An analog VLSI design for a neuron with a choice of learning rules

Mark James Neal

Aberystwyth, UK

Received 27 January 1998; accepted 22 March 1999

Abstract

An analog implementation of a neuron using standard VLSI components is described. The node is capable of both delta-rule and simple error-correcting learning. Decomposition into functional blocks allows the parts of the design to be easily separated and understood. The connectivity problem is eased by serially encoding inputs so that all nodes in a layer are connected to a single line carrying activations from the previous layer. Performance implications of the architecture are considered. The design was simulated with the Spice transistor level simulator. Schemas for interconnection of large numbers of nodes and simulations of the circuitry required are presented. Results show that effective learning is achieved by both algorithms. Implementation of multiple learning rules in a single neuron is demonstrated as an effective way of increasing flexibility in neural network hardware implementations. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Analog VLSI; On-chip learning; Delta-rule learning; Bucket-brigade

1. Introduction

Implementation of neural networks in VLSI is usually considered only for dedicated one-off applications which require a single learning rule or more often a recall-only architecture. This is mainly due to the difficulty of implementing flexible architectures or multiple learning rules in hardware. The design presented here provides a modular design for a neural network node which can be reconfigured to provide various different functionalities at run-time. The provision of a variety of
learning rules which is central to the construction of general-purpose neurocomputers, is shown to be possible and efficient with respect to the use of chip area. This flexibility is achieved by decomposing the node into simple functional blocks which can be switched in and out using simple control signals.

The design uses analog techniques in order to take advantage of the efficient use of space which can thus be achieved. It is also possible that the inaccuracies which can occur within analog circuits are beneficial to some neural network algorithms. The analog node described was simulated and developed using the Spice transistor level simulator, with device characteristics similar to those produced by the 3 μm fabrication process at Edinburgh. The design is viable and implementable with current technology.

The division of the node into functional blocks improves the prospects of gradual upgrading of the design, and is the first step toward implementation of general-purpose neurocomputing chips.

2. The node design

The functionality of a feedforward node lends itself very well to decomposition into the following blocks: the weighted synapses, the summing section of the node, the active part of the node implementing the transfer function, and the weight update circuitry. In an analog implementation of this model the summing function is trivial, and consists of a single capacitor. For this reason it was combined with the active part of the node into a single block. The more complex sections of the node are those containing the weighted synapses, and the weight update circuitry. The synaptic section is further decomposed into two parts, the part which stores the weights, and the part which implements the multiplication function. The weight update circuitry (controlled by a single signal for each different learning rule) adds and removes charge from the weight capacitors as required by that rule. Many researchers have found synapses to be problematic as they must fulfil both multiplication and weight storage functions (e.g. [11,4]). There are two ways in which synapses can be optimised (with regard to silicon area), reducing the area used for storing the weights which is currently difficult, and reduction of space in implementation of the multiplication function which can be more readily addressed by current technology.

1 A certain lack of accuracy i.e. noise is certainly not detrimental, and may be beneficial in some circumstances. [13,1,10].
2 3 μm p-well CMOS process of the Edinburgh Microfabrication Facility, Univ. of Edinburgh, UK.
3 Capacitors seem essential for weight storage, however, there are problems; leakage can be reduced only by the use of specialized processes and size can only be reduced at the expense of increased perimeter per unit area, and therefore increased leakage. Implementation of less leaky capacitors seems important in improving the viability of VLSI neural systems.
4 Other implementations have attacked the problem by reducing the complexity of the multiplier to three or four transistors each and ignore the errors introduced [7,3,4].
By serially encoding the inputs to the node, the use of a single multiplication circuit could be achieved. This allowed the use of a more complex and more accurate circuit than would have been possible if a multiplier had been included at each synapse. The serial encoding of the inputs also eases the connectivity problem as all nodes in a layer can be connected to a single line which carries all the output values from the previous layer. These values are applied to the input line one at a time, using a bucket-brigade delay line to sequentially apply the inputs. The signals used to pulse the bucket-brigade are also used to clock a single pulse down a simple shift register thus synchronizing activation of each of the weights with the associated input from the bucket-brigade (see Fig. 1). The inputs to the node are fed to one input of the Gilbert multiplier [8] circuit and the weights are each in turn fed to the other input. The multiplier output is stored as a quantity of charge on a capacitor. The voltage on this capacitor then represents the sum of the products of the weights and their respective inputs. This capacitor feeds the axon (which is a simple three transistor amplifier) and careful choice of device characteristics causes its output to be a sigmoid function of the voltage on the capacitor.

