Technical University of Denmark

Hardware Learning in Analogue VLSI Neural Networks

Lehmann, Torsten; Bruun, Erik

Publication date:
1995

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Lehmann, T., & Bruun, E. (1995). Hardware Learning in Analogue VLSI Neural Networks. Kgs. Lyngby, Denmark: Technical University of Denmark (DTU).

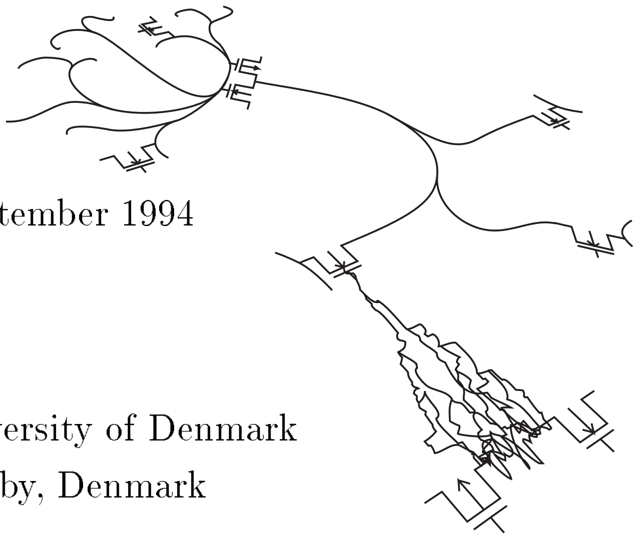
ELECTRONICS
INSTITUTE

Hardware Learning in Analogue VLSI Neural Networks

A thesis by

Torsten Lehmann

In partial fulfillment of
the requirements for the degree of
Doctor of Philosophy



September 1994

Technical University of Denmark
DK-2800 Lyngby, Denmark

Typeset using TEX
(plain format ver. 3.1415N,
PhDMac ver. 1.31 by TL)

Edition 1.03

Abstract

English

In this thesis we are concerned with the hardware implementation of learning algorithms for analogue VLSI artificial neural networks. Artificial neural networks (ANNs) are often successfully applied to problems for which no algorithmic solution exist, but can be described by examples. ANNs are fault tolerant and parallel of nature; analogue VLSI is a technology that can efficiently exploit these properties providing high performance systems. Analogue VLSI implementations of recall mode ANNs are maturing, but the equally important problem of implementing programming (or learning) hardware for these is still in its infancy.

We shall present the analogue VLSI implementation of two supervised, gradient descent learning algorithms for ANNs: the error back-propagation learning algorithm (BPL) for layered feed forward ANNs and the real-time recurrent learning algorithm (RTRL) for general recurrent networks. Both algorithms teach a cascadable analogue VLSI chip set for ANNs which we shall also describe. This chip set use simple capacitive weight storage with a digital RAM back-up memory. The BPL algorithm is implemented on-chip based on a novel bidirectional principle resulting in a very modest hardware increase compared to the recall mode system. The RTRL algorithm is implemented as add-on hardware to the recall mode system, using a compromise between computational speed and hardware consumption. The implementations of several algorithmic variations are also considered (eg. weight decay and momentum). Results from measurements on the fabricated chips are presented as well as measurements on a recall mode system.

We display the novel category of gradient descent like algorithms, non-linear gradient descent, which are better suited for hardware implementations than ordinary gradient descent; both in terms of accuracy and hardware consumption. Further, we argue that ANN ensembles should be used to improve performance of analogue neural systems. Also included are novel considerations on analogue computing accuracy, offset compensation, derivative computation, analogue memories, network topologies, process parameter dependency canceling, and learning in systems with RAM back-up, among other things.

We conclude that though the technology is promising for implementing learning algorithms much research is still needed; both at a algorithmic level and at a implementation level.

Dansk

I denne afhandling skal vi beskæftige os med hardware implementeringer af indlæringsalgoritmer til analoge VLSI kunstige neurale netværk. Kunstige neurale netværk (artificial neural networks, ANNs) er ofte med held brugt på problemer for hvilke der ikke eksisterer nogen løsningsalgoritme, men som kan beskrives ved hjælp af eksempler. ANNs er af natur fejltolerante og parallelle; analog VLSI er en teknologi, som effektivt kan udnytte disse egenskaber til implementering af systemer med stor ydeevne. Analog VLSI implementeringer af fast-programerede ANNs er ved at modnes, men det ligeså vigtige problem at implementere programerings (eller indlærings) hardware til disse er stadig i sin spæde begyndelse.

Vi skal her præsentere analoge VLSI implementeringer af to overvågede, gradient nedstignings indlæringsalgoritmer til ANNs: “Error back-propagation” indlæringsalgoritmen (BPL) til lagdelte netværk uden tilbagekobling samt “real-time recurrent learning” algoritmen (RTRL) til generelle, tilbagekoblede netværk. Begge algoritmer oplærer et kaskadekoblet, analogt VLSI chipsæt til ANNs, som vi også skal beskrive. Dette chipsæt benytter et simpelt kapacitivt vægtlager med en digital RAM hukommelse til vægtopfriskning. BPL algoritmen er baseret på et nyt bidirektionelt princip og implementeres internt på ANN chipsættet ved brug af ganske lidt ekstra hardware. RTRL algoritmen implementeres med extern hardware og som et kompromis mellem beregningshastighed og hardware forbrug. Implementeringer af forskellige varianter af algoritmerne bliver også overvejet (fx. vægt henfald og moment). Resultater fra målinger på de fremstillede chips vil blive præsenteret, såvel som målinger på et system baseret på ANN chipsættet.

Vi fremviser den ny kategori af gradient nedstignings lignende algoritmer, ikkelineær gradient nedstigning, som er bedre egnet til hardware implementeringer end sædvanlig gradient nedstigning; både med hensyn til præcision og hardware forbrug. Endvidere påpeger vi at ANN “ensembles” bør benyttes for at forbedre ydeevnen af analoge neurale systemer. I teksten præsenteres også nye overvejelser angående, blandt andet, præcision af analoge beregnende enheder, offset kompensering, differentialkvotient beregning, analoge hukommelser, netværks topologier, udligning af proces-parameter afhængighed, og indlæring i systemer med RAM opfrisknings hukommelse.

Vi konkluderer at, selvom teknologien er lovende for implementering af indlæringsalgoritmer, er megen forskning stadig nødvendig; både på et algoritmemæssigt niveau samt på et implementeringsmæssigt niveau.

Preface

The present thesis is a partial fulfillment of the requirements for the degree of *Doctor of Philosophy* (licentiatgraden, Philosophiae Doctor, Ph.D.). The work was carried out at the *Electronics Institute*, the *Technical University of Denmark*, and was funded on a scholarship from the Technical University of Denmark. Professor *Erik Bruun* of the Electronics Institute was supervisor.

I have tried to make this thesis a coherent presentation of *hardware learning in analogue VLSI neural networks* — though by no means exhaustive. This has made it necessary to include work that is not entirely my own and I will state it clearly whenever this is the case. In particular, my fellow Ph.D. student, John Lansner, was responsible for large parts of the work in *implementation of neural networks.* Thomas Kaulberg was responsible for the *op-amps* (and *current conveyors*) used on the chips. A Masters student of mine, Jesper Schultz, did much of the work on the *sparse input synapse chip* architecture. Finally, John Hertz, Benny Lautrup and Anders Krogh were responsible for the development of *non-linear back-propagation.* I have tried to refer other authors whenever possible; what is not refered (apart from well known matters) is, for most parts, my own work. Some people will undoubtedly find parts of the text provocative (eg. when I argue that one should refrain from using floating gate memories for synapse strengths) — please do not take offense, it is by no means a mark of disrespect of other peoples work; merely personal views (as well as a deliberate attempt to provoke which I think is sound for development in any field of research).

Regarding the *layout* of the thesis: Disgracefully (some would say) I have the habit of using parenthesis quite often. Note that often they act like subordinate clauses; their contents being important. Figure- and equation numbers, which are globally enumerated, carry chapter labels as superscripts for the ease of location (as figure 37^4 and (9^4)). In most places references to other peoples work (as *Sánchez-Sinencio and Lau* [206]) are incomplete in the sense that only a few of the relevant references are displayed; references that cover material related to the text are usually preceded by "cf." or "se also". *Italics* font are used for emphasis and for concepts that are found in the index. I use "we" as as the personal pronoun throughout the thesis as I think this eases reading.

The *thesis* is organized as follows:

In the *introduction*, the field of *analogue VLSI neural networks*, including *hardware learning*, is briefly introduced. Motivations for the research are given and the objective of the thesis is defined.

In the *implementation of the neural network* chapter, the neural network architecture that will make the basis of the rest of this thesis is presented. The results of the chip-in-the-loop training also serves as a standard of reference for on-chip learning implementations.

In the *preliminary conceptions on hardware learning* chapter, the choices of learning algorithms for implementation are considered and I elaborate on general considerations on hardware learning.

In the *implementation of on-chip back-propagation* chapter, the first hardware learning system is described. It is based on a simple, but elegant, idea and was meant to be just a minor part of the thesis (my initial work was that of implementing RTRL). As it turned out, however, there was a lot of hard work involved in verifying this simple idea (work worth at least $1\frac{1}{2}$ years). The work has borne fruit, though, in several paper invitations.

The second hardware learning system is described in the *implementation of RTRL hardware* chapter. The system architecture of this work was developed during my studies for the Masters degree. As for the first system, there was a lot of hard work involved in the implementation and testing of the experimental system. A work that is not complete at the time of writing.

In the *thoughts on future analogue VLSI neural networks* chapter, I have collected some odds and ends of the field which did not fit into the other chapters: During my Ph.D. study, a lot of ideas have come to my mind on network architectures, subcircuits, learning algorithms, etc. Also, I have formed my own personal opinion on several matters. These thoughts are not all mature; however, I find it important to propagate this information to the scientific community in order that other scientists can benefit from the ideas.

Finally, in the last chapter, *conclusions*, the conclusions are drawn. In this section, I have tried to emphasize what is my own contributions to science (denoted by "*we*" or "*our*").

The *appendices* hold material that is of interest mostly to the meticulous reader.

The *enclosures* hold material that is of interest only to the reader who want to carry on my work; and they serves as documentation for my work.

Being organized in a project oriented way, many considerations are placed in the bulk of the text, where it is used, for readability rather than being organized in a logical manner. I hope the index will prove adequate for locating such considerations.

Lyngby
September 1994

Torsten Lehmann

Acknowledgements

I should like to thank the *analogue integrated electronics group* at the *Electronics Institute* for valuable discussions during this work; in particular, Gudmundur Bogason, Erik Bruun, Thomas Kaulberg, John Lansner and Peter Shah. Thanks to the members of CONNECT for discussions on neural networks; especially Lars Kai Hansen and Anders Krogh. Also, thanks to Ole Hansen of the Mikroelektronik Centeret who was always ready with answer and a helping hand. Thanks to Mogens Yndal Petersen of the Electronics Institute for the layout and soldering of numerous PCBs. Thanks to the DTU EUROCHIP staff who endured many questions and pushed deadlines during chip manufacturing.

Thanks to Peter A. Toft, John A. Lansner, Lars Kai Hansen, Thomas Kaulberg and Gudmundur Bogason for valuable criticism of the thesis.

Finally, thanks are due to the *Danish Technical Research Council*, the *Danish Natural Science Council* and *Analog Devices, Denmark* for financial support.

Contents

Abbreviations

AC	Alternating Current
A/D	Analogue/Digital
ADC	Analogue to Digital Converter
ANN	Artificial Neural Network
ASIC	Application Specific Integrated Circuit
BiCMOS	Bipolar/Complementary MOS integrated technology
BJT	Bipolar Junction Transistor
BPL	Back-Propagation Learning
CCII	Current Conveyor of second generation
CCO	Current Controlled Oscillator
CMOS	Complementary MOS integrated technology
CPS	Connections Per Second
CUPS	Connection Updates Per Second
D/A	Digital/Analogue
DAC	Digital to Analogue Converter
DC	Direct Current
DNA	DesoxyriboNucleic Acid
DSP	Digital Signal Processor
EEPROM	Electrical Erasable Programmable ROM
FLOPS	FLoating-point Operations Per Second
FSM	Finite State Machine
GCPS	Giga CPS
GCUPS	Giga CUPS
IPM	Inner Product Multiplier
IS	Input Scale
ISA	Industry Standard Architecture
LBM	Lateral Bipolar Mode
LSB	Least Significant Bit
MCPS	Mega CPS
MCUPS	Mega CUPS
MDAC	Multiplying DAC
MFLOPS	Mega FLOPS
MLP	Multi Layer Perceptron
MOS	Metal Oxide Semiconductor
MOSFET	MOS Field Effect Transistor

MOST	MOS Transistor (MOSFET)
MPC	Multi Project Chip
MPW	Multi Project Wafer
MRC	MOS Resistive Cell
mRNA	Messenger RNA
MSB	Most Significant Bit
MVM	Matrix-Vector Multiplier
NARV	Normalized Average Relative Variance
NLBP	Non-Linear Back-Propagation
NLRTRL	Non-Linear RTRL
NLSM	Non-linear Synapse Multiplier
OBD	Optimal Brain Damage
OR	Output Range
PC	Personal Computer
PC AT	IBM PC AT type compatible computer
PCB	Printed Circuit Board
PFM	Pulse Frequency Modulation
PLA	Programmable Logic Array
PSRR	Power Supply Rejection Ratio
PWM	Pulse Width Modulation
RANN	Recurrent Artificial Neural Network
RAM	Random Access Memory
RGC	Regulated Gain Cascode
RNA	RiboNucleic Acid
ROM	Read-Only Memory
RTRL	Real-Time Recurrent Learning
SAR	Successive Approximation Register
TTL	Transistor-Transistor Logic
UV	Ultra Violet
VLSI	Very Large Scale Integration
WSI	Wafer Scale Integration

Symbols

A list of the less commonly used *symbols* appearing in the thesis is given here. For standard symbols refer to the literature (eg. *Geiger et al.* [77], *Hertz et al.* [95] or *Sánchez-Sinencio and Lau* [206]).

$\star$ As index: index runs over all possible values.

$\otimes$ Vector multiplication by coordinates ($\underline{\xi} \otimes \underline{\zeta} = [\xi_1\zeta_1, \xi_2\zeta_2, \ldots]^{\mathrm{T}}$).

$*$ Convolution operator.

$\mathrm{b}\xi$ As Superscript: $\zeta^{\mathrm{b}\xi}$: bit number ξ of ζ.

B Number of bits; B_ξ is a B bit discretization/resolution of ξ. Also bandwidth.

$B_{\mathrm{M}\xi}$ Memory necessary to store ξ.

C_{ox} MOS gate oxide capacitance per unit area.

d_k Neuron target value.

D_ξ Non-linearity of ξ.

$D_{\mathrm{A}\xi}$ Accuracy of ξ.

E_{NARV} Normalized average relative variance error measure.

$g_k(s_k)$ Neuron activation function; also as vector of scalar functions, $\underline{g}(\underline{s})$.

i ANN weight (neuron) index, $i \in U$.

I Set of ANN input indices (m).

I_{sm} Maximal signal current.

j ANN input/output index, $j \in I \cup U$.

$\mathcal{J}$ Cost function (instantaneous), eg. quadratic ($\mathcal{J}_{\mathrm{Q}}$) or entropic ($\mathcal{J}_{\mathrm{E}}$).

$\mathcal{J}_{\mathrm{tot}}$ Total cost function.

k ANN neuron index, $k \in U$.

k' p_{ij}^k (index k) RTRL chip access variable.

K' MOST process transconductance parameter (μC_{ox}).

L Number of layers in ANN. Also MOST channel length.

LSB_B Sometimes used non unit-less for *bit absolute measures*.

l ANN layer number (often indexing as superscript), $l \in \{\underset{\text{input}}{1}, 2, \ldots, \underset{\text{output}}{L}\}$.

Δl_{ofs} Geometric device size offset (eg. on L).

m ANN input index, $m \in I$.

