (15,100)-ES with all recombination types
- Termination: error below 10^{-3} (0% classification error) or 10,000 epochs (BP), 100,000 function evaluations (ES)

This rather easy task can be learned by BP and all ES variants as well. From table 4.3 we see that BP solves the problem in 48 of 50 runs within 1434 function evaluations on average. The performance of the various ES recombination schemes is diverse. The most successful ones are those with intermediate or averaging recombination of step sizes. The two best recombination schemes (di, dA) solve the problem in all 50 runs yielding a better success rate then backpropagation. The high standard deviation and the distribution of runs (figure 4.5) forbids the comparison of averaged values. When we fall back upon the minimum number of function evaluations as criterion then BP is 2-3 times faster than the best four ES variants.

One could object that most ES solved the problem in fewer than 50 generations and an inappropriate population size accounts for the large number of function evaluations. Thus, a smaller (2,20)-ES (ii) was tested as well and for such small problems this can reduce the number of needed function evaluations.

An interesting observation about the behavior of the ES can be made when looking at the step sizes of figure 4.5. In the bottom figure all runs are sorted by the number of generations needed to reach the optimum. For each run the number of generations and final step sizes in the population are given. There is a clear correlation between large step sizes in a population and long running times. All runs that successfully solved the problem in fewer than 50 generations (50%) ended with their step sizes being within a range of 10^{-2} and 10^{2}. In connection with the error landscapes of figure 4.3 (row 2 and 3), we see that this is a "reasonable" range for mutations[6] in which the sigmoid units of the network are not likely to be saturated. Larger step sizes can force one or more units into saturation and the evolution process might get caught on the planes caused by saturated units.

When such a plane is reached an ES behaves as if the underlying objective function is a constant function. In this case, intermediate or averaging recombination

[6]The probability of a mutation step of the same absolute value as the corresponding step size is approximately 0.32.

on the step sizes in combination with mutation causes them to grow exponentially. See section A.6 in the appendix for an illustration of this effect.

This observation leads to the idea of restricting the minimally and maximally allowed step size and weights in the population. In appendix A.5.4 several countermeasures to prevent large weights and vanishing weight changes are investigated. Neither the bounding of weights and step sizes to "reasonable" ranges nor a step size correction leads to an increase of the success rate. In fact, the opposite can be observed. A closer look reveals that some runs with relatively large step sizes in the population can still be successful and that such large step sizes are needed to escape from the planes. An example is run number 47 in figure 4.5 (bottom) that solved the problem with the largest step size above 10^{13}.

	suc. rate	min	max	mean	sd
BP	48:50	1,169	1,777	1,434	278
ES (dA)	50:50	2,100	31,000	6,492	6,481
ES (di)	50:50	2,200	76,600	12,624	15,955
ES (ii)	48:50	3,000	74,700	19,068	21,738
ES (iA)	48:50	3,000	87,400	13,435	21,100
ES (AA)	48:50	3,100	66,200	15,572	20,606
ES (Ai)	45:50	4,000	97,700	19,640	29,008
ES (dd)	27:50	2,600	44,600	6,962	10,737
ES (nn)	31:50	2,500	22,400	5,880	5,470
ES (ig)	33:50	4,700	23,200	7,348	3,326
ES (id)	27:50	2,800	27,900	5,211	4,744
ES (Ag)	35:50	5,200	17,600	9,637	3,297
ES (Ad)	26:50	3,100	10,400	5,626	1,801
ES (dg)	32:50	3,300	12,700	5,400	1,584
ES (Ag)	35:50	5,200	17,600	9,637	3,297
(2,20)-ES (ii)	46:50	560	19,800	3,224	5,069

Table 4.3: XOR: Comparison of backpropagation and evolution strategy. Success rate and number of function evaluations needed to reach the error limit.

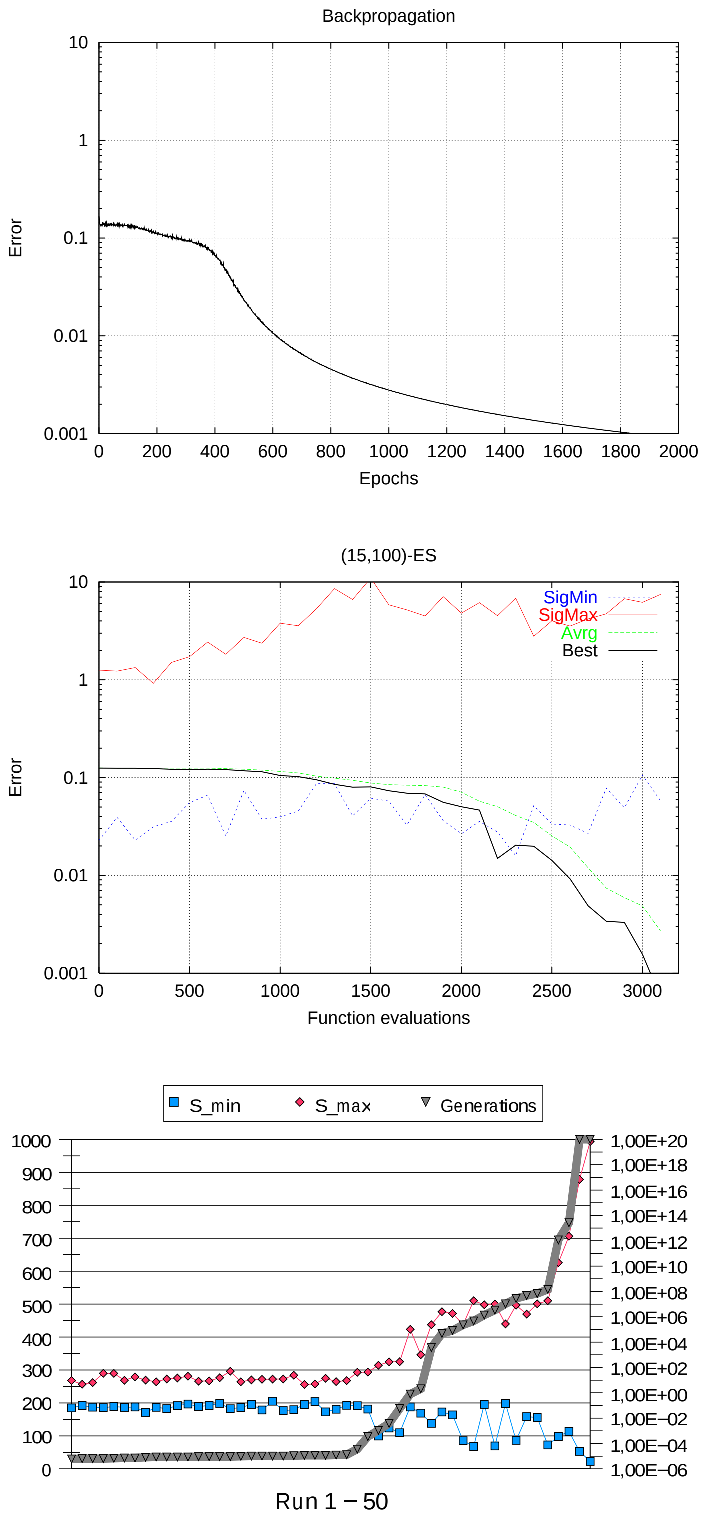


Figure 4.5: XOR: (Top) Typical BP learning learning rate $\eta = 0.8$ and no momentum. Success after 1,847 function evaluations (epochs). (Middle) An example (15,100)–ES run with "ii" recombination. Success after 3,100 function evaluations (31 generations). (Bottom) Results of all runs with smallest (S_min) and largest (S_max) step sizes. Runs are sorted by the number of generations and one can observe a clear correlation between large step sizes in a population and a large number of generations.

4.3.3 6-bit parity

- Network: 6-6-6-1, fully connected (91 weights, 64 patterns)
- Algorithm 1 (BP): $\eta = 0.1$, $\alpha = 0$
- Algorithm 2 (ES): (15,100)-ES with all recombination types
- Termination: error below 10^{-4} (0% classification error) or 1,000,000 epochs (BP) 10,000,000 function evaluations[7] (ES)

This task is more difficult than the 2-bit parity problem, but it can still be learned by BP and some ES variants. Both learning curves in figure 4.6 are dominated by long periods of stagnation and occasional sharp drops in error.

From table 4.4 we see that BP solves the problem reliably in 10 of 10 runs but with a large standard deviation. From the 13 recombination schemes only 5 are able to solve the task and the most successful ones are again those with intermediate or averaging recombination of step sizes (ii, Ai). A high standard deviation and the exponential distribution of runs in figure 4.6 forbids the comparison of averaged values. When we fall back upon the minimum and maximum number of generations as criterion then BP is 3 to 4 times faster in the best case and 5 times in the worst case. Some of the strategies with geometric recombination are also able to solve the problem but not as reliably as BP or the other ES variants.

The correlation between large step sizes and long running times in the bottom figure is even more obvious than before. In combination with the ES run where we see exponentially increasing step sizes, this indicates the presence of large flat planes in the weight space. Both observations suggest that the ES spends most of the time on the planes of saturated units.

	suc. rate	min	max	mean	sd
BP	10:10	17,913	739,277	437,534	274,484
ES (ii)	8:10	55,900	4,007,900	1,015,590	1,379,480
ES (ig)	1:10	218,500	-	-	-
ES (dg)	4:10	68,000	489,300	277,075	172,667
ES (AA)	2:10	468,600	3,008,000	-	-
ES (Ai)	9:10	76,200	3,560,200	952,722	1,228,420
ES Wienhold	best 10 of 20	-	-	1,509,600	-

Table 4.4: Parity-6: Comparison of backpropagation and evolution strategy. Success rate and number of function evaluations needed to reach the error limit.

[7]As we now have to cope with 91 parameters instead of 9, the number of allowed function evaluations was increased by a factor of 10. Additional results with RProp and bootstrapping can be found in the appendix A.1.3.

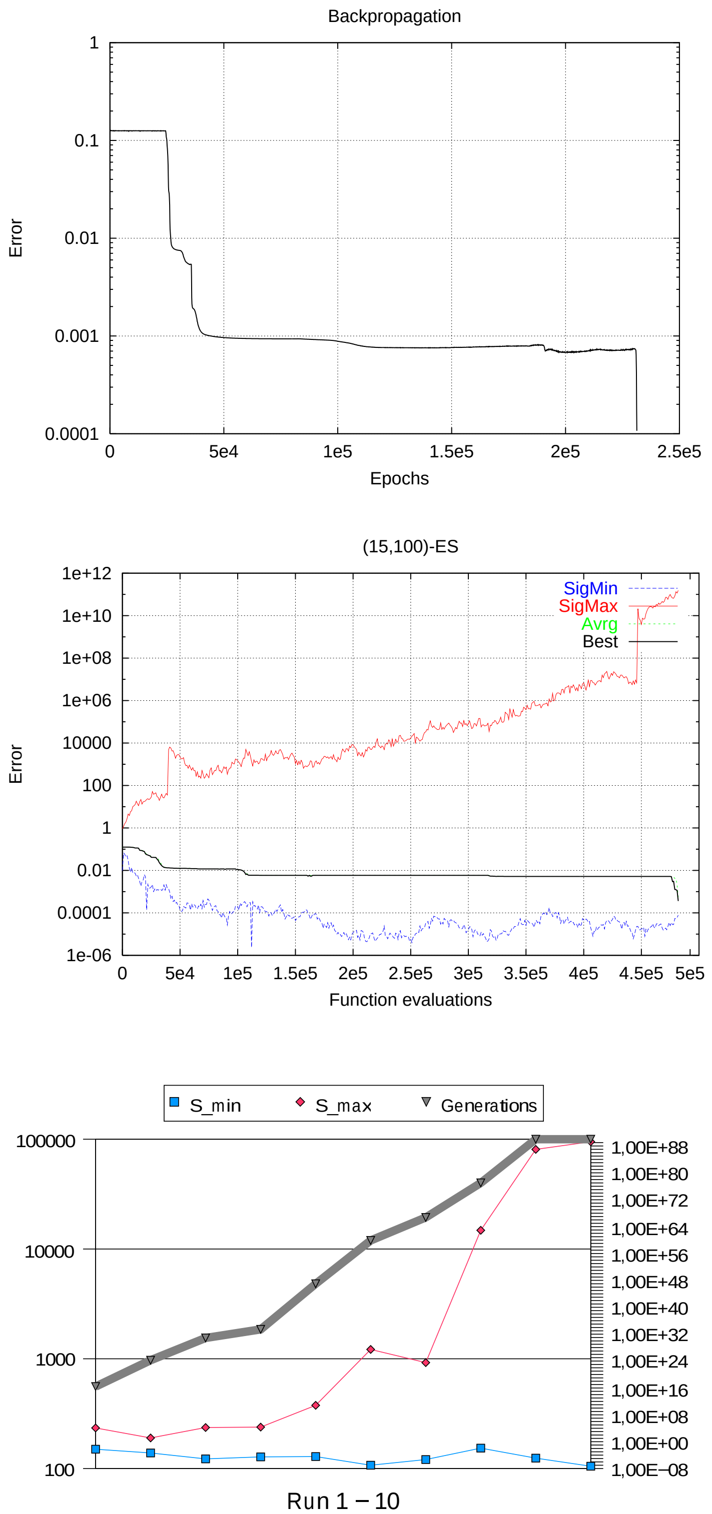


Figure 4.6: Parity-6: (Top) Typical BP learning with learning rate $\eta = 0.1$ and no momentum. Success after 231,410 function evaluations (epochs).(Middle) An example (15,100)–ES run with "ii" recombination. Success after 482,900 function evaluations (4,829 generations). (Bottom) Results of all runs with smallest and largest step sizes. Runs are sorted by the number of generations and one can observe a clear correlation between large step sizes in a population and a large number of generations.

4.3.4 9-bit parity

The 9-bit parity problem is the largest problem here in terms of weight space. Instead of 9 or 91 parameters we now have to cope with 694 parameters.

- Network: 9-63-1, fully connected (694 weights, 512 patterns)
- Algorithm 1 (BP): $\eta = 0.1$, $\alpha = 0$
- Algorithm 2 (ES): (15,100)-ES with all recombination types
- Termination: error below 10^{-4} (0% classification error) or 1,000,000 epochs (BP) 10,000,000 function evaluations (ES)

Even though this task is the most difficult of our parity problems, it can still reliably be learned by BP in 10 of 10 runs, whereas none of the ES runs succeeded in reaching the error limit. As is typical for parity problems, the learning curves in figure 4.7 are dominated by long periods of stagnation and occasional sharp drops in error indicating saturated units.

In figure 4.3 a small part of the weight space at the beginning of the search with randomly initialized weights is shown, while figure 4.4 shows the same weights close to an optimum. The weights were selected to represent different functional areas of the 9-63-1 architecture.

The example ES run with exponentially increasing step sizes and the largest step sizes ($> 10^{12}$) at the end of all ES runs suggest that the ES wanders on the planes of saturated units. This is even more likely with a growing number of parameters because a single too large weight causes saturation of the corresponding unit. Even though no solution was found by an ES figures 4.6 and 4.7 suggest that a larger number of function evaluations (i.e. 10^8 or 10^9) might lead to a solution. Such a large number was beyond our computing capabilities and would also yield a dramatic performance gap between ESs and BP.

	suc. rate	min	max	mean	sd
BP	10:10	71,342	596,712	295,748	226,453

Table 4.5: Parity-9: Success rate and number of function evaluations needed to reach the error limit. There were no successful runs with an ES.

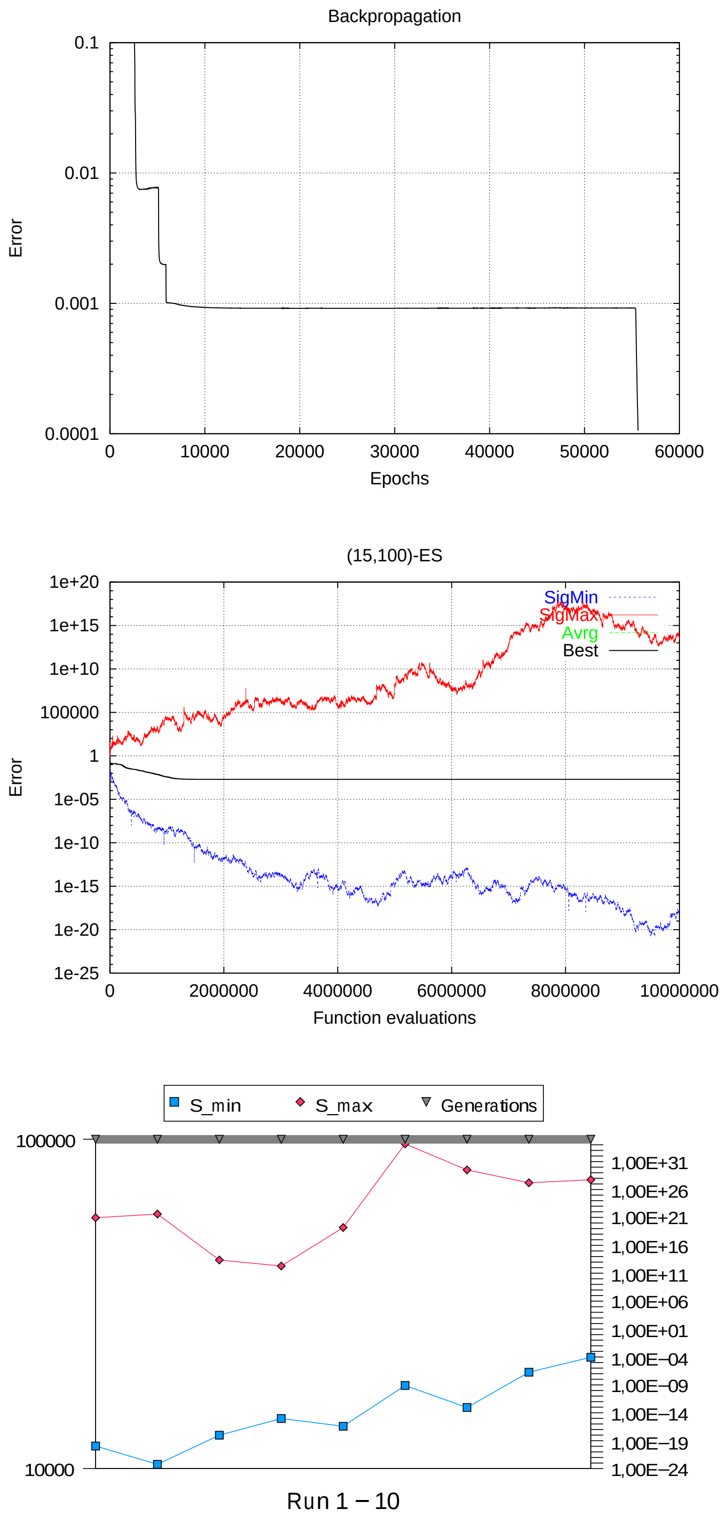


Figure 4.7: Parity-9: (Top) Typical BP learning with learning rate $\eta = 0.1$ and no momentum. Success after 556,600 function evaluations (epochs). (Middle) An example (15,100)–ES run with "ii" recombination. No success after 10,000,000 function evaluations. (Bottom) Results of all runs with smallest and largest step sizes.

4.3.5 Two Spirals

The two-spirals problem is an artifical benchmark problem in which the network has to learn to discriminate between two inter-twined spirals in the 2-D plane. Each spiral consists of 97 points and cycles three times around the origin without overlapping (see figure 4.8) [44]. For this problem the number of hidden units plays a crucial role for the classification and generalization quality of the network. While a 100% correct classification of a fixed number of patterns can be achieved relatively easily, it is rather difficult to achieve a perfect generalization. The more units a network has, the better the generalization and the easier the classification task. Thus, two network sizes were investigated and compared with respect to their classification and generalization ability: one moderate-sized network with 50 hidden units and one network with as few as 25 hidden units.

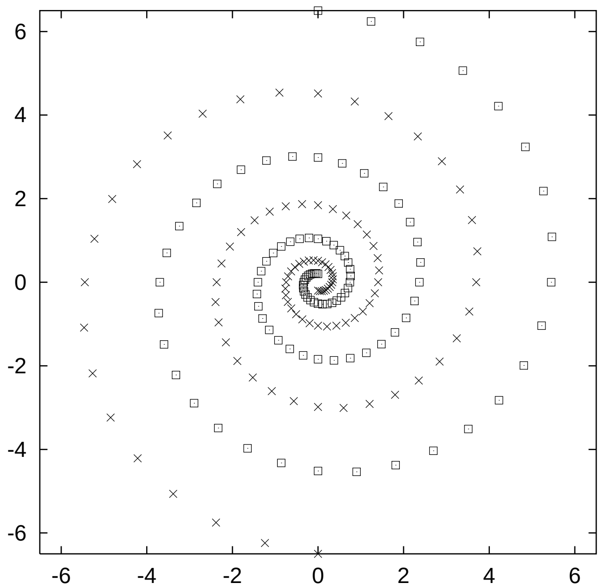


Figure 4.8: Spirals: Task of learning to tell two inter-twined spirals apart. Each spiral consists of 97 data points.

- Network-A: 2-25-2, fully connected (127 weights, 194 patterns)
- Network-B: 2-50-2, fully connected (252 weights, 194 patterns)
- Algorithm 1 (BP): $\eta = 0.2$, $\alpha = 0.1$
- Algorithm 2 (ES): (15,100)-ES with all recombination types
- Termination: error below 10^{-3} (0% classification error) or 100,000 epochs for BP and 10,000,000 function evaluations for ES

On the very difficult task to teach the small network A, backpropagation achieved only a success rate of 3 out of 10. The ES was even worse with only a single successful run that needed about 100 times the number of function evaluations. ESs with other than intermediate or averaging recombination (ii, Ai) did not succeed at all.

For the larger network B backpropagation solved the problem reliably in 10 out of 10 runs and achieved a perfect classification within 11,443 epochs on average (table 4.6). The ESs were able to solve the task but not as reliably as BP and, on average, 175 to 250 times slowlier than BP.

	suc. rate	min	max	mean	sd
NN-A-BP	3:10	14,006	63,056	43,336	25,910
NN-B-BP	10:10	6,298	22,801	11,443	5,751
NN-A-ES (ii)	1:10	1,438,900	-	-	-
NN-B-ES (ii)	4:10	1,206,900	5,509,000	2,880,500	1,960,630
NN-B-ES (Ai)	3:10	1,682,300	2,183,500	2,009,070	283,204

Table 4.6: Spirals: Comparison of backpropagation and evolution strategy. Success rate and number of function evaluations needed to reach the error limit.

A correlation between failed runs (network B) and large step sizes can be seen in the bottom figure 4.10. All failed runs ended with the smallest step size σ_{min} above a value of 10^{30} suggesting that those runs suffer from the saturation of hidden units. With a large number of hidden units the network can compensate for saturated units as the task can be solved with fewer still functional units. The fewer hidden units a network has, the more difficult it is to compensate for saturated units, since the task gets more difficult with fewer units.

The plots in figure 4.9 show the typical classification and generalization capabilities for the two network architectures both trained by BP (left) or an ES (right). The surface in the background of the two spirals is the *response surface*. It gives the network's response to an input at the specific (x, y)-coordinate. A network with perfect generalization would yield two neatly inter-twined spiraling areas with the original training patterns in the middle of them.

No significant visual difference of the response surfaces can be seen between BP or ES-trained networks. The surfaces of other successful runs exhibit very similar surfaces for ES and BP. A difference can be seen between the two network architectures. The smaller network exhibits a better generalization than the larger one independently from the used training algorithm. The generalization capabilities of both training algorithms are similar in case of the two-spirals problem.

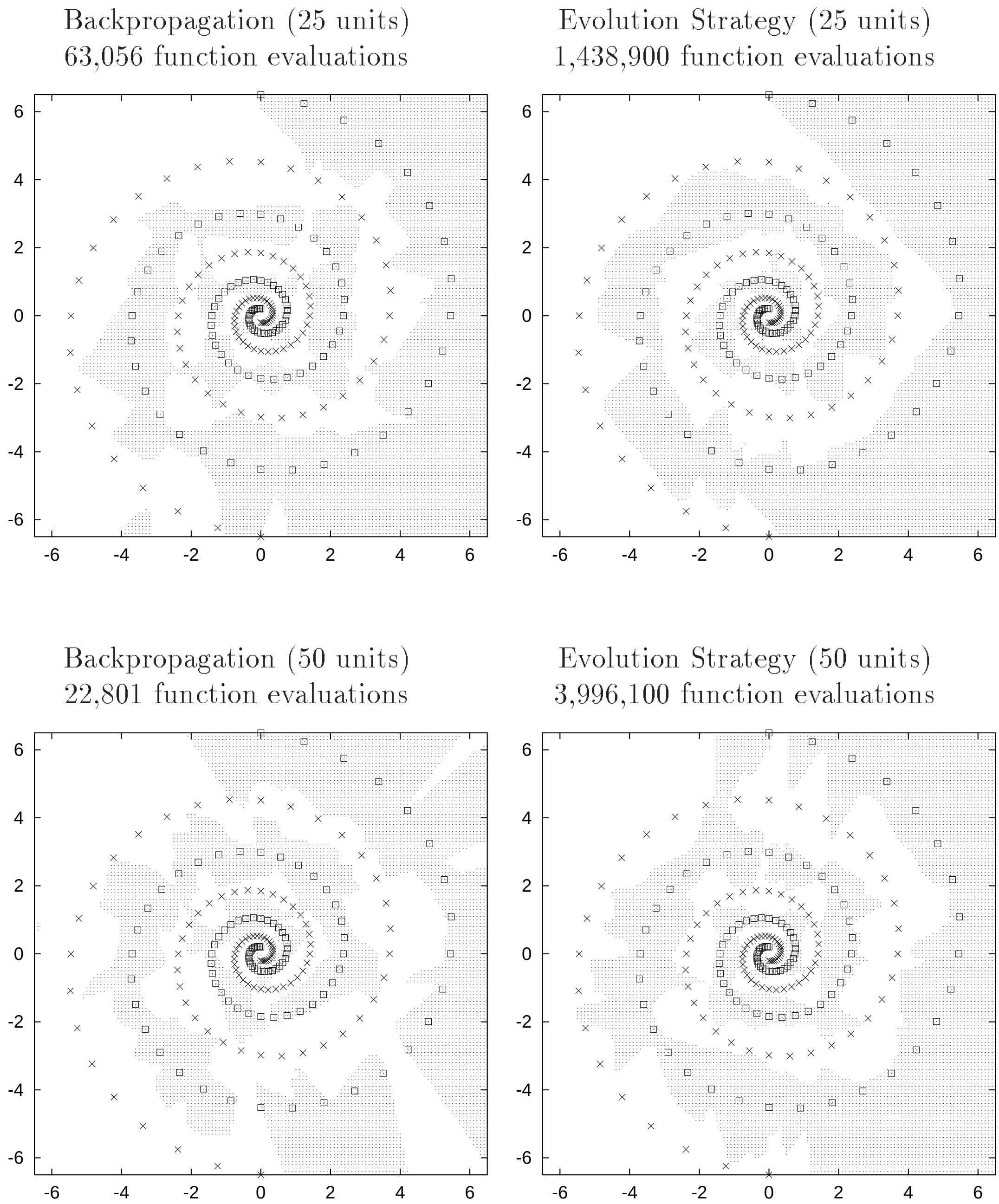


Figure 4.9: Spirals: Classification and generalization capabilities of two different network architectures (25 and 50 hidden units) trained with BP and an ES.

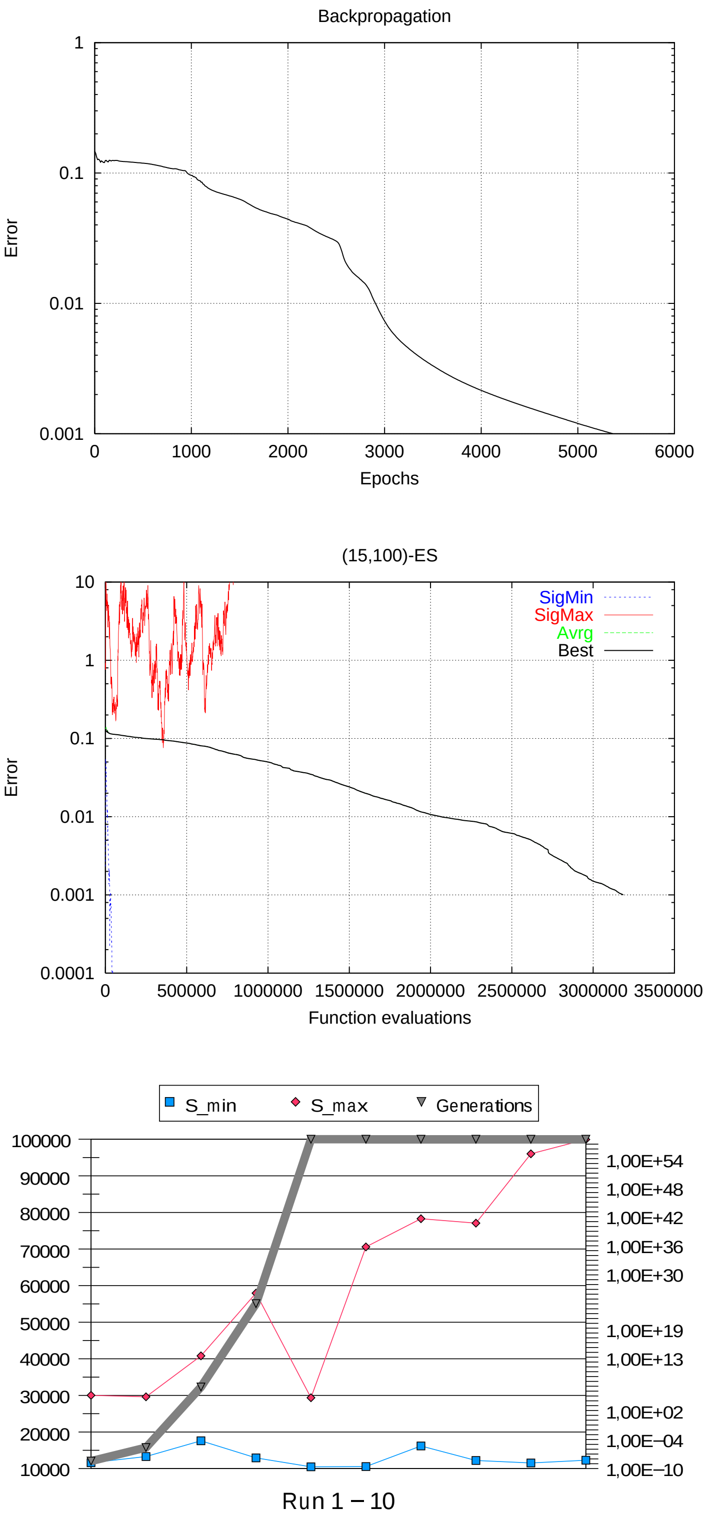


Figure 4.10: Spirals: (Top) Typical BP learning with learning rate $\eta = 0.1$ and no momentum. Success after 8,220 function evaluations (epochs). (Middle) An example (15,100)–ES run with "ii" recombination. Success after 3,185,500 function evaluations (31,855 generations). (Bottom) Results of all runs with smallest and largest step sizes. Runs are sorted by the number of generations, and one can observe a correlation between large step sizes in a population and a large number of generations.

