Evolvable Hardware for Generalized Neural Networks

Masahiro Murakawa Shuji Yoshizawa Isamu Kajitani* Tetsuya Higuchi**

University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo, Japan
* University of Tsukuba, 1-1-1 Tennoudai, Tsukuba, Ibaraki, Japan
** Electrotechinical Laboratory, 1-1-4 Umezono, Tsukuba, Ibaraki, Japan

Abstract

This paper describes an evolvable hardware (EHW) system for generalized neural network learning. We have developed an ASIC VLSI chip, which is a building block to configure a scalable neural network hardware system. In our system, both the topology and the hidden layer node functions of a neural network mapped on the chips are dynamically changed using a genetic algorithm. Thus, the most desirable network topology and choice of node function (e.g. Gaussian or sigmoid) for a given application can be determined adaptively. This approach is particularly suited to applications requiring ability to cope with time-varying problems and real-time timing constraints. The chip consists of 15 Digital Signal Processors (DSPs), whose functions and interconnections are reconfigured dynamically according to the chromosomes of the genetic algorithm. Incorporation of local learning hardware increases the learning speed significantly. Simulation results on adaptive equalization in digital mobile communication are also given. Our system is two orders of magnitude faster than a Sun SS20 on the corresponding problem.

1 Introduction

The traditional applications of neural networks focused on the *off-line* learning of a given function using a single network whose weights are gradually modified. In recent years, the alternative approach of *on-line* adapting by reshaping the network itself has been attracting renewed attention [Fiesler, 1994]. The on-line approach has the advantages of efficiency and flexibility which are impossible with the off-line approach. We embody this on-line approach with evolvable hardware (EHW) [Higuchi *et al.*, 1992][Higuchi *et al.*, 1994]. Ability of this method to dynamically adapt to changing situations is particularly suited to practical industrial applications.

However, optimal performance for a given application is produced by an architecture with the most suitable topology and the most appropriate node functions (i.e. sigmoid or Gaussian). Further, to meet the time constraints imposed by real-time applications, neural network hardware systems need to be 'tailored' to the size of the ideal network for the problem. In general, it is very difficult to design an optimal neural network and process it with scalable parallel hardware.

To solve these two problems, we have developed (1) a learning scheme which utilizes genetic algorithms (GAs) to automatically select both the optimal network topology and the node functions, and (2) an evolvable hardware chip that functions as a building block for configuring a scalable neural network.

A concept of EHW is an innovative hardware design methodology for truly adaptive hardware systems [Higuchi, 1997]. In systems designed by EHW concepts, both the choice of the hardware function on each processing element and the specification of these elements' interconnections are determined by a GA [Goldberg, 1989] and reconfigured dynamically. The particular chip that we describe in this paper is an example of hardware designed by the EHW concept aimed at generalized neural network processing.

The paper is organized as follows. Section 2 reviews related work on learning neural networks with GAs, highlighting especially some of the shortcomings that our learning scheme will address. Section 3 describes the concept of EHW. Section 4 explains our new scheme and how EHW chips are dynamically configured to process neural networks. In Section 5, we report simulation results on the problem of adaptive equalization in digital mobile communication. Section 6 briefly describes the chip, and Section 7 gives our conclusions.

2 Evolutionary Neural Networks

Many researchers have worked on designing neural networks with GAs. Since there are many thorough reviews on this work [Schaffer *et al.*, 1992][Yao, 1993], we do not attempt an exhaustive review here. Such approaches typically utilize a GA to evolve the optimal topology of an appropriate network, and then use back-propagation to train the weights. However, using this approach online for industrial applications would be difficult because of the slow speed of both the back-propagation and GA components.