The updating of the weights is carried out by a well-separated piece of circuitry which is connected to the weight array output line (which is used to increment or decrement the weights), the target value for the node and the output line from the node (which is held at the current activation level during the learning pass). The small number of interaction points between the node and the learning circuitry makes possible the addition of (very simple) isolation circuitry and thereby a choice of learning rules which can be selected via a single control line. The weight update circuitry presented below implements a close approximation to the delta-rule, but as there is only a single copy of the learning circuitry per-node much more complex learning circuitry could be used without a large penalty in terms of chip-area. A second block of weight update circuitry which allows error-correcting learning to be selected is also shown, in order to demonstrate the possibility of switching between multiple learning rules.

**Fig. 1.** The functional decomposition employed in the design.
2.1. The weight storage array

The weights for the node are stored as voltages in an array of capacitors $c_1$ to $c_n$ (see Fig. 2).

Because the capacitor array uses a common output line, which has a significant capacitance, it is necessary to reset this line between the selection of weights. Failure to do so results in incorrect weight values being applied to the multiplier.

2.2. The multiplicative synapse

Each of the weight/input pairs is applied in turn to the multiplicative synapse as the input stream arrives at the node. This serial presentation of each of the input values in turn to the multiplier both dramatically reduces the number of transistors required for a node, and allows all of the inputs to a node to be passed down a single line, allowing much greater connectivity to be achieved. Fig. 4 shows the overall connectivity schema and demonstrates the small number of connections required between layers. The cost is execution time, which increases linearly with the number of connections per node. The circuitry for the synapse is shown in Fig. 2.

2.3. The axon design

The axon used is a simple three transistor amplifier which takes its input from the capacitor which stores the sum of the multiplier outputs. The resulting output of the axon provides the activation level of the node. The design of the axon is shown in Fig. 3.
2.4. The node function

From the above it can clearly be seen that the multiplicative synapse computes the products of the weights and inputs:

\[ w_{ij} \]

and that the capacitor between the synapse and axon computes the sum of these products:

\[ \sum_{j=0}^{n} w_{ij} \cdot j. \]

The final stage of the node, the axon amplifier, computes a function \( f \) of the sum of the products, where \( f \) approximates a steep sigmoid function. Thus the functionality of the node can be summarized as:

\[ a = f \left( \sum_{j=0}^{n} w_{ij} \right), \]

where \( a \) is the activation level of the node.

3. Input and control signals for the node

The signals required during a simple feedforward pass are (see Fig. 1):

- A selection signal (S1) to put the required capacitor voltage onto the weight array output line, which is drawn from a simple shift-register.
- A reset signal (S2) to clear the weight array output line, after each input-weight pair has been multiplied.
- A reset signal (S3) to clear the summing capacitor, at the beginning of the forward pass.
- The input signals (S4) from the previous layer consisting of a series of pulses equal in number to the number of connections to this node.
In order to perform a learning-pass the following additional signals are required:

- A signal (S5) to isolate the summing capacitor from the synapses, and thus retain the output state of the node.
- A signal (S6) to activate the weight update circuitry, by switching on isolation transistors.
- A target value (S7) with which to compare the actual state of the node.

The complexity of circuitry required to generate these signals can be minimized by using a simple shift-register to pass a synchronization pulse along the capacitor array. The pulse required to cause the next step down this shift-register can also be used to perform other functions such as resetting the output from the weight array. The synchronization pulse passing down the shift-register can also be used, after activating each of the weights in turn to reset the summing capacitor after a feedforward pass. This simply involves connecting the gate of the reset transistor to the tail of the shift-register (i.e. using S1 to generate S3).

A simple bucket brigade arrangement is also used to pass the inputs to each node as required. This is achieved by connecting the outputs from nodes in preceding layers to different points in the bucket-brigade and passing their output voltages down the bucket brigade to the tail which is connected to the input line to the node(s) in the next layer (see Fig. 4). The same control pulses can be used for both the bucket-brigade and the shift-register in order to obviate the need for complex synchronization circuitry.

