

Proc. of the International Workshop on Application of Neural Networks to Telecom., 2. Stockholm, Sweden May 22–24 1995. Erlbaum Pub.

A Technique for Mapping Optimization Solutions into Hardware

Timothy X Brown

Bell Communications Research

Morristown, NJ 07960

timxb@bellcore.com

Abstract

Feedback neural architectures have addressed a myriad of optimization problems—almost exclusively in slow software simulation. Speed is promised only when implemented in high-speed recurrent hardware. Unfortunately, hardware's discrepancies from ideal conspire to produce poor solutions. This paper describes a recurrent hardware-in-the-loop learning procedure that maps idealized solutions of optimization tasks into non-ideal recurrent hardware. The technique is demonstrated on a general purpose analog VLSI chip with winner-takes-all and resource allocation problems. For all problem sizes that fit into the chip the network finds very good solutions with high probability. Theoretical analysis and empirical measurements indicate that the noise limits the hardware to 30 neurons, but this increases dramatically with reduced noise.

1 Background

Feedback neural networks have addressed optimization problems important to telecommunications such as resource allocation, traveling salesman, and scheduling. These have been adapted to a variety of applications in switching, control, and communications [1,2,3]. When designed correctly, they always satisfy the hard constraints defining valid solutions [1]. By applying annealing techniques, solution quality can be made near optimal [4]. The real advantage for these problems—demonstrated almost exclusively in software simulation—is promised when implemented in high-speed massively parallel neural hardware.

2 The Problem

The many feedback connections used in solutions to these optimization problems, plus the necessity of simultaneous settling of the many neurons required to guarantee stable valid solutions requires either time-consuming precise numerical calculations or, when speed and power consumption are critical, parallel analog hardware such as[5,6,7]. Unfortunately, when the carefully balanced optimization parameters are mapped to analog VLSI hardware; process and temperature variations, non-ideal components, and limited precision result in violated constraints, invalid solutions or significantly sub-optimal solutions. It is well known that the variety of ideal-software/hardware mismatch errors that are difficult to observe and/or quantify necessitate hardware-in-the-loop learning[6,7], i.e. ideal weights are first loaded, and then example patterns presented; performing learning whenever outputs are in error. For optimization, this is complicated by the need for a recurrent architecture learning algorithm; and recurrent hardware with adjustable synapses. Also, the complicated structures that arise, bring doubt on whether hardware-in-the-loop learning can be used with very large optimization problem sizes.

3 A Solution

3.1 The Training Procedure

The Boltzmann algorithm presents an input pattern, settling the network while the output units 'float', and then settling again with the output units 'clamped' to the correct values. Synapses use the difference in neuron activation to update their weights[8]. This local learning property is ideal for recurrent problems. An optimization problem typically has many equivalent good or optimal solutions which, while equivalent, typically have very different outputs. An arbitrary output solution chosen for learning will likely be different from the networks's solution and disrupt already learned weights. Successful training requires an algorithm that when given a potential network solution finds a nearby optimal solution.

3.2 The Hardware

Any analog hardware with suitable recurrent connections, software adjustable synapses, and a variable gain can with software implement the mean field Boltzmann algorithm. This paper was greatly facilitated by using an analog fully-interconnected 32-neuron neural network experimental prototype chip with on-chip Boltzmann learning[5]. This device performs the neural computations in parallel analog circuits while the weights are stored in 5 bit (-15 to 15) digital registers.

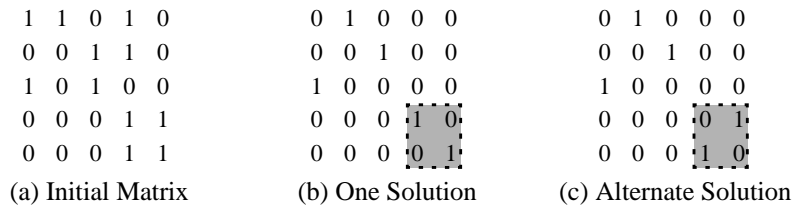


Figure 1: Example of the Binary Resource Allocation Problem. Initial matrix, (a), is reduced to no more than one 1 per row and one 1 per column (b). Equivalent alternate solutions can exist (c).

More importantly, the chips implement the Boltzmann and mean field learning algorithms on chip as part of each synapse. Currently the chips are mounted on a VME board under software control of an attached workstation.

4 Two Test Problems

The first is a stylized version of the winner-takes-all (WTA) problem that takes as input a vector of N 1's and 0's and chooses a single "winner" from any one of the 1's. The neural solution consists of N neurons with inhibitory connections between each pair of neurons. This WTA is a simple selector problem, but serves as a basis for the next problem.