4.3.6 Chemical Engineering

The following problem stems from chemical engineering where a neural network was used to acquire an internal model of a chemical process and predict certain properties of chemical compounds. Further details can be found in chapter 5. The performance of the ES on this problem was thoroughly studied because of its practical application.

In addition to the variation of the recombination scheme, the number of mutation step-sizes σ_i, either 1 or 25, the number of recombination partners, local or global, and coupled or decoupled recombination were included in the parameter study.

- Network: 4-4-1, fully connected (25 weights, 214 patterns)
- Algorithm 1 (BP): $\eta = 0.8$, $\alpha = 0.2$
- Algorithm 2 (ES): (15,100)-ES with all recombination types.
- Termination: error below $5 \cdot 10^{-5}$ or 250,000 epochs for BP and 10,000,000 function evaluations for ES.

	suc. rate	min	max	mean	sd
BP	8:10	22,202	176,210	80,488	54,376
ES (ii)	3:10	2,718,000	4,158,000	3,314,000	751,351
ES (Ai)	3:10	4,430,400	6,011,600	5,462,130	894,139

Table 4.7: Chemical Engineering: Comparison of backpropagation and evolution strategy. Success rate and number of function evaluations needed to reach the error limit.

Here, backpropagation succeeds in 8 of 10 runs with an average of 80,488 epochs (table 4.7). Again, only ESs with intermediate or averaging recombination (ii, Ai) are able to solve the task. With only 3 successful runs they are less reliable and 41 to 67 times slower than BP. Unlike the other problem, this one does not exhibit a correlation between running time and step sizes (figure 4.11). At the end of all runs the step sizes are still within a “reasonable” range and there is also some variation in the population which can be concluded from the difference between best and average individuals. Results might improve with longer running time.

A look at the prediction quality of the network (see chapter 5, figure 5.10) reveals that only one out of all ES-trained networks performs comparably to backpropagation. All other networks give rather poor results.

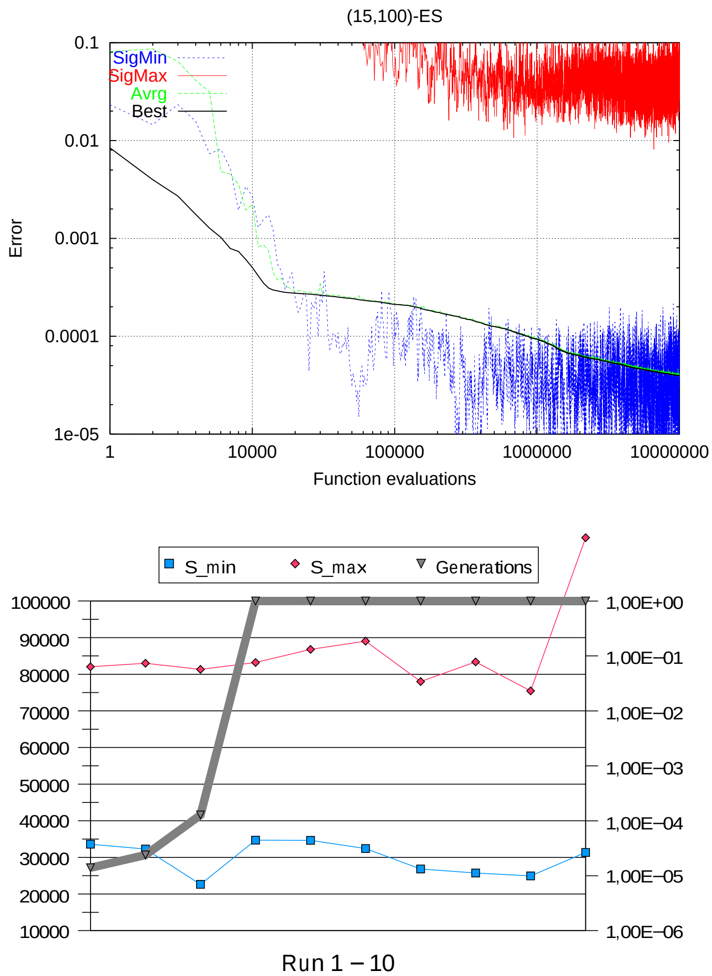


Figure 4.11: (Top) An example (15,100)–ES run with "ii" recombination. Success after 4,158,000 function evaluations (41,580 generations). (Bottom) Results of all runs with smallest and largest step sizes. Runs are sorted by the number of generations.

With only one overall step size all runs got stuck at high error levels with small variances. In all runs the populations collapsed around a single individual. This holds for all recombination schemes. All experiments are documented in the appendix A.5.

With the intermediate recombination, one can observe a significant difference between global and local recombination. Both variations are able to find a good solution, but the local recombination delivers better results on average. Local recombination on the object variables combined with global recombination on the step sizes is comparable to local recombination on both parameter sets. All experiments are documented in the appendix A.4.

4.4 Scaling Properties of Evolution Strategies and Backpropagation

In the previous experiments we have seen a decreasing performance of ESs with a growing problem dimension. The 9-dimensional 2-bit parity problem can easily solved by an ES while larger problems exhibited a reduced success rate and a drastic increase in the number of function evaluations needed. The following experiments investigate the scaling properties of backpropagation and evolution strategies. The general n-bit parity problem can easily be used to gradually increase the network size with the number of bits used. Thus, the problem dimension can be scaled with the number of bits and used hidden units. We generated network architectures to solve the parity problem for n bits by using $n^2 - n$ hidden units. This yields $(n+2) \cdot (n^2 - n) + 1$ numbers of weights and biases (problem dimension) in the network.

Problem	2-bit	3-bit	4-bit	5-bit	6-bit	7-bit	8-bit	9-bit
Patterns	4	8	16	32	64	128	256	512
Hidden units	2	6	12	20	30	42	56	72
Dimensions	9	31	73	141	241	379	561	793

Table 4.8: n-bit parity problems: Network architecture and the corresponding problem dimensions.

All experiments were performed with the following settings:

- Network: n-$(n^2 - n)$-1, fully connected ($(n+2) \cdot (n^2 - n) + 1$ weights, 2^n patterns)
- Algorithm 1 (BP): $\eta = 0.6$, $\alpha = 0.1$
- Algorithm 2 (ES): (15,100)-ES with (ii) recombination
- Termination: error below 10^{-3} for all problems except for 9-bit parity where 10^{-4} was used (0% classification error) or 1,000,000 epochs (BP), 10,000,000 function evaluations (ES)

The scaling behavior of both algorithms is shown in figure 4.12. Up to 141 dimension (5-bit parity) the number of function evaluations needed to solve the problem increases for BP and ES alike. However, the ES needed about 2.8 to 13 times more function evaluations. For problem dimensions larger than 141 the learning gets easier for BP while it gets more difficult for the ES. The number of needed function evaluations increases while the success rate (figure 4.13) decreases. An exception is the 5-bit parity problem where the ES is slightly better than BP.

Bits	suc. rate	min	max	mean	median	sd
2	10:10	1,175	236,152	24,923	1,595	174,218
3	10:10	1,593	5,681	4,129	3,619	1,297
4	10:10	1,999	59,262	27,798	7,593	24,233
5	10:10	29,587	486,739	209,488	96,561	139,134
6	8:10	22,135	339,638	131,780	76,948	93,476
7	10:10	15,435	974,613	225,041	47,093	330,552
8	10:10	19,380	299,195	103,849	63,389	77,104
9	10:10	14,346	121,326	51,340	33,368	33,085

Table 4.9: Scaling behavior of backpropagation on the n-bit parity problems.

Bits	suc. rate	min	max	mean	median	sd
2	10:10	3,500	135,200	35,170	7,230	51,189
3	10:10	9,100	11,500	10,220	10,148	912
4	10:10	18,500	49,200	31,530	28,711	10,563
5	10:10	42,700	584,600	177,000	89,469	184,442
6	6:10	236,400	1,129,000	612,593	689,967	341,552
7	2:10	637,800	1,024,000	1,338,840	830,900	273,085
8	0:10					
9	0:10					

Table 4.10: Scaling behavior of the evolution strategy on the n-bit parity problems. All runs for the 8- and 9-bit problems failed.

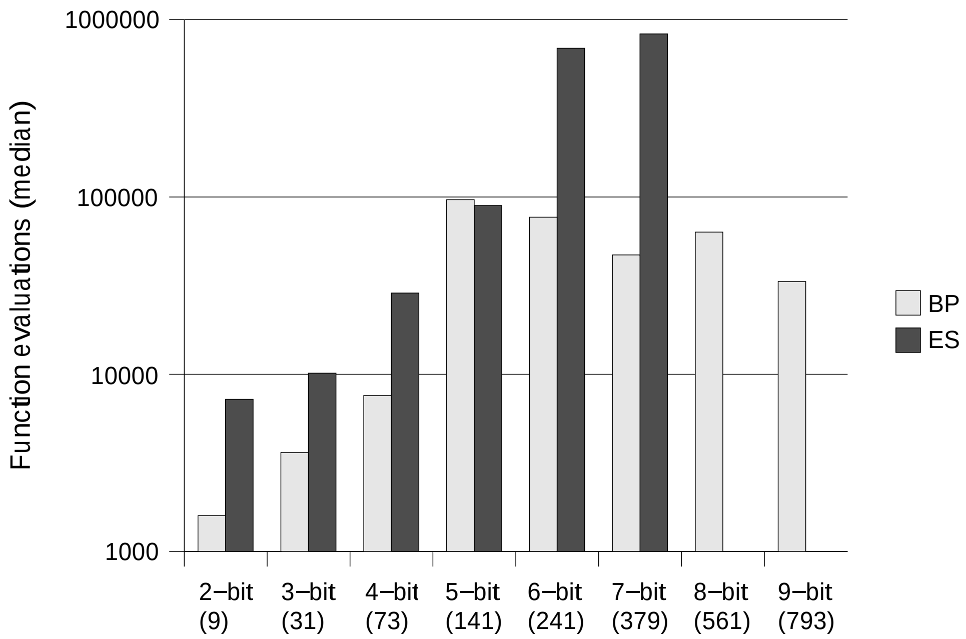


Figure 4.12: n-bit parity: Scaling behavior of backpropagation and ES. The x-axis gives the number of bits used and the corresponding problem dimension while the y-axis shows the median number of needed function evaluations.

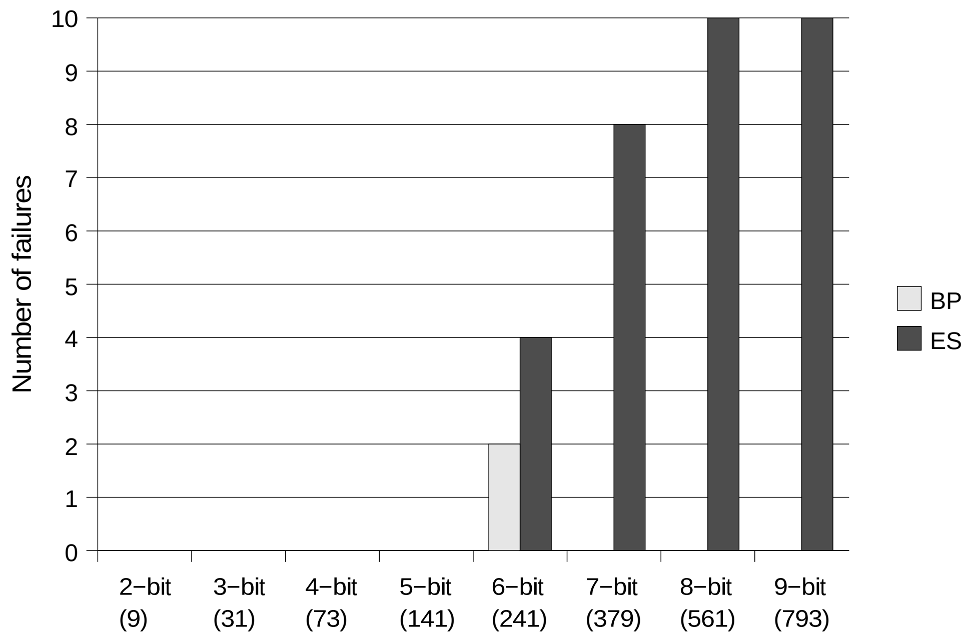


Figure 4.13: n-bit parity: Failure rate for backpropagation and ES. The x-axis gives the number of bits used and the corresponding problem dimension while the y-axis shows the number of failed runs from a total of 10 runs.

The above results suggest that an ES is much more likely to suffer from a large dimensionality than backpropagation. While backpropagation solved all problems without any change to the algorithm or parameter settings in almost all cases, a (15,100)-ES worked fine for small problem dimensions but failed for very large search spaces. This observation is supported by the findings of Kursawe who presented simulation results where a poorly parameterized (15,100)-ES could be forced to fail on the simple sphere model ([43], p. 51). This is illustrated by figure A.42 in the appendix.

Ostermeier assumes that the population size must be much larger than the problem dimension to enable the step size adaption [60]. For the simple and theoretically well understood sphere model, it is known that the population should grow with the number of problem dimensions [9] for optimal progress, but these results cannot be transfered easily to more complex objective functions. Hence, the optimal population size is not known in advance for a new problem. As the population size was not investigated in this parameter study, it might well be the case that larger populations and longer running times lead to successful training. This might also lead to a further increase in the number of needed function evaluations which would widen the performance gap. As the backpropagation algorithm does not suffer that much from the curse of dimensionality it should be preferred at higher problem dimensions.

The above results do not allow the conclusion that ESs generally fail for large problem dimensions[8], but they identify the problem size as problematic. The optimization seems to become more difficult and a "correct" parameterization is more important. The population size might be a critical parameter here, that should be scaled with the problem dimension. Whether this result is only valid for the weight space of a neural network or more generally remains open.

[8]During the research that lead to this chapter several problems with more than 300 parameters (7-bit parity) could be solved by ESs.

4.5 Training without Gradient Information

It is well known that sigmoid functions are not the only possible choice as activation functions for units. It was shown that other non-linear activation functions can be used as well [82]. Several differentiable activation functions and their influence on the performance of the network have been investigated by [27, 46].

Due to the lack of derivatives non-continuously differentiable activation functions have not been used in feed-forward networks. For a network with such an activation function the backpropagation algorithm cannot be used as gradient information is not easy to get. To explore the capabilities of such activation functions other means of training had to be found. Here, ESs as training algorithms can be very useful.

As the two-spirals and parity problems are non-linear classification problems they can, in principle, also be learned by networks with other non-linear activation functions. In the following experiments we took the two different sizes of the two-spirals problem and the 2- to 9-bit parity problems and replaced the sigmoid function with the non-linear threshold function (see def. 2.2). All experiments were performed with the same settings as described in the previous sections.

n-bit parity problems

Bits	suc. rate	min	max	mean	median	sd
2	10:10	200	2,700	1,000	491	926
3	10:10	500	16,100	5,480	2,288	4,448
4	10:10	14,000	201,000	66,780	32,819	65,717
5	9:10	82,400	477,300	344,311	226,673	152,702
6	1:10	579,600	579,600	579,600	579,600	0

Table 4.11: n-bit parity: Results for ES-trained networks with threshold functions. All runs for the 7-, 8-, and 9-bit problems failed.

In figure 4.15 the success rates for ES-trained networks with sigmoid and threshold units are compared. The higher failure rates on the 5-, 6-, and 7-bit parity problems suggest that the problem becomes more difficult with threshold functions.

For the two smaller 2- and 3-bit parity problems a network with threshold functions even outperforms the same network with sigmoid functions (figure 4.14) while it needs more function evaluations for the larger problems. Except the 7-bit parity problem all parity problems that could be solved by networks with sigmoid units were also solved by networks with threshold units.

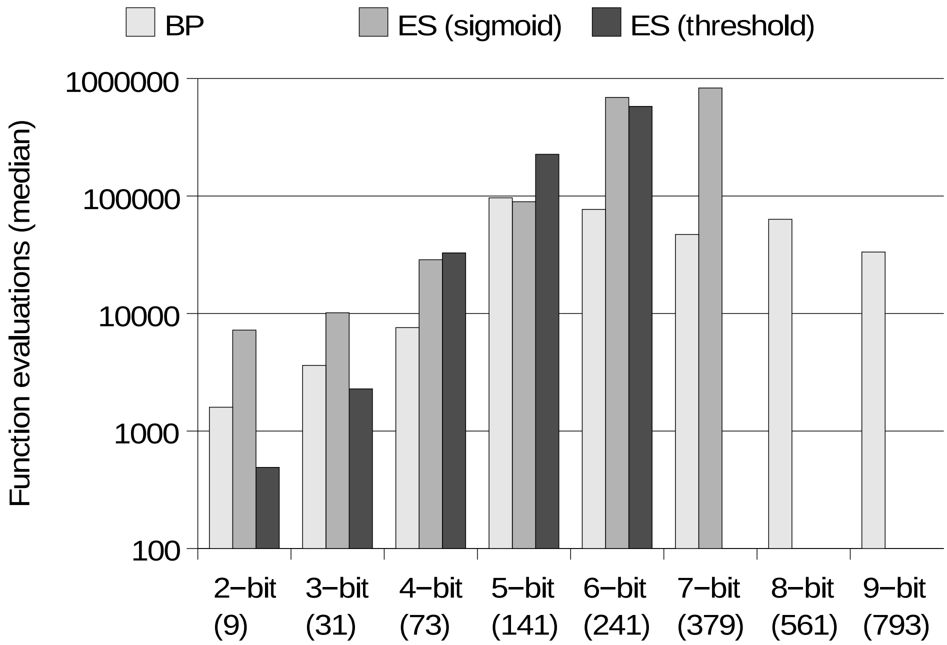


Figure 4.14: n-bit parity: Classification with threshold functions. Comparison of function evaluations of BP (sigmoid), ES (sigmoid), and ES (threshold). The x-axis gives the number of bits used and the corresponding problem dimension while the y-axis shows the number of needed function evaluations.

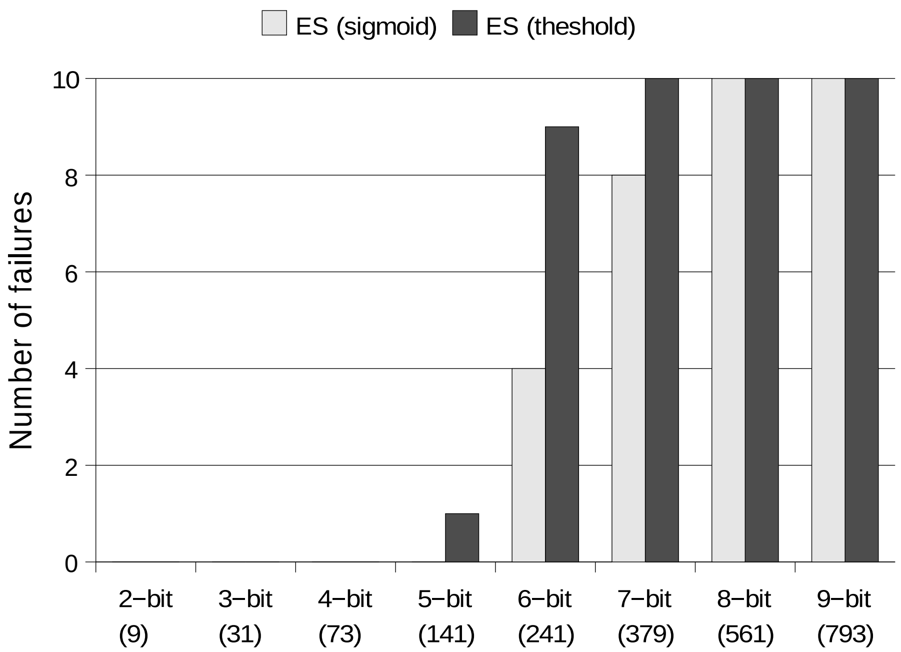


Figure 4.15: n-bit parity: Comparison of failures rates for ES-trained networks with sigmoid and threshold functions. The x-axis gives the number of bits used and the corresponding problem dimension while the y-axis shows the number of failed runs.

Two-spirals problems

The parity problems were purely binary problems with binary inputs and outputs, which made them an ideal candidate for threshold functions. The two-spirals problem instead has real-valued inputs that form smoothly inter-winded structures which are, intuitively, better approximated by smooth activation functions. Thus, the difficulty is increased with the usage of threshold functions.

	suc. rate	min	max	mean	sd
NN-A-ES (25, sigmoid)	1:10	1,438,900	-	-	-
NN-A-ES (25, threshold)	1:10	3,993,300	-	-	-
NN-B-ES (50, sigmoid)	4:10	1,206,900	5,509,000	2,880,500	1,960,630
NN-B-ES (50, threshold)	1:10	7,562,000	-	-	-

Table 4.12: Spirals: Results for ES-trained networks with threshold functions. Success rate and number of function evaluations needed to reach the error limit.

For both problem sizes solutions can be found by ESs. Due to the poor success rate of only 1 out of 10 for both problem sizes, a direct performance comparison is not permitted. Only in the case of the larger problem, it can be concluded from the higher success rate that networks with sigmoid units can be trained more easily than networks with threshold units.

The response surfaces in figure 4.16 show the classification and generalization capabilities for the two networks with threshold activation functions trained with an ES and for the same network with sigmoid activation functions trained by BP.

Despite the fact that the success rate is significantly lower and the number of needed function evaluations is higher than for networks with sigmoid functions, the generalization measured by the visual image of the response surfaces is better for networks with threshold functions.

The diverse results on the n-bit parity problem and the two-spirals problem do not allow a general conclusion about the difficulty of training networks with threshold units, but the above results show that ESs can be used successfully to train networks with non-linear and non-continuously differentiable activation functions. For smaller problem sizes and purely binary tasks, they do not only outperform ESs with sigmoid activation functions but also outperform BP-trained networks with sigmoid units.

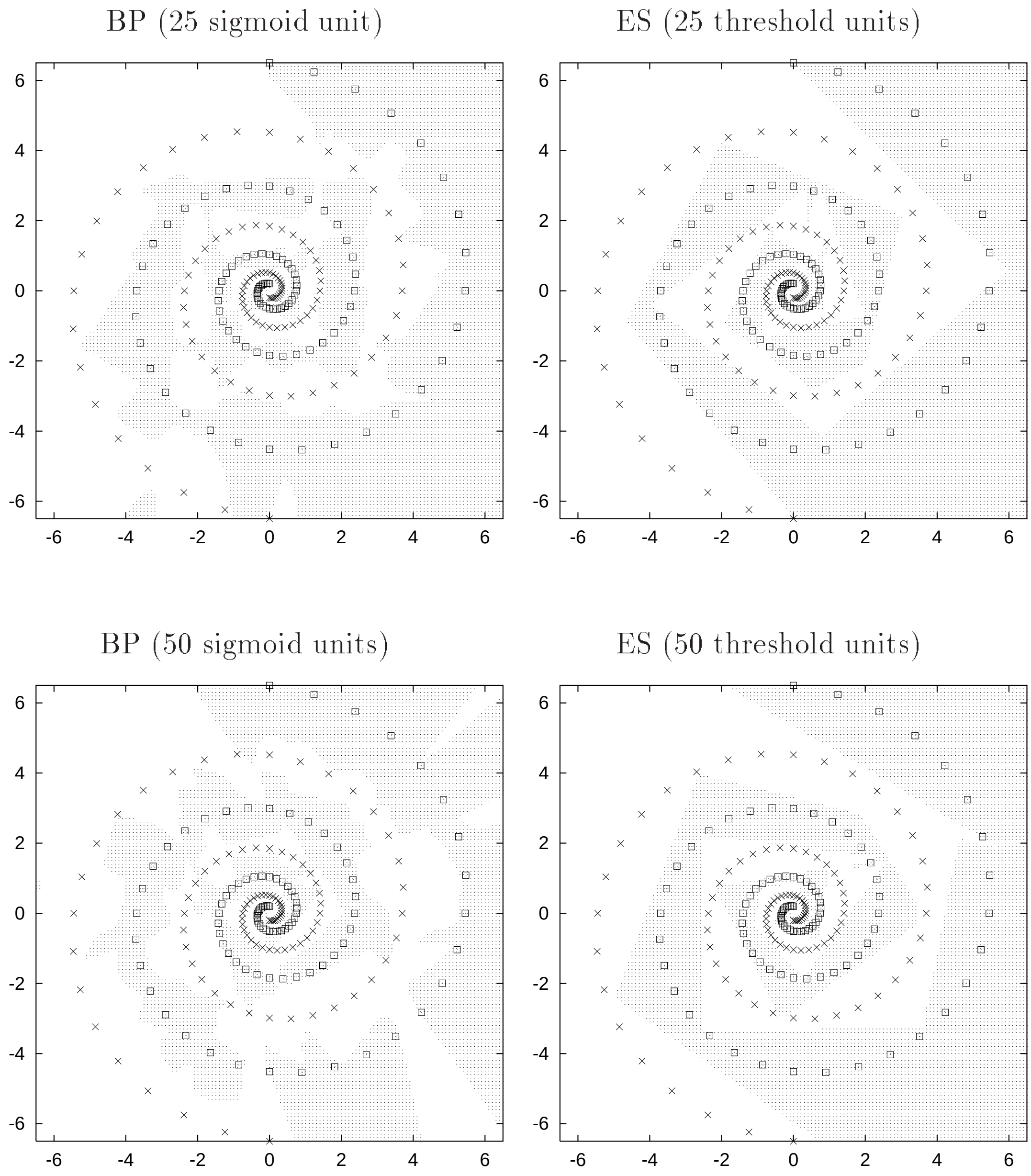


Figure 4.16: Spirals: Classification and generalization capabilities of two different network architectures with threshold functions trained with ES.

4.6 Discussion and Conclusions

Experiments show that ESs are, in principle, capable of successful weight tuning in neural networks. This also holds when the network consists of non-continuously differentiable activation functions. Nevertheless, they are much less efficient in terms of computational effort and not as reliable as properly tuned backpropagation. The following table 4.13 summarizes the success and failure of varying recombination types. Intermediate recombination of object variables and step sizes is the best choice closely followed by global-averaging recombination of step sizes.

		σ				
		d	i	A	g	n
x	d	failed	failed	failed	O	-
	i	failed	X	X	failed	-
	A	failed	X	O	failed	-
	n	-	-	-	-	failed

Table 4.13: The success of varying recombination types for network training. Here, "failed" means that only the very easy XOR problem could be solved, "X" denotes at least one successful run in all other problems excluding the 9-bit parity, "O" denotes at least two successful runs for one more problem, while "-" denotes no experiments.

Explanations why a certain recombination scheme fails or succeeds might be derived from the structure of the search space and probably improper values for τ' and τ. When training neural networks equipped with sigmoid units, difficult regions of the search space are planes caused by saturated units together with narrow and not axis parallel valleys in those planes.

A closer look at the initial error surfaces given in figure 4.3 reveals that in some dimensions (row 2) a step size larger than 10 is sufficient to reach a plane with 0.32 probability[9]. Even close to the optimum such planes can easily be reached with large σ (see figure 4.4 row 1).

Why is intermediate/averaging recombination successful despite the exponentially growing step sizes on the planes? Exponentially growing step sizes should not be mistaken for exponentially growing weights. Weights start oscillating around their center with growing orders of magnitude and are either caught on a plane or jump from one plane to another. Due to the averaging effect of recombination there is still a diminishing chance of getting back into the sensitive areas of the sigmoid units.

[9] With the normal distribution, more than 99.99 percent of all mutations lie within the range of $3 \times \sigma$ [29], and the chance for a mutation larger than σ is approximately 0.32.

When caught on a plane with large step sizes, discrete recombination only resamples the weights from too large positive or too large negative weights, so that the search performs a random walk on the planes. Here, the only chance of compensating for too large or too small values is to have a weight connected to the same unit with the same absolute value but an opposite sign. This is called epistasis. With the averaging effect of intermediate recombination the ES still has a chance of leaving a plane.

Within narrow corridors it is necessary to have one very small step size in one direction in order not to leave to corridor and a large one to make progress in one direction. If the corridor is not parallel to an axis, such as that in figure 4.3, the situation is even worse. Here, progress is rather slow due to the necessarily small step sizes in two directions that are needed to remain within the corridor.

Another reason for the failing of most recombination schemes might be the fixed population size. In this context Ostermeier points out problems that might be caused by adaptation mechanism of the step sizes. "The problem is that to enable step-size adaptation the resulting population sizes have to be much larger (... about 10 $\times$ problem dimension ...) than necessary concerning the object parameter optimization" [60]. For the 2, 6 and 9-bit parity problems we would have needed a population of 90, 910 and 6940 according to his suggestion. The large two-spirals problem and chemical engineering problem would need λ to be 2520 and 250.

An open question is: If large step sizes cause premature stagnation, then why does geometric recombination, which does not suffer from exponential growth, fail?

The investigation on the scaling properties of both algorithms suggest that ESs are much more likely to suffer from a large dimensionality than backpropagation. While BP solved all problems without any change to the algorithm or parameter settings in almost all cases, an (15,100)-ES worked for small problem dimensions but failed more often in case of large search spaces.

To explore the capabilities of non-continuously differentiable activation functions other means of training than backpropagation are needed. This chapter has shown that ESs can be used successfully as training algorithms for networks with such functions. Small networks with threshold functions that are trained by an ES can even outperform the same network with sigmoid functions trained by backpropagation.

The following results and advices were gained from one or two problems only and should be considered with care:

- Local (sexual) recombination always yields better results than global recombination. In most cases results are very similar, but in no case global recombination outperformed the local one. See section A.5.3.

- Imposing "reasonable" upper and lower bounds on the step sizes and weights and a weight dependent correction of step sizes does not help to increase the overall success rate and should be avoided. See section A.5.4.
- The overall mutability τ' is an application specific parameter. Increasing it yields better performance for some problems and worse for others. See section A.5.5.
- Strategy parameters and object variables should be coupled during recombination. The coupling was always beneficial in network training and caused no significant difference with classical ES test functions. See section A.5.2.

Questions revisited:

1. *Can* ESs *be used for the training of neural networks?*
   Yes - but only for small and moderate problems sizes. The best recombination scheme in almost all cases is local (sexual), that is, two parents[10], intermediate recombination of x and σ with coupled object variables and strategy parameters, and n step sizes. The second-best choice is global averaging recombination of x and intermediate recombination of σ.
2. *Can* ESs *overcome the problem of being trapped in non-global local minima like backpropagation?*
   It is not clear whether local minima are problematic but except for the very small 2-bit parity problem the failure rates of the best ESs are always higher than for backpropagation. Here, ESs suffer from flat areas caused by saturated units.
3. *How do* ESs *and BP scale with the problem dimension?*
   Without parameter adjustments an ES is more likely to suffer from a large dimensionality than BP.
4. *How do* ESs *and BP compare in terms of computation effort?*
   For the easiest task an ES needs more than a factor of 2 function evaluations and several orders of magnitude for larger tasks than BP.
5. *Which are there advantages and disadvantages of* ESs *as training algorithms?*
   A clear advantage over gradient-based training is that ESs can be used in networks with non-continuously differentiable activation functions. A disadvantage is that ESs have much more parameters to adjust (recombination scheme, population size, selection pressure, selection type "+" or "," , mutation factors τ and τ'), and those parameters are sensitive to changes [43].

