Exploring the Parameter Space of a Genetic Algorithm for Training an Analog Neural Network

Steffen G. Hohmann, Johannes Schemmel, Felix Schürmann, Karlheinz Meier
Kirchoff-Institute for Physics, Heidelberg University, Schröderstrasse 90,
69120 Heidelberg, Germany
e-mail: hohmann@kip.uni-heidelberg.de
phone: Germany-(0)6221-544329

Abstract

This paper presents experimental results obtained during the training of an analog hardware neural network. A simple genetic algorithm is used to optimize the synaptic weights. The parameter space of this algorithm has been intensively scanned for two learning tasks (4 and 5 bit parity). The results provide a quantitative insight into the interdependencies of the evolution parameters and how the optimal settings are predetermined by the learning problem. It is observed that population sizes in the order of 15 in connection with mutation rates of about 1% yield the best performance of the training when using moderate selection. The optimal population size is found to be independent of the learning task. A significant improvement of the training success can be achieved, when the role of crossover is reduced and higher mutation rates combined with stronger selection are applied. The observations are shown to be essentially independent of the signal to noise ratio within the analog hardware.

1 Introduction

Artificial neural networks as a computational model and simulated evolution as an optimization procedure are both biologically inspired concepts that have been successfully applied to numerous problems. Their combination to evolutionary artificial neural networks (EANNs) has been widely studied in the past years [Yao, 1999]. The processing of information within a neural network allows a high degree of parallelism, which motivates a special hardware implementation. Our group has recently developed an analog neural network chip optimized for the training with evolutionary algorithms that exploits the high degree of parallelization possible in modern VLSI[1] technologies [Schemmel, 2001]. The chip is integrated in a training and test setup that allows processing of in the order of 1000 individuals per second. This makes it feasible to approach another critical aspect of evolutionary optimization: The performance of an evolutionary algorithm depends on the tuning of several parameters that can have a massive impact on the success and the speed of the optimization process [De Jong, 1975]. Depending on the applied algorithm and the chosen problem, the interrelation of the various parameters can be very complex. Although recent theoretical investigations on the dynamics of genetic algorithms show promising results [Rogers, 2001], the complexity of realistic problems - e.g. the training of neural networks - greatly obstructs their theoretical analysis. In practice the optimal evolution parameters for a given task and a specific algorithm still have to be approximated by trial and error, guessed by experience or have to be found by other optimization procedures themselves [Grefenstette, 1986]. What additionally impedes thorough investigations of the interaction of evolution parameters is the limited amount of available experimental data, as it is usually costly to obtain [Schaffer, 1989], [De Jong, 1975].

This paper discusses the results obtained from more than 260,000 evolution runs carried out to train neural networks for two different tasks, which corresponds to several billions of tested individuals. A total of about 950 parameter settings are evaluated and analyzed with respect to the success of the learning process within a fixed number of fitness evaluations. The aim of this investigation is to clarify the influence of the various evolution parameters on the performance of the training algorithm. To promote a deeper understanding of the results and to motivate further theore-

[1]Very Large-Scale Integration

tical analysis, the genetic representation and the evolutionary algorithm have been chosen as to maintain a close resemblance to classical genetic algorithms (GAs) [Goldberg, 1989]. Furthermore, the tackled learning tasks (4 and 5 bit parity) are small in size, but known to be challenging problems for neural network training. It is assumed that the main aspects of the results can qualitatively be transfered to more complex tasks.

2 Hardware Realization of the Neural Network

The analog neural network chip used throughout the experiments is based on the concept of a feedforward network. It consists of 64×64 synapses that connect each of the 64 input neurons with each of the 64 output neurons. The analog operation of the chip is limited to the synaptic weights and the inputs of the output neurons. Both, input (I_j) and output signals (O_i) of the network are digital. A synapse is activated, when the value of the corresponding input neuron is set to 1. This way, the weight multiplication is reduced to an addition: The weight of each synapse (w_{ij}) is stored as charge on a storage capacitor and each output neuron compares the sum of the signals of all its activated synapses with a fixed reference voltage (b). If the reference value is exceeded, the neuron fires.