M Number of inputs in ANN.

M_k	Number of inputs to neuron k.
N	Number of neurons in ANN.
$N_{\mathcal{A}}$	Number of letters in input alphabet $\mathcal{A}$.
N_{epc}	Number of epochs ANN is trained.
$O(.)$	In most places implicitly "order of N": $O(f(N))$.
N_{E}	Number of networks in ANN ensemble.
N_l	Number of neurons in layer l of ANN.
ofs	As index: offset error.
p^k_{ij}	Neuron derivative variables for RTRL.
$p^k_{\mathrm{N}ij}$	Neuron derivative variables for non-linear RTRL.
res	As index: resolution.
R_{D}	Dynamic range.
s_k, s^l_k	Neuron net input; also as vector, $\underline{s}$.
t	Time. Often unit less.
$t_{\xi\,\mathrm{pd}}$	Propagation delay when calculating ξ.
T	Set of neuron indices (k) for which a target exist.
T_{cyc}	Learning cycle time.
T_{epc}	Training epoch; $t \in T_{\mathrm{epc}}$ when training.
T_k	Indicates if $k \in T$ ($T_k = 1 \iff k \in T$).
U	Set of neuron indices (k).
V_{sm}	Maximal signal voltage.
V_{t}	Thermal voltage: $V_{\mathrm{t}} = kT/q$.
V_{T}	MOSFET threshold voltage.
w_{ij}, w^l_{ij}	Connection strength from input/neuron j (layer $l-1$) to neuron i (layer l); also as matrix, $\underline{\underline{w}}$.
$\underline{\underline{w}}_{\square}$	$\underline{\underline{w}}$ without the columns corresponding to the inputs.
$\underline{w}_{\star\star}$	Connection strengths arranged as vector.
Δw_{ij}	Connection strength change.
Δw_{min}	Connection strength change threshold.
W	MOST channel width.
x_m	Input to ANN; also as vector, $\underline{x}$.
y_k, y^l_k	Neuron activation; also as vector, $\underline{y}$.
z_j, z^l_j	Neuron input: ANN input or neuron activation; also as vector, $\underline{z}$.
α_{FC}	BJT forward emitter-collector current gain.
α_{mtm}	Learning momentum parameter.
α_{N}	NLBP domain parameter.
β	MOSFET transconductance parameter: $\beta = W/L \cdot \mu C_{\mathrm{ox}}$.
β_{t}	Activation function steepness ($\beta_{\mathrm{t}} = \partial g(s)/\partial s\big\|_{s=0}$).
γ_{F}	Derivative or Fahlman perturbation.
δ^l_k	Weight strength error for back-propagation; also as vector, $\underline{\delta}^l$.

$\delta^l_{\mathrm{N}k}$	NLBP weight strength error.
δ_ξ	Droop rate of sampled signal (eg. weight drift in mV/s).
ε_k, ε^l_k	Neuron activation error; also as vector, $\underline{\varepsilon}$.
ϵ_{dec}	Weight decay parameter.
ζ	General quantity ("random" variable) or free running index.
η	ANN learning rate. Also neuron specific, η_k.
Θ_k	Neuron threshold.
Λ	(Weight) restoration efficiency.
μ	Carrier surface mobility.
ξ	General quantity ("random" variable) or free running index.
σ_I	Current mismatch: standard deviation compared to reference device.
σ^k_{ij}	Neuron net input derivative variables for RTRL.
ϕ_ξ	Clock phase ξ.

Generally, the rules below are obeyed, though deviations do appear. Not all different kinds of signals are distinguished by the rules; the context in which a symbol appear must supply this lack of information. The lack of a consistent, usable standard necessitates this unfortunate "definition".

For electrical signals, the case of the symbol indicates whether it is a *DC signal* or an *AC signal*. For the first case (bias voltages, quiescent currents, etc.) we use upper case letters, eg. I_{bias}; for the second (small signal quantities, instantaneous values, etc.) we use lower case letters, eg. i_{signal}.

The *font* of the subscript indicates if the subscript refer to another symbol (italics, eg. v_y) or if the subscript is descriptive (roman, eg. v_{out}).

The case of a descriptive index usually indicates whether this is an compound abbreviation (upper case, eg. v_{GS}) or not (lower case, eg. v_{ofs}).

List of figures

Chapter 1

Introduction

This thesis describes the analogue VLSI implementation of two supervised learning algorithms for artificial neural networks. One is the *error back-propagation* learning algorithm for a layered feed-forward network and the other is the *real-time recurrent learning* algorithm for a general recurrent network. Both operate in discrete time on a *cascadable analogue VLSI neural network* that has a digital random access backup memory for the weights. Also included in this thesis is the implementation of a cascadable analogue VLSI neural network as well as some general conceptions on hardware learning and thoughts on future analogue VLSI neural networks.

During the last decade or so, the field of *artificial neural networks* (ANNs, see appendix B) has matured. Artificial neural networks are no longer “magic devices” but powerful tools — when used in the right manner — for classification problems and similar tasks for which no algorithmic solution is known. The ANN foundation — both theoretically and on an application level — growing increasingly solid, hardware implementations (primarily analogue and digital VLSI, see appendix C) for high performance systems have begun to emerge. VLSI implementations of recall-mode ANNs are maturing and the field is ready for the step towards fully adaptive VLSI ANNs (ie. including learning). This is the objective of the present thesis: to study analogue VLSI implementations of computational neural networks, with the emphasis on learning hardware implementations. In this introduction motivations for using analogue VLSI to implement ANNs and learning algorithms are described and the objective of the present work is defined.

1.1 Implementing ANNs in analogue hardware

Why use analogue hardware? Artificial neural networks can easily be simulated on standard, digital von Neumann computers. These general purpose computers are rapidly moving into every imaginable part of our society. They are the subject of *very* intense research world-wide, and the competition among manufactures is *very* hard. The computational performance is growing exponentially over time. How can we possibly hope to compete with them? We can not. There are niches, though, where analogue integrated neural networks have the advantage. In this section we will examine these.

- *Parallelism*: In the ever present pursuit for faster data processing, two approaches are possible: One is to use faster systems (eg. higher clock frequencies). During the recent years, this procedure has been exploited to the utmost limit: interfacing to and communication with these very fast systems is ever more difficult (*Seitz* [213]). The other way to speed up data processing is to use parallel data processing elements. This is no trivial task, though, as many problems can not be parallelized (*Almasi and Gottlieb* [8], *Leighton* [148]). Neural networks are inherently parallel and are therefore easily mapped on parallel hardware. Further, though analogue data processing elements are inherently slower than their digital equivalents, the analogue versions of the neural network computing primitives, multiplication and addition, can be much smaller than their digital equivalents. Thus massively parallel neural systems are efficiently implemented using analogue VLSI, giving a potential for *very* fast data processing (eg. *Murray et al.* [175], *Ismail and Fiez* [106], *Graf and Jackel* [82]).

  This claim requires a couple of remarks: As the exact mapping on parallel hardware depends heavily on the network topology (which is application dependent), the massively parallel neural systems should be *application specific* rather than general purpose (*Lehmann* [142], *Jackel* [110], *Mead* [162]). Further, an analogue neural network should not be thought of as an accelerator for a von Neumann computer as using a serial computer to supply the data for a massively parallel neural network would, in most cases, severely limit the performance. Also note that, for very high precision computations digital circuits are required; however, it is generally believed that the precision offered by analogue components is sufficient in many neural systems (though, cf. the following chapters, *Edwards and Murray* [67], *Hollis et al.* [97], *Tarassenko et al.* [238]).

  Massively parallel analogue neural networks have been reported by *Arima et al.* [16], *Castro et al.* [42] and *Masa et al.* [157] among others.

- *Asynchronousness*: Many neural networks are asynchronous in nature. This asynchronousness can be efficiently exploited. Asynchronous (or *self timed*) systems have a number of distinct advantages over systems governed by a clock (*Seitz* [213], *Sutherland* [233], *Ramacher and Schildberg* [194], *Murray et al.* [175]):

  - o Synchronous systems must be designed to run at a conservative clock frequency to ensure functionality in worst case situations; asynchronous systems run at the maximum speed of the present hardware. Furthermore, if a certain component is the bottleneck of the system, this can be replaced by a faster component with immediate improvement of the overall performance.
  - o When increasing a system clock frequency, communication between components becomes a problem as it is very difficult to distribute the system clock (without skew) over a large area. Asynchronous communication is usually the solution.
  - o In synchronous systems all components change states (and thus draw current from the power supply) simultaneously at the clock edges. This puts very heavy demands on the tolerable power supply peak currents, capacitive decoupling, etc. The heavy peak currents also introduce a lot of noise. Asynchronous systems are power averaging.
  - o Real world interfacing is basically an asynchronous task; using asynchronous systems for this is the natural approach and eliminates, for instance, problems associated with *metastability* (*Seitz* [213], *Gabara et al.* [73]).

  In spite of the very attractive features of asynchronous systems, few "conventional" digital data processing systems have been successfully implemented because of the hardware overhead needed for handshaking (*Sparsø et al.* [227]). Pure analogue systems have no need for handshaking and are thus well suited to implement asynchronous systems. (Though in systems with feed-back the stability must be considered.)

  Asynchronously operated analogue neural networks have been reported by *Alspector et al.* [9], *Hollis and Paulos* [98] and *Mead* [162] among others.

- *Fault tolerance*: A well trained ANN is insensitive to small weight changes as $\partial\mathcal{J}/\partial w_{kj} \approx 0$ at the equilibrium (where $\mathcal{J}$ is the error cost and w_{kj} is a weight, cf. (36^{B})). However it is *not* insensitive to the complete loss of a connection (due to a short circuit, a RAM fault, radiation, etc.) as the network must have as simple an architecture as possible to ensure good generalization ability. To ensure fault tolerance, even down to the hardware level, it is necessary to introduce redundant hardware. This is in favour of hardware implementations of neural networks; in particular analogue hardware as the cost of an extra synapse is relative low. In this context, a new emerging technology, *wafer scale integration* (WSI), deserves mentioning, as fault tolerant systems are crucial to the applicability of WSI. This technology is well tailored to the implementation of massively parallel neural networks (*Yasunaga et al.* [273]).

- *Low power applications*: The use of sub-threshold operated MOSFETs offer the possibility of extremely low power systems. Though digital systems can function in sub-threshold, analogue systems carry more information per wire and fewer transistors per operation and thus inherently use less power (cf. eg. *Ismail and Fiez* [106]).

  Low power analogue neural networks have been reported by *Leong and*

Jabri [149] and *Mead* [162] among others.

- *Real-world interfacing*: As well as being asynchronous, real-world interfaces are often required to be analogue. Analogue neural networks obviously eliminate the need for A/D and D/A converters, which is an attractive feature. However, this becomes of paramount importance when the data is applied in massive parallelism. The use of hundreds or thousands of high speed A/D converters would seldom be justified. For low power systems with real-world interfaces it is also of great importance that no power is wasted in the processes of A/D and D/A converting.

  Analogue neural networks with real-world interfaces have been reported by *Leong and Jabri* [149], *Masa et al.* [157] and *Mead* [162] among others.
- *Regularity*: The regularity of artificial neural networks makes them well suited for massively parallel implementations: The design effort can be put in designing a few efficient components which are used repeatedly and interconnected in a regular way.

To conclude: At least two niches for analogue integrated neural systems exist, both possibly asynchronous or with redundant hardware:

- Massively parallel, application specific systems having a parallel real-world interface.
- Small, low power, application specific systems with a real-world interface.

In this work we will be predominantly interested in the massively parallel analogue neural networks rather than the low power ones.

Today, several applications using analogue integrated neural networks have been reported. For instance high energy particle-detector track-reconstruction devices (*Masa et al.* [157]), implantable heart cardioverters/defibrillators (*Leong and Jabri* [149]) and silicon cochleas, retinas and motion sensors (*Mead* [162], *Park et al.* [185], *Cao et al.* [38]).

Though most systems reported embrace the principles above, general purpose analogue neural systems have been reported (*Mueller et al.* [173], *Van der Spiegel et al.* [228], *Satyanarayana et al.* [207]). Systems that do embrace the above principles can be found in *Bibyk and Ismail* [23, 24], *Bruun et al.* [32], *Corso et al.* [53], *Eberhardt et al.* [65], *Hollis and Paulos* [98], *Kub et al.* [126], *Lansner and Lehmann* [130, 131], *Linares-Barranco et al.* [151], *Mead* [162], *Moon et al.* [169], *Murray et al.* [175, 176], *Neugebauer and Yariv* [180] and *Ramacher and Rückert* [193].

1.2 Implementing learning algorithms in analogue hardware

Because of the non-ideal characteristics, analogue integrated neural networks are most often taught using *chip-in-the-loop training* (*Castro et al.* [42], *Eberhardt et al.* [66]) — that is, rather than down loading predetermined weights for a given application, each individual chip (or system) is trained by (i) applying an input pattern to the chip, (ii) compute the network error on the basis of the target values and the actual chip outputs, and (iii) adjust the weights on the chip according to the learning algorithm such that the network error decreases. This can accommodate for offset errors, non-linearities, etc.

There is a wealth of different training approaches which quite easily can be programmed on a host computer for chip-in-the-loop training. The question now arising is: Why should we sacrifice the flexibility of chip-in-the-loop training using a host computer in favour of implementing learning algorithms in analogue hardware? The reasons are similar to the reasons for implementing ANNs in analogue hardware in the first place:

Performing similar operations on all synapses/neurons of a regular system composed of synapses and neurons, many learning algorithms have the same properties as neural networks when implemented in analogue hardware:

- *Parallelism*: Learning is computationally a very heavy task. Typically of the order $O(N^4)$ or $O(N^6)$ in a system with N neurons. (Compare to the recall-mode task which is of the order $O(N^2)$, assuming the system has $O(N^2)$ synapses.) In terms of speed, it is therefore of even greater importance to utilize inherent parallelism for the learning algorithm than for the neural network itself. Fortunately many learning algorithms can be parallelized to a great extent. The arguments for using analogue hardware to utilize the parallelism are the same as above.

  Massively parallel implementations of learning algorithms have been reported by *Arima et al.* [16] among others.
- *Adaptability*: Adaptive neural systems are continuously taught while being used. At least two situations exist where the learning algorithm must be embedded in the system hardware: (i) In large adaptive systems, where it is crucial to utilize the inherent parallelism of the learning algorithm. (ii) In small, adaptive, low-power systems, where a host computer is not available. In these two application areas it is likely that the neural network is analogue — and the arguments that advocated the use of analogue hardware for the neural network holds for the implementation of the learning algorithm too.
- *Asynchronousness*: Many learning algorithms can be formulated in continuous time, which enables asynchronous, analogue hardware implementations of learning algorithms. An asynchronous, analogue neural network with asynchronous, analogue interfaces is most naturally taught by such a learning algorithm.
- *Fault tolerance*: Some learning algorithms can be formulated to operate on local data in such a way that the learning algorithm can be embedded in the

(extended) synapse and neuron hardware. In a fault tolerant analogue neural network, the inclusion of such a learning algorithms does not sacrifice the fault tolerance for the system as a whole.

- *Low power applications*: As is the case for the analogue neural networks, analogue implementations of learning algorithms can use MOSFETs operated in sub-threshold for extremely low-power applications.
- *Data conversion*: The learning algorithm needs access to inputs, outputs and intermediate variable of the neural network. If the neural network is analogue, the use of analogue hardware for the learning algorithm eliminates the need of A/D and D/A converters (cf. above).
- *Regularity*: As is the case for the neural networks, many learning algorithms are regular in structure; thus the design of the equivalent hardware is inexpensive.

Combining these properties with the conclusion in section 1.1, we conclude that at least two niches for analogue hardware implementations of learning algorithms exist, both possibly asynchronous or with redundant hardware:

- Massively parallel, possibly adaptive, application specific systems having a parallel real-world interface.
- Small, adaptive, low power, application specific systems with a real-world interface.

