Feedforward Neural Network Initialization: an Evolutionary Approach

Leandro Nunes de Castro Eduardo Masato Iyoda Fernando José Von Zuben Ricardo Gudwin
{lnunes, emi, vonzuben, gudwin}@dca.fee.unicamp.br
State University of Campinas – UNICAMP – DCA – FEEC
Campinas – SP – Brazil

Abstract

The initial set of weights to be used in supervised learning for multilayer neural networks has a strong influence in the learning speed and in the quality of the solution obtained after convergence. An inadequate initial choice of the weight values may cause the training process to get stuck in a poor local minimum or to face abnormal numerical problems. Nowadays, there are several proposed techniques that try to avoid both local minima and numerical instability, only by means of a proper definition of the initial set of weights. The focus of this paper is in the application of genetic algorithms (GA) as a tool to analyze the space of weights, in order to achieve good initial conditions for supervised learning. GA's almost-global sampling compliments connectionist local search techniques well, and allows us to find some very important characteristics in the initial set of weights for multilayer networks. The results presented are compared, for a set of benchmarks, with that produced by other approaches found in the literature.

1. Introduction

The efficacy and efficiency of supervised learning in multilayer neural networks strongly depend on the network topology, the neurons' activation function, the learning rule, and the initial values of the weights.

Optimal instances for these items are usually unknown *a priori* because they depend mainly on the particular training set to be considered and on the nature of the solution [17].

Here we assume that the network topology, the neurons' activation function and the learning rule have already been determined in a proper manner, though not necessarily the optimal one. Under these conditions, a successful training process turns to depend solely on a good instantiation of the set of weights, that is, one that guides the training process to a high-quality solution, out of poor local minima and abnormal numerical problems.

The importance of a good choice for the initial set of weights is stressed by Kolen and Pollak [7]. They showed that it is not feasible to perform a purely global search for obtaining the optimal set of weights. So, for practical purposes, the learning rule should be based on optimization techniques that employ local search to find the optimal solution [15]. But local search implies that the solution has a strong relation to the initial condition, because each initial condition belongs to the basis of attraction of a particular local minimum, which will attract this solution [5].

Here we consider potential hybrids of genetic algorithms (GA) and connectionist algorithms from the perspective of the state spaces they search through and their respective methods. In brief, GA proceeds by almost *globally sampling* over the space of alternative solutions, while gradient techniques proceed by *locally searching* the immediate neighborhood of a current solution. This suggests that using GA to provide good "seeds" from which a gradient method continues to search will be effective. Notice that, as the search using GA is not purely global, we are not allowed to ask for optimal seeds, but only good ones.

2. Non-evolutionary initialization techniques

Fahlman [4] performed studies about random weight initialization techniques for multilayer neural networks. He proposed the use of a uniform distribution over the interval [−1.0, 1.0], but experimental results showed that the best initialization interval to the problems he dealt with varied in ranges between [−0.5, 0.5] and [−4.0, 4.0].

Some researchers tried to determine the best initialization interval using other neural network parameters.

Kim and Ra [6] calculated a lower bound for the initial length of the weight vector of a neuron to be $\sqrt{\alpha / d_{in}}$, where α is the learning rate and d_{in} is the neuron *fan-in*.

Boers and Kuiper [2] initialize the weights using a uniform distribution over the interval

$\left[-3/\sqrt{d_{in}},\ 3/\sqrt{d_{in}}\right]$, without any mathematical justification.

Nguyen and Widrow [13] proposed a simple modification of the random initialization process. The weights connecting the output units to the hidden units are initialized with small random values over the interval [–0.5, 0.5]. The initial weights at the first layer are designed to improve the learning capabilities of the hidden units. Using a scale factor, $\beta = 0.7(q)^{1/p}$, where q is the number of hidden units and p is the number of inputs, the weights are randomly initialized and then scaled by $\mathbf{v} = \beta \frac{\mathbf{v}}{\|\mathbf{v}\|}$, where $\mathbf{v}$ is the first layer weight vector.

