Vita

Philipp Koehn was born in Erlangen, Germany on August 1, 1971 as the second son of Sigrid and Johannes Koehn. He attended the Buechenbach-Nord elementary school and graduated from the Albert-Schweitzer-Gymnasium in 1990. Since 1990, he has studied at the University of Erlangen-Nuremberg Computer Science. He entered the University of Tennessee at Knoxville in Fall 1993, and received a Master degree in Computer Science in 1994.

and neural networks: optimizing connections and connectivity", in: *Parallel Computing 14*, p. 347-361, North-Holland.

(Whitley, 1991): D. Whitley, S. Dominic, R. Das: "Genetic Reinforcement Learning with Multilayer Neural Networks", in: *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 562-569, Morgan Kaufmann.

(Whitley, 1992): Darell Whitley, J. David Schaffer, Larry J. Eshelman: "Combinations of genetic algorithms and neural networks: A survey of the State of the Art", in *Proceedings of the International Workshop on Combinations of genetic algorithms and neural networks*, p. 1-37, Baltimore, IEEE.

(Whitley, 1993): Darell Whitley, Stephen Dominic, Rajarshi Das, and Charles W. Anderson: "Genetic Reinforcement Learning for Neurocontrol Problems", in *Machine Learning*, 13, p. 259-284, Kluwer.

(Wieland, 1991): Alexis P. Wieland: "Evolving Neural Network Controllers for Unstable Systems", in: *International Joint Conference on Neural Networks*, Vol. II, pp. 667-673, IEEE.

(Williams, 1994): Bryn V. Williams, Richard T. J. Bostock, David Bounds and Alan Harget: "Improving Classification Performance in the Bumptree Network by Optimising Topology with a Genetic Algorithm", `ftp://archive.cis.ohio-state.edu (128.146.8.52) /pub/neuroprose /williams.wcci94.ps.Z`

(Wong, ?): Francis Wong: "Genetically Optimized Neural Networks", `ftp://archive.cis.ohio-state.edu (128.146.8.52) /wong.nnga.ps.Z`

The symbol * indicates that I did not use these sources for this thesis, although I refer to them in the text.

formance Analysis of Neural Network Architectures", Technical Report 16/1992, University of Koblenz, Germany, `ftp://archive.cis.ohio-state.edu (128.146.8.52) /pub/neuroprose/schiff.nnga.ps.Z`

(Schiffmann, 1992b): W. Schiffmann, M. Joost, R. Werner: "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons", `ftp://archive.cis.ohio-state.edu (128.146.8.52) /pub/neuroprose/bp_speedup.ps.Z`

(Schiffmann, 1993): W. Schiffmann, M. Joost, R. Werner: "Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons", in: *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, pp. 675-682, Innsbruck.

(Srinivas, 1991): M. Srinivas and L.M. Patnaik: "Learning Neural Network Wights Using Genetic Algorithms - Improving Performance by Search Space Reduction", in: *Proceedings of the International Joint Conference on Neural Networks*, pp 2331-2336, IEEE.

(Syswerda, 1989): Gilbert Syswerda: "Uniform Crossover in Genetic Algorithms", in: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 2-9, Morgan Kaufmann.

(Tanese, 1989): Reiko Tanese: "Distributed Genetic Algorithms", in: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 434-439, Morgan Kaufmann.

(White, 1993): David W. White: "GANNet: A genetic Algorithm for Searching Topology and Weight Spaces in Neural Network Design", Dissertation at the University of Maryland.

(Whitley, 1989): D. Whitley: "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best", in: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 116-121, Morgan Kaufmann.

(Whitley, 1990): D. Whitley, T. Starkweather et C. Bogart: "Genetic algorithms

Network Connectivity", in: *International Conference of Neural Networks*, Vol. II, pp. 863-868, San Francisco, IEEE.

(McInerney, 1993): Michael McInerney and Atam P. Dhawan: "Use of Genetic Algorithms with Back-Propagation in Training of Feed-Forward Neural Networks", in: *International Conference of Neural Networks*, Vol. I, pp 203-208, San Francisco, IEEE.

(Miller, 1989): Geoffrey F. Miller, Peter M. Todd and Shailesh U. Hedge: "Designing Neural Networks using Genetic Algorithms", in: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 379-384, Morgan Kaufmann.

*(Montana, 1989): D. Montana and L. Davis: "Training feedforward neural networks using genetic algorithms", in: *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 762-767, Morgan Kaufmann.

(Montana, 1991): David J. Montana: "Automated Parameter Tuning for Interpretation of Synthetic Images", in: *Handbook for Genetic Algorithms*, pp. 282-311.

(Munro, 1993): Paul W. Munro: "Genetic Search for Optimal Representation in Neural Networks", in: *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, pp. 675-682, Innsbruck.

(Rojas, 1991): Raul Rojas: "Theorie der neuronalen Netze", Springer Verlag.

(Rumelhart, 1986)*: D. Rumelhart, G. Hinton and R. Williams: "Learning Internal Representation by Error Propagation", in: *Parallel Distributed Processing*, p. 318-362, MIT Press.

(Schiffmann, 1991): Wolfram Schiffmann, Merten Joost, Randolf Werner: "Performance Evaluation of Evolutionary Created Neural Network Topologies", in:. *Parallel Problem Solving from Nature 2*, pp. 292-296, H.P. Schwefel and R. Maenner, Springer Verlag.

(Schiffmann, 1992): W. Schiffmann, M. Joost, R. Werner: "Synthesis and Per-

International Joint Conference on Neural Networks, Vol. II, pp. 397-404, IEEE.

(Lindenmayer, 1968): Aristid Lindenmayer: "Mathematical Models for Cellular Interactions in Development", in: *Journal of Theoretical Biology*, Vol. 18, pp. 280-299.

(Lindgren, 1992): Kristian Lindgren, Anders Nilsson, Mats G. Nordahl and Ingrid Rade: "Regular Language Inference Using Evolving Neural Networks", in: *Proceedings of the International Workshop on Combinations of genetic algorithms and neural networks*, p. 75-86, Baltimore, IEEE.

(Manderick, 1989): Bernard Manderick and Piet Spiessen: "Fine-Grained Genetic Algorithms", in: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 428-433, Morgan Kaufmann.

(Mandischer, 1993): Martin Mandischer: "Representation and Evolution of Neural Networks", in: *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, pp. 643-649, Innsbruck.

(Maniezzo, 1993): Vittorio Maniezzo: "Searching among Search Spaces: hastening the genetic evolution of feed-forward neural networks", in: *International Joint Conference on Neural Networks and Genetic Algorithms,* p. 635-642, Innsbruck.

(Maniezzo, 1994): Vittorio Maniezzo: "Genetic Evolution of the Topology and Weight Distribution of Neural Networks", in: *IEEE Transactions of Neural Networks*, Vol. 5, No. 1, pp 39-53.

(Marti, 1992a): Leonardo Marti: "Genetically Generated Neural Networks I: Representational Effects", in: *IEEE International Joint Conference on Neural Networks*, Vol. IV, p. 537-542.

(Marti, 1992b): Leonardo Marti: "Genetically Generated Neural Networks II: Searching for an Optimal Representation", in: *IEEE International Joint Conference on Neural Networks*, Vol. II, p. 221-226.

(McDonnell, 1993): John R. McDonnell and Don Waagen: "Evolving Neural *tions of genetic algorithms and neural networks*, p. 108-122, Baltimore, IEEE.

(Happel, 1994): Bart L. M. Happel and Jacob M. J. Murre: “Design and Evolution of Modular Neural Network Architectures”, in: *Neural Networks*, Vol. 7, Nos. 6/7, pp. 985-1004.

(Harp, 1989): Steven Alex Harp, Tariq Samad, Aloke Guha: “Towards the Genetic Synthesis of Neural Networks”, in: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 360-369, Morgan Kaufmann.

(Harp, 1991): Steven A. Harp and Tariq Samad: “Genetic Synthesis of Neural Network Architecture”, in: *Handbook of Genetic Algorithms*, pp. 202-221.

(Jacob, 1993): Christian Jacob and Jan Rehder: “Evolution of neural net architectures by a hierarchical grammar-based genetic system”, in: *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, pp. 72-79, Innsbruck.

(Karunanithi, 1992): Nachimuthu Karunanithi, Rajarshi Das and Darell Whitley: “Genetic Cascade Learning for Neural Networks”, in: *Proceedings of the International Workshop on Combinations of genetic algorithms and neural networks*, p. 134-145, Baltimore, IEEE.

(Kitano, 1990a): Hiroaki Kitano: “Designing Neural Networks Using Genetic Algorithms with Graph Generation Systems”, in: *Complex Systems*, No. 4, p. 461-476.

(Kitano, 1990b): Hiroaki Kitano: “Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms”, in; *Eighth National Conference on Artificial Intelligence*, Vol. II, pp 789-795, AAAI, MIT Press.

(Koza, 1991): John R. Koza: “Genetic Programming”, MIT Press.

(Koza, 1991b): John R. Koza and James P. Rice: “Genetic Generation of Both the Weight and Architecture for a Neural Network”, in: *Proceedings of the* *rithms*, pp. 80-85, Morgan Kaufmann.

(Gruau, 1992): Frederic Gruau: “Genetic synthesis of boolean neural networks with a cell rewriting developmental process”, in: *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, p. 55-74, Baltimore, IEEE.

(Gruau, 1992b): Frederic Gruau: “Cellular Encoding of Genetic Neural Networks”, research report #92-21 of Ecole Normale Superieure de Lyon, `ftp://lip.ens-lyon.fr (140.77.1.11) /pub/Rapports/RRRR92/RR92-21.ps.Z`

(Gruau, 1993): Frederic Gruau and Darrel Whitley: “Adding Learning to the Cellular Development of Neural Networks: Evolution and the Baldwin Effect”, in: *Evolutionary Computation*, No. 1, Vol. 3, pp. 213-233, MIT Press.

(Gruau, 1994): Frederic Gruau: “Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm”, PhD thesis at Ecole Normale Superieure de Lyon, `ftp://lip.ens-lyon.fr (140.77.1.11) /pub/Rapports/PhD/PhD94-01 -E.ps.Z`

(Fogel, 1993): David B. Fogel: “Using Evolutionary Programming to Create Neural Networks that are Capable of Playing Tic-Tac-Toe”, in: *International Conference of Neural Networks*, Vol. II, pp. 875-880, San Francisco, IEEE.

*(Holland, 1975): J.H. Holland: “Adaption in natural and artificial systems”, University of Michigan Press, Ann Harbor.

(Hancock, 1991): P. J. B. Hancock: “Gannet: Genetic design of a neural network for face recognition”, in: *Parallel Problem Solving from Nature 2*, pp. 292-296, H.P. Schwefel and R. Maenner, Springer Verlag.

(Hancock, 1992): Peter J.B. Hancock: “Genetic Algorithms and permutation problem: a comparison of recombination operators for neural net structure specification”, in: *Proceedings of the International Workshop on Combina-*

Bibliography

*(Bersini, 1992): H. Bersini and G. Seront: “In search of a good crossover between evolution and optimization”, in: *Parallel Problem Solving from Nature 2*, p. 479-488, Elsevier, Amsterdam.

(Belew, 1989): Richard K. Belew: “When both individuals and populations search: Adding simple learning to the Genetic Algorithm”, in: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 34-41, Morgan Kaufmann.

(Bishop, 1993): J.M. Bishop and M.J. Bushnell: “Genetic Optimisation of Neural Network Architectures for Colour Recipe Prediction”, in: *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, pp. 719-725, Innsbruck.

(Boers, 1992): Egber Boers and Herman Kuiper: “Biological metaphors and the design of modular artificial neural networks”, Master thesis at Leiden University, the Netherlands.

(Bornholdt, 1992): Stefan Bornholdt and Dirk Graudenz: “General Asymmetric Neural Networks and Structure Design by Genetic Algorithms”, in: *Neural Networks*, Vol. 5, pp. 327-334, Pergamon Press.

(Caruana, 1988): R.A.Caruana and J.D. Schaffer: “Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms”, in: *Proceedings of the Fifth International Conference on Machine Learning*, p. 153-161, Morgan Kaufmann.

(Dennet, 1991): Daniel C. Dennet: “Consciousness explained”.

(Eshelman, 1989): L.J. Eshelman, R.A. Caruana, and J.D. Schaffer: “Biases in the Crossover Landscape”, in: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 10-19, Morgan Kaufmann.

(Goldberg, 1989): David E. Goldberg: “Zen and the Art of Genetic Algorithms”, in: *Proceedings of the Third International Conference on Genetic Algo-*

Bibliography

Outlook

So far , many ideas regarding encoding strategies have been published. Also, a couple of empirical studies have produced intuition for the application of genetic algorithms to neural networks. This field, however , lacks attempts to create a theoretical account. A few principal properties - such as the structural-functional mapping problem - have been proposed, but the discussion of these is purely based on the interpretation of different empirical results.

GANN systems have already been applied to a couple of real world tasks. Future work should focus on robust GANN systems that can be used for commercial applications. The range of applications has been outlined.

evolved by the genetic algorithm, after it finished.

Genetic weight evolution does not scale up. Also, the preferred use of the mutation operator in small networks suggests that the genetic algorithm is not able to profit from its ability of parallel hyperplane sampling. Obviously , the sequential coding of weights does not sufficiently provide bit patterns, that can be exploited by the GA. There are multiple ways the weights can be set in order to solve a given problem. So, the genetic algorithm rather works as a genetic hillclimber .