[10]Note that two parents should not be mistaken for a population of $\mu = 2$. See section 3.1.3.

This chapter has empirically shown that the backpropagation algorithm outperforms ESs with respect to all our criteria in almost all cases. All previous claims that ESs, at least in their traditional variations, outperform neural network learning algorithms are proven wrong either by counter examples or by extensive parameter studies. With a growing number of problem dimensions the learning becomes more difficult for ESs and the "correct" parameterization is crucial.

"Nobody should make use of EC in case where good old methods like Linear and Dynamic Programming, quasi-Newton, or theoretically well underpinned methods work. None of the EAs would do the job better nor even as good as those" (H.-P. Schwefel, 1994 [74]). If such methods are not applicable as it is the case in neural networks with non-continuously differentiable activation functions, EAs are feasible methods.

Chapter 5

CI Methods in Chemical Engineering

This chapter investigates the utility of CI methods in one area of Chemical Engineering. It will compare the performance of neural networks and evolutionary algorithms and combinations of them on real engineering problems. An encoding of chemical compounds is proposed that allows the application of both paradigms and establishes a basis for comparisons.

Solutions found by CI methods are presented that compare to the best physically motivated methods known so far and even outperform them in several ways.

5.1 Motivation

In chemical engineering the simulation of chemical plants and the design processes are important tasks. To simulate and design such plants chemical engineers need to know about certain properties like, how chemicals react with each other or how they behave under influence of heat or pressure. The knowledge of such properties, which can be gained during real experiments, is a necessity to simulate and build such plants. The problem is that there are millions of chemical compounds known yet and experimental data are often not available.

For this reason there is a need for calculation methods which are able to predict those thermodynamic properties. Such methods are often physically motivated models based upon measurements of real experiments (see figure 5.1).

In this work, properties under consideration concern either pure components where the heat of vaporization has to be predicted or mixtures where the heat of mixing should be predicted.

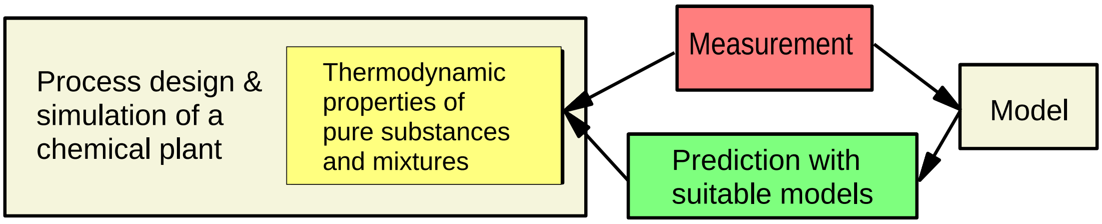


Figure 5.1: Motivation: Process design relies on chemical properties that have to be measured or predicted.

Usually models are developed that have a physical background and where the model parameters have to be fitted with the aid of experimental data. This often leads to nonlinear regression models with a multi-modal objective function where evolution strategies are successfully used [22, 21, 23].

There are more than 13 million known organic compounds and each year approximately another 1.000.000 can be added. The impossibility of measuring such large data leads to group contribution models, where molecules are divided into so-called functional groups. Physical interactions are no longer specified on the molecular level but on the level of function groups. Each functional group of a molecule gives a contribution to the thermodynamic property and the sum of all contributions has to be calculated. One gains a small number of functional groups instead of a large number of existing molecules. Hence, the number of interactions is significantly reduced. The group contribution principle is illustrated in figure 5.2.

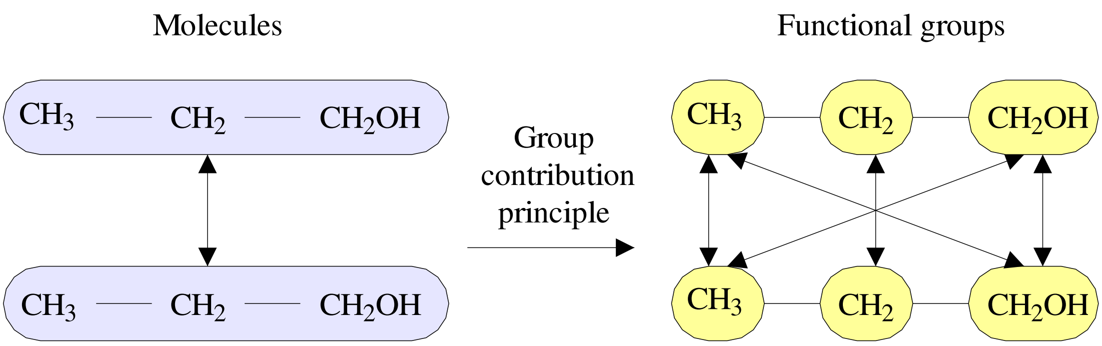


Figure 5.2: The group contribution principle. Functional groups within molecules contribute to chemical property instead of the whole molecule.

A new way for the calculation and prediction of thermodynamic properties is the use of neural networks in various ways. First, descriptors have to be derived from the molecular structure of chemical compounds. Given the descriptors, experimental data for a specific thermodynamic property can be used to generate training data for a neural network. Predictions of this thermodynamic property are then possible by using the molecular structure for a chemical compound without the need for experimental data. In a first investigation the enthalpy of vaporization was used.

In section 5.2 a brief overview of the models used is given followed by a section 5.3 describing the data. Section 5.4 continues with an experimental comparison of physical models, networks trained with backpropagation, networks trained with evolution strategies and a combination of the latter two.

5.2 Models for the Enthalpy of Vaporization

5.2.1 Physical Models

The physical background for the enthalpy of vaporization ΔH_v consists of electrostatic interactions forced by the atoms of the molecules. Equations can be derived from statistical thermodynamics in order to describe the interactions between molecules (first level) and between functional groups of these molecules (second level). Physical models, such as UNIFAC (UNIversal Functional Activity Coefficient) [18] were developed in order to describe the real behavior of liquid mixtures. The part of the UNIFAC model, which summarizes the interactions between functional groups of the molecules within a pure liquid were taken as a basis for the development of the so-called UNIVAP model (UNIversal enthalpies of VAPorization) [41, 88, 86]. This model consists of sums of exponential terms, which include the interaction parameters and the temperature. In contrast to UNIFAC an extended temperature dependence was used in order to describe the behavior of the enthalpy of vaporization in principle. For the UNIVAP model it was difficult to reach the critical point, where the enthalpy of vaporization reaches zero. Therefore a modified temperature dependence was used in this investigation (UNIVAP).

The interactions are weighted by the surface fractions of functional groups of a molecule. The interaction parameters have to be fitted to experimental data of enthalpies of vaporization. This leads to a non-linear regression problem of which the objective function consists of the *mean absolute error* (MAE Eq. 5.1) over all N experimental data points between the calculated values $\Delta H_v^{calc.}$ (physical model) and the experimental data $\Delta H_v^{exp.}$:

$$MAE = \frac{1}{N} \sum_{N} \left| \Delta H_v^{calc.} - \Delta H_v^{exp.} \right| \tag{5.1}$$

Other error functions like the *mean relative error* (MRE Eq. 5.2) or the *root mean squared relative error* (RMSRE Eq. 5.3) might also be used, but especially the RMSRE is not very expressive because it puts too much emphasis on data points close to the critical value.

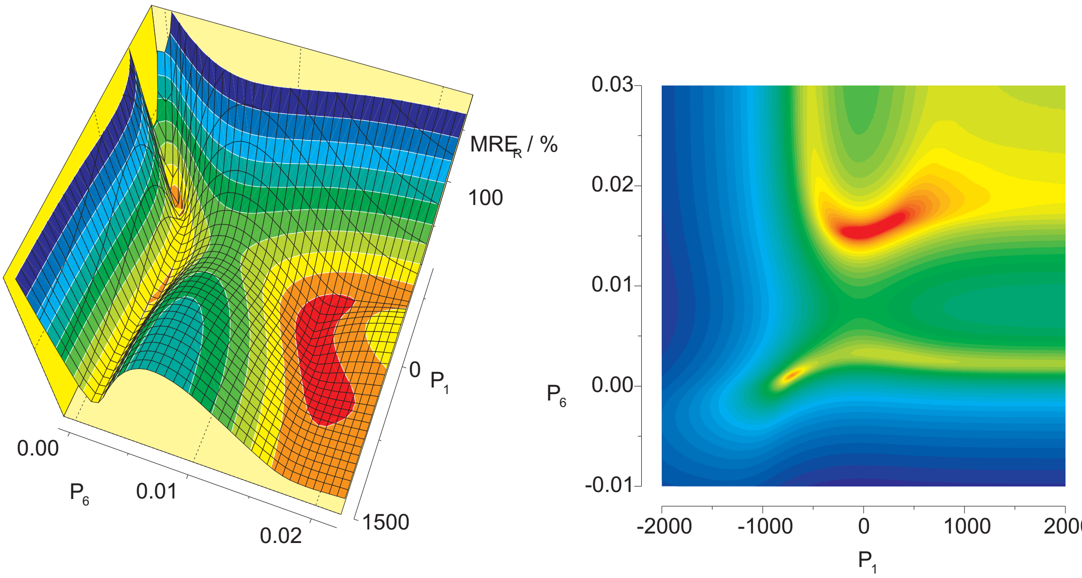


Figure 5.3: Multimodality in a physical model (2 of 27 dimensions, parameters P_1 and P_6 varied and all others held constant).

$$MRE = \frac{1}{N} \sum_N \frac{\left|\Delta H_v^{calc.} - \Delta H_v^{exp.}\right|}{\Delta H_v^{exp.}} \tag{5.2}$$

$$RMSRE = \sqrt{\frac{1}{N} \sum_N \left(\frac{\Delta H_v^{calc.} - \Delta H_v^{exp.}}{\Delta H_v^{exp.}}\right)^2} \tag{5.3}$$

Due to the complex structure of the physical model, especially the exponential terms, multimodality usually occurs. In figure 5.3 the multimodality of a physical model is illustrated. To cope with the difficulty of the corresponding optimization problem, [22, 21, 87, 86] propose an encapsulated evolution strategy (see section 3.2 for further details) for solving this problem.

Another theoretical approach is the so-called EBGCM (Enthalpy Based Group Contribution Model) [42, 86] in order to describe the enthalpy of mixing of binary liquid mixtures. This model is similar to UNIFAC, but has a slightly different background. It was used to derive an equation for the enthalpy of vaporization, which is similar to the UNIVAP model. This so-called EBGVAP model (Enthalpy Based Group contribution model for enthalpies of VAPorization) was used in our investigation, too. For UNIVAP and EBGVAP three parameters for the interactions between functional groups of the same type have to be fitted by non-linear regression. For interactions between different kinds of functional groups six parameters have to be estimated. In principle the enthalpy of vaporization of a molecule i can be calculated as follows:

$$\Delta H_v = \sum_k \nu_k^{(i)} \varepsilon_k^{(i)} \quad / \, kJ/mol \tag{5.4}$$

R is defined as the universal gas constant of 8.314 J/(mol·K) and $\nu_k^{(i)}$ is the number of groups of kind k within the molecule i. The term $\varepsilon_k^{(i)}$ / kJ/mol is called group enthalpic factor of group k. This factor can be written for UNIVAP (Eq. 5.6) and for EBGVAP (Eq. 5.7):

$$x = \left[\frac{\frac{\partial \Psi_{km}}{\partial T}}{\sum_p \Theta_p^{(i)} \Psi_{pm}} - \Psi_{km} \frac{\sum_p \Theta_p^{(i)} \frac{\partial \Psi_{pm}}{\partial T}}{\left(\sum_p \Theta_p^{(i)} \Psi_{pm} \right)^2} \right] \tag{5.5}$$

$$\varepsilon_k^{(i)} = RT^2 Q_k \left[\frac{\sum_m \Theta_m^{(i)} \frac{\partial \Psi_{mk}}{\partial T}}{\sum_m \Theta_m^{(i)} \Psi_{mk}} \right] + \sum_m \Theta_m^{(i)} \cdot x \tag{5.6}$$

$$\varepsilon_k^{(i)} = RT^2 Q_k \left[\sum_m \Theta_m^{(i)} a_{mk} \frac{\sum_p \Theta_p^{(i)} \frac{\partial \Psi_{pk}}{\partial T}}{\sum_p \Theta_p^{(i)} \Psi_{pk}} + \sum_m \Theta_m^{(i)} \sum_j \Theta_j^{(i)} a_{jm} \cdot x \right] \tag{5.7}$$

with Q_k defined as the relative van der Waals surface of group k, and the surface fraction of a group m within a molecule i can be calculated with:

$$\Theta_m^{(i)} = \frac{\nu_m^{(i)} Q_m}{\sum_p \nu_p^{(i)} Q_p} \tag{5.8}$$

The interaction parameter Ψ_{mk} between the groups of kind m and k is defined as:

$$\Psi_{mk} = \exp\left(\frac{-\Delta u_{mk}}{RT} \right) \tag{5.9}$$

The equations for the temperature dependence for UNIVAP (Eq. 5.10) and EBGVAP (Eq. 5.11) are:

$$\Delta u_{mk} = R \cdot (a_{mk} + b_{mk} T + \exp(c_{mk} T^2) \cdot \mathrm{K}) \tag{5.10}$$

$$\Delta u_{mk} = \left(b_{mk}T + \exp\left(c_{mk}T^2\right) \cdot \text{J/mol}\right) \tag{5.11}$$

Here a_{mk}, b_{mk} and c_{mk} are the interaction parameters, which have to be fitted. Considering Eq. 5.6 and 5.7, the heat of vaporization in Eq. 5.4 should have the unit J/mol. Usually, heats of vaporization extend over a range between 0 J/mol and $> 10^5$ J/mol. This leads to difficulties in the optimization procedure, because exponential terms describing the temperature dependence as given in Eq. 5.10 and 5.11 cannot correlate data within this large range with satisfying results. A factor of about 1000 is introduced and therefore the output of Eq. 5.4 is set to kJ/mol.

5.2.2 Neural Networks

Neural networks are able to acquire an internal model of a process by learning from examples. After successful training the network will be a model for the process which led to the experimental data. Theoretical results show that feed-forward networks are capable of arbitrarily exact function approximation, given an unlimited number of free parameters or infinite precision [31].

In these experiments simple feed-forward networks with sigmoid activation functions (Eq. 2.3) are used. The network model can be written as:

$$out = \frac{1}{1 + e^{-\left(\left(\sum_{i=1}^{n} w_{out,i} \times hid_i\right) + w_{ou,bias}\right)}} \tag{5.12}$$

$$hid_i = \frac{1}{1 + e^{-\left(\left(\sum_{j=1}^{m} I_j \times w_{i,j}\right) + w_{i,bias}\right)}} \tag{5.13}$$

with n as the number of hidden units, m as the number of input units and I_j as input value for unit j.

As training algorithms for the network weights $w_{i,j}$ we employed the standard backpropagation algorithm, RProp (Resilient Propagation) and various (μ,λ) evolution strategies.

5.3 Description and Preparation of the Data

Before the models can be fitted to experimental data, descriptors had to be derived from the molecular structure of chemical compounds. Using the descriptors, data for the enthalpy of vaporization can be used to generate training data for the neural network.

5.3.1 Selection of Descriptors

The question arises which descriptors should one choose to describe the process itself. There are several possibilities for the definition of descriptors [22]:

- number of atoms
- molar mass
- single bonds
- topological parameters describing the connectivity between atoms.
- dipole moment
- surface fractions
- temperature

In the following experiments we used *surface fractions* of the functional groups and the *temperature* as descriptors which are also used by group contribution models. Experimental data was gathered for several functional groups, and different problem sizes (number of functional groups) were generated to evaluate the scaling behavior of the models. In table 5.1 we see the dimension and number of model parameters for three different datasets.

	3 groups	5 groups	20 groups
Data points	429	645	1549
Dimensions	4	6	21
Model parameters	25	41	390

Table 5.1: Three different datasets with growing number of functional groups involved and the related problem dimensions.

5.3.2 Selection of Data

The experimental data concerning the enthalpy of vaporization (table 5.2) were taken from different data handbooks [49, 85, 80]. Data for three different classes of chemical compounds were used: normal alkanes, 1-alcohols, and branched alcohols. These data were chosen for the investigation of three (3MG), five (5MG) and twenty (20MG)[1] functional groups, the so-called main groups: CH_3, CH_2 and

[1]Due to the complexity of the task, only very few experiments were made with 20 main groups. Those experiments were also restricted to neural network learning because the problem size was intractable for the physical models.

CH_nOH. The group CH_nOH contains the functional groups CH_3OH, CH_2OH and CHOH. The experimental data for all data sets cover a temperature range from 92 K to 776 K. The number of carbon atoms in the n-alkanes ranges from 2 (Ethane) to 19 (Nonadecane), for the 1-alcohols from 1 (Methanol) to 14 (Tetradecanol) and for the branched alcohols from 4 (2-Methyl-2-propanol) to 6 (2-Methyl-2-pentanol). The preprocessing steps and experimental settings were the same for the 3MG and 5MG data sets.

Group interaction for 3 main groups	n_p	$n_{data,total}$	$n_{data,training}$
CH_3 CH_3/CH_3 CH_2/CH_2 CH_2	12	248	128 (51.61 %)
CH_nOH CH_nOH/CH_3 CH_nOH/CH_2 CH_nOH	15	181	86 (47.51 %)
total:	27	429	214 (49.88 %)
Group interaction for 5 main groups	n_p	$n_{data,total}$	$n_{data,training}$
CH_3 CH_3/CH_3 CH_2/CH_2 CH_2	12	248	130 (52.42 %)
CH CH/CH CH_2/CH CH_3	15	133	58 (43.94 %)
C C/C CH/C CH_2/C CH_3	21	52	28 (53.85 %)
CH_nOH CH_nOH/CH_3 CH_nOH/CH_2 CH_nOH	15	181	89 (49.17 %)
CH CH_nOH/C CH_nOH	12	40	22 (55.00 %)
total:	75	654	327 (50.00 %)

Table 5.2: Number of experimental data for the different group interactions.

5.3.3 Partitioning for Cross-Validation

After generating a data set (either 3MG or 5MG) it was subdivided into 3 classes: training (50%), validation (25%) and test (25%) set. Only the training set was used to adapt the parameters for all our models. The validation set could be used during the adaptation process to evaluate the algorithm's performance on unknown data and stop the adaptation process if the error on the validation set increases. The validation set does not have any influence in the learning procedure of the NN or the adaptation process of the physical models UNIVAP and EBGVAP. Validation and test set therefore measure the generalization ability of all models. In this context it was important to choose the partitioning of the data so that all group interactions are equally present in all partitions. The distribution of the data in the 3MG and 5MG data sets can be seen in table 5.2.

5.3.4 Transformations

For the use with the neural network the data has to be normalized to be used as input values to the neurons. Several normalization techniques were investigated:

- Linear scaling (Eq. 5) over all inputs (unit activation range $[0.0 \ldots 1.0]$ and $[-1.0 \ldots 1.0]$).
- Separate linear scaling of main-groups, temperature and enthalpy (unit activation range $[0.0 \ldots 1.0]$ and $[-1.0 \ldots 1.0]$)
- Linear and separate linear scaling to the interval $[0.1 \ldots 0.9]$ and a unit activation range $[0.0 \ldots 1.0]$.
- Separate hyperbolic scaling (Eq. 2.28) to the interval $[0.1 \ldots 0.9]$ and a unit activation range $[0.0 \ldots 1.0]$.

In all of the following experiments the data was normalized via separate linear transformations of main-groups, temperature and enthalpy to the interval $[0.1 \ldots 0.9]$ with a unit activation range of $[0.0 \ldots 1.0]$. Network responses outside of this interval were mapped onto the boundaries and then re-transformed to the original scale. This scaling technique delivered the best results in preliminary experiments.

5.4 Experiments and Results

There was one basic question which was the starting point of these experiments: "Can neural networks be used to model chemical processes?" Throughout the experiments several other questions arose which also had to be answered: "How does a neural network compare to physical models?", "How good is the scaling behavior of the different methods?" "What is the best architecture, training algorithm and parameter setting for such a network?", "If evolution strategies are good at optimizing physical models, can they also be used to optimize the neural network's weights?"

All these questions and their answers lead to a rather complex experiment structure. Figure 5.4 illustrates the main structuring of the experiments and results.

Comparing different methods or models is at least two-fold. On the one hand a fair comparison should allow all models the same number of free parameters to adjust to the problem. On the other hand, one can say that it is sufficient if a model performs well on formerly unseen data regardless of the number of parameters it needed. In both cases our main concern is the generalization capability.

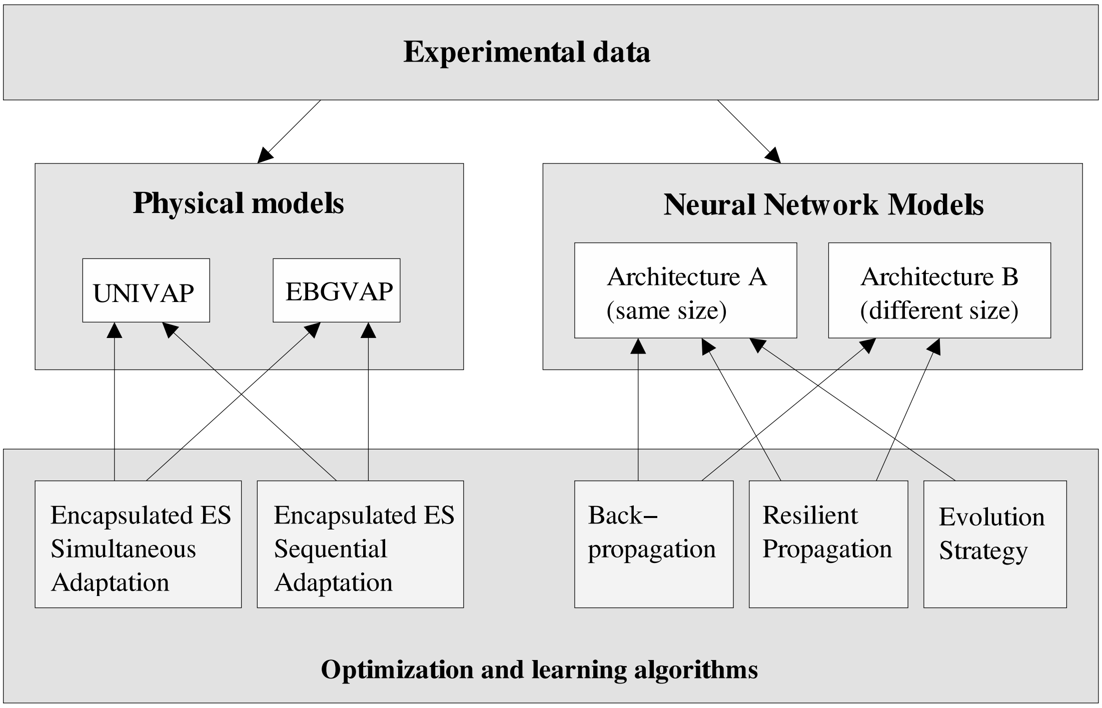


Figure 5.4: Different CI approaches under investigation.

Some of these experiments were designed to find good neural models under the most similar conditions for the calculations as the physical models. In these experiments the number of adjustable parameters was approximately the same for all models. Experiments of this type are tagged as “fixed”.

The other type of experiments (tagged as “free”) was designed to search for good results independent of the number of free network parameters (weights) used. In this case we are not restricted to a certain number of weights in the neural network and the difficulty is to find the optimal structure of the network[2].

Figure 5.5 illustrates the usage of the physical and neural network model. In the physical model interaction parameters between the functional groups have to be adapted in order to approximate the actual heat of vaporization ΔH_v. In case of the neural network the weights have to be adjusted to reflect the correlation between the functional groups and their heat of vaporization ΔH_v.

[2]Another important issue that is not addressed here is the problem of finding a good model equation for the temperature dependency of the physical model. Here, only the structure of the network is under investigation.

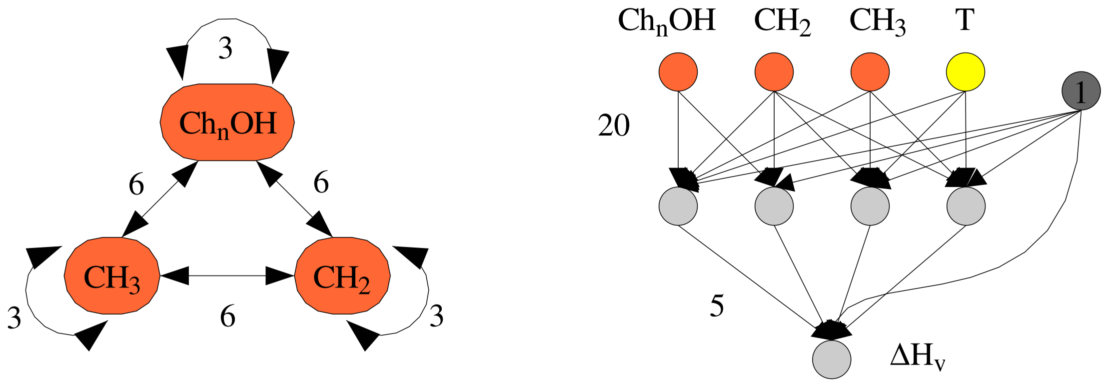


Figure 5.5: Physical model (left) with 27 interaction parameters between functional groups, and neural network (right) with 25 weight parameters.

5.4.1 Physical Model Experiments

Only the training set was used for the non-linear regression of the interaction parameters. Two different fitting methods were investigated: sequential fitting (seq) and simultaneous fitting (sim). With the sequential methods the parameters were computed successively, i. e. first the 12 parameters for the interactions $CH_3 \leftrightarrow CH_3$, $CH_3 \leftrightarrow CH_2$ and $CH_2 \leftrightarrow CH_2$ were fitted to the training data set. After this optimization process (corresponding to 3MG in table 5.2), these 12 parameters are used in the fitting procedure of the remaining 15 parameters of the interactions $CH_nOH \leftrightarrow CH_nOH$, $CH_3 \leftrightarrow CH_nOH$, and $CH_2 \leftrightarrow CH_nOH$, because data points of substances are used, which contain the main groups CH_3 and CH_2, too. The advantage of a sequential fitting procedure is to keep the dimension space as small as possible. These sequential experiments for the physical models were done with the aid of a repeatedly started encapsulated evolution strategy [21] by using a multidimensional but non-correlated step-length control and a global averaging recombination of the objective and the strategic variables: $[\texttt{AA 4+8(AA 7+19)}\ ^{300}]^{30}$ (see section 3.2). The encapsulated evolution strategies were run five times with $\approx 13.6 \cdot 10^6$ objective function evaluations each, in order get a fairly good local minimum, at least.

The determined results were optimized by a multi-start simplex-algorithm [56] with 50 different runs of 2500 iterations each. The best result for UNIVAP (seq) and EBGVAP (seq) can be found in tables 5.3 and 5.4.

In contrast to this sequential regression of the model parameters a simultaneous regression (sim) of all 27 (3MG) respectively 75 (5MG) parameters was investigated by using the same encapsulated ES as for the sequential experiments. The determined errors (MAE Eq. 5.1) of these runs were improved by a multi-start simplex-method as well by using 50 different runs of 3000 iterations each.

5.4.2 Neural Networks Experiments (Backpropagation)

The learning rate η and the architecture of the network (number of hidden units and connections) have the strongest influence on the performance of the network [52]. In case of the fixed experiments we only have to find a good learning rate η. The learning rate was varied with a fixed architecture which had approximately the same number of free parameters (weights) as the UNIVAP respectively as the EBGVAP model.

In case of the free experiments a limited parameter study was carried out. With the best learning rate found, we searched for a good network architecture through variation of the number of hidden units[3]. All runs were performed 10 times. For a documentation of the results see the appendices A.4.3 and A.4.4.

5.4.2.1 Variation of the Learning-Rate

The architecture of the network was fixed at 4 input, 4 hidden and 1 output units (4-4-1) for the 3MG data and at 6 input, 5 hidden and 1 output units (6-5-1) for the 5MG data, to have approximately the same number of free parameters ($25 = 4 \times 4 + 4\,\text{bias} + 4 + 1\,\text{bias}$) as the UNIVAP and EBGVAP methods.

For both data sets (3MG and 5MG) the learning rate was varied between $\eta = 0.001$ and $\eta = 10.0$, the momentum term α was fixed to 0.2. A training run was stopped after it reached the error limit ($msse \leq 5 \times 10^{-5}$) or exceeded a maximum number of 100,000 pattern presentations (epochs). None of the runs showed an excessive overfitting of the data. The error on the validation set very seldom increased with a decreasing error on the training set. Because of this observation the stopping criterion was only defined on the training set.

The figures 5.6 and 5.7 show the error curves for 10 different runs (3MG) with the best learning rate which was used throughout all other experiments. The left-hand side figure gives the error on the training set, and on the right-hand side we see the validation error. If an error curve reaches the base of the graph it satisfied a specified error limit for the whole training set. Networks with very low learning rate never reached the specified error limit, due to the very slow learning progress. A rate too high resulted in oscillating error curves. The figures look similar for the 5MG data set.

5.4.2.2 Variation of the Number of Hidden Units

After the variation of the learning rate η we used the best rate as a constant for the hidden unit search. The number of hidden units varied between 1 and 40. Networks with less then 3 units failed to learn the task. Up to 40 units

[3]This does not mean that both parameters are independent of each other. We consider this value to be a first estimate to start with.

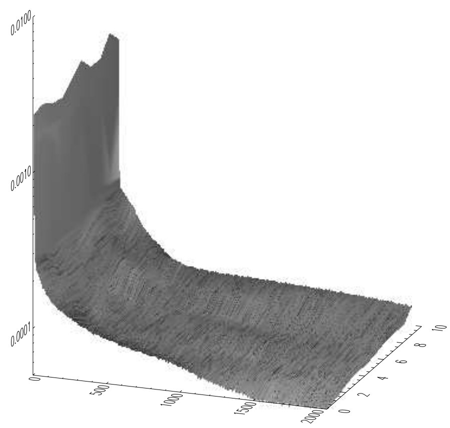


Figure 5.6: 3MG: training set errors at learning rate $\eta = 0.8$. The figure shows the error (Eq. 2.6) of the networks (y-axis) over 100,000 training epochs (x-axis) for all 10 runs (z-axis). Only every 50th epoch is shown.

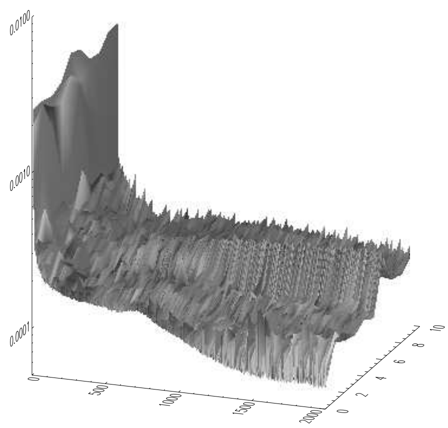


Figure 5.7: 3MG: validation set errors at learning rate η=0.8. The figure shows the error (Eq. 2.6) of the networks (y-axis) over 100,000 training epochs (x-axis) for all 10 runs (z-axis). Only every 50th epoch is shown.

the results on training as well as validation data were almost independent of the number of units employed. We therefore used our initial 4-4-1 (NN-A) and a 4-6-1 (NN-B) network for the 3MG data and a 6-5-1 (NN-A) and a 6-9-1 (NN-B) network for the 5MG data. In some cases networks with slightly more or slightly less units perform a little better. These networks are included in the comparison. All networks were trained with backpropagation as well as RProp. For the final training runs 250,000 epochs were used instead of the 100,000 of the parameter study.