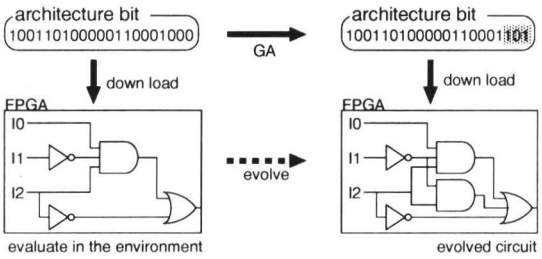

Figure 1: Evolvable Hardware at the Gate-Level

To improve the learning speed, radial basis function (RBF) networks [Powell, 1987][Poggio and Girosi, 1990] combined with genetic algorithms may be an appealing choice. Indeed, the learning speed of RBF Networks (RBFNs) can outperform multi-layer perceptron (MLP) by up to three orders of magnitude [Moody and Darken, 1989].Billings, for example, has worked on the genetic synthesis of RBFNs [Billings and Zheng, 1995].

However, compared with the MLP, RBFN requires large numbers of hidden layer nodes, particularly for high-dimensional input/output spaces (the "curse of dimensionality"). It was therefore our idea to mix the use of RBFs and sigmoid functions within a single architecture. Further, rather than specifying *a priori* how the two functions should be combined, we developed a method of using a GA to automatically tailor the node functions in a network to a given problem. To reduce the learning time, a local learning algorithm is first applied to bring the network weights to a reasonable level. This local learning is performed in parallel by hardware.

One significant benefit of our approach is that the network structure can vary in time. This is not possible with conventional neural network hardware, but our chip allows the optimal network structure and node functions to be dynamically reconfigured even while the network is being used on-line. In other words, our hardware system is a parallel processor where the number of processing elements is varied by a GA to continually produce the best performance.

3 Evolvable Hardware

The method of the evolvable hardware design is to change dynamically following two hardware configurations according to the GA chromosomes: (1) choices of a hardware function on each processing element of a software-reconfigurable device and (2) specification of interconnections between these elements. These reconfigurations can continue on-line to improve performance adaptively. In the conventional hardware design, it is necessary to prepare all the specifications of the hardware functions in advance. On the other hand, EHW can be reconfigured without such specifications. From this, we can see that the EHW concept provides a contrasting bottom-up hardware design methodology to the conventional top-down methodology. Thus, EHW is particularly suited to real-time applications where no hardware specification can be given in advance.

To realize the EHW concept, most existing research employs Field Programmable Gate Arrays (FPGAs) and Programmable Logic Devices (PLDs) as software-reconfigurable devices. Their internal circuit connections and node logic functions can be reconfigured by downloading binary strings, called *architecture bits*. The basic idea of these researches is to regard the architecture bits as chromosomes and to evolve good hardware structures by applying GAs to these strings, as shown in Figure 1.

Attempts to apply most research on EHW to practical problems, however, would suffer from the common problem that only relatively small circuits can be evolved. This is because the hardware evolution is based on primitive gates such as AND-gates and OR-gates; we call the evolution at this level *gate-level* evolution. The hardware functions resulting from gate-level evolution are not typically powerful enough for use in industrial applications.

In order to solve this problem, we have proposed a new type of hardware evolution: *function-level* evolution [Murakawa *et al.*, 1996]. Our proposal is that if hardware is genetically synthesized from high-level hardware functions (such as adders and multipliers) instead of primitive gates (like AND and OR gates), more useful hardware functions will be obtained. Depending on the application, the high-level function and the topology of the interconnection need to be determined carefully. This suggests that there will be a variety of EHW architectures at the function-level. The particular chip that we describe in this paper is an example of hardware designed for the function-level evolution. In the chip, the high-level functions can be directly implemented by a single DSP to perform the neural network processing (e.g. summation, calculation of the sigmoid functions or RBFs). The interconnections and the functions of each DSP are then determined by the GA chromosomes.

4 Evolvable Hardware for Generalized Neural Networks

We have developed a learning scheme for a generalized neural network [Murakawa *et al.*, 1997]. We describe the genetic learning and then show how the network is mapped onto FPMD (Field Programmable Multiple DSPs) chips. FPMD is an evolvable hardware chip specially designed for implementing generalized neural networks.

