

Using Quasirandom Numbers in Neural Networks

Peter G. Anderson Roger S. Gaborski Ming Ge Sanjay Raghavendra
Mei-Ling Lung

Abstract

We present a novel training algorithm for a feed forward neural network with a single hidden layer of nodes (i.e., two layers of connection weights). Our algorithm is capable of training networks for hard problems, such as the classic two-spirals problem.

The weights in the first layer are determined using a quasirandom number generator. These weights are frozen—they are never modified during the training process.

The second layer of weights is trained as a simple linear discriminator using methods such as the pseudoinverse, with possible iterations.

We also study the problem of reducing the hidden layer: pruning low-weight nodes and a genetic algorithm search for good subsets.

Neural Network Structure

Our goal is to construct feed-forward neural networks, such as Fig. 1, which can solve low dimensional two-class discrimination problems, such as the famous two-spirals problem [5]. Solving means that the network must learn the training set and generalize (interpolate and extrapolate) that learning in a reasonable way. The training data for the two-spirals problem, along with the generalization ability of one of our networks, is shown in Fig. 2. In most of our experiments, we use 200 points on each spiral, 400 training exemplars in all. The points lie in a $2a \times 2a$ square in R^2 centered at the origin. We have experimented with various values of a in the range 1.0–4.0.

A feed forward network to classify those points has three inputs, x_0 , x_1 , and x_2 . The pair (x_1, x_2) represents the input data to be classified, and $x_0 \equiv 1$ is a fixed bias input to allow us to use zero as the threshold for the cells in the next layer. The single hidden layer consists of N nodes, denoted $\mathbf{y} = (y_1, y_2, \dots, y_N)$. The output is the scalar, z . The transfer function of the

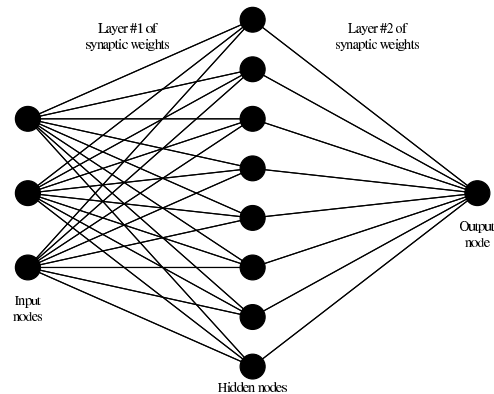


Figure 1: A 3–8–1 feed-forward neural network.

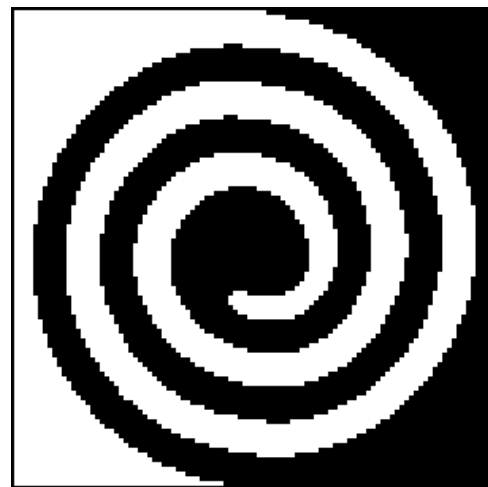


Figure 2: The two-spirals problem. A network was trained with 230 hidden units on 400 exemplars in the $[-4.0, 4.0]^2$ square. The coloring shows the sign of the network's outputs at points covering that square. The maximum absolute error for the training exemplars is 0.028.

network is

$$z = \sigma_2 \left(\sum_{k=1}^N v_k \sigma_1 \left(\sum_{j=0}^2 w_{kj} x_j \right) \right)$$

The nonlinearity, σ_1 , at the hidden node can be one of several; we have used: *tanh*, *sine*, the step function *algebraic sign*, a piecewise linear sigmoid, and a piecewise rational sigmoid $x/(1+|x|)$. The nonlinearity, σ_2 , at the single output node is *algebraic sign* (we generally ignore σ_2 , that is, treat it as the identity function, and train networks with targets ± 1).