5.4.3 Neural Networks Experiments (Evolution Strategy)

In this experiment we substituted the backpropagation algorithm with an evolution strategy. Some authors [94] reported good results when training a network with an ES. Again we systematically searched for a good parameterization of the (15,100)–ES. Parameters under consideration were the number of mutation step-sizes σ_i and the recombination scheme used on the object variables x_i (the network weights) and the step sizes. Each parameter setting was run for 100,000 generations (10,000,000 function evaluations) and repeated 10 times to have some statistical validity. All of the 12 possible combinations (see table 4.1) of discrete, intermediate, global averaging and geometric recombination of object variables and step size variations as well as a variant without recombination were done with either $n_\sigma = 1$ and $n_\sigma = 25$, local and global recombination. See appendix A.5.1 for detailed results of the parameter study.

None of the parameter settings lead to good and reliable results. Only one out of all ES trained networks performed comparably to backpropagation. All other networks give rather poor results. The quality of the average result did slightly improve when using backpropagation as local search procedure (an additional training of 250,000 epochs) after ES optimization but was not as good as backpropagation alone. Figure 5.8 shows the best run, which can be regarded as a very rare event, with a (15,100)–ES. Because of the poor performance no ES experiments were performed on the 5MG data set. Typical results for other recombination schemes are given in the appendix A.5.1.

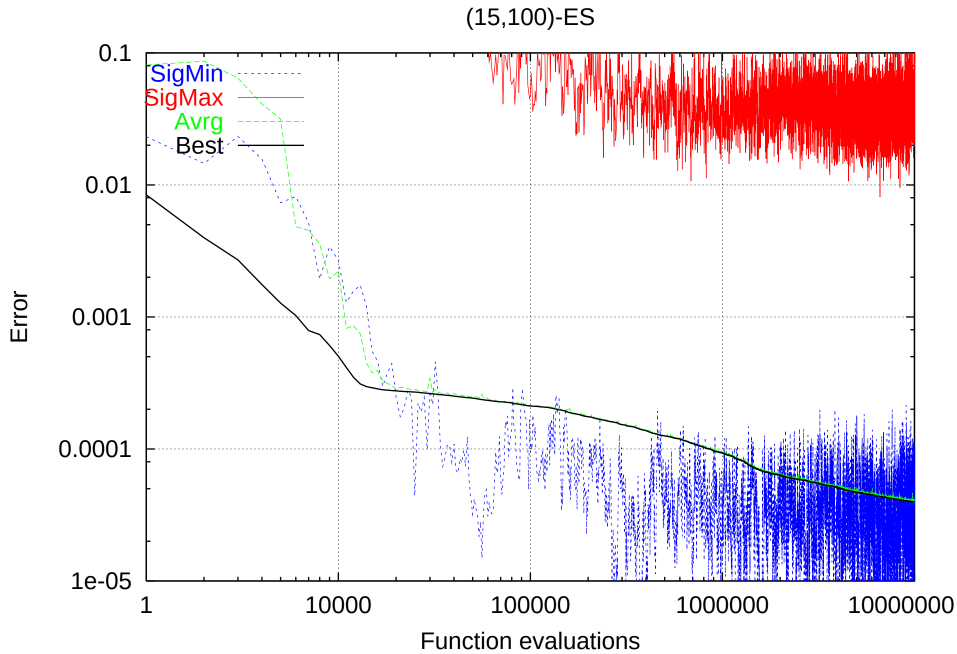


Figure 5.8: 3MG: (15,100)–ES with n step sizes and intermediate recombination of weights and step sizes. Depicted are the best (Best) and average (Avrg) individual as well as the minimal (SigMin) and maximal (SigMax) step size in the population over 10,000,000 function evaluations.

5.4.4 Results and Comparison

5.4.4.1 The 3 Main Groups Data Set (3MG)

For a comparison between the physical models and NN, the two network architectures with learning rates gained by the previous experiments are used. Architecture A has 4 hidden units and nearly the same number of free parameters (25 weights) as the UNIVAP respectively the EBGVAP model (27). Architecture B performs alike and has 6 hidden units (37 weights).

List of experiments and their parameters:

1. Parameters for NN-A (4-4-1): epochs=250,000
   a) with backpropagation (NN-A-BP), η=0.8
   b) with RProp (NN-A-RP) , $\Delta_0 = 0.2$, $\Delta_{max} = 50.0$

2. Parameters for NN-B (4-6-1): epochs=250,000
   a) with backpropagation (NN-B-BP), η=0.8
   b) with RProp (NN-B-RP), $\Delta_0 = 0.2$, Δ_{max}=50.0

3. Parameters for NN-A-ES (4-4-1): best (15,100)–ES, $n_\sigma = n$, intermediate recombination of x_i and σ_i (100,000 generations)
   a) best network of all experiments
   b) average network

Table 5.3 and figure 5.9 give an overview of all experiments. In the first place, the results determined by the physical models UNIVAP and EBGVAP show, that the newer group contribution model EBGVAP is more suitable than UNIVAP for the correlation and prediction of heats of vaporization because of its better physical background. On the other hand the results show a superiority of a sequential fitting procedure. The neural network with plain backpropagation performs even slightly better than the best physical model. Trained with RProp the network clearly outperforms all physical models. Similar results are achieved by a network with a linear instead of a sigmoid output unit.

From the errors of the validation and test set we can derive the generalization capabilities of the different models. The best generalization is given by network A (backpropagation and RProp) very closely followed by EBGVAP (seq), whereas the worst generalization is delivered by the same network trained with an evolution strategy and the UNIVAP (sim) model. For a more detailed view of the errors see appendix A.4.1.

As an additional test for the generalization ability, all data of an ethane molecule in a range from 92 K to 305 K was used. In figures 5.10 and 5.11 we compare all models on the enthalpy prediction for ethane. It can be seen that the physical model EBGVAP and the neural network perform equally well on this task, except for the critical regions near $T \to T_{cr}$ and $\Delta H_v(T_{cr}) \approx 0$ J/mol, where the network outperforms all other models. The prediction by using the network that was trained with an ES is however characterized by a discontinuity near $T \approx 105$ K, which is not interpretable thermodynamically. The prediction of this neural network model can therefore be only used over a part of the temperature

	UNIVAP (seq)	EBGVAP (seq)	UNIVAP (sim)	EBGVAP (sim)
Train	0.681	0.617	0.881	0.720
Valid	0.941	0.750	0.966	0.829
Test	0.557	0.576	1.003	0.766
All	0.716	0.640	0.933	0.759
	NN-A-BP	NN-B-BP	NN-A-ES (best)	NN-A-ES (avrg)
Train	0.652	0.570	0.612	1.143
Valid	0.566	0.878	0.876	1.536
Test	0.686	0.703	0.747	1.357
All	0.638	0.679	0.711	1.292
	NN-A-RP	NN-B-RP	NN-A-BP (lin)	
Train	0.425	0.464	0.559	
Valid	0.604	0.690	0.818	
Test	0.560	0.600	0.641	
All	0.504	0.555	0.645	

Table 5.3: 3MG: Mean absolute error (Eq. 5.1) per pattern for different data sets and models.

range. The network trained with backpropagation suffers from a very small discontinuity at $T \approx 160$ K while RProp does not show any discontinuity and is therefore plausible from a thermodynamic viewpoint.

Almost all networks trained with an ES only give a poor approximation of the enthalpy curve. In comparison to table 5.3 the superiority of EBGVAP over UNIVAP can be also seen in the figures 5.10 and 5.11 because of the smaller deviation at temperatures smaller than $T = 150$ K and at temperatures near the critical temperature of T_{cr}(Ethane)= 305.4 K.

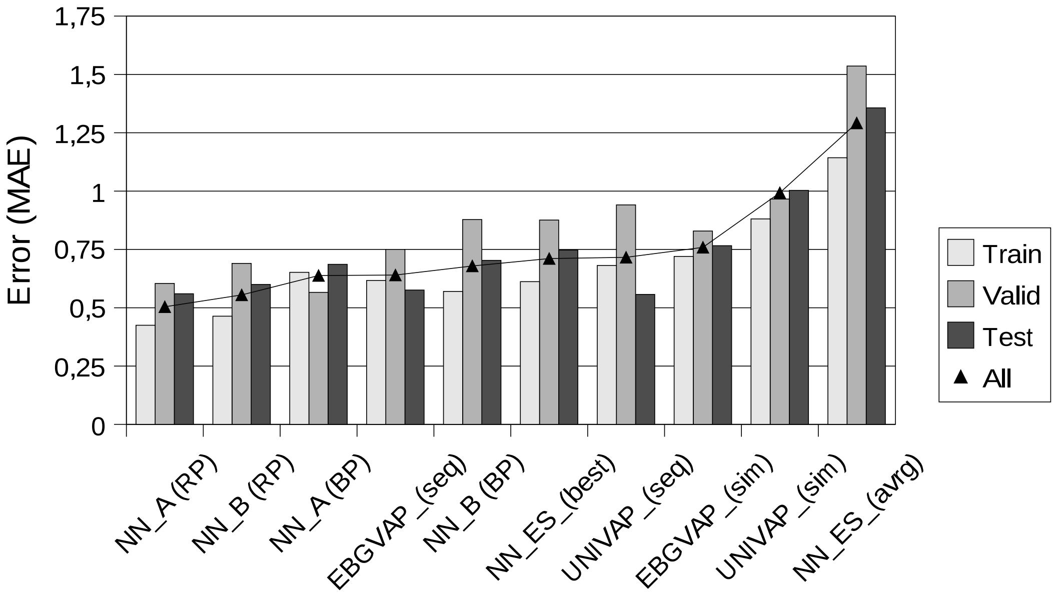


Figure 5.9: 3MG: Performance comparison of all methods. Methods are sorted in ascending order with respect to their overall error. The best three ranks on the left-hand side are held by neural network models.

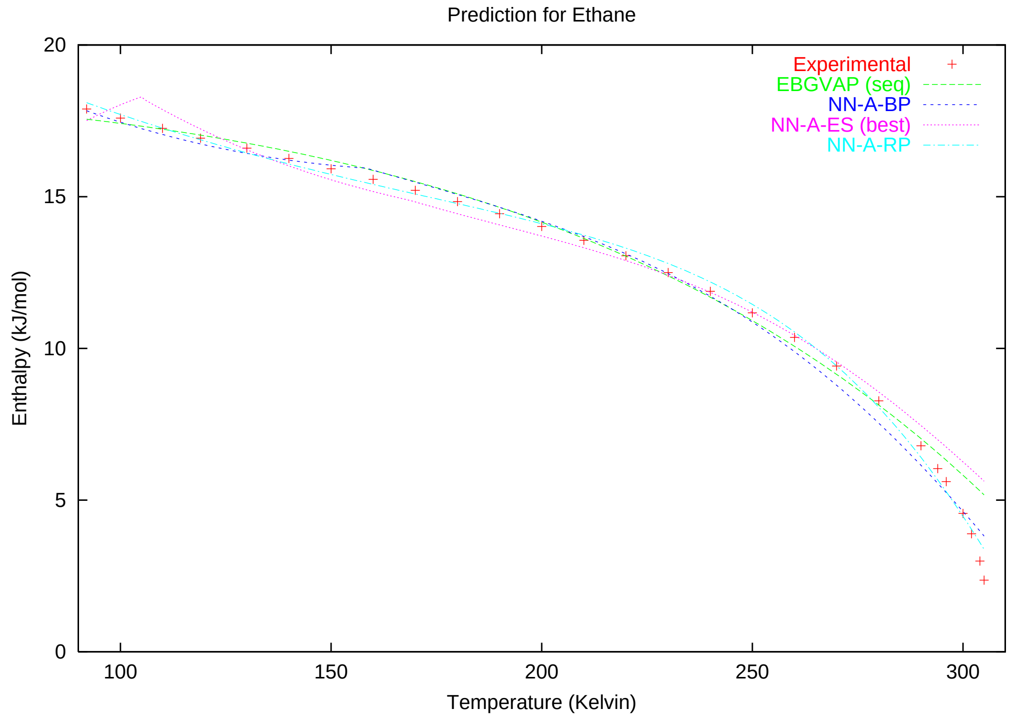


Figure 5.10: 3MG: Several well performing models (EBGVAP (seq), NN-A-BP, NN-A-ES (best), NN-A-RP) predicting the enthalpy of ethane over the whole temperature range. Discontinuity of NN-A-ES at $T \approx 105$ K and of NN-A-BP at $T \approx 160$ K.

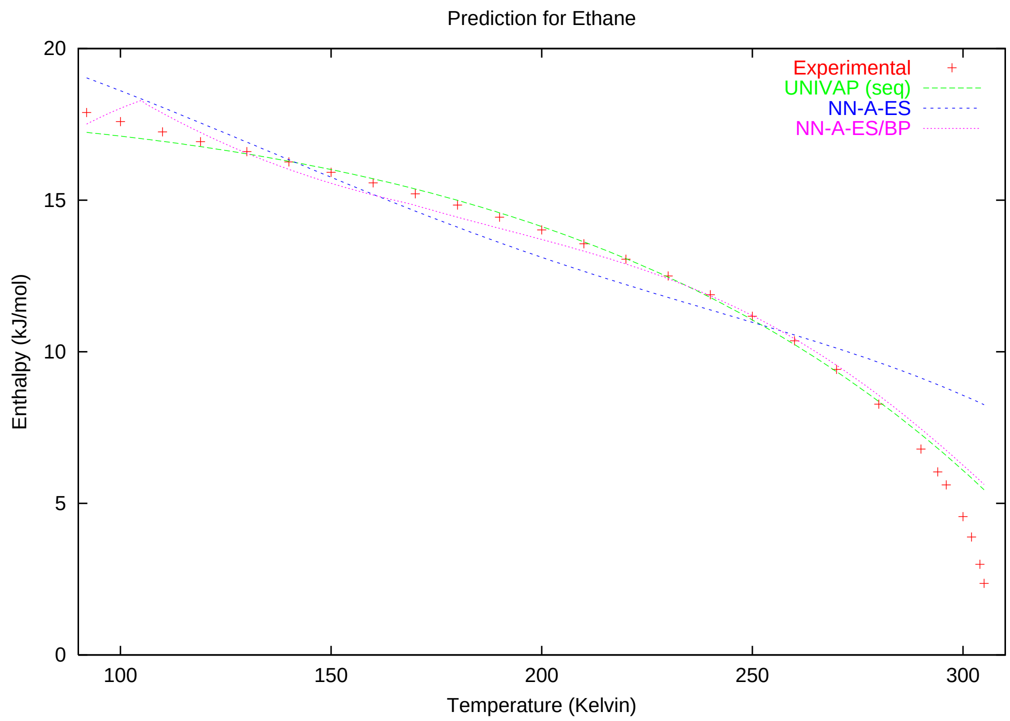


Figure 5.11: 3MG: Several poorly performing models (UNIVAP (seq), NN-A-ES (average), NN-A-ES/BP) predicting the enthalpy of ethane over the whole temperature range. Discontinuity of NN-A-ES at $T \approx 105$ K.

5.4.4.2 The 5 Main Groups Data set (5MG)

As in the 3MG experiments, we took two network architectures with learning rates gained by the previous experiments. Architecture B has 5 hidden units (41 weights), architecture A has 9 hidden units (73 weights) that is nearly the same number of free parameters of the physical models UNIVAP and EBGVAP (75 parameters).

1. Parameters for NN-A (6-9-1): epochs=250,000
   a) with backpropagation (NN-A-BP), η=0.9
   b) with RProp (NN-A-RP), $\Delta_0 = 0.2$, Δ_{max}=50.0

2. Parameters for NN-B (6-5-1): epochs=250,000
   a) with backpropagation (NN-A-BP), η=0.9
   b) with RProp (NN-A-RP), $\Delta_0 = 0.2$, Δ_{max}=50.0

Table 5.4 and figure 5.12 give on overview of all 5MG experiments. Again we see that the EBGVAP model is superior to the UNIVAP model but both neural networks perform better than the best physical model. Network B has only 55 % of the free parameters of the models UNIVAP and EBGVAP but gives slightly better results, whereas network A with nearly the same number of parameters is significantly better. With the increased problem size the simultaneous adaptation method of parameters loses even more ground compared to the sequential method. The results concerning the physical models show the need of decreasing the dimension of the variable space. It is obvious, that a simultaneous optimization (sim) of 75 parameters in all did not lead to satisfying results. To split the optimization procedure of all 75 parameters into several sequential optimizations (seq) by using already fitted parameters as constants leads to the best results which can be seen in the figures A.14 and A.15 in the appendix.

In figures 5.13 we again compare all models on the enthalpy prediction for ethane.

When we take a closer look at the performance of the sequential and the simultaneous adaptations of the physical model, we clearly see that the sequential method outperforms the other on both problem sizes. A detailed comparison of both fitting methods is given in the appendix A.4.2.

5.4.4.3 Larger Data Sets

This experiment was mainly designed to show the scaling behavior of all methods. Here, we only have one network architecture and one physical model. The number of parameters to adapt is 390 for the EBGVAP model and 392 for the neural

	UNIVAP (seq)	EBGVAP (seq)	UNIVAP (sim)	EBGVAP (sim)
Train	0.667	0.609	1.764	1.275
Valid	1.050	0.805	2.077	1.579
Test	1.180	0.939	2.013	1.707
All	0.891	0.740	1.904	1.459
	NN-B-BP	NN-A-BP	NN-B-RP	NN-A-RP
Train	0.702	0.564	0.684	0.412
Valid	0.649	0.589	0.614	0.396
Test	0.904	0.744	0.841	0.514
All	0.737	0.614	0.705	0.433

Table 5.4: 5MG: Mean absolute error (Eq. 5.1) per pattern for different data sets and models.

network. The network was trained with backpropagation and RProp. For the EBGVAP model only the better sequential adaptation method was investigated and, in contrast to the 3MG and 5MG experiments, all available data was used. For the networks models, results are given for training with all available data and for training only with training data. In both cases the errors are given for all three data sets.

1. Parameters for NN-A (21-17-1): epochs=100,000
   a) with backpropagation (NN-A-BP), η=0.8
   b) with RProp (NN-A-RP), $\Delta_0 = 0.2$, Δ_{max}=50.0

Table 5.5 gives an overview of all 20MG experiments, and figure 5.14 compares models for all data sizes. Again, the neural networks perform better than the physical model. The neural network's outperform the physical model by almost a factor of 3. Even a network that has only seen 50% of the data performs twice as good as the physical method.

The following two figures summarize the performance of the best models for all problem sizes. We can clearly see that the neural network models scale very well with the growing problem size, whereas even the best physical model with sequential adaptation of parameters suffers from decreasing performance.

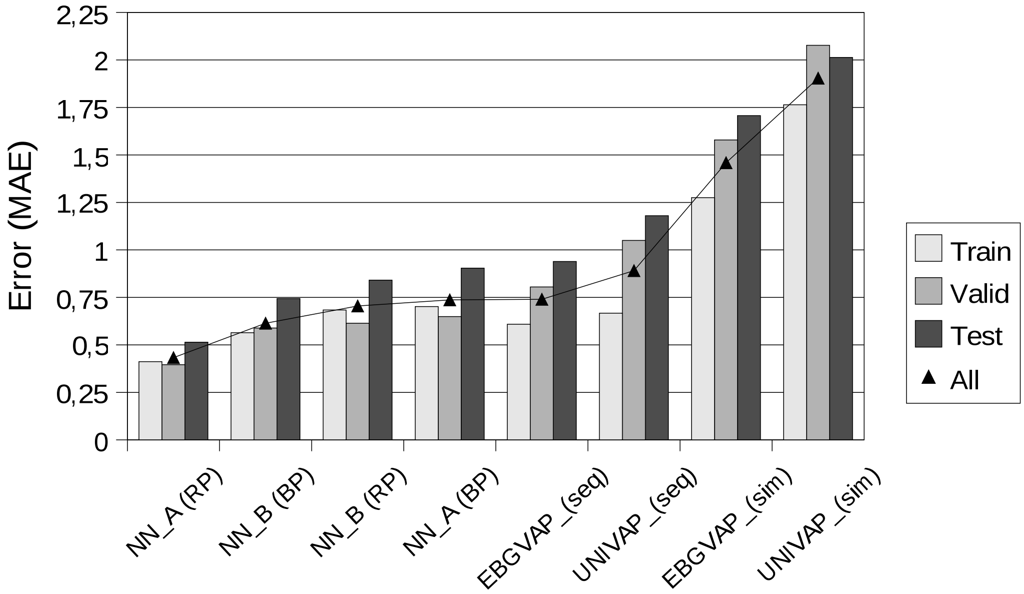


Figure 5.12: 5MG: Performance comparison of all methods. Methods are sorted in ascending order with respect to their overall error. The best four ranks on the left-hand side are occupied with neural network models.

	NN-A-BP (all)	NN-A-RP (all)	EBGVAP (seq)
Train	0.627	0.779	-
Valid	0.668	0.813	-
Test	0.670	0.889	-
All	0.648	0.695	1.902
	NN-A-BP	NN-A-RP	
Train	0.737	0.650	
Valid	0.998	0.900	
Test	1.049	0.820	
All	0.880	0.755	

Table 5.5: 20MG: Mean absolute error (Eq. 5.1) per pattern for different data sets and models.

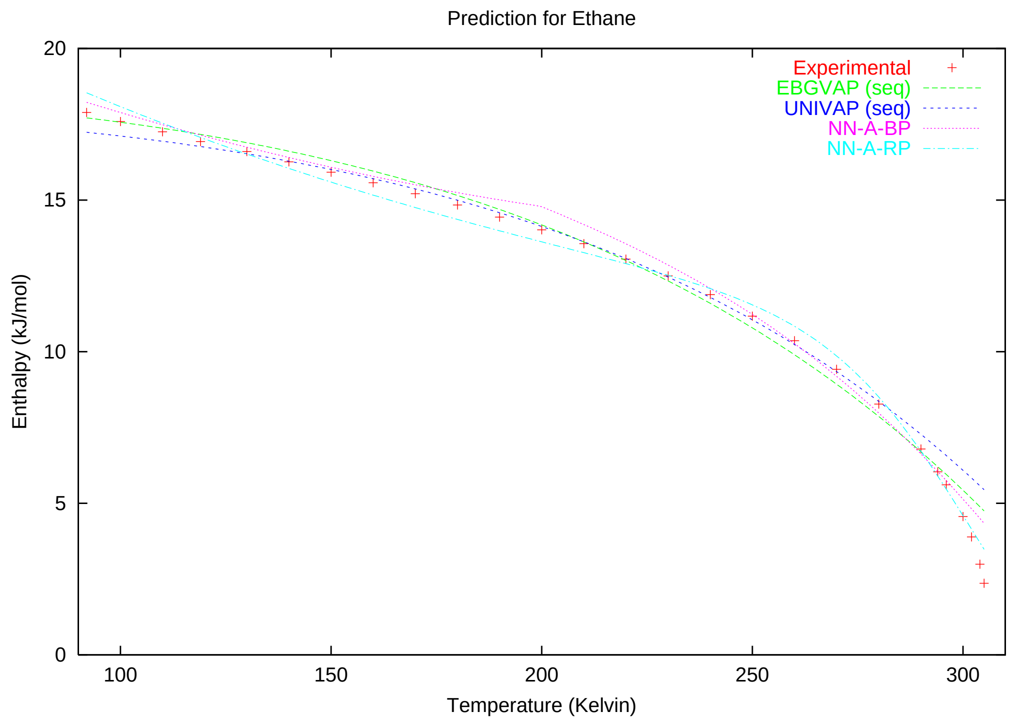


Figure 5.13: 5MG: Several well performing models (EBGVAP (seq), UNIVAP (seq), NN-A-BP, NN-A-RP) predicting the enthalpy of ethane over the whole temperature range.

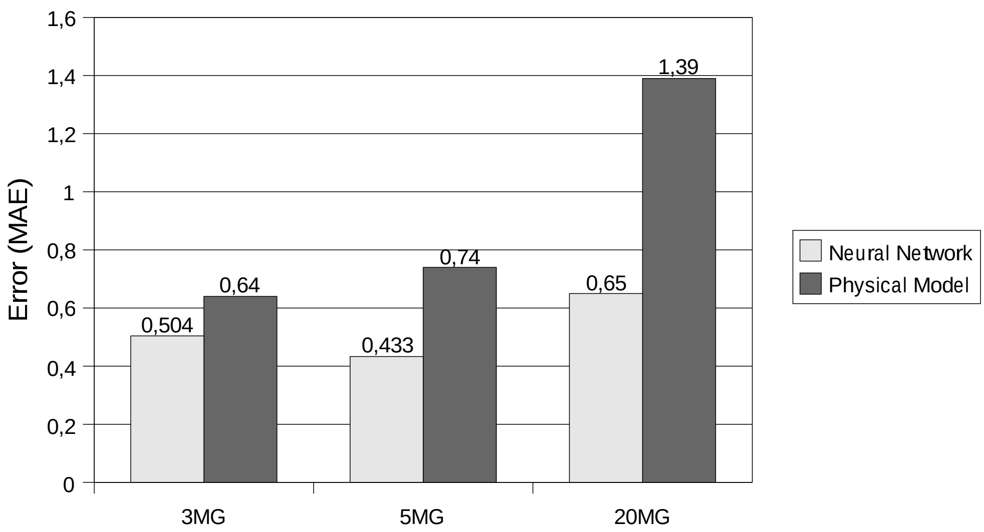


Figure 5.14: Performance comparison of the best neural network and the best physical model on varying problem sizes.

5.4.5 Run-Time Comparison of Different Approaches

Another evaluation criterion for the neural network and the physical model approach is the computational effort that is needed to achieve approximately the same level of performance. Both approaches are complex and different in their algorithmic nature. Hence, a formal runtime analysis is not feasible, instead the CPU-time per model evaluation and the total number of evaluations needed to achieve the same error level are used[4]. We compare the best physical model EBGVAP with simultaneous and sequential adaptation with two neural network architectures. The sequential adaptation of parameters is much more time consuming than the simultaneous adaptation. Depending on the number of main groups involved, the time consumption grows approximately by a factor equal to the number of sequential adaptation steps[5].

In case of the physical model a single evaluation corresponds to one objective function evaluation plus the fraction of time that is consumed by the ES for this single evaluation. Here, most of the time is spent during the calculation of the objective function (the physical model). For the neural network one evaluation is equivalent to a learning epoch.

Table 5.6 lists the time in CPU-milliseconds for a single evaluation and the total number of evaluations needed. For the two smaller problems 3MG and 5MG the physical models need ≈ 3 times as much CPU-time as the neural networks. For the larger 20MG problem the factor is only ≈ 1.2.

problem size	EBGVAP (sim)	EBGVAP (seq)	evaluations	NN-A	NN-B	evaluations
3MG	6.72	13.44	1,368,244	2.38	2.75	250,000
5MG	13.99	69.95	1,368,244	4.22	5.61	250,000
20MG	84.19	2,525.7	1,368,244	70.90		100,000

Table 5.6: Time consumption of physical and neural network models for various problems sizes. The figures give the number of CPU-milliseconds for a single model evaluation and the total number of evaluations needed.

When we multiply these figures by the total number of evaluations needed to achieve the results of this chapter, we end up with a drastically changed relation. In table 5.7 and figure 5.15 these figures are given in CPU-minutes. Compared

[4] A MIPS-R10000 250 MHz processor was used in all tests. All figures were gained by averaging the time consumption of both methods over 10,000 evaluations

[5] For the 3MG problem the factor was 2, for the 5MG problem the factor was 5, and for the large 20MG problem the factor could only be estimated to be between 20 and 40 (30 was used in these calculations). See section 5.4.1 for further details.

problem size	EBGVAP (sim)	EBGVAP (seq)	NN-A
3MG	153	306	10
5MG	319	1,595	18
20MG	1,920	57,596	118

Table 5.7: Total time consumption for various problems sizes. The figures give the number of CPU-**minutes** for the whole optimization.

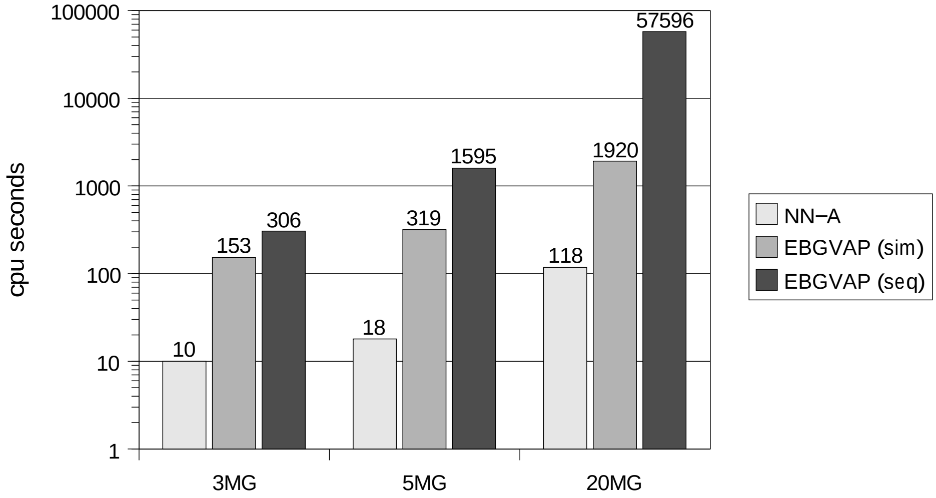


Figure 5.15: Comparison of time consumption for neural networks and physical models on all problem sizes.

to the simultaneous adaptation method, which is the one that is most similar in terms of the working principle, the neural network is at least 15 times faster than the physical model. A fairer comparison with the better sequential adaptation method gives a speed-up between 30 and nearly 500. The performance relation for the simultaneous adaptation method and the neural network seems to be constant with growing problem size, while it grows by a factor depending on the problem size for the sequential method. The time needed for the evolutionary training of the neural network is 24 times higher than that of backpropagation.

The largest 20MG problem can be solved in just two hours by the neural network approach, while the physical model, with the rather poor performing simultaneous adaptation, needs 32 hours. The sequential method cannot be used for this problem size because of its time consumption. It would have required approximately 40 days computing time.

5.5 Discussion and Conclusions

The most important result of this investigation is the good ability to correlate as well as to predict the enthalpy of vaporization with neural and physical methods. Neural networks with simple backpropagation or RProp training are as good as the physically based group contribution methods UNIVAP and EBGVAP and especially at critical temperatures even slightly better. Trained with RProp the network outperforms all other models. Predictions by networks, however, are sometimes characterized by discontinuities within the thermodynamically significant temperature range, which is shown in the figures 5.10, 5.11, and 5.13. These points are not thermodynamically interpretable, so that a prediction that exhibits such a discontinuity should only be used with care. None of the networks trained with RProp suffered from discontinuity.

