Gain Variation in Recurrent Error Propagation Networks

Steven J. Nowlan
Department of Computer Science
University Of Toronto[1]

INTRODUCTION

Neural networks have received much attention recently as plausible models for studying the computational properties of massively parallel systems. Learning algorithms have been developed (Rumelhart et al., 1986; Ackley et al., 1985) that enable these networks to learn internal representations, allowing them to represent complex non-linear mappings. Two distinct types of networks have been studied quite extensively. The first of these uses analog (continuous-valued) units with a sigmoidal I/O function (Hopfield and Tank, 1985), and an error propagation algorithm for updating the weights to minimize an error function (Plaut et al., 1986; Plaut and Hinton, 1987). Most of these studies have focused on strictly feedforward networks. The second type of network employs stochastic binary units and symmetric connections. From an initial state these networks approach a low temperature fixpoint (stable state), which represents a local minimum of a global energy function. The weights in such networks may be updated by examining the difference in statistics between the states with inputs clamped and unclamped at thermal equilibrium (Ackley et al., 1985).

Recent work has shown some interesting relationships between these two distinct models. Peterson and Anderson (1987b) have developed a continuous approximation to the Boltzmann machine algorithm, in which the stochastic binary units of the Boltzmann machine are replaced with analog units whose states are mean field approximations to the average states of corresponding stochastic binary units at equilibrium. They have shown significant speedup in convergence and improved generalization for interesting problems (Peterson and Anderson, 1987a). Hopfield (1987) has shown that for a certain class of statistical estimation problems, the statistical network and the analog network have very closely related properties and learning algorithms. Provided that four conditions are met, the error propagation update rule for the weights in an analog feedforward network is a mean field approximation to the update rule for a statistical network using the Boltzmann machine algorithm. These four conditions are: the analog network must use a symmetric divergence (Pearlmutter and Hinton, 1986) rather than the more common mean square error function; the statistical averaging in the two state network is performed over the hidden and output units, but not the input units; the two networks have a small number of outputs; and the networks have only a single layer of non-interacting hidden units[2].

This work explores another parallel between statistical and analog networks. Recurrent analog networks often show better convergence if a global gain term is introduced which may be varied over a single settling (Hopfield, 1984). The result is a procedure similar to simulated annealing (Kirkpatrick et al., 1983). An error propagation scheme is presented which allows an analog network to "learn" its own gain variation schedule, and experimental results for a constraint satisfaction task show an order of magnitude speedup in learning when this approach is used.

ERROR PROPAGATION IN RECURRENT ANALOG NETWORKS

The recurrent analog networks that are considered here use a synchronous updating procedure, and place no restrictions on the nature of the connectivity matrix. The activation level of a unit at time t is a non-linear function of the input to the unit at time t:

$$y_{j,t} = g(x_{j,t}) \tag{1}$$

One possibility for g, used in our simulations, is the logistic function, $g(x) = 1/(1+e^{-x})$. The input is a weighted sum of the states of units in the previous time step:

$$x_{j,t} = G_t \sum_i w_{ji} y_{i,t-1} \tag{2}$$

The term G_t is a global gain term which premultiplies the input for every unit. There are also two distinguished subsets of units: $\mathcal{I}$ the set of input units and $\mathcal{O}$ the set of output units. The states of units in $\mathcal{I}$ are determined by the environment.

[1] The author is currently visiting the University of Toronto, while completing a Ph.D. at Carnegie-Mellon University

[2] This means that to first order the hidden units have no effect on each other.

