VLSI NEURAL NETWORK WITH DIGITAL WEIGHTS AND ANALOG MULTIPLIERS

Vincent F. Koosh, Rodney Goodman

California Institute of Technology
Pasadena, CA 91125
darkd@micro.caltech.edu
http://www.micro.caltech.edu

ABSTRACT

A VLSI feedforward neural network is presented that makes use of digital weights and analog multipliers. The network is trained in a chip-in-loop fashion with a host computer implementing the training algorithm. The chip uses a serial digital weight bus implemented by a long shift register to input the weights. The inputs and outputs of the network are provided directly at pins on the chip. The training algorithm used is a parallel weight perturbation technique[1]. Training results are shown for a 2 input, 1 output network trained with an AND function, and for a 2 input, 2 hidden unit, 1 output network trained with an XOR function.

1. INTRODUCTION

Training an analog neural network directly on a VLSI chip provides additional benefits over using a computer for the initial training and then downloading the weights. The analog hardware is prone to have offsets and device mismatches. By training with the chip in the loop, the neural network will also learn these offsets and adjust the weights appropriately to account for them. A VLSI neural network can be applied in many situations requiring fast, low power operation such as handwriting recognition for PDA's or pattern detection for implantable medical devices[2].

There are several issues that must be addressed to implement an analog VLSI neural network chip. First, an appropriate algorithm suitable for VLSI implementation must be found. Traditional error backpropagation approaches for neural network training require too many bits of floating point precision to implement efficiently in an analog VLSI chip. Techniques that are more suitable involve stochastic weight perturbation[1],[3],[4],[5],[6],[7], where a weight is perturbed in a random direction, its effect on the error is determined and the perturbation is kept if the error was reduced; otherwise, the old weight is restored. In this approach, the network observes the gradient rather than actually computing it.

Serial weight perturbation[3] involves perturbing each weight sequentially. This requires a number of iterations that is directly proportional to the number of weights. A significant speed-up can be obtained if all weights are perturbed randomly in parallel and then measuring the effect on the error and keeping them all if the error reduces. Both the parallel and serial methods can potentially benefit from the use of annealing the perturbation. Initially large perturbations are applied to move the weights quickly towards a minimum. Then, the perturbation sizes are occasionally decreased to achieve finer selection of the weights and a smaller error. In general, however, optimized gradient descent techniques converge more rapidly than the perturbative techniques.

Next, the issue of how to appropriately store the weights on-chip in a non-volatile manner must be addressed. If the weights are simply stored as charge on a capacitor, they will ultimately decay due to parasitic conductance paths. One method would be to use an analog memory cell [8],[9]. This would allow directly storing the analog voltage value. However, this technique requires using large voltages to obtain tunneling and/or injection through the gate oxide and is still being investigated. Another approach would be to use traditional digital storage with EEPROM's. This would then require having A/D/A (one A/D and one D/A) converters for the weights. A single A/D/A converter would only allow a serial weight perturbation scheme that would be slow. A parallel scheme, which would perturb all weights at once, would require one A/D/A per weight. This would be faster, but would require more area. One alternative would remove the A/D requirement by replacing it with a digital counter to adjust the weight values. This would then require one digital counter and one D/A per weight.

2. CIRCUITS

2.1. Synapse

A small synapse with one D/A per weight can be achieved by first making a binary weighted current source (Figure 1) and then feeding the binary weighted currents into diode connected transistors to encode them as voltages. We then feed these voltages to transistors on the synapse to convert them back to currents. Thus, we achieve many D/A's with only one binary weighted array of transistors. It is clear, that the linearity of the D/A will be poor because of matching errors between the current source array and synapses which may be located on opposite sides of the chip. This is not a concern because the network will be able to learn around these offsets.