In conclusion, the application of GANN systems for weight Optimization is reduced to the field of small networks, especially for control or other tasks, for which no error feedback exists. Since genetic algorithms operate on simple fitness, i.e. quality evaluations, they can be applied to a larger class of problems. Back-propagation training requires the information of a target output and cannot be used in a couple of control tasks.

Architecture Optimization

Even simple weight-based encoding shows promising results for the task of architecture optimization. Consistent progress can be observed in a small number of generations, when the problem of a noisy fitness function was able to be overcome by a sufficient number of training processes.

However , architecture optimization by a GANN system comes with a extremely high computational cost. If only one specific problem has to be examined, the search for a good architecture using a genetic algorithm can hardly be justified. Using a relatively bad architecture and spending computation time on excessive training will yield better results.

The application of GANN architecture optimization is feasible, if the goal is a good topology for a large class of similar problems. In these cases, the high computation time is justified, since it has to be performed only once. Then, for a particular instance of the problem the optimized architecture can be used.

3.5 Conclusions

We successfully applied the genetic algorithm paradigm to neural networks in the task of weight as well as architecture Optimization.

Weight Optimization

For weight Optimization, the GANN systems outperforms traditional neural network training, when small networks were used. In these small search spaces, the global approach of genetic algorithms proves to be roughly comparable to the local back-propagation. We also showed that genetic algorithms do considerably better than random search for weight values.

A mix of mutation and crossover operators proves to be most successful. Especially the crossover operator alone is not able to find good weight settings as fast as in combination with mutation.

The high number of encoding bits increases the performance considerably with hardly any increase of computational cost. An optimal range of weights is $(-20, 20)$ for the sine function, independent from the number of encoding bits. This range also resulted in good performance on other problems.

A large generation size and the use of a distributed genetic algorithm has also shown success. These strategies decrease the possibility of getting stuck in local minima and preserve the diversity among the individuals of the population.

However, the introduction of gray coding was not satisafactory, since no performance improvement could be observed for neither boolean nor real-valued functions.

Also, the Baldwin effect has no significant impact on the genetic search. The search space seems to be continuous enough, so that no further guidance by NN training is required. Hence back-propagation learning during the run of the genetic algorithm can not be considered as a reasonable strategy. However, back-propagation may be used to fine tune a weight setting that was tures. The assigned fitness for the best individuals fall remarkably below their actual fitness, which is indicated by the reevaluations. Better results could be achieved with 20 training processes and E=50,000. The evolved architectures reach error that even come very close to the ones in the E=100,000 run, although only half the number of training epochs are used.

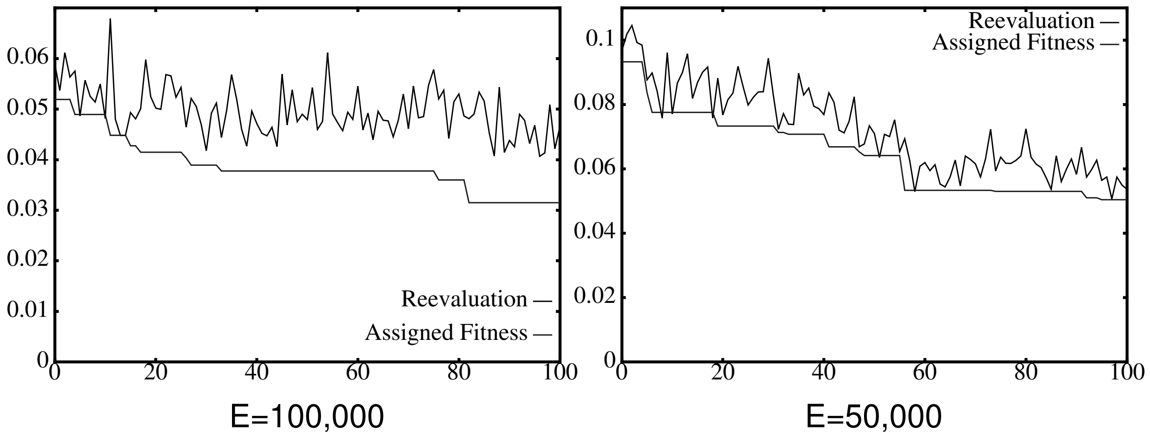
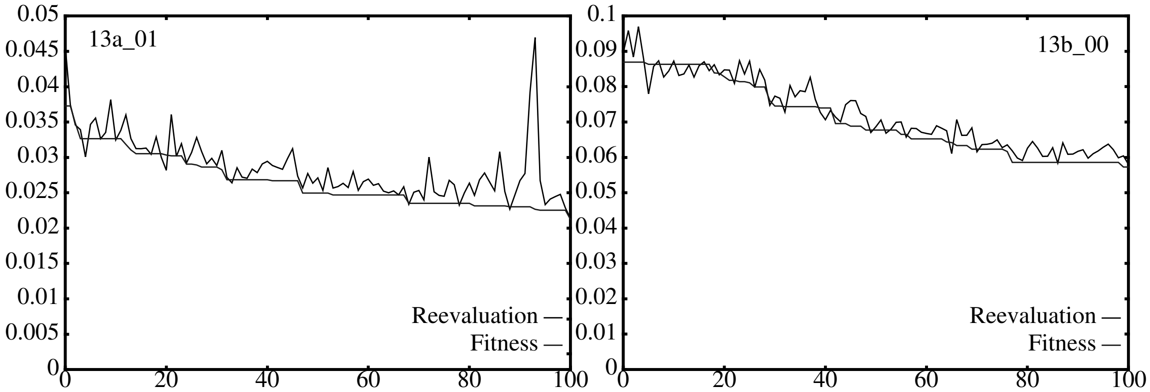
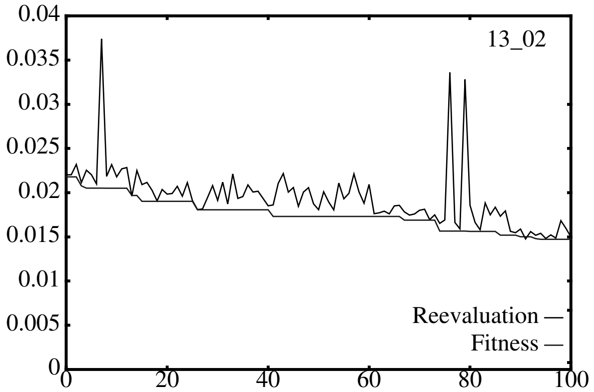


Figure 3.21: Architecture Optimization for Sine



E = 50,000 E = 20,000



E = 100,000

Figure 3.20: Architecture Optimization for the 1-Bit Adder

Sine

Finally, in experiment 18, we treat the sine function. Since 17 training patterns are involved, a run of experiment 18 takes about two days computation time on a SUN SPARC5m70.

Experiment 18: 100 generations, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover-mutation rate 20. Varying number of epochs, varying number of evaluations, learning rate 0.8. Sine with 17 training patterns. Weight-based encoding: 13 total nodes, 1 index bit, mutation rate 20. Two-point-crossover.

The results of runs for E=50,000 and E=100,000 are displayed in figure 3.21 in the usual fashion. The run for 100,000 suffers from noisy architec-

Figure 3.19 displays a network that was generated in a E=100,000 run. It has a more complicated structure. But also here, unit 3 is almost abolished and the nodes 5 and 2 are merely bias units.

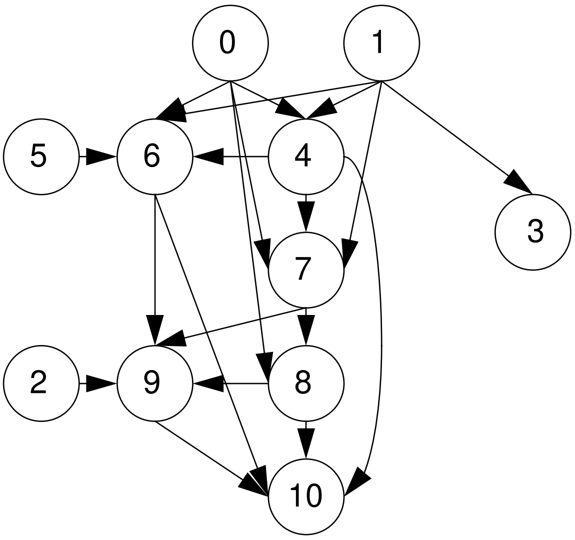


Figure 3.19: Network generated during a E=100,000 run

The shown architectures enhance the impression that a higher number of generations would result in even better networks. In case of the last architecture, a simple pruning of the connection from node 1 to 3 would help.

1-Bit Adder

In experiment 17 we evolve networks for the 1-bit adder . The evolution of the fitness values of the best individuals are plotted in figure 3.20. For each of the three different parameter settings for E and for the number of training processes, a typical run is displayed.

The results indicate that also for the 1-bit adder , progress towards improved architectures is smooth is consistent.

Experiment 17: 100 generations, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover -mutation rate 20. Varying number of epochs, varying number of evaluations, learning rate 0.8. 1-bit adder . Weight-based encoding: 13 total nodes, 1 index bit, mutation rate 20. Two-point-crossover .

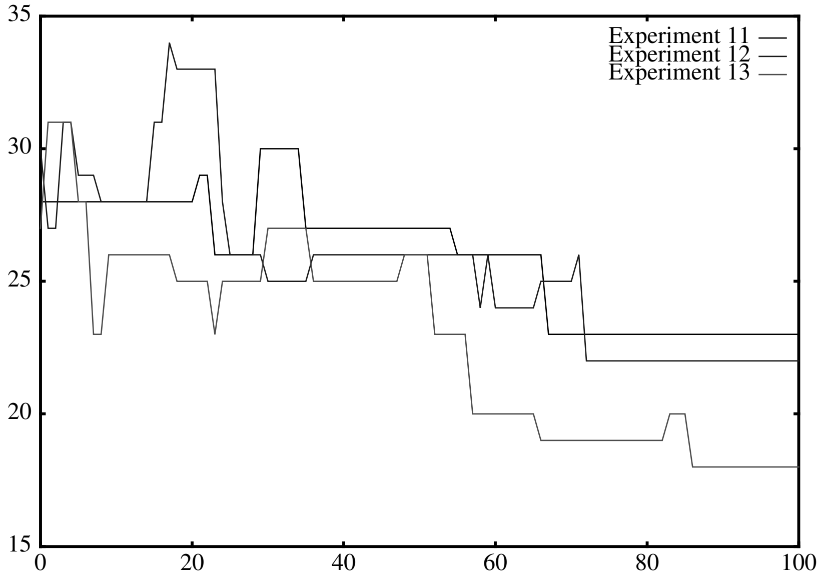


Figure 3.17: The Reduction of Network Size

The Architecture of Evolved Networks

Figure 3.18 and 3.19 give an impression of the topology of the evolved neural networks. Apparently , the GA reduced the number of effectively used nodes: In Figure 3.18, node 5 is not used anymore at all, and the nodes 2, 4, and 6 are reduced to a system of connected bias units. The core of the network is a layered 2-4-1 network with interior nodes 7, 8, 9 and 3.

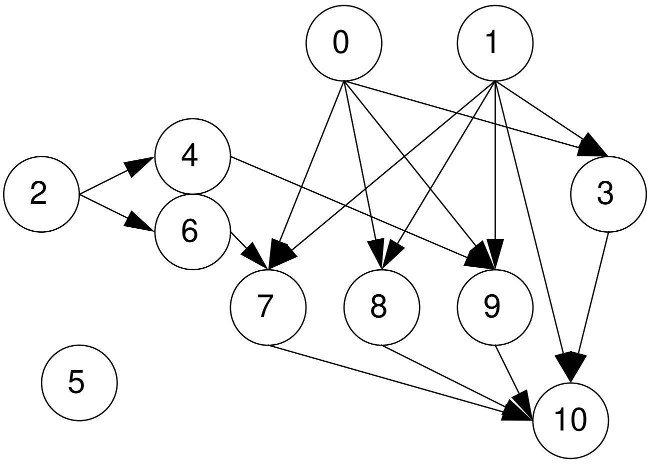


Figure 3.18: Network generated during a E=50,000 run

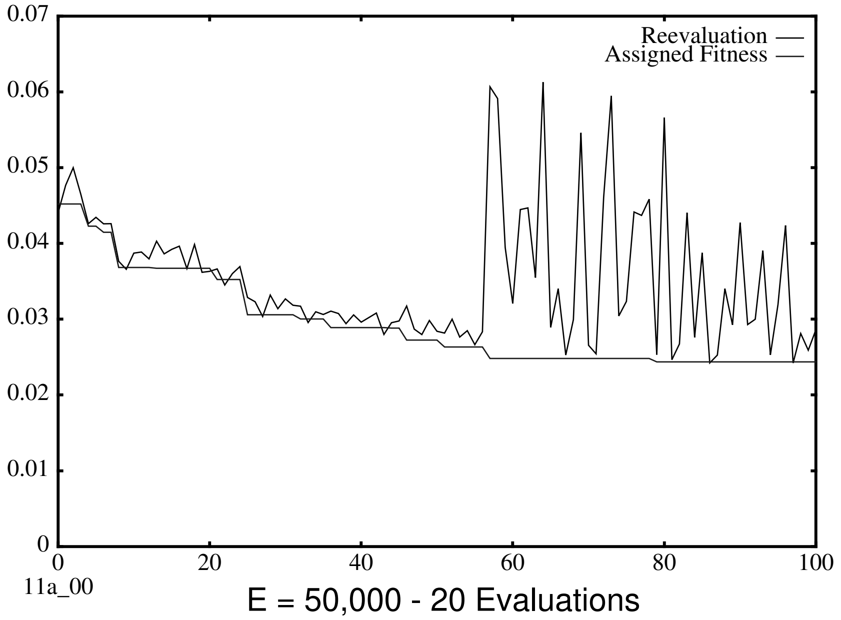


Figure 3.16: Problem of Bias towards "noisy" Architectures

Since the genetic algorithm only uses one of these evaluations - the first - an accidentally good static fitness evaluation will give such a "noisy" architecture a high ranking. Noisy architectures are a problem, since their quality can not be adequately assessed: Their fitness value is much more influenced by the random initial weights. This contradicts our goal to find good architectures independent from initial weights.