As for the neural networks, it is believed in certain circles that the limited precision of analogue hardware is sufficient for the implementation of (certain) learning algorithms, because of the present feed-back. Some even argue that limiting effects as noise (*Murray and Edwards* [177]) can improve learning ability. Certain offset errors, however, can be completely destructive for the learning scheme (*Montalvo et al.* [168], *Lehmann* [139, 140]), and it is not yet generally accepted whether or not this prohibits analogue implementations of learning algorithms (though a few systems have been reported, eg. *Alspector et al.* [9], *Shima et al.* [220] and *Valle et al.* [251]). Thus research on implementing analogue learning hardware still has a wealth of unexplored areas. This is to what we will commit the following chapters.

Architectures for analogue hardware learning can be found in *Alspector et al.* [9], *Arima et al.* [16], *Card* [39], *Caviglia et al.* [44], *Hollis et al.* [99], *Jabri and Flower* [107], *Lehmann* [139, 141, 142], *Linares-Barranco et al.* [151], *Macq et al.* [154], *Matsumoto and Koga* [160], *Montalvo et al.* [168], *Murray* [174], *Reyneri and Filippi* [196], *Schneider and Card* [210], *Shima et al.* [220], *Tarassenko and Tombs* [237], *Valle et al.* [251], *Wang* [256] and *Woodburn et al.* [270].

Chapter 2

Implementation of the neural network

Rather than implementing a learning system all at once, we have chosen a sequential approach, first implementing an acting neural system, and second implementing learning hardware for this system. In this chapter the implementation of the artificial neural network that is the core of the systems in this thesis will be presented. The chapter includes reflections on the choice of network models and topologies suitable for an analogue VLSI implementation. Also, the choices of hardware topologies and essential subcircuits (memories, multipliers and thresholding hardware) are discussed — with the future implementation of hardware learning in mind. Several examples from the literature are given. After the presentation of the design of and measurements on our cascadable ANN solution, and after the presentation of system level measurements, reflections on future work are given: the inclusion of process parameter dependency canceling and temperature compensation. A summary concludes the chapter.

2.1 The artificial neural network model

As the very first thing, we must decide on a model for our artificial neural network. There are three properties that this model must possess; it must be:

- General purpose
- Simple
- Suitable for the technology

It was argued in chapter 1 that analogue ANNs have to be application specific. With no particular application in mind at this point, we shall deviate slightly from this principle without actually violating it: the object is to design a set of general purpose *building block components* or *modules* (*Eberhardt et al.* [65], see also *Mueller et al.* [173]). Application specific *systems* can then be composed of a number of these.

Analogue computational hardware is typically limited to a relative precision of about 1% (eg. *O'Leary* [182], see appendix C). For this reason, and for the reason of limiting the hardware cost, it is preferable to use a *simple ANN model.* Some researchers try to model the biological mechanisms of neural networks very closely (*Grillner et al.* [86], see also *MacGregor* [153]) or use other complicated network models. This should not be necessary for computational neural networks as many of the properties of these are owing to the structure and non-linearity of the system.

It is of paramount importance that our ANN model is compatible with the restrictions imposed by the analogue VLSI technology (eg. *Mead* [162]). Otherwise the advantages of using the technology in the first place would be lost. It must be absolutely clear that we thus can not justify the implementation of an arbitrary model (just as we can not justify the use of analogue integrated ANNs for an arbitrary application as argued in the previous chapter); the *model must be easy to map on the hardware* in terms of both topology and computation primitives.

Our first objective will be to implement an acting system — which can be refined later if necessary.

2.1.1 The neurons

Using *stochastic neurons* gives the possibility of implementing very powerful networks such as *Boltzmann machines* (*Hertz et al.* [95]). The *activation*, y_k, of the stochastic neuron k, typically of value 1 or -1, is probabilistically determined:

$$\Pr(y_k = 1) = 1 - \Pr(y_k = -1) = g_k(s_k)\,,$$

where s_k is the *neuron k net input* and $g_k(.)$ is the *activation function* (cf. appendix B). Stochastic systems can very efficiently explore the state space of the system's free parameters during a learning process. They are somewhat slow, however, as the outputs must be averaged over time to find the probability distributions of the outputs, and the stochastic processes are not very well suited for analogue signal processing (though see *Alspector et al.* [9] for a VLSI implementation that come around these problem using different kinds of *annealing* processes).

A very general, deterministic, network model uses *higher order neurons* (*Wulff* [271], *Giles et al.* [79]):

$$\begin{aligned} y_k &= g_k(s_k) \\ &= g_k\left(\sum_j w_{kj} z_j + \sum_{j_1 \le j_2} w'_{kj_1 j_2} z_{j_1} z_{j_2} + \sum_{j_1 \le j_2 \le j_3} w''_{kj_1 j_2 j_3} z_{j_1} z_{j_2} z_{j_3} + \cdots \right) , \end{aligned} \tag{1^2}$$

where the w_{kj}s, $w'_{kj_1 j_2}$s, $w''_{kj_1 j_2 j_3}$s, ... are the *connection strengths* and the z_{j_ξ}s are the *neuron k inputs*† (cf. appendix B). The highest number of z_{j_ξ}-factors gives the order of the neuron. Though higher order networks can be very efficient compared to "conventional" (ie. first order) networks, they map poorly on VLSI because of their high structural dimensionality (a Dth order network has a $D+1$-dimensional structure). Thus, using first order deterministic neurons is preferable from a VLSI implementation point of view. This is also theoretically the most well-studied one which is also significant.

Finally there is the choice of the neuron *transfer function*. The simplest possible choice would be setting $g_k(.) \equiv \mathrm{sign}(.)$ (a *hard limiter*) which is used in *Hopfield networks* and which is well suited for an analogue VLSI implementation (*Hollis and Paulos* [98], *Sánchez-Sinencio and Lau* [206]). This, however, would sacrifice the generality of the system (obviously continuous valued outputs would be impossible; also, many learning algorithms rely on a smooth transition from "low" to "high" neuron output). The choice, therefore, is to set $g_k(.) \equiv g(.)$, where $g(.)$ is a sigmoid-like function, which is a sufficiently general solution (networks using this kind of neurons can approximate any limited function (*Lapedes and Farber* [134])).

2.1.2 The network

Ideally we should put no constraints on the *network topology*. However, as we shall see in the following, sparse interconnections between neurons will be difficult to implement in general. Thus the choice is to use fully interconnected (groups of) neurons which is the most general topology.

Using an unconstrained topology imposes another problem if *feedback* is present: instability. An unknown number of neurons in a feedback loop would cause an unknown phase shift at high frequencies and might lead to oscillations. The problem is further complicated by the fact that signs and magnitude of the gains (weights) in the system change during learning. A solution is to place the feedback as shown in figure 74^{B} and ensuring a single dominating pole in the loop (refer to *Hollis and Paulos* [98], *Linares-Barranco et al.* [151], *Graf and Jackel* [82], and

† z_{j_ξ} can be either a *network input*, x_{j_ξ}, or the output *activation* from another *neuron*, y_{j_ξ} (see sections 4.1.1 and 5.1.1).

Mueller et al. [173], eg., for such systems). In *non-relaxation systems*† the network time constants should — in some way — match the time constants of the input data and of possible learning hardware. In this case it is easier to use a *discrete time feedback* (ie. a sampler) in a general system, though the asynchronousness is sacrificed. Even using discrete time systems, there is still a wealth of problems to which we can apply analogue ANNs, and most of the systems in this thesis will be designed to work in discrete time.

2.2 Mapping the algorithm on VLSI

Before presenting our analogue integrated ANN solution, we shall have a look at different aspects of such implementations. More specifically we shall discuss different architectures, signalling methods, memories, multiplication and thresholding circuits — with the future implementation of learning algorithms in mind.

2.2.1 Architecture

The architecture of a *small, low power, application specific system* (cf. chapter 1) must be tailored to the application. The *building block components* for such systems are thus the atomic parts of neural networks — synapses and neurons — and would have the form of, say, a cell library to a CMOS process. Though the discussions in the present thesis are meant for massively parallel systems, many of the considerations apply equally well for small, low power systems. Only, the circuits should be replaced by low power ones. We shall use *strong inversion circuits* rather than subthreshold ones as the former are inherently faster than the latter.

For *massively parallel, application specific systems*, the level of integration of the *building block components* is preferably very high (this reduces design time). Unfortunately, this puts constraints on the architecture and minimizing these constraints is one of the objects of *VLSI neural networks* design.

Reformulating (1^2) for a vector of first order neurons (cf. (30^{B})) — corresponding to, say, a layer — we have:

$$\underline{y} = \underline{g}(\underline{s}), \qquad \text{where} \quad \underline{s} = \underline{\underline{w}}\,\underline{z}\,, \tag{2^2}$$

where $\underline{g}(.)$ is a vector of sigmoid-like functions. Assuming we have a parallel operated *matrix-vector multiplier* (*MVM*) that gives as output the multiplication of its input vector and a stored matrix, the number of rows in the multiplier is increased

† Systems where the network is *not* allowed to settle to a steady state before the next input pattern is applied (*Williams and Zipser* [267] and others); these are used in time sequence data processing.

simply by adding another multiplier with the same input vector. The number of columns is increased by adding the output vector to that of another multiplier. As dimensions are easily added to a vector of functions, the implementation of an ANN that is *fully interconnected* (between layers, if layered) and which can be scaled to an arbitrary size, is feasible using two building block components (*Eberhardt et al.* [65], *Lansner and Lehmann* [131], *Shima et al.* [220]). This is shown in figure 1^2; it is assumed that adding the outputs from several multipliers is done simply by connecting their outputs together (cf. next section). This *cascadability* is *most* important. We shall use the terms *synapse chip* (the multiplier) and *neuron chip* (the squashing functions) for the two modules. Further, we shall refer to the rows/columns of $\underline{\underline{w}}$ as *rows of synapses* and *columns of synapses.*

For a recurrent network, an elegant approach is to place synapses on the neuron chip as illustrated in figure 2^2 (here: $\underline{y} = \underline{g}(\underline{\underline{w}}\underline{y})$) (*Duong et al.* [63]). This makes module interconnection easier.

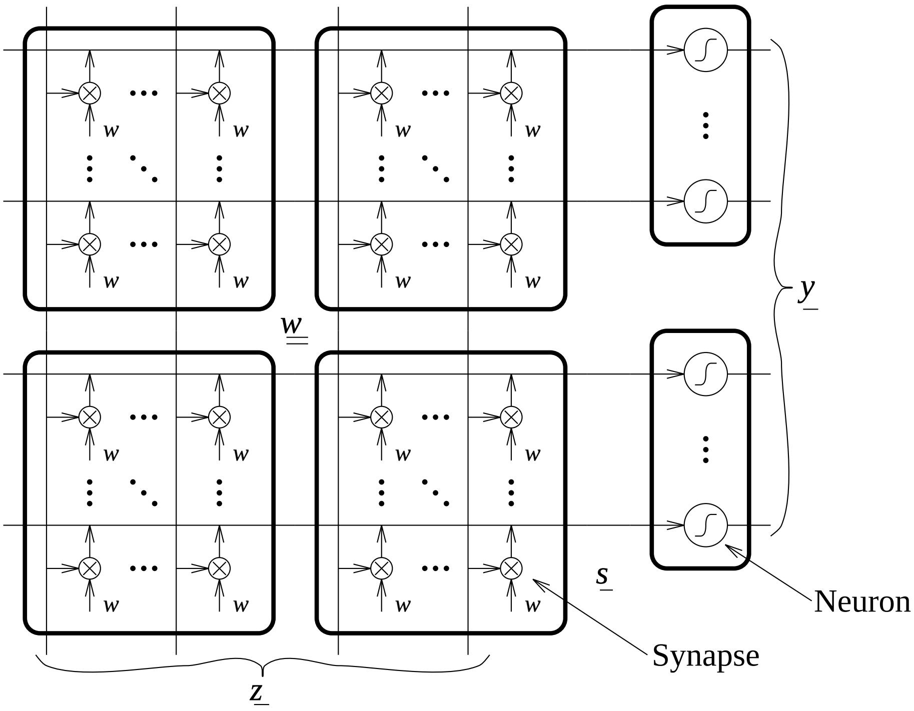


Figure 1^2: Expandable neural network. *This topology can implement systems of arbitrary size, fully connected between the layers.*

One could expect *routing* problems in systems with rigorously interconnected units. The distributed and regular placement of the synapses in the above systems, however, practically eliminates this problem (massive inter-chip communication is still inconvenient, though). For sparse, random connectivity routing would consume considerably more area per synapse.

Obviously, any first order ANN topology can be mapped on one of the above systems by setting some of the connection strengths equal to zero (feedback and extra layers can be added in figure 1^2). If the system is known to have sparse

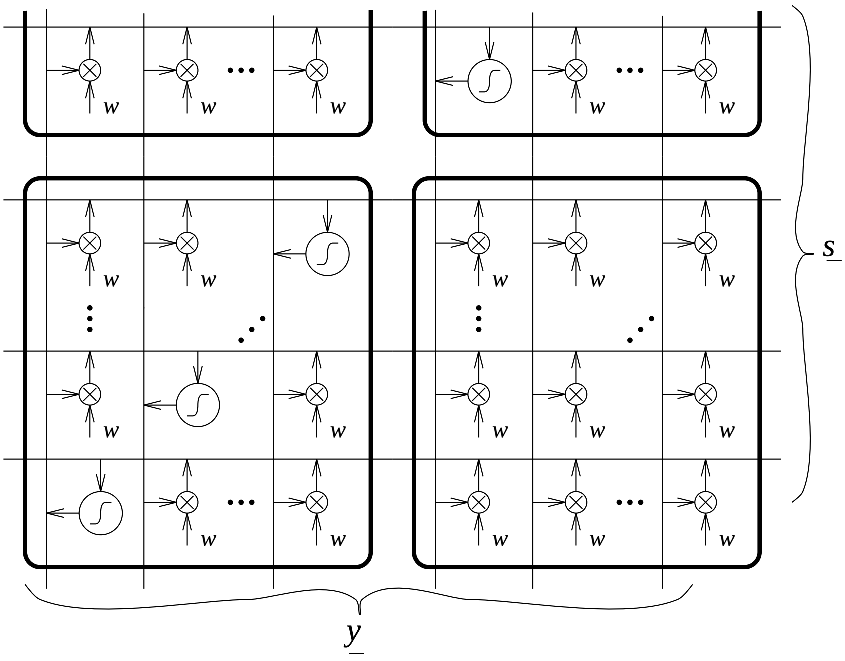


Figure 2^2: Expandable recurrent neural network. *This topology can also implement systems of arbitrary size. The neurons must not be larger than the synapses in order not to waste area.*

connectivity, though, it would be preferable not to waste hardware for all the null-connections. This could be accomplished by "folding" the synapse matrix, in a way similar to the folding of sparse PLAs, if the structure of the network is known in advance (*Bruun et al.* [32]). Often it is not, however; and certainly not when implementing general neural architectures.

Solving a problem with unknown properties, one would typically arrive at the sparse architecture by *pruning* (ie. removing unnecessary connections) a fully connected network, eg. using *optimal brain damage* (*OBD*) (*Le Cun et al.* [55], see also *Larsen* [135]). Thus preferably, a reconfigurable neural network should be able to emulate a fully connected one during the pre-pruning phase. As, depending on how it is used, OBD can remove as few as 50%–75% of the connections, and as simple synapses can be very small, care should be taken that interconnections and routing switches does not take up more area than left free by the reduced number of synapses. Another way to avoid "wasting" hardware in a pruned network would be to use the "null-connections" of a fully connected architecture to introduce redundancy in the system.

We believe that the fully connected "building block" topology of figure 1^2 is, though simple, a very capable one. We shall use this in the present work.

A number of systems with *reconfigurable network topologies* have been proposed in the literature (*Mueller et al.* [173], *Satyanarayana et al.* [207], *Graf and Henderson* [84], and others). Though it can be questioned if a "random" connected neural network can be mapped efficiently on these systems, they do provide a general problem-solving environment. Also, the reconfigurability can be used to alter the ANN topology during training and to map out defective blocks.

