Alopex: A Correlation-Based Learning Algorithm for Feed-Forward and Recurrent Neural Networks

K. P. Unnikrishnan, and K. P. Venugopal

Abstract

We present a learning algorithm for neural networks, called Alopex. Instead of *error gradient*, Alopex uses *local correlations* between changes in individual weights and changes in the global error measure. The algorithm does not make any assumptions about transfer functions of individual neurons, and does not explicitly depend on the functional form of the error measure. Hence, it can be used in networks with arbitrary transfer functions and for minimizing a large class of error measures. The learning algorithm is the same for feed-forward and recurrent networks. All the weights in a network are updated *simultaneously*, using only *local* computations. This allows complete parallelization of the algorithm. The algorithm is *stochastic* and it uses a 'temperature' parameter in a manner similar to that in simulated annealing. A heuristic 'annealing schedule' is presented which is effective in finding global minima of error surfaces. In this paper, we report extensive simulation studies illustrating these advantages and show that learning times are comparable to those for standard gradient descent methods. Feed-forward networks trained with Alopex are used to solve the MONK's problems and symmetry problems. Recurrent networks trained with the *same* algorithm are used for solving temporal XOR problems. Scaling properties of the algorithm are demonstrated using encoder problems of different sizes and advantages of appropriate error measures are illustrated using a variety of problems.

K. P. Unnikrishnan is with the Computer Science Department, GM Research Laboratories, Warren, MI 48090; and with the Artificial Intelligence Laboratory, University of Michigan, Ann Arbor, MI 48109.

K. P. Venugopal is with the Medical Image Processing Group, University of Pennsylvania, Philadelphia, PA 19104.

1. Introduction

Artificial neural networks are very useful because they can represent complex classification functions and can discover these representations using powerful learning algorithms. Multi-layer perceptrons using sigmoidal non-linearities at their computing nodes can represent large classes of functions (Hornik, Stichcomb, and White, 1989). In general, an optimum set of weights in these networks are learned by minimizing an error functional. But many of these functions (that give error as a function of weights) contain local minima, making the task of learning in these networks difficult (Hinton, 1989). This problem can be mitigated by (i) choosing appropriate transfer functions at individual neurons and appropriate error functional for minimization and (ii) by using powerful learning algorithms.

Learning algorithms for neural networks can be categorized into two classes.[1] The popular back-propagation (BP) and other related algorithms *calculate explicit gradients* of the error with respect to the weights. These require detailed knowledge of the network architecture and involve calculating derivatives of transfer functions. This limits the original version of BP (Rumelhart, Hinton, and Williams, 1986) to feed-forward networks with neurons containing smooth, differentiable and non-saturating transfer functions. Some variations of this algorithm (Williams and Zipser, 1989, for example) have been used in networks with feedback; but, these algorithms need non-local information, and are computationally expensive.

A general purpose learning algorithm, without these limitations, can be very useful for neural networks. Such an algorithm, ideally, should use only locally available information; impose no restrictions on the network architecture, error measures or transfer functions of individual neurons; and should be able to to find global minima of error surfaces. It should also allow simultaneous updating of weights and hence reduce the overhead on hardware implementations.

Learning algorithms that do not require explicit gradient calculations may offer a a better choice in this respect. These algorithms usually *estimate the gradient* of the error by local measurements. One method is to systematically change the parameters

---

[1] Methods that are not explicitly based on gradient concepts have also been used for training layered networks (Minsky, 1954; Rosenblatt, 1962). These methods are limited in their performance and applicability and hence are not considered in our discussions.

(weights) to be optimized and measure the effect of these changes (perturbations) on the error to be minimized. Parameter perturbation methods have a long history in adaptive control, where they were commonly known as the "MIT rule" (Draper, and Li, 1951; Whitaker, 1959). Many others have recently used perturbations of single weights (Jabri, and Flower, 1991), multiple weights (Dembo, and Kailath, 1990; Alspector *et al.*, 1993), or single neurons (Widrow, and Lehr, 1990).

A set of closely related techniques in machine learning are Learning Automata (Narendra, and Thathachar, 1989) and Reinforcement Learning (Barto, Sutton, and Brouwer, 1981). In this paper we present an algorithm called 'Alopex'[2] that is in this general category. Alopex has had one of the longest history of such methods, ever since its introduction for mapping visual receptive fields (Harth, and Tzanakou, 1974). It has subsequently been modified and used in models of visual perception (Harth, and Unnikrishnan, 1985; Harth, Unnikrishnan, and Pandya, 1987; Harth, Pandya, and Unnikrishnan, 1990), visual development (Nine, and Unnikrishnan, 1993; Unnikrishnan, and Nine, 1993), for solving combinatorial optimization problems (Harth, Pandya, and Unnikrishnan, 1986), for pattern classification (Venugopal, Pandya, and Sudhakar, 1991 & 1992b), and for control (Venugopal, Pandya, and Sudhakar, 1992b). In this paper we present a very brief description of the algorithm and show results of computer simulations where it has been used for training feed-forward and recurrent networks. Detailed theoretical analysis of the algorithm and comparisons with other closely related algorithms such as reinforcement learning will appear elsewhere (Sastry, and Unnikrishnan, 1993).

2. The Alopex Algorithm

Learning in a neural network is treated as an optimization problem.[3] The objective is to minimize an error measure, E, with respect to network weights $\mathbf{w}$, for a given set of training samples. The algorithm can be described as follows: consider a neuron i with an interconnection strength w_{ij} from neuron j. During the n^{th} iteration, the weight w_{ij} is updated according to the rule,[4]

---

[2] *Alopex* is an acronym for *Al*gorithm fo*r* *p*attern *ex*traction, and refers to the *alopecic* performance of the algorithm.

[3] Earlier versions of this have been presented at conferences (Unnikrishnan, and Pandit, 1991; Unnikrishnan, and Venugopal, 1992).

[4] For the first two iterations, weights are chosen randomly.

$$w_{ij}(n) = w_{ij}(n-1) + \delta_{ij}(n) \tag{1}$$

where $\delta_{ij}(n)$ is a small positive or negative step of size δ with the following probabilities:[5]

$$\delta_{ij}(n) = \begin{cases} -\delta \text{ with probability } p_{ij}(n) \\ +\delta \text{ with probability } 1-p_{ij}(n) \end{cases} \tag{2}$$

The probability $p_{ij}(n)$ for a negative step is given by the Boltzmann distribution:

$$p_{ij}(n) = \frac{1}{1 + e^{-\frac{C_{ij}(n)}{T(n)}}} \tag{3}$$

where $C_{ij}(n)$ is given by the correlation:

$$C_{ij}(n) = \Delta w_{ij}(n) \,.\, \Delta E(n) \tag{4}$$

and $T(n)$ is a positive 'temperature'. $\Delta w_{ij}(n)$ and $\Delta E(n)$ are the changes in weight w_{ij} and the error measure E over the previous two iterations (Eqs. 5a and 5b).

$$\Delta w_{ij}(n) = w_{ij}(n-1) - w_{ij}(n-2) \tag{5a}$$

$$\Delta E(n) = E(n-1) - E(n-2) \tag{5b}$$

The 'temperature' T in Eq. (3) is updated every N iterations using the following 'annealing schedule':

$$T(n) = \frac{1}{N \cdot M} \sum_i \sum_j \sum_{n'=n-N}^{n-1} | C_{ij}(n') | \quad \text{if } n \textit{ is a multiple of } N \tag{6a}$$

$$T(n) = T(n-1) \quad \textit{otherwise.} \tag{6b}$$

M in the above equation is the total number of connections. Since the magnitude of Δw is the same for all weights, Eq. (6a) reduces to:

$$T(n) = \frac{\delta}{N} \sum_{n'=n-N}^{n-1} | \Delta E(n') | \tag{6c}$$

2.1 Behavior of the Algorithm

Equations (1) - (5) can be rewritten to make the essential computations clearer.

---

[5] In simulations, this is done by generating a uniform random number between 0 and 1 and comparing it with $p_{ij}(n)$

$$w_{ij}(n) = w_{ij}(n-1) + \delta \cdot x_{ij}(n-1) \tag{7}$$

δ is the step size and x_{ij} is either +1 or -1 (randomly assigned for the first two iterations).

$$x_{ij}(n-1) = \begin{cases} x_{ij}(n-2) \text{ with probability } p_{ij}(n) \\ -x_{ij}(n-2) \text{ with probability } 1-p_{ij}(n) \end{cases} \tag{8}$$

where

$$p_{ij}(n) = \frac{1}{1 + e^{\delta \cdot \frac{\Delta E(n)}{T(n)}}} \tag{9}$$

From Eqs. (7) - (9) we can see that if ΔE is negative, the probability of moving each weight in the same direction is greater than 0.5. If ΔE is positive, the probability of moving each weight in the opposite direction is greater than 0.5. In other words, the algorithm favors weight changes that will decrease the error E.

