UNIVERSITY OF OSLO
Department of informatics

An Analog Neural Network with On-Chip Learning

Roy Ludvig Sigvartsen

Main Subject Thesis

August 11, 1994

Preface

The work presented in this Cand. Scient. thesis has been carried out at Dept. of Informatics, University of Oslo. I started working on my thesis September 1992 and the work has given me a stimulating and challenging time here at the institute. I thanks my supervisor Yngvar Berg and Tor Sverre Lande for all advice and help in the process which have resulted in this thesis. I would also thank the colleagues in the analog VLSI group for always giving constructive talks.

Thanks to my parents for always encouraging me in the work. And finally I want to express my special thanks to Heidi who has supported me both financially and emotionally through the research and writing.

Blinderen, August 1994

Roy Ludvig Sigvartsen

Contents

List of Figures

1 Introduction

1.1 Analog neural networks

Today, fast digital computers help humans in daily tasks. However, in most tasks the human brain is superior to the computers. A good example is the processing of visual information. If we could copy a bit of the architecture in the brain, very powerful computers may be built. Therefore neural network models have received an extensively increased attention in the past 10 years. The models are drawn from our current knowledge of biological neural systems. An example is the use of “neuron” and “synapse” which can be found in brains and neural networks.

Our aim is to build systems that can understand images, speech and other similar tasks in the human world. Most of the developments in this research field is implemented on digital computers. The recent growing interest in the field also originates from the constantly development of faster digital computers.
Why are neural networks so attractive? They are valuable on several aspects:

- They are adaptive: they can learn from new data.
- They can generalize: they can classify data which only broadly resembles the already learned data.
- They can handle imperfect or incomplete data, offering a degree of fault tolerance.
- They are non-linear: they can capture relationship of large complexity.
- They are highly parallel: their operations can be executed simultaneously.

The parallel computations in neural networks are not being completely exploited in existing solutions since most of them are being executed on serial computers. However, since the computers in the past years have been substantial faster, acceptable results are still being obtained. Special purpose parallel hardware that utilizes the massively parallel processing in neural networks have a great commercial potential. In this research field digital, analog and mixed digital/analog systems have been proposed. A review of the latest publications in this field shows an advancement of analog systems.

It can be explain with the following arguments:

☞ In analog systems (and neural networks) a high degree of fault tolerance is allowed since it may not be critical if a few transistors do not function. This is not true for digital systems.

☞ It is possible to build circuits which have a remarkable power of computation compared to their sizes and complexity. One example is the computation of the activation function and its derivative (in the neuron). The analog circuit computing these two functions only involves 7 transistors! Such a high computational density is impossible for digital systems to achieve.

☞ Using analog CMOS, low power consumption is achieved, especially when operating the CMOS transistor in the subthreshold region.

☞ The brain is also “analog”. Understanding information processing in biological systems in addition to the physics of analog signals, even more efficiently signal processing in neural networks can be obtained. An example is the summation of synapses at the input to a neuron. When using current as the output signal type for all synapses, only hard wiring of the synapse outputs are necessary to perform summation.

However, analog systems are not robust to noise, but that is not a requirement for neural networks. Besides, it has been shown that noise assists neural networks to learn [Hertz][Lehman][Murray].

1.2 The motivation of this thesis

You may have already guessed from the above discussion that the subject of this thesis is an implementation of a neural network in analog CMOS. The neural network will be of feed-forward type and the learning algorithm is back-propagation. The main basis of the thesis is work done by [Soelberg]. He showed how a neuron and a weight (synapse) can be build in analog CMOS. The proposed network implementation (a 2-1-1 network) did not worked as expected. The pros and cons described by [Soelberg] were the fundament for this thesis.

A chip with a sample network is implemented and it will be shown how it is possible to obtain on-chip learning for an analog neural network with back-propagation learning.
A long term analog memory with UV-light adaption is used as the weight storage elements. Current-mode differential signals are used as the main signal type. A special weight updating scheme is used since the network is continuous in time and no clocking is required. The pattern presentation interval and the UV-light intensity determine the learning rate η and the size of the weight increments. A minimized weight module is obtained including a multiplier of smaller size and larger linear operation range than the multiplier

used by [Soelberg]. The analog memory and the coupling between weights and neurons are improved.

The thesis is organized into 6 sections. The second section describes fundamentals of feed-forward neural networks and the back-propagation algorithm. Building blocks in analog CMOS which can be used in neural networks are discussed in chapter 3. How analog CMOS circuits may be connected to compute neural network operations are focused in chapter 4. Chapter 5 discusses our implementation of a 4-3-2 network and on-chip learning of the network. The discussion and conclusion are included in chapter 6.

Four extra sections are included after the reference list. Appendix A gives a short introduction to analog CMOS. Appendix B contains a history which describes back-propagation. Appendix C gives a more detailed summary of the chip which includes transistor diagrams of the neuron and the weight modules in addition to a description of the input and output pads on the chip. Appendix D includes a paper published in the journal: *Analog Integrated Circuits and Signal Processing*.

2 Artificial Neural Network

There exist several classes of neural network architectures. It is not always easy to choose the most appropriate architecture for a given problem. The first thing to do is to examine your training set. If it includes the correct outputs (targets) you can choose networks containing "learning with a teacher" (supervised learning). Applying supervised learning a direct comparison of the outputs of the network with known correct answers is carried out.

However, sometimes your training set do not includes any learning goals. Then the only information available is the correlations of the training pairs. The network is expected to create categories from these correlations (unsupervised learning). And the outputs can be a clustering, a dimensionality reduction or a feature extraction of the inputs.

The future goal for our neural network implementation is to learn speech-, pattern recognition and other similar human tasks. For these problems the answer for a given set of input is known, but the function is unknown. Thus an implementation of a neural network with supervised learning is our choice.

2.1 Feed-forward network

The power of multi-layer networks was realized already in the late 60s, but only when Rummelhart and McClelland [Rummelhart] showed how to make them learn, these networks appeared to be useful. The network topology we are going to use includes two layers as illustrated in figure 2.1. (The rule for this thesis are: when counting the number of layers, the input layer is not included.) These two layers are the hidden layer and the output layer.

Each layer has a number of units (or neurons) and to each of these neurons many synapses (weights) are connected. For each neuron in the layer below it exists a weight to each neuron in the layer above. The weights job is to scale the contribution from the neuron in the layer below. Input to a neuron is a summation of all the weights connected to the neuron. The output of a neuron is a threshold function of its input. We may choose the threshold function (or activation function) to be either a sign-function, a linear or semi-linear function, or a sigmoid function.

We use a sigmoid function because this function gives us continuous-valued outputs which are nonlinear, derivable and kept between fixed bounds. A sigmoid function is easy to implement in analog CMOS.

A neuron has a bias (threshold) which will ensure that the output of the neuron is non-zero if the input is zero. A feed-forward network can map any function only if the right architecture is used. The number of hidden units must be a choice which depends on the complexity of the input patterns.

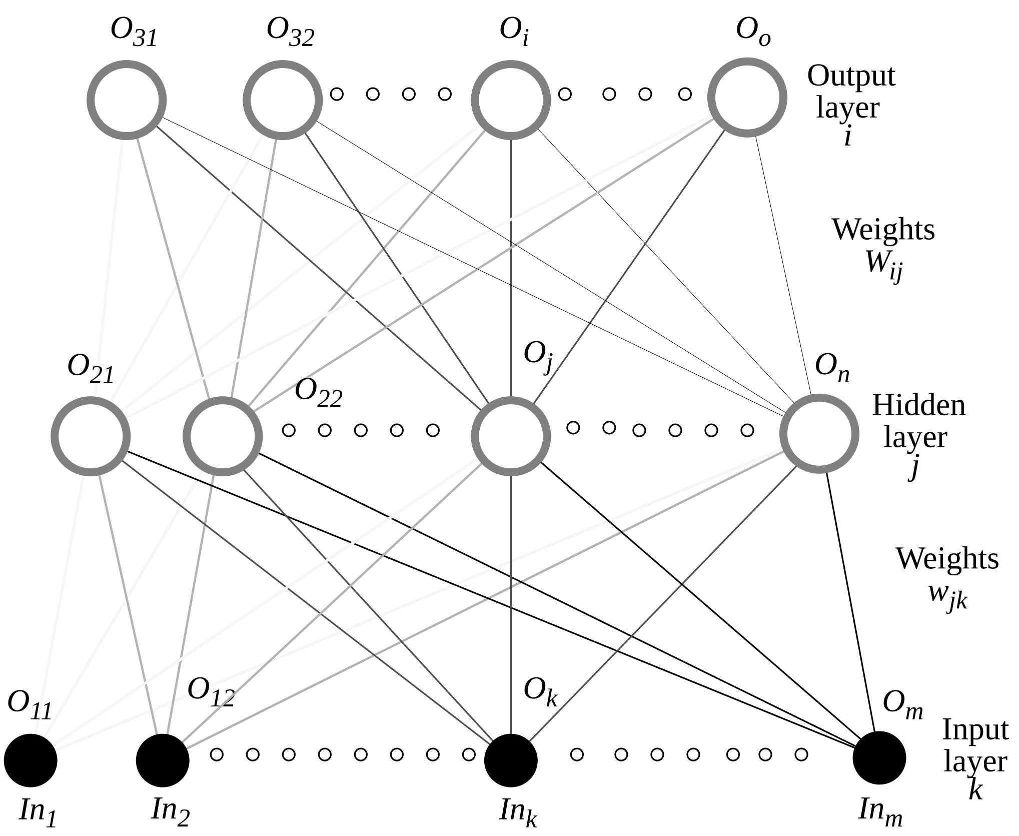


Figure 2.1 : ***A two layer feed-forward neural network.***
When counting the number of layers, the input layer is kept outside. Each line between two neurons is a synaptic connection (a weight) which performs a multiplication $(o \cdot w)$ The network is a m-n-o sized network (m input, n hidden, and o output neurons).

2.2 Back-propagation

To recall a function with a feed-forward network, a set of weights has to be found that performs the desired mapping. The back-propagation (backprop) algorithm [Rummelhart] is an optimized scheme to find a solution set. The algorithm is based on a gradient descent optimization procedure and should be thought of as an algorithm for computing $\frac{\partial E}{\partial w}$ for each weight in the network.

I heard a good story once from the Internet (author: Warren Sarle) which explains the backprop algorithm in a funny way. This story may be found in Appendix B. The algorithm is based on a training set of patterns which contains input and target vectors. Each vector (or pattern) is presented for the network in a repeated order and the error, which measures how far away the network is from a solution set, can be found. A typical way to calculate the error is to use the sum-square error measure:

$$E = \frac{1}{2}\sum_{pi}\left(O_i^p - t_i^p\right)^2 \quad \text{where} \tag{2.1}$$

O_i^p is the output of neuron *i* for pattern *p*,

t_i^p is the correct output (target) of neuron *i* for pattern *p*.

As evaluated in many books and articles [Hertz] [Rummelhart] the updating rules for the weights using the error measure in eq. (2.1) and following gradient descent ($\frac{\partial E}{\partial w}$) are:

For the hidden layer to output neuron connections:

$$\Delta W_{ij} = \eta \delta_i O_j \quad \text{where} \tag{2.2}$$

η is the learning rate which decide the step size,

δ_i is the error computed in the output layer and

O_j is the output of the hidden neuron *j*.

For the input layer to hidden neuron connections:

$$\Delta w_{jk} = \eta \delta_j O_k \quad \text{where} \tag{2.3}$$

δ_j is the error computed in the hidden layer and

O_k is the output of the input neuron *k*.

Before computing the weight changes, the errors have to be calculated:
The error calculated for the output neuron:

$$\delta_i = (t_i - O_i)\, O'_i \quad \text{where} \tag{2.4}$$

O_i is the output of the output neuron i and
O'_i is the derivative of the output neuron i.

The error calculated in the hidden neurons:

$$\delta_j = O'_j \sum_i W_{ij} \delta_i \tag{2.5}$$

Note that the δ-errors are propagated from output down to input in an opposite direction of the feed-forward signals, hence the name back-propagation. In the original standard backprop algorithm the weight updating was employed only after all the patterns were presented[1]. This approach required additional accumulation storage for each weight. The more commonly used method is to update each weight before presenting a new pattern (on-line updating). We do not truly follow the gradient $\frac{\partial E}{\partial w}$ because the network calculates a new error between each pattern presented. The network will then have different values of E for each pattern.

2.3 Learning with backprop

To find out what is required for a network to learn its training patterns is a non-trivial problem and depends on several parameters which will be discussed.

Convergence

A network converge when the error E (in eq. (2.1)) has reach a limit ε. The limit describes how large error rate we can tolerate. If the limit is small, let us say that only 0.01% error is tolerated, the network may not converge or the time it takes to converge will be large. Then it is possible for the network to become overtrained(i.e. bad generalization). A moderate choice of the limit ε is necessary to ensure convergence.

Initial conditions

The theory says that a network should start with randomly chosen weights. If we pick large weight values which will give us derivatives approximately equal to zero, the convergence time will increase rapidly and the backprop algorithm may be stuck in a local minimum. To avoid such situations it is important to pick random weights which gives neuron output values in the dynamic switching range of its activation function.

If you know something about the mapping function, it would be an idea to set the weights closest to the values you think the network will have at the end of the training.

1. This method is called off-line updating (batch mode).

Local minima

If the backprop algorithm falls into a local minimum, it may be stuck. Local minima may result in that networks will never converge or they will converge to a wrong solution, Usually, networks which have fallen into a local minimum use long time to get out it and the convergence time will be increased. A typical local minimum is one in which two or more weight updates cancel each other. To avoid such cancellations we can either add a *little* noise to the patterns or present the patterns in random order [Hertz].

Mapping function and the architecture

In most cases the architecture of a neural network is dependant on the type of function we want to map. The number of hidden neurons have to be varied with the complexity of the mapping function and of course also with the number of inputs. If you use too few hidden units, the network may not converge. If you use too many, the network gets too many free parameters and this may result in over-fitting[1].

In order to achieve good generalization the training set have to be large. A rule of thumb is to use more training patterns than the number of free parameters included in the network. If we do not have such a large set, one possibility is to enlarge the set with many noisy variations of the original set. The convergence time will increase, but we get a much better generalization.

Learning rate η

It is difficult to know the optimal value of η. Some use a constant value, others may alter the value as a function of time and others again use an adaptive variation of the value η. The adaptive choice may automatically regulate η after the following rules:

- When the backprop algorithm in the last repetitions has decreased the error E, the algorithm may increase the speed of the reduction in the error E by increasing η.
- When the error E has been increased in the last repetitions, η should be decreased.

Due to this adjustment a more efficient and optimal training is achieved, however, in a hardware implementation such adjustment will involve unwanted extra control logic. In our implementation the most desirable and cheapest way of setting η, is to update it as a function of time. Two adjustment techniques have shown good results in simulations. The first is to start with a high value of η (large step size) and then decrease it slowly. The second technique is to first start with a low rate (small step size) and increase it for a while before we decrease it again.

1. The training patterns will give a small error but new patterns which is unknown for the network will give a large error rate. (Bad generalization).

3 Basic ANN computations in Analog CMOS

Those who are familiar with the CMOS transistor operating in week inversion may continue reading. Otherwise you should read Appendix A first.

If you review the mathematical neural network equations in chapter 2, you can see that we need to implement basic operations as summation, subtraction and multiplication in our implementation. If the implementation should be a digital system we would begun to build an adder, a subtracter and a multiplier. However, building an analog system, we should take advantage of the physics of the two signal types in analog system: voltage and current.

Using current signal representation we may easily achieve summation (and subtraction) by only physically connecting signals together (Kirchhoffs current law). By using the CMOS transistor in weak inversion we obtain an operation range for current signals from *fA* up to *nA* (theoretically).

Voltage signals have the advantage that they may be distributed to many high-ohmic nodes (as gates on a CMOS transistor). Voltage signals should be applied when a neuron output is assigned to many synapses.

Multiplication can be performed by various circuits. Important issues when choosing multiplier are: linear range, offset problems, size and type of input/output signals.

Another mathematical operation we need is derivation, which is not a trivial operation to implement in an analog system. We do, however, this in an elegant manner as will be shown later.

When choosing the circuits for our network, we have to think of how they may be connected together. Some circuits need differential input/output representation and other single representation. With one signal connection between each circuits a large amount of routing space is saved. However, a reference signal has to be routed to those circuits having differential signal input. In addition, some of these circuits may have different demands on the reference signal value, which means that several reference signals have to be applied on the chip. With two connections between each circuits, each signal can be split into a positive and a negative component. When routing this type of representation, you use a lot of extra space. But you do not have to deal with reference signals and global routing. So we have chosen mostly differential current signal representation.

Another solution would be to use a combination of both single and differential representation. Linking two or more circuits we could use one bidirectional current signal. And for those circuits which need differential inputs a small converter could be applied to convert the one bidirectional signal into two unidirectional signals. This inquire that the converter should only contain of a few transistors if this representation should be of any advantage.

3.1 Multiplier

There exist several multiplier circuits operating in weak inversion. A major problem for these multipliers is the limited linear range. Often they do not satisfy the requirements of accuracy in certain calculation because of transistor mismatch and temperature variations. It is very important to choose the right multiplier and we have tested various circuits to find multipliers which gives best accuracy and fits with the signal representation.

3.1.1 A modified transamp with increased linear operation range

A transconductance amplifier (transamp) is a circuit that amplifies a voltage difference into a current signal. The current output signal is scaled by a bias current I_b determined by a bias voltage V_b. The scaling can be viewed as a multiplication. When using the transamp as a multiplier we wish to operate it in the linear output range. However, this region is small, only 60mV linear operation range. It exist a number of techniques to increase the linear region, including capacitive division and source degeneration.

A widely used source degeneration technique for the transamp is described in [Watts] and is illustrated in figure 3.1. In the circuit it has been placed two diodes between the differential pair and the bias transistor. With this extension the linear range is increased to 144mV, but the common-mode operating range will be reduced significantly. To guarantee such a inflexible input restriction, extra logic on the inputs have to be included. The result of this will be an increase of the network size instead of a minimizing.

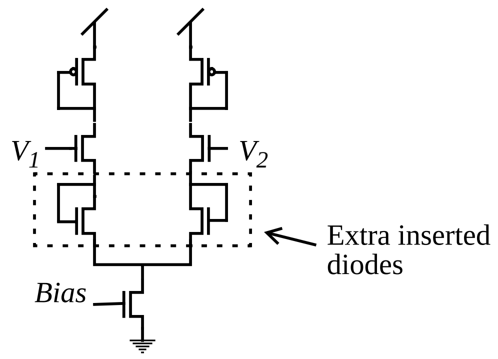


Figure 3.1 : ***A simple source degeneration technique to increase the linear range.***
The disadvantage of this circuit is the reduced common-mode operation range.

Another source degeneration technique is described in [Torrance]. In this technique several differential pairs may be connected in series in such a way that the input voltage is divided *n* times if the number of sections are *n*. The voltage input to each differential pair is $(V_1 - V_2)/n$.

An example with three differential pair sections are shown in figure 3.2 a) and performs the function:

$$I_{out} = I_1 - I_2 = I_b \tanh\left(\frac{\kappa q (V_1 - V_2)}{6kT}\right) \tag{3.1}$$

The linear range is increased to 180mV and the common-mode operating range is not reduced. If we only use two sections the linear range becomes 120mV. By using p-type diodes instead of n-type, as shown in figure 3.2 (b), the linear range increases to 140mV. A *hspice* simulation of the circuit along with a simulation of a standard transamp is shown in figure 3.2 (d). The linear range of the modified transamp with *n*=2 is 2.3 times larger than the linear range of a standard transamp.

Equation (3.1) can be applied as a multiplications between a unidirectional current-signal (I_b) and a differential voltage signal ($V_1 - V_2$). The modified transamp in figure 3.2 (b) is chosen to perform the multiplications in the feed-forward computations. In these computations the linear range of the multipliers is critical. [Soelberg] proposed a Gilbert multiplier in the feed-forward computations. This multiplier contains of 11 transistors, has 5 signal input lines, and the linear range of the multiplier is only 60mV. The multiplier in figure 3.2 (b) contains of 9 transistors, has 4 signal input lines, and the linear range is 140mV. The dynamical behavior and the layout size are improved.

3.1.2 Four quadrant multiplier

A transamp can only perform a two quadrant multiplication. In some back-propagation calculations a four quadrant multiplication is required. One remarkable small and practical four quadrant multiplier is described by [Toumazou]. It is based on the translinear principle (first proposed by [Gilbert]) and is originally implemented with bipolar transistors. It is easy to convert bipolar transistors to CMOS transistors working in subthreshold region, since the drain current has an exponential behavior. Unfortunately simulations of this multiplier showed some offset problems that will limit the usage of this multiplier.

Another well-known four quadrant multiplier is the Gilbert multiplier [Mead]. It performs the function:

$$_{out} = I_b \tanh\left(\frac{\kappa}{V_T}\frac{V_1 - V_2}{2}\right)\tanh\left(\frac{\kappa}{V_T}\frac{V_3 - V_4}{2}\right). \tag{3.2}$$

In this equation two differential voltage signals are multiplied. V_T is the termal voltage kT/q and at room temperature it is equal to around 25mV.