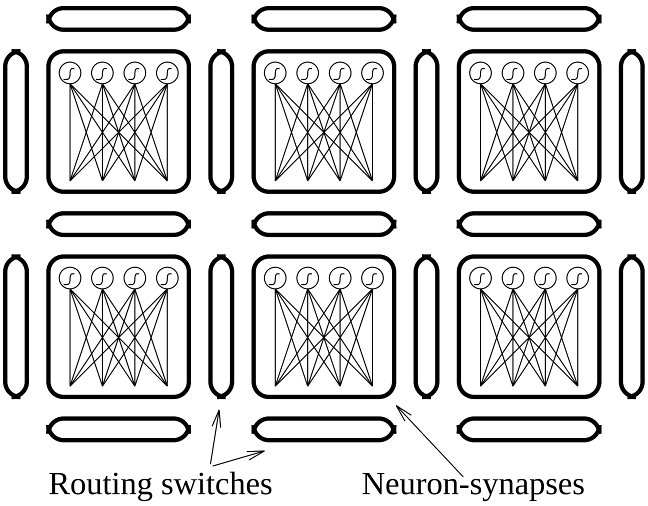


Figure 3[2]: Reconfigurable neural network. *The philosophy of this kind of topology is to implement a general neural computer.*

A particularly interesting reconfigurable ANN is found in *Satyanarayana et al.* [207], see figure 3[2] and 14[2], p. 27. In this implementation the "lumped" synapses and neurons above are replaced with "*distributed neuron-synapses*": The neuron squashing circuit is distributed among the connected synapses and can be connected in parallel with other neuron-synapses, ensuring that the routing switch area is kept reasonably low, as indicated in figure 3[2].

2.2.2 Signalling

The *domains* in which the various signals are carried are closely related to the needs of the matrix-multiplier above — or the needs of a synapse:

- The output from a neuron (or a network input) must easily be distributed to a column of synapses.
- The outputs from a row of synapses must easily be accumulated.

Distributing a signal is most easily done using a voltage as this can be detected using high impedance sensors in parallel (ie. MOS gates). In the current domain the addition of analogue signals is simply done by connecting the input wires to the output wire. Thus, using synapses with voltage inputs and current outputs satisfies the above requirements — which is fortunate as multipliers typically have voltage inputs and current outputs. This is illustrated in figure 4[2]. A variation of the current output scheme is to use charge packages which can be accumulated on an integrator.

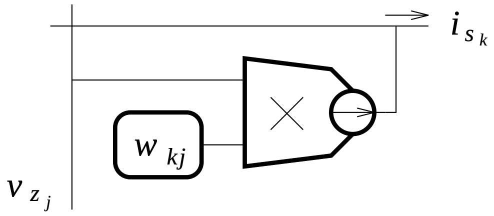


Figure 4^2: Typical electronic synapse. *The multiplier has voltage inputs and current output to ensure the cascadability of synapses.*

Analogue signals carried in the voltage/current domains are sensitive to noise; for instance coupled via the power supply or capacitive/inductive parasitics. In a *pulse stream neural network*, the noise sensitivity of the neuron outputs is efficiently reduced by moving the information from the voltage domain to the time domain — for instance using *pulse frequency modulation* (*PFM*) or *pulse width modulation* (*PWM*) (*Murray et al.* [175, 176]): A digital voltage signal can be easily distributed and regenerated, and the temporal information is insensitive to most noise sources. The noise sensitivity of the synapse outputs is not so easily reduced because of the requirement for easy accumulation. The synapse outputs would thus typically be charge packages (the connections strengths multiplied by the stream of input pulses). To get the full advantage of the noise insensitive neuron outputs it is therefore important that the synapse-to-neuron connections are kept at a minimum, local area. That is, only neuron outputs should be used for inter-chip communication.

The disadvantages of pulsed neural networks is a reduction in speed: Given a bandwidth B of our system, we can process $2B$ data points (*Tugal and Tugal* [250]) in a pure analogue system† whereas only $2B/100$ in a PFM neural network with a dynamic range of 40 dB.

The above are the most commonly used signalling methods in integrated neural network contexts, though other methods exist (*Neugebauer and Yariv* [180], *Murray et al.* [175], *Mead* [162], *Webb* [259], *Mortara and Vittoz* [171]). We shall use continuous valued signalling in the voltage and current domains in this work as this is inherently the fastest signalling method compatible with a simple synapse architecture.

As seen in figure 4^2, the typical electronic synapse consists of two components: a

† That is the fundamental Nyquist upper limit, assuming we use sinc ($\mathrm{sinc}(x) \stackrel{\mathrm{def}}{=} \sin(x)/x$) pulses in a linear system. A more realistic measure would be, for instance, B data points (per second): assuming the system has a single dominating pole at the frequency $f_{3\mathrm{dB}}$, the output corresponding to a step input would settle to 8 bit accuracy within the time $1/f_{3\mathrm{dB}} = 1/B$ (*Lehmann* [140]); ie. we could process B data points.

multiplier and a connection strength memory cell. As the number of synapses in an ANN (mostly) scale as $O(N^2)$, where N is the number of neurons, reducing the synapse area has been one of the major objects of integrated neural network research. Thus a discussion of memory cells and multipliers are the subject of the following two sections.

2.2.3 Memories

Storing analogue signals are by no means simple; no true, efficient analogue electronic memory exists today. Thus, the storage of the synaptic strengths is a major concern in analogue ANNs research; the solutions found in the literature are compromises of one kind or another, most of which can be put in one of the following categories:

- Capacitive storage
- Storage using special process facilities
- Digital storage

Capacitive storage The simplest method for storing an analogue signal is to put a charge on a capacitor and reading this using the *very* high impedance gate terminal of a MOSFET (*Tsividis and Satyanarayana* [246]). The drawback of this method is that the leakage current (primarily) through the sampling switch (or some other weight changing device) eventually exhausts the weight. Several approaches to reduce the leakage current are possible. For instance using a differential scheme as shown in figure 5^2, which cancels the predominant source-bulk reverse biased junction current. (This scheme also cancels the offset error due to charge injection.) Alternatively, using a low offset voltage buffer, one can ensure a 0 V voltage drop across the source-bulk diode, efficiently eliminating this leakage (see *Shah* [217], *Vittoz et al.* [254], *Horio and Nakamura* [101]). Whatever method employed, though, weight decay can not totally be eliminated and some kind of refresh is necessary.

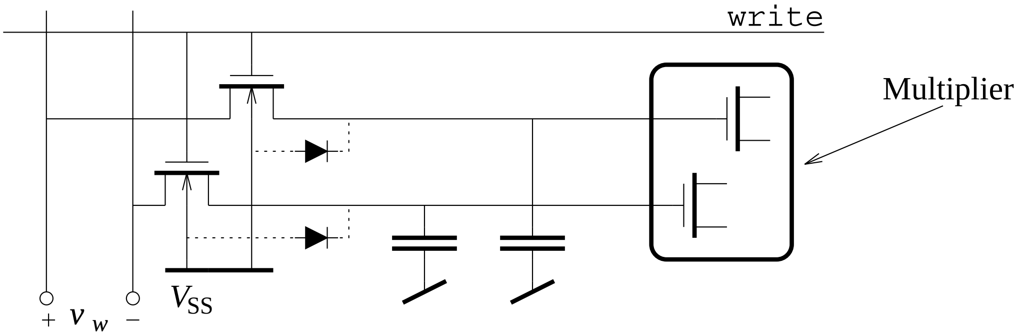


Figure 5^2: Capacitive storage. *Several approaches to refreshing the charge and reducing leakage (as the differential scheme shown) are possible.*

Most *refreshing schemes* rely on *quantizing the weights* and using these discrete valued weights to recharge the weight capacitors. One of the more obvious approaches to do so is to use a digital *RAM backup memory*: The weights are stored digitally in the RAM and the capacitors are periodically refreshed via a D/A converter (*Lansner and Lehmann* [131], *Eberhardt et al.* [65], *Jackson et al.* [111]). One would typically have serial access to the weight capacitors as well as to the words in the RAM; thus a count of $O(1)$ D/A converters would be needed†. As digital RAM is very cheap, this is an accountable solution (for large systems!). The serial access to the updating of weights is a severe limitation for a system with hardware learning: It will necessarily be an order $O(N^2)$ slower than a system with parallel weight access. However, the much discussed issue of too coarse weight discretizing during learning (see eg. *Hollis et al.* [97], *Tarassenko et al.* [238], and the following chapters) is less pronounced using this architecture than most others: first, as only $O(1)$ D/A and A/D converters are needed, it is accountable to use high precision converters; second, the RAM backup can have words of arbitrary widths (in number of bits) and thus very small weight changes can be accumulated (cf. the following chapters).

It is also possible to employ a "*quantize-regenerate*" refreshing scheme: The voltage on the weight capacitor is periodically compared to a discrete number of reference voltages (eg. in the form of a staircase ramp) and the capacitor voltage is "regenerated" to the closest reference (*Vittoz et al.* [254], *Björk and Mattisson* [25], *Horio and Nakamura* [101]). For very high precision weights (compared to the weight droop rate), it is necessary to place the regeneration circuit at the synapse sites to allow parallel weight refresh. For lower precision weights, a column, say, can share this circuit, but a voltage buffer must be placed at the synapse site to drive the capacitance on the wire connecting the column to the regeneration circuit. In either case, the quantize-regenerate technique is more area demanding than simple capacitive storage with a RAM backup. For systems with on-chip learning, the quantize-regenerate refreshing scheme is not particularly well suited because of the required high resolution of the weights — unless the learning scheme is so fast that several weight updates can be accumulated between successive weight refreshes (compare to the following "analogue adjustment").

An altogether different approach to refreshing relies on the presence of a learning scheme: refresh by *relearning*: During an idle phase of the neural network it is trained using an epoch, say, of the original training data, thus restoring the weights (cf. eg. *Valle et al.* [251], *Woodburn et al.* [270]; see also *Arima et al.* [16]). The obvious disadvantages with this approach are (i) that the network can not run continuously and (ii) that the whole training set needs to be stored in the system (though see chapter 6.3). If the training scheme employed is an *unsupervised learning* scheme, refresh by relearning can be employed in a more elegant way: Learning can be applied on each input pattern, eliminating the need for an idle phase and the storing of the training data (*Schneider and Card* [210]). (Such weight refresh is also applicable if *learning with a critic* is employed: a *reinforce-*

† As before: for a system with N neurons.

ment signal can usually be extracted from the environment of an acting system at a minimum cost (*Alstrøm* [10]).) The trouble with this approach is that the network will tend to forget the classification of scarcely occurring input patterns. Whether this is acceptable, or indeed an advantage, is strongly dependent on the application.

In most situations it is necessary to be able to read the contents of the weight matrix; for backup purposes, for example, or for transferring the network state to another network (retraining would be necessary). It is possible to do so without direct weight access if the outputs from the matrix-vector multiplier are accessible: Applying $z_\xi = \delta_{\xi j}$ † as inputs to the matrix-vector multiplier yields as outputs $s_i = w_{ij} + w_{ij\,\mathrm{ofs}}$, where $w_{ij\,\mathrm{ofs}}$ is the MVM output offset error (first order approximation) which would have to be canceled‡.

Special process facility storage *Non-volatile analogue memories* can overcome the leakage problems of capacitive storage. The most popular of these is *floating gate storage* where a charge is trapped on the completely insulated (floating) gate of a MOSFET, thus programming the threshold voltage (cf. *Sze* [235]), see figure 6^2 (*Horio and Nakamura* [101], *Vittoz et al.* [254]; compare to figure 77^{C}). (The MOSFET would be the input transistor(s) of the multiplier in figure 5^2.) There exists numerous ways of trapping the charge on the floating gate; some compatible with standard CMOS processes, other requiring special process steps as those in EEPROM processes, for example. The programming is usually carried out by (i) applying a high voltage across the gate oxide, thereby forcing a tunneling current to charge the gate (*Carley* [41], *Lee et al.* [137]) or (ii) exposing the gate to UV light, thereby inducing a parallel conductance caused by the generation of holes/electron pairs in the oxide (*Benson and Kerns* [22], *Abusland and Lande* [5]).

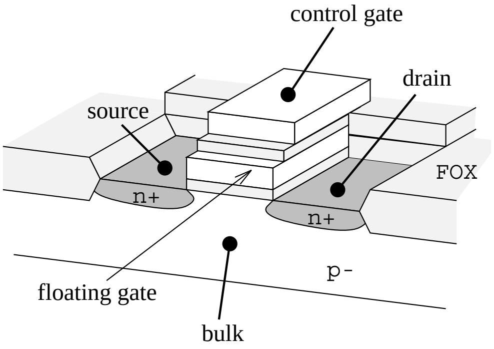


Figure 6^2: Floating gate MOSFET. *Schematic drawing of physical floating gate MOST. The control-gate/ floating-gate overlap need not, as in the figure, be on top of the channel.*

A completely different approach is *amorphous silicon storage* (*Reeder et al.* [195]). Here the resistance of a vanadium-amorphous silicon-chromium sandwich can be programmed by applying high-voltage pulses — much like the way floating gate MOSFETs are programmed electrically.

† Kronecker's delta: $\delta_{\xi j} = \begin{cases} 1, & \text{for } \xi = j \\ 0, & \text{for } \xi \neq j \end{cases}.$

‡ Note that, unless the offset error is very small, such a readout would not be accurate enough for a learning rule (as (9^4)).

Though these, often quite small, *special process facility memories* are the only true analogue memories existing today, several matters weigh against their use in analogue VLSI neural networks:

The writing on these analogue memories usually wear the devices. A typical floating gate device can endure in the order of 10000 full scale changes, for example. This is sufficient for programmable recall-mode systems (as in *Castro et al.* [42]), but for adaptive systems it is not: Though weight changes can be accumulated on a short term memory to reduce the number of device programmings, such systems are in general *taught by example* (*on-line learning* rather than *batch learning*) and continuously over time. Thus, the number of weight changes scale as $O(N^2t)$ (at the least), quickly exhausting the endurable number of device programmings.

The *driving force of VLSI* processes is digital electronics (primarily RAM, microprocessors, etc.). Thus a state of the art process will be tuned to digital requirement (eg. a 5 V, 0.7 μm, single poly, triple metal, n-well CMOS process without precision components). To get access to state of the art processes, the analogue circuit designer must submit to the potentials offered by digital processes. Further, some argue that special analogue devices (high resistive polysilicon, floating capacitors, precision components — even BiCMOS processes) will eventually cease to be available to the average designer because of the future role of most analogue circuits: interfaces to *digital signal processors* (*DSP*s) on *mixed analogue digital integrated circuits* (*NEAR* [3], see also *Tsividis* [247]). For this reason, one should have a *very* good reason to use special process steps for analogue circuits — especially in *VLSI* analogue circuits, as these would be exceedingly expensive. In this context, it is important to note that in most systems minimizing the *synapse cost* and power dissipation rather than the *synapse area* is the objective!

Even non-volatile analogue memories compatible with standard CMOS processes should be used with caution: They rely on un-documented features of the process which (i) must be characterized experimentally by the designer, (ii) would probably be subjects to immense process variations, and (iii) could possibly be changed without notice by the vendor. One very important characteristic, for instance, is the *memory life time* which one should have a fairly good idea of before considering a production (even floating gate devices do degrade; though on a time scale of years rather than seconds as for capacitive storage). Further, the need for high voltages or UV light for programming is inconvenient. (See also *Murray et al.* [176].)

Digital storage The problems with weight degrading, weight wearing and special processing steps can be overcome if one is willing to refrain from using analogue storage.

Unless very high resolution synapse strengths are needed, digital memories consume more area than simple analogue memories: The size of a digital memory scale as $O(\log w_{\text{res}})$ whereas analogue scale as $O(w_{\text{res}})$, where w_{res} is the *resolution* of the synapse strength (limited by noise; cf. eg. *Geiger et al.* [77]). For typical ANN system which require a weight resolution of 8–16 bit, the analogue solution is usually the smallest by far.