The results concerning the sequential versus simultaneous parameter optimization of the 3MG and the 5MG data sets show the need of a relatively small dimension of variable space by carrying out a sequential optimization, where thermodynamic information is included as much as possible. Optimization of interaction parameters by using already fitted parameters as constants can lead to incompatibilities during the fitting procedure [42], but the results are obviously better in terms of mean absolute errors, which is shown in the figures A.14 and A.15. Neural networks, instead, have no need for a sequential adaptation method and do therefore not suffer from incompatibilities. During the training all parameters (weights) are adapted simultaneously.

The comparison of the results for UNIVAP and EBGVAP shows the influence of the structure of the model itself. Further investigations could use evolutionary algorithms to optimize the structure of the models with regard to the temperature dependence. For the neural networks it can be stated that the use of surface fractions of functional groups as descriptors for a neural network leads to good results for both correlation and prediction. The big advantage of this new procedure is, that the molecules can easily be divided into functional groups, which makes it easy to use in engineering applications and allows the direct comparison of neural networks and physical models, due to the same input information. The investigations concerning the architecture of the neural networks show, that a simple network structure is sufficient and a more complex network with short-cut connections or additional hidden layers does not give better results. In this context evolution strategies as training algorithms and combinations of ES with backpropagation failed to deliver good models in almost all experiments. It is worth to notice that ESs did not fail completely on the task, but, given the best possible parameter setting, they perform like a mediocre physical model.

From a thermodynamic point of view, it is interesting that a simple method like a neural network can give similar and even much better results in comparison with much more complicated, physically motivated models. If a physical model gives results with a quality less than a neural model, the physical model should be improved. However, in chemical engineering there are many thermophysical properties, which are usually not described by physical methods, but by incremental methods. These methods, for example, for critical data i.e. normal boiling points and so on, could be replaced by neural networks. However, these results are first steps in developing efficient network structures for our purpose and especially investigations with more functional groups will give a better comparison between physical models and neural networks.

An additional advantage of the neural models over the physical models is that their computational effort is much lower, and they exhibit a much better scaling behavior. While the physical models are, in terms of computational effort, only feasible up to a certain small number of dimensions (5 main groups with 41 model parameters), the neural models also perform well at a higher problem dimension (20 main groups with 390 model parameters). From the viewpoint of computation time, neural networks clearly outperform the best physical models.

Chapter 6

Summary

This thesis analyzed evolution strategies as an alternative to gradient-based neural network training. It investigated two CI methods, neural networks, evolution strategies, and a combination of both for solving difficult problems in the field of chemical engineering.

In chapter 4 it was shown that evolution strategies are, in principle, capable of successful weight-tuning in neural networks. This also holds when the network consists of non-continuously differentiable activation functions. Here, EAs are feasible means as backpropagation cannot be applied because of the not so easy to get gradient information.

A clear advice of how to recombine the weights of a neural network could be found. The best recombination scheme is: two parents (local), intermediate recombination of the object variables x and step sizes σ. It can further be said that the coupling of object variables and strategy parameters is beneficial when training networks and that it is not a critical parameter for other problems. Despite the capability of ESs to optimize the weights of a neural network, it must be said that except for small problems they are less efficient in terms of computational effort. For the easiest task an ES needs more than twice as many function evaluations and several orders of magnitude more for larger tasks.

In comparison with backpropagation ESs have more parameters to adjust (recombination scheme, population size, selection pressure, selection type ($\mu +, \lambda$), mutation factors τ and τ'), and those parameters are often sensitive to changes.

The investigation of the scaling properties of both algorithms suggest that ESs are much more likely to suffer from a large dimensionality than backpropagation. With a growing number of problem dimensions the learning becomes more difficult for ESs and the "correct" parameterization is crucial.

The hope that the evolutionary search might overcome the problem of getting trapped in non-global local minima like gradient-based methods do should be abandoned. This work suggests to recur to evolution strategies for neural network training only when no gradient information is available in the network. In this case ESs deliver promising results on small and moderate binary classification tasks.

In chapter 5 difficult real-world problems from the field of chemical engineering were solved with the aid of neural networks and evolutionary algorithms. The most important result of this investigation is the good ability to correlate as well as to predict the enthalpy of vaporization with neural and physical methods. It could be shown that neural networks with simple backpropagation or RProp training outperformed the best physical group-contribution methods.

The novel use of surface fractions of functional groups as descriptors for a neural network leads to good results for both correlation and prediction. A big advantage of this new procedure is that molecules can be divided easily into functional groups, which makes it easy to use in engineering applications. From a thermodynamic point of view, it is interesting that a simple method like a neural network can give similar and even much better results in comparison with much more complicated, physically motivated models. If a physical model gives results with a quality less than a neural model, the physical model should be improved.

An additional advantage of the neural models over the physical models is that their computational effort is much lower, and they exhibit a much better scaling behavior. While the physical models are, in terms of computational effort, only feasible up to a certain small number of dimensions, the neural models can also be employed for higher problem dimensionalities.

In chemical engineering there are many thermophysical properties, which are usually not described by physical methods but by incremental methods. Such methods for the prediction of critical data, normal boiling points and so on, could also be replaced by neural networks. A first investigation into the prediction of critical values (appendix A.3) delivered promising results. Compared to an incremental method a neural network trained on half of the available data is only slightly worse in terms of prediction quality. If a network has all data available it outperforms all other methods. From the viewpoint of chemical engineering this is a very promising result. Given the molecular structure and the surface fractions of the functional groups, a single model is sufficient to predict all three critical values.

Appendix A

Appendix

The following sections will document subsidiary results mentioned in earlier chapters or deal with more technical issues. They should be read with their context in the main text corpus in mind.

A.1 Factors that Influence Learning

A.1.1 Activation Functions

Experiments with different activation functions show that the logistic function is not always the right choice (see [46] and section 4.5). Decomposable problems with linear parts might benefit from linear activation functions, whereas oscillating time series are in some cases better modeled with trigonometric functions.

A.1.2 Slope of the Activation Function

The following two learning experiments illustrate the influence of the slope of the sigmoid activation function. In the first experiment the 2-bit parity problem (XOR), that needs non-linear activation functions, is used while the second experiment employs a digit classification problem [48, 50]. For each value of the activation functions slope γ 10 runs where performed and averaged. Only those runs that reached a fixed error-limit were considered successful and used for averaging.

From figure A.1 we learn that the speed as well as the success rate increases with larger γ. A slope of $\gamma = 4.0$ yields a highly non-linear activation function (see 2.1) and results in fastest training. Almost linear activation functions with $\gamma = 0.1$ lead to stagnation of the learning in all runs.

A differing observation can be made for the digit classification task. Figure A.2 shows the same variations of γ with a different outcome. A slope of $\gamma = 1.0$ leads to the fastest and most reliable learning.

From those two experiments it can be seen that the slope parameter γ has a significant influence on the speed as well as on the success rate of the neural network training. A "good" setting of the slope parameter is highly application specific. These facts make γ an ideal candidate to be included in an evolutionary search for good networks.

In this context some work has been done by Han, Moraga, and Sinne [27]. They investigated the effect of different activation functions and their parameterization (slope, dynamic range, and symmetry) in different layers of a feed-forward network. To find a good initial setting of the parameters they employed a genetic algorithm. It could be shown that properly parameterized networks exhibit faster learning and good generalization.

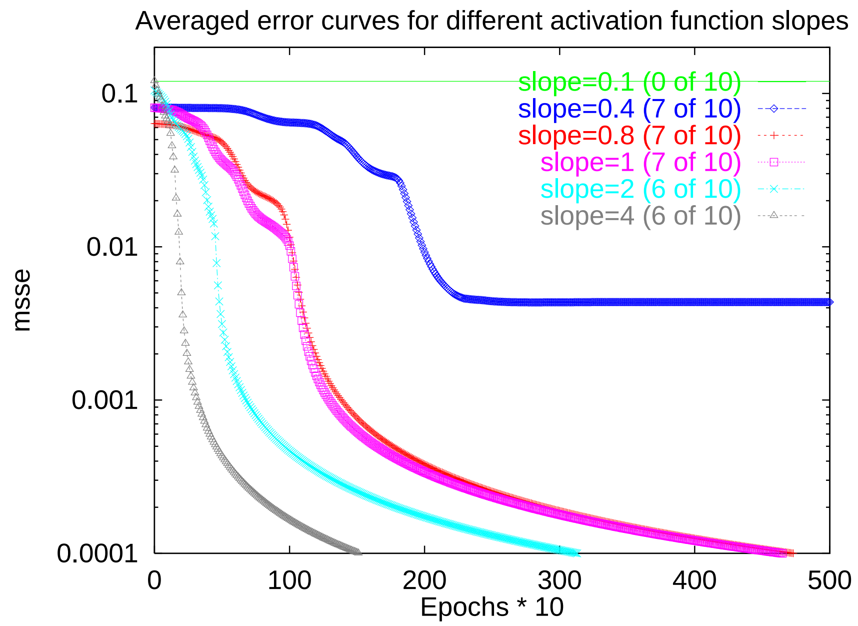


Figure A.1: XOR: Training ($\lambda = 0.8, \eta = 0.2$) with different activation function slopes. The averaged training errors over 10 runs and, in parentheses, the success rates are depicted.

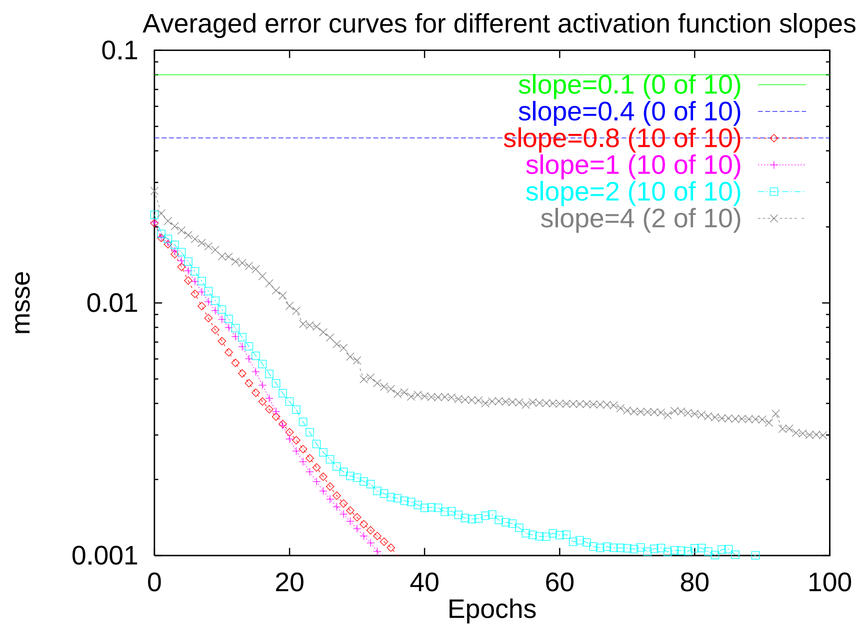


Figure A.2: Digit: Training ($\lambda = 0.8, \eta = 0.2$) with different activation function slopes. The averaged training errors over 10 runs and, in parentheses, the success rates are depicted.

A.1.3 Pattern Presentation

When presenting patterns to the network, one has to choose between random pattern presentation, where the sequence of all patterns is sampled anew for each epoch, and normal presentation, where the sequence remains the same over the whole training process.

Sometimes it can be advantageous to "pre-train" a network with a subset of patterns and continue the training with the whole the training data. This technique is called *bootstrapping* and can significantly reduce the training time as it can be seen in the following experiments.

- Problems: 6- and 9-bit parity
- Network 1: 6-6-6-1, fully connected (91 weights)
- Network 2: 6-63-1, fully connected (694 weights)
- Algorithm (BP): $\eta = 0.1$, $\alpha = 0$
- Termination: error below 10^{-4} (0% classification error) or 1,000,000 epochs.
- Runs: 10 runs. Only successful runs are averaged.

Here, bootstrapping was only explored for the parity problems of section 4.3.1. For the 6-bit parity problem the same network architecture (6-6-6-1) was trained with randomized presentation of all 64 patterns and with bootstrapping where the network was "pre-trained" with 6 ($\approx$ 10%) randomly chosen patterns for 10,000 epochs. In both cases 10 runs where performed and a maximum number of 1,000,000 epochs was allowed to reach the error limit of 10^{-4}. Only those runs that succeeded to reach 0% classification error were used for averaging.

In case of the 9-bit parity problem the architecture (6-63-1) was trained with randomized presentation of all 512 patterns and with bootstrapping where the network was "pre-trained" with 51 ($\approx$ 10%) randomly chosen patterns for 10,000 epochs. Again 10 runs where performed with a maximum of 1,000,000 epochs and an error limit of 10^{-4}.

	suc. rate	min	max	mean	sd
(6-bit) normal training	10:10	17,913	739,277	437,534	274,484
(6-bit) bootstrapping	10:10	17,103	63,578	32,017	13,647
(9-bit) normal training	10:10	71,342	596,712	295,748	226,453
(9-bit) bootstrapping	10:10	19,585	53,709	35,211	11,844

Table A.1: Comparison of normal training and bootstrapping. Given are the number of epochs to reach the error limit.

With bootstrapping the training time is significantly reduced. In case of the 6-bit parity problem normal training needed about 13.6 times more epochs than bootstrapping and about 8.4 times more epochs are needed in case of the 9-bit parity problem. In figures A.3 and A.4 typical learning curves for the 6-bit problem are shown with and without bootstrapping.

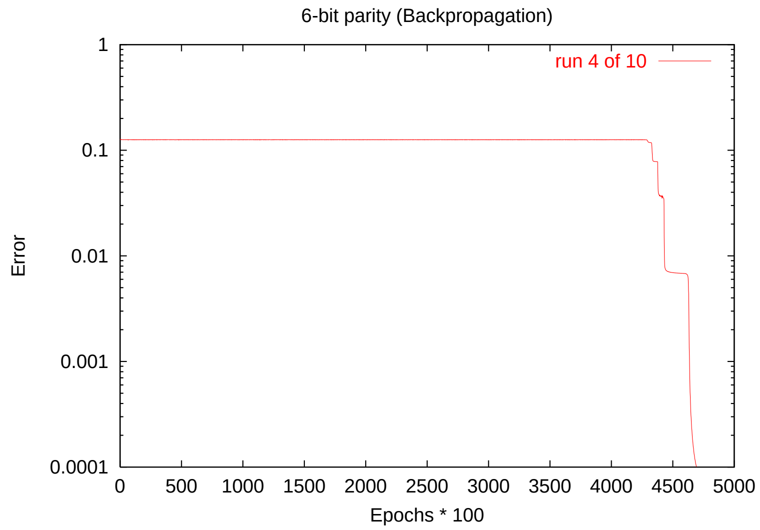


Figure A.3: Learning with randomized presentation of all 64 patterns.

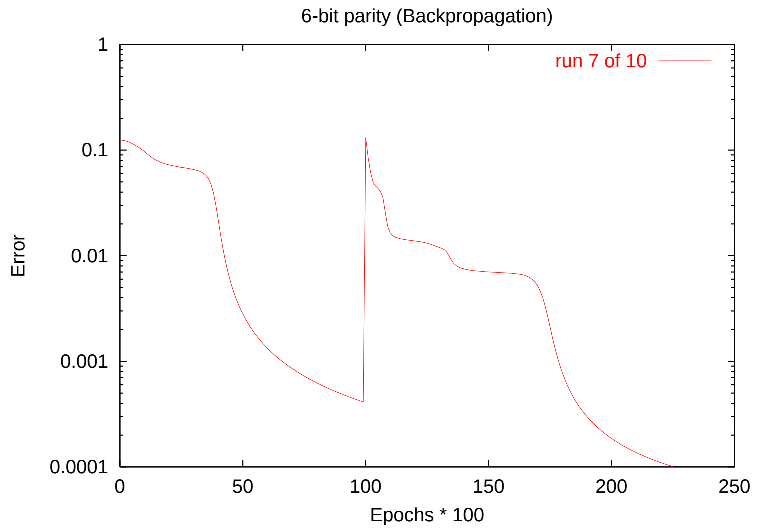


Figure A.4: Learning with bootstrapping. Pre-training with 6 randomly chosen patterns for 10,000 epochs.

The 6-bit parity experiment was repeated with RProp as training algorithm to see whether bootstrapping works in this case as well.

- Algorithm 1 (RPprop): $\Delta_0 = 0.2$, $\Delta_{max} = 50.0$
- Algorithm 2 (RPprop): $\Delta_0 = 0.2$, $\Delta_{max} = 50.0$, with weight decay $\alpha = 8.0$

	suc. rate	min	max	mean	sd
BP: normal	10:10	17,913	739,277	437,534	274,484
BP: bootstrapping	10:10	17,103	63,578	32,017	13,647
RProp: normal	1:10	3,500	-	-	-
RProp: bootstrapping	1:10	2,400	-	-	-
RProp: weight decay	7:10	1,820	9,215	3,727	2,361
RProp: weight decay boot.	2:10	4,540	29,822	-	-

Table A.2: Comparison of normal training and bootstrapping for RProp training (6-bit parity). Given are the number of epochs to reach the error limit.

For this task RProp is much more likely to get trapped in a non-global local optimum than backpropagation. Due to the small number of successful RProp runs no conclusions can be drawn about the effect of bootstrapping. If we introduce a weight decay term RProp becomes much faster than backpropagation but it's success rate is 30% lower. When using bootstrapping with the more successful weight decay RProp gets stuck is most cases.

A.2 Complexity Issues in Neural Networks

The following section introduces some basic definitions from the complexity theory. These are necessary to understand the theorems about learnability and topology determination in feed-forward networks. The following theorems of Steven Judd [36] [37] and, based on them, the work of Jyh-Han Lin and Jeffrey Scott Vitter [47], show that learning as well as the problem of topology determination are NP-complete in general. This is one motivation for using an EA as a heuristic search procedure for good architectures [51, 52].

A.2.1 Computational Complexity Theory

A basic concept in complexity theory is the term NP-completeness [19]. It denotes decision problems that can be solved by a non-deterministic Turing machine within polynomial time bounds. When considering the complexity of an problem the most important question is whether it belongs to the class P or NP.

Definition A.2.1 (Class NP, class P, decision problem)
Given a language L and a word x, then the task of deciding if $x \in L$ or $x \notin L$ is called a decision problem.

The class NP *comprises all decision problems that can be solved by a non-deterministic Turing machine within polynomial time.*

Problems that can be solved by a deterministic Turing machine within polynomial time belong to the class P.

Definition A.2.2 (Polynomial reducible, NP-complete)
Let $\sum_1$ and $\sum_2$ be alphabets. A language $L_1 \subseteq \sum_1^*$ is polynomial reducible *to another language $L_2 \subseteq \sum_2^*$ if a polynomial transformation $f : \sum_1^* \rightarrow \sum_2^*$ exists and the following holds:*

$$\forall x \in \sum\nolimits_1^* : x \in L_1 \Leftrightarrow f(x) \in L_2 \tag{A.1}$$

The notation $L_1 \leq_P L_2$ means that L_1 is polynomial reducible to L_2. A decision problem $L_2 \subseteq \sum_2^*$ is called NP-complete *if and only if $L_2 \in NP$ and $\forall L_1 \in NP$: $L_1 \leq_P L_2$.*

The NP-completeness of a problem was first proven by Cook for the satisfiability problem (SAT) [19]. With SAT a first representative of the class of NP-complete problems was found. To prove the NP-completeness of a new problem one has to reduce a known problem (i.e. SAT) to the new problem.

A.2.2 Loadability in Feed-Forward Networks

The work of Steven Judd [34] [35] [36] [37] has shown that learning in feed-forward networks is NP-complete in general. Judd does not use the term learning, instead he coins the new term *loadability*, which is more general in his context. In contrast to learning, loading a network does not only mean changing weights but assigning activation functions and weights to the nodes of the network.

This section will give some results concerning the loadability of mappings with respect to classes of activation functions, followed by complexity issues with respect to different network architectures.

First, Judd proposes a classification of activation functions in order to use this classes instead of single functions throughout his argumentation [34].

Definition A.2.3 (LSF, BF, QLF, LF)
Activation functions from the binary domain, that is f:$\mathbb{B}^n \to \mathbb{B}$, are divided into two classes.

- *Let LSF be the set of all linear threshold functions.*
- *Let BF be the set of all binary functions.*

Activation functions from the real domain $\mathbb{R}^n \to A$ can be divided into the following classes.

- *Let QLF be the set of all monotonous functions that are based on linear combinations of the inputs (quasi linear functions).*
- *Let LF be the set of all logistic functions. $f(x) = \frac{1}{1-e^{-x}}$*

It is necessary to extend the formal description of a network architecture if we want to use it along with Judds formalism.

Definition A.2.4 (Configuration)
A network architecture A is given as the 5-tupel $A = (U, H, I, O, DE)$ (see def. 2.1.9) . The activation functions are linear threshold units $f \in LSF$ and the input units are given by:

$$pred(u_i) = \{u_k | (u_i, u_k) \in DE\} \tag{A.2}$$

A network configuration *is defined as assignment of activation functions to each unit $K : H \to LSF$ such that $f_i = K(u_i)$ is the activation function of unit i.*

Definition A.2.5 (Loadable)
Let F be a feed-forward network with architecture A and $f : D \to \mathbb{R}^m$ a mapping with $D \subseteq \mathbb{R}^n$ that has to be learned. Let $F_A : \mathbb{R}^n \to \mathbb{R}^m$ be the function which is realized by the network F. Then the mapping f is loadable by F_A if a configuration K exists such that $\forall x \in D$ holds $F_A(x) = f(x)$.

Theorem A.2.1 (Loading with LF is NP-complete)
The task of finding a feed-forward network F_A with logistic activation functions from LF such that a given mapping $f : D \to \mathbb{R}^m$ is loadable by F_A is NP-complete.

With the proof of this theorem [36] Judd showed that the problem of finding a neural network architecture, with the common logistic activation function that realizes a given function is NP-complete in the general case [34] [36].

Restricting the activation function to subclasses does not help. The theorem also holds for the class of quasi linear functions QLF as well as for the binary functions BF. Even for the simple linear threshold functions LSF the loading problem is NP-complete. The theorem is rather general and independent of the training algorithm used. It shows that no training algorithm exists which can solve the learning task in P-time if $P \neq NP$. Note that it also holds for training NNs with evolutionary algorithms.

Now the question arises whether other restrictions might relax the problem. Researching constraints on the network architecture is a straight forward approach. With the aid of graphs classes of network architectures can be distinguished and for these classes it can be proven if the loading problem is in P or NP.

The following section will focus on the loading of mappings of the type $\mathbb{B}^n \to \mathbb{B}$ and less the 2^n mappings. As activation functions linear threshold functions $f \in LSF$ are used. To specify architecture classes we need some definitions.

Definition A.2.6 (Support cone of a unit)
Let $O := \{u_1, u_2, .., u_m\} \subset U$ be the set of output units of a feed-forward network, then the support cone *of a unit $u_i \in O$ can be defined as:*

$$sc(u_i) := \{u_i\} \cup \{sc(u_j) | u_j \in pred(u_i) \cap H\} \tag{A.3}$$

The support cone of a unit contains all predecessor units that potentially influence the output of that unit. Based on this definition architectures with certain restrictions on unit support cones can be defined.

Definition A.2.7 (Shallow architecture)
Let O be the set of output units then

$$F_p(u_i) := \{ \text{ activation function for } sc(u_i) | u_i \in O\} \tag{A.4}$$

defines the set of all activation functions in the support cone of the output unit u_i. *A feed-forward network has* a shallow architecture *if*

$$\exists c > 0 : \forall u_i \in O \text{ holds: } |F_p(u_i)| \leq c \tag{A.5}$$

A constant c is given as bound for the support cone of that unit.

The limitation of shallow architectures therefore has a bounding effect on the number of layers, that is the depth of the network, the number of predecessor units, called maximum fan-in, and the number of possible activation functions and weights of a network.

Theorem A.2.2 (Loading in shallow architectures is NP-complete)
Let F_A *be a feed-forward network with shallow architecture* A *and* $f : D \subseteq \mathbb{R}^n$ *the mapping to be loaded. The problem of loading* f *into* F_A *is NP-complete.*

The proof can be done by reduction of the NP-complete 3SAT problem to the loading problem [35], [36].

Definition A.2.8 (Support cone interaction graph)
Let $O := \{u_1, .., u_m\}$ *be the set of output units of a network. The graph* $G := (O, E)$ *with* $E := \{(u_i, u_j) : sc(u_i) \cap sc(u_j) \neq \emptyset\}$ *as edge set is called* support cone interaction graph.

Such a graph defines details about the interaction between the support cones of all output units and can be used to further restrict network architectures.

Theorem A.2.3 (Loading in #P)
Loading shallow architectures whose support cone interaction graphs are trees can be accomplished in #P.

The proofs of Judd [35], [36] make clear that the learning of mappings in feed-forward networks is NP-complete for most architectures in the general case. For some very limited classes of architectures it has been proven the learning can be accomplished in polynomial time.

Restrictions on architectures play a key role in theoretical research. Other important restrictions on the mappings that might be learned in #P by a neural network remain to be researched.

A.2.3 Complexity of Finding Optimal Topologies

Jyh-Han Lin and Jeffrey Scott Vitter [47] carry on the work of Judd. Their research is considered with the complexity of learning specific mappings and the complexity of finding an optimal architectures for specific problems. This section will recapitulate those results that show that the problem of finding an optimal architecture for a given mapping is NP-complete with respect to the size of the network.

We will consider networks with threshold functions from LSF and binary classifications $f : \mathbb{B}^n \rightarrow \mathbb{B}$ that have to be learned. First of all some formalism on the comparability of different networks and an optimality criterion shall be given.

Definition A.2.9 (Functional size of a network)
Let F be a network given by a 5-tuple according to definition 2.1.9, then the functional size of a network $|F|$ *is defined as the number of units $|H|$ that are not input units.*

We now consider networks with a functional size $k \geq 2$. The following theorems consider networks that have 2 or more hidden units.

Definition A.2.10 (Equivalent network, optimal network)
Two networks F_1 and F_2 are equivalent, *written as $F_1 = F_2$, if they compute the same set of functions.*

One can write $F_1 \subseteq F_2$ if all functions computed by F_1 are also computed by F_2.

A network F is said to be optimal *if, for all F_2 with $F_1 \subseteq F_2$, $|F_1| \leq |F_2|$ holds.*

According to this definition a network can be considered optimal if it has the smallest functional size to perform the computation of a given mapping. No other network exists with fewer non-input units that computes the same set of functions.

A simple decision problem is the question whether a network's output is different from 0.

Theorem A.2.4 (Decision of output $\neq$ zero)
The set of non-zero networks shall be given by $NZ := \{F | \exists x : F(x) \neq 0\}$. Then the decision problem $F \in NZ$ is NP-complete.

Proof A.2.1 (Sketch of proof)
The proof can be performed by reduction of 3SAT. Starting from a boolean expression B a network F_B can be constructed that realizes B. The boolean expression B is satisfiable if and only if $F \in NZ$. Due to the NP-completeness of 3SAT the problem $F \in NZ$ is also NP-complete.

Another theorem deals with the question whether two networks produce different output when given the same input.

Theorem A.2.5 (Decision of inequality)
The set of pairs of networks with different output behavior is given by $DO := \{(F_1, F_2) | \exists x : F_1(x) \neq F_2(x)\}$. The problem of deciding whether a network pair $(F_1, F_2) \in DO$ is NP-complete.

Proof A.2.2
The proof follows immediately from the fact that $F \in NZ$ *is a special case of* $(F_1, F_2) \in DO$. *We can construct* F_2 *as a network with output* 0 *for all inputs.*

It is an important question, given a network F, if an equivalent and optimal network F_{opt} exists.

Theorem A.2.6 (Optimal equivalent network)
Given a network F *with size* $k := |F|$ *and* $k \geq 2$. *The set of optimal equivalent networks is defined as* $OEN := \{(F_{opt}, k) | \forall x : F(x) = F_{opt}(x) \wedge |F_{opt}| \leq k\}$. *Then, the decision problem* $(F, k) \in OEN$ *is NP-complete.*

Proof A.2.3 (Sketch of proof)
The theorem can by proven by reduction of the output problem (A.2.4). Starting with a network $F_1 \in NZ$ *we can construct a network* F_2 *with additional input and output. Now* $F_2 \in NZ$ *if and only if* $(F_2, 0) \notin OEN$.

With these theorems it is shown that comparing the functionality of two given networks with more than one hidden unit is not feasible in the general case. The problem of finding an equivalent or optimal network is also NP-complete. We can conclude that no algorithm exists that, given a network, can find an optimal network with the same functionality in polynomial time.

A.3 Prediction of Critical Values in Chemical Engineering

A.3.1 Motivation

In chemical engineering the access to data for critical values for substances is of importance for the development of physical models for the calculation of thermodynamic properties (see chapter 5). Gaining such values by experiments is very costly, often inaccurate and because of thermodynamic instabilities very often not even feasible. There is a growing need for the development of suitable methods to correlate and to predict critical values like critical temperatures T_c, critical pressures p_c, and critical molar volumes v_c.

Two different kinds of methods are distinguished by Geyer [20]: Correlation methods and incremental methods. Both approaches achieve good results only through specialization on certain substances or certain critical values. They trade off accuracy against universality and suffer from specialization. A comparison of the best correlation and incremental methods yields the following results: Critical temperatures $T_c < 1\%$, critical pressures $1\% < p_c < 2\%$, and critical molar volumes $2\% < v_c < 6\%$.

In this section promising experiments motivate the usage of neural networks as an accurate and universal method for the predication of critical values. First, networks are used to learn *one* of the three critical values at a time, later these networks are compared to networks that had to learn all three critical values at a time. Three networks with varying size were explored for each task.

A.3.2 Data Preprocessing

In the figures A.5 through A.7 the degree of coverage for main groups used is shown. From a total of 403 different substance, 197 substances covered the critical temperature T_c, 141 substances covered the critical pressure p_c, and 116 substances covered the critical mol volume v_c. The different weighting is caused by the availability of experimental data. Main groups in the diagram without data do not matter during training because in the actually used group representation, such groups are subsumed under the CH_n group.

Networks trained with one critical value at a time receive all available data of the specific value while networks trained with all three values receive only data of 104 substances for which all values are known. As in the previous section the data was divided into a training set (50%), a validation set (25%), and a test set (25%). Table A.3 gives an overview of the partitioning of the data with respect to the critical value.

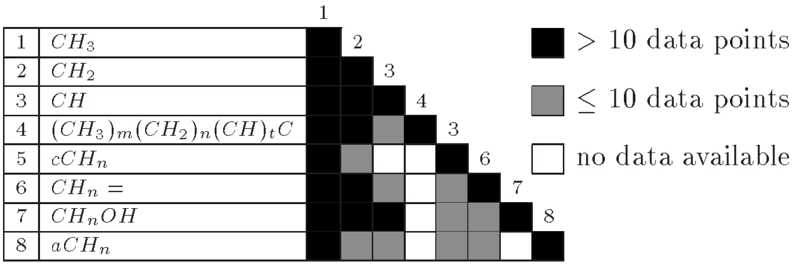


Figure A.5: Coverage of main groups for the 197 T_c-data points.