$R \times C$ binary resource allocation (BRA) takes an $R \times C$ matrix of 1's and 0's and reduces 1's to 0's so that the matrix has at most one 1 in each row and one 1 in each column while maximizing the number of 1's in the final matrix. WTA is $1 \times N$ or $N \times 1$ BRA. The neural solution consists of $R \times C$ neurons with inhibitory connections between pairs of neurons in the same row or column. The problem is significant since problems such as packet arbitration can be reduced to it [1].

BRA is equivalent to the cardinality graph matching problem which has a polynomial-time algorithm solution[9]. This polynomial algorithm allows us to compute the optimal target solutions readily. Figure 1 shows an example 5×5 problem and solution. As can be seen, the optimal solution is not always unique.

5 Experiments

We tested both the WTA and BRA problems using the 32 fully-connected neuron VLSI chip. The current hardware interface requires that each input uses one neuron requiring both an input and output matrix. Thus a 3×3 problem requires at least 18 neurons. The initial weights were prescribed by the theory in [1] and after scaling to the $[-15, +15]$ range of the hardware are shown in Table 1. Due to fixed offsets in the neuron range, additional bias input units were needed to guar-

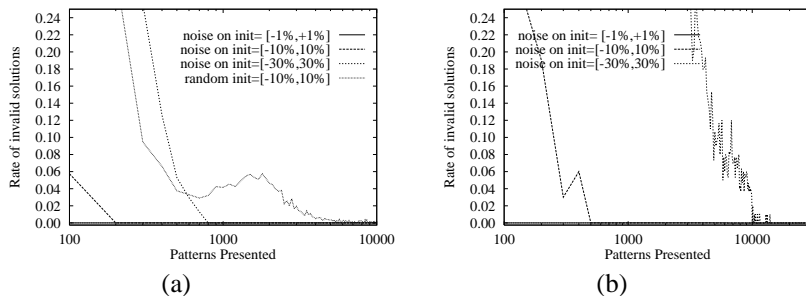


Figure 2: Simulated Learning of 3x3 (a) and 8x8 (b) BRA with Different Levels of Initial Noise in the Weights. Shows rate of invalid solutions as a function of learning cycles.

antee a large enough bias level. The number of bias units was found to be $\sim R+C$, although for larger sizes this was restricted by available neurons. To reduce the degrees of freedom we incorporated prior knowledge about the network connectivity. All extraneous synapses such as those between neurons not in the same row or column were permanently turned off and the one-to-one connections between input and output neurons were frozen at their initial values.

Random patterns were generated by setting elements in the initial matrix to 1 with probability 0.3. A learning cycle consisted of presenting a random pattern to the network, reading the output, calculating the nearest optimal pattern using the max-flow algorithm in [9] and using that as the target pattern for on-chip mean-field learning. An experimental run consisted of at least 4,000 (typically 10,000) learning cycles followed by 20,000 testing cycles. A solution during testing was judged by both its row/column constraint satisfaction, and whether it is an optimal solution. An experiment was the average of 20 runs.

5.1 Results: Software

To understand the algorithm's performance we first looked at a software simulation of the hardware. We ran a series of experiments where different noise levels were added to the prescribed initial weights. The noise was uniform zero mean with different maximum magnitude. Figure 2 shows learning curves for different perturbations on the initial ideal weights, and for completely random initial weights. Since the simulated network is ideal, with small noise, the network

Table 1: Initial Weights.

output intra-row/column	input to output	total bias
-15	15	-8

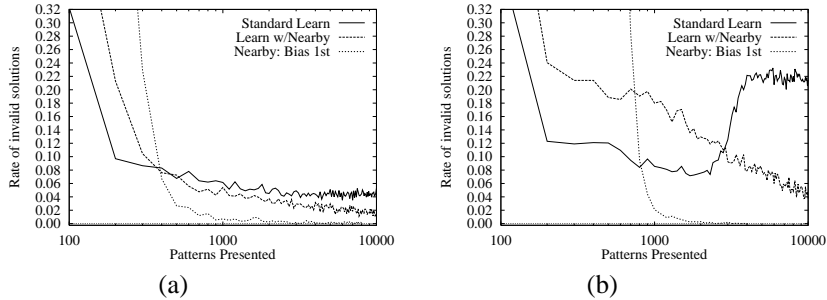


Figure 3: Hardware Learning of 3x3 BRA (a) and 10 WTA (b) with: Arbitrary Optimal Patterns; Nearby Optimal Patterns; and Nearby Optimal Pattern and Bias First.

never makes a mistake. As noise is added, the network initially produces poor and even invalid solutions, but eventually delivers an error-free final set of weights for both the 3x3 and the much larger 8x8 problem; suggesting that the hardware neural network can achieve good results even if weights are not initially ideal and it must learn them from example.