The most severe problem with embedding digital circuitry in an analogue system is the need for data converters; the area of such (monotonous) converters typically scale as $O(w_{\mathrm{res}}^2)$ (cf. *Pelgrom et al.* [187], *Lakshmikumar et al.* [129] and *Geiger et al.* [77]). Using digital synapse strength storage require a *digital to analogue converter* (*DAC*) at each synapse site, and though using a multiplying DAC eliminates the need for the synapse multiplier, this will be the most area consuming part of the synapse.

In spite of the area penalties of digital weight storage, there are several application areas where such is very useful; especially in small systems that can not tolerate the need for support hardware (as RAM). *Flower and Jabri* [71] use digital synapse strength storage in an implantable heart cardioverter-defibrillator, for instance.

For analogue systems that include hardware learning, there is another obstruction connected with digital storage; the need for *analogue to digital converters* (*ADC*s) to write the synapse strengths. Using a parallel weight updating in such a system is area inefficient (though see *Hollis et al.* [99], *Shima et al.* [220]) because of the necessary high weight resolution during learning (typically 10–16 bit) (*Hollis et al.* [97], *Lehmann* [139, 140], *Tarassenko et al.* [238], *Höhfeld and Fahlman* [96], *Brunak and Hansen* [31]). Inspired by the fact that the weight changes during learning is usually much smaller than the necessary resolution of the recall-mode network (say 6–8 bit), it has been proposed to use an *analogue adjustment* (an analogue "bit") to the digital memory, which is active during learning, see figure 10[2] (*Lehmann et al.* [146], *Lansner* [133], *Bruun et al.* [35]): The weight changes determined by the on-chip learning algorithm are accumulated on the analogue memory. When the equivalent of 1 LSB has been accumulated, the digital word is decreased or increased and the analogue adjustment is reset. This operation basically requires two one bit ADCs and a digital adder, which could be shared by a column of synapses.

Of other synapse memories, read-only memories should be mentioned (*Masa et al.* [157], see also *Mead and Ismail* [163], *Mead* [162]) which for instance could be programmed by transistor sizing. Read-only memories, though, are not interesting in the context of hardware learning.

In this work, we have chosen the simple capacitive storage method with RAM backup — not because it is particularly well suited for learning; it is not — because this is a simple, reliable concept which allow the most dense synapse packaging. The learning scheme will have to submit to this storage method.

2.2.4 Multipliers

Unlike analogue memories, analogue multipliers are very easily implemented in, say, CMOS — provided the inherent offset and non-linearity are acceptable. Many different multiplier architectures have been proposed in the literature (see *Kub et al.* [126], *Neugebauer and Yariv* [180], *Bibyk and Ismail* [23], *Hollis et al.* [98], *Massengill* [158], *Woodburn et al.* [270], *Saxena and Clark* [208], *Sánchez-Sinencio and Lau* [206]) and we shall restrict our examination to only a few.

The desired synapse multiplier key characteristics are the following (cf. above):

- Small
- Current output
- Voltage inputs. One of these should have a *very* high input impedance (ie. a MOS gate) so that the capacitance on this node can be used for the synapse strength storage.

Prior to implementing a synapse multiplier there are two questions we need to answer. First: do we need a *four quadrant multiplier*? The synapse strengths in a general neural network needs to be bipolar, thus we need two quadrant synapse multiplication†. Often the neuron activation function (eg. tanh(.)) yields as output a bipolar value, indicating that we would need a four quadrant synapse multiplication. However, doing a simple linear transformation on the activation function (from bipolar units (superscript "±1"), $y_k^{\pm1} \in [-1,1]$, to unipolar units (superscript "01"), $y_k^{01} \in [0,1]$):

$$\begin{aligned} g_k^{01}(.) &\equiv \tfrac{1}{2} g_k^{\pm1}(.) + \tfrac{1}{2} \quad \Rightarrow \\ w_{kj}^{01} &= 2 w_{kj}^{\pm1} \\ \Theta_k^{01} &= \Theta_k^{\pm1} + \sum_j w_{kj}^{\pm1} \end{aligned} \quad , \qquad (3^2)$$

where Θ_k is the neuron threshold (cf. (30^{B})), we see that there is — in a mathematical sense — no need for a bipolar activation function. (As would be expected because pulse stream networks, as our brain, are operational.) When it comes to learning, though, it is advantageous to use bipolar neuron outputs: Learning algorithms typically have a factor $\epsilon_k y_j$ in the weight updating rule for w_{kj}, where ϵ_k is the neuron k error. Thus, if unipolar neurons are used ($g_k(.) \equiv \frac{1}{2}\tanh(.) + \frac{1}{2}$, say), the weight change is negligible when y_j is close to the lower extreme value (0), regardless of the neuron error. For this reason, ANNs using bipolar neurons tend to learn faster (*Stornetta and Huberman* [230], *Haykin* [93], see also *Le Cun et al.* [56]). It should be noted that the transformation (3^2) can be applied to learning algorithms as well as to networks. This would yield slightly more complicated weight

† Actually, by adding a constant prior to the multiplication and do a subtraction afterwards ($s = wz = (w + w_0)z - w_0 z$) only one quadrant multiplication is strictly necessary (*Johnson et al.* [115], see also *Woodburn et al.* [270]). However, this method is bound to introduce additional offset errors which turns out to be a major problem in analogue VLSI neural networks (cf. the following text).

updating rules and different rules for w_{kj} and Θ_k, though, which is undesirable. A quite different motivation to use a four quadrant synapse multiplier is that this will be needed to the implementation of the learning hardware (cf. the following chapters) and it reduces design time and error probability to reuse building blocks.

The second question is: do we need a *linear multiplier*? Doing *gradient descent* learning, one needs to know the $\partial s_k / \partial w_{ij}$ derivative, where s_k is the sum of the synapse outputs. To fulfill this need, the multiplier should be linear (to be in compliance with our simple ANN model) or at least have a computable transfer function derivative. However, this is a requirement somewhat more strict than needed: The inherent fault tolerance of ANNs relaxes this requirement and for ANN chips taught using chip-in-the-loop training or hardware training inaccuracies can be eliminated to a great extent by the learning algorithm (*Eberhardt et al.* [66], *Lehmann* [139, 140], *Montalvo et al.* [168], *Castro et al.* [42], *Valle et al.* [251], *Card* [40], *Leong and Jabri* [149], *Woodburn et al.* [270], among others; see also section 4.4.3). What is of greater importance than the multiplier linearity is its *dynamic range*, usually restricted to about 60 dB, which puts constraints on the networks that can be mapped on a given topology. In this connection, the multiplier output *offset error* is very important: When connecting the outputs from many synapse multipliers the output offsets will accumulate, easily giving a resulting offset that is greater than the maximum output of a single synapse (if the multiplier has a systematic offset, this is very probable). While in principle this resulting neuron input offset error can be canceled by adjusting the *bias*, the dynamic range of the bias synapse is thus easily exceeded, if steps to prevent this is not taken (eg. offset canceling). These chip specific offset errors, and other process variation related inaccuracies, are the reasons why analogue ANNs in general need to be taught using chip-in-the-loop training. Also, in analogue systems with on-chip learning, even small multiplier offsets can cause severe problems, cf. the following chapters.

Gilbert multiplier A very popular four quadrant multiplier is the *Gilbert multiplier* shown in figure 7^2 (*Kub et al.* [126], *Schneider and Card* [210]). Assuming a square law approximation for the saturated MOSFETs†, one can show that the output current is given by:

$$i_{wz} = i_{wz+} - i_{wz-} \approx \sqrt{\tfrac{1}{2}\beta_w\beta_z} \cdot v_w v_z \,,$$

when v_w is small in the sense that the differential output current of the upper differential pairs are linear in v_w‡. β_w and β_z are the transconductance parameters for the upper two and the lower differential pairs respectively.

When used as a synapse multiplier, a row of synapses can share the current mirror that is needed to take the $i_{wz+} - i_{wz-}$ difference (we call this circuit the *current differencer*) thereby saving a current mirror per synapse — though for a

† $i_{\mathrm{D}} = \frac{1}{2}\beta(v_{\mathrm{GS}} - V_{\mathrm{T}})^2$.

‡ More precisely $v_w \ll \sqrt{\frac{2I_{\mathrm{B}}}{\beta_w}}\sqrt{1 - \sqrt{\frac{\beta_z v_z^2}{I_{\mathrm{B}}} - \frac{\beta_z^2 v_z^4}{4I_{\mathrm{B}}^2}}}$.

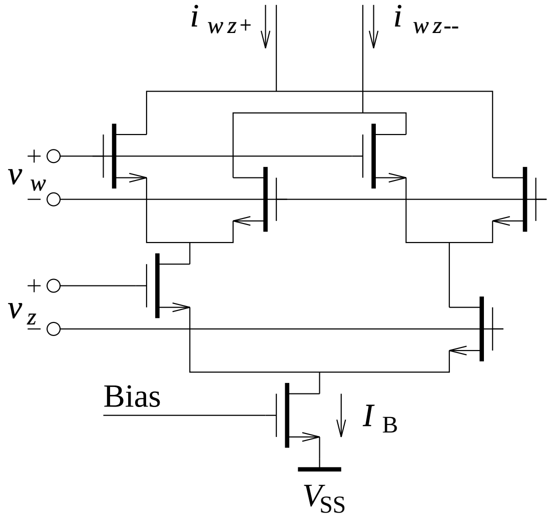


Figure 7²: MOS Gilbert multiplier. *All transistors work in saturation. A wide range version is possible by adding a number of current mirrors.*

given accuracy of the difference, the total transistor area devoted to this task can not be decreased, cf. appendix C.3. Note, that if local current differencing is used and the multiplier output is not directly compatible with the neuron input, utmost care should be taken in the design of the required signal converter as not to loose the good accuracy in this component.

For a design with a restricted supply voltage, a wide range version of the multiplier is easily implemented by the addition of a number of current mirrors. A "folded" version of the multiplier is also a possibility.

The MOS resistive circuit Another, very linear, multiplier is the *The MOS resistive circuit* (*MRC*) shown in figure 8² (*Czarnul* [57], *Khachab and Ismail* [123], *Tsividis et al.* [245]). When ensuring virtual short-circuit of the output terminals, the differential output current is given by:

$$i_{wz} = i_{wz+} - i_{wz-} = \beta v_w v_z = \frac{1}{r_{\text{MRC}}} v_z \,.$$

The transistors operate in the triode region. Though requiring four matched transistors, the circuit has several nice properties: It cancels out most non-linearities of the MOS transistors, making the difference current very linear in both v_w and v_z. The difference current is independent of the threshold voltage making it insensitive to bulk effect and substrate noise. Further, the circuit is quite fast as the high frequency effects of parasitic capacitances also tend to cancel out. The disadvantage of the circuit is that the triode mode operation require a somewhat large power supply voltage. For differential mode signals on the v_z terminals, the circuit acts as two controlled resistors (cf. figure 9²) with resistances r_{MRC} as defined above — an observation that is often very convenient when analyzing circuits with MRCs.

The need for a virtual short-circuit prevents the feasibility of local current subtraction when this circuit is used as a synapse multiplier — thus the circuit will usually exhibit relative large output offset errors. However, consisting of only four transistors, the synapse density can be very high when using this multiplier. Actually, even higher synapse density is possible if single ended signalling is employed:

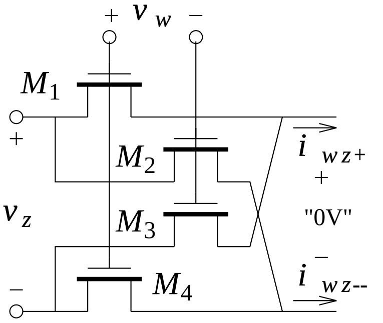


Figure 8[2]: MOS resistive circuit multiplier. *All transistors work in triode mode; the output potentials must be equal. The triode mode operation restricts the dynamic range.*

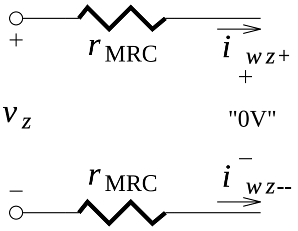


Figure 9[2]: MRC resistive equivalent. *This convenient equivalent circuit is valid for differential mode signals only.*

Using one of the v_z terminals as a constant reference, and forcing the output potentials to be at that same potential, two of the transistors will have zero drain-source voltage and can thus be eliminated as they do not conduct any current (*Flower and Jabri* [71]). The problem with this approach is that very low impedance summing nodes are necessary at the outputs, capable of sinking current from many synapses.

Multiplying DAC When using a digital synapse memory in an analogue system, a *multiplying DAC* (*MDAC*) must be used as the synapse multiplier. The disadvantage of this approach is its excessive area consumption. However, at the expense of reduced accuracy (cf. above, appendix C.3), the *scaling* and *summing* circuit of the DACs can be shared by (say) a row, k, of synapses. This way only B identical voltage controlled current sources are needed at the synapse sites for an B bit resolution, see figure 10[2](*Bruun et al.* [32], *Dietrich* [59], see also *Van der Spiegel et al.* [228]). The transconductor and the diode coupled transistor (common to a column, j) ensures a current proportional to the input voltage v_{z_j} for all the synapse current sources that have the corresponding weight bit $w_{kj}^{\mathrm{b}\xi}$ set. The currents on the B output lines are scaled and summed by the current adder (common to a row), producing the resulting output current. The accuracy of this solution is primarily determined by the diode coupled transistor which has to match all the synapse current sources of a column. The resolution is determined by the local current source matching and the accuracy of the scaling current adder. At a very modest area increase, the circuit can handle bipolar inputs (*Dietrich* [59], *Lehmann et al.* [146], *Lansner* [133]). Such a 4×4 synapse chip designed at our institute has been fabricated in a standard 1.5μm CMOS process (giving an accuracy of about 1%), proving the applicability of the scheme (*Dietrich* [59], *Lehmann et al.* [146]).

When doing on-chip learning in a system with digital synapse weights, the weight resolution can be temporary increased during learning by the addition of an *analogue adjustment*, as noted in section 2.2.3. To get an accurate weight

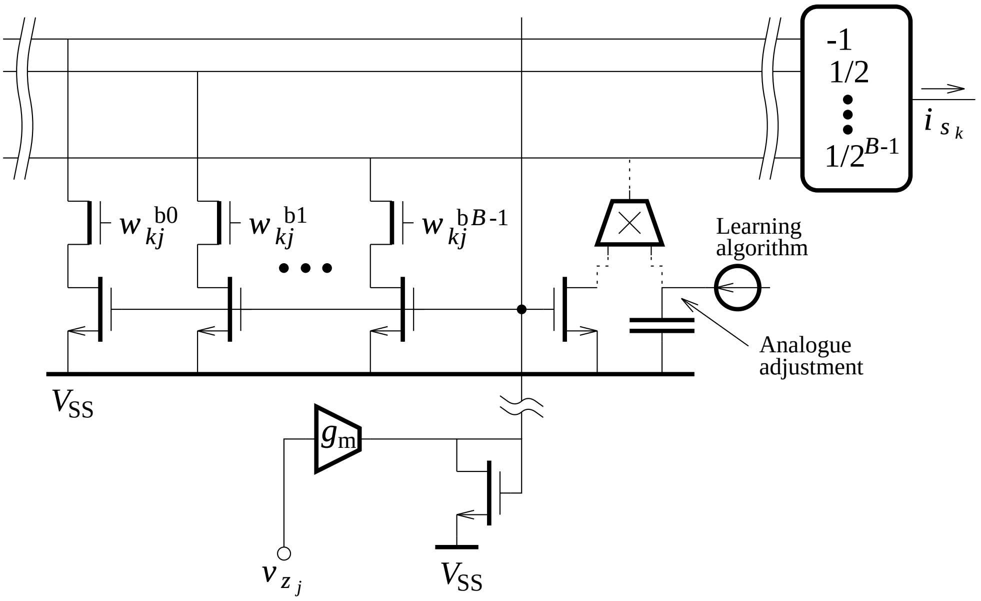


Figure 10^2: Multiplying DAC synapse. *Simplified schematic for positive inputs. The switch transistors are controlled by a local weight register (bits $w_{kj}^{b\xi}$; not shown). The value of the optional "analogue adjustment" is controlled by an on-chip learning algorithm.*

multiplication during learning, an analogue multiplier should be added to the multiplying DAC, as indicated in the figure. Improved learning would result by the omission of this multiplier, however, as the network errors would be calculated on the actual resulting network rather than on an intermediate network with higher weight resolution (*Lansner* [132]). See also section 2.6.