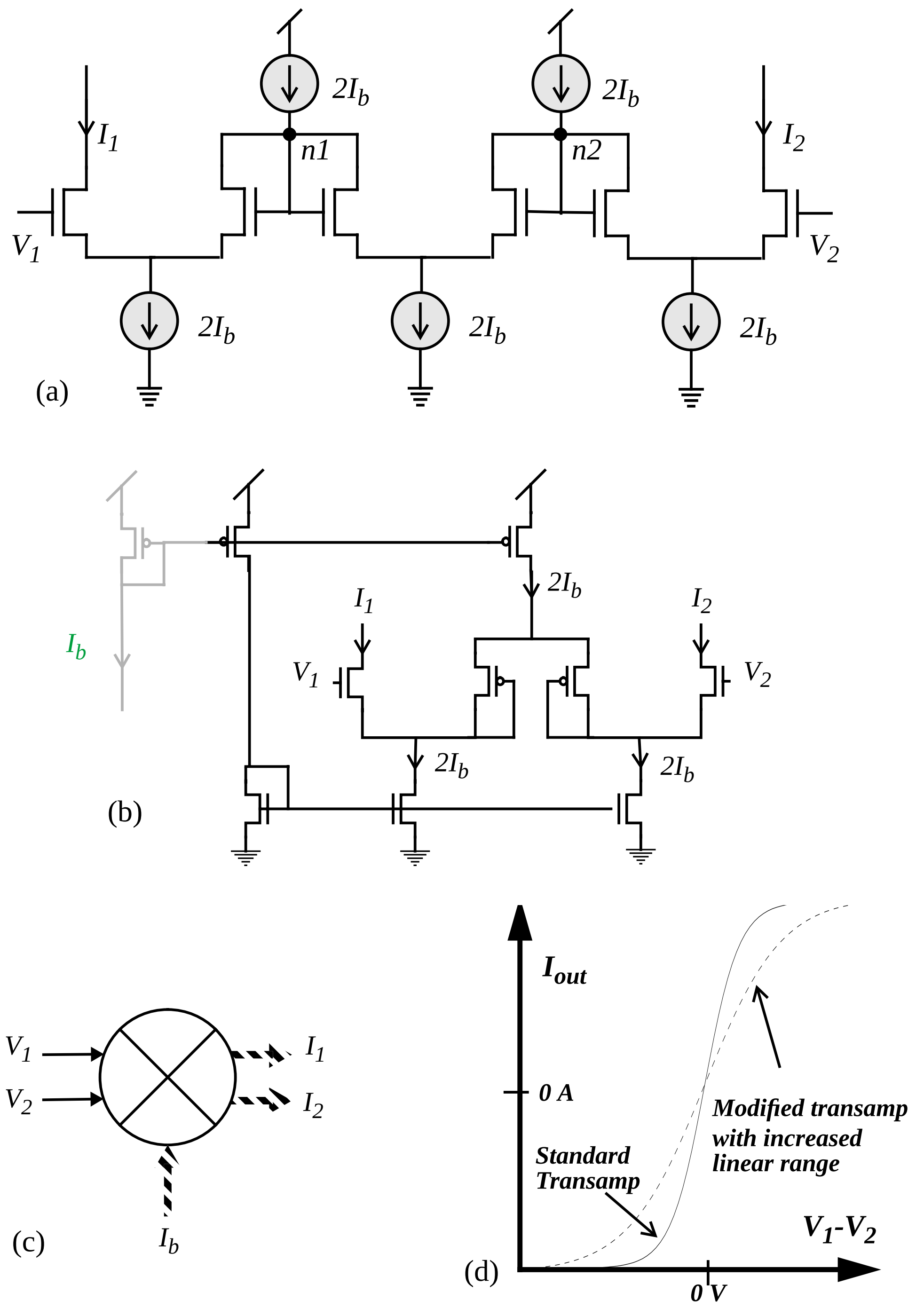


Figure 3.2 : ***A modified transamp with increased linear range for use in multiplications. (a) demonstrate the principle, with three differential pair sections. The voltage between*** V_1 ***and*** $n1$, $n1$ ***and*** $n2$, $n2$ ***and*** V_2 ***are*** $(V_1 - V_2/3)$. ***The linear range will increase by a factor of three. (b) illustrates the circuit used in our network. It has two sections and uses p-type diodes instead of n-type to increase the linear range.***
(c) shows the chosen symbol for the modified transamp when it is used as a multiplier and (d) shows a simulation of both the standard and the modified transamp. The linear range is increased by 80mV.

By some modification the multiplier may perform the function:

$$I_{out} = (I_1 - I_2) \tanh\left(\frac{\kappa}{V_T}\frac{V_3 - V_4}{2}\right) \qquad (3.3)$$

Equation (3.3) is a multiplication between a differential current signal $I_1 - I_2$ and a differential voltage signal $V_3 - V_4$.

3.2 Analog UV-memory

One of our goal in the VLSI implementation of a neural network is to store and update an analog memory on-chip. A promising approach is the floating gate technique. With this technique we may store a charge on an isolated gate of a transistor (floating gate) with extreme low leakage. Adjustment of the voltage on this gate may be accomplished either by Fowler-Norheim tunneling, hot carrier injection or ultraviolet (UV) light exposure which will be used in our implementation. By using UV-light exposure we are able to inject electrons through (the edges of) a CMOS capacitor onto the floating gate node. Thus it is possible to increase or decrease the value on the memory by small steps which is required by many neural network algorithms.

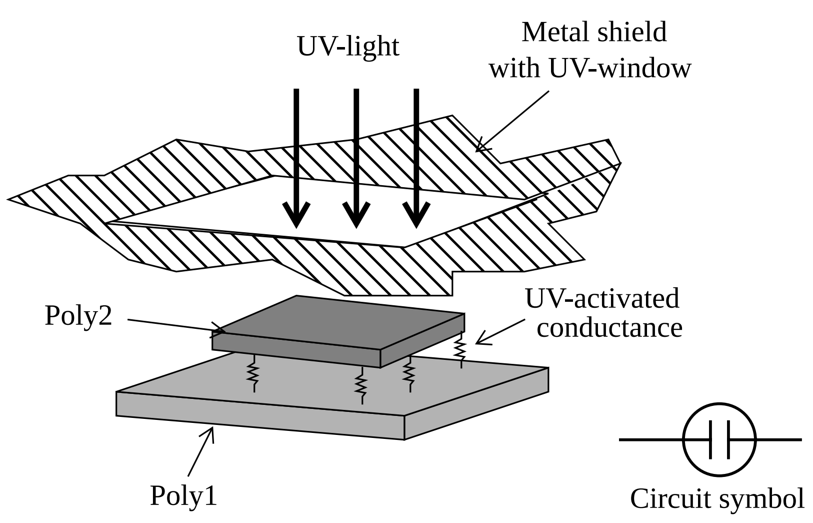


Figure 3.3 : ***Physical view of the UV-structure.***
The poly1 layer is connected to a gate of a transistor. The UV-activated conductance is also drawn in the figure.

3.2.1 Physical description

To program the floating gate, a capacitor, made of overlapping poly1 - and poly2 layer has to be connected to the gate. The insulator, silicon dioxide (SiO_2), is separating these two silicon layers, but when SiO_2 is exposed by UV-light, it conducts with a small conductance.

To ensure that only the UV-structures is exposed, we use a metal shield over the rest of the chip. However, a small amount of UV-light waves are being reflected under the metal shield. Benson and Kerns [Benson] showed that the reflection under the shield attenuates exponential as a function of the distance from the UV-window. Since the UV-activated conductance is depending on the UV-light intensity, UV-structures with different time constants can be build. Moving the UV-window a bit away from the UV-structure, we achieve a smaller UV-activated conductance which means a larger time constant. A memory with an UV-window not exactly above the UV-structure is applied in our implementation. To prevent reflections under the shield we have in our design built guard rings of poly1 to metal1 contacts outside the UV-structure.

The edges of the capacitor in the UV-structure are one of the factors that decide the size of the UV-activated conductance. To achieve high programming speed it is important to layout the capacitor with long edges.

3.2.2 Circuit description

The poly1 node in figure 3.3 is usually called the control node while the poly2 node is the floating gate node. The UV-activated conductance is a nonlinear function of the voltage difference between the control gate and the floating gate [Maher] [Benson]. Especially for small voltage differences the conductance will be small. Benson and Kerns [Benson] proposed a tanh-function dependency. Our approach is to always keep a large voltage difference between the control gate and the floating gate in order to achieve higher programming speed.

[Maher] introduced a second capacitor C_{cap} connected to the floating gate to remove the total load capacitance on the floating gate. The other input to this capacitor is an inverted input V_{cap} of the control gate voltage V_{cg} as illustrated in figure 3.4.

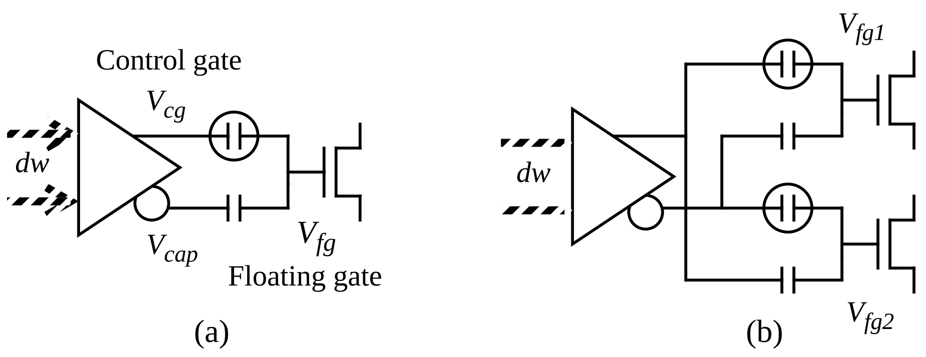


Figure 3.4 : ***Circuit description of the analog UV memory.***
V_{cap} ***is an inverted voltage of*** V_{cg} ***. dw is a current difference which is amplified to a large voltage difference. Figure (a) shows a typical floating gate memory and figure (b) shows a differential voltage representation which will be used in our implementation. The actual value of the memory is then*** V_{fg1} -V_{fg2}.

To remove the total load capacitance the signal V_{cap} and V_{cg} have to have symmetrical characteristic. [Soelberg] introduced a digital inverter to invert V_{cg}, but this led to an unstable floating gate voltage on V_{fg} when V_{cg} and V_{cap} switched. An improved solution will be to let a transamp with a differential output control V_{cg} and V_{cap} as shown in figure 3.5. One important detail with this implementation is how the voltage on V_{cap} is computed. To minimize the symmetrical switching mismatch between V_{cg} and V_{cap}, V_{cap} is computed directly from V_{cg}. The only source of errors in the symmetrical characteristic will be the current mirrors inverting the voltage V_{cg}. The extra time used to invert V_{cg} is not a problem since the time constant of the UV-memory is considerable larger.

The unsymmetrical characteristic of V_{cg} and V_{cap} may affect the voltage on the floating gate due to the capacitive division of the V_{cg} and V_{cap}. To avoid such disturbance after finished programming, V_{cg} and V_{cap} can be set to a fixed voltage level. Locking V_{cg} and V_{cap} at a fixed value may be accomplished by using n-type transistors to pull down the voltages on the control gate and V_{cap}.

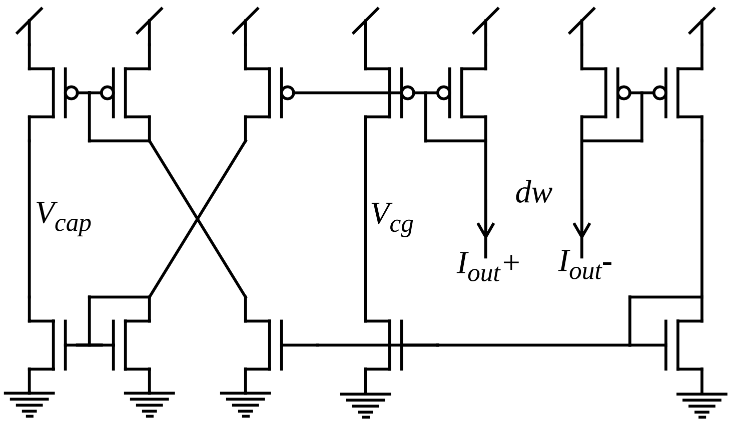


Figure 3.5 : ***The amplifier stage used to program the UV-memory. The current inputs $I_{out}+$ and $I_{out}-$ are outputs from a differential pair.***

3.2.3 Resolution of the UV-memory

All neural networks have to include some kind of storage of the weight values. A well-known problem for the storages is if they have the required resolution. The main reason is that these neural network storage mechanisms in some way touch the digital regime. Either the storage is totally digital or the long-term storage is digital but before/after training these values are put through a DAC/ADC. A digital value is always quantized and it is necessary to know how many bits is required to obtain a wanted accuracy.

An analog storage, however, that is never digitalized, will not have an exact lower level of accuracy because of a quantization problem. The accuracy in analog storages are determined by the level of noise included in the storages.

[Tarassenko] reports a detectable accuracy of 1:1000 on their analog memory (10 bits accuracy). The memory value is held on the gate capacitance of a FET. The disadvantage of this type of analog storage is that it needs to be refreshed regularly. After finished training all the memories are digitally saved. With 1:1000 accuracy on both the memories and the analog multipliers used they have successfully simulated on-chip learning for a neural network.

Our UV-memory alone has an higher accuracy of 1:1000 if the whole operation range is used. However, our problem is that only a constrained range of the memory is interesting because the output of the memory is an input to a multiplier with a linear range of 140mV (see figure 3.2). If the memory shall work inside the linear range with an accuracy of 1:1000, the UV-memory has to achieve programming steps small as 0.14mV.

[Murray] talks about how analog noise in the memories actual assist the learning process in neural network. He says that you may see the inaccuracy in analog memories as spread of “actual” values of the memory. But the memories maintain its accuracy as a time average. And if the learning process is sufficiently slow, as is for our network, it “sees trough” the relatively low levels of noise in an analog system. So analog memories have fundamentally different accuracy problems than digital memories. And therefore it is probably not meaningful to talk about exact values of the resolution of our memory and compare it with digital resolution.

[Lehman] demonstrated that constrained analog weights with added noise, enhanced the probability of learning in neural networks. The weights used in his network correspond to the weights in our network in a such way that they are constrained and they include noise. Thus the accuracy of our weights do not have to reach a limit before a successful learning of the network is obtain.

3.2.4 Measurements on the UV-memory

Usually we look at programming steps as voltage increments between two measurements. In a neural network the time between two measurements may be equalized to the time each pattern is presented. If we want to decrease the steps, we may increase the measure frequency (decrease the time each pattern is presented) or decrease the UV-light intensity.

The UV-light source used, is an old eprom-eraser which expose light with wavelengths of 254nm and with an effect of 4W. To adjust the intensity of the UV-light exposed to the chip, two methods can be used. The obvious one is to alter the distance between the chip and the light source. The light intensity has an exponential dependency on the distance. The second method is to filter the light exposed to the chip with various degree of attenuation. We have two filters that only amplify lights with wavelengths around 250nm. The first one (filter A10) has an amplification of 0.1 at 254nm and the second (filter B24) has an amplification of 0.24 at 254nm. Measurements showed that the programming speed of the weights had an exponential dependency on the difference between applying filter B24 and filter A10. For small UV-light distance the difference was large (at 3cm: 8.7 times), and for larger distances the difference was smaller (at 12cm: 4 times)

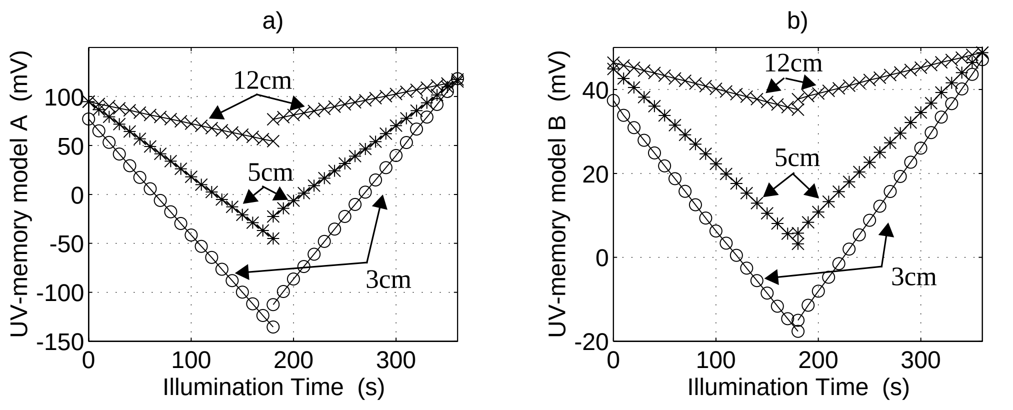


Figure 3.6 : ***Measurements on the UV-memory.***
Examples of how a memory can be programmed are shown. Figure (a) shows the fastest changing memory (model A) and figure (b) shows the slower memory (model B). For both figures the UV-light source distance has been altered between 3cm, 5cm and 12cm. The filter B24 has been applied for all measurements.

Two types of the UV-memory with different time constants have been applied. The UV-memory model with smallest time constant (model B) has a half size UV-window that is moved 6µm to the left of the floating-gate node (see figure C.5). The window is placed in a such way that UV-light is only exposed to one of the edge of the capacitor poly1-poly2 (capacitor between control node and floating-gate node, see also figure 3.3). The time constant for the memory was decreased by a factor of 3.3 (mean value).

When updating the UV-memory we may only increase or decrease the value stored. Figure 3.6 shows both increasing and decreasing values of the two different UV-memory models (model A and model B) for various light-source distances. From figure 3.6 you may see that when shifting from negative to positive programming steps the value on the memory jumps about 20mV. This is due to non-symmetrical behavior of the amplifier stage in figure 3.5. Especially when I^+_{out} and I^-_{out} (shown in figure 3.5) are not symmetrical matched (the common mode signal $I^+_{out} + I^-_{out} \neq 0$) this effect will occur.

Figure 3.7 illustrates the problem discussed above for three UV-memory models of type A and for one model of type B. Compare to the UV-memory discussed in [Soelberg] our UV-memory has a considerable improved switching characteristic, but may still be improved. A particular unwanted offset is the voltage difference obtained after the input has switched sign. To reach a higher level of accuracy, such offsets have to be removed.

If you compare the switching characteristic of model A and B in figure 3.7, model B has a significant smaller offset error than model A. The inconsistence between these two models are due to a very poor layout of the UV-memory model A. The routing of the floating gate node is unfortunately not kept at a minimum. Extra gate capacitances are included for the model A due to the unnecessary routing of the floating gate. In addition, the routing area is probably different for the two UV-structures included in the differential memory circuit which can be observed as a larger switching mismatch of the memory. The UV-memory model B do not suffer of such a horrible routing. Therefore a more stable UV-memory should be obtainable for the whole network.

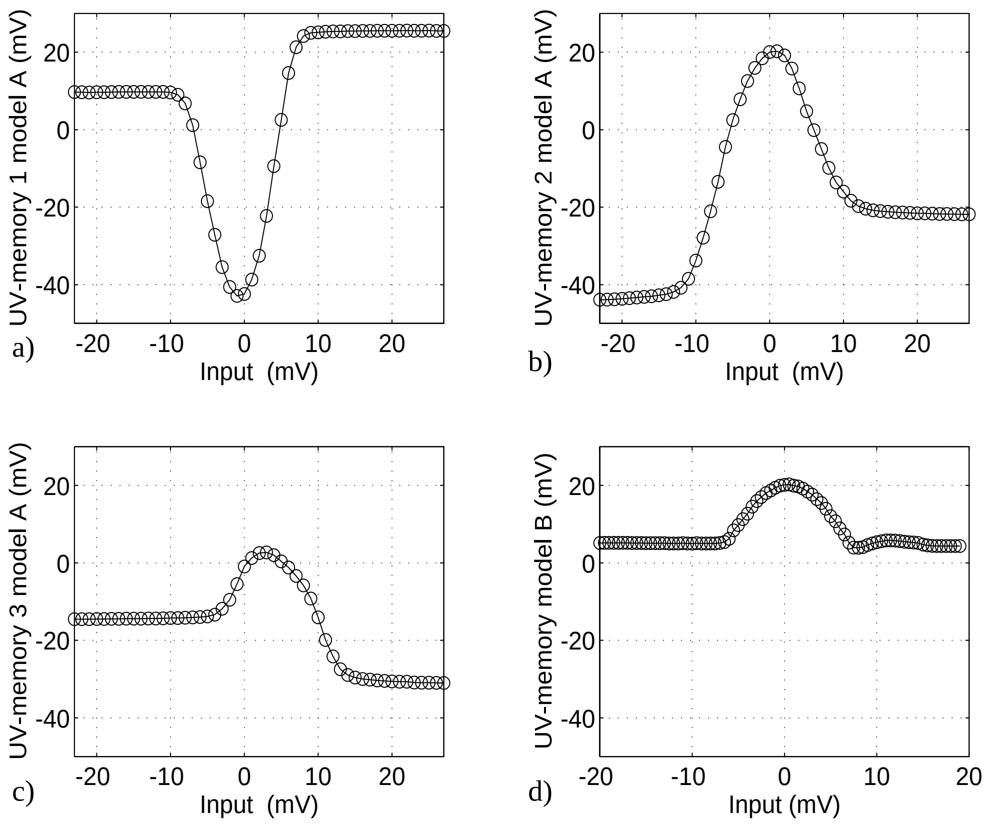


Figure 3.7 : ***Measurements of the dynamical behavior of four UV-memories. The figures show the characteristic of four UV-memories when the programming direction is switched (UV-light source is turned off). The*** *input* ***is a differential voltage presented to a Gilbert multiplier for model A and transamp for model B. These two circuits produce the input signals*** I^+_{out} ***and*** I^-_{out} ***to the amplifier stage in figure 3.5.***

3.3 Other circuits

3.3.1 Sigmoid function

In the algorithm of neural networks, a threshold function is used in a similar way as the behavior of a biological neuron. For our network we have chosen the sigmoid functions:

$$(O_a+) - (O_a-) = \tanh(\beta h) \text{ and } O_b = \frac{1}{1+e^{-\beta h}} \tag{3.4}$$

Both of these functions can be calculated by transamps. A transamp which only output one of the currents in its differential pair (I_1 in figure A.5) will output O_b. A transamp which outputs both of the currents flowing in each leg of the differential pair will output (O_a+) and (O_a-). Figure 3.8 shows circuit symbols for the two activation functions, figure 3.8a) for O_b and figure 3.8 b) for $(O_a+) - (O_a-)$.

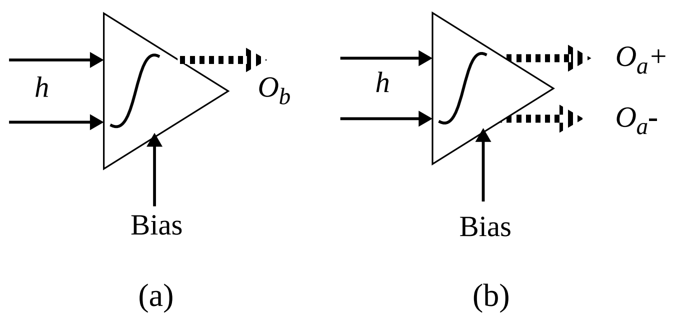


Figure 3.8 : ***Circuit symbol for the activation function circuits. Figure (a) shows a transamp with the output*** O_b ***in equation (3.4) (One unidirectional signal). Figure (b) shows a transamp with the output*** $(O_a+) - (O_a-)$ ***in equation (3.4) (two unidirectional signals) Solid lines are voltage and stippled lines are current signals.***

3.3.2 Derivation

We have to calculate the derivative of a sigmoid function. [Delbrück] showed how this may be accomplished (see figure 3.9). By only adding two transistors to the transamp circuit, we are able to calculate the derivative of a sigmoid function which is a powerful analog computation.

