An investigation of practical issues in translating algorithms based on back-propagation into analogue, VLSI circuits

Robin Woodburn

A thesis submitted for the degree
of Doctor of Philosophy to the
Faculty of Science
University of Edinburgh
1996

Abstract

One of the most widely-used artificial neural networks is the multi-layer perceptron, trained by error-back-propagation (the 'back-propagation algorithm'). Commonly, the network is implemented as a serial-computer simulation, but there has been considerable interest in translating it into hardware. The most difficult translation into analogue VLSI is the 'learning' part of the algorithm, that is the part which involves calculating the output errors and making appropriate modifications to the analogue weights representing the connections between nodes. For this reason, most analogue hardware implementations train weights held off-chip in a digital representation; the weights are converted to an analogue representation for storage on the chip which comprises the network.

This thesis examines the Virtual Targets algorithm, based on back-propagation, but with some modifications which render it more amenable to translation into analogue VLSI circuits which can 'learn on-chip'. I describe several circuits, designed to exploit our research group's pulse-stream approach to analogue VLSI, which provide four-quadrant multiplication, and calculate differences, signs and error-derivatives. Results, from simulation and from a chip fabricated with the circuits, are given.

A consideration of other approaches to the problem of learning on-chip makes it clear that key issues are weight-storage, and a means of modifying the weights. I explain why calculating exact weight-changes is difficult, and give the results of simulation experiments leading to a further simplification of the Virtual Targets algorithm which makes it possible to train the network using fixed increments and decrements of the weights. I show the results of tests of circuits on a second chip, designed with implementation of the entire algorithm in mind, and assess the likelihood of such an implementation being successful.

I place this analysis in the context of the search for 'intelligent' machines, and ask how far designs such as my own might contribute to such a machine. I also make some suggestions on the most fruitful directions for analogue designs of artificial neural networks.

Declaration

This thesis is entirely my own work. The work I describe in it is also my own, unless otherwise indicated.

Acknowledgements

Ah! The invidious chance to thank all those immediate colleagues who've helped me during the course of my project. Alan, my supervisor, for giving me the opportunity to work in what has been the most demanding and most interesting work I've ever done, and for giving me so much freedom to tackle things the way I wanted. Martin, my second supervisor, who, as he explained something for the fourth time, must have wondered if I would ever get it, but who valiantly resisted the burning temptation to call me 'turnip head'. Alister, Donald and Steve, who steered me through a difficult first year, and Alister again for all the help in the details of circuit-design and testing. Drew, Andy and Geoff, three terrific engineers, for being great to work with and for making what seemed impossible just a question of approaching it the right way; they'll surely be successful in their new careers. Andy and Drew, again, and Dwayne for all the constructive criticism of the thesis. Torsten for the talks on on-chip learning. Mike, Emma, John and Richard for the beer and blether. Catherine, for thinking like me. To all of these and Peter, Andy M, David, Bill, Kostis, Colin, Ken (never won an argument yet), Kevin, Anne, Lynne, Andy C, Mark and Neil, the colleagues I've worked with, for their friendship which made my time at Edinburgh such a good one.

Finally, it is traditional to thank your nearest and dearest, not that they had much choice in the matter. Ruth had to put up with an awful lot but, on the bright side, she can now talk with confidence about the characteristics of four-quadrant multipliers and the virtual targets algorithm.

Dedication

To my mother and father, who would have been very pleased.

Table of Contents

Chapter 1

Introduction

This thesis is an investigation of the possibility of building a machine, in the form of a microchip, using analogue and digital circuits, which can itself adapt its outputs in light of changes to its inputs. More specifically, it investigates the translation of a variant of a well-known neural-network algorithm into hardware in such a way that the weights in the network can be adjusted by circuits on the chip, so that the chip can "learn" autonomously, that is without the support of a conventional computer. To consider this possibility, I have had to examine a whole range of issues associated with designing and simulating neural networks, and building and using chips, and these are explored in the later chapters.

The emphasis, like much of research in electronic engineering, is on "how" : How do we change weights in an artificial neural network (ANN)[1] so that it will respond correctly to a set of patterns? How do we approximate a parabola with an electronic circuit? How do we build a four-quadrant multiplier? How do we put a set of electronic components together to build a system on a chip? How do we drive that system so that it will perform like an ANN?

As someone who studied psychology, not engineering, at university, I very quickly noticed the emphasis on "how" in engineering research. The *techniques* by which

[1] ANNs are computational systems which are held to mimic, to some degree, the computational abilities of biological systems by using large numbers of simple, interconnected nodes. The network adapts to changing inputs by having its weighted connections modified in strength.

one can carry out a task — rather than the "why" questions, the basic reasons for investigating the techniques — seem to dominate.

In Appendix B, I have placed quotations from a selection of leading figures who work in the field of ANN research. Some of these quotations may seem faintly ridiculous; they did to me. Yet I had a sympathy for the people I have quoted, because they are asking "why" they are doing their research. Their implicit answer is that they are trying to emulate in some way the processes, like seeing and hearing, which people find so easy and machines so difficult. Some of the claims researchers make are a little far-fetched, and so I have included quotations from Mead and Hinton who have an objective to understand, and perhaps emulate, brain function, but are nevertheless, it seems to me, level-headed about how close we are to success.

Because I think the "why" questions are important, I felt unable to ignore some more fundamental issues about what we can reasonably expect from neural networks, including ones instantiated in hybrid analogue-digital hardware, and whether the "learning" exhibited in such networks is comparable to what we commonly call learning in people. Partly, this was because I am fascinated by these issues; partly, because some of the claims of researchers have left me incredulous; and, partly, because, amidst 20-hour days trying to beat some design software into behaving sensibly, it is difficult but important to look where you are heading, and why. Appendix A is therefore devoted to these topics.

1.1 Aim of the project

The aim of the project was to study the issues raised by the translation of the virtual targets algorithm[2] into analogue VLSI circuits.

We already had circuits, designed earlier by some colleagues, to perform some parts of this task. The questions we asked ourselves initially were as follows.

---

[2]Strictly speaking, the algorithm embodies a feed-forward, multi-layer perceptron, trained by back-propagation of the derivatives of the errors. The popular version is commonly known as the 'back-propagation algorithm', and the virtual targets algorithm is a variant of it.

Could we invent circuits for the remainder of the task? If so, could we invent a scheme to put all the parts together in a system? If this was possible, could we make some assessment of the system's capability to learn on-chip?

To meet this aim, I designed one chip to test some preliminary ideas, re-assessed the algorithm in light of what I had learned, and designed a second chip which would be capable, if embedded in the right system, of testing the complete algorithm.

I describe these designs in a way which explains how my ideas developed as the project progressed, and why I came to my final conclusions.

1.2 Detailed goals

Not all the detailed goals, of course, were clear at the start of the project; they emerged over time. The following list gives an idea of the technical goals which had to be achieved.

1.2.1 Goals for the circuits

- Establish what are the functions making up the virtual targets algorithm, and how these functions fit together.
- Given that the existing Epsilon chip can perform a forward pass through an multi-layer network of perceptrons, decide what parts of that can be used, and what additional circuits are required for the virtual targets algorithm.
- Hence design circuits to perform
  - four-quadrant multiplication
  - a difference calculation
  - a sign calculation
  - a 'sigmoid-prime' function.
- Simplify the algorithm. In light of the simplification, establish how to

  - make fixed increments or decrements to a stored weight
  - make scaled increments to stored targets
  - carry out an 'error pass' through the network to back-propagate error-terms.

- Find a way of putting all the circuits together in a system which could test if learning is possible on chip, and make a simple test of it.

1.2.2 Other goals

In view of the importance of links between areas in what is a multi-disciplinary field, I also set myself the goal of making links with work on machine intelligence :

- Assess how true are claims that artificial neural networks (ANNs) are 'brain-like', and how far they contribute towards the idea of intelligent machines. Make suggestions about how engineers should view these ideas.

- Assess the value of analogue implementations of ANNs compared to digital ones.

- Suggest how analogue circuits could best contribute to ANN research.

1.3 Structure of the thesis

The thesis is structured as follows :

Chapter 2 reviews three algorithms from the point of view of the ease with which they can be implemented in hardware.

Chapter 3 examines issues in on-chip learning, considers the published literature, and looks in detail at several examples designed by other research groups.

Chapter 4 describes the circuits, designed for the first chip, to implement some basic functions of the virtual targets algorithm, together with their simulation and test results.

Chapter 5 shows how I simplified the virtual targets algorithm in light of my progress.

Chapter 6 puts in place the final pieces of the jigsaw which is a chip for on-chip learning. It describes the circuit designed to modify weights and the architecture of a second chip which, in a complete system, might instantiate the entire algorithm.

Chapter 7 gives results of tests of the second chip and assesses the merits of the circuits for on-chip learning.

Chapter 8 presents my final conclusions.

Chapter 2

A Review of Relevant Algorithms and Hardware Implications

2.1 Introduction

This Chapter introduces some basic terminology necessary to understand the remainder of the thesis and considers the motivation for the study. It then looks at the mathematical background to the back-propagation algorithm, considers two alternatives to it, and then compares their software performance and hardware implications. Finally, the Chapter describes the empirical effect of one of the terms in the back-propagation algorithms.

2.2 Terminology

This section introduces some of the terms that are necessary to understand the algorithms discussed in this chapter.

Artificial neural networks (ANNs) are a computational paradigm in which groups of simple processors, or processing nodes, are connected in parallel, generally in layers. (The terminology used to describe the layers varies; the following description establishes the terminology used throughout this study.) The first, or input, layer receives input signals from the outside world (see figure 2–1). No node in the input layer has any computational function, but serves merely to distribute each of the input signals to the succeeding layer. The output layer receives inputs from the preceding layer and provides output signals to the outside world. Between the input layer and the output layer may be any number of

intervening layers, known as hidden layers, but often the number of hidden layers is only one. Unlike the input-layer processors, each of the processors in the other layers has some computational function. The nature of that function may vary from layer to layer, but generally all the processing nodes in a single layer have the same computational function.

For the networks considered in this thesis, the processing nodes are called perceptrons, and a network with several layers of such nodes is called a multi-layer perceptron (MLP) network.

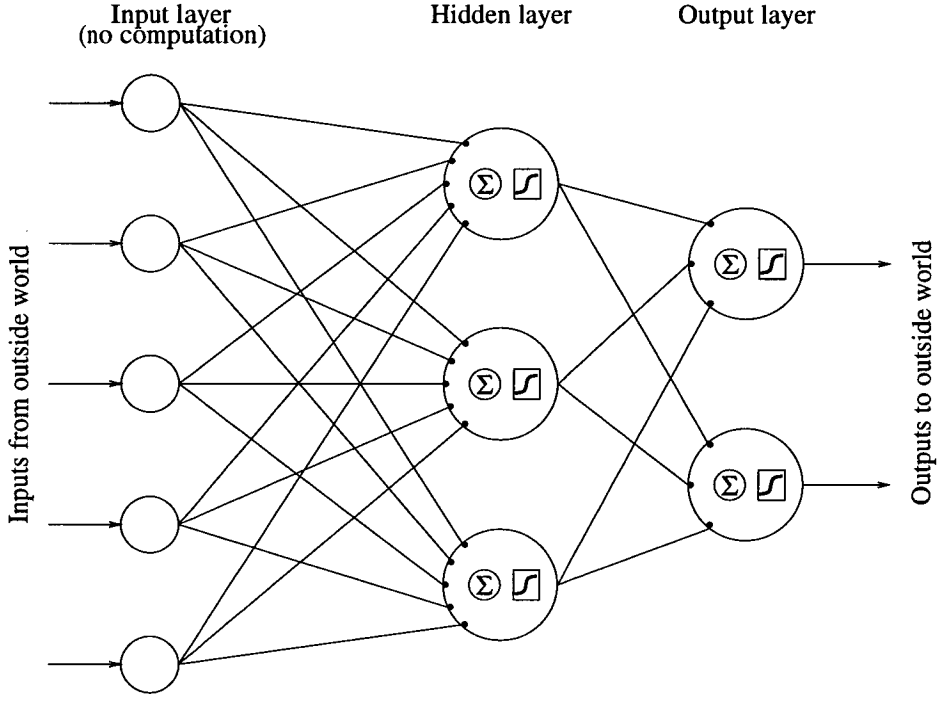


Figure 2–1: *A simple, fully-connected multi-layer perceptron*

Each processing node in the hidden or output layers receives several input connections, and provides a single output connection with a value called its 'output state' (see Figure 2–2). Each of the input connections, representing the state of another node, is weighted, and the weighted values are then summed, to produce the node's 'activation'. The activation value is then mapped by a mathematical function into an output state. For reasons explained in the literature (Beale and Jackson, 1990, Hertz et al., 1991), this mathematical function is usually a differentiable but non-linear function, hence the common term 'non-linear mapping function'. If the node is in a hidden layer, the output state will be dis-

tributed to nodes in the next layer. If the node is in the output layer, the output state will represent one of the outputs of the network.

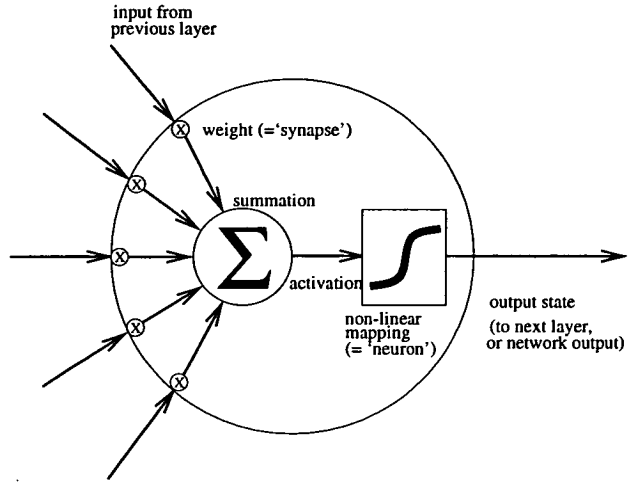


Figure 2–2: *A computational node, found in the hidden and output layers*

Because work on ANNs originally drew inspiration from biological networks, two of the hardware components of the processing nodes have been given the biological names 'synapse' and 'neuron'. The degree to which the artificial components resemble the biological ones is very slight, but the terminology occurs so frequently in the literature that I have reluctantly adopted it. There is no great consistency in the way different groups use the terms; components are lumped into synapses or neurons as is expedient. However, broadly speaking, the point at which one processing node receives a connection from another node, is called the 'synapse', and it is here that the input connection is weighted. Since weighting is multiplication, 'synapse' is a synonym for 'multiplier'. The mechanism by which the weighted connections are summed can, at least in analogue chips, be accomplished so simply that the summation process has not been deemed worthy of a biological name, and this mechanism is usually considered part of the synapse. The term 'neuron' is usually given to those components that calculate the non-linear mapping function to produce the final activation value.

Processing in such a network proceeds (at least conceptually) as follows. A series of input vectors or patterns (which can be digital or analogue data) is presented, vector by vector, to the input layer, which distributes each element of the vector to each of the nodes in the hidden layer. Each hidden-layer node simultaneously processes the vector by weighting and summing the input elements and perform-

ing a non-linear mapping to produce, at the node's output, one element of the hidden-layer output-state vector. This vector is in turn presented to each node in the output layer, which processes it in the same way as the hidden-layer processed the inputs. The output-layer's output vector is the output of the network. The whole process of transforming a series of input vectors into a series of output vectors is known as a 'forward pass'. A forward-pass is, then, a series of repeated *multiplication—summing—non-linear-mapping* operations.

'Learning' consists of modifying all the weights in each of the hidden- and output-layers after each forward pass, according to a particular learning rule. The modification is carried out, over many iterations, in such a way that either the network is forced to produce a specific output vector for a specific input vector, or it stabilises in a state where the response to different input vectors is in some way interesting to the operator.

2.3 Motivation for this study

Our thinking on this study began with a consideration of the type of MLP applications that would require 'on-line' learning, that is learning where the weight-set would be changing constantly. In other words, the characteristics of the application would be such that the weights necessary for successful operation could not be calculated beforehand.

For many applications, developing the weight set during computer simulation may be inconvenient because of the long training times involved, but the methodology is not fatal to their success. For example, financial applications, database-retrieval, hand-writing recognition and medical diagnosis may all, depending on the circumstances, be carried out 'off-line' to achieve acceptable solutions. Adaptability may not even be advantageous in these cases, since the output characteristics of the problem may have been well-defined.

In other applications, however, the input data may not be well-controlled, may arrive in large quantities in analogue form, and may require to be dealt with in real time. Examples are robotics and sensor-motor control, speech recognition, natural-language applications, process-control, image-processing and machine vision. In these circumstances, the neural network system needs to be adaptable at high speed. The point about such systems is that they may require to re-

spond to inputs some of which are stable or very slowly-changing, and so can be pre-learned while other inputs may be rapidly-changing and so require 'learning' or 're-learning' over a period of time. On-chip learning brings closer the prospect of systems with this capability, and hence the possibility of embedded or autonomous systems, where analogue VLSI finds its strongest justification.

2.4 Back-propagation : gradient-descent using back-propagation of error

Our preference was for an algorithm that would provide a system capable of dealing with real-world applications of neural networks. The most obvious choice is then back-propagation (Rumelhart et al., 1986), because this algorithm can accommodate a whole range of pattern-recognition and signal-processing tasks from medical diagnosis to air-combat manoeuvre selection (Maren et al., 1990).

Back-propagation is a gradient-descent algorithm, so called because it attempts to minimise a measure of error : the error measure is envisaged as a surface, like that of hills and valleys in multi-dimensional space, in which the network, to find a solution, must reach a minimum, preferably *the* minimum of the lowest point in the surface. To move towards a solution, that is in the direction of minimum error, the network requires a gradient term for each network weight. These gradient terms are used to reduce the network error by changing the weights on the connections in the direction of reduced error.[1]

2.4.1 Mathematical background to back-propagation

Mathematical analysis results in a series of learning equations, described in Appendix D, which are derived from a gradient-descent minimisation of the

[1]There are several variations on back-propagation as a means of training MLPs (Hertz et al., 1991). Cost functions other than the sum-of-squared-errors can be minimised, for example measures of entropy. Optimisation techniques other than gradient descent are also available, for example Newton's method and conjugate gradient methods. None of these variations is considered in this thesis.

sum squared error of the output neuron. While the mathematical derivation is easily accessible in various textbooks, for example (Beale and Jackson, 1990, Hertz et al., 1991), and so is not repeated in full here, a description of the key features of the analysis is helpful for a comparison of the back-propagation, virtual targets and weight perturbation algorithms, to be given shortly.

The key points are these :

1. The algorithm aims to minimise the error at the network output by changing weights at the interconnections between nodes. The error is defined as proportional to the square of the distance of each output-node's actual output from the desired, or target, output.

2. To minimise the error, we require to know the overall change in the error with respect to each weight. We cannot compute such a change directly, but we can compute the change indirectly by successive application of the chain rule.

Notation

The notation used in the remainder of this section is as follows. E_p is the error function for pattern p, t_{pk} represents the target output for pattern p on node k, O_{pk} represents the actual output at that node, and w_{kj} is the weighted connection from node j to node k. For a three-layer network, layer i is the input layer, layer j the hidden-layer, and layer k the output layer.

Definitions

The following definitions apply :

- The error is defined as :

$$E_p = \frac{1}{2}\sum_k (t_{pk} - O_{pk})^2 \tag{2.1}$$

  The introduction of the $\frac{1}{2}$ simplifies the consequential mathematics.

- The activation for each unit for each pattern p (ie the weighted sum of the inputs to a node) is defined as :

$$net_{pk} = \sum_j w_{kj} O_{pkj} \tag{2.2}$$

- The output of each unit is a sigmoid function of the activation for that unit, and is defined as :

$$O_{pk} = \sigma_k(net_{pk}) = \frac{1}{1 - e^{-(net_{pk})}} \tag{2.3}$$

Finding $\partial E_p / \partial w_{kj}$

The algorithm aims to minimise the error at the network output by changing weights at the interconnections between nodes. We therefore require to know the overall change in the error with respect to each weight $\Delta E_p / \Delta w_{kj}$, a computation which cannot be made directly. However, we can derive the computation indirectly by successive application of the mathematical chain rule, in the following way.

$$\frac{\partial E_p}{\partial w_{kj}} = \frac{\partial E_p}{\partial net_{pk}} . \frac{\partial net_{pk}}{\partial w_{kj}} \tag{2.4}$$

The term $\partial E_p / \partial net_{pk}$ is defined as :

$$\frac{\partial E_p}{\partial net_{pk}} = -\delta_{pk} \tag{2.5}$$

and δ_p is known colloquially as the "delta term". The term $\partial net_{pk} / \partial w_{kj}$ can be shown to simplify to O_{pj}.

Hence :

$$- \frac{\partial E_p}{\partial w_{kj}} = \delta_{pk} O_{pj} \tag{2.6}$$

and consequently, the weight-change to reduce the error is :

$$\Delta_p w_{kj} = \eta \delta_{pk} O_{pj} \tag{2.7}$$

where η is a gain term.

This equation applies to output and hidden layer alike, but the δ_p term is calculated differently for each layer.

Finding δ_p for the output layer

As before, we cannot compute δ_p directly, but we can do so indirectly by using the chain rule :

$$\delta_{pk} = -\frac{\partial E_p}{\partial net_{kj}} = -\frac{\partial E_p}{\partial O_{pk}} \cdot \frac{\partial O_{pk}}{\partial net_{pk}} \tag{2.8}$$

The term $-\partial E_p / \partial O_{pk}$ simplifies to $(t_{pk} - O_{pk})$.

The term $\partial O_{pk} / \partial net_{pk}$ is $\sigma'(net_{pk})$, ie the derivative of the output with respect to the activation of that node, or *sigmoid prime*, which can be shown to simplify to :

$$\frac{\partial O_{pk}}{\partial net_{pk}} = \sigma(net_{pk})(1 - \sigma(net_{pk})) \tag{2.9}$$

$$= O_{pk}(1 - O_{pk}) \tag{2.10}$$

Hence, for the output layer :

$$-\frac{\partial E_p}{\partial w_{kj}} = \delta_{pk} O_{pj} \tag{2.11}$$

$$= (t_{pk} - O_{pk}) \left[O_{pk}(1 - O_{pk}) \right] O_{pj} \tag{2.12}$$

and consequently the weight-change to reduce the error is :

$$\Delta_p w_{kj} = \eta_k (t_{pk} - O_{pk}) \left[O_{pk}(1 - O_{pk}) \right] O_{pj} \tag{2.13}$$

Finding δ_p for the hidden layer

Just as for the output layer, the delta term for the hidden layer is :

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial O_{pj}} . \frac{\partial O_{pj}}{\partial net_{pj}} \tag{2.14}$$

As for the output layer, the term $\partial O_{pj}/\partial net_{pj}$ simplifies to $\sigma'(net_{pj})$, and so to $O_{pj}(1 - O_{pj})$.

In expanding δ_p for the output layer, we were able to express the term $-\partial E_p/\partial O_{pk}$ as a function of the output error $(t_{pk} - O_{pk})$. Unfortunately, we have no explicit targets for the hidden layer, and so we must express the equivalent term, $-\partial E_p/\partial O_{pj}$, in some other way. As it turns out, we can show that this term can be expressed in terms of δ_p for the output layer, so that :

$$-\frac{\partial E_p}{\partial O_{pj}} = \sum_k \delta_{pk} w_{kj} \tag{2.15}$$

Hence :

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} O_{pi} \tag{2.16}$$

$$= \left[\sum_k \delta_{pk} w_{kj}\right] [O_{pj}(1 - O_{pj})] O_{pi} \tag{2.17}$$

and consequently the weight-change to reduce the error is :

$$\Delta_p w_{ji} = \eta_j \left[\sum \delta_{pk} w_{kj}\right] [O_{pj}(1 - O_{pj})] O_{pi} \tag{2.18}$$

$$= \eta_j \left[\sum (t_{pk} - O_{pk}) O_{pk} (1 - O_{pk}) w_{kj}\right] [O_{pj}(1 - O_{pj})] O_{pi} \tag{2.19}$$

We have circumvented the absence of explicit targets for the hidden layer by back-propagating the error (and the derivative of the sigmoid, the sigmoid prime)

through the weights which connect the hidden and output layers[2]. By this means we can establish the effect of hidden-layer weights on the output errors.

2.4.2 Problems with back-propagation

I make an explicit comparison of the consequences, for hardware, of the weight-change equations 2.13 and 2.19 later, in Section 2.4.1, but we can note two important points now : that the equations for each layer are substantially different; and that equation 2.13 for changing weights in a node in the output layer uses information local to that node, whereas equation 2.19 requires information passed back from the layer above. It is these two problems that encouraged us to consider alternatives to back-propagation.

There has been considerable progress recently in translating the back-propagation algorithm into hardware (for example (Valle et al., 1992, Jabri and Flower, 1992)), but the problem is a knotty one. Doubts persist as to the likelihood of success in practice (Tarassenko and Tombs, 1993, Cairns, 1995). For one thing, error gradient terms must be calculated and back-propagated in some form to previous layers in the network, which is in itself difficult to do. For another, changes in the weighted connections are frequent, and have to be very small, that is a high precision is required, otherwise the network will never find a solution or will be unstable in its solution.

2.4.3 Alternatives to back-propagation

Two options for attacking the problem of translating algorithms based on back-propagation into hardware have been explored at Oxford (weight perturbation) and Edinburgh (the virtual targets algorithm).

---

[2]Although only one hidden-layer is considered as part of this thesis, the approach outlined here is valid for any number of hidden layers.

2.5 Alternative approaches : the virtual targets algorithm and weight perturbation

This section describes alternatives to plain back propagation, in the form of the virtual targets algorithm and weight perturbation. Virtual targets is a "top-down" approach, attempting to alter the requirement to back-propagate the error, as a means of facilitating the algorithm in hardware. The motivation for weight perturbation is the same, but its approach is "bottom-up", eliminating some of the complex calculations of back propagation in favour of measuring changes at the network output.

2.5.1 Virtual targets

Murray developed the virtual targets algorithm as a way of adapting back-propagation to the constraints of silicon implementation. Its rationale and details are described in the following sections.

2.5.2 Overview of the algorithm

The virtual targets algorithm is one of a family of target-based algorithms, whose potential for VLSI implementation was initially identified in (Murray, 1991). The algorithm is, like the back-propagation algorithm, one in which the learning rule specifies how weights can be altered in the light of the network's response to repeated presentation of input patterns. The degree and direction of each weight-change is determined by, among other things, the size of the error between each output node's actual response to an input pattern and the target response that an external agency has dictated should be correct. The target-response for each output node is therefore determined by the nature of the classification problem being considered; for example, we know beforehand that, in response to a binary pattern representing the character 'B', one of the 26 output-nodes of the network (one node for every character in the alphabet) should signal **ON** while the remainder signal **OFF**. Unlike back-propagation, some simplifications to the algorithm are bought at the price of providing explicit targets for the hidden-layer nodes too. Although the algorithm specifies that these hidden-layer targets are

reset at various points, their values are not generally known, but evolve according to certain equations in much the same way that the weights do.

2.5.3 Precursors of the algorithm

The back-propagation algorithm changes only the weights during learning, and the outputs of the hidden layer are determined primarily by the input vectors and the weights (and some additional factors such as the learning rate or the 'temperature' or gain of the sigmoid function). By contrast, (Grossman et al., 1989) viewed a network's 'internal representations' (effectively the outputs of the hidden layer) as additional variables during learning, whose values would be determined by factors in addition to the input vectors and the weights. On Grossman's view, the problem is shifted from one of minimising an output error or cost function to one of searching for useful internal representations which will lead to a good solution. Krogh and Rohwer (Krogh et al., 1990, Rohwer, 1990) have also used this notion, and Krogh has formalised the idea by constructing a cost function for his algorithm which is an explicit function of these internal representations, and which is systematically reduced as the algorithm runs.

Murray synthesised these various approaches (Murray, 1992b, Murray, 1992a) by introducing an explicit desired or target state for each hidden node, updated continuously during learning. The details were developed pragmatically, not as a means of improving on back-propagation, nor to provide a mathematical analysis of target algorithms, but to facilitate on-chip learning in hardware.

The key points of the virtual targets algorithm are that :

1. the weight-change equations in the hidden layer and output layer are equivalent. They replicate back-propagation's equation for the output-layer weights (Equation 2.13), which is much simpler than back-propagation's weight-change equation for the hidden layer (Equation 2.19).

2. hidden-layer and output-layer nodes are rendered identical in function, at the expense of introducing hidden-layer targets for each input vector.

3. the problem of back-propagation of error is effectively altered to one of modifying the target states during learning.

The algorithm is, effectively, a different way of back-propagating the error. It is not mathematically equivalent (since it does not guarantee gradient descent), but is functionally equivalent in that it performs much like back propagation in classification tasks.

Murray showed (Murray, 1992b, Murray, 1992a) that the algorithm behaved as though two forces were at work and sometimes competing : one force being the movement of the hidden-layer weights to reduce the hidden-layer error; the other force being learning on the hidden-layer targets to reduce the network error.

Since the algorithm does not guarantee gradient descent, one would expect that the network error might sometimes increase (ie the algorithm would demonstrate 'hill climbing'), which Murray did indeed observe.

The crucial test of the algorithm, of course, was whether it would work. It succeeded on two standard MLP test problems, the parity task (can the network distinguish between binary vectors exhibiting odd or even parity?) and the encoder-decoder task (can the network encode an N-bit pattern into $log_2 N$ bits and then decode this representation?) although, like back-propagation, the algorithm sometimes became stuck in local minima. Furthermore, a comparison between back-propagation and virtual targets on a real-world task, recognising vowel sounds from a vowel database of male and female speakers, showed that the two algorithms had similar generalisation performance, while virtual targets had a learning-speed advantage (Murray, 1992b, Murray, 1992a).

In summary, then, virtual targets and back-propagation have broadly similar levels of performance, and are of the same order of complexity, but virtual targets removes the distinction between hidden-layer and output-layer nodes, and allows for the weight-change equations in each layer to use local information, at the expense of introducing explicit hidden-layer targets.

2.5.4 Details of the virtual targets algorithm

The full equations for the virtual targets, and for the back-propagation, algorithms can be compared in Table D–1 in Appendix D

From the point of view of VLSI, the virtual targets algorithm has two advantages over back-propagation. Firstly, the means of updating weights in a node

uses only information that is local to that node, simplifying the circuitry necessary for a hardware implementation; in back-propagation, information for updating the hidden-layer weights has to be passed back from the output layer. Secondly, the weight-update strategies for both hidden- and output-layer neurons are identical (whereas they are different for back-propagation), which means that, once neuron circuits have been designed, they can be replicated for the whole network, whichever layer they are in. The price to be paid is that, in addition to the output target-states (ie teaching patterns used in training), a target-state has to be introduced for each hidden-layer neuron, and these hidden targets require information to be fed back from the layer above.

The virtual targets algorithm simplifies the weight-update strategy sufficiently to make hardware implementation a more practical prospect than for back-propagation.

The algorithm for the training phase for an $I - J - K$ network is outlined in Figure 2–3.

2.5.5 Weight perturbation

Colleagues at Oxford chose to investigate weight perturbation algorithms. Their motivation (as with the study described in this thesis) was to facilitate the building of hardware MLP networks with on-chip learning.

Weight perturbation takes various forms (Cairns, 1995), but its simplest form (Jabri et al., 1993) is as follows. Instead of *calculating* the error / weight gradients $\partial E_p/\partial w_{kj}$, as does back-propagation, the technique *measures* them directly using a finite-difference approximation, which is computationally simple but nevertheless effective, if slow compared to back-propagation (Cairns, 1995). The procedure is as follows :

1. Apply an input vector to the network.

2. Measure the network error.

3. Perturb a weight by an amount $pert_{kj}$ and apply the input vector again.

4. Re-measure the error and calculate the change in the error.

1. Carry out a forward-pass calculation to produce hidden- and output-layer vectors :
   - Apply input pattern $\{O_{ip}\}$, and read out the states $\{O_{jp}\}$ and $\{O_{kp}\}$ of the hidden and output nodes.
2. Calculate initial values for the hidden-layer targets :
   - Assign targets $\{T_{jp}\}$ for the hidden nodes such that $\{T_{jp}\} = \{O_{jp}\}$.
3. Repeat 1 and 2 for all input patterns.
4. Present patterns in random order and allow :
   (a) weights to evolve according to the following equations :

$$\frac{\delta W_{kj}}{\delta t} = \eta_{weights} O_{jp} O'_{kp} \epsilon_{kp} \tag{2.20}$$

$$\frac{\delta W_{ji}}{\delta t} = \eta_{weights} O_{ip} O'_{jp} \epsilon_{jp} \tag{2.21}$$

   where
   - $\eta_{weights}$ is a gain-term representing weight learning-speed;
   - $\{O_{jp}\}$ and $\{O_{ip}\}$ are the inputs from the previous layer;
   - O'_{kp} and O'_{jp} represent the derivatives of the activation function (the 'sigmoid-prime' terms); and
   - ϵ_{kp} and ϵ_{jp} are the error-terms where $\epsilon_{kp} = T_{kp} - O_{kp}$ and $\epsilon_{jp} = T_{jp} - O_{jp}$.

   (b) hidden-layer targets to evolve according to the following equation :

$$\frac{\delta T_{jp}}{\delta t} = \eta_{targets} \sum_{k=0}^{K} W_{kj} \epsilon_{kp} \tag{2.22}$$

   where
   - $\eta_{targets}$ is a gain-term representing target learning-speed;
   - W_{kj} is a weight on the connections between the hidden- and output-layers; and
   - ϵ_{kp} is the error term where $\epsilon_{kp} = T_{kp} - O_{kp}$.

Figure 2–3: *The virtual targets algorithm.*

5. Repeat steps 3 and 4 for each weight in the network.

6. Update all the weights in the network.

7. Repeat steps 1 to 6 for all input vectors.

8. Repeat step 7 until the network has learned.

Hence :

$$\frac{\partial E_p}{\partial w_{kj}} = \frac{E_{diff}}{pert_{kj}} \tag{2.23}$$

$$= \frac{E_p(w_{kj} + pert_{kj}) - E_p(w_{kj})}{pert_{kj}} \tag{2.24}$$

and so the weight-change equation becomes :

$$\Delta w_{kj} = -\frac{\eta}{pert_{kj}} [E_p(w_{kj} + pert_{kj}) - E_p(w_{kj})] \tag{2.25}$$

The computational simplicity[3], then, comes from the fact that, to establish the amount by which a weight must be altered requires only a difference calculation, scaled by a factor $\eta/pert_{kj}$.

2.6 Implications for hardware

This Section examines explicitly the hardware implications of each of the three algorithms (back-propagation, virtual targets, and weight perturbation), considered earlier. The comparison is summarised in Table 2–1. The following subsections consider points where the comments in the table need amplification.

---

[3]The term 'computational simplicity' can be misleading. Although the computation specified here is superficially very simple (for example, on a digital machine), to carry it out in analogue hardware proves rather difficult (Cairns, 1995).

Feature	Back propagation	Virtual targets	Weight perturbation
Speed of training (in epochs)	Slightly slower than VT	Fastest	Not significantly slower than BP
Speed of presentation per epoch	About the same		Slow
Classification performance	About the same		
Computational complexity :			
Forward pass	About the same		
Backward pass	Conveying output-layer error to hidden-layer weights	Conveying output-layer error to hidden-layer targets	Conveying network error to hidden-layer weights
Synapses	Must be bi-directional		Can be uni-directional
Hidden-layer weights	Non-local information must be back-propagated	Only local information required	Non-local information must be back-propagated
Output-layer weights	Only local information required		
Greatest complexity	Calculating hidden-layer weight updates	Calculating hidden-layer target updates	Detecting and measuring small changes in network error
Equivalence of hidden- and output-layer nodes	Weight update very different	Nodes effectively the same	
Storage required	Input patterns Weights Output targets	Input patterns Weights Output targets Hidden-layer targets	Input patterns Weights Output targets Error / weight gradients
Possible limiting factors	Precision in forward pass Precision in weight update	Precision in forward pass Precision in weight update Precision in target update	Precision in forward pass Precision in weight update Detection of very small errors

Table 2–1: Comparison of the three algorithms

Speed

Since the motivation for the virtual-targets and weight-propagation algorithms is to facilitate hardware implementations of MLP networks, extensive comparisons of their simulation performance with back propagation are not available. However, we can say that :

1. virtual targets has a slight learning advantage over back-propagation on the speaker-identification task examined by (Murray, 1992b, Murray, 1992a), learning in around 500 training epochs[4] compared with around 1500 for back propagation. On a series of tasks (speaker-identification, classification of medical data and image-region classification), Cairns found weight perturbation and back propagation to have learned successfully within 2000 epochs.

2. the time taken for an epoch is inevitably slower for weight perturbation. For K training patterns, an epoch for back propagation and virtual targets will be in the order of :

$$T_{epoch} = (T_{forwardpass} + T_{weightupdate}) \times K$$

while, for the same problem using weight perturbation, a network with N weights will require :

$$T_{epoch} = (T_{forwardpass} + T_{weightupdate}) \times K \times (N + 1)$$

Classification performance

Classification performance is measured by training on a set of training vectors, and then measuring the network's ability to classify correctly on a second set of test training vectors which are chosen to represent, as far as possible, the whole set of vectors which the network will be required to classify. The results may

[4]A learning epoch is one in which all patterns from the training set have been presented once.

be presented in various ways, for example as the percentage of input vectors classified correctly (*generalisation performance*), or as the percentage incorrectly classified (*cross-validation classification error*). Performance of an algorithm is highly dependent on the nature of the classification the network is required to make.

Murray found the virtual-targets and back-propagation algorithms to be broadly comparable in their success rates in classifying unseen data (Murray, 1992b, Murray, 1992a) on the vowel-recognition task described earlier. Cairns found the classification properties of weight perturbation and back propagation to be similar on three classification tasks (speaker-identification, classification of medical data and image-region classification) (Cairns, 1995), provided the size of perturbation was sufficiently small[5].

Computational complexity

Although a broad comparison is made here, a definition of 'computational complexity' is not simple. Some computations which are very simple in digital simulation prove difficult in translation into analogue hardware (see footnote on page 21), while, as we see later in translating the virtual targets algorithm into hardware, the reverse is also true. A quantitative comparison could only be made with a detailed design of a system for each algorithm, but would consider :

- the degree of parallelism in the design;
- the number of calculations required;
- the complexity of the analogue circuits to carry out the calculations (eg number of transistors, space-consumption, power-consumption);
- whether circuits could be designed to be replicated many times or whether specialised circuits would be required;

[5]In the limit of small perturbations, as the change in weight approaches zero, the weight updates generated by weight perturbation are identical to those obtained by error back-propagation.

- signal-routing complexity within and between chips.

Trade offs between hardware complexity and algorithmic performance

From Table 2–1, it is clear that there is no easy alternative to back propagation as a means of implementing MLP networks in hardware. Nevertheless, virtual targets and weight perturbation have sufficient advantages to make detailed investigation of their merits worthwhile.

It appears, from simulation results, that the virtual-targets and weight-perturbation algorithms reduce complexity in particular areas without loss of classification performance, although the length of an epoch is increased using weight perturbation, due to the increased number of forward passes required. The price in hardware terms is that of increased storage (for both virtual targets and weight perturbation) and the detection of small output changes (for weight-perturbation).

2.7 Analogue hardware and the pulse-stream approach

Chapter 3 reviews, in detail, the use of analogue hardware for ANNs. This Section outlines the pulse-stream approach used in the project.

The pulse-stream approach is a hybrid of digital and analogue techniques in which data is pulse-encoded but computation is done in analogue form (Murray et al., 1991). An example of the technique is given in Appendix E. The advantages of analogue circuitry can be retained, but communication between modules or chips can be by binary pulses which encode data using, for example, pulse frequency or pulse width. In more detail :

- analogue computation can be compact, fast, asynchronous and free of quantisation effects, and so is preferable to digital computation.

- digital signals are robust and easily transmitted and regenerated, and so preferable to analogue signals as a means of communication.

These advantages have been verified over a number of years by our research group. As explained in Chapter 3, although I considered alternatives to pulse-stream techniques, I adopted them myself.

2.8 Sigmoid derivative

In this section, I consider the derivative of the sigmoid function, also known as the sigmoid prime, and its effect on the performance of the virtual targets algorithm. The term appears in the equations for updating weights in both the back propagation and virtual targets algorithms.

As part of an entirely pragmatic series of simplifications to the virtual targets algorithm, described in detail in Chapter 5, I removed the term, and made an estimate of the consequences in simulation. The criteria for success in these simplifications were the practical ones that the algorithm should still classify correctly, and that the algorithm should be easier to implement in hardware. I was able to meet these criteria, at the expense of longer learning speeds, ie the network took more epochs than previously to learn to discriminate different input vectors.

2.8.1 The sigmoid function and its derivative

The general form of the sigmoid function is $1/(1+e^{-x})$. A *threshold* term, θ, and a *temperature* term, T, are added to control the zero crossing point and the slope or gain of the function, so it is normally expressed as $1/\left(1+e^{-(x-\theta)T}\right)$. The shapes of sigmoid curves for different values of θ and T are shown in Figure 2–4.

As explained in Section 2.4.1, given an output which is a sigmoidal function of the activation, the derivative of the output with respect to the activation of that node is :

$$\sigma'(net_{pk}) = O_{pk}(1-O_{pk}) \tag{2.26}$$

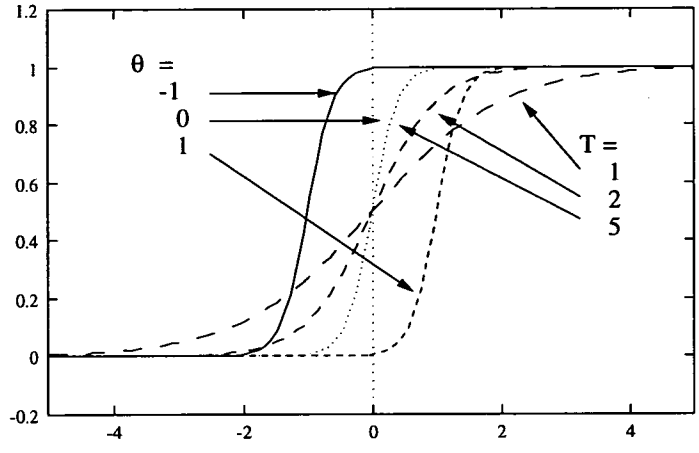


Figure 2–4: *Various sigmoidal curves*

and this function is illustrated in Figure 2–5

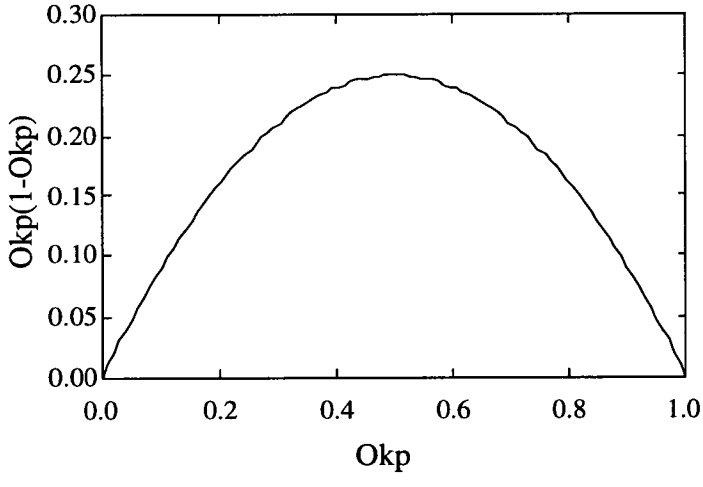


Figure 2–5: *Graph of the sigmoid prime.*

As shown in the table of learning equations in Appendix D, the term appears in the weight-update equations for hidden and output layers for both the back-propagation and virtual-targets algorithms. As errors are back-propagated to the hidden-layer weights, the effect of the error is scaled by multiplication with the sigmoid-prime term. The value of the sigmoid prime goes to zero as a unit's output O_p approaches 0.0 or 1.0 (which Fahlman called 'flat spots'), and never exceeds 0.25. Therefore even if the error on a unit which is almost fully **ON** or almost fully **OFF** is near the maximum, only a tiny fraction of the error will be back-propagated. The unit may then stick in its **ON** or **OFF** state, taking a

large number of epochs to change or, if the back-propagated error is so small as to be difficult to distinguish from zero, perhaps never recovering.

Fahlman identified this difficulty with back-propagation (Fahlman, 1988) in an investigation of how various network parameters might be 'tuned' to improve the characteristics, including learning speed, of the algorithm. He tried various alternatives to the use of the sigmoid prime, including :

- adding a constant to the term (so that its minimum value was 0.1 and its maximum 0.35), dramatically improving learning times;
- replacing the function with a constant, thus eliminating the use of the term altogether, which still reduced learning times, although not so dramatically;
- replacing the function with another which combined use of a constant with a random value in the range 0.0 to 0.5, which caused the network to behave in much the same way as the use of the constant alone.

Fahlman concluded :

> *The primary lesson from these experiments is that it is very useful to eliminate the flat spots by one means or another. ... A slight modification of the classic sigmoid-prime function [adding a constant step to the term] did the job best, but replacing this step with a constant ... reduces the learning speed by about 20%. This suggests that this general family of learning algorithms is very robust, and will give you decent result however you scale the error, as long as you don't change the sign or eliminate the error signal by letting the sigmoid-prime function go to zero.*

I concluded that this gave me empirical justification for removing the term, with results similar to Fahlman's, as is described in more detail in Chapter 5.

Fahlman himself noted that the success of his approach might be problem- and network-dependent. I cannot find in the literature that this point has been investigated, although Looney has confirmed Fahlman's original result (Looney, 1996), and there is evidence, for multi-hidden-layer networks, that amplifying the effect of the sigmoid-prime on hidden-layers more distant from the output layer can improve learning speed (Han and Moraga, 1995, Sarkar, 1995).

2.9 Conclusions

The back-propagation algorithm and its two derivatives, virtual targets and weight-perturbation, have similar classification abilities, although weight-perturbation requires a higher number of forward passes during each epoch, which extends its learning time in software simulation.

All three algorithms are complex to render into hardware, but virtual targets has some features which ease the complexity at the expense of extra storage, while weight perturbation alters the complexity from that of calculating weight changes to that of detecting small changes at the outputs, again at the expense of extra storage.

For the back-propagation algorithm, the sigmoid-prime term can be replaced by a constant without serious performance deficit. This empirical finding proves important in simplifying the virtual targets algorithm, as described in Chapter 5.

Chapter 3

Issues in on-chip learning

3.1 Introduction

I consider here the motivation for providing learning on-chip, and consider the respective merits of digital and analogue implementations. I look at the published literature, and classify the different implementations in terms of the ways in which they store weights and modify them, two key issues in this field. Finally, I describe some implementations in detail.

3.2 What is on-chip learning?

Designers can implement hardware neural networks, whether they are digital or analogue, in several ways. One way of thinking about these implementations, and so of classifying them, is the method they use to determine a set of weights (ie to 'learn') and to provide these weights on a chip.

3.2.1 Learning off-chip

The majority of implementations have external learning, that is learning that does not take place on the chip. A general-purpose, serial computer trains the network by generating a weight-set appropriate for the task in hand. This procedure al-

lows the designer to train the network and test it on test-data or cross-validation data. Although the serial nature of general-purpose computers prevents them from exploiting the speed of a parallel network, computers nevertheless provide the designer with the flexibility and design tools to allow him to check and re-check his work. Once the designer is satisfied that the network operates satisfactorily, the final weight-set can be down-loaded to the hardware, which is then used for the intended application. For networks that use digital hardware, the down-loading means storage of the weights in digital registers. For analogue chips, the weights must also be stored, and this can be done in purely analogue circuitry, using non-volatile storage techniques such as floating-gate storage or the much newer technique of amorphous-silicon resistors (Holmes et al., 1995, Holmes et al., 1993). However, even for analogue systems, the most common form of storage is digital registers, using digital-to-analogue conversion to provide weights in analogue form on the chip itself. The weights themselves are generally represented as charge on capacitors. Since charge is 'volatile', ie it leaks away, some means has to be found of refreshing the values of the weights.

3.2.2 Chip-in-the-loop learning

A second method of determining an appropriate set of weights is by chip-in-the-loop learning. This technique is applied exclusively to analogue chips as a means of compensating for variations in the performance of arithmetic functions of circuits located at different places on the chip. The designer carries out a training phase, on a serial computer, to generate a suitable weight-set, and down-loads the weight-set to the hardware network. Because of process variations on the chip, a computation, for example a multiplication of two variables, from a circuit at one place on the chip may give a different result from a copy of the circuit a few hundred microns distant. This can degrade network performance. However, if the supporting computer carries out a further training phase by applying inputs to, and reading outputs from, the hardware, while re-adjusting the weights, the hardware performance will rise. The re-adjustment of weights compensates for within-chip variations. Intel's analogue ETANN chip (Tam et al., 1990) has used the 'chip-in-the-loop' scheme, as has a back-propagation algorithm on a mixed optical and analogue-electronics network (Frye et al., 1991). Our own group has used this technique for the EPSILON chip as a means of implementing a vari-

ant of the back-propagation algorithm (Churcher et al., 1993), to be described in Chapter 4.

3.2.3 Learning on-chip

A third method of implementation is by what Card and Schneider call *in situ* learning (Card and Schneider, 1992), that is the learning mechanism is located on the chip itself, and hence the chip carries out the training of the network to produce a suitable weight-set, without any need for serial-computer simulation. As we shall see, automatic adjustment of analogue weights on a chip is not an easy objective to achieve.

3.3 The advantages of on-chip learning

On-chip learning offers the following advantages over external and chip-in-the-loop learning :

- **Speed.** Clearly, an ANN algorithm that runs on parallel hardware will run faster than the same algorithm running on a serial computer; that is, after all, what the algorithms are designed to do. If the hardware can accomplish the learning phase too, so much the better. However, the issue of speed is not a simple one. A circuit that can adjust an on-chip weight must be replicated at every site to be truly parallel. This may make heavy demands in terms of space, and therefore there may be trade-offs between space-saving and complexity : the more parallel the implementation, and hence the more space required, the more simple, and hence the less sophisticated and slower, the circuitry involved. Furthermore, the network can only realise any speed-enhancements in practice if the system can present data to, and read it from, the chip in a way which does not cause bottlenecks. We cannot, in other words, separate questions of the speed of learning on-chip from the nature of the circuitry on the chip and the speed of the system as a whole.

- **Autonomous learning.** A major advantage of neural networks is that they do not need programming but, provided they are carefully designed,

can be trained from examples to perform a task. If the chip can perform the training phase, then it is conceivable we could design autonomous systems that can learn and re-learn in real time. As with the issue of speed, autonomous learning is not a simple matter to achieve. Currently, we do not have algorithms that are applicable to a wide range of problems without fundamental changes to the network. For example, a network running the back-propagation algorithm, of which the algorithm described in this thesis is a variant, needs careful design to give good results. Once we train a network for one task, say pattern-classification, we may have to re-train it for a new task and also change the number of inputs and neurons. Nevertheless, the ability to learn autonomously does suggest potential benefits.

- **Compensation for analogue variations.** Cells at different places in an array of analogue circuits will display different characteristics due to process variations across the chip, a factor that can be accommodated by using chip-in-the-loop learning. On-chip learning offers this same advantage, but can also compensate for differences in the circuitry that holds the weights and adjusts them.

- **Adaptation to constantly-changing environments.** Just as with process variations, on-chip learning can compensate for changes in the environment, such as the temperature of the chip or in response to use or changes in the surroundings. For example, when we drive a car, our performance varies from day to day, and varies even in the course of a single journey. Weather or lighting conditions can change, as may the surface of the road, the density of traffic, or our own abilities and reactions as we become fatigued through effort or more alert after passing the scene of a recent accident. The whole environment – the outside world, the car and our own bodies – changes. A neural network that is in a perpetual state of learning could be capable of responding to a whole kaleidoscope of complex relationships. This is by no means a new idea, since learning systems have existed for many years in the guise of adaptive control systems, for example for telephone echo cancellation or linear-predictive coding of speech. Neural networks could support a rich range of inputs and outputs in environments that are changing constantly.

- **Compensation for charge-leakage.** If we use charge on capacitors to store analogue weights, the weight circuitry can be compact and simple, but

the weights must be refreshed in value from time to time to neutralise the leakage of charge from the capacitors. As an alternative to weight-refresh we can re-run the learning phase at intervals, until suitable weights are re-learned.

3.4 The use of analogue, rather than digital, hardware

Before looking at the reasons why one might choose analogue rather than digital hardware, I want to suggest that digital hardware offers such advantages that, at least at present, it is likely to be the prime choice. Digital implementations are not restricted in the algorithms they can instantiate, but can be reasonably easily re-configured to reflect the differences between a range of algorithms. Digital arithmetic is not susceptible to process variations and so is highly accurate; that is, an arithmetic result can be produced repeatedly wherever the function is located on a chip, between chips of a different kind, and even with structures of a different kind. Precision is not infinite, but it can be high, as is dynamic range, both these factors being dependent on the number of bits chosen; at least with current algorithms, precision is not a problem (although the space occupied by a large number of bits is). The digital-design process is well-understood, and is amenable to automation or, at worst, to algorithmic approaches that can reduce the amount of trial-and-error steps required and the likelihood of error. Chip-manufacturing processes are, generally speaking, designed with digital circuits in mind, as are the models of transistors that manufacturers produce.

By contrast, analogue hardware has many problems :

- Usually we have to commit analogue hardware to a particular algorithm, since each algorithm requires different calculations that must be realised with different circuits, each of which must be carefully designed before manufacture.

- It is difficult to design analogue circuits that work well; each designer has to expend many hours of effort developing a personal methodology and a 'feel' for circuits, and the way they work, that is difficult to automate, or even to explain. We may have to design circuits to accommodate the vagaries of

standard chip-manufacturing processes, and we must interpret the results of simulations using knowledge that only experience can provide.

- Analogue arithmetic is highly susceptible to process variations and so is inaccurate to a degree that is difficult to predict, both across a single chip and between chips.

- It is difficult to achieve a high dynamic range because of noise due to electromagnetic pickup and switching.

- The analogue voltages and currents that represent the different variables are subject to offsets.

- Weights represented as charge on a capacitor have to be refreshed periodically.

Despite these problems, analogue approaches are held to have certain advantages that I now want to examine.

- *Analogue implementations are more compact than digital ones.* This advantage, if genuine, is certainly a good reason for building analogue neural networks. The main constituent of any artificial network is the array of synapses, that is weight-storage and multipliers. Synapses on an analogue chip can take up much less space than their digital equivalents. However, as with most such assertions, the advantage is not overwhelming; by the time we take into account the requirements of off-chip refresh circuitry, we may have simply shifted the problem elsewhere. This is not to say that analogue designs cannot show an advantage over digital ones, only that a meaningful comparison is rarely straightforward.

  In any case, I am not convinced that compactness is yet a major issue in the design of neural hardware. Clearly, there is an advantage for any engineered system in minimising costs, and silicon-usage is one of those costs. Because biological systems, in particular the human brain, are held to be massively parallel in nature, there seems to be an assumption that artificial networks will benefit from a concomitant massive parallelism, even though the biggest software implementations currently use only a few hundred neurons. We can draw an analogy here between the current state of neural network re-

search and research into the use of parallel processors a decade ago. At that time, exaggerated predictions were being made about the impact of huge numbers of identical parallel processors that would render serial computing redundant. In practice, of course, the design of algorithms to exploit parallelism proved rather difficult, unpredicted bottlenecks appeared, and progress in the use of parallel processors in reality took a much more stately course.

- *Analogue circuitry is infinitely precise.* It is true that limited resolution is not a feature of analogue hardware (although see the footnote on page 116), as it is with digital hardware, but the problems of noise and inaccuracy make this advantage difficult to exploit.

- *Analogue hardware can consume very little power.* This is an undoubted advantage of analogue designs; it is possible to operate transistors subthreshold and still achieve very useful results (Mead, 1989), but there are, so far, few applications that make this an important virtue (Jabri et al., 1993).

The greatest virtues of neural network hardware — namely that once designed it needs little or no programming and, because of redundancy, reduced testing — apply to digital and analogue implementations alike. My conclusions are that the advantages of analogue hardware are still questionable, although they may emerge more clearly as research into algorithms matures and as applications become more diverse.

3.5 Research into hardware parallel architectures

The essence of neural networks is their parallelism, and we can only exploit this on a parallel architecture. Parallel software and hardware architectures both suffice. VLSI implementations offer a further advantage that many hundreds or even thousands of neurons can be created in a small space, with a consequent massive parallelism and resistance to error because of redundancy. Analogue implementations offer the prospect of bigger networks in a smaller space, although it seems we do not yet know the best way of exploiting this advantage.

From the preceding discussion, it is clear that these benefits are bought at the price of producing networks that are difficult to design and inflexible to use. This means that, for the future, we need to be clear about both the application we have in mind and the means of achieving it before an analogue network becomes the network of choice.

There are, nevertheless, two good reasons why such research is necessary. One is that costs (of space, power, design-time and so on) are always important, and hence occasions when compact, low-power circuitry will be required are bound, sooner or later, to arise, for example in the case of autonomous vehicles. The other reason is that digital simulation can never tell us enough about the performance of real-world analogue circuits. This is partly because simulating a circuit and building one are two entirely different tasks. It is also because some issues (for example, the question of stability in a network with hidden-layers) is difficult or impossible to investigate in simulation. The only sure way is to build the circuit and try it.

3.6 Strategies in research on-chip learning

We could consider approaches to ANNs, and hence to on-chip learning, as falling somewhere between two extremes.

At one extreme are the approaches that concentrate on the mathematical properties of ANNs as a means of solving real-world commercial or engineering problems, such as economic forecasting or face-recognition. Although claiming inspiration from biology, the link between ANNs and biological networks is tenuous. Within this approach, on-chip learning is seen as useful because it is the obvious next step for implementing algorithms, such as back-propagation, in their entirety on a chip, or because the technique would aid performance in rapidly-changing environments, such as a robot might encounter. Publications on on-chip learning lie overwhelmingly toward this end of the spectrum, and the technical work described in this thesis is firmly rooted in this tradition.

At the other extreme are approaches that try in some way to emulate biological mechanisms. The justification for on-chip learning in such cases are that it is an integral part of the system. I include it here, albeit briefly, because the 'biological'

approach represents some of the best work in VLSI design, and because I am intellectually attracted to it, despite my own work being from the other tradition.

Reasons advanced by researchers for choosing a particular strategy for on-chip learning vary. For example, the approach might :

- owe, in some degree, its inspiration to neurobiology.
- owe little to neurobiology, but demonstrates some interesting properties.
- use a popular algorithm.
- use an algorithm that gives a prospect of real-world applications.
- facilitate implementation in hardware.
- demonstrate a hardware feature in which the research group specialises
- allow large, parallel networks to be implemented.
- make the best use of the materials available.