Size of the Networks

Randomly generated networks with 1 1 nodes have an average of 27 connections. However , the best standard architectures use a much lower number of nodes and connections. For instance, the 6-node fully connected network has 15 connections.

Figure 3.17 illustrates that the genetic algorithm is able to reduce the number of connections from the initial 27 down to around 20. The figure displays the network sizes of the same best individuals, whose fitness values are discussed in figure 3.14 and 3.15.

Table 17: Performance of Evolved and Standard Architectures

Architecture	E=20,000	E=100,000
Trained for E=20,000	0.07853	0.019189
Trained for E=50,000	0.093884	0.018650
Trained for E=100,000	0.218535	0.017688
6 node fully-connected	0.056107	0.017559
2-5-1 layered	0.097225	0.025024
3-4-1 incl. bias unit	0.091618	0.015099

Notice that the genetic algorithm is restricted to only 100 generations. The graphs of figure 3.14 and 3.15 suggest that a longer run of the experiments beyond 100 generations will resulted in even better architectures.

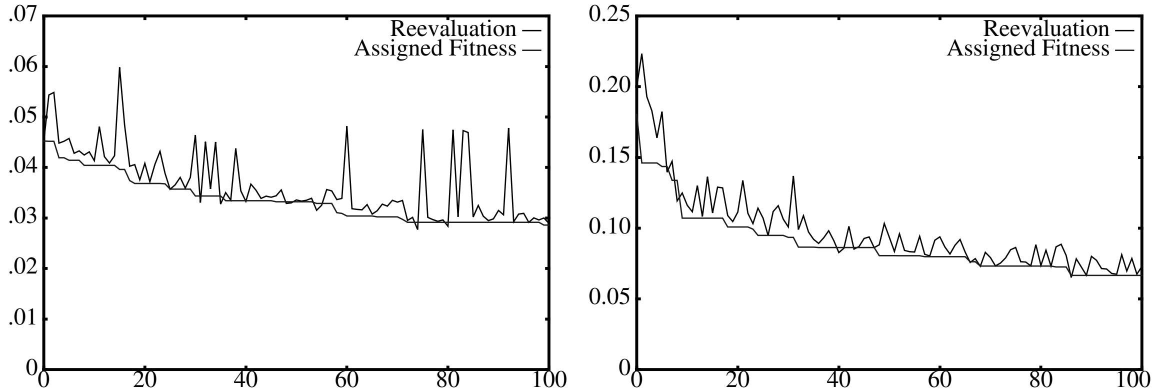
Bias towards "noisy" Architectures

Figure 3.16 illustrates how the problem of a "noisy" fitness function may significantly influence the performance of the GANN system. A run of experiment 12 is displayed. At generation 58 an architecture becomes the best individual, whose performance depends on a much larger scale on the initial weight setting. This results into reevaluation values between 0.0246553 and 0.061286.

total nodes, 1 index bit, mutation rate 20. Two-point-crossover.

Experiment 16: 100 generations, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover -mutation rate 20. Epochs = 20,000 divided by connection count, 50 evaluations, learning rate 0.8. XOR. Weight-based encoding: 1 1 total nodes, 1 index bit, mutation rate 20. Two-point-crossover.

The results are shown in Figure 3.15. The performance of the GANN system using this parameter setting is similar to experiment 14. However, especially in the setting with 50 evaluations, a much smoother and more consistent progress can be observed.



11a_01 E= 50,000 - 20 Evaluations

11b__10 E = 20,000 - 50 Evaluations

Figure 3.15: Variation of Training Length and Number of Evaluations

Table 17 compares the performance of the best architectures found by the genetic algorithm and the best standard architectures. The performance of the found architectures is comparable with best instances of standard topologies. In nearly all cases, a better fitness than the layered 2-5-1 architecture could be achieved

The results also illustrates that the networks were trained for a specific purpose: The net that was evolved for E=20,000 is the best net for E=20,000 among the evolved networks and beats also two standard architectures, but it ranks second to last for E=100,000. On the other hand, the net that was trained for E=100,000 is the overall third best net for this task, but shows poor performance for E=20,000.

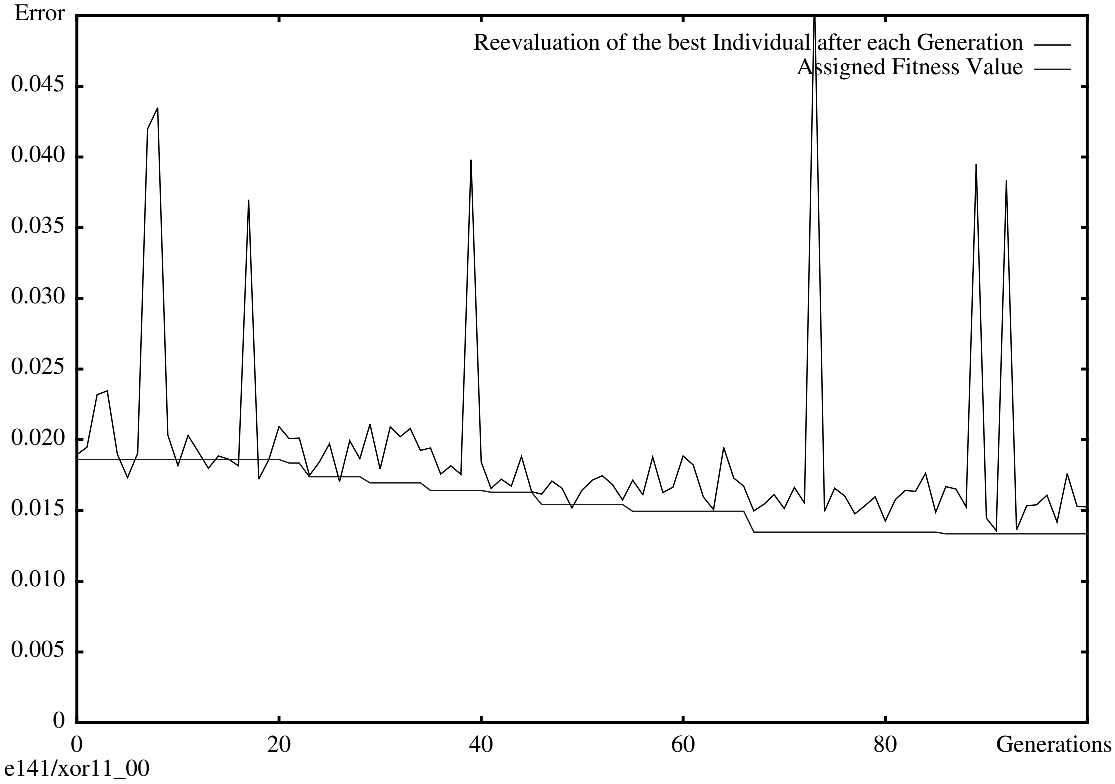


Figure 3.14: Fitness Evolution of Optimizing XOR-Architecture (E=100,000; 10 Evaluations)

So, the genetic algorithm was able to find an good architecture and made consistent progress in this task until it was stopped after 100 generations. The resulting network is comparable with the best of the standard architectures.

Impact of Training Length and Number of Evaluations

In experiment 15 and 16 we investigate, if the genetic algorithm can be improved by more accurate fitness evaluations. We train each generated architecture 20 times, and in the 13th experiment 50 times in order to get an adequate average error . On the other hand, the training length is decreased to E=50,000 and E=20,000. So, the run of these experiment variations takes about the same computation time as a run of experiment 14.

Experiment 15: 100 generations, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover -mutation rate 20. Epochs = 50,000 divided by connection count, 20 evaluations, learning rate 0.8. XOR. Weight-based encoding: 1 1 tion time on the mentioned machine.

Figure 3.14 displays a typical run of the experiment. It illustrates the impact of the noisy fitness function quite well. Two lines are displayed: The “noisy” line displays fitness evaluations of the best individual, that are computed at the end of each generation. The other line displays the fitness value of the best individual: This fitness value was generated after the creation of the individual and is used by the genetic algorithm.

Both lines show continuous progress towards a fast learning architecture: Beginning with a best individual with fitness 0.0186142, finally an fitness of 0.0133564 is reached. However, these good values do not neccessarily represent the quality of the architecture. They also profit from accidentally good random initial weights during the evaluations. A more accurate measurement is displayed in the “noisy” line. It represents the reevaluation of the best individual after each generation. Its average also improves from circa 0.02 to circa 0.015.

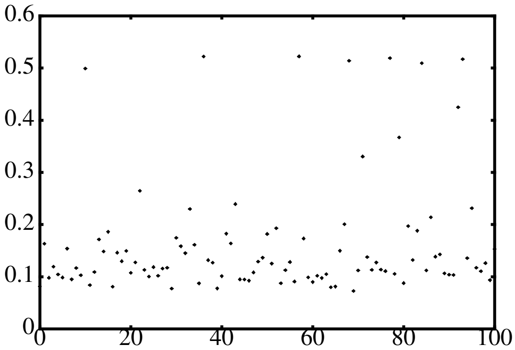


Figure 3.12: Error Sampling for 6-node (E =10,000)

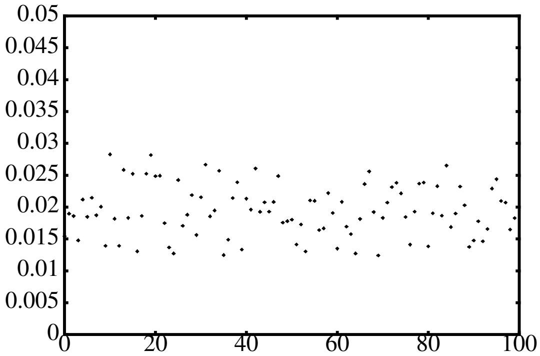


Figure 3.13: Error Sampling for 6-node (E=100,000)

These considerations urge for a high value of E and a high number of training processes. However, repeated long training processes are very time consuming. One training process for E=100,000 takes about 3 seconds computation time, when using the research tool on a SUN SPARC 5m70. For E=10,000, only 0.3 seconds are required.

Optimizing NN Architectures Using Weight-Based Encoding

In weight-based encoding of an architecture, each connection of a fully connected architecture is encoded by one bit: the index bit. If it is set, the connection exists, otherwise not. Each network can be represented by a string of these index bits. Thus, the original GA operators can be applied.

Since each weight has a fixed interpretation in a given fully connected architecture, we have to restrict the search initially to architectures with a maximum number of nodes.

Experiment 14: 100 generations, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover-mutation rate 20. Epochs = 100,000 divided by connection count, 10 evaluations, learning rate 0.8. XOR. Weight-based encoding: 1 1 total nodes, 1 index bit, mutation rate 20. Two-point-crossover.

In experiment 14 we investigate, how efficient the genetic algorithm finds a good network architecture using weight-based encoding. A run of the experiment with E=100,000 over 100 generations takes about ten hours computathat the success of training depends on the initial weight values. Since, we merely optimize the architecture, the weights are initially set to random values. So, if we repeat training evaluations with different initial weight, we get different results.

Figure 3.1 1 displays how much the results vary for the 5-node fully connected network. 100 samples of 1,1 11 epoch (E/cc = 10,000/9) training are shown. 7% of the cases are complete failures with a rms-error of about 0.5. Figure 3.12 shows the same picture for the 6-node network. For E = 100,000, the results vary merely between 0.01 and 0.03 (see figure 3.13).

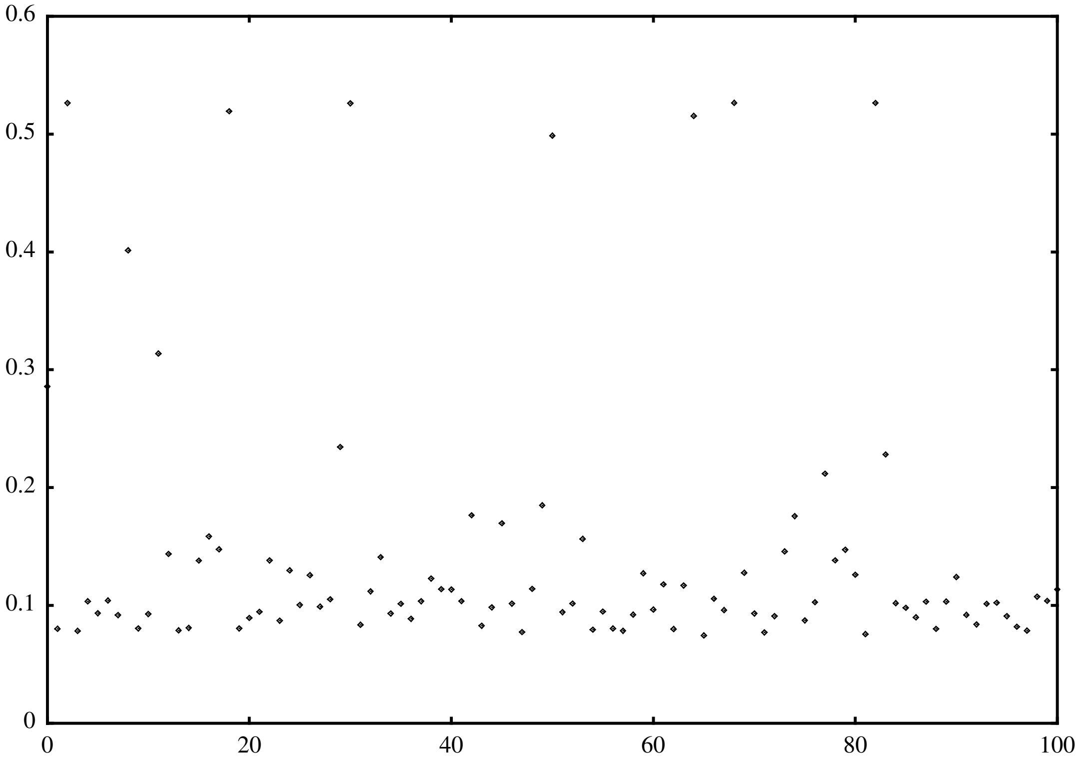


Figure 3.11: Evaluations after 10,000 E for 5-node Fully-Connected Net

The table illustrates the problems that come along with the fitness evaluation of architectures that are generated by the GA. In order to achieve the best possible evaluation, training over a high number of epochs has to be performed. Also, error evaluation based on random weights is a noisy fitness function. the more training processes performed, the more representative the resulting evaluation value.