3.3.3 Subtraction of two voltages

To find an easy way to substract two voltages, we use a transamp to do this. When it is working in its linear range, a real scaled substraction is performed. That works well for our use.

3.3.4 Current to voltage converter

When connecting transamps with current outputs to Gilbert multipliers with voltage inputs, the current outputs have to be converted to voltage signals first. In the subthreshold region a current signal works over several order of magnitude. But the linear input voltage of a transamp or a Gilbert multiplier only works in the mV-region. A converter circuit should map the wide range current signal into a voltage within the constrained input range

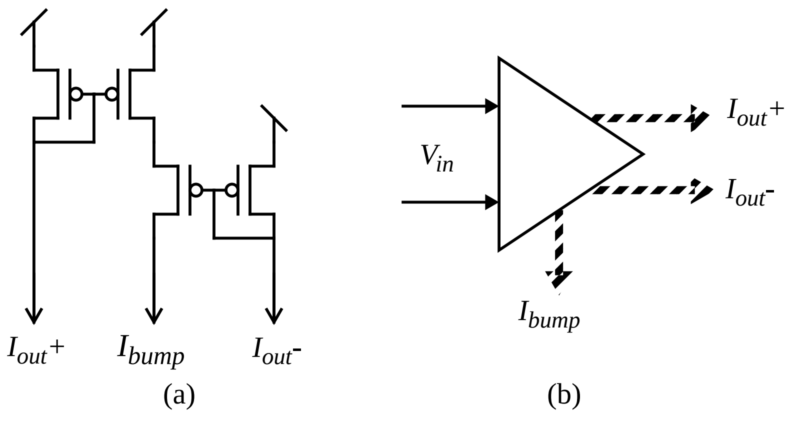


Figure 3.9 : ***Computation of the derivative I_{bump}.***
$I_{out}+$ and $I_{out}-$ are outputs of a differential pair and perform a tanh function when subtracting them. While I_{bump} performs a $1/cosh^2$ function which is the derivative of the tanh function. Figure (b) shows the circuit symbol for a transamp which also outputs the bump signal I_{bump}.

of transamps and Gilbert multipliers. One possibility is to use an inverse-sinh conversion [Kerns]. But this converter circuit includes around 11 transistors and uses a bidirectional current input and voltage output.

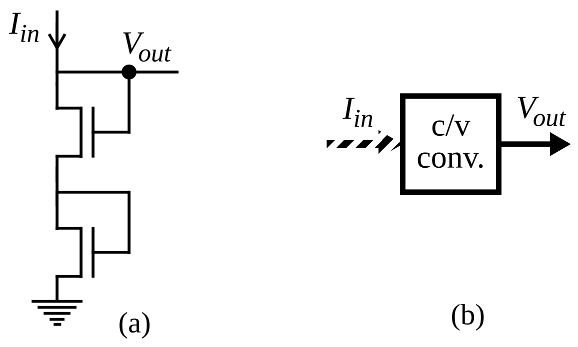


Figure 3.10 : ***A simple I-V converter.***
It converts a current signal to a voltage signal with a logarithmic compression. Figure (b) shows a symbol used in the rest of the thesis.

Another solution is to use a logarithmic compression conversion. This may be achieved by using diode-coupled transistors as show in figure 3.10. [Soelberg] used only one diode-coupled transistor for each signal. The operation range for one n-type diode do not exceeds 1.5V. If we add an extra diode at the top of the first one, we may get a more useful operation range (between 1V and 3V depending on the W/L ratio of the diodes). The conversion is proportional to:

$$V \sim 2\ln\left(\frac{I}{I_0}\right)$$

3.4 Summary

The basic circuits used in our implementation of a neural network have been presented. The circuits shown have been extensively tested in the simulators *hspice* and AnaLOG before they were pick out to be applied in the implementation. The proposed circuits are covering every operations required for a neural network. The two most important circuits in a neural network are the multipliers and the memories. These two types of circuits have been thoroughly discussed.

An modified transamp with increased linear range is proposed as the multiplier to be used in the feed-forward computations. A floating gate memory with UV-adapation is proposed as the storage of the weights. The memory circuit has an improved dynamical behavior compared to the circuit used by [Soelberg]. Two different memory circuits are applied, model A and model B. Model A, which has the smallest time constant, are being used as the weight storage while model B, which has the most stable behavior, are being used as the threshold storage.

Each circuit operates properly alone. However, putting them together into a neural network and trying to operate them in their operation range synchronously is not obtained straightforward. The problems relating to this work will be discussed in next chapter.

4 Feed-forward and Back-propagation Computations in Analog CMOS

A presentation of how an ANN are implemented in analog CMOS is described in this chapter. The first part will present the forward calculation and the second part the error and update calculations.

4.1 The Feed-forward computation

In chapter 2 the rules of a feed-forward neural net with back-propagation learning was described. This chapter will show how the feed-forward part may be implemented in analog CMOS. Forward calculation of the network is working when being in learning (training)- and in recall-mode. Thus it is important that this calculation do not diverge in these two modes.
Important issues for this part is:

- Which activation function to use.
- Which multipliers to use.
- Which signal representation to use.

In chapter 3 we answered these questions generally. Now we want to build a whole network with these selected circuits and signals.
In chapter 3.3 two different activation function for the neurons were selected:

$$g_1(net) = \frac{1}{1 + e^{-(\beta \cdot net)}} \tag{4.1}$$

$$ {}_2(net) = \tanh(\beta \cdot net) \tag{4.2}$$

net is the input to the neuron which is a summation of weights. These two activation functions can be implemented by a transamp (see chapter 3.3.1). The activation function in equation (4.1) is always positive and may be represented in analog CMOS as an unidirectional current signal. The input and hidden neurons use this activation function because it only requires one output signal for each neuron. With one input signal instead of two implies a simplification of the multiplier circuits connected to the activation function circuits. The output neurons use the activation function in equation (4.2). A conversion of the function in equation (4.2) to a digital function is easy. If the function is positive, then the output is logical high. If the function is negative, then the output is logical low.

$$O_j = I_{b_j}\frac{1}{1+e^{-\beta\left(\sum_{k=1}^{m} O_k \cdot \tanh(\alpha w_{jk}) + I_{ON} \cdot \tanh(\alpha\Theta_j)\right)}}$$

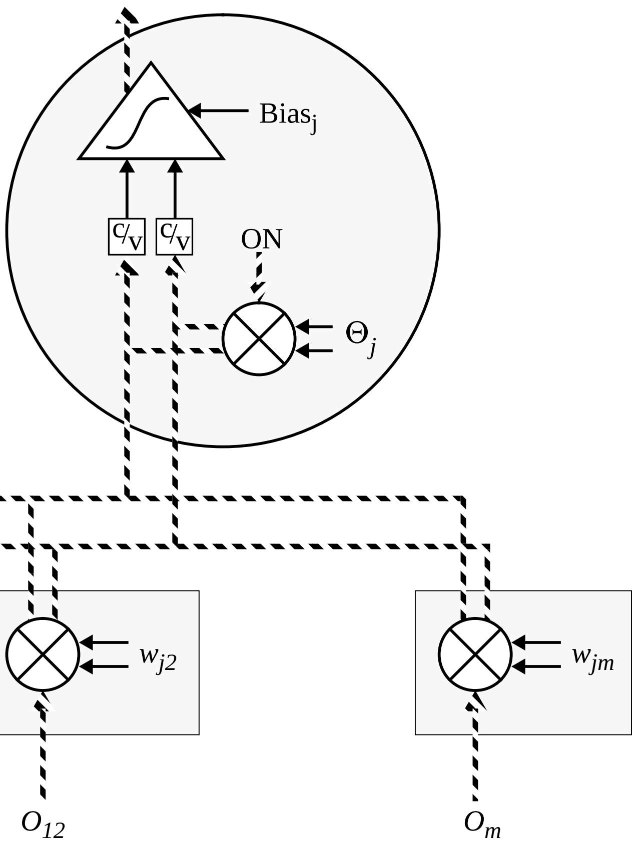


Figure 4.1 : ***Feed-forward computation implemented in analog CMOS. The figure describes the computations from input layer*** *k* ***to hidden layer*** *j*. ***The input*** O_k ***is the output from the neurons in the input layer. The large grey circle demonstrates the neuron-module and the gray boxes the weight-modules for feed-forward computations. Stippled lines are current signals and solid lines are voltages.***

The computation of hidden neurons is shown in figure 4.1. First every weighted neuron output from the layer below $_k \cdot w_{jk}$ is summed together with the weighted threshold $\Theta_j \cdot$ ‘ON‘ . All these signals are represented as currents to be able to use Kirchhoffs current law and just wire these signals together to perform the summation. The contribution of Θ_j is scaled by the input *ON* which can be looked at as a neuron that is always on.

In the equation in figure 4.1 it is assumed that the weights w_{jk} are inside the linear range of the multipliers. This consideration may not always be true but it is not critical for the back-propagation learning[Lont][Lehman]. However, it seems like that the size of the linear range of these multipliers is of more significance.

The current output O_j of the neuron described in figure 4.1 has a few extra parameters which are not included in the theory:

I_{b_j}	: Determined by Bias_j,
β	: Thermal voltage for activation function circuit, Defined as $\kappa q / 2kT$,
α	: Thermal voltage for multiplier circuits, Defined as $\kappa q / 4kT$,
I_{on}	: determined by ON.

A transconductance amplifier may also be used as a multiplier. If the voltage-input is ΔV_{in} and the bias current $I_b = O_+$, then the amplifier performs a multiplication between a single quadrant current signal and a voltage difference:

$$I_{out} = O_+ \cdot \tanh\left(\frac{\kappa}{nV_T} \cdot \Delta V_{in}\right) \quad \text{where} \tag{4.3}$$

V_T is the thermal voltage. The linear range of the tanh-function increases proportional with the parameter n. In chapter 3.1 it was shown how we may increase the parameter n to achieve higher linear range. The multipliers in figure 4.1 are modified transamps with $n = 2$.

Measurements

Figure 4.2 shows measurements of an output neuron O_i. In figure 4.2 (a) we have disconnected all the neurons in hidden layer (O_j) and the output O_i was only depending on Θ_i. This voltage was modified by programming it with UV-light exposure. So the X-axis in figure 4.2 (a) is programmed values of Θ_i and Y-axis is obtained by measurements between each programming period. The curve shows a typical sigmoid function.

Figure 4.2 (b) is measurements of output neuron O_i, but here is the X-axis a voltage input to the input neuron O_k. As you can see, this sigmoid function has a large bump. It looks like O_i is going to be positive but something happens and it stays negative. The reason is that when O_i is reaching -10nA, the programming direction of the weights is beginning to switch sign. As described in chapter 3.2.4 the value on the memories may jump up to 60mV while the programming direction is switching sign. These large jumps in the weight values are causing the transient in the output of neuron O_i. Why the output of O_i is not becoming positive, must be due to the variation in weight values for different signs of the programming direction.

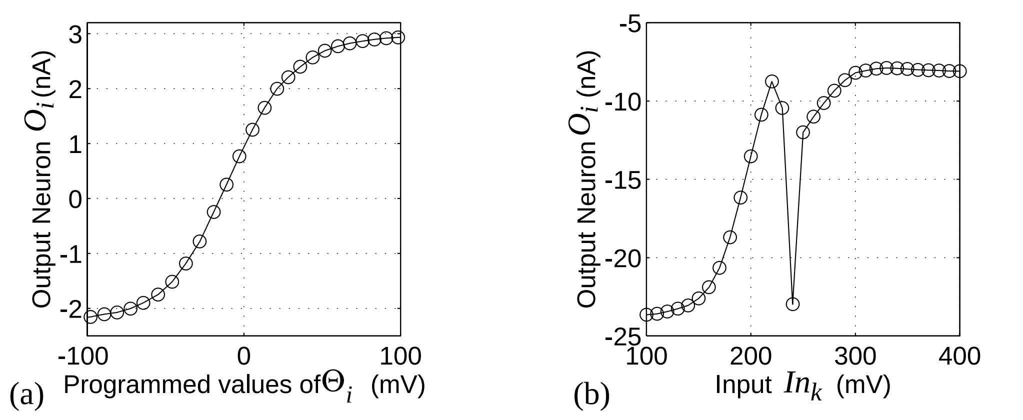


Figure 4.2 : ***Measurements on the feed -forward part.***
Shown in (a): output of neuron O_i as a function of programmed values of Θ_i when all the neuron O_j in the hidden layer are off ($Bias_j$ = Gnd). Shown in (b): output of neuron O_i as a function of the input In_k to the network. The bump in figure b) is caused by unstable weights.

In recall-mode the problem with unstable weights is easy to eliminate. You have to clamp the sign of the programming direction to a fixed value and not let it interfere with the back-propagation computation. But under training-mode it is not so trivial to compensate for this error since it is the back-propagation computation which determines the programming direction. A more carefully UV-memory layout to obtain a stable value on the memory may help to eliminate this error.

Since the offset error is critical only during the training-mode, a simpler storage with less errors could be used in this mode. Afterwards, the values on these storages could be stored in the UV-memory before enter the recall-mode. A simple storage could be a capacitor which have to be updated frequently. [Tarassenko] used the gate-capacitance of a transistor to hold analog values under the training of the network and the capacitor had to be refreshed every 10ms. The refreshing requires a clocking and such an arrangement may conflict with the continuous-time computation in the network.

4.2 Back-propagation computation

In chapter 2 the rule for back-propagation was described. The calculation of back-propagation can be divided into two categories: the computation of the errors and the computation of the weight updating.

4.2.1 Computing the errors in analog CMOS

There are two important errors in the back-propagation algorithm; δ_i and δ_j as shown in equations (2.4) and (2.5). Common for the calculations of these errors is the derivative

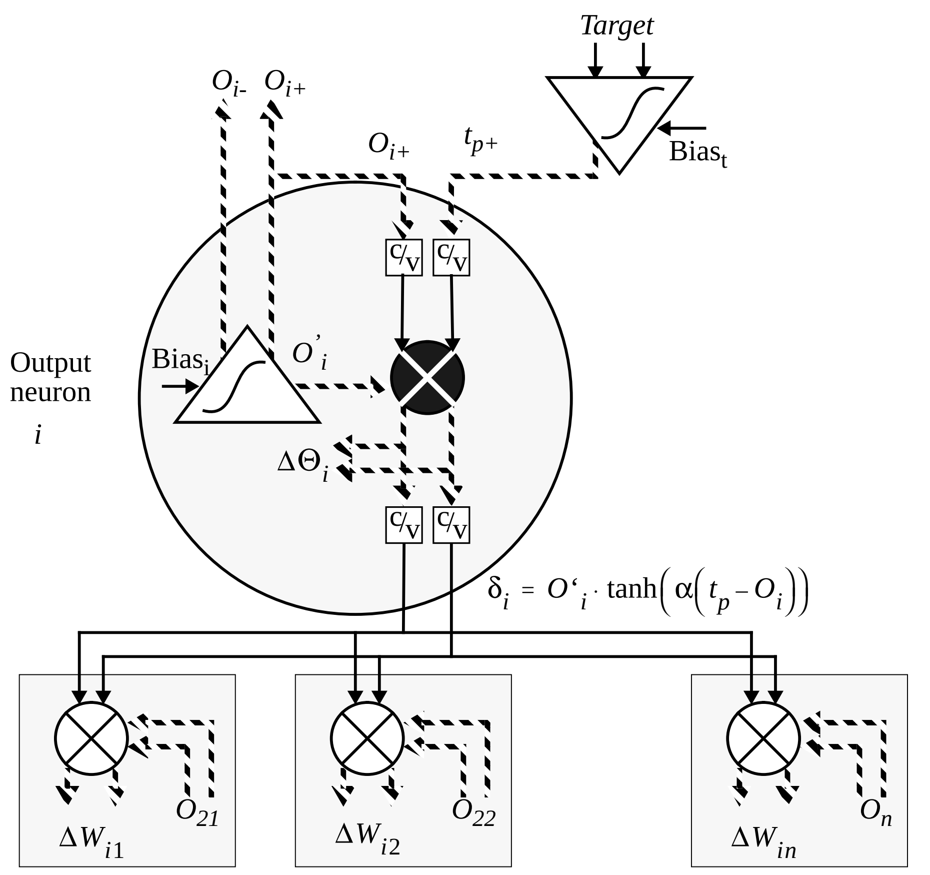


Figure 4.3 : ***Computing the error*** δ_i ***and the weight updates*** ΔW_{ij}***.*** ***The increment*** ΔW_{ij} ***gives the direction of the updating of*** W_{ij}***.*** ΔW_{ij} ***is the current input to the amplifier stage in figure 3.5. The constant*** α ***in the calculation of*** δ_i ***is defined as*** $\alpha = \kappa q / 2kT$***. Stippled lines are current signals and solid lines are voltages. The white multipliers are Gilbert multipliers with one differential voltage input and one differential current input. The black multiplier is a standard transamp with differential current output. The derivative*** O'_i ***controls the bias current of this transamp.***

of the activation function. The derivative may be achieved nicely by the bump circuit [Delbrück]. From this circuit the bump signal (the derivative of the tanh function) is a current (one quadrant) output. This signal may easily be multiplied with a voltage difference in a transconductance amplifier. See figure 4.3 and figure 4.5 where O'_i and O'_j are the bump signals.

Normal assertion for accuracy in back-propagation is at least 16-bit precision. However, Murray [Murray] showed that the learning was enhanced by introducing noise (20%) on the weight calculation and the activation function in an analog neural network. Therefore the accuracy and the linear range of the multipliers involved in calculation of the errors are not critical for the final result. The special weight updating scheme explained in chapter 4.2.2 will also strengthen this conclusion.

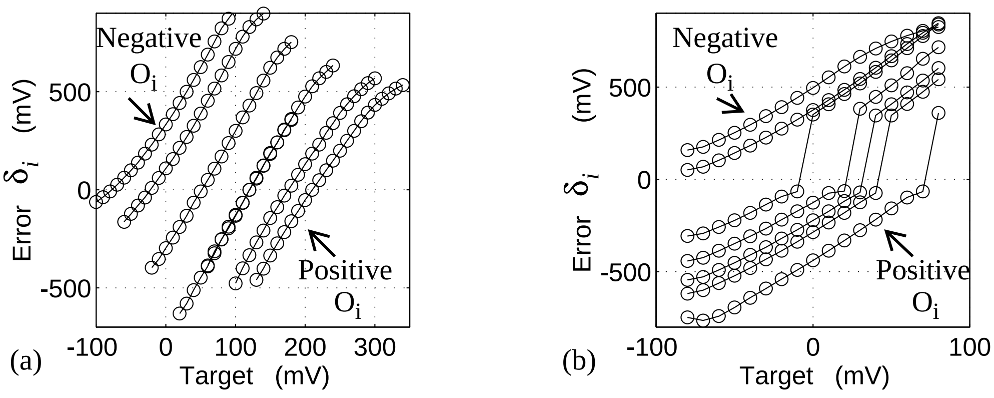


Figure 4.4 : ***Measured response of the error δ_i.***
Target is swept for different values of O_i. In (a) the sign of the programming direction of the weights is clamped to a fixed value and $Bias_i = Bias_t$. In (b) the programming direction is determined by the back-propagation computation, hence the jump in the error δ_i which is caused by unstable weights.

Figure 4.3 illustrates how computing of the error δ_i is done. Before the signal δ_i leaves the neuron module it is converted to a voltage. Thus it is easy to distribute the signal to many nodes. Also shown in figure 4.3 is the calculations of the weight change $\Delta W_{ij} = \delta_i \cdot O_j$ and the threshold change $\Delta\Theta_i = \delta_i \cdot$ 'O'. The weight change is obtained in the weight-module by a Gilbert multiplier with one differential current input and one differential voltage input. The output of this multiplication is input to an UV-memory amplifier stage shown in figure 3.5. The neuron output 'O' included in the threshold change computation describes a neuron which always is 'on' (high). It is, however, unnecessary to include a multiplication with this neuron since it is constant on and therefore the calculation of $\Delta\Theta_i$ is identical to δ_i. The current signal $\Delta\Theta_i$ is input an UV-memory amplifier stage.

Measured response of δ_i is shown in figure 4.4 for different values of $_i$. Figure 4.4 (a) shows results when the sign of every weight programming directions are clamp to a fixed value and when $Bias_i = Bias_t$. The results in figure 4.4 (b) are obtained when the back-propagation is computing the sign of the programming direction. When δ_i is switching sign it jumps from 350mV to -50mV because of the change in the values of the weights due to non-symmetrical capacitances in the UV-structure (described in chapter 3.2.4). From measurements, we found out that the jump in δ_i when switching sign may be adjusted by the ratio $\mathrm{Bias}_t/\mathrm{Bias}_i$. Using this, it was possible to minimize the error shown in figure (b).

A layout error (two signals were exchanged in the layout) demanded a different use of the error signal δ_i. To get positive increments of the weights, δ_i had to be negative (if O_j was positive). δ_i had to have opposite sign of the wanted weight change direction. This could be obtained by setting *target* with opposite sign of the direction wanted. As a consequence of this approach, δ_i will operate outside the range where it is unstable. A further discussion can be read in the following chapter 4.2.2.