3. The evolutionary approach

The combination of GA and a training algorithm for supervised learning in feedforward network arises from the observation that the local search performed by the training algorithm (conjugate gradient or other gradient descent procedure) is well complemented by the almost-global search sampling performed by the GA. Gradient descent procedures always extract some characteristics of their local neighborhood to determine the search direction to go through. Sampling techniques like GA, on the other hand, are effective because they ensure broad coverage over the entire domain; see Figure 1. The GA works by collecting information from the search space through generations, and then using this information to guide subsequent sampling towards promising regions [1]. In this paper we use this GA capability to find the best way of initializing the weights in order to achieve the global optimum or at least a good local optimum of the cost function with little computational effort, i.e., in a few iterations.

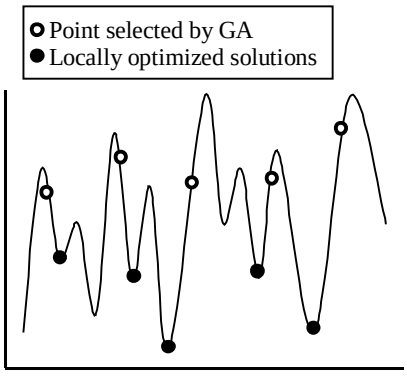


Figure 1: Sampling and searching.

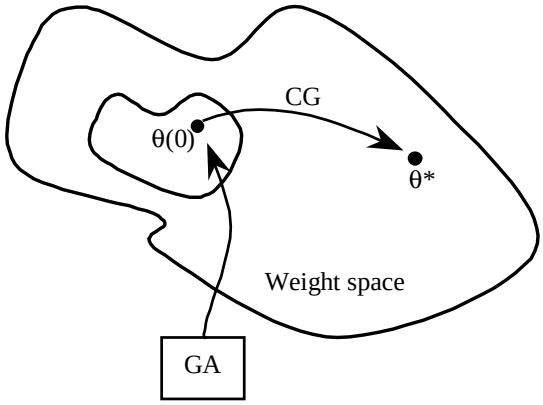


Figure 2: Selecting θ(0) and searching the optimum.

When placed in the context of connectionist networks, the GA is used to create "seeds": starting points from which a local search technique proceeds. This corresponds to using the GA to prescribe the initial weight vector θ(0). A schematic view of this hybrid construction is shown in Figure 2. The GA selects an initial weight vector for each individual in the population, and each one is allowed to learn with CG (conjugate gradient), until convergence. A local minimum is detected when the norm of the gradient vector is smaller than a previously specified threshold μ. The convergence is achieved when the sum-squared error (SSE) is smaller than another specified threshold ε.

Thus the GA will sample those regions of weight space from which it is reliably possible to reach good solutions via gradient strategies.

What we want to study is the relation between the starting point in weight space and the speed of convergence. The GA is seeking a θ(0) that guarantees a fast way of reaching the solution ultimately found by a gradient technique.

3.1 The description of the GA

In this section we describe the main properties of the GA implemented. The basic steps of a GA can be listed as follows [10]:

1. start with a randomly generated population of n m-bit chromosomes (candidate solutions to a problem);
2. calculate the fitness $f(x)$ of each chromosome x in the population;
3. repeat the following steps until n offspring have been created:

- select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness.
- with probability p_c (crossover probability), cross over the pair at a randomly chosen point to form two offspring. If no crossover takes place, form two offspring that are exact copies of the parents.

- mutate the two offspring at each locus with probability p_m (mutation probability) and place the resulting chromosomes in the new population.

4. replace the current population with the new one.
5. go to step 2.

Each iteration of this process is called a *generation*. The number of epochs for convergence is considered to be the fitness of an individual.

In the floating point (FP) implementation each chromosome vector was coded as a vector of floating numbers. Each element was forced to be within the desired range, and the operators were carefully designed to preserve this requirement. This FP implementation does not need a special kind of crossover, but makes it necessary the definition of a new mutation operator.