Some of these reasons are stated explicitly in publications, or during presentations at conferences, while some are implicit and never overtly stated.

The techniques also greatly vary. For example, some researchers concentrate on inventing circuits to perform particular functions such as modifying weights. Others build complete systems. Yet others concentrate on the conditions under which learning could best operate.

I have chosen to illustrate the diversity of the field by choosing these examples :

- emulating a biological function
- implementing a model of a behavioural phenomenon, but without reference to the underlying biology, and with the emphasis on taking account, from the start, of the limitations of analogue hardware
- implementing a complete back-propagation system with non-volatile weights

- investigating the efficacy of a variation of back-propagation that simplifies implementation in hardware

3.6.1 Emulating a biological function

This kind of approach has a good pedigree in that its leading figure is Carver Mead, renowned for developing VLSI solutions to neural problems. He was very explicit in his original aims. These were to understand biological systems (because of their power), to model them in silicon, and to apply them to problems that even the largest digital computers found intractable. His 'neuromorphic' approach identifies different structural levels in the nervous system (Faggin and Mead, 1990). At the lowest, it develops silicon analogues of the computational primitives of the nervous system, using the physics of semiconductors (for example, the exponential dependence on gate-voltage of the drain current of a transistor operating in the sub-threshold region). At the next level, it attempts to organise these primitives to perform complex computational tasks, such as sensory pre-processing or 'learning'. At the top level, it develops an architecture capable of solving a practical problem, such as character recognition. In circuit terms, Mead has tried to make his designs adaptive, to compensate for inter-device and inter-circuit variations and to make the architectures reconfigurable. His hope was that his designs would achieve the robustness and fault-tolerance of natural systems.

Mead continues to apply his expertise to real-world problems through his commercial activities (McDonald,, 1992). Although some of the shine has been rubbed off his original notion, I believe it is still valid, as I explain in my conclusions to this thesis. For example, Mead and his group have tried to model auditory processing, (Lyon and Mead, 1989, Lazzaro and Mead, 1990, Lazzaro and Wawrzynek, 1993), as have others (Liu et al., 1992, Rosen et al., 1994).

An interesting variant on this work is the implementation of a model of olfactory processing (dealing with the sense of smell) (Shoemaker et al., 1992). The motivation for the implementation was twofold : that the area of the olfactory bulb in question proved to have interesting clustering properties that might be replicated in VLSI; and that the work might elucidate the computational principles of real nervous systems.

Learning in the system was modelled on long-term potentiation (referred to on page 144). The scheme circumvented some of the problems of learning in ANNs such as back-propagation networks (to be discussed later in the thesis), in that weights were incremented in fixed increments of 5% to 10% of the range, over a range of only two or three times the value of naive weights.

The investigators took a pragmatic approach to modelling the system, in that they followed biological principles closely on some VLSI modules in their system, but were expedient in their use of circuits in producing other functions. Theirs is a good example of clever circuits being used on an interesting problem, to aid understanding rather than offer real-world applications.

Another investigator who has applied analogue VLSI ideas to this area is Elias (Elias, 1993), who uses fairly simple models of very-low-level neural structures, namely the dendrite (one of the structures of nerve cells to which synapses connect). His work is interesting on two counts. The first is that his circuits can realise temporal-encoding, a well-known feature of real neurons; he stimulates chains of simple *RC* circuits that emit different responses depending on the physical distance of the stimulus from the output. The second matter of interest is that he seems able to build useful feature detectors that can respond to, for example, lines moving in particular directions.

3.6.2 Implementing a model of a behavioural phenomenon

A colleague, Torsten Lehmann, takes an approach that combines biological inspiration (the Klopf algorithm) with an understanding of the limitations of analogue hardware, based on his experience in trying to implement the back-propagation algorithm (Lehmann, 1995).

Klopf's so-called *drive-reinforcement* model (Klopf, 1988) takes the high-level, psychological ideas of drives and reinforcements and applies them to individual neurons : the *drives* become sufficiently strong signals and the *reinforcements* become changes in the signal levels. This approach to associative learning, which is derived directly from animal learning, is able to simulate a wide range of classical-

conditioning data (Levine, 1991). It is, essentially, a differential Hebbian model.[1] There are two additional complexities : in order to account for the observation that in classical conditioning there is an optimal interval between stimuli, the change in post-synaptic activity is delayed in time; and the change in synaptic efficacy is proportional to present efficacy, because in animal learning there is an initial, S-shaped acceleration in learning. The mathematical interpretation of the Klopf rule is shown in Figure 3–1.

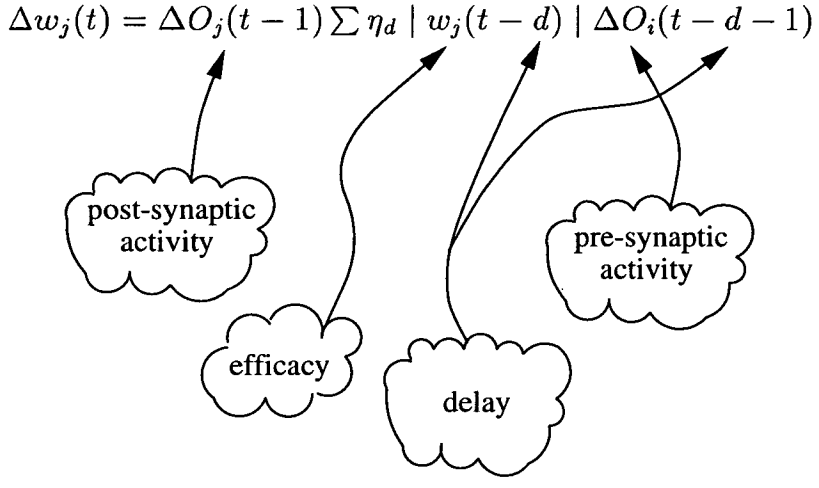


Figure 3–1: *Klopf's weight-change rule.*

Lehmann's medium of choice is analogue, pulse-stream, VLSI, neural networks, and he takes the inherent offsets and imprecision of analogue electronics into account from the beginning (Lehmann, 1995). He argues that an unsupervised learning algorithm such as Klopf's is not only biologically plausible but more likely to provide an efficient implementation in, for example, robotics.

Whereas the pulse-stream network described in this thesis uses pulse-width modulation, Lehmann favours pulse-frequency modulation, and he has designed novel circuits including a synapse, current-mode and charge-mode weight-change modules, and bandpass filters, needed because the implementation is a free-running, asynchronous one.

---

[1]Plain Hebbian models cross-correlate pre- and post-synaptic activities, while differential Hebbian models cross-correlate *changes* in pre- and post-synaptic activities.

The significance of Lehmann's conclusions lies in the fact that he had already designed a chip set (Lehmann, 1994) that could, in principle, instantiate a range of algorithms, including the back-propagation algorithm, as well as learn on-chip. He concluded from this work (Lehmann, 1993) that learning in gradient-descent algorithms was severely affected by a range of problems, including offset errors on signals and the difficulty of calculating accurately the derivative signal, and it was this understanding that led him to a different approach.

3.6.3 Implementing a complete back-propagation system with non-volatile weights

A Norwegian group (Berg et al., 1996, Sigvartsen, 1994, Soelberg et al., 1994) have attempted to implement a complete back-propagation network with on-chip learning. Their system uses : a slightly-modified version of the algorithm; continuous-time, analogue circuits; and analogue, non-volatile storage in the form of floating-gate memories. This is an unusual combination of techniques, made more unusual in that the means of learning, that is modifying the floating-gate weights, is a UV-light source. To control the size of any increment or decrement of the weights, the intensity of the light is varied.

In explaining their approach, the group make the usual token acknowledgement to biological systems, but emphasise the technical prowess of the hardware, namely that : the feedback of signals during learning can compensate for offsets due to transistor mismatch; power-consumption is very low since they use transistors in their sub-threshold regions; and the network, if large, would show fault-tolerance (although the network they have built is so small this advantage does not apply in practice). The group have devoted their time, not to the design of special circuits (since the circuit ideas are largely borrowed), but to building and testing a complete system.

The group's design is a considerable technical feat, firstly because they have had to translate an algorithm that is discrete-time and digital into continuous-time, analogue circuits; and secondly because they have successfully combined the electronics with the UV-light sources necessary to modify the rather cumbersome floating-gate storage.

Their greatest achievement is that they have a machine with which they can

investigate, in a complete analogue system, phenomena such as stability and the effects of noise, about which there is much theorising but little practical knowledge.

3.6.4 Combining hardware design with simulation

My colleagues at Oxford have made an analysis of a technique called weight-perturbation (Cairns, 1995), that avoids the need for calculating, during the learning phase, the direction the weights should be adjusted. (The nature of the algorithm is described in Section 2.5.5.)

The Oxford group's strategy for assessing the utility of hardware was in contrast to my own attempt to translate as much of the virtual targets algorithm as possible into hardware. They designed one chip to perform the forward pass of a MLP and used it with chip-in-the-loop training to compare error back-propagation and different weight-perturbation techniques. They then used the data they had accumulated in software simulations of real-world tasks (speaker identification, medical data analysis and region classification) to make an assessment of the precision with which weights must be updated for on-chip learning to be achieved. There is much to be said for this alternative approach in that conclusions can be drawn with minimum effort invested in actually building hardware.

3.7 Implementations of on-chip learning

When considering implementations of on-chip learning, we should be aware that modifiable weights are a reasonably recent innovation in analogue VLSI. Historically, hardware neural networks have progressed from architectures where the weights are fixed according to pre-calculated values, through systems using programmable weights (currently the most popular choice), to adaptive systems. The published work that I consider in detail here is only concerned with the last of these, namely adaptive systems

The normal course of events in turning an algorithm simulated on a serial computer into a hardware implementation is to build modules that will perform the different basic functions, replicate the modules several (perhaps many) times, and

combine them into a parallel system. In looking at a list of hardware implementations, we can expect to see many different ways of building the basic modules. However, the features that are of most interest in a study of on-chip learning are :

- choice of algorithm
- the method by which weights are stored
- the means by which weights are changed
- the design of the synapse
- the design of the neuron

Table C–1 in Appendix C, lists published work on digital and analogue implementations of on-chip learning or (in a few cases) work that is of particular interest in a study of on-chip learning. It is obvious from the list that there is an enormous variety of approaches. Some work has particular applications in mind, but many are concerned only with investigating particular circuits. What is not obvious from the list is that certain issues are of key importance and I turn now to these.

3.8 Key issues : weight-storage and weight-modification

The questions of weight-storage and weight-modification are particularly difficult ones (Eberhardt et al., 1992), and it will help to put the work of this study into context to examine the work of other groups on these issues.

3.8.1 Weight storage

As indicated at the start of the chapter, weight-storage can take these forms :

- **fixed weights** Weights are calculated in simulation and then their values fixed or 'hard-wired' in some way. Such an approach, though not very common now, is described in (Mead, 1989).

- **digital weights** Weights are stored externally in digital form, and so are accurately known, with a precision dependent on the number of bits represented. As with fixed-weights, the values in store can be maintained indefinitely.

- **mixed digital and analogue weights** The weights are stored digitally, but converted to analogue values in a primary store in the form of charge on a capacitor. The primary store may require regular refreshment from the digital store if the values are not to decay through charge-leakage.

- **truly analogue weights** Floating-gate structures store analogue values in a non-volatile manner, so no external refresh is necessary. Amorphous-silicon technology shows some promise, but is at an early stage of development. Although both these technologies are non-volatile, reprogramming the weights is much slower than is reprogramming capacitive weights.

To highlight the different approaches, Table 3–1 classifies each of the approaches in Table C–1 in terms of the method of primary and secondary storage the researchers have adopted. From the table we can easily infer the following points.

- Some approaches to on-chip learning are entirely digital.

- Of those which use an analogue primary store with digital secondary storage, all represent weights as charge on a capacitor; sometimes that capacitor is explicitly provided, sometimes it is the gate-capacitance of a transistor that in turn controls a current.

- A number of researchers have taken advantage of the fact that, if the network is learning continuously, and the speed at which learning is taking place is sufficiently fast, refresh is unnecessary : the weights' constant adjustment in time itself acts as a refresh. (One researcher (Montalvo et al., 1994b, Montalvo et al., 1994a) proposes capacitive storage in the learning phase and floating-gate storage once the appropriate weights have been calculated and so is included twice in Table 3–1.)

Purely digital implementations	Hybrid analogue-digital implementations using capacitive storage with digital backup	Purely analogue implementations			Included for other reasons
		Capacitive storage with refresh-by-learning	Capacitive storage with refresh by read-store-and-write	Floating-gate storage	
Duranton and Sirat, 1990	Choi and Salam, 1993	Arima *et al*, 1991a, 1991b, 1992	Linnares-Barranco *et al*, 1993	Abusland and Lande, 1994 / Berg *et al*, 1996	Alspector *et al*, 1989, 1992
Eguchi *et al*, 1991	Cohen and Andreou, 1992	Dolenko and Card, 1993a, 1993b, 1995		Kim *et al*, 1992	Botros and Abdul-Aziz, 1993
Hammerstrom, 1990	Ghosh *et al*, 1994	Donald and Akers, 1993		Meador *et al*, 1991	El-Masry *et al*, 1992
Myers *et al*, 1992	Lehmann, 1994	Ibrahim and Zaghloul, 1990		(Montalvo *et al*, 1992, 1994a, 1994b)	Frye *et al*, 1991
Salam and Wang, 1991	Macq *et al*, 1992	Montalvo *et al*, 1992, 1994a, 1994b		Shibata and Ohmi, 1995	Hollis and Paulos, 1994
Shima *et al*, 1992	Schwartz *et al*, 1989a, 1989b	Schneider and Card, 1991a, 1991b			Säckinger *et al*, 1992
Theeten *et al*, 1990	Wang, 1993a, 1993b	Watola and Meador, 1992			van Daalen *et al*, 1994
Tomberg and Kaski, 1991					

Table 3–1: *Classification of published work in terms of method of primary and secondary storage*

- The division between refresh-by-learning and digital back-up probably reflects the confusion within the community over the best use to which adaptive systems can be put. There is no clear candidate for an application that requires a fast learning phase and then fixed weights thereafter; nor for that matter one that requires continuous weight-adjustment, and so no back-up.

3.8.2 Weight-modification

Each of the forms of weight-storage has advantages and disadvantages when we consider the issue of weight-modification :

- **fixed weights** The question of weight-modification does not, of course, arise

- **digital weights in digital and hybrid implementations** Digital storage is well understood, easily-controllable, resistant to decay in value, and easy to increment or decrement. This accounts for its relative popularity in the

list of implementations in Tables C–1 and 3–1. Set against these advantages, digital storage takes up a lot of space on a silicon chip. Some algorithms, such as back-propagation, are held to require a precision of 12 bits or more, which makes space a serious constraint. One solution is to keep the storage off-chip (incremented or decremented by an on-chip learning signal) and refresh the charge on small capacitors on-chip.

- **truly analogue weights** Refresh-by-learning is a truly adaptive method, although yet to be proved in an application. Some researchers (Castello et al., 1991) see floating-gate solutions as the best combination of silicon-area and storage-capability; others think small weight-increments on floating-gates prove too difficult (Schwartz et al., 1989b). In addition to the speed disadvantage noted in section 3.8.1, floating-gate storage is a technology that is not as easily available to researchers (including my group) as are other technologies capable of implementing digital storage or mixed digital/analogue storage. This, and the fact that knowledge about its design is not as common compared to other designs, probably explains its infrequency in the published literature.

3.9 Examples of weight-storage and weight-modification

In this section I look at an example of each of the categories of weight-storage listed in Table 3–1, and explain how weights are modified.

3.9.1 Example 1 : purely digital storage

Alspector's group designed a Boltzmann machine, a network in which the neurons have binary values; the network learns by optimising energy-states (a technique known as 'simulated annealing'), rather than by the delta-rule (Maren et al., 1990). The algorithm, which derives from statistical dynamics, is usually described as analogous to a physical solid that slowly cools, moving from a high-energy to a low-energy state. In network terms, as the 'temperature' (replicated by a noise source) reduces, the proportion of neurons in a high-energy state

reduces also, but the rule for updating the weights connecting units is probabilistic, so that sometimes the network jumps to a higher-energy state, allowing it the chance to escape local minima in the energy surface (Beale and Jackson, 1990).

The modified Boltzmann machine (Alspector, 1989, Alspector et al., 1989) uses a neuron that can produce both a step-function (as required by the original algorithm) and a non-linear function such as $tanh$[2], with a noise amplifier that can add noise to (or 'increase the temperature of') the neuron output. The chip's synapse is shown as the block-diagram of Figure 3–2[3]. Five flip-flops comprise the digital weights (4 bits and sign). The synapse achieves multiplication as follows. A zero-weight disconnects the synaptic connection. A non-zero weight is converted to an analogue conductance by a set of pass-transistors with graduated, binary-conductance ratios. As the weight changes, the logic selects the appropriate combination of pass-transistors, to give a combined conductance that increases monotonically in steps from -15 to +15 with increasing weight.

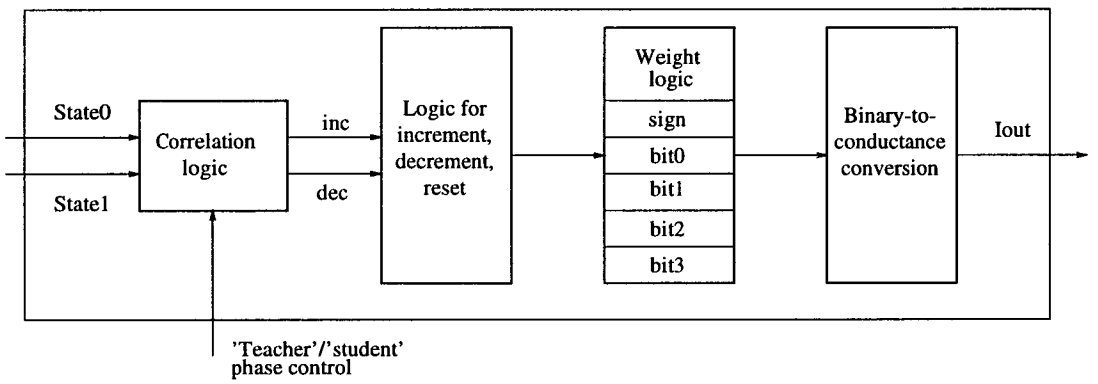


Figure 3–2: *Block-diagram of the synapse on Alspector's Boltzmann chip*

Learning is in two phases : in the 'teacher' phase, the inputs are 'clamped' with an input-pattern, the outputs with a desired-state pattern; in the 'student' state the output neurons are unclamped and run free. If, over these two phases, two interconnected neurons are correlated (ie have the same binary state), then

---

[2]The *tanh* function is equivalent to the sigmoid function, except that output values can vary between -1 and 1.

[3]Diagram adapted from Alspector's own figures.

the correlation-logic increments the synapse-weight between them; otherwise it decrements the weight.

Alspector produced a deterministic version of the Boltzmann machine that uses mean-field learning (Alspector et al., 1992). Conceptually, in deterministic machines a neuron produces a real number (rather than a binary output), representing the probability of a unit being in an **ON** state, ie the units' outputs represent the output-probabilities directly which, at least in simulation, means much faster learning. In this version of the machine, Alspector was able to vary the gain of the neuron with temperature, to sharpen the output from a *tanh* function to a step-function. Although weight-storage is the same as in the earlier version, the neuron output is a voltage, so the synapse now implements a $weight \times voltage$ multiplication to produce an output current.

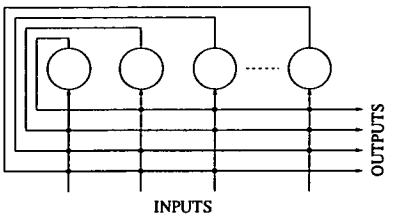
Alspector's solution to weight-storage exploits the ease with which digital logic can increment or decrement a weight, but his circuits are still able to offer analogue multiplication, and analogue outputs from neurons.

3.9.2 Example 2 : a hybrid implementation

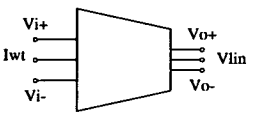
Ghosh's group has implemented a Hopfield network (Ghosh et al., 1994a, Ghosh et al., 1994b). The network is single-layered and fully connected. Each neuron connects to every other neuron, but not to itself, through weighted connections, and the connections between any two nodes are symmetrical. One way to depict such a network is in a crossbar configuration, as shown in Figure 3–3(a).

The Hopfield network is auto-associative : given a pattern as input, it will regenerate it, even if the pattern is noisy or incomplete. To recall a stored pattern, an input is applied, and the output of each neuron feeds back to all the other neurons until the network eventually settles into a state of equilibrium, when it produces the correct output. Learning a pattern does not involve repeated cycles of applying a pattern, reading the outputs, and adjusting the weights, as is the case with back-propagation. Rather, the weights are calculated in a single step, directly from the patterns that the network must learn. The algorithm, which is a form of Hebbian learning, involves multiplying the input vector by its transpose to produce a matrix, that is then added to the weight-matrix to store the pattern.

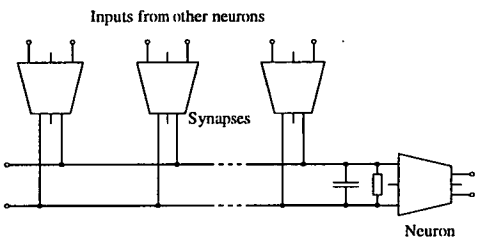
Ghosh's group has built on earlier work where weights are resistors and neurons



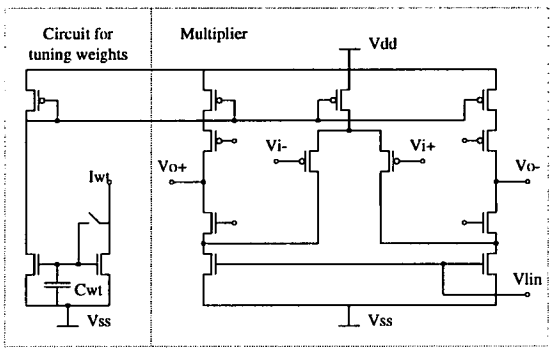
(a) Crossbar configuration for a Hopfield network



(c) Ghosh's symbol for his OTA synapse and neuron



(c) OTAs connected as a network



(d) Schematic of the synapse and tuning circuit

Figure 3–3: *The design of Ghosh* et al *for elements of a Hopfield network*

are op-amps that sum weighted currents. Problems associated with early work include : the need to have two neurons, one for excitatory connections, and the other with an inverted output for inhibitory connections (since resistors cannot take negative values); translating a pre-calculated weight-matrix into resistor values, while taking account of parallel resistances and the output impedances of op-amps; and the large size and inconsistency of silicon resistors. This more recent work overcomes these problems. Moreover, it provides a means of tuning the synapses, which is of course impossible with fixed resistances. It does so by using operational transconductance amplifiers (OTAs) as synapses and neurons (see Figure 3–3(b)[4]).

The OTA at each synapse provides four-quadrant multiplication (so furnishing excitatory and inhibitory weights), and supplies a differential output current to the neuron. At the neuron, an OTA configured as an integrator sums the synapse currents, as shown in Figure 3–3(c).

[4]Diagrams adapted from Gosh's own figures

As a multiplier, the OTA's output current is proportional to the product of the differential input voltage, derived from the neuron outputs, and the transconductance of the differential stage (see Figure 3–3(d)). This in turn is set by the bias current of the tail transistor, derived from the weight-capacitor. The weight is stored in an on-chip SRAM. When addressing circuitry isolates a particular synapse, the switch closes and a current DAC drives a current proportional to the weight through transistor M_{wt}, setting its gate-voltage. When the switch opens, the capacitor C_{wt} maintains the weight-current (although it must be refreshed periodically). The mirror transistors copy this current to the tail transistor, and enable the correct multiplication. (V_{lin} controls the linearity of the transconductance.)

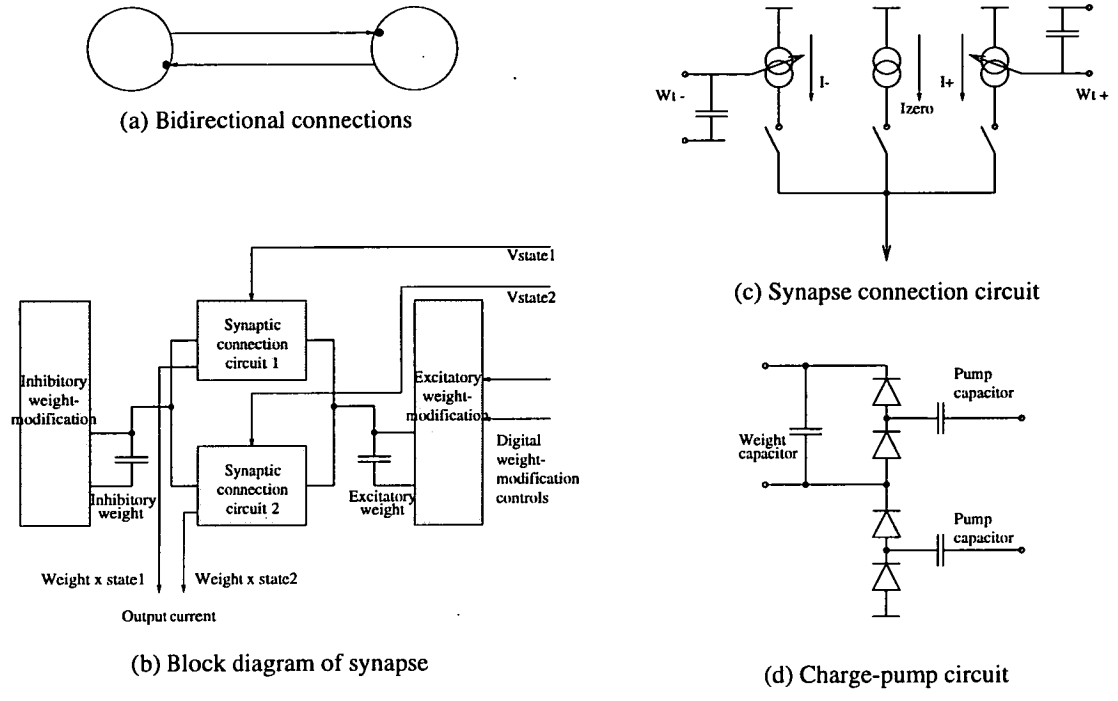
A network constructed out of these building blocks, but only tested in simulation, seems capable of recalling stored bipolar patterns, even if some of the input patterns are corrupted. The group has not yet provided a way of calculating, on-chip, the correct weights for particular sets of patterns, but has found a way of adjusting the weights for different sets.

3.9.3 Example 3 : capacitive storage with refresh-by-learning

The various learning chips produced by Arima's group (Arima et al., 1991a, Arima et al., 1991b, Arima et al., 1992), each bigger than the last, were among the first to demonstrate on-chip learning. In many ways, they were ahead of their time, achieving goals that other groups, including our own, are still trying to emulate. Like Alspector's, the chips ran a form of Boltzmann, mean-field algorithm, with the consequence that their neuronal outputs were bistate. The chips originally came in sets of two, one bearing the synapses being cascaded with the other bearing the neurons. Ultimately, synapses and neurons appeared on the same chip.

The synapse design was essentially the same over the various designs, and is shown in Figure 3–4[5].

[5]Diagrams adapted from Arima's own figures



(a) Bidirectional connections

(c) Synapse connection circuit

(b) Block diagram of synapse

(d) Charge-pump circuit

Figure 3–4: *Circuits on the Boltzmann chip of Arima* et al

Each synapse is bidirectional and symmetrical, as shown in Figure 3–4(a), on the assumption that $W_{ij} = W_{ji}$. A synapse comprises two 'synaptic connection circuits' (one for each direction) and two capacitive stores, one to represent excitatory or positive weights, the other inhibitory or negative weights (Figure 3–4(b)). Each synaptic connection circuit (Figure 3–4(c)) is, in effect, a set of switched current-sources, one of fixed value to represent a zero weight, and the other two being controlled by the voltage on the weight-capacitors. The states of the interconnecting neurons control the switches; a 'firing' (ie **ON**) neuron causes the synapse to supply weighted currents to the next stage, otherwise it supplies the zero-current. In the neuron, a comparator reads the summed currents and flips if the value is over threshold.

I considered this design a possibility for my own system. Charge-pumps, again controlled by switches (Figure 3–4(c)), bump the weight-capacitors up or down according to the learning rule. The capacitors are very small ($0.5pF$), even by VLSI standards, so only a 10% change was possible, but this seems to have been enough for the network to associate correctly 98% of input patterns with the correct outputs, provided various Hamming-distance constraints were observed.

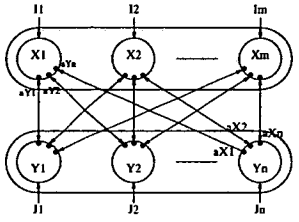
Arima's group saw charge-leakage, and hence 'forgetting', as a problem (Arima et al., 1991b), even if learning was continuous; if the holding-time is short compared to the learning time, which might be the case if the set of input patterns is large, then leakage destroys the learned associations. Their later work developed a weight-refresh scheme to overcome this problem (Arima et al., 1992).

3.9.4 Example 4 : capacitive storage with read-store-and-write refresh

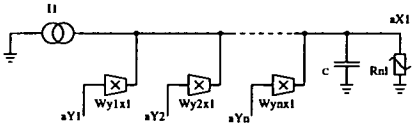
A Spanish group has implemented an algorithm not commonly found in VLSI, the bidirectional associative memory (BAM) (Linares-Barranco et al., 1993). The algorithm allows the network to learn pairs of patterns so that, after learning, one pattern will stimulate the network to produce the other (Levine, 1991). Kosko (Kosko, 1988) developed a dynamical system of differential equations for a general heteroassociative link (ie each pattern in a pair is different in structure and, perhaps, in length) between collections of nodes in networks of the type shown in Figure 3–5 (a). Conceptually, each interconnection has only one bidirectional weight. When pairs of patterns are presented to the pairs of inputs, the outputs of each layer are fed back to the other and a learning rule adjusts the weights. Kosko showed that, where activity-pattern vectors (ie the output vectors) are bistate (binary or bipolar), the weights will converge to a state of equilibrium. Although there are discrete-time versions of Kosko's analysis, it maps naturally onto a continuous-time system, which is what Linares-Barranco's group have built.

The network can use several learning rules, the simplest, and the one used by Linares-Barranco *et al*, being the Hebbian rule : connection strengths are increased with correlated activity of interconnected nodes. The activity of a node is a function of three terms, the nodes's own present input, the weighted sum of inputs from nodes in the other layer, and a non-linear mapping of the node's present activity. The circuit-blocks shown in Figure 3–5(b)[6] for node $Y1$ implement the three terms. An external current-source represents the input I_{y1}, the synapses are transconductance multipliers, and a circuit that behaves like a non-linear resistor makes the non-linear mapping. The voltages on the output nodes

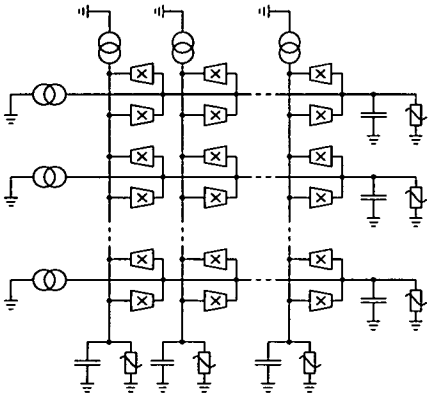
[6]Diagrams adapted from Linares-Barranco's own figures



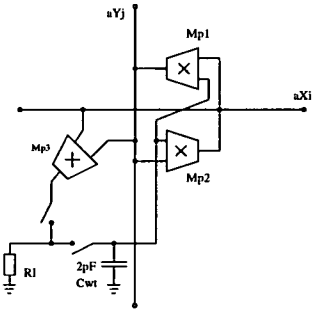
(a) A BAM network



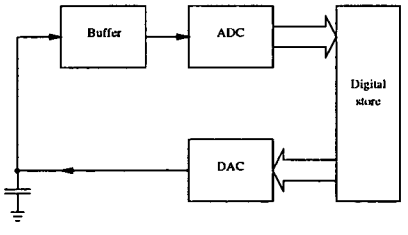
(b) Schematic of a neuron and weighted inputs



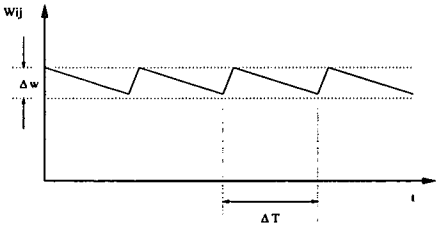
(c) Two layers of neurons, arranged in a grid



(d) Schematic of a single synapse, showing single weight, twin multipliers and weight-modification mechanism



(e) Block digram of weight-refresh scheme



(e) Weight-refresh cycle

Figure 3–5: *The design of Linares-Barranco* et al *for a BAM network*

represent the activity outputs. The whole network is implemented on silicon in the very elegant array design shown in Figure 3–5(c) : y-nodes are arranged in rows, with the columns providing weighted inputs from the x-nodes, while the x-nodes are arranged in columns, with the rows supplying weighted inputs from the y-nodes.

Each synapse comprises three transconductance multipliers (Figure 3–5(d)) and instantiates the Hebbian rule. M_{p1} and M_{p2} are the weighting multipliers. M_{p3} (also a transconductance multiplier, but connected as a transconductance amplifier with negative feedback) provides the load resistance R_L. During the learning phase, the input patterns produce activity voltages that drive the inputs to the M_{p3} amplifier, and so provide the weight-voltage on C_{wt}. This is fed back, in turn, to provide a new set of activity voltages. After a few presentations of the pattern set, the weights settle into an equilibrium state (Linares-Barranco et al., 1993). The network has learned the patterns, and the switches isolate C_{wt}.

The refresh scheme, shown in Figure 3–5(d) and (e), prevent leakage-currents from destroying the weight-values. The buffer feeds the weight-voltage to an ADC and latch, the latch increments to the next level (one of eight), and the DAC writes the corresponding voltage back to C_{wt}. The scheme keeps the weight-values within a finite interval, and moreover allows the weights to be monitored.

The chip carries 5 neurons in each layer (and so 25 weights) and was able to learn its maximum theoretical capacity of two pairs of patterns. One pattern would stimulate the network to produce its pair, despite weight-deviations in 6 of the 25 weights, due to circuit-mismatch. Hence, although very small, the network worked.

3.9.5 Example 5 : purely analogue implementations with floating-gate storage

One group has combined weight-perturbation techniques with both short-term capacitive, and long-term floating-gate, storage, to produce a system that learns on chip and maintains its resultant weight-set over time (Montalvo et al., 1992, Montalvo et al., 1994b, Montalvo et al., 1994a).

The weight-perturbation algorithm works, like back-propagation, by gradient-descent, but circumvents the difficulty of calculating the error-derivative (as back-

propagation requires) by measuring the gradient instead. The algorithm approximates the derivative by measuring the network error, perturbing a weight, measuring the error again, and then modifying the weight in the direction that will reduce the error. The advantages are that the algorithm needs no complex calculation, and requires no exact knowledge of the network's characteristics, since the gradient-measurement takes automatic account of the multiple imperfections of a real-world analogue implementation. The disadvantage is that the algorithm is only semi-parallel. For example, circuitry can perturb, in parallel, all the weights that connect a single hidden node to the output nodes, since each weight affects only one output, but the algorithm must attend to each hidden node in turn, to avoid multiple effects on a single output. (There are stochastic variants of the basic algorithm that perturb all weights in parallel.) However, the advantages make weight-perturbation an increasingly popular approach.

Figure 3–6[7] shows the system. The synapse comprises a weight-modification module, a dynamic weight with a perturbation mechanism, a long-term store and a multiplier (Figure 3–6(a)). C_{dy} is the dynamic store to which the increment and decrement signals can add or subtract charge in the form of the currents through transistors $M1$ and $M2$ (Figure 3–6(b)).

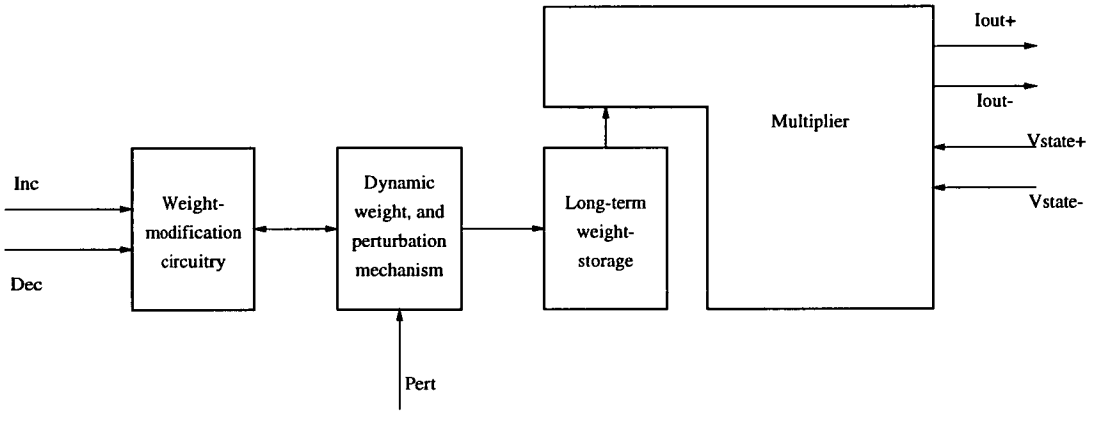
The synapse consists of a double-differential stage with two current sinks : $F1$, which has a fixed gate-voltage reference, supplying a zero-current I_1; and $F2$, which can vary I_2 around the value of I_1, providing a four-quadrant multiplication.

The charge on C_{dy} varies I_2 directly, by varying the gate-voltage on $F2$, but $F2$ can also store the charge more permanently using high-voltage pulses to transfer the charge to the floating gate.

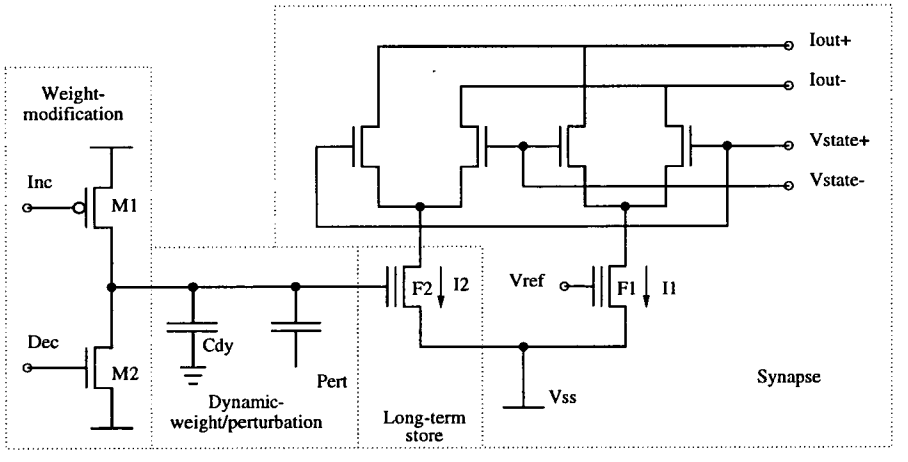
The system provides dynamic and long-term storage, and uses an algorithm that requires neither great accuracy nor complex calculation. The authors claim to be able to limit offsets due to charge-injection on C_{dy}, and furthermore to have developed circuits that compensate for temperature effects, but they have so far presented no results of the network running the algorithm.

---

[7]Diagrams adapted from Montalvo's own figures.



(a) Block diagram of the synapse



(b) Circuit schematic

Figure 3–6: *The design of Montalvo* et al *for combining short- and long-term storage on the same chip*

3.10 Conclusions

On-chip learning is a technique that places the circuitry that adapts weights onto a chip, rather than have the weights calculated on a supporting computer. The technique, although difficult to achieve, offers several advantages over other methods of determining an appropriate set of weights.

On-chip learning can be implemented in digital or analogue hardware, and it appears that, at least at the moment, the advantages of the digital approach make it the medium of choice. Nevertheless, several digital and analogue approaches exist that attempt to place learning circuitry on a chip.

Notwithstanding the variety in approaches, the issues of how to store a weight, and how to change it, are key ones. Examples drawn from throughout the field demonstrate that a range of algorithms can be accommodated to on-chip learning, although the networks tend to be small, and operate with varying degrees of success. These approaches each use different methods of storing and changing weights, and each has its merits and defects; no one approach seems overwhelmingly better than another, and this is partly because, as yet, there is no clear candidate for an application of on-chip learning.

Chapter 4

Hardware functions from the VT algorithm

4.1 Introduction

In this chapter, I show how I translated the virtual targets algorithm into functions capable of being realised in silicon.

4.2 Translating the Algorithm into Hardware

4.2.1 Building on previous work

Colleagues here recently produced a chip called EPSILON, designed to carry out the forward-pass of a number of different algorithms, including back-propagation (Hamilton et al., 1992, Hamilton et al., 1993). The chip could carry out the *multiplication* → *summing* → *non-linear-mapping* operations referred to in section 2.2. The general approach to the design and the performance of the chip in many ways rivalled that of the ETANN chip (Tam et al., 1990) referred to in section 3.2, which was a product of the very large and successful commercial R & D laboratory. The EPSILON chip comprises an array of two-quadrant multipliers, a simple summation method involving summing currents on an electrical node,

and a series of circuits to calculate the activation of a neuron and convert it into an appropriate output, ie all the operations described in Figure 2–2.

Since the forward-pass operations are a major part of the virtual targets algorithm, it seemed an obvious decision to build on this work. Our ability to carry out the operations of a forward pass was taken for granted and I focussed on the weight- and target-modification operations described in Figure 2–3.

4.2.2 Basic hardware principles

A number of basic guidelines have been established in our group that can be sensibly applied to any analogue-hardware implementation of an ANN :

1. A single layer of a network can be translated naturally into a grid pattern, as illustrated in Figure 4–1. To implement a complete network comprising hidden and output layers, there are several possible solutions. There can be enough synapses to implement all the nodes; the outputs of the first layer can be routed off-chip and then back on again; or, if the nature of the signals allows, chips can be cascaded together.

2. It makes sense to represent inputs to and outputs from the chip as voltages. For inputs, it is easier and more accurate to distribute voltages around a grid of synapses. For outputs, it is easier to pass voltages than currents between cascaded chips.

3. Outputs of synapses are generally represented as currents, because summation can be achieved simply and elegantly on a single electrical node according to Kirchoff's current law.

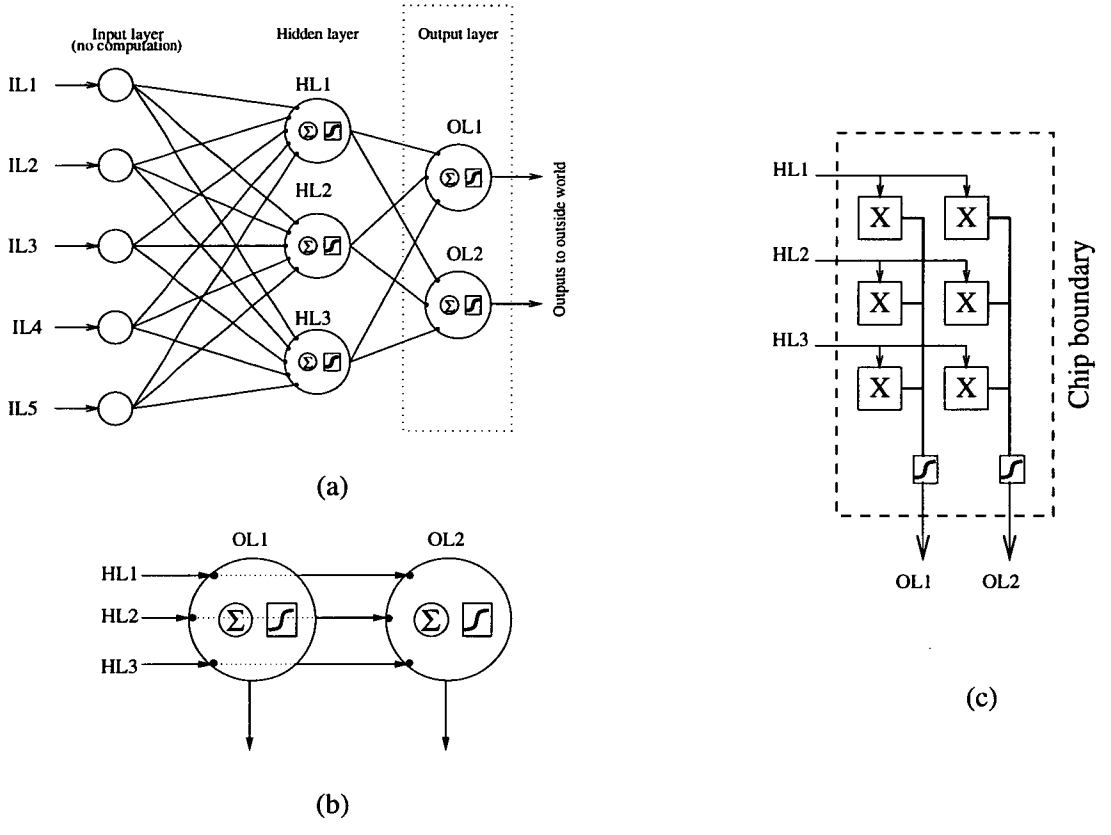


Figure 4–1: *Translating one layer of an ANN, in this case the output layer, into a grid pattern for implementation on silicon. (a) The original network, with the output-layer nodes labelled OL1 and OL2, and the inputs labelled HL1 - HL3. (b) The output-layer nodes rearranged so that each input signal is directed horizontally to the same synapse in each node. The synapses are represented as black dots. (c) The way circuits implementing the various functions might actually be laid out on silicon. Synapses are now represented by squares. Input signals are passed along the rows. Each column contain all the synapses for a particular node. The outputs of the synapses in each column are summed down the column and, at the column-foot, the non-linear mapping is applied to each sum.*

4.3 Implementing the forward-pass equations on EPSILON

This section considers the forward-pass equations and the way in which they were implemented on the EPSILON chip. To understand how this was done is important in appreciating how I developed the original EPSILON design for the virtual targets algorithm and on-chip learning.

4.3.1 Forward-pass equations

The forward pass can be expressed in the following equations for the outputs of the hidden and output layers :

$$O_j = \sigma\left(\sum_{j=0}^{J} W_{ji} O_i\right)$$

$$O_k = \sigma\left(\sum_{k=0}^{K} W_{kj} O_j\right)$$

where the sigmoid function $\sigma(x) = 1/\left(1 + e^{-(x-\theta)T}\right)$

As I have already noted, these equations can be decomposed into a series of *multiplication* → *summing* → *non-linear-mapping* operations, and circuits implementing the various functions laid out in a grid-pattern on silicon. EPSILON was designed in this way.

4.3.2 The architecture and circuits of EPSILON

Figure 4–2 summarises the EPSILON architecture. A detailed understanding is unnecessary, and I only go into further detail when this is important to understand the work of this investigation.

- **Multiplication** The multiplication is a two-quadrant operation because the weights W_{kj} and W_{ji} are bipolar while the input states O_k and O_j

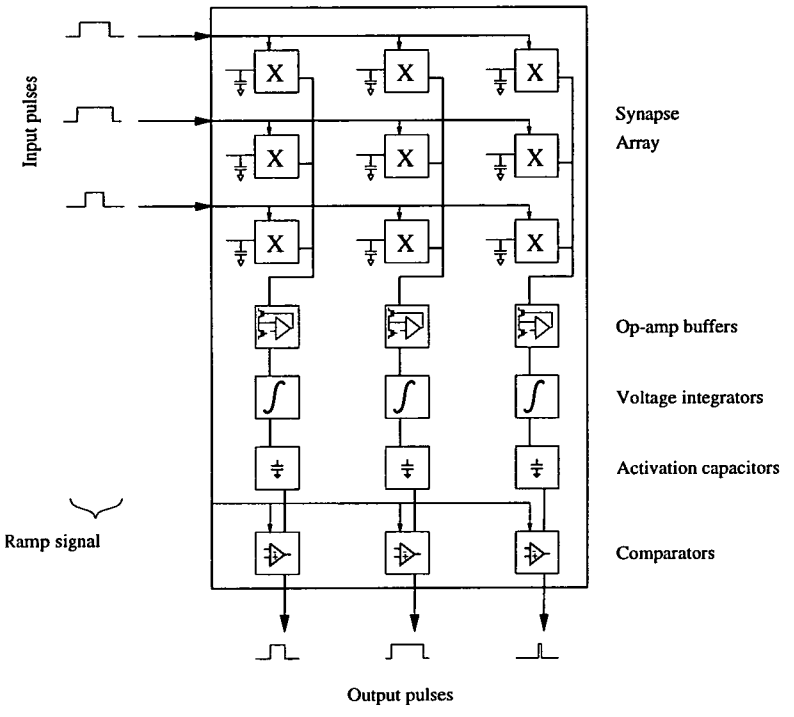


Figure 4–2: *A much-simplified description of the architecture of the EPSILON chip*

are unipolar. On EPSILON, input states are represented as voltage pulses of varying width, the width encoding the value of the state. Weights are represented as charge stored on a capacitor at each synapse site. The output signal from each synapse, the result of a *weight-voltage* $\times$ *state-pulse* multiplication, is a current-pulse.

- **Summation** The output current-pulses from a column of synapses are summed simply on a single electrical node.

- **Non-linear mapping** The means of converting the summed currents, representing the activation, into a value representing the output-state of a node is rather complicated. Each column has a series of circuits to accomplish the transformation. First a buffer-circuit and operational-amplifier convert the summed current-pulses into a series of positive and negative voltage-pulses. A voltage integrator re-converts the voltage-pulses into positive and negative currents that, in the form of charge, are dumped onto or drawn off from an activation capacitor. The final voltage on the capacitor is the weighted sum of the input signals. This voltage is converted into a pulse

via a comparator by use of an off-chip, programmable ramp-voltage. In this manner, all state signals, whether input or output, are pulse-width modulated signals. It also means that the shape of the ramp-voltage determines the mapping between the weighted sum and the node activation, which can, within reason, be any linear or non-linear mapping. As explained by one of its inventors (Churcher, 1993), the ramp shown in Figure 4–2 is a two-sided sigmoid 'on its side' that encodes this particular form of non-linear mapping.

4.4 Implementing the weight-modification equations

In the virtual targets algorithm, the learning rule is represented by modification of weights (as with back-propagation) and additionally by modification of targets which is considered in Section 4.5. As we shall see, although some of the components of the learning rule at first sight seem difficult to implement in analogue electronics, a suitable choice of representation for signals can greatly ease the problems, and some calculations can be done in an analogue way by what are essentially digital signals.

4.4.1 The equations

The equations according to which the weights are modified can be expanded as shown in Figure 4–3. The form of the equation for each layer is identical, and comprises four terms.

The first of these is a gain term, which can be thought of as a means of increasing or reducing the size of the weight-change. The remaining three require to be multiplied together, and comprise the input from the previous layer, a 'sigmoid-prime' term and an error-term. The first task was to develop means of implementing each of the terms electronically. The second task was to find a means of multiplying the three terms together, which seemed a much more difficult proposition.

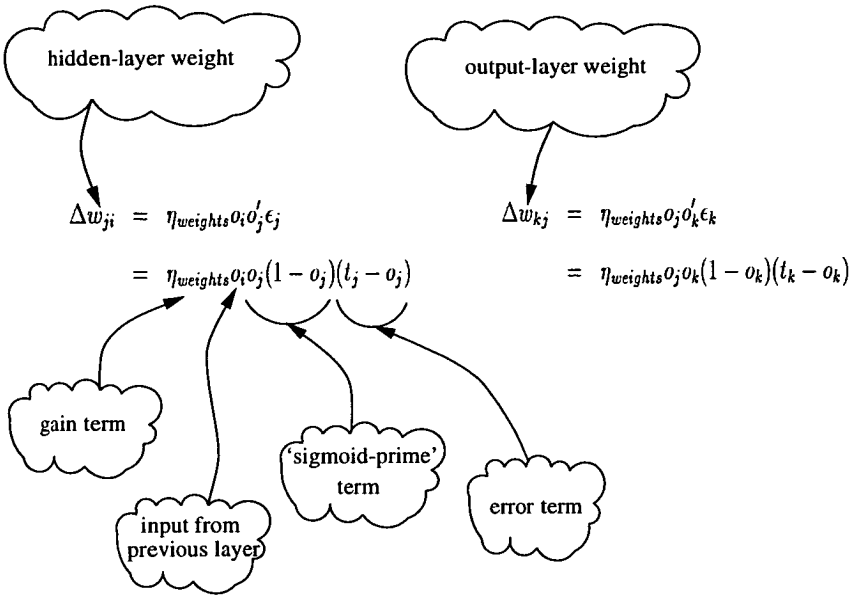


Figure 4–3: *The weight-modification equations for the hidden and output layers. Individual terms in the hidden-layer equation are identified by the names in the bubbles. Since the form of the equations is identical in the two cases, the terms for the output-layer equation are equivalent to those in the hidden layer.*

4.4.2 Input and output signals

I considered changing the form of representation of input and output (state) signals from pulse-width modulation to some other form, perhaps pulse-frequency modulation which had also been investigated by our group, and perhaps even an entirely different form. However, the advantages of retaining the pulse-width modulation scheme were overwhelming. The group had had plenty of experience in the design of pulse-width modulated circuits, the operation makes inter-chip communication possible and the EPSILON design could be used for the forward-pass calculations.

4.4.3 Implementing the 'sigmoid prime' term

The sigmoid prime term is of the form $O_k(1-O_k)$, and the means of implementing it is to consider it as an analogue function with a state input O_k and an output $O_k(1 - O_k)$. The graph of the function is shown in Figure 4–4(a). Our initial consideration of this problem involved a switched-capacitor circuit with a rather

complex clocking scheme and, as an alternative, an inverter circuit in which transistor characteristics were used to try and replicate the features of the curve. It then occurred to us that an accurate representation of the curve was probably not important for the success of the algorithm, but only its general shape. We therefore considered implementing it as the approximation shown in Figure 4–4(b).

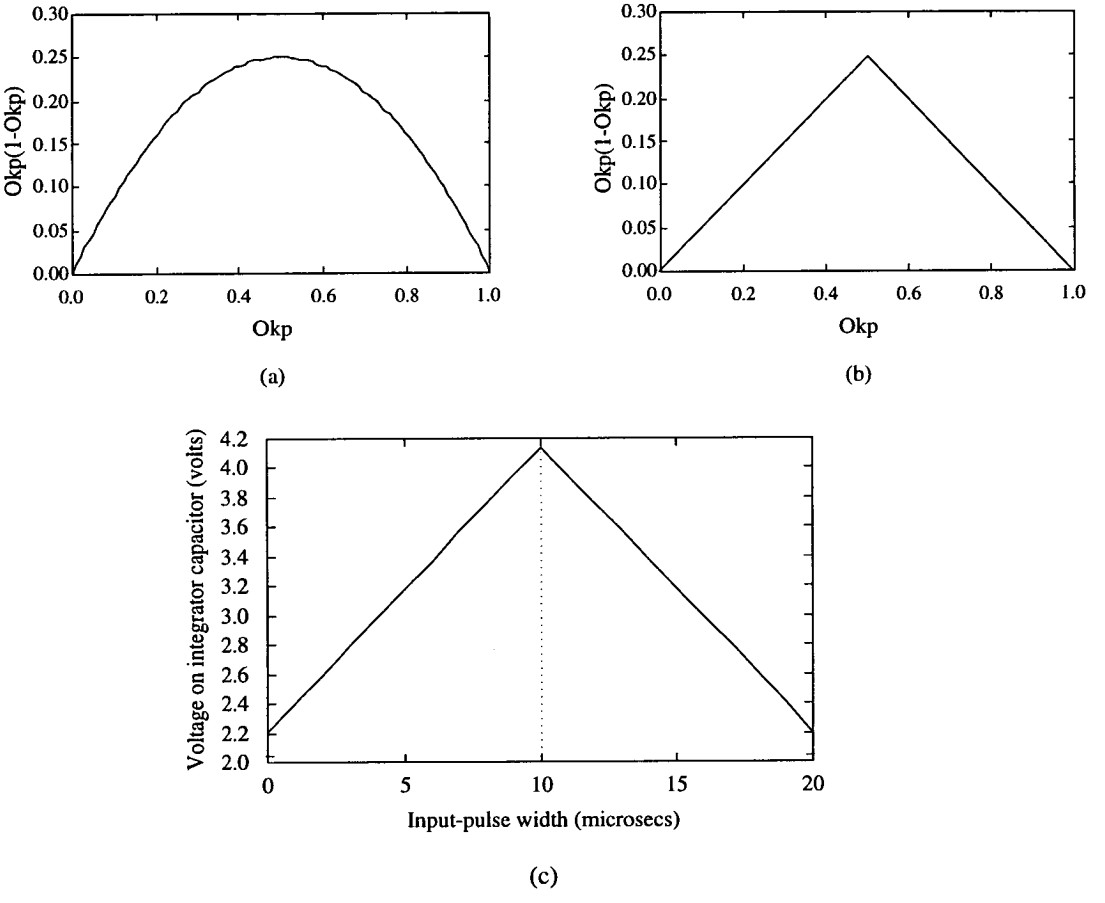


Figure 4–4: *The 'sigmoid-prime' term. (a) Graph of the function $O_k(1 - O_k)$. (b) A more-simply implemented approximation.*

In fact, once we have chosen pulse-modulated signals as the states, to produce the sawtooth version of the function proves very simple using an XOR gate[1]. The basic idea is illustrated in Figure 4–5.

[1]The original idea was that of my second supervisor, Martin Reekie

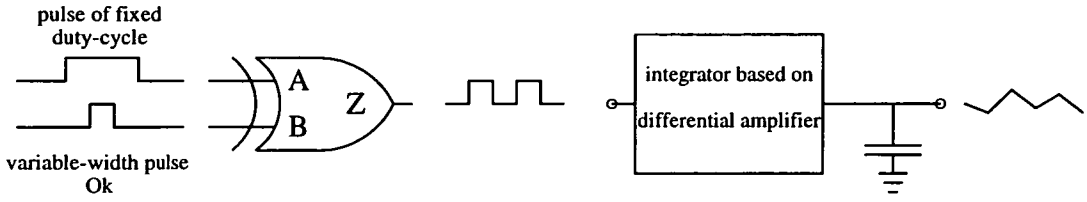


Figure 4–5: *The fundamentals of the circuits to implement the 'sigmoid-prime' term. Appropriate input signals to the XOR gate produce a series of pulses at the output that can be integrated to give the correct result.*

An XOR gate receives two pulsed inputs, a fixed, control signal, and a pulse-width signal representing the state. The gate's output is a pulse-train whose **ON**-time-*minus*-**OFF**-time rises and then falls, like the sawtooth curve illustrated in Figure 4–4, as the state-pulse increases from zero to some maximum width. The following circuit is a voltage integrator that, during the **ON**-time, sources current onto an integration capacitor, and during the **OFF**-time, sinks current from the capacitor. The final calculation is the result of the function.