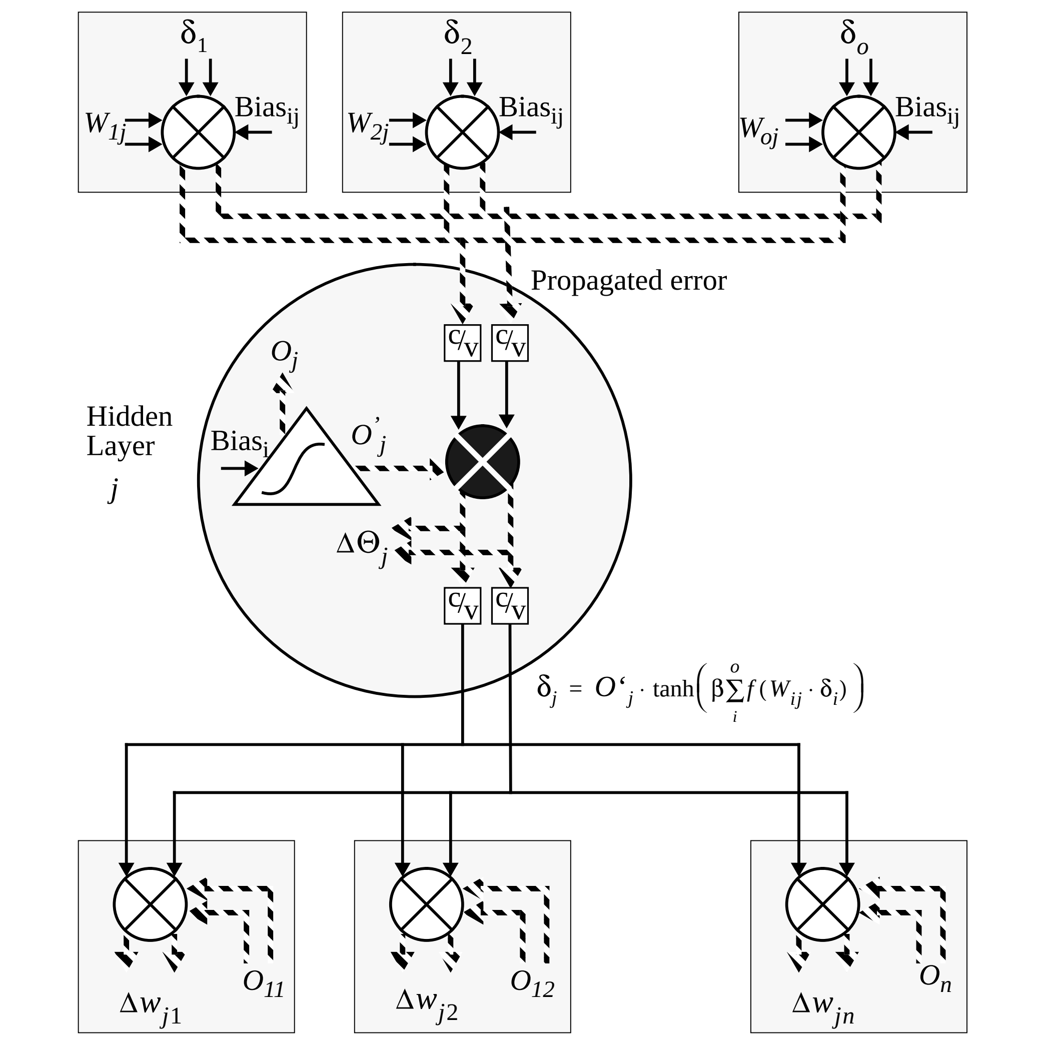


Figure 4.5 : ***Computing the error*** δ_j ***and the weight updates*** Δw_{jk}
This error is depending on each error δ_i ***in the output layer. The white multiplier circuits shown in the figure are Gilbert multipliers and the black one is a standard transamp used as a multiplier. The result from the multiplication*** $W_{ij} \cdot \delta_i$ ***is:*** $f(W_{ij} \cdot \delta_i) = I_{b_{ij}} \tanh(\alpha W_{ij}) \tanh(\alpha\delta_i)$ ***where*** α ***is*** $\kappa q / kT$***.***
Stippled lines are current signals and solid lines are voltages.

Figure 4.5 demonstrate how the hidden layer error δ_j is computed. It is very similar to the computation of δ_i. However, the computation of δ_j includes the propagated error which is a summed value of the weighted values $W_{ij} \cdot \delta_i$ for $i = 1, 2,..., o$. The multipliers used in these calculations are Gilbert multipliers with two differential voltage inputs. The output of these multipliers are current signals and are summed together by just wiring them together.

Figure 4.6 (a) shows measurements of the error δ_j as a function of the output layer error δ_1 for different values of the output layer error δ_2. As the observant reader may have

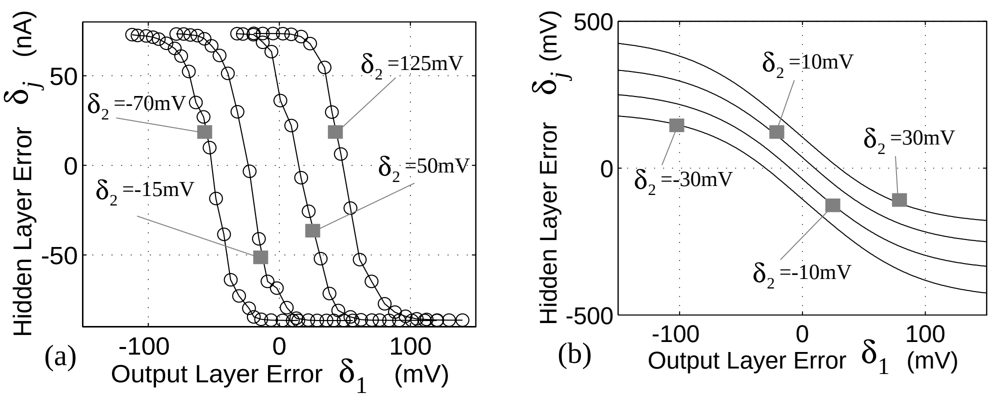


Figure 4.6 : ***Measured and theoretical result of the error signal*** δ_j**.**
The measured result in figure a) is obtained by putting the real error δ_j ***through a transamp to get a bidirectional current signal. The X-axis in figure a) is obtained by sweeping the target*** t_1 ***for different values of*** t_2 ***and measuring*** δ_j**,** δ_1 ***and*** δ_2 ***at the same time. Figure b) shows simulations in*** *hspice* ***of the hidden layer error. The result illustrates how a theoretical*** δ_j ***should look like.***

already seen, the signal type of δ_j in figure 4.5 and figure 4.6 (a) is not identical. In figure 4.6 (a) the δ_j is a current signal. This is due to few pads on the chip and by converting δ_j to a bidirectional signal we are able to only use one output pad for this signal. The measured result in figure 4.6 (a) is an output of a transamp with the original error δ_j as input. The result illustrates how the summed propagation error and then again δ_j are depending on the output layer errors δ_i. The measured results correspond to the theoretical results in figure 4.6 (b) which shows simulations of the network in *hspice*. The simulations illustrate the original δ_j and how it variates with the output layer errors δ_i.

4.2.2 Updating the weights in analog CMOS.

Equations (2.2) and (2.3) demonstrates how the weights are updated. It is important to understand how this computation is implemented in analog CMOS because our method departs from the theory and implementations in software.
Equations (2.2) and (2.3) computes the size of the incremental change on a weight. In a computer program this will execute perfectly. A pattern is being presented at the inputs and the targets. The weights are updated, not synchronously but one at a time, before next pattern will be presented. This method works in the discrete-time domain.

In our design the weights will be updated synchronously (parallel computing) and this is done in continuous-time. Thus equations (2.2) and (2.3) is difficult to implement directly into analog CMOS because the equations are designed for discrete-time computations.
The way to go is to let the time each input/target pattern pair is present to the network, decide the size of the increment. And let equations (2.2) and (2.3) only decide the sign. (The intensity of the UV-light illumination is also a factor which determines the size of the increments.) Figure 4.7 displays the weight updating scheme for our network.

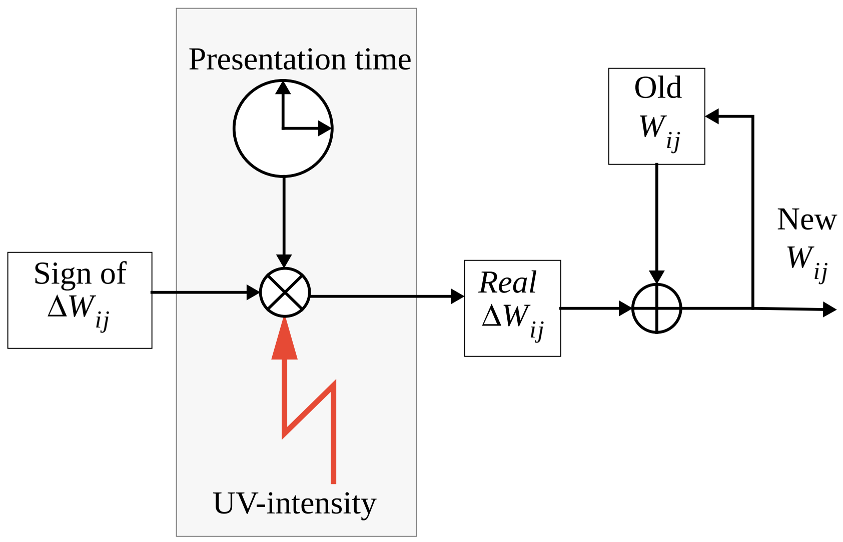


Figure 4.7 : ***The weight updating scheme.***
A schematic diagram that illustrate how the updating of the weight is done. The weight W_{ij} ***is updated with an increment*** ΔW_{ij}***. The size of the increment is determined by the time a pattern is presented and the intensity of the UV-light exposure.***

This approach has some good facilities:

- All the weights are being updated all at the same time.
- We control the size of the increment ΔW outside the chip (this may be compared to controlling the learning rate η.)

But this approach has also some unwanted effects that we have to consider:

- It is only the UV-light that determines the learning is on or off.
- All the increments ΔW for one pattern have nearly the same size.

Figure 4.8 illustrate how the sign of the computed increment ΔW_{ij} determines the sign of the increase or decrease in W_{ij}. Unfortunately the sign of ΔW_{ij} is inverse of the sign of the increase in W_{ij}. During drawing of the chip layout, two signals in a differential signal have been switched. One way out of this layout error problem is to set an inverse value on *target*. When *target* is positive, we want a negative output and when *target* is negative, we want a positive output. The method will work but the error δ_i will not behave as expected. The computation of δ_i will not approach zero when O_i approach the wanted output. A continuing of the weight updating when O_i is equal to wanted output may lead to that the network can forget already learned patterns. The convergence time will be increased because unnecessary weight updating for learned patterns will slow down the learning of unlearned patterns.

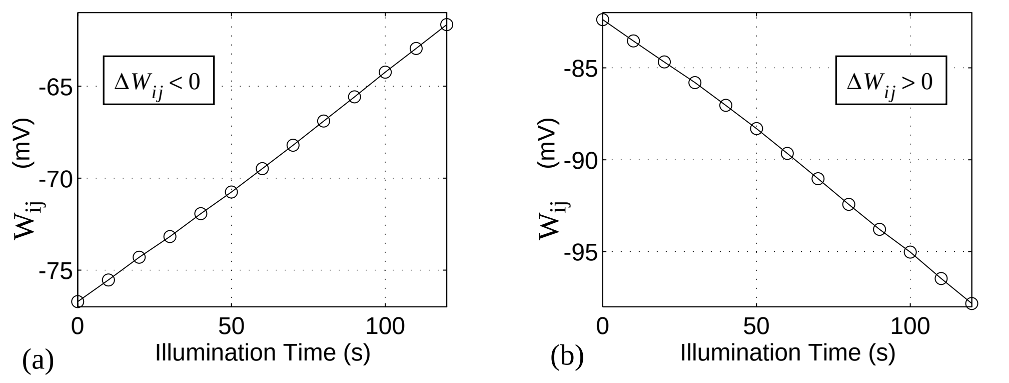


Figure 4.8 : ***Programmed increments as a function of*** ΔW_{ij}.
The measured results show how computed values of ΔW_{ij} ***is increasing and decreasing the weight*** W_{ij} ***when the UV-light is on. As you may see the values of*** W_{ij} ***increases when*** ΔW_{ij} ***is negative and vice versa. Due to equations (2.2) and (2.3) this is wrong behavior and is caused by a layout error under realization of the chip. This may be compensated by switching the sign of the target. When target is negative we want a positive output and vice versa.***

Since equations (2.2) and (2.3) only will determine the sign and one (two with UV-light) global parameter decides the size of all the increments ΔW, identical sized increments may lead to cancellations of previous increments. This problem can be solved with a different pattern presentation scheme (also called training scheme). Usually one pattern pair is presented one at a time with the same time between each pair and in a sequential repeating order.

There exist two alternatives to eliminate the cancellations:

> If we pick a random length of the presentation time for each pattern, each weight increment will differ from last increment and we get no cancellation effects. With this tactic it will probably be important that the total presentation time for each pattern at the end of training are equal for all patterns.

or:

> If we let the presentation time for each cycle of the training set differ, we get a similar effect as the alternative above. For example we may start with a long presentation time for each cycle and reduce it for each repetition until end of training. (also proposed in chapter 2.3). The weights will for a such training scheme first be adjusted roughly and during further training the weights will approach more and more exact values.

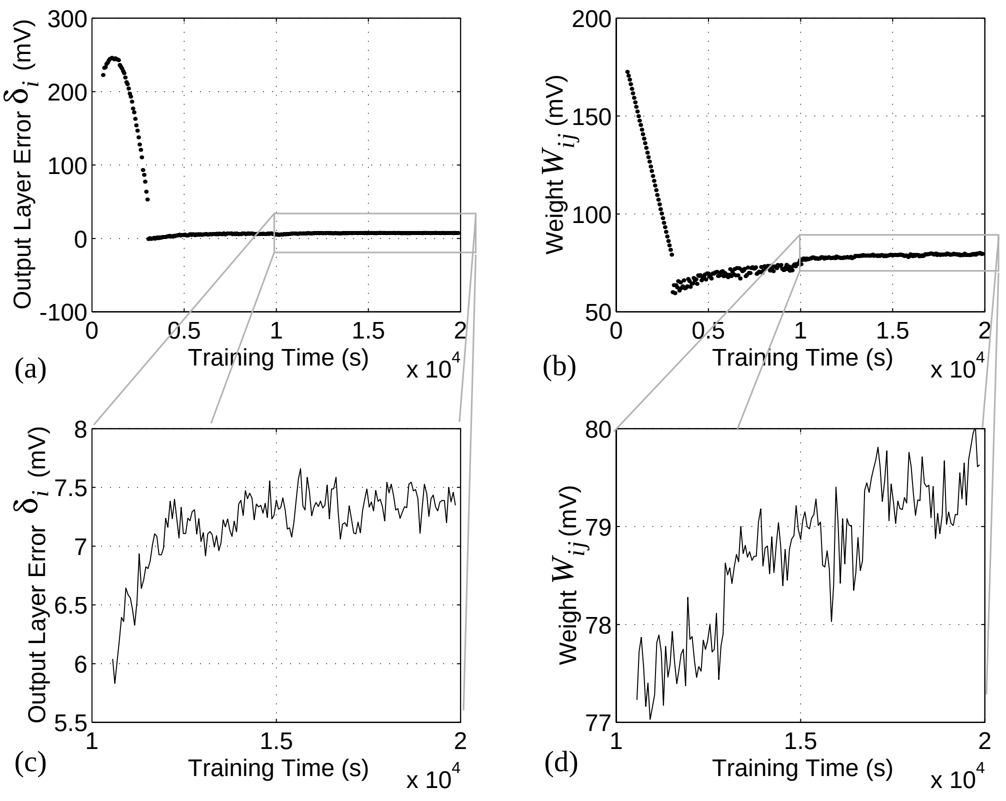


Figure 4.9 : ***Measurements on the stability problem.***
Figure (a) shows the error δ_i while it is decreasing toward zero and stays there. Figure (c) is a zoomed window of δ_i in (a). Figure (b) shows programming of a weight w_{ij} which is a function of δ_i. Observe how stable w_{ij} is when δ_i is almost zero, only 3.9% drift of normal programming speed. Figures (c) and (d) are zoomed windows of figures (a) and (b) respectively. Measurements is taken every minute.The filter A10 was used during programming and the distance between the UV-light source and the chip was 15cm.

Neither of these two methods were tested when actually training the network. It appeared that the weight changes for pattern to pattern differed under training, due to noise in small weight increments. For larger and more complex pattern sets applying these two presentation schemes may be necessary.

Sequential repeating order of the training patterns may also cause oscillations of the weight values. You may think of it as a local minimum. By picking out the order randomly we eliminate this problem. This training scheme has also been reported as a way to shorten the training time [Weiss].

We have just talked about the weight updates as positive or negative increments. Usually, when the network converge, the errors δ become small (values near zero). The decrease in δ should have some effect on the size of the increments ΔW or else the network may have difficulties with converging. During the design of the analog UV-memory it was an important task to consider the state where the sign input $\delta \cdot O$ is almost zero. Of course it is significant to stop the updating of the weights if the errors are near zero. But to include extra logic to control this, was out of the question. One of our main issues was to minimize the area of the weights. Instead of designing extra logic, we exploited the behavior of the implemented UV-memory. When $\delta \cdot O \approx 0$ the control node and the capacitive node have almost identical voltage (approximately 2.5V). Thus the voltage difference between control node and floating gate node becomes smaller. In chapter 3.2 it was described how a change in the floating gate voltage is depending on the voltage difference between the control node and the floating gate node. With a smaller voltage difference the size of the increments and decrements will be considerably smaller. And this, in addition to the feedback error computation, will probably be enough to get the network to converge.

Figure 4.9 shows measurements on the network. One pattern is presented at the inputs and the targets, and the UV-light source is turned on. Two nodes in the network are examined, the output error δ_i and the weight W_{ij}. W_{ij} is updated as a function of the sign from the multiplication $\delta_i \cdot O_j$. What will happen when δ_i approaches zero? In chapter 4.2.1 it was discussed the situation where δ_i was approaching zero because O_i was approaching t_i. In figure 4.9 δ_i approaches zero because the derivative O'_i approaches zero. The shape of δ_i follows the shape of a typical bump signal (O'_i). When δ_i is close to 0V, the updating of W_{ij} are approximately stopped. W_{ij} stabilizes as we had hoped. In normal programming mode the speed on the programming in figure 4.9 is 1.8mV/min, while when $\delta_i \approx 0$ the programming speed is only 0.07mV/min. The weight W_{ij} only drift 3.9% of normal programming speed when δ_i is close to 0V. This is a very interesting result. The stabilizing of the weights when the derivative O'_i is approximately zero will help the network to hold its weight values inside an interesting range (-200 > W_{ij} < 200mV). Because of the fixed output bounds of a neuron output it is unnecessary to change those weights that are trying to increase this neuron output if it already have reach its maximum value. A drawback is that if the network starts with many weight values outside the interesting range these values may never enter the interesting range. Therefore a satisfactory initiation of the network is especially important.

4.3 The threshold Θ

Updating of the threshold do not differ much from weight updating:

$$\Delta\Theta = \eta \cdot \delta \cdot `O` \tag{4.4}$$

The only difference is that the neuron $`O`$ is always on (logical high). Therefore the multiplication with the neuron $`O`$ is not necessary. The sign of threshold updating are then only determined by the error δ. In figure 4.3 and figure 4.5 $\Delta\Theta$ is shown as a copy of the current version of δ before δ is converted to voltage. $\Delta\Theta$ is fed into the UV-memory amplifier stage shown in figure 3.5.

In the feed-forward computation, however, we wish to scale or sometimes disconnect the contribution of the threshold and therefore an extra multiplier is included in our network in order to obtain these opportunities. In figure 4.1 the threshold Θ is scaled (from voltage to current) before it is summed together with the rest of the net-input to a neuron. The scaling is determined by the bias signal *ON* which is identical to the neuron ‘*O*‘. Hence it is possible to regulate the magnitude of Θ to match it to the net-input signal to a neuron. The network implementation proposed by [Soelberg] did not include the threshold scaling facility. The absence of this facility was the major factor that his network did not work.

[Soelberg] stated that weight updating should be faster than updating of thresholds. To enlarge the time constant on the threshold we had to implement an UV-memory structure that had a higher time constant. The realization of such a structure was explained in chapter 3.2. The UV-structure model B described in this chapter has been applied for the storage and updating of the threshold Θ. This structure has about three times larger time constant than the structure the weights are using (model A).

4.4 Summary

It has been showed how the theoretical equations described in chapter 2 may be implemented in analog CMOS. Measured results of the computation of these equations are also included. Calculations in analog CMOS have often constrained range of operation. An example is the multipliers used. The linear operation range to these multiplies are limited within 140mV. In the back-propagation calculation the range of the multipliers used is of second order importance, mainly because of our special weight updating scheme. Normally these computations are deciding the size of each weight update increment. But in our system the size is determined by how long a pattern is present, the UV-light intensity and if the back-propagation error is close to zero or not. The weight increment computation only determines the sign of the increment.

As discussed before, noise is a major factor in analog system. Noise includes transistor mismatch, temperature offsets, interference from the setup as electromagnetic fields, unwanted offsets caused by the UV-light and so on. Many has concluded that analog CMOS is too imprecise to ever handle tasks such as neural networks. But [Murray] showed how noise enhanced the computations in these tasks when applying analog CMOS. The discovering of weight jumps up to 15% of its interesting operation range in our UV-memories was not desirable. In addition, a layout error almost spoiled everything, we asked ourselves: Is our network going to learn?

5 A 4-3-2 Neural Network

A feed-forward neural network with 4 input, 3 hidden, 2 output neurons and back-propagation learning is implemented on a chip processed at MOSIS. In this chapter the network implementation will be thoroughly discussed. First we consider how the network behaves and how it may be trained. Secondly we demonstrate on-chip learning.

5.1 Why a 4-3-2 sized network?

When designing the neural network we wanted a larger network than the typical XOR-network. [Soelberg] designed such a network (a 2-1-1 network) but he also included a lot of test structures, and that was not of any interest for us, mainly because of the good result he achieved on the test structures.

The size of the network was chosen from the criteria:

- the size of the chip
- the number of inputs to be synchronously controlled
- the number of outputs to be measured
- which type of mapping function we want to learn.

Our network should have analog inputs and analog outputs although we think about them as digital inputs and outputs. Functions we want to map are in first hand boolean functions. Later, if we succeeds, more complicated analog function may be mapped in improved implementations.

Usually neural network architectures are designed after finding the training set. When designing a network that is larger than absolute the minimum size, you can tolerate that not every neuron have to operate as expected, exploiting the high level of fault tolerance in analog systems and neural networks. For example if a neuron is locked at a fixed value, another neuron may “take over” its job. Examples on this will be shown later.