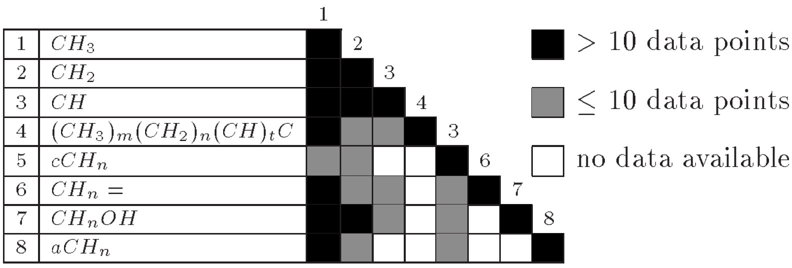


Figure A.6: Coverage of main groups for the 141 p_c-data points.

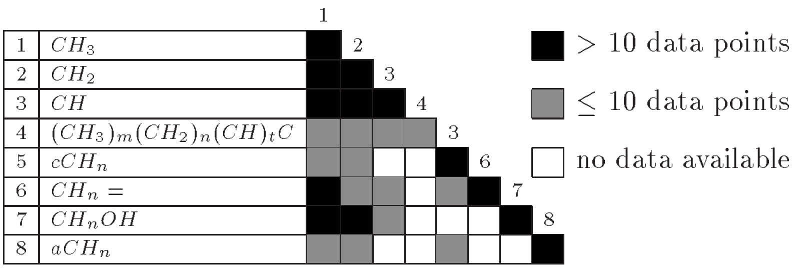


Figure A.7: Coverage of main groups for the 116 v_c-data points.

critical value	training	test+validation	total
T_c	99	98	197
p_c	71	70	141
$V_{m,c}$	58	58	116
$T_c, p_c, V_{m,c}$	52	52	104

Table A.3: Partitioning of data for training and generalization with respect to the critical value.

A.3.3 Networks Architectures and Experiment Design

One set of experiments was designed to explore the capabilities of neural networks to function as a model for the prediction of a single critical value (figure A.8). For each of the three critical values, three different network architectures with a varying number of hidden units (6, 10, 20) were explored. The second set of experiments used networks that were designed to learn all three critical values at a time (figure A.9). Again, three networks with varying number of hidden units (6, 10, 20) were explored for each task. As training algorithm backpropagation with a learning rate $\eta = 0.2$ and a momentum term $\alpha = 0.2$ was used and given a fixed number of $200,000$ learning epochs. Each network architecture was trained 10 times and the network with the smallest generalization error was chosen for comparison.

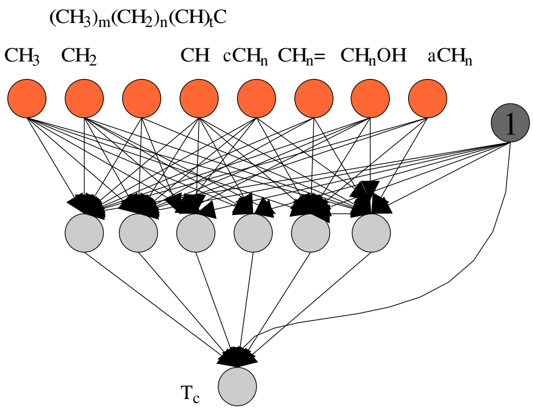


Figure A.8: Architecture of the 8-6-1 network for one critical value (T_c).

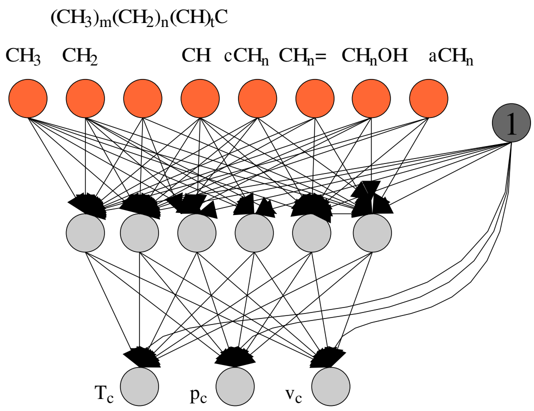


Figure A.9: Architecture of the 8-6-3 network for all critical values (T_c, p_c, v_c).

A.3.4 Results

Table A.4 presents the mean relative error (Eq. 5.2) of the training and validation+test[1] set for the networks trained with one critical value. All networks exhibit approximately the same errors regardless of the number of hidden units used. The smallest network with 6 hidden units seems to be sufficient for this problem. The error on the validation+test set is slightly higher but does not vary much with the number of hidden units. Only networks with 20 hidden units show an significant over-fitting effect due to the large number of weights available.

With a mean relative error of approximately 1.4% (T_c), 3.0% (p_c), and 3.4% (v_c) on data sets that have not been used for training the results are promising. A direct comparison with classical methods would be miss-leading because of their specialization on certain substance and is therefore omitted here.

[1]This is the joint set of validation and test set.

critical value	network	training MRF/%	validation+test MRF/%	all MRF/%
T_c	(8-6-1)	1.19	1.42	1.31
	(8-10-1)	1.15	1.42	1.29
	(8-20-1)	1.20	1.51	1.36
p_c	(8-6-1)	1.94	2.90	2.42
	(8-10-1)	2.13	3.03	2.58
	(8-20-1)	2.15	3.15	2.65
v_c	(8-6-1)	2.10	3.38	2.74
	(8-10-1)	1.96	3.35	2.66
	(8-20-1)	2.06	3.51	2.79

Table A.4: Results of neural networks with one output.

Table A.5 presents the mean relative error (Eq. 5.2) of the training and validation+test set for the networks trained with all three critical values. The critical value under consideration is underlined in the table. The two larger networks with 10 and 20 hidden units exhibit a 20% - 30% less training error for T_c than the previous networks but their validation+test set error is 40% - 60% larger. There is an overall increase in error for this two networks of 10% - 20%.

critical value	network	training MRE/%	validation+test MRE/%	all MRE/%
$\underline{\mathbf{T_c}}, p_c, v_c$	(8-6-3)	1.16	2.96	2.06
	(8-10-3)	0.90	2.23	1.57
	(8-20-3)	0.82	2.12	1.47
$T_c, \underline{\mathbf{p_c}}, v_c$	(8-6-3)	0.01	2.58	1.30
	(8-10-3)	0.01	2.62	1.32
	(8-20-3)	0.01	3.08	1.55
$T_c, p_c, \underline{\mathbf{v_c}}$	(8-6-3)	1.42	2.91	2.17
	(8-10-3)	1.19	2.98	2.09
	(8-20-3)	1.10	3.15	2.13

Table A.5: Results of neural networks with three outputs.

This effect can be seen with the other critical values p_c and v_c as well. The most dramatic over-fitting can be observed with p_c where the training error is two orders of magnitude smaller than the generalization error. Despite this immense difference the generalization error is still slightly better than that of the previous networks with only a single critical value. The over-fitting effect can be seen in

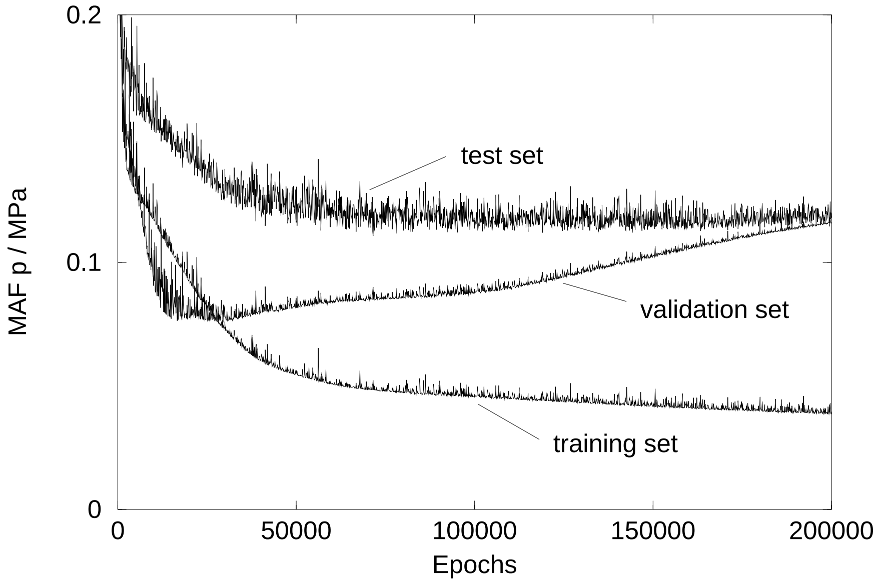


Figure A.10: Overtraining of the p_c data (8-20-3 network).

figure A.10 where the error (Eq. 2.6) on the critical pressure p_c over 200,000 learning epochs is given. After about 11,000 epochs the validation error starts to increase and the training was stopped to achieve a good generalization. The same network was also trained with all available data to incorporate the maximum possible information. The training time for this run was increased to 1,000,000 epochs. Although no statement about generalization can be made for such a network, the results might indicate how well a network could perform if there were enough data.

Table A.6 compares all neural networks with an incremental method. The first row gives the errors of the 8-20-1 network from table A.4, the second row gives the errors of the 8-20-3 network from table A.5, the third row shows the errors of the same network with early-stopping, the fourth row depicts the errors of a network trained with all available data and prolonged training time. The early-stopping method results in 2 to 5 times larger errors for all values. With mean relative errors of approximately 0.6% (T_c), 1.2% (p_c), and 1.2% (v_c) the networks trained with all data perform best of all methods[2].

In figure A.11 the results of all methods are compared and we can draw some conclusions: If we leave aside the generalization capabilities of the methods and focus only on their overall performance, measured by the mean error of training, validation and test data, then the clear winner is the neural network approach. Compared to the incremental method a network trained on half of the available

[2]Only the error of the critical pressure p_c is much larger indicating an ill partitioning of the data with respect to this value.

method	training set MRE / %			validation+test set MRE / %			all MRE / %		
	T_c	p_c	v_c	T_c	p_c	v_c	T_c	p_c	v_c
$\mathrm{NN}_{\mathrm{train(1)}}$	1.20	2.15	2.06	1.51	3.15	3.51	1.36	2.65	2.79
$\mathrm{NN}_{\mathrm{train(3)}}$	0.82	0.01	1.10	2.12	3.08	3.15	1.47	1.55	2.13
$\mathrm{NN}_{\mathrm{stopped(3)}}$	3.32	5.72	6.02	4.14	6.50	7.02	3.73	6.11	6.52
$\mathrm{NN}_{\mathrm{all(3)}}$	0.62	1.69	1.33	0.57	1.15	1.17	0.60	1.42	1.25
Inc. method (3)	1.49	14.28	4.37	1.30	12.01	2.99	1.40	13.5	3.68

Table A.6: Comparison of neural networks with an incremental method.

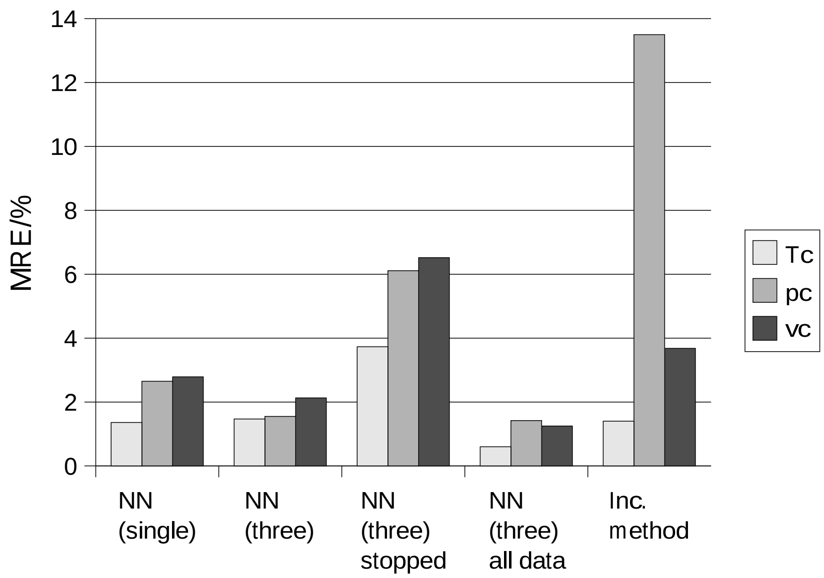


Figure A.11: Comparison of all methods on the critical values T_c, p_c, v_c.

data is only slightly worse on the critical temperature T_c, more then 8 times better on the critical pressures p_c, and 1.7 times better on the critical volumes. The same network with early-stopping is 2 to 3 times worse on T_c and v_c and 2 times better on p_c. This allows the conclusion that early-stopping does not always help to improve the overall performance. If a network has all data available it outperforms all other methods.

With a mean relative training error of 1% critical values can be predicted with 2-3% uncertainty. Form the viewpoint of chemical engineering this is a very promising result. Given the molecular structure and the surface fractions of the functional groups, a single model is sufficient to predict all three critical values.

A.4 Experiments in Chemical Engineering

This section gives subsidiary results and further illustrating figures on the experiments in chemical engineering from chapter 5.

A.4.1 Errors on Different Datasets

Figure A.12 shows the sorted errors of the backpropagation-trained network (NN-A-BP) on all data sets in case of the 3MG problem. Except for the critical regions close to $\Delta H_v(T_{cr}) = 0$ J/mol, the network comes very close to the desired values. In figure A.13 we see the errors for the same network trained with an evolution strategy. The errors are slightly higher for all three data sets and increase drastically near the critical regions.

A.4.2 Adaptation Methods for Physical Models

A closer look at the performance of the sequential versus the simultaneous adaptation of the physical model (figures A.14 and A.15) reveals that the sequential method outperforms the other on both problem sizes.

The superiority of the sequential fitting procedure can be explained by the negative influence of increasing numbers of parameters for the optimization process. A number of 27 simultaneously fitted parameters in all makes the adaptation of the strategic variables used by encapsulated evolution strategies more difficult in contrast to sequential fitting procedures, which result in smaller dimensions. With the increased problem size from 3MG to 5MG the simultaneous adaptation methods is 2 - 3 times worse than the sequential one and the generalization performance is even worse. The neural network instead is not sensitive to an increase in the number of free parameters.

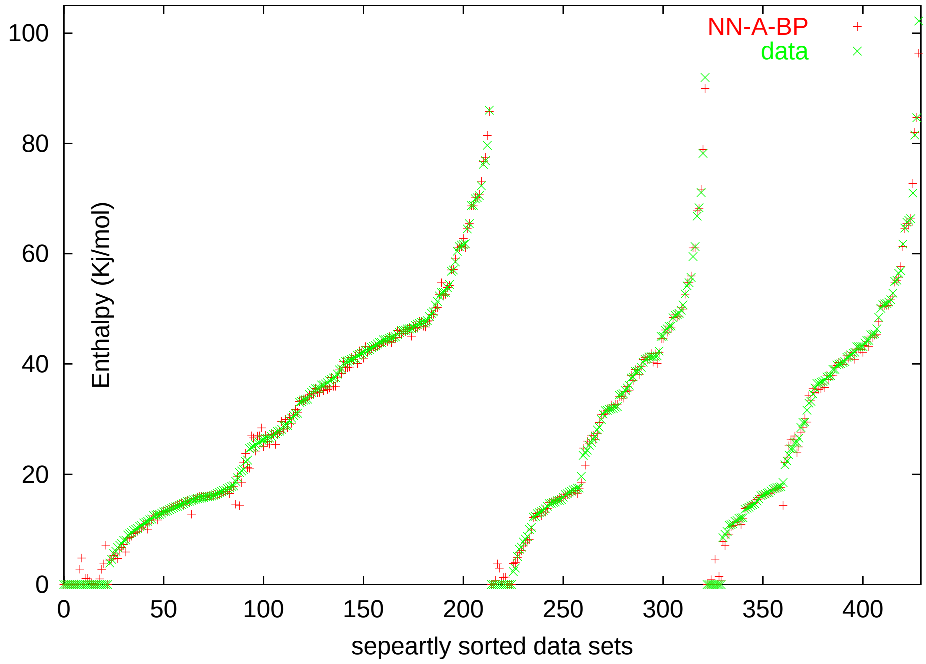


Figure A.12: 3MG: NN-A-BP (backpropagation) errors on training, validation, and test set.

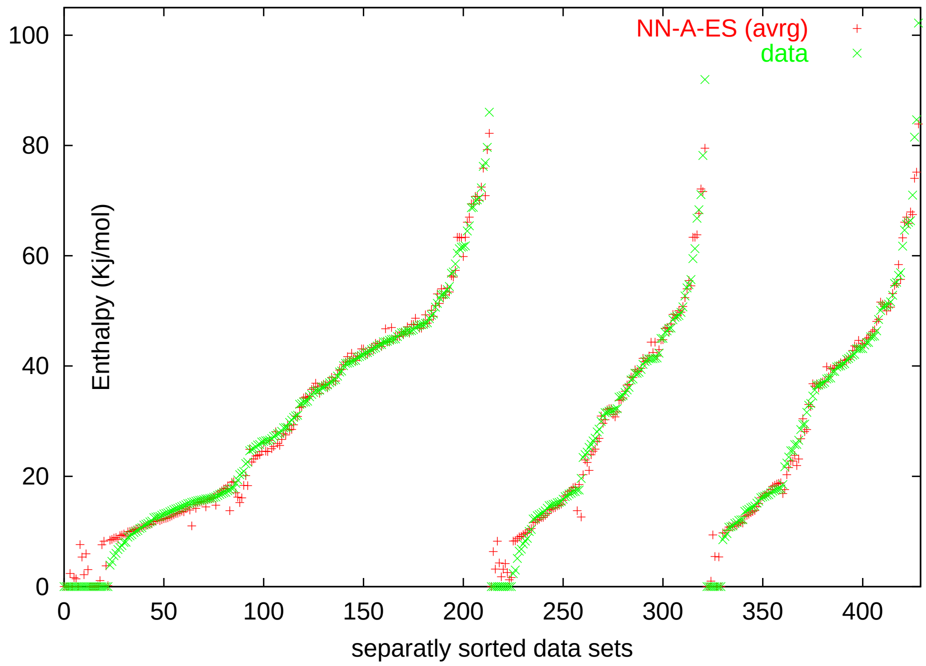


Figure A.13: 3MG: NN-A-ES (evolution strategy) errors on training, validation, and test set.

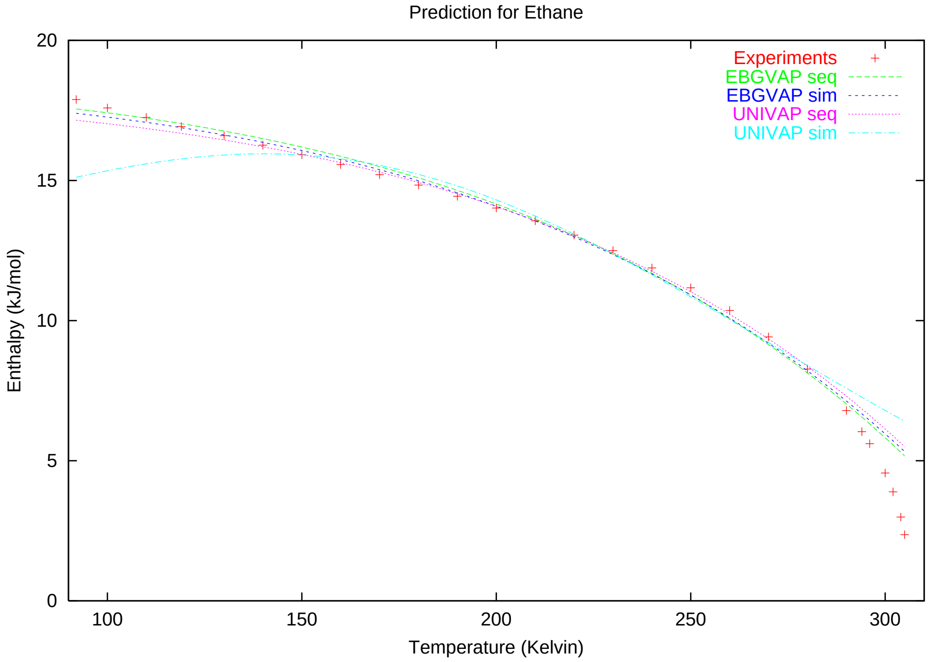


Figure A.14: 3MG: Physical models predicting the enthalpy of ethane over the whole temperature range. Comparison of sequential versus simultaneous adaptation.

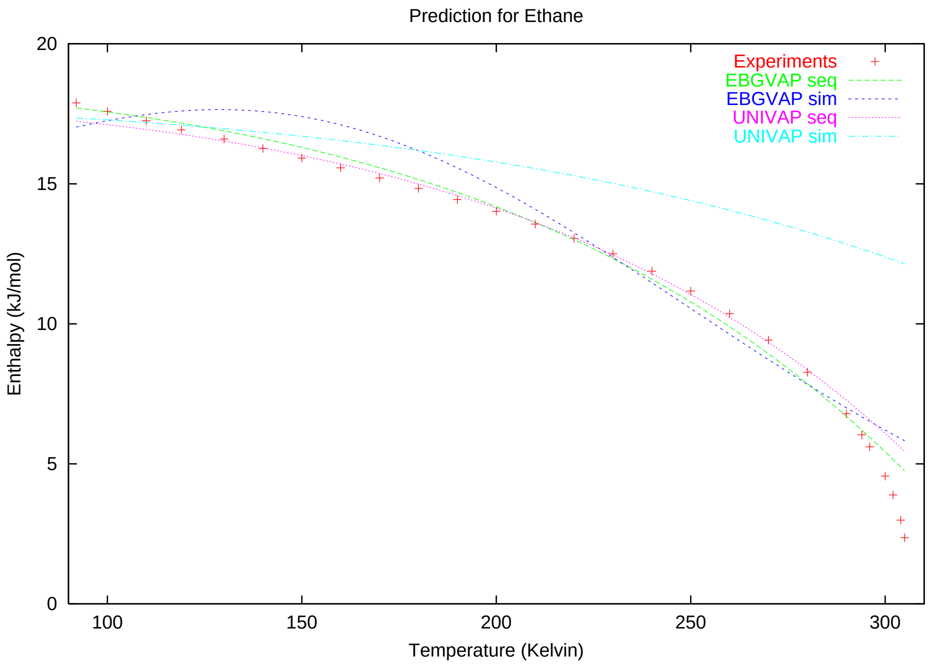


Figure A.15: 5MG: Physical models predicting the enthalpy of ethane over the whole temperature range. Comparison of sequential versus simultaneous adaptation.

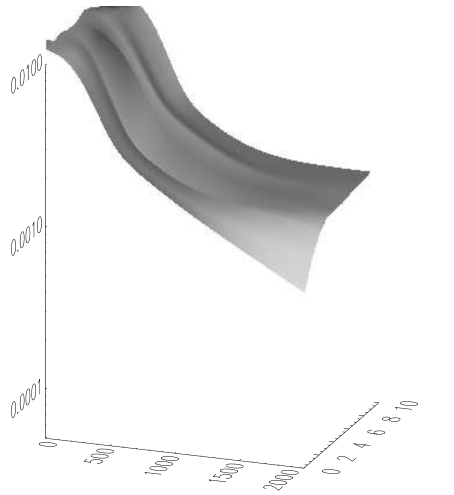
A.4.3 Variation of the Learning Rate

The following figures were gained by a limited parameter study. The learning rate of the network was studied over a wide range to find out which learning rates can be considered as a starting point for good network learning. The architecture of the network was fixed at 4-4-1 to have approximately the same number of free parameters ($5 \cdot 4 + 5 = 25$) as the physical methods for the 3MG data set and fixed at 6-9-1 ($6 \cdot 9 + 10 = 73$) for the 5MG data set. Because the error curves of the training and validation set are very similar only figures and interpretations for the 3MG data set are given.

The learning rate η was chosen from the set $\{0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,$-$0.7, 0.8, 0.9, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0\}$, the momentum term α was fixed at 0.2. Experiments started with a very low learning rate ($\eta = 0.001$) and ended with a far to high rate of ($\eta = 10.0$). Half of the 21 learning rates where chosen from the "normal" interval of learning rates ($[0.1, \ldots, 1.0]$).

A training run was stopped after it reached an error limit ($tss \leq 5 \times 10^{-5}$) or exceeded a maximum number of 100,000 epochs in case of the 3MG data and because of the larger data set 200,000 epochs in case of the 5MG data set.

The following figures show the msse error of the networks (y-axis) over the training epochs (x-axis) for all 10 runs (z-axis). Only every 50th epoch is shown. The lefthand-side figure always gives the error on the training set, whereas the righthand-side figure shows the error on the validation set.



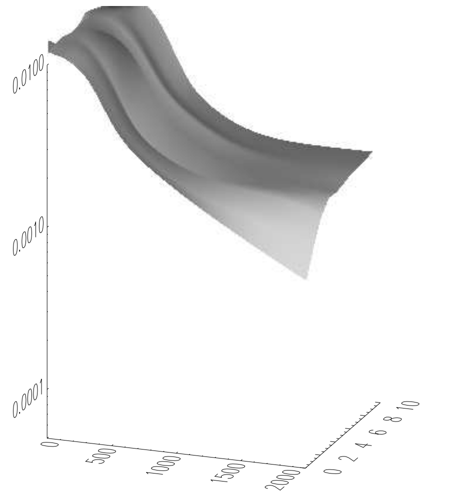
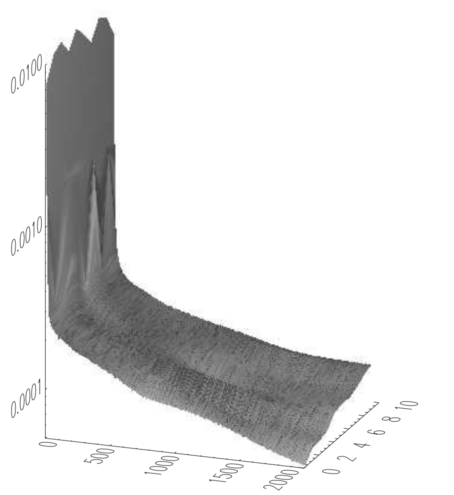


Figure A.16: Training error (η=0.001) **Figure A.17:** Validation error (η=0.001)

Too small learning rate: The surfaces of A.16 and A.17 are very smooth, due to the smoothness of each single run. The learning progress is very slow but steady. None of the training runs reached the specified error-limit within the given time. The error curves of training and validation sets are very similar and no over-training is observed.



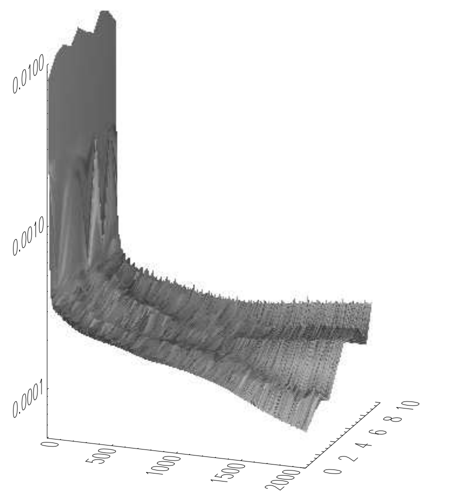


Figure A.18: Training error (η=0.2) **Figure A.19:** Validation error (η=0.2)

Moderate small learning rate: The surfaces of A.18 and A.19 are smooth. The learning progress is slow but steady. Only 1 of 10 runs reached the specified error-limit within the given time. In 2 of 10 runs we observe a very small increase in validation error.

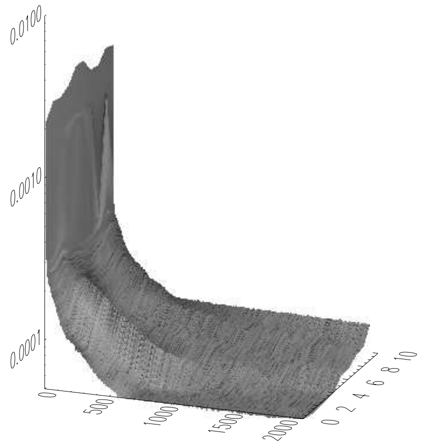


Figure A.20: Training error (η=0.7)

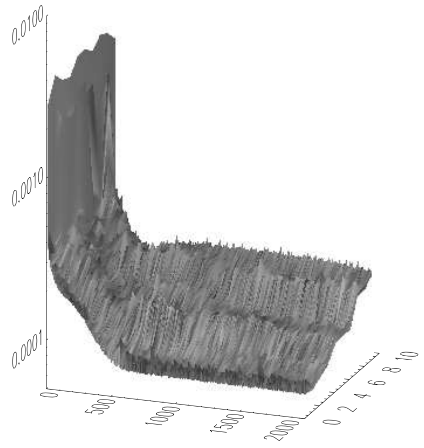


Figure A.21: Validation error (η=0.7)

Good learning rate: The surfaces of A.20 and A.21 are smooth with some small perturbations in the validation set. The learning progress is good and steady. Here 4 of 10 runs reached the specified error-limit within the given time. In 1 of 10 runs we observe a very small increase in validation error.

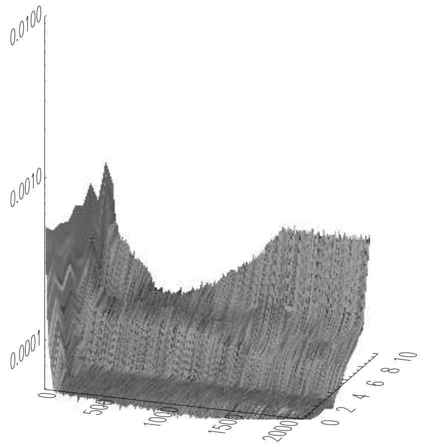


Figure A.22: Training error (η=8.0)

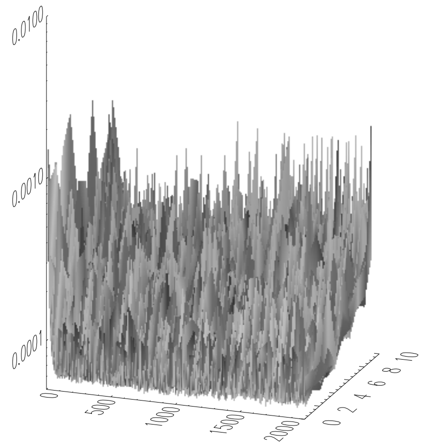


Figure A.23: Validation error (η=8.0)

Far to high learning rate: The surface of the training errors A.22 is rough. Two runs reached the error limit very fast but most runs got stuck at a high error level. The surface of the validation errors A.23 is covered with high peaks, resulting from large oscillations during training.

A.4.4 Variation of the Number of Hidden Units

After variation of the learning rate the best rate was used as a as constant for the hidden unit search. The number of units was varied between 1 and 40. Networks with too few units, less than 3, were not able to solve the task while networks with too many units did not gain anything from the additional units but suffer from prolonged training time because of their size. The performance of networks with more than 6 hidden units did not improve any more.

The following figures show the msse error (Eq. 2.6) of the networks (y-axis) over the training epochs (x-axis) for all 10 runs (z-axis). The lefthand-side figure always gives the error on the training set, whereas the righthand-side figure shows the error on the validation set.