The result rests on the idea of a fixed time-frame in which pulsed signals are centred on a particular time in the frame. To see how this is done in practice, some exemplar signals are shown in Figure 4–6[2]. The time-frame chosen is 40μs. The signal applied to input A of the XOR gate is in every case a fixed 50% duty-cycle pulse train, centred on the 20μs point in the frame. Input B receives the pulse representing the state O_k, centred on the 10μs point, and examples encoding three state-values of low-value, one half and one are shown. The gate's output is a series of pulses of a number and duration dependent on the width of the O_k pulse. As the O_k pulse increases in width, the **ON**-time-*minus*-**OFF**-time of the gate's output pulses rises correspondingly, and then falls.

I chose a standard circuit for an XOR gate, laid it out using silicon design tools, and simulated it using Hspice.

---

[2]The graphs shown here are Hspice simulations of circuits extracted from VLSI layout.

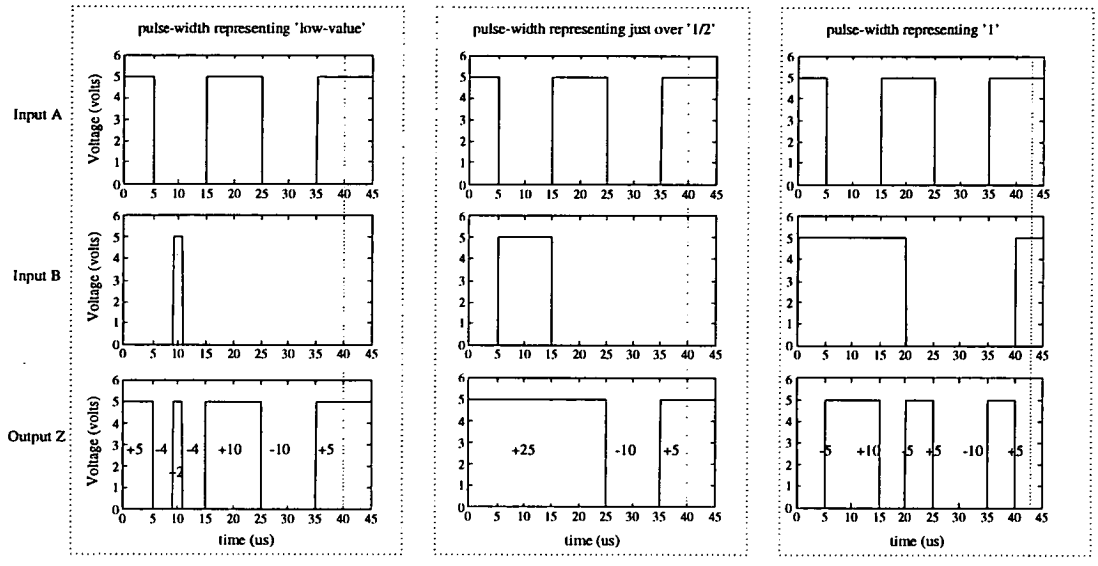


Figure 4–6: *Calculation of the 'sigmoid-prime' term, shown by exemplar inputs to, and corresponding outputs from, the XOR gate. The output signal's ON-time-minus-OFF-time varies in an analogue manner, as explained in the text.*

4.4.4 Implementing the error term

The error-term is of the form $T_k - O_k$, where T_k is a target value and O_k is a state value. If both these values are represented by a pulse-width modulation signal, the result can again be computed using an XOR gate. This time the result does not depend on the two pulses being centred on a point in the time-frame, but is achieved if they are coincident in this way (as shown in Figure 4–7) or have coincident leading or trailing edges.

The calculation of this term is so simple it needs no further explanation, other than to say that the result can be used in exactly the same way as a state-value can be used.

The circuit used for the XOR gate for the sigmoid prime and error-term functions is shown in Figure 4–8.

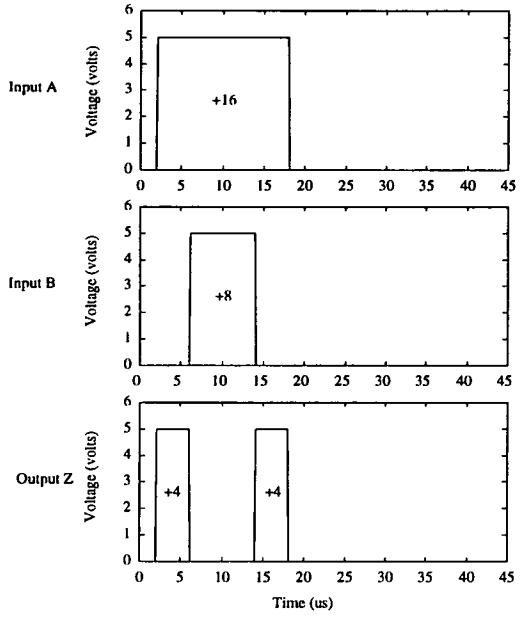


Figure 4–7: *Calculation of the error term, shown by exemplar inputs to, and corresponding outputs from, the XOR gate.*

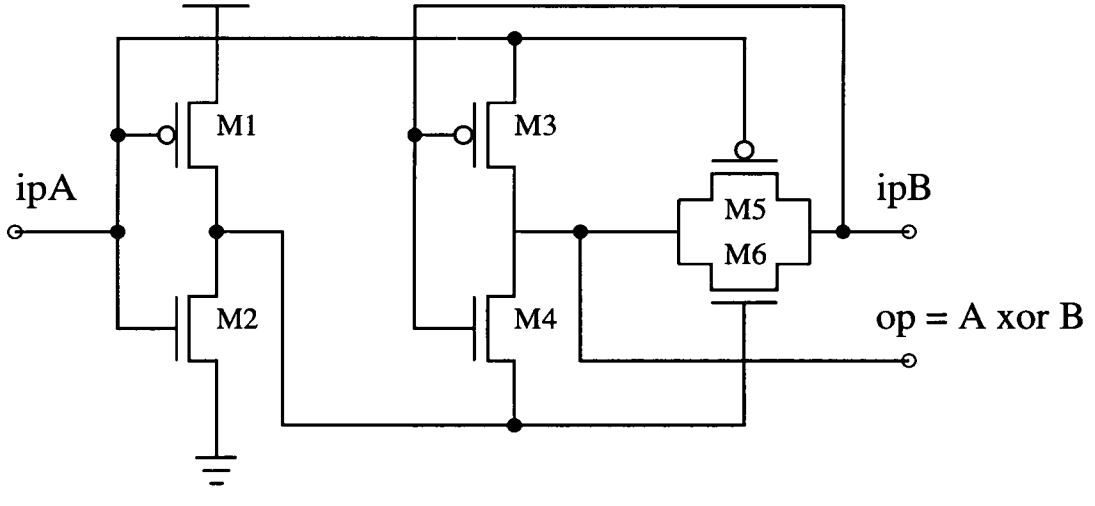


Figure 4–8: *The standard circuit for the XOR gate used for the 'sigmoid-prime' and error-term functions*

4.4.5 Implementing a sign circuit

The calculation of an error-term that is the difference of two values raises the issue of sign, since one must determine whether a negative result will be generated. The idea of centring the pulses that represent targets and errors on a point in a time-frame again makes the solution easy. This time, the function can be implemented using an S-R flip-flop (see Figure 4–9), the only constraint being that, whatever the inputs, the flip-flop should not settle in an indeterminate state.

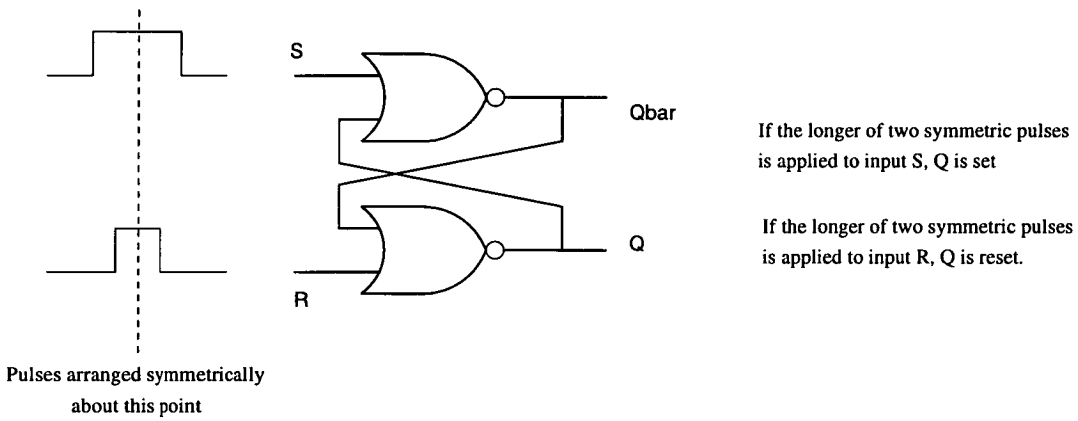


Figure 4–9: *Using an S-R flip-flop to determine the sign of the result of the subtraction of one pulse from another*

I chose a standard circuit for an S-R flip-flop, and simulated it using Hspice. The only difficulty was in ensuring the circuit never settled into an indeterminate state. This involve choosing non-standard transistor sizes for some of the transistors. I then laid out the circuit using silicon design tools, and re-simulated it to verify its operation under all input conditions. The circuit is shown in Figure 4–10.

4.5 Implementing the target-modification equation

We can use some of what we have learned from implementing the weight-modification equations in implementing the target-modification equation. This issue only affects the hidden-layer targets because, as the virtual targets algorithm

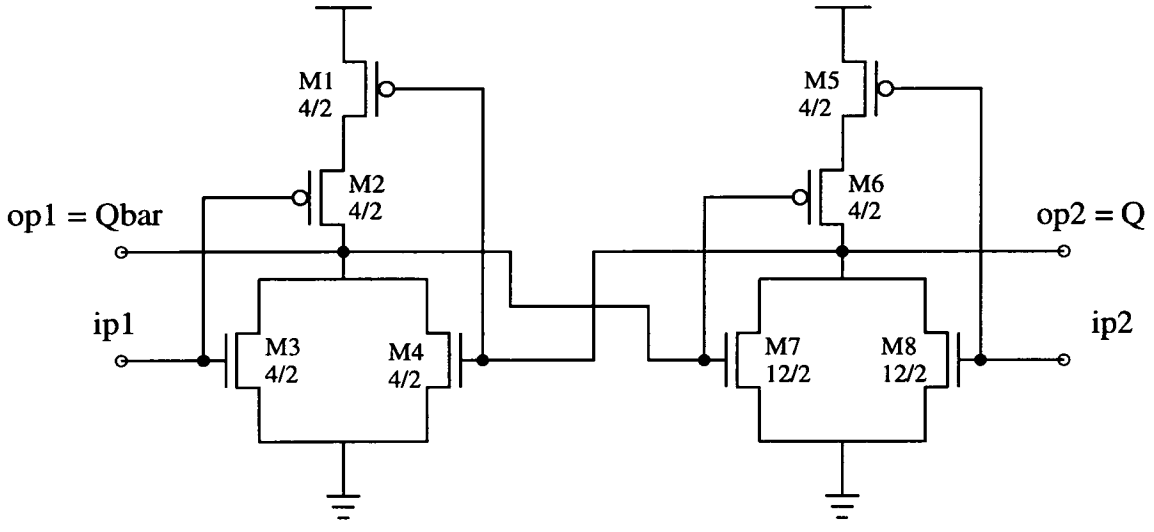


Figure 4–10: *The flip-flop circuit used for the sign function. Judicious choice of transistor sizes ensures the flip-flop never settles in an indeterminate state.*

is a supervised-learning algorithm, targets for the outer layer have to be specified explicitly as part of the problem that the network is trying to solve.

4.5.1 The equation

The equation for update of the hidden-layer targets is :

$$\Delta T = \eta_{targets} \sum_{k=0}^{K} W_{kj} \epsilon_{kp}$$

We can immediately note three things : that the error-term that occurs in the weight-change equation reappears here; that two terms must be multiplied together; and that the results of several multiplications must be summed. The equation is, then, very like the forward-pass equations referred to in Section 4.3.1 on page 62. The difference is that, whereas the forward-pass equations require a two-quadrant multiplication, both the weight and error terms are bipolar, and so a four-quadrant multiplication is required.

I approached this problem, therefore, from the point of view that, if a suitable four-quadrant multiplier could be developed, the implementation of the target-

modification equation would be, to all intents and purposes, solved. However, the design for a suitable multiplier took considerable thought.

4.5.2 The EPSILON synapse

Our start-point was the two-quadrant multiplier developed for the EPSILON chip and shown in Figure 4–11.

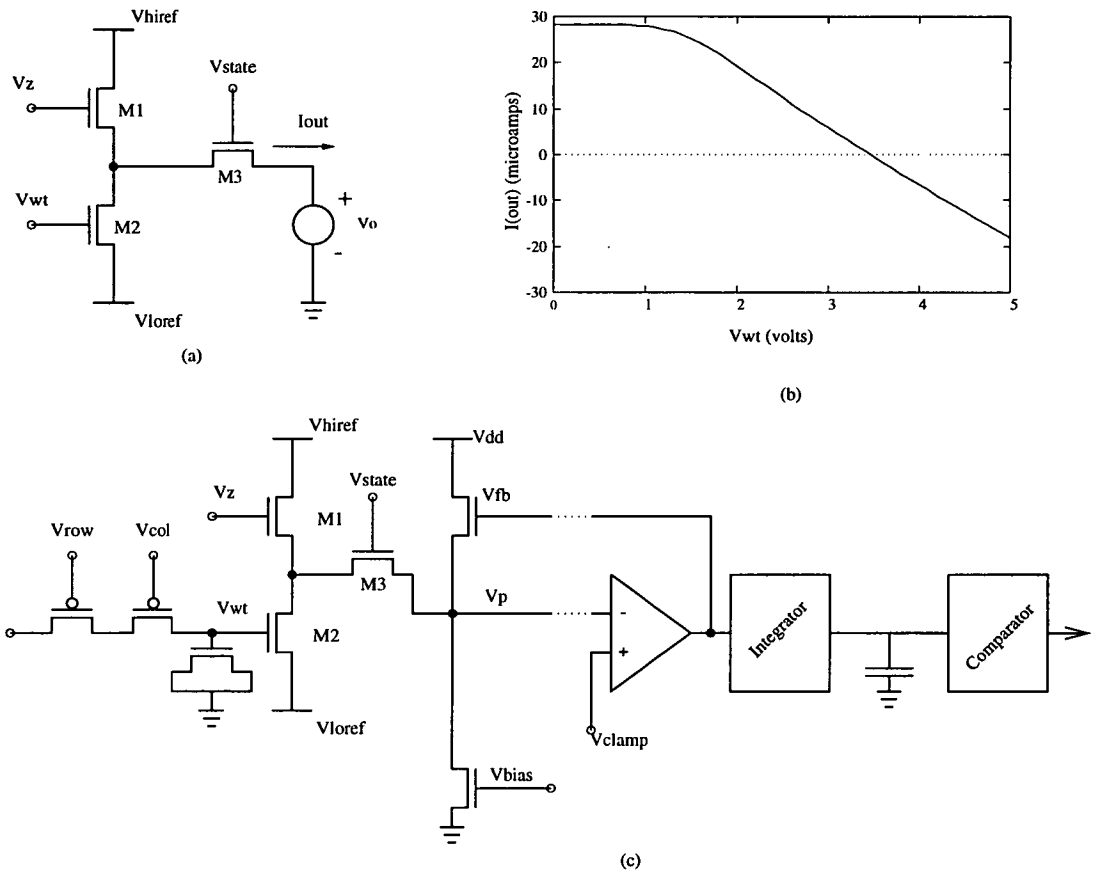


Figure 4–11: *The EPSILON synapse. (a) The transconductance amplifier, which is at the core of the circuit. (b) The amplifier's output characteristics when v_{out} is varied. (c) The complete synapse and functional blocks that make up a complete synapse and neuron.*

The transconductance amplifier consists of only three transistors supplied by rails with a voltages differing by around 1 volt, which keeps M_1 and M_2 in the linear region of operation. The voltage V_z is fixed, so that transistor M_1 acts as a constant-current-source. The voltage v_{wt} on M_2 represents a weight that, as it

rises, causes M_3 to sink varying currents. The resultant current through M_3 varies linearly, as shown, from positive to negative values, ie in two quadrants, provided the rail voltages are low compared to v_{wt}. If v_{state} is a digital pulse, i_{out} is a current-pulse whose amplitude reflects the weight-voltage and whose duration reflects the state-value. If the current-pulse is integrated, a *weight* $\times$ *state* multiplication is achieved. If the output currents from a set of parallel synapses are integrated, the result represents a sum of products, as the equations require.

For the target-modification equation, another circuit producing four-quadrant multiplication is required. Alternatively, the performance of the EPSILON circuit requires to be extended to four-quadrants, but it is not immediately obvious how this might be done.

4.5.3 Option 1 : the Gilbert multiplier

The Gilbert multiplier (Kub et al., 1990, Schneider and Card, 1991b) is very popular , for all kinds of applications, not only for ANNs, and its basic structure and output characteristics are shown in Figure 4–12.

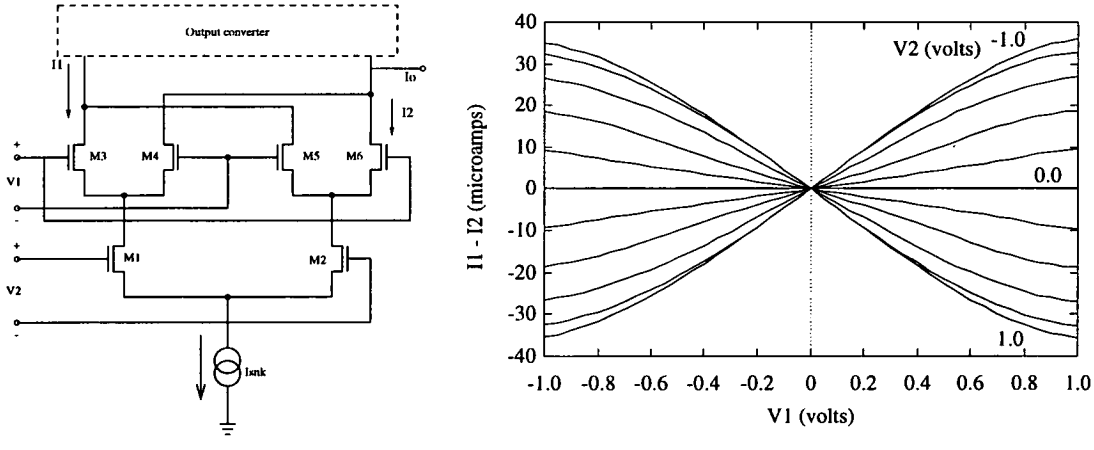


Figure 4–12: *Basic structure of the Gilbert multiplier, and its output characteristics . The circuit's operation is described in the text.*

Analysis that assumes the square-law approximation for the transistors working in saturation, and hence that their drain-source current is $i_D = \frac{1}{2}\beta(v_{GS} - V_T)^2$, shows that the difference of the output currents i_1 and i_2 is :

$$i_o = i_1 - i_2 \simeq \left(\sqrt{\frac{1}{2}\beta_u\beta_l}\right)(v_1 v_2)$$

where β_u and β_l are the transconductance parameters of the upper two differential pairs and the lower differential pair, respectively. In other words, i_o is the product of the two differential input voltages.

The circuit has much to commend it. Its operation is well verified, the output characteristics are regular and the curves well-spaced, and a choice of output converter can provide a single-ended voltage or current.

A disadvantage is that the gain of the circuit can be very large, making the differential voltages that can be multiplied to give a linear output correspondingly small. The wide-range version[3] also comprises around 20 transistors. However, the greatest disadvantage is in trying to develop a circuit of which our group had no experience and which was not a pulse-mode circuit. All the inputs to the multiplier are differential voltages, not pulses where states are encoded in time.

4.5.4 Option2 : the Dupuie multiplier

A beautifully simple multiplier with good output characteristics is the Dupuie multiplier (Dupuie and Ismail, 1990), shown along with its output characteristics in Figure 4–13.

The inventors envisaged the multiplier output-nodes being connected to an operational amplifier to compute the current difference. Unlike the Gilbert multiplier, and like the EPSILON synapse, correct operation relies on the transistors being held in their linear region. In other words, the voltage over the drain and source terminals, v_{DS}, must be kept low (say between 1 and 1.5 volts), compared to the voltages on the transistor gates,v_{GS}, (say 3 to 4 volts); in these circumstances, the transistors' drain-source currents, and hence the output currents, are a linear function of the gate-voltages. The multiplier is able to achieve four-quadrant operation because the drain and source terminals can be reversed, and current can

---

[3]The term 'wide-range' refers to the multiplier's ability to multiply voltages near V_{DD} or ground

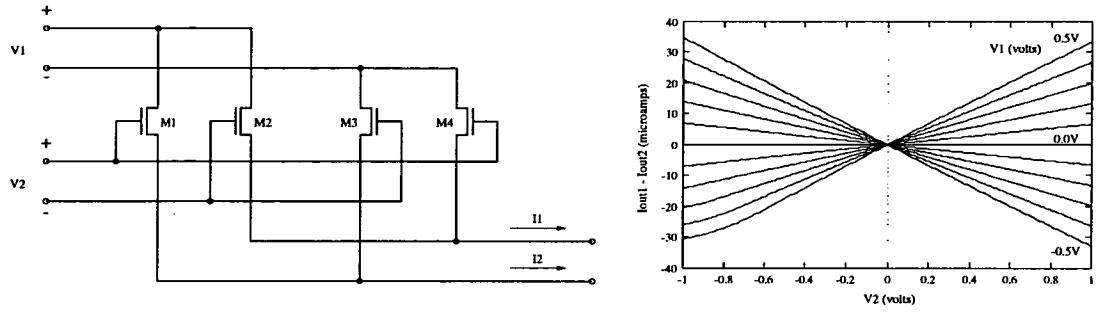


Figure 4–13: *Basic structure of the Dupuie multiplier, and its output characteristics . The circuit's operation is described in the text.*

flow through the transistors in either direction. There is a 2-transistor version of the same circuit with similar characteristics.

This circuit, like the Gilbert multiplier, requires inputs in voltage, rather than pulse, mode.

4.5.5 Option 3 : twin EPSILON synapses

Another possibility is the development of the original EPSILON synapse into a four-quadrant form. Two ways in which this might be done are shown in Figure 4–14. V_{wt+} is a positive weight, while V_{wt-}, which is stored at the same time, is its negative 'mirror' around a zero voltage. Pulses that represent states are designated 'positive' or 'negative'. In the scheme shown in Figure 4–14(a), control logic determines which position switch S adopts, but an objection would be that charge-sharing between the capacitors, as the switch changes position, might seriously distort the weight representation. In Figure 4–14(b), this objection is overcome : a twin-synapse performs the four-quadrant multiplication, and this time the control-logic determines which half of the synapse is used.

In circumstances where learning takes place off-chip, say on a supporting PC, the problem of calculating a positive weight and its negative mirror is easily solved in software. After each calculation, the computer down-loads new values to the chip. However, an objection to both the schemes outlined here is the difficulty of providing a weight-voltage and its mirror when the weights are changing. One problem is that of incrementing a positive weight and decrementing a negative one by the same increment. Another problem is of matching capacitances, with the

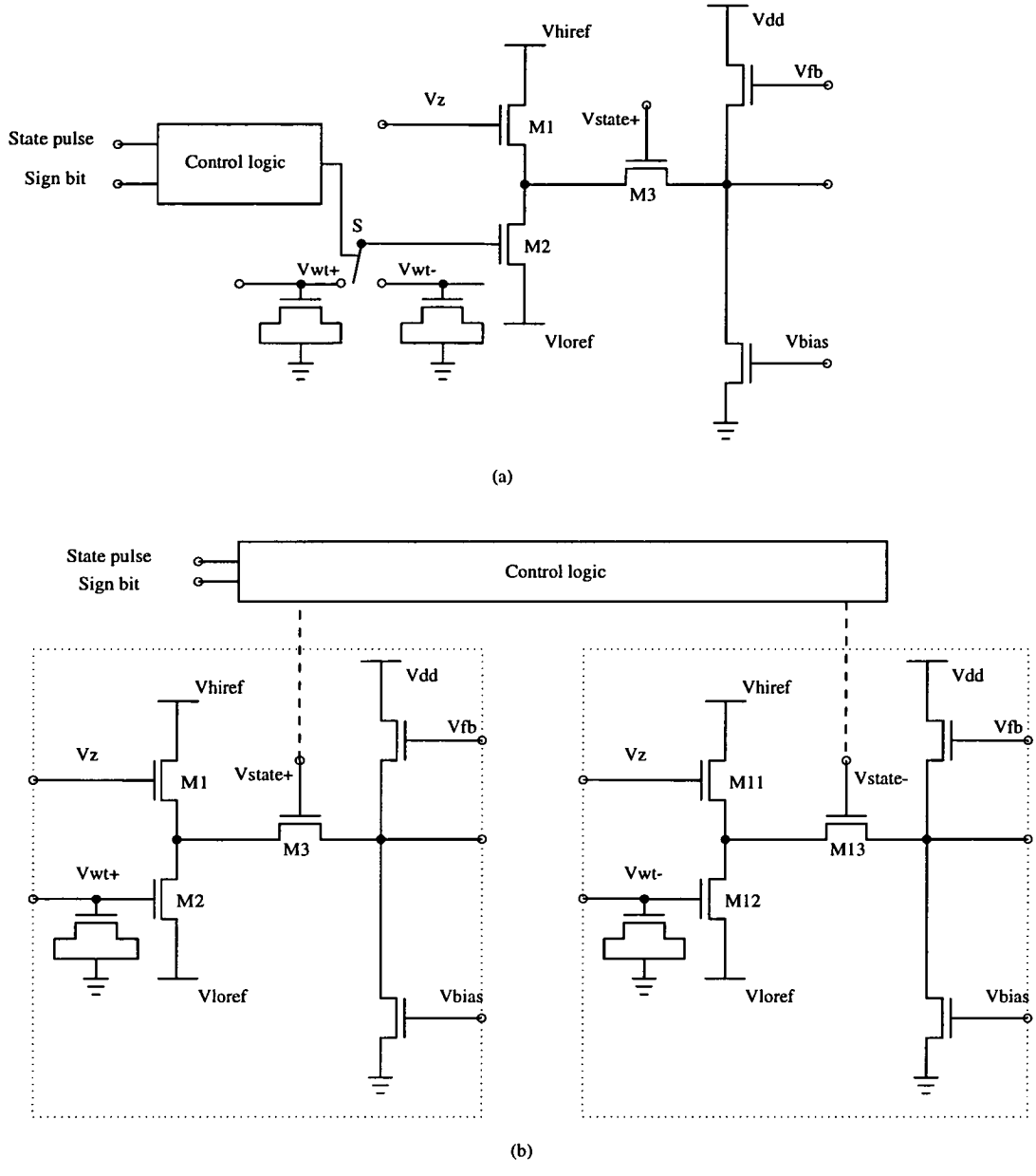


Figure 4–14: *Schemes for implementing the EPSILON synapse as a four-quadrant multiplier. (a) Two capacitors, one representing a positive weight and the other its negative mirror, serve one synapse, and a switch selects the appropriate capacitor. (b) To circumvent charge-sharing, twin-synapses are used and control logic selects which of a 'negative' or 'positive' pulse is directed to the appropriate synapse.*

danger that the variation between the two capacitors exacerbates the difficulty of matching increments and decrements.

4.5.6 Option 4 : twin EPSILON synapses operated in parallel

A solution that is not without problems, but nevertheless is a substantial improvement on the schemes proposed so far, is shown in Figure 4–15. I have already noted, in connection with the Dupuie multiplier (Section 4.5.4), that a transistor operating in its linear region not only provides a current that is linearly related to its gate-voltage, but also can have its drain and source voltages reversed. This makes the original EPSILON synapse, in all essentials, symmetrical to the rail voltages. The left synapse in Figure 4–15 is a normal EPSILON synapse, while the right synapse has its connections reversed so that transistor M_2 acts as a constant-current sink, while transistor M_1 sources a current that varies linearly with the weight voltage.

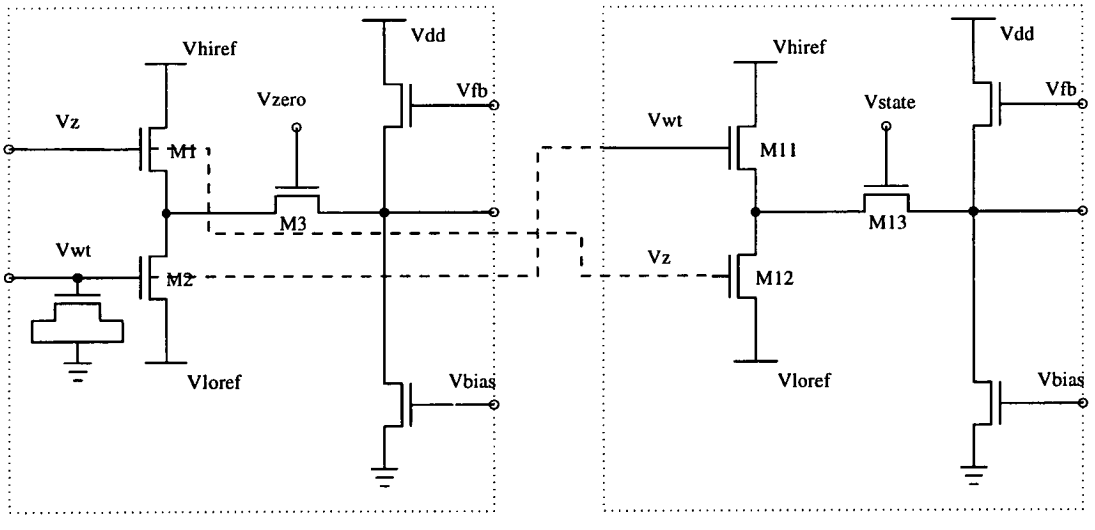


Figure 4–15: *A four-quadrant multiplier. A single capacitor drives twin synapses, in one of which the V_z and V_{wt} connections have been reversed. A 'zero'-state pulse is supplied to one synapse, while the other receives a normal state pulse.*

The circuit is by no means perfect, as the simulation results of Figure 4–16 show. The curves are reasonably symmetrical in the x and y planes, but the intended

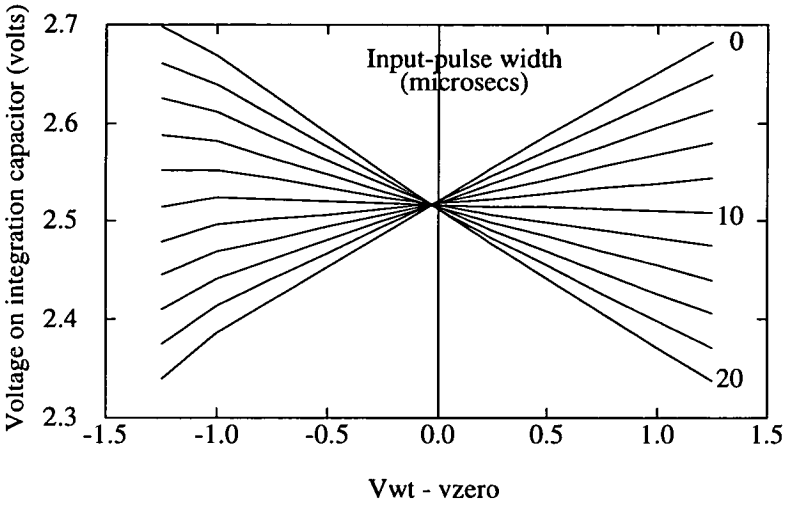


Figure 4–16: *Output characteristics of the four-quadrant multiplier.*

zero output voltage of 2.5v is actually offset. The size of this offset is dependent on a number of factors concerned with the operation of the synapse circuit itself and other circuitry that makes up the whole neuron. An analysis of the twin synapse is given in Appendix E, which shows that that part of the offset due to the synapse is caused the asymmetry of its two halves. This imbalance can be corrected (in a manner explained in Appendix E) by the addition of a current source to the circuit. There are several practical solutions to the correction of the offset but, due to approaching chip deadlines, I left this matter aside.

4.6 Summary of design work

Although I had undertaken a wide-ranging consideration of the possibilities for several circuits, the final decisions condensed into the following straightforward conclusions :

- *Retaining the pulse-mode approach has strong advantages.* Unfortunately, the use of the elegant Gilbert or Dupuie multipliers is precluded, at least without substantial further development, because they operate with voltage-mode, not pulse-mode, inputs. Set against this disadvantage, the sigmoid-prime, error and sign circuits which, at first sight, seem difficult to design turn out to require no more than slightly modified digital gates.

- *Building on the design work of the EPSILON chip solves a number of immediate problems.* The forward-pass equations can already be implemented. The target-modification equation can also be implemented, except that a four-quadrant multiplication is required.

- *A four-quadrant multiplier with the essentials of the correct functionality can be derived from the EPSILON synapse.* This also retains the use of pulse-mode signals and makes it easy to interface to other EPSILON circuits.

- *The question of implementing the weight-modification equations remained to be solved.* As things stood at this point, the equation required a three-term multiplication in which each term was a pulse.

4.7 Test chip : design, testing and results

This section explains the architecture of the test chip and describes how I built a test-system, which gave me valuable experience in providing the various voltage, current and pulse signals that drive the EPSILON circuits and those of my own design.

4.7.1 Objectives

The design of the test-chip had, in terms of results, a single objective : to assess the basic functionality of the four-quadrant multiplier and sigmoid-prime circuits. For reasons of space and time, the error- and sign-circuits, although designed and laid out in the design tools, were not committed to silicon, on the assumption that their simplicity more or less guaranteed their ability to operate as predicted.

4.7.2 Chip architecture

The architecture of the chip is shown in Figure 4–17. The chip comprised 4 synapses, with their associated neurons (op-amp buffer, integrator and comparator), together with an **XOR** gate and associated integrators and comparators.

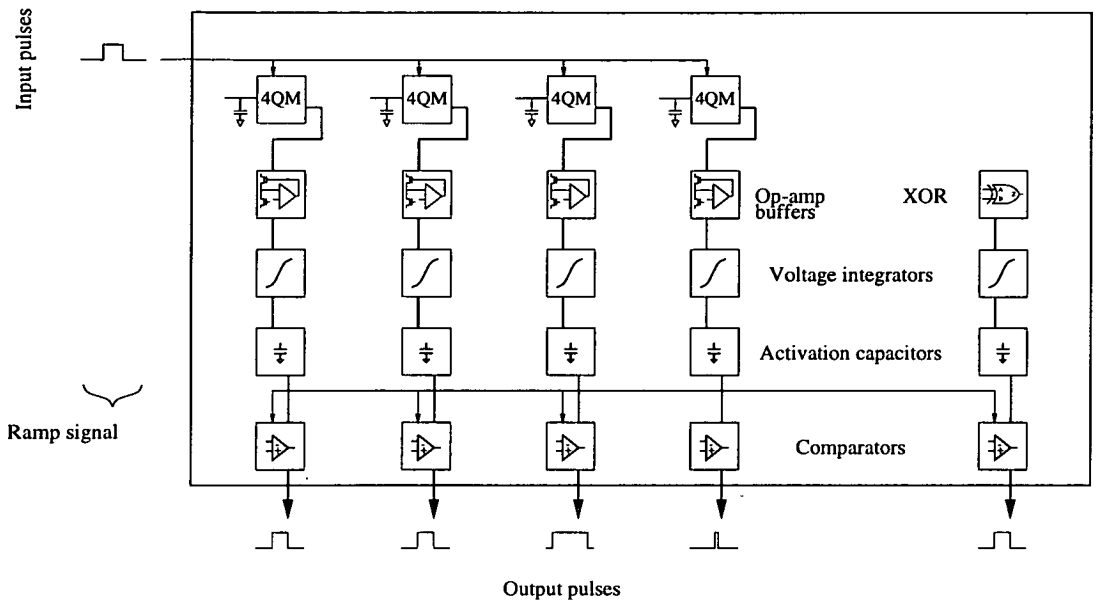


Figure 4–17: *Architecture of the test chip, with 4 synapses, a sigmoid-prime circuit, and support circuits.*

4.7.3 Testing

Further details of the chip's design and testing are given in Appendix F

4.7.4 Results from the test chip

The results of the test procedure confirmed the functionality of both the four-quadrant multiplier and the sigmoid-prime circuits. Figure 4–18 repeats the simulation results shown earlier, alongside results taken from the chip.

The results proved very difficult to obtain. The EPSILON circuits are difficult to set up, because accuracy in their several current and voltage supplies can be critical. The test board and associated state-machine proved very noisy, possibly because the clock-rate of 2MHz (divided down to 1MHz for clocking the ROM), required to give a resolution of $1\mu s$ pulses, was probably near the limits of the state-machine's performance. Graph (b) of Figure 4–18 shows results from a single run of a single synapses-and-neuron column; the curves seem to show two distinct zero-points, circled on the graph, the cause of which was very difficult to identify. Graph (c) shows results of the average of a number of runs, with

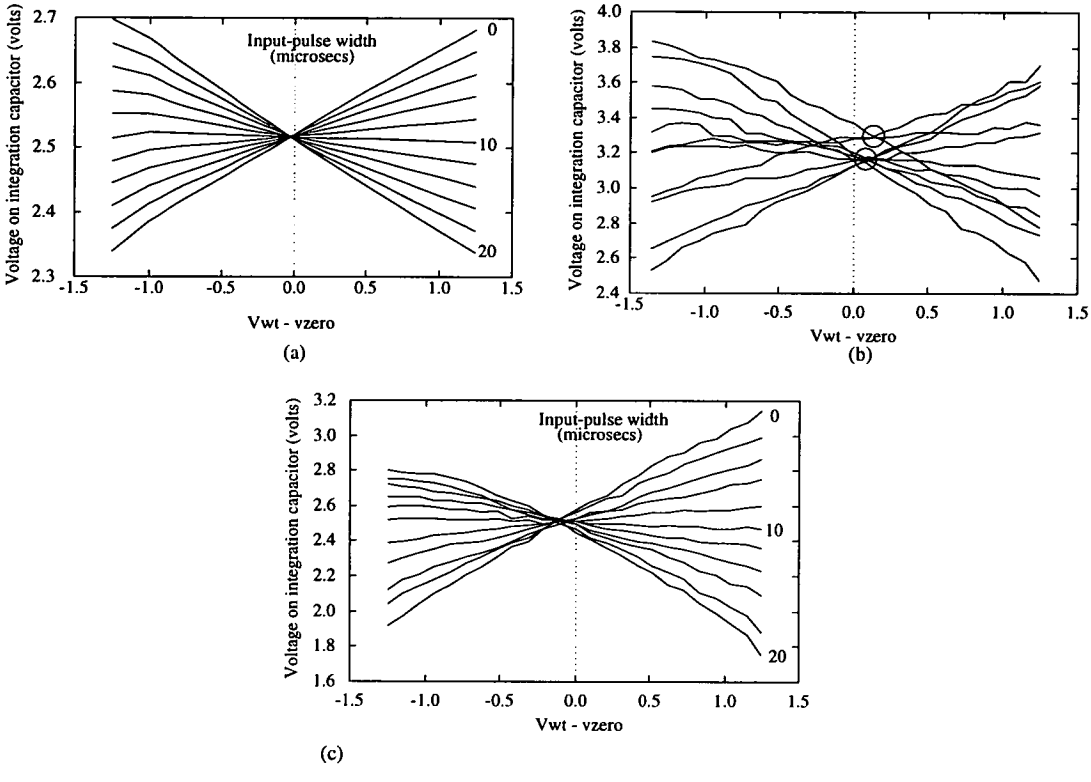


Figure 4–18: *Results of tests of the four-quadrant multiplier. (a) Simulation results. (b) Results from a single synapse and neuron. (c) Results from a single synapse, averaged over 10 runs, and corrected for double-zero point. Chip integration-capacitor voltages are inferred from pulse-outputs which had previously been calibrated.*

the double-zero point corrected artificially. The characteristics of the measured results are asymmetrical about the zero-weight and $2.5V$ axes. Set against these points, the general characteristics are as expected, the curves are reasonably evenly-spaced, and the output range is clearly scalable.

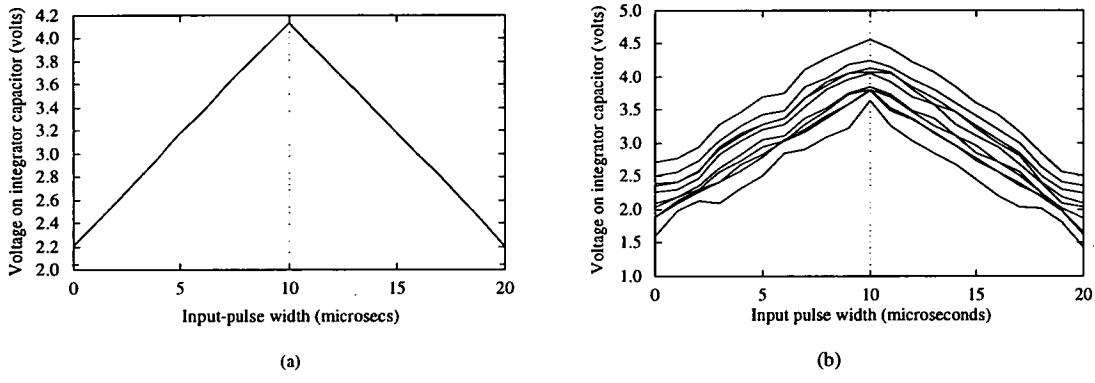


Figure 4–19: *Results of tests of the sigmoid-prime circuit. (a) Simulation results. (b) Results from 10 chips. Chip integration-capacitor voltages are inferred from pulse-outputs which had previously been calibrated.*

For the sigmoid-prime circuit, testing was reasonably easy, and Figure 4–19 shows the results. Although there is a variation between chips, I did not anticipate that accuracy in the calculation would be critical. As with the multiplier results, however, I did expect a problem with establishing a zero-point.

4.8 Summary and conclusions

Given an MLP algorithm with advantages over back-propagation for implementation in VLSI, I was able to break some of the required functions down into a series of modules capable of being realised with hybrid circuits. The modules concerned four-quadrant multiplication, the sigmoid-prime term, the difference of two variables, and the sign of a calculation. Although I considered several possibilities from the literature for a four-quadrant multiplier, I was able instead to find an option that matched well with, and allowed us to use, much of the work that had gone into developing the EPSILON chip. The use of pulse-width signals also made design of circuits for the sigmoid-prime, difference, and sign terms much easier than had at first seemed possible.

I constructed a chip to test the functionality of the four-quadrant and sigmoid-prime circuits and concluded that, while there were problems, notably with noise and the difficulty of setting up circuits, the basic functionality existed to make further development of the algorithm worthwhile.

Chapter 5

Simplification of the algorithm

I still needed to solve two large problems. One concerned the complexity of the weight-change equation. The other problem, consequent on the first, was a means a making the weight-change appropriately. I discuss the second of these problems in Chapter 6, but here I consider the complexity of the weight-change equation and what might be done to simplify it.

5.1 Software simulation and hardware computation

In some ways, digital-computer simulations of neural networks sit uneasily in relationship to analogue hardware, as Table 5–1 shows[1]. If an algorithm does not work in a software simulation, then it is very unlikely to work in hardware; but the fact that the algorithm *does* work in software only suggests the possibility of success in analogue hardware, not how it might be achieved, nor how the hardware might behave in practice. The transformation from software to hardware is a

[1]The Table introduces the terms 'accuracy' and 'precision'. There seem no universally-agreed definitions. Kirk (Kirk, 1993) defines them like this : 'accuracy' is how closely a measurement agrees with our expectations ("getting what you want") and precision is the degree of agreement of repeated measurements, and so a quantification of the useful information present ("knowing what you've got").

question of subjective judgements rather than a systematic application of well-understood rules.

Software Simulation	Hardware implementation
Precision depends on hardware but can be implemented higher degrees in software	Very high precision (but value not known precisely)
Perfect accuracy	Accuracy imperfect
Timing of calculations and updates can be controlled precisely	Timing can be stepped, but little control over exact timing of changes in individual circuits
Processing is generally serial (and slow)	Processing can be parallel (and fast)
No change in values over time or space	Values of variables change due to leakage currents, changes in temperature, variations in process, etc
Tiny changes can be made to values, and dynamic range is high	Changes may have to be large, and dynamic range may be low
Mistakes can be easily rectified	Mistakes have to be circumvented or otherwise overcome
'Tweaks' are easy to implement	'Tweaks' may be difficult to introduce
Skills for writing code are important	Many different skills, including coding, are required
Results are repeatable	Results of repeated computations may vary due to noise, temperature changes, etc

Table 5–1: *Comparison of features of digital-computer simulations and analogue-hardware computation.*

With software simplifications in mind, I asked these questions :

- How could the algorithm be simplified?
- Could the changes be reduced to questions of proportionality or scale?
- Could a single measure of success be developed for tests of the simplified algorithm?
- Could the changes be translated into hardware that was easy to control?

5.2 Changes to the target-modification algorithm

The target-modification equation decrees that hidden-layer targets will evolve so : $\Delta T = \eta_{targets} \sum_{k=0}^{K} W_{kj} \epsilon_{kp}$. My original simulations did not constrain hidden-layer target-values (ie the limits were those of the computer), and increments to target-values were differently scaled to decrements. I constrained the values between **0** and **1**, made increments and decrements the same, and ensured a straightforward proportionality between the target-change and the sum-of-products, so that : $\Delta T \propto \sum W_{kj} \epsilon_{kp}$. In this way, the target-modification circuit would only have to scale the result of the sum-of-products calculation.

This gave me a baseline for further simulations, and Figure 5–1 graphs the results of learning on the character-recognition problem referred to in Section 2.5.4.

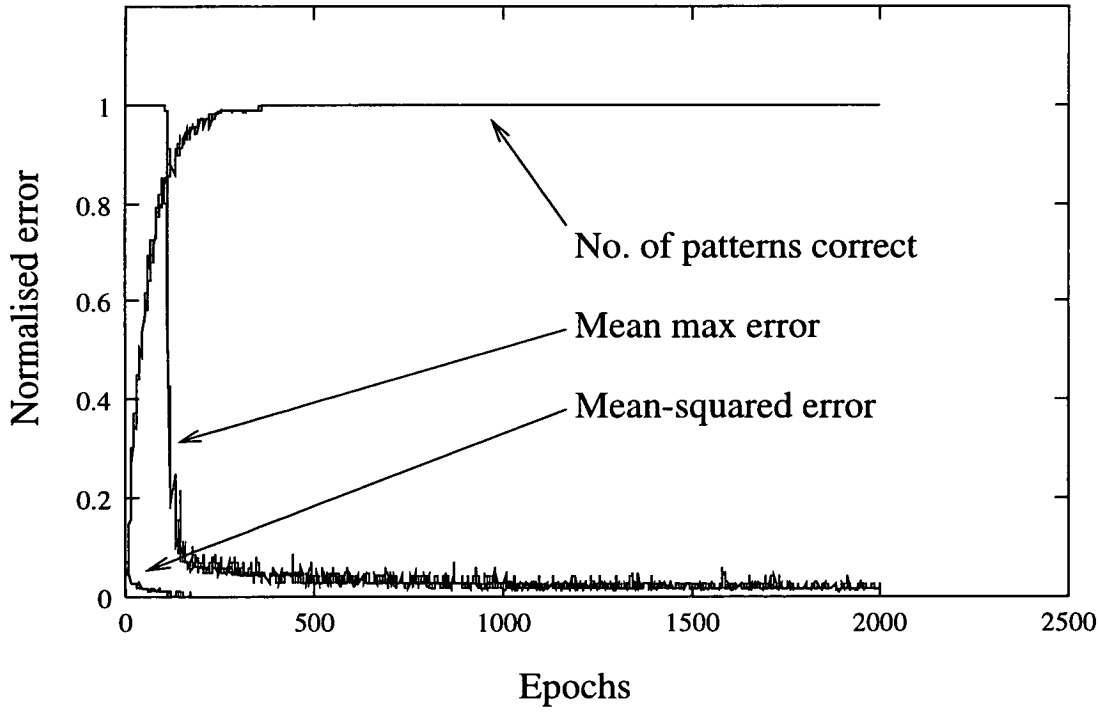


Figure 5–1: *Simulation results using the simplified target-modification algorithm. These results were used as a baseline for comparison with future simulations*

The points to be noted from the graph are that : the value for the number of patterns correctly recognised has been normalised, so that a value of 1 indicates that all patterns have been recognised; the mean-square error is included because it is that error that is being minimised according to the virtual targets algorithm; and the total error is included because of its usefulness in indicating the stability of learning[2].

Further simulations showed that the algorithm would classify all patterns correctly within 750 epochs.

5.3 Changes to the weight-modification algorithm

From the starting point described in the last section, I considered the weight-modification equations. The equations for the hidden- and output-layer weights are shown in Figure 5–2 (which is simply a copy of Figure 4–3).

Of the four terms involved, the three terms other than the gain term are represented as pulses. The first term I considered was the sigmoid-prime term.

5.3.1 Simplifying the sigmoid-prime term

I had already made a simplification of this term, by replacing the original characteristics, which are shaped like a parabola, with a triangular characteristic (see Figure 4–4 on page 66). However, Fahlman had investigated other possibilities of improving learning in back-propagation networks, including shifting the sigmoid-prime's output range from 0.0 $\longrightarrow$ 0.25 to 0.1 $\longrightarrow$ 0.35 (Fahlman, 1988).

---

[2]The definitions of these error-terms, graphed in this and following figures, is as follows. An error is the difference between an actual output-value from any one node and the target-value for that node. The 'mean max error' is the mean of the maximum error for each pattern, that is the largest single error for any particular pattern-presentation, averaged over all patterns. The 'mean-squared error' is the mean square of all output errors for all nodes for all patterns.

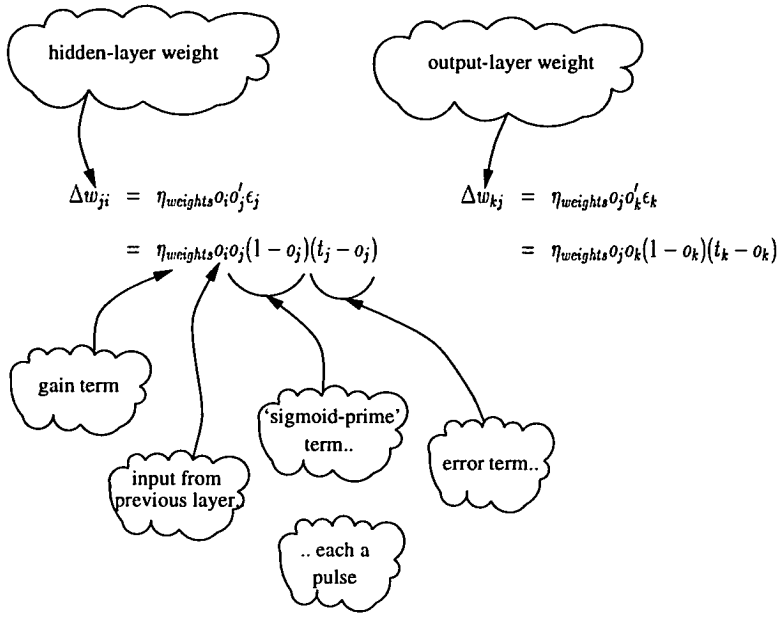


Figure 5–2: *The weight-modification equations for the hidden and output layers.*

His reasoning was as follows. The output of the function is zero when o_{pj} is zero or one, and reaches a maximum of 0.25 when o_{pj} is 0.5. In standard back-propagation, output units tend to get turned off hard during the early stages of learning, and get stuck in the zero state. The more hidden units there are, the more likely output units are to get stuck. As an error is back-propagated through the network, the error is multiplied by the sigmoid prime; even if such an output represents the maximum possible error (if a unit is incorrectly close to zero or one), the shape of the sigmoid-prime graph ensures that only a tiny fraction of the error is passed back to the unit's weights and previous layers. The term heavily discourages learning on nodes that are very **ON** or **OFF**, and attenuates learning on nodes that are midway between the extremes.

After experimentation had shown that Fahlman's recipe of raising the function's output range was as valid for the virtual targets algorithm as for back-propagation, I decided to remove the term altogether, and to scale the gain-term. This change produced a corresponding improvement in the speed of learning, as shown by the graphs in Figure 5–3.

For different values of the gain-term, learning was very quickly successful, in well under 500 epochs instead of around 500 as before. Learning is smooth, as shown by the curve for the total error, until the gain-term is raised to a value of 0.100.

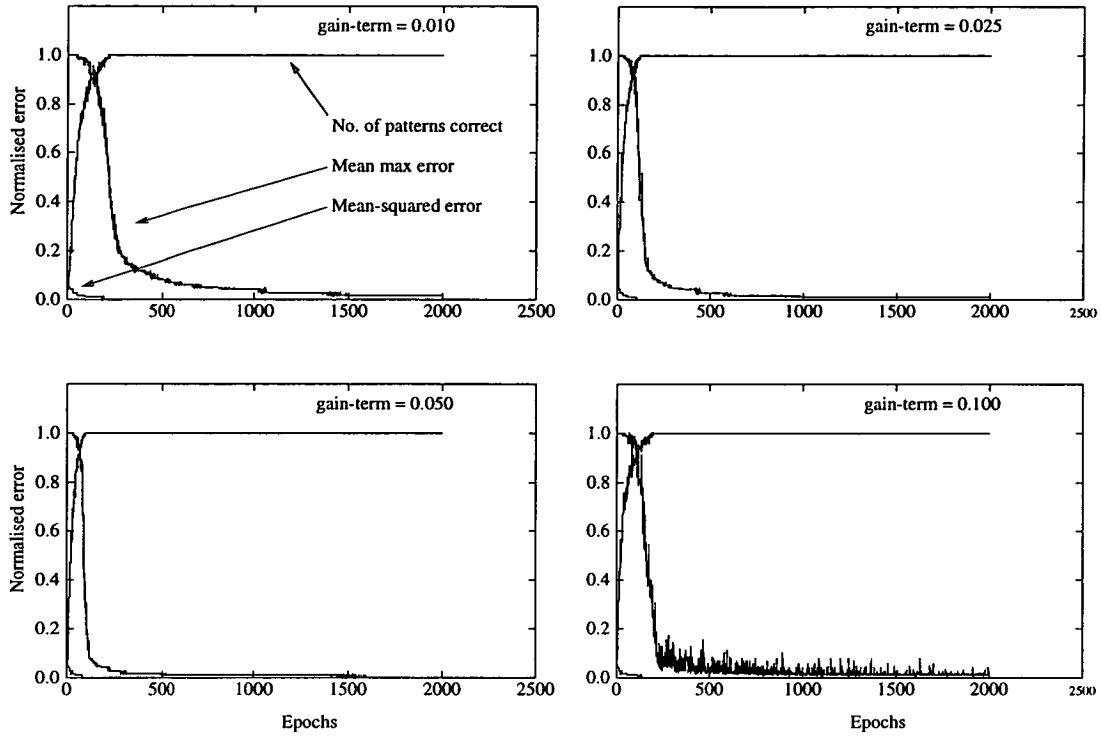


Figure 5–3: *Simulation results with the sigmoid-prime term removed from the algorithm and the gain-term rescaled and given different values.*

I could not, of course, assume with any certainty that a hardware implementation would behave in the same way as the software, but it seemed reasonable to conclude that removal of the sigmoid-prime term was no impediment to learning, and would perhaps enhance it. Far from some simplification being necessary, I could ignore the term completely.

5.3.2 Simplifying the remainder of the weight-modification equation

The weight-modification equation had now been reduced to this form[3] (for the output-layer weights) :

$$\Delta w_{kj} = \eta_{weights} o_j (t_k - o_k)$$

[3]This equation corresponds to the delta rule for the simplest kind of networks.

In other words, successful learning can take place when the change in the weight is proportional only to the error and the input from the previous layer.

This led me to wonder if learning could still be achieved with a simplification that allowed weight-modification in what Shoemaker calls a 'trinary' manner, in other words three types of modification involving a fixed increment, a fixed decrement of equal value, or no-change (Shoemaker et al., 1991). Were learning successful in such circumstances, it would open up the possibility of a weight-modification circuit that only demanded a 'bump' in the right direction.

I therefore experimented with a software function that carried out this simple recipe, with the caveat that the no-change state should apply in circumstances where either the error or the input was so small that it fell below some minimum threshold. If the error was positive, the weight was increased, and if negative it was reduced. My expectation was that, if the idea worked, then its success would depend on the correct step-size, and this did indeed prove a little tricky to define.

The graphs in Figure 5–4 show the results.

There are several problems with these results. As the step-size increases from 0.01 to 0.20, the learning becomes increasingly bumpy. When it is least bumpy, ie when the step-size is very small (0.01), learning is, as one might expect, very slow, taking around 6000 epochs to classify all patterns correctly. On the other hand, even with a step-size of 0.10, classification seemed reasonably stable.