The 4-3-2 network has 6 analog inputs. Each pattern has to be represented as a 6 bit word: 4 inputs and 2 targets. To take advantage of the whole network at once, we need 6 voltage sources to be simultaneously programmed. To be able to measure the result the

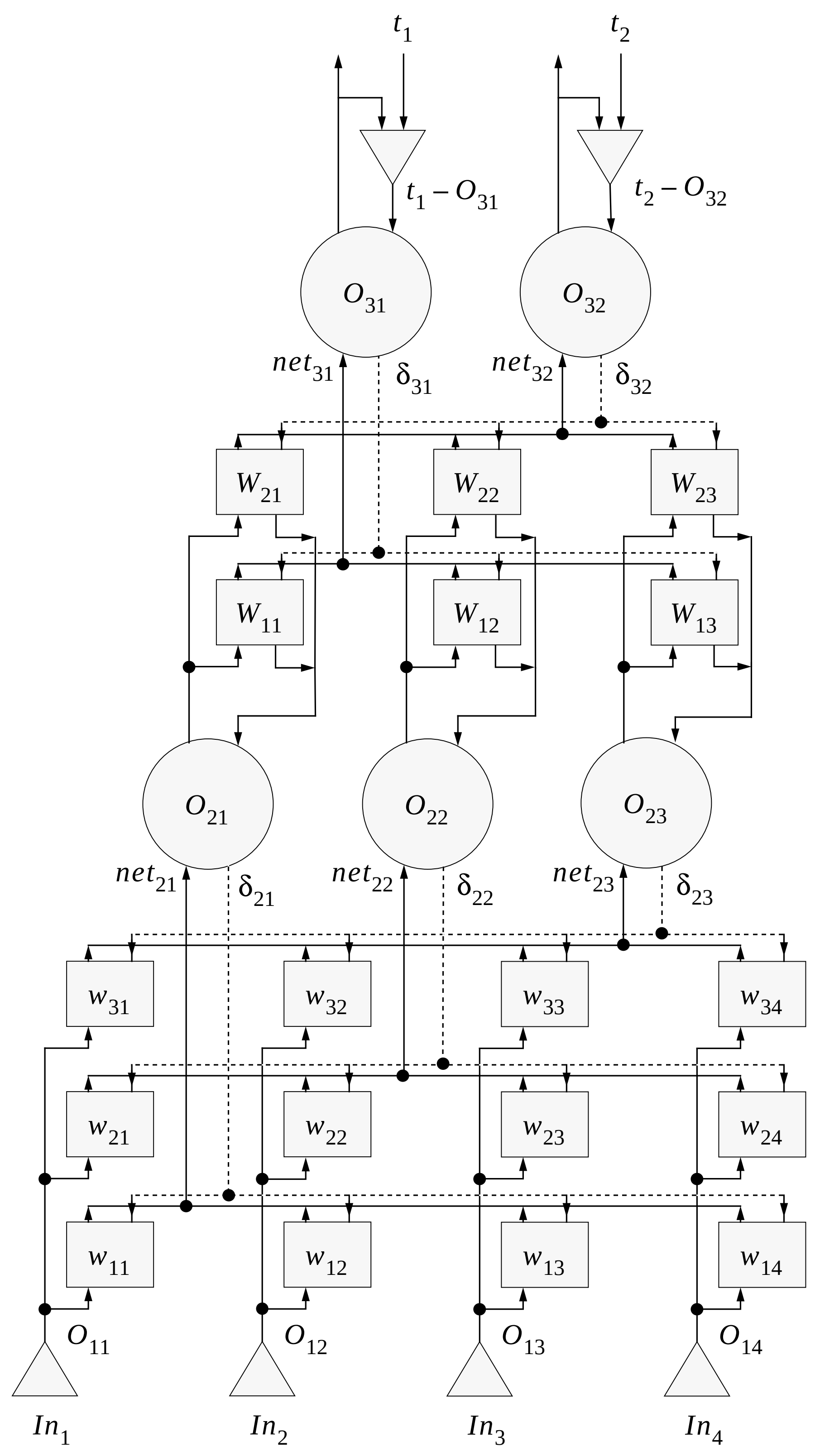


Figure 5.1 : ***The 4-3-2 feed-forward network***

first idea was to measure the two output layer errors in addition to the two output neurons. But after testing the chip we found out that it was enough to measure the two output neurons during training, mainly due to the special target presentation applied which altered the output layer errors in a different manner than assumed at the beginning.

The 4-3-2 network contains of 22 weights and thresholds but only 5 neurons as can be seen in figure 5.1. If we had added one hidden neuron, seven more weights had to be included. This shows that it is necessary to carefully choose the number of hidden neurons. After having picked out the number of input and output neurons, the number of hidden neurons was determined in according to the space left on the chip.

5.2 Dynamical behavior and time constants

Our analog system has a lot of parameters which can be adjusted. When starting measuring on the chip, our first problem was to match signals. With 5 bias voltages, 3 other reference voltages and 6 input voltages it was in the beginning difficult to get any reasonable relevant measurements. In addition, initiation of 22 weight values also led to difficulties. How do we get the network to operate as we wanted? First we had to study the matching of signals inside the network.

The 6 input voltages were fed directly into 6 transamps and a reference voltage set to 2.5V worked as the negative differential voltage component for all the transamps. Therefore these 6 voltages should be adjusted around 2.5V ± 150 mV.

The output of a neuron is a tangents hyperbolic function which are normally constrained between the two currents $\pm I_b$. An increase in the output is avoided if the input is expanding outside the operation range due to the fixed boundary output range.

Also included in the neuron is the derivative of the tanh function (bump signal). This signal is approximately zero when the tanh function is at its operation range borders. The bump signal operates in our network as a bias voltage for a transamp which computes the error δ (see figure 4.3). When the bump signal approaches zero, the output of the transamp also approaches zero. An example on this is illustrated in figure 4.9 (a) which shows measured results of δ when the derivative O' approaches zero. The shape of δ follows a typical bump signal shape. In these measurements the bump signal operates as desired, it stops the programming of the weights. But as experienced with other signals, offset errors in differential pairs could give a positive or negative offset when their bias currents are almost zero. With this offset error the programming of the weights would not stop as desired. As a consequence of this we tried to increase the bias voltages to increase the operation range of the bump signal. It would be a better solution to amplify the bump signal in the design by increasing the width of the transistors that calculate the bump signal.

The operation range for the input to a neuron is determined by summed weights. How large the range is, will be a function of the number of weights. If the result of the summing of weights exceeds the operation range of the neuron, in worst case by 2-3 times, the convergence time will grow. The extra time applied is due to the time the network may use to adjust net inputs which are largely outside the operation range of the neurons.

Therefore to match the operation range of a neuron a weighting between the number of summed weights connected to it and the weights linear range should be carried out.

As described in chapter 2.3 the theory says that the weights should start with random values. The UV-memory has an operational range of about 4V but the multiplier with the UV-memory as one of the inputs, has only a dynamical operation range of about 250mV (140mV linear range). And with no extra logic to set a fixed value on the UV-memory, this mismatch between operation range was probably the most difficult problem to overcome.

[Pineda] showed how relationships between time constants in a feed-forward neural network have to be satisfied before the network can learn. He stated that the time constant for the forward propagation has to be smaller than the back-propagation which again has to be smaller than the time constant of the weight adaption. Measurements have shown that these criteria are satisfied in our network.

5.3 Training the network in practice

5.3.1 Initiation of the network

As stated before, the weights should start with random values. But since the weight values only have an interesting range of about 150-250mV and a dynamical operation range of 1-4.9 V they have to be initiated to ensure that they are inside the interesting range.

Pull-down transistors have been included at each control node in the UV-memory. The intention of these transistors was to apply them when we wanted to lock the value on the memories with the result that the memories not were disturbed by the back-propagation signals. In addition it was the purpose to use the pull-down transistors to initiate the UV-memory by programming all the memories towards Gnd. The initiation gave poor results. The memories did not enter the interesting range. And when training of the network was activated, the programming of the weights showed a “stairs-step” effect. A “stairs-step” effect means that positive increments are not identical sized with negative increments for identical programming parameters.

The effect is also reported by [Abusland94]. His solution to this initiation problem was to put the network through several up and down programming steps. In our network we held all the inputs constant and altered the two target inputs up and down several times which was the key to successful training. All the weight values entered the interesting range and the network had been initilized with random values. However, sometimes one or two hidden layer errors δ_j were stuck at values near zero. For these errors the derivative O' was close to zero. How this affected the training of the network, is discussed in chapter 5.4.

5.3.2 Pattern presentation technique

In general, patterns are presented sequentially during training in a repeating order. As discussed in chapter 4.2.2 this may lead the network into a local minimum. A better solution will be to pick out the order randomly. This technique has been applied in the training of our network.

The special weight updating scheme uses the time each pattern is on (pattern presentation interval) and the UV-light intensity to determine the size of the increments. Adjusting the size of the presentation intervals may speed up the convergence time. You can compare it with regulation of the learning rate η . This rate is often used as an instrument to get faster out of local minima. A normal regulation technique of the learning rate is to start with a high value and then decrease the value. This technique could also be applied on the length of the presentation intervals. But in practice it was enough to only pick out the order randomly, because the weight increments did not canceled out due to the noise present in analog systems in addition to the random presentation order.

The training of the network follows the algorithm:

```
UV-light source ON
For (the number of repetition)
    Begin
        For (the number of patterns)
            Begin
                Put on a pattern in random order
                Measure
                Let it stay on in n seconds
                Measure
            End
        If (error <= a chosen limit) then training finished
        (regulate n)
    End
UV-light source OFF
```

The statement <***For*** *(the number of repetition)*> may be replaced with <***While*** *(not training finished)*> if we have knowledge in front of the training that *error* will reach ***limit***. If the error don't approach ***limit***, the network will not converge and the training will never end.

5.4 On-chip learning

5.4.1 Learning four patterns

We start to show on-chip learning for a simple boolean function to demonstrate that the back-propagation algorithm is working in analog CMOS. Four patterns is picked out from the boolean function defined as $O_{31} = In_3In_4$. The patterns are shown in table 5.1.

The selected function is linear separable. A linear separable problem must have the ability to separate its boolean outputs with a line or a plane in its input space. The correctly separation of the outputs for the function $O_{31} = In_3In_4$ is shown in figure 5.2. All linear separable problems can be mapped by a feed-forward network with only one layer (perceptron).

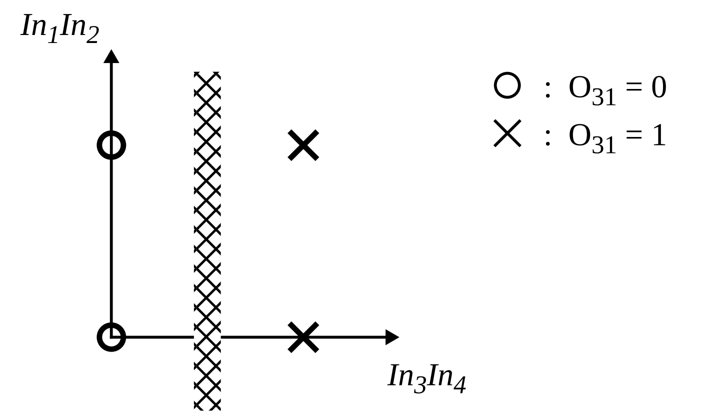


Figure 5.2 : ***A linear separable function.***
The figure demonstrates the separation of the boolean function In_3In_4 ***into a group with outputs logical high and a group with outputs logical low.***

Table 5.2 shows parameters which are set off-chip. The three parameters UV-filter, UV-light source distance, and pattern presentation interval are determining the weight and threshold time constant. For this training session we used a relative large time constant. The network would still have converge if a smaller time constant had been used for training of this simple function.

Figure 5.3 demonstrates the training period and the recall period. Figure 5.3 (a) shows measurements during the training. First the chip is initiated, secondly the presentation algorithm in chapter 5.3.2 is used. You can see how the two patterns with targets equal to "0" are first positive and move towards negative values as wanted. The training duration is 3000 seconds and with a pattern interval of 15 seconds the four patterns are repeated 50 times.

Figure 5.3 (b) shows retrieval of the four patterns when the UV-light is turned off and the target is set to a fixed value. As you can observed the network has successfully learned the selected function.

Figure 5.3 (c) shows the calculated mean square error during the training. The error is calculated from the equation:

$$\text{MSE} = \frac{1}{2p} \sum_{p=1}^{n} (t_1 - O_{31})^2 \tag{5.1}$$

Where p is the number of patterns, t_1 is the correct output and O_{31} is the actual output. Since O_{31} is a current in the *nano*ampere region, the value of t_1 has to be converted. t_1 is chosen from where the output O_{31} is stable. From figure 5.3 (a) the value for t_1=0 is -24nA and 22nA for t_1=1.

An interesting finding in this training session was found. The network has an ability to handle faults relating to hidden neurons which is stuck at one of their operation range border. Before and after training every hidden neurons and hidden layer errors were measured. The measured results showed that two of the hidden neurons and their hidden layer errors had not changed at all. This is interesting because it tells us that it is not necessary for all hidden neurons to work if we want to train the network. The reason these neurons were stuck, is not satisfying initiation of the weights connected to those neurons.

Table 5.1 : ***Four patterns from the boolean function*** In_3In_4 **.**

Number	Input				Target
	In_1	In_2	In_3	In_4	t_1
1	0	0	0	0	0
2	0	0	1	1	1
3	1	1	0	0	0
4	1	1	1	1	1

Table 5.2 : ***Training parameters.***

UV-filter	UV-light source distance	Pattern intervals	Weight time constant	Threshold time constant
A10	12cm	15s	0.86mV/interval	0.25mV/interval

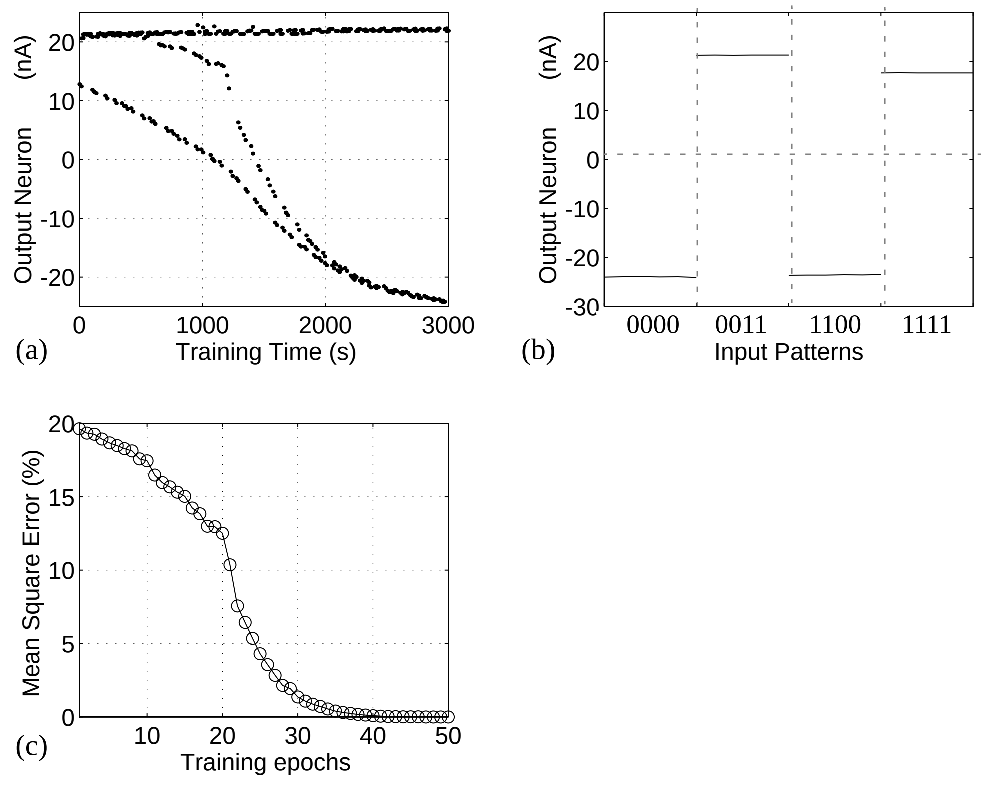


Figure 5.3 : ***Measured results of learning and recall of the 4 patterns: 0000, 0011, 1100 and 1111.***
Figure a) shows measurements during training, figure b) illustrates the measured result of the recall of the training patterns, and figure c) shows calculated mean square error of the measurements in figure a).

5.4.2 Learning a four pattern XOR function

A classical problem to map is the XOR-function. This function requires hidden neurons if it is mapped by a feed-forward network. More complex functions are involving the XOR-function as a subproblem.

We have chosen to train the network with four patterns from the XOR-function $In_1In_2 \oplus In_3In_4$ as shown in table 5.3. This function is not linear separable. As illustrated in figure 5.4, more than one line has to be applied to divide the function into target groups (“0” and “1”). Therefore this function has to be solved with a network including hidden neurons.

First we initiated the network so the weights were inside the interesting range. And then we started the training. For this training session, the filter B24 was preferred.

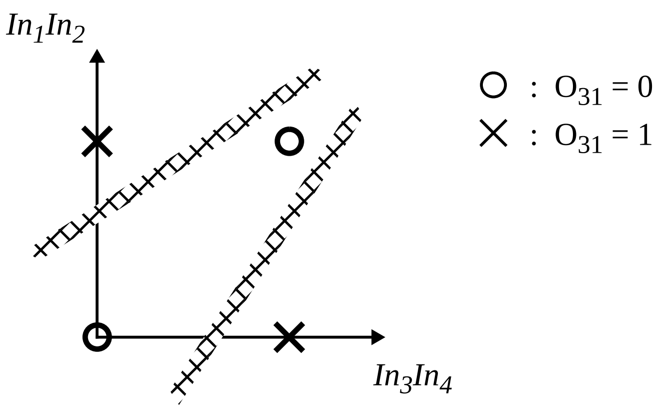


Figure 5.4 : ***The XOR-function is not linear separable. It is not possible to separate the function into two groups with only one line (or one plane).***

Pattern intervals was set to 10 seconds and the weight time constant was measured to be 2.25mV/interval as described in table 5.4. The number of repetitions was scheduled to 125 and then the total training time became 5000 seconds.

The results are shown in figure 5.5. In figure (a) it is shown the measurements of the output during training. After 30-40 repetitions the network output approaches close to the correct output. As you also may observe, one of the patterns with negative target (‘*’ points) first seems to change sign. However, the continued training shows that the output for this pattern is changing more and more against the wanted target.

After training the UV-light is turned off and the training patterns is tested on the network as shown in figure 5.5 (b). For this training session, retrieval of the patterns had to include the targets. The main reason was that the network needed the back-propagation signals identical to those calculated under training to obtain the same result. The network could not tolerate that weight values were not changed due to change of programming direction (described in figure 3.7).

Figure 5.5 (c) shows the mean square error calculated from the equation (5.1). The target t_1 was set to -32nA for t_1=0 and 20nA for t_1=1. After 30-40 repetitions the error decreases rapidly and the reduce in the error continues slowly, as expected, until the training is finished.

Table 5.3 : ***Four patterns from the boolean function*** $In_1 In_2 \oplus In_3 In_4$.

Number	Input				Target
	In_1	In_2	In_3	In_4	t_1
1	0	0	0	0	0
2	0	0	1	1	1
3	1	1	0	0	1
4	1	1	1	1	0

Table 5.4 : ***Training parameters for the XOR problem.***

UV-filter	UV-light source distance	Pattern intervals	Weight time constant	Threshold time constant
B24	12cm	10s	2.25mV/interval	0.62mV/interval

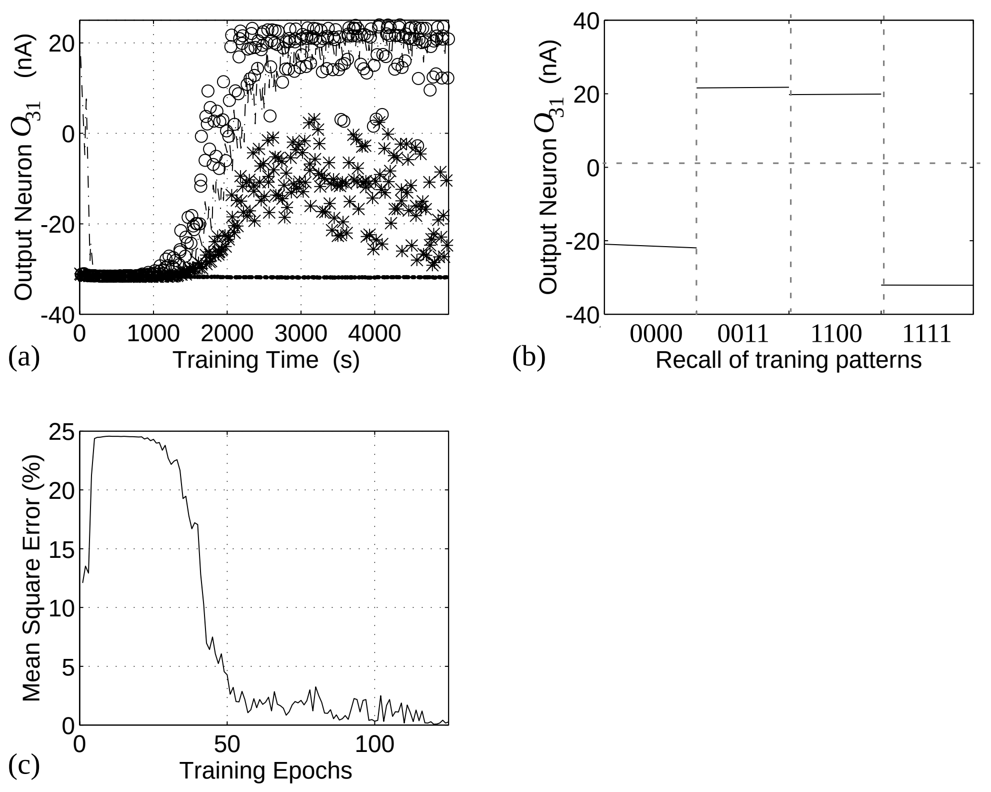


Figure 5.5 : ***Measured results of learning and recall of the XOR-function. Figure a) shows measurements during training. '*' is input 0000, 'o' is input 0011, '-' is input 1100, and '.' is input 1111. Figure b) illustrates the measured result of the recall of the training patterns, and figure c) shows the calculated mean square error of the measurements in figure a)***

5.4.3 Learning an eight pattern function

To show on-chip learning for a larger set of training pairs an eight pattern function was selected. The boolean function for these patterns is defined as $(in_1in_2)(in_3+in_4)$. The patterns are shown in table 5.5. The eight other patterns included in this function will be used as test patterns to demonstrate the networks ability to generalize. These test patterns are shown in table 5.7. The chosen function is linearly separable, as illustrated in figure 5.6. This time both of the outputs are trained with the same function. Thus it is possible to compare the output behavior for these two outputs.