Quasirandom Numbers for W

In our solution, the size, N , of the hidden layer is large (using $\sigma_1 = \tanh$, we use at least $N = 168$ to solve the nested, three-revolutions, two-spirals problem in the $[-4.0, 4.0]^2$ square). The connection weights matrix, $\mathbf{W} = \{w_{kj}\}$, from the input to the hidden nodes is computed once and then frozen. Traditionally, feed-forward networks' weights are initialized to small pseudorandom values prior to back propagation training, with the expectation that a point in weight-space so chosen is sufficiently close to a point that solves the problem, so that the gradient descent search will be able to zero in on that desired point [6]. A problem with this approach is that pseudorandomly chosen points will cover the space unevenly: they will form clumps and leave gaps. Our approach is to use "quasirandom" numbers [11] to achieve even coverage, and, if enough hidden nodes are supplied in our network, a solution network can be uncovered using the weights *as given* rather than found by adjusting the weights. The connection weights, (w_{k0}, w_{k1}, w_{k2}) , from the input to hidden node y_k , are points in the cube $[-1.0, 1.0]^3$ in three-space. We generate them in an "equidistributed" manner as developed experimentally in terms of "linear pixel shuffling" [1, 3].

Our recipe for the weights, (w_{k0}, w_{k1}, w_{k2}) , is as follows. Let α_0 be the unique positive real solution of $\alpha_0(\alpha_0 + 1)^3 = 1$. Let $\alpha_1 = \alpha_0(\alpha_0 + 1)$ and $\alpha_2 = \alpha_0(\alpha_0 + 1)^2$. So,

$$(\alpha_0, \alpha_1, \alpha_2) = (0.380278, 0.524889, 0.724492)$$

The point in three-space, $(\alpha_0, \alpha_1, \alpha_2)$, appears to be a suitable generalization of $\tau = (\sqrt{5}-1)/2 = 0.618034$ (τ satisfies $\tau(\tau + 1) = 1$; $\tau + 1$ is the Golden Mean), in one-space, in the following sense. The irrational τ is maximally difficult to approximate with rationals [4]; therefore, the fractional parts of its first N multiples, $\{k\tau\} = k\tau - [k\tau]$, for $k = 0, \dots, N-1$, are very

smoothly distributed in the interval $[0.0, 1.0]$ (this sequence is particularly suitable for Monte Carlo integration [9]). We use the points $(\{k\alpha_0\}, \{k\alpha_1\}, \{k\alpha_2\})$, for $k = 0, \dots, N-1$, as N smoothly distributed points in $[0.0, 1.0]^3$. Our first-layer weights are taken as

$$w_{kj} = 2\{k\alpha_j\} - 1$$

for $j = 0, 1, 2$ and $k = 0, \dots, N-1$.

A Linear Solution for V

Denote the training exemplars for the problem at hand by $(\mathbf{x}^{(m)}, z^{(m)})$, for $m = 1, \dots, M$ (for our two-spirals problem, M is 400, with 200 points on each spiral). The first layer of weights is frozen, so we evaluate only once the activation values of the hidden nodes corresponding to the training exemplars:

$$\mathbf{y}^{(m)} = \sigma_1 \mathbf{W} \mathbf{x}^{(m)}, \quad m = 1, \dots, M$$

(the $\mathbf{y}^{(m)}$ are column N -vectors) and obtain the second layer of weights as a least squares (i.e., linear) approximation to the M problems

$$z^{(m)} = \mathbf{V} \mathbf{y}^{(m)}$$

using the pseudoinverse of \mathbf{Y} , the $N \times M$ matrix whose m -th column is $\mathbf{y}^{(m)}$. Let \mathbf{Z} be the row M -vector whose m -th element is $z^{(m)}$. The matrix problem to solve is

$$\mathbf{V} \mathbf{Y} = \mathbf{Z}$$

The matrix \mathbf{Y} is not square and invertible, so use

$$\mathbf{V} = \mathbf{Z} \mathbf{Y}^T (\mathbf{Y} \mathbf{Y}^T)^{-1} = \mathbf{Z} \mathbf{Y}^\#$$

calling $\mathbf{Y}^\#$ the pseudoinverse of \mathbf{Y} . This solution to \mathbf{V} minimizes the total squared error:

$$\hat{E} = \sum_{m=1}^M (\mathbf{V} \mathbf{y}^{(m)} - z^{(m)})^2$$

rapidly giving us a feed-forward neural network capable of solving some hard problems.