4.1 Genetic Learning

The generalized neural network considered here is defined as follows:

$$y = f(\boldsymbol{x}) = \sum_{k=1}^{n} w_k \mu_k(\boldsymbol{x}) \,. \tag{1}$$

where n is the number of the hidden layer node functions. For simplicity, each function $\mu_k(\boldsymbol{x})$ is either radial

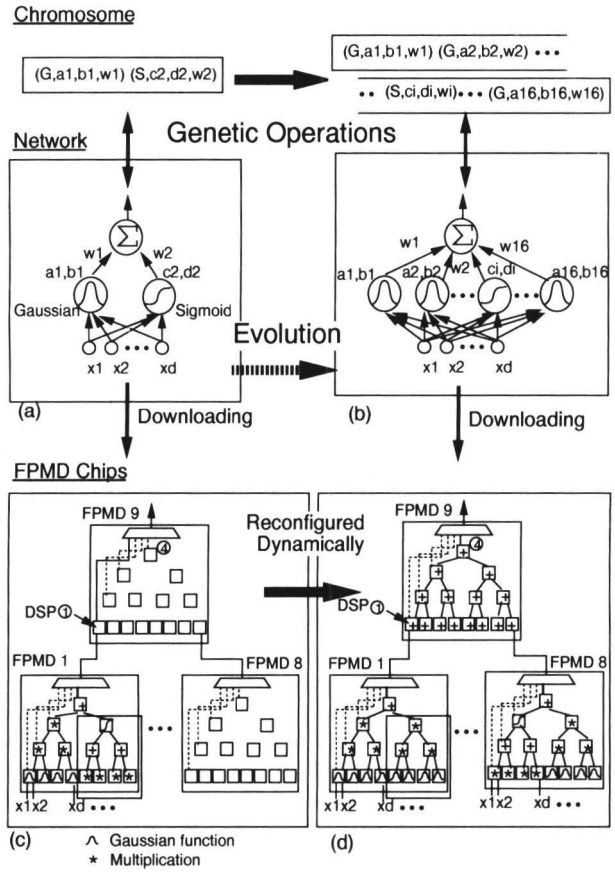

Figure 2: Evolvable Hardware for Generalized Neural Networks

basis function (RBF) or sigmoid function:

$$\mu_k(\boldsymbol{x}) = \prod_{i=1}^{d} exp(-(x_i - a_{ik})^2/b_{ik}) \quad (2)$$

$$or$$

$$= \frac{1}{1 + exp(\sum_{i=1}^{d} c_{ik}x_i - d_k)}. \quad (3)$$

The number of outputs in the network is assumed to be one, but the architecture can be readily extended to cope with multi-output problems.

The genetic learning determines the network topology (e.g. the number of nodes:n) and the choice of node functions (e.g. Gaussian or sigmoid function) adaptively for a given application.

The w_k and the parameters of the node functions (e.g. $a_{ik}, b_{ik}, c_{ik}, d_k$) are tuned by local learning with the steepest descent method. In Table 1, descriptions of the genetic operators are given (for more details, see [Murakawa *et al.*, 1997]).

Figure 2 illustrates this genetic learning. A chromosome of the GA represents one network. The network is evolved by applying the genetic operators to the chromosome. For example, Figure 2 shows how a network with two hidden layer nodes (a) is evolved to have 16 nodes (b).

Table 1: Genetic Operators

Coding	a chromosome consists of n genes
Gene	a gene represents one hidden layer node (node function and the parameter)
Selection	tournament selection (tournament size is 2)
Crossover	modified two-point crossover
Mutation	three types of mutations:
	Insertion of a node - insert a RBF node or a sigmoid node
	Deletion of a node - delete a node selected randomly
	Replacement of a node - change the node function
Random Immigrant	10% of the population are replaced with new individuals created randomly
Local Learning	tune the node parameters by iterations of the steepest descent method

4.2 Mapping on the FPMD Chips

In Section 4.1, we have described how the most desirable network structure and the choice of the node functions are determined with genetic learning. Here we show how the obtained network is mapped on the FPMD chips and how they are reconfigured dynamically.