If the input neuron activation value (or the network input) is binary, the multiplier architecture can be *very* simple (*Murray et al.* [175], *Woodburn et al.* [270], *Graf and Jackel* [82], *Johnson et al.* [115]). Basically, a weight controlled current source is either switched to the multiplier output or not, depending on the state of the input, requiring as few as three (or even one) transistors for unipolar neuron activation. See also section 2.3.3.

Using a *synapse multiplier* which is *non-linear* in v_w (*NLSM*) (eg. $\propto \sinh(v_w)$) can reduce the problem of limited *dynamic range* (*Van der Spiegel et al.* [228], *Hollis et al.* [99], *Valle et al.* [251], *Kwan and Tang* [128]). As it is the magnitude (*and* sign) rather than the actual value of the weight that is of importance to the ANN performance, the inevitably reduced resolution for large weights is of less concern. Using MOS transistors operated above threshold achieving an exponential characteristic is not easy. A square-law non-linearity, however, is easily achieved as indicated in figure 11^2. This particular circuit also has the advantage that an offset free zero weight can be ensured (ignoring subthreshold currents) by proper biasing, see figure 12^2. Also, the flat plateau around zero output current will tend to trap small weights at efficiently zero value — the multiplier could be said to be "*self-pruning*" — which might improve the ANN generalization ability.

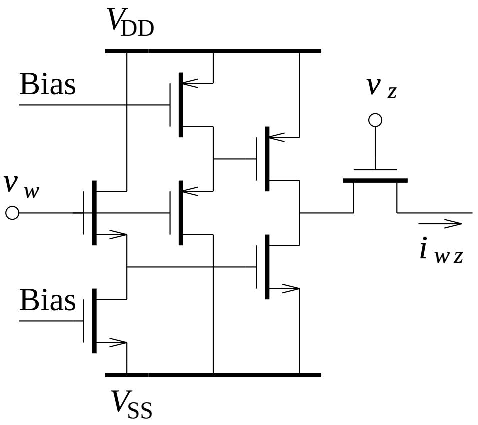


Figure 11[2]: Simple non-linear synapse multiplier. *The four leftmost transistors act as level shifters that ensure only one of the saturated, middle "weight transistors" are turned on. The multiplication is performed by the rightmost switch transistor.*

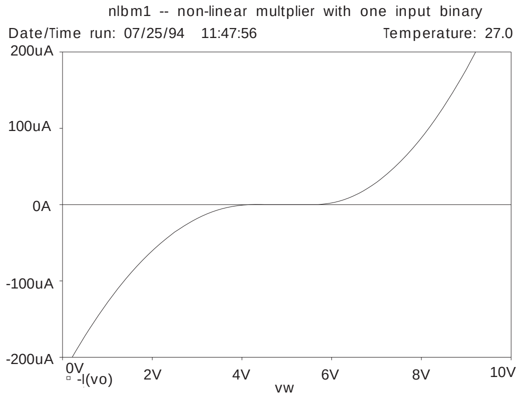


Figure 12[2]: Weight-output characteristic of NLSM. *A simple double (asymmetric) square law characteristic. Notice the null range which ensure zero output offset error.*

In this work we have chosen to use the MOS resistive circuit multiplier. There are several reasons to this: (i) The MRC is a small, fast multiplier which is important to the applicability of massively parallel analogue ANNs. (ii) The learning algorithm, which we are to add at a later stage, not being determined in advance, we should not reject the possibility of implementing a gradient descent algorithm. The extent to which an unknown synapse non-linearity can be canceled by the learning scheme is problem- and algorithm dependent; a reasonably linear multiplier is the safe choice. (iii) The MRC is a very versatile component. One can, for instance, implement a voltage in, voltage out multiplier/divider ($v_{\mathrm{out}} = v_{\mathrm{in}1}v_{\mathrm{in}2}/v_{\mathrm{in}3}$), using two MRCs and an op-amp, which is independent of process variations (cf. the following chapters). This will be needed for the learning scheme and we can thus reuse our multiplier cell which reduces the possibility of design errors. In this connection it should be mentioned that there are several other possible choices for process variation insensitive voltage in, voltage out multipliers (eg. *Wang* [257]; see also *Sakurai and Ismail* [204], *Coban and Allen* [49], *Botha* [28]).

2.2.5 Activation functions

The last thing that needs to be considered before implementing the analogue neural network is the threshold function. If a binary valued neuron transfer function is sufficient for the application at hand, the neuron circuit can be very simple (*Bibyk and Ismail* [23], *Hollis and Paulos* [98]). If, on the other hand, a continuous valued non-linearity is sought (as in our case) the circuit is often somewhat more complex. This complexity is usually not a major concern, though, as there are only N neurons compared to the order $O(N^2)$ synapses. The exact shape of the neuron transfer function is usually irrelevant (cf. eg. *Sánchez-Sinencio and Lau* [206]). What is more important is its qualitative shape; eg. that it is monotonous and saturates for numerically large inputs. This is a very attractive feature which means that transfer functions easily implemented in the technology can be chosen.

Pulse frequency neuron The neuron schematic is of course strongly dependent on the network signalling domains. For pulse frequency neural networks, for instance, a neuron must be a non-linear *current controlled oscillator* (*CCO*). A sample implementation of such a neuron can be seen in figure 13[2] (*Murray et al.* [176]).

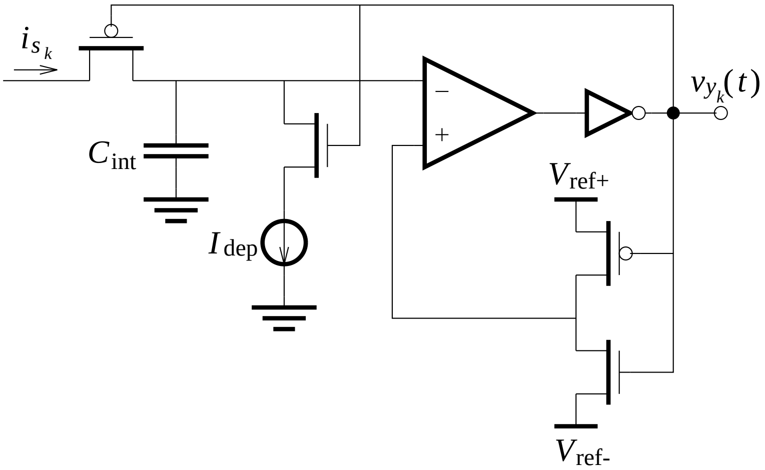


Figure 13[2]: Pulse frequency neuron. *The output voltage alternates between digital high and low at a frequency determined, in a non-linear way, by the input current.*

The pulse frequency will be in the range $f_{\text{act}} \in [0, I_{\text{dep}}/C_{\text{int}}(V_{\text{ref}+} - V_{\text{ref}-})]$. This particular circuit does not have a particularly low input impedance which must be compatible with the synapse multipliers.

Distributed neuron Using continuous valued current in/voltage out signalling for the neurons, the neuron non-linearity can be achieved simply by applying a non-linear load to the summed synapse outputs — the same wire would then be used as both neuron input and output (we assume that the loads at the neuron output has a very high impedance). This approach makes it very easy to distribute the neuron hardware on the synapses rather than using conventional lumped neurons, see figure 14[2] (*Satyanarayana et al.* [207]). The distributed approach has the obvious advantage that the current range in the distributed elements is that of a single synapse — thus making the system truly scalable to an arbitrary size. The function implemented by the *distributed neuron-synapse* approach is

$$y_k = g_k\left(\frac{1}{M_k}\sum_j w_{kj} z_j\right),$$

where M_k is the number of inputs to the resulting neuron, k. Some argue that the typical weight magnitudes of a neuron is often proportional to $1/M_k$ (though, see section 2.3.1), in which case this very factor can actually improve the effective dynamic range of the synapses by a factor M_k (*Satyanarayana et al.* [207]; see also *Eberhardt et al.* [65]). Note that the distributed neuron elements must be kept small as their number scale as $O(N^2)$.

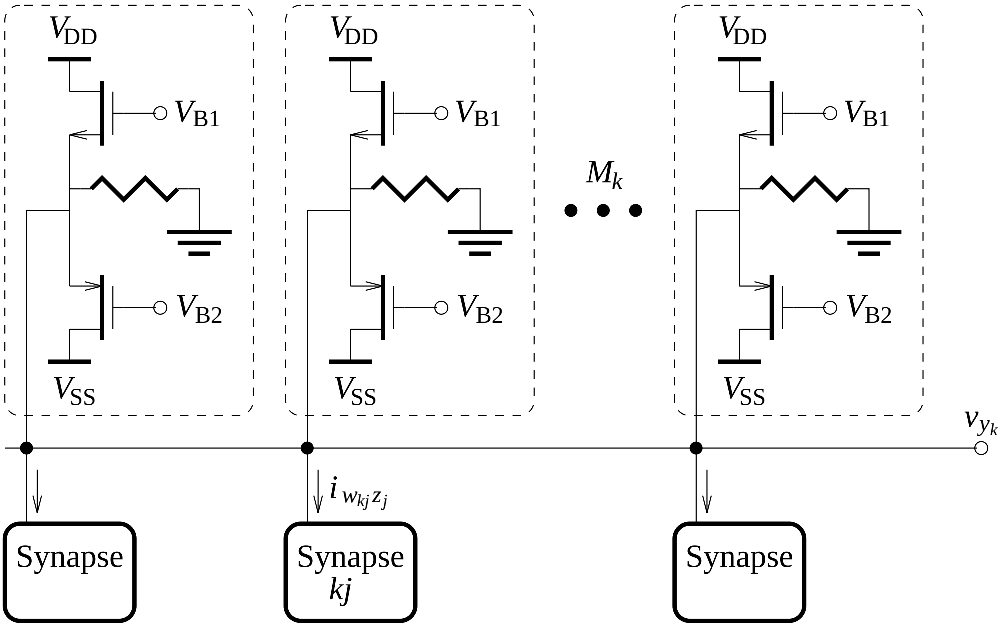


Figure 14[2]: Distributed neuron. *The input current and output voltage are carried on the same wire. The transfer function for this particular neuron does not saturates in itself; the synapse would contribute to the transfer characteristic. This distributed approach makes the network truly scalable.*

Hyperbolic tangent neuron If we are to add learning hardware to an acting neural network, we will (most probably) not have access to the neuron net inputs, the s_ks. For the implementation of a gradient descent algorithm $\partial y_k/\partial s_k$ needs to be computed. Thus the choice of a *hyperbolic tangent transfer function* — the transfer function of a bipolar differential pair — is well tailored to this situation: we have

$$\frac{\partial}{\partial s_k}\tanh(s_k) = 1 - \tanh^2(s_k).$$

Unfortunately, the differential pair has voltage input and current output, rather than the other way around, which makes the use of input and output transresistances necessary. A hyperbolic tangent neuron implementation can be seen in figure 15^2, p. 29.

Bipolar transistors are not available in standard CMOS processes. However, well MOS transistors can be operated in *lateral bipolar mode* (*LBM MOSFET*) which turns on a reasonably good (though somewhat slow) bipolar transistor, see appendix C.2. Though it is against the philosophy of section 2.2.3 to use non-documented devices, the offense is not too severe in this case: using a fairly simple regulating circuit, the primary parameter of the LBM MOSFET, the emitter-collector current gain, can be measured and adapted to. See section 2.7.

We shall discuss the necessity of computing the transfer function derivative in later chapters; as well as problems related to this calculation. Further, we shall examine other neuron circuits. For now, "not constraining the future implementation of the learning hardware" motivates our choice of transfer function, which is this hyperbolic tangent one implemented using LBM MOSFETs.

Many other neuron architectures can be found in the literature, to which we refer the interested reader (eg. *Mueller et al.* [173], *Schneider and Card* [210], *Sánchez-Sinencio and Lau* [206], *Mead* [162]).

2.3 Chip design

In this section we will describe the chip set developed at our institute. Considerations on the choices of the central components were given in the previous section. A description of the chip set was published in *Lansner and Lehmann* [130, 131] (see also *Lehmann* [141, 142], *Schultz* [211]). The cascadable chip set consist of a neuron chip and a synapse chip, having the topologies shown in figure 1^2.

The chip set was designed primarily to test the ANN functionality, and thus as little hardware as possible was included on the chips — to reduce the possibility of malfunctioning chips. This design methodology has proven successful: all the designed chips worked after first processing (though errors occurred). The price for this reduced error probability is basically that the chips need a large number of biases (voltages and currents) which severely complicates their use. As we, unfortunately, did not have sufficient time to design a complete, volume manufacturable set of chips, problems related to self-biasing, temperature compensation, etc. are not experimentally covered in this thesis; though *very* important to VLSI design.

The integrated *CMOS process* we shall use is a standard analogue 12 V, 2.4 μm, double poly, double metal, n-well CMOS process. In order to put as few constraints as possible on the analogue building blocks, we have chosen a rather large, ±5 V, power supply. This was convenient as the different components on the first chip set was developed concurrently by three different designers (myself, John A. Lansner and Thomas Kaulberg). In a future implementation, the building blocks should be redesigned to a standard 5 V (or 3.3 V) digital process.

A *design strategy* that has been employed is to reuse components whenever possible, in order to reduce the possibility of design errors and design time. This is true for on-chip "micro components" (as the op-amp) as well as "macro components" (as the matrix-vector multiplier) as we shall see in the following chapters.

A few preliminary, general system aspect considerations are needed before the actual chip designs. These can be found in appendix D.1.

2.3.1 The neuron chip

The neuron chip design was done by John A. Lansner (see *Lansner* [133]). The schematic of a neuron on the neuron chip is shown in figure 15[2]. The core of the circuit is the bipolar differential pair implemented using two LBM MOSFETs. The differential output current of this pair is converted to a single ended voltage by the "output range" (OR) MRC (and op-amp). (For the op-amp schematic, see appendix E.1.) This is again buffered so the neuron can drive the relative low impedance input of the synapse chip (cf. below). At the input of the neuron, the "input scale" (IS) MRC (and op-amp) is likewise placed, acting as the input transresistance that converts the input current to voltage needed to drive the differential pair. The resulting transfer function is the following:

$$\begin{aligned} v_{y_k} &= \frac{1}{\beta_{\mathrm{OR}} V_{\mathrm{OR}}} \alpha_{\mathrm{FC}} I_{\mathrm{B}} \tanh\left(\frac{i_{s_k}}{2\beta_{\mathrm{IS}} V_{\mathrm{IS}} V_{\mathrm{t}}}\right) \\ &= R_{\mathrm{OR}} \alpha_{\mathrm{FC}} I_{\mathrm{B}} \tanh\left(\frac{R_{\mathrm{IS}} i_{s_k}}{2V_{\mathrm{t}}}\right) \end{aligned} , \qquad (4^2)$$

where I_{B} is the bias current and α_{FC} is the *emitter-collector current gain*. Because of the undesired vertical collector of the LBM MOSFET, connected to the substrate, we have $\alpha_{\mathrm{FC}} = -i_{\mathrm{C}}/i_{\mathrm{E}} \overset{\text{very}}{\approx} 0.5$. The output voltage v_{y_k} is referred to $V_{\mathrm{ref}} = -2\,\mathrm{V}$ to be compatible with the synapse inputs.