5.2 Results: Hardware

Figure 3 shows learning curves for 10-input WTA and 3x3 BRA problems. Despite the recurrent nature of the problem, the network learns to less than 5% errors within 10K iterations. Figure 3 also shows the erratic and poor performance when learning with an arbitrary optimal solution as opposed to the nearest optimal solution. Learning experiments showed that most significant weight changes were in the biases and that learning was greatly facilitated if the bias weights were first set to the correct value. Figure 3 shows the same learning when the first half of the learning cycles were used to find the bias, and the second half were used to learn all of the weights (the on-chip learning mechanism could be disabled for individual synapses under software control). In this case the network learns better solutions faster. Because of the improved performance, this learning first on bias and then on all weights was used for all of the later experiments.

To test the ability to learn problems in a variety of different dimensions, experiments were done for [2–15]x1, [2–7]x2, and [3–5]x3 size networks (all combinations that could fit onto the 32 neuron chip). A successfully trained network was defined as one that would find the optimal solution 99.9% of the time. Figure 4 shows the probability of a successful run as a function of the problem size. As can be seen problems can with high probability be learned for sizes up to 12x1, 6x2, and 4x3. The performance for the 5 larger sizes was limited due to lack of available bias neurons on the 32 neuron chip (e.g., the 15x1 and 5x3 problems

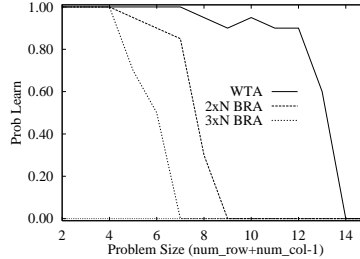


Figure 4: Probability of Learning to 99.9% Accuracy vs. Problem Size.

had only 2 bias neurons available). This was confirmed by observing in these cases that at the end of learning the bias weights were always saturated.

6 Extensibility to Larger Sizes

The previous section demonstrates in hardware the WTA/BRA for different sizes. We analyze the limits to the hardware from two perspectives—systematic weight variations and ambient system noise. For this analysis we focus on setting a bias weight, T_B , for each output neuron that works with all input combinations. With no noise, a range of bias weights will all work. As the noise grows, this range narrows; an effect that is amplified with more neurons. In this analysis we ignore learning and calculate the probabilities that any successful solution exists.

We look first at systematic weight variations with the following model. The connection between any two associated neurons (in same row or column) is nominally the maximum negative weight, $-T_{MAX}$. Each weight has a fixed additive error uniformly distributed in $[-\epsilon/2, +\epsilon/2]$. Since for large errors, we can adjust the weights to within the weight precision, we assume ϵ is less than the precision of the weights. The probability of there being a working bias is a function of the neuron on and off voltage, V_{ON} and V_{OFF} , and the size of the problem, $n = R + C - 1$. If all of a neuron's associates are off, then the input from the bias neuron should exceed the input from the associates so the neuron turns on:

$$V_{ON}T_B > N(-(n-1)V_{OFF}T_{MAX}, V_{OFF}\sqrt{n-1}\sigma_p), \quad (1)$$

where $N(\mu, \sigma)$ is a μ mean, σ standard deviation normal variable and $\sigma_p = \epsilon/\sqrt{12}$ is the uniformly distributed precision's standard deviation. Similarly if one or more associated neurons are on then the neuron should turn off:

$$V_{ON}T_B < N(-(V_{ON} + (n-2)V_{OFF})T_{MAX}, \sqrt{V_{ON}^2 + V_{OFF}^2(n-2)}\sigma_p) \quad (2)$$

Normalizing by $V_{ON}T_{MAX}$ and taking the difference between (1) and (2) we get the normalized range of allowable values for T_B :

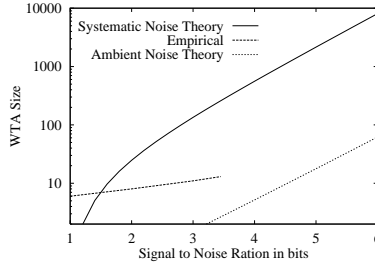


Figure 5: Precision Limitations on Addressable Problem Size.

$$R = N(1 - V_{\text{OFF}}/V_{\text{ON}}, \sqrt{1 + (V_{\text{OFF}}/V_{\text{ON}})^2(2n - 3)(\sigma_p/T_{\text{MAX}})}) \quad (3)$$