The FMPD chip is a building block to configure a scalable neural network hardware. An arbitrary size of neural network hardware can be configured with multiple FPMDs because the chip includes 15 DSPs connected in a binary tree shape as shown in Figure 2(c).

The obtained neural network by genetic learning is immediately mapped on FPMDs. For example, the network in Figure 2(a) can be mapped onto FPMD No.1 in Figure 2(c). In this case, as the network is still small, only one FPMD is used.

Each DSP can perform any arithmetic function. So, for example, by using the seven DSPs in FPMD No.1 in Figure 2(c) (on the right side), a sigmoid function is effectively implemented with the binary tree connections utilizing the inherent parallelism. Binary tree connections are also very useful when the summation of outputs is calculated. For example, the FPMD No. 9 in Figure 2(d) is configured to conduct the summation in parallel.

The functions and interconnections of the FPMD chips are dynamically controlled by rewriting the chromosome. For example, the output of the FPMD No.9 in Figure 2(c) is connected to the DSP No.1. After evolution the DSP No.4 is connected to the output in Figure 2(d). Also, the 7 DSPs on the right side of the FPMD No.1 in Figure 2(c) calculate the sigmoid function. After reconfiguration they are changed to the Gaussian function in Figure 2(d).

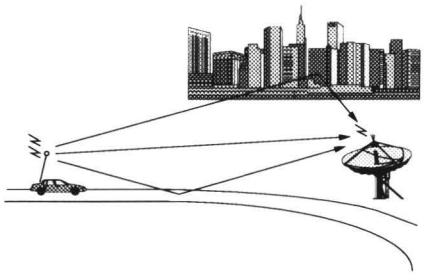

Figure 3: Mobile Communication

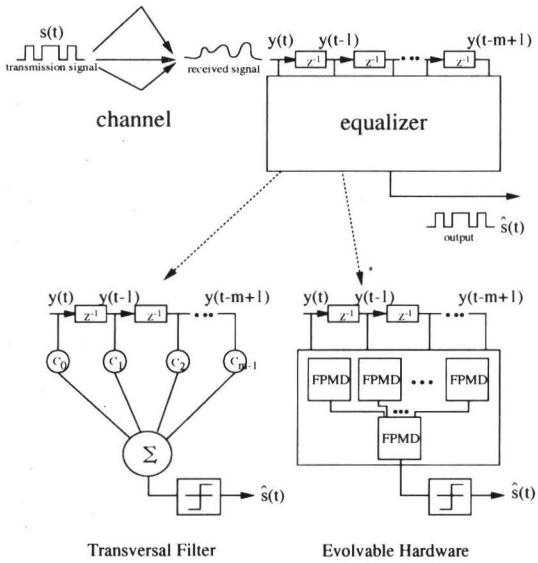

Figure 4: Adaptive Equalizers

5 Adaptive Equalizer in Digital Mobile Communication

To examine the performance of our system, we conducted a simulation of adaptive equalization in digital mobile communication. In particular, the ability to adapt to a dynamically changing environment was of special concern to us.

High-speed communications channels are often impaired by linear and non-linear channel distortion and additive noise. To obtain reliable data transmission in such communications systems, adaptive equalizers are required [Proakis, 1988]. In digital mobile communications, the channel can be influenced by environmental conditions such as landscape and the presence of buildings (Figure 3). The task of the equalizer is to recover the transmitted symbols based on the channel observation y(t).

Existing adaptive equalization techniques for time-varying channels employ a linear transversal filter (Figure 4). However, if the non-linear channel distortion is too severe, adaptive equalizers based on such linear transversal filters suffer from severe performance degradation. For such channels, non-linear adaptive equalizers based on neural networks were proposed [Chen *et al.*, 1990][Chen *et al.*, 1991]. But the algorithms are so complicated for hardware implementation.