Figure 3.10 displays some of the data of table 16: For the longest training process, the 3-4-1 layered ranks the best among the four displayed architectures. However , it is ranked second after E = 5,000 and third after E = 10,000 and 20,000. So, the short term convergence does not predict long term convergence, even within the same topology class. The 5-node fully connected network seems to superior to its 6-node competitor up to E = 20,000, until it is ultimately outperformed.

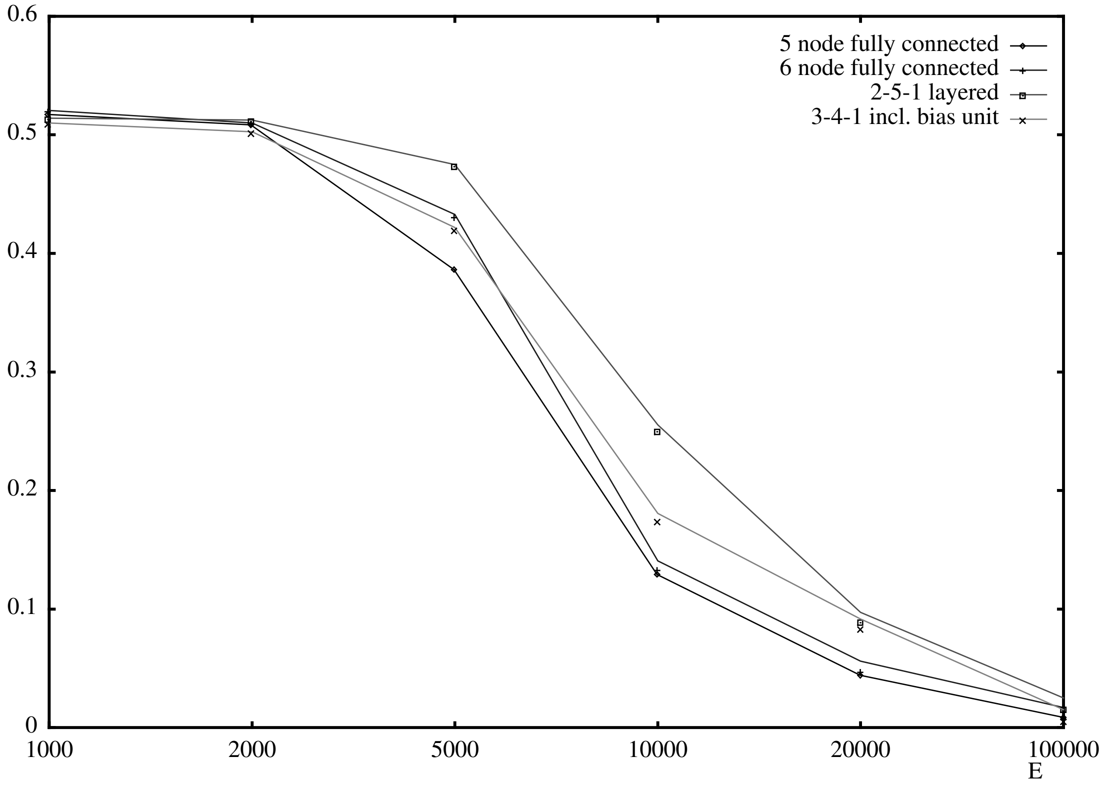


Figure 3.10: Error Development for Standard Architectures

The other problem for defining the fitness of an architecture lies in the fact

Table 16: Performance Evaluation of Various Network Architectures for XOR

E	1000	2000	5,000	10,000	20,000	100,000
4 node fully-c	0.515437	0.513334	0.484624	0.453512	0.451519	0.450719
5 node fully-c	0.518312	0.509715	0.389778	0.137291	0.053577	0.018675
6 node fully-c	0.520498	0.510036	0.432788	0.140613	0.056107	0.017559
7 node fully-c	0.525658	0.519585	0.408227	0.149738	0.060961	0.019101
8 node fully-c	0.529587	0.524147	0.466310	0.292121	0.085560	0.019935
9 node fully-c	0.534662	0.530759	0.510064	0.363000	0.131906	0.021986
10 node fully-c	0.536131	0.532083	0.507876	0.353022	0.110811	0.023429
11 node fully-c	0.542059	0.536737	0.514340	0.380702	0.152209	0.027108
2-2-1	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
2-3-1	0.503402	0.496261	0.426042	0.293187	0.258096	0.250959
2-4-1	0.515189	0.508591	0.471117	0.293429	0.132421	0.025543
2-5-1	0.514120	0.512444	0.474998	0.255315	0.097225	0.025024
2-6-1	0.521576	0.512424	0.459168	0.264913	0.102529	0.025856
2-7-1	0.528192	0.520914	0.487832	0.325130	0.102028	0.026940
2-8-1	0.526797	0.523003	0.490040	0.331040	0.146517	0.030287
3-3-1 incl. bias	0.507148	0.501188	0.411446	0.224329	0.191503	0.217216
3-4-1 incl. bias	0.509988	0.502421	0.421896	0.180641	0.091618	0.015099
3-5-1 incl. bias	0.517983	0.513984	0.481745	0.272044	0.108480	0.018168
3-6-1 incl. bias	0.524419	0.519634	0.480121	0.251478	0.073099	0.018310
3-7-1 incl. bias	0.524781	0.519920	0.499058	0.311401	0.079394	0.019666
3-8-1 incl. bias	0.526714	0.521591	0.496229	0.312837	0.113252	0.021580

3.4 Experiments on Architecture Optimization

The optimization of the architecture of neural networks is a completely different task from weight optimization. While for the later , the GANN system has to compete with back-propagation, no generally accepted strategy exists to find a good architecture for an arbitrary task.

Traditionally , a standard topology is selected and the number of nodes follow rules of thumb. Standard topologies are for instance layered networks with a small number of hidden layers or fully connected networks.

Optimal Topology for XOR

First, we examine how much the choice of architecture influences the training process, and how we can evaluate a certain architecture. We investigated the performance of different architectures for the XOR function. The performance is measured by the error of the network after training. The network is initialized to random weights. Since the training of large networks, the number of epochs depends on the number of connection of each network:

$$epochs = \frac{E}{connections}$$

Table 16 gives the evaluation of different architectures for the XOR problem. The displayed fitness is the average error of 10 evaluations. Three different topology classes are investigated: fully connected feed-forward networks, layered architectures and layered architectures with a bias unit in the input layer . A bias unit is a hidden node that has the constant activation 1. The results suggest that the 6-node fully connected network is the best competitor for the longest training period.

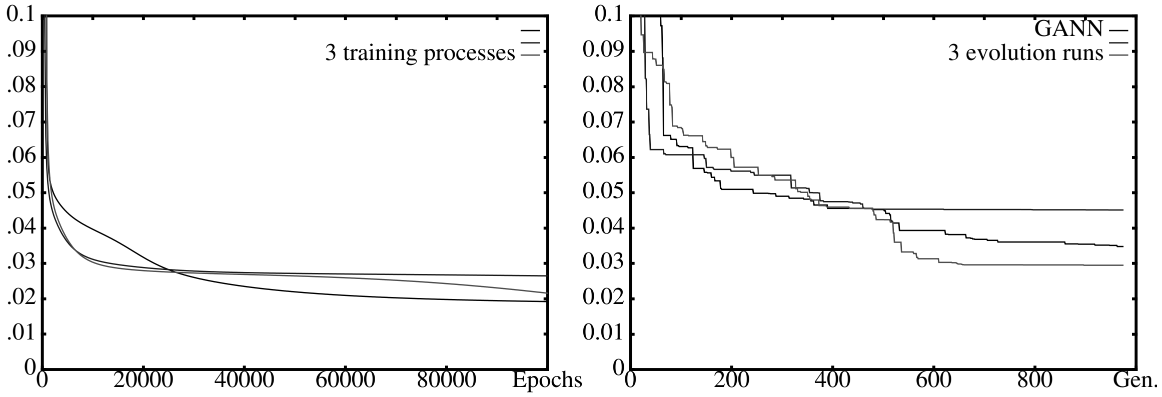


Figure 3.8: Backprop vs. GANN and the Gaussian Bell Curve

Experiment 13: 970 generations, 10 population with 30 individuals, distribution cycle 200, 5 mutations and 5 crossovers with crossover -mutation rate 20. 1 epoch, learning rate 0.8. 2-dimensional gaussian bell curve with 25 training patterns. Weight-based encoding: 7 total nodes, 7 bits/weight (no index), mutation rate 20. Varying types of crossover . Binary ands gray coding. Scaling factor 0.08.

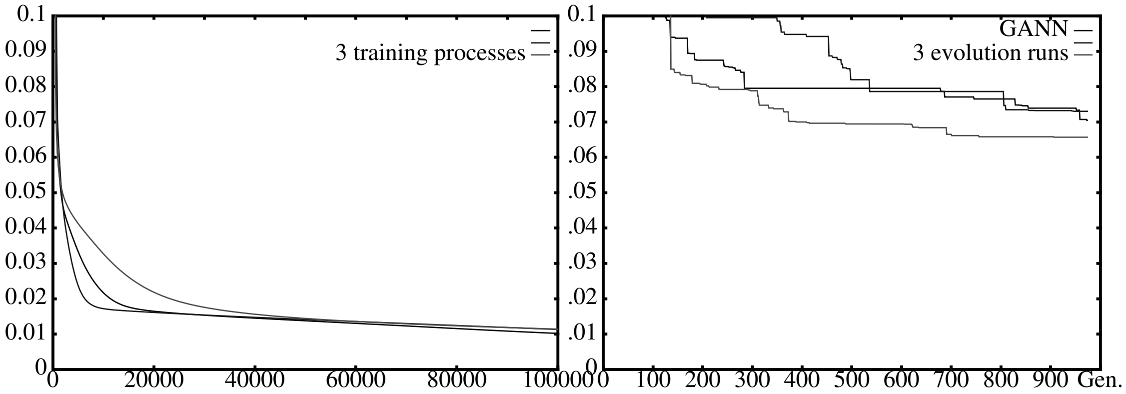


Figure 3.9: Backprop. vs GANN and the 2D Gaussian Bell Curve

Experiment 11: 970 generations, 10 population with 30 individuals, distribution cycle 200, 5 mutations and 5 crossovers with crossover -mutation rate 20. 1 epoch, learning rate 0.8. 4-period sine with 17 training patterns. Weight-based encoding: 7 total nodes, 7 bits/weight (no index), mutation rate 20. Varying types of crossover . Binary ands gray coding. Scaling factor 0.08.

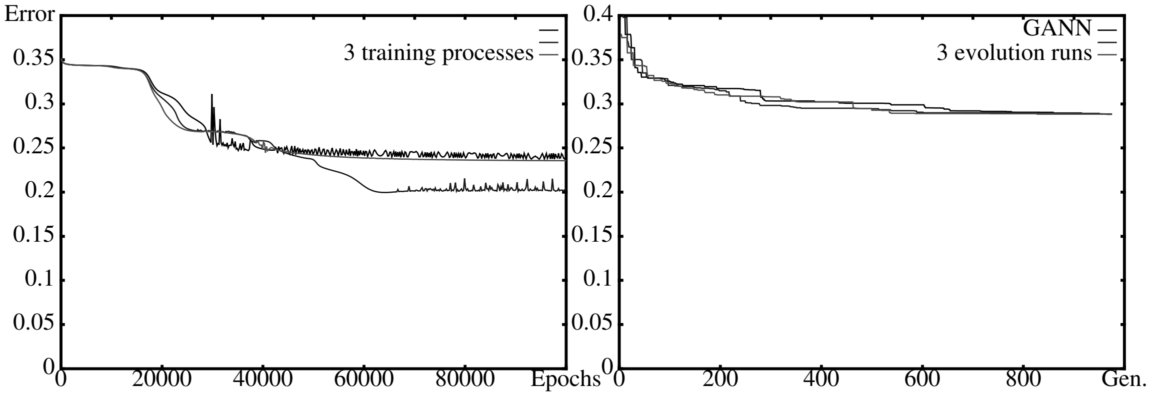


Figure 3.7: Backprop. vs GANN and the 4-Period Sine Function

For each experiment, a figure (3.7, 3.8, and 3.9) displays three runs of the experiment in its right graph. As a comparison, three training processes of backpropagation are displayed in the left half of the figure. The backpropagation algorithm ran for each training process over 100,000 epochs. The GANN system generated 100,000 individuals.