For the 400 points on two spirals in the square $[-1.0, 1.0]^2$ and $\sigma_1 = \tanh$, this method yields networks that learn the training set as follows:

1 revolution	22 hidden nodes	0 errors
2 revolution	54 hidden nodes	0 errors
3 revolution	116 hidden nodes	0 errors
4 revolution	400 hidden nodes	1 error

We compared the quasirandom values for \mathbf{W} with the pseudorandom numbers as supplied with the J programming language (our implementation language for most experiments) [10]. The results for the pseudorandom \mathbf{W} are as follows:

1 revolution	18 hidden nodes	0 errors
2 revolution	60 hidden nodes	0 errors
3 revolution	385 hidden nodes	0 errors

The Iterated Pseudoinverse

We have demonstrated that the quasirandom numbers used for the weights \mathbf{W} enable us to construct mappings from 3-space to N -space that permit a straightforward least squares errors technique to easily locate a separating hyperplane for the elements of two classes, such as the points in two spirals in 2-space. However, the two point sets may be linearly separable, yet the least squares technique cannot locate a separating hyperplane. The summed squares of network errors, prior to applying σ_2 , is a good error measure to minimize, but it is not the best one. Rather than \hat{E} , we should minimize

$$E = \sum_{m=1}^M (\sigma_2 \mathbf{V} \mathbf{y}^{(m)} - z^{(m)})^2$$

to locate a hyperplane that has the *smallest number* of errors. To achieve that result (often to the point of no errors), we employ the following iterative technique, where the nonlinear function σ_2 is *algebraic sign*. (This was described in [2] for training a digit recognition system.)

1. Determine the following matrices:

\mathbf{W} and \mathbf{Y} as described above

$$\mathbf{A} := \mathbf{Z} \mathbf{Y}^T$$

$$\mathbf{B} := \mathbf{Y} \mathbf{Y}^T$$

$$\mathbf{V} := \mathbf{A} \mathbf{B}^{-1}$$

2. Iterate the following training epoch (until success or exhaustion):

for $m := 1$ to M

if $z^{(m)} \neq \sigma_2(\mathbf{V} \mathbf{y}^{(m)})$ then

retrain exemplar m as follows:

$$\mathbf{A} := \mathbf{A} + z^{(m)} \mathbf{y}^{(m)T}$$

$$\mathbf{B} := \mathbf{B} + \mathbf{y}^{(m)} \mathbf{y}^{(m)T}$$

recalculate $\mathbf{V} := \mathbf{A} \mathbf{B}^{-1}$

The main loop in this procedure should be iterated 10–50 times, or until the error decreases to zero. The retraining of exemplar m is shown above in case it gets the wrong answer with the present net (i.e., the present \mathbf{V}). This process can also improve the performance of a network that produces the correct answer for every training exemplar in terms of the correct algebraic sign

for $\mathbf{V} \mathbf{y}^{(m)}$: retrain exemplar m in case the absolute error prior to applying σ_2 is too large; that absolute error is $|z^{(m)} - \mathbf{V} \mathbf{y}^{(m)}|$. Such retraining may train networks with smaller hidden layers and whose generalization ability is improved.

We experimented with the iterated pseudoinverse with a sequence of four two-spiral problems: 1-, 2-, 3-, and 4-revolutions. The data points were in the square $[-1.0, 1.0]^2$. The computing effort to achieve zero errors with the least number of hidden nodes was as follows:

1 revolution	16 hidden nodes	5 iterations
2 revolution	44 hidden nodes	22 iterations
3 revolution	88 hidden nodes	33 iterations
4 revolution	271 hidden nodes	50 iterations

Fig. 3 shows the efficacy of this iteration.

Reducing the Hidden Layer

The networks that we have described above solve a wide variety of problems at the cost of using a very large hidden layer of N cells, many of which are of little value in any particular problem. Fahlman [5] describes feed-forward networks with as few as 12 hidden nodes that solve the two spirals with three revolutions (Fahlman’s hidden nodes are not in a single hidden layer but in a cascaded arrangement with one node in each layer; every node receives input from *all* the nodes that precede it). It is rumored that a network with one hidden layer with 30 nodes can be trained using back propagation to solve this problem.

We have investigated a small variety of methods of determining workable small subsets of the hidden layer of N elements: locating and removing the cells associated with small values of v_k (measured in absolute value) and using a genetic algorithm to search the space of subsets.