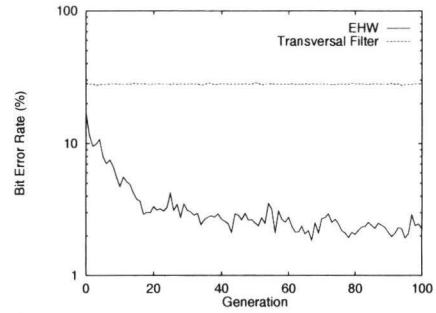

Figure 5: Learning Performance of the EHW-Based Equalizer (SNR: 15 dB)

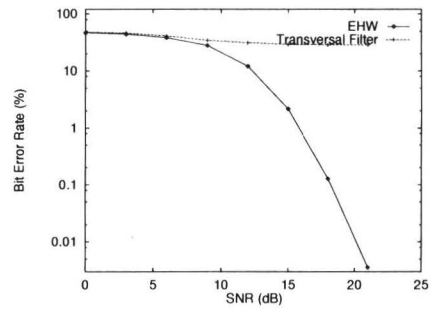

Figure 6: Bit Error Rate of the EHW-Based Equalizer versus SNR

To overcome these difficulties, we apply our system to the adaptive equalizer. A communications system that employs the EHW-based adaptation equalizer is shown in Figure 4. The transmitter sends a *known training sequence* to the receiver, and the receiver adjusts the EHW-based equalizer so that it reproduces the correct transmitted symbols.

5.1 Learning Performance of the EHW-Based Equalizer

We simulated the learning performance of the proposed EHW-based equalizer. The transfer function of the channel was given by $G(z) = 1.0 + 1.5z^{-1}$, and zero-mean white Gaussian noise was added to the output of the channel. The order of the equalizer was 2 (m = 2).

A training set of eight data points was generated at every generation. Using a population size of 80, the fitness of each individual was determined by $n/8$, where n was the number of correct classifications by the EHW. The bit-error-rate (BER) was defined as the ratio of misclassified to correct symbols in the output of the best-of-generation individual. The BER was evaluated at every generation based on 10^5 random input symbols. Simulations were carried out which restricted the maximum

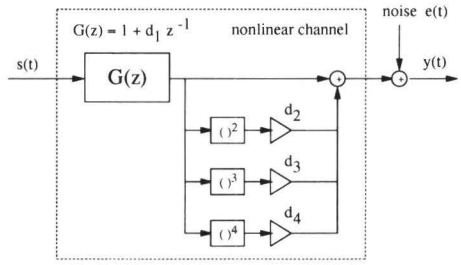

Figure 7: Non-linear Transmission Channel used in the Simulations

number of the hidden layer nodes to 20.

Figure 5 shows the learning performance of this simulation for a signal-to-noise ratio (SNR) of 15 dB. The solid curve was obtained by averaging the results of 100 independent runs. The broken line shows the learning curve of a transversal-filter-based equalizer, whose total number of training sequences was the same as that of the EHW. As can been seen, the BER of the EHW-based equalizer is far lower. This is due to the ability of the generalized neural network to synthesize non-linear functions.

Figure 6 shows the BER versus SNR achieved by the EHW-based equalizer at generation 100. We found that a significant improvement in the BER could be achieved by the EHW-based equalizer, especially at a high SNR.

5.2 Adaptive Equalization of Time-Varying Channels

In real communications systems, the characteristics of the channel are usually time-varying. Hence, adaptive equalizers are required to follow such changes and compensate for the channel distortion.

We therefore simulated the performance of the EHW-based adaptive equalizer for time-varying channels, using the non-linear channel shown in Figure 7. The transmitted sequence is passed through a linear channel whose transfer function is $G(z) = 1 + d_1 z^{-1}$, and the output of the channel is added to the nonlinear harmonics. The value of the gain coefficients d_2, d_3,and d_4 determines how severe the nonlinear distortion will be. Such non-linear channel models are frequently encountered in data transmission over digital satellite links. The linear transversal-filter-based adaptive equalizer can not compensate for such non-linear channel distortion.