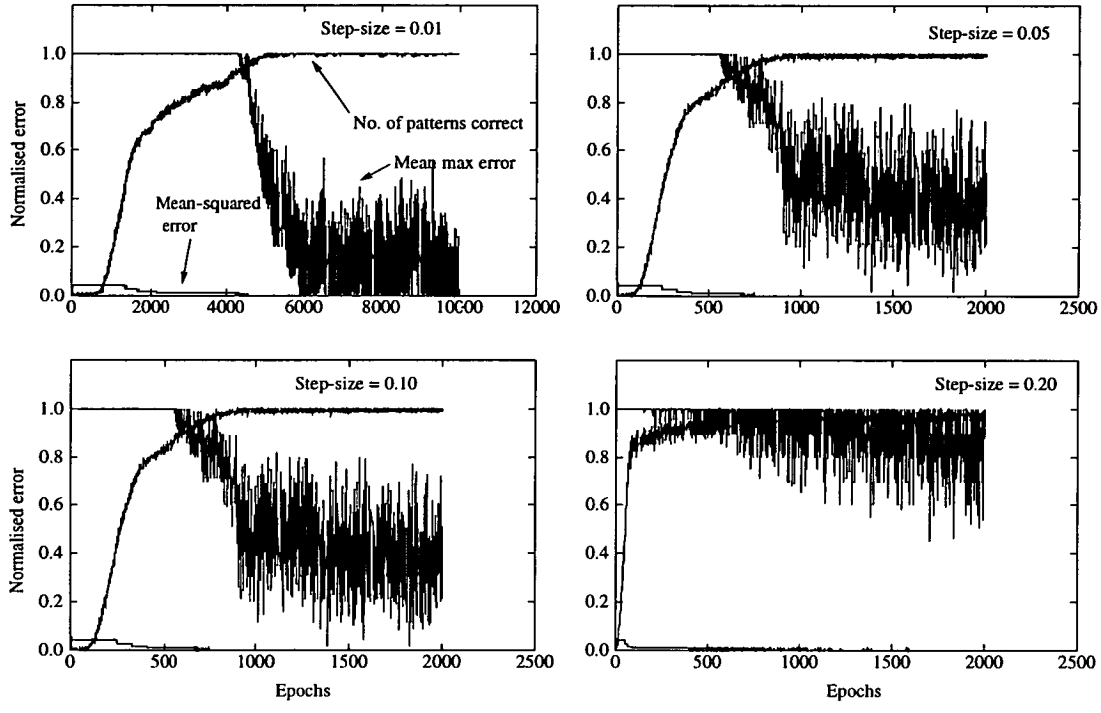


Figure 5–4: *Simulation results using an algorithm where weights are 'bumped' in the correct direction. Note that the range of the x-axis on the first graph is 10.000 epochs, while that on the other three is 2,000.*

5.4 Using the forward-pass circuits efficiently

Calculating a network-layer's output states (what I call the 'forward pass') requires a sum-of-products calculation, and so does the equation for calculating the change to the hidden-layer targets. The equations are shown together in Figure 5–5.

Each is a sum-of-products calculations; assuming, in analogue circuits, the representation of the variables is the same for the input-states and errors (say, pulses), then the same multipliers could be used for both calculations.

The problem is that the equations require that the flow of data through an array of synapses to calculate output states is orthogonal to the direction of flow for hidden-layer targets.

To be clear about this idea, the virtual targets algorithm requires that data should

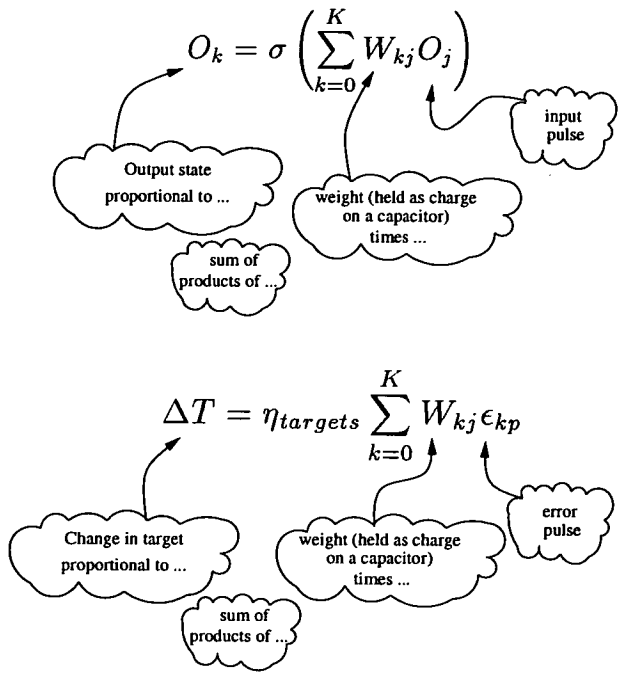
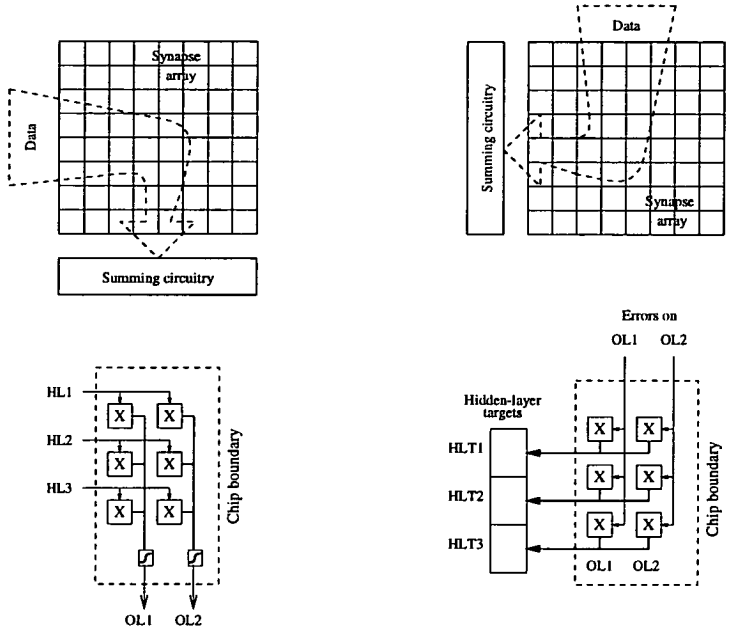


Figure 5–5: *Preferred representations of signals to realise the equations for computation of output states and for changes to the hidden-layer targets.*

pass through the array in two ways. The first way is for the forward pass through each layer, in which input states are weighted, the results are summed and the sigmoid function is applied, to give an output state for each neuron in that layer. As explained in Figure 4–1 on page 61, a forward pass is equivalent to driving inputs along rows in a chip array and summing the results of the weightings on a common node at the foot of the columns. The idea is depicted in Figure 5–6(a).

The second way in which data passes through the array is the backward pass of the error terms in order to compute the hidden-layer targets. For a chip array, this is equivalent to driving signals representing the error-terms down the columns, and summing the results along the rows, as shown in Figure 5–6(b).

If we can find a way of performing both kinds of calculation, on the same multiplier array, the circuit-design can be simplified. A way of doing this is explained in Chapter 6.



(a) Forward pass

(b) Backward ('error') pass

Figure 5–6: *Conceptual representation of the flow of data through the synapse array on a chip for the forward (a) and backward (b) passes.*

5.5 Final alterations to the algorithm

I made final changes to the algorithm at a later stage, when recoding it for the PC on which I carried out circuit tests. The changes involved the removal of a software procedure I called 'jamming the targets'[4]. The algorithm was recoded for the network shown in Figure 5–7, chosen for a very small test of the circuits.

---

[4]This was a means of keeping the relationship between actual hidden-layer outputs and hidden-layer targets fairly close. The procedure checked the level of the largest single error on the output layer : if the error fell close to zero, the actual outputs were copied to the hidden-layer targets. Previous changes, notably constraining values of the hidden-layer targets between 0 and 1, rendered this unnecessary. I would like to thank Andy Myles for many helpful discussions that aided me in making a reordering of events in the algorithm that made it intuitively more sensible

Simulations showed that, if the fixed steps used to increment or decrement weights were too coarse, then the network would not learn. However, within wide limits, any step size finer than the maximum increment with which learning would take place was acceptable; the finer the step size, the longer the network took to train. We will see the importance of this result in Chapter 7.

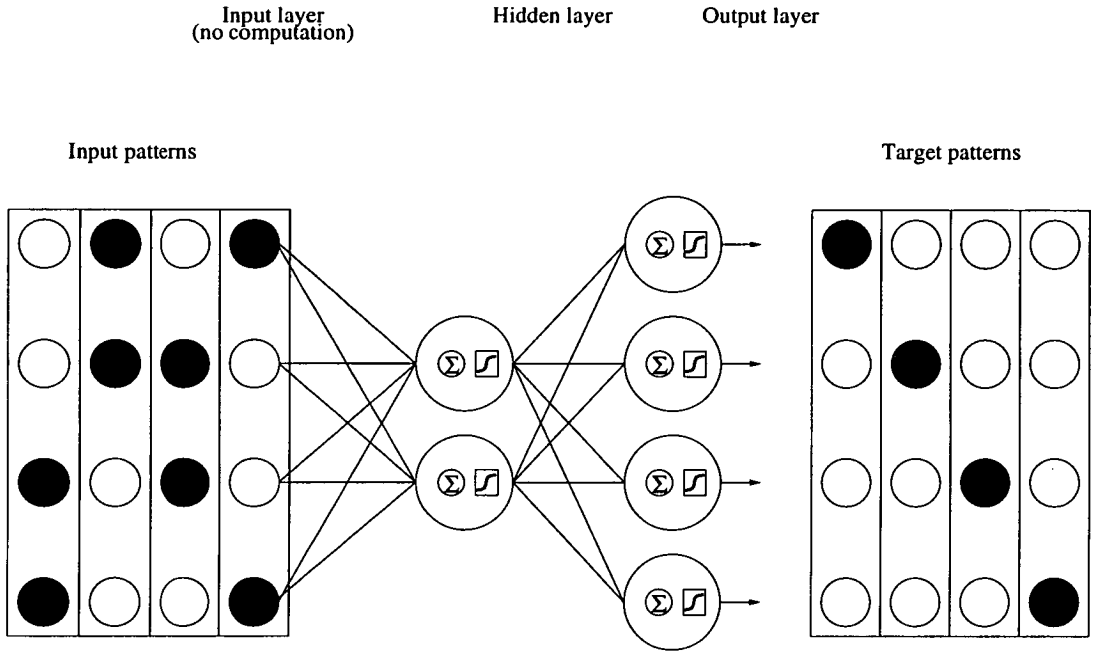


Figure 5–7: *The test network.*

5.6 Conclusions from the software simulations

The idea of simplifying the algorithm to facilitate hardware implementation had been successful, although not unconditionally. We had been able to draw the following conclusions from the work in this chapter :

- *hidden-layer targets* – it would be sufficient, for the success of the algorithm, to modify targets in proportion to the sum-of-products of the weights and errors, so that : $\Delta T \propto \sum W_{kj} \epsilon_{kp}$;

- *sigmoid-prime term* – far from being important for learning, this term could be disregarded altogether;

- *weight modification* – although I could not be certain, I could be hopeful that a simple, fixed-step, increment or decrement in weight-values would be sufficient for learning to take place, provided the error and input from the previous layer were above a threshold, and in a direction dependent on the sign of the error. However, the step size would have to be small.

I turned then to the question of a mechanism for modifying weights and targets.

Chapter 6

Elements of a system for the algorithm

6.1 Introduction

In Section 6.2, I describe my investigation of means to change weights represented by charge on capacitors, and the decision I made on a circuit to carry out this task. In Section 6.3, I explain the scheme I developed for transferring my ideas onto silicon and, in Section 6.4, the design of my second chip.

6.2 A circuit for changing the weights

I considered four possibilities for a circuit to modify weights :

- Schwartz's switched-capacitor weight-modifier
- Transistor substrate pump
- Arima's charge-pump
- the EPSILON voltage-integrator

and finally decided on the last of these. I describe the circuits themselves, and my reasons for my final choice, in this Section.

6.2.1 The Schwartz weight-modifier

One of the ideas included in Table 3–1 in that of modifying weights by a switched-capacitor circuit (Schwartz et al., 1989b, Schwartz et al., 1989a, Schwartz and Samalam, 1990, Schwartz and Samalam, 1991). A schematic diagram of the circuit is shown in Figure 6–1.

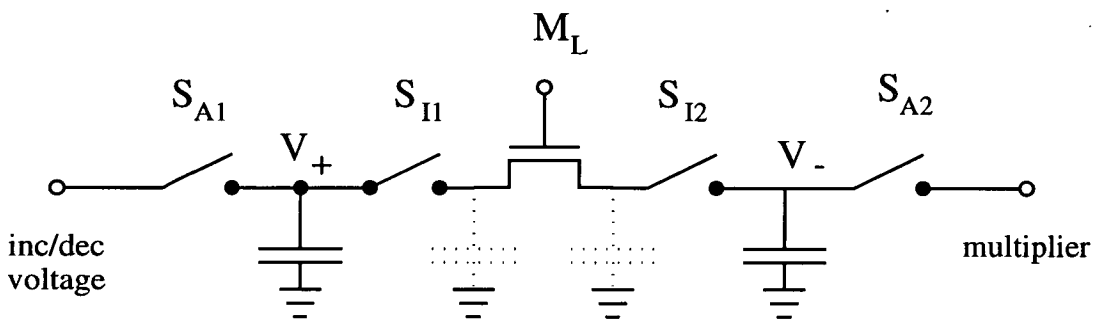


Figure 6–1: *The Schwartz weight-modifying circuit.*

The idea is to exploit charge-injection, a phenomenon normally avoided by designers of switched-capacitor circuits. A pair of capacitors, representing V_+ and V_-, are connected by a string of MOS transistors. S_A are access transistors, to allow the capacitors to be charges to pre-set voltages, and S_I allow the capacitors to be isolated from M_L, a long transistor used to inject charge into the capacitor nodes. Since S_I are minimum size, and so cause little charge-injection into connecting nodes, we can think of them as ideal switches, and consider only the charge-injection process caused by pulsing transistor M_L.

At the start of the process, all switches are open. Using a clocking scheme, S_{I1} is closed and M_L turned **ON** long enough for electrostatic equilibrium to be reached. If S_{I1} is opened, then S_{I2} closed, and M_L turned slowly off, mobile charge in the transistor channel is injected onto V_-, lowering its voltage. The reverse sequence raises its voltage.

The authors of this work claim that very fine adjustments are possible with this scheme.

6.2.2 Transistor substrate pump

Another possibility exploits a phenomenon recognised almost three decades ago (Brugler and Jespers, 1969), and illustrated in Figure 6–2. When pulsing a transistor gate-voltage with a negative pulse, there was an apparent injection of charge across the source-substrate and drain-substrate junctions. If the transistor was unpulsed there was a negative reverse-leakage current. If it was pulsed with negative pulses, there was a positive current, broadly linearly-proportional to pulsing frequency. Perhaps it would be possible to exploit this phenomenon.

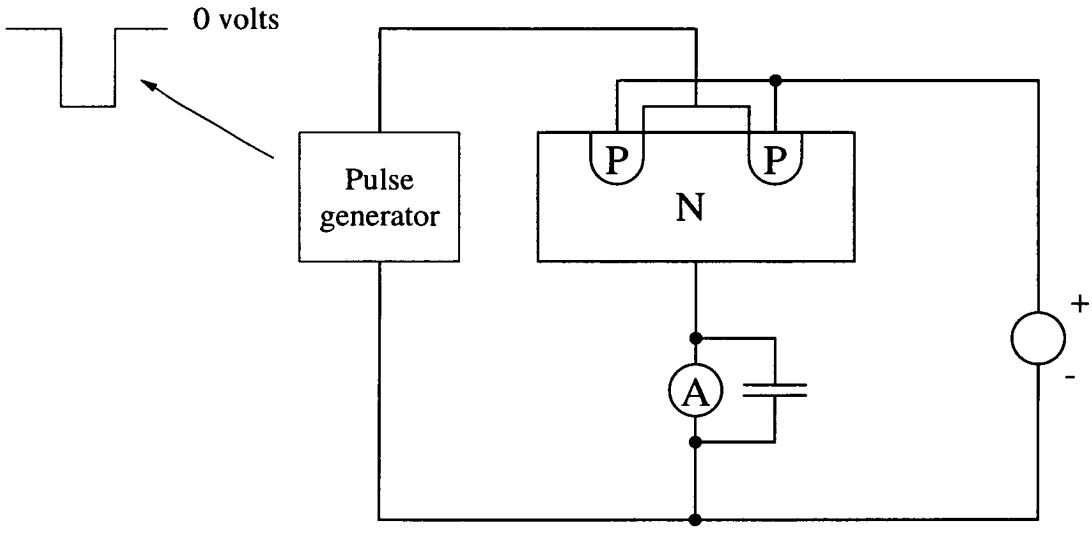


Figure 6–2: *Brugler's experiment on transistor charge-pumping. The voltage source keeps the drain-substrate and source-substrate junctions in reverse bias, and the pulse takes the gate-voltage below the substrate voltage.*

6.2.3 Arima's charge-pump

A circuit diagram of Arima's charge-pump circuit (Arima et al., 1991a), together with a simulation of its characteristics, is shown in Figure 6–3.

The capacitors are realised by the gate-capacitance of transistors, and $5V$ pulses are applied to V_{down} (during $\Delta T1$) or V_{up} (during $\Delta T2$) to charge or discharge the capacitor. The inputs to the pump were stimulated, for these simulations, by 50% duty-cycle wave-forms with an **ON**-time of $1\mu s$, ie 150 pulses during ΔT, giving small increments or decrements of, at most, $20mV$, and often less; this is

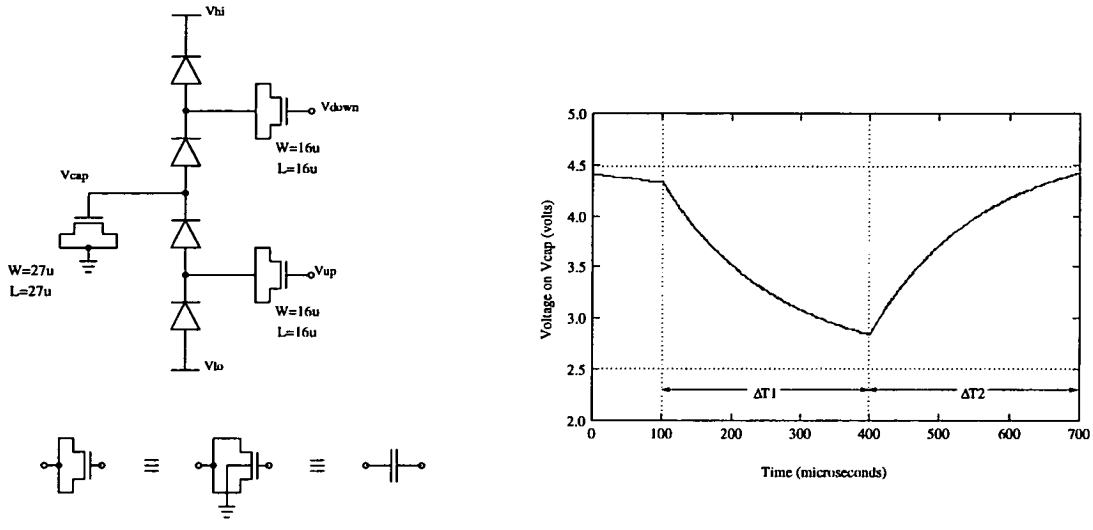


Figure 6–3: *(a) Circuit diagram of a charge-pump; (b) The pump discharging and charging a capacitor.*

much better than the 10% change in value Arima reports (Arima et al., 1991a), of course on a different process. The curves are very non-linear (which is unlikely to be important for learning as long as the change in charge is in the correct direction), but approach asymptotic limits well within the synapse-capacitors' $2.5V \longrightarrow 5.0V$ range. Nevertheless, the idea looked a very good one.

6.2.4 The EPSILON voltage-integrator

I had been considering switched current-sources as a weight-change mechanism. In principle, these are simple (see Figure 6–4(a)), being a means of sourcing current onto, and sinking it from, a storage capacitor; in practice, it can be tricky to make small and consistent changes to the store of charge, because of switching noise which is coupled onto the weight-node and because, as the switches are turned off, charge is injected into the node. However, it occurred to me that the voltage-integrator, designed by Donald Baxter for EPSILON, already provided, in its output stage, something very like the switched current-sources (see Figure 6–4(b)).

This gave me the idea of using the integrator to adjust weights up or down in the manner illustrated in Figure 6–5. The voltage-control switches one of two analogue voltages onto the integrator's input, and so controls whether current

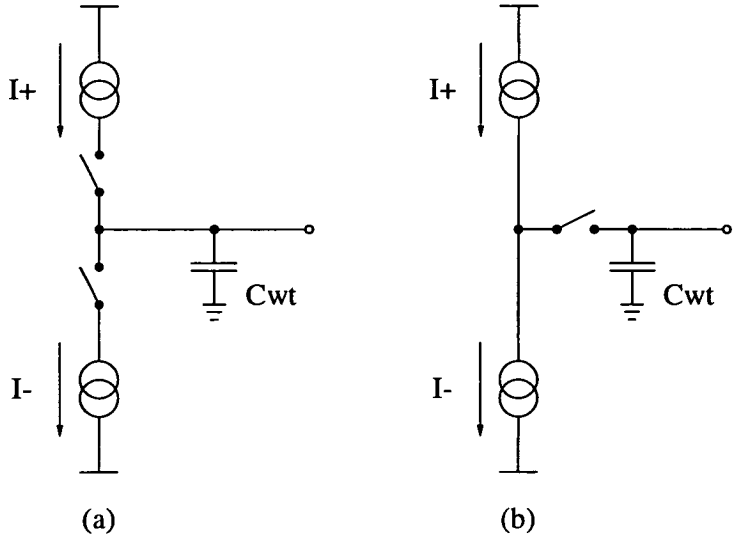


Figure 6–4: *(a) The principle of switched current-sources. (b) The principle as interpreted in the output-stage of EPSILON's voltage-integrator.*

will be sourced onto, or sunk from, the weight capacitor. The two gain controls, $Gain_+$ and $Gain_-$ determine the quantity of charge that is dumped or removed, and the pulse-control switch provides a further control, by making it possible to vary the time the weight capacitor is connected to the source or sink. Some addressing circuitry and a reset switch complete the picture.

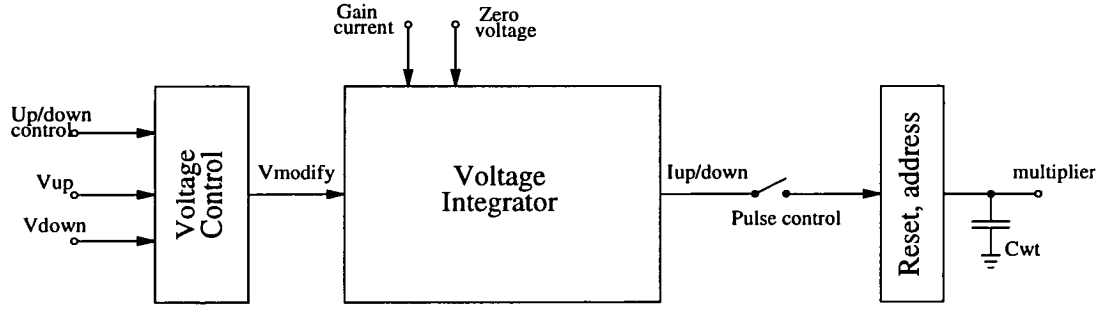


Figure 6–5: *Block-diagram of the weight-modification circuit based on a voltage-integrator.*

The integrator itself is illustrated in greater detail in Figure 6–6. I_{gain} provides the tail current, via a cascode current-mirror, to bias a differential stage. When the differential inputs are equal, transistor M_4 mirrors a current of magnitude

$I_{gain}/2$, to the output stage where transistor M_6 steps it down to a current-source of $I_{gain}/4$.

I_{zero}, of the same magnitude as I_{gain}, is mirrored directly, *via* a cascode current-mirror, to the output stage, where it is stepped down to $I_{gain}/4$. It acts as a current sink which, at stasis, exactly matches the current source, giving an output current of zero. A differential voltage at the inputs disturbs the equilibrium of the system, causing a positive or negative current at the output.

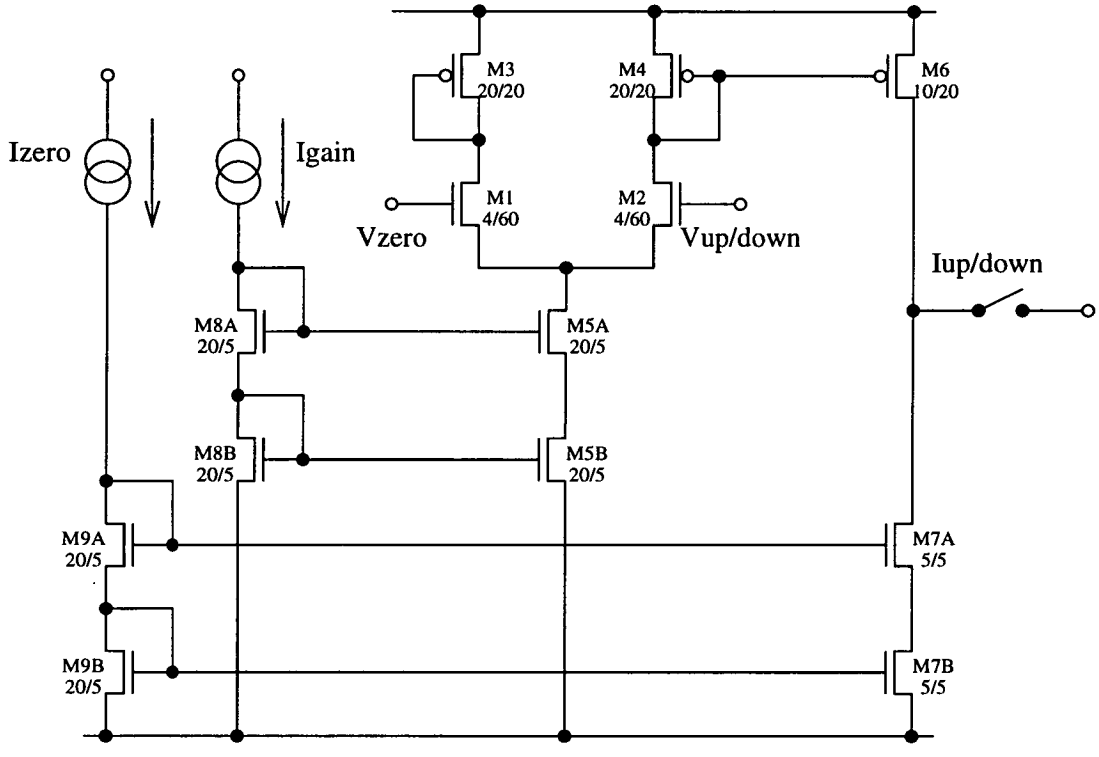


Figure 6–6: *Schematic of the weight-modification circuit based on a voltage-integrator.*

I anticipated that the system would be difficult to set up, since currents and voltages would have to be set quite precisely, and I would have to infer equilibrium over an array of synapses and neurons. On the other hand, there were several ways in which the magnitude of the output current, and hence the step-size of any change to the weights, could be controlled :

- by making the integrator tail-current small;
- by supplying only a small differential voltage at the integrator inputs;

- by varying the on-time of the switch controlling the output current;
- by fabricating a large weight capacitor.

Simulation results are shown in Figure 6–7. The (off-chip) current-sources were set at $1\mu A$. With a V_{zero} of $3.75V$, the up- and down-voltage inputs were set at $4.0V$ and $3.6V$ respectively, and the pulse-control switch was pulsed on a 50% duty-cycle with an **ON**-time of $1\mu S$. In Figure 6–7(a), during the up-cycle, the weight capacitor was raised from its mid-point value of $3.75V$ to $4.88V$, where it saturated as the integrator's output stage was turned off. The step-size during this cycle was never more than $35mV$. For the down-cycle, the weight capacitor was reset to its mid-point value of $3.75V$, and dropped in steps of around $30mV$ to its low point of $2.0V$, and below. In Figure 6–7(b), the weight was set to its minimum of $2.5V$ and nudged up and then down again.

The weight-change process is clearly different in the up-cycle and the down-cycle; in Figure 6–7(a), it takes around 60 steps to rise, and 80 steps to fall, from the mid-point value to the limits. Nevertheless, the circuit does provide small-step weight changes in a predictable way.

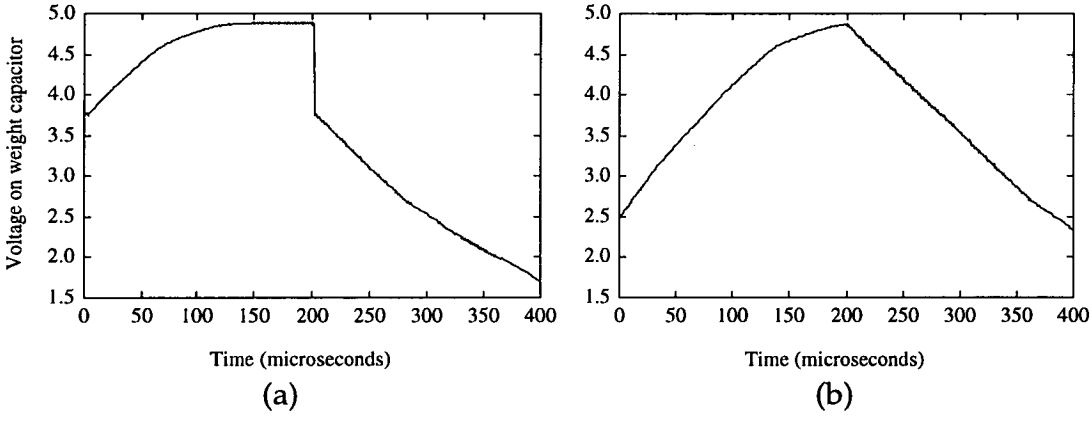


Figure 6–7: *Simulation results of the integrator-based weight-modification circuit.*

Since the integrator was one of my group's existing designs, and since I had insufficient time in which to make tests of the alternatives, I chose the integrator as the means of making changes to the capacitive weights.

6.3 An architecture for a second chip

The various computations that make up the virtual-targets algorithm are quite complex, and the analogue circuits required to approximate the functions even more so. I decided, therefore, that I would have to make some compromises that would enable me both to instantiate as much as possible of the algorithm on a chip, and at the same time test each part of the functionality in case any one, or more, parts did not work.

I developed a scheme for arranging the various modules on a chip, and decided on an architecture that would be within the silicon budget, would test several of the modules, and could, conceivably, implement the whole algorithm if there was time. The plan for a single layer of an MLP network is shown in Figure 6–8. The diagram is conceptual, not a representation of the chip layout. The synapses are arranged in an array, into which there are, conceptually, two sets of inputs. The first set is that of the input patterns (or inputs from a previous layer) which are fed 'in the rows and down the columns' to produce, at the neuron outputs, a set of summed $weight \times state$ calculations. The second set of inputs is the error calculations for the layer, which are fed 'down the columns and out the rows' to produce, at the neuron outputs, a set of summed $weight \times error$ calculations. The neurons involved in these computations are, of course, the same physical neurons in each case.

The errors, and their polarities, are computed from the target states and from the output states from that layer.

The weight-modification circuit changes weights in response to the error signal (if the error is of sufficient magnitude) and the sign of the error (to determine the direction of the weight-change).

A set of targets is, like the synapses, arranged in an array. The target-modification circuit changes the targets in proportion to the sum of product of the errors and weights.

A two-layer network, with both hidden and output layers, could be realised as shown in Figure 6–9. Chip 1 realises the hidden-layer, while chip 2 realises the output layer. Although the hidden-layer neurons exist on chip 1, the hidden-layer

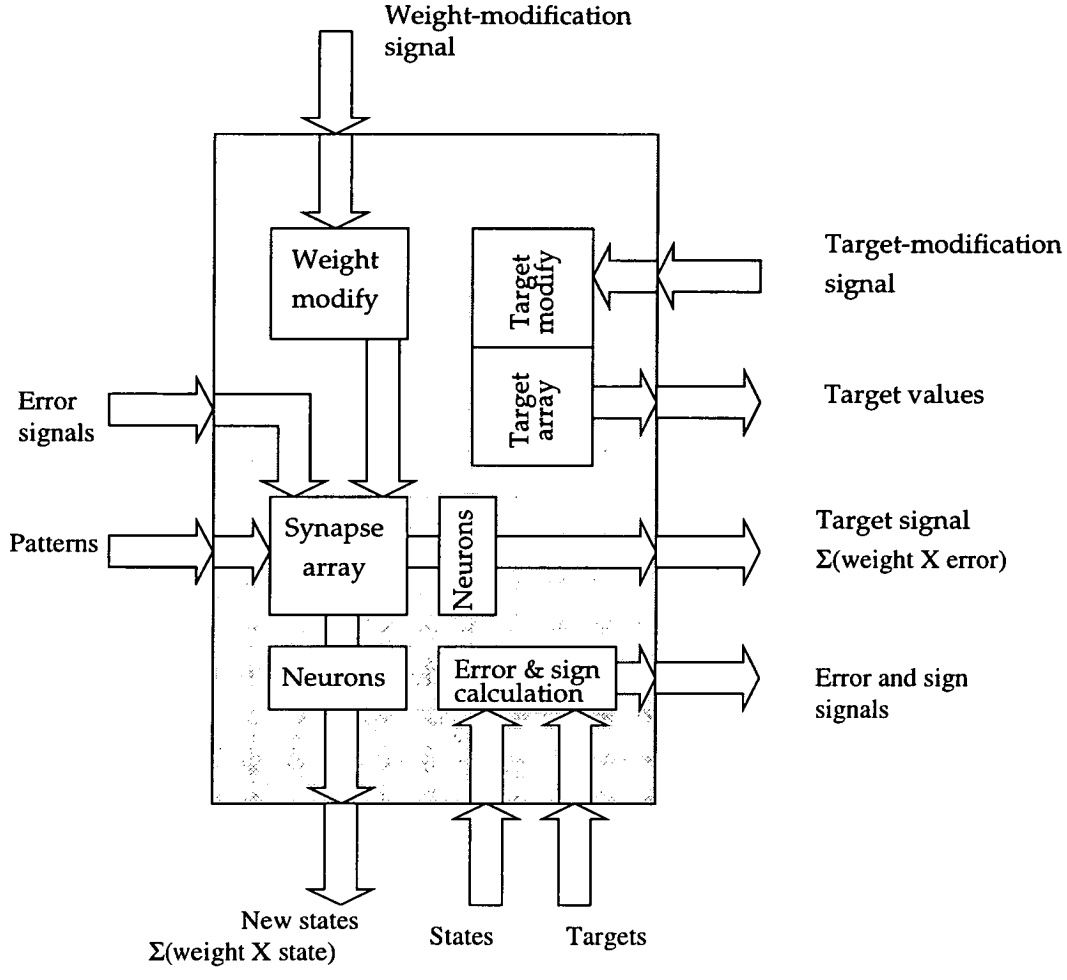


Figure 6–8: *Architecture of a chip embodying a single layer.*

targets are stored on chip 2, because the information required to update them comes from the output layer. Since the two chips would be identical, this means the target store on chip 1 would be redundant.

The rather complex way in which the two chip could be combined is as follows[1].

A pattern is applied ① to the inputs of the synapse array of chip 1, to produce a set of summed *weight* × *state*, hidden-layer outputs ②. These states form the inputs ② to the synapse array of chip 2, which performs a further summed *weight* × *state* computation to produce the output-layer outputs ③.

[1]Numbers presented like so in this description : ③ refer to numbers in Figure 6–9

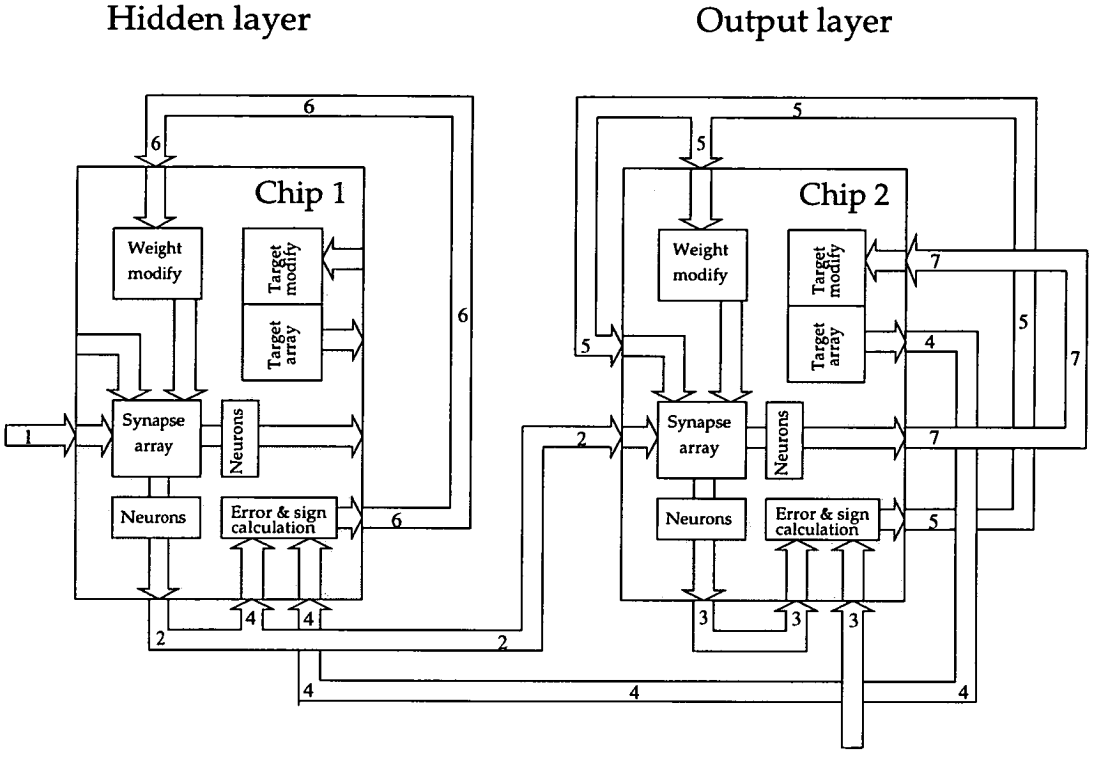


Figure 6–9: *How two chips embodying a single layer can be combined to instantiate a two-layer network.*

The errors for each layer, and their signs, are calculated like so. For the output layer errors, the error circuits on chip 2 use the output-layer states and the predetermined targets for the pattern ③, producing the output-layer errors ⑤. For the hidden-layer errors, the error circuits on chip 1 use the hidden-layer targets on chip2 and the hidden-layer states ④, producing the hidden-layer errors ⑥.

The error signals and their signs for the output layer ⑤ and the hidden layer ⑥ are fed back to modify the weights in each layer. The errors from the output layer are also fed back ⑤ to the synapse array on chip 2, where they provide the inputs to compute a summed *weight* × *error* computation ⑦ from which new target-values can be computed.

6.4 Design of the second chip

To move from the architecture described in Section 6.3 to the design of a chip, I decided on two initial constraints. The weight capacitors should be large, to allow me to make very small increments and decrements, and the number of neurons (and hence the target array) should be as large as possible.

The design was configured for placement in a printed-circuit board a colleague had designed for a bus-based system in which several 'neural modules' could be interconnected[2]. The board carried sockets for the analogue chip and for a digital Xilinx FPGA chip, on which I could design the many digital modules required to drive the analogue circuits.

6.4.1 Forward and backward passes through the array

As explained in Section 5.4, it would be advantageous to use the multiplier both for the forward and for the backward 'error' pass.

A solution requires a redesign of the array with switching to select rows or columns as appropriate. Although the switching is quite complex in practice, the principle is easily described, as shown in Figure 6–10. The same principle can be applied to switching input states into rows or columns (Figure 6–10(a)) as for switching output currents (Figure 6–10(b)).

6.4.2 Architecture

As a result, I was able to accommodate a 64×64 array of synapses, providing weighted inputs to 8 neurons with capacitors of around $7pF$ in value. An overview of the chip layout is shown in Figure 6–11.

---

[2]I wish to express my gratitude to Geoff Jackson for this initial idea, for generously offering to share the equipment he had put so many hours into building, and for his help in fitting my work in with his own.

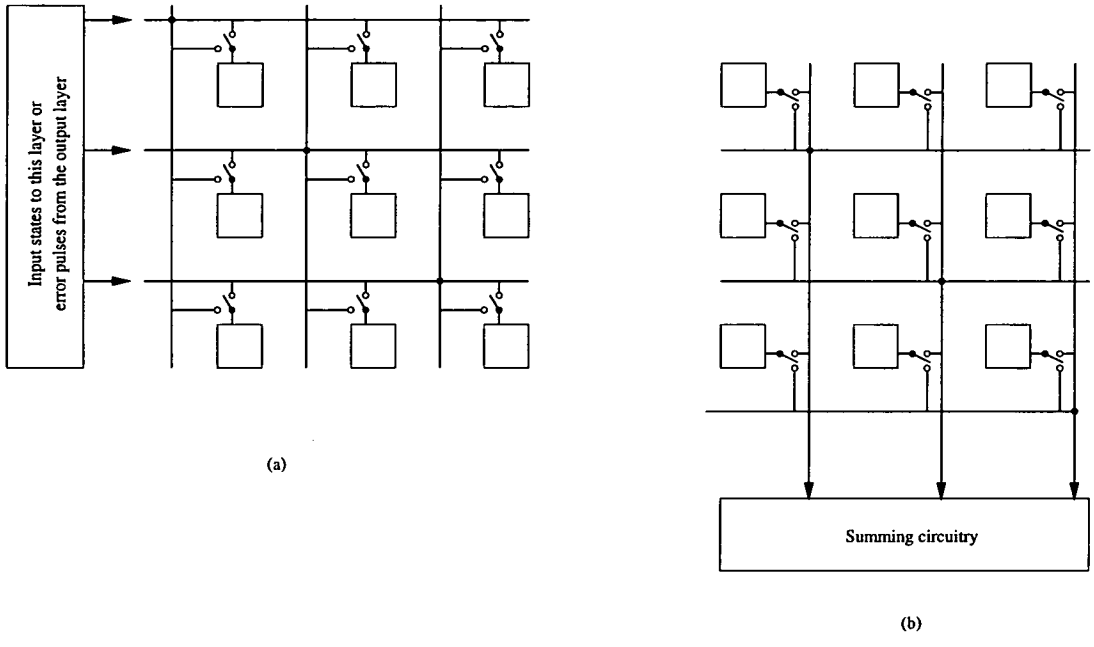


Figure 6–10: *(a) The switching arrangement to switch input states to the layer (for a forward pass) or error pulses (for a backward pass). (b) Switching output currents between rows and columns. All connections are switched simultaneously onto rows or columns, necessitating only one off-chip control line.*

6.5 Summary and conclusions

I carried out the work described in this chapter in light of the simplifications I had made to the virtual-targets algorithm, and focussed on the means by which I would be able to modify weights. I was able to use a circuit, as the basis of a weight-modification scheme, that appeared at least as good as the alternatives available to me, and with the advantage that I had already used it and knew its characteristics.

As a result, I was able to develop an architecture for a chip that would realise a single layer of an MLP network, and yet be capable of being cascaded with another chip to instantiate the complete virtual-targets algorithm.

In the next chapter, I explain how I tested the various modules of the chip, and present my results.

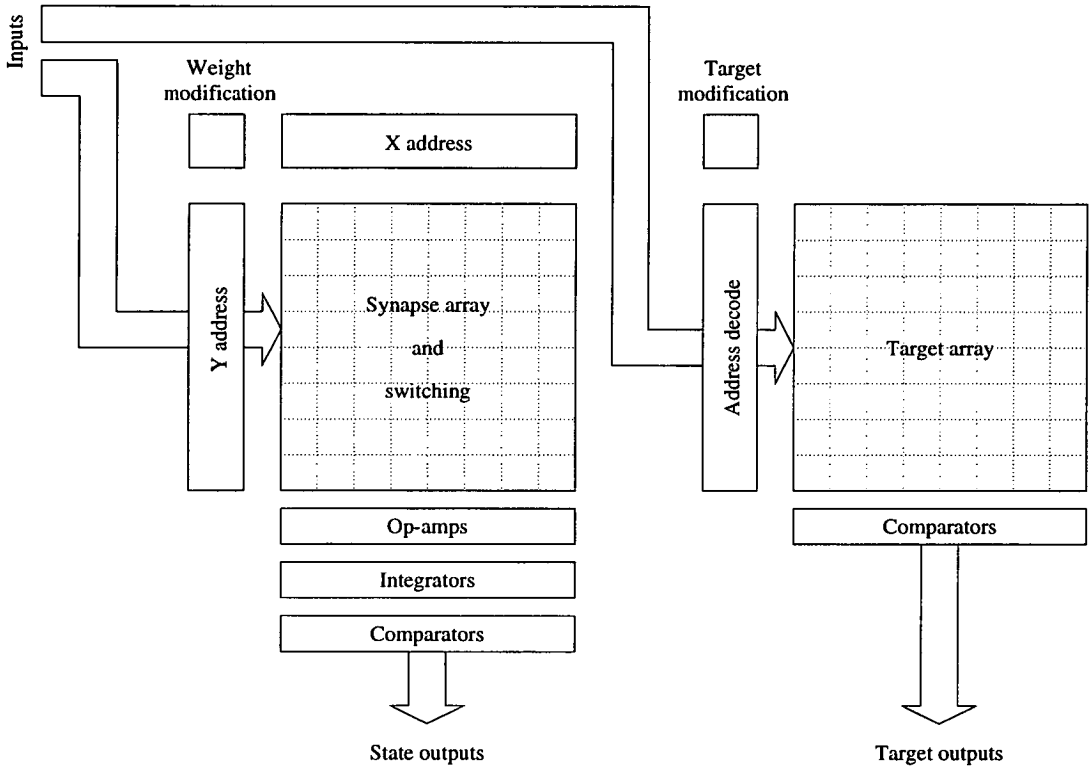


Figure 6–11: *Overview of the layout of the chip.*

Chapter 7

Final tests and assessment

7.1 Introduction

Section 7.2 describes the experimental setup to test the various parts of the chip. Section 7.3 presents results from testing the multiplier and weight-modification circuits. In light of these results, Section 7.4 examines some of the issues relating to achieving learning on chip. Section 7.5 presents the results of trials of a simple problem on the chip. Section 7.6 suggests how these experiments might continue. In Section 7.7, I draw conclusions on the success of my approach.

7.2 Testing the chip

Further details of the design and testing of the second chip are given in Appendix F.

7.3 Results from tests on individual modules

I present here the results from the multiplier and weight-modification circuits.

7.3.1 Multiplier

Results from tests of the multiplier are shown in Figure 7–1.

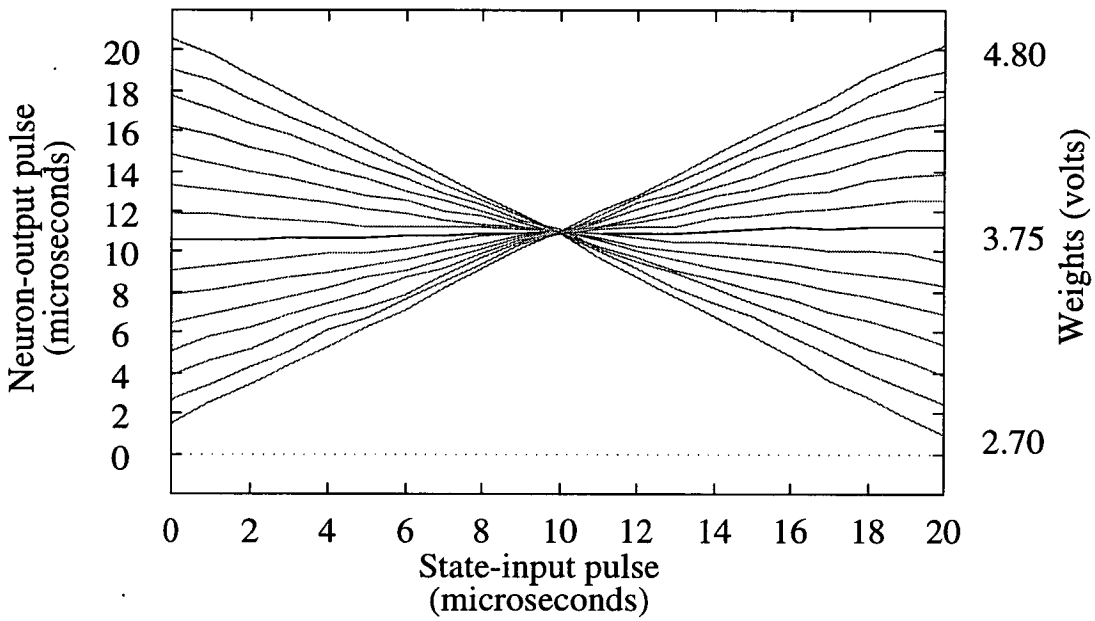


Figure 7–1: *Results from tests of the multiplier. Results represent averages of 50 readings from one column of synapses, produced by storing the same weight at all synapses in the synapse array and applying the same input pulses to each input-row in the array.*

The graph shows the multiplication of an input state, represented as a pulse-width signal, by a weight, represented as charge on a capacitor. The curve for a 'zero' weight of $3.75V$ is almost completely horizontal and very close to the 'zero' output-pulse of $10\mu s$ width. The curves are evenly-spaced, are reasonably linear, and all go through the same zero point very close to the 'zero' input-pulse of $10\mu s$ width. The shape of the curves is also reasonably symmetrical around the x- and y-planes. They are, in short, among the best circuit-results our group has produced.

These results are a considerable improvement on the results from testing my first chip, presented in Figure 4–18 on page 81. The reason for this is, simply, many hours of trials and experimentation to set the system up to produce the most accurate results. The effort to do this is considerable, but the reward is that, once set up, the multipliers behave reliably, and the results are eminently repeatable, with the exception explained shortly.

The variation in measurements over a series of readings is quantified in Figure 7–2.

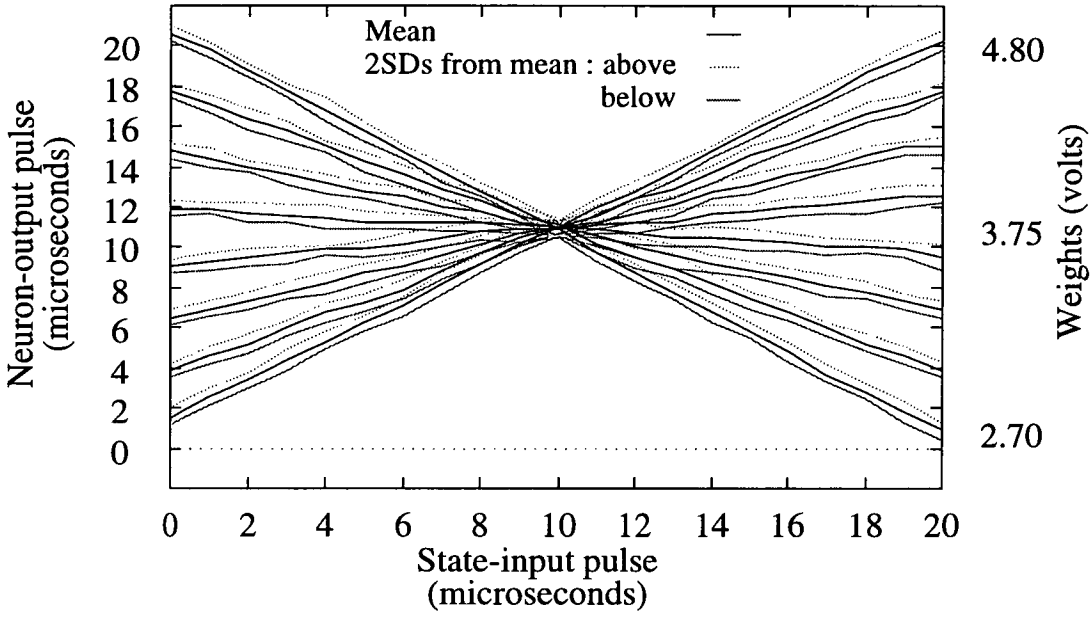


Figure 7–2: *Results from tests of the multiplier. Every alternate curve from the previous figure is repeated here, along with adjacent curves representing two standard deviations on either side of the mean (ie approximately 95% of readings).*

The exception to the repeatablility of results, mentioned above, is that there is a serious problem with offsets at the output-pulse 'zero' point. The consequence is that a 'zero' set for one column of synapses will probably not be replicated accurately at the other columns. The problem manifests itself as a translation of the curves along the y-axis (ie 'up and down' the page in Figures 7–1 and 7–2), all other characteristics of the curves being preserved. This was a known problem discovered by the designers of the Epsilon chip (Hamilton et al., 1993), from which much of the present work was derived.

The designers of EPSILON attributed the offsets to difficulties with the integrator circuit, and my results support this conclusion. As explained in Sections 5.4

and 6.4.1, my chip can accommodate both a forward pass (where state inputs are distributed along the rows of the synapse array, with each synapse in a column contributing to the activation) and an 'error' pass (where error-inputs are distributed down the columns of the array, and each synapse in a row contributes to the activation). Hence, depending on the way the array is switched, the activation represents the contribution of either a complete column or a complete row of synapses. Tests of the multipliers in each configuration produced results which were indistinguishable, strongly suggesting that circuits other than the synapses caused the offsets.

A summary of the configuration of the multiplier and associated circuits is shown in Table 7–1.

Voltage sources (volts)	Multiplier	V_z	4.36
		V_{hiref}	1.50
		V_{loref}	0.50
		V_{bias}	3.10
	Op-amp	V_{clamp}	1.00
	Integrator	V_{integz}	3.76
Current sources (μ amps)	Integrator	$I_{balance}$	10
		I_{tail}	10
	Comparator	$I_{comparator}$	30
Pulse widths (μ seconds)		Zero pulse	10
		Input range	0 — 20
		Output range	0 — 20
Accuracy, defined as : $\left(1-\left[\frac{\max\lvert\xi_{3sd}-\xi_{mean}\rvert}{\xi_{max}-\xi_{min}}\right]\right)\times 100$			95%

Table 7–1: Summary of multiplier configuration

7.3.2 Weight-change circuit

The results from tests of the weight-change circuit are shown in Figure 7–3. The circuit schematic in figure right is a reminder of Figures 6–4 and 6–6 on pages 100 and 101.

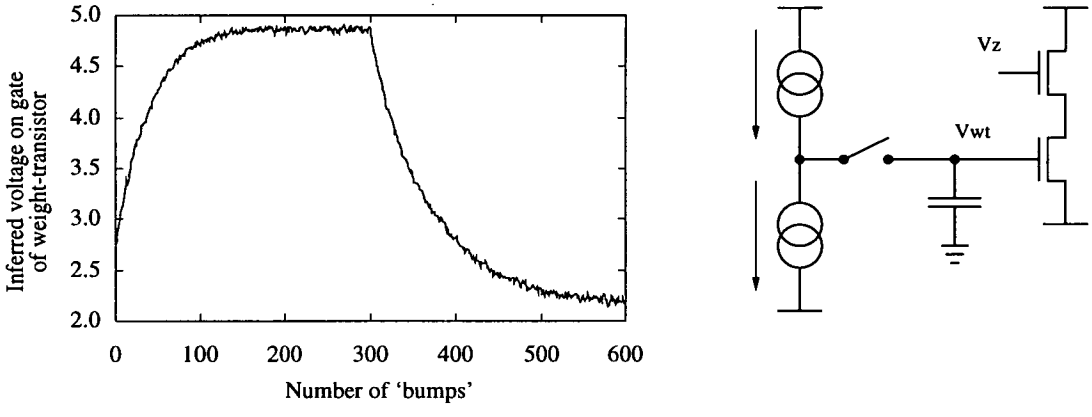


Figure 7–3: *Results from tests of the weight-change circuit.*

The voltage V_{wt} on the gate of the transistor, held as charge on the weight capacitor, cannot be read directly. Instead, the voltage has to be inferred from the results of a *weight* × *state* multiplication. For these results, 8 identical state-pulses were applied to the synapse array, each of whose weights was set to an identical starting-value ($2.70V$). All the weights in the array were then nudged upwards until saturation, at just under $5.00V$, and then nudged back down again. The voltages shown against the y-axis in Figure 7–3 are inferred from an earlier calibration; this related a weight voltage, applied to every synapse in the array, to a pulse-width output from a single neuron at the foot of a column of synapses.

Details of the way in which the weight-modification circuit was set up are shown in Table 7–2.

Voltage sources (volts)	V_{up}	3.78
	V_{down}	3.74
Current sources (μ amps)	$I_{balance}$	5
	I_{tail}	5
Pulse-time (μ seconds)		5
Realistic range (volts)		3.00 → 4.50
No. of steps to move weight over 90% of range		100
Maximum step size (millivolts)		80
Mean step size (millivolts)		20 ≡ 6.2 bits

Table 7–2: Details of how the weight-modification circuit was set up

7.4 Issues raised by the various circuits

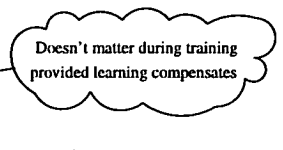
This Section considers the various technical issues involved in light of the functions we can produce for the virtual targets algorithm, and the general characteristics of the circuits to execute them.

7.4.1 Learning rates

As a result of experience like that of using chip-in-the-loop learning described on page 31, it is generally agreed that the process of training a network, by modifying the weights, itself compensates for many of the faults of, and non-uniformities in, analogue circuits. In simulation, the learning rate is usually small, which may slow learning but prevent instability in the number of patterns recognised. The immediate difficulty in placing the learning on-chip is that the learning rate has to be high enough to overwhelm the faults. We can see why this is so by considering some of the factors involved in weight adaptation, weight resolution and weight decay, and the effect of offsets and accuracy.

7.4.2 Weight adaptation

Weight adaptation can be characterised as equation (a) in Figure 7–4.[1]



$$\Delta W = \Delta W_{ideal} + \Delta W_{offsets} + \Delta W_{leakage} + \Delta W_{charge-injection} + \cdots$$

(a)

$$\Delta W = \Delta W_{bump} + \Delta W_{offsets} + \Delta W_{leakage} + \Delta W_{charge-injection} + \cdots$$

(b)

Figure 7–4: *Equations to characterise adaptation of weights.*

ΔW_{ideal} refers to the exact calculation of the 'correct' direction and distance to move the weight towards a network solution. Within the constraints that the distance must not be too little so that learning never takes place, nor too much so that a solution is never found, the exact distance the weight travels does not matter. However, it is crucial that the components of the equation, in concert, push the weight in the correct direction, to increase the weight or reduce it, otherwise the network as a whole will never find a solution.

To some extent, the idea of fixed increments or decrements to the weights, as is used in the simplified version described in this thesis, and shown in equation (b) in Figure 7–4, acts in the same way as varying the learning rate : W_{bump} simply has to be chosen so that the weight moves in the proper direction. Provided the effect of offsets, charge injection and the rest is not too great, the question of the correct increment then reduces to that of the resolution that can be achieved in adjusting the weights. This idea is simple, at least in concept, if not in execution.

---

[1]Equation (a) is from (Annema and Wallinga, 1995)

7.4.3 Weight decay

In general, the problem of weight decay on capacitive weights, due to charge leaking away is ignored in learning circuits on the grounds that continuous learning will compensate. Even a little decay, perhaps as low as 1%, may be enough to prevent learning (Dolenko and Card, 1995), but it should be possible to prevent such decay by re-learning at a fast enough rate. At the worst, the chip could be cooled a few tens of degrees below room temperature to reduce decay-rates to acceptable levels. Bipolar decay (that is, decay from a positive or negative value towards zero) seems to be less of a problem than unipolar decay (that is decay towards the most positive or negative value)(Mundie and Massengill, 1991); in most networks, weight decay is unipolar, leading these authors to suspect that the resolution required for weights in the learning phase is higher than generally believed.