The input impedance is non-linear and strongly dependent on (though always smaller than) the resulting transresistance R_{IS}. One should not be too offended by this non-ideal load of the current source: When the tanh is saturated, even severe non-linearities in the transresistance (or in the synapse output stage, caused by the non-linear, finite load) are indifferent as an input current (i_{s_k}) error would be indistinguishable at the neuron output. The differential pair saturates for a differential input voltage of a few V_t ($\tanh(4V_{\mathrm{t}}/2V_{\mathrm{t}}) = 0.96$), which is thus the voltage shift from the input reference (0 V) that must be tolerated by the synapse chip (regardless of R_{IS}). This is easily accomplished.

Though the input transresistance is adjustable, it has a dynamic range of only 30 dB–40 dB. In other words, the *neuron transfer function steepness*, β_{t}, is adjustable within this range (or the *effective maximum synapse weight*, $|w|_{\max}$, if one prefers to think of the transfer function with a fixed steepness, $\beta_{\mathrm{t}} = 1$). Clearly,

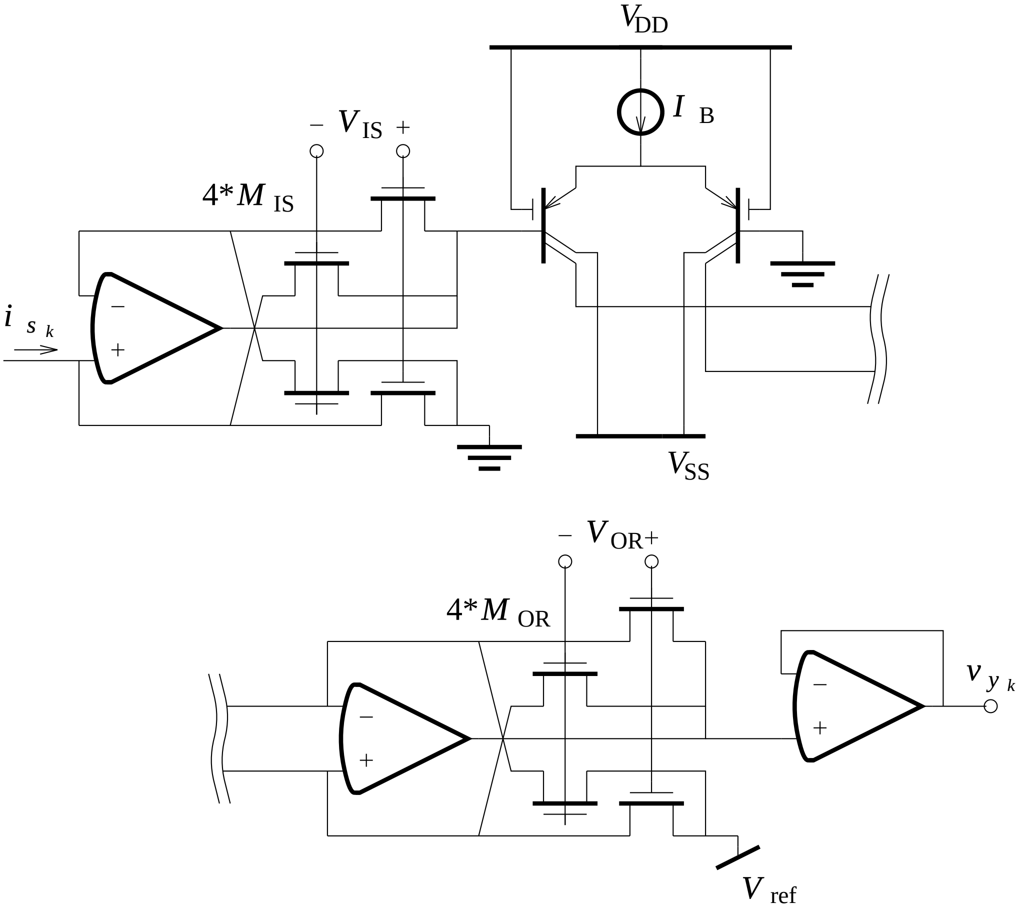


Figure 15[2]: Hyperbolic tangent neuron. *Basically a BJT differential pair (using parasitic components) embedded in transresistances.*

in a system with an arbitrary large number of synapses connected to each neuron, this dynamic range does not allow a neuron to be saturated only if all synapses are exiting it — as would be the case if the steepness scaled as $1/M_k$, where M_k is the number of connected synapses. The dynamic range of our neuron steepness is sufficient to cancel process variations, but not much more. In many classification problems, it is often seen that the neurons in a trained network (both output and hidden ones) are most often saturated (see eg. *Brunak and Lautrup* [30], *Krogh et al.* [125]; see also *Williams and Zipser* [268]); that is, they act more or less as hard limiters — clearly, this is not possible if the steepness scales as $1/M_k$. If one could ensure that the transresistance was adjustable down to $0\,\Omega$, both extremes could be embraced (*Eberhardt et al.* [65]; see also *Mueller et al.* [173]), but using a lumped neuron approach this would prove most difficult: the lumped neuron would have to be able to sink an arbitrarily large current. Further, even such a system would not be able to handle the situation where all synapses but one must agree to make a decision, which in turn can be overruled by the last synapse (a situation which is not uncommon in the brain (*Rumelhart et al.* [200])). The unfortunate conclusion is, that in order to be absolutely general, the *dynamic range* of a *synapse* must scale as M_k, which is incompatible with analogue VLSI. This, once again, stress the importance of making the dynamic range of the synapse as large as possible.

In order to make reasonably simple neurons and learning algorithms, we have chosen the "fixed" neuron steepness approach. The steepness has been selected to be compatible with "typical" weight magnitudes found in reported systems and our own simulations. It should be noted that, if the non-linearity introduced by the non-linear neuron input impedance is acceptable, the neuron steepness can be reduced simply by adding a parallel external resistance at the input. It could be one of the objectives of further research to enhance the neuron steepness dynamic range (the steepness should be governed by the learning algorithm).

2.3.2 The synapse chip

The synapse chip consists of a number of *inner product multipliers* (*IPM*) that multiply the input vector ($\underline{v}_{z_\star}$)† with a row of the stored matrix ($\underline{V}_{w_{k\star}}$). Such an inner product multiplier is shown in figure 16^2 (see eg. *Bibyk and Ismail* [23]). The difference of the summed synapse MRC outputs is taken by the op-amp with MRC feedback, which also ensures the required virtual short-circuit of the synapse outputs. The resulting voltage is transformed by the transconductance ($g_{\mathrm{m}k}$) to the output current (i_{s_k}). (For the op-amp and the transconductance schematics, see appendix E.1 and E.2.) The resulting transfer function is the following:

$$i_{s_k} = \frac{g_{\mathrm{m}k}}{W_0/L_0 \cdot V_{\mathrm{C}}} \sum_j W_j/L_j \cdot V_{w_{kj}} v_{z_j} \ , \tag{5^2}$$

where W_ξ/L_ξ are the MRC width/length ratios and V_{C} controls the total transconductance. This control voltage is used to adjust the effective maximum synapse weight, though the dynamic range allow only for small adjustments (as compensating process variations), as was the case for the neuron steepness adjustments.

The schematic of a single synapse is shown in figure 17^2. The synapse strength is stored in a differential manner on capacitors at each synapse site; this way offset due to *charge injection* (*Shieh et al.* [219], *Wegmann et al.* [260]) is canceled, as well as (the differential) charge leakage due to the reversed biased drain-bulk diodes of the sampling switches; provided that the components match. To ensure random synapse access, the sampling switches are controlled by a NAND gate rather than directly by the row/column select signals provided by the row- and column decoders (as in *Lee et al.* [137] or *Kub et al.* [126]). For minimum geometry transistors for the gate, the area overhead is acceptable. See also appendix D.1.1.

† "$\star$" meaning that the index runs over all possible values:
$\underline{v}_{z_\star} \overset{\mathrm{def}}{=} [v_{z_1}, v_{z_2}, \ldots, v_{z_{M_k}}]^{\mathrm{T}}$

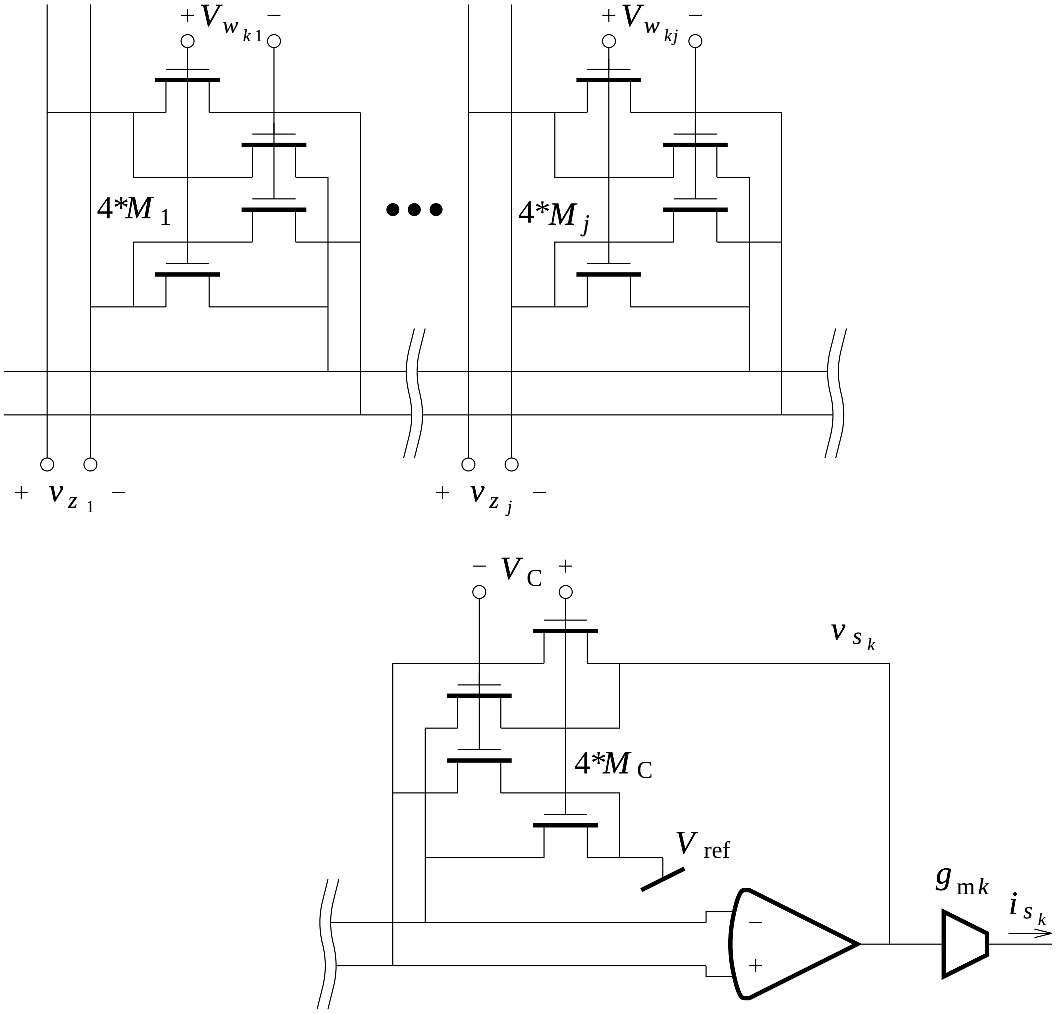


Figure 16^2: Inner product multiplier. *One MRC per input vector dimension is required (upper part). The MRC feed back opamp ensures virtual short-circuit at the MRC outputs; the transconductance converts the opamp output voltage to a current for cascadability.*

The second generation synapse chip Using an op-amp with MRC feedback as the synapse differencer, followed by a transconductance to obtain the desired output current is somewhat indirect. A simpler and more accurate approach is to use a *current conveyor* (see appendix E) to take the difference while ensuring the virtual short-circuit, as shown in figure 18^2. See also chapter 4. The $i_{s_k}+$ input is lead directly to the output and the $i_{s_k}-$ is negated by the current conveyor and added to the output. The y-x voltage follower ensures the virtual short-circuit. To avoid DC common mode currents in the synapses the output potential of the current conveyor should be close to V_{ref}, which requires a slightly changed neuron schematic (cf. chapter 4). This solution does not allow the *effective maximum synapse weight* to be tuned as above. Process variations must thus be canceled by scaling the weights, which reduces the dynamic range of the synapse weights slightly. The adjustments are carried out automatically during learning, though,

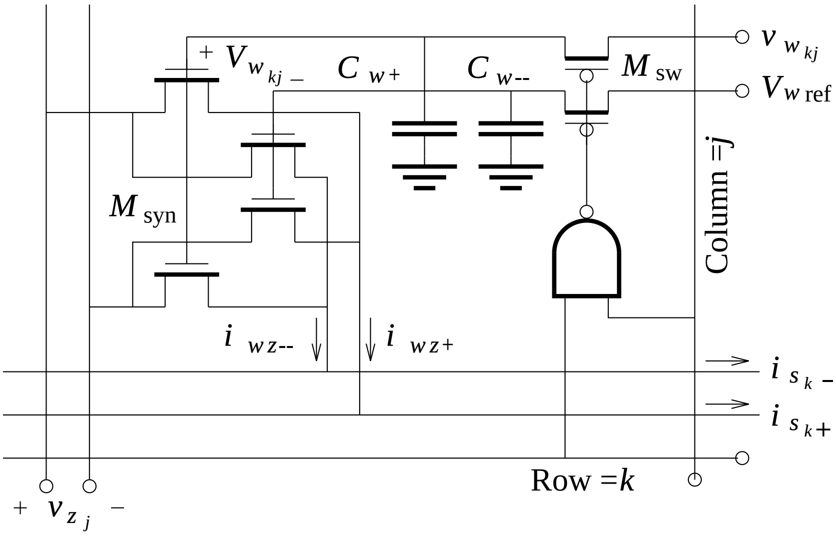
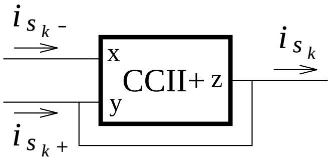


Figure 17[2]: Synapse schematic. *In addition to the MRC multiplier weight-storage (differential, capacitive) and access (switch transistors and* NAND *gate) circuit is placed at the synapse sites.*

without any weight scaling learning mechanism.

Figure 18[2]: Current conveyor differencer. *This circuit gives as output $i_{s_k} = i_{s_k+} - i_{s_k-}$ while buffering the y-node voltage to the x-node. The input voltage is thus determined by the output load.*



It is also possible to use a current-mode op-amp with resistive feedback as a differencer. This way the effective maximum synapse weight can be adjusted while preserving the good accuracy of a simple differencer.

The use of one of these *current-mode* operational devices to our *current-mode* signal processing is the superior choice, when considering both speed and accuracy of the circuit (*Bruun* [33], *Bruun et al.* [35]).

2.3.3 Sparse input synapse chip

Often the inputs to a neural network are taken from a *discrete input alphabet* consisting of a number, $N_{\mathcal{A}}$, of symbols (or *letters*), $\alpha_1, \alpha_2, \ldots, \alpha_{N_{\mathcal{A}}}$. To avoid false distance relations among the different letters, unary coding of the letters is usually employed; that is, $N_{\mathcal{A}}$ network input lines are assigned to every logical input (or *letter input*), X_ξ:

$$\begin{array}{lccccc}
X_\xi \equiv & x_{\xi N_{\mathcal{A}}+0}, & x_{\xi N_{\mathcal{A}}+1}, & x_{\xi N_{\mathcal{A}}+2}, & \ldots, & x_{\xi N_{\mathcal{A}}+N_{\mathcal{A}}-1} \\
\alpha_1 & 1 & 0 & 0 & \cdots & 0 \\
\alpha_2 & 0 & 1 & 0 & \cdots & 0 \\
\vdots & & & & \ddots & \\
\alpha_{N_{\mathcal{A}}} & 0 & 0 & 0 & \cdots & 1
\end{array}$$

We notice that the network inputs are used *sparsely*. One can use -1 instead of 0 as the inactive value, as discussed in section 2.2.4. In this particular case, however, the choice of 0 can actually improve learning: only the weights related to the present input letter in each *letter input* will be modified (by a typical algorithm, cf. above).