We used a *non-uniform mutation* [9], that is a special dynamic mutation operator aimed at both improving single element tuning and reducing the disadvantage of random mutation in the FP implementation.

The new operator is defined as follows: if $s_v^t = \langle v_1,...,v_m \rangle$ is a chromosome (t is the generation number) and the element v_k was selected for mutation, the result is a vector $s_v^{t+1} = \langle v_1,...,v_k^{'},...,v_m \rangle$, where

$$v_k^{'} = \begin{cases} v_k + \Delta(t, UB - v_k) & \text{if a random digit is 0,} \\ v_k - \Delta(t, v_k - LB) & \text{if a random digit is 1,} \end{cases} \quad (1)$$

and LB and UB are lower and upper domain bounds of the variable v_k: –1 and 1, respectively. The function $\Delta(t,y)$ returns a value in the range $[0,y]$ such that the probability of $\Delta(t,y)$ being close to zero increases as t increases. This property causes this operator to search the space uniformly initially (when t is small), and very locally at later stages, thus increasing the probability of generating the new number closer to its successor when t is large.

We used the following function:

$$\Delta(t, y) = y\left(1 - r^{(1-t/T)^b}\right) \quad (2)$$

where r is a random number from [0,1], T is the maximal generation number, and b is a system parameter determining the degree of dependency on iteration number (we set $b = 5$).

4. Alternative algorithms and benchmarks

We make the analysis of the weights determined by the GA and compare its results with five other methods applied to five benchmark problems. The methods compared were UNIFORM, BOERS [2], WIDROW [13], KIM [6], OLS [8], where these methods, with the exception of OLS, were described in Section 2. The method UNIFORM is the most commonly used uniform distribution over the interval [-1.0, 1.0].

To specify the benchmark problems used, let N be the number of samples, m the chromosome size (number of parameters to be adjusted, i.e., the dimension of the weight vector θ), SSE the desired sum squared error (stopping criterion) and *net* the net architecture represented by $[n_i\text{-}n_h\text{-}n_o]$. Where n_i is the number of inputs, n_h is the number of hidden units and n_o is the number of outputs of the network.

The benchmarks used for comparison were:

- parity 2 (XOR) problem: N = 4, m = 9, *net*: [2-2-1], SSE = 0.1;
- parity 3 problem: N = 8, m = 16, *net*: [3-3-1], SSE = 0.1;
- parity 4 problem: N = 16, m = 25, *net*: [4-4-1], SSE = 0.1;
- sin(x): N = 41, m = 16, *net*: [1-5-1], SSE = 0.1;
- ENC/DEC: the family of encoder/decoder problem is very popular and is described in [4]. N = 10, m = 157, *net*: [10-7-10].

The training algorithm used was the Moller scaled conjugate gradient [11], with the exact calculation of the second order information [14]. The net has hyperbolic tangent as the activation function for the hidden units and linear output units. The desired SSE was 0.1 in every case. In all GA's evolved, we performed 100 generations with a population of 50 individuals, p_c = 50% and p_m = 10%.

5. Simulation results

Table 1 presents the comparison among the non-evolutionary methods for the five benchmarks tested. For each problem and initialization technique, 10 runs were performed and the results to be presented are the minimum, maximum, mean and standard deviation of the number of epochs necessary for convergence.

Table 2 shows the result for the GA. In this table we present the number of epochs necessary for convergence of the best and worst individual in the population. The mean number of epochs and the standard deviation (of the number of epochs) of the population are also presented.

We are going to compare the weights (parameters) of some individuals evolved by the GA with some initial weight vectors determined by the best initialization technique for each benchmark. Tables 3 and 4 show that the mean value of the best initial weight vectors are approximately zero and Figure 4(a) and (b) show that there is a balance between the amount of negative and positive values in the weight vector.