The temperature T in Eq. (3) determines the stochasticity of the algorithm. With a non-zero value for T, the algorithm takes biased random walks in the weight space towards decreasing E. If T is too large, the probabilities are close to 0.5 and the algorithm does not settle into the global minimum of E. If T is too small, it gets trapped in local minima of E. Hence the value of T for each iteration is chosen very carefully. We have successfully used the heuristic 'annealing schedule' shown in Eq. (6). We start the simulations with a large T, and at regular intervals, set it equal to the average absolute value of the correlation C_{ij} over that interval. This method automatically reduces T when the correlations are small (which is likely to be near minima of error surfaces) and increases T in regions of large correlations. The correlations need to be averaged over sufficiently large number of iterations so that the annealing does not freeze the algorithm at local minima. Towards the end, the step size δ can also be reduced for precise convergence.

The use of a controllable 'temperature' and the use of probabilistic parameter updates are similar to the method of simulated annealing (Kirkpatrick, Gelatt, and Vecchi, 1983). But Alopex differs from simulated annealing in three important aspects: (i) the correlation ($\Delta E \cdot \Delta w$) is used instead of the change in error ΔE for weight updates; (ii) all weight changes are accepted at every iteration; and (iii) during an iteration, all weights are updated simultaneously.

2.2 "Universality" of the algorithm

The algorithm makes no assumptions about the structure of the network, the error measure being minimized, or the transfer functions at individual nodes. If the change in the error measure is broadcast to all the connection sites, then the computations are completely local and all the weights can be updated simultaneously. The stochastic nature of the algorithm can be used to find the global minimum of error function. The above features allow the use of Alopex as a learning algorithm in feed-forward and recurrent networks, and for solving a wide variety of problems.

In this paper we demonstrate some of these advantages through extensive simulation experiments. Convergence times of Alopex for solving XOR, parity, and encoder problems are shown to be comparable to those taken by back-propagation. Learning ability of Alopex is demonstrated on the MONK's problems (Thrun, *et al*., 1991) and on the mirror symmetry problem (Peterson, and Hartman, 1989) that have been used extensively for benchmarking. Scaling properties of Alopex are investigated using encoder problems of different sizes. The utility of annealing schedule for overcoming local minima of error surfaces is demonstrated while solving the XOR problem. Since Alopex allows the usage of different error measures, we show that the use of an information theoretic error measure (Hopfield, 1987; Baum, and Wilczek, 1988; Unnikrishnan, Hopfield, and Tank, 1991), instead of the customary squared error results in smoother error surfaces and improved classifications. Finally we demonstrate its ability to train recurrent networks for solving temporal XOR problems. It should be stressed that in all these experiments, the *same* learning module was used for these diverse network architectures and problems.

3. Simulation Results

In this section we present results from an extensive set of simulation experiments. The algorithm has three main parameters; the initial temperature T, the step-size δ, and the number of iterations N over which the correlations are averaged for annealing. The initial temperature is usually set to a large value of about 1000. This allows the algorithm to get an estimate of the average correlation in the first N iterations and reset it to an appropriate value according to Eq. (6). Hence this parameter does not affect the simulations substantially. N is chosen empirically, and usually has a value between 10 and 100. Again, this is not a very critical parameter and for most of the iterations, is

not optimized. On the other hand, δ is a critical parameter, and is chosen with care. We have found that a good initial value is about 0.001 to 0.01 times the dynamic range of the weights. We terminate learning when the output-neuron responses are within 0.1 of their targets for the entire training set.

3.1 Comparisons with other learning algorithms

The first set of experiments were done to compare the convergence time of Alopex with back-propagation. Alopex was used to train multi-layer perceptrons with sigmoidal transfer functions, using the mean-squared error measure. Table 1 shows the performance of Alopex and a standard version of the back-propagation on the XOR, parity, and encoder problems. A 2-2-1 network was used for solving the XOR, a 4-4-1 network was used for solving the (4 bit) parity, and a 4-2-4 network was used for solving the (4 bit) encoder problem. The average number of iterations taken by the two algorithms over 100 trials are given in Tbl. 1.

TABLE 1 HERE

We can see that the average number of iterations taken by Alopex is comparable to those taken by back-propagation. It should be pointed out that in Alopex all the weights are updated simultaneously and hence with a parallel implementation, the computation time taken per updating would be much less than that of back-propagation.

The next set of experiments were done to compare Alopex with Reinforcement Learning and Learning Automata. The multiplexer task, which involves learning a six-input boolean function, has been solved using both these methods (Barto, 1985; Mukhopadhyay, and Thathachar, 1989). Of the six input lines, four carry data and two carry addresses. The task is to transmit the appropriate data, as specified by the address, to the output line. Following Barto (1985), we chose a network with six linear input units, four sigmoidal hidden units and a sigmoidal output unit, with 39 parameters (34 weights and five thresholds) to adjust. The training data was continuously fed into the network and the parameters were updated after every 64 examples. The training was stopped when 1000 consecutive examples were correctly classified. Following Mukhopadhyay and Thathachar (1989), we created three tasks with three different sets of address lines. Table 2 shows the average number of updates (over 10 trails, each starting with a different set of weights) needed for solving each of the tasks.[6] From

[6] Mukhopadhyay and Thathachar (1989) specifies the convergence criterion as the correct

Tbl. 2 we can see that Alopex compares favorably with these algorithms. Since the updating and stopping criteria are slightly different in the three studies, the numbers can not be compared directly. Table 3 shows the number of iterations taken (from one initial set of weights) for different step-sizes, using the mean-squared error and the log error (see section 3.4).

TABLES 2 AND 3 HERE

The third set of experiments were done to compare Alopex with weight perturbation methods. Figure 1 shows the mean square error as a function of iterations for the XOR problem. A 2-2-1 network was used. The data for weight perturbation and back-propagation is taken from Jabri and Flower (1991). For a small step-size (δ=0.008), the error decrement for Alopex is fairly smooth and it takes about the same number of iterations as the other two methods to converge. The convergence can be speeded up by using larger steps, as shown by the plot for δ = 0.03. The error decrement is no longer smooth.

FIGURE 1 HERE

3.2 The MONK's problems

These are a set of three classification problems used for extensive bench-marking of machine learning techniques and neural network algorithms (see Thrun *et al.*, 1991 for details). Samples are represented by six, discrete-valued attributes and each problem involves learning a binary function defined over this domain. Problem 1 is in standard disjunctive normal form. Problem 2 is similar to parity problems and combines different attributes in a way that makes it complicated to describe in disjunctive or conjunctive normal forms using only the given attributes. Problem 3 is again in disjunctive normal form, but contains about 5% misclassifications. In the database, 124 randomly chosen samples are designated for training the first problem, 169 for training the second problem and 122 for training the third problem. The entire set of 432 samples are used for testing.

A feed-forward network with 15 input units, 3 hidden units, and an output unit was trained to solve these problems. The network contained sigmoidal non-linearities and Alopex was used to minimize the mean-squared error. The network learned to

---

classification of the 64 training examples. With this criterion, the number of iterations are lower.

classify the first test set with 100% accuracy after 5,000 iterations and the second test set after 10,000 iterations. The third test set was correctly classified after 1,000 iterations and Fig. 2 shows the network output for the 432 samples.

FIGURE 2 AND TABLE 4 HERE

Table 4 compares the performance of feed-forward perceptrons trained using standard back-propagation, back-propagation with weight decay, the cascade-correlation technique, and Alopex on these problems. We can see that Alopex is the only method capable of correctly learning all the three problems. It should be noted that about 25 learning methods were compared in Thrun, *et al.*, but none of them achieved 100% accuracy on all three test sets. These experiments show that Alopex can be used as a powerful, general learning algorithm.

3.3 The mirror symmetry problem

The mirror symmetry problem has also been used for bench-marking learning algorithms (see Peterson, and Hartman, 1989; Sejnowski, Kienker, and Hinton, 1986; Barto, and Jordan, 1987). The inputs are NxN-bit patterns with either a horizontal, a vertical, or a diagonal axis of symmetry and the task of the network is to classify them accordingly. For comparing numerical generalization accuracies, we used the fixed training set paradigm described in Peterson, and Hartman (1989). Ten sets of 4x4-bit data, with each set containing 100 training samples, were used in the experiments. A feed-forward network with 16 input units, 12 hidden units and 3 output units were trained on each one of these data sets and the training was terminated when all the training samples were correctly classified according to the "mid-point" criteria.[7] The generalization accuracy was determined on the remaining 9 sets of data, using the same criterion. Experiments were done using patterns where the elements had probabilities of 0.4, 0.5, and 0.6 for being on. Table 5 shows the generalization accuracies and average number of training iterations. Alopex was used to minimize the mean-squared error measure and the log error measure (see below). The accuracies for Mean Field Theory Learning (MFT) and back-propagation (BP) are also shown. The generalization accuracy for Alopex is slightly better in one case and is considerably better in the other two cases.[8]

---

[7] The responses of "correct" output units should be greater than 0.5 and the responses of "incorrect" output units should be less than 0.5.