Eliminating Low-Weight Nodes

Since neural networks are reputed to be robust and can continue to function properly when their weights are slightly altered, we suspected that the nodes associated with those small values could simply be removed from the network with little damage to the network’s performance. This can be accomplished by identifying the elements of \mathbf{V} in order of their absolute value, smallest to largest, and successively setting those weights to zero and testing the performance of the resulting network. The disappointing results of this experiment—the network is almost immediately drastically broken—along with the more promising results of a related experiment, are shown in Fig. 4. (These experiments used spirals with three revolutions in the square $[-4.0, 4.0]^2$.)

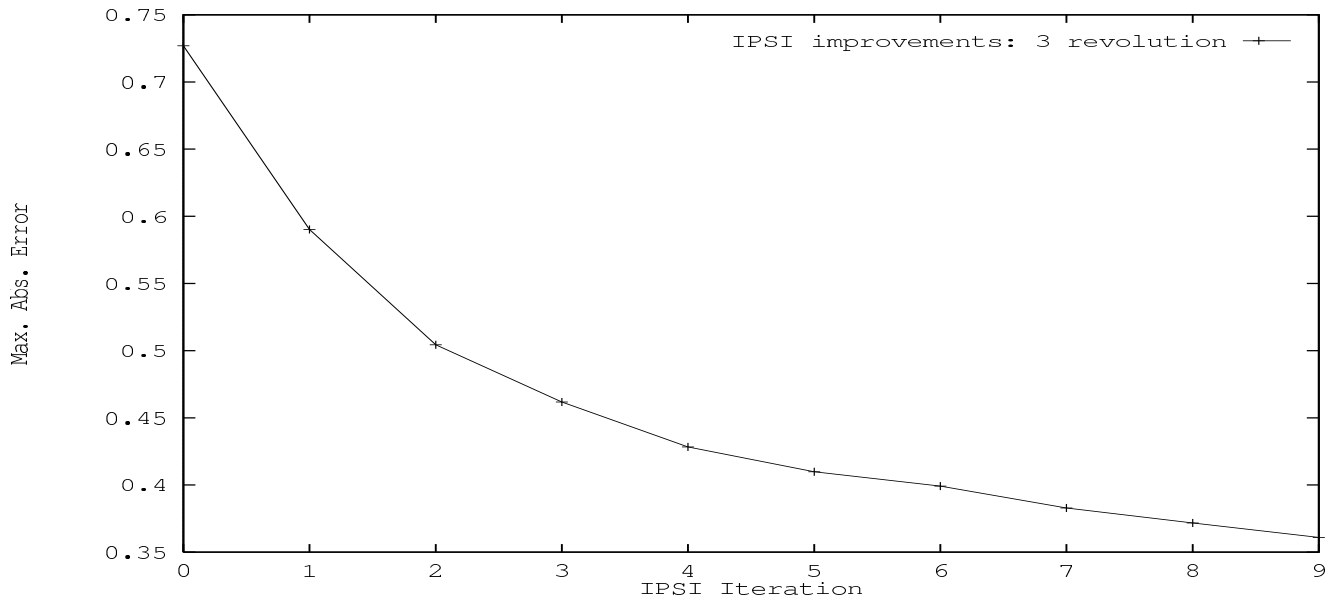


Figure 3: Using the iterated pseudoinverse to improve a network. The network solves the three-revolution problem with no errors; successive iterations reduce the maximum absolute error (the vertical axis). The network had 169 hidden nodes, the fewest needed to solve this problem in $[-4.0, 4.0]^2$ with the simple linear method.