To make an analysis of the neurons' activation, consider that $\mathbf{z}$ is the hidden activation vector, $\mathbf{y}$ the output activation vector, and $\bar{\mathbf{y}}$ its mean value. The columns of $\mathbf{z}$ and $\mathbf{y}$ represent the unit number (hidden or output) and its rows the sample index. A hidden unit is said to be saturated if its activation is not contained in the

[-*a*, *a*] interval, where *a* is a pre-specified threshold. Table 5(a) shows the result of a forward pass of the best and worst individuals evolved for the parity 2 problem. Due to the size of the network and amount of information, Table 5(b) shows only part of the results for the ENC/DEC problem.

6. Conclusions and comments

Comparing Tables 1 and 2 we can see that in the majority of the cases the GA is able of determining an individual that guarantees faster convergence (the best and the average of the population evolved) than the methods proposed in the literature. If the GA was allowed to explore a broader range of values, with weights distributed over the interval [-4, 4] for example, it would be certainly possible to determine the optimum values for the weights without the need of a gradient descent technique but with a higher computational cost, because GA do not make direct use of first and second-order information from the error surface, as done by gradient and conjugate gradient methods.

The zero mean of the best initial weight sets (Tables 3 and 4) allows us to conclude that initializing the weights predominantly in the approximately linear part of the activation function makes the training faster and less subject to numerical instability. The equilibrium in the amount of negative and positive weight values (Figure 4) leads to the conclusion that the weights have to be well distributed around the origin in the weight space in order to guarantee a broad coverage of the search space.

Another important aspect is that the mean value of the outputs of the best individuals are also close to zero, in contrast to the respective values of the worst ones. The low value of the mean together with the equilibrium in the amount of positive and negative signs gives the idea of symmetry in the weight vector, in the sense that positive values are very close in amount and precision to the negative ones for the best individuals (see Table 5 and Figure 3). In some cases, like in the approximation of the sin(*x*) function, the worst individual evolved is about ten times slower than the best one (see Table 2). If we make a forward run with the best and worst weight vectors, it is possible to realize that the best individual causes the saturation of some units while the worst does not. In these situations, we notice that some saturation caused by the GA in the initialization weight vector leads to very fast convergence, but the non-saturated vectors still gives rise to reliable initialization intervals. As a consequence, previously determined rules for weight initialization can present a poor performance or even spectacularly fail in some cases, because contradictory properties can arise in the best initial weight sets associated with different problems. So, the robustness provided by GA methods for weight initialization turns to be the main factor supporting their application in replacement of more specific methods.

References

[1] **Belew, R., K., McInerney, J., Schraudolph, N., N.**, "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning", *CSE TR #CS90-174*, 1990.

[2] **Boers, E.G.W. & Kuiper, H.** "Biological Metaphors and the Design of Modular Artificial Neural Networks", Master Thesis, Leiden University, Netherlands, 1992.

[3] **Chen, S., Chung, E.S. & Alkadhimi, K.** "Regularised Orthogonal Least Squares Algorithm for Constructing Radial Basis Function Networks", *Int. J. Control*, v. 64, n° 5, pp.829-837, 1996.

[4] **Fahlman, S.E.** "An Empirical Study of Learning Speed in Back-Propagation Networks", *Tech. Rep.*, CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, September 1988.

[5] **Hertz, J., Krogh, A. & Palmer, R.G.** Introduction to the Theory of Neural Computation. Addison-Wesley Publishing Company, 1991.

[6] **Kim, Y.K. & Ra, J.B.** "Weight Value Initialization for Improving Training Speed in the Backpropagation Network", *Proc. of the IEEE International Joint Conf. on Neural Networks*, vol. 3, pp. 2396-2401, 1991.

[7] **Kolen, J.F. & Pollack, J.B.** "Back Propagation is Sensitive to Initial Conditions", *Technical Report TR 90-JK-BPSIC*, 1990.

[8] **Lehtokangas, M., Saarinen, J., Kaski, K. & Huuhtanen, P.** "Initializing Weights of a Multilayer Perceptron by Using the Orthogonal Least Squares Algorithm", *Neural Computation*, v.7, pp. 982-999, 1995.