### 3.1. The weight update circuitry

Delta-rule-based circuitry and simple error-correcting circuitry are implemented for the node, and can be selected by a single control-line each. This is possible due to the effective modularity of the circuitry made possible by the presence of a single synaptic multiplier. This allows a single line to be used for updating all the weights stored at the node, as well as a single line to be used to sense the incoming input-values for each connection to the node. Schematically the delta-rule circuitry has only four connections to the rest of the node, each of which can be isolated sufficiently well using a single transistor. The error-correcting circuitry is even simpler, and has only three connections to the rest of the node.

Examples of the weight changes performed and the activation levels resulting, from the use of these circuits when simulated using SPICE are shown in Figs. 7-10. The simulations were performed using the complete node as described here with two weight capacitors, and signals for those two inputs being received after propagation down a bucket-brigade delay-line as described above.

#### 3.1.1. Error-correcting learning circuitry

The circuitry shown for the simple error-correcting learning is intended to show that simple learning algorithms can be effective, and to demonstrate that the node is capable of selecting between two different learning rules. The circuit simply detects
whether the target voltage is greater than or smaller than the actual output voltage, and increases or decreases the weight if the input to the synapse is large enough. This mediation of the weight change by the input voltage is achieved using a single transistor as a multiplier (see Fig. 5). This is effectively an implementation of a simple version of the delta-rule which cannot treat low-voltage inputs as significant, and as such is limited in the functions that it will be capable of learning. This is due to the mediating transistor acting as a two-quadrant multiplier rather than a full four-quadrant multiplier that the delta-rule requires.

3.1.2. Delta-rule learning circuitry

The delta-rule learning circuitry as presented implements the delta-rule [14] to a close approximation. The circuits used are standard ones and no clever tricks or fringe effects are employed. The two operational amplifiers shown in the circuit...
Fig. 5. The weight-update circuitry for a simple error-correcting learning rule.

Diagram are simple eight transistor units connected as differential amplifiers (see Fig. 6). One conducts when the target voltage is greater than the actual output voltage, and the other in the opposite case. The outputs from these two amplifiers then affect the state of the push–pull pair of transistors connected to their outputs. The state of the common connection of this pair of transistors provides a voltage which represents the difference between the target and actual output values for the node, \( t - o \) where \( t \) is the target value for the node and \( o \) is the current output value. This common connection is then used as one of the inputs to a Gilbert multiplier [8] identical to that used in the synapse of the node. The other input to the multiplier is taken from the current input to the node (which is associated with the weight which is to be updated). The effect of this multiplication is thus:

\[
(t - o) \times i,
\]

where \( i \) is the input value for the current weight/synapse. This is the weight-update value called for by the delta-rule as defined in [14]

\[
\Delta w_i = (t - o)i_i,
\]

where \( \Delta w_i \) represents the weight change required for the current weight and \( i_i \) represents the input value for the current weight/synapse.

\[5\] A single operational amplifier could be used if a double-ended power supply was used; however as a single-ended power supply is used everywhere else it was deemed to be extravagant to include one solely for this circuit.
4. Evaluation

4.1. Implementation in silicon

The circuits presented above were all designed using components available in a standard MOS fabrication process. The evaluation presented below uses the circuits presented above with SPICE models based on a 3 μm process. The transistor parameters used are as given in Table 1. The implementation of circuits containing passive components in VLSI is relatively problematic compared to the construction of circuits consisting solely of transistors. However, the construction of capacitors of small values is now relatively common using any of a variety of methods [2]; probably the most effective is the use of a simple metal over diffusion structure. The capacitors used in the simulations presented below were 0.5 pF although smaller values could undoubtedly be used especially if leakage characteristics could be improved. For capacitors of 0.5 pF a region measuring approximately 80 μm × 80 μm would be required to implement a simple metal over diffusion capacitor [15]. Other techniques could be used which may be more efficient in terms of space, the most promising of which is to use transistor structures themselves as capacitors [15].