Throughout the presented experiments, the chip is operated in a special mode: All input neurons are combined to pairs. Each pair is used as one differential input. This is done automatically by setting the value of every second input neuron to the inverted value of its neighbour: If one input neuron is switched off, its inverted counterpart is switched on and vice versa. In this *combine* mode the operation of the network can be modeled by a slightly modified Perceptron formula with the activation function $g(x)$ of the output neurons set to the Heaviside function $\Theta(x-b)$:

$$O_i = \Theta\left(\sum_{j=0}^{31} [w_{i,2j+1} \cdot I_{2j+1} + w_{i,2j+2} \cdot (1 - I_{2j+1})] - b\right) \quad \text{with } I, O \in \{0,1\} \tag{1}$$

Hence, the connection of one combined input neuron to one specific output neuron is characterized by a set of two weights. The effective number of available input neurons is reduced to 32. On the other hand, the number of activated input neurons remains the same, independent of the input data. This has several advantages for the performance of the chip. A further discussion of these topics lies beyond the scope of this article. For more detailed technical information see [Schemmel, 2001].

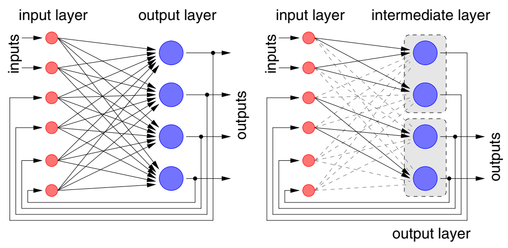

Figure 1: **Left**: A recurrent network. **Right**: Configured as a two-layer network by setting some synapses to zero (dashed lines).

The network operates within a discrete time update scheme, i.e. Equation 1 is calculated once for each network cycle. For the chip to be used as a recurrent network, the signals of half of the output neurons can be fed back to the input neurons. In this mode the output of the network at a given time does not only depend on the actual input, but also on the previous network cycle. The feedback can be used to configure the network as a virtual multi-layer Perceptron if the appropriate weights are set to zero. Both cases are schematically illustrated in Figure 1. If the chip is used as a multi-layer network with n layers, a corresponding number of network cycles is used to propagate a signal from the input to the output neurons. The maximum possible network frequency exceeds 50 MHz, which results in more than 200 giga connections per second and makes the implementation of networks with several layers practical.

3 Implementation of the Training Algorithm

The network is trained by a generational genetic algorithm with ranking selection, a chromosomewise one-point crossover and a single-gene mutation operator. Throughout our experiments the architecture of the network - i.e. the number of used input and output neurons, the active feedback connections and the number of network cycles - is fixed for each learning problem. The genome only contains information about the synaptic weights of the individual networks to configure the chip.

3.1 Genetic Representation

A genome consists of 64 chromosomes maximum, each describing the complete set of 64 synaptic weights that connect one output neuron to all input neurons. One weight is coded as a floating point number from the interval $[-1,\ 1]$ and is regarded as a complete single gene. Within one chromosome these weights are arranged to form a linear string.

Neither all neurons, nor all synapses of the array are necessarily used for a learning problem. In practice the genome only contains those chromosomes which parameterize the output neurons of the chip that are used either as inner neurons in hidden layers or as actual output neurons of the network. For technical reasons the chromosomes still contain information about all connecting synapses even if not all of the 64 input neurons are used. The parts of the genome which describe unused synapses have negligible effect on the performance of the represented network and thus do not influence the fitness of the individual.

3.2 Genetic Operators

During recombination a common one-point crossover operator is used and is successively applied to each corresponding pair of chromosomes provided by the two involved individuals. A new cut point is randomly and uniformly selected for each chromosome pair. The weights are regarded as the smallest units of the genome and always swapped as a whole. Accordingly, the mutation operator acts on the complete gene by replacing the weight to be mutated by a new value randomly and uniformly selected from the interval $[-1,\ 1]$. Both, crossover and mutation do not distinguish between parts of the genome that are relevant for the fitness and those which are inactive.

3.3 Selection and Reproduction Scheme

The algorithm acts on a population of ϕ genomes and the initial population is chosen by setting all genes of all individuals to random values uniformly distributed within the interval $[-1,\ 1]$. After evaluation of the fitness of each individual, the new generation is created from the preceeding population by successively executing three steps: selection, recombination and mutation. The selection scheme is rank based and is controlled by a parameter ρ called the replace percentage. After a fitness ranking of all individuals, the worst $n = \lfloor \rho \cdot \phi \rfloor$ genotypes are replaced by the best individual of the population. As long as $\rho \neq 0$ the worst individual is replaced even if $n \leq 1$. Thus if γ_i denotes the genotype at position i in the ranking we can write:

$$\gamma_i = \gamma_1 \quad \forall i \in \begin{cases} \{\phi - n + 1, \ldots, \phi\} & \text{if } n \geq 1 \\ \{\phi\} & \text{if } n < 1 \wedge \rho \neq 0 \\ \{\} & \text{if } \rho = 0 \end{cases}$$
$$n = \lfloor \rho \cdot \phi \rfloor \tag{2}$$