7.4.4 Weight resolution within a fixed range

The numerical values for weights generated in computer simulations have to be matched, in analogue implementations, between the limits of a fixed range. Expressed in terms of digital hardware, the analogue representation is 'fixed point'. The range determines the weights' dynamic range and the number of steps required to move from one of the limits to the other. Ideally, the weights should never exceed their bounds, otherwise weights will be indistinguishable, but they should use as much of the range as possible to exploit the dynamic range.

What does this mean for analogue hardware? Precision is not infinite, being limited ultimately by the charge on a single electron[2]. This is not an easy issue to resolve, but we can make at least a rough estimate of the demands on the hardware by using the following translations. (The translation is commonly expressed in digital terms because of the digital origins of the algorithms.) For an analogue implementation, the equivalent number of bits n for a weight increment ΔW is :

---

[2]Arima *et al* used 0.5pF capacitors in their network. To raise the charge on such a capacitor so that the voltage rises by $1mV$, that is one part in 1000 for a $1V$ range, would require the removal of around 3000 electrons.

$$n = log_2\left(\frac{W_{max} - W_{min}}{\Delta W}\right)$$

the corollary being that the largest permissible increment ΔW on a weight, given a desired equivalent number of bits n is :

$$\Delta W = \frac{W_{max} - W_{min}}{2^n}$$

A reasonable rule-of-thumb is that, to represent weights in a range of $1V$:

$$\begin{aligned} 8\,bits &\equiv 4.00mV \quad (1\,part\,in\,250) \\ 12\,bits &\equiv 0.25mV \quad (1\,part\,in\,4000) \end{aligned}$$

The simulations I carried out generated hidden-layer weights within the range $-10 \rightarrow +10$ and output-layer weights within the range $-20 \rightarrow +20$. How such ranges might be matched to the hardware is illustrated in Figure 7–5.

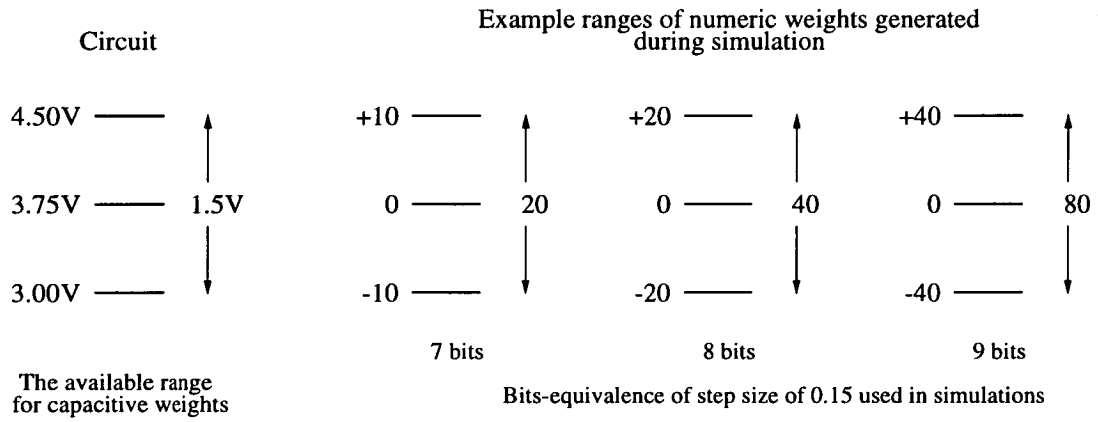


Figure 7–5: *Matching numerical weights generated during simulations to the available circuit range.*

In an extensive series of simulations, Cairns put the resolution required in the learning phase at 12 bits (Cairns, 1995), although other authors suggest 14 bits or more (Mundie and Massengill, 1991).

Clearly, the results of the weight-bumping circuit shown in Figure 7–3 on page 113 fall short of the necessary resolution. As I have already indicated, the design

allows for some improvement. To meet the resolution demands I have outlined, over a $1.5V$ range would require :

$$\begin{array}{rcll} 6.2\ bits & \equiv & 70.00mV & 1\ pt\ in\ 75 \quad (baseline) \\ 8\ bits & \equiv & 5.90mV & 1\ pt\ in\ 250 \quad (\times 3\ improvement) \\ 12\ bits & \equiv & 0.37mV & 1\ pt\ in\ 4000 \quad (\times 53\ improvement) \\ 16\ bits & \equiv & 0.03mV & 1\ pt\ in\ 66000 \quad (\times 900\ improvement) \end{array}$$

To achieve these improvements would be very difficult.

The weight resolution depends, in practice, on the signal range that the synapse can accommodate. I designed my system according to the scheme shown in Table 7–3. Under this scheme, the input range for the weights is 1.5V; with a reconfiguration of the voltage and current supplies to the chip, the input range could probably be increased to 2.5V, as illustrated in Table 7–3.

One reason this issue is not a simple one is that, in addition to questions of offsets and accuracy, to be discussed shortly, noise must be taken into account. Researchers disagree on whether noise is beneficial (Murray and Edwards, 1994) or detrimental (Dolenko and Card, 1993a, Dolenko and Card, 1995) to learning, but since all investigations involve simulation, this is a matter which will probably only be settled empirically.

What solutions are there to the problem of the high weight resolution needed for learning? Lehmann suggests (Lehmann, 1994) that high-resolution weights be held digitally, with obvious consequences for silicon resources if these are on-chip, and for system complexity if they are off-chip; or, alternatively, that 'probabilistic rounding' be used so that, for a weight-change less than the minimum resolution, the weight change is carried out with a certain probability.

7.4.5 Offsets

Some published work (Annema and Wallinga, 1995, Dolenko and Card, 1993a), including that of a colleague (Lehmann, 1994), suggests offsets are a serious problem. The problem exists with offsets on the weights themselves, so that different weights have different zero points, and also on the neurons, where learning can take place even after the output error is zero. Either or both of these offsets

Scheme used in trials		Possible alternative scheme
Weight range (volts)		
Upper bound	4.50	4.75
Zero	3.75	3.50
Lower bound	3.00	2.25
Bits equivalent of weight-change (millivolts)		
8 bits	5.90	9.70
12 bits	0.36	0.61

Table 7–3: *The input range of weights on the system, and the equivalent step size for 8 and 12 bits equivalence*

can make learning impossible. The only solution is some means of cancelling the offsets.

7.4.6 Accuracy

Given neural networks' much-vaunted error tolerance, it is a relief to find that inaccuracies in computation due, for example, to process variations, or to non-linearities or variable gains in multipliers, do not seem to be a serious impediment to learning. Experiments (part-simulation, part-hardware) on an opto-electronic network showed that miscalculations of the error to be back-propagated, and so of the appropriate weight changes, had negligible effects on learning (Frye et al., 1991). Learning also seems to be tolerant of variations in multiplier gains which are fixed at the start of simulation experiments (so replicating fabrication variations), although it seems less tolerant to random variations, and so to noise (Dolenko and Card, 1995).

7.5 Trials

I have run a small number of trials of the algorithm using the scheme shown in Appendix F and the network shown in Figure 5–7.

The chip calculates the sum of the *weight* × *state* products for the hidden layer using 4 input weights and a bias weight for each of 2 neurons.

The PC implements the remainder of the functions, namely :

- the sigmoid function
- learning on the hidden-layer weights and the hidden-layer targets. The PC calculates the step changes to the weights and downloads new values via a weight DAC, onto the chip.
- the complete output layer including error calculations.

7.5.1 Chip-in-the-loop experiment

A first experiment involved generating hidden-layer weights using a simulation run entirely on the PC. The final weights, saved once the algorithm had learned to recognise the four input patterns, were then downloaded onto the chip, and a chip-in-the-loop session was run as explained in the introduction to this Section. The chip-in-the-loop session allowed the weights to adjust themselves to compensate for imperfections on the chip and therefore to produce a new and valid 'solution' to the problem of recognising the four patterns.

The experiment demonstrated that the network solution, as measured by the number of patterns recognised correctly, very quickly became, and remained, perfectly stable over a long number of epochs. The weights on the hidden layer continued to vary during this time, but always in a way that left the solution intact. This shows that, as predicted, the learning loop could compensate for imperfections in the forward-pass computation and for any charge leakage from the weight capacitors.

7.5.2 Weight-range experiment

The second experiment was a repeat of the first, but this time with the weights on the chip set initially close to zero. Although the PC still carried out the learning in the hidden and output layers, this time it had to generate a solution from scratch, rather than modify weights close to a solution.

Learning has proved difficult under these circumstances. The hidden-layer weights fairly quickly tended to gravitate towards the upper limit of their range, making further learning impossible.

However, occasionally, the system would show evidence of learning. Figure 7–6 shows two graphs, one the characteristic graph of a simulation and the other the results of a chip-in-the-loop session. The network has learned to recognise two of the four patterns consistently, albeit over a large number of epochs.

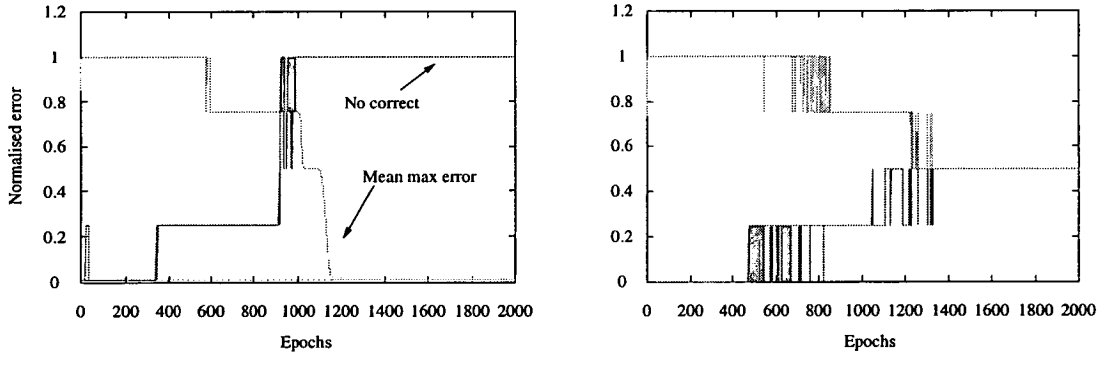


Figure 7–6: *Results of learning trials : (a) simulation; (b) chip.*

7.5.3 Revisiting weight-range issues

Finding suitable values for some of the constants used in the algorithm (for example the threshold level and gain of the sigmoid, and the step sizes for the weights and hidden-layer targets) is difficult even in simulation. There is a balance to be struck between the step sizes for weights and targets that has to be discovered by trial-and-error methods, and that, if chosen incorrectly, can inhibit or prohibit learning. Perhaps it is not surprising that finding suitable values with a chip in the loop is equally difficult.

The second experiment used the PC to modify numeric weights, which were then mapped on to voltage values supplied to the chip using an 8-bit DAC, used over only a part of its full range. A step size of 0.15 in a range of $-10 \rightarrow +10$ for the hidden-layer weights has around a 7-bit equivalence, just within the DAC's own resolution. The fact that the system showed evidence of learning is suggestive that, with a weight-modification circuit capable of greater resolution, learning would be possible, but the only true test would be to try it.

7.6 Further experimentation

More experiments of the kind I have described require to be carried out to establish the conditions under which learning can take place. A plan to do this is as follows :

- Increase the weight range, from the values used in the experiment to the alternative scheme shown in Table 7–3. This would require a complete reconfiguration of the chip and further tests of the multiplier and weight-modification circuits to establish zero points and signal output ranges. This represents a good deal of time-consuming work but would extend the resolution of the system. The system also needs to be redesigned so that digital circuitry captures and stores output pulses from the output neurons. Currently, output pulses are captured on, and readings taken from, a storage oscilloscope. This is extremely convenient from the point of view of observing the signals to monitor progress, and tracking down hardware bugs, but is very slow.

- Attempt to increase the number of steps the weight-modification circuit requires to move a weight between its upper and lower bounds; as explained in Section 7.4.4, a considerable improvement is possible.

- Try to train the weights using the weight-modification circuit, instead of the PC.

7.7 Conclusions

After careful setting-up, the multiplier worked as predicted, in both the forward-pass and error-pass modes, and excellent results were obtained from the system as a whole. There is a problem of offsets on the outputs which was discovered and solved by my colleagues who designed the Epsilon chip, an issue which I have not addressed.

The experimental set-up described here is a good test-bed for further examining issues which are well-understood from simulations but not from experience with real hardware.

The performance of the weight-modification circuit could be improved to produce much smaller step sizes than those demonstrated in the test results (around 4-bits equivalence), probably to 9- or 10-bits equivalence and even beyond. However, further trials would be needed to prove that this improvement would be sufficient to permit learning.

The evidence from published work is very strong that a weight resolution of around 12-bits equivalence is required during the training phase.

Nevertheless, the system did show evidence of learning, even using an 8-bit DAC to convert numerical results from a PC to voltages delivered to the chip.

I have been able to demonstrate that analogue circuits can be developed for implementing every aspect of the virtual targets algorithm, provided that the algorithm is used in its simplified form. Furthermore, the different modules can be integrated to create an entire, two-layer network. The algorithm can be implemented, then, in analogue hardware, provided that hardware is operated at its limits on trivial problems.

Chapter 8

Conclusions

8.1 Introduction

In this last chapter, I present my final conclusions. The chapter also acts as a summary of the work presented in the thesis.

8.2 Issues in on-chip learning

8.2.1 Summary

I have defined the term 'on-chip learning' and compared digital and analogue approaches. There is no single strategy for designing chips with on-chip learning, and consequently the variety of implementations is very large. However, the issues of weight-storage and weight-modification are important for all researchers. I have categorised different approaches in terms of weight- storage, and then looked in detail at five examples which show how other researchers have tackled the issues.

8.2.2 Conclusions

On-chip learning is a technique which offers advantages over other methods of determining an appropriate set of weights, but there is no major application ready-to-hand.

Although analogue hardware has advantages over wholly-digital implementations, the balance of advantages still lies with digital hardware.

The different approaches to on-chip learning use different methods of storing and changing weights. Each has its merits and defects; no one approach seems overwhelmingly better than another.

8.3 Translating the VT algorithm into analogue VLSI circuits

8.3.1 Summary

I have explained the fundamental characteristics of a MLP, feed-forward network, and the virtual targets algorithm in particular. The algorithm has been established as an interesting candidate for translation into analogue VLSI.

It proved possible to design circuits for a number of the functional components of the algorithm, while adapting these functions to my research group's distinctive pulse-stream approach. The circuits comprised, initially, a four-quadrant multiplier, a 'sigmoid-prime' circuit, a sign circuit and a difference circuit.

8.3.2 Conclusions

Test of the circuits, either in simulation on extracted layouts (the difference and sign circuits), or from a fabricated chip (the 'sigmoid-prime circuit' and the multiplier) demonstrate the basic functionality required.

8.4 Simplification of the algorithm : summary

Despite considerable progress on circuit design, the algorithm embodied features that resisted translation into analogue hardware. However, major simplifications were possible, including the abandonment of the 'sigmoid-prime' circuit, and the use of a weight-update algorithm that entailed a very simple computation (increment, decrement or leave-unchanged) and a fixed magnitude of increment or decrement.

8.5 Elements of a system for the algorithm

8.5.1 Summary

After investigation of a number of possible circuits, a weight-change circuit, capable of dumping or removing charge on a capacitor, was adapted for the purpose of incrementing or decrementing weights represented as charge on capacitors. The same design was used for the task of incrementing and decrement the hidden-layer targets, which were represented in the same way as the weights.

By introducing some switching circuitry, controlled by a single control-line off-chip, it proved possible to reduce the rather complex computation for the hidden-

layer targets to an 'error-pass' through the synapse array that would require only scaling to properly represent the target-modification equation.

With these matters settled, I was able to draw up a complete scheme for instantiation of the algorithm, and so design, and have fabricated, a second chip.

8.6 Final results and assessment

8.6.1 Summary

The effort of great care in setting up the chip's voltage and current supplies was rewarded with excellent results from tests of the multiplier. The switching scheme, to allow 'forward' and 'error' passes, worked as predicted. The weight-modification circuit demonstrated the correct functionality, with more work required to achieve a small step-size.

This represented significant progress. There were now circuits to implement every major function of the algorithm, and a scheme to put these functions together into a system.

I examined a number of issues related to various practical aspects of learning on-chip, that showed how heavy were the demands on a weight-modification circuit.

Trials of the chip, tested in a loop with a supporting PC, showed that a network with 'correct' weights would remain stable, but learning was difficult to achieve, even when weights were modified on the PC and down loaded onto the chip.

8.6.2 Conclusions

The performance of the weight-modification circuit could be improved to produce much smaller step-sizes than those demonstrated in the test results (around 4-bits equivalence), probably to 9- or 10-bits equivalence and even beyond.

The evidence from published work is very strong that a weight-resolution of around 12-bits equivalence is required during the training phase. These results are generally from simulations, but might preclude learning on the system I have designed.

Nevertheless, the system did show evidence of learning, even using an 8-bit DAC to convert numerical results from a PC to voltages delivered to the chip. The only means of proving the success of the approach would be to use a suitably-configured weight-modification circuit.

The experimental set-up described here could be developed fairly easily into a test-bed for further examining these issues.

It would be possible to solve some problems evident in my assessment, with more work on the circuits and their configuration in a system. However, it appears there are fundamental problems (offsets and weight-resolution) which may mean back-propagation-like algorithms are not demonstrated to best advantage using analogue VLSI.

Analogue circuits can be developed for implementing every aspect of the virtual targets algorithm, provided that the algorithm is used in its simplified form. Furthermore, the different modules can be integrated to create an entire, two-layer network. The algorithm can be implemented, then, in analogue hardware, provided that hardware is operated at its limits on trivial problems. However, it is clear from the work described here and from the work of other investigators that to consider a hardware implementation of such an algorithm for real-world problems, or even for more complex artificial problems, is simply unrealistic.

8.7 The use of the virtual targets algorithm : conclusions

The algorithm in the form described in this thesis offers several advantages over standard back-propagation, which have been exploited in the work described here to produce functional analogue circuits. The results described in Chapter 7 are encouraging, in that learning may well be possible when attempted with an admittedly rather trivial problem.

There remain some questions that remain to be answered on whether the virtual targets algorithm represents a sufficient advance on standard back-propagation to make it a successful alternative for on-chip learning.

One concerns the introduction of the hidden-layer targets as the price of simplifications in other aspects of the algorithm. As the algorithm stands currently, a set of targets, equivalent to the number of hidden-layer nodes, is required for every pattern in the input set. For problems other than small ones, the necessary storage might prove a burden. Furthermore, the most complex component of back-propagation is the computation to distribute errors, measured at the output, backwards through the network, a computation which is neither removed nor simplified by using virtual targets. The consequent benefits are the two originally identified (and described in Section 2.5.4) as a weight-update rule requiring only local information, and the rule's application equally to the hidden and output layers. A third advantage to emerge with the design of the scheme to implement the entire algorithm (and described in Section 6–8) is that the information to be passed backwards between the layers is reduced to an error term, represented by a pulse. It is of course a matter of judgement to say whether these advantages outweigh the disadvantages.

Another question relates to the removal of the sigmoid-prime term from the equations. This effected a major simplification. However, this simplification would also apply to standard back-propagation. Furthermore, I have made no assessment of the disadvantages of the term's removal on, for example, more difficult problems, or on the ability of the network to generalise its representation of input patterns to other patterns of a similar class that have not previously been presented to it.

A third concerns the introduction of fixed-step changes to the weights. This also effected a twofold simplification : the calculation of weight-changes was rendered much-less complex; and consequently the circuit to realise the step-changes became much simpler. Perhaps this simplification would apply with equal success to standard back-propagation. Even different step-sizes for the hidden and the output layers would be a small price to pay for removing the need for storage for the hidden-layer targets.

8.8 Algorithms and analogue VLSI

8.8.1 Summary

A great deal of excellent work has been carried out on building analogue, digital and hybrid circuits and systems to implement ANNs. A huge strength of this work has been its interdisciplinary nature, spanning as it does fields as varied as statistical mechanics, analogue circuit-design and neurobiology.

There is now a very good understanding of the demands that mathematical models make on analogue circuits, and of the difficulties which render many of these models unsuitable for analogue VLSI.

8.8.2 Conclusions

There has been an over-emphasis on translating mathematical models into analogue VLSI. These models, many of which have very interesting properties, are easy to explore in conventional-computer simulation, but difficult to translate into analogue form. Furthermore, the motivation for the translation into hardware, and into analogue hardware in particular, is rarely clear, since most digital implementations show considerable advantages over their analogue counterparts.

8.9 Artificial neural networks and analogue VLSI : conclusions

Investigations into the use of analogue VLSI as a means of implementing ANNs is at a cross-roads. Many researchers, rather naively, expected that, while understanding little of function in real brains, they would be able to build machines with human characteristics, albeit simple ones. They rested their hopes on several ideas, which have turned out to be unreliable. One was that 'massive parallelism', in itself, using large arrays of identical, simple processors, was enough to realise complex functions. Another was that, for reasons of space, speed and cost, hardware versions of simulated networks would have a ready application in preference to computer simulations.

From an engineering perspective, I see ANNs developing in three directions. One is as a branch of statistics for applications such as pattern-recognition. ANN researchers may find this rather a harsh environment to work in since their research is going to be judged against decades of intensive statistical research, instead of as a, supposedly entirely new, paradigm of parallel processing.

The second direction is in hardware development. For most applications, solutions using only software will probably be desirable for their flexibility. For the small number of instances where hardware versions of networks (although no major one has emerged as yet), I consider digital hardware to be the likely choice, since digital implementations can generally match analogue speeds, and they retain the advantages of flexibility, and of precision and accuracy. Analogue electronics, it seems to me, will achieve only a slight foothold in this market where, for example, there are highly particular reasons for using it, such as the low-power application being investigated by Jabri (Jabri et al., 1993).

A third direction is in modelling brain function, either for applications or to illuminate brain processes. If the hope is applications, then researchers must make a serious reassessment of the likely outcomes. If the hope is to illuminate brain function, then I see a serious role for analogue electronics as an investigative tool, along with other tools such as simulation and the modern, non-invasive techniques such as positron emission tomography and magnetic resonance imaging. The work of Shoemaker and Elias, which I described in Section 3.6.1, in modelling low-level

neural function and producing interesting results, demonstrates what might be achieved. Because of the emphasis on mathematical models and on real-world applications for these models, I believe this area is very under-exploited.

Appendix A

Intelligence and learning in people and machines

A.1 Neural networks and the brain

In the past, many authors of works on ANNs referred to the similarities between neural computation in a machine and the structures and functions of the brain. Some drew a quite specific analogy between the two, referring to *"brainlike devices"* (Caudill and Butler, 1990), *"brain style computation"* and *" brainlike systems"* (Rumelhart, 1990); or arguing that *"neural modelers currently start at the lowest level, building networks of model neurons and synapses and expecting intelligent behavior to emerge from the aggregation of various neural forms and knowledge learning"* (Alspector, 1989). Others were more circumspect : *"The belief that there are common quantitative foundations for both brain science and artificial intelligence has come and gone and come again"* (Levine, 1991); *"The approach of neural computing is to capture the guiding principles that underlie the brain's solution to [parallel] problems"* (Beale and Jackson, 1990); *"It is arguable that 'neural' should be purged from the vocabulary of this field – perhaps* Network Computation *would [be] more accurate ... "* (Hertz et al., 1991). All agree, however, that ANNs are in some way inspired by the brain. As the quotations in Appendix B indicate, some investigators even liken neural network functions to the highest levels of brain function.

These considerations led me into an exploration of ideas about learning and thinking in people and machines, and this chapter lays out my conclusions. One difficulty is that neural network researchers now work in so many disciplines, with such different motivations and viewpoints, that the common language they use sometimes obscures rather than illuminates the work they are doing. Of course, any researcher worth his salt has an interest, however casual, in all aspects of neural network research and its applications; this makes it all the more important to place one's work in context.

The following sections consider the main schools of thought in the world of machine intelligence and ask why the belief that machines can act in an intelligent way is so persistent. I look at objections to the whole idea of machine intelligence and their validity. I examine the need for engineers to consider notions of consciousness and, as a consequence, suggest what neurobiologists and those engaged in computational intelligence have to offer each other. I relate this idea of inter-disciplinary research to the particular field of learning. Finally, I propose a common-sense stance for engineers to take on how they can use their skills to contribute to research in machine intelligence.

A.2 The aims of ANN research

The early hopes for ANN research, that the paradigm would provide us with radical new insights into human functions, leading us in their turn into new algorithms with stunning applications, have not been realised. As a consequence, many researchers are disillusioned with the idea of parallels between ANNs and the brain. In recent years, neural techniques have come to be seen as little more than another statistical network in the toolbox. One way of looking at this is to say the field has matured and become more realistic. Another way to think of it is that the original biological inspiration has served its purpose and might as well be abandoned. I believe this is a mistake, and, in my final conclusions, I suggest that ANNs can provide their most powerful justification in modelling brain function, provided of course we keep a realistic perspective on what we can achieve. I explain the reasons for my beliefs in the following sections.

A.3 Viewpoints on intelligence and machines

What are the main schools of thought on machine intelligence? Enquiries into the potential of computing machines have spawned a huge literature, with many shades of opinion represented but, at the risk of caricature, we can distinguish three main viewpoints.

A.3.1 'Strong AI'

The first is generally called 'strong AI' (artificial intelligence). The idea can be summarised as follows. All thinking is computational, the sort of rule-bound and algorithmic thinking done by computers. The feeling (perhaps the illusion) that we are conscious is simply the outcome of the computation, although we do not yet fully understand the connection between the two. This viewpoint is exemplified by the work of three past giants in the history of computer science, Turing, Newell and Simon (McCorduck, 1979). Turing formalised the idea of computation, established the formal properties of symbol-manipulation, and showed that any problem, if sufficiently specified, could be solved computationally on his universal machine. He suggested that such a machine, by its formal structure, could perhaps emulate the mind. Even more importantly, he established the basis of a science of function divorced from structure; in other words, the substrate of thought — valves and tubes, silicon or brain — is irrelevant to the functions that can be carried out. Following Turing's lead, Newell and Simon argued that manipulation of symbols is the essence of intelligence, and hence that a symbol-manipulating machine, including one with a Von Neumann architecture, could exhibit intelligence.

This belief in the potential of computers to be intelligent led to an explosion of effort in so-called 'artificial intelligence', using symbol-manipulation in rule-based systems to emulate everything from medical diagnosis to scientific creativity. The ideas of these original thinkers have spread into many fields such as philosophy, psychology, cognitive science, computational linguistics and engineering.

A.3.2 'Weak AI'

The second position, 'weak AI', holds that conscious awareness is a unique feature of the brain (perhaps only truly the brain of man), and cannot be found in machines. However, computing machines can simulate consciousness and, in practice, we cannot know whether we are dealing with a machine or a human mind. A modern exponent of this view is Edelman (Edelman, 1994), whom I mention in greater detail in Section A.7 below. On this view, machines can, conceivably, perform any human function of perception, analysis or action without necessarily developing 'understanding' in the human sense.

A.3.3 Computational intelligence

The third view, exemplified by (Penrose, 1995), and to which I adhere, is that of computational intelligence, which proposes to avoid concepts of mind and understanding in the practical pursuit of 'machine intelligence'. Like the proponents of 'weak AI', we believe that conscious awareness is a property of the brain, that can one day be understood. However, consciousness and understanding are something entirely other than computational thinking. On this view, we will be able to build machines that can, with varying degrees of success, perform human functions, but rarely, if at all, in the way that people do. We will never be able to simulate human consciousness, nor understanding, and we will always be able to tell, ultimately, whether we are dealing with a mind or a machine.

In the next section I explain why the tendency to associate human and machine intelligence is so strong, and so misleading, and justify my belief in the computational intelligence viewpoint.

A.4 The Forces Behind Machine Intelligence

Proponents of the strong-AI and weak-AI viewpoints make much of the parallels between human intelligence on the one hand, and computational approaches to human functions (such as speech or image recognition, or expert systems), on the other. They argue, justifiably, that some machines can behave in a human-like manner, and that our understanding of computational approaches has assisted

our understanding of the workings of the brain, However, I believe our tendency to see these parallels owes less to the success of computational approaches in mimicking human function than to two deep-seated cultural factors, firstly our belief that man is a complex machine, and secondly our drive to automate. In my view, these cultural forces lead us to make more of the parallels between people and machines than is justified by the evidence.

A.4.1 Man as machine

We can trace the modern view of man as a machine back to the work of Descartes (1596-1650), who viewed all material beings, including people, as being ruled by the same mechanical laws. To read Descartes' practical science (Descartes, 1972), along with the works of contemporaries such as Harvey on the circulation of the blood (Harvey, 1628), for all their errors of fact and flavour of vitalism, gives one a real sense of the powerful forces that their new scientific method had unleashed. For the first time, these works demonstrated, in a methodical and detailed way, a means of understanding the operation of the body in terms of mechanics and hydraulics, ideas that were well understood from their application in machines. Furthermore, scientists could test their ideas by measurement; for example, Harvey was able to show that the weight of the volume of blood pumped by the heart in an hour exceeded that contained in the whole body, and so must circulate rather than being continuously created and destroyed. It is difficult to overstate the influence of this mechanical view of the body, because it has indirectly stimulated so much of modern medical scientific technique such as blood transfusions, the fitting of artificial limbs and the replacement of complete organs. We take it for granted that our bodies are like cars, able for much of our lives to survive on repairs or replacement parts until some terminal disintegration destines us for the scrap heap. No doubt, at stages during this steady advance, there were areas that it was believed science would never conquer, and yet now we believe even our genes can be engineered from one species to another.

A.4.2 The drive for automation

The second force at work that encourages our belief in the similarities between people and machines is the quest to automate. The interplay between economic pressures to replace people by machines and the increasing formalisation of tasks

at work has stimulated the automation of human abilities. Requiring people to work with machines encourages work practices that are algorithmic in nature, and this in turn makes it more likely that the work can be automated. Integrating computer technology into the office makes this as true of intellectual tasks as of manual ones.

Descartes viewed the mind as being of a fundamentally different nature to the body. The idea that machines might also replicate brain functions is, therefore, much more recent. However, the belief that we could, at least in principle, reduce every aspect of man's behaviour, including his mental abilities, to a system of rules, certainly predates the electronic computer. Here is the management thinker, Frederick Taylor, explaining in 1912 his principles of so-called scientific management, which heavily influenced Henry Ford's production-line system :

> '[One of the duties of management] ... is the deliberate gathering in on the part of those on the management's side of all of the great mass of traditional knowledge, which in the past has been in the heads of the workmen, and in the physical skill and knack of the workman, which he has acquired though years of experience. The duty of gathering in of all this great mass of traditional knowledge and then recording it, tabulating it and, in many cases, finally reducing it to laws, rules and even to mathematical formulae, is voluntarily assumed by the scientific managers.'
>
> (Taylor, 1947)[1]

A.5 Objections to the Idea of Truly Intelligent machines

What are the objections to the idea of truly intelligent machines? Clearly, if we can design a machine that can manipulate and instantiate the rules and mathematical formulae referred to by Taylor, then we can easily imagine a machine that can behave just like a person; and if the machine behaves like a person, might

---

[1] p40

the two not then be indistinguishable? And might we not then be justified in claiming the machine as "intelligent"?

Without pretending that the answer to this question is straightforward, I assert that it is "no". The most trenchant critic (and the most entertaining, in a very entertaining field) is Hubert Dreyfus (Dreyfus, 1992, Dreyfus and Dreyfus, 1988), whose objections are threefold. Firstly, promising beginnings in machine intelligence are consistently exaggerated into a golden future that never materialises. Secondly, human behaviour cannot be replaced by rules, because people do not follow rules, except in very constrained circumstances, in their everyday lives. Thirdly, even if rules could be designed to replicate some element of human behaviour, some circumstance would inevitably arise requiring yet another, as yet unspecified, set of rules, leading to an infinite regress. The first of these objections is conceded by most investigators. On the second and third, Dreyfus has a great deal of support form fields as varied as philosophy (Polanyi, 1958), psychology (Suchman, 1987) and knowledge systems (Collins, 1990).

Of the many writers in this area, Searle takes the most sensible approach. Another severe critic of the 'strong-AI' viewpoint, he argues (Searle, 1980, Searle, 1987) that the problem is that machines lack 'intentionality', because mental states are "directed at or about objects and states of affairs in the world". Mind and intentionality are properties of neural systems — neurophysiological events do not cause mental events, which are a feature of neurophysiological systems with certain properties.

What, then, is different about ANNs? The argument that the brain functions by computation, that it develops and learns by minimising some kind of cost function, and that ANNs are useful models or hypothesis-generators which illuminate real brain processes, can be very persuasive (Churchland and Sejnowski, 1992). At the very least, connectionists[2] have avoided the trap of 'strong AI' enthusiasts that we can circumvent the question of biological phenomena altogether by instantiating the brain in computers.

My own feeling is that connectionism has fallen short so far on a number of counts. The first is that, as Dreyfus argues about conventional AI, the field has made claims that it has failed to fulfil, and claims, such as many of those in

[2]The term is often used to describe those who study ANNs

Appendix B, that it is most unlikely to fulfill. Secondly, Searle is surely right in saying that people are not passive receptors of data but are agents with purposive behaviour. Thirdly, many connectionists make little more than a token gesture towards biological networks, while expecting human-like functions to emerge from their efforts. Lastly, we have confused the ability to replicate basic human abilities, for example pattern-discrimination, with the transcendent ability to recognise an object for what it is; we learn and remember, in large part, through our ability to ascribe meaning to events, an ability which the designer supplies to ANNs by interpreting clustering in unsupervised networks, or telling a supervised network what responses are correct and to what degree generalisation of results is acceptable.

Although many proponents of ANNs take it as read that neural networks more nearly approximate real-brain processes than do artificial symbol systems, this may not be so (Churchland, 1989, Feldman and Ballard, 1982, Fodor and Pylyshyn, 1988). The argument, to paraphrase it crudely, is as follows. On the one hand, can ANNs be models both of *brain structure* and of the way people *represent* the world in their minds, ie carry out cognition (Churchland)? Or can ANNs, at best, be models only of brain structure (Fodor)? For my part, I agree with Fodor that, for connectionists to construe ANNs as models of cognition as Churchland does (and Rumelhart, for that matter – (Rumelhart and McClelland, 1986)), is not sustainable. Alternatively, if ANNs are models of implementation, then we should pay more attention to real biological structures and abandon the idea that cognitive abilities will somehow manifest themselves out of ANN architectures.

A.6 Consciousness, the nervous system, and computation

Does any of this matter? As engineers, can we not dispense with the notion of consciousness, and concentrate on the algorithms that will put human functions like face or speech recognition within our grasp? Following Searle's lead, is the question for us not : what kind of neural organisation would have the features to embody the minimum necessary for, say, colour vision? The answers are : yes, it does matter and, no, we cannot ignore consciousness, because the brain is a

system for acquiring knowledge about the world, and knowledge (of whose face we see, of what that person is saying) depends on consciousness. Some people with damage to a part of the visual cortex, can 'see' directional motion in an object without being conscious that they are doing so, a phenomenon known as 'blindsight' (Zeki, 1993). These patients' vision is useless, since they cannot acquire any knowledge about the world through it. It seems that the integrity of the visual cortex is essential for the conscious experience of vision.

Nevertheless, the mutual dependence of neurobiology and computational approaches is strong. For example, computational approaches have taught us that some human activities that have a high intellectual content, like playing chess, are highly amenable to algorithmic solutions. By contrast, knowledge of brain architecture and function can be a catalyst for devising plausible and powerful algorithms that may give us insights into brain function and provide us with the tools to build machines to carry out real-world tasks (Churchland and Sejnowski, 1988).

A.7 Recent Developments

The preceding criticisms do not diminish the advances made in recent decades in our understanding of what is, and is not, possible, and has radically altered our notions of the nature of intelligence. A huge strength of ANN research is its multi-disciplinary nature, stimulating a healthy cross-fertilisation in ideas, for example in Brooks' work (Brooks, 1991) on building intelligent robots.

Another promising development is the growth of interest in new ways of thinking about these problems, stimulated by, among other things, the ideas of phenomenology. Here we think of people not as passive receivers of sense-data that are processed in some mysterious way to transform them into meaning, but rather as agents in the world, actively seeking meaning and performing actions in a context already charged with meaning. At this stage, the phenomenological approach does not offer a clear research programme in the way that neural-network research has done — although, interestingly, Terry Winograd, an erstwhile, and talented, champion of conventional artificial intelligence, now teaches and writes (Winograd and Flores, 1986) in phenomenological terms. However, it does suggest some very auspicious augmentations to more traditional

approaches, beginning in psychology (Suchman, 1987) and running through the design of human-computer interfaces (Winograd and Flores, 1986) to robotics engineering (Brooks, 1989, Beer, 1990, Connell, 1990).

Edelman (Edelman, 1994) takes many of these ideas a stage further. For him, the crucial difference between computers (and indeed any mechanistic, that is chemical or physical, description of the brain) and real brains is evolutionary morphology, in other words the dynamic arrangement of the constituents of body organs, including the brain, to show system properties. In Edelman's view, the striking properties of brains cannot be separated or abstracted from the topological arrangement of the brain itself. Intelligence cannot be 'disembodied' by abstracting it and installing it in software or hardware or a combination of both. Any truly intelligent machine, which would necessarily have its 'brain' organised on evolutionary principles, would have to be able to connect causally-unconnected outside events for its own adaptive needs — no trainer to tell it what was correct or incorrect, no operator to retrain it in the event of 'error', no homunculus to determine the 'meaning' of its own self-organisation.

A.8 Learning in psychology, neurobiology and ANNs

I consider in this section the relationship of 'learning', as it is commonly understood, to 'learning' in neurobiology and ANNs.

A.8.1 Psychological views of learning

Since learning and memory are psychological phenomena, the opinions of psychologists ought to be helpful here. Unfortunately they are not, for these reasons.

Firstly, there is no simple definition of learning, because it is a complex phenomenon (Thornton, 1992), which cannot be reduced to an equation or two.

Secondly, and astonishingly, until recently, most textbooks treated learning and memory as entirely separate, discussing one and omitting the other entirely. In this respect, connectionism, as a discipline that unifies the two phenomena, has

had the advantage over the psychologists until very recently, when it was argued that, since learning, remembering and forgetting all occur in the same biological context, their adaptive functions must be intertwined (Boulton, 1994).

Thirdly, categorisation of types of learning is a mess (because it is a complex phenomenon). Psychologists commonly distinguish five areas of learning :

1. *Developmental learning*, that is learning, for example to perceive objects, as we grow;

2. *Non-associative learning* including habituation (reduction in response to continued stimulus) and sensitisation (increase in response to repeated stimulus), and also including reflexes;

3. *Associative or stimulus-response learning*, including classical conditioning (eg in "Pavlov's dogs" experiments) and operant conditioning[3] (association between a stimulus and a response such as is used in shaping circus animals' behaviour);

4. *Procedural learning* (learning 'how'), of which *motor learning* is a form. This is really a special form of associative learning;

5. *Declarative learning*, of which *relational learning* is a form, for instance learning about relationships between objects in space (spatial learning), between events in the past (episodic learning) or between another person's behaviour and our own (observational learning).

Of these categories, associative learning is a mechanism while the others are phenomenological observations (about which, admittedly, a great deal is known). Psychologists have, historically, expended a lot of effort on the study of associative learning but it cannot explain the other categories except in the most rudimentary terms.

---

[3]The distinction between classical and operant conditioning is not a very clear one. Classical conditioning exploits reflexes, while operant conditioning uses spontaneous animal behaviour for which there is no explicit stimulus, for example rats pressing the lever in an experimental box.

A.8.2 Neurobiological views of learning

Neurobiologists ask similar questions to ANN researchers. For example : Where does learning take place? How is information about a learned event acquired and encoded? How is the information retrieved? They also have a strong candidate for a mechanism of learning, called *long-term potentiation* (Byrne, 1988, Brown et al., 1988). This interesting model of learning is based on the observation that, in some parts of the brain (notably the hippocampus), a brief burst of high-frequency stimulation in a nervous-system pathway leads to augmentation of the post-synaptic cell's response to subsequent, normal, stimuli.

A.8.3 ANN views of learning

Machine learning is now a field of enquiry entirely separate from the psychological and neurobiological fields. Machine learning is considered to take place when a computer system automatically generates a new data structure or program out of an old one, and so irrevocably changes itself, with some purpose (Anzai, 1992). This, of course applies to ANNs.

The discovery of long-term potentiation, referred to in Section A.8.2, has been seen as exciting because it is the kind of mechanism recognised by connectionists as Hebbian learning. Hebb (Hebb, 1949) proposed an associative mechanism[4] in the brain that has been interpreted as an algorithm in which weighted connections are strengthened as the result of correlations between the responses of connecting nodes. Long-term potentiation has four characteristics required of learning (Groves and Rebec, 1992) : it is initiated by brief stimuli, so can capture transient events; only inputs carrying information are involved (as is true in ANNs); there is cooperativity , ie one node must 'fire' as the other one does (as is the case in artificial Hebbian networks); and there is associativity between nodes in time or strength (as is also the case in artificial Hebbian networks).

---

[4] "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." (p62)

A.8.4 The relationship between 'learning' in psychology, neurobiology and ANNs

Relating all these phenomena is very difficult, because our common-sense notions of learning encompass everything from learned reflexes, to an intricate skill, to the tenets of eastern philosophy. It is not at all obvious what is the link between neurobiological mechanisms and higher-level psychological views of learning, and the relationship between psychological views and ANNs is even more tenuous still. There are some superficial similarities between learning in ANNs and in people, particularly for rote learning and learning a motor action (for example a tennis stroke). However, it should be clear that, as the term 'learning', as commonly understood, symbolises such a complex phenomenon, to draw comparisons between the artificial and the human is, to say the least, premature.

The large variety of angles from which different researchers approach the problem is one of the fields greatest strengths. Nevertheless, many ANN algorithms are biologically unrealistic and discovering their biological counterparts, if any, will be difficult (Mitchison, 1989). If engineers acknowledge this truth then each area of the field has much to learn from the others.

A.9 An engineering approach to neural networks

In my view, 'artificial intelligence' as envisaged by the researchers discussed in the previous sections, even if possible in principle, is so remote a prospect that it is not worth considering. The questions to be answered for an engineer are therefore :

- What stance would it be reasonable for an engineer to take in relation to work in other neural-network fields?
- What aims should engineers working in research in neural networks have?

Bezdek (Bezdek, 1992) has made a commendable but not wholly satisfactory attempt to answer these questions by defining 'computational intelligence' in relation to biological (that is real-world) intelligence and artificial intelligence, that

is the type of machine intelligence that is often claimed to comprise 'knowledge'. Bezdek says artificial intelligence actually comprises 'knowledge tidbits', the numerical information, rules and constraints that the investigator uses to replicate intelligent behaviour in a machine. Computational neural networks provide a low-level, computational strategy for carrying out some kind of pattern-classification, and nothing more. As Bezdek says : "[A back-propagation network] 'learns' (its parameters, the weights in the network) in exactly the same sense that the EM algorithm for finding maximum likelihood estimators from labelled data does". Unfortunately, Bezdek's strictures on avoiding being seduced by mentalistic phrases that, by being interpreted in an anthropomorphic way, convey something more profound and substantial than is justifiable, are usually ignored.

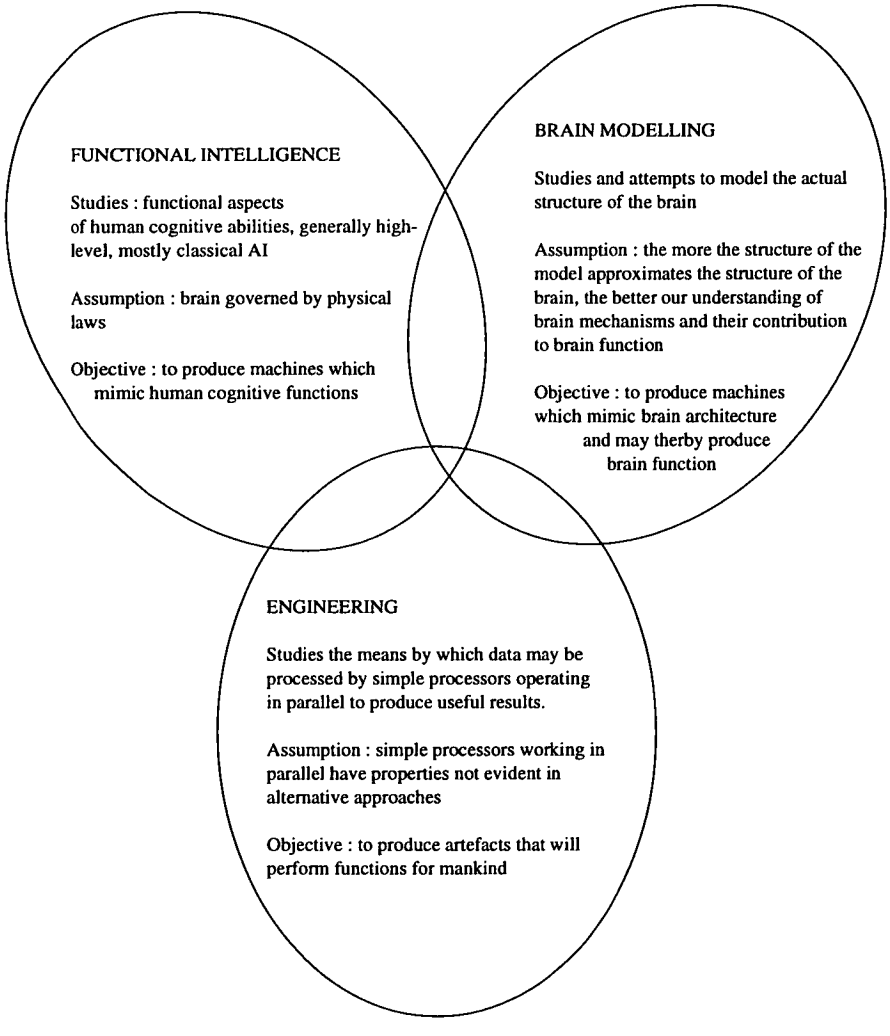


Figure A–1: *Different kinds of enquiry*

Building on Bezdek's work, I offer the classification shown in Figure A–1 to illustrate the relationship of neural networks in engineering to neural networks in two other fields. The classification is of course a gross over-simplification of the huge variety of uses to which neural networks are put, but the simplification has the benefit of clarifying what, to my mind, are serious confusions in the minds of researchers in all fields.

On my interpretation, brain-modelling is a perfectly respectable pursuit with the aim of investigating the characteristics of real brains and representing these characteristics in a way that is at least as useful as any other modelling technique that gives us an insight into human performance. My opinion is that such models are as valid for our time as information-processing models of the brain were in the 60s or telephone-exchange models were before that; they do not, by any stretch of the imagination, describe reality, but they are a useful way of thinking about problems that enables us to make testable predictions about human performance. However, clearly, the more the models are based on our actual understanding of how components of the brain and the nervous system operate, the more likely the models are to tell us something about real brains. Hence, there will be a propensity (but not an overwhelming one) for models to use components that we believe are like the components of the brain, of which neural networks may provide an example. This description would obviously cover much neural-network research, but also much research in conventional artificial intelligence. It will continue to open up many avenues of interest, to help our understanding of what intelligence actually is, and to highlight and clarify what is particular about people that makes us different from machines.

I choose the term functional intelligence to indicate that, in this second field, an explicit aim is to produce models and machines that try to mimic cognitive functions, both low-level (eg pattern discrimination) and high-level (eg creativity, game-playing), in other words those aspects of human performance that we normally think of as intelligent. I say functional because this research can proceed without necessarily concerning itself about the mechanisms by which people carry out these functions, nor with the structural details of the areas of the brain that are involved. Thus a chess-playing machine fits into this category even though it is realised in software or electronic hardware and not in biological neurons, nor even in a crude analogue of neurons, and even though the mechanisms by which it operates (sophisticated algorithms searching through a complex 'game-space') are rather unlikely to be the ones human chess-players use.

I make it clear here that researchers in both functional intelligence and brain-modelling may choose to believe that their models are 'truly' intelligent, that they show evidence of 'thinking', 'sleeping', 'dreaming' or whatever mentalistic labels they wish to ascribe to them but, for the reasons I have outlined, I disagree, and predict that such labels will not be justifiable in the foreseeable future, if ever. If these studies advance more rapidly than I predict then, in 100 years, I may be willing to change my, by then disembodied, mind.

I think that engineers, represented in the third field in the diagram, have a unique position in this triptych, because they are wedded neither to the idea of producing intelligent machines, nor to the need to demonstrate that their models have any association with real brains, any more than aeroplanes have an association with birds or trains with horses. Engineers can take a magpie approach to problems, stealing whatever bright and shiny idea might seem attractive, with the aim of solving problems in whatever way seems practical. We can keep an interested, if sceptical, eye on developments in these fields, in the hope that they will turn up some profitable technique or helpful notion. Our objectives are both easier to define and, in some ways, more difficult to realise, in that we are trying to produce artefacts that are useful to mankind, using a method, neural networks, for which the problems of implementation are formidable[5]. In this enterprise, it is to our advantage to renounce, as far as is practical, words like 'intelligence', 'learning' and 'recall' that lend an unjustified radiance to our work.

The three fields in the diagram are shown as overlapping because each draws on the others for knowledge and inspiration. For example, the engineer may design a robot along lines developed in the world of functional intelligence, as Brooks and his students have done; or they may develop electronic models of real-world neural structures (Shoemaker et al., 1992).

---

[5]The prime reasons for using ANNs are that : (1) unlike conventional computers, they need no explicit programming, but adapt based on examples of similar problems; and (2) they can be effective at solving problems where solutions are difficult or impossible to define (Rees, 1996).

A.10 Summary and conclusions

In the myriad of views on the relationship of minds, computational machines and the physical world, we can distinguish three main positions, which I have characterised as strong AI, weak AI and computational intelligence. I have looked briefly at each of these positions, explaining the historical forces behind the desire to link human functions and machines.

I have then examined the notion of learning in people and machines and demonstrated the fragility of the link between them. Finally, I have come to a conclusion about the approach engineers should take and the objectives they should aim for.

There are strong cultural factors that seduce us into seeing a greater similarity than actually exists between machines and people, particularly where 'thinking' is concerned. In this respect, it is unfortunate that ANN investigators without a knowledge of the elements of neurophysiology have adopted so much biological terminology for artificial networks, as these terms lend an unjustified radiance to their work that confuses outside observers, and even those working in the field itself.

Although the idea of 'intelligence' and 'mind' as emergent properties of computational systems has a long history, there are strong objections to it.

Notions of 'biological inspiration' are surrounded by confusion. Engineers should be clear about the motivation for their appeals to biology :

- to produce functionality. Biology may give us ideas (eg 'discounting the luminance') but the maxim here is 'function, not form'.

- to model biological systems (because of their power) in silicon and apply the models to problems which conventional computers find intractable — the 'neuromorphic' approach. The driving force here remains functionality, but from a distinctly biological perspective. Since the brain is so imperfectly understood, to 'do things the way the brain does' is very difficult, and a great deal of effort is likely to be necessary before any function approximating human function, for instance 'seeing' or 'hearing', is achieved.

- to model brain function, with the aim of better understanding how biological systems work. This is an area with few investigators and, in my view, poorly exploited.

The phenomenon of learning in people is a complex one, and psychologists, neurobiologists and ANN researchers see it from rather different perspectives. A unified theory of learning, memory and recall does not appear to be imminent.

Engineers have a unique place in ANN research. Their motivation to build useful machines, forces them to confront real-world difficulties. Ill-defined notions can be subjected, by the very nature of the discipline, to rigorous testing and to justification by results.

Nevertheless, engineers should be aware that human consciousness, and people's ability to acquire knowledge about, and act in, the world, are very special characteristics, and most unlikely to be replicated in machines. Computers, whatever their nature, will never have 'subjective awareness', nor will they 'experience sensations', nor 'be conscious'. Any investigator who believes that these human characteristics will emerge from non-conscious, computational building-blocks has a number of difficult questions to answer, such as : What would a programme of research to discover the foundations of these characteristics comprise? If there is a link between physical substrates and consciousness, why would silicon be a good substrate to use for the building-blocks? If we did have conscious machines, what would they be like, and what use would they be?

Appendix B

Quotations from workers in the field of ANNs

'We're beginning to understand the way that these connection-weights [in biological networks], that is the ways one nerve-cell affects another, are changed by different environments, and it's through that set of rules of changes of connection-weights that we think we'll be able to understand the whole of this [the brain's] complexity; how it was, even, that Mozart and Einstein were able to have their amazing creativity ...' †[1]

John Taylor
Director
Centre for Neural Networks
King's College, London

'Networks are a state of mind and I think that some day in the future your best friend may be a neural network.' †

Terry Sejnowski
The Howard Hughes Medical Institute At The Salk Institute

'Now there is a state we can create [in a neural network with light- and dark-sensors] simply by cutting the sensors off, which is

[1]Quotations marked †are from "Equinox : Teaching computers to think", broadcast by Channel 4 Television in March 1992.

a bit like sleep in human beings. At that point, the neural network starts running off on its own, and has flashes of the images that it's learned. Now that may be what dreaming's all about ...' †

Igor Aleksander
Professor of Neural SystemsImperial College, London

'Will it be possible one day to hold a rational intelligent conversation with a computer? This question has occupied the greatest scientific brains of the post-war period ... the debate was stirred up with some remarkable claims from Igor Aleksander, one of our most respected computer engineers. Yes, he says, it will be possible, one day, to converse with computers. And, more astonishingly, Aleksander and his team ...claim to have created a computer brain that is based on the same principles as its human counterpart: it dreams, it thinks, it forms mental images, suffers emotion, and even dabbles in free will.'[2]

Christopher Lloyd
Journalist

'...he [Humphreys] is also thinking about a silicon chip which one day might be programmed with names, addresses and telephone numbers and then be fitted into the brains of Alzheimer's disease patients, to provide perfect recall for faltering memories. ... "You could put an empty silicon chip in and the brain itself could store information. It sounds far-fetched but I think it is feasible. ... One might hope that, if you look at the future, the process will be indistinguishable: when the brain cells are interacting with the silicon, they are not aware that they are not interacting with another brain cell.'[3]

Professor Colin Humphreys
Head of Metallurgy and Material Science
Cambridge University

---

[2] *Sunday Times*, 29 May 1994.

[3] *The Guardian (Science)*, 24 March 1994.

'Scientists in the United States are working on a new generation of computers [using bioelectronics] which ... [according to Signal magazine] ... "could lead toward an extremely fast machine that might match or correspond with the human operator's intellect" ... "It's almost like a circuit board of real cells ... Once we have [used the technique in] artificial limbs, we will progress towards artificial intelligence" ...'[4]

Dr James Hickman
Research Chemist
Science Applications International Corporation
McLean, Virginia

'In the next few decades we will have machines and computers with far superior brain power and computational capacity than humans ... Even the most conservative estimates indicate that in the early part of the next century accessible computational power with known artificial means, such as electronic circuitry, will be greater than that of humans ... Clearly, it will be possible in the next 50 years to achieve artificial intelligence systems which are not only more intelligent than humans, but also exhibit a significant number of advantages: they will be faster, more reliable, quicker to learn, more robust, and usually more accurate ... I don't believe there is anything to stop robots and machines being creative in the same way humans are. They could even have emotions.'[5]

Kevin Warwick
Professor of Cybernetics
Reading University

'This [noise in neural networks] is the equivalent of internal imagery. It's like staring at a blank screen and having internal images parade past ... You can be led astray into thinking that I'm saying

---

[4] *The Scotsman*, 2 February 1994.

[5] *The Glasgow Herald*, 12 September 1995.

that the stream of consciousness is nothing more than noise being translated into different thoughts. But that's not the case ... You've got noise, which is producing neural images in no distinct order, but those helter-skelter thoughts can now excite whole chains of associations. That is the systematic part of intelligence.'

'People will start to ask the question : how am I distinct from these machines? And I think the inevitable answer, though sidestepped for decades, is that we're the same, Once we looked at the birds and emulated what was needed to fly, and now we're flying. And now I think we know enough about the brain that we can make real brains and do the things we thought were sacrosanct, like creation.'[6]

Steven Thaler
Inventor of the 'Creativity Machine'

'We [our group at Caltech] build systems that are models of pieces of the nervous system like the retina, or the cochlea in your ear. What I mean by a model is a system that in some way does a similar function, systems which emulate, in some way, pieces of the nervous system ... ' †

'I think at the present time we have enough technology to build anything we could imagine. The problem is we don't know what to imagine, we don't understand enough about how the nervous system computes, to really make more complete thinking systems ... ' †

Carver Mead
Moore Professor of Computer Science
Caltech

---

[6] *New Scientist, 20 January, 1996*

'We really are nowhere near having [neural networks with] the sophistication of something like a cat or a dog or a rat. These are much more sophisticated information-processing systems; we're a very long way away from coming anywhere close to real biological intelligence, and I think it'll be quite a long time before we get anywhere near it ...' †

Geoffrey Hinton
Canadian Institute for Advanced Research
University of Toronto

Appendix C

Implementations of, and circuits for, on-chip learning

The table on the following pages lists, as a guide to published work in this area, digital and analogue implementations of on-chip learning, together with other published work which is of interest from the point of view of this thesis.

The table lists the implementations in alphabetical order by the first-named author of the published paper or papers in which they are described. Within the table itself are descriptions of the application which the authors had in mind, and the algorithm which they have contrived, or attempted, to implement. The table then explains the main circuits used to implement the functions which are common to ANN algorithms : to provide a weight, and to change that weight; for multiplication at a synapse; for summation of weighted inputs and for a non-linear mapping to a neuron output; and for any other additional function that is necessary.