[9] **Michalewicz, Z.**, "Genetic Algorithms + Data Structures = Evolution Programs", 3rd. ed., Springer Verlag, 1996.

[10] **Mitchell, M.**, "An Introduction to Genetic Algorithms", MIT Press, 1996.

[11] **Moller, M.F.** "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", *Neural Networks*, vol. 6, pp. 525-533, 1993

[12] Neural Networks Benchmarks: ftp://ftp.ics.uci.edu/pub/machine-leraning-databases.

[13] **Nguyen, D. & Widrow, B.** "Improving the Learning Speed of Two-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights", in *Proc. Int. Joint Conf. N. Net. (IJCNN)*, Ann Arbor, MI, vol. 3, pp. 21-26, 1990.

[14] **Pearlmutter, B.A.** "Fast Exact Multiplication by the Hessian", *Neural Computation*, vol. 6, pp. 147-160, 1994.

[15] **Shepherd, A.J.** "Second-Order Methods for Neural Networks – Fast and Reliable Methods for Multi-Layer Perceptrons", Springer Verlag, 1997.

[16] **Stäger, F. & Agarwal, M.** "Three Methods to Speed up the Training of Feedforward and Feedback Perceptrons", *Neural Networks*, vol. 10, n° 8, pp. 1435-1443, 1997.

[17] **Thimm, G. & Fiesler, E.** "High-Order and Multilayer Perceptron Initialization", *IEEE Trans. on Neural Networks*, vol. 8, n° 2, pp. 349-359, 1997.

Appendix

Table 1: Simulation results for the benchmarks and methods tested. *Max*, *min* and *mean* are the maximum, minimum and mean number of epochs, respectively. δ is the standard deviation of the number of epochs.

Problem	Method	*Max*	*Min*	*Mean*	δ
parity 2 (XOR)	UNIFORM	95	7	33.60	32.86
	BOERS	129	6	42.90	48.32
	WIDROW	93	8	20.50	25.92
	KIM	49	7	19.80	12.49
	OLS	166	6	46.70	46.50
parity 3	UNIFORM	240	11	36.10	71.89
	BOERS	62	10	25.10	16.01
	WIDROW	25	10	19.60	5.17
	KIM	46	10	18.90	10.41
	OLS	148	29	65.70	36.16
parity 4	UNIFORM	856	42	442.80	244.55
	BOERS	465	42	202.60	153.31
	WIDROW	320	36	166.30	104.60
	KIM	379	41	121.10	103.86
	OLS	560	200	394.70	108.01
sin(*x*)	UNIFORM	123	41	63.90	26.70
	BOERS	65	24	38.90	13.94
	WIDROW	107	29	48.80	23.64
	KIM	136	38	108.50	29.50
	OLS	331	45	181.80	90.82
ENC/DEC	UNIFORM	91	11	30.10	30.72
	BOERS	40	12	22.21	7.94
	WIDROW	68	7	24.50	22.53
	KIM	298	10	38.17	69.07
	OLS	276	7	97.78	102.03

Table 2: Simulation results for the GA tested on the benchmarks. *Max*, *min* and *mean* are the maximum, minimum and mean number of epochs, respectively. δ is the standard deviation of the number of epochs of the population.