In the following step, all n replaced individuals are consecutively crossed with a new partner randomly selected among the whole population. Since γ_1 and the last n genotypes of the population are identical, it is possible that an individual is actually combined with a clone of itself which renders the crossover operator noneffective. On the other hand, one individual can be chosen more than once for crossover and may be changed repeatedly with cumulating effects. After the recombination step, the whole population is mutated whereby every single gene of each genotype in the population is changed with a fixed probability μ in the way described in 3.2. The resulting population of genotypes after the mutation step forms the new generation.

It should be noted that due to the applied mutation strategy even the best individual is not guaranteed to survive the generation step completely unchanged. Hence, the algorithm does not implement an elitist reproduction scheme in the sense of the usual definition [De Jong, 1975] though it is clearly biased towards a fast reproduction of the best indivdual.

3.4 General Considerations

In respect to the applicability of training neural networks, more sophisticated algorithms or codings are conceivable and have been numerously presented in the literature [Yao, 1999]. On the other hand, both, the presented coding and the algorithm combine direct compatibility to the used hardware with high simplicity, allow fast implementation and have proven to be capable of successfully training a neural network. Furthermore, the genetic representation of the problem and the applied operators retain a close resemblance to those used in classical GAs [Goldberg, 1989]. On the genotype level, the whole algorithm can completely be described by three variables ϕ, ρ and μ. This should allow for a better understanding of the performance of this algorithm depending on the parameters.

4 Experimental Setup

4.1 Technical Environment and Scale Factor

The network chip is connected to a standard PC by a PCI interface board that carries its own RAM and a field programmable gate array (FPGA). In the current stage of the setup, the training algorithm is implemented in C++ and executed on the host computer. At the beginning of an evolution run the testpatterns and target data of the learning task are sent to the RAM of the PCI-card where they remain throughout

learning problem	4BP	5BP
used genes	38	58
total genes	256	320
ineffective genes	218	262
No. of testpatterns	16	32
maximum fitness	160	320
threshold	157	317
individuals/sec	1061	949

Table 1: Specification of the applied learning problems.

the whole evolutionary process. During the evolution the data for each individual is sent to the network via the FPGA using a 16 bit digital to analog converter (DAC) that generates the analog weight values from the gene data. Once the weight values have been stored within the synapse array, the testpatterns are consecutively applied to the network. The results are read back by the FPGA and are used to calculate the fitness of the individual.

Due to the analog operation of the chip the reference voltage and weight values in the network are subject to fluctuations. The repeated application of the same input pattern does not necessarily yield identical outputs, especially, if the input signals of one or more output neurons are close to the reference value. For the selected problems it is thus necessary to present each test pattern repeatedly. The performance of the network can be evaluated by counting the number of correct bits in the output of the network summed over all successive presentations of the pattern. Besides reflecting the stability of the solution, this number also yields a smoother performance measure. It is therefore used as the fitness of the individual.

The range of possible weight values to be stored as charge within the synapse array is technically limited. The gene values from the interval $[-1,\ 1]$ have to be mapped onto this range of possible weights. We thus introduce the weight scaling factor $\omega \leq 1$ that controls this mapping. If $\omega = 1$, the whole technically possible range is exploited. If $\omega < 1$, only the corresponding fraction of the full range is used. Since the absolute average size of the signal fluctuations remains constant independent of the choice of ω, the variation of this parameter can be used to control the signal to noise ratio of the weights. The latter is expected to have considerable impact on the performance of the network.