Although this is a quite crude comparison, we have to observe that backpropagation shows similar results to the genetic weight evolution.

Experiment 12: 970 generations, 10 population with 30 individuals, distribution cycle 200, 5 mutations and 5 crossovers with crossover -mutation rate 20. 1 epoch, learning rate 0.8. Gaussian bell curve with 17 training patterns. Weight-based encoding: 5 total nodes, 7 bits/weight (no index), mutation rate 20. Varying types of crossover . Binary ands gray coding. Scaling factor 0.08.

Crossover Type and Gray Coding

In Experiment 10 we investigate, if the three standard crossover operators differ in their performance on the task of evolving weights for the sine function. The best design of the population that was found in the experiment 9 is used: 10 subpopulations with 30 individuals each and distribution cycle of 200.

Experiment 10: 970 generations, 10 population with 30 individuals, distribution cycle 200, 5 mutations and 5 crossovers with crossover-mutation rate 20. 1 epoch, learning rate 0.8. Sine with 17 training patterns. Weight-based encoding: 5 total nodes, 7 bits/weight (no index), mutation rate 20. Varying types of crossover. Binary ands gray coding. Scaling factor 0.08.

The results are reported in table 15 in the usual fashion. Uniform crossover seems to be superior to one- and two-point crossover. However, the difference is not large and may be even in the range of measurement error. Also, no significant difference between gray and binary coding can be observed, although the results suggest that gray coding is a bad choice for two-point crossover.

Table 15: The Impact of Crossover Type and Gray vs. Binary Coding

	One-Point	Two-Point	Uniform
Binary	0.036879	0.036089	0.035413
Gray	0.034778	0.041122	0.034802

Application to Other Tasks

After optimizing all the parameters of the GANN system, we finally examine in experiment 11, 12 and 13, how the its performance compares to backpropagation. Three real-valued functions are examined: 4-period sinus, the Gaussian bell curve and a two-dimensional Gaussian bell curve. The functions are trained to a fully connected 7-node, 5-node and 7-node architecture respectively.

the error of the measurement. However, population sizes smaller than 30 rank certainly last.

Table 13: The Impact of Generation Size

10	20	30	50	100	200
0.079219	0.077692	0.060396	0.069216	0.048445	0.059918

Distributed Genetic Algorithm

In experiment 9 we examine, if the use of a distributed genetic algorithm improves the evolution of good weight settings. Distributed GA were introduced in [Tanese, 1989], and are discussed in section 1.3 of this thesis. In the experiment, 10 subpopulations with 30 individuals each are used. Again, 100,000 individuals are generated during one run of the experiment.

In three different versions of the experiment, the distribution cycle is varied. The distribution cycle is the number of generations, after which the best individuals of each generation are exchanged.

Experiment 9: 970 generations, 10 population with 30 individuals, 5 mutations and 5 crossovers with crossover-mutation rate 20. 1 epoch, learning rate 0.8. Sine with 17 training patterns. Weight-based encoding: 5 total nodes, 7 bits/weight (no index), mutation rate 20. Two-point-crossover. Binary coding. Scaling factor 0.08.

Table 14 presents the results of the three different versions of the experiment in the usual fashion - the fitness value 10th best run out of 20 is displayed. Three different distribution cycles are used. The results indicate that the distributed GA shows superior performance compared to the standard GA. Also, a high distribution cycle is the best choice.

Table 14: Distributed GA

Cycle 10	Cycle 50	Cycle 200
0.045303	0.039761	0.038429

Table 12: Quality of n-Bit Encoding Using Optimal Range

3-node range 20	5-node range 20	7-node range 19.2	9-node range 20.48
0.083744	0.064960	0.0562101	0.053783

The more encoding bits are used, the better the results. A higher number of encoding bits slightly increases the computation time: The computation time for a single run of the experiments varies from 9.7 minutes for 3-bit encoding to 11.2 minutes for 9-bit encoding on a SUN IPX. A much higher number of encoding bits, however, would result in a relatively stronger slowdown, since different data types than 16-bit integers for the weight storage have to be used.

Population Size

In all our experiments so far, a population size of 30 has been used. Since in each generations 10 GA operators were applied to the individuals, up to 10 new individuals replaced the worst individuals, while at least two thirds of old individuals remained in the population.

In experiment 8, we examine how different population sizes effect the performance of the GANN system. To achieve comparable results the number of generations was adapted to the population size, so that always 100,000 individuals were generated.

Experiment 8: Varying number of generations, 1 population with varying number of individuals, 5 mutations and 5 crossovers with crossover-mutation rate 20. 1 epoch, learning rate 0.8. Sine with 17 training patterns. Weight-based encoding: 5 total nodes, 7 bits/weight (no index), mutation rate 20. Two-point-crossover. Binary coding. Scaling factor 0.08.

Table 13 reports the results. Again, the rms error of the median (10th best run out of 20) is displayed. Clearly, a larger population yields to better performance: the empirical data favors the size 100, although this may be based on

Clearly , both the number of encoding bits and the range of weights have high impact on the efficiency of the GANN system. The influence of the scaling factor on the performance of the neural network depends on number of bits: For the 3-bit encoding the factor 1 worked the best, whereas for the 9-bit encoding the factor 0.1 resulted in the lowest terminal error .

To localize the optimal scaling factor , we ran additional variations of experiment 7 for the 7-bit and 9-bit encoding schemes. The results are shown in Table 10 and 11, only the median (the 10th best out of 20 runs) is displayed.

Table 10: Search for the Best Weight Range Using 7 Encoding Bits

factor 0.20 max. 12.8	factor 0.25 max. 16.0	factor 0.30 max. 19.2	factor 0.40 max. 25.6	factor 0.50 max. 32.0
0.072611	0.068293	0.0562101	0.064460	0.088252

The results are quite interesting. Although a different optimal scaling factor for both cases has been found - 0.3 for 7-bit and 0.08 for 9-bit - the actual optimal range of the weights is quite similar . The GANN system seems to find the best networks, if the weights are limited to values in the interval (-20,20).

Table 11: Search for the Best Weight Range Using 9 Encoding Bits

fact. 0.05 rng. 12.80	fact. 0.06 rng. 15.36	fact. 0.07 rng. 17.92	fact. 0.08 rng. 20.48	fact. 0.10 rng. 2.560	fact. 0.15 rng. 38.40	fact. 0.20 rng. 51.20
0.072213	0.075602	0.060600	0.053783	0.070743	0.084441	0.071368

Using this range for 3-bit and 5-bit encoding (using scaling factors of 4 and 1.25 respectively) enhances the impression that this is the optimal range, independent from the number of encoding bits: Also here, better results than with other scaling factors can be achieved. Table 12 presents the terminal error of the median run in all four encoding cases.

Table 9: Impact of Encoding Bits and Range of Weights

	3 bits (-4, +4)	5 bits (-16, +16)	7 bits (-64, +64)	9 bits (-256, +256)
factor 0.1	0.313010	0.227216	0.084543	0.032426
	0.313010	0.227216	0.087498	0.036018
	0.313010	0.227216	0.087849	0.046769
	0.313010	0.227256	0.089561	0.049013
	0.313010	0.227256	0.101269	0.049217
	0.313010	0.232908	0.114790	0.055787
	0.313010	0.232949	0.115402	0.057749
	0.313010	0.232949	0.118045	0.058833
	0.313213	0.232949	0.119068	0.066740
	0.313213	**0.233084**	**0.119340**	**0.070743**
	0.313213	0.234149	0.121218	0.073356
	0.313264	0.234178	0.122774	0.075433
	0.313264	0.234178	0.124372	0.082707
	0.313365	0.234178	0.124788	0.126809
	0.313365	0.234184	0.126510	0.136825
	0.313365	0.234211	0.137003	0.138920
	0.313365	0.234226	0.141770	0.146683
	0.313365	0.234226	0.148802	0.147172
	0.313365	0.234298	0.155459	0.148235
	0.313365	0.234666	0.156998	0.150765
factor 0.03	0.330737	0.307095	0.215711	0.0727724
	0.330737	0.307095	0.215711	0.0739605
	0.330737	0.307095	0.215713	0.0761268
	0.330737	0.307095	0.215721	0.0780194
	0.330737	0.307095	0.215725	0.0835465
	0.330737	0.307095	0.215938	0.0989931
	0.330737	0.307095	0.22262	0.0996362
	0.330737	0.307095	0.222724	0.0996622
	0.330737	0.307249	0.222724	0.0999415
	0.330737	**0.307249**	**0.222725**	**0.102609**
	0.330737	0.307249	0.222729	0.104635
	0.330737	0.307249	0.222729	0.105248
	0.330737	0.307249	0.222739	0.108213
	0.330737	0.307249	0.222744	0.108351
	0.330737	0.307249	0.222765	0.108464
	0.330737	0.307251	0.222796	0.128752
	0.330737	0.307404	0.222944	0.134128
	0.330737	0.307649	0.222996	0.143609
	0.330737	0.307649	0.222996	0.154041
	0.330737	0.307649	0.223004	0.165353

Table 9: Impact of Encoding Bits and Range of Weights

	3 bits (-4, +4)	5 bits (-16, +16)	7 bits (-64, +64)	9 bits (-256, +256)
factor 1.0	0.153030	0.044309	0.052438	0.0661776
	0.153030	0.045748	0.067474	0.101586
	0.153030	0.045926	0.069318	0.133243
	0.165535	0.046576	0.073104	0.135788
	0.167299	0.049201	0.076985	0.138132
	0.167299	0.059204	0.107452	0.150616
	0.167299	0.059749	0.116326	0.150978
	0.167299	0.059842	0.121792	0.151314
	0.167299	0.060958	0.136346	0.151638
	0.167299	**0.071386**	**0.136933**	**0.157729**
	0.167299	0.071605	0.139090	0.159126
	0.181069	0.079629	0.139795	0.170319
	0.181069	0.093596	0.149087	0.174345
	0.181457	0.094930	0.149235	0.183583
	0.181457	0.108731	0.151369	0.187701
	0.181734	0.109770	0.151847	0.189298
	0.184193	0.115010	0.155842	0.192337
	0.192114	0.126416	0.156825	0.201894
	0.192114	0.146723	0.159297	0.206178
	0.203813	0.150069	0.165454	0.227227
factor 0.3	0.248386	0.118632	0.0295233	0.0498435
	0.248386	0.118632	0.0316353	0.0704057
	0.248386	0.118865	0.0334778	0.0714226
	0.248386	0.119028	0.0368493	0.0716892
	0.248386	0.119028	0.040791	0.0766553
	0.253577	0.120198	0.0488674	0.0917237
	0.253577	0.120619	0.050332	0.106842
	0.253577	0.124684	0.0535794	0.119931
	0.253577	0.145036	0.0539525	0.130911
	0.253577	**0.145036**	**0.0562101**	**0.144254**
	0.253577	0.145088	0.0672379	0.146182
	0.253577	0.145185	0.0679378	0.148769
	0.254062	0.160298	0.0688299	0.149224
	0.254062	0.160930	0.0715112	0.150575
	0.255607	0.163454	0.0799214	0.152315
	0.255607	0.165005	0.0885259	0.15547
	0.255607	0.166743	0.0974661	0.161709
	0.255607	0.175946	0.0989347	0.169699
	0.255607	0.185054	0.103806	0.1777
	0.255607	0.185305	0.144998	0.191477

feedback, since it merely operated on this kind of fitness information.

Hence, for the rest of this section, we will investigate, how the evolution of weight settings for real-valued functions can be improved. The sine function serves as the chosen prototypical example. We will investigate the influence of the GA parameters on the performance of the GANN in the task of evolving weight settings for the this function.

Number of Encoding Bits and Range of Weights

So far, we always used 7 bits for the encoding of each weight, which resulted in relatively high weight values between -64 and +64, without 0. This proved to be most efficient for boolean functions such as XOR and the n-bit adder. Real valued functions, however, require fine-tuned output values, hence fine-tuned weights. In case of boolean functions, the desired output values are extremes of the sigmoid threshold function, which has the properties

$$\lim_{x \to \infty} \sigma(x) = 1$$

$$\lim_{x \to -\infty} \sigma(x) = 0$$

So, high weight values support boolean output.

In experiment 7 we search for the optimal number of encoding bits and the optimal weight range for the real valued function sine by varying these parameters. The results of 20 runs for each constellation are reported in table 9: the fitness values after 10,000 generations are shown, the median (10th best) is printed in bold type.

Experiment 7: 10000 generations, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover-mutation rate 20. 1 epoch, learning rate 0.8. Sine with 17 training patterns. Weight-based encoding: 5 total nodes, varying number total bits/weight (no index), mutation rate 20. Two-point-crossover. Binary coding. Varying scaling factor.