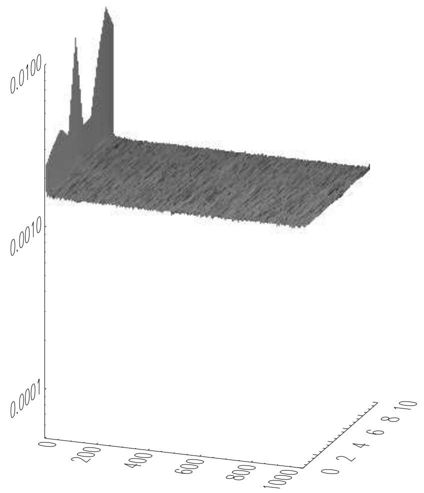


Figure A.24: Training error (U=1)

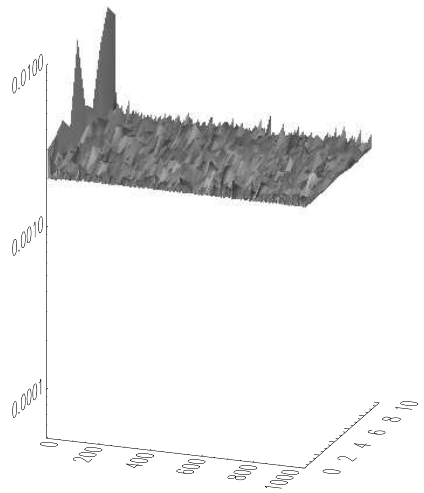


Figure A.25: Validation error (U=1)

Too few units: The two flat surfaces A.24 and A.25 show that there is no learning progress. All runs got stuck at a high error level. Networks with only 1 hidden unit could not solve the problem.

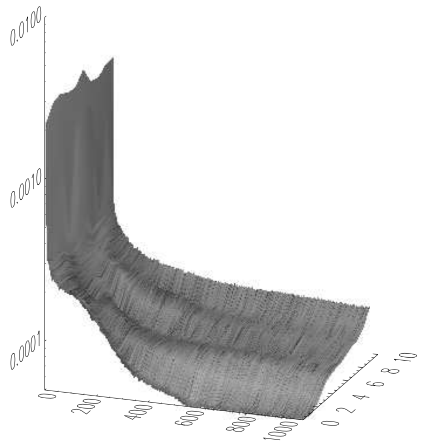


Figure A.26: Training error (U=4)

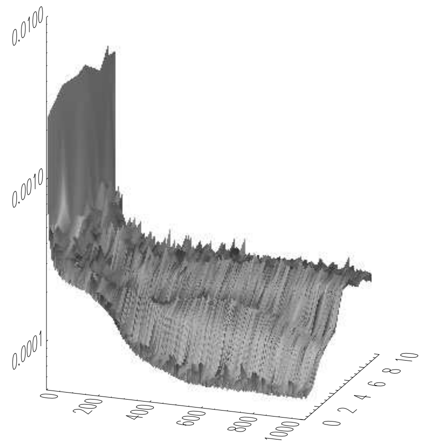


Figure A.27: Validation error (U=4)

Sufficient number of units: The surfaces of A.26 and A.27 are smooth with some bumps in the validation set. The learning progress is good and steady. Here 2 of 10 runs reached the specified error-limit within the given time.

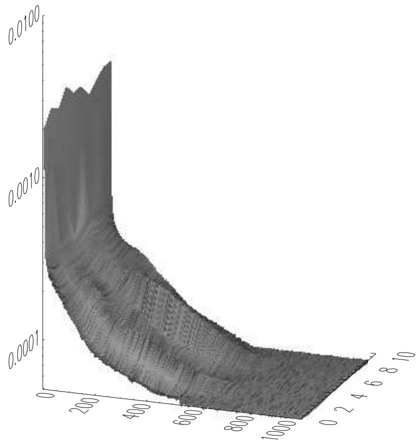


Figure A.28: Training error (U=6)

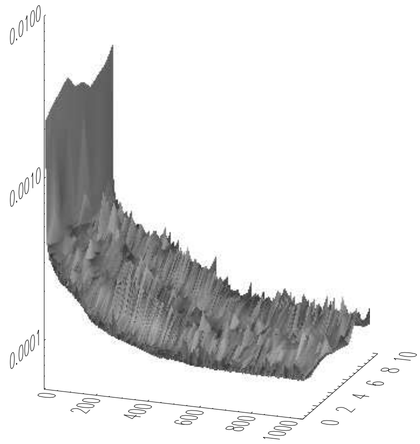


Figure A.29: Validation error (U=6)

Best number of units: The surfaces of A.28 and A.29 are smooth with some bumps in the validation set. The learning progress is good and steady. Here 10 of 10 runs reached the specified error-limit within the given time. None of the runs showed on over-training.

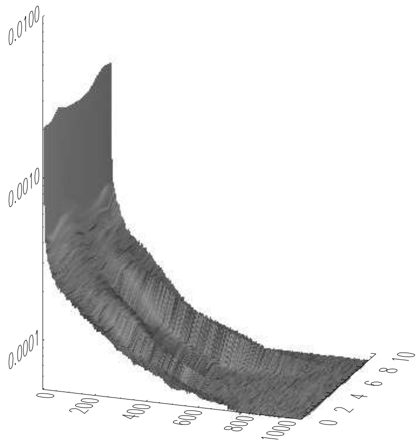


Figure A.30: Training error (U=40)

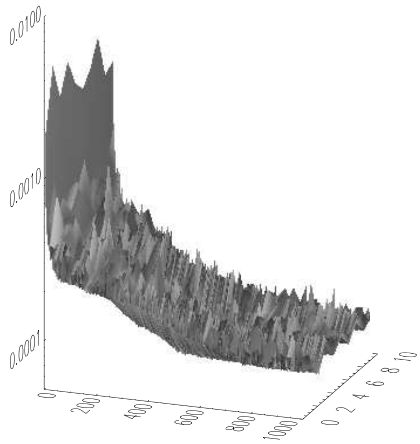


Figure A.31: Validation error (U=40)

Too many units: The surfaces of A.30 is smooth but the surface A.31 of the validation error shows larger error changes throughout learning. The learning progress is good and 10 of 10 runs reached the specified error-limit within the given time. None of the runs showed on over-training.

A.5 ES Parameter Studies on NN Training

This section presents results mentioned in earlier chapters. These parameter studies were made because the setting proposed in the literature [2] (local discrete recombination of object variables and global intermediate recombination of strategy parameters) for the ES did not deliver useful results.

A.5.1 Chemical Engineering: Recombination and Strategy Parameters

The initial motivation of this parameter study was to improve the quality of the neural network as a model for the prediction of chemical properties, and if successful, replace the backpropagation algorithm for training by an evolutionary approach.

This section will present experimental results for different recombination schemes of a (15,100)–ES on 25 parameters of the 3 main group problem. Every scheme was given 100,000 generations (10,000,000 pattern presentations) and repeated 10 times to gain statistical validity. In almost every experiment none of the 10 runs showed promising results, so no more runs were performed.

All of the following variations were done with either one step size or n step sizes. For all runs local (sexual) recombination was used. The more successful runs with n step sizes were also done with global recombination.

The figures on the right-hand side give a typical example run of a recombination scheme. It shows the best (Best) and average (Avrg) individual network (weight vector) in the population and the smallest (SigMin) and largest (SigMax) step size σ in the population. The error (y-axis) and the number of generations (x-axis) are both on a logarithmic scale.

The figures on the left-hand summerize the final results for all 10 runs. The runs are sorted by the number of generations and the graph shows the final best individual, the smallest and largest step size in the population for each run. A run is considered successful if it reached an error comparable to that of backpropagation ($\approx 5 \times 10^{-5}$). The error (MSSE Eq. 2.6) of the best individual is aligned at the left y-axis and values for the step sizes are aligned at the right y-axis. Both are on a logarithmic scale and often differ by several orders of magnitude.

The table below the figures allows conclusions about the diversity of the final results for all 10 runs. It gives the mean (mean), the minimum (best), the maximum (worst) and the standard deviation (sd) of the best and average individual at the end of each run. Additionally, the difference between the best and average individual can be seen as an indicator for the diversity of the population. A small difference between the best and average individual indicates a collapsed population and a stagnation, either premature or close to the optimum. In some cases the step sizes allow conclusions about whether progress is still likely or premature stagnation happened. If the maximum step size in a population is above a certain very large value then at least one unit of the network is saturated and not functional anymore. If the maximum step size is too small then the probability for significant weight changes becomes too small as well. In both cases it is likely that the strategy fails to find the optimum.

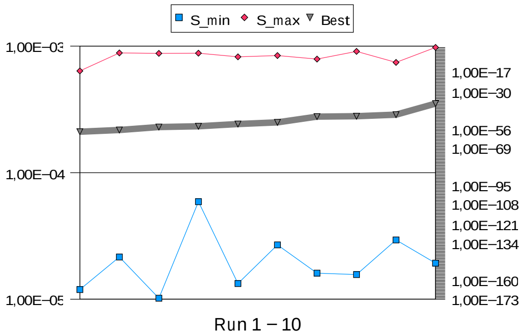
The following table A.7 allows a direct comparison with backpropagation learning.

	best of 10	worst of 10	mean	sd	suc. rate
Backpropagation	2.20e-05	6.10e-05	3.93e-05	1.68e-05	8:10
Evolution Strategy	2.40e-05	1.70e-04	6.65e-05	4.14e-05	3:10

Table A.7: 3MG: Final results for 10 backpropagation (250,000 function calls) and 10 ES runs (10,000,000 function calls).

Recombination with one and n Step Sizes

The figures throughout this section always show the slightly better local recombination scheme with n step sizes, while the results for one step size and the results for global recombination are only summarized in the tables below the figures.



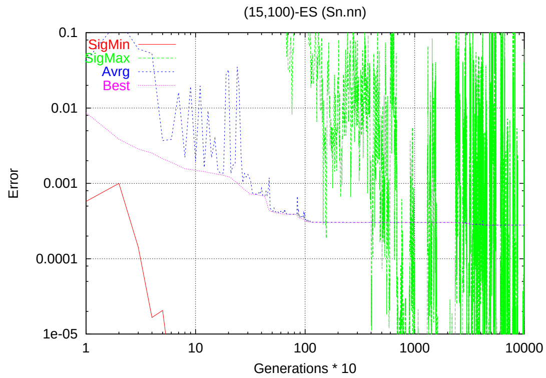


Figure A.32: (Left) all sorted runs of an (15,100)–ES without recombination. Depicted are the final best individual, the smallest and largest step size in the population for each run. (Right) A typical example run with the best (Best) and average (Avrg) individual network (weight vector) in the population and the smallest (SigMin) and largest (SigMax) step size σ in the population.

$\#\sigma$	individual	best	worst	mean	sd
1	best	0.000209	0.000275	0.000237	1.78e-05
	average	0.000209	0.000275	0.000237	1.78e-05
n	best	0.000211	0.000352	0.000258	4.24e-05
	average	0.000211	0.000352	0.000258	4.24e-05

Table A.8: Results for an ES without recombination. The success rates for all variations are 0:10.

No recombination: All runs without recombination got stuck at a high error level. In most runs the smallest and largest step size in the population went below 10^{-5} quite early (after 10%-20% of the total generation time). This can be seen as an indicator for the fact that no more or at most only very small progress can be made because of the too small weight changes[3]. The fitness distance to the average individual is zero in all runs, which indicates a collapsed population.

[3]See section A.5.4 for a further discussion of limiting and forcing of weight changes.

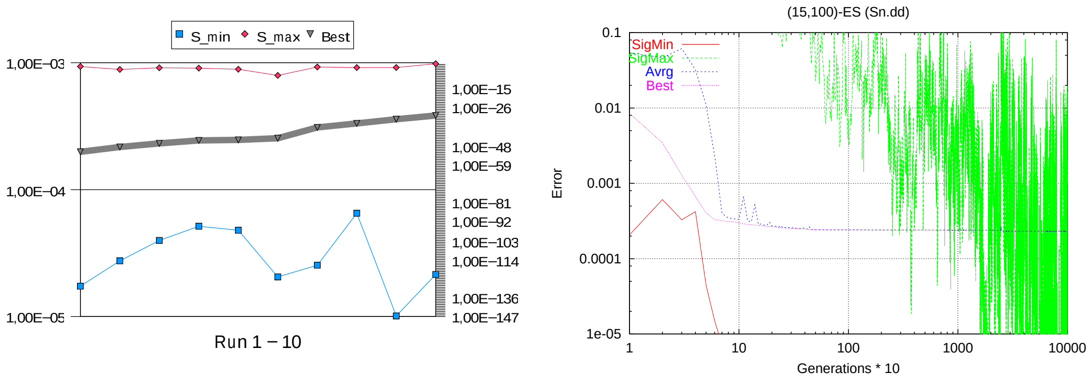


Figure A.33: (Left) all runs of an (15,100)–ES ("dd" recombination) and an example run (right).

$\#\sigma$	recombination	individual	best	worst	mean	sd
1	local	best	0.000260	0.001432	0.000398	0.000364
		average	0.000260	0.001432	0.000398	0.000364
n	local	best	0.000199	0.000384	0.000278	6.373e-05
		average	0.000199	0.000384	0.000278	6.373e-05
n	global	best	0.000213	0.000317	0.000250	2.915e-05
		average	0.000213	0.000317	0.000250	2.915e-05

Table A.9: Results for an ES with "dd" recombination. The success rates for all variations are 0:10.

Discrete recombination of weights and discrete recombination of step size: Same behavior as without recombination. All runs got stuck at a high error level. In most runs the smallest and largest step size in the population went below 10^{-5} quite early (after 10% - 20% of the total generation time). The fitness distance to the average individual is 0 - 3×10^{-7} indicating a collapsed population. There is no significant performance difference between local and global recombination.

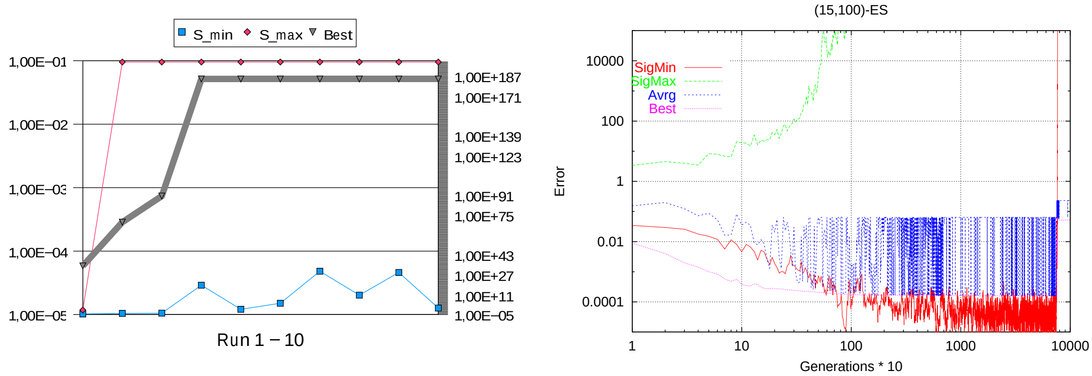


Figure A.34: (Left) all runs of an (15,100)–ES ("di" recombination) and an example run (right).

$\#\sigma$	recombination	individual	best	worst	mean	sd
1	local	best	8.80e-05	0.000171	0.000130	2.85e-05
		average	8.80e-05	0.000171	0.000130	2.85e-05
n	local	best	5.90e-05	0.051730	0.036319	0.024814
		average	5.90e-05	0.232913	0.186416	0.098024
n	global	best	0.23291	0.232913	0.232913	0.0
		average	0.23291	0.232913	0.232913	0.0
n	local/global	best	0.00143	0.232913	0.060495	0.106426
		average	0.00148	0.232913	0.060509	0.106418

Table A.10: Results for an ES with "di" recombination. The success rates for all variations are 0:10.

Discrete recombination of weights and intermediate recombination of step size: Most runs got stuck at a very high error level. In 7 out of 10 runs the smallest and largest step size in the population drift apart by more than 100 orders of magnitude. With the local recombination scheme, the smallest step sizes in the populations went above 10^5 and the error got stuck at a fixed value in most runs. The global recombination worsens these problems. After 10% of the total number of generations the smallest step sizes went above 10^{200} for all runs. With only one step size the results are better but all runs got stuck at a similarly high error level.

Additional 10 runs with local recombination on the object variables and global recombination of the step sizes were not successful.

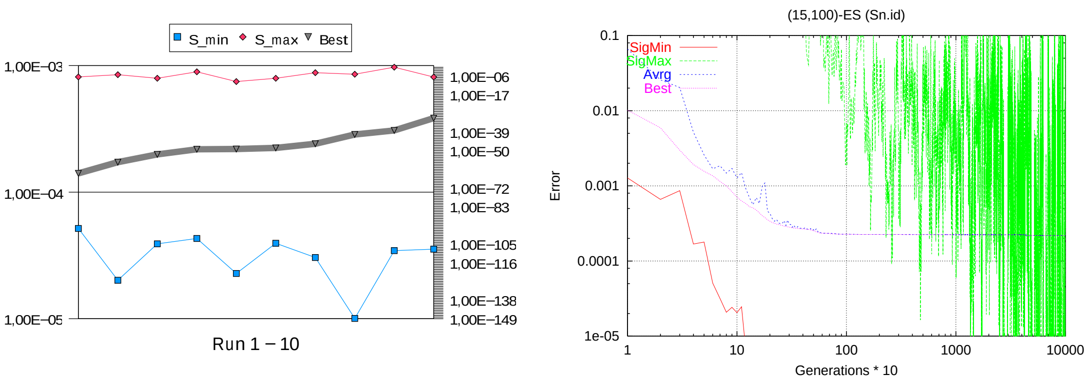


Figure A.35: (Left) all runs of (15,100)–ES ("id" recombination) and an example run (right).

$\#\sigma$	recombination	individual	best	worst	mean	sd
1	local	best	0.000277	0.001425	0.000626	0.000547
		average	0.000277	0.001425	0.000626	0.000547
n	local	best	0.000141	0.000385	0.000239	7.06e-05
		average	0.000141	0.000385	0.000239	7.06e-05
n	global	best	0.000178	0.000357	0.000237	5.31e-05
		average	0.000178	0.000357	0.000237	5.31e-05

Table A.11: Results for an ES with "id" recombination. The success rates for all variations are 0:10.

Intermediate recombination of weights and discrete recombination of step size: This recombination scheme delivers almost the same results and runs exhibit the same behavior as with "dd" recombination. All runs got stuck at a high error level. In most runs the smallest and largest step size in the population went below 10^{-5} quite early (after 10% - 20% of the total generation time). Other runs exhibit a behavior where the step sizes vary by several orders of magnitude. The fitness distance to the average individual is 0 in all runs indicating a collapsed population. There is no significant performance difference between local and global recombination.

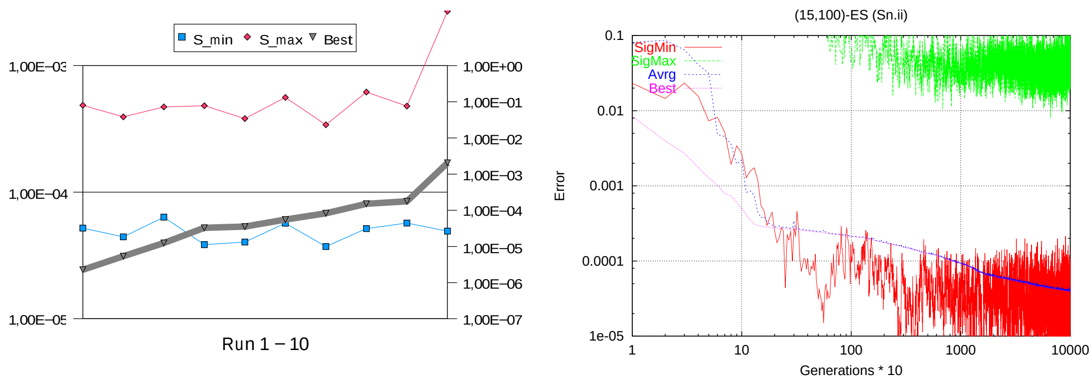


Figure A.36: (Left) all runs of (15,100)–ES ("ii" recombination) and an example run (right).

$\#\sigma$	reco.	indiv.	best	worst	mean	sd	s. rate
1	local	best	0.000186	0.000226	2.12e-04	1.18e-05	0:10
		average	0.000186	0.000226	2.12e-04	1.18e-05	
n	local	best	2.40e-05	0.000170	6.65e-05	4.14e-05	3:10
		average	2.50e-05	0.000171	6.70e-05	4.16e-05	
n	global	best	2.40e-05	0.000547	1.30e-04	1.52e-04	3:10
		average	2.40e-05	0.000547	1.30e-04	1.52e-04	
n	local/global	best	2.70e-05	0.000142	7.26e-05	3.45e-05	3:10
		average	2.70e-05	0.000142	7.26e-05	3.45e-05	
n	geometric	best	0.000252	0.000363	3.10e-04	4.09e-05	0:10
		average	0.000252	0.000363	3.10e-04	4.09e-05	

Table A.12: Results for an ES with "ii" and "ig" recombination. Given are the success rates and the errors.

Intermediate recombination of weights and intermediate/geometric recombination of step size: This recombination scheme delivers successful runs and results that are comparable to other methods. The step sizes are still within a "reasonable" range and there is also some variation in the population. The fitness distance to the average individual is $3 \cdot 10^{-7}$ - $1 \cdot 10^{-6}$ indicating that even further progress is possible.

With "ii" recombination one can observe a significant difference between global and local recombination. Both variations are able to find a good solution, but the local recombination delivers better results on average with a much smaller standard deviation. Local recombination on the object variables combined with global recombination on the strategy variables is comparable to local recombination on both parameter sets. None of the runs with geometric recombination of step sizes succeeded. Here, it seems as if all runs got stuck in a local optimum.

In the case of only one step size all runs got stuck at a higher error level with no difference between the average and the best individual indicating a collapsed population.

From the fact that none of the recombination schemes with only one step size was successful one can conclude that this problem needs more than a single step size.

A.5.2 Coupled versus Decoupled Strategy Parameters

In an ES the recombination of the strategy parameters can either be coupled to the object variables or decoupled. To figure out which variant performs better, the experiments with the most promising parameter setting (n step sizes and intermediate recombination of object variables and strategy parameters) were repeated with strategy parameters decoupled from the object variables. An offspring may now get its step sizes from other parents than it got the object variables from.

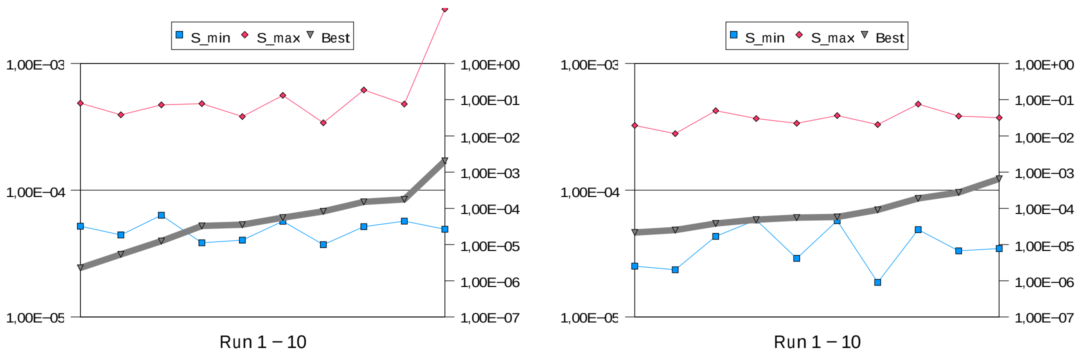


Figure A.37: All runs of (15,100)–ES (intermediate/intermediate recombination) decoupled (left) and coupled parameter sets (right).

Decoupled weights and step sizes: If we look at the mean individual, the results are similar to the experiment with coupled parameter sets, but on average, the best individual has almost twice the error (table A.14). The fitness distance to the average individual is 0 for all 10 runs which indicates collapsed populations.

The decoupling of strategy parameters and object variables during recombination seems to cause a loss of diversity and we see a slightly degraded performance in comparison with the coupled variant. This observation is confirmed by experiments on the 2-bit parity problem (table A.14).

This result might be plausible if we reconsider the application. In the context of neural networks training algorithms one can think of the strategy parameters

coupled object variables and strategy parameters					
individual	best	worst	mean	sd	suc. rate
best	2.4e-05	0.000170	6.65e-05	4.14e-05	3:10
average	2.5e-05	0.000171	6.70e-05	4.16e-05	
decoupled object variables and strategy parameters					
individual	best	worst	mean	sd	suc. rate
best	4.6e-05	0.000123	7.06e-05	2.43e-05	2:10
average	4.6e-05	0.000123	7.06e-05	2.43e-05	

Table A.13: Results for an ES with coupled and decoupled parameters.

as local learning rates for the weights. When training a network with backpropagation like algorithm such a local learning rate is semantically coupled to the corresponding weight. Ripping apart a weight and its associated learning rate does not seem to make much sense here.

chemical engineering					
	suc. rate	min	max	mean	sd
coupled	3:10	27,180	41,580	33,140	7,513
decoupled	2:10	46,190	85,250	65,720	-
2-bit parity (XOR)					
	suc. rate	min	max	mean	sd
coupled	48:50	3,000	74,700	19,068	21,738
decoupled	46:50	3,100	98,600	17,567	24,366

Table A.14: Comparison of coupled and decoupled evolution strategies with "ii" recombination. Given are the success rates and the number of function evaluations needed to reach the error limit.

To see whether a coupling of object and strategy parameters is beneficial in general, two classical ES test functions were used as well. The sphere model and the Rastrigin function:

$$\text{Spehre:} \qquad \sum_{i=1}^{n} x_i^2 \tag{A.6}$$

$$\text{Rastrigin:} \qquad 3 \cdot n + \sum_{i=1}^{n} (x_i^2 - 3 \cdot cos(2 \cdot \pi \cdot x_i)) \tag{A.7}$$

Again a (15,100)-ES with both "di" and "ii" recombination was run for 5000 generations. In figure A.38 we see the performance on the sphere model with coupled and decoupled parameters. Depicted are 50 averaged runs. For both recombination schemes, there is only a very small difference between coupled and decoupled recombination. The same can be observed for the Rastrigin function.

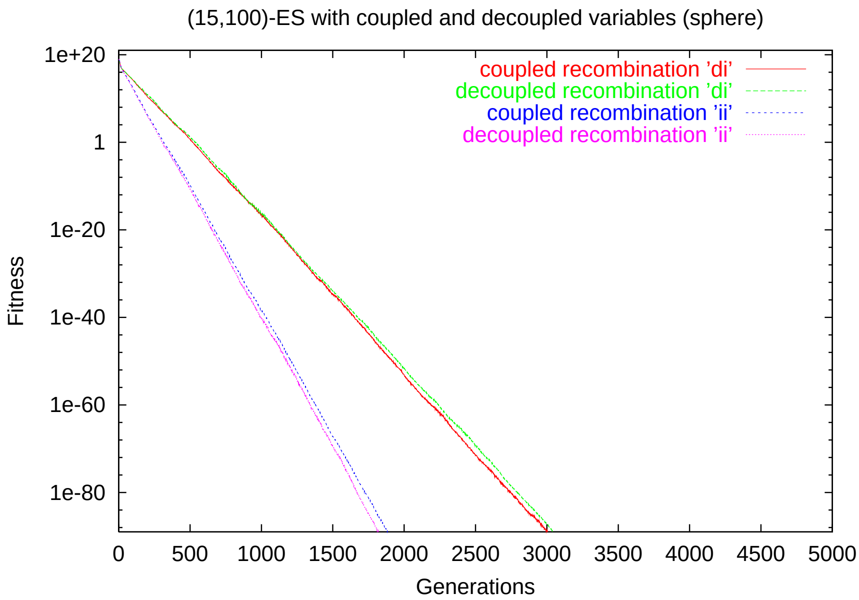


Figure A.38: Sphere model: 50 ES runs with coupled and decoupled object and strategy parameters. No significant difference can be observed.

A.5.3 Local versus Global Recombination

All experiments documented in section A.5.1 showed that local recombination always yields better results than global recombination. In most cases results are very similar, but in no case global recombination outperformed local one. This observation is confirmed by experiments on the 2-bit parity problem (section 4.3).

	reco.	suc. rate	min	max	mean	sd
ES (ii)	local	10:10	3,100	109,300	33,280	43,151
	global	10:10	3,800	59,200	14,030	17,099
ES (di)	local	10:10	2,300	71,100	17,740	26,649
	global	10:10	3,400	47,600	15,620	15,038
ES (dd)	local	8:10	2,500	60,500	11,750	19,818
	global	4:10	3,000	3,600	3,350	400
ES (id)	local	6:10	2,700	749,900	127,867	304,733
	global	4:10	57,800	833,400	591,450	363,407
ES (nn)	local	8:10	3,300	11,900	5,350	2,858
	global	7:10	3,600	8,800	5,714	2,099
ES (ig)	local	8:10	5,300	16,700	8,200	3,879
	global	8:10	5,000	8,600	6,000	1,355

Table A.15: XOR: Local versus global recombination. Given are the success rates and number of function evaluations needed to reach the error limit.

A.5.4 Bounded Weights and Step Sizes

One can observe that with growing step sizes in the population the number of generations needed to achieve a given learning task increases. Another observation is that most ES runs that failed ended with the largest and smallest step size in the population drifted far apart from each other.

When we take a closer look at the weight changes over time we can observe two effects. The most obvious one is that too large step sizes cause to large weights. Also, some weights get stuck at a certain value because of too small step sizes. A straight forward countermeasure is to impose bounds (constraints) on the step sizes or on the weights. Another measure that ensures weight changes is a *step size correction* that forces the step sizes to be at least a fraction of the weights values, such that $\sigma_i \geq \varepsilon \cdot |x_i| >$ S_min (see [75], p. 150).

Here, the following approaches were investigated on the two parity problems 2-bit (section 4.3.2) and 6-bit parity (section 4.3.3):

a) Upper and lower bounds on the step sizes.

b) Upper and lower bounds on weights and step sizes.

c) Step size correction with $\varepsilon = 0.05$.

a) Upper and lower bounds on the step sizes: Restricting the minimum (S_min) and maximum (S_max) allowed step size in the population does not help to increase the overall success rate. In fact, the opposite can be observed.

For the 2-bit parity problem, the step sizes were bounded to the range where 2/3 of the successful runs ended. The following two figures illustrate the difference between an ES with unbounded step sizes (figure A.39) and bounded step sizes (S_min $<$ 0.0001, S_max $<$ 10) (figure A.40). A larger upper bound of S_max $<$ 100 delivered the same results. The termination criterion was an error of 10^{-3} or 1000 generations (100,000 function evaluations).

In case of the bounded step sizes the success rate decreases from 48 to 42 (table A.16). A closer look reveals that some runs with relative large step sizes in the population can still be successful and that they might need such large step sizes to escape from local minima or get off the flat spots at the boundaries of the activation functions.