4.2 Learning Task, Network Architecture and Success Condition

Within the presented experiments the network is trained for two tasks: 4 bit parity (4BP) and 5 bit parity (5BP). Being a linearly nonseparable problem the calculation of parity requires the chip to be configured as a two layer network [Hertz, 1991]. Hence, a number of two network cycles is used in each case. The hidden layer consists of three and four neurons for the 4BP and 5BP problems respectively. Since no weight is forced to be zero, shortcut connections from the input layer directly to the output neuron are possible. Figure 2 shows the resulting architectures for both problems. Similar architectures for the N bit parity task are reported in the literature [Pujol, 1998]. Since the neurons are operated in *combine* mode (Section 2), all synaptic connections are actually described by two separate weights that have to be optimized independently. For 4BP this results in $19 \times 2 = 38$ used weight values (see Figure 2). As the genome describes the complete set of synaptic weights for each of the four used output neurons (Section 3.1), it contains $4 \times 64 = 256$ genes, leaving 218 genes ineffective. The corresponding values for 5BP are shown in Table 1. To take into account the effects introduced by noise, each test pattern of the current task is applied ten times throughout all experiments which multiplies the maximum possible fitness by ten. The training is considered successful if the fitness of the best individual in the population averaged over the last five generations is better than a given threshold. The resulting values of maximum possible fitness and the chosen threshold values can also be found in Table 1. The thresholds are chosen empirically as to allow for small fluctuations in the performance of the found solution, which is a realistical approach for the setup used. Due to the applied success criteria, the length of an evolution run can obviously not be measured with a higher precision than ± 5 generations. The approximate average number of individials per second that can be processed by the setup depends on the population size ϕ and the number of testpatterns that are applied for the fitness evaluation. The corresponding values, when using a standard PC[2] and a population size of 100, are included in Table 1.

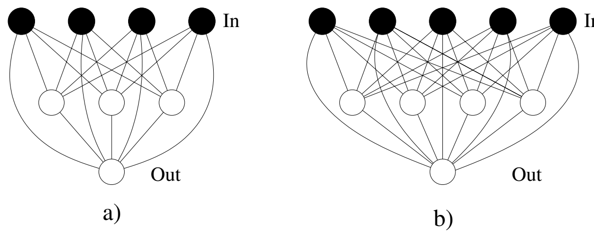

Figure 2: The used architectures for the 4BP problem (a) and the 5BP problem (b).

[2]In this case an AMD Athlon XP 1700 is used.

5 Experiments and Results

Several measures for the performance of a genetic algorithm are conceivable [De Jong, 1975]. Since it is a stochastic search procedure, comparing the performance among variations of a genetic algorithm with different parameters is not unproblematic [Schaffer, 1989]. The probability of the algorithm to reach the success condition described in 4.2 within a given number of fitness evaluations depends on the chosen evolution parameters. It will be regarded as the performance measure of a parameter configuration throughout the remainder of this article and is referred to as the yield Y given in percent. This rather practical approach has been chosen because the number of fitness evaluations is nearly proportional to the computation time needed by the serial implementation of the algorithm.

As discussed in the previous sections, the used training algorithm can be controlled by four parameters: The population size ϕ, the replace percentage ρ and the mutation rate μ (Section 3) parameterize the simulated evolution, while the weight scaling factor ω is specific for the used analog hardware (Section 4.1). The goal of the conducted experiments is to illuminate the influence of these parameters on the yield Y. Given a defined parameter setting and a learning task the yield is evaluated by conducting a number of N_E evolution runs and counting the successes N_S to calculate the success probability p and the yield using $p = N_S/N_E = Y/100$. Since all evolution runs are independent, we can assume the outcome of the experiment to be binomially distributed and estimate the error of the measured Y as $\Delta Y = \sqrt{pq/N_E} \times 100$ with $q = 1 - p$. The measurand Y is evaluated for parameter settings located on three selected two-dimensional hyperplanes of the four-dimensional parameter space as a complete scan would be impracticable. The following sections summarize the results obtained during the scans of the investigated subspaces for the various learning problems. The experiments have been conducted using two identical setups that were continuously running for a period of about eight weeks.

5.1 Population Size and Mutation Rate

This section presents the results of a simultaneous variation of the parameters ϕ and μ while keeping $\rho = 20\%$ and $\omega = 0.9$. For each learning task ϕ and μ have been varied within the ranges shown in Table 2, which also lists the total number of parameter settings n_s that has been evaluated for each problem. To use the computation time more efficiently, the trials have not been distributed uniformly within these ranges but have been clustered in regions with stronger variations of $Y(\phi, \mu)$. The results are qualitatively the same for both tasks and their analysis shall be discussed for 4BP as an example. Figure 3 shows the yield Y as a function of the mutaion rate μ for three different values of the population size ϕ. Given a fixed ϕ the yield shows a clear dependence of μ: Y exhibits a maximum at a specific ϕ-dependent value $\mu_{max}(\phi)$. While $\mu_{max}(\phi)$ itself increases with growing ϕ, the maximum yield $Y(\mu_{max})$ decreases. A closer examination reveals that for a fixed ϕ the data can be fitted satisfactorily by the function