The weight time constant was chosen to be 1.13mV/interval as described in table 5.6. First the number of repetitions was set to 200 and with a pattern interval of 5 seconds the total training time became 8000 seconds. After these repetitions retrieval of the training patterns was measured for the two outputs. This is shown in figure 5.7 (b) (O_{31}) and in figure 5.8 (b) (O_{32}). As you can see, O_{31} has not learned the function yet, but O_{32} has already learned it. The retrieval of the training patterns is done with the targets locked at a fixed value and the UV-light source turned off.

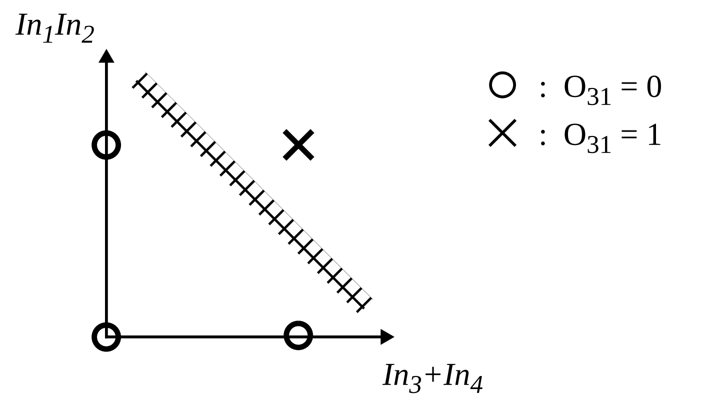


Figure 5.6 : ***The eight patterns function is linear separable***

Since O_{31} had not learned all the eight patterns, it would be interesting to see if O_{31} could learn all the patterns while O_{32} could still store all learned outputs. Therefore 100 additional repetitions (4000 seconds) were executed. The total training time became 12000 seconds. As you can observe in figure 5.7 (c), O_{31} has at last learned all eight patterns. But O_{32} in figure 5.8 (c) has 'forgotten' the previously learned pattern 1100. It can sincerely be a problem if the outputs are not able to finished the learning of all patterns at the same time. Since we can not stop the weight updating of some weights and continue updating of others, such problems may occur. One of the main reasons outputs have not synchronously learned all patterns, can be that the random initiation of the weights connected to each output neuron differ too much. Thus some of the output neurons need longer training time to learn up their weights to give the neurons correct outputs. Probably this would not happen if we had succeeded with the stopping of the weight programming when the error δ was zero due to identical O_{31} and t_1.

It should be said that if the targets were included when retrieving the training patterns, both of the outputs were giving correct outputs. It looks like that the variation of the weight values due to switching of the programming direction sign, influences the two outputs to not simultaneously learn all eight patterns.

Figure 5.9 shows the calculated mean square errors for each of the outputs (figure (a) and (b)) in addition to the mean of the outputs (figure (c)). In these calculations (see equation (5.1) for details) it is used -32.2nA for t_1=0, +17.4nA for t_1=1, -32.8nA for t_2=0, and +23nA for t_2=1.

From these measurements you can clearly see that both of the outputs have learned the function. After about 190 repetitions (7600 seconds) the network output has under 0.03% error. From this it may be concluded that the network has learned its training patterns. However, when we tried to recall these patterns, one of the outputs had not learned its training patterns as described above due to swing in weight values. Thus it exists a minor variation at the outputs which depends on whether or not the targets are included in the retrieval phase. To minimize the error at the output the targets should be included in the recall mode for our implementation.

.

Table 5.5 : ***Eight training patterns from the boolean function*** $In_1 In_2 (In_3 + In_4)$ **.**

Number	Input				Target	
	In_1	In_2	In_3	In_4	t_1	t_2
1	0	0	0	0	0	0
2	0	0	0	1	0	0
3	0	0	1	0	0	0
4	0	0	1	1	0	0
5	1	1	0	0	0	0
6	1	1	0	1	1	1
7	1	1	1	0	1	1
8	1	1	1	1	1	1

Table 5.6 : ***Training parameters for the eight pattern function.***

UV-filter	UV-light source distance	Pattern intervals	Weight time constant	Threshold time constant
B24	12cm	5s	1.13mV/interval	0.31mV/interval

Table 5.7 : ***Eight testing patterns from the boolean function*** $In_1 In_2 (In_3 + In_4)$ **.**

Number	Input				Target	
	In_1	In_2	In_3	In_4	t_1	t_2
1	0	1	0	0	0	0
2	0	1	0	1	0	0
3	0	1	1	0	0	0
4	0	1	1	1	0	0
5	1	0	0	0	0	0
6	1	0	0	1	0	0
7	1	0	1	0	0	0
8	1	0	1	1	0	0

Generalization

Normally, mapping of boolean functions by neural networks is done by learning of every patterns included in a function. This is mainly because it always exists a finite number of patterns in the functions. And when every combinations of the patterns is known for a selected boolean function, all these patterns may be included in the training set. Perfect generalization can be obtained, which means that it exist no unknown patterns and the network has learned everyone.

However, one of the features of neural networks is that we do not have to know the function we want to map. In large neural networks with thousands of input neurons it exits not always a knowledge of which function the patterns are describing. It will probably exist a number of input patterns not included in the training set which will have unknown outputs. For these cases it would be interesting to analyze the network to find out what it will do with patterns which never have been presented for the network; to find out how good the generalization characteristic of the network is.

The network was tested with the eight reminding patterns from the chosen boolean function $(In_1 In_2)\ (In_3 + In_4)$. These patterns are listed in table 5.7. The measured result of this test is shown in figure 5.7 (d) for O_{31} and in figure 5.8 (d) for O_{32}. A comparison of the measurements and table 5.7, shows that the network has achieved 100% generalization. The good result may be a consequence of training with a simple function. The generalization ability should be tested for more complex functions and a mean value should be obtained from repeated training lesson.

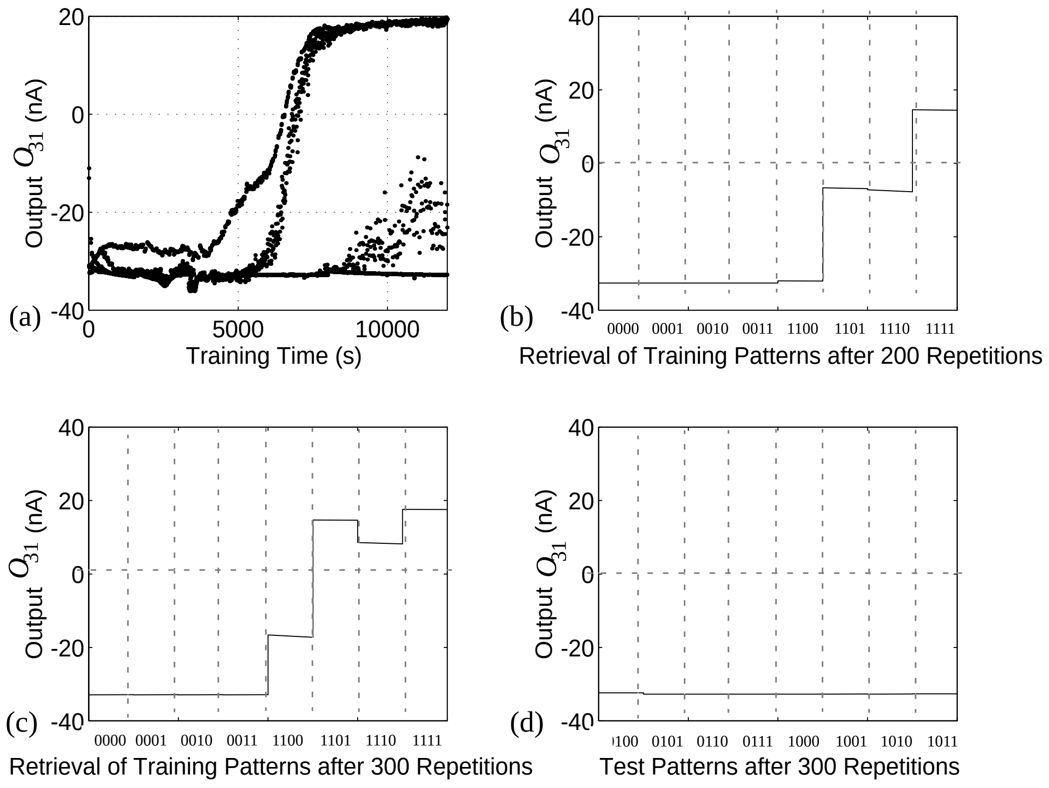


Figure 5.7 : ***Measured results of learning and recall of eight patterns for output** O_{31}. **Figure a) shows measurements during training. Figure b) shows measurements of the recall of the learning patterns with UV-light source off after 200 repetitions. Figure c) shows the same as (b) but after 300 repetitions. In figure d) the measured results from the presentation of the test patterns are shown.***

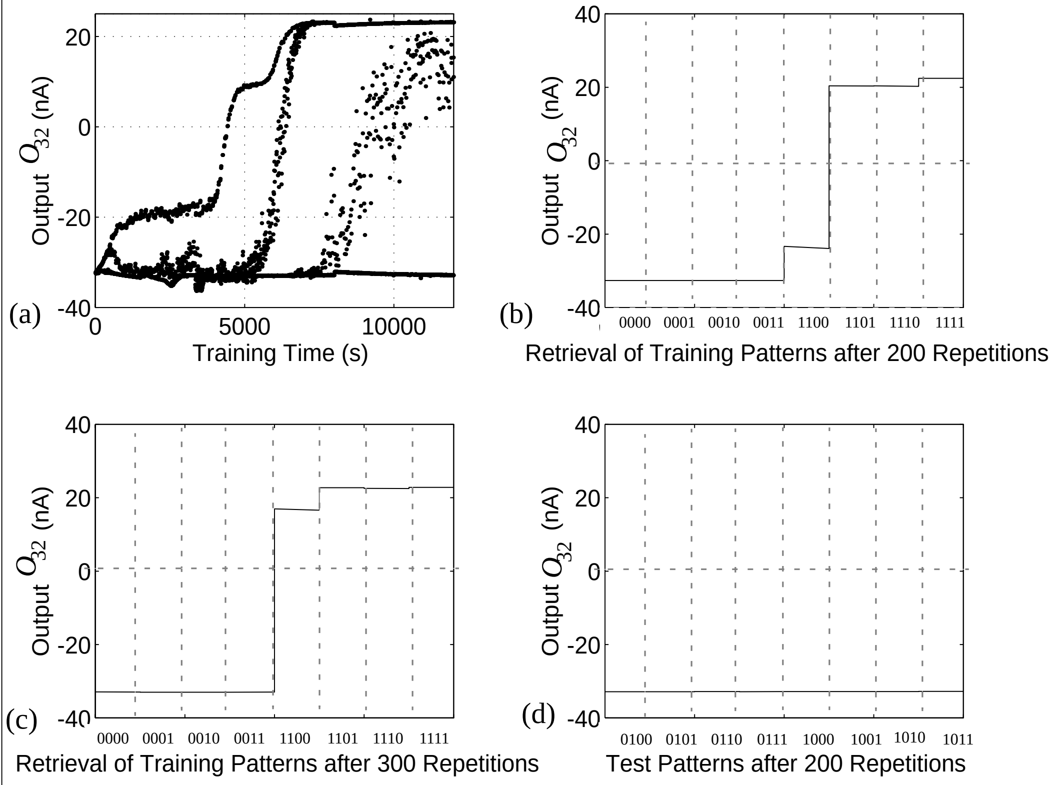


Figure 5.8 : ***Measured results of learning and recall of eight patterns for output*** O_{32}. ***Figure a) shows measurements during training. Figure b) shows measurements of the recall of the learning patterns with UV-light source off after 200 repetitions. Figure c) shows the same as (b) but after 300 repetitions. In figure d) the measured results from the presentation of the test patterns are shown.***

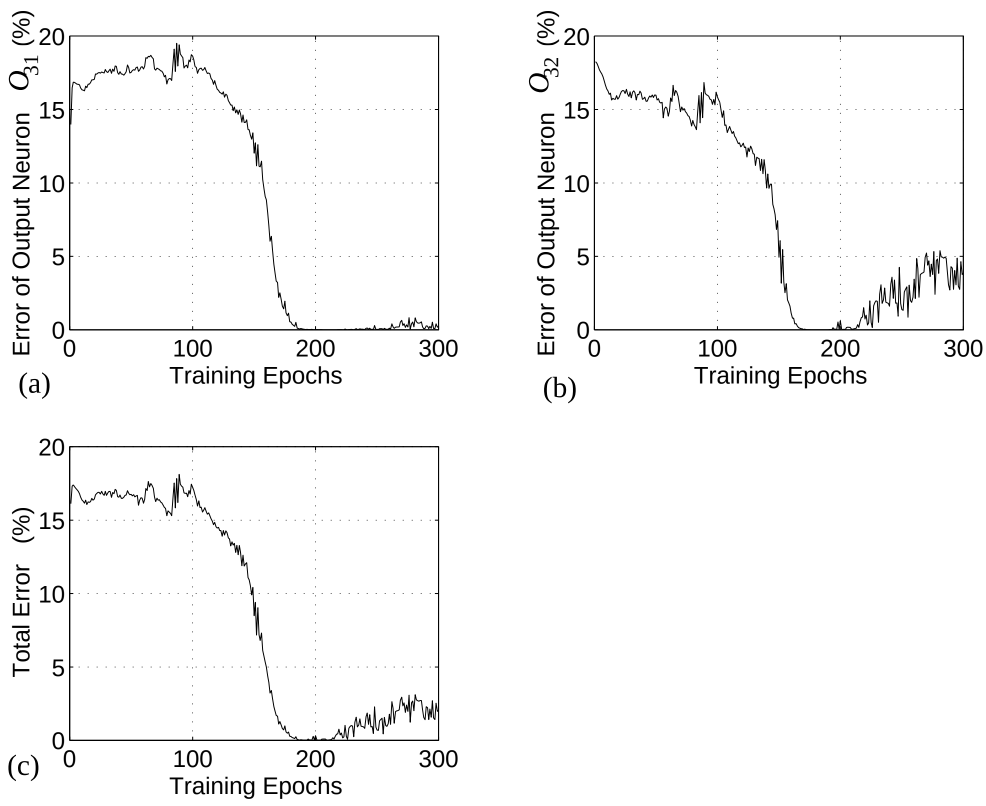


Figure 5.9 : ***The mean square error during training for the outputs. Figure a) shows the calculated MSE for*** O_{31} ***and figure b) for*** O_{32}***. Figure c) shows the average of the calculated results in (a) and (b).***

5.4.4 Incomplete learning

In previous chapter it was stated that when retrieving the training patterns, targets should be included. In this chapter it will be shown how targets may totally dominate the output when they are included in the retrieval phase.

A new training was started:
First we tried to initiate the network, but it was not successful. The hidden neurons did not respond for any alteration of the inputs. Some weights in the input to hidden layer connections probably were heavily exceeded the interesting operation range. Of course a new initiation could solve this problem. However, it would be interesting to see if the network could handle such situations. Eight patterns from a XOR-function listed in table 5.9, were picked out and training of 250 repetitions were started. The weight time constant described in table 5.8, was set to a large value because the training was planned to last over the night.

The measured results are shown in figure 5.10. Figure (a) shows the progress of the output O_{31} during training. Observe how nice the output is divided into two target groups as wanted.

Table 5.8 : ***Training parameters for the eight patterns XOR function.***

UV-filter	UV-light source dis-tance	Pattern intervals	Weight time constant	Threshold time constant
A10	12cm	30s	1.72mV/interval	0.50mV/interval

Table 5.9 : ***Eight patterns from the boolean function*** $In_2 \oplus In_3$.

Number	Input				Target
	In_1	In_2	In_3	In_4	t_1
1	0	0	0	0	0
2	0	0	0	1	0
3	0	0	1	0	1
4	0	0	1	1	1
5	1	1	0	0	1
6	1	1	0	1	1
7	1	1	1	0	0
8	1	1	1	1	0

Figure 5.10 (c) shows the calculated mean square error. After 100 repetitions it looks like that the network has learned the patterns. And a small drift is causing the network to develope an offset (from 140 to 250 repetitions). After the training phase we recalled the training patterns. The result is shown in figure 5.10 (d). The targets are included in the recall. It appears that the network could separate the patterns for high and low targets correctly. Only a offset created by a drift in the weights after an early finished learning of patterns, is destroying the network from achieving successful training after 250 repetitions.

However, new measurements of the hidden neurons showed that these neurons were still stuck at almost the same values. It was obviously that a variation in the output could never be caused by an input variation. The retrieval of the training patterns was performed once again, but this time the targets were locked at a fixed value. The result can be seen in figure 5.10 (b). Changing sign of the fixed value resulted in sign change of the output for all of the patterns.

The incorrect learning of the network is related to two conditions. The first one is the alteration in weight values due to switching of the sign of the weight programming. The switching of the sign is a function of the targets. Therefore an output variation is caused by a variation in weights which again is caused by a variation of a target.

Secondly, when the input to a hidden neuron is large, the derivative (bump signal) is approximately zero. Then the hidden layer error δ_j becomes almost zero and we have a condition identical to the one illustrated in figure 4.9 where the weight updates of w_{jk} are practically zero. The stop of weight updating when the derivative is zero is not always desirable and is one of the main drawbacks of the back-propagation algorithm. During the time that a neuron output has its maximum value and we want the output to be, as an example, of opposite sign, the derivative of the neuron output is stopping, or slowing down the process of changing the neuron output. It exist improvements of the standard back-propagation which handle this type of unwanted effects in a more efficient way. Hence he learning algorithm should be improved in a redesigned implementation.

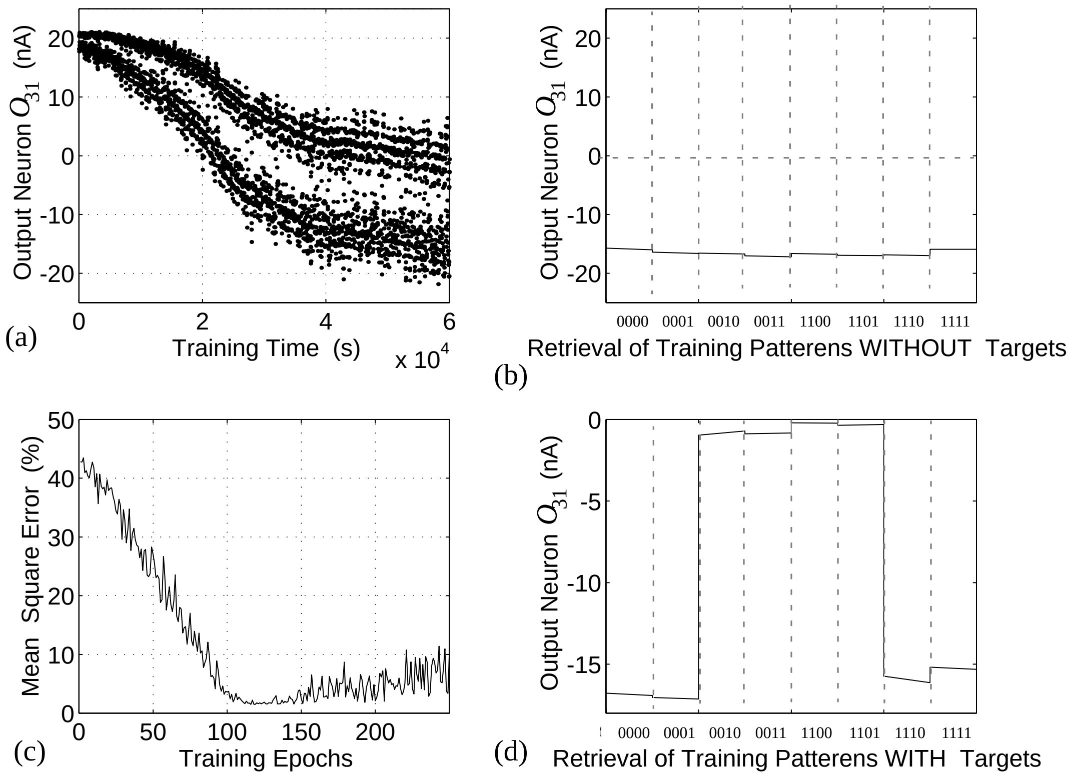


Figure 5.10 : ***An attempt to learn the eight patterns XOR function. All three hidden neurons were stuck at a fixed value during the training. Figure a) shows measurements during training. Figure b) and d) shows recall of the training patterns. From these two figures it should be obvious that it is the target s that determine the outputs. Figure c) shows calculated mean square error.***

6 Discussion and Conclusion

6.1 Discussion and improvements

It was shown in chapter 5 how the network could learn. The most important criterium stated to obtain learning, was the ability to ensure an appropriate initiation of the weights. If the weights are outside the interesting operation range (150-250mV), the network will probably not managing to learn the training patterns.

The initiation method used (a large number of up and down programming steps) was difficult to accomplish. We had to ensure that none of the derivative O' were near zero. This requirement was not easy to put through all the time. In chapter 5.3.1, two of the hidden neurons were reported stuck at their maximum value because of poor initiation. For this training session it was enough with one hidden neuron operative to learn the training patterns. However, in chapter 5.4.4 all three hidden neurons were reported out of function and it was impossible to learn the training patterns.

A suggestion of an improvement of the initiation problem could be to include a special mode where the initiation took place. One solution is to apply the V_{ref} signal which was planned to be used in the recall mode. The signal V_{ref} is already routed around the chip and can easily be connected to the derivative O'. When no scaling of O' is wanted, V_{ref} could be set to 0V. During the initiating of the network, V_{ref} could be set to 0.8V. Thus the initiation of the weights would not be stopped due to derivatives $O' = 0$.