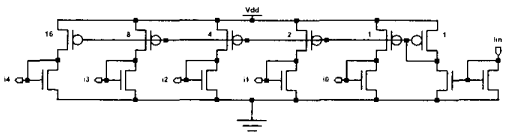

Figure 1: Current Source Circuit

The synapse[6],[2] is shown in Figure 2. The synapse performs the weighting of the inputs by multiplying the input voltages by a weight stored in a digital word denoted by b0 through

0-7803-6685-9/01/$10.00©2001 IEEE

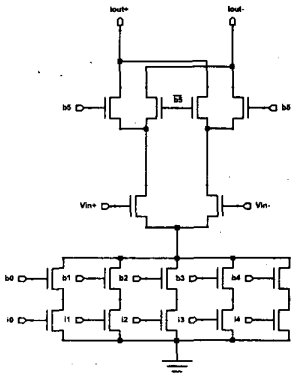

Figure 2: Synapse Circuit

b5. The sign bit, b5, changes the direction of current to achieve the appropriate sign.

In the subthreshold region of operation, the transistor equation is given by[10]:

$$I_d = I_{d0} e^{\kappa V_{gs}/U_t}$$

and the output of the synapse is given by[10],[2]:

$$\Delta I_{out} = I_{out+} - I_{out-} = \begin{cases} +I_0 W \tanh\left(\frac{\kappa(V_{in+}-V_{in-})}{2U_t}\right) \\ b5 = 1 \\ -I_0 W \tanh\left(\frac{\kappa(V_{in+}-V_{in-})}{2U_t}\right) \\ b5 = 0 \end{cases}$$

where W is the weight of the synapse encoded by the digital word and I_0 is the least significant bit (LSB) current.

Thus, in the subthreshold linear region, the output is approximately given by:

$$\Delta I_{out} \approx g_m \Delta V_{in} = \frac{\kappa I_0}{2U_t} W \Delta V_{in}$$

In the above threshold regime, the transistor equation in saturation is approximately given by:

$$I_D \approx K(V_{gs} - V_t)^2$$

The synapse output is no longer described by a simple tanh function, but is nevertheless still sigmoidal with a wider "linear" range.

In the above threshold linear region, the output is approximately given by:

$$\Delta I_{out} \approx g_m \Delta V_{in} = 2\sqrt{K I_0}\sqrt{W}\Delta V_{in}$$

It is clear that above threshold, the synapse is not doing a pure weighting of the input voltage. However, since the weights are learned on chip, they will be adjusted accordingly to the necessary value. Furthermore, it is possible that some synapses will operate below threshold while others above depending on the choice of LSB current. Again, on-chip learning will be able to set the weights to account for these different modes of operation.

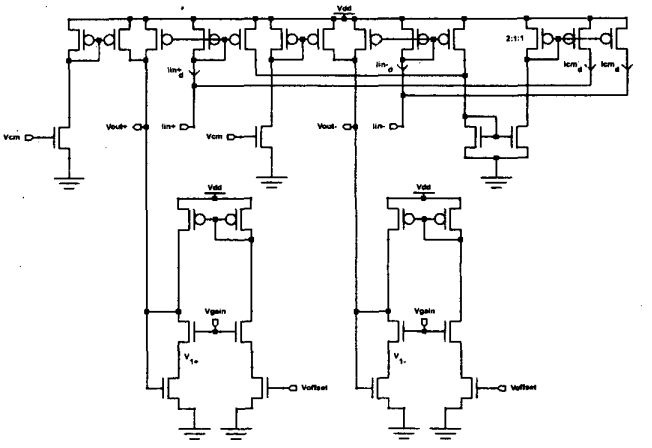

Figure 3: Neuron Circuit

2.2. Neuron

The synapse circuit outputs a differential current that will be summed in the neuron circuit shown in Figure 3. The neuron circuit performs the summation from all of the input synapses. The neuron circuit then converts the currents back into a differential voltage feeding into the next layer of synapses. Since the outputs of the synapse will all have a common mode component, it is important for the neuron to have common mode cancellation[2]. Since one side of the differential current inputs may have a larger share of the common mode current, it is important to distribute this common mode to keep both differential currents within a reasonable operating range.