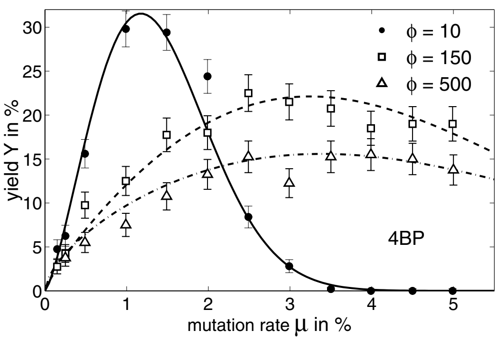

Figure 3: The yield Y as a function of μ for different values of ϕ and the 4BP problem ($\rho = 20\%$).

learning problem	4BP	5BP
population size ϕ	5-700	5-600
mutation rate μ in %	0.15-5	0.15-5
max. No. fitness eval.	16800	39600
No. of tested settings (n_s)	239	153
N_E per setting	400	300
opt. pop. size (ϕ_{opt})	14	13
opt. mut. rate (μ_{opt})	1.38	0.83
max. yield (Y_{opt}) in %	31.8	34.0
$\chi^2/n_{d.o.f.}$	1.36	2.96

Table 2: Parameters and results of the ϕ vs. μ experiments.

$$Y_F(\phi, \mu) = F_1(\phi) \cdot \mu^{F_2(\phi)} \cdot e^{F_3(\phi) \cdot \mu^2} \qquad (3)$$

and that the ϕ-dependece of the data can be modeled by setting

$$F_i(\phi) = f_{i1} \cdot \phi^{f_{i2}} + f_{i3} \quad , \quad i \in \{1, 2, 3\}, \qquad (4)$$

whereby we introduce the nine fit parameters f_{ij}. The combination of Equations 3 and 4 is fitted to the whole set of n_s evaluated points $Y_{kl} = Y(\phi_k, \mu_l)$, i.e. the Gauss-Newton method[3] is used to set the fit parameters with regard to the minimization of

$$\chi^2 = \sum_{k,l} \frac{(Y_{kl} - Y_F(\phi_k, \mu_l))^2}{\Delta Y_{kl}^2} \quad . \qquad (5)$$

[3]The fit routine is implemented in MATLAB.

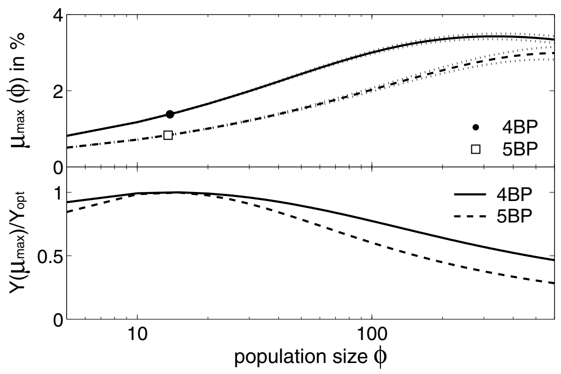

Figure 4: Fit results: The optimal mutation rate $\mu_{max}(\phi)$ and the normalized yield $Y(\mu_{max})/Y_{opt}$ as a function of ϕ for both tasks ($\rho = 20\%$). The symbols mark the parameters for overall maximum yield.

The fit results for the shown values of ϕ are presented in Figure 3 as line plots. It can be seen that the fit reproduces the data. The resulting χ^2 divided by the number of degrees of freedom $n_{d.o.f.} = n_s - 9$ is given in Table 2. The same fit procedure can be executed for 5BP, yielding different sets of f_{ij}. It is to be noted that Equations 3 and 4 are chosen empirically and are not motivated by a theoretical model so far. Nevertheless, they describe $Y(\phi, \mu)$ for both learning tasks reasonably well as can be seen from the values $\chi^2/n_{d.o.f.}$ in Table 2. For the following analysis the maximum number of fitness evaluations allowed for an evolution to fulfill the success condition is set for each learning task separately (see Table 2). The values are chosen as to lead to comparable maximum yields when using the respective optimal evolution parameters predicted by the fits. This is done to assure that the results of the analysis allow a comparison between the tasks although they vary in difficulty.