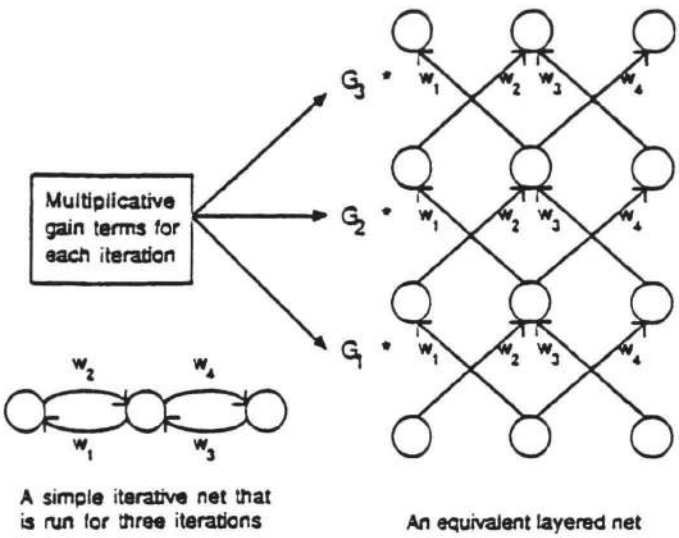

Figure 1: A recurrent network and the equivalent layered network. Corresponding weights in layers must be identical (*i.e.* w_1 has the same value in all layers), but the gain terms (G_i) vary between layers. Each layer corresponds to the state of the recurrent network at a different point in time.

The state vector for the recurrent network at time t can be treated as the vector of states of the t^{th} layer in an equivalent layered network (figure 1). The trajectory of the state vector for the recurrent network is then represented by the state vectors of successive layers in the equivalent network. Since the weights in the recurrent network are stationary (fixed during the settling of the network), the set of weights between successive layers in the layered representation must be identical.

The dynamics for these networks can be expressed by finding the continuous differential equations equivalent to the discrete difference equations (1) and (2). This produces the following set of coupled differential equations:

$$\frac{dy_j}{dt} = -y_j + g(x_j) + I_j \qquad (3)$$

where

$$x_j = G \sum_i w_{ji} y_i \qquad (4)$$

I_j is defined to be zero for all units not in $\mathcal{I}$. Equation (3) is a simple transformation of an equation that has been studied by both Amari (1972) and Hopfield (1984). Amari showed that in randomly connected networks with the dynamics of (3) the attractors were either stable or bistable.

When the attractors are fixpoints, one can use such networks to perform constraint satisfaction searches. The activities of units encode the values of the parameters of a problem, and the weights on the connections encode the constraints between the parameters. This approach has been used for classic optimization problems (Hopfield and Tank, 1985) and for parsing (Selman and Hirst, 1987). Coding the constraints into the weights by hand becomes a formidable task for large problems. This raises the possibility of devising learning algorithms which will manipulate the weights of a recurrent network to model the constraints of a specific problem through some training procedure.

There are several ways in which to pose the problem of modifying the weights. One approach would be to consider the fixpoint of the network for a specific input, and compute some error measure based on the distance of this fixpoint from a desired fixpoint. One could then use gradient descent to modify the weights to minimize this error measure (Pineda, 1987). An alternative is to consider not just the fixpoint, but the entire trajectory of the network. This approach was first suggested by Rumelhart and Hinton (1986) and is the approach taken here.

Consider a trajectory of length k for a recurrent net-

work. There is an equivalent layered representation of this trajectory which has k distinct layers (figure 1). The standard back-propagation algorithm may be applied to this layered representation, if we define an error measure E for the final states of units in $\mathcal{O}$. Using this approach we can derive partials for the weights:

$$\frac{\partial E}{\partial w_{ji}} = \sum_t \frac{\partial E}{\partial x_{j,t}} G_t y_{i,t-1} \tag{5}$$

Note that the back-propagation proceeds through the sequence of states in the trajectory, and in particular that the partial of E with respect to w_{ji} will vary along that trajectory. Since the weights are stationary, some form of time averaging is required. Equation (5) uses a uniform time averaging, although versions which favour the terms near the end of the trajectory could also be used.[3] Thus the weights are being updated based on the average derivative over the trajectory. The disadvantage of using this approach to modifying the weights in the network is that it becomes necessary to store the entire trajectory for the back propagation phase. However, the advantage is that the network can be trained not just to have certain limit behaviour, but also to have certain behaviour along the trajectory followed to the limit. One obvious example is to force the network to learn fixpoints as attractors, by penalizing bistable behaviour during the last few states of the trajectory. To allow control over the trajectory, the back propagation procedure is modified slightly to allow directly observed error terms in time steps other than the last to be added to the back propagated terms for units in $\mathcal{O}$. (This is equivalent to specifying desired states for intermediate layers in a layered network.)