$$I_{in+} = I_{in+_d} + \frac{I_{in+_d} + I_{in-_d}}{2} = I_{in+_d} + I_{cm_d}$$

$$I_{in-} = I_{in-_d} + \frac{I_{in+_d} + I_{in-_d}}{2} = I_{in-_d} + I_{cm_d}$$

$$\Delta I = I_{in+} - I_{in-} = I_{in+_d} - I_{in-_d}$$

$$I_{cm} = \frac{I_{in+} + I_{in-}}{2} = \frac{I_{in+_d} + I_{in-_d} + 2I_{cm_d}}{2} = 2I_{cm_d}$$

$$\Rightarrow I_{in+_d} = I_{in+} - \frac{I_{cm}}{2} = \frac{\Delta I}{2} + \frac{I_{cm}}{2}$$

$$\Rightarrow I_{in-_d} = I_{in-} - \frac{I_{cm}}{2} = -\frac{\Delta I}{2} + \frac{I_{cm}}{2}$$

If the ΔI is of equal size or larger than I_{cm}, the transistor with I_{in-_d} may begin to cutoff and the above equations would not exactly hold; however, the current cutoff is graceful and should not normally affect performance. With the common mode signal properly equalized, the differential currents are then mirrored into the current to voltage transformation stage. This stage effectively takes the differential input currents and uses a transistor in the triode region to provide a differential output. This stage will usually

be operating above threshold, because the V_{offset} and V_{cm} controls are used to ensure that the output voltages are approximately mid-rail. This is done by simply adding additional current to the diode connected transistor stack. Having the outputs mid-rail is important for proper biasing of the next stage of synapses. The above threshold transistor equation in the triode region is given by $I_d = 2K(V_{gs} - V_t - \frac{V_{ds}}{2})V_{ds} \approx 2K(V_{gs}-V_t)V_{ds}$ for small enough V_{ds}. If K_1denotes the prefactor with W/L of the cascode transistor and K_2 denotes the same for the transistor with gate V_{out}, the voltage output of the neuron will then be given by:

$$I_{in} = K_1(V_{gain} - V_t - V_1)^2 \approx K_2\,(2(V_{out} - V_t)V_1)$$

$$V_1 = V_{gain} - V_t - \sqrt{\frac{I_{in}}{K_1}}$$

$$I_{in} = 2K_2(V_{out} - V_t)\left(V_{gain} - V_t - \sqrt{\frac{I_{in}}{K_1}}\right)$$

$$V_{out} = \frac{I_{in}}{2K_2(V_{gain} - V_t) - 2\sqrt{\frac{K_2^2}{K_1}I_{in}}} + V_t$$

$$\text{if } \frac{W_1}{L_1} = \frac{W_2}{L_2} \text{ then } K_1 = K_2 = K,$$

$$\Rightarrow V_{out} = \frac{I_{in}}{2K(V_{gain} - V_t) - 2\sqrt{KI_{in}}} + V_t$$

$$\text{for small } I_{in},\ R \approx \frac{1}{2K(V_{gain} - V_t)}$$

Thus, it is clear that V_{gain} can be used to adjust the effective gain of the stage.

Using these two circuit building blocks it is possible to construct a multilayer feed-forward neural network. Note that the nonlinear squashing function is actually performed in the next layer of synapse circuits rather than in the neuron as in a traditional neural network. However, this is equivalent as long as the inputs to the first layer are kept within the linear range of the synapses. The biases for each neuron are simply implemented as synapses tied to fixed bias voltages. Also, depending on the type of network outputs desired, additional circuitry may be needed for the final squashing function.

3. TRAINING ALGORITHM

The neural network is trained by using the parallel perturbative weight update rule[1]. The perturbative technique requires generating random weight increments to adjust the weights during each iteration. These random perturbations are then applied to all of the weights in parallel. In batch mode, all input training patterns are applied and the error is accumulated. This error is then checked to see if it was higher or lower than the unperturbed iteration. If the error is lower, the perturbations are kept, otherwise they are discarded. This process repeats until a sufficiently low error is achieved. The following is an outline of the algorithm:

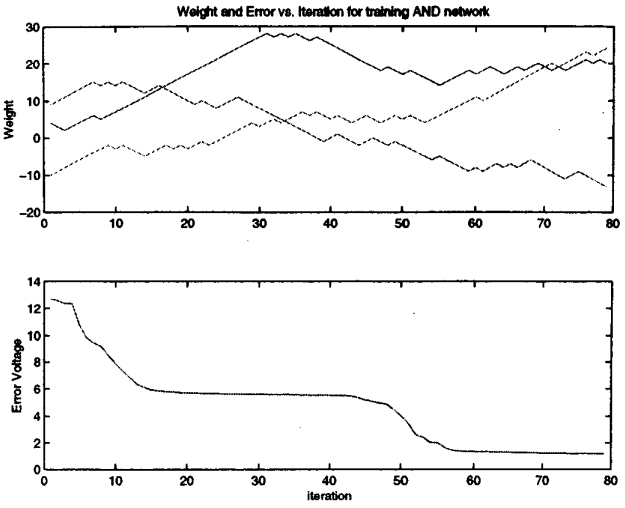

Figure 4: Training of a 2:1 network with AND function

```
Initialize Weights;
Get Error;
while(Error > Error Goal);
   Perturb Weights;
   Get New Error;
   if (New Error < Error),
      Weights = New Weights;
      Error = New Error;
   else
     Restore Old Weights;
   end
end
```

4. TEST RESULTS

A chip implementing the above circuits was fabricated in a 1.2μm CMOS process. All synapse and neuron transistors were 3.6μ/3.6μ. An LSB current of 100nA was chosen for the current source. The above neural network circuits were trained with some simple digital functions such as 2 input AND and 2 input XOR. The results of some training runs are shown in Figures 4-5. As can be seen from the figures, the network weights slowly converge to a correct solution. Since the training was done on digital functions, a differential to single ended converter was placed on the output of the final neuron. This was simply a 5 transistor transconductance amplifier. The error voltages were calculated as a total sum voltage error over all input patterns at the output of the transconductance amplifier. Since V_{dd} was 5V, the output would only easily move to within about 0.5V from V_{dd} because the transconductance amplifier had low gain. Thus, when the error gets to around 2V it means that all of the outputs are within about 0.5V from their respective rail and functionally correct. A double inverter buffer can be placed at the final output to obtain good digital signals. At the beginning of each of the training runs, the error voltage starts around or over 10V indicating that at least 2 of the input patterns give an incorrect output.

Figure 4 shows the results from a 2 input, 1 output network

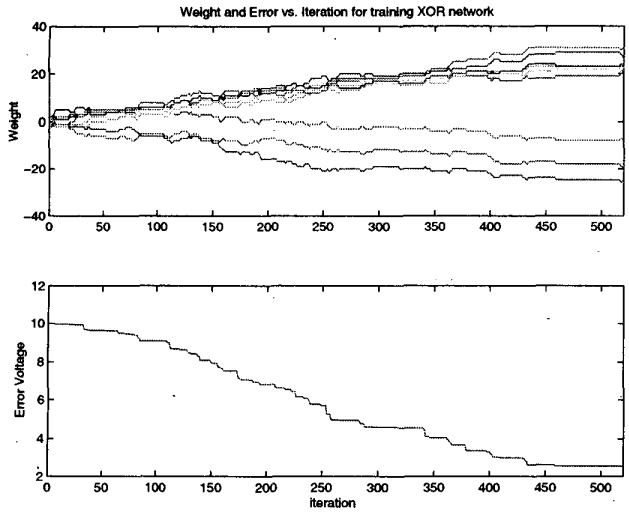

Figure 5: Training of a 2:2:1 network with XOR function starting with small random weights