Table C–1: List of main features of on-chip learning implementations

Authors	Application	Algorithm	Weight	Weight-change	Synapse	Neuron	Others	Remarks
Abusland and Lande, 1994	Associative memory	Hopfield net	Twin voltages on analogue floating-gate mapped onto UV-activated conductances	UV-light intensity (for learning speed) and exposure-time (for weight-change)	Gilbert multiplier	Transconductance amplifier		Same group as Berg *et al*
Alspector *et al*, 1989	XOR, unsupervised competitive	Modified Boltzmann machine	Digital, converted to analogue conductance	Digital increment or decrement	Conductance increases linearly with digital value	Double differential amplifier giving *tanh* function	Noise amplifiers to simulate temperature	
Alspector *et al*, 1992	Parity problem and generalisation of XOR	Boltzmann mean-field network	Digital	Digital increment or decrement	Voltage x weight produces current	Amplifiers with variable gain	Noise-generator summing in form of a current	
Arima *et al*,1991a, 1991b, 1992	Pattern recognition	Boltzmann machine	Capacitors, one excitatory and one inhibitory	Charge-pump giving 10% change - resolution	Current weighted according to excitatory or inhibitory capacitor-voltage	Summed currents into comparator which flips if Vref greater than threshold	Weight refresh is by learning	
Berg *et al*, 1996, Sigvartsen, 1994	XOR	Back-propagation	Twin voltages on analogue floating-gate mapped onto UV-activated conductances	UV-light intensity (for learning speed) and exposure-time (for weight-change)	Differential pair maps weight-voltages x state-current onto current output	Differential pair maps votage-difference onto single-ended current	Complete circuits to implement entire algorithm	Continuous-time system
Botros and Abdul-Aziz, 1993	Pattern association	Back-propagation	Digital implementation using FPGA, with learning off-chip					Simple, at least as concept

Table C-1 (cont): List of main features of on-chip learning implementations

Authors	Application	Algorithm	Weight	Weight-change	Synapse	Neuron	Others	Remarks
Choi and Salam, 1993	Pattern association and XOR	Back-propagation modified so derivative-of-sigmoid terms removed	Capacitive, with steady-state value being read and stored off chip	Continuous-time circuit settles into steady state	MOS transistor current control	Operational amplifier		Simulation only
Cohen and Andreou, 1992	Not stated	Hebbian learning, Herault-Jutten neuro-morphic network	Capacitive	"Bump" circuit to alter multiplier's current in small steps	Gilbert, sub-threshold	Not stated		Simulation only
Dolenko and Card, 1993	None : investigation of hardware properties	Back-propagation	Capacitive	Multiplier charges or discharges capacitor	Gilbert, super-threshold		Learning-rate is change-time. Refresh is repeated pattern-presentation	Simulation only
Donald and Akers, 1993	Ostensibly real-time control, but actually a classifier	Modified Hebbian learning algorithm, due to Oja (unsupervised learning)	Capacitive. Refresh method not stated.	"Statistical sampling" capacitor charged or discharged, then switched-capacitor shift onto weight - capacitor	Current proportional to weight injected into summing node, then integration on capacitor	Threshold circuit producing binary output		Refresh implicitly by learning
Duranton and Sirat, 1990	Image- or speech-recognition	Various	Fully digital CMOS implementation					Sigmoid executed off-chip
Eguchi *et al*, 1991	Inverted pendulum, character-recognition	Modified back-propagation	Digital implementation using logic-gates and stochastic pulse-trains					Conference paper never published in journal

Table C-1 (cont): List of main features of on-chip learning implementations

Authors	Application	Algorithm	Weight	Weight-change	Synapse	Neuron	Others	Remarks
El-Masry *et al*, 1992	Not applicable since weight-change building-block circuit		OP-amp output voltage	Switching of voltages at op-amp input	Not applicable since weight-change building-block circuit			Simulation only
Frye *et al*, 1991	Ballistic trajectory as a test of system-identification	Modified back-propagation	Length of bar of light projected onto photo-conductive array	Off-chip	Transimpedance amplifier	Differential summing amplifier		Opto-electronic, included here for relevance to VLSI circuits
Ghosh *et al*, 1994a, 1994b	Not stated	Hopfield, Boltzmann, cellular networks	Capacitive, controlling tuning current	Tuning-current proportional to weight varies bias-transistor gate-voltage	OTA multiplies differential input voltage times transconductance, varied by bias - current	OTA	Digital storage envisaged	Simulation only
Hammerstrom D, 1990	General-purpose machine suitable for `large, real world problems'	Several	Digital implementation					
Hollis and Paulos, 1994	Classification and functional mapping	Semi-parallel weight perturbation	Digital, controlling switched current sources	Increment/ decrement	Differential circuit with tail-current controlled bu#y weighted switched current sources	Same differential pair as the synapse		Simulation only
Ibrahim and Zaghloul, 1990	Not applicable : implementation of a weight-change cell only		Capacitive	Schwartz modifier (see under *Schwartx et al*)	Gilbert	Not applicable		
Kim *et al*, 1992	Not applicable : implementation of a synapse cell only		Resistor conductance implemented as floating-gate transistor	Not applicable	Input-voltage times resistor conductance	Not applicable		

Authors	Application	Algorithm	Weight	Weight-change	Synapse	Neuron	Others	Remarks
Lehmann, 1993, 1994, 1995	Pattern recognition	Back-propagation	Differential voltages on gate-capacitances	Elegant synapse design permits backward-propagation calculation in serial fashion	MOS resistive cell produces differential current from differential voltage	Differential pair of parasitic bipolar transistors	Digital storage	
Linares-Barranco *et al*, 1993	Bi-directional associative memory (BAM)	Hebbian learning	Capacitive	Via multiplier	Transconductance multiplier	Transconductance multiplier	Refresh by reading weights and adding a little charge	
Lindblad *et al*, 1995	Character recognition	Radial-basis function	Digiatal implementation of 'zero-instruction-set computer'					Learning circuitry is on-chip
Macq *et al, 1992*	Not applicable, since building-block circuits. Kohonen network envisaged		Current	Analogue winner-takes-all circuit detects smallest distance and adjusts a column of synaptic array			Refresh by ADC and DAC, reading stored current, writing next-upper reference	Concerned only with analogue storage of adjustable weights. On-chip learning predicted but not implemented.
Meador *et al*, 1991	Not stated	Not stated	Floating-gate storage	Re-programming of floating-gate	Voltage-controlled switch converts incoming pulse-trains to sequence of charge-packets on or off weight-capacitor	Relaxation oscillators producing variable pulse-frequency output		Not yet "auto-adaptive"

Table C-1 (cont): List of main features of on-chip learning implementations

Authors	Application	Algorithm	Weight	Weight-change	Synapse	Neuron	Others	Remarks
Montalvo *et al*, 1992, 1994	Not applicable : building-block circuits only		During learning, capacitive; after learning, floating-gate storage		Multiplication of differential input-voltage times differential currents produced by floating-gate transistors	Operational amplifier converts summed currents to differential output voltage		Combines learning capability with non-volatile storage
Myers *et al*, 1992	Building-blocks for real-time image and signal processing	Back-propagation	Fully-digital implementation. Trains with 8 bit weights (rather than the usual 12 bits) by using pseudo-random noise sources. Learning on the chip.					
Säckinger *et al*, 1992	Character recognition	Five-layer, feed-forward	Capacitive, refreshed by on-chip DAC with resolution of 6 bits (+ 4 scaling)	Off-chip	Multiplying DAC (analogue weight times digital state)	Digital, via current-summing and ADC		The ANNA chip, capable of various topologies
Salam and Wang, 1991	Pattern and character recognition	Modified Hopfield net	Voltages on gates of inter-connecting transistors, set to **high** or **low**	Learns to store a pattern		Conceptually op-amp with RC feed-back, implemented as double inverter using RC parasitics		Continuous-time "digital" implement-ation
Schneider and Card, 1991b, 1991c	Not stated	Mean-field network with Hebbian learning (deterministic version of Boltzmann machine)	Capacitive	Access-transistors to weight capacitors used sub-threshold to dribble charge on or off	Super-threshold Gilbert, or "inverter" with **high** and **low** voltages set so transistor works in linear region	Super-threshold Gilbert with output current converted to a voltage	Refresh by learning	

Table C-1 (cont): List of main features of on-chip learning implementations

Table C-1 (cont): List of main features of on-chip learning implementations

Authors	Application	Algorithm	Weight	Weight-change	Synapse	Neuron	Others	Remarks
Schwartz *et al*, 1989a, 1989b, 1990, 1991	Not applicable, since building-block circuits.		Capacitive	Switched-capacitor system, making small weight-changes possible	Not applicable			Digital primary store envisaged, or constant learning
Shibata and Ohmi, 1992, 1995	None stated or demonstrated	Various	Threshold voltage of floating-gate vMOS transistor	Vth altered by injecting charge through gated node onto floating-gate using programming pulses	All signals in voltage mode using vMOS transistors. Binary states given weight by subtracting Vth. Charge distributed over common floating gate sums outputs of each vMOS transistor.			Cunning implement-ation
Shima *et al*, 1992	Identity-mapping problem	Simplified back-propagation	Static RAM (but claim could be analogue) backing up current	A-to-D conversion, increment or decrement, then D-to-A conversion	Gilbert multipliers	Summed currents into transimpedance amplifier	Maximum-value detector for sigmoid-derivative	Synapses and weight-change on one chip, neurons on another
Theeten *et al*, 1990		Back-propagation with local learning rule	Digital implementation					Same research group as Duranton and Sirat
Tomberg and Kaski, 1991	None stated	Back-propagation	Digital implementation					
van Daalen *et al*, 1994	None stated	Back-propagation	Not stated. Concerned only with output neuron			Stochastic bit-stream implementation based on digital coounter	Neuron also calculates derivative of output as required by back-propagation	Can be considered a digital implement-ation

Authors	Application	Algorithm	Weight	Weight-change	Synapse	Neuron	Others	Remarks
Wang, 1993a, 1993b	None stated	Back-propagation	Capacitive, refreshed by DAC	Charge-transfer between buffered capacitors	Wide-range Gilbert	Combination of transimpedance amplifer and differential amplifier		Synapses on one chip, neurons on another
Watola and Meador, 1992	Clustering applications	Competitive-pulse Hebbian learning	Capacitive	Switched-current sources to dribble charge on or off	Current modulated by pulsing on gating transistor	Not applicable	Refresh unnecessary because applications have statistics which vary more rapidly than refresh	Asynchronous pulse-mode (ie pulse-density) implement-ation

Table C-1 (cont): List of main features of on-chip learning implementations

Appendix D

Table of learning equations

The table on the following page shows the learning equations for the virtual targets and back-propagation algorithms.

In the implementation of the virtual targets algorithm described in this thesis, weight and target updates are carried out 'stochastically'; in other words, every epoch, each input pattern is presented and the weights and targets are updated after every pattern presentation.

Both the virtual targets and the back-propagation algorithms make use of a term called the 'sigmoid-prime'. As explained in Chapter 5, the virtual targets algorithm was simplified by removing the term; hence the table shows the equations with and without the 'sigmoid-prime'.

Virtual Targets Algorithm	
with 'sigmoid prime' term	**without term**
Output-layer weights :	
$\Delta w_{kj} = \eta_w O_j O'_k \varepsilon_k$	
$= \eta_w O_j O_k (1 - O_k)(t_k - O_k)$	$= \eta_w O_j (t_k - O_k)$
Hidden-layer weights :	
$\Delta w_{ji} = \eta_w O_i O'_j \varepsilon_j$	
$= \eta_w O_i O_j (1 - O_j)(t_j - O_j)$	$= \eta_w O_i (t_j - O_j)$
Hidden-layer targets :	
$\Delta t_j = \eta_t \sum w_{kj} \varepsilon_k$ $= \eta_t \sum w_{kj} (t_k - O_k)$	
Back-propagation Algorithm	
Output-layer weights :	
$\Delta w_{kj} = \eta_w O_j \delta_k$	
$= \eta_w O_j O_k (1 - O_k)(t_k - O_k)$	$= \eta_w O_j (t_k - O_k)$
Hidden-layer weights :	
$\Delta w_{ji} = \eta_w O_i \delta_j$	
$= \eta_w O_i O_j (1 - O_j) \sum w_{kj} \delta_k$ $= \eta_w O_i O_j (1 - O_j) \sum w_{kj} O_k (1 - O_k)(t_k - O_k)$	$= \eta_w O_i \sum w_{kj} (t_k - O_k)$

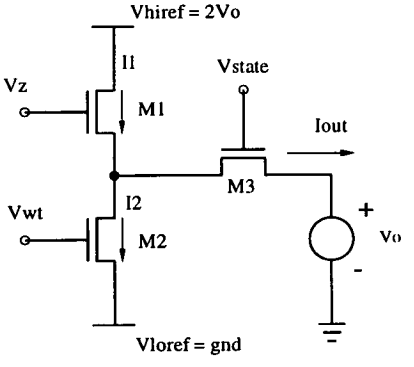
where

- i, j and k are indices for the input, hidden, and output layers, respectively
- w is a weight
- O_i, O_j and O_k are the outputs of the input, hidden, and output layers, respectively
- η_w and η_t are weight and target gain-terms, respectively
- ε is the error-term such that $t_k - O_k$ represents a desired minus an actual output

Table D–1: The virtual-targets and back-propagation equations

Appendix E

Analysis of the twin-synapse circuit



1

Assumptions :

1. Transistors M_1 and M_2 operate in the linear region, so that :
   $i_{DS} = \beta v_{DS}\left[(v_{GS} - V_T) - \frac{1}{2}v_{DS}\right]$
2. The widths and lengths of M_1 and M_2 are equal so that :
   $\beta_{M_1} = \frac{\mu_o C_{ox} W}{L} = \beta_{M_2}$
   where μ_o is the surface mobility parameter, C_{ox} is the capacitance per unit area of the gate oxide, W is the width, and L is the length.
3. The effect of transistor M_3 in its **ON** state can be ignored.

Figure E–1: *The model of a single synapse to be analysed.*

The model of a single synapse to be analysed is shown in Figure E–1.

$$\begin{aligned} i_1 &= \beta v_{DS}\left[(v_{GS} - V_T) - \frac{1}{2}v_{DS}\right] \\ &= \beta V_o\left[(V_z - V_o - V_T) - \frac{1}{2}V_o\right] \\ i_2 &= \beta V_o\left[(v_{wt} - V_T) - \frac{1}{2}V_o\right] \\ i_{out} &= i_1 - i_2 \\ &= \beta V_o\left[V_z - v_{wt} - V_o\right] \end{aligned}$$

Since β, V_0 and V_z are constants, then :

$$i_{out} \propto -v_{wt}$$

The twin-synapse model shown in Figure E–2 can be analysed, rather informally, in the same way as the single synapse.

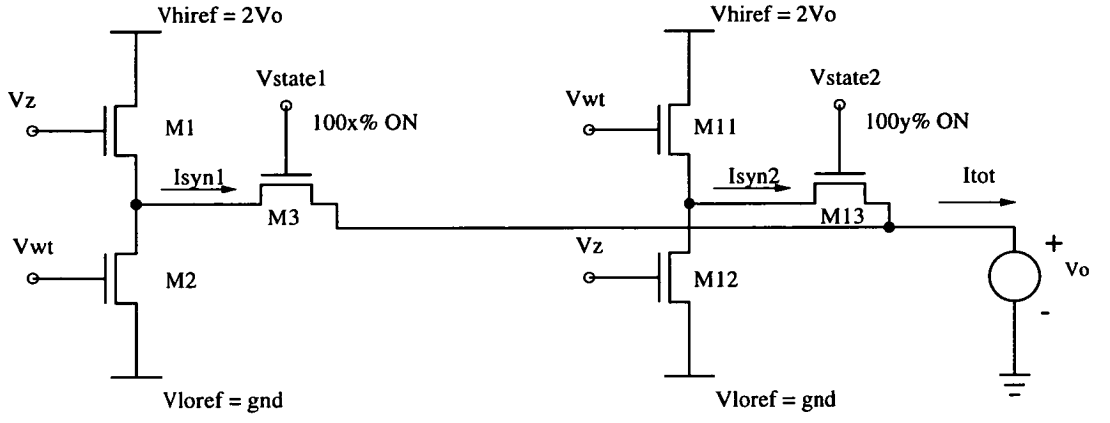


Figure E–2: *The model of a twin synapse to be analysed. The assumptions made for the single-synapse model apply.*

The analysis is as follows :

$$\begin{aligned} i_{syn_1} &= \beta V_o [V_z - v_{wt} - V_o] \\ i_{syn_2} &= \beta V_o [v_{wt} - V_z - V_o] \end{aligned}$$

Assume a time-frame in which *state*1, acting as a zero pulse, occupies half the frame, as shown in Figure E–3. Hence, *state*2, which is of variable width within the frame can, at the extremes, occupy none or all of the frame, Hence, *state*1 $\in [\frac{1}{2}]$ and *state*2 $\in [0, 1]$.

Let $x = \frac{1}{2}$ (for a 50% occupation of the frame), and let $z = (y - \frac{1}{2}) \Rightarrow z \in [-\frac{1}{2}, \frac{1}{2}]$.

Then :

$$\begin{aligned} i_{tot} &= i_{syn1} + i_{syn2} \\ &= \beta V_o [(V_z - v_{wt} - V_o)x + (v_{wt} - V_z - V_o)y] \\ &= \beta V_o \left[\frac{1}{2}V_z - \frac{1}{2}v_{wt} - \frac{1}{2}V_o + zv_{wt} - zV_z - zV_o + \frac{1}{2}v_{wt} - \frac{1}{2}V_z - \frac{1}{2}V_o\right] \\ &= \beta V_o [z(v_{wt} - V_z) - V_o(z + 1)] \end{aligned}$$

The effect of this is that $i_{out} \propto v_{wt} - C$ where C is an offset current which has to be 'injected' into the synapse to achieve the correct zero point. The offset could be corrected using the current source shown in Figure E–4.

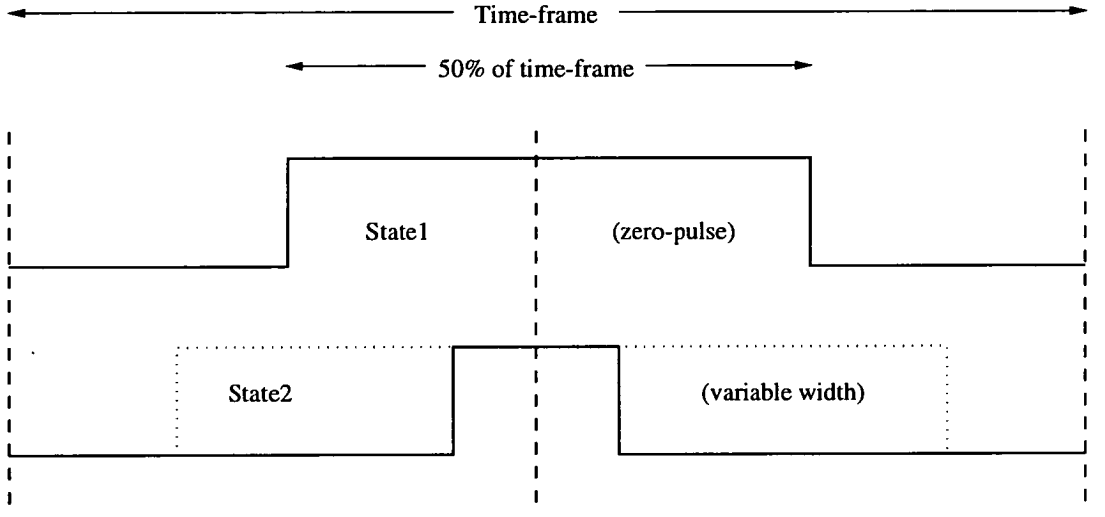


Figure E–3: *How the time-frame is occupied by the zero-pulse and a variable-width pulse*

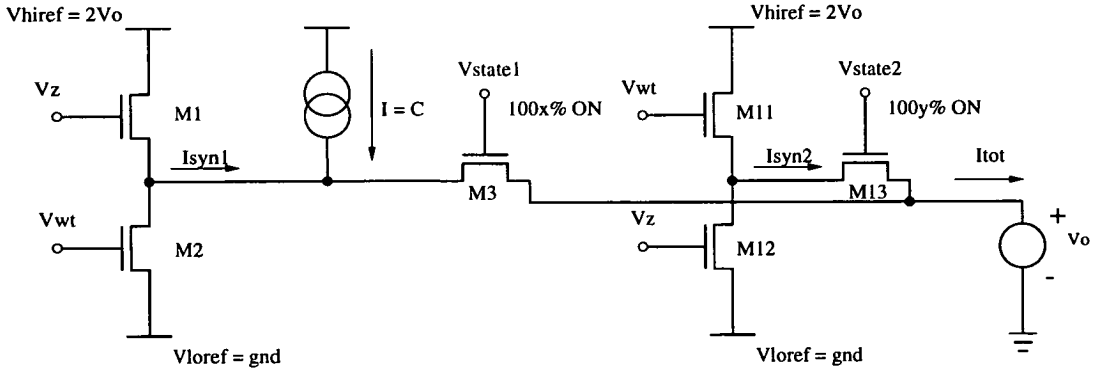


Figure E–4: *Correcting the offset*

Appendix F

Details of the two chips

This Appendix contains details of the design and testing of the two chips described in this thesis.

Details of design and test of chip described in Chapter 4

Process	:	ES2 1.5μm
Design tool	:	Cadence Edge
Test board	:	Wire-wrap board
Main controller	:	IBM PS2/30
Analogue voltage inputs	:	Unity-buffered from PSU
Analogue current inputs	:	*Via* resistors from PSU
Pulsed inputs	:	Programmable state machine. The machine, driven by its own clock, used an instruction-set, stored in EEPROM, to determine the state of digital signals in each clock-cycle. By controlling these states, a set of control and pulsed-input signals could be generated.
Output signals	:	Read into PS2 from Philips 3365 storage oscilloscope via GPIB interface

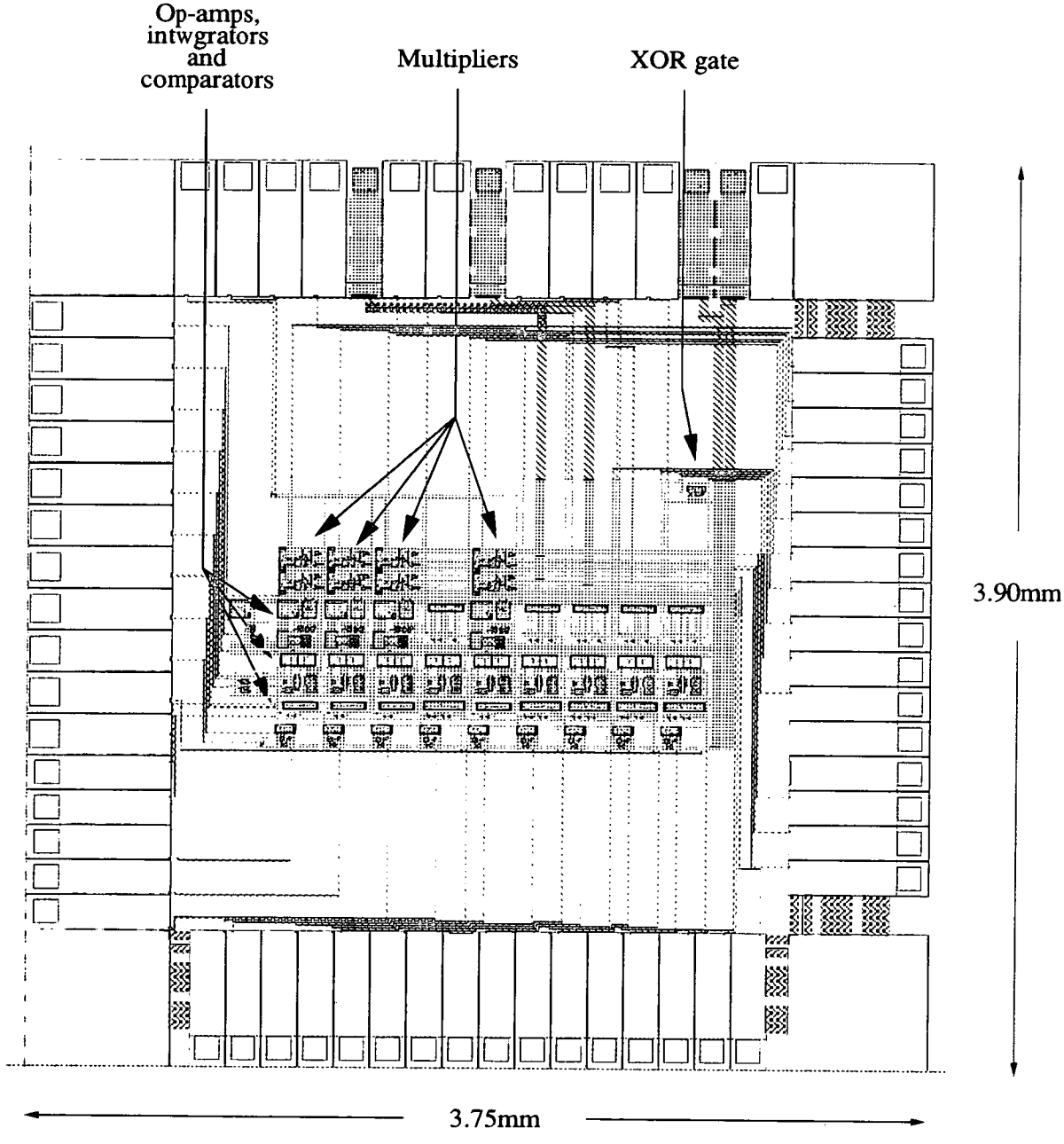


Figure F–1: *Plot of first chip.*

Testing the first chip

The chip was tested using the equipment shown in Figure F–2.

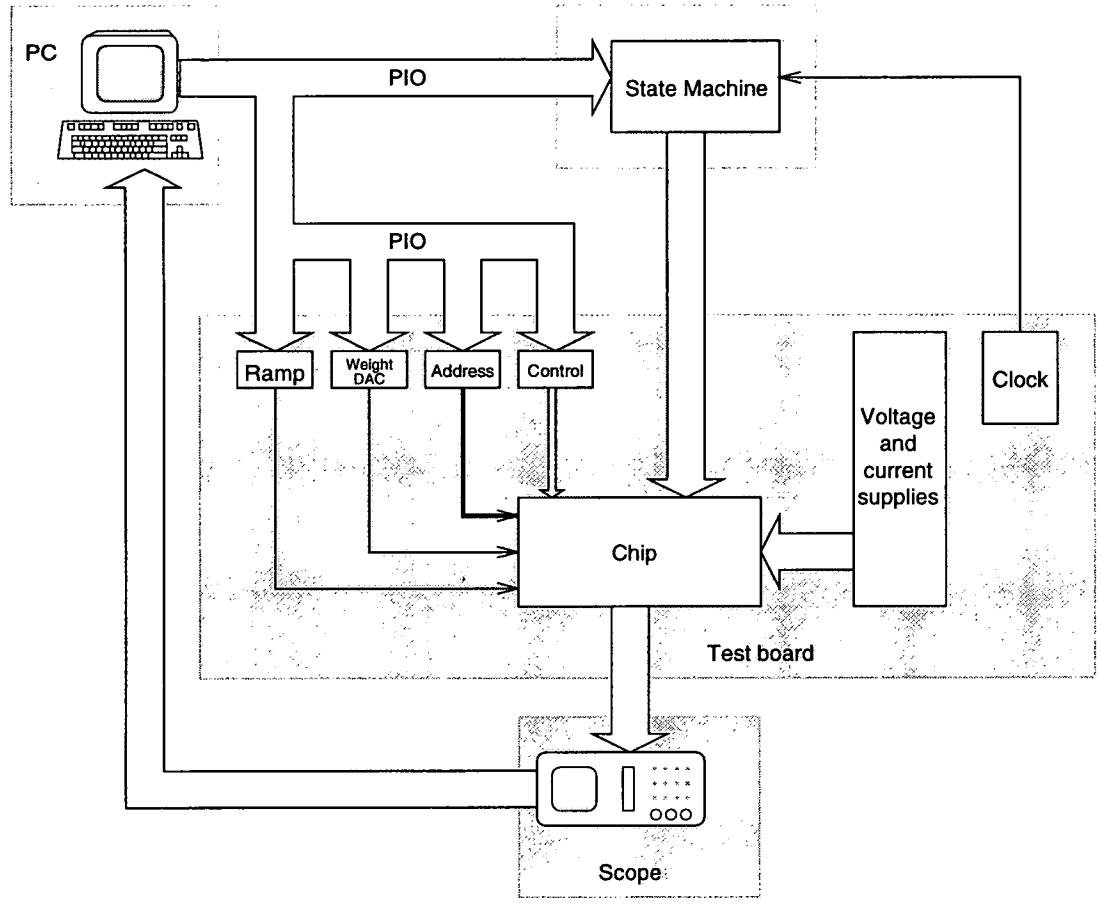


Figure F–2: *Design of system for testing the chip.*

An IBM PS/2 controlled the dynamic signals for the experiments, that is the pulse inputs; and the addressing and control signals and DAC that enabled varying weights to be loaded as charge on the weight-capacitors of the synapses. A state-machine[1], effectively a programmable ROM with its own clock, supplied the pulse

[1]Grateful thanks are due to Alister Hamilton, who had developed this state-machine for another purpose, and to Andy Myles who very kindly used his substantial *awk* skills to rewrite the compiler that drove the machine and provided a complete state-machine-code testing system.

inputs themselves, as well as a number of other control signals (enable and reset signals for the activation capacitor, and the signal to start a ramp from the ramp DAC). The static signals, that is the several current and voltage supplies necessary for the op-amp, integrator and output-pulse circuits, derived from a main power-supply; resistor-ladders, buffered by op-amps, supplied the voltages, and potentiometers controlled the currents. A digital oscilloscope captured the output pulses and sent data back to the PC via a GPIB bus.

Details of design and test of chip described in Chapter 6

Process	:	ES2 1.5μm
Design tool	:	Cadence Opus DFII
Test board	:	PCB, designed by Geoff Jackson, and wire-wrap board
Main controller	:	IBM PS2/30
Analogue voltage inputs	:	Unity-buffered from PSU
Analogue current inputs	:	*Via* resistors from PSU
Pulsed inputs	:	XILINX field-programmable, gate-array (FPGA) chip. Compiling software was run on a Sun workstation and downloaded to the FPGA chip by a serial interface. The design was done schematically, using Cadence Opus DFII; small digital schematics can be combined hierarchically from library components to produce complex circuits. An example schematic, of the driver for the shift-register which addressed the synapse array, is shown in Figure F–3.
Output signals	:	Read into PS2 from Philips 3365 storage oscilloscope via GPIB interface

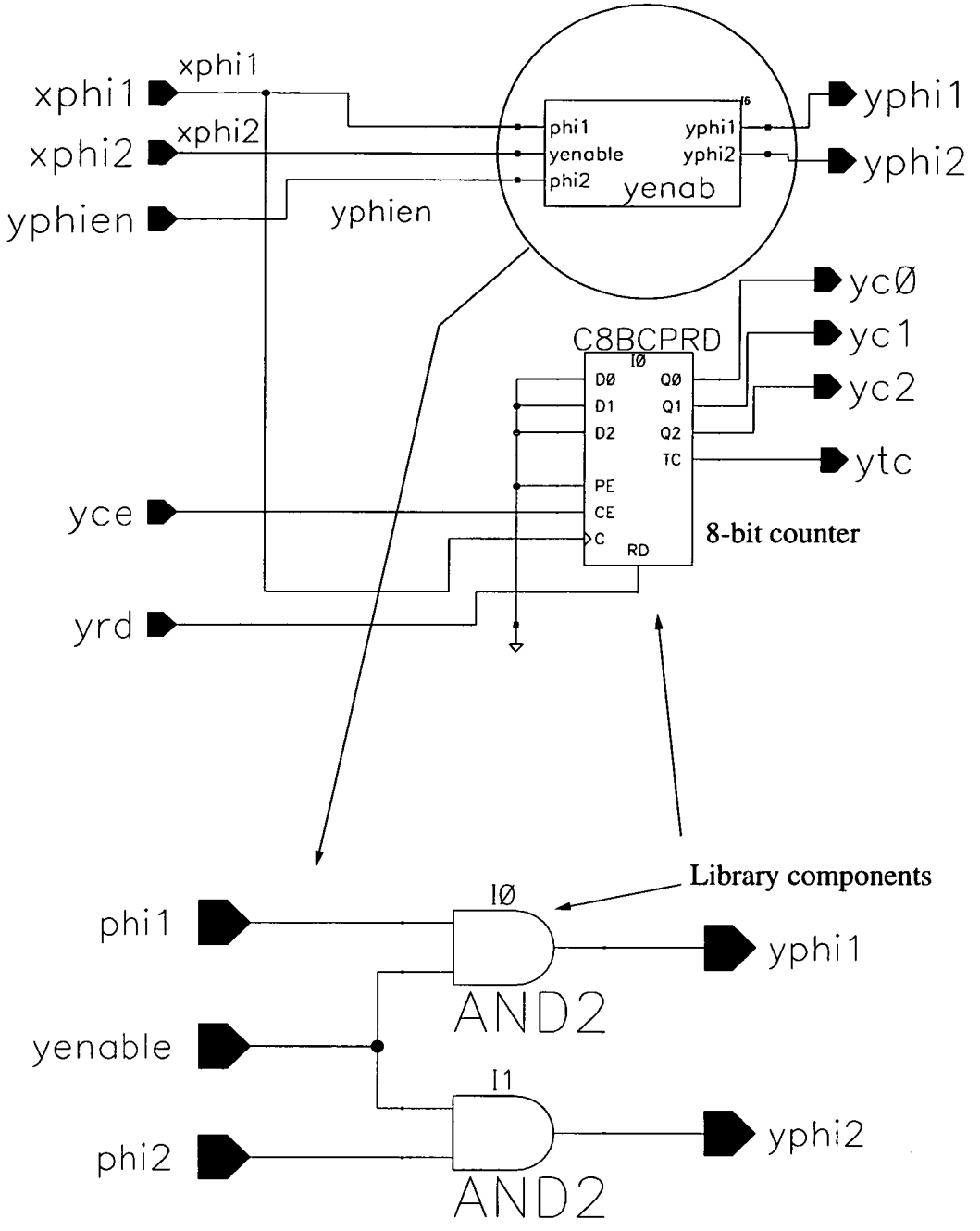


Figure F–3: *Schematic of the Y shift-register driver*

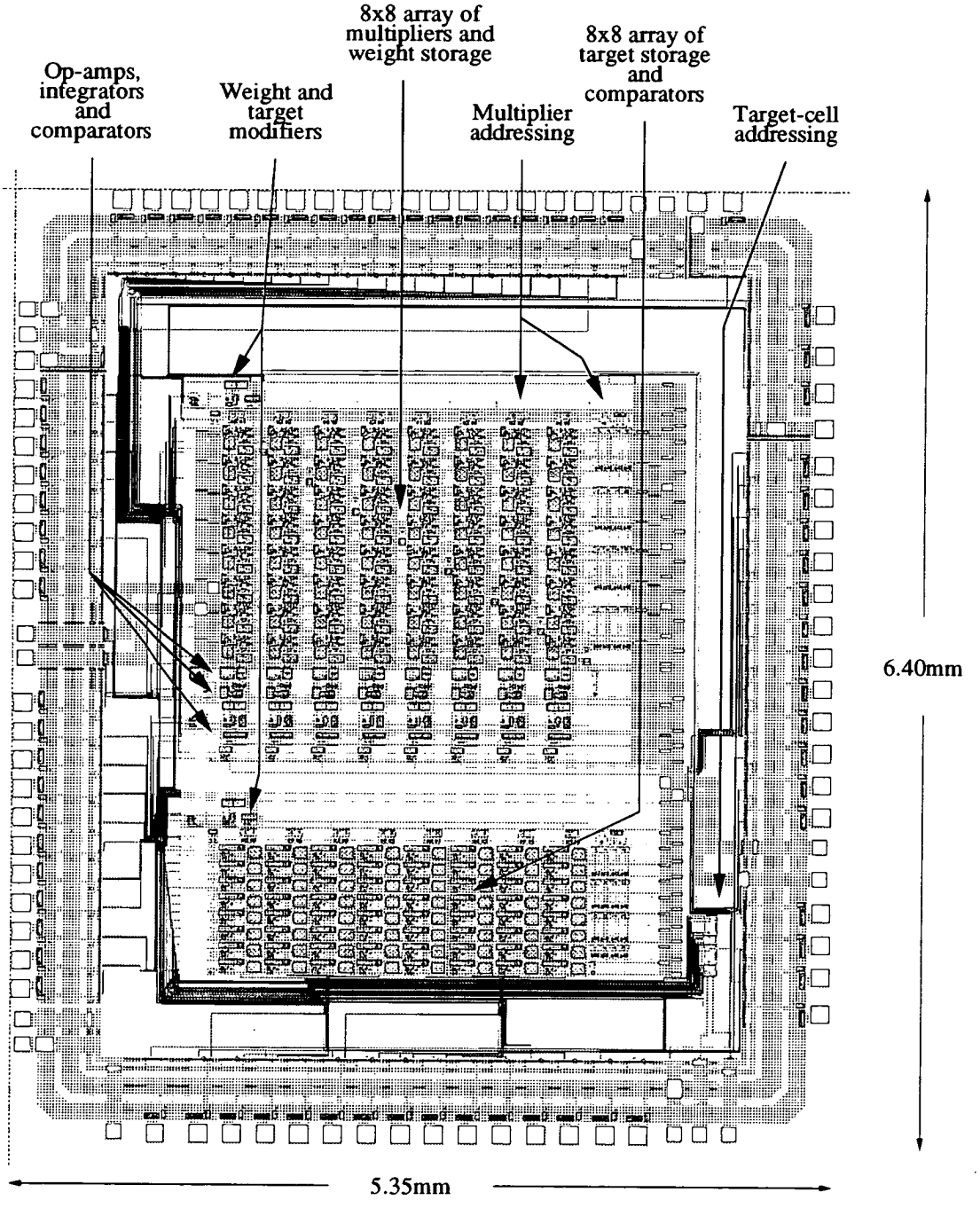


Figure F–4: *Plot of second chip.*

Testing the second chip

The chip was installed in a printed circuit board and driven by a combination of digital and analogue hardware, and by digital software down-loaded onto a Xilinx FPGA (field-programmable gate array) chip. Using the Xilinx chip meant that I could design and redesign small pieces of support circuitry to test individual parts of the chip, and then reuse these pieces in a larger and more comprehensive design to carry out trials of on-chip learning. Each digital module comprised standard building blocks such as multiplexers and flip-flops, together with a set of state machines to control the order and timing of events.

The arrangement of the various components is depicted in Figure F–5.

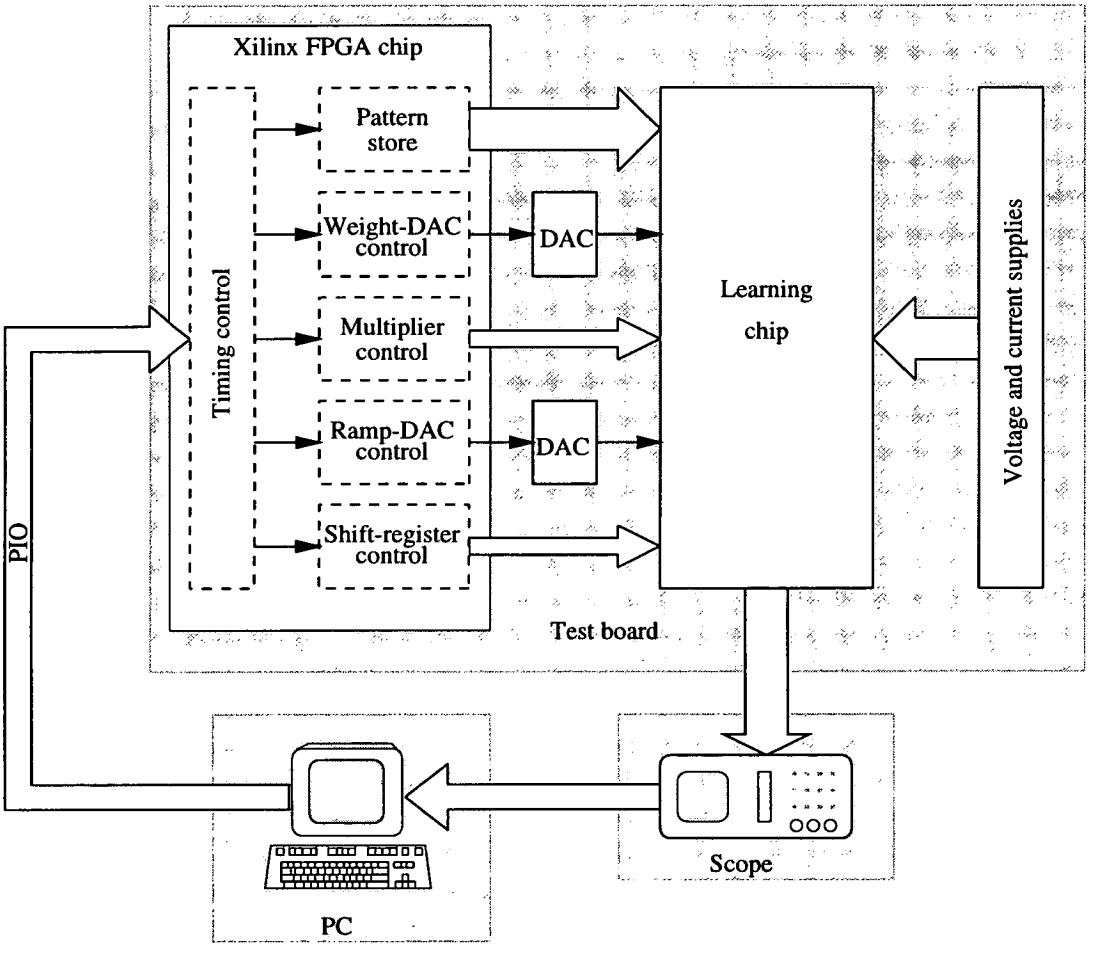


Figure F–5: *Arrangement for testing the chip.*

Appendix G

Related papers

JFIT Technical Conference, 1993

Woodburn R, Murray A F and Reekie H M, 1993. "On-chip learning in neural networks". In *Proceedings of the JFIT Technical Conference, Keele, March, 149 - 156.*

ON-CHIP LEARNING IN VLSI NEURAL NETWORKS USING HYBRID ANALOGUE/DIGITAL TECHNIQUES

Robin Woodburn, Alan F Murray and H Martin Reekie *

1 Introduction

This paper describes a test-chip design for on-chip learning in a VLSI neural network which makes the network not only programmable but also truly adaptive.

The current emphasis in neural-network implementations is on two types of programmable system. The first is fixed-function systems, where an appropriate weight-set is evolved during computer simulation and then down-loaded to the network, which is then used as a programmable, high-speed system. The second type is simulation accelerators, where the architecture may be highly flexible in the types of network which it can embody, but which is optimised for neural operations.

The aim of the current design is to move from programmability to adaptability, by creating a VLSI implementation which will be able to evolve a suitable weight-set – that is to learn – on-chip.

2 Applications of On-chip Learning

For many applications, developing the weight parameters during computer simulation may be inconvenient because of the long training times involved, but the meth-

*Robin Woodburn is the main contact, tel 031-650-5665, email rjw@ee.ed.ac.uk. All authors are members of staff of the Department of Electrical Engineering, University of Edinburgh, King's Buildings, Mayfield Road, Edinburgh EH9 3JL. The project's grant number is SERC 17060

odology is not fatal to their success. For example, financial applications, database-retrieval, hand-writing recognition and medical diagnosis may all, depending on the circumstances, be carried out 'off-line' to achieve acceptable solutions. Adaptability may not even be advantageous in these cases, since the output characteristics of the problem may have been well-defined.

In other applications, however, the input data may not be well-controlled, may arrive in large quantities in analogue form, and may require to be dealt with in real time. Examples are robotic and sensor-motor control, speech recognition, natural-language applications, process-control, image-processing and machine vision. In these circumstances, the neural system must be adaptable at high speed. The point about such systems is that they may require to respond to inputs some of which are stable or very slowly-changing, and so can be pre-learned while other inputs may be rapidly-changing and so require 'learning' or 're-learning' over a period of time. On-chip learning brings closer the prospect of systems with this capability, and hence the possibility of autonomous systems.

3 Development of On-chip Learning

The idea of on-chip learning is not new. In some senses it is an obvious next step from programmable systems : long training times on serial computer architectures make means of accelerating the process on truly parallel implementations very attractive. Furthermore, researchers have been moving incrementally towards the idea with 'chip-in-the-loop' techniques for analogue devices : the weight-matrix derived from simulation is down-loaded, and then another, short, learning phase is carried out in which the network is used for the forward computation while a supporting computer updates the weights, until the network stabilises. The resultant weight-matrix is thereby adjusted to compensate for the inevitable disparities in analogue computations due to process variance. The 'chip-in-the-loop' scheme has been used for Intel's analogue ETANN chip [1], for the back-propagation algorithm on a mixed optical and analogue-electronics network [2], and indeed for our own group's EPSILON chip as a means of implementing a variant of the back-propagation algorithm [3], to be described shortly.

4 Implementations of On-chip Learning

Several other groups are investigating the issue of on-chip learning. These investigations are primarily digital [4, 5, 6, 7, 8], although some hybrid schemes are

beginning to make an appearance [9, 10].

The digital implementations tend to be flexible schemes capable of supporting a wide variety of neural algorithms including back-propagation, as well as algorithms with similar characteristics such as those for digital signal processing. The number of neurons tends to be small (in the 10s), but multiplexing techniques and the cascadability of chips permits larger networks to be created.

The analogue systems, like the one described here, are network-specific. The number of neurons tends to be greater (high 10s and 100s), with one example being cascadable to create larger networks [11].

Analogue learning implementations raise a number of fundamental issues. For example, the back-propagation algorithm, which has given such an impetus to neural-network research and applications, is notoriously difficult to render into hardware. One reason is that weight-updates involve non-local information, which has to be fed back from the output to the hidden layer (and this can be difficult to organise efficiently in digital hardware also). Another is that the weight-update strategy for output-layer neurons is different from the strategy for hidden-layer neurons. The virtual-targets algorithm goes a considerable way towards overcoming these problems.

A whole system needs to be created in addition to the building-blocks for computation. This means that different methods for encoding states and values, and the means for feeding these signals to different parts of the network, must be considered very carefully.

Process-variations will inevitably affect the operation of the network. Although out group has had some success in confronting this problem through the design of process-invariant circuits [12], it cannot be entirely overcome. In designing the system, a judgement has to be made about what is or is unlikely to be successful. For example, in a weight-modification circuit, imprecision in computation can probably be tolerated provided the polarity of the result (and hence the direction in which the weight is adjusted) is correct. Computer and Spice simulations help develop a feel for what is right, but ultimately the silicon implementation is the only real test.

5 The Virtual-Targets Algorithm

The virtual-targets algorithm simplifies the weight-update strategy sufficiently to make hardware implementation a more practical prospect than for back-

propagation.

It is not important to understand the detail of the algorithm to appreciate its advantages over back-propagation, although the algorithm for the training phase for an $I-J-K$ network is outlined in Figure 1 and it is described in detail elsewhere [13, 14]. The first advantage is that weight-update uses only local information, simplifying the circuitry necessary for a hardware implementation.

1. Calculate initial values for the hidden-layer targets :
   (a) Apply input pattern $\{O_{ip}\}$, and read out the states $\{O_{jp}\}$ and $\{O_{kp}\}$ of the hidden and output nodes.
   (b) Assign targets $\{T_{jp}\}$ for the hidden nodes such that $\{T_{jp}\} = \{O_{jp}\}$.
2. Repeat 1 for all input patterns.
3. Present patterns in random order and allow :
   (a) weights to evolve according to the following equations :

$$\frac{\delta W_{kj}}{\delta t} = \eta_{weights} O_{jp} O'_{kp} \epsilon_{kp} \tag{1}$$

$$\frac{\delta W_{ji}}{\delta t} = \eta_{weights} O_{ip} O'_{jp} \epsilon_{jp} \tag{2}$$

   where

   - $\eta_{weights}$ is a gain-term representing weight learning-speed;
   - $\{O_{jp}\}$ and $\{O_{ip}\}$ are the inputs from the previous layer;
   - O'_{kp} and O'_{jp} represent the derivatives of the activation function (the 'sigmoid-prime' terms); and
   - ϵ_{kp} and ϵ_{jp} are the error-terms where $\epsilon_{kp} = T_{kp} - O_{kp}$ and $\epsilon_{jp} = T_{jp} - O_{jp}$.

   (b) hidden-layer targets to evolve according to the following equation :

$$\frac{\delta T_{jp}}{\delta t} = \eta_{targets} \sum_{k=0}^{K} W_{kj} \epsilon_{kp} \tag{3}$$

   where

   - $\eta_{targets}$ is a gain-term representing target learning-speed;
   - W_{kj} is a weight on the connections between the hidden- and output-layers; and
   - ϵ_{kp} is the error term where $\epsilon_{kp} = T_{kp} - O_{kp}$.

Figure 1: *The virtual-targets algorithm.*

The second advantage is that the weight-update strategies for both hidden- and output-layer neurons are identical (whereas they are different for back-propagation), which means that, once neuron circuits have been designed, they can be replicated for the whole network. The price to be paid is that, in addition to the output target-states (ie teaching patterns used in training), a target-state has to be introduced

for each hidden-layer neuron, and these hidden targets require information to be fed back from the layer above.

6 Translating the Algorithm into Hardware

The new design builds on previous work carried out by our group, [15, 3], which has proved circuits for an analogue, feed-forward network capable of providing multiply-accumulate operations for a variety of neural algorithms including virtual targets; the virtual-targets algorithm is currently realised on our EPSILON chip using the 'chip-in-the-loop' technique referred to earlier.

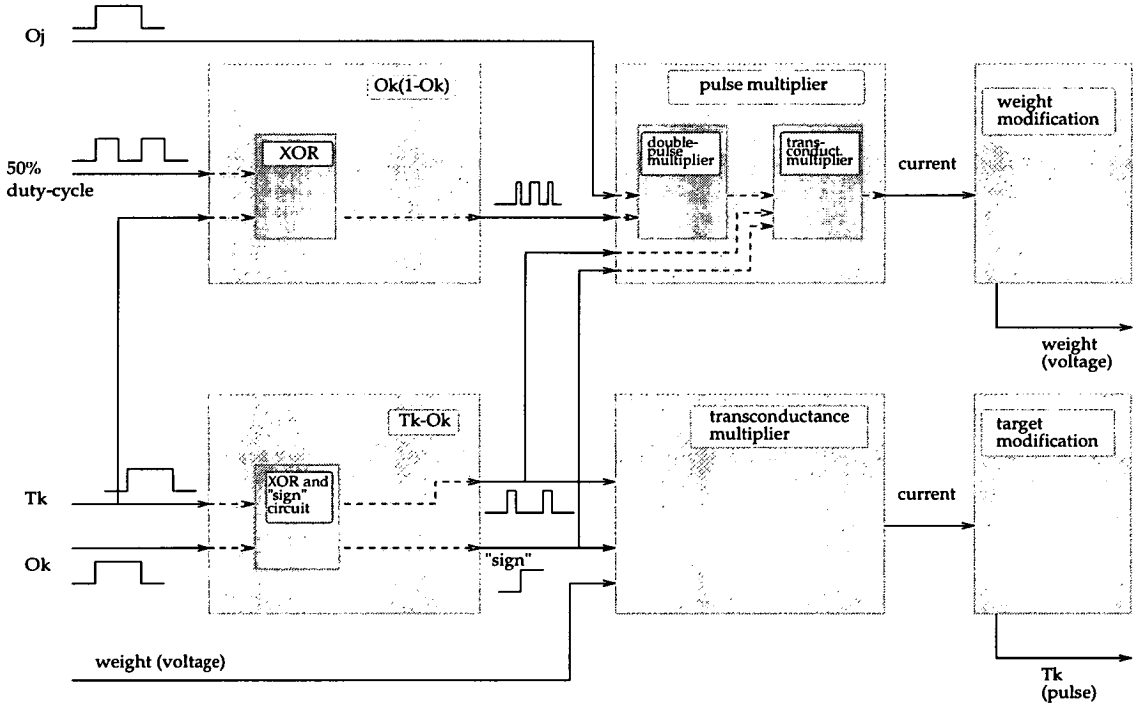


Figure 2: *Functional block diagram of the computational elements of the chip*

The design is still incomplete. However, a functional block diagram of the computational elements of the design is shown in Figure 2, and details of work so far are as follows :

Hybrid chip The chip will be a hybrid one, to reflect previous work by our group, and to blend the merits of both digital and analogue technology. Our approach is to use whichever form of technology is the best or most convenient in the circum-

stances. In general, we prefer analogue circuitry for synaptic multiplications on the grounds, firstly, that such circuitry occupies a much smaller silicon area than the digital equivalent and, secondly, that summation can be simply and elegantly achieved if the results of multiplication are currents which can be summed on a node. We prefer digital signals for neural states because they can encode analogue values on a time axis and yet are robust and easily transmitted within chips or across chip boundaries.

Neural states Neural states will take on analogue values whose instantaneous value will be represented by the width of an individual pulse. Our group has considered, and used, a number of different representations in the past [15]. However, pulse-width modulation seems, at least in the first instance, the most appropriate representation for the current scheme for on-chip learning because of the ease with which it may be employed. For the virtual-targets algorithm, a neural state O_k is continuously-valued such that $0 < O_k < 1$.

Target states Target states will be identical in form to neural states. For virtual targets, target-states T_k are continuously-valued such that $0 < T_k < 1$.

Error-signals Error-signals, which represent the difference between target- and output-states, are bipolar and continuously-valued such that $-1 < \epsilon_k < 1$. Since both the target- and output-states are represented by pulses, simple digital circuitry can be used to latch a 'sign-bit' to represent the polarity of the error computation. The magnitude can also be easily computed by applying each pair of target and output pulses to an **XOR** gate.

Sigmoid-prime The derivative of the activation function, also known as the 'sigmoid-prime', is of the form $O_k(1 - O_k)$, where O_k is a neural state. The most convenient means of representing this computation is as a function, whose input O_k will give an output which is an inverted parabola which cuts the x-axis at 0 and 1. Such an output characteristic is not very easily generated by an analogue circuit, but a reasonable approximation, the output characteristic of which is a triangular wave, can be generated by applying the input pulse O_k to one input of an **XOR** gate whose other input is stimulated by a square-wave with a 50% duty-cycle.

Weight-states Weights are held as charge on a capacitor. A weight can then be modified by adding charge to, or removing it from, the capacitor. For virtual targets, weights are continuously-valued and bipolar.

Transconductance multiplier Our group has developed a transconductance multiplier [15], with highly linear characteristics, for multiplying weights by states. The input stage of the multiplier produces a current proportional to the weight-voltage, W_{kj}, which is then pulsed by a switch-transistor controlled by the neural state, O_k. For a particular neuron, the resulting output current can be summed with

other synaptic outputs and integrated over a period of time into a voltage. This voltage represents the multiplication of W_{kj} by O_k. The multiplier operates in two quadrants, with weights being positive or negative while states are positive only. However, for the virtual-targets algorithm, a four-quadrant multiplier is required since the target-modification equation (equation 3 in Figure 1) requires multiplication of two bipolar values. One solution would be to arrange the multiplier in pairs, each with its own weight-capacitor, one of which represents a positive value and the other a negative one. The disadvantage of this approach is that process-variations may cause mis-matches in the operation of the paired multipliers, leading to inconsistencies in the results of the computations, so other possibilities are being considered.

Generating pulses to represent neural and target states As neural and target states are continuously-valued, the width of the pulses which represent the states must also be varied. Post-synaptic neural activity is represented as a voltage, while targets can also be represented as a voltage *via* charge stored on a capacitor. To generate pulses from these voltages requires only a two-stage comparator with an inverter output driver. If a ramp voltage is applied to one input of the comparator, it will produce a pulse output the width of which is set by the neural- or target-state voltage. The ramp voltage can easily be generated off-chip and globally distributed to all neurons and targets in parallel.

Remaining work This leaves two significant pieces of work still to be decided, the double-pulse multiplier, and weight-refresh circuitry. Current preference for the double-pulse multiplier is for a charge-pump system to produce a voltage which can be fed directly into the transconductance multiplier.

References

[1] Tam S M, Gupta B, Castro H A, and Holler M. "Learning on an analog VLSI neural network chip". In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Los Angeles, 701 - 703*, 1990.

[2] Frye R C, Rietmann E A, and Wong C C. "Back-propagation learning and nonidealities in analog neural network hardware". *IEEE Transactions on Neural Networks, 2 (1), (January), 110 - 117*, 1991.

[3] Churcher S, Baxter D J, Hamilton A, Murray A F, and Reekie H M. "Generic analog neural computation — the EPSILON chip". In *Advances in Neural Information Processing Systems 5, eds C L Giles, S J Hanson and J D Cowan*, 1993.

[4] Duranton M and Sirat J A. "Learning on VLSI : a general-purpose digital neurochip". *Phillips Journal of Research, 45 (1), 1 - 17*, 1990.

[5] Theeten J B, Duranton M, Mauduit N, and Sirat J A. "The LNeuro-chip : a digital VLSI with on-chip learning mechanism". In *Proceedings of the International Neural Networks Conference, Paris, 593 - 596*, 1990.

[6] Eguchi H, Furuta T, Horiguchi H, Oteki S, and Kitaguchi T. "Neural network LSI chip with on-chip learning". In *Proceedings of the International Joint Conference on Neural Networks (vol I), July 8 - 12, Seattle, 453 - 456*, 1991.

[7] Hammerstrom D. "A VLSI architecture for high-performance, low-cost, on-chip learning". In *Proceedings of the International Joint Conference on Neural Networks (vol II), San Diego, 537 - 544*, 1990.

[8] Tomberg J and Kaski K. "Digital VLSI architecture of backpropagation algorithm with on-chip learning". In *Proceedings of the International Conference on Artificial Neural Networks (vol 2), Espoo, Finland, 1561 - 1564*, 1991.

[9] Arima Y, Mashiko K, Okada K, Yamada T, Maeda A, Kondoh H, and Kayano S. "A self-learning neural network chip with 125 neurons and 10K self-organisation synapses". *IEEE Journal of Solid-state Circuits, 26 (4), (April), 607 - 611*, 1991.

[10] Salam F M A and Wang Y. "A real-time experiment using a 50-neuron CMOS analog silicon chip with on-chip digital learning". *IEEE Transactions on Neural Networks, 2 (4) (July), 461 - 464*, 1991.

[11] Arima Y, Mashiko K, Okada K, Yamada T, Maeda A, Notani H, Kondoh H, and Kayano S. "A 336 neuron, 28K-synapse, self-learning neural network chip with branch-neuron-unit architecture". *IEEE Journal of Solid-state Circuits, 26 (11), (November), 1637 - 1643*, 1991.

[12] Baxter D J. "Process-tolerant VLSI neural networks for applications in optimisation", unpublished PhD Thesis, Edinburgh University. 1992.

[13] Murray A F. "Analog noise-enhanced learning in neural networks circuits". *Electronics Letters, 2 (17), 1546 - 1548*, 1991.

[14] Murray A F. "Multi-layer perceptron learning optimised for on-chip implementation". *Neural Computation, 4 (3), 336 - 381*, 1992.

[15] Hamilton A, Murray A F, Baxter D J, Churcher S, Reekie H M, and Tarassenko L. "Integrated pulse stream neural networks : results, issues and pointers". *IEEE Transactions on Neural Networks, 3, (3) (May), 385 - 393*, 1992.