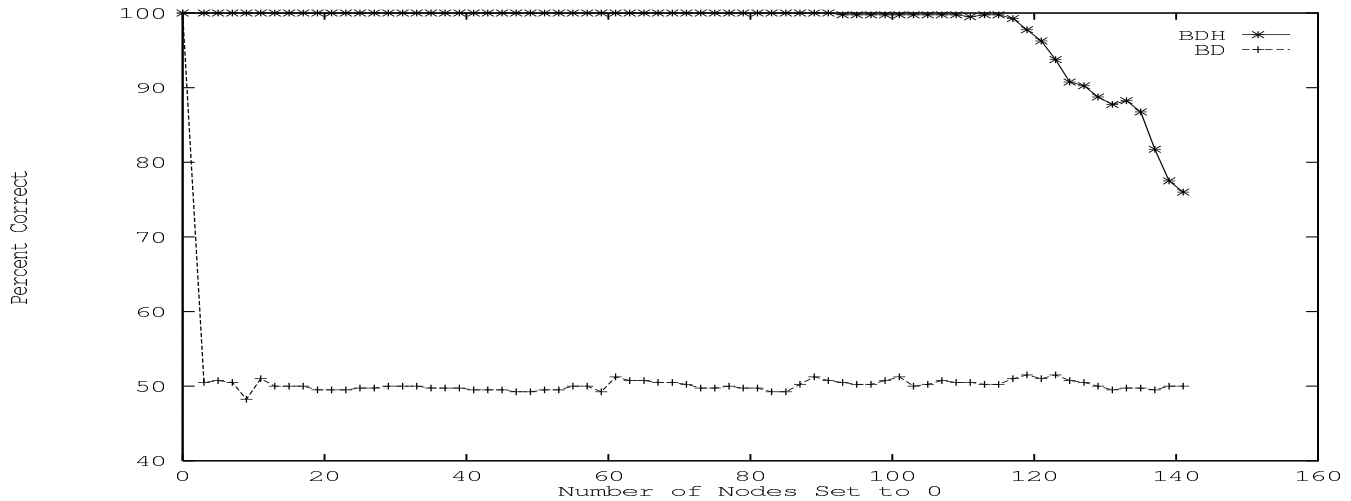


Figure 4: Performance degradation of a 3-230-1 network whose hidden nodes associated with small values of $|v_k|$ are removed from the network. “+” is the experiment in which those values of v_k are successively forced to zero. “*” is the experiment in which those nodes are removed from the network and the network is retrained.

The related experiment, also shown in Fig. 4, is to identify the small $|v_k|$ positions, as above, but to delete the corresponding nodes in the hidden layer prior to determining \mathbf{V} . This improved hidden node pruning approach allows us to construct a 3-139-1 network that solves the two-spirals three-revolutions problem and a 3-114-1 network that only makes one error.

Genetic Algorithm Searching

A genetic algorithm [8] is an indirect technique for searching for solutions to problems for which there may be no easy direct technique. A genetic algorithm simply needs the ability to evaluate the fitness (e.g., suitability, or payoff, or cost) of any proposed solution. A genetic algorithm runs in a manner inspired by natural, biological evolution, working with a population of proposed solutions. Each of the solution individuals in the population has a fitness that is used to rank them. The fitter members are chosen as parents that can have offspring combining, with luck, the good qualities of both parents. Population control keeps the size of the population from exploding by ridding it of its least fit members. Eventually, with time and luck, the process will evolve a super individual which meets the search criteria.

Fitness

The problem at hand is to locate a small neural network capable of learning a particular problem. The simple neural network that we have already constructed has a hidden layer consisting of the nodes $\{y_1, y_2, \dots, y_N\}$. What we seek is a small subset $S = \{s_1, s_2, \dots, s_k\}$ of the integer sequence $\{1, 2, \dots, N\}$, such that the neural net whose hidden layer nodes are $\{y_{s_1}, y_{s_2}, \dots, y_{s_k}\}$ solves the given problem. The fitness of S depends upon the size, k , and upon, C , the number of correct training exemplars the network built using S achieves. A suitable fitness function may take the form

$$fitness(S) = C - \beta k$$

where β is a suitable positive constants. We have tried $1 \leq \beta \leq 3$.

Creating Offspring

We can represent the sets S as sequences of zeros and ones of length N , where a one in position j indicates that j is a member of the subset. The size k of such a set can be measured as $k = \sum_1^N p_j$.

Having chosen two high-fitness individuals to serve as parents, say

$$P = \{p_1, p_2, \dots, p_N\}$$

$$Q = \{q_1, q_2, \dots, q_N\}$$

select a subscript position i , with $1 \leq i < N$, as a crossover point. The two children of P and Q are

$$\{p_1, p_2, \dots, p_i, q_{i+1}, \dots, q_N\}$$

$$\{q_1, q_2, \dots, q_i, p_{i+1}, \dots, p_N\}$$

This is known as one-point crossover [8]. The offspring may be mutated by randomly modifying some of their entries. Generally a very small amount of this mutation is needed.

Selecting Parents

In our implementation, the parents are chosen by means of a tournament model. Select T' individuals from the present population and choose the fittest of them to serve as parent P ; then select T'' (which may be the same as or different from T') other individuals and choose the most fit to serve as parent Q . We use either $T' = T''$ or $T' = T'' + 1$, so we may describe these cases in terms of $T = T' + T''$. We have used $T = 3, 4, 5$, and 6 . The larger values of T tend to force the algorithm to be “elitist” and choose most parents from among the very best in the current population. This policy causes the average population fitness to increase rapidly at the expense of the rapid loss of the features that may initially be present in some of the lesser fit individuals. Smaller values of T require longer search times (more children creations), but the quality of the eventual answers may be superior.