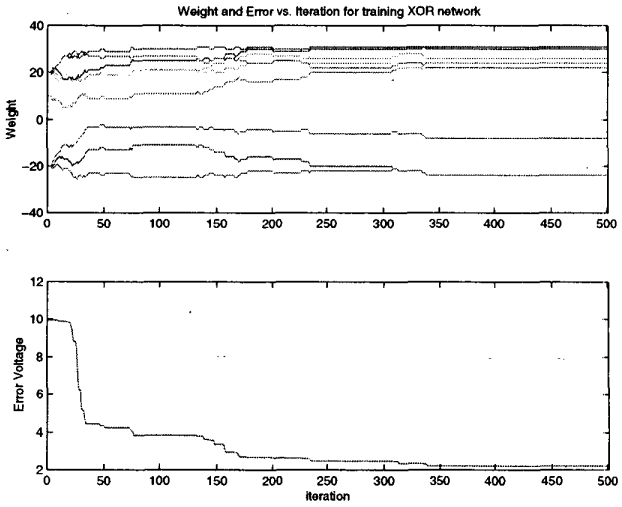

Figure 6: Training of a 2:2:1 network with XOR function starting with "ideal" weights

learning an AND function. This network has only 2 synapses and 1 bias for a total of 3 weights. The weight values can go from -31 to +31 because of the 6 bit D/A converters used on the synapses.

Figure 5 shows the results of training a 2 input, 2 hidden unit, 1 output network with the XOR function. The weights are initialized as small random numbers. The weights slowly diverge and the error monotonically decreases until the function is learned. As with gradient techniques, occasional training runs resulted in the network getting stuck in a local minimum and the error would not go all the way down.

Figure 6 shows the same network trained with XOR, but the initial weights are chosen as mathematically correct weights for ideal synapses and neurons. Although, the ideal weights should, in theory, start off with correct outputs, the offsets and mismatches of the circuit cause the outputs to be incorrect. However, since the weights start near where they should be, the error goes down rapidly to the correct solution. This is an example of how a more complicated network could be trained on computer first to obtain good initial weights and then the training could be completed with the chip in the loop.

5. CONCLUSIONS

A VLSI implementation of a neural network has been demonstrated. Digital weights are used to provide stable weight storage. Analog multipliers are used because full digital multipliers would occupy considerable space for large networks. Although the functions learned were digital, the network is able to accept analog inputs and provide analog outputs for learning other functions. A parallel perturbation technique was used to train the network successfully on the 2-input AND and XOR functions.

6. REFERENCES

[1] J. Alspector, R. Meir, B. Yuhas, A. Jayakumar, and D. Lippe, "A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks", *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman Publishers, vol. 5, pp. 836-844, 1993.

[2] R. Coggins, M. Jabri, B. Flower, and S. Pickard, "A Hybrid Analog and Digital VLSI Neural Network for Intracardiac Morphology Classification", *IEEE J. of Solid-State Circuits*, vol. 30, no. 5, pp. 542-550, May 1995.

[3] M. Jabri, B. Flower, "Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks", *IEEE Tran. on Neural Networks*, vol. 3, no. 1, pp. 154-157, Jan. 1992.

[4] B. Flower, M. Jabri, "Summed Weight Neuron Perturbation: An O(N) Improvement over Weight Perturbation", *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman Publishers, vol. 5, pp. 212-219, 1993.

[5] G. Cauwenberghs, "A Fast Stochastic Error-Descent Algorithm for Supervised Learning and Optimization", *Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufman Publishers, vol. 5, pp. 244-251, 1993.

[6] P. W. Hollis, J. J. Paulos, "A Neural Network Learning Algorithm Tailored for VLSI Implementation", IEEE Tran. on Neural Networks, vol. 5, no. 5, pp. 784-791, Sept. 1994.

[7] G. Cauwenberghs, "Analog VLSI Stochastic Perturbative Learning Architectures", *Analog Integrated Circuits and Signal Processing*, 13, 195-209, 1997.

[8] C. Diorio, P. Hasler, B. A. Minch, C. A. Mead, "A Single-Transistor Silicon Synapse",*IEEE Tran. on Electron Devices*, vol. 43, no. 11,pp. 1972-1980, Nov. 1996.

[9] C. Diorio, P. Hasler, B. A. Minch, C. A. Mead, "A Complementary Pair of Four-Terminal Silicon Synapses", *Analog Integrated Circuits and Signal Processing*, vol. 13, no. 1-2, pp. 153-166, May-June 1997.

[10] C. Mead, *Analog VLSI and Neural Systems*, New York: Addison-Wesley, 1989.