[8] The average number of iterations can not be compared, as Peterson and Hartman updates

TABLE 5 HERE

3.4 Usefulness of different error measures

In most of the studies reported above, we had used the mean-squared error measure. When the output nodes are sigmoidal, this error function has an upper and lower bound and may contain multiple minima even for a single layer network (no hidden units). Alopex can be used for minimizing arbitrary error measures. In this section we demonstrate the advantage of using an information theoretic (log) error measure. The classification error in this case is defined as:

$$E = \sum_i target_i \log \left(\frac{target_i}{output_i}\right) + (1 - target_i) \log \left(\frac{1-target_i}{1-output_i}\right) \qquad (10)$$

where the targets for the output units are either 0 or 1.[9] For a network with one layer of connections (no hidden units), and containing sigmoid non-linearities at output nodes, this error function has been shown to contain only a single minimum (Unnikrishnan, Hopfield, and Tank, 1991).

Table 6 shows the average number of iterations (over 100 trials) taken by networks using the squared and log errors to solve the XOR, parity, and encoder problems. The number of times these networks failed to converge after 20,000 iterations, are also shown in this table. For the XOR problem, a network using the log error got 'stuck' during 4% of the trials while a network using the squared error got stuck during 19% of the trials. A network using back-propagation, and hence the squared error, got stuck during 14% of the trials.

FIGURE 3 (a-e) AND TABLE 6 HERE

The improved performance of networks using the log error is due to the fact that these error surfaces are much smoother and contain fewer local minima. Figure 3a shows the network used for the XOR problem and Figs. 3b-e show the error surfaces around the solution point. The surfaces are plotted with respect to pairs of weights, holding the other weights at their final, converged values. We can see that the surfaces for the log error are much smoother than those for the squared error.

---

the weights after 5 patterns are presented, while we update the weights after all the 100 patterns are presented.

[9] Since derivatives of transfer functions are not explicitly calculated in Alopex, targets for learning can be 1.0 or 0.0.

Networks using the log error always converged faster during our experiments. For example, the third MONK's problem was solved by a network using the log error after only 665 iterations, while a network using the squared error took 1000 iterations. This is also evident in the data shown in Tbl. 5 for the symmetry problem. Networks using the log error consistently converged faster (and generalized a little better).

3.5 Using the 'annealing schedule' to reach global minimum

The annealing schedule described in Eq. (6) automatically controls the randomness of the algorithm and it has been successfully used on many occasions to reach global minima of error surfaces. Figure 4 illustrates a case for the XOR network shown in Fig. 3a. Alopex was used to minimize the log error. The path taken by the algorithm to reach the solution point is plotted over the error surface with respect to two of the weights. The algorithm had to overcome several local minima to reach the global minimum. (These minima are not completely evident in the figure as the other weights are held at their optimum values for plotting the error surface. These weights were changing during learning.)

FIGURE 4 HERE

3.6 Scaling properties of Alopex

The ability of Alopex to learn in networks with large number of output classes was investigated using encoder problems of different sizes. Table 7 shows average number of iterations in 25 trials. A network using the squared error could not solve problems bigger than 8 bits, but one using the log error could successfully learn problems up to 32 bits long that we attempted. The error per bit during these learning experiments are shown in Fig. 5. These results show that with appropriate error measures, Alopex can be used in networks with large numbers of output nodes.

FIGURE 5 AND TABLE 7 HERE

3.7 Learning in networks with feedback

Conventional feed-forward networks have limited ability to process real-time temporal signals, model dynamical systems, or control them. We investigated the ability of the Alopex algorithm for training recurrent networks that could be used more effectively for such applications. Three-layered networks with totally interconnected hidden layers (including self loops) were used to solve temporal XOR problems with

various delays. The task is to make the network output at time t, the XOR of the input at time $t-\tau$ and the input at time $t-(\tau+1)$. For this, the network needs to store values from $\tau+1$ time-steps in the past.

A randomly generated, 3000 bits long string was used for training and another 100 bits long string was used for testing. Alopex was used to minimize the squared error. A network with two hidden units (1-2-1 network) was able to learn the $\tau = 0$ problem in 6,000 iterations. The $\tau = 1$ problem was learned by a 1-4-1 network in 4,668 iterations and the $\tau = 2$ problem was learned by a 1-6-1 network in 27,000 iterations. Figure 6b shows the output of the last network along with the test data and Fig. 7 shows the average error per pattern for the three networks during learning.

FIGURE 6 (a-b) AND FIGURE 7 HERE

4. Neurobiological Connection

In this paper, we have presented Alopex as an algorithm for artificial neural networks, It was originally developed for modeling aspects of brain functions and the following three characteristics make it ideal for these purposes:

(i) it is able to handle hierarchical networks with feedback;

(ii) it is a correlation based algorithm; and

(iii) it is a stochastic algorithm.

The mammalian sensory systems are organized in a hierarchic fashion and there are extensive interconnections between neurons within a layer and between neurons in different layers (Van Essen, 1985). During development, some of these feedback connections are established even before the feed-forward connections (Shatz, *et al.*, 1990). We have extensively used simulations of multi-layer networks with feedback to investigate the dynamics of sensory information processing and development (Harth, and Unnikrishnan, 1985; Harth, Unnikrishnan, and Pandya, 1987; Harth, Pandya, and Unnikrishnan, 1990; Unnikrishnan, and Nine, 1991; Janakiraman, and Unnikrishnan, 1992; Janakiraman, and Unnikrishnan, 1993; Unnikrishnan, and Janakiraman, 1992; Nine, and Unnikrishnan, 1993; Unnikrishnan, and Nine, 1993). In all these studies, Alopex is used as the underlying computational algorithm.

In the nervous system, mechanisms such as the NMDA receptors are capable of computing temporal correlations between inputs in a natural fashion (Brown, Kairiss, and Keenan, 1990). Computer simulations of known neural circuitry in the mammalian

visual system have demonstrated the capability of these mechanisms for carrying out Alopex (Sekar, and Unnikrishnan, 1992; Unnikrishnan, and Sekar, 1993).

There is considerable randomness in the responses of neurons (VerVeen, and Derksen, 1965). Stochastic algorithms like Alopex takes this aspect of the nervous system into consideration.

5. Discussion

Our simulations show that Alopex is a robust, general purpose learning algorithm. By avoiding explicit gradient calculations, it overcomes many of the difficulties of current methods. We have used the *same* algorithm to train feed-forward and recurrent networks and for solving a large class of problems like XOR, parity, encoder, and temporal XOR. With appropriate error measures, it is able to learn up to 32-bit encoder problems. Our results on the MONK's problems are the best ones reported in literature. The generalization results on the 4x4 symmetry problems, using a fixed training set, are better than the ones quoted for BP and MFT. Results on the switching problem shows that the network takes comparable number of iterations to solve this task as taken by reinforcement learning or learning automata. Other recent studies, reported elsewhere, has shown the applicability of Alopex for solving diverse problems such as recognition of underwater sonar targets and handwritten digits (Venugopal, Pandya, and Sudhakar, 1991 & 1992a), and control of non-linear dynamics of an underwater vehicle (Venugopal, Pandya, and Sudhakar, 1992b).

A continuous-time version of Alopex, using differentials instead of differences, and integrals instead of sums, has been developed recently (E. Harth, personal communication). It has been implemented in analog hardware and used for a variety of adaptive control applications. Analog VLSI implementations of these circuits may make real-time learning in neural networks possible.

The algorithm uses a single scalar error information to update all the weights. This may pose a problem in networks with large (hundreds) number of output units and in networks with large number of hidden layers. Learning may have to be subdivided in these networks. A preliminary mathematical analysis of convergence properties of Alopex has been done and will be presented elsewhere (Sastry, and Unnikrishnan, 1993).

Finally we would like to say that it was the long history of Alopex in brain modeling that prompted us to investigate it as a learning algorithm for artificial neural networks. We believe that, after all, knowledge of biological neural functions would be useful in developing effective learning algorithms for artificial neural networks.

Acknowledgements:

The authors wish to thank P. S. Sastry for many useful discussions and the action editor for pointing out some of the relevant literature. The clarity of this manuscript has been greatly improved by comments from Jim Elshoff, Erich Harth, Don Jones, P. S. Sastry, R. Uthurusamy, Wayne Wiitanen, and Dick Young. Milind Pandit and Harmon Nine helped us with some of the simulations and Tom Boomgaard helped us with some of the figures. Part of this work was done when KPV was a visitor at GM Research Labs.

References

Alspector, J., R. Meir, B. Yuhas, A. Jayakumar, and D. Lippe, 1993. A parallel gradient descent method for learning in analog VLSI neural networks, *in* Advances in neural information processing systems, Morgan Kaufman (*in press*).

Barto, A. G., 1985. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiol.*, 4, 229-256.