We can find a valid bias if $R > 0$. In the hardware, $V_{\text{OFF}} = 0.7\text{V}$ and $V_{\text{ON}} = 4.3\text{V}$. Conservatively, $V_{\text{OFF}}/V_{\text{ON}} = 1/4$. Using this value, Figure 5 plots the signal-to-noise ratio (SNR), $(1+T_{\text{MAX}})/\epsilon$, such that a bias can be found for all n associated neurons with 99.9% probability as a function of n . Similar analysis on errors in the neuron output voltages yields:

$$R = N(1 - V_{\text{OFF}}/V_{\text{ON}}, \sqrt{2(n - 1)\sigma_N/V_{\text{ON}}}) \quad (4)$$

where σ_N is the voltage variance. Figure 5 plots the SNR such that a bias can be found with 99.9% probability as a function of n . Comparing the two noises, clearly the network is not limited by systematic offsets—since, as the software results illustrate, these can be countered by adjusting the weights—and noise is a dominant factor. For the noisy VME system our measured SNR is about 6 bits indicating problems of up to size 60 can be tackled.

To get an idea for the largest size problems that we can solve in the hardware, we re-ran the simulations by permanently fixing T_{MAX} to smaller values, and observing the largest size WTA problem that could be learned with 99.9% accuracy. This is also plotted in Figure 5 (Note that it is not directly comparable with the other plots). The full 5 bits could not be used before the bias limits were reached. But extrapolating we see that problems of size 20–30 are possible, implying an ambient noise level closer to 5 bits. Steps are being taken to reduce the noise since according to Figure 5 every bit higher SNR doubles the feasible largest size. Using a more typical analog SNR of 8 bits, we see that problems up to size 200 are possible (e.g. 100x100 BRA with 10,000 neurons).

7 Discussion and Conclusions

We have shown in both software and analog hardware that the discrepancy between an ideal set of weights and a noisy weights can be corrected for with hardware-in-the-loop learning even in the difficult case of an optimization task with a recurrent architecture. This was demonstrated in hardware using an on-chip recurrent learning algorithm on a resource allocation task with excellent results on sizes up to 12x1, 6x2, and 4x3. Larger sizes were not always successful due to a limited number of bias units. Analysis of systematic offsets and noise on the neuron outputs suggested that systematic offsets will not limit significantly but ambient noise may limit the system to problems of size 200. Direct measurements in hardware suggested sizes of ~30, consistent with the noisy VME environment. New modified boards will both reduce noise and allow us to tackle problems with up to 64 neurons and probe these predictions. These encouraging results bring the promised speed of parallel analog hardware closer to significant optimization tasks.

Acknowledgments

I would like to thank Tony Jayakumar for help with the analog hardware and Josh Alspector for useful discussions.

References

- [1] Brown, T.X., "Neural Networks For Switching," in *Neural Networks in Telecom.*, ed. Yuhas, B., Ansari, N., Kluwer, Boston, 1994, pp. 11–36.
- [2] Kechriotis, G., Manolakos, E., "Implementing the Optimal CDMA Multiuser Detector with Hopfield Neural Networks," *Proc. of the Inter. Workshop on Appl. of Neural Networks to Telecom.*, ed. Alspector, J., et al., Erlbaum, Hillsdale, NJ, 1993, pp. 60–66.
- [3] Barhen, J., Toomarian, N., Protopopescu, V., "Optimization of the Computational Load of a Hypercube Supercomputer Onboard a Mobile Robot," *Applied Optics*, Vol. 26, No. 23, pp. 5007–5014, 1987.
- [4] Peterson, C., Anderson, J.R., "A Mean Field Learning Algorithm for Neural Networks", *Complex Systems*, Vol. 1, No. 5, pp. 995–1019, 1987.
- [5] Alspector, J., Jayakumar, A., Luna, S., "Experimental Evaluation of Learning in a Neural Microsystem," in *Advances in Neural Info. Proc. Sys. 4*. ed. Moody, J.E., et al., Morgan Kaufmann, San Mateo, CA, 1992, pp. 871–878.
- [6] Eberhardt, S.P., Duong, T., Thakoor, A.P., "Design of Parallel Hardware Neural Network Systems from Custom Analog VLSI "Building Block" Chips," *Proc. of the Inter. Joint Conf. on Neural Networks, 1989*, Vol. II, pp. 183–190.

- [7] Holler, M., Tam, S., Castro, H., Benson, R., “An Electrically Trainable Analog Neural Network (ETANN) with 1024 ‘Floating Gate’ Synapses,” *Proc. of the Inter. Joint Conf. on Neural Networks*, 1989 Vol. II, pp. 191–196.
- [8] Ackley, D.H., Hinton, G.E., Sejnowski, T.J., “A Learning Algorithm for Boltzmann Machines”, in *Cognitive Science*, 1985, pp. 147–169.
- [9] Lawler, E.L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976, p. 195