The term G_t in equation (5) determines the steepness of the non-linearity in the recurrent network, and has been referred to as the system gain (Hopfield and Tank, 1985). This term may be allowed to vary during the settling of the network. For example, an increasing gain in a network with mutually inhibitory connections can implement a winner-take-all network that converges quickly. Hopfield and Tank (Hopfield and Tank, 1985) found that increasing the gain slowly as their analog network settled increased the quality of the solution found by the network. They suggested that increasing the gain in this fashion was analogous to following the effective field solution from a high temperature, resulting in a final state near the thermodynamic ground state of the system.

Rather than determining a gain schedule in advance, an error propagation scheme can be used to decide how G_t should vary over the trajectory of the network. G_t is optimized by performing gradient descent in the error measure:

$$\frac{\partial E}{\partial G_t} = \sum_j \sum_i \frac{\partial E}{\partial x_{j,t}} w_{ji} y_{i,t-1} \tag{6}$$

Once again averaging is necessary, in this case over all units in the network, since G_t is a global premultiplier.

Equations (5) and (6) can be combined to produce an algorithm that will allow networks to exhibit desired trajectories from initial states. These trajectories correspond to a constraint satisfaction search via relaxation. An algorithm of this form is described by Nowlan (Plaut et al., 1986).

PERFORMANCE ON A CONSTRAINT SATISFACTION PROBLEM

The author investigated the performance of the gain variation error propagation algorithm through simulations on some small problems (Plaut et al., 1986). For simple coding and sequencing tasks a network that learns a variable gain schedule learns to solve a problem several times faster than a similar network with fixed gain. In addition, the learned gain variation schedules outperformed several hand designed schedules on the same tasks. These tasks are all expressed in terms of I/O mappings, a prespecified output was required for each input. This makes the dynamics to be learned quite a bit easier. It is possible for the network to learn a trajectory from each input to the desired output without constructing a true attractor (a stable state with low error and a region of attraction around it) for that output.[4] A task in which it is necessary to construct robust attractors which represent the problem constraints provides a much richer domain in which to study the performance of gain variation.

The problem selected is the *n queens* problem. This is a classical constraint satisfaction problem that was studied extensively by early AI researchers (Nilsson,

[3] The G_t terms actually do weight the derivatives, so the time averaging is not truly uniform.

[4] Additional simulations showed that this was in fact the case for several of the experiments discussed.

1980; Feigenbaum and Barr, 1981). The general problem is to place n queens on an n by n grid of squares, such that there is no vertical, horizontal, or diagonal line through the grid that contains more than 1 queen. This problem is easily mapped into a network: Each cell of the grid is represented by a unit in the network, and each unit is fully connected to every other unit including itself.[5] In addition, each unit has an external input line to carry environmental input. In this special case every unit is both an input and an output unit ($\mathcal{I} = \mathcal{O}$). One nice feature of this problem is that it can be easily scaled.

Given a random initial state vector, the network is required to settle into a final state which represents a valid solution to the n queens problem. A valid solution is one in which n units are above the *on level* (0.9), and all the rest are below the *off level* (0.1). In addition, no two *on* units can lie on the same vertical, horizontal, or diagonal line. To solve this problem, the network must construct stable attractors for the valid solutions to the problem and the set of attraction basins must span the entire input space.

Since this task differs from the typical I/O mapping tasks given to error propagation algorithms, some care must be taken in deciding on an error measure. Given a random initial input state, there are in general many final states which are equally acceptable as solutions. One possibility is to measure the distance of the actual output from each of these final solutions, and take the minimum distance as the error to be propagated. This results in a form of nearest neighbour error measure.