Barto, A.G., 1986. Game-theoretic cooperativity in networks of self-interested units, *in* Neural Networks for computing, J.S. Denker(Ed.), American Institute of Physics.

Barto, A. G., R. S. Sutton, and P. S. Brouwer, 1981. Associative search network: a reinforcement learning associative memory. *Biol. Cybern.*, 40, 201-211.

Barto, A. G., and M. I. Jordan, 1987. Gradient following without back-propagation in layered networks. *in* Proc. IEEE First Ann. Intl. Conf. Neural Nets., pp. II629-II636.

Baum, E. B., and F. Wilczek, 1988. Supervised learning of probability distributions by neural networks, *in* Neural information processing systems, D. Z. Anderson(Ed.), American Institute of Physics.

Brown, T. H., E. W. Kairiss, and C. L. Keenan, 1990. Hebbian synapses: Biophysical mechanisms and algorithms. *Ann. Rev. Neurosci.*, 13, 475-512.

Dembo, A., and T. Kailath, 1990. Model-free distributed learning. *IEEE Trans. Neural Networks.* , vol. 1, pp. 58-70.

Draper, C. S., and Y. T. Li, 1951. *Principles of optimalizing control systems and an application to the internal combustion engine*, ASME Publications.

Harth, E., A. S. Pandya, and K. P. Unnikrishnan, 1986. Perception as an optimization process. *in* Proc. IEEE Conf. CVPR., pp. 662-665.

Harth, E., A. S. Pandya, and K. P. Unnikrishnan, 1990. Optimization of cortical responses by feedback modification and synthesis of sensory afferents. A model for perception and REM sleep. *Concepts in Neurosci.*, 1, 53-68.

Harth, E., and E. Tzanakou, 1974. Alopex: A stochastic method for determining visual

receptive fields. *Vision Res.*, 14, 1475-1482.

Harth, E., and K. P. Unnikrishnan, 1985. Brainstem control of sensory information: a mechanism for perception. *Int. J. Psychophysiol.* , 3, 101-119.

Harth, E., K. P. Unnikrishnan, and A. S. Pandya, 1987. The inversion of sensory processing by feedback pathways: a model of visual cognitive functions. *Science* , 237, 187-189.

Hinton, G.E. 1989. Connectionist learning procedures. *Artificial Intelligence*, 40, 185-234.

Hopfield, J. J. 1987. Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proc. Natl. Acad. Sci. USA* , 84, 8429-8433.

Hornik, K., M. Stichcomb, and H. White, 1989. Multilayer feed-forward networks are universal approximators. *Neural Networks*, 2, 359-366.

Jabri, M., and B. Fowler, 1991. Weight perturbation: an optimal architecture and learning technique for analog VLSI feed-forward and recurrent multilayer networks. *Neural Computation*, 3, 546-565.

Janakiraman, J., and K. P. Unnikrishnan, 1992. A feedback model of visual attention. *in* Proc. IJCNN, III-541 - III-546.

Janakiraman, J., and K. P. Unnikrishnan, 1993. A model for dynamical aspects of visual attention. *in* Proc. Computation and Neural Systems '92 (*in press*).

Kirkpatrick, S., C. Gelatt, and M. P. Vecchi, 1983. Optimization by simulated annealing. *Science* , 220, 671-680.

Minski, M. L., 1954. *Theory of neural-analog reinforcement systems and its application to the brain-model problem*, Princeton University.

Mukhopadhyay, S., and M. A. L. Thathachar, 1989. Associative learning of boolean functions. *IEEE Trans. SMC.* , 19, 1008-1015.

Narendra, K. S., and M. A. L. Thathachar, 1989. *Learning automata: an introduction*, Prentice Hall.

Nine, H. S., and K. P. Unnikrishnan, 1993. The role of subplate feedback in the development of ocular dominance columns. *in* Proc. Computation and Neural Systems '92 (*in press*).

Peterson, C., and E. Hartman, 1989. Explorations of the mean field theory learning algorithm. *Neural Networks* , 2, 475-494.

Rosenblatt, F., 1962. *Principles of neurodynamics*, Spartan books.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986. Learning internal representations by error propagation, *in* Parallel Distributed Processing, vol. 1, MIT Press.

Sastry, P. S., and K. P. Unnikrishnan 1993. On the convergence properties of the Alopex algorithm. *manuscript in preparation*.

Sejnowski, T. J., P. K. Kienker, and G. E. Hinton, 1986. Learning symmetry groups with hidden units: beyond the perceptron. *Physica D* 22, 260-275.

Sekar, N. S., and K. P. Unnikrishnan, 1992. The Alopex algorithm is biologically plausible. *in* Abstr. Learning and Memory meeting, Cold Spring Harbor Laboratory, pp. 50.

Shatz, C. J. *et al*., 1990. Pioneer neurons and target selection in cerebral cortical development. *in* Cold Spring Harbor Symposium on Quantitative Biology, vol. 55.

Thrun, S. *et al*., 1991, The MONK's problems: A performance comparison of different learning algorithms, Carnegie Mellon University, CMU-CS-91-197, December, 1991.

Unnikrishnan, K. P., J. J. Hopfield, and D. W. Tank. 1991. Connected-Digit Speaker-Dependent Speech Recognition Using a Neural Network with Time-Delayed Connections. *IEEE Trans. Signal Proc.* , 39, 698-713.

Unnikrishnan, K. P., and J. Janakiraman, 1992. Dynamical control of attention through feedback pathways: A network model. Soc. Neurosci. Abstr. 18: 741.

Unnikrishnan, K. P., and H. S. Nine, 1991. Cortical feedback to LGN may play a major role in ocular dominance column development. Soc. Neurosci. Abstr. 17: 1135.

Unnikrishnan, K. P., and H. S. Nine, 1993. Role of subplate 'feedback' in the

formation of ocular dominance columns: A model. *submitted* .

Unnikrishnan, K. P., and M. S. Pandit 1991. Learning in hierarchical neural networks with feedback. *in* Abstr. Neural Nets. Comp. Conf., Snowbird, UT.

Unnikrishnan, K. P., A. S. Pandya, and E. Harth, 1987. The role of feedback in visual perception *in* Proc. IEEE First Ann. Intl. Conf. Neural Nets., pp. IV259-IV267.

Unnikrishnan, K. P., and N. S. Sekar, 1993. A biophysical model of the Alopex algorithm. Soc. Neurosci. Abstr. 19: 241.

Unnikrishnan, K. P., and K. P. Venugopal, 1992. Learning in connectionist networks using the Alopex algorithm. *in* Proc. IJCNN I-926 - I-931.

Van Essen, D. C., 1985. Functional organization of primate visual cortex. *in* Cerebral Cortex, Vol 3, A. Peters and E. G. Jones, Eds. Plenum Press, NY.

Venugopal, K. P., A. S. Pandya, and R. Sudhakar, 1991. Continuous recognition of sonar targets using neural networks. *in* Automatic Target Recognition, F. Sadjadi, Ed. Proc. SPIE, vol. 1471, 43-54.

Venugopal, K. P., A. S. Pandya, and R. Sudhakar, 1992a. Invariant recognition of 2-D objects using Alopex neural networks. *in* Applications of Artificial Neural Networks, S. K. Rogers, Ed. Proc. SPIE, vol. 1708 (*in press*).

Venugopal, K. P., A. S. Pandya, and R. Sudhakar, 1992b. A recurrent network controller and learning algorithm for the on-line learning control of autonomous underwater vehicles. *(submitted).*

Verveen, A. A., and D. W. Derksen, 1965. Fluctuations in membrane potential and the problem of coding. *Kybernetik* 2, 152-160.

Whitaker, H. P., 1959. An adaptive system for control of aircraft and spacecraft. *in* Institute for Aeronautical Sciences, paper 59-100.

Widrow, B., and M. A. Lehr, 1990. 30 years of adaptive neural networks. perceptron, madaline, and back-propagation. *Proc. IEEE*, 78, 1415-1442.

Williams, R. J., and D. Zipser, 1989. A learning algorithm for continually running

fully recurrent neural networks. *Neural Computation*, 1, 270-280.

Table Captions

Table 1: Average number of iterations taken by back-propagation and Alopex to solve three different problems. The average was taken over 100 trials with different initial weights. For back-propagation; the learning rate and momentum were 0.9 & 0.7 respectively for XOR, 0.5 & 0.8 respectively for parity, and the same for encoder. For Alopex; δ and N were 0.35 & 20 respectively for XOR, 0.1 & 30 for parity, and 0.05 & 100 for encoder.

Table 2: Number of iterations taken by Alopex, Learning Automata (LA), and Reinforcement Learning (ARP) to solve the four-bit switching problem. See Mukhopadhyay and Thathachar (1989) for a description of the three tasks. The data for LA is taken from the above paper and the ARP data is taken from Barto (1985). Slightly different updating and stopping criteria are used in each method and hence the three can not be compared directly. For Alopex and LA, each task was run 10 times and for ARP one of the tasks were run 30 times. For Alopex, δ was 0.0025 and N was 10.