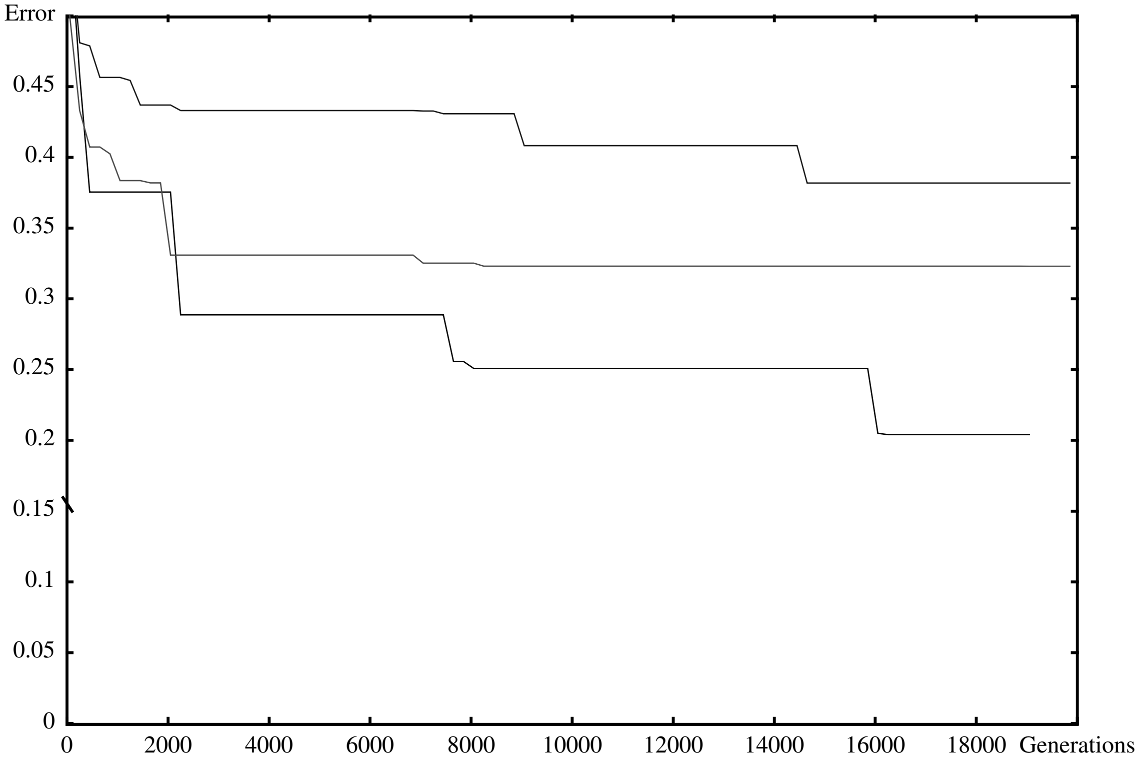


Figure 3.6: GANN and the 2-Bit Adder

This finding is consistent with reports of other research [Lindgren, 1992]. The training of NN connection weights with GA is generally feasible only for small networks. D. Whitley suggests in [Whitley , 1993] to apply GANN systems to control problems that rely on reinforcement learning. Since no target output is given, back-propagation cannot be used as a training method.

In general, control problems demand the training of a real-valued function. In case of the pole-balancing problem [W ieland, 1991], the input values represent state information such as angle and velocity of the pole. The output values represent action information such as the force that has to be applied to the pole to keep it in balance.

So, basically we can model each reinforcement control task with the training of a real valued function. The feedback we receive from the problem is merely the quality of the solution, not the optimal action as requested by back-propagation. But the GANN system can easily deal with just such reduced

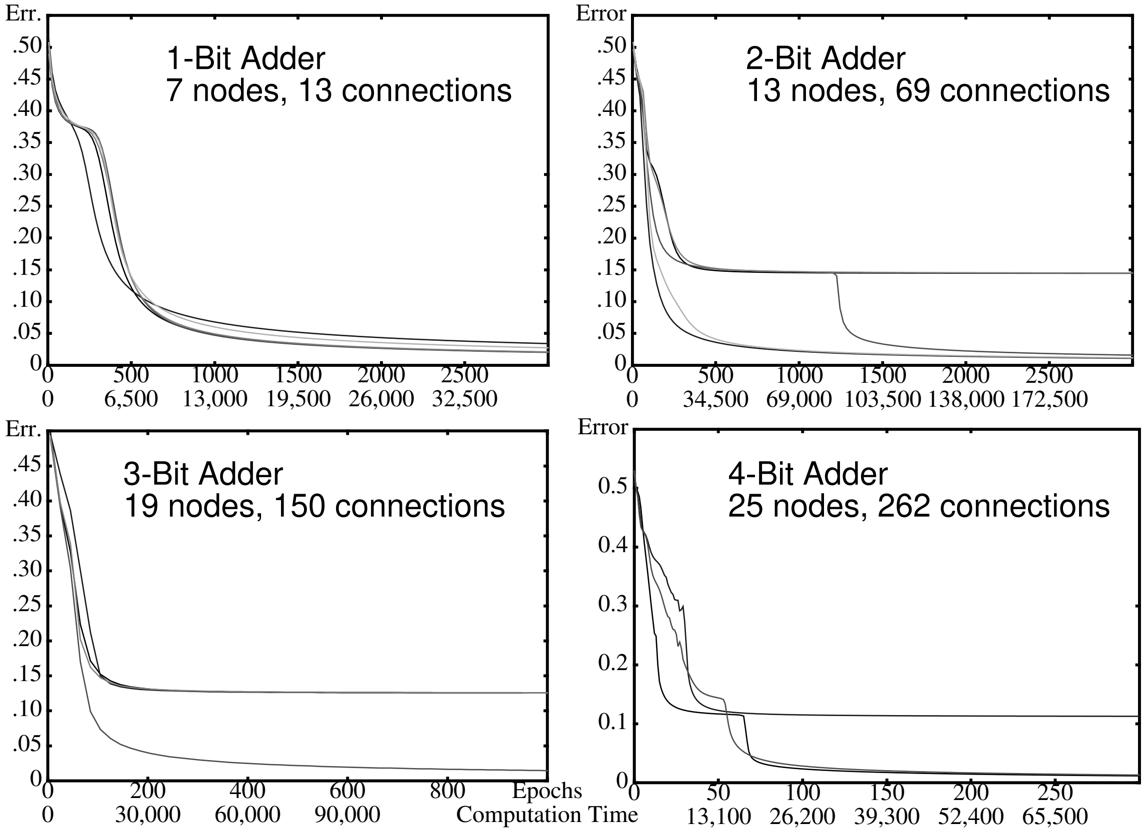


Figure 3.5: Backpropagation and the n-Bit-Adder

However, the GANN system has many more problems with relatively complex versions of the *n* bit adder. Figure 3.6 displays the evolution of the fitness value of the best individuals in three runs of experiment 6.

> **Experiment 6:** 20000 generations, target fitness 0.01, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover-mutation rate 20. 1 epoch, target error 0.01, learning rate 0.8. 2-Bit Adder. Weight-based encoding: 13 total nodes, 7 total bits/weight (no index), mutation rate 20. Two-point-crossover. Binary coding.

It illustrates that already the 2-bit adder is a task beyond the capabilities of our system. Even with a large number of generations, the systems can not come close to a weight setting that would result in an acceptable error below 0.05. Other parameter settings can not significantly improve the genetic algorithms, as additional unreported experiments suggest.

So, we have to conclude that neural network training during the run of the genetic algorithm does not pay off. In contrary , it slows down the evolution process dramatically: In the case of the research tool used, including training over 100 training cycles increased the computation time by 3000 percent.

Thus, in the following experiments, back-propagation training is not used anymore.

Scalability

In the previous experiments we have examined very small problems with up to 2 input and 2 output nodes (1-bit adder), and up to 17 training patterns (sine). The results were quite promising: the genetic algorithm was able improve the weight values for the neural network much faster than back-propagation.

In order to investigate the scalability of the GANN system, we examine different complexities of the adder problem. For an n bit adder , a fully connected network with $6n + 1$ nodes, i.e. 3 n hidden nodes was used. Figure 3.5 illustrates, how back-propagation deals with the first four instances of the problem.

The graphs display the error against the number of epochs and computation time, which is measured in terms of the number of epochs times number of connections to be trained. The multiple runs of back-propagation suggest that with increasing problem size, the chance of getting stuck in a local minimum increases, which is in these cases an error of about 0.1 to 0.15. But on the other hand, the convergence speed of successful runs remains about the same: After about 30,000 time units, a fairly good weight setting is reached.

Table 8: The Effect of Different Number of NN Epochs (1-Bit Adder)

1	10	20	50	100	200	500
120	190	100	220	170	250	100
140	250	160	250	200	260	120
210	260	210	300	210	280	170
250	270	210	340	250	310	180
290	270	220	370	270	340	200
300	280	230	380	300	360	200
310	280	270	430	300	380	230
340	310	280	440	330	410	240
360	320	360	440	330	410	250
360	320	370	460	400	440	270
400	370	380	470	410	440	270
410	370	430	470	410	470	280
410	380	450	480	460	480	280
440	390	450	490	470	490	320
460	420	450	540	490	570	330
470	440	460	560	490	590	330
470	450	470	570	580	600	340
470	450	490	570	590	620	350
530	510	490	570	670	680	350
580	520	530	580	670	690	390
600	530	570	580	710	690	390
610	540	610	610	730	690	400
620	600	660	620	740	700	400
630	600	670	630	760	720	470
630	**710**	**690**	**640**	**840**	**780**	**470**
650	710	690	690	850	790	480
670	800	690	710	960	800	490
870	830	720	750	1020	810	510
1000	870	730	790	1180	870	520
1000	970	790	850	1180	910	580
1030	970	900	860	1250	930	600
1030	1090	960	1050	1290	950	650
1030	1120	980	1090	1390	1150	660
1100	1230	990	1110	1570	1210	740
1150	1330	1000	1200	1580	1250	840
1180	1630	1070	1230	1630	1520	880
1340	1740	1470	1760	1830	1840	920
1840	1970	1850	2260	2320	1900	1000
2630	2150	2450	2320	2720	2250	1150
3450	2160	2510	2940	3530	2470	1180
5890	2280	2560	4630	5130	2600	1180
6260	2320	2650	5790	5160	2720	1230
8010	3190	2700	5980	5400	2730	1460
11080	3660	2930	6390	5560	3580	1490
15260	6170	4720	9930	5760	5310	1650
16910	8570	7560	12140	13750	10750	1750
20030	11370	10030	18050	15940	20030	2400
20030	20030	20030	19170	20030	20030	6400
20030	20030	20030	20030	20030	20030	10820
99630	20030	20030	20030	20030	20030	20030

Table 7: The Effect of Different Numbers of NN Epochs (XOR)

1	10	20	50	100	200	500
50	30	80	50	90	40	50
60	40	90	60	110	50	90
80	70	100	60	120	60	110
90	70	120	80	140	70	110
90	90	120	150	150	90	120
110	90	130	160	160	100	140
130	100	130	160	180	120	160
150	110	130	170	190	120	160
150	120	140	180	200	120	180
180	120	150	180	200	140	190
200	150	160	180	210	180	200
200	150	180	200	220	190	210
230	150	200	200	230	210	220
230	180	220	230	230	230	230
240	180	250	230	250	230	250
240	180	250	250	250	250	260
250	220	260	270	260	250	290
250	220	270	270	260	260	320
260	230	270	290	270	270	330
270	240	290	290	270	300	340
290	240	290	300	280	310	340
290	260	290	300	280	310	350
300	270	310	320	300	320	370
320	270	320	340	300	340	380
380	**280**	**320**	**340**	**310**	**340**	**390**
380	290	380	350	320	350	390
390	380	390	360	360	360	410
390	390	400	360	370	370	410
520	390	450	380	390	400	440
520	400	470	380	390	420	470
520	420	520	420	390	440	470
560	420	550	420	410	450	480
630	430	680	430	410	450	480
640	480	690	440	410	450	500
660	500	740	450	420	480	570
720	510	790	530	420	490	630
750	510	1070	560	440	510	670
810	520	1070	580	490	510	760
1030	560	1080	850	670	570	850
1050	570	1310	900	710	810	980
1420	810	1660	980	850	830	1100
1840	920	1800	1030	900	850	1260
2370	1010	2110	1660	940	880	1330
2590	1200	2170	1870	1790	1390	1390
3250	1340	2180	2060	2070	1940	1440
6610	1520	3250	2100	5080	2850	1440
8090	1610	3270	2150	8670	2930	1650
9030	2150	3640	2290	9030	3200	1980
9030	9030	4970	2810	9030	4580	2280
9030	9030	9030	2920	9030	9030	2460

The results suggest that the relation between the number of mutation and crossover operators does not really matter, except that both, especially mutation, are required for an optimal use of the GANN system. Also, the choice between gray or binary coding seems to be somewhat irrelevant.

Baldwin Effect

In all of our experiments to this point, the networks generated by the genetic algorithm were trained for 100 epochs prior to fitness evaluation. This is only a reasonable strategy, if the Baldwin effect (see section 2.1) improves the genetic search considerably. Training for 100 epochs takes about 95% of the computation time of the GANN system, which has to be justified.

Experiment 4 and 5 investigate the Baldwin effect for the XOR function and the 1-Bit Adder. The number of training epochs was varied from 1 to 500 in both cases.

> **Experiment 4:** 900 generations, target fitness 0.01, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover-mutation rate 20. Varying number of epochs, target error 0.01, learning rate 0.8. XOR. Weight-based encoding: 5 total nodes, 7 total bits/weight (no index), mutation rate 20. Two-point-crossover. Binary coding.

> **Experiment 5:** 2000 generations, target fitness 0.01, 1 population with 30 individuals, 5 mutations and 5 crossovers with crossover-mutation rate 20. Varying number of epochs, target error 0.01, learning rate 0.8. 1-Bit Adder. Weight-based encoding: 7 total nodes, 7 total bits/weight (no index), mutation rate 20. Two-point-crossover. Binary coding.

For XOR, no significant difference can be observed for using different number of epochs. The median is in all cases between 280 and 390. The same observation can be made for the 1-Bit Adder: The data does not suggest any significant impact of the Baldwin effect for this task, although the median number of generations is unusually high for the experimental setting with 500 epochs.