The resistors used in the delta-rule weight-update circuits presented are of non-critical values, and are simply used as a voltage divider. The resistance values in the error-correcting learning circuitry are more critical in value and would require careful layout in order to allow for process variations and local fabrication variations. The
<table>
<thead>
<tr>
<th>Gate length</th>
<th>Gate width</th>
<th>Threshold voltage</th>
<th>Oxide thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 μm</td>
<td>7 μm</td>
<td>0.8 V</td>
<td>45 nm</td>
</tr>
</tbody>
</table>

use of transistors biased in the triode region for the less-critical resistance values would be the most efficient implementation method. This would probably prove unsatisfactory for the more critical values which would require a structure with a more linear response. These may be implemented either as diffused resistors for the lower resistance values or as well resistors for the higher resistance values [9].

The final component which is used in the circuitry is an operational amplifier. This is used in the delta-rule learning circuitry for accuracy when generating the weight updates. The circuit used is a simple two-stage design using eight transistors and a single capacitor [2]. The circuit was chosen as the a compact design with a low component count and sufficient accuracy, gain and output range. The circuit is shown in Fig. 11.

4.2. SPICE simulations

SPICE simulations of the node learning two-input patterns were carried out. The diagrams showing the results are summarised as given in Table 2 follow for the delta-rule.

Examination of the voltage traces in Figs. 7 and 8 reveals that target voltages are attained in a similar fashion when the target states are approached from both an over-excited and an under-excited state. The small amplitude oscillations seen about the target voltage are due to a slightly excessive learning rate. Reduction of the learning rate removes the oscillations, but due to weight decay the target voltage is reached very slowly. This is because the size of the weight changes is similar to the weight-decay between learning epochs. Control of the learning rate is achieved by varying the resistance between the output of the weight update circuitry and the weight storage array. This resistance is in fact a single transistor which is controlled by a voltage applied to its gate.

Fig. 9 shows some equivalent runs to Figs. 7 and 8, but using a smaller learning rate, and exhibits the slower convergence expected when using a smaller learning rate. Thus a compromise value for the learning rate to be used must be struck in a similar way as is required for conventional neural network simulations.

Similar results were obtained for the error-correcting learning rule (results not shown here).

4.3. Weight update and decay

The upper plot in Fig. 10 shows the trace for two weights throughout the training period and the lower plot shows a single trace in more detail. The upper plot shows
Table 2

<table>
<thead>
<tr>
<th>Target (V)</th>
<th>Initial weights (mV)</th>
<th>Final weights (mV)</th>
<th>Fig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>800 800</td>
<td>748 754</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>800 800</td>
<td>728 730</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>700 700</td>
<td>750 754</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>700 700</td>
<td>729 734</td>
<td>8</td>
</tr>
</tbody>
</table>

Lower learning rate

| 4          | 700 700               | 748 749            | 9    |
| 1          | 800 800               | 730 732            | 9    |

Fig. 7. Response of neuron whilst learning using the full delta-rule circuitry. The picket-fence traces are the neuron response and the horizontal traces the target values in both cases. Initial weights were set to 800 mV.

that weight changes become smaller as the target value is approached, resulting in a curve whose slope reduces as the final value is approached. The lower plot shows the effect on a weight of the various stages during the activation cycle for the node. The upward jumps in voltage are due to the learning circuitry injecting charge during the learning phase of activation. The downward steps are due to the charge which is lost from the weight capacitor when the weight is accessed for use in the multiplication process with its associated input. From the diagram, it can be seen that the weight is being updated on only every other activation cycle. This is due to the necessity of calculating the node’s current output before calculating the weight changes required, in exactly the same way as with any other neural network implementation. Clearly, when the size of weight increments is the same
Fig. 8. Response of neuron whilst learning using the full delta-rule circuitry. The picket-fence traces are the neuron response and the horizontal traces are the target values in both cases. Initial weights were set to 700 mV.

Fig. 9. Response of neuron whilst learning using the full delta-rule circuitry with a smaller learning rate. The picket-fence traces are the neuron response and the horizontal traces are the target values in both cases. Initial weights were set to 700 mV.

as the size of the weight decay caused by the weight access the learning algorithm ceases to improve the solution and the weight value effectively remains constant. This situation can be seen in Fig. 10 (bottom) where the the network has effectively converged.
Fig. 10. Traces showing the weight changes during learning using the full delta-rule circuitry.