Originally, the computation of the derivative O' is correct in accordance to the theory. However, sometimes especially at the beginning of training, the derivative O' may reduce the learning speed. In our network this situation may destroy the ability of learning as stated in chapter 5.4.4. The problem with the derivative O' in the standard back-propagation has been widely discussed in the theory [Fahlman][Hertz].

One solution could be to remove the calculation of the derivative O'. This can be done by applying an another error function estimator called relative entropy [Hertz]:

$$E = \sum_{ip} \frac{1}{2}\left(1 + t_i^p\right)\log\left(\frac{1 + t_i^p}{1 + O_i^p}\right) + \frac{1}{2}\left(1 - t_i^p\right)\log\left(\frac{1 - t_i^p}{1 - O_i^p}\right) \qquad (6.1)$$

It can be shown that the output layer error only becomes (when using gradient descent):

$$\delta_i = t_i - O_i \qquad (6.2)$$

As you can see, no derivative O' is included. The problems at the start of training are eliminated. However, the output layer error δ_i do not become equal to zero when the input to a output neuron exceeds its operation range. Thus the derivative can not prevent this input to exceed the input operation range. Notice also that this only concerns the output layer. In the hidden layer error we still have to include the calculation of the derivative O'.

[Fahlman] found a compromise:

$$\delta_i = (t_i - O_i)\ (O_i' + \alpha) \qquad (\alpha > 0) \qquad (6.3)$$

Equation (6.3) takes advantage of the features in the sum square error estimation as well as in the relative entropy error estimation. The parameter α could be set to a relatively high value before training and then be decreased during the training. The inclusion of α can in additional be applied to the hidden layer error δ_j. In an implementation this may be incorporated identically to the proposed solution on the initiation problem discussed above.

Another solution including stop of the weight updating, can be obtained by looking at the error:

$$E = \frac{1}{2}\sum (t_i - O_i)^2$$

When when error E is equal to zero, the network has learned the pattern presented. Therefore this pattern, strictly speaking, need no updating and next pattern may immediately be presented. We could also scale the time each pattern is presented in harmony with the size of the error E to speed up the learning. The only drawback is that these calculations have to be executed off-chip.

As described in chapter 3.2, we used two almost identical UV-models where the difference was the time constants. One for the weights W_{ij} and w_{jk}, and one with larger time constant for the thresholds Θ_i and Θ_j. Measurements of the influence of the scaling in the learning showed excellent results. The weights were adjusting themselves according to the threshold. A similar scaling could be extended to also yielding the weights W_{ij} and w_{jk}. w_{jk} could have a faster time constant than W_{ij}. This might lead to avoidance of some types of local minima and fasten up the convergence time.

In general, there are few restrictions of which type inputs, outputs and targets should be in feed-forward network. They could be boolean or continuous valued types. In our network inputs are scaled to be a current between zero and I_{b_k} (a bias current determined by the bias voltage $Bias_k$). If we want boolean inputs, we can define zero current as “0” (false) and $I_{b_{in}}$ as “1” (true). If we want continuous valued inputs, we have to operate in the linear range between zero current and I_{b_k}.

Outputs and targets operate in the range $-I_{b_i}$ to $+I_{b_i}$. If we want boolean outputs, we may define negative current as “0” and positive current as “1”. If we want continuous valued outputs, we have to operate the outputs (and the targets) in the linear range between $-I_{b_i}$ and $+I_{b_i}$. Usually, obtaining linear outputs, the activation function in outputs neurons is dropped. Thus these neurons will have only the summation of weights as their output. The above discussed item has shown that we may achieve mapping for both boolean functions and continuous valued functions in our network.

Probably the most complex boolean function a network with 4 inputs can learn, is the 4 bits parity problem. Parity problems are defined as “1” if the input pattern contains an odd number of “1”s and “0” otherwise. The XOR problem is a 2 bits parity problem. [Rummelhart] stated that a feed forward network requires at least N hidden neurons to solve parity for N-bits input patterns. Thus it is not possible for our network to solve the 4 bits parity problem since it has only 3 hidden neurons. We tested this function and the network did not manage to learn all the patterns included in the 4 bits parity function. An improved implementation should have at least identical number of hidden neurons and inputs. [Rummelhart] also showed that the convergence time is reduced linearly with the logarithm of the number of hidden neurons. The number of hidden neurons should therefore not be minimized for other reasons than limited space on the chip.

During the measurements we used 6 voltage sources to adjust the inputs. The voltage sources were programmed serially. This operations required at least 2 seconds for each repetition of a pattern. Thus less than 5 seconds presentation intervals were not used. For larger networks an improved method should be found. The UV-light distance could be reduced in such a way that the weight programming speed was increased by a factor of 5 compared to the speed used during the training. If we only reduced the UV-light distance and not the pattern presentation interval, the learning rate η reached a value which brought the network into a local minimum. To reduce the interval and obtain shorter convergence time a faster adjustment of the inputs and the targets have to be applied.

One solution is to use a computer with a parallel interface which is controlling analog multiplexers which again are controlling sample and holds circuits operating as voltage sources [Teeffelen]. Another solution is to only use digital inputs [Abusland94]. However, such a method will limit the operation of the network since a network with digital inputs only can map boolean functions.

6.2 Conclusion

A design of a feed-forward neural network with back-propagation on-chip learning in analog CMOS has been proposed. It works in continuous-time and has long term analog memories. With the structure used on the fabricated chip, it is possible to build networks of any size, also wafer scale. It has been shown that back-propagation implemented in analog CMOS operating in subthreshold region is possible. The high accuracy in memory and back-propagation described for digital systems appeared to be inessential for our analog system.

Some unwanted offsets in the memory circuit are reported, and when not compensating for this error it was sometimes difficult to retrieve all the learned patterns. To ensure on-chip learning of hard tasks such as the parity problem, a more carefully design of the memory programming circuit should be carried out.

Although we succeeded with the initiation of the network in most cases, an improved method should be found to ensure a fast and reliable initiation.

The layout error described on page 34 resulted in a different target presentation the originally planned. With this compensating technique the network could not take full advantage of the interesting findings regarding the stability of the memories shown in figure 4.9. However, the network still managed to learn its training patterns. This shows how powerful the back-propagation algorithm is, especially in analog systems.

The bottleneck in our system is the learning speed, due to the UV-memory. Fortunately, the programming speed for our memories is independent of the network size. Thus when going up to wafer scale integration, our network may compete even with digital processors working in parallel. Appropriate tasks for networks on such hardware may be speech- and pattern recognition where the number of free parameters exceeds 10^4. Since we can not reduce the time constant for the UV-memory after processing the chip, several other techniques have been proposed to speed up the convergence time in this thesis.

6.3 Further work

Continuous development of the UV-memory is taken place at our institute. From this work it has been constructed circuits with other utilities such as the UV-light programmable voltage reference [Abusland93].

To speed up the programming time of the UV-memories, new ideas have to be found. In these days a new UV-structure is under testing, where the poly1 to poly2 capacitance is replaced with a diffusion to poly1 capacitance (a transistor with common drain and source). The development of the UV-structure has to proceed to increase the programming speed.

An improved method of controlling the inputs to a network should be found. The method applied for our network has limited operation. The learning speed could have been increased at least by a factor of 5 if a faster adjustment of the inputs had been applied.

An advancement of our network could be a larger network with a special task to solve. A larger network can be implemented on a larger chip, but such a chip will increase the chip development cost. The fault tolerance allowed for expensive chips is minimal. It is, however, possible to build many small chips at low-cost and connect these chips into a network [Lansner]. The chips can be divided in two groups: neuron chips and weight chips. An flexible architecture can be obtained because the number of applied neurons can be adjusted even after the processing of the chips.

Bibliography

[Abusland93] Aanen Abusland, Tor Sverre Lande
Local Generation and Storage of Reference Voltages in CMOS Technology Using UV-light
Proc. 11th European Conf. Circuit Theory and Design, Vol. I, 1993, pp. 281-286

[Abusland94] Aanen Abusland
A CMOS Analog Hopfield Net with Local Adaption and Storage of Weights
Cand. Scient. thesis, Institute of Informatics, The University of Oslo, February 1994

[Allen] Philip C. Allen, Douglas R.Holberg
CMOS Analog Circuit Design
Holt, Rinehart and Winston 1987

[Benson] Ronald G. Benson, Douglas A. Kerns
UV-Activated Conductances Allow for Multiple Time Scale Learning
IEEE Transactions on Neural Network, Vol. 4, No. 3, May 1993

[Delbrück] Tobi Delbrück
"Bump" Circuits for Computing Similarity and Dissimilarity of Analog Voltages
Caltech 1991

[Fahlman] S.E Fahlman
Fast-Learning Variations on Back-Propagation: An Empirical Study
In Proceedings of the 1988 Connectionist Models Summer School pp 38-51, San Mateo: Morgan Kaufmann

[Gilbert] B. Gilbert
Translinear Circuits: a Proposed Classification
Elecronic letters, Vol. 11, pp. 14-16, 1975

[Hertz]	John Hertz, Anders Krogh, Richard G. Palmer *Introduction to the Theory of Neural Computation* Addison-Wesley, 1991
[Kerns]	D. A. Kerns *Experiments in Very Large-Scale Analog Computation* California Institute of Technology PhD thesis 1993
[Lansner]	John A. Lansner and Torstein Lehmann *An Analog CMOS Chip Set for Neural Networks with Arbitrary Topologies* IEEE Transactions on Neural Networks, Vol. 4, No. 3, May 1993
[Lehman]	A. von Lehman, E.G. Paek, P.F. Liao, A. Marrakchi, J.S. Patel *Factors Influencing Learning by Back-Propagation.* IEEE International Conference on Neural Networks, Vol. I, pp. 335-341, San Diego 1988
[Lont]	Jetzy B. Lont, Walter Guggenbühl *Analog CMOS Implementation of a Multilayer Perceptron with Nonlinear Synapses* IEEE Transactions on Neural Network, Vol. 3, No. 3, May 1992
[Maher]	M. A. Maher *New UV-Memory Writing Scheme* (Unpublished) 1992
[Mead]	Carver Mead *Analog VLSI and Neural System* Addison-Wesley 1989
[Murray]	Alan F. Murray *Multilayer Perceptron Learning Optimized for On-Chip Implementation: A Noise-Robust System* Neural Computation, Vol. 4, No. 3, pp 366-81 1992 Cambridge,Mass. ISSN 0899-7667
[Pineda]	F. J. Pineda *Dynamics and Architecture for Neural Computation* Journal of Complexity, Vol. 4, pp. 216-245, 1988
[Rummelhart]	D. E. Rummelhart, J. L. McClelland *Parallel Distributed Processing - Explorations in the Microstructure of Cognition* Vol.1 Foundations MIT Press 1986

[Soelberg] Knut Soelberg, Roy Ludvig Sigvartsen, Tor Sverre Lande, Yngvar Berg.
An Analog Continuous-Time Neural Network..
Analog Integrated Circuits and Signal Processing
Vol. 5, page 235-246, 1994

[Tarassenko] Loinel Tarassenko, Jon Tombs
On-Chip Learning With Analouge VLSI Neural Networks
in Proceedings of the Third MicroNeuro conference, pp. 163-174, Edinburgh, April 1993

[Teeffelen] J. J. M. van Teeffelen
Interfacing Neural Network Chips with a Personal Computer
Master thesis, Faculty of Electrical Engineering, Eindhoven University of Technology, August 1993

[Torrance] R. R. Torrance, T. R. Viswanathan, J. V. Hanson
CMOS Voltage to Current Transducers
IEEE Transactions on circuits and system, Vol. CAS-32, No. 11, November 1985

[Toumazou] C. Toumazou, F. J. Lidgey, D. G. Haigh
Analogue IC design: the Current-Mode Approach
Peter Peregrinus Ltd. 1990

[Tsividis] Y. Tsividis
Moderate Inversion in MOS Devices
Solid State Electronics, Vol. 25, No. 11 1982 pp. 1099-1104

[Watts] L. Watts, D. A. Kearns, R. F. Lyon, C. A. Mead
Improved Implementation of the Silicon Cochlea
IEEE Journal of Solid-State Circuits Vol. 27 No. 5 May 1992

[Weiss] Shalom M. Weiss, Casimir A. Kulikowski
Computer Systems That Learn
Morgan Kaufmann, San Mateo, California 1991

A Analog CMOS

A.1 CMOS transistor

A simple way to explain how a CMOS-transistor works is to compare it with a switch which has two states: ON or OFF. Figure A.1 shows a circuit symbol of a n-channel CMOS transistor. The CMOS-process includes two types of transistors, one n-channel type transistor which has electrons as charge carriers, and one p-channel type transistor which has holes as charge carriers. The circuit symbol for a CMOS transistor has three terminals. A voltage V_{gs} (between the Gate - and the Source terminal) decide if a current I_{ds} will flow from the Drain - to the Source terminal..

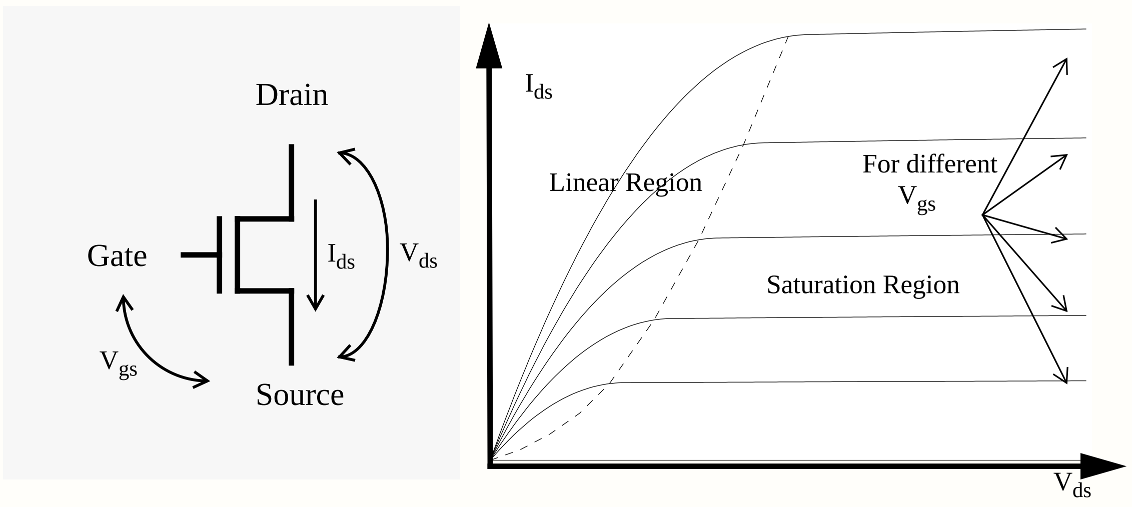


Figure A.1 : ***Circuit symbol for and n-channel transistor and a simulation of it in* hspice. *V_{ds} is swept from 0V to 2V for various voltages of V_{gs} between 1.5V and 2.5V***

We may model the behavior of the transistor in an analog fashion. Figure A.1 shows the current I_{ds} as a function of the voltage V_{ds}. It is possible to split this function into two region:

- One linear region where I_{ds} is dependent of V_{ds} (among others) and
- One saturation region where I_{ds} is almost *in*dependent of V_{ds}.

Common for the linear region and the saturation region shown in figure A.1 is that $V_{gs} > V_T$ where V_T is the threshold voltage. If $V_{gs} < V_T$ then the current I_{ds} is equal to approximately zero. Although this statement can be read in many books, it is not exactly true. When $V_{gs} < V_T$ it flows a tiny, but well-defined current through the channel of the transistor

We can say that the voltage V_{gs} splits the behavior of the current I_{ds} into two region (see figure A.2):

- Strong inversion region ($V_{gs} >> V_T$). Here the square-law function of the current I_{ds} yields [Allen] and the current in the channel is flown by drift.
- Week inversion region ($V_{gs} << V_T$). Here has the current I_{ds} an exponential behavior [Mead] and the dominant current-flow mechanism is diffusion.

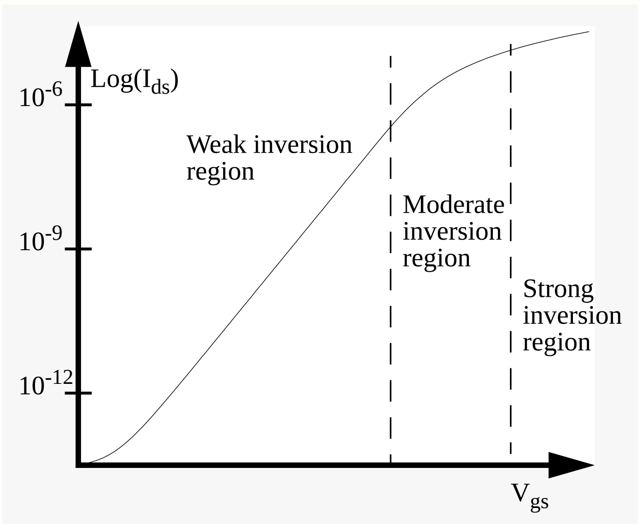


Figure A.2 : ***Simulation of a n-channel (n-type) transistor in* hspice *with BSIM parameters.***
V_{gs} is swept from 0V and up to 1.3V and V_{ds} is kept at a constant voltage (2.0V).

In the transition between the two regions none of these above mention behavioral yields alone. This region is called the moderate inversion region [Tsividis] and the range is typical 0.5V.

A.1.1 Subthreshold operation (week inversion)

A good electrical model of the current flowing through a n-type CMOS transistor which operating in the subthreshold region, is described in [Mead]:

$$I_{ds} = I_0 e^{-\frac{q\kappa V_g}{kT}} \left(e^{\frac{qV_s}{kT}} - e^{\frac{qV_d}{kT}} \right) \tag{A.1}$$

where

I_0 is a physical constant (including the width and the length of the transistor.)
q is the electron charge (positive for p- (holes) and negative for n-type transistors (electrons)).
k is the Boltzmanns constant.
T is the Temperature.
κ describes the reduced gate effectiveness.

The term $\frac{kT}{q}$ is often called the termal voltage and named V_T and usually has a magnitude equal to 25mV at room temperature. The model for p-channel type transistor is similar to equation (A.1) but all the voltages have opposite signs.
We may shorten the equation (A.1) and scale the tree voltages V_g, V_s and V_d by $\frac{kT}{q}$:

$$I_{ds} = I_0 e^{\kappa V_g} (e^{-V_s} - e^{-V_d}) \tag{A.2}$$

$$I_{ds} = I_0 e^{\kappa V_g - V_s} (1 - e^{-V_{ds}}) \tag{A.3}$$

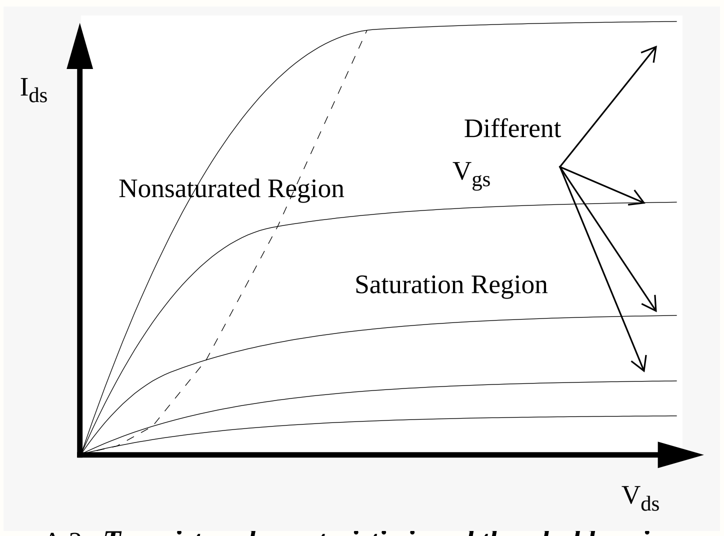


Figure A.3 : ***Transistor characteristic in subthreshold region. V_{ds} is swept a few 100mV.***

The drain current I_{ds} saturates when $V_{ds} \geq 4\frac{kT}{q}$ because then the value of the term $e^{-V_{ds}}$ becomes near zero. Equation (A.3) becomes:

$$I_{ds} = I_{sat} = I_0 e^{(\kappa V_g - V_s)} \tag{A.4}$$

Figure A.3 shows this region (Saturation region) where I_{ds} is theoretical independent of V_{ds} . But as you may see in the figure I_{ds} is increasing slowly with V_{ds}

A.2 Effects to consider in analog CMOS

A.2.1 Early effect

The current I_{ds} increases slowly with the voltage V_{ds} (in the saturation region) which can be seen in Figure A.3. The increase in I_{ds} is due to the increase in the width of the depletion layer (surrounding the drain) when the drain voltage increases.The growing in the depletion layer width is decreasing the channel length. A shorter channel results in a shorter path for the electrons to flow through which has the effect that it will increase the current I_{ds} . This effect is called the Early effect [Mead] and if we include it into equation (A.4), we get:

$$I_{ds} = I_{sat}\left(1 + \frac{V_{ds}}{V_0}\right) \tag{A.5}$$

Where V_0 is a constant that describes the channel length modulation (for a given transistor size). To make the transistors least affected by the Early effect you have to build the transistors with a long channel.

A.2.2 Body effect (substrate effect)

A problem in the fabrication of the chips is the pollution of the substrate. This will lead to decreased efficiency of the gate voltage:

> When we increase the source voltage V_s (for a n-type transistor) we have to increase the gate voltage V_g even more to keep the current I_{ds} constant.

Therefore the parameter κ is introduced in equation (A.1) to compensate for this effect.

A.2.3 Transistor mismatch

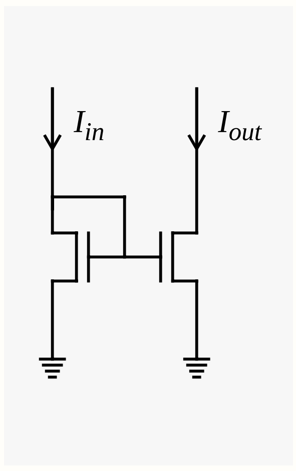
Unfortunately matching transistors in the subthreshold region is difficult. The current flowing in two identical transistor (size and biasing) may differ by a factor of 2 [Mead]. One of the major reason is the nonuniform doping on the chip. It is possible to improve the matching by increasing the size of the transistors. The matching error decreases roughly as the square root of the increase in the transistor area [Watts].