An even more sophisticated training method, a form of shaping, may be used to produce good results and reduce the training time. The network is first presented with noisy versions of solution vectors as input. A solution is chosen at random, and then noise uniformly distributed between 0 and η (initially 0.2) is added to units that are off and subtracted from units that are on. This noisy vector is clamped to the external inputs, and the network allowed to settle for γ cycles. During the last three cycles, the mean square distance between the output vector and the solution vector used to generate the input is calculated as the error measure. The error is taken over the last three cycles to force the network to learn attractors which are fixpoints. The network is trained in this fashion until its average error (normalized by the vector lengths) is less than 20 percent over the last 50 trials. At this point, the same training regime is continued, except that the input is presented during the first cycle only, rather than being clamped. During this second phase η is gradually increased to 0.4. Once the average error over 50 trials is less than 10 percent, a final training phase is performed in which initial states are randomly generated, and the nearest neighbour error measure is used over the last three trials. The first phase of training establishs the attractors, the second phase stabilizes the attractors independant of external input, and the third phase ensures robustness of the attractors.

The simulation results are summarized in Table 1. One obvious anomaly in the table is the difference between the 5 queens and 6 queens problems. The gain variation technique shows a clear advantage for the 5 queens problem (an order of magnitude improvement), but the performance with and without gain variation is nearly identical for the 6 queens problem. The answer lies in the second column of the table. There are only 4 solutions for the 6 queens problem, and two of these solutions are simple mirror images of the other two. It is quite easy for the network to set the unit biases to favour the small set of units which appear in these 4 solutions. This "trick" makes establishing stable attractors quite easy. On the other hand for the 5 queens problem, and the larger problems, there are sufficiently many solutions so that almost every unit is active in at least one solution, so the biases alone cannot be used to give the network a head start. Under these circumstances, the dynamical behaviour of the network becomes much more important, and so a much stronger advantage is shown by the gain variation algorithm.

The set of weights learned by the network for one example of the 5 queens problem (figure 2) shows that in its solution the network has extracted the essential constraints of the task. Each unit has learned to develop a positive weight to itself, and negative weights along all horizontal, vertical and diagonal lines which the unit lies on. This means that all of the units have a natural tendency to turn on, and along each line (in any orientation) a winner-take-all network has formed, so that the stable states are those in which only one unit on a particular line is on. This is a very natural representation of the original problem constraints, which stipulated that no

[5] The network must learn which of these connections are really needed to solve the task.

		No Gain Variation				Gain Variation			
n	Solutions	Phase 1	Phase 2	Total	ϵ	Phase 1	Phase 2	Total	γ-ϵ
4	2	115	420	677	0.1	70	232	350	1.0×10^{-3}
5	10	6371	11048	19820	0.01	371	1048	1540	1.0×10^{-4}
6	4	371	1048	1420	0.1	397	698	1242	1.0×10^{-3}
7	40	20600	57480	83200	0.01	1280	4160	6080	1.0×10^{-4}
8	92	50000	—	50000	0.01	1960	12600	16200	1.0×10^{-4}

Table 1: Simulation results for various sizes of the n queens problem. ϵ is the size of the weight step, γ-ϵ is the size of the gain update. The numbers under the columns Phase 1, Phase 2, and Total are the number of weight updates performed in each training phase (see text) and in total. One update was performed after every 20 training examples. The results for the 4 queens problem are averaged over 50 runs, for the 5 and 6 queens problems the averages are over 20 runs. Only 2 runs are reported for the 7 queens, and 1 run for the 8 queens. The algorithm with no gain variation was not able to meet the 20 percent error criteria for phase 1 for the 8 queens problem and was terminated after 50000 updates.