IEEE Micro, 1994

Murray A F, Churcher S, Hamilton A, Holmes A J, Jackson G B, Reekie H M and Woodburn R, 1993. "Pulse-stream VLSI neural networks". *IEEE Micro, 14, (3), 29 - 39.*

Pulse Stream VLSI Neural Networks

EPSILON, a large, working, VLSI device, demonstrates pulse stream methods in the wider context of analog neural networks. EPSILON uses dynamic weight storage techniques, but a nonvolatile alternative is desirable. To that end, we have developed an amorphous silicon memory, which we present in experiments incorporating the device in a modest pulse stream neural chip. We have also developed a target-based training algorithm, which we demonstrate in a prototype learning device using a realistic problem. Finally, we explore system-level problems in experiments with a second version of EPSILON in a small, autonomous robot.

Alan F. Murray

Stephen Churcher

Alister Hamilton

Andrew J. Holmes

Geoff B. Jackson

H. Martin Reekie

Robin J. Woodburn

University of Edinburgh

Because the pulse stream technique does not quantize information explicitly, it preserves the high resolution of analog processing. In addition, it communicates by exchange of binary pulses with fixed amplitude, thus exploiting many benefits of digital circuitry. The pulse stream technique uses digital signals to carry information and to control analog circuitry, while storing further analog information on the time axis. A number of techniques exist for coding a neural state $0 < S_i < 1$ onto a pulsed waveform V_i with frequency υ_i, amplitude A_i, and pulsewidth δ_i. An earlier work[1] reviewed these techniques.

We first used pulses in the neural context in 1986.[2] Now we bring the pulse stream story up to date. We have built large network chips using the pulse stream technique. We have developed algorithms that place the training, as well as the forward-pass computation, on the silicon substrate. We are using novel memory devices to work toward nonvolatile weight storage with fast programming time, and we have developed "stripped-down," application-oriented chips for inclusion in small mobile robots. Here, we describe these concurrent, related but distinct, projects.

First, however, we must clearly state our attitude to pulse stream methods for neural networks. They are effective, efficient, and attractive in many applications. They have a good biological precedent. But we do not claim that these methods are best in all circumstances. Rather, they form one of a range of possible neural-network techniques that includes digital,[3] weightless,[4] purely analog,[5,6] and several hybrid schemes. This list is far from exhaustive.

Pulse stream hardware: EPSILON

The EPSILON (Edinburgh Pulse Stream Implementation of a Learning-Oriented Network) chip is a 3,600-synapse section of a neural network implemented in 1.5-μm CMOS (complementary metal oxide semiconductor) technology. With the goal of building as large a neural network as possible in silicon, we used the following circuits.

Synapses. The synapse design is based on the standard transconductance multiplier circuit, previously the basis of monolithic analog transversal filters in signal-processing applications.[7] Such multipliers use MOS transistors in their linear region of operation to generate output currents proportional to the product of two input volt-

0272-1732/94/$04.00 © 1994 IEEE

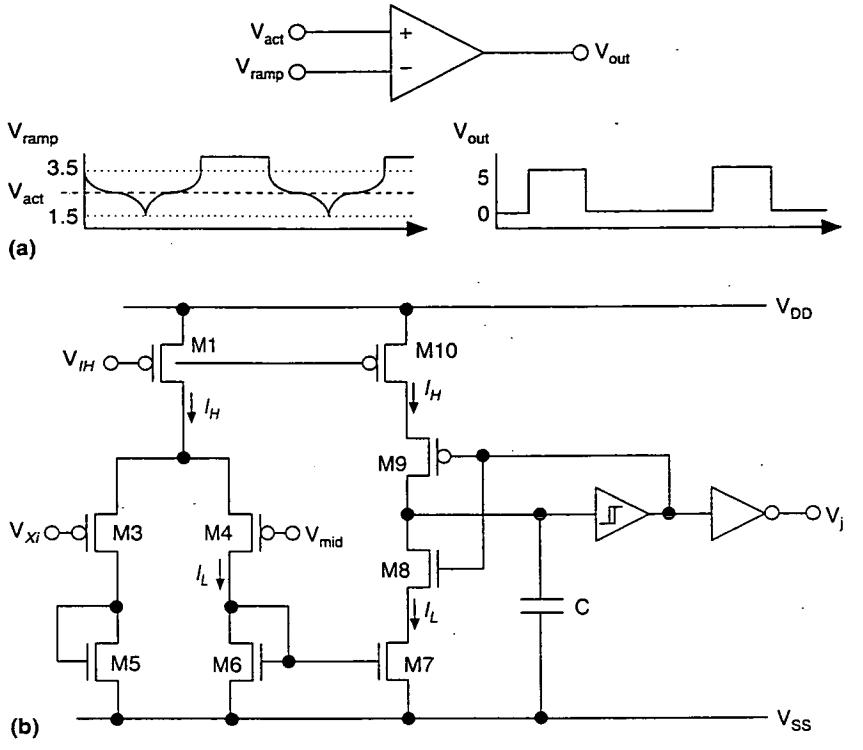


Figure 1. Pulsewidth modulation neuron (a); pulse frequency modulation neuron (b).

ages. We adapted this concept for pulsed neural networks by fixing one of the input voltages and using a neural state to gate the output current. In this manner, the synaptic weight controls the output current's magnitude, which is multiplied by the incoming neural pulses. The resultant charge packets subsequently integrate to yield the total postsynaptic activity voltage.

The linearity of the synaptic output, as a function of input state, is very high.[8] The variation of synapse response with synaptic weight voltage is fairly uniform. Therefore, the design contains the effects of across- and between-chip process mismatches tolerably well.

Neurons. To reflect the diversity of neural network forms and possible applications, we included two different neuron designs on the EPSILON chip. We designed the first, a synchronous pulsewidth modulation neuron, with vision applications in mind. This circuit can guarantee network computation times, thereby eliminating the data dependency inherent in pulse frequency systems. The second neuron design uses asynchronous pulse frequency modulation. Although hampered by data-dependent calculation times, its wholly asynchronous nature makes it ideal for neural network architectures that embody temporal characteristics—feedback networks and recurrent networks. Like the synapse, both circuits minimize transient noise injection and tolerate process variations.

Pulsewidth modulation. The main disadvantage with this technique appears to be its synchrony; neurons all switch together, causing larger power supply transients than in an asynchronous system. We circumvented this problem, however, with a double-sided pulse modulation scheme.

Figure 1a illustrates the operation of the pulsewidth modulation neuron. The neuron itself is a two-stage comparator with an inverter output driver. The inputs are the integrated postsynaptic activity voltage V_{act} and the reference voltage V_{ramp}, generated off chip and distributed to all neurons. As Figure 1a shows, the output changes state whenever the reference signal crosses the activity voltage. The shape of the reference signal determines the transfer function; when the signal is generated by a RAM lookup table, the function is user programmable. Figure 1a shows the signal that should be applied for a sigmoidal transfer characteristic. The sigmoid signals are "on their sides" because the input (or independent variable) is on the vertical axis. The use of a double-sided ramp voltage generates a symmetrical pulse at about the midpoint of the ramp, reducing the occurrence of coincident edges and relieving the problem of switching transients on the power supplies. Furthermore, because the analog element (that is, the ramp voltage) is effectively removed from the chip, and the circuit itself merely functions as a digital block, the system is immune to process variations.

Pulse frequency modulation. Figure 1b illustrates the second neuron design, basically a voltage-controlled oscillator (VCO) with a variable-gain sigmoidal transfer characteristic. The circuit achieves oscillation via the hysteretic charge and discharge of capacitor C by currents I_H and I_L . I_H sets the constant output pulsewidth, while I_L controls interpulse spacing (and hence output frequency). I_L itself is determined by the activity voltage V_{Xi} via the differential stage of transistors M3 to M6. This differential stage gives the VCO its sigmoidal characteristic, and additional current injected and removed at appropriate points in this stage produces gain variations (Figure 1b omits this circuitry for the sake of clarity).

EPSILON specifications. The synapse and neuron circuits underpin the EPSILON chip, fabricated by European Silicon Structures using its ECPD15 (1.5-µm, double-metal, single poly-CMOS) process. Each chip uses a single layer of synaptic connections and accepts inputs as either analog voltages (for direct interface to sensors) or pulses (for communication with other chips and with digital systems). Table 2 on page 36 gives the full EPSILON specifications.

Demonstration. EPSILON's first real-world problem was the implementation of a 54:27:11 multilayer perceptron (MLP) to classify 11 different vowel sounds spoken by each of 33 speakers. Analog outputs of 54 band-pass filters formed the input vectors.

We initially trained the MLP on a Sun Sparcstation, using a subset of 22 patterns. Learning used the virtual-targets algorithm, with 0 percent noise.[9] Next we downloaded the weight set to EPSILON. Then we restarted the training, but this time we used EPSILON to evaluate the forward-pass phases of the network. At the end of training, EPSILON identified all 22 training patterns.

Subsequently, we presented 176 unseen test patterns to the MLP, which correctly classified 65.34 percent of these vectors. This compared very favorably with similar generalization experiments carried out on a Sparc, in which the best result was 67.61 percent.

EPSILON's second major application consisted of investigations into image region labeling with an MLP. The problem: Given a set of features derived from an image region, use those features, combined with similar features from neighboring regions, to classify the region.

In particular, we trained an MLP to discriminate between regions that were roads and those that were not roads. The experiments were relatively straightforward, consisting of comparisons between forward-pass (also called recall mode) generalization abilities of networks implemented on EPSILON and as computer simulations. We developed six different synaptic weight sets for a 45:12:2 MLP from six different training data sets on a Sparc workstation. We measured the generalization capabilities of the weight sets against six different test vector sets (one for each weight set). Table 1 shows the results.

EPSILON's performance compares well with equivalent Sparc simulations. Indeed, its mean generalization ability was only 4 percent below that of the simulated networks—certainly within one standard deviation. It achieved this performance without the need for chip-in-loop training.

Table 1. Comparison of EPSILON's generalization performance and equivalent simulations.

Network implementation	Regions correct (Mean %)	(Std. dev. %)
EPSILON	63.57	4.86
Simulation	67.56	8.33

Choice of an on-chip learning algorithm

On-chip learning is an essential feature if network chips are to become autonomous neural systems addressing real-time, real-cost applications. Furthermore, in embedded systems, where analog VLSI technology finds its strongest justification, the chip's ability to adapt in situ is almost essential.

Several groups are investigating on-chip learning, primarily in the digital domain, although some schemes, like our own system, use hybrid digital-analog technologies. Digital systems are flexible and support a variety of neural algorithms including back propagation. Network size (in other words, the number of neurons) is restricted, however. Digital circuitry is area hungry.

Analog neural systems take advantage of the technology's greater compactness to implement larger numbers of neurons (up to several hundred) and can be cascaded to create larger networks. However, they support a restricted range of algorithms. Although the EPSILON chip can implement several algorithms, it performs only the forward pass, and an associated computer performs the learning phase proper. This chip-in-loop training[5] is a successful, although time-consuming and clumsy, process.

If an analog chip can support only one algorithm, that algorithm must be selected with care.

Researchers have made considerable progress recently in translating the back-propagation algorithm into hardware.[10] We have used an algorithm called virtual targets, one of a family of target-based algorithms related to back-propagation.[11] This algorithm, although far from a panacea for all the problems of VLSI learning, has two great advantages. First, the means of updating weights uses only local information, simplifying the circuitry necessary for a hardware implementation. Second, the weight-updating strategies for both hidden- and output-layer neurons are identical (they are markedly different with respect to back propagation). Identical neuron circuits can be replicated for the whole network, and the virtual-targets chip is inherently more flexible in supporting different MLP architectures.

In developing a virtual-targets test chip, we concentrated on three issues:

- compact, four-quadrant, pulse stream multiplication;
- implementing the derivative of the sigmoidal activation function $s(x)$, $s'(x) = \partial s(x)/\partial x$ with respect to neural activity x; and
- the minimum value by which a weight can be adjusted.

Four-quadrant multiplication. Transistors M1, M2, and M3 in Figure 2 form the transconductance multiplier referred

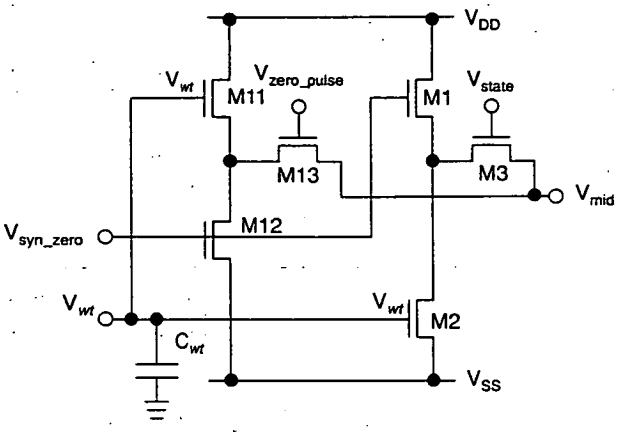


Figure 2. Four-quadrant transconductance multiplier.

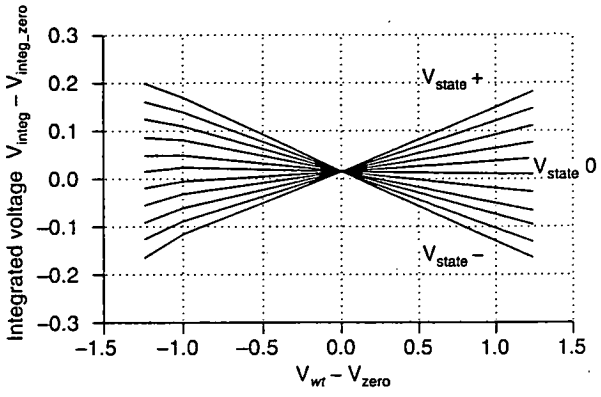


Figure 3. Four-quadrant characteristics of transconductance multiplier (simulation results).

to earlier with highly linear characteristics for multiplying weights by states. V_{DD}, V_{mid}, and V_{SS} are chosen such that M1 and M2 operate in their linear regions. The gate voltage of M1 is fixed to provide a zero point. When the gate voltage of M3 is held at V_{DD} so that M3 conducts, and the weight, represented by charge on capacitor C_{wt}, is at a low point on the range, the current through M2 is smaller than that through M1. The V_{mid} node serves to sink the excess, positive current. If the weight is at a high point on the range, V_{mid} sources a corresponding current through M2. Hence, the current through the V_{mid} node varies linearly and inversely with the charge on the capacitor.

Pulses on M3's gate voltage, which represent neural states (pulse duration represents the neuron's activation level), can control the current sourced or sunk by V_{mid} in time. Integrating the resulting current pulse (whose amplitude reflects the weight and whose duration represents the state) over time computes the multiplication of weight by state. The summation of a number of these computations can be achieved simply and elegantly on a single circuit node.

Although this circuit, which can be laid out on silicon very compactly, has given excellent performance, it is two-quadrant in nature because neural states are unipolar and weights are bipolar. For the virtual-targets algorithm, a four-quadrant multiplier is essential because in addition to a weight-by-state multiplication, a weight-by-error multiplication is necessary, and weight and error values are both bipolar.

Transistors M11, M12, and M13 (Figure 2) are the solution providing four-quadrant multiplication. These additional transistors form a second multiplier in parallel with the first, but this time the V_{syn_zero} and V_{wt} connections are reversed. To provide a zero-state point, analogous to the zero-current point set by V_{syn_zero}, the pulse applied to M13 is fixed in duration at 10 µs—midway between the shortest pulsewidth (0 µs) representing a negative error and the longest pulsewidth (20 µs) representing a positive error. At the same time, a pulse that can vary in duration from 0 µs to 20 µs is applied to M3. Just as before, integrating the resultant current pulses in time computes a four-quadrant, error-by-weight multiplication, as shown in Figure 3.

The $s'(x)$ function. Weight changes evolve in the virtual-targets algorithm according to equations given in an earlier article.[11] One of the terms in a weight change equation is the derivative of a sigmoidal activation function $s'(x)$. This function takes the form $O' = O(1 - O)$. Circuits that will give both a sigmoid and its derivative already exist.[12] In our implementation, input and output states are represented by digital pulses whose duration reflects the activation level.

Again, the circuit is gratifyingly simple, since $s'(x)$ does not have to be exact for learning to be successful. An XOR gate performs computation. One gate input receives a pulse of fixed duration (10 µs in our scheme), while the other receives a pulse that varies symmetrically around the first—in other words, from 0 µs to 20 µs. The gate's output is integrated by a differential circuit, whereby an on pulse sources a charge onto a capacitor, while an off pulse removes the charge, as illustrated in Figure 4. When the integration is complete, the output is a triangular function of the duration of the variable pulse.

Figure 5a shows the ideal form of $s'(x)$, constrained on the x axis to values between 0 and 1 by the sigmoid activation function. Figure 5b shows a triangular approximation to $s'(x)$, obtained by the XOR circuit. The integrator-capacitor voltage can be converted back to a pulsewidth signal; Figure 5c shows this final approximation to $s'(x)$.

Weight adaptation. Since weights are held as charge on a capacitor, weight adaptation requires that this charge be incremented or decremented by very small amounts.

Weight adaptation involves several important issues. The first is accuracy of the computation, defined as how closely the result matches expectations. Clearly, if a weight is to be changed, the computation on which the change depends should be accurate. A second, more important issue is the direction of change. Although some inaccuracy in computation is tolerable, it is vital that a computation aiming to increase a weight does in fact increase it. The third issue is precision, defined as the degree of agreement of repeated measurements of a quantity.

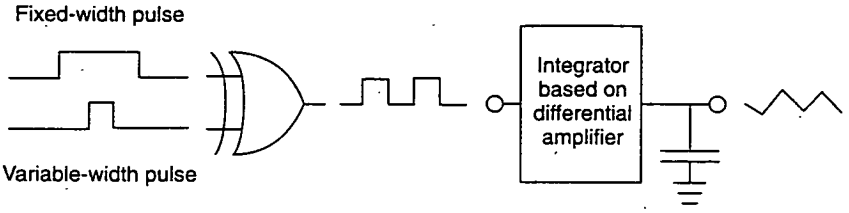


Figure 4. Integration of pulses from XOR gate.

We plan to resolve these issues by careful analysis of the properties of the circuits fabricated and equally careful simulation and analysis of their idiosyncrasies as part of a learning network. The chips developed using the virtual-targets algorithm will be amenable to integration into truly autonomous (for example, robotic) systems, where the ability to learn *in situ* will be essential.

Memory devices for pulse stream synapses

In dynamic-storage design such as EPSILON weights are stored as voltage on storage capacitors. The fact that this voltage will decay with time necessitates some form of external refresh circuitry, with a consequent increase in system complexity. Thus, a technique that allows nonvolatile, on-chip storage of synaptic weights is highly desirable. For this reason, a number of neural network designs use EEPROM (electronically erasable programmable read-only memory) technology.[5]

In most of these schemes a synaptic weight is stored as the difference in threshold voltage between two floating gate transistors. This technique allows both inhibitory and excitatory weights to be implemented. But programming EEPROMs is a slow process. We have developed an alternative approach using amorphous-silicon (αSi:H) analog memory devices for fast, nonvolatile weight storage.

αSi:H analog memory devices. Researchers at Dundee and Edinburgh Universities developed the αSi:H analog memory during a long-standing program of research into the switching properties of thin αSi:H films.[13,14] The device we are currently working with consists of a 0.1-μm-thick layer of αSi:H sandwiched between vanadium and chromium electrodes, as shown in Figure 6, next page.

After an initial forming process consisting of a series of relatively high voltage pulses, the device can be programmed into a resistance state between 1 kohm and 1 Mohm. The programming pulses are typically 120 ns in duration with a magnitude of between 2V and 6V. The physical changes that take place during forming and programming are not yet completely clear. What is certain is the following:

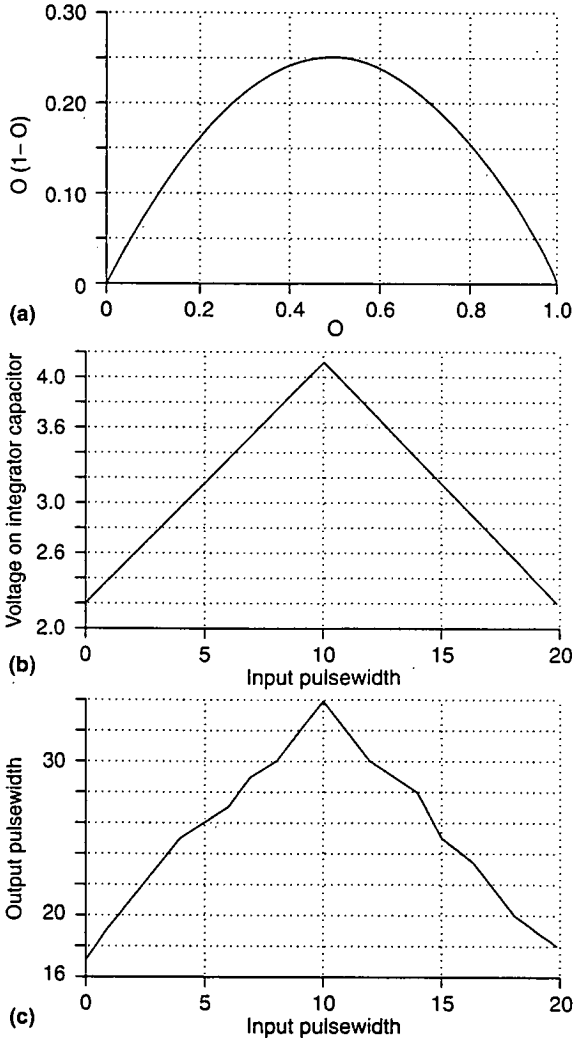


Figure 5. Ideal form of $s'(x)$ function (a); output after pulses from XOR gate are integrated for input pulses of different duration (b); output after integrated computations are converted from a voltage to a pulsewidth signal (simulation results) (c).

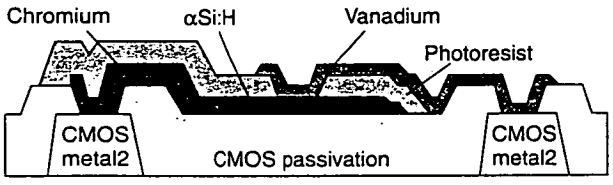


Figure 6. αSi:H memory fabricated on surface of conventional CMOS chip.

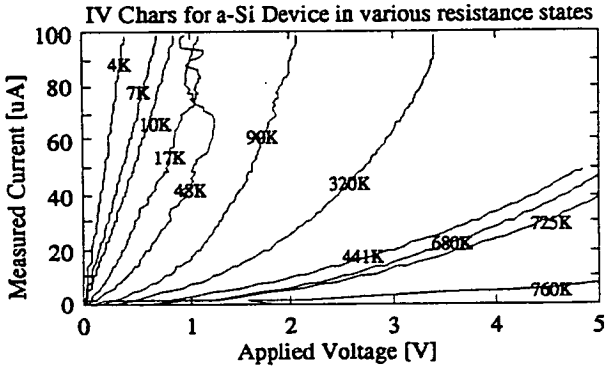


Figure 7. αSi:H memory in various resistance states.

- The forming process creates a submicron vertical filament through the αSi:H layer. This filament contains vanadium, through which conduction proceeds.
- The active volume for conductivity changes is extremely small—perhaps only a few atomic spacings in depth.
- The programmed memory is nonvolatile (for months) and shows no signs of fatigue with repeated reprogramming.

αSi:H and neural networks. British Telecom Research Labs first demonstrated the αSi:H analog memory device in the neural context. Researchers used a chip containing an array of αSi:H devices to provide the synaptic coupling to a bank of external operation-amplifier neurons. They built a test board with a 10×10 array of αSi:H devices to solve the benchmark XOR problem. While the system demonstrated that αSi:H could provide synaptic weight storage, this simplistic approach has some disadavantages. For example, it requires external neuron circuitry and allows only positive weights.

To overcome these problems, we used the αSi:H device to replace the storage capacitors in the EPSILON design while retaining the pulse stream synapse and neuron circuits. As a first step, we had to verify that the αSi:H and CMOS circuits could be integrated on one substrate. We designed and fabricated a test chip with the αSi:H devices on top of the CMOS layers (including passivation), as shown in Figure 6. This chip included various CMOS test circuits, along with some straightforward two-terminal memory devices. Figure 7 shows a set of current-to-voltage characteristics from one of these memory devices in various resistance states.

The test chip's function was simply to probe the problems associated with fabricating αSi:H devices on a CMOS substrate and subsequently programming them via CMOS circuitry. Our many areas of concern included potential damage to the CMOS circuitry by the αSi:H fabrication and forming processes, and difficulties in providing an appropriate programming waveform through MOSFETs (metal-oxide semiconductor field-effect transistors). After investigating these problems and developing solutions, we designed and fabricated a synapse circuit that actually uses αSi:H for weight storage.

The αSi:H synapse. Our latest chip includes five different synapse designs, each represented by a test block on the chip. Each test block contains four αSi:H synapses and one neuron. Although each design is slightly different, Figure 8 summarizes the basic operation of a synapse. The αSi:H device stores what is effectively a weight current. The weight current is subtracted from a zero current and is then gated by the input pulsewidth-modulated signal. The resultant charge packet is then summed on the integration capacitor. To convert the accumulated integration voltage into a pulsewidth signal, we use a neuron that is effectively a comparator. Applying a ramp voltage to the noninverting terminal causes the output to change state when the value on the integration capacitor is exceeded. This produces a pulsewidth-encoded output.

Testing of this new chip is incomplete, but we have verified the operation of the CMOS synapse and neuron circuits by using external carbon resistors in place of the αSi:H memory devices.

Real-world applications

Although we can test a VLSI device simply by verifying that the output currents and voltages are as they should be, the true test is its performance in practical use—its ability to carry out applications. We are working to make EPSILON the core of a neural network system suitable for application-based implementations. We are using this system-level approach combined with the advances in analog memory and on-chip learning described earlier to provide a framework for evaluating these emerging techniques in an application-based environment.

System requirements. We do not envisage that neural networks will ever replace conventional digital computing but rather that neural processing will serve as an interface between the real world and digital computing. To achieve

this, we must package analog neural processing power so as to interface to a conventional host processor while demanding as little computational overhead as possible when embedded in an application. For EPSILON and its derivatives, computational overhead comes in the following forms:

- weight memory management,
- learning algorithm implementation,
- pulse modulation conversion,
- chip timing and control, and
- data communication.

Technological developments such as the αSi:H memory and the on-chip learning algorithms may help solve the first two overhead problems. However, the other three require some form of processing and control external to the chip. Therefore, we have devised a system-level framework with the following functions:

- control of one or more pulse stream neural processors;
- communication between pulse stream neural processors;
- communication between pulse stream neural processors and the host processor;
- possible addition of specialized training and/or learning processors; and
- possible addition of specialized signal-conditioning cards—for example, image-processing cards.

The pulse stream neural processor will

- perform autonomous weight refresh,
- perform all data conversions to and from pulse code modulation for data communication with the host, and
- require minimum external control signals and processing overhead.

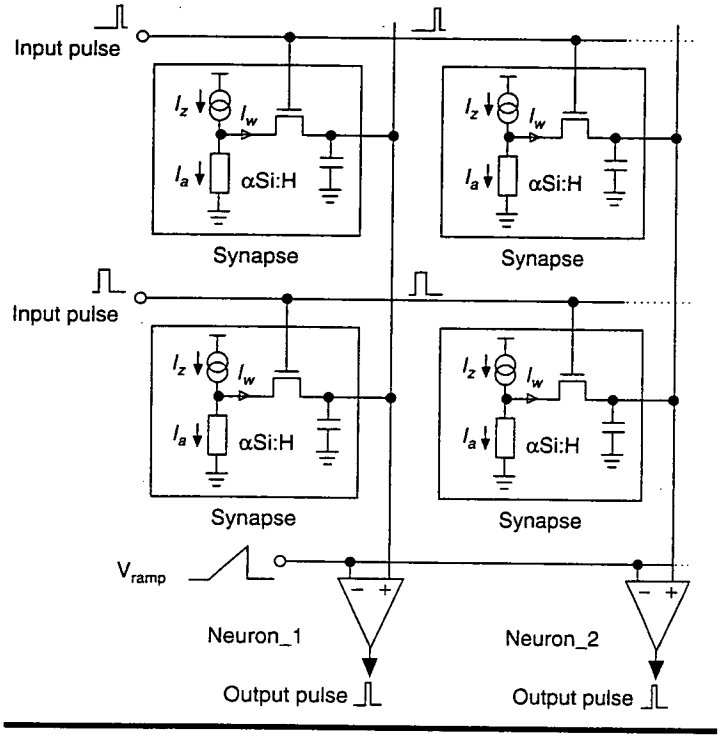


Figure 8. Pulse stream synapses incorporating αSi:H memory devices.

EPSILON II. To aid the development of a pulse stream neural network system, we have designed a new chip, EPSILON II. A smaller, more efficient device, EPSILON II embodies an optimal approach to an application in which analog VLSI circuitry is likely to be preeminent: integrating relatively small networks as compact, analog-input devices.

EPSILON II consists of a single-layer network with up to 32 input and 32 output neurons. Neural state inputs to the device may be either analog, for direct sensor interfacing, or pulses modulated in width (PWM) or frequency (PFM), for cascading chips in multilayer networks. State outputs are therefore either PWM or PFM signals.

Architecture improvements. The circuits performing EPSILON II's synapse and neuron functions are almost identical to those on EPSILON. We made minor circuit and layout modifications, principally to the self-biasing circuits, to improve detailed performance. Significant changes of the chip's architecture make EPSILON II much more flexible than its predecessor. Table 2, next page, compares the features of the two chips.

We changed the EPSILON architecture to increase the ease of implementing diverse neural algorithms. The 32-input, 32-output architecture allows simple interfacing to digital systems. We added digital recovery of analog input data by converting each of the 32 analog input channels into PWM form and feeding this digital data via the 32 output pins to external hardware. This improvement facilitates efficient implementation of learning procedures that require knowledge of input data, such as the back-propagation learning algorithm.

Each neural state input is individually programmable as either an analog or a digital signal so that raw analog sensor data can be fused with preprocessed digital data from other EPSILON II devices or separate digital data sources.

The temporal coding of PFM signals makes possible the implementation of neural feedback structures. To this end, we made the activity of each neuron individually program-

Table 2. Comparison of EPSILON and EPSILON II specifications.

Characteristics	EPSILON	EPSILON II
No. of state input pins	30	32
No. of actual state inputs	120, multiplexed in banks of 30	32
Input modes	Analog, PWM or PFM	Analog, PWM or PFM
Input mode programmability	All analog/all digital	Bit-programmable
No. of state outputs	30, directly pinned out	32, directly pinned out
Output modes	PWM or PFM	PWM or PFM
Digital recovery of analog inputs	No	Yes, PWM-modulated
No. of synapses	3,600	1,024
Additional autobias synapses	None	4 per output neuron
No. of weight load channels	2	1
Weight load time (ms)	3.6	2.3
Weight storage	Dynamic	Dynamic
Programmable activity voltage	No	Yes
Maximum speed (cps)	360 million	102.4 million
Technology	1.5 μm CMOS	1.5 μm CMOS
Die size	9.5 × 10.1 mm	6.9 × 7 mm
Packaging	144-pin PGA	120-pin PGA
Maximum power dissipation (mW)	350	320

cps: connections per second

mable to allow initialization of the network for use in these structures.

Analog performance improvements. Circuit techniques that minimize performance variation of the original EPSILON chip's individual analog components have proved successful. We have, however, measured variation across large arrays of synapses. In part, we attribute them to power supply variations across the chip, but the whole issue of process sensitivity in large analog-synapse arrays remains a fertile research area.

EPSILON II incorporates measures for reducing power supply variations across the chip and improvements of the circuits that automatically set up chip bias voltages. It also incorporates the autobias technique developed on the original EPSILON. The autobias technique dedicates a number of synapses associated with each output neuron to set the zero point of that neuron, thus removing any residual mismatches. By defining a suitable zero point, the autobias technique ensures that multiplying a zero state vector by a zero weight vector produces a zero output. EPSILON II specifically dedicates an additional four synapses per output neuron to the autobias task. While this technique sets the zero point of all output neurons, it has no effect on the gain of the synapse-neuron combination, which is also prone to variation. At present, we rely on the learning algorithm to adjust for such variations.

Pulse stream system framework. To allow flexibility in the development of applications of EPSILON-like devices, we are developing a generic system framework for pulse stream neural processing (Figure 9). The framework uses low-cost, industry-standard, bus-based processing cards in a rack system into which an EPSILON card can be plugged. The standard digital bus controls the EPSILON devices, and the conventional digital processor provides the raw computing power necessary for the learning process. Although the digital processor currently performs learning, the framework will eventually include the developments in on-chip learning and nonvolatile weight storage technology described earlier. The addition of a dedicated analog bus allows communication and pipelining between EPSILON cards and direct access to the real world.

The framework provides an environment that we can reconfigure rapidly to assess neural network hardware solutions to a multitude of problems. The system's modularity and a wide range of commercially available analog and digital cards largely eliminate the need for extensive circuit board design.

Instinct-rule robot application. Edinburgh University's Department of Artificial Intelligence has developed a small, autonomous, instinct-rule robot based on a software exemplar.[15] It is a powerful demonstration of the EPSILON II chip's use in the generic system framework. Figure 10 illustrates the robot schematically, including the EPSILON II device that serves as the essential programmable neural link between

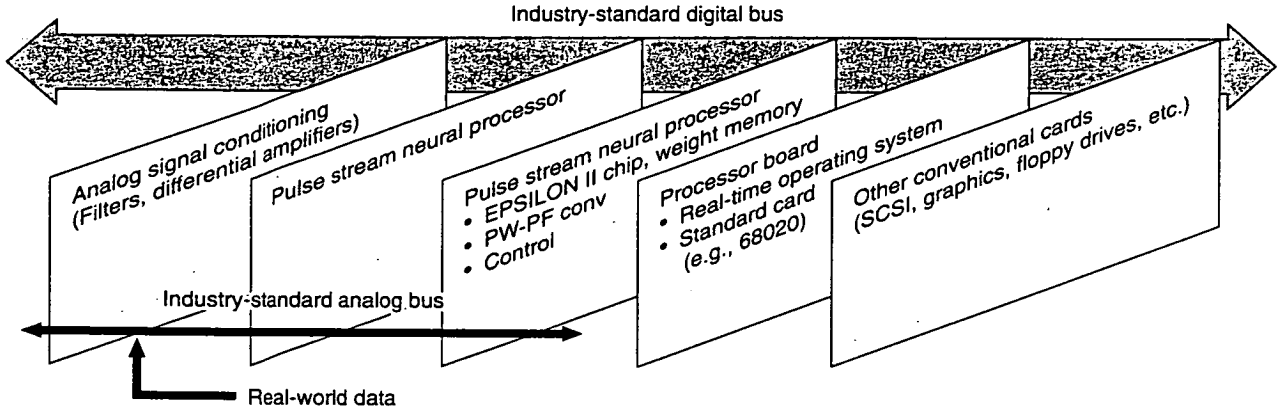


Figure 9. Generic framework for pulse stream neural computation.

the analog sensors and the instinct-rule base that underpins the robot's adaptive behavior.

The instinct-rule robot registers forward motion by means of two front-mounted feelers and a simple detector on the free-rotating front caster. The feelers are simple binary switches that give the robot an indication of obstacles in its path. A pattern associator neural network links the sensor data to the instinct-rule controller. Instinct rules such as "Keep crash sensors inactive" and "Get bored—change direction" allow the robot to learn simple behavior such as following a corridor. In real applications, such a system could provide fail-safe behavior for a more complex robot whose full guidance system failed. The additional use of historical information allows the robot to perform maze-following tasks.

We are extending the sensitivity and range of sensors interfaced to the neural network to increase the scope of the instinct rules. For example, the use of force-sensitive resistors as bend sensors will allow the implementation of an analog feeler. A directional photodiode and the instinct rule "Keep light sensor active" will allow the robot to learn to follow a light source.

EPSILON II operates at the boundary between the analog real world and the digital world of conventional computing. The main advantages of an analog VLSI solution to neural network applications are evident in our robotic application. They include the following:

- Direct interfacing to analog signals without analog-to-digital converters and analog signal multiplexing, resulting in economies of system size, speed, and power consumption.
- The ability to fuse direct analog sensor data with digital sensor data processed elsewhere in the system. In the robot application, this digital data may be historical sensor data or data conventionally processed from the camera.
- Hardwired neural algorithm. There is no need to program the neural algorithm in software because it is hardwired in VLSI circuitry. A host processor currently performs learning off chip.
- Distributed processing. Several EPSILON II devices can be embedded in a system to allow multiple and/or mul-

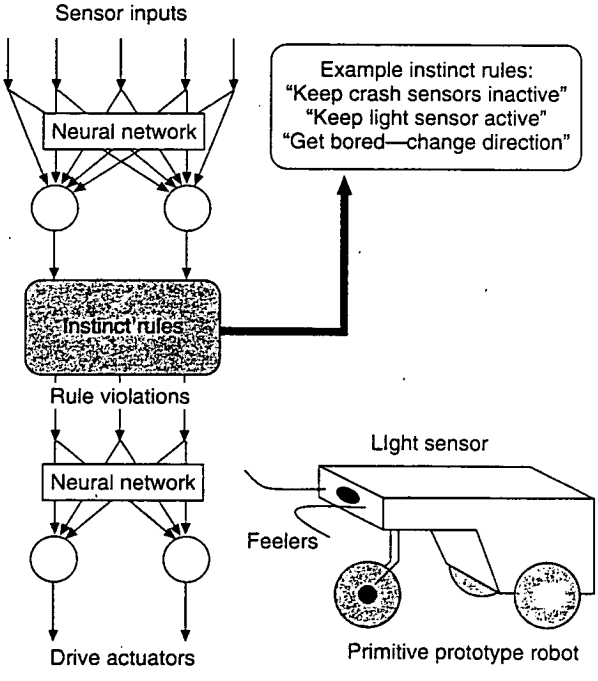


Figure 10. A simple instinct-rule robot.

tilayer networks. The real-time applications environment described makes this an attractive possibility.
- Speed. Table 2 lists guaranteed calculation times in connections per second. The speed of software solutions is not so readily defined. This has implications for real-time applications, where a guaranteed speed performance is essential.

ADVANCES IN CIRCUIT TECHNIQUES, learning algorithms, memory technology, and neural applications have sprung from our pulse stream work. Many interesting problems remain to be solved—memory yield, process variations, and system packaging come to mind as obvious issues. However, we believe we have developed an optimal approach for embedded analog neural systems in a wide variety of contexts. We also consider it essential that the proponents of analog (or equivalent pulsed) methods are realistic about exactly where such chips will be useful. As simple simulation accelerators, they are entirely inappropriate. As components in a cost-effective, real-time, compact embedded system, they are likely to be invaluable.

Acknowledgments

We thank the Engineering and Physical Sciences Research Council, British Aerospace, Thorn-EMI, British Telecom, and the Commonwealth Scholarship Commission for financial support.

References

1. A.F. Murray, D. Del Corso, and L. Tarassenko, "Pulse Stream VLSI Neural Networks—Mixing Analog and Digital Techniques," *IEEE Trans. Neural Networks*, Vol. 2, No. 2,1991, pp. 193-204.
2. A.F. Murray and A.V.W. Smith, "Asynchronous Arithmetic for VLSI Neural Systems," *Electronics Letters*, Vol. 23, No. 12, June 1987, pp. 642-643.
3. D. Hammerstrom, "A Highly Parallel Digital Architecture for Neural Network Emulation," in *VLSI for AI and Neural Networks*, Plenum, New York, 1991, pp. 357-366.
4. T. Clarkson and C.K. Ng, "Multiple Learning Configurations Using 4th Generation pRAM Modules," *Proc. Int'l Conf. Microelectronics for Neural Networks*, UnivEd Technologies, Edinburgh, 1993, pp. 233-240.
5. M. Holler et al., "An Electrically Trainable Artificial Neural Network (ETANN) with 10240 'Floating Gate' Synapses," *Proc. Int'l Joint Conf. Neural Networks*, 1989, pp. 191-196.
6. A.G. Andreou et al., "Current-Mode Subthreshold MOS Circuits for Analog VLSI Neural Systems," *IEEE Trans. Neural Networks*, Vol. 2, No. 2, 1991, pp. 205-213.
7. P.B. Denyer and J. Mavor, "MOST Transconductance Multipliers for Array Applications," *IEE Proc.*, Pt. 1, Vol. 128, No. 3, June 1981, pp. 81-86.
8. A.F. Murray et al., "Pulse-Firing Neural Chips for Hundreds of Neurons," *Neural Information Processing Systems (NIPS) Conf.*, Morgan Kaufmann, San Mateo, Calif., 1990, pp. 785-792.
9. A.F. Murray and P.J. Edwards, "Synaptic Weight Noise During MLP Training: Fault Tolerance and Training Improvements," *IEEE Trans. Neural Networks*, Vol. 4, No. 4, 1993, pp. 722-725.
10. T. Lehmann, "A Hardware-Efficient Cascadable Chip Set for ANNs with On-Chip Back Propagation," *Int'l J. Neural Systems*, Vol. 4, No. 4, Dec. 1993, pp. 351-358.
11. A.F. Murray, "Multi-Layer Perceptron Learning Optimised for On-Chip Implementation—A Noise-Robust System," *Neural Computation*, Vol. 4, No. 3, 1992, pp. 366-381.
12. G. Bogason, "Generation of a Neuron Transfer Function and Its Derivative," *Electronic Letters*, Vol. 29, No. 21, 1993, pp. 1867-1869.
13. M.J. Rose et al., "Amorphous Silicon Analog Memory Devices," *J. Non-Cryst. Sol.*, Vol. 115, 1989, pp. 168-170.
14. A.A. Reeder et al. "Application of Analog Amorphous Silicon Memory Devices to Resistive Synapses for Neural Networks," *Proc. Materials Research Council Spring Meeting*, Vol. 258, 1992, pp. 1081-1086.
15. U. Nehmzow, "Experiments in Competence Acquisition for Autonomous Mobile Robots," PhD thesis, Univ. of Edinburgh, 1992.

Alan F. Murray is a professor of neural electronics at the University of Edinburgh. Previously he worked as a research physicist and as an integrated-circuit design engineer. Neural networks, particularly hardware issues and applications. are his primary research interest. In 1986. he developed the pulse stream method of neural integration, which has since been implemented and extended in close collaboration with Lionel Tarassenko of Oxford University's Department of Engineering Science.

Murray received a BSc in physics and a PhD in solid-state physics from the University of Edinburgh. He is a member of the IEE, the INNS, and a senior member of the IEEE. He has written more than 120 publications, including undergraduate and research textbooks on neural networks.

Stephen Churcher works as a design engineer with Xilinx Development Corporation, with interests in advanced FPGA architectures and their applications. He received a BSc and a PhD in electronics and electrical engineering from the University of Edinburgh. His graduate research centered on the design of analog CMOS VLSI neural networks for computer vision tasks, with emphasis on region classification in natural scenes. He is an author of 17 publications in this field.

Alister Hamilton is a lecturer in the Department of Electrical Engineering at the University of Edinburgh, where he worked on the development of EPSILON under a research fellowship. Earlier he worked on real-time medical image processing at the Medical Research Council Department of Pattern Recognition and Automation and then as a lecturer at Napier University.

Hamilton has a BSc in communication and electronic engineering from Napier University, an MSc in digital techniques from Heriot-Watt University, Edinburgh, and a PhD from Edinburgh University. He is a member of the IEE and has authored 21 publications.

Andrew J. Holmes received a BEng degree in electrical engineering from Edinburgh University. He is currently studying toward the PhD degree in the same department. His research interests include analog VLSI circuits and artificial neural networks.

Geoff B. Jackson received the BE degree in electrical engineering and an ME degree, both from the University of New South Wales, Sydney, Australia. He is currently working toward the PhD degree at the University of Edinburgh under an award from the Commonwealth Scholarship Commission. His research interests center on neural network hardware and its applications.

H. Martin Reekie is a senior lecturer in electrical engineering at the University of Edinburgh. His primary research interest is analog VLSI for neural networks. Earlier he worked on charge-coupled devices and analog VLSI realizations of wave digital filters using gallium arsenide technology.

Reekie received a BSc in mathematics and statistics and a PhD in electronics from the University of Edinburgh. He is a member of the IEE and a senior member of the IEEE. He has authored over 40 publications, including two undergraduate textbooks, and is currently finishing an undergraduate textbook on analog circuits.

Robin J. Woodburn hopes to obtain his PhD from Edinburgh University, where he is researching VLSI circuits for neural networks. He graduated from Aberdeen University with a BSc in psychology, before taking up a career as a manager. He has since worked, much more happily, with computer systems and human-computer interfaces. He holds an MSc in computer systems engineering from Edinburgh University.

Send correspondence to A.F. Murray, Dept. of Electrical Eng., University of Edinburgh, Edinburgh, Scotland, EH9 3JL; or afm@uk.ac.ed.ee.

International Symposium on Circuits and Systems, 1994

Woodburn R, Reekie H M and Murray A F, 1994. "Pulse-stream circuits for on-chip learning in analogue VLSI neural networks". In *Proceedings of the IEEE International Symposium on Circuits and Systems, vol 4, 103 - 106.*

Pulse-stream circuits for on-chip learning in analogue VLSI neural networks

Robin Woodburn
Department of Electrical Engineering
University of Edinburgh
King's Buildings,
Mayfield Road,
Edinburgh EH9 3JL
email rjw@ee.ed.ac.uk

H Martin Reekie & Alan F Murray
Department of Electrical Engineering
University of Edinburgh
King's Buildings,
Mayfield Road,
Edinburgh EH9 3JL
email martin@ee.ed.ac.uk, afm@ee.ed.ac.uk

ABSTRACT

We explain the advantages of the **Virtual-Targets** algorithm for learning in a VLSI Multilayer Perceptron, outline the design issues, and describe fabricated circuits that implement the main algorithmic functions. The test-chip described is a precursor to a larger device incorporating full on-chip learning.

ACKNOWLEDGEMENTS

The authors would like to thank the U.K. Science and Engineering Research Council for financial support.

INTRODUCTION

Long training times on serial computer architectures make on-chip learning in truly parallel implementations an essential step in developing autonomous neural systems for real-time, real-cost applications. Furthermore, in embedded systems, where analogue VLSI finds its strongest justification, the ability to adapt *in situ*, in the absence of a host computer, is almost essential.

Several groups are investigating on-chip learning. These investigations are primarily digital, although there are schemes which are, like our own system, hybrid digital/analogue technologies. The digital systems are flexible, and support a variety of neural algorithms including back-propagation, but network size is restricted.

Analogue neural systems take advantage of the technology's greater compactness to implement larger numbers of neurons (up to several 100), and can be made cascadable to create larger networks, but are inflexible in the range of algorithms supported. Although the EPSILON chip ([1]) produced by our group can implement several algorithms, the chip only performs the forward-pass, and the learning phase proper is carried out on an associated computer. This has been referred to as the "chip-in-the-loop" process.

Clearly, then, the design of an analogue chip in which only one algorithm can be supported requires that the algorithm be selected with care.

CHOICE OF ALGORITHM

Our preference was for an algorithm which would provide a system capable of dealing with real-world applications of neural networks. The most obvious choice is then back-propagation, because this algorithm can accommodate a whole range of pattern-recognition and signal-processing tasks from medical diagnosis to air-combat manoeuvre selection. We also have a joint interest, with the University's Department of Artificial Intelligence, in robotics applications, because robots can be made to carry out learning behaviour which appears intelligent, even though the underlying processes are very simple [2]. The constraints of a small, autonomous robot are precisely those which demand the compactness of analogue VLSI, low power, and direct analogue-sensor interfaces.

There has been considerable progress recently in translating the back-propagation algorithm into hardware (for example [3, 4]), but the problem is a knotty one. Doubts persist as to the likelihood of success in practice [5]. We have preferred to approach this problem using an algorithm known as "Virtual Targets" - one of a family of target-based algorithms - whose potential was initially identified by Murray [6]. This algorithm has two great advantages : firstly, the means of updating weights uses only local information, simplifying the circuitry necessary for a hardware implementation; and secondly, the

weight-update strategies for both hidden- and output-layer neurons are identical (whereas they are different for back-propagation), which means that, once neuron circuits have been designed, they can be replicated for the whole network.

ISSUES IN ON-CHIP LEARNING

Several issues have to be addressed in on-chip learning, among them : precision; accuracy; means of refreshing weights; the minimum value by which weights can be adjusted; providing four-quadrant multiplication; implementing the derivative of the activation function, at the output, also known as the "sigmoid-prime"; and passing error-signals, representing differences between expected and actual output states, between layers in the network.

The test-chip, and consequently this paper, concentrates on three of these issues, namely :

1. compact, four-quadrant pulse-stream multiplication;
2. the minimum value by which a weight can be adjusted; and
3. implementing the "sigmoid-prime".

FOUR-QUADRANT MULTIPLICATION

Our group has developed a transconductance multiplier [1] (see Figure 1) with highly linear characteristics, for multiplying weights by states.

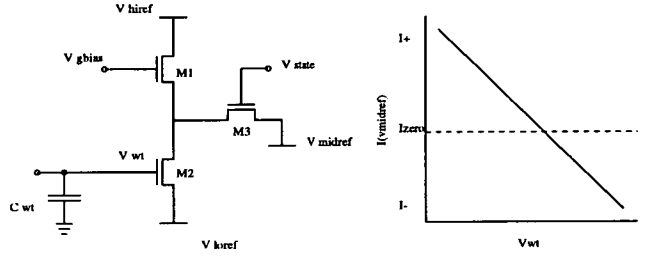


Figure 1: *Transconductance multiplier*

V_{hiref}, V_{midref} and V_{loref} are chosen such that the two transistors $M1$ and $M2$ operate in their linear regions; typically, the difference between V_{hiref} and V_{midref}, and that between V_{midref} and V_{loref} is $0.5V$. The gate-voltage of $M1$ is fixed, to provide a zero-point. When the gate-voltage of $M3$ is held at V_{dd}, so that $M3$ conducts, and the weight, represented by charge on capacitor C_{wt}, is at a low point on the range, $M2$ conducts a small current relative to that through $M1$. The V_{midref} node serves to sink the excess, positive, current; if the weight is at a high point on the range, V_{midref} sources a corresponding current through $M2$. The graph of Figure 1 indicates the relationship.

The current being sourced or sunk by V_{midref} can be controlled in time by pulsing the gate-voltage of $M3$, which represents neural states. Pulse duration represents the neuron's level of activation. If the resulting current-pulse (whose amplitude reflects the weight and whose duration represents the state), is integrated over time, then the multiplication of *weight* $\times$ *state* is computed. Summation of a number of these computations can be achieved simply and elegantly on a single circuit node.

While this circuit, which can be laid out in VLSI in a very compact form, has given excellent performance, it is two-quadrant in nature, because neural states are unipolar and weights bipolar. For the virtual-targets algorithm, a four-quadrant multiplier is required because, in addition to a *weight* $\times$ *state* multiplication, a *weight* $\times$ *error* multiplication is necessary, and weight and error values are both bipolar.

The solution is remarkably simple (see Figure 2).

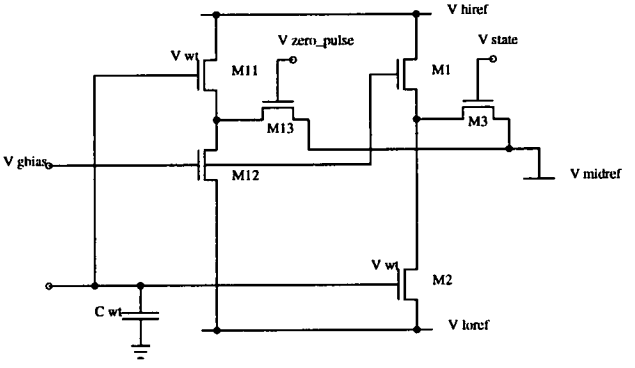


Figure 2: *Four-quadrant transconductance multiplier*

Transistors $M1, M2$ and $M3$ are identical to those in the two-quadrant multiplier. Transistors $M11, M12$ and $M13$ are a second multiplier in parallel with the first, but this time with the V_{gbias} and V_{wt} connections reversed. To give a "zero-state" point, analogous to the zero-current point set by V_{gbias}, the pulse applied to $M13$ is fixed in duration at a mid-point between the shortest pulse-width ($0\mu s$) representing a negative error and the longest pulse width ($20\mu s$) representing a positive error. At the same time a pulse which can vary in duration from $0\mu s$ to $20\mu s$ is applied to $M3$. If the resultant current pulses are integrated in time, just as before, a four-

quadrant, *error* × *weight* multiplication is computed, as shown in Figure 3.

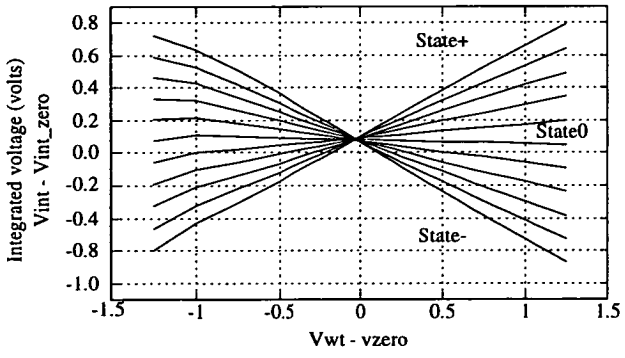


Figure 3: *Four-quadrant characteristics of the transconductance multiplier (simulation results).*

THE SIGMOID-PRIME FUNCTION

Weight-changes evolve, in the virtual-targets algorithm, according to the equations described in [7]. One of the terms in a weight-change equation is the derivative of the sigmoid activation function, also referred to here as the "sigmoid-prime" function, which takes the form $O'_k = O_k(1 - O_k)$.

Circuits already exist which will give both a sigmoid and its derivative[8]. We have been working on an implementation more compatible with our group's approach, where input and output states in the neural network are represented by digital pulses whose duration reflects the level of activation.

The circuit is very again simple. It relies on the fact that the sigmoid-prime does not have to be exact for learning to proceed. Computation is performed by an XOR gate, one input of which receives a pulse of fixed duration (10μs in our scheme) while the other receives a pulse which varies symmetrically around the first, in other words from 0μs to 20μs. The gate's output is integrated by a differential circuit, whereby an **ON** pulse sources charge onto a capacitor, while an **OFF** pulse removes charge, as illustrated in Figure 4.

The ideal sigmoid-prime is shown in Figure 5(a), constrained on the x-axis to values between 0 and 1 by the sigmoid activation function. A triangular approximation to the sigmoid-prime (Figure 5(b)) is obtained by the XOR circuit described above. The integrator-capacitor voltage can be converted back to a pulse-width signal, and this final approximation to the sigmoid-prime is shown in Figure 5(c).

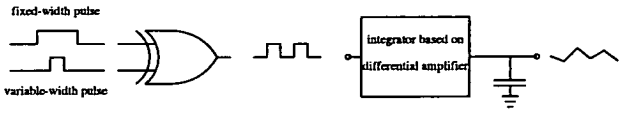


Figure 4: *Pulses from the XOR gate are integrated. When the integration is complete, the output is a triangular function of the duration of the variable pulse.*

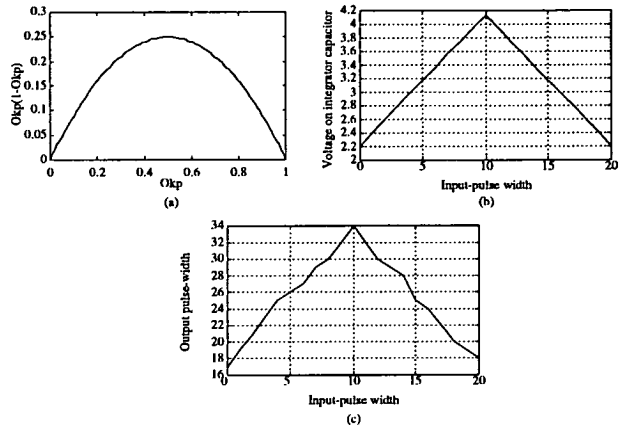


Figure 5: *(a) Ideal form of the sigmoid-prime; (b) Output once the pulses from the XOR gate have been integrated, for input-pulses of different duration; (c) Output once the integrated computations have been converted from a voltage to a pulse-width signal. (Simulation results).*

WEIGHT ADAPTATION

Consider the circuit block-diagram shown in Figure 6, and let us assume that the first integrator has just carried out the sigmoid-prime computation described in Section . The result of the computation is held as a voltage on a capacitor, which can vary in value from V_{low} to V_{high}, with a mid-point of V_{mid}. We can set the voltage range to be such that voltages between V_{mid} and V_{high} are considered to be positive results, while voltages between V_{low} and V_{mid} are considered to be negative results; in other words, V_{mid} is the zero point.

The second integrator in Figure 6 also controls a capacitor on which we may imagine a weight is stored as charge. While the second integrator's input switch is connected to V_{mid}, the charge being accumulated on the weight-capacitor is equal to that being removed, and so the weight remains unchanged. However, if the input to the second integrator is switched to connect to the voltage held on the computation capacitor, then the weight-capacitor's voltage will rise or fall; charge will

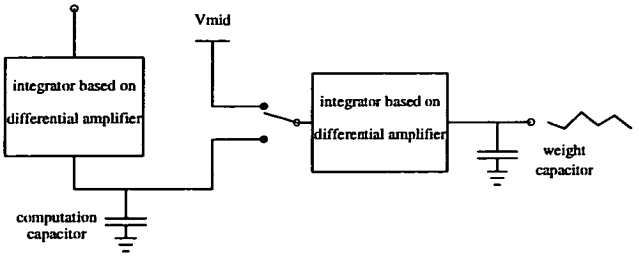


Figure 6: *A circuit for incrementing or decrementing weights.*

accumulate on the weight-capacitor if the computation voltage is above V_{mid}, and charge will be removed from the weight-capacitor if the computation voltage is below V_{mid}.

The size of the weight change depends upon the length of time during which the input switch is connected to the computation capacitor. This has two great advantages : the connection-time can be controlled by pulses, which is highly compatible with our pulse-stream approach; and the connection-time subsequently dictates the "learning rate".

There are several important issues here. The first is **accuracy of the computation** (defined as how closely the result matches our expectations [9]). Clearly, if a weight is to be changed, then it is desirable that the computation on which the change depends is accurate. If the computation produces a weight change that is too large, the weight-change may "overshoot", and instability may result. Conversely, if the hardware-computed change is too small, then the network may take an inordinate time to learn.