Table 3: Number of iterations taken by Alopex to solve the switching task for different step-sizes (δ) and error measures. All networks were started with the same initial conditions.

Table 4: Performance of different gradient descent methods and Alopex on MONK's problems. For Alopex, δ was 0.01 and N was 10. Training was terminated when the network had learned to correctly classify all the test samples. (Data for back-propagation and cascade-correlation is from Thrun, *et al.,* 1991.)

Table 5: Average number of training iterations and generalization accuracies for the 4 x 4 mirror symmetry problem. Data for back-propagation (BP) and mean field theory learning (MFT) are taken from Peterson and Hartman (1989). For Alopex, δ was 0.003 and N was 10.

Table 6: Comparison of experiments using different error measures. An experiment was categorized as 'stuck' if it did not converge after 20,000 iterations. The learning parameters used for back-propagation (BP) and Alopex are the same as in Table 1.

Table 7: Average number of iterations (over 25 trials) for encoder problems of

different sizes. δ was 0.005 for the 4-bit and 8-bit networks, 0.004 for the 16-bit network, and 0.001 for the 32 bit network. N was 10 for all the networks.

Figure Captions

Figure 1: The mean square error as a function of iterations for learning the XOR problem with a 2-2-1 network. Back-prop - Back-propagation algorithm; Wt-pert - weight perturbation algorithm; Alpx (sml) - Alopex with a small step size (δ = 0.008); Alpx (lrg) - Alopex with a large step size (δ = 0.03). The plots for back-propagation and weight perturbation are reproduced from Jabri and Flower (1991).

Figure 2: Network responses for the 432 test samples of the third MONK's problem. The network was trained on 122 training samples for 1000 iterations. The test samples are shown on a 18x24 grid, following the convention of Thrun *et al.* (1991). The height of the blocks represent the magnitude of network responses, with the maximum of 1.0 (deep red) and minimum of 0.0 (deep blue). The six marked samples were deliberately mis-classified in the training set. Responses to these samples show the robustness of the algorithm. At this point in training, the network correctly classifies all the 432 samples. See text and Thrun *et al.* (1991) for details of MONK's problems.

Figure 3: a) Schematic of the network used for solving the XOR problem. The weights in the network are labeled w_1 through w_6. Individual neurons contains sigmoidal nonlinearities. b) -e) Error surfaces around the solution point (center) for square (b & d) and log (c & e) errors. Figures b & c show the surfaces with respect to w_3 and w_6. Figures d & e show them with respect to w_4 and w_5. All the other weights (and biases) were held at their optimum values for plotting the surfaces.

Figure 4: Path taken by the Alopex algorithm to reach the global minimum of error for XOR. It is superimposed on the final error surface with respect to w_3 and w_6. The actual error surface encountered by the algorithm during learning is different from this final one since the other weights, which are held at their optimum value for this plot, changes during learning.

Figure 5: Error per bit for different encoder problems as a function of learning iterations. Alopex was used to minimize the log error.

Figure 6: a) Schematic of networks used to solve the temporal XOR problems. They contained a totally interconnected (including self loops) hidden layer. b) Output of a network (solid line) trained to solve the $\tau = 2$ temporal XOR problem. It is superimposed on the target sequence (dotted line). We can see that the two plots are almost identical. The network has learned to solve this problem with high accuracy.

Figure 7: The average error per bit for the three networks trained to solve temporal XOR problems, as a function of learning iterations. The three networks are of different sizes. δ was 0.005 and N was 10 for all three networks.

TABLE 1

	BP	**Alopex**
XOR (2-2-1 network)	1175	478
Parity (4-4-1 network)	595	353
Encode (4-2-4 network)	2676	3092

TABLE 2

	Task1			Task2			Task3			Overall		
Algorithm	**Lo**	**Hi**	**Av**	**Lo**	**Hi**	**Av**	**Lo**	**Hi**	**Av**	**Lo**	**Hi**	**Av**
Alopex	6306	14532	9851	7141	16619	11249	6206	13872	9948	6206	16619	10349
LA	10659	15398	12628	10748	15398	12641	10403	12939	11917	10403	15398	12395
ARP	*	*	*	*	*	*	*	*	*	37500	350000	133149

TABLE 3

Step-size	Sqr Error	Log Error
0.01	12,382	7,802
0.0075	10,421	6,327
0.005	9,416	15,385
0.0025	6,341	12,910
0.001	11,666	9,289
0.0005	13,657	14,714
Average	10,647	11,071

TABLE 4

Learning algorithm	Problem #1	Problem #2	Problem #3
BP	100%	100%	93.1%
BP with weight decay	100%	100%	97.2%
Cascade correlation	100%	100%	97.2%
Alopex	100%	100%	100%

TABLE 5

	PROB 0.5		PROB 0.4		PROB 0.6	
Learning Algorithm	**Aver. no: of iters.**	**Gen. accuracy(%)**	**Aver. no: of iters.**	**Gen. accuracy(%)**	**Aver. no: of iters.**	**Gen. accuracy(%)**
Alopex (Log)	4417	74.0	3966	79.3	4412	79.8
Alopex (Square)	8734	73.3	7735	73.7	6141	76.1
Mean Field Theory	*	70	*	63	*	62
Back-propagation	*	71	*	64	*	63

TABLE 6

	XOR (2-2-1 NETWORK)		PARITY (4-4-1 NETWORK)		ENCODER (4-2-4 NETWORK)	
Error Measure	**Av. no: of iters.**	**% of times 'stuck'**	**Av. no: of iters.**	**% of times 'stuck'**	**Av. no: of iters.**	**% of times 'stuck'**
Log	1484	4	297	0	2996	0
Square	478	19	353	0	3092	1
BP (Square)	1175	15	595	0	2676	0

TABLE 7

Size of input and network	Average number of iterations	
	Square error	Log error
4 Bits (4-2-4 net)	1734	1775
8 Bits (8-3-8 net)	6771	13,077
16 Bits (16-4-16 net)	(No convergence)	20,280
32 Bits (32-5-32 net)	(No convergence)	75,586

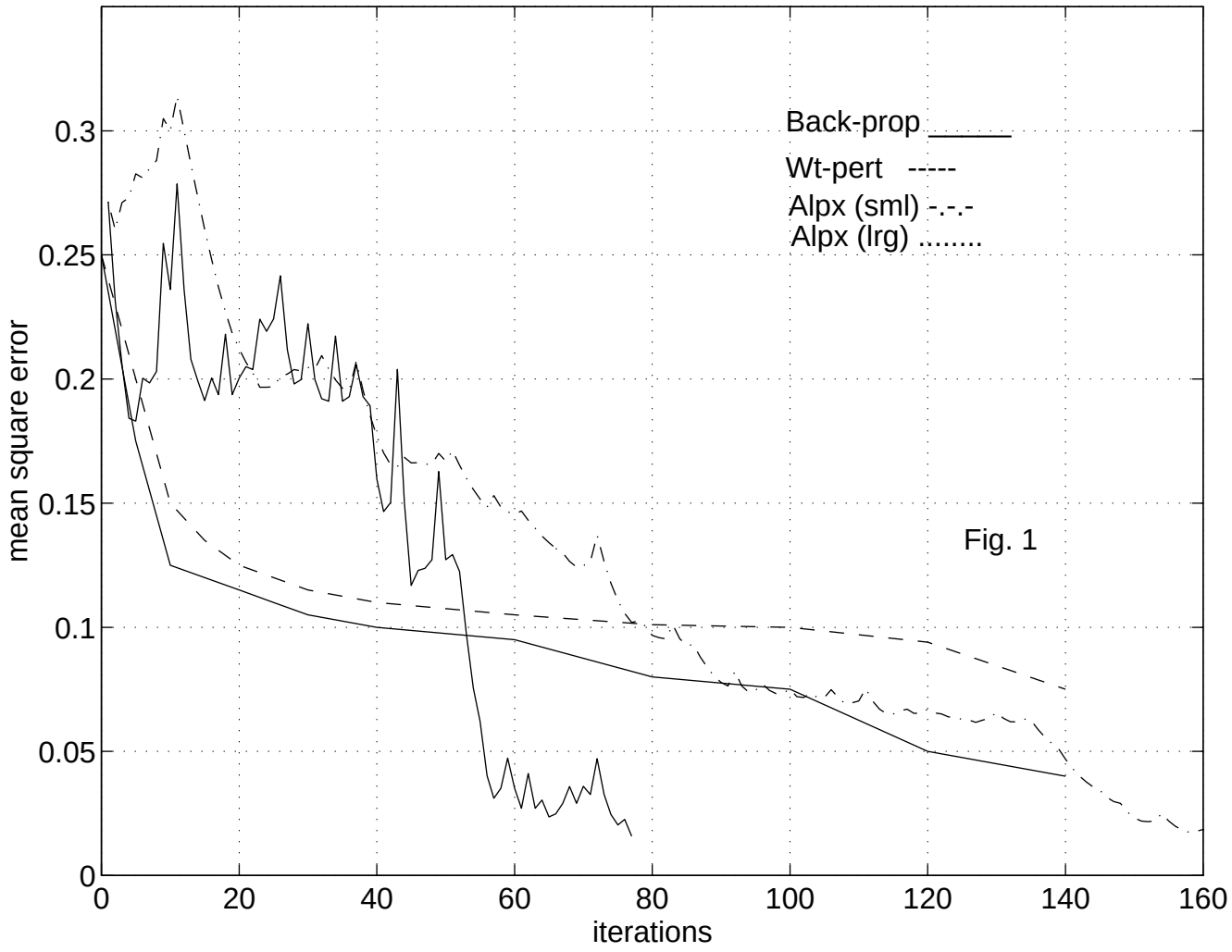
Back-prop
Wt-pert -----
Alpx (sml) -.-.-
Alpx (lrg) .....
mean square error
iterations
0
0.05
0.1
0.15
0.2
0.25
0.3
0
20
40
60
80
100
120
140
160