Population Control

The initial population is a collection of random subsets of $\{1, 2, \dots, N\}$. The population is maintained at its initial size (in the range 100–200) and kept in order sorted by fitness. As new individuals are created, they are sorted into the population with the least fit being eliminated.

A small population will rapidly evolve its most fit member and stagnate (with no mutation, the population will rapidly converge to a state in which all individuals are identical). A larger population will evolve slower, generally to a better solution.

Figs. 5–7 show the results of a genetic algorithm search for small networks to solve, or nearly solve, the two-spiral problems. The data points on the two spirals for these experiments were scaled to fit within the square $[-3.0, 3.0]^2$. Prior to the genetic search, the pseudoinverse technique was iterated ten times, each time retraining the 10% of the exemplars with the greatest absolute error. Each subset S (genetic algorithm individual) was used to extract submatrices \mathbf{A}_S ,

Population size	β	Goal # nodes	Final # nodes	Smallest # errors
100	2.2	15	15	0
100	2.2	10	12	0
100	2.3	8	6	8
200	2.3	8	6	8

Figure 5: The experiments done on 1-revolution spirals. The number of starting hidden nodes was 50.

Population size	β	Goal # nodes	Final # nodes	Smallest # errors
100	2.2	40	40	2
100	2.2	30	40	0
100	2.3	30	36	11
200	2.3	30	37	4

Figure 6: The experiments done on 2-revolution spirals. The number of starting hidden nodes was 150.

the selected columns of \mathbf{A} , and \mathbf{B}_S , the selected rows and columns of \mathbf{B} . The fitness of S is then the fitness of the network using the weights vector $\mathbf{V} := \mathbf{A}_S \mathbf{B}_S^{-1}$.

Genetic Algorithm Variation

The description of a genetic algorithm that we gave above is just one of many possibilities. Here we present one alternative that we experimented with.

An alternative method to search for small working networks is to determine at the outset what size, k , the hidden layer should be, and constrain every subset in the population to have that size. The fitness function in this case depends only on the network’s performance. (This approach was taken in [7] to discover a 300-feature classifier for hand printed digits given a working 1500-feature classifier.)

Creation of two children from two parents needs to be handled more delicately. Our solution was to replace one-point crossover with uniform crossover, which randomly assigns each parent’s entry to one or the other of the children. It is straightforward to modify this process to enforce the number of ones each child gets.

A mutation operation that preserves the number of ones in an individual can be achieved by interchanging two randomly chosen entries.

The results of two experiments are shown in Figs. 8 and 9. Without any iterations, we were able to solve the two-spirals problem in the square $[-4.0, 4.0]^2$, with

Population size	β	Goal # nodes	Final # nodes	Smallest # errors
100	2.5	80	77	30
100	2.5	70	60	81
200	2.0	70	75	20
200	2.5	70	49	86
300	2.0	70	71	13
300	2.2	70	72	19
300	2.0	60	53	98
400	2.0	70	82	26
300	2.5	70	55	103
400	2.5	60	52	98
500	2.5	70	38	94
500	2.2	70	81	24
600	2.2	70	79	29
400	2.2	70	75	14
400	2.2	70	74	16
600	2.3	70	83	22
600	2.0	95	100	0

Figure 7: The experiments done on 3-revolution spirals. The number of starting hidden nodes was 250.

two revolutions using 100 hidden nodes and the 300 quasirandom weights. The genetic algorithm searched for a network with exactly 40 hidden nodes to solve this problem. Both experiments shown used a population of 100 with the subset size fixed at 40. The first experiment used no mutation; after the creation of 4,000 individuals, the population appears to have converged to 100 copies of a network that does not solve the problem. The second experiment used a single mutation (interchanging one pair of entries chosen at random from every newly created individual—an operation that might not modify that individual); it discovered a solution to the problem in less than 10,000 individual creations, and it never converged.

Conclusions

We have shown that the quasirandom numbers are excellent replacements for pseudorandom numbers for neural network initializations. Networks can be rapidly created using those numbers for the first layer and a linear technique for the second layer to solve hard problems without being excessively large.