Table 6: XOR with Gray Coding

10 mut	9 m 1 cro	8 m 2 cro	7 m 3 cro	6 m 4 cro	5 m 5 cro	4 m 6 cro	3 m 7 cro	2 m 8 cro	1 m 9 cro	10 cro
30	50	30	30	30	30	30	30	30	30	30
30	70	30	30	90	30	50	50	50	50	50
130	70	30	70	140	30	50	70	80	50	60
160	90	30	80	140	30	90	70	90	60	90
180	90	80	110	150	50	110	70	90	60	110
190	90	90	150	150	50	120	80	120	70	120
210	110	130	160	160	60	120	90	120	90	120
260	120	140	170	170	80	140	130	120	100	120
280	130	140	190	190	90	140	140	130	100	130
380	130	140	210	190	100	160	150	140	130	220
520	150	160	210	200	110	160	150	170	140	240
540	150	180	210	200	110	170	160	190	140	340
560	160	190	210	200	130	180	160	190	150	360
590	180	230	230	210	160	200	200	240	170	360
640	190	250	240	230	190	210	200	250	170	410
780	200	260	260	240	200	220	220	250	200	440
780	220	270	260	270	200	250	220	260	200	460
800	240	300	280	290	210	250	240	260	240	500
860	250	310	290	300	210	260	240	280	250	570
880	260	340	300	310	220	290	250	290	260	660
900	310	350	310	340	230	320	260	310	260	800
910	320	360	310	360	250	330	260	360	270	960
950	320	370	310	380	250	350	260	370	270	990
990	340	380	310	390	270	360	310	380	280	1640
1000	**360**	**400**	**320**	**390**	**280**	**360**	**310**	**490**	**290**	**1680**
1060	400	410	330	390	290	380	320	520	330	1870
1060	430	420	350	390	290	380	330	540	340	1920
1080	440	440	380	400	300	390	330	550	350	2080
1090	480	440	400	400	310	410	340	590	360	2140
1120	500	440	410	440	310	420	340	630	370	3030
8240	580	450	420	450	320	450	380	670	380	3130
9030	590	470	430	450	320	470	390	670	390	3480
9030	620	470	440	460	330	550	420	690	410	3640
9030	640	480	460	500	330	560	450	710	410	4180
9030	650	480	460	500	360	590	480	780	470	4230
9030	650	510	470	510	370	820	550	850	470	4970
9030	670	520	540	540	370	960	580	970	560	5180
9030	700	530	560	580	380	1450	600	1070	560	5880
9030	720	540	600	630	420	1480	620	1080	560	6540
9030	730	580	600	630	430	1710	640	1090	610	6740
9030	760	590	620	640	440	1830	680	1190	700	6900
9030	770	630	620	780	440	2040	930	1210	730	7270
9030	800	670	660	860	540	2130	960	1220	1160	9030
9030	880	680	880	960	560	2240	1320	1240	1340	9030
9030	900	700	1000	990	630	2540	1340	2090	1370	9030
9030	930	750	1100	1250	670	7070	2210	2180	1670	9030
9030	930	830	1430	1600	730	9030	3030	2970	1770	9030
9030	980	900	1550	2440	770	9030	3700	3060	6580	9030
9030	1680	920	3470	5710	2570	9030	4860	6600	9030	9030
9030	2050	5440	4120	9030	5520	9030	9030	9030	9030	9030

Table 5: XOR with Binary Coding

10 mut	9 m 1 cro	8 m 2 cro	7 m 3 cro	6 m 4 cro	5 m 5 cro	4 m 6 cro	3 m 7 cro	2 m 8 cro	1 m 9 cro	10 cro
30	30	30	40	30	90	30	30	70	30	50
100	40	50	60	70	110	30	40	70	30	50
180	40	60	80	90	120	70	60	70	40	60
190	50	90	80	90	140	70	80	90	40	80
210	120	110	100	100	150	70	110	100	80	90
210	130	130	110	120	160	80	110	110	90	100
220	130	180	120	160	180	100	110	110	120	100
220	170	190	120	170	190	130	120	120	140	110
240	190	190	160	170	200	150	130	130	140	110
270	210	190	170	170	200	160	130	130	160	120
310	220	220	190	190	210	160	130	140	190	140
320	230	230	190	210	220	180	140	190	210	160
320	230	240	200	210	230	190	160	200	220	160
330	250	240	200	220	230	200	160	210	230	240
350	250	240	210	230	250	200	180	220	230	290
350	290	250	220	240	250	210	190	260	250	340
380	290	250	230	250	260	220	190	270	270	400
380	300	260	280	250	260	220	200	270	320	460
390	310	270	280	260	270	220	200	330	320	490
390	320	280	290	270	270	230	240	330	320	500
400	320	290	300	280	280	250	260	350	430	510
430	390	300	310	280	280	310	260	360	450	520
440	440	320	320	290	300	310	260	360	510	620
490	440	330	330	290	300	320	280	360	540	650
540	**490**	**350**	**340**	**310**	**310**	**330**	**300**	**420**	**640**	**670**
540	530	360	380	310	320	340	310	510	680	800
620	570	390	390	330	360	350	310	510	710	810
640	570	420	420	340	370	350	330	530	750	830
690	580	440	420	350	390	400	350	540	830	990
710	630	470	460	350	390	410	370	540	950	1100
720	640	480	460	370	390	480	400	620	1020	1200
730	650	480	480	380	410	500	410	620	1110	1480
750	660	490	500	430	410	520	450	790	1130	1510
850	670	490	520	530	410	570	460	790	1140	1640
880	690	510	560	610	420	590	520	910	1180	1790
900	740	540	560	620	420	630	520	910	1230	2980
1040	810	580	610	660	440	750	530	1250	1360	3080
1050	820	580	650	680	490	760	570	1430	1420	4490
1080	820	580	670	730	670	770	620	1430	1460	4580
1100	900	610	720	730	710	790	730	1560	1630	5280
1140	900	660	800	750	850	1090	940	2080	1890	5460
1170	950	680	840	770	900	1500	1000	2080	1940	5640
1230	1000	700	850	780	940	1810	1190	2230	3230	6010
1240	1080	700	880	860	1790	2010	1350	3970	3440	7180
1260	1180	800	940	940	2070	2070	1660	3970	5870	7700
1290	1440	940	970	990	5080	2120	1710	4400	8770	8780
1360	1440	1040	1010	1100	8670	2480	2660	6780	9030	9030
1750	1500	1170	1020	2330	9030	4190	3610	9030	9030	9030
2170	1860	4950	1020	2400	9030	4750	9030	9030	9030	9030
2800	6100	6460	1130	5180	9030	4810	9030	9030	9030	9030

Table 4: Fitness after 900 Generations for Sine

10 mutations	5 mutations, 5 crossovers (cro-mut-rate 2)	5 mutations 5 crossovers (cro-mut-rate 20)	10 crossovers (cro-mut-rate 20)
0.0262927	0.0262347	0.0241474	0.0309672
0.0343965	0.0335057	0.0273759	0.0342695
0.0362376	0.0338501	0.0285026	0.0405706
0.0369155	0.0345520	0.0314216	0.0433516
0.0374821	0.0346752	0.0327999	0.0461073
0.0386209	0.0375245	0.0366941	0.0473017
0.0396562	0.0379173	0.0371354	0.0477327
0.0413063	0.0396147	0.0386946	0.0495430
0.0412722	0.0409227	0.0401896	0.0645487
0.0417484	0.0423005	0.0405493	0.0757391
0.0434693	**0.0443224**	**0.0420645**	**0.0801280**
0.0438430	0.0444928	0.0437945	0.0894230
0.0448293	0.0446978	0.0457060	0.0885142
0.0478439	0.0449794	0.0491868	0.0930273
0.0484388	0.0450663	0.0542150	0.1128250
0.0486863	0.0492701	0.0558863	0.1309260
0.0505658	0.0642964	0.0562682	0.1310140
0.0518176	0.0916236	0.0647300	0.1399360
0.0607796	0.1311020	0.1005830	0.1520440
0.0639127	0.1352570	0.1359760	0.1553080
0.0740992	0.1473870	0.1477130	0.1712730

The general picture is the same as for XOR: Both crossover and mutation are required for a optimal set of GA operators. Crossover alone is a weak operator . Also, a crossover-mutation rate of 20 seems to be better than 2.

The Optimal Number of Crossover and Mutation Operators

So far , we have found that mutation alone works better than crossover alone, but the best option is a mixed set. In a extended version of the first two experiments, we are looking for the optimal composition of this set. Table 5 and 6 present the result of these experiments in detail, the median is printed in bold type.

Selection of Operators and Encoding Type for Sine

To verify these findings, we examine the evolution of a five-node fully-connected neural network for the sine function. Sine is a more difficult task, as a random search for an solution suggests: Within 80,000 randomly generated networks, only very few have fitness values below 0.1.

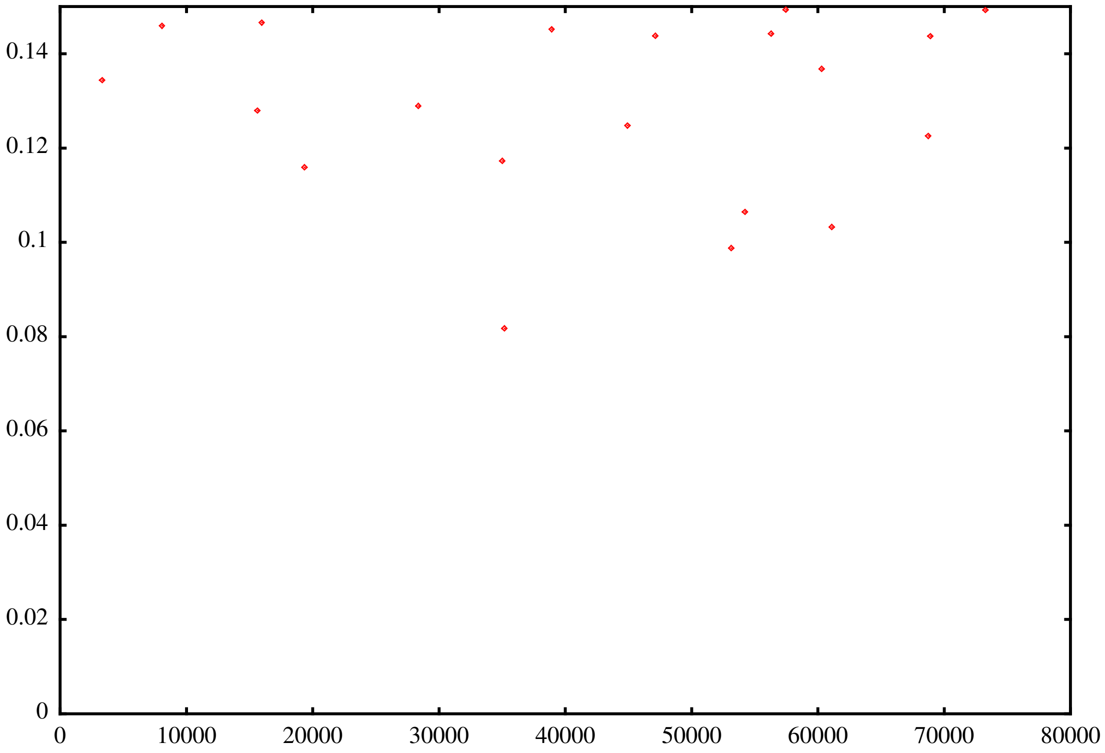


Figure 3.4: Best Fitness Values of Random Sine Networks

Experiment 3: 900 generations, target fitness 0.001, 1 population with 30 individuals, variable number of crossovers, mutations and crossover-mutation rate. 100 epochs, target error 0.01, learning rate 0.8. Sine with 17 training patterns. Weight-based encoding: 5 total nodes, 7 total bits/weight (no index), mutation rate 20. Two-point-crossover.

Using the same parameters as in experiment 1, the GANN system never reaches the target fitness of 0.001 in the 21 runs of each version of the experiment. Table 4 displays the final fitness values after 900 generations. The median - the 11th best run out of 21 - is printed again in bold type.

Table 3: Experimental Results of Evolving XOR using Gray Coding

10 mutations	5 mut., 5 cro. cro-mut-rate 2	5 mut., 5 cro. cro-mut-rate 20	10 crossovers
30	30	40	30
30	30	60	50
40	110	60	60
130	130	80	90
160	130	80	110
180	130	90	120
190	140	120	120
210	140	120	120
250	140	130	130
260	150	130	220
280	150	130	240
310	150	140	250
380	170	140	340
450	200	170	360
520	210	170	360
540	230	180	410
540	240	180	440
560	250	180	460
570	260	200	500
580	290	210	570
590	300	210	660
640	330	220	800
650	340	240	960
660	350	250	990
660	**360**	**250**	**1640**
680	420	260	1680
730	520	260	1870
740	530	290	1920
780	550	300	2080
780	580	310	2140
800	590	310	3030
860	600	320	3130
880	600	370	3480
880	700	380	3640
880	700	390	4230
900	720	410	4970
910	740	430	5180
950	740	450	5880
990	790	500	6540
1000	790	500	6740
1060	800	530	6900
1060	850	530	7270
1080	910	540	>9030
1090	950	560	>9030
1120	1050	560	>9030
1120	1200	590	>9030
1250	2340	630	>9030
1440	2430	630	>9030
1620	3200	740	>9030
1810	5600	1000	>9030

We examine how gray coding changes the performance of the GANN system in experiment 2. This experiment uses the same parameter as experiment 1, except for the fact that gray coding is applied to the single weight values. The detailed results of 50 runs is shown in Table 3.