Fig. 1

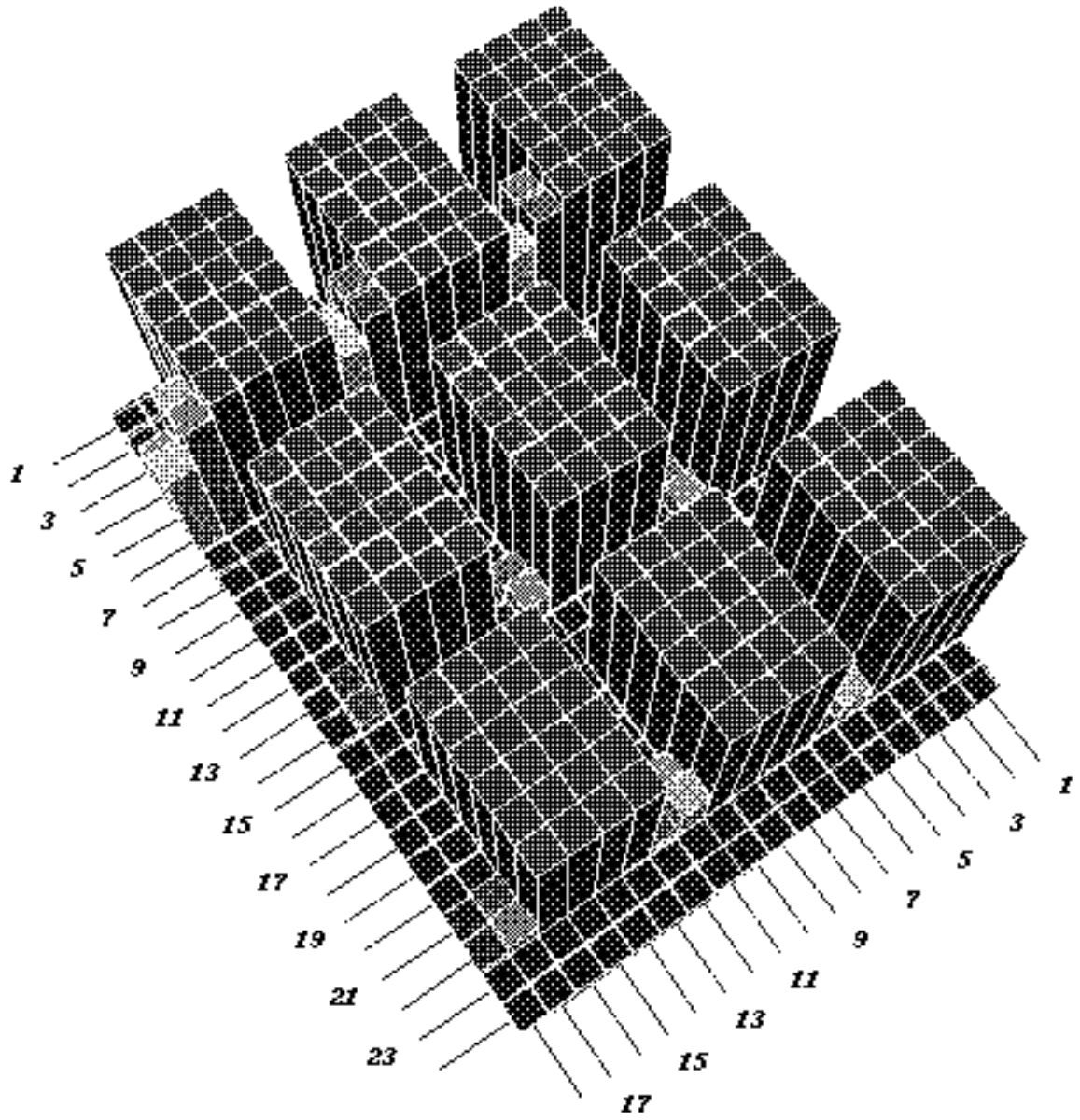

Fig. 2

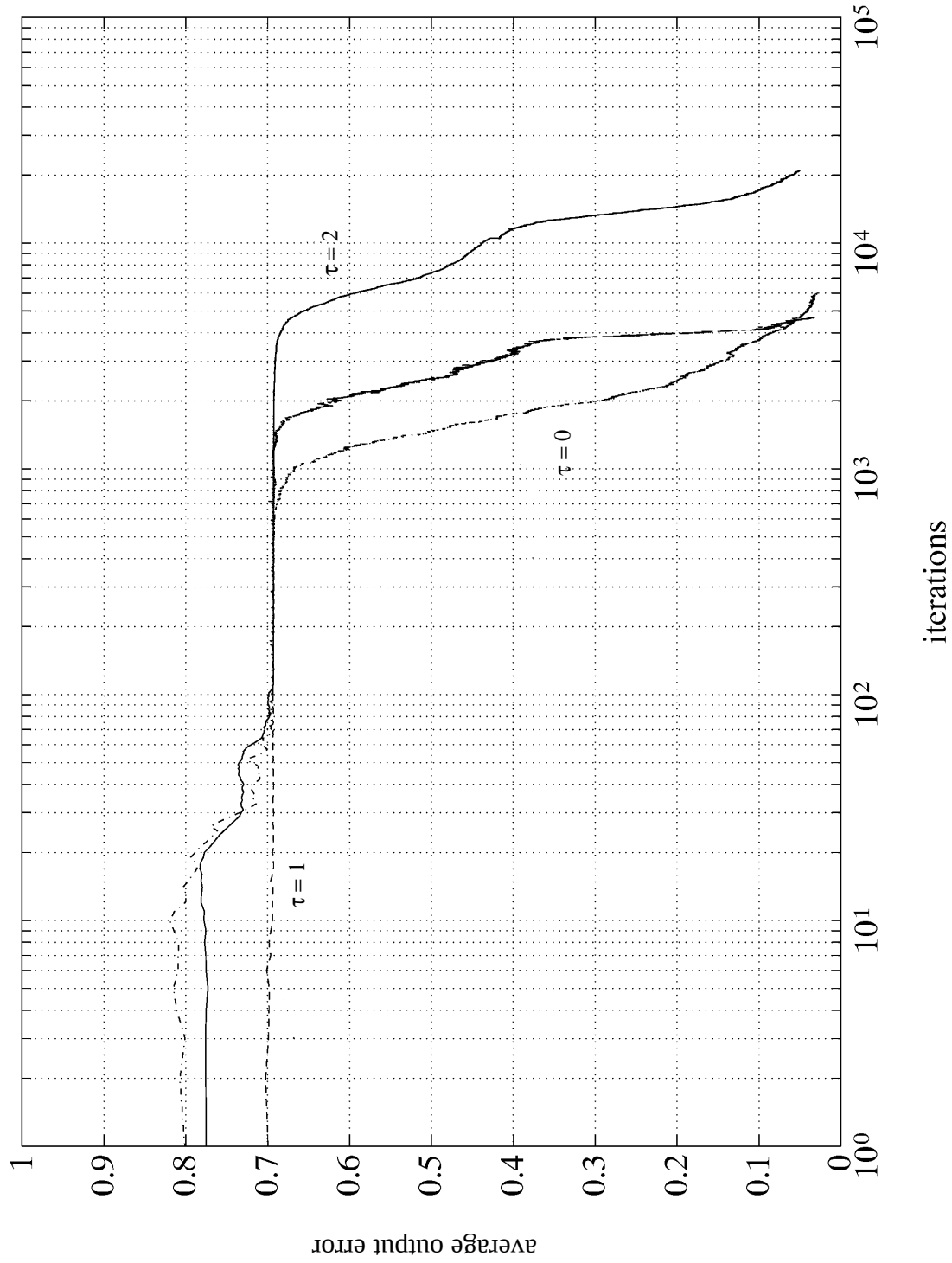
average output error
iterations
τ = 2
τ = 1
τ = 0
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10^0
10^1
10^2
10^3
10^4
10^5

Fig. 7.