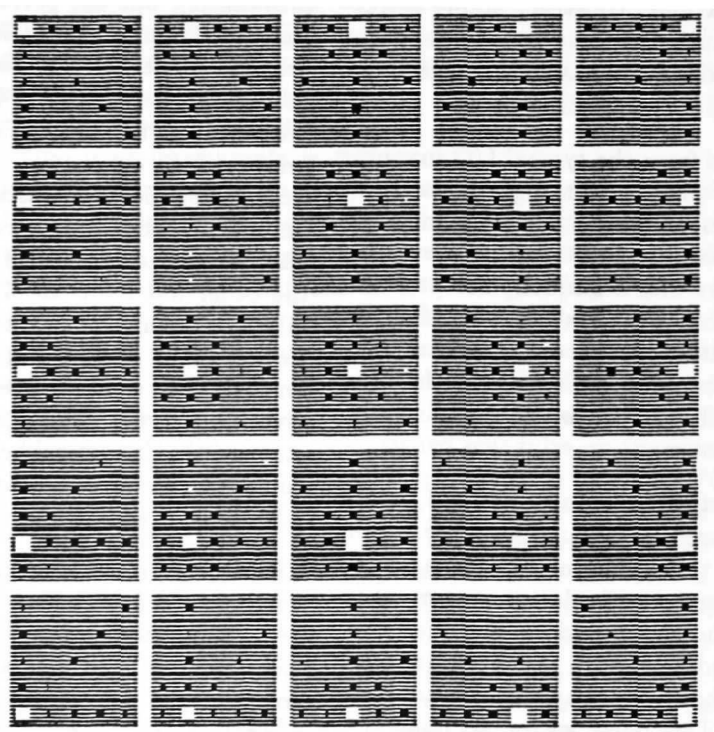

Figure 2: Weights learned for the 5 queens problem. The weight display is recursive, each large grey square represents one unit. Within each large grey square is a 5 by 5 grid of squares which represent the weight of the connection from that unit to every other unit in the network. Black squares represent negative weights, white squares represent positive weights, and the size of the square represents the magnitude of the weight. Weight decay was applied to drive all non-essential weights to zero, to simplify the weight display.

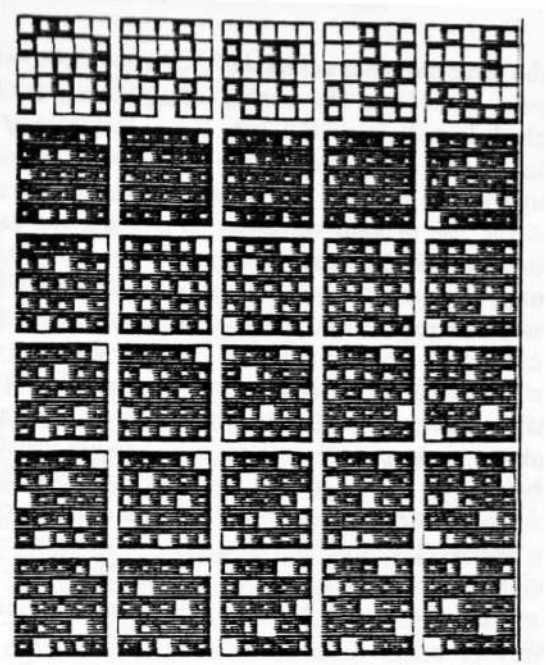

Figure 3: Display of the dynamics of the 5 queens problem for 5 different initial states. Each column represents the settling of the network for one case. The large black squares contain the activity levels of all 25 units in the network, arranged in a 5 by 5 grid. The size of the white square in each grid position is proportional to the activity of the corresponding unit. In all cases the state reached at the end of 6 cycles is a stable fixpoint.

two queens could lie on the same horizontal, vertical or diagonal line.

The dynamic behaviour of a 5 queens network can be seen in figure 3. In the third of the 5 examples we can see the network placing a queen in the third row when the initial configuration contains no dominant unit in this row. In the fifth example we can see competition between two initially dominant units in both the first and second rows, and also the creation of a dominant unit in the third row. The performance of the network is quite robust, even on highly ambiguous inputs.

Our experiments with the simple I/O mapping networks showed a tendency for the learned gain schedules to be "annealing" schedules, starting at a low gain and increasing it as the network settled (Plaut et al., 1986). This same effect is observed in the simulations for the n queens problem.

DISCUSSION

Varying the gain for a recurrent network as it settles has been suggested elsewhere (Hopfield and Tank, 1985; Pineda, 1987), as has an analogy between gain in recurrent analog networks, and temperature in statistical networks (Amari, 1972; Hopfield, 1984; Pineda, 1987). The unique aspect of this work is the use of an error propagation scheme to *simultaneously* optimize the weights and gain schedule for a recurrent network. The empirical results presented here show that for constraint satisfaction problems in which a complex attractor structure must be developed, the parallel optimization of the weights and gain schedule can produce an order of magnitude speed-up in the convergence of the optimization. What is not clear is whether this speed-up is obtained by following a shorter path to the same weight region that would be reached by optimizing the weights alone, or whether a qualitatively different region of weight space is reached by the combined optimization.