Given the two sets of fit parameters f_{ij} that describe the landscapes $Y(\phi, \mu)$ for each task, one can study the resulting function $\mu_{max}(\phi)$, which is shown in the upper half of Figure 4 within the investigated ranges on a logarithmic ϕ-scale. The calculated parameters f_{ij} exhibit a certain error due to the uncertainty ΔY of the measured Y. This leads to a finite precison of the prediction for $\mu_{max}(\phi)$ illustrated by the dotted lines. Both curves exhibit a similar behaviour: The optimal mutation rate shows a strong increase with ϕ for small population sizes but saturates for large populations. For 4BP it even slightly decreases for higher values of ϕ. Investigations on other optimization problems find an optimal mutation rate that is independent of the population size [Rogers, 2001] while others report the opposite correlation [Schaffer, 1989]. The lower half

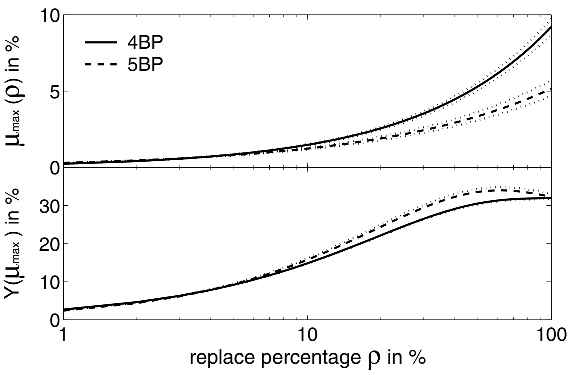

Figure 5: Fit results: $\mu_{max}(\rho)$ and $Y(\mu_{max})$ as a function of ρ for both tasks ($\phi = 100$).

of Figure 4 displays the yield $Y(\mu_{max})$ at the optimal mutation rate for a given ϕ divided by the yield Y_{opt} reached in the global maximum (ϕ_{opt}, μ_{opt}). It is remarkable that the yield displays a distinct maximum at a population size that is nearly equal for both tasks. Comparison to the upper half of Figure 4 reveals, that the regions of roughly constant μ_{max} are in fact regions of a low yield compared to the global maximum. The parameters (ϕ_{opt}, μ_{opt}) that lead to the highest possible yield as predicted by the fits are shown in Table 2 and are marked in Figure 4. Besides the similarity in the respective values of ϕ_{opt} it is noticable that they are significantly lower than those commonly suggested ([DeJong, 1975], [Grefenstette, 1986]) but are higher than the lowest possible ϕ as recommended by theoretical investigations concerning serial genetic algorithms [Goldberg, 1989b]. Other investigations on function optimization find optimal mutation rates that depend on the length l of the bit string used as genome like $\mu_{opt} = 1/l$ [Bäck, 1993]. The values in Table 2 seem to be in accordance with this observation if l is defined as the number of used genes (Table 1). However, since the used coding differs from a binary representation, the application of this formula is problematic.

5.2 Replace Percentage and Mutation Rate

In a set of experiments similar to those in Section 5.1 the replace percentage ρ and mutation rate μ have been varied while keeping $\phi = 100$ and $\omega = 0.9$. The scanned ranges of ρ and μ and the total number of evaluated settings is listed in Table 3. Once again the results are qualitatively the same for both tasks and the analysis of the data proceeds similar to Section 5.1: For a fixed replace percentage ρ the dependency of the yield $Y(\rho, \mu)$ on μ can be fitted by the function

$$Y_F(\rho, \mu) = G_1(\rho) \cdot \mu^{G_2(\rho)} \cdot e^{G_3(\rho) \cdot \mu^2} \qquad (6)$$

learning problem	4BP	5BP
replace percentage ρ in %	5-100	10-100
mutation rate μ in %	0.5-10	1-10
max. No. fitness eval.	16300	40000
No. of tested settings (n_S)	270	62
N_E per setting	400	300
opt. rep. pctg. (ρ_{opt}) in %	100	65
opt. mut. rate (μ_{opt})	9.2	4.0
max. yield (Y_{opt}) in %	32.0	34.5
$\chi^2/n_{d.o.f.}$	1.37	3.31

Table 3: Parameters and results of the ρ vs. μ experiments.