The second, more important, issue is one of the **direction** of change. While some level of inaccuracy in computation is tolerable, it is vital that a computation which requires a weight-increase does in fact increase it; should the weight be incorrectly decreased, then the network will almost certainly never train successfully.

The third issue is one of **precision** (defined as the degree of agreement of repeated measurements of a quantity [9]). One way of looking at this problem is to determine the smallest amount by which a weight can be incremented or decremented and the difference be measured. Noise in analogue circuits will also affect precision, and might be quantified as the ratio of the size of the noise-floor to the weight-range.

To date, our studies indicate that the analogue imperfections brought about in a pulse-stream learning system are at least tolerable, in the context of neural training; in fact, analogue noise can be positively beneficial ([10]). These issues can only be resolved by careful analysis of the properties of the circuits fabricated, coupled with equally careful simulation and analysis of their idiosyncrasies as part of a learning network.

REFERENCES

[1] Churcher S, Baxter D J, Hamilton A, Murray A F, and R. H. M, "Generic analog neural computation — the EPSILON chip," in *Advances in Neural Information Processing Systems 5, eds C L Giles, S J Hanson and J D Cowan*, 1993.

[2] Nehmzow U, "Experiments in competence acquisition for autonomous mobile robots, unpublished PhD Thesis, Edinburgh University." 1992.

[3] Valle M, Caviglia D D, and Bisio G M, "An experimental analog VLSI neural chip with on-chip back-propagation," in *Proceedings of the 18th European Solid-state Circuits Conference, pp 203 - 206*, 1992.

[4] Jabri M and Flower B, "Weight perturbation : an optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks," *IEEE Transactions on Neural Networks, 3 (1), 154 - 157*, 1992.

[5] Tarassenko L and Tombs J, "On-chip learning with analogue VLSI neural networks," in *Proceedings of the Third International Conference on Microelectronics for Neural Networks, pp 163 - 174*, 1993.

[6] Murray A F, "Analog noise-enhanced learning in neural networks circuits," *Electronics Letters, 2 (17), 1546 - 1548*, 1991.

[7] M. A. F, ""Analog VLSI and multi-layer perceptrons — accuracy, noise and on-chip learning"," *Neurocomputing, 4 (1992), 301 - 310*, 1992.

[8] Bogason G, "Generation of a neuron transfer function and its derivative," *Electronic Letters (in press, accepted September 1993)*, 1993.

[9] Kirk D B, "Accurate and precise computation using analog VLSI with applications to computer graphics and neural networks, unpublished PhD Thesis, California Institute of Technology." March 1993.

[10] Murray A F and Edwards P J, "Analogue synaptic noise — a hardware nuisance, or an aid to learning ?," in *Proceedings of the Third International Conference on Microelectronics for Neural Networks, pp 121 - 129*, 1993.

IEEE Circuits and Devices, 1996

Woodburn R and Murray A F, 1996. "Pulse-stream techniques and circuits". *IEEE Circuits and Devices, 12, (4), 43 - 47*

Pulse-Stream Techniques and Circuits

Robin Woodburn and Alan F. Murray

The pulse-stream technique for analogue computation combines many of the advantages of the analogue and digital domains. It was originally developed for artificial neural networks (ANNs), where it still finds most of its applications.

Pulse-Stream Encoding

Pulse-stream signals encode analogue information in the time domain by modulating the *width* of a single pulse or the *frequency* of a stream of pulses, as shown in Fig 1. For a pulse-width modulated (PWM) signal, the width of the pulse represents an analogue value with a maximum value determined by the largest pulse-width, and a minimum value determined by the narrowest pulse-width that is detectable. A PWM signal can carry out a computation within the time of the widest pulse (although there are likely to be computational overheads which add to this time). The maximum frequency of the signal therefore depends on the widest pulse.

A pulse-frequency modulated (PFM) signal uses fixed-width pulses that vary in frequency. The largest analogue value is represented by the maximum frequency, and the smallest by the minimum frequency. A PFM signal can take longer to carry out a computation than its PWM counterpart, because the PFM pulses have to be aggregated to establish the frequency of operation. There is, however, no restriction (in principle) on the minimum PFM signal.

For neural applications, PFM signals have the appeal that they are closest in form to the asynchronous spiking of real neurons. Most neural algorithms bear only a superficial resemblance to biological neural networks, so the decision to use PWM or PFM will depend purely upon practical considerations. (Phase-encoding of information, which is common in biological systems, is also possible.)

Computing with an Analogue Two-quadrant Multiplier

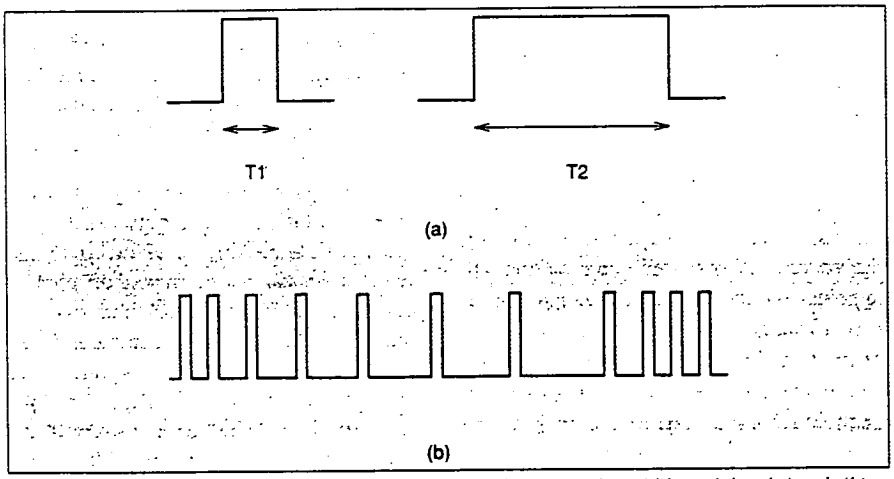
At its simplest, the pulsing circuit for a PWM signal can be reduced to a switch, a current source, and a current sink. An example of the use of such a switch for two-quadrant multipliers is shown in Fig. 2. The constant-current source, $I_{balance}$, is connected in series with the voltage-controlled current-sink, I_{sink}, which can draw currents ranging from zero to a value greater than $I_{balance}$. Current-source and sink are connected through a switch, S, to a voltage source, V_{clamp}.

Consider what happens when switch S is closed. If I_{sink} is smaller than $I_{balance}$, a positive current flows. If I_{sink} exactly matches $I_{balance}$, no current flows. If I_{sink} is larger than $I_{balance}$, a negative current flows. Hence, the amplitude and polarity of the current through the switch is determined by the voltage-control circuitry in the current sink, and the duration of the current is dependent on the time the switch is closed. In short, the circuit produces an output current of a magnitude and sign determined by $I_{balance}$ and I_{sink}, in bursts or pulses of width and frequency determined by the characteristics of the switch.

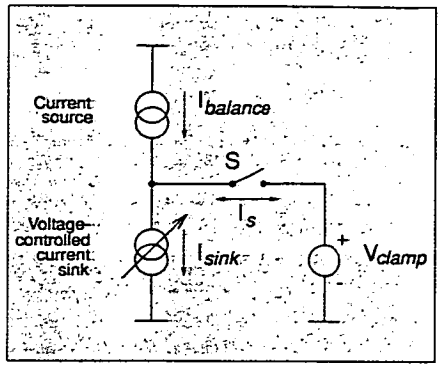
This basic idea was developed for ANN applications to compute the summed product of connection-weights and neural states. The actual circuit, designed on a chip by a former member of our group, Donald Baxter, is shown in Fig 3, along with its transfer characteristics.

Transistors M1 and M2 are connected to power rails having a voltage difference that is low (V_{hi} - V_{lo} is in the order of 1.0V), forcing the transistors to operate in the linear region. M_1 acts as the constant-current source, and M_2 as the current sink. The V_{hi}, V_{lo}, and clamp voltages are all supplied by voltage sources off-chip, preventing them from varying with load; for this reason, we can consider the transistors as current sources, rather than as conductances. The current sink is controlled by $V_{\omega i}$, an analogue value representing a connection-weight. The geometries of M_1 and M_2, along with the voltages V_{zero} and V_{clamp}, are selected to cancel out nonlinear terms in the transistor characteristics, rendering the multiplier admirably linear.

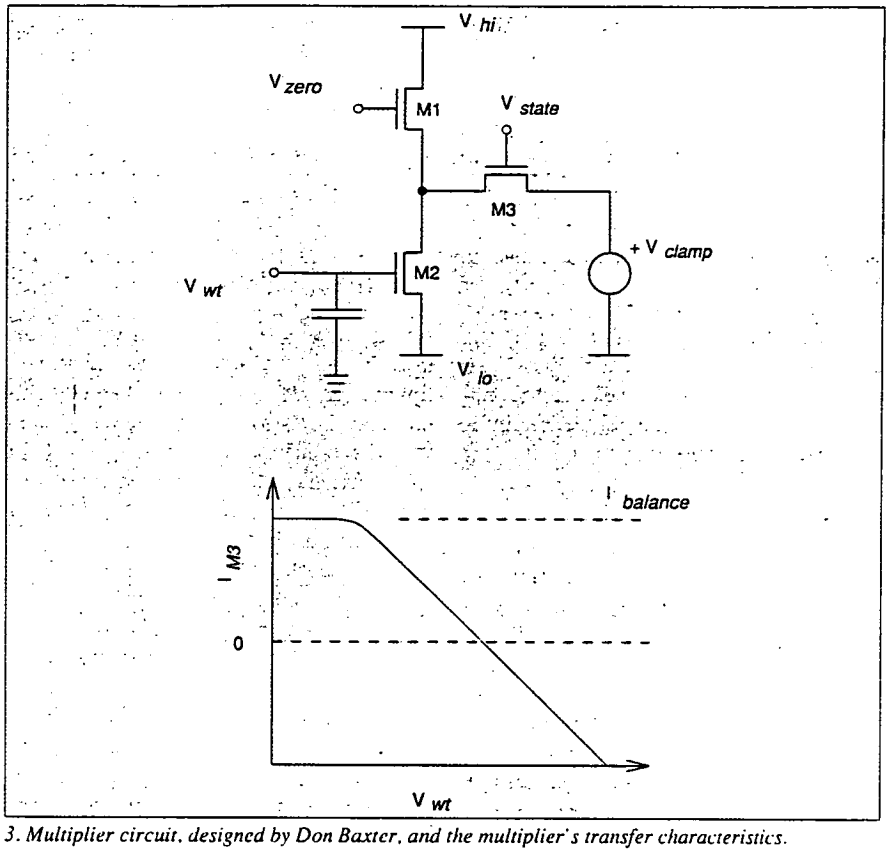
Transistor M_3 acts as a switch whose input is pulsed between 0V and 5V to represent a neural state. The current through M_3 is thus a pulse whose amplitude and polarity depend on $V_{\omega i}$ and whose width depends on V_{state}. In this way, we can multiply an input, the neural state V_{state}, by a stored synaptic weight, $V_{\omega i}$.



1. Encoding analogue information in pulse-stream signals: (a) a pulse-width modulated signal; (b) a pulse-frequency modulated signal.



2. A two-quadrant, pulse-stream multiplier.



3. Multiplier circuit, designed by Don Baxter, and the multiplier's transfer characteristics.

Additional Circuits to Calculate Sum of Products

The multiplier is, in itself, not sufficient to meet our application. Neural networks require the computation of several weight-times-state products, and then the summation of these products, as shown in Fig. 4. In our system, obtaining this product requires two operations. The first is to sum the currents, which is simple, because currents can be summed on a single node, according to Kirchoff's laws.

Because the states are encoded in time, the summed currents must then be integrated in time to give the correct result. To do this is a little more complicated. The current pulses must first be converted into voltage pulses that are, in turn, integrated by a conventional voltage integrator. This aggregated sum-of-products is called the neural activation.

The circuits to perform the integration are shown in Fig. 5. Transistors M_4 and M_5 act as a matched active load onto which the synapse current is delivered. The op-amp maintains the voltage difference between the two op-amp inputs at zero; in other words, it modulates the gate-voltage on M_4 to ensure that the central node is held at V_{clamp}. In this way, the op-amp converts the synapse current pulses into voltage pulses at the op-amp output.

The voltage-pulses are re-converted by the voltage integrator into currents that are aggregated onto the activation capacitor, to give a voltage, V_{activ}, representing the sum-of-products computation. The integrator design is based on a differential stage, which controls an output stage that dumps charge onto an activation capacitor (when the voltage pulses are positive), or removes them (when voltage pulses are negative). For simplicity, details of the integrator are not given here.

Converting Analogue Inputs and Outputs into Pulses

Since analogue values are encoded as binary pulses, many of the advantages of analogue and digital signals can be realised in pulse-stream designs. For example, we can 'read' the voltage on the activation capacitor using a simple comparator, and an inverted, double-sided, analogue ramp provided by a DAC, as shown in Fig 6. As the ramp falls and rises again, the output of the comparator produces a pulse.

The shape of the ramp can also be chosen to implement nonlinear functions. For example, multilayer perceptrons (MLPs) require the sigmoid function shown in Fig. 6. This function can be determined off-chip by generating an appropriate double-sided, nonlinear ramp, so that the comparator output produces a pulse which is a function of the 'sigmoidal ramp.' The same principle can be applied to other functions such as the Gaussian function required by radial-basis-function (RBF) networks.

The multiplier, op-amp, integrator, and comparator circuits are on a single chip, and the inputs to the chip are analogue voltages (which can be stored digitally, off-chip), and pulses. If comparators are placed on chip at the inputs, we can interface directly to analogue voltages. The outputs from the chip will be pulses, which can be transmitted directly to other similar analogue chips or to digital circuits.

Centered Pulses

We have chosen to modulate pulse widths symmetrically about a center line by using double-sided ramps. This centering of pulses has several benefits. It reduces noise introduced by circuit switching and power surges on the chip, because the rising and falling edges of pulses of different widths occur at different times. Also, some computational functions are rendered very simple.

Figure 7 illustrates, as an example, how the value of one pulse might be subtracted from another using an XOR gate. The output is a series of short pulses which, provided the time frame in which they occur is controlled, can be used in another computation. The sign of the subtraction can also be determined easily, using an 'analogue' SR flip-

flop (i.e., one which cannot settle in an indeterminate state). If the longer of the two pulses is applied to input S, Q is set; if the longer pulse is applied to R, Q is reset. Naturally, if the asynchronous nature of the PFM approach is important to a particular application, pulse-centering is not an option.

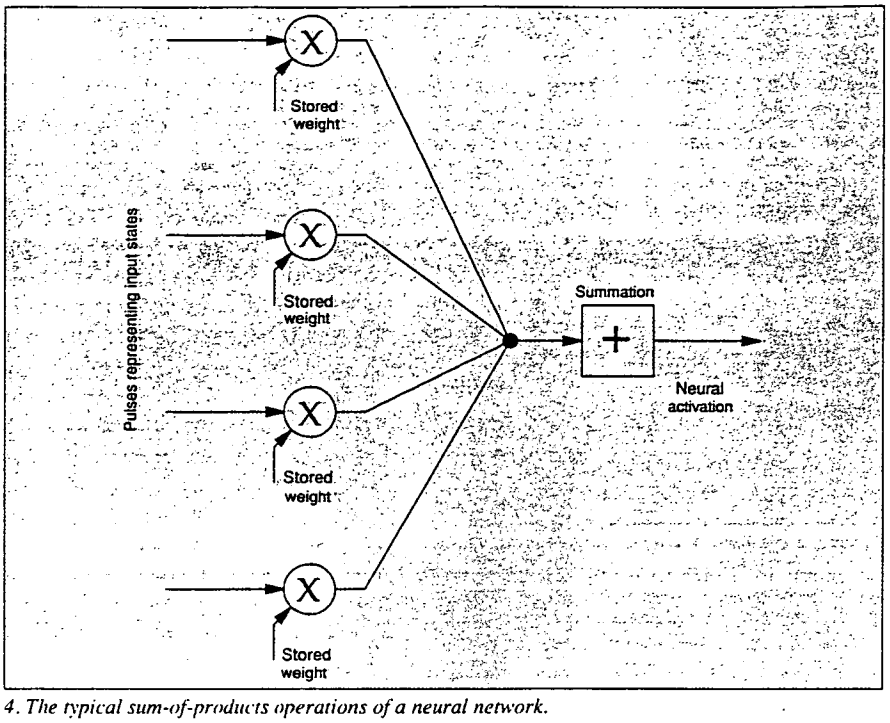
Quantisation

To what extent are time-encoded signals quantised in the systems we have described? With PWM, this depends on our means of providing pulses. For example, the application might require that we represent the state signals in an analogue fashion, and centered precisely, to allow accurate calculation of, say, the difference of two pulses or the sign of a pulse. If state pulses are provided using the comparator and an analogue, double-sided ramp, either for inputs to the chip or outputs from it, then the pulsed signals are truly analogue, and precisely centered.

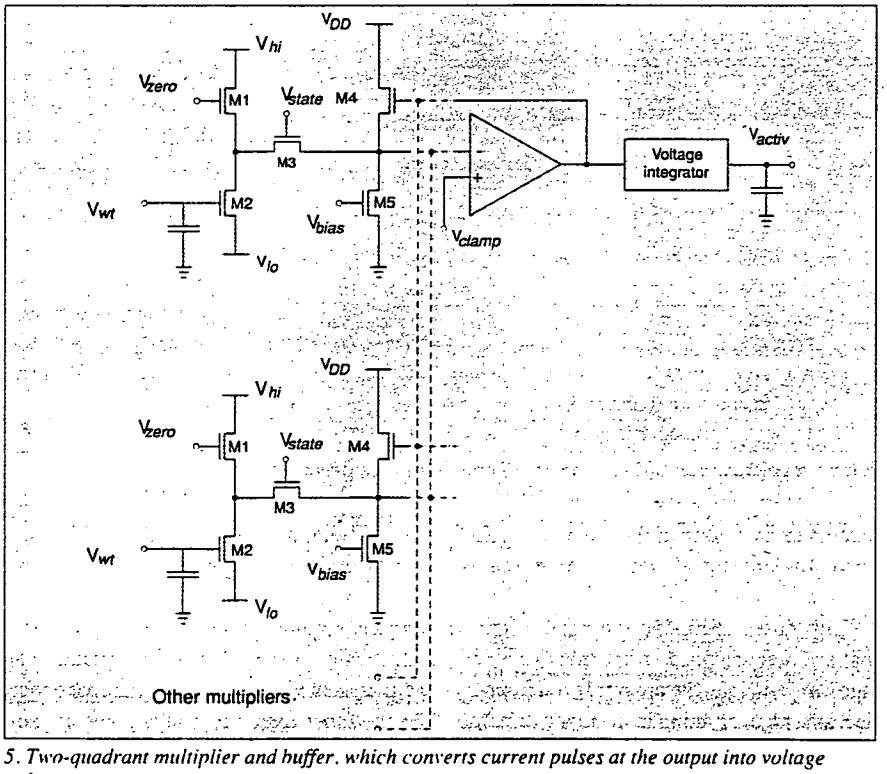
PFM signals are also truly analogue. However, to aggregate the pulses to carry out the computation will probably take considerably longer than the 'per-pulse' computation of PWM.

Where high accuracy is not important, and some level of quantisation is tolerable, then, for simplicity, we might provide the ramp to the comparator using a DAC. The DAC, of course, produces a stepped ramp, not an analogue ramp, and introduces quantisation effects. Also, we can usefully 'store' pulses in RAM, as shown in Fig 8.

To 'fire' the pulses, we use a rapidly-clocked counter to address successive locations, starting with address 0 and ending with the highest address in the RAM. Again, this technique introduces quantisation effects, and there is the disadvantage that pulses made up of an odd number of addresses cannot be exactly centered on a midpoint. The level of quantisation depends on the clock speed; to reduce the quantisation, the clock speed can be increased. There are practical limits to the clock speed, since increasing the clock speed means more RAM is required for a pulse of a given width.



4. The typical sum-of-products operations of a neural network.

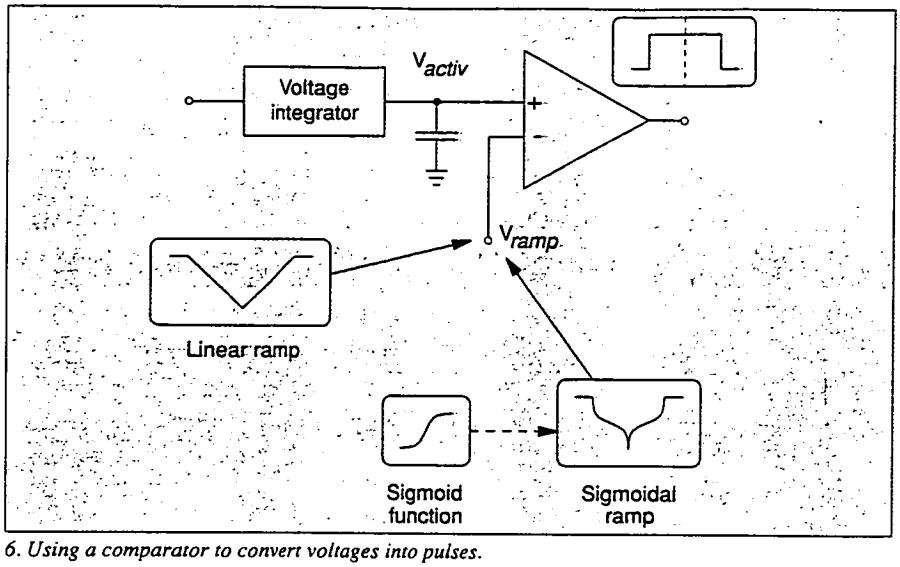


5. Two-quadrant multiplier and buffer, which converts current pulses at the output into voltage pulses.

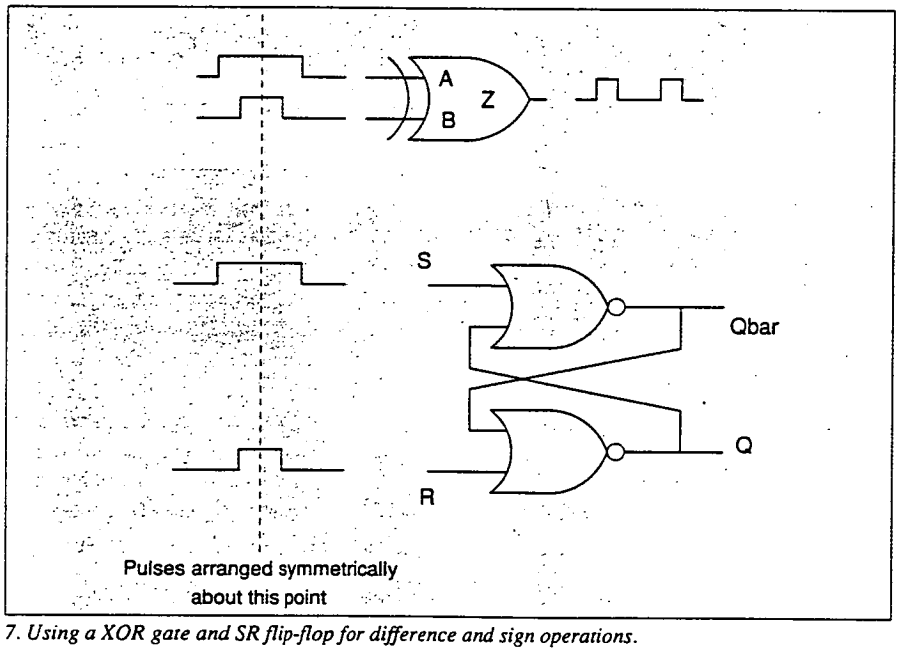
Process Dependence of Signals versus Scaled VLSI Technologies

All the designs presented here were fabricated on VLSI processes that used 0V and 5V supply rails. For example, the current-pulses described previously, which represent a weight times state multiplication, are partly encoded in time (the state variable) and partly amplitude modulated (the weight variable).

As VLSI device-dimensions are reduced, the operating voltage is often decreased (a 3.3V operating voltage is now quite common). This means the range over which the amplitude of any signal may be modulated also decreases. Sometimes it may be possible to redesign the circuit to allow for the scaling. Also, signal-to-noise ratios and dy-



6. Using a comparator to convert voltages into pulses.



7. Using a XOR gate and SR flip-flop for difference and sign operations.

namic ranges worsen as the process is scaled down. In short, signals which are amplitude-modulated scale poorly.

This raises a further question: do signals that are time encoded suffer this disadvantage? In principle, the answer is no. In practice, as with everything in the real world, the answer is not so simple. Establishing the exact timing of events (for example, the rise or fall point of a pulse edge) is difficult because of small variations in rise and fall times, and so pulses are susceptible to jitter. In addition, the circuits used to produce the pulses can themselves be dependent on the scaling factor. However, it would be fair to say that encoding signals in time gives additional freedom in the design of analogue circuits. At one extreme, if the designer is creative enough to encode all signals stochastically (using PFM), the circuits can achieve a measure of scaling independence. At the other extreme, if all signals are amplitude modulated, then the circuits are heavily scaling dependent, as device dimensions and operating voltages are reduced. Many of our research-group's circuits combine signals that are amplitude modulated with signals using time encoding, and so these circuits lie somewhere between the extremes. Changes in technologies certainly present many challenges to the analogue designer, and pulse-stream techniques can help in meeting these challenges.

ANN and Other Applications

Thus far our research group has adapted the design principles described here for MLPs, RBF networks, robotics, and analogue filters. The basic pulse-stream technique can be applied widely, where its combination of pseudo-analogue behaviour, robustness, and simplicity are valuable. The only fundamental constraint is the level of difficulty of the analogue function to be implemented.

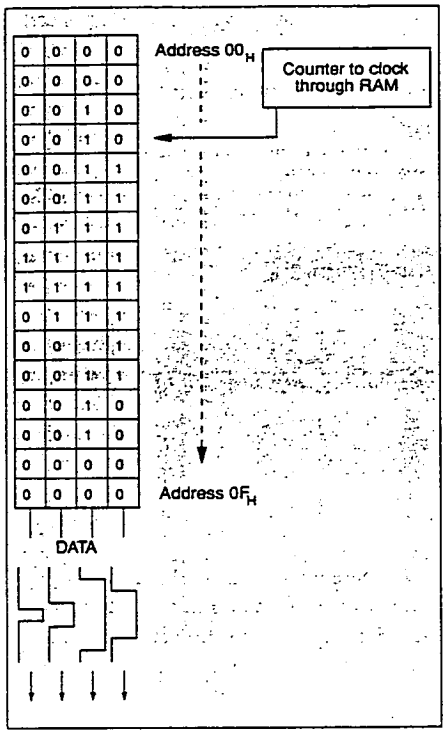
The primary disadvantage of the technique is that effort has to be expended in developing additional analogue and digital circuitry to produce and to store the data encoded in the pulses, and to generate some rather unusual ramps. Even then, we have found that the solutions often reduce to tried-and-tested digital techniques.

We hope the circuits we have described here encourage you to try some designs of your own. Several other pulse-stream efforts have been reported at ISCAS and elsewhere. As an 'exercise for the student,' try selecting the transistor dimensions and voltages V_{hi}, V_{lo}, and V_{zero} in Fig. 3 to render the pulsed current through M_3 a purely linear function of the weight voltage $V_{\omega t}$. Once you've done that, explore the range over which $V_{\omega t}$ can vary before the circuit "falls apart."

Further Reading

The majority of applications to date are neural ones, but these techniques have been applied outside the field. Meador and Hylander [1], for example, have designed a pulse-coded, winner-take-all network. The network measures the distance between an input vector, representing some pattern that exists in the outside world, and stored 'prototype' vectors, to determine which prototype most nearly approximates the input. This kind of network, common in neural applications such as self-organising feature maps, can also be useful for vector quantisation and coding, and for statistical data clustering.

Some approaches are explicitly biological. De Yong and Fields [2] have used their knowledge of biological signals as inspiration for applications in signal processing and control systems. Biological neurons communicate using trains of pulse-like action potentials, fired continuously or in bursts. The pulse-trains encode timing, frequency,



8.'Firing' pulses by clocking through a RAM.

and phase relationships that the authors exploit in their artificial networks, with the aim of storing complex patterns for signal processing problems.

Elias [3] uses fairly simple models of very low level neural structures, namely the dendrite (one of the structures of nerve cells to which synapses connect). His work is interesting on two counts. The first is that his circuits can realise temporal-encoding, a well-known feature of real neurons; he stimulates chains of simple *RC* circuits that emit different responses depending on the physical distance of the stimulus from the output. The second matter of interest is that, by using the spatial characteristics of several dendrites working in parallel, he is able to build useful feature detectors that can respond to, for example, lines moving in particular directions.

The three approaches described so far in this section can all be found in an interesting collection of articles in [4], where you can also find several other ideas ranging from the realisation of Boolean functions [5], to simple interfacing of networks to the analogue world using pulse-density modulation [6].

Finally, we mention an application from our own group. Papathanasiou and Hamilton [7] have moved away altogether from neural structures, using pulse coding for a filter building-block they call the Palmo filter. Fundamental to filter structures is the integrator, which they realise using pulse-stream techniques. Somewhat like the synapse described in Fig. 2, their design uses PWM to represent the magnitude of signals. The pulses control the ON time of transistors to gate positive and negative currents, which are accumulated on a capacitor. The researchers have found ways of reducing the effects of variations in process, and intend to design a 'programmable' filter chip with an array of filter taps that can implement a range of different filters.

Various other aspects of our research group's activities, including some of those described here, are illustrated in [1-10].

Acknowledgements

The authors would like to thank Tor Sverre Lande for encouraging us to write sections 6 and 7, and also Edgar Sanchez-Sinencio, and Andreas Andreou for their very helpful comments on the drafts of this article. Donald Baxter, Steven Churcher, and Alister Hamilton developed many of the orignal ideas that have made pulse-stream techniques successful. ***CD***

Robin Woodburn is a researcher in the University of Edinburgh's Electrical Engineering Department (e-mail: rjw@ee.ed.ac.uk). *Alan Murray* is Professor of Neural Electronics in the same Department (e-mail: afm@ee.ed.ac.uk).

References

1. J. L. Meador and P. D. Hylander, "Pulse coded winner-take-all networks," in *Silicon Implementation of Pulse Coded Neural Networks*, M. E. Zaghloul, J. L. Meador, and R. W. Newcomb, Eds., pp. 79-99, Kluwer Academic Publishers, Boston, Ma, 1994.

2. M. DeYong and C. Fields, "Silicon neurons for phase and frequency detection and pattern generation," in *Silicon Implementation of Pulse Coded Neural Networks*, M. E. Zaghloul, J. L. Meador, and R. W. Newcomb R~W, Eds., pp. 65-77. Kluwer Academic Publishers, Boston, Ma, 1994.

3. J. G. Elias, "Silicon dendritic trees," in *Silicon Implementation of Pulse Coded Neural Networks*, M. E. Zaghloul, J. L. Meador, and R. W. Newcomb, Eds., pp. 39-63. Kluwer Academic Publishers, Boston, Ma, 1994.

4. J. L. Meador and P. D. Hylander, *Silicon Implementation of Pulse Coded Neural Networks*, Kluwer Academic Publishers, 1994.

5. M. deSavigny and R. W. Newcomb, "Realization of Boolean functions using a pulse coded neuron," in *Silicon Implementation of Pulse Coded Neural Networks*, M. E. Zaghloul, J. L. Meador, and R. W. Newcomb, Eds., pp. 65-77. Kluwer Academic Publishers, Boston, Ma, 1994.

6. J. Tomberg J, "Synchronous pulse density modulation in neural network implementation," in *Silicon Implementation of Pulse Coded Neural Networks*, M. E. Zaghloul, J. L. Meador, and R. W. Newcomb, Eds., pp. 65-77, Kluwer Academic Publishers, Boston, Ma, 1994.

7. K. Papathanasiou and A. Hamilton, "Pulse based signal processing: VLSI implementation of a Palmo filter," in *Proceedings of the International Symposium on Circuits and Systems*, Atlanta, May 1996, in press.

8. A. Hamilton, A. F. Murray, D. J. Baxter, et. al., "Integrated pulse stream neural networks: results, issues and pointers," *IEEE Transactions on Neural Networks*, 3(3), 385-393, 1992.

9. G. Jackson and A. F. Murray, "Competence acquisition in an autonomous mobile robot using hardware neural techniques," in *Advances in Neural Information Processing Systems*, 8, in press. The MIT Press, Cambridge, MA, 1996.

10. R. Woodburn, H. M. Reekie, and A. F. Murray, "Pulse-stream circuits for on-chip learning in analogue VLSI neural networks," in Proceedings of the IEEE International Symposium on Circuits and Systems 4, London, 103-106, 1994.

Bibliography

Abusland A and Lande T S (1994). "An analog continuous-time micropower hopfield net". In *Proceedings of the IEEE International Conference on Neural Networks, III, 1860 - 1865.*

Alspector J (1989). "Neural-style microsystems that learn". *IEEE Communications Magazine, November, 29 - 36.*

Alspector J, Bhusan G, and Allen R B (1989). "Performance of a stochastic learning microchip". In Touretzky D S, editor, *Advances in Neural Information Processing Systems 1 , 749 - 760.* Morgan Kaufmann, San Mateo, Ca.

Alspector J, Jayakumar A, and Luna S (1992). "Experimental evaluation of learning in a neural microsystem". In Moody J E, Hanson S J, and Lippman R P, editors, *Advances in Neural Information Processing Systems 4, 871 - 878.* Morgan Kaufmann, San Mateo, Ca.

Annema A J and Wallinga H (1995). "Analog weight adaptation hardware". *Neural Processing Letters, 2, (3), 7 - 11.*

Anzai Y (1992). *Pattern Recognition and Machine Learning.* Academic Press, San Diego, Ca.

Arima Y, Mashiko K, Okada K, Yamada T, Maeda A, H, K., and S, K. (1991a). "A self-learning neural network chip with 125 neurons and 10K self-organisation synapses". *IEEE Journal of Solid-state Circuits, 26 (4), (April), 607 - 611.*

Arima Y, Mashiko K, Okada K, Yamada T, Maeda A, Notani H, Kondoh H, and Kayano S (1991b). "A 336 neuron, 28K-synapse, self-

learning neural network chip with branch-neuron-unit architecture". *IEEE Journal of Solid-state Circuits, 26 (11), (November), 1637 - 1643.*

Arima Y, Murasaki M, Yamada T, Maeda A, and Shinohara H (1992). "A refreshable analog VLSI neural networkchip with 4000 neurons and 40k synapses". In *Digest of Technical papers from the IEEE International Solid-State Circuits Conference, San Francisco, 132 - 133 and 265.*

Beale R and Jackson T (1990). *Neural computing : an introduction.* Adam Hilger, Bristol.

Beer R D (1990). *Intelligence as adaptive behavior : an experiment in computational neuroethology.* Academic Press, Boston.

Berg Y, Sigvartsen R L, Lande T S, and Abusland A (1996). "An analog feed-forward neural network with on-chip learning ". *Analog Integrated Circuits and Signal Processing, 9, 65 - 75.*

Bezdek J C (1992). "On the relationship between neural networks, pattern recognition and intelligence". *International Journal of Approximate Reasoning, 6, 85 - 107.*

Botros N M and Abdul-Aziz M (1993). "Hardware implementation of an artificial neural network". In *Proceedings of the IEEE International Conference on Neural Networks, San Francisco, 1252 - 1257.*

Boulton M E (1994). "Conditioning, remembering, and forgetting. *Journal of Experimental Psychology : Animal Behavior Processes, 20, (3), 219 - 231.*

Brooks R A (1989). "A robot that walks; emergent behaviors form a carefully evolved network". *MIT Artificial Intelligence Laboratory AI Memo 1091.*

Brooks R A (1991). "Intelligence without representation". *Artificial Intelligence, 47, 139 - 159.*

Brown T, Chapman P, Kairiss E, and Keenan C (1988). "Long-term synaptic potentiation". *Science, 242, 724 - 728.*

Brugler J and Jespers P (1969). "Charge pumping in MOS devices". *IEEE Transactions on Electron Devices, 16, (3), 297 - 302.*

Byrne J H (1988). "Cellular analysis of associative learning". *Physiological Reviews, 67, (2), 329 -439.*

Cairns G (1995). *Learning with analogue VLSI multi-layer perceptrons.* Unpublished PhD dissertation, Department of Engineering Science, Oxford University.

Card H C and Schneider C R (1992). "Analog CMOS neural circuits - *in situ* learning". *International Journal of Neural Systems, 3 (2), 103 - 124.*

Castello R, Caviglia D D, Franciotta M, and Montecchi F (1991). "Self-refreshing analogue memory cell for variable synaptic weights". *Electronics Letters, 27, (20) September 1991, 1871 - 1872.*

Caudill M and Butler C (1990). *Naturally intelligent systems.* The MIT Press, Cambridge, Massachussets.

Choi M and Salam F (1993). "Implementation of feedforward artificial neural nets with learning using standard CMOS technology". In *Proceedings of IEEE International Symposium on Circuits and Systems, Singapore, 1509 - 1512.*

Churcher S (1993). *VLSI neural networks for computer vision.* Unpublished PhD dissertation, Department of Electrical Engineering, Edinburgh University.

Churcher S, Baxter D J, Hamilton A, Murray A F, and Reekie H M (1993). "Generic analog neural computation — the EPSILON chip". In *Advances in Neural Information Processing Systems 5, eds C L Giles, S J Hanson and J D Cowan.*

Churchland P M (1989). *A neurocomputational perspective : the nature of mind and the structure of science.* The MIT Press, Cambridge, Massachussets.

Churchland P S and Sejnowski T J (1988). "Perspectives on cognitive neuroscience". *Science, 242, 741 - 745.*

Churchland P S and Sejnowski T J (1992). *The Computational Brain.* MIT Press, Cambridge, Ma.

Cohen M and Andreou A (1992). "MOS circuit for nonlinear Hebbian learning". *Electronics Letters, 28, (6), 591 - 593.*

Collins H M (1990). *Artificial experts : social knowledge and intelligent machines.* The MIT Press, Cambridge, Massachussets.

Connell J H (1990). *Minimalist mobile robotics : a colony-style architecture for an artificial creature.* Academic Press, Boston.

Descartes R (1972). *Treatise of man.* Harvard University Press, Cambridge, Massachussets.

Dolenko B K and Card H C (1993a). "Neural learning in analogue hardware : effects of component variation from fabrication and from noise". *Electronics Letters, 29, (8), 693 - 694.*

Dolenko B K and Card H C (1993b). "The effects of analog hardware properties on backpropagation networks with on-chip learning". In *Proceedings of the IEEE International Conference on Neural Networks, San Francisco, vol 3, 110 - 115.*

Dolenko B K and Card H C (1995). "Tolerance to analog hardware of on-chip learning in backpropagation networks". *IEEE Transactions on Neural Networks, 4, (5), 1045 - 1052.*

Donald J and Akers L (1993). "A neural processing node with on-chip learning". In *Proceedings of IEEE International Symposium on Circuits and Systems, Chicago, vol 4, 2748 - 2751.*

Dreyfus H L (1992). *What computers still can't do : a critique of artificial reason.* The MIT Press, Cambridge, Massachussets.

Dreyfus H L and Dreyfus S E (1988). "Making a mind versus modeling a brain : artificial intelligence at a branch point". *Artificial Intelligence, 117.*

Dupuie S T and Ismail M (1990). "High frequency CMOS transconductors". In Toumazou C, Lidgey F J, and Haigh D G, editors, *Analogue IC Design : the Current-mode Approach*, IEE Circuits and Systems, Series 2, chapter 5, pages 181 - 238. Peter Peregrinus Ltd, London.

Duranton M and Sirat J (1990). "Learning on VLSI : a general-purpose digital neurochip". *Philips Journal of Research, 45, (1), 1 - 17.*

Eberhardt S, Tawel R, Brown T, Daud T, and Thakoor A (1992). "Analog VLSI neural networks : implementation issues and examples in optimization and supervised learning". *IEEE Transactions in Industrial Electronics, 39, (6), 552 - 564.*

Edelman G (1994). *Bright air, brilliant fire : on the matter of the mind.* Penguin, London.

Eguchi H, Furuta T, Horiguchi H, Oteki S, and Kitaguchi T (1991). "Neural network LSI chip with on-chip learning". In *Proceedings of the International Joint Conference on Neural Networks, Seattle, vol 1, I-453 - I-456.*

El-Masry B, Maundy B J, and Abu-Allam E (1992). "Weight storage elements for analog implementation of artificial neural networks". In *Proceedings of the 35th Midwest Symposium on Circuits and Systems, 2, Chapter 395, 1520 - 1523.*

Elias J G (1993). "Artificial dendritic trees ". *Neural Computation, 5, 648 - 664.*

Faggin F and Mead C (1990). "VLSI implementation of neural networks". In Zornetzer S F, Davis J L, and Lau, editors, *An introduction to neural and electronic networks*, chapter 13, pages 275 - 292. Academic Press, San Diego, Ca.

Fahlman S E (1988). "An empirical study of learning speed in back-propagation networks". *Technical Report CMU-CS-88-162, Carnegie Mellon University.*

Feldman J A and Ballard D H (1982). "Connectionist models and their properties". *Cognitive Science, 6, 205 - 254.*

Fodor J A and Pylyshyn Z W (1988). "Connectionism and cognitive architecture : a critical analysis". *Cognition, 28, 3 - 71.*

Frye R C, Reitman E A, and Wong C C (1991). "Back-propagation learning and nonidealities in analog neural network hardware". *IEEE Transactions on Neural Networks, 2, (1), 110 - 117.*

Ghosh J, LaCour P, and Jackson S (1994a). "OTA based neural network architectures with on-chip tuning of synapses". In *Proceedings of IEEE 7th International Conference on VLSI design, 71 - 76.*

Ghosh J, LaCour P, and Jackson S (1994b). "OTA based neural network architectures with on-chip tuning of synapses". *IEEE Transactions on Circuits and Systems - II : Analog and Digital Signal Processing, 41, (1), 49 - 58.*

Grossman T, Meir R, and Domany E (1989). "Learning by choice of internal representations". In Touretzky D S, editor, *Advances in Neural Information Processing Systems 1 , 73 - 80.* Morgan Kaufmann, San Mateo, Ca.

Groves P and Rebec G (1992). *An introduction to biological psychology.* Wm C Brown, Dubuque, Ia, USA.

Hamilton A, Churcher S, Edwards P J, Jackson G B, Murray A F, and Reekie H M (1993). "Pulse stream VLSI circuits and systems : the Epsilon neural network chipset". *International Journal of Neural Systems, 4, (4), 395 - 405.*

Hamilton A, Murray A F, Baxter D J, Churcher S, Reekie H M, and Tarassenko L (1992). "Integrated pulse stream neural networks : results, issues and pointers". *IEEE Transactions on Neural Networks, 3, (3), 385 - 393.*

Hammerstrom D (1990). "A VLSI architecture for high-performance, low-cost, on-chip learning". In *Proceedings of the International Joint Conference on Neural Networks (vol II), San Diego, 537 - 544.*

Han J and Moraga C (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In Sandoval J, editor, *Lec-*

ture Notes in Computer Science 930 : from Natural to Artificial Computation. Springer-Verlag, Berlin.

Harvey W (1628). *Movement of the heart and blood in animals.* Everyman's Library (1963).

Hebb D O (1949). *The organization of behavior : a neuropsychological theory.* John Wiley and Sons, New York.

Hertz J, Krogh A, and Palmer R G (1991). *Introduction to the theory of neural computing.* Addison-Wesley, Redwood City.

Hollis P W and Paulos J J (1994). "A neural network learning algorithm tailored for VLSI implementation". *IEEE Transactions on Neural Networks, 5, (5), 784 - 791.*

Holmes A J, Gibson R, Hajto J, Murray A F, Owen A E, Rose M J, and Snell A J (1993). "Use of a-Si:H memory devices for non-volatile weight storage in artificial neural networks". *Journal of Non-crystalline Solids, 166, 817 - 820.*

Holmes A J, Murray A F, Churcher S, and Hajto J (1995). "Pulsestream synapses with non-volatile analogue amorphous silicon memories". In Tesauro G, Touretzky D, and Leen T, editors, *Advances in Neural Information Processing Systems 7, 763 - 769.* The MIT Press, Cambridge, Ma, Cambridge, Ma.

Ibrahim F and Zaghloul M (1990). "Design of modifiable-weight synapse CMOS analog cell". In *Proceedings of the International Symposium on Circuits and Systems, New Orleans, vol 4, 2978 - 2981.*

Jabri M and Flower B (1992). "Weight perturbation : an optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks". *IEEE Transactions on Neural Networks, 3 (1), 154 - 157.*

Jabri M, Pickard S, Leong P, and Xie Y (1993). "Algorithmic and implementation issues in analog low power learning neural network chips". *Journal of VLSI Signal Processing, 6, 67 - 76.*

Kim S, Shin Y, Bogineni N, and Sridhar R (1992). "A programmable analog CMOS synapse for neural networks". *Analog Integrated Circuits and Signal Processing, 2, 345 - 352.*

Kirk D B (1993). *Accurate and precise computation using analog VLSI with applications to computer graphics and neural networks.* Unpublished PhD dissertation, California Institute of Technology, March.

Klopf A H (1988). "A neuronal model of classical conditioning". *Psychobiology, 16, 85 - 125.*

Kosko B (1988). "Bidirectional associative memories". *IEEE Transactions on Systems, Man and Cybernetics, 18, 49 - 60.*

Krogh A, Thorbergsson G, and Hertz J (1990). "A cost function for internal representations". In Touretzky D S, editor, *Advances in Neural Information Processing Systems 2, 733 - 740.* Morgan Kaufmann, San Mateo, Ca.

Kub F J, Moon K K, Mack I A, and Long F M (1990). "Programmable analog vector-matrix multipliers". *IEEE Journal of Solid-State Circuits, vol SC-25 (1), 207 - 214.*

Lazzaro J and Mead C (1990). "A silicon model of auditory localization". In Zornetzer S F, Davis J L, and Lau, editors, *An introduction to neural and electronic networks*, chapter 8, pages 155 - 173. Academic Press, San Diego, Ca.

Lazzaro J and Wawrzynek J (1993). "Silicon auditory processors as computer peripherals". In Hanson S J, Cowan J D, and Giles C L, editors, *Advances in Neural Information Processing Systems 5, 820 - 827.* Morgan Kaufmann, San Mateo, Ca.

Lehmann T (1993). "A hardware efficient cascadable chip set for ANN with on-chip backpropagation". *International Journal of Neural Systems, 4 (4), 351 -358.*

Lehmann T (1994). *Hardware learning in analogue VLSI neural networks.* Unpublished PhD dissertation, Technical University of Denmark, September.

Lehmann T (1995). "Implementation issues of self-learning pulsed integrated neural systems". In *Proceedings of 13th NORCHIP Conference, Copenhagen.* In press.

Levine D S (1991). *Introduction to neural and cognitive modeling.* Lawrence Erlbaum, New Jersey.

Linares-Barranco B, Sanchez-Sinencio E, Rodriguez-Vazquez A, and Huertas J (1993). "A CMOS analog adaptive BAM with on-chip learning and weight refreshing". *IEEE Transactions on Neural Networks, 4, (3), 445 - 455.*

Lindblad T, Lindsey C S, Minerskjöld M, Sekhniaidze G, Székely G, and Eide Å(1995). "Implementing the new zero instruction set computer (ZISC036) from IBM for a Higgs search". *Nuclear Instruments and Methods in Physics Research A, 357, 192 - 194.*

Liu W, Andreou A G, and Goldstein M H (1992). "Voiced-speech representation by an analog silicon model of the auditory periphery". *IEEE Transactions on Neural Networks, 3, (3), 477 - 487.*

Looney C G (1996). "Stabilization and speedup of convergence in training feedforward neural networks". *Neurocomputing, 10, (1), 7 - 31.*

Lyon R F and Mead C (1989). "Electronic cochlea". In Mead C, editor, *Analog VLSI and Neural Systems, 279 - 302.* Addison-Wesley, Reading, Ma.

Macq D, Legat J, and Jespers P (1992). "Analog storage of adjustable synaptic weights". In *Applications of Neural Networks III — Proceedings of the International Society for Optical Engineers, Orlando, vol 1709, 712 - 718.*

Maren A, Harsten C, and Pap R (1990). *Handbook of Neural Computing Applications.* Academic Press, San Diego, Ca.

McCorduck P (1979). *Machines who think : a personal inquiry into the history and prospects of artificial intelligence.* W H Freeman, San Francisco.

McDonald, J. A. (1992). "Neural nets are starting to make sense". *Biosensors and Biolelectronics, 7, (9), 621 - 626.*

Mead C (1989). *Analog VLSI and neural systems.* Addison-Wesley.

Meador J, Wu C, Nintunze N, and Chintrakulchai P (1991). "Programmable impulse neural circuits". *IEEE Transactions on Neural Networks, 2, (1), 101 - 108.*

Mitchison G (1989). "Learning algorithms and networks of neurons". In Durbin R, Miall C, and Mitchison G, editors, *The Computing Neuron,* chapter 3, pages 35 - 53. Addison Wesley.

Montalvo A J, Gyurcsik R S, and Paulos J J (1994a). "Building blocks for a temperature-compensated analog VLSI neural network with on-chip learning". In *Proceedings of the International Symposium on Circuits and Systems, vol 6, 363 - 366.*

Montalvo A J, Hollis P, and Paulos J (1992). "On-chip learning in the analog domain with limited precision circuits". In *Proceedings of the International Joint Conference on Neural Networks, vol 1, I-196 - I-201.*

Montalvo A J, Paulos J J, and Gyurcsik R S (1994b). "An analog VLSI neural network architecture with on-chip learning". In *Proceedings of the IEEE International Conference on Neural Networks, Orlando, 1364 - 1368.*

Mundie D B and Massengill L W (1991). "Weight decay and resolution effects in feedforward artificial neural networks". *IEEE Transactions on Neural Networks, 2, (1), 168 - 170.*

Murray A F (1991). "Analog noise-enhanced learning in neural networks circuits". *Electronics Letters, 2 (17), 1546 - 1548.*

Murray A F (1992a). "Analog VLSI and multi-layer perceptrons — accuracy, noise and on-chip learning". *Neurocomputing, 4 (1992), 301 - 310.*

Murray A F (1992b). Multi-layer perceptron learning optimised for on-chip implementation. *Neural Computation, 4 (3), 336 - 381.*

Murray A F, Del Corso D, and Tarassenko L (1991). "Pulse-stream VLSI neural networks mixing analog and digital techniques". *IEEE Transactions on Neural Networks, 2, (2), 193 - 204.*

Murray A F and Edwards P J (1994). "Synaptic weight noise during MLP training : Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training". *IEEE Transactions on Neural Networks, 5, (5), 792 - 802.*

Myers D J, Cox V, Harbridge J, Orrey D, Williamson C, and Naylor D (1992). "A high performance digital processor for implementing large artificial neural networks". *BT Technological Journal, 10, (3), 134 - 143.*

Penrose R (1995). *Shadows of the Mind.* Vintage, London.

Polanyi M (1958). *Personal knowledge : towards a post-critical philosophy.* Routledge and Kegan Paul.

Rees C (1996). *Neural computing learning solutions : user survey.* Department of Trade and Industry, London.

Rohwer R (1990). "The 'moving targets' training algorithm". In Touretzky D S, editor, *Advances in Neural Information Processing Systems 2, 558 - 565.* Morgan Kaufmann, San Mateo, Ca.

Rosen D J, Rumelhart D E, and Knudsen E I (1994). "A connectionist model of the owl's sound localization system". In Cowan J D, Tesauro G, and Alspector J, editors, *Advances in Neural Information Processing Systems 6, 606 - 613.* Morgan Kaufmann, San Mateo, Ca.

Rumelhart D E (1990). "Brain style computation : learning and generalization". In Zornetzer S F, Davis J L, and Lau, editors, *An introduction to neural and electronic networks*, chapter 21, pages 405 - 420. Academic Press, San Diego, Ca.

Rumelhart D E, Hinton G E, and Williams R J (1986). "Learning internal representations by error propagation". In Rumelhart D E, McClelland J L, et al., editors, *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, volume 1, chapter 4, pages 318 - 362. MIT Press, Cambridge, Ma.

Rumelhart D E and McClelland J L (1986). "PDP models and general issues in cognitive science". In Rumelhart D E, McClelland J L, et al., ed-

itors, *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, volume 2, chapter 4, pages 110 - 149. MIT Press, Cambridge, Ma.

Sackinger E, Boser B E, Bromley J, LeCun Y, and Jackel L (1992). "Application of the ANNA neural network chip to high-speed character recognition". *IEEE Transactions on Neural Networks, 3, (3), 498 - 505.*

Salam F and Choi M (1990). "An all-MOS analog feedforward neural circuit with learning". In *Proceedings of the International Symposium on Circuits and Systems, vol 4, 2508 - 2511.*

Salam F and Wang Y (1991). "A real-time experiment using a 50-neuron CMOS analog silicon chip with on-chip digital learning". *IEEE Transactions on Neural Networks, 2, (4), 461 - 464.*

Sarkar D (1995). "Methods to speed up error back-propagation learning algorithm". *ACM Computing Surveys, 27, (4), 519 - 542.*

Schneider C and Card H (1990). "Analog VLSI models of mean field networks". In Delgadofrias J G and Moore W R, editors, *VLSI, artificial intelligence and neural networks*, chapter 39, pages 185 - 194. Plenum Publishing, New York.

Schneider C and Card H (1991a). "Analog CMOS Hebbian synapses". *Electronics Letters, 27, 9, 785 - 786.*

Schneider C and Card H (1991b). "Analog CMOS synaptic learning circuits adapted from invertebrate biology". *IEEE Transactions on Circuits and Systems, vol CAS-38 (12), 1430 - 1438.*

Schneider C and Card H (1991c). "CMOS mean field learning". *Electronic Letters, 27, (19), 1702 - 1704.*

Schneider C and Card H (1992). "Analog CMOS constrastive Hebbian networks". In *Applications of Neural Networks III — Proceedings of the International Society for Optical Engineers, Orlando, vol 1709, 726 - 735.*

Schneider C and Card H (1993). "Analog CMOS deterministic Boltzmann circuits". *IEEE Journal of Solid-State Circuits, 28, (8), 907 - 914.*

Schwartz D, Howard R, and Hubbard (1989a). "Adaptive neural networks using MOS charge storage". In Touretzky D S, editor, *Advances in Neural Information Processing Systems1 , 761 - 768.* Morgan Kaufmann, San Mateo, Ca.

Schwartz D, Howard R, and Hubbard W (1989b). "A programmable analog neural network chip". *IEEE Journal of Solid-State Circuits, 24, (2) 313 - 319.*

Schwartz D and Samalam V (1990). "Learning, function approximation and analog VLSI". In *Proceedings of the International Symposium on Circuits and Systems, 2441 - 2445.*

Schwartz D and Samalam V (1991). "An analog VLSI splining network". In Lippman R P and Moody J, editors, *Advances in Neural Information Processing Systems 3, 1008 - 1014.* Morgan Kaufmann, San Mateo, Ca.

Searle J (1980). "Minds, brains, and programs". *The Behavioral and Brain Sciences, 3, 417 - 424.*

Searle J (1987). "Minds and brains without programs". In Blakemore C and Greenfield S, editors, *Mindwaves.* Basil Blackwell, Oxford.

Shibata T, Kosaka H, Ishii H, and Ohmi T (1995). "A neuron-MOS neural network using self-learning-compatible synapse circuits". *IEEE Journal of Solid-State Circuits, 30, (8), 913 - 922.*

Shibata T and Ohmi T (1992). "A self-learning neural-network LSI using neuron MOSFETs". In *Digest of Technical Papers from the Symposium on VLSI Technology, vol 9, 84 - 85.*

Shima T, Kimura T, Kamatani Y, Itakura T, Fujita Y, and Iida T (1992). "Neuro chips with on-chip back-propagation and/or Hebbian learning". *IEEE Journal of Solid-State Circuits, 27, (12), 1868 - 1876.*

Shoemaker P A, Carlin M J, and Shimabukuro R L (1991). "A back propagation learning with trinary quantization of weight updates". *Neural Networks, 4, 231 - 241.*

Shoemaker P A, Hutchens C, and Patil S (1992). "A hierarchical clustering network based on a model of olfactory processing". *Analog Integrated Circuits and Signal Processing, 2, 297 - 311.*

Sigvartsen R L (1994). *An analog neural network with on-chip learning.* Unpublished MSc dissertation, Universit of Oslo Deparment of Informatics, August.

Soelberg Y, Sigvartsen R L, Lande T S, and Berg Y (1994). "An analog continuous-time neural network". *Analog Integrated Circuits and Signal Processing, 5, 235 - 246.*

Suchman L A (1987). *Plans and situated actions : the problem of human-machine communication.* Cambridge University Press, Cambridge.

Tam S M, Gupta B, Castro H A, and Holler M (1990). "Learning on an analog VLSI neural network chip". In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Los Angeles, 701 - 703.*

Tarassenko L and Tombs J (1993). "On-chip learning with analogue VLSI neural networks". In *Proceedings of the Third International Conference on Microelectronics for Neural Networks, pp 163 - 174.*

Taylor F (1947). *Scientific Management.* Harper and Row.

Theeten J B, Duranton M, Mauduit N, and Sirat J A (1990). "The LNeuro-chip : a digital VLSI with on-chip learning mechanism". In *Proceedings of the International Neural Networks Conference, Paris, 593 - 596.*

Thornton C J (1992). *Techniques in computational learning.* Chapman and Hall, London.

Tomberg J and Kaski K (1991). "Digital VLSI architecture of back-propagation algorithm with on-chip learning". In *Proceedings of the International Conference on Artificial Neural Networks (vol 2), Espoo, Finland, 1561 - 1564.*

Valle M, Caviglia D D, and Bisio G M (1992). "An experimental analog

VLSI neural chip with on-chip back-propagation". In *Proceedings of the 18th European Solid-state Circuits Conference, pp 203 - 206.*

van Daalen M, Zhao J, and Shawe-Taylor J (1994). "Real-time output derivatives for on chip learning using digital stochastic bit strean neurons". *Electronic Letters, 30, (21), 1775 - 1777.*

Wang Y (1993a). "A modular analog CMOS LSI for feedforward neural networks with on-chip BEP learning". In **.

Wang Y (1993b). "Analog CMOS implementations of backward error propagation". In *Proceedings of IEEE International Conference on Neural Networks, San Francisco, 701 - 706.*

Watola D and Meador J (1992). "Competitive learning in asynchronous-pulse-density integrated circuits". *Analog Integrated Circuits and Signal Processing, 2, 323 - 344.*

Winograd T and Flores F (1986). *Understanding computers and cognition : a new foundation for design.* Ablex, Norwood, New Jersey.

Zeki S (1993). *A vision of the brain.* Blackwell Scientific Publications, Oxford.