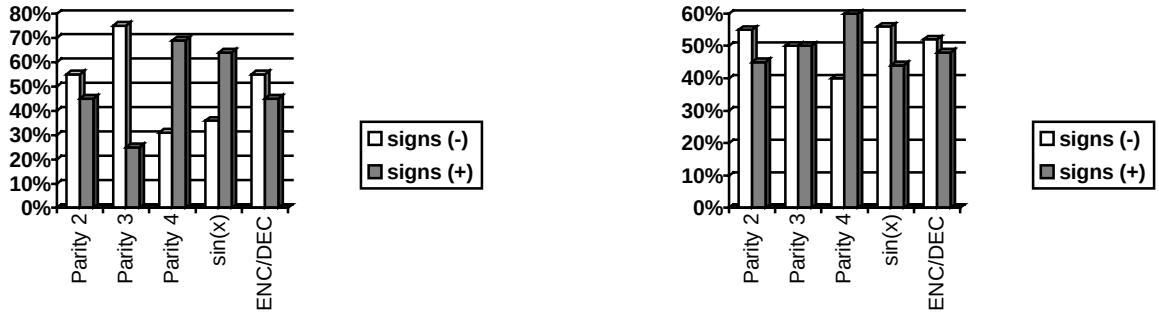
Problem	*Max*	*Min*	*Mean*	δ
parity 2 (XOR)	50	5	13.52	10.38
parity 3	96	7	15.93	12.20
parity 4	2000	15	220.10	446.94
sin(*x*)	151	14	46.48	31.74
ENC/DEC	159	7	25.80	35.39

Table 3: Analysis of the optimal weight sets evolved by the GA. *Max*, *min* and *mean* are the maximum, minimum and mean of the values of the best weight vectors, respectively. δ is the standard deviation of these values.

Problem	*Max*	*Min*	*Mean*	δ
parity 2 (XOR)	0.82	-0.99	-0.10	0.76
parity 3	0.84	-0.96	-0.30	0.56
parity 4	0.99	-0.99	0.10	0.62
sin(*x*)	0.94	-0.97	0.22	0.60
ENC/DEC	0.99	-0.99	-0.05	0.56

Table 4: Analysis of the optimal weight sets defined by some initialization techniques. *Max*, *min* and *mean* are the maximum, minimum and mean of the values of the weight vector, respectively. δ is the standard deviation of these values.

Method	Problem	*Max*	*Min*	*Mean*	δ
KIM	parity 2 (XOR)	1.01	-1.19	0.02	0.73
KIM	parity 3	0.21	-0.21	-0.01	0.15
WIDROW	parity 4	1.19	-1.22	0.07	0.71
BOERS	sin(*x*)	1.33	-1.19	-0.06	0.73
WIDROW	ENC/DEC	1.26	-1.27	-0.04	0.60



(a) (b)

Figure 3: Percentage of (-) and (+) signs in the individuals contained in the weight vector. (a) Best individual evolved, by the GA, for each problem. (b) Best weight set determined by the methods presented in Table 4.

Table 5: Forward run with the best and worst individual evolved. (a) Parity 2 (XOR) problem. (b) ENC/DEC problem.

Non-saturation interval: [-0.8, 0.8]

Best individual:
Hidden and output activation for each sample:

$$\mathbf{z} = \begin{bmatrix} -0.76 & 0.70 \\ -0.97 & -0.81 \\ 0.57 & 0.99 \\ -0.39 & 0.45 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 0.54 \\ 0.16 \\ -0.54 \\ 0.12 \end{bmatrix}$$

$\overline{\mathbf{y}} = 0.07$
Sample 2: hidden units 1 and 2 saturated
Sample 3: hidden unit 2 saturated

Worst individual:

$$\mathbf{z} = \begin{bmatrix} -0.35 & 0.97 \\ -0.81 & 0.81 \\ 0.21 & 0.51 \\ -0.51 & -0.45 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} -1.04 \\ -0.89 \\ -0.71 \\ 0.07 \end{bmatrix}$$

$\overline{\mathbf{y}} = -0.64$
Sample 1: hidden unit 2 saturated
Sample 2: hidden units 1 and 2 saturated

(a)

Non-saturation interval: [-0.9, 0.9]

Best individual:
$\overline{\mathbf{y}} = -0.03$
Sample 1: hidden unit 2 saturated

Worst individual:
$\overline{\mathbf{y}} = 0.13$
Sample 1: hidden unit 5 saturated
Sample 2: hidden unit 5 saturated
Sample 3: hidden unit 1 saturated
Sample 4: hidden unit 6 saturated
Sample 5: hidden unit 4 saturated
Sample 10: hidden unit 5 saturated

(b)