> **Experiment 2:** 900 generations, target fitness 0.001, 1 population with 30 individuals, variable number of crossovers, mutations and crossover-mutation rate. 100 epochs, target error 0.01, learning rate 0.8. XOR. Weight-based encoding: 5 total nodes, 7 total bits/weight (no index), mutation rate 20. Gray coding.

Table 2 presents how binary and gray coding compare, if we use the median as a measurement. The median in this case is the result of the 25th best run out of 50 for each experiment. Since the mean average is harmed by a few bad runs, the median is a better choice to evaluate the performance of weight training.

Table 2: Comparison of Gray and Binary Coding

	10 mutations	5 mutations 5 crossovers cro-mut-rate 2	5 mutations 5 crossovers cro-mut-r. 20	10 crossovers cro-mut-r. 20
Binary Coding	540	360	320	670
Gray Coding	660	360	280	1640

The results show that for both gray and binary coding a mixture of crossover and mutation operators converges faster than either crossover or mutation alone. Also, mutation seems to be slightly more effective in the task of reaching a solution fast than crossover.

Also, a crossover mutation rate of 20 is superior to a rate of 2 in both cases. This rate defines how often mutation is applied to the offspring of crossover. So, with a rate of 20, the offspring is mutated only in 1 out of 20 cases.

However, the introduction of gray coding is not preferable. On the contrary, it slows down the evolutionary process in most cases.

While the vast majority of random individuals have fitness values of circa 0.5, 0.707 and 0.866, an empirical study of 10000 random individuals also produced 4 individuals that reach the target fitness of 0.001. Figure 3.2 provides more detail of this study .

Considering this data, a random search has to examine 2500 individuals in order to find one with the desired target fitness. If we compare this with the 350 individuals in one of the GANN experiments, we can state that GANN outperforms random search by the factor 2500/350, which is about 7.

Gray Coding

Studies [Caruana, 1988] have shown that the performance of genetic algorithms may be increased by using an alternate form of encoding integer numbers called gray coding. Figure 3.3 displays the gray codes for the numbers 0-7 opposing to their binary codes.

Number	Gray Codes	Binary Codes
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

Figure 3.3: Examples of Gray Codes

For gray coding, the hamming distance between two neighboring numbers $(n, n + 1)$ is constant 1 for any n. The hamming distance is the number of different bits between two codes. For binary coding it varies. In the example given, it is up to 3 for (3,4). These “hamming cliffs” make the fine-tuning transition task for the GA difficult.

Empirical studies on genetic algorithms in general suggest that gray coding improves the GA for some functions, or at least have the same effect as binary coding.

the back-propagation training of its individuals for their evaluation. Each individual is trained over 100 epochs, which totals in 35000 epochs. So, both approaches lead to astonishingly similar results.

However , this comparison has to be questioned in a couple of ways. It applies only to this special situation. Both, the performance of the GANN system and neural network training alone depend on a variety of parameter that do not equally apply to both approaches. A generation of a genetic algorithm is something principally different than an epoch of a neural network.

GANN vs. Random Search

A different criterion to evaluate the quality of GANN system is its performance in comparison to random search. Random search in this case means the random generation of individuals by the encoding strategy . We use the same parameters as in experiment 1.

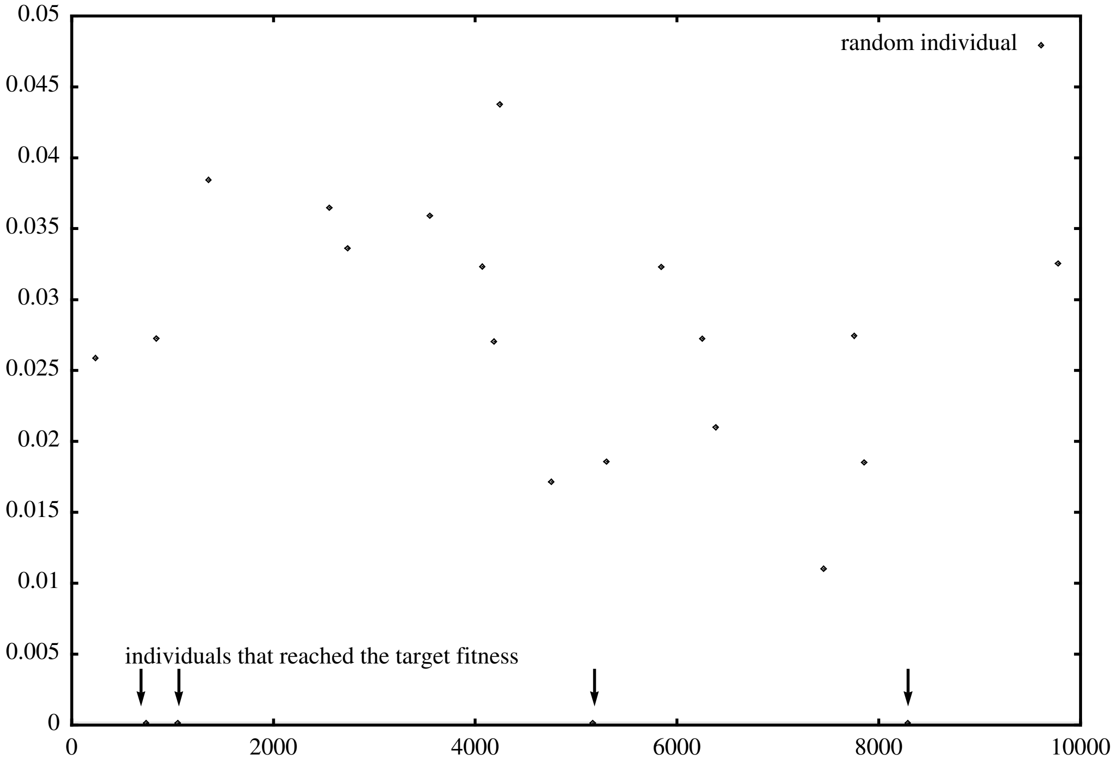


Figure 3.2: Best Fitness Values of Random Individuals

The table suggests that an average of about 300 to 700 individuals have to generated in order to find a network that suffices the target fitness.

GANN vs. Backpropagation Training

How can we judge these results? Figure 3.1 displays, how plain back-propagation deals with the situation, if the initial weights are set to random values in the interval $[-1, 1]$.

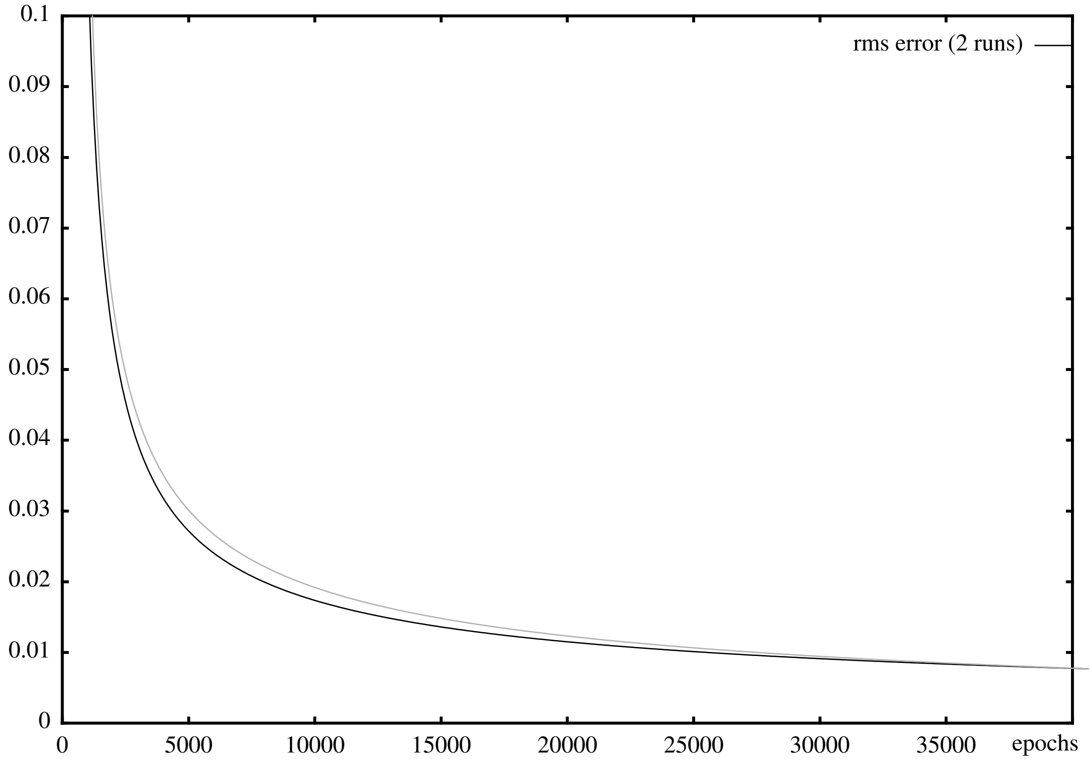


Figure 3.1: Backpropagation Training of the XOR Function

The figure suggests that backpropagation training takes about 35000 training epochs to train XOR to a five node fully-connected network. How does that relate to the generation of 350 individuals by the GANN system (average case for 5 crossover, 5 mutation, cro-mut-rate 20). During the experimental run of both algorithms on a SPARC workstation, about 20 seconds of computation time was spent in both cases.

The GANN system spends the vast majority of its computational time in

Table 1: Number of Individuals Evaluated for XOR (50 experimental runs)

10 mutations	5 mut., 5 cro. (cro-mut-rate 2)	5 mut., 5 cro. (cro-mut-rate 20)	10 crossovers (cro-mut-rate 20)
30	30	90	50
100	100	110	50
180	100	120	60
190	110	140	80
210	120	150	90
210	130	160	100
220	140	180	100
220	170	190	110
240	180	200	110
270	190	200	120
310	220	210	140
310	230	220	160
320	240	230	160
320	240	230	240
330	260	250	250
350	270	250	290
350	270	260	340
380	280	260	400
380	280	270	460
390	300	270	490
390	300	280	500
400	320	280	510
430	330	300	520
440	340	300	620
490	350	310	650
540	**360**	**320**	**670**
540	360	360	800
620	390	370	810
640	400	390	830
690	410	390	990
710	440	390	1100
720	440	410	1480
730	450	410	1510
750	450	410	1640
850	470	420	1790
880	480	420	2980
900	530	440	3080
1040	530	490	4490
1050	530	670	4580
1080	560	710	5280
1100	590	850	5430
1140	590	900	5640
1170	650	940	6010
1230	750	1790	7180
1240	780	2070	7700
1260	1000	5080	8780
1290	1000	8670	>9030
1360	1010	>9030	>9030
1750	1890	>9030	>9030
2170	2670	>9030	>9030

Each weight of a given architecture is represented by n bits. The index bit indicating the existence of a connection is optional. If it is used, merely $m = n - 1$ bits encode the weight value. If a network is created, the weights are initially set to integer values between $-2^{m-1} - 1$ and $2^{m-1} + 1$, excluding 0.

It is also possible to encode just the connectivity of the network. In this case, the weights are set to random values in the interval $[-1, 1]$. The range of the weight values may be changed by a scaling factor.

The encoding strategy supports one-point, two-point and uniform crossover.

3.3 Experiments on Weight Optimization

The first section of experiments focuses on weight-optimization. A fixed architecture is used, and merely the weight values are encoded in the parameter string. The experimental setting is based on [Whitley, 1990].

The first experiment tackles XOR. A fully-connected five node architecture is used and the target is a fitness of 0.001 (root mean square error of 0.01 within 10 epochs).

Experiment 1: 900 generations, target fitness 0.001, 1 population with 30 individuals, variable number of crossovers, mutations and crossover-mutation rate. 100 epochs, target error 0.01, learning rate 0.8. XOR. Weight-based encoding: 5 total nodes, 7 total bits/weight (no index), mutation rate 20. Two-point-crossover.

The results of 50 runs of the experiment are presented in table 1, ordered with the best ones first. The median is printed in bold type. The numbers in the table indicate the number of individuals generated during the run of the experiment, until the target fitness is reached. This is 30 random individuals for the initial population and 10 generated by crossover and mutation for each generation.

4 Periods of Sine

In this variation of the function above, four periods of sine are used. This results in a more difficult problem. Also, the function has to be slightly transformed in order to fit into the [0,1]x[0,1] input-output space:

$$\frac{1}{2}(\sin(8\pi x) + 1)$$

Gaussian Bell Curve

Also, the Gaussian Bell Curve, which is widely applied in the field of statistic is used as a problem for the evaluation of GANN systems. The formula for this function is:

2-Dimensional Gaussian Bell Curve

Finally , a real valued function with two input parameters is also considered. The function is the two-dimensional version of the above described Gaussian Bell Curve. It is defined as:

Figure 3.0 displays the graphs of the three real-valued functions.

3.2 Encoding Strategies

We implemented a weight-based encoding strategy that is based on Whitley's GENITOR (refer to section 2.2.1). The system supports all kinds of architectures, but we focused especially on fully connected topologies, i.e. all feedforward connections are possible.

respectively.

Sine

Opposed to the boolean functions, sine takes real valued inputs and produces real valued outputs. To keep both input and output ranges in the limits between 0 and 1, we used training patterns for the slightly adapted function:

$$\frac{1}{2}\left(\sin\left(2\pi x\right)+1\right)$$

A graph is displayed in figure 3.0. An arbitrary number of training patterns can be used. More patterns increase the accuracy of the problem.