As before, we simulated the bit-error-rate achieved by the EHW-based adaptive equalizer whose order m was 2. Simulations were performed for the case in which d_1 changed drastically during evaluation (Figure 8) and for the case in which d_1 changed gradually (Figure 9). In the simulations, the coefficients were set to $d_2 = 0.6, d_3 = 0.5$, and $d_4 = 0.4$. The length of the training sequence was 8. For the genetic learning, the maximum number of the hidden layer nodes was restricted to 20 and the population size was 80. The results were averaged over 100 independent runs.

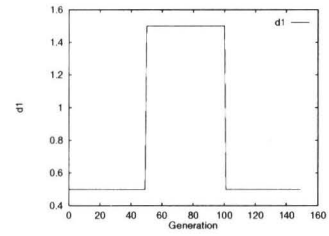

Figure 8. Time-varying Channel (drastic change in d_1)

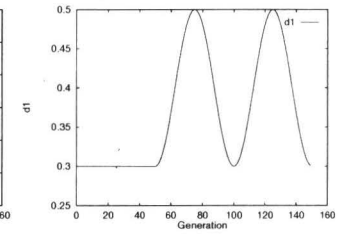

Figure 9. Time-varying Channel (gradual change in d_1)

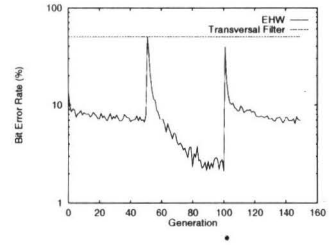

Figure 10. Adaptive Equalization of the EHW-based Equalizer (SNR: 15 dB)

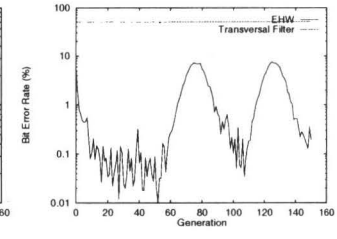

Figure 11. Adaptive Equalization of the EHW-based Equalizer (SNR: 15 dB)

The BERs achieved by the EHW-based equalizer for the channels of Figure 8 and Figure 9 are shown in the graphs of Figure 10 and Figure 11, respectively. These graphs demonstrate that EHW-based equalizers have the ability to follow both drastic and gradual environmental change.

6 Hardware Implementation

We have developed a prototype board and an ASIC VLSI chip. The FPMD chip is designed to implement generalized neural networks.

6.1 FPMD chip

The FPMD chip includes 15 DSPs connected in a tree shape. The tree size is genetically controlled. Within a chip, broadcast hardware is included to speed up local learning. Local learning is conducted in parallel at each DSP. The chip has 250K gates and operates at a 33 MHz clock speed. It will be available in July, 1997. For the learning of the adaptive equalizer, we have a simulation result that the execution with 9 FPMD chips takes 1.3 seconds while the execution on Sun SS20 takes 2 minutes.

6.2 The prototype board

The prototype board is designed to enable *on-line* adaptation. As shown in Figure 12, the prototype board includes two sets of FPMD chips. One of the two FPMD sets is an *execution* FPMD set, which actually processes the incoming data. The other set is a *learning* FPMD set, which is continually used to find a better hardware

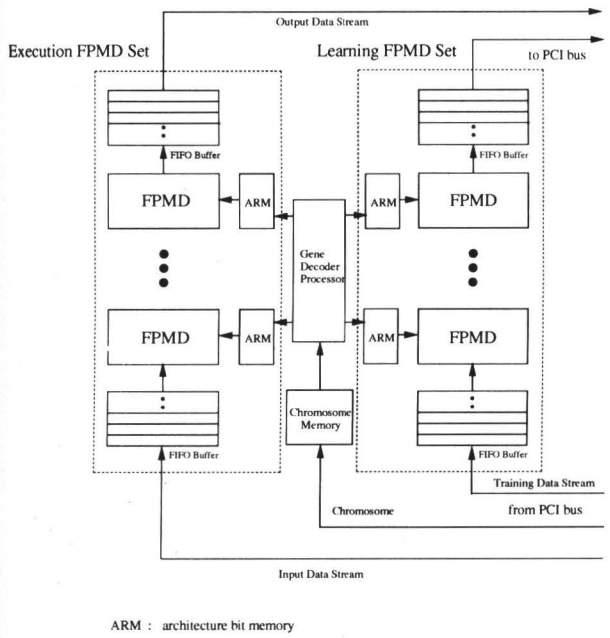

Figure 12: Prototype Board

structure by genetic learning while the execution FPMD set is processing the existing best work. The training data are updated at regular intervals, enabling the architecture to keep track of a changing environment.