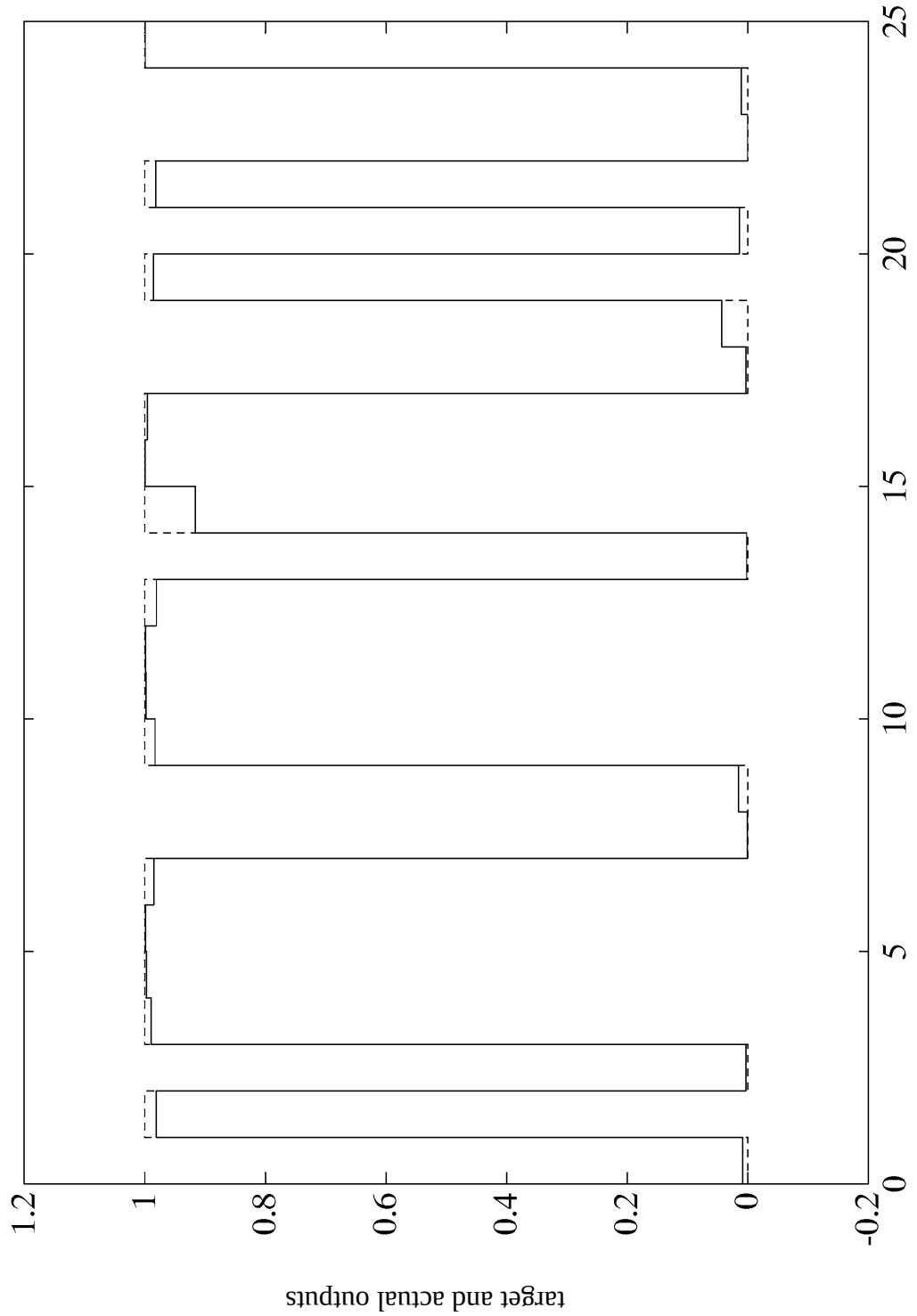
target and actual outputs
time steps
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25

Fig. 6. (b).

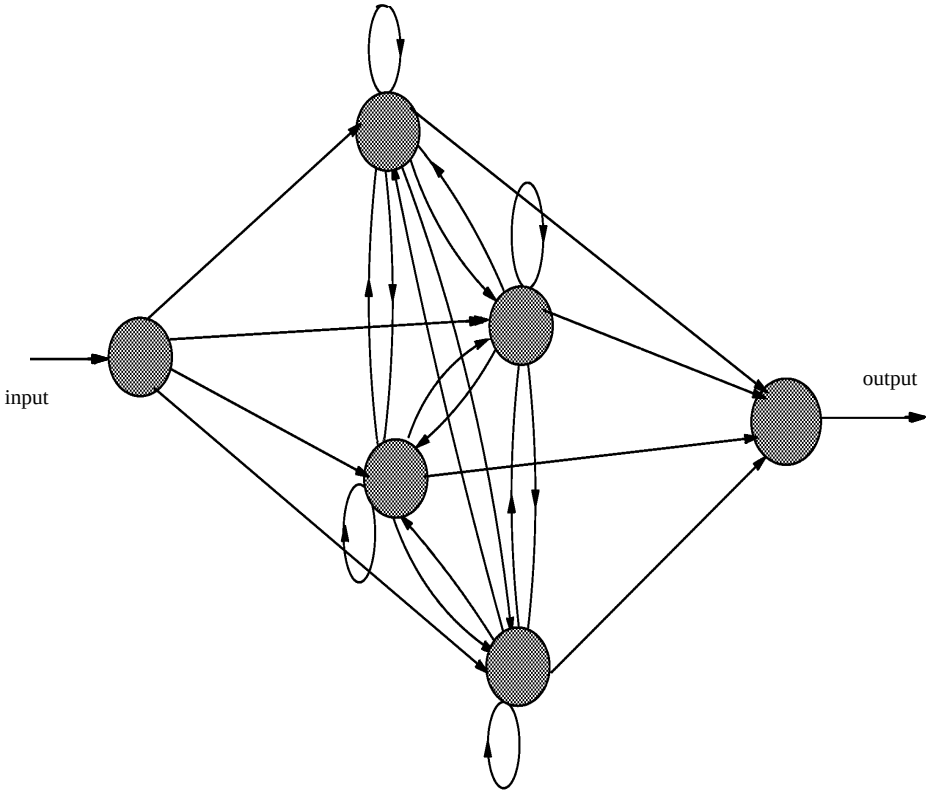

Fig. 6. (a).

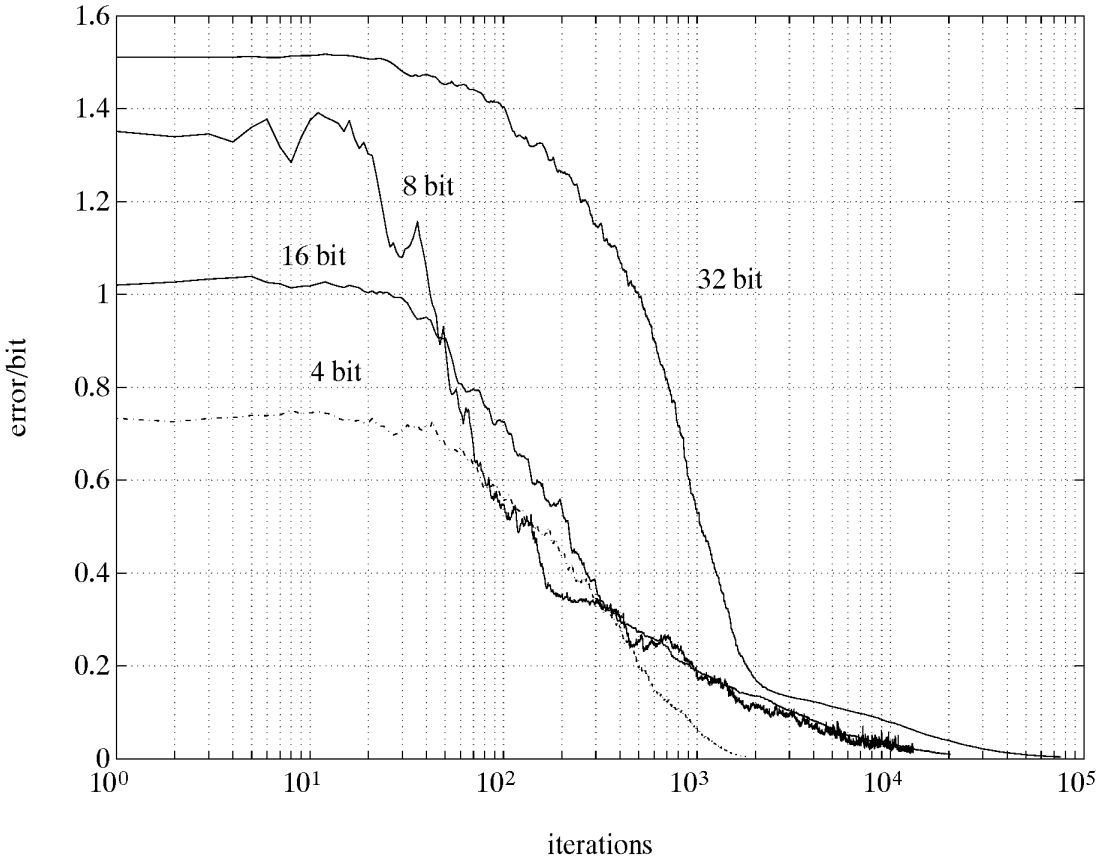
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
error/bit
8 bit
16 bit
32 bit
4 bit
10^0
10^1
10^2
10^3
10^4
10^5
iterations

Fig. 5.