Some simple simulations tend to support the latter hypothesis. If an n queens network that has been trained with gain variation has its gain schedule modified so that the gain is a constant value (for example the mean of the gain schedule), qualitatively different behaviour is observed from the network. Additional stable states are observed, which do not correspond to solutions to the n queens problem, and which are not stable when the network is

allowed to settle using the original gain schedule. This behaviour is expected. Annealing the analog network with the gain schedule will force the network into states nearer the global error minimum, and away from local minima with relatively high error. It would appear that the use of gain variation allows the network to find a set of weights that has an attractor structure with many high error spurious attractors, in addition to the low error attractors that correspond to solutions to the task. It is apparently much easier to find a set of weights with an attractor structure of this form, rather than one with attractors which correspond only to task solution points. This leads to the hypothesis that the speed-up in convergence is obtained by allowing a much larger region of weight space to satisfy the problem, and that the combined optimization leads to a qualitatively different region of weight space than by optimization of the weights alone.

There is an additional factor which may account for some of the speed-up in learning observed with the parallel optimization of the weights and gain schedule.[6] The error surface for many problems that back propagation is applied to is characterized by ravines with steep sides in most directions, but a shallow descent in one direction. Once the weight vector is aligned with the floor of the ravine, one can move quite rapidly along the floor of the ravine by simply adjusting a global gain term. To examine this effect, several of the 5-queens simulations were repeated with a gain term that was learned by error propagation, but was constrained to be constant during a settling (as the weights were constrained). This algorithm was a factor of two to three faster than the simple back-propagation algorithm, but still five to six times slower than the parallel optimization of the weights and gain schedule, suggesting that the ability to scale all of the weights accounted for a small part of the speed-up observed in the n-queens problem. Nevertheless, the speed-up was significant enough to suggest using a similar scale factor in layered networks.

REFERENCES

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169.

Amari, S. (1972). Characteristics of random nets of analog neurons. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(5):643–657.

Feigenbaum, E. A. and Barr, A., editors (1981). *The Handbook of Artificial Intelligence*. Volume 1, Pitman, London.

Hopfield, J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Nat. Acad. Sci. USA, Bio.*, 81:3088–3092.

Hopfield, J. and Tank, D. (1985). "neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152.

Hopfield, J. J. (1987). Learning algorithms and probability distributions in feed-forward and feed-back networks. *PNAS*, 84:8429 – 8433.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

Pearlmutter, B. A. and Hinton, G. E. (1986). G-maximization: an unsupervised learning procedure for discovering regularities. In Denker, J., editor, *Neural Networks for Computing*, American Institute of Physics.

Peterson, C. and Anderson, J. (1987a). *Neural Networks and NP-complete Optimization Problems: A Performance Study on the Graph Bisection Problem*. Technical Report MCC-EI-287-87, Microelectronics and Computer Technology Corportation, Austin, TX.

Peterson, C. and Anderson, J. (1987b). *A Mean Field Theory Learning Algorithm For Neural Networks*. MCC Technical Report EI-259-87, Microelectronics and Computer Technology Corporation, Austin, TX 78759-6509.

Pineda, F. (1987). Generalization of back propagation to recurrent and higher order neural networks. In *Proceedings of IEEE Conference on Neural Information Processing Systems*, IEEE, Denver, Colorado.

Plaut, D., Nowlan, S., and Hinton, G. (1986). *Experiments on Learning by Back Propagation*. Department of Computer Science CMU-CS-86-126, Carnegie-Mellon University, Pittsburgh,PA.

Plaut, D. C. and Hinton, G. E. (1987). Learning sets of filters using back-propagation. *Computer Speech and Language*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by back-propagating errors. *Nature*, 323:533–536.

Selman, B. and Hirst, G. (1987). Parsing as an energy minimization problem. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing*, pages 155–168, Pitman, London.

[6] This was suggested by Geoff Hinton and Mike Mozer, personal communication.