If the performance of the learning FPMD set is improved, then the execution FPMD set is reconfigured using the chromosome obtained by the learning FPMD set. Thus, the system realizes *on-line* adaptation.

The design of the prototype board allows it to be connected to the PCI bus of a personal computer. The GA calculation is performed outside the board, by the personal computer.

7 Conclusion

We have described a learning scheme and an evolvable hardware chip for generalized neural network learning. This development is aimed at the use in practical industrial applications, especially those which require the ability to cope with time-varying problems and real-time timing constraints. The need for such applications has been increasing recently. So, in addition to the adaptive equalizer, we are now working on loss-less data compression and ATM buffer control.

References

[Billings and Zheng, 1995] S. A. Billings and G. L. Zheng. Radial basis function network configuration using genetic algorithms. *Neural Networks*, 8(6):877–890, 1995.

[Chen *et al.*, 1990] S. Chen, G. J. Gibson, F. N. Cowan, and P. M. Grant. Adaptive equalization of finite nonlinear channels using multilayer perceptrons. *Signal Processing*, 20(2):107–119, 1990.

[Chen *et al.*, 1991] S. Chen, G. J. Gibson, F. N. Cowan, and P. M. Grant. Reconstruction of binary signals using an adaptive radial basis function equalizer. *Signal Processing*, 22(1):77–93, 1991.

[Fiesler, 1994] E. Fiesler. Comparative bibliography of ontogenic neural networks. In *International Conference on Artificial Neural Networks*, 1994.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, 1989.

[Higuchi *et al.*, 1992] T. Higuchi, T. Niwa T., Tanaka H. Iba, H. Garis, and T. Furuya. Evolvable hardware with genetic learning. In *Simulation of Adaptive Behavior.* MIT Press, 1992.

[Higuchi *et al.*, 1994] T. Higuchi, H. Iba, and B. Manderick. Evolvable hardware with genetic learning. In H. Kitano, editor, *Massively Parallel Artificial Intelligence.* MIT Press, 1994.

[Higuchi, 1997] T. Higuchi, editor. *First International Conference on Evolvable Systems.* Springer Verlag, 1997.

[Moody and Darken, 1989] J. Moody and C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[Murakawa *et al.*, 1996] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi. Hardware evolution at function level. In *Parallel Problem Solving from Nature IV*, pages 62–71. Springer Verlag, 1996.

[Murakawa *et al.*, 1997] M. Murakawa, S. Yoshizawa, I. Kajitani, and T. Higuchi. On-line adaptation of neural networks with evolvable hardware. In *Seventh International Conference on Genetic Algorithms.* Morgan Kaufmann, 1997.

[Poggio and Girosi, 1990] T. Poggio and F. Girosi. Networks for approximation and learning. *Proc. IEEE*, 78:1481–1497, 1990.

[Powell, 1987] M. J. D. Powell. Radial basis functions for multivariable interpolation: A review. In M. G. Cox, editor, *Algorithms for Approximation*, pages 143–167. Clarendon Press, 1987.

[Proakis, 1988] J. Proakis. *Digital Communications.* Prentice Hall Inc., 1988.

[Schaffer *et al.*, 1992] J. D. Schaffer, D. Whitley, and L. J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In L. D. Whitley and J. D. Schaffer, editors, *Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE Computer Soc. Press, 1992.

[Yao, 1993] X. Yao. A review of evolutionary artificial neural networks. *Int. J. Intelligent Systems*, 8(4):539–567, 1993.