Several techniques—the iterated pseudoinverse, weight removal with retraining, and genetic algorithms—can be applied singly or in combination to

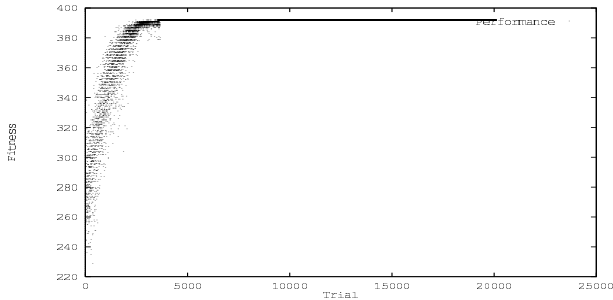


Figure 8: An experiment with no mutation converges early, failing to solve the two-spirals, two-revolutions problem with 40 hidden units.

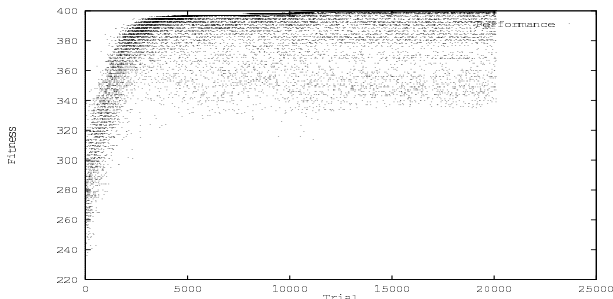


Figure 9: An experiment with a little mutation solved the two-spirals, two-revolutions problem with 40 hidden units.

create smaller networks.

The Authors

Peter G. Anderson is a Professor in the Departments of Computer Science and Imaging Science at the Rochester Institute of Technology. Address correspondence to him at anderson@cs.rit.edu.

Ming Ge, Sanjay Raghavendra, and Mei-Ling Lung, are graduate students pursuing their Masters Degrees in Computer Science at RIT.

Roger S. Gaboriski does medical imaging research at Kodak Health Imaging Systems, Inc.

Acknowledgement

The authors wish to thank Alex Mirzaoff and Eastman Kodak Company for support of this project.

References

[1] Peter G. Anderson, "Advances in linear pixel shuffling," Proceedings of the 1994 International Con-

ference on Fibonacci Numbers and their applications. (to appear)

- [2] Peter G. Anderson and Roger S. Gaboriski, "The polynomial method augmented by supervised training for hand printed character recognition," *Proceedings of the International Conference on Neural Networks and Genetic Algorithms 1993. Artificial Neural Networks and Genetic Algorithms, Proceedings of the International Conference in Innsbruck, Austria, 1993*, R. F. Albrecht, C. R. Reves, and N. C. Steele, Eds., Springer-Verlag.
- [3] Peter G. Anderson, "Linear pixel shuffling for image processing: an introduction," *Electron. Imag.* 2: 147-154(1993).
- [4] J. W. S. Cassels, *An Introduction to Diophantine Approximation*, Cambridge Tracts in Mathematics and Mathematical Physics, No. 45, Cambridge University Press, Cambridge (1957).
- [5] Scott E. Fahlman and Christian Lebiere, "The cascade-correlation learning architecture," Carnegie-Mellon University Technical Report no. CMU-CS-90-100, February 14, 1990.
- [6] Laurene Fausett, *Neural Network Architectures*, Prentice Hall, Englewood Cliffs, NJ, 1993.
- [7] Roger S. Gaboriski, Peter G. Anderson, Christopher Asbury, and David Tilly, "Genetic algorithm selection of features for hand-printed character identification," *Artificial Neural Networks and Genetic Algorithms, Proceedings of the International Conference in Innsbruck, Austria, 1993*, R. F. Albrecht, C. R. Reves, and N. C. Steele, Eds., Springer-Verlag.
- [8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York, 1989.
- [9] U. Grenendar and W. Freiberger, *A Short Course in Computational Probability and Statistics*, Applied Mathematical Sciences 6, Springer Verlag, New York, NY (1971).
- [10] Kenneth E. Iverson, *Introduction to J*, Iverson Software, Inc., 33 Major St., Toronto, Ontario, Canada, 1992.
- [11] H. Niederreiter, "Quasi Monte Carlo methods and pseudorandom numbers," *Bull. AMS*, 84(6): 957-1041(1978).
- [12] Uma Srinivasan, "Polynomial discriminant method for handwritten digit recognition," *SUNY Buffalo Technical Report*, December 14, 1989.