and a closer investigation reveals, that the ρ-dependency can be modeled as

$$G_1(\rho) = g_1 \cdot e^{g_2 \cdot \rho} + g_3 \quad (7)$$
$$G_2(\rho) = g_4 \quad (8)$$
$$G_3(\rho) = g_5 \cdot \rho^{g_6} \quad . \quad (9)$$

Note that the exponent $G_2 = g_4$ of μ in Equation 6 does not depend on ρ and that this model uses just six parameters instead of nine as in Section 4. Fitting the set of Equations 6-9 to the data, one finds that $g_4 = 1.008$ for 4BP and $g_4 = 1.514$ for 5BP. If g_4 is not regarded as a fit parameter but is set manually to 1 and 1.5 for 4BP and 5BP respectively, the quality of the fit does not suffer measurably. The resulting values of $\chi^2/n_{d.o.f.}$ are shown in Table 3. It is remarkable that g_4 apparently exhibits constant simple values that seem to depend only on the learning task. While this article is written, additional data is gathered for the 5BP task. More data points will improve the quality of the fit and might allow a more reliable prediction for g_4. However, it can be foreseen that the conclusions of the following analysis will not be affected.

Given the fit parameters g_l, it is possible to investigate the optimal mutation rate μ_{max} for a given ρ and the yield $Y(\mu_{max})$ at this point. Figure 5 shows the results. For both tasks, the optimal mutation rate μ_{max} increases monotonically with increasing ρ. This is understandable, since the diversity in the population is reduced significantly when ρ increases. Thus the introduction of new genes by mutation becomes more important. The yield $Y(\mu_{max})$ also increases with ρ. For 4BP the maximum possible yield is in fact obtained at $\rho = 100\%$, while for 5BP a maximum is reached at $\rho \approx 65\%$. Additional data will clarify if the latter observation is an authentic feature or an effect introduced by the limited precision of the fit. However, it is obvious that high replace percentages ρ in combination with high mutation rates μ result in an optimal yield Y. Note that the choice of $\rho = 100\%$ renders the crossover operator noneffective and the search is driven solely by the selection of the best individual and the parallel mutation of its ϕ clones. It has previously been suggested, that this kind of strategy leads to better performance than algorithms based mainly on recombination [Schaffer, 1989]. Especially, with regard to the training of neural networks, the crossover operator is known to be problematic [Yao, 1999]. The presented results verify these considerations.

learning problem	4BP	5BP
scale factor ω	0.13-1.0	0.13-1.0
mutation rate μ in %	0.5-4.5	0.6-3.6
max. No. fitness eval.	20000	40000
No. of tested settings (n_S)	126	84
N_E per setting	400	300
opt. mut. rate (μ_{opt}) in %	2.6	1.6

Table 4: Parameters and results of the ω vs. μ experiments.

5.3 Weight Scaling and Mutation Rate

In this section we discuss the effects of a variation of the weight scaling factor ω on the success of the training algorithm. At $\phi = 50$ and $\rho = 20$ the yield Y is measured for various settings of μ and ω as given in Table 4. In accordance to the previous observations, Y reaches a maximum at a certain optimal mutation rate μ_{max} for each value of ω. The maximum can be found by fitting the data with a function similar to those presented in the foregoing sections:

$$Y_F(\omega, \mu) = H_1(\omega) \cdot \mu^{H_2(\omega)} \cdot e^{H_3(\omega) \cdot \mu^2} \quad (10)$$

This is done for each value of ω. The determined optimal mutation rates $\mu_{max}(\omega)$ together with the corresponding yields $Y(\omega, \mu_{max})$ are presented in Figure 6. It can be seen that the results are essentially independent of the learning task: Small values of ω result in a low yield or even inhibit the success of the training completely. Using a small scale factor leads to a poor signal to noise ratio of the weights and the performance of the network suffers accordingly. Under these condidtions it is not possible for the algorithm to find a stable solution. An increase of ω improves the signal to noise ratio. This makes it easier for the algorithm to find a network with a stable performance despite the presence of noise. Thus the maximum yield increases. For a scale factor higher than approximately 0.6 the yield is seen to be independent of ω within the given precision. It can be concluded that the success of the training is not limited by the noise of the weights once a specific signal to noise ratio is exceeded.