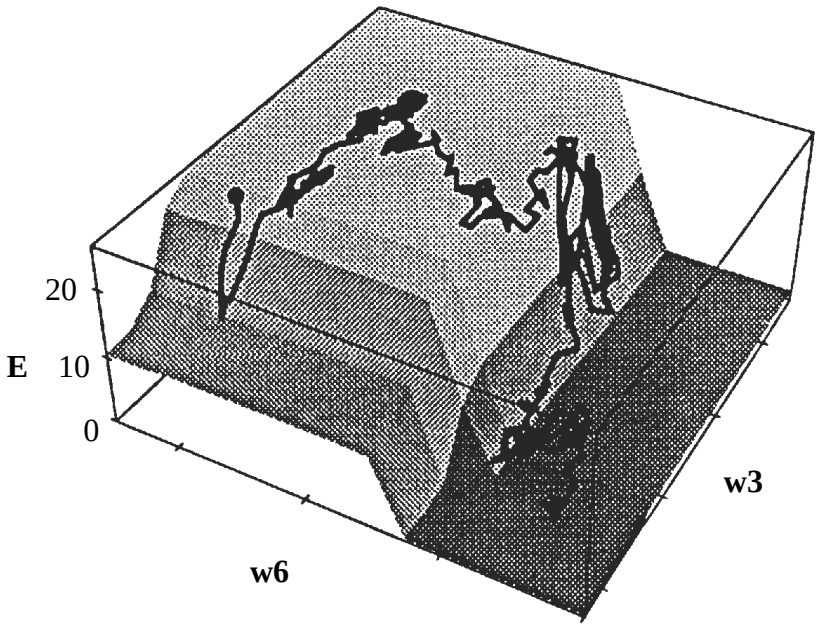

Fig. 4.

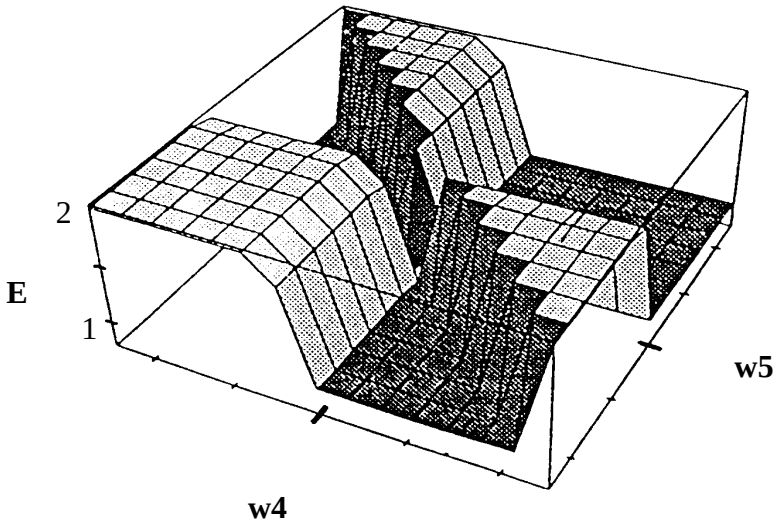

Fig. 3. (d).

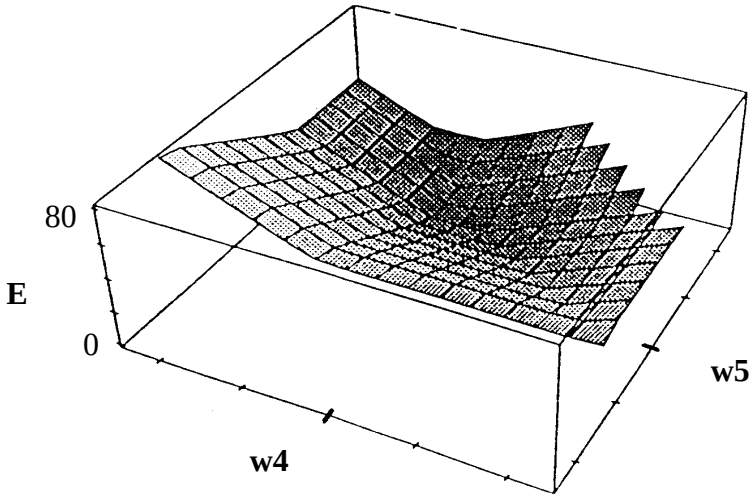

Fig. 3. (e).

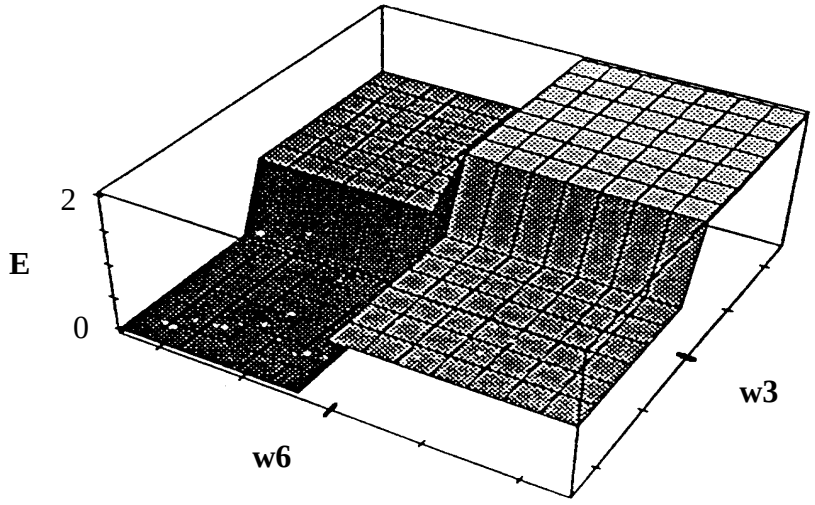

Fig. 3. (b).

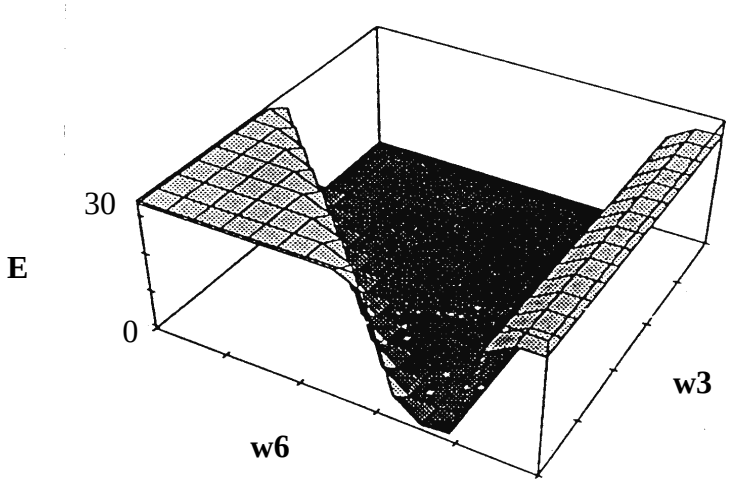

Fig. 3. (c).

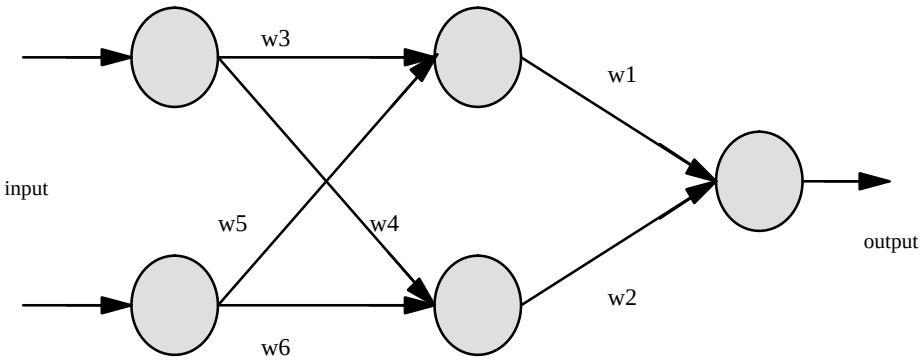

Fig. 1. (a)