Fig. 11. The simple operational amplifier used in the delta-rule learning circuit.
4.4. Performance considerations

The time required for a node activation cycle as currently configured is 500 μs. This consists of two time slices, one for each of the two weight/input pairs. Each weight requires 250 μs to be addressed whether learning is in progress or not. So the time required for a complete node cycle is simply the number of weights at the node times 250 μs. So for a node with a realistic number of weights, say 20, the node cycle will be 20 × 250 μs = 5000 μs = 5 ms.

The 250 μs access time for a weight allows a very simple calculation of the number of calculations per second for a single node. This is simply \(\frac{1000000}{250} = 4000\) connections/s. If we take a large but realistic number of nodes to be implemented on a single piece of silicon, say 10000 on 1.5 cm\(^2\) [12], then we arrive at a figure of 40 000 000 connections/s. This is 150 times faster than a small simulation program carrying out delta-rule learning on a Sun Microsystems Ultra Enterprise 2 model 2170, which takes 3.1 s to execute 800 000 connections (Fig. 11).

To date, no attempt has been made to optimize the speed at which the circuitry operates, but it is very likely that cycle times much shorter than those shown here will be achievable. The limiting factor (in terms of speed) will be the time required to deposit sufficient charge on the summing capacitor used as the input to the axon. Clearly, using a smaller implementation technology and thereby smaller capacitors will increase both speed and the number of nodes which will fit onto a given area of silicon.

5. Discussion

The main aim driving the design of the node was to make the addition of learning circuitry relatively simple. This has been achieved, through the use of the capacitor array and single multiplier to perform the functions of the synapses. Multiplexing the inputs to the node has effectively allowed many of the other parts of the node to be drastically reduced in size. Only one multiplier and set of learning circuitry is required to serve all the synapses, and this combined with the reduction in area due to the single transmission line between layers of nodes makes the design viable in terms of chip-area. The penalty that is paid for the flexibility (both in connectivity patterns and in selection of learning rules) is that networks constructed in the way suggested will only perform with O(n) where n is the number of inputs per node. This must be contrasted with simulations as well as fully parallel implementations.

The number of weights implemented at each node dictates the maximum connectivity implementable with this design. This is clearly limited by the silicon area available and the number of nodes which are required; thus a balance must be struck between the number of nodes and the maximum connectivity available. Fewer nodes will require fewer weights to implement fully connected networks, and so with smaller numbers of nodes full connection will be possible. With larger numbers of nodes there will be a trade-off between connectivity and size of network.
The construction of many different types of learning circuitry within the same nodes brings the advent of a truly flexible general-purpose neural processor a step closer. This has only been made possible by using serial encoding of the signals between layers in the network. There is a further problem which such encoding brings, which is that the network must indeed consist of layers of significant numbers of nodes in each layer for there to be any real speed-up over software simulation.

6. Conclusions and future work

The obvious extension of this work is the implementation of the full back-propagation algorithm. The alteration of this rule and the circuitry presented to implement the back-propagation algorithm requires the addition of a mechanism for generating the derivative of the output function of the node, and a mechanism for the passing of the error values to the previous layer. The use of a bucket-brigade would be the most obvious way to implement this.

The node design described shows a realisable design for an artificial neuron with multiple on-chip learning rules using current technology. Implementation of a flexible and reconfigurable neural chip is not as distant as it may appear; further work which is required concerns the architecture and control of such neuro-computers in order that the best use can be made of the inclusion of multiple on-chip learning rules in a multi-purpose node design such as that presented here. The possibilities for enhancement of performance also require detailed investigation. The pinnacle of achievement for such a design, would be one capable of being reconfigured in software to implement many different neural network algorithms efficiently for both learning and re-call.

7. For further reading

The following references are also of interest to the reader: [5] and [6].

References


Mark Neal completed his B.Sc. and Ph.D. at the University of Wales Aberystwyth in the Department of Computer Science. His Ph.D. was completed in 1993 and was concerned with the design of systems combining conventional and neural computing elements. He then went on to work on a variety of research projects using neural networks and artificial intelligence techniques for data analysis and robotics. He is now a lecturer at Aberystwyth with research interests in artificial intelligence, neural networks and their application to robotics.