A.2.4 Temperature variations

CMOS transistors are considerable sensitive to temperature variations especially when they are operating in subthreshold region. From the equation (A.1) you can see that the current I_{ds} increases with an increase in the temperature. With this increase will also the transistor mismatch increase and in a worst case analyze a mismatch may give different results for different temperature.

A.3 Current mirror

The most common operation in analog circuit design is to copy a current. In figure A.4 the circuit shown copies the current I_{in} a single time. With this circuit we can copy the current I_{in} as many times as we want to because it is no load on the input. The current mirror shown in figure A.4 can only copy unidirectional currents.



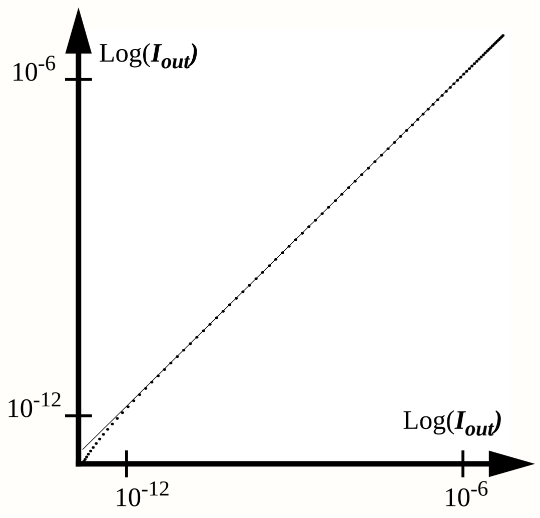


Figure A.4 : ***Current mirror***
The points in the right figure is a simulation of the current mirror and the line is an exact copy.

A.4 Transconductance amplifier (transamp)

Often we represent a signal as the difference between two voltages. To manipulate these type of signals we usually use some variant of the differential pair shown in figure A.5 a). A variant is the transconductance amplifier (transamp) which only includes a current mirror on the top of the pair shown in figure A.5 b). The output of a transamp is a sigmoid function [Mead]:

$$I_{out} = I_1 - I_2 = I_b \tanh\left(\frac{\kappa(V_1 - V_2)}{2}\right) \qquad (A.6)$$

(Remember that all voltages are scaled by $\frac{kT}{q}$)

Where I_b is the current through the transistor controlled by the voltage V_b, and I_1 plus I_2 is defined as:

$$I_1 = I_b \frac{e^{\kappa V_1}}{e^{\kappa V_1} + e^{\kappa V_2}} \text{ and } I_2 = I_b \frac{e^{\kappa V_2}}{e^{\kappa V_1} + e^{\kappa V_2}} \tag{A.7}$$

What can this amplifier calculate? It performs at least a substraction of two voltages and amplify this with a tangents hyperbolic function into a current.
We will use the transamp:

- in multiplication (of I_b and tanh)
- in substraction (of V_1 and V_2)
- to perform a sigmoid function of the input

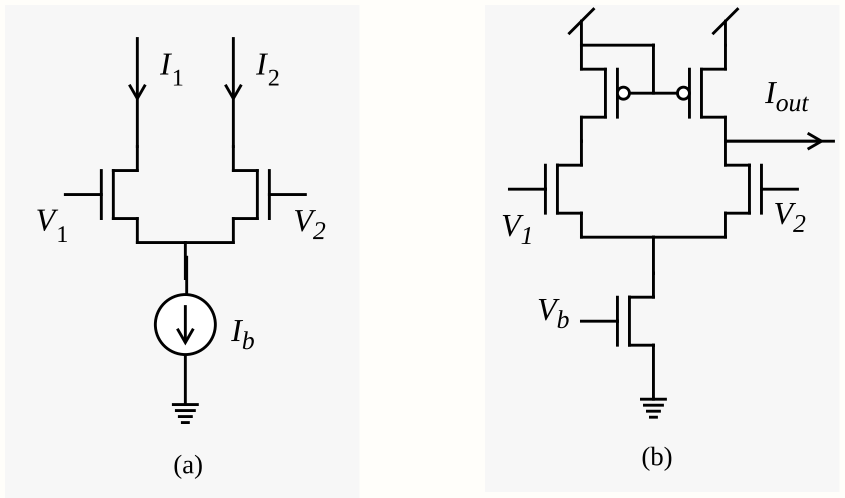


Figure A.5 : ***A differential pair and a transconductance amplifier***

B A Kangaroo Story

This story was found in the Internet news group *comp.ai.neural-nets* and was written by Warren Sarle. It explains the standard backprop algorithm in a easy and funny way.

Training a network is a form of numerical optimization, which can be likened to a kangaroo searching for the top of Mt. Everest. Everest is the global optimum, but the top of any other really high mountain such as K2 would be nearly as good. We're talking about maximization now, while neural networks are usually discussed in terms of minimization, but if you multiply everything by -1 it works out the same
Initial weights are usually chosen randomly, which means that the kangaroo may start out anywhere in Asia. If you know something about the scales of the inputs, you may be able to get the kangaroo to start near the Himalayas. However, if you make a really stupid choice of distributions for the random initial weights, or if you have really bad luck, the kangaroo may start in South America.
In standard backprop, the kangaroo is blind and has to feel around on the ground to make a guess about which way is up. He may be fooled by rough terrain unless you use batch training. If the kangaroo ever gets near the peak, he may jump back and fourth across the peak without ever landing on the peak.

C Details of the Chip

An more extensive presentation about the chip will be shown to demonstrate details such as the layout of the UV-structures and the chip. In addition, neuron and weight modules at transistor level are shown.

The layout was don on late winter of 1994 and the chip was processed at Mosis in the spring. The process was a 2.0 µm p-well Orbit semiconductor run. The Mosis test results showed a threshold voltage for a minimum n-channel type transistor equal to 0.98V and for a minimum p-channel type transistor -0.75V.

C.1 Neuron and weight modules at transistor level

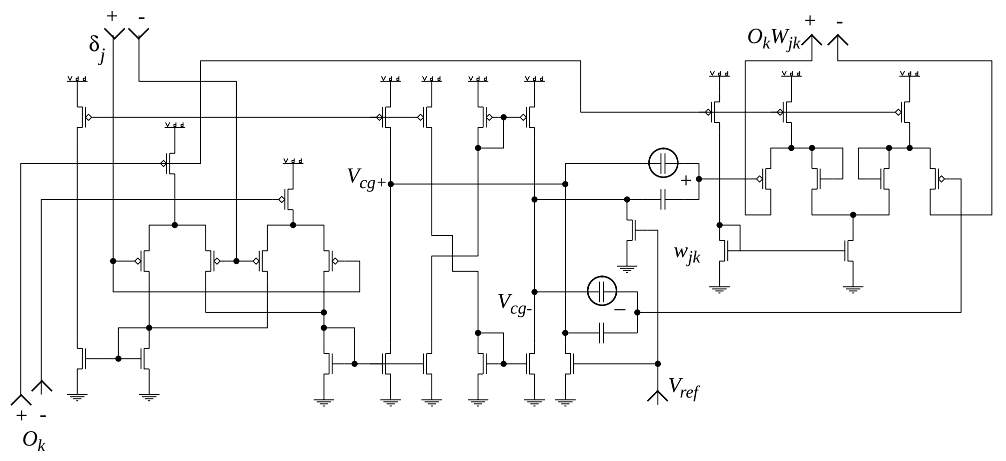


Figure C.1 : ***The weight module** w_{ij} **at transistor level.***

Since the layout of the chip is divided into neuron and weight modules, it might be interesting for the reader to see these modules at transistor level. It exists two weight modules: One for the input to hidden layer connections (w_{jk} - module) and one for the hidden to output layer connections (W_{ij} - module). The difference between these two modules is that the W_{ij} - module also includes a Gilbert multiplier to compute the propagated error. Figure C.1 describes the w_{jk} - module. It contains 28 transistors in addition to a differential UV-structure.

The hidden neuron and the output neuron are designed identically. The neuron module with output labels is shown in figure C.2. The module contains 48 transistors and a differential UV-structure. The input neurons contain only a transamp which includes 5 transistors.

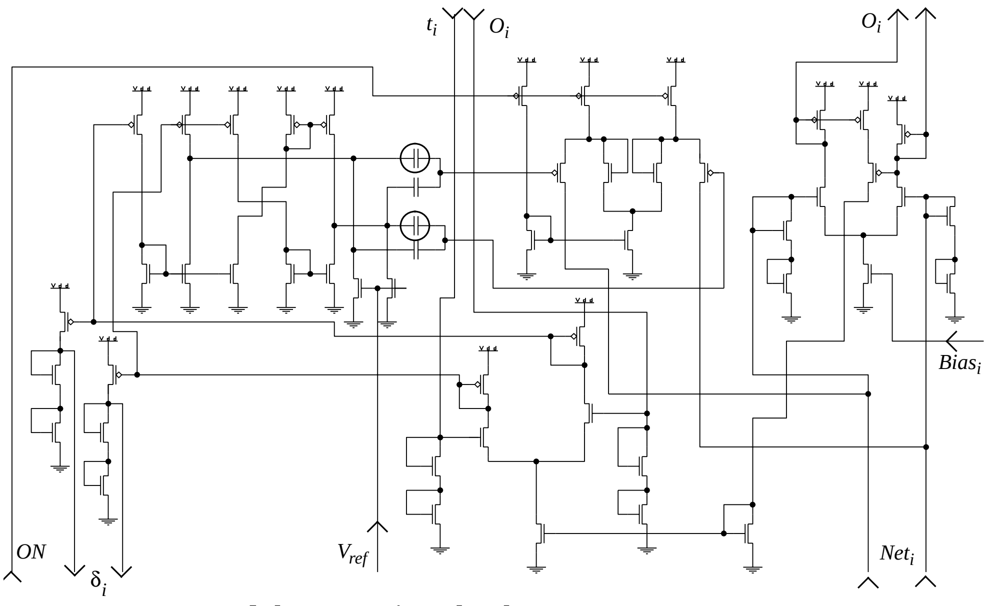


Figure C.2 : ***Neuron module at transistor level.***
The labels shown in the figure indicate that this neuron is a output neuron.

C.2 The layout

The chip layout was drawn by the help of the IC design system NE$_W$OL. Netlists from NE$_W$OL and the transistor simulator program AnaLOG were successfully compared before sending the chip to processing. Figure C.3 a floorplan for the chip is shown. The figure is a photo taken at our institute where we have a microscope with a RGB connection to a Silicon Graphics computer (Indy). The figure is magnified 4000 times.

The layout size of a w_{jk} - module is $300x280\mu m$, and for a W_{ij} - module $400x300\mu m$. For a neuron module the layout size is $440x300\mu m$. A small degree of effort was done to minimize these modules. It is possible to build larger network at Mosis, however, more circuitry outside the chip has to be build to control the inputs and outputs to a larger network.

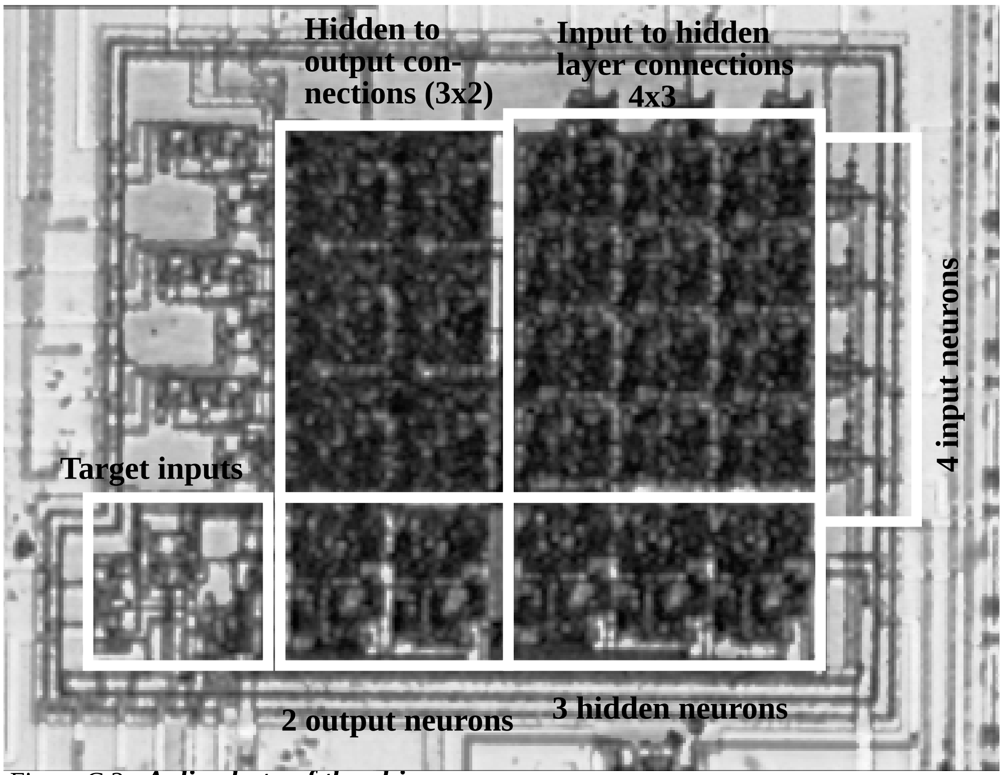

Hidden to output con-nections (3x2)
Input to hidden layer connections 4x3
4 input neurons
Target inputs
2 output neurons
3 hidden neurons

Figure C.3 : ***A die photo of the chip.***
The figure displays the floorplan of the chip. The size of the network without the extra read-out circuitry is 2000x2000 µ m.

A photo of the UV-structure model A is shown in figure C.4. The photo is magnified 110000 times. The UV-window may be seen as a dark area and it is placed over the poly2 area. Figure C.5 illustrates photo of the UV-structure model B. In this model the UV-window (marked with a white circle) is only covering one side of the poly2 area.

Figure C.6 describes a diagram of the padframe, which demonstrates where the inputs and the outputs are.

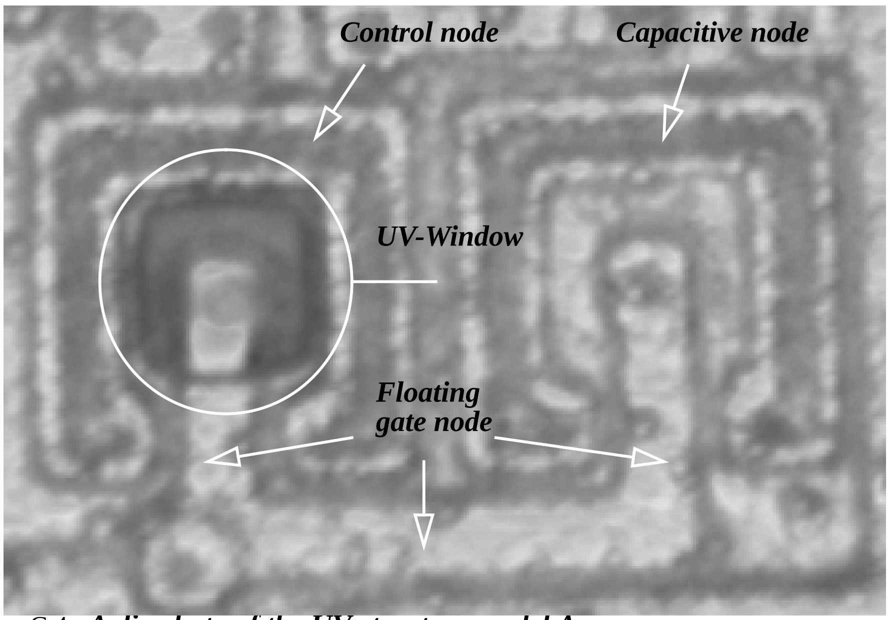


Figure C.4 : ***A die photo of the UV-structure model A. The structure in the figure has a real size of 100x50 μm***

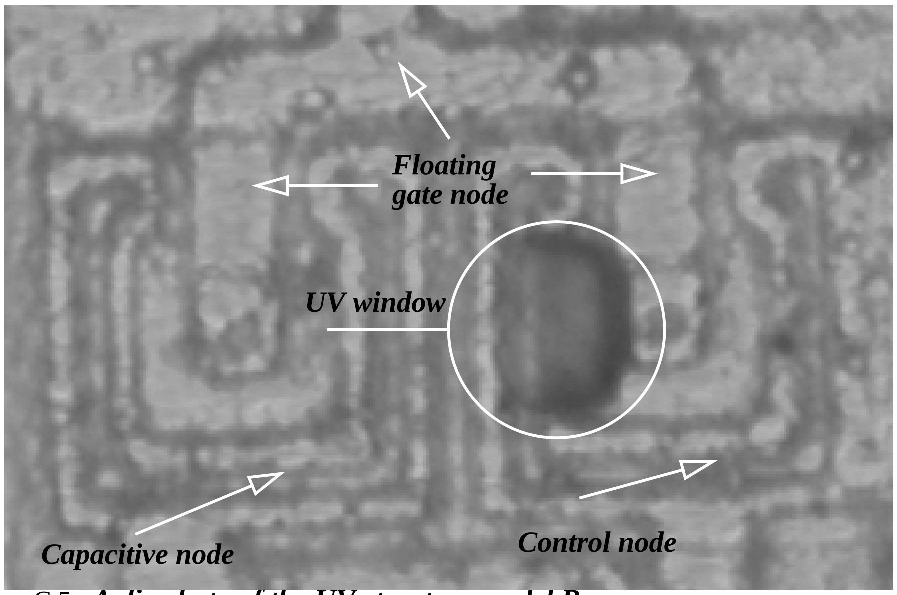
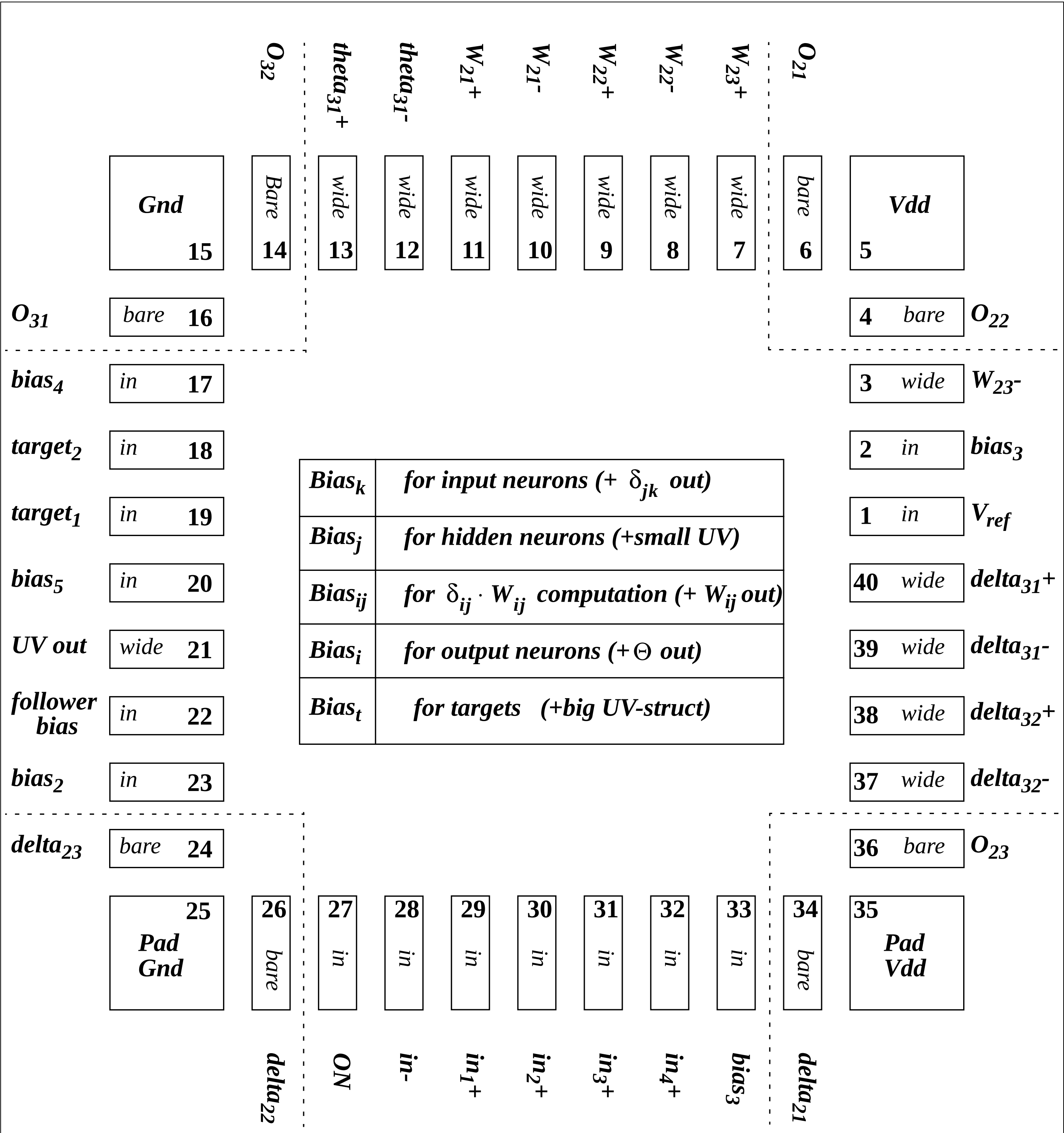


Figure C.5 : ***A die photo of the UV-structure model B. The structure in the figure has a real size of 100x50 μm***



$Bias_k$	for input neurons (+ δ_{jk} out)
$Bias_j$	for hidden neurons (+small UV)
$Bias_{ij}$	for $\delta_{ij} \cdot W_{ij}$ computation (+ W_{ij} out)
$Bias_i$	for output neurons (+Θ out)
$Bias_t$	for targets (+big UV-struct)

Figure C.6 :

D The paper

Knut Soelberg visited the NORCHIP seminar in Finland 1992. The talk he had at this seminar resulted in an invitation from a journal to send an extended version. Since I was measuring on his chip at this time, the job to extend his work was handed over to me. The final result was published in the journal: *Analog Integrated Circuits and Signal Processing*. The paper is included in the next pages.

At the moment a short version of this thesis is under development. This paper is planned to be sent to the NORCHIP seminar of 1994 held in Gothenburg (deadline 1. September).