If the success rate is of importance then imposing "reasonable" upper and lower bounds on the step sizes should be avoided.

b) Upper and lower bounds on the weights and step sizes: When imposing bounds on the weights, several countermeasures can be taken to prevent weights from leaving the bounds. Three different bounding strategies are explored:

1. Reset weight and step size to their upper bounds (limitXS):
   $x_i := X\_max, \quad \sigma_i := S\_max$
2. Reset weight to upper bound and initialize step size (initS):
   $x_i := X\_max, \quad \sigma_i := rand(0, S\_max)$
3. Initialize weight and step size (initXS):
   $x_i := rand(X\_min, X\_max), \quad \sigma_i := rand(S\_min, S\_max)$

None of the bounding methods lead to a higher success rate than an unbounded ES. Among the bounding strategies the results are diverse. In case of the XOR problem (table A.16) limiting step sizes (limitS) and limiting weights as well (limitXS) performed best, while the latter can be clearly favored in case of the 6-bit parity problem (table A.17).

c) Step size correction: A step size correction was introduced as an additional measure to b). In case of the XOR problem (table A.16) it clearly improves the results of the bounding strategies, while it worsens them drastically for the 6-bit parity problem (table A.17). Again, the best results are those gained by an unbounded ES.

Restricting step sizes and weights to "reasonable" bounds or the coupling of step sizes to the weight with the sigma correction does not help to increase the overall success rate when training neural networks. In fact, the opposite could be observed for the two test problems.

method	suc. rate	min	max	mean	sd
normal	48:50	3,000	74,700	19,068	21,738
limitS	42:50	2,900	70,000	11,969	14,172
b) weights bound to $(-20, 20)$, step sizes bound to $(0.0001, 10)$					
limitXS	41:50	3,100	53,900	7,285	8,991
initS	34:50	3,300	12,900	5,135	2,097
initXS	38:50	3,200	57,400	8,255	10,754
c) step size correction of $\varepsilon = 0.05$ plus b)					
limitXS	43:50	3,400	78,800	13,241	17,121
initS	44:50	3,100	65,700	7,093	10,691
initXS	44:50	4,100	86,300	9,556	15,513

Table A.16: Parity-2 (XOR): ES-learning with upper and lower bounds and step size correction. Given are the success rates and the number of function evaluations.

method	suc. rate	min	max	mean	sd
normal	8:10	55,900	4,007,900	1,015,590	274,484
b) weights bound to $(-20, 20)$, step sizes bound to $(0.0001, 10)$					
limitXS	8:10	80,700	4,231,000	1,582,760	1,753,570
initS	7:10	53,000	1,378,800	339,260	581,655
initXS	8:10	48,600	3,434,000	642,450	1,159,780
c) sigma correction of $\varepsilon = 0.05$ plus b)					
limitXS	5:10	732,600	5,587,800	2,789,080	2,347,170
initS	7:10	957,700	4,247,800	2,501,940	1,334,090
initXS	1:10	3,081,600	3,081,600	3,081,600	0

Table A.17: Parity-6: ES-learning with upper and lower bounds and step size correction. Given are the success rates and the number of function evaluations.

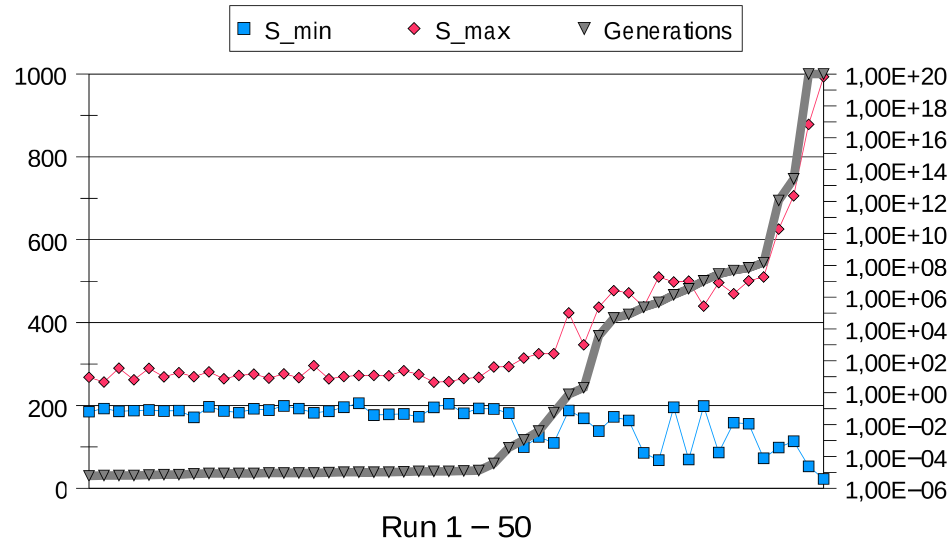


Figure A.39: XOR: 50 ES run with unbounded step sizes. Depicted are the number of generations needed, the minimum step size (S_min), and the maximum step size (S_max) in the population. The success rate is 48 of 50 runs.

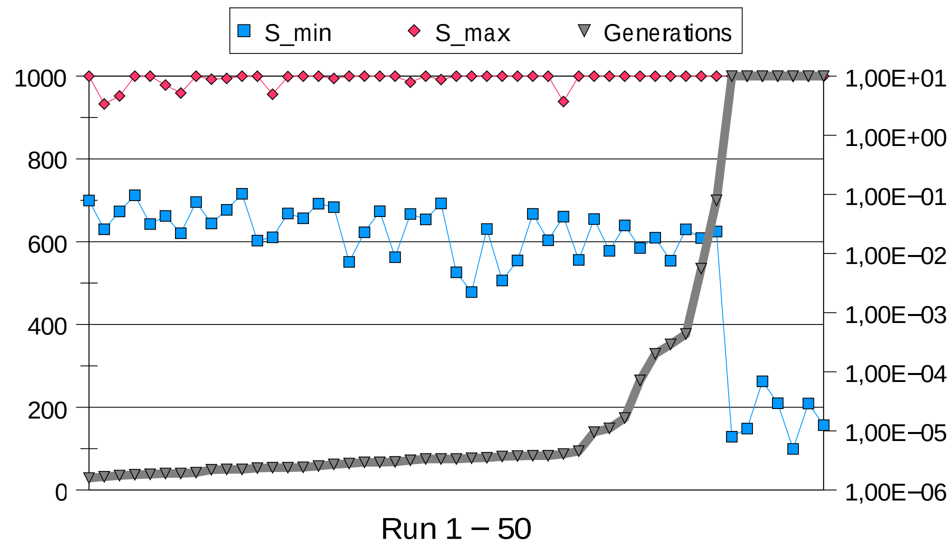


Figure A.40: XOR: 50 ES run with bounded step sizes (S_min < 0.0001, S_max < 10). Depicted are the number of generations needed, the minimum step size (S_min), and the maximum step size (S_max) in the population. The success rate is 42 of 50 runs.

A.5.5 Variations of the Overall Mutability

In the following experiment the overall mutability was increased from $\tau' = 1/\sqrt{(2n)}$ to $\tau' = 1/\sqrt{(n)}$. The influence of the parameter τ has not been researched in this parameter study, but the diverse results on the 2-bit and 6-bit parity problems suggest that τ is a relevant and application specific parameter and supports the findings of [43].

In case of the 2-bit parity problem the overall success rate as well as the number of generations needed to reach the error limit improves (table 4.3 and table A.18) with τ adjusted according to the common heuristic (section 3.1.2). The opposite can be observed in case of the 6-bit parity problem (table 4.4 and table A.19), where most recombination schemes fail to achieve the task with the suggested τ' and significant improvement can be made choosing a higher mutability of $\tau' = 1/\sqrt{(n)}$.

	suc. rate	min	max	mean	sd
ES (ii)	8:10	4,100	954,800	142,675	330,332
ES (di)	10:10	3,800	82,400	29,590	28,000
ES (dd)	5:10	3,200	4,900	3,900	872
ES (id)	4:10	3,700	5,000	4,375	585
ES (nn)	7:10	2,700	29,800	13,671	11,830
ES (ig)	6:10	4,000	10,300	8,100	2,194

Table A.18: Parity-2 (XOR): Success rate and number of function evaluations needed to reach the error limit. With τ' set to $1/\sqrt{(n)}$ instead of $1/\sqrt{(2n)}$.

	suc. rate	min	max	mean	sd
ES (ii)	8:10	88,500	3,994,600	887,562	1,313,640
ES (di)	8:10	111,900	7,115,400	1,874,620	2,410,130
ES (dd)	3:10	116,700	3,843,300	1,362,330	2,148,590
ES (id)	1:10	155,400	155,400	155,400	-

Table A.19: Parity-6: Success rate and number of function evaluations needed to reach the error limit. There were no successful runs with “nn” and “ng” recombination. With τ' set to $1/\sqrt{(n)}$ instead of $1/\sqrt{(2n)}$.

	suc. rate	min	max	mean	sd
ES (ii)	7:10	2,433,500	5,469,700	3,394,200	1,052,000

Table A.20: Spirals: Success rate and number of function evaluations needed to reach the error limit. With τ' set to $1/\sqrt{(n)}$ instead of $1/\sqrt{(2n)}$.

A.6 ESs on a Constant Objective Function

The error surfaces in the neural networks weight space are characterized by large and very flat planes that are caused by saturated activation functions. Caught on such a plane an ES behaves as if the underlying objective function is a constant. In this case intermediate and averaging recombination on the step sizes together with mutation causes them to grow exponentially. Accordingly, the object variables perform a random walk with growing order of magnitude. Figure A.41 illustrates this.

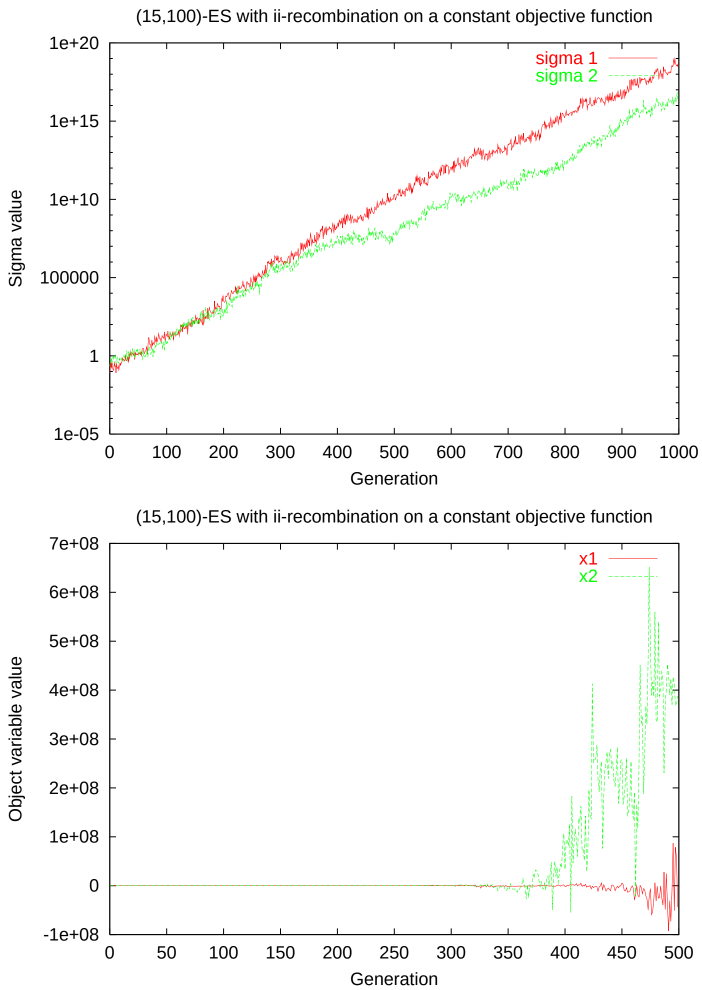


Figure A.41: The behavior of an ES on a constant 30 dimensional objective function. (Above) intermediate recombination of the step sizes together with mutation causes them to grow exponentially. (Below) the corresponding object variables perform a random walk with growing order of magnitude.

The median of the log normal distribution used for the mutation of step sizes (see Eq. 3.5) is 1 (e^0) and guarantees the neutrality of the process in the absence of selection and averaging recombination. Together with intermediate recombination and without selection, this is no longer the case.

Even though the median of each distribution is 1, the median of the joint distribution of two or more averaged mutations is larger than 1. Thus, intermediate recombination of step sizes together with mutation introduces a drift into the process. This effect has also been observed by Obalek [58]. In the absence of selective pressure this leads to the observed growth of the step sizes.

When using discrete, geometric or no recombination of step sizes at all, we do not observe an exponential growth of the step sizes.

A.7 Progress on the Sphere Model

Kursawe presented simulation results where a poorly parameterized ES failed on the simple sphere model ([43], p. 51) for larger problem dimensions. This effect is illustrated by the following figure A.42. A (15,100)-ES with n step sizes, global discrete recombination of object variables, global intermediate recombination of step sizes, and coupled step sizes was used. The starting point was: $x_i = -10^9$ with $\sigma_i = 2 \cdot 10^7$.

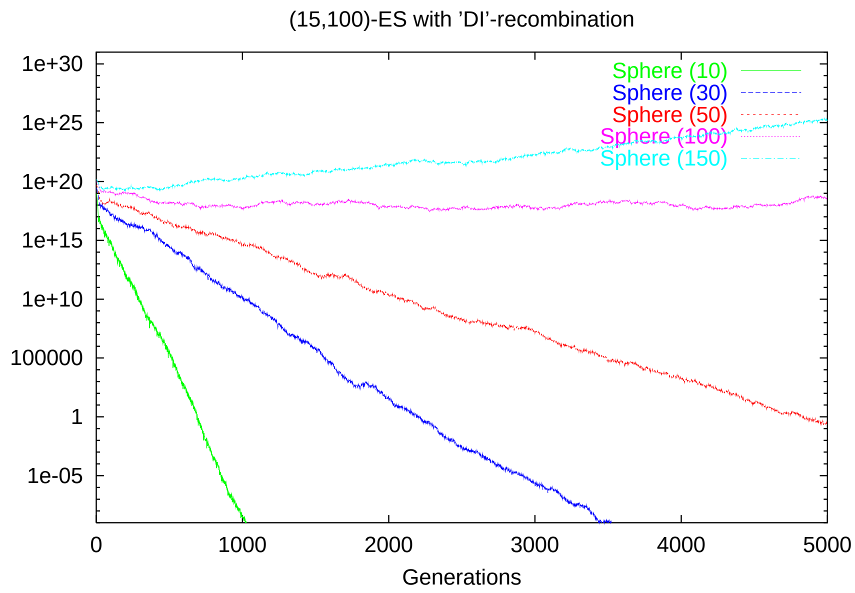


Figure A.42: Sphere model with varying problem dimensions (10, 30, 50, 100, 150): (15,100)-ES with n step sizes, global discrete recombination of object variables, and global intermediate recombination of step sizes.

This does not mean that ESs generally fail for large problem dimension[4], but the optimization becomes more difficult and the "correct" parameterization gets more important.

[4]In chapter 4 several problems with more than 300 parameters could be solved by ESs.

List of Figures

List of Tables

Bibliography

[1] J. T. Alander. An indexed bibliography of genetic algorithms and neural networks. Report Series 94-1-NN, University of Vaasa, Department of Information Technology and Production Economics, 1994. ftp://ftp.uwasa.fi/cs/report94-1/gaNNbib.ps.Z.

[2] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1996.

[3] T. Bäck. An overview of parameter control methods by self-adaptation in evolutionary algorithms. *Fundamenta Informaticae*, 35(1-4):51–66, 1998.

[4] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997.

[5] T. Bäck and H.-P. Schwefel. Evolution strategies I: Variants and their computational implementation. In G. Winter, J. Périaux, M. Galán, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science, Proc. First Short Course EUROGEN'95*, pages 111–126. Wiley, New York, Las Palmas de Gran Canaria, Spain, December 4–8, 1995.

[6] J. E. Baker. Reducing bias and inefficiency in the selection algorithm and their application. In J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21. Lawrence Erlbaum Assoc, 1987.

[7] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and Its Application.* dpunkt Verlag, Heidelberg, 1998.

[8] R. Battiti. Using mutual information for selecting features in supervised neural network learning. *IEEE Transactions on Neural Networks*, 5(4):537–550, 1995.

[9] H.-G. Beyer. Toward a theory of evolution strategies: on the benefit of sex – The $(\mu/\mu, \lambda)$-theory. *Evolutionary Computation*, 3(1):81–111, 1995.

[10] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[11] T. Blickle. Tournament selection. In Bäck et al. [4].

[12] P. P. Bonissone. Hybrid algorithms: the symbiosis. In Ruspini et al. [70].

[13] H. Braun. *Neuronale Netze Optimierung durch Lernen und Evolution.* Springer, Berlin, Heidelberg, 1997.

[14] G. Deco and D. Obradovic. *An Information-Theoretic Approach to Neural Computing.* Springer, Berlin, 1996.

[15] S. E. Fahlmann. Faster learning variations of back–propagation: An empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51. Morgan Kaufmann, San Mateo, CA, 1988.

[16] S. E. Fahlmann. The recurrent cascade-correlation architecture. In *Neural Information Processing Systems – NIPS 3.* Morgan Kaufmann, San Mateo, CA, 1991. ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.rcc.ps.Z.

[17] E. Fiesler and R. Beale, editors. *Handbook of Neural Computation.* Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1998.

[18] A. Fredenslund, R. L. Jones, and J. M. Prausnitz. Group-contribution estimation of activity coefficients in non-ideal liquid mixtures. In *AIChE Journal*, volume 21, pages 1086–1099. 1975.

[19] M. R. Gary and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

[20] H. Geyer. *Entwicklung und Untersuchung von Gruppenbeitragsmethoden zur Vorhersage thermodynamischer Stoffgrössen unter Verwendung von Methoden der Computational Intelligence.* Dr.-Ing. Dissertation, University of Dortmund, Department of Chemical Engineering, Chair of Thermondynamics, 1999.

[21] H. Geyer, P. Ulbig, and S. Schulz. Encapsulated evolution strategies for the determination of group contribution parameters in order to predict thermodynamic properties. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature 5*, pages 978–987. Springer, Berlin, 1998.

[22] H. Geyer, P. Ulbig, and S. Schulz. Use of evolutionary algorithms for the calculation of group contribution parameters in order to predict thermodynamic properties. Part 2: Encapsulated evolution strategies. In *Computers and Chemical Engineering 23*, volume 7, pages 955–973. 1999.

[23] H. Geyer, P. Ulbig, S. Schulz, and P. Bräuer. Vergleich zwischen klassischen und verschachtelten Evolutionsstrategien am Beispiel einer nichtlinearen Regression an Oberflächenspannungen in $\mathbb{R}^2$. Technical Report of the Collaborative Research Center 531 "Computational Intelligence" CI-66/99, University of Dortmund, March 1999.

[24] C. Goerick and T. Rodemann. Evolution strategies: An alternative to gradient based learning. In *Proceedings of the International Conference on Engineering Applications of Neural Networks (EANN-96)*, London, 1996.

[25] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading. Addison-Wesley, MA, 1989.

[26] J. Grefenstette. Rank-based selection. In Bäck et al. [4].

[27] J. Han, C. Moraga, and S. Sinne. Parametric feedforward networks. Technical Report 590, University of Dortmund, Department of Computer Science, November 1995.

[28] P. J. B. Hancock. A comparison of selection mechanisms. In Bäck et al. [4].

[29] J. Hartung. *Statistik: Lehr und Handbuch der angewandten Statistik*. Oldenbourg, München, Wien, 10th edition, 1995.

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

[31] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[32] C. Igel and M. Hüsken. Improving the Rprop learning algorithm. In H. Bothe and R. Rojas, editors, *Proceedings of the Second International Symposium on Neural Computation, NC'2000*. ICSC Academic Press, 2000.

[33] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. Technical Report UM-CS-1987-117, University of Massachusetts, Amherst, Department of Computer Science, November 1987.

[34] J. S. Judd. Learning in networks is hard. In M. Caudill and C. Butler, editors, *Proceedings of IEEE First International Conference on Neural Networks, San Diego*, pages 685–692, 1987.

[35] J. S. Judd. On the complexity of loading shallow neural networks. *Journal of Complexity*, 4(3):177–192, Sept. 1988.

[36] J. S. Judd. *Neural Network Design and Complexity of Learning.* The MIT Press, Cambridge, MA, 1990.

[37] J. S. Judd. How loading complexity is affected by node function sets. In S. J. Hanson, G. A. Drastal, and R. L. Rivest, editors, *Computational Learning Theory and Natural Learning Systems*, volume 1. MIT Press, Cambridge, MA, 1994.

[38] C. Kappler. Are evolutionary algorithms improved by large mutations? In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV, Int'l Conf. Evolutionary Computation*, volume 1141 of *Lecture Notes in Computer Science*, pages 346–355. Springer, Berlin, 1996.

[39] H. Kitano. Empirical studies on the speed of convergence of neural network training using genetic algorithms. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 789–795. MIT Press, Cambridge, MA, 1990.

[40] H. Kitano. Neurogenetic learning: An integrated method of designing and training neural networks using genetic algorithms. Technical Report CMU-CMT-92-134, Carnegie Mellon University, 1992.

[41] M. Klüppel, S. Schulz, and P. Ulbig. Univap - a group contribution method for the prediction of enthalpies of vaporization of pure substances. In *Fluid Phase Equilibria 102*, pages 1–15. 1994.

[42] C. Kracht, T. Friese, P. Ulbig, and S. Schulz. Development of an enthalpy based group contribution g^e-model (EBGCM). In *The Journal of Chemical Thermodynamics 31*, volume 5, pages 587–614. 1999.

[43] F. Kursawe. *Grundlegende empirische Untersuchungen der Parameter von Evolutionsstrategien – Metastrategien.* Dr.-rer. nat. Dissertation, University of Dortmund, Department of Computer Science, 1999.

[44] K. Lang and M. Witbrock. Learning to tell two spirals apart. In D. Touretzky and G. H. d T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann, San Mateo, CA, 1988.

[45] Y. LeCun, L. Bottu, G. B. Orr, and K.-R. Müller. Efficient backpropagation. In Orr and Müller [59], pages 9–50.

[46] S.-W. Lee. *Generalisierung und Optimierung der Cascade-Correlation-Architekturen unter Verwendung von neuen Aktivierungsfunktionen.* Dr.-rer. nat. Dissertation, University of Dortmund, Department of Computer Science, 1998.

[47] J.-H. Lin and J. S. Vitter. Complexity issues in learning by neural nets. Technical Report CS-90-01, Department of Computer Science, Brown University, Jan. 1990.

[48] A. Linden. Untersuchung von Backpropagation in konnektionistischen Systemen. Diploma thesis, University of Bonn, Informatik-Report Nr. 80, 1990.

[49] V. Majer and V. Svoboda. *Enthalpies of Vaporization of Organic Compounds: A Critical Review and Data Compilation.* Blackwell Scientific Publications, Oxford, 1985.

[50] M. Mandischer. Genetische Algorithmen zur Optimierung konnektionistischer Modelle. Diploma thesis, University of Dortmund, Department of Computer Science, 1992.

[51] M. Mandischer. Genetic optimization and representation of neural networks. In P. Leong and M. Jabri, editors, *Proc. Fourth Australian Conf. Neural Networks*, pages 122–125, Melbourne, February 1–3, 1993. Sydney University, Department of Electrical Engineering.

[52] M. Mandischer. Evolving recurrent neural networks with non-binary encoding. In *Proc. Second IEEE Int'l Conf. Evolutionary Computation (IEEE/ICEC'95), vol. 2*, pages 584–589, Perth, Australia, Nov. 29–Dec. 1, 1995. IEEE Press, Piscataway NJ.

[53] M. Mandischer, H. Geyer, and P. Ulbig. Comparison of neural networks, evolutionary techniques and thermodynamic group contribution methods for the prediction of heats of vaporization. Technical Report of the Collaborative Research Center 531 "Computational Intelligence" CI–70/99, University of Dortmund, September 1999.

[54] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity (reprint). In J. A. Anderson and E. Rosenfeld, editors, *Neurocomputing: Foundations of Research*, pages 18–28. MIT Press, 1988.

[55] D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 762–767, San Mateo, CA, 1989. Morgan Kaufmann.

[56] J. A. Nelder and R. Mead. A simplex method for function minimization. In *Computer Journal 7*, pages 308–313. 1965.

[57] S. C. Ng, H. Leung, and A. Luk. Evolution of connection weights combined with local search for multi-layered neural networks. In T. Fukuda, T. Furuhashi, and D. B. Fogel, editors, *Proceedings of the IEEE International Confernce on Evolutionary Computation (ICEC96)*, Nagoya, Japan, May 20–22, 1996. IEEE Press, Piscataway, NJ.

[58] J. Obalek. Rekombinationsoperatoren für Evolutionsstrategien. Diploma thesis, University of Dortmund, Department of Computer Science, 1994.

[59] G. B. Orr and K.-R. Müller, editors. *Neural Networks: Tricks of the Trade*, volume 1524 of *LNCS*. Springer, Wien, 1998.

[60] A. Ostermeier, A. Gawelczyk, and N. Hansen. Step-size adaption based on non-local use of selection information. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 189–198. Springer, Berlin, 1994.

[61] V. W. Porto, D. B. Fogel, and L. J. Fogel. Alternative neural network training methods. *IEEE Expert*, 10(3):16–22, 1995.

[62] L. Prechelt. Proben1 – A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Department of Computer Science, University of Karlsruhe, Germany, September 1994.

[63] L. Prechelt. Early stopping - but when? In Orr and Müller [59], pages 55–70.

[64] I. Rechenberg. *Evolutionsstrategie'94*. Werkstatt Bionik und Evolutionstechnik; 1. F. Frommann und G. Holzboog Verlag, Stuttgart, 1994.

[65] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons - from backpropagation to adaptive learning algorithms. *International Journal of Computer Standards and Interfaces*, 16:265–278, 1994.

[66] M. Riedmiller. Rprop - description and implementation details. Technical report, Institute for Logic, Complexity, and Deductionssystems, University of Karlsruhe, January 1994. http://www.ira.uka.de/ftp/pub/neuro/riedmiller/rprop.details.ps.Z.

[67] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The Rprop algorithm. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pages 568–591, San Francisco, April 1993.

[68] M. Rudnick. A bibliography: The intersection of genetic search and artificial neural networks. CS/E 90-001, Department of Computer Science and Engineering, Oregon Graduate Institute, 1990.

[69] D. E. Rummelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, MA, 1986.

[70] E. H. Ruspini, P. P. Bonissone, and W. Pedrycz, editors. *Handbook of Fuzzy Computation*. Institute of Physics Publishing, Bristol, 1997.

[71] M. Schmitt. *Komplexität neuronaler Lernprobleme*. Peter Lang Verlag, Frankfurt am Main, 1996.

[72] N. Schraudolph. Centering neural network gradient factors. In Orr and Müller [59], pages 207–226.

[73] H.-P. Schwefel. *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary Systems Research*. Birkhäuser, Basle, Switzerland, 1977.

[74] H.-P. Schwefel. On the evolution of evolutionary computation. In J. M. Zurada, R. J. M. II, and C. J. Robinson, editors, *Computational Intelligence – Imitating Life*, pages 116–124. IEEE Press, Piscataway NJ, 1994.

[75] H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology. Wiley Interscience, New York, 1995.

[76] H.-P. Schwefel and T. Bäck. Evolution strategies II: Theoretical aspects. In G. Winter, J. Périaux, M. Galán, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science, Proc. First Short Course EUROGEN'95*, pages 127–140, Las Palmas de Gran Canaria, Spain, December 4–8, 1995. Wiley, New York.

[77] H.-P. Schwefel and F. Kursawe. On natural life's tricks to survive and evolve. Technical Report of the Collaborative Research Center 531 "Computational Intelligence" CI–18/98, University of Dortmund, January 1998.

[78] H.-P. Schwefel and G. Rudolph. Contemporary evolution strategies. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *Advances in Artificial Life, Third European Conf. Artificial Life, Granada, Spain, June 1995, Proc.*, pages 893–907. Springer, Berlin, 1995.

[79] H.-P. Schwefel, G. Rudolph, and T. Bäck. Contemporary evolution strategies. Technical Report of the Systems Analysis Research Group SYS–6/95, University of Dortmund, Department of Computer Science, December 1995.

[80] K. A. Simmrock, R. Janowsky, and A. Ohnsorge. *Critical Data of Pure Substances, Vol. I and II.* Dechema, Frankfurt, 1986.

[81] J. Smith and T. C. Fogarty. Self adaption of mutation rates in a steady state genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC 96)*, pages 318–323, 1996.

[82] M. Stinchcombe and H. White. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *Proceedings of the International Joint Conference on Neural Networks (Washington)*, volume 1, pages 613–617. IEEE Press, Long Beach, CA, 1989.

[83] M. B. Stinchombe. Theoretical considerations for choosing a network topology. In Fiesler and Beale [17], chapter B.

[84] K. Swingler. *Applying Neural Networks - A Practical Guide.* Academic Press, 1996.

[85] V. Sychev, A. A. Vasserman, V. A. Zagoruchenko, G. A. Spiridonov, and V. A. Tsymarny. Thermodynamic properties of ethane (vol. 4). In *National Standard Reference Data Service of the USSR, A series of property tables*, pages 90–93. Springer, Berlin, 1987.

[86] P. Ulbig. *Gruppenbeitragsmodelle UNIVAP & EBGCM.* Dr.-Ing. Dissertation, University of Dortmund, Institute for Thermodynamics, 1996.

[87] P. Ulbig, T. Friese, H. Geyer, C. Kracht, and S. Schulz. Prediction of thermodynamic properties for chemical engineering with the aid of computational intelligence. In *Progress in Connectionist-based Information Systems, Proceedings of the International Conference on Neural Information Processing and Intelligent Information Systems*, volume 2, pages 1259–1262. Springer, New York, 1997.

[88] P. Ulbig, M. Klüppel, and S. Schulz. Extension of the UNIVAP group contribution method: enthalpies of vaporization of special alcohols in the temperature range from 313 to 358 K. In *Thermochimica Acta 271*, pages 9–21. 1996.

[89] A. Ultsch. Kopplung deklarativer und konnektionistischer Wissensrepräsentation. Technical Report 352, University of Dortmund, Department of Computer Science, 1990.

[90] A. Ultsch. *Konnektionistische Modelle und ihre Integration mit wissensbasierten Systemen (Habilitationsschrift).* University of Dortmund, Department of Computer Science, Dortmund, 1991.

[91] A. Ultsch, R. Hannuschka, U. Hartmann, M. Mandischer, and V. Weber. Optimizing symbolic proofs with connectionist models. In Kohonen, Mäkisara, Simula, and Kangas, editors, *Artificial Neural Networks*, pages 585–590. North-Holland, Amsterdam, 1991.

[92] P. Webros. *Beyond regression: new tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, Cambridge, MA, 1974.

[93] D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trails is best. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[94] W. Wienhold. Minimizing the system error in feedforward neural networks with evolution strategy. In S. Gielen and B. Kappen, editors, *Proceedings of the International Conference on Artificial Neural Networks*, pages 490–493. Springer, London, 1993.

[95] A. Zell. *Simulation Neuronaler Netze.* Addison-Wesley, 1994.