The errors of the derived optimal mutation rates depend on the distinctness of the respective maximum in $Y(\omega, \mu)$ and on the precision of the data. The latter decreases, when the yield is small. The clearness

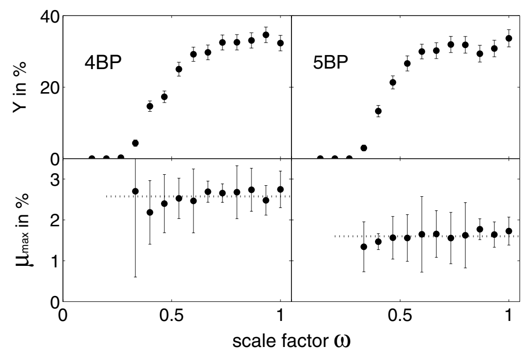

Figure 6: Yield Y and optimal mutation rate μ_{max} as a function of ω for 4BP (left) and 5BP (right).

of the maximum varies slightly between the measurements at different ω. Within the estimated error the optimal mutation rate μ_{max} shows no dependency on the scale factor ω and the signal to noise ratio. This even holds for regions of ω, where the yield Y is still seen to be affected by the weight noise. The previously described experiments were conducted using $\omega = 0.9$. In this domain, both, the optimal mutation rates and the yields are not measurably influenced by the weight fluctuations. It can be inferred that the presented results concerning the interaction of the parameters are not influenced by the presence of noise. However, it shall be repeated that the weights are still subject to fluctuations even for $\omega = 1$. It is not possible to test the system without any noise. On the other hand, software simulations of the system indicate that a minimal level of noise can in fact be advantageous for the training. Future studies with a new generation of analog network chips [Schemmel, 2002] have to investigate the behaviour of the system under a further reduction of the signal noise.

6 Conclusions

We have presented detailed experimental data showing the influence of the evolution parameters on the success of the training of an analog neural network. A quantitative analysis of the data has been conducted and reveals strong dependencies between the parameters. The results suggest that population sizes in the order of 15 and mutation rates of about 1% are to be preferred if the algorithm relies mainly on recombination and applies moderate selection. The optimal population size seems to be independent of the learning problem. The performace of the algorithm improves significantly when a stronger selection is applied and recombination is given only a small role or is even neglected completely. In this case, emphasis has to be placed on mutation, thus higher mutation rates are favorable. We have shown, that these conclusions are not influenced by the noise of the analog weight values within the chip and should in principle be transferable to other systems. At this moment, we do not have a thorough theoretical understanding of the results. It is reasonable to assume that the qualitative behaviour common to both tasks might be a feature of the algorithm, while the concrete values of the optimal parameters could in principle be connected to characteristics of the specific tasks and the used network architectures. We hope that our results encourage further theoretical or experimental research in this direction[4].

References

T. Bäck (1993). Optimal Mutation Rates in Genetic Search. *International Conference on Genetic Algorithms*, University of Illinois, Urbana-Champaign, pp. 2-8.

K. DeJong (1975). The Analysis and Behaviour of a Class of Genetic Adaptive Systems. *PhD thesis*, University of Michigan. Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76-9381.

D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, MA.

D. E. Goldberg (1989b). Sizing Populations for Serial and Parallel Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA, pp. 70-79.

J. J. Grefenstette (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16, No. 1, pp. 122-128.

J. A. Hertz, A. Krogh, R. G. Palmer (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, Inc., Redwood City, CA.

J. Pujol, R. Poli (1998). Evolving Neural Networks Using a Dual Representation with a Combined Crossover Operator. *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC)*, pp. 416-421.

A. Rogers, A. Prügel-Bennett (2001). A Solvable Model of a Hard Optimization Problem. *Theoretical Aspects of Evolutionary Computing*, pp. 209-224, Springer.

J. D. Schaffer, R. A. Caruana, L. J. Eshelman, R. Das (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA, pp. 51-60.

J. Schemmel, K. Meier, F. Schürmann (2001). A VLSI Implementation of an Analog Neural Network suited for Genetic Algorithms. *Proceedings of the International Conference on Evolvable Systems 2001*, Springer, pp. 50-61.

J. Schemmel, F. Schürmann, S. Hohmann, K. Meier (2002). An Integrated Mixed-Mode Neural Network Architecture for Megasynapse ANNs. To appear in the *Proceedings of the International Joint Conference on Neural Networks 2002.*

X. Yao (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE*, Vol. 87, No. 9, pp. 1423-47.

[4] The data and the MATLAB macros are available at *http://www.uni-heidelberg.de/vision/projects/evo_fpnn_related.html* .