Sample applications As examples of applications where unary input coding are used, we shall mention *prediction of splice sites* and *word hyphenation*:

In the *human genom project*, which aim is to map all human genes, the prediction of *splice sites* in pre-mRNA molecules (a copy of part the information in a DNA molecule) is an important task. Much of the DNA information is "junk" that does not code for proteins and much of this junk DNA is scattered about in DNA sequences that *does* code for proteins. Prior to the protein synthesis, the body cells cut out these junk sequences of the pre-mRNA molecules. The junk-code boundaries are the *splice sites*, see figure 19[2] (*Stryer* [231], *Brunak et al.* [29]). These splice sites are difficult to predict and are dependent on the context of the nucleotide sequence. Applying neural networks to this problem has proven to be very successful (*Brunak et al.* [29]). The input alphabet for this ANN application has $N_{\mathcal{A}} = 4$ letters, A, T (or U), G, and C, corresponding to the four nucleotide bases of DNA (or RNA). *Brunak et al.* used two layer perceptrons with the order of 300 letter inputs (ie. 1200 network inputs) and 200 hidden and one output sigmoid neurons for this classification task.

Another application that uses a discrete input alphabet is *word hyphenation*. Ignoring context dependent hyphenation (as `de-sert` vs. `des-ert`), a good solution can be found using an ANN with an input window of a few letters: *Brunak and Lautrup* [30] used a two layer perceptron with eight letter inputs and 30 hidden and one output sigmoid neurons to hyphenate Danish words. The input alphabet of this classification task has $N_{\mathcal{A}} = 30$ letters corresponding to the 29 letters of the Danish alphabet (a, b, ..., z, æ, ø, å) plus a "null" letter.

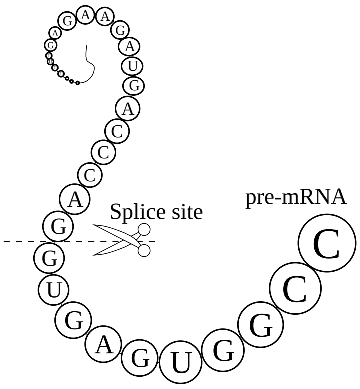


Figure 19^2: Nucleotide sequence. *A splice site is shown for a sample schematic pre-mRNA molecule; RNA is composed of a sequence of the bases adenosine, cytidine, guanosine and uridine.*

The sparse, unary coding of the letter inputs — say, applying 29 zero inputs for each 1 input — clearly exploits the input bandwidth of a "standard" synapse chip poorly. For systems with, say, hundreds of letter inputs even a small reduction in the required input bandwidth can reduce the system cost considerably. This is the motivation to develop a special input layer synapse chip for networks with a discrete input alphabet — a *sparse input synapse chip.*

The basic idea of this synapse chip is simply to use binary coding of the input alphabet and decode this to unary coding on-chip. Standard digital CMOS design techniques can be used for the decoding or, alternatively, a demultiplexor (controlled by the letter input) can be used to redirect a current as shown in figure 20^2. The figure shows the current demultiplexor for a two bit letter input and one of the four synapse columns connected to the demultiplexor. As discussed in section 2.2.4, the synapse multiplier can be very simple when the z input is binary; eg. as in figure 11^2. One can also use the demultiplexed current to derive the bias currents for synapse transconductors as shown in figure 20^2; only one of the four columns of synapse transconductances will have bias currents.

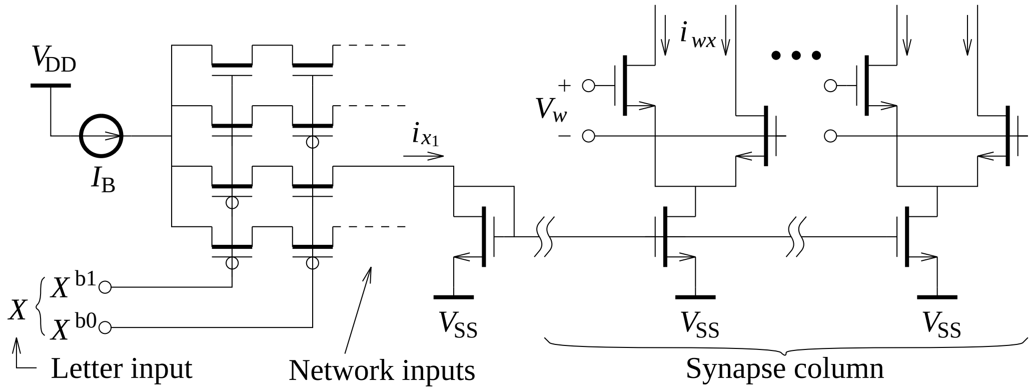


Figure 20^2: Sparse input synapse chip column. *A binary coded letter input X corresponds to, say, four unary coded ANN inputs: The bias current I_B is demultiplexed (left) to one of the four synapse columns (one shown to the right) corresponding to the letter input. Turning off or on the synapse transconductance bias currents act as multiplication by 0 or 1.*

To implement a non application specific sparse input synapse chip, it is necessary that the demultiplexor is reconfigurable — that is, one must be able to change the number of outputs and control lines to fit the applications. The design of such a reconfigurable sparse input synapse chip was done by Jesper S. Schultz (see *Schultz* [211]). As the required input bandwidth to drive a given number of synapse columns scale as $O(\log(N_{\mathcal{A}})/N_{\mathcal{A}})$, one can not exploit both input bandwidth and chip area 100% efficiently for all $N_{\mathcal{A}}$ (though, the input signals can be time multiplexed when $N_{\mathcal{A}}$ is low (to improve the exploitation of the chip area) and it is usually not of paramount importance to fully exploit the input bandwidth when $N_{\mathcal{A}}$ is high). It is therefore necessary to select an "ideal" $N_{\mathcal{A}}$ where the number of synapse rows is tuned to the input bandwidth. Also, when the number of letters in the input alphabet is not a power of two, the binary input coding prohibits a 100% efficient exploitation of both input bandwidth and chip area. Thus, even a reconfigurable sparse input synapse chip is "non application specific" rather than "general purpose".

2.4 Chip measurements

A 4 neuron neuron chip and a 4 × 4 synapse synapse chip has been fabricated in a standard 2.4μm CMOS process. In this section, we shall present measurements on these chips (first published in *Lansner and Lehmann* [130, 131]). A table of the most important chip characteristics can be found in appendix D.1.

2.4.1 The neuron chip

The measurements on the neuron chip were done by John A. Lansner (see *Lansner* [133]). In figure 21^2 measured *neuron transfer characteristics* for different values of the input scale voltage V_{IS} can be seen. The maximum transfer function non-linearity (compared to an ideal tanh) is $D_g \lesssim 2\%$ of the output range and the non-linearity of the derivative is $D_{dg} \lesssim 10\%$.

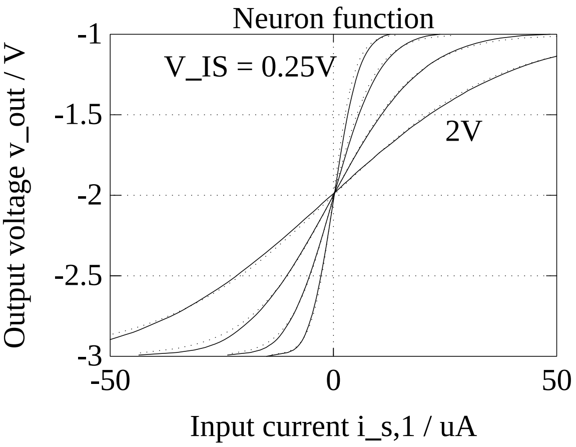


Figure 21^2: Measured neuron transfer function. *Characteristics for different input scale control voltages V_{IS}. The dotted lines are ideal tanh curves. The input offset has been canceled.*

This low non-linearity proves the applicability of the LBM MOST differential pair and the possibility of accurately computing the derivative on the basis of the neuron output (see also section 4.4.2).

2.4.2 The synapse chips

The measured *synapse transfer characteristics* for a single synapse can be seen in figure 22[2]. The characteristics showed a good linearity ($D_{wz} \lesssim 3\%$ or 5 bits accuracy) — with the exception of the case with negative $V_{w_{kj}}$ values and positive v_{z_j} values ($D_{wz} \lesssim 16\%$). This is because it was necessary to lower V_{SS} to ensure a reasonable output current swing (due to a layout error). This non-linearity is by no means prohibitive for the application of the synapse chip; chip-in-the-loop training or a future on-chip learning mechanism can easily compensate for it (cf. eg. *Castro et al.* [42], *Valle et al.* [251]; see also section 5.2.1).

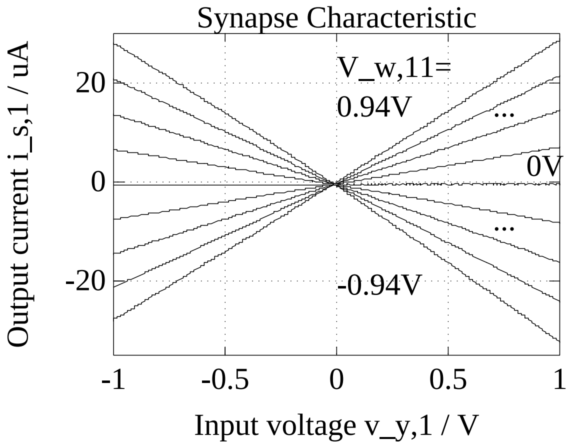


Figure 22[2]: Measured synapse characteristics. *Characteristics for different stored weight voltages V_w. The output offset has been canceled.*

The *weight matrix resolution* was measured to $V_{w\mathrm{res}} \lesssim 2\,\mathrm{mV}$ or 10 bit at the least for a 2 V range of "matrix voltages". Smaller changes are possible but lies below the noise floor at the synapse chip output. For a recall-mode system the resolution is sufficient for a wide range of applications (cf. eg. sections 2.2.3, 5.2.1).

Second generation synapse chip For measurements on a synapse chip with current conveyor based synapse differencing refer to chapter 4.4.

The output *offset currents* on the synapse chip and the input *offset current* on the neuron chip are quite large; approaching in magnitude the maximum synapse output current. The reason could be (in addition to component mismatch) that the opamps have low gains ($< 60\,\mathrm{dB}$), which together with opamp offset voltages of 2 mV would give the measured current offsets. This, however, is not necessarily a major problem (provided that the network is trained and used using the same chips) as the offset currents just displaces the neuron biases. Likewise the matrix offset voltages (which are relative small) could be used as small, random, initial weights when the network is trained. It should be noted that the offset errors are (mostly) non-systematic.

Even with the large current offsets, the chip set characteristics are compatible with many ANN applications (though it is not as general as an ideal, simulated network, of course). The primary limitation of the network is the limited dynamic range of the synapses. For enlarging the application area, ensemble methods can be employed (cf. section 6.3.1).

2.4.3 Chip compound

Interconnecting a synapse- and a neuron chip, the combined transfer characteristics can be measured. This is shown in figure 23[2] for different values of the synapse strength, verifying the synapse-/neuron-chip compatibility. The *step response* of the synapse-neuron combination is shown in figure 24[2]. The delay through one layer of an ANN based on our chips can be measured on this curve: for an 8 bit output accuracy we have $t_{l\mathrm{pd}} \lesssim 2.6\,\mu\mathrm{s}$, corresponding to 6 MCPS per synapse chip. As the synapse chip propagation delay should be largely independent of the number of synapses, a full-size (100 × 100 synapses) synapse chip would be expected to do 3.8 GCPS†.

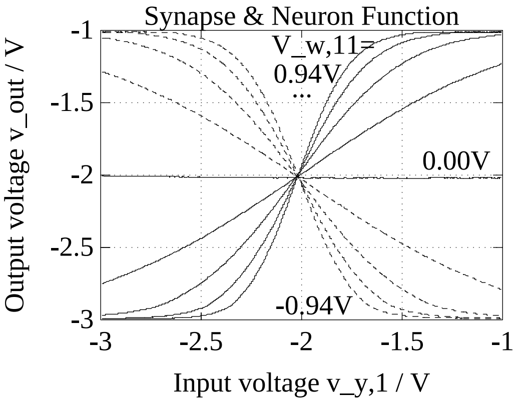


Figure 23[2]: Measured synapse-neuron transfer characteristics. *Characteristics of combined chips (ANN layer) for different stored weight voltages V_w.*

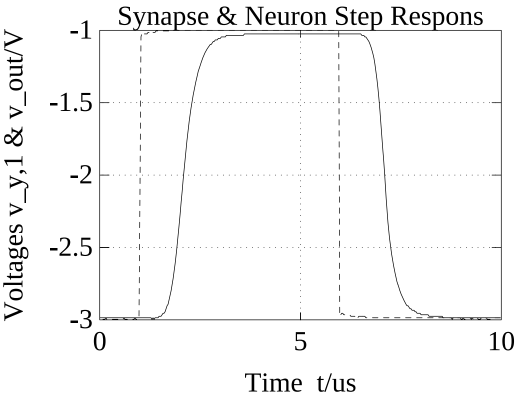


Figure 24[2]: Measured synapse-neuron step response. *Combined chips. The dashed line is the input signal. The effect of five cascaded opamps can be seen in the evidently high order transfer function.*

† For comparison: A typical (1994) workstation would be able to to about 57 MFLOPS (HP 9000/735 125 MHz, *HP Direct* [2]) or almost two orders of magnitude below the computational power of a single chip.

2.5 System design

The system design was done by John A. Lansner (see *Lansner* [133]). To verify the functionality of the chip set, a two layer *test perceptron* based on it was implemented at our institute. Using five synapse chips and two neuron chips, an 8-4-4 architecture was implemented as shown in figure 25^2. (A layout error caused one of the neurons on the neuron chip to be disconnected; also one of the inputs on the synapse chip was malfunctioning. Thus this architecture.) The two linear output neurons was implemented by simple resistors.

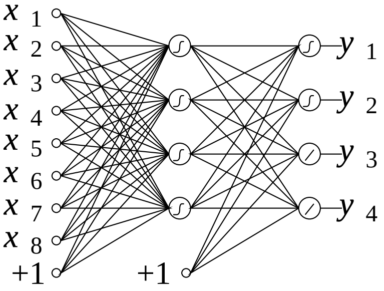


Figure 25^2: Two layer test perceptron. *This simple architecture is capable of solving a large range of non-trivial tasks.*

A standard PC interface was added to test and teach the system (cf. figure 26^2). The synapse strengths are stored in a 16 bit RAM and are periodically refreshed via a 12 bit DAC. Both output and hidden neurons are accessible from the PC via ~ 10 bit ADCs. The inputs are driven by 12 bit DACs.

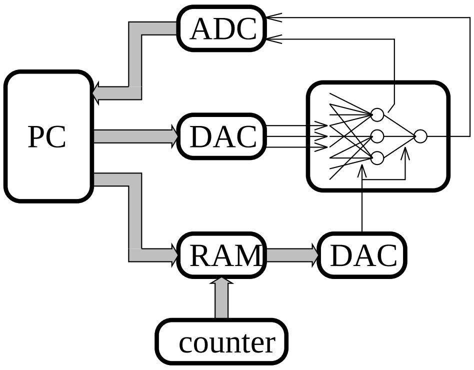


Figure 26^2: Test perceptron system architecture. *To test the ANN system it is embedded in a digital system. In a real-world application, only the weight backup would be digital.*

2.6 System measurements

The system measurements were done by John A. Lansner (see *Lansner* [132, 133]). For a realistic performance evaluation, a well known real-world data set was applied to the hardware system; namely the *sunspot time series* (*Weigend et al.* [261]). This semi periodic time series is the yearly average of dark blotches on the sun, see figure 27^2 (the data is normalized to be within the range $[0, 1]$). Using a *tapped delay line* to feed the *ANN* the sunspot activity of the latest M years, the ANN must predict the activity for the following year. (Note that the data set complexity approximately matches the network architecture, which is essential for obtaining a good generalization ability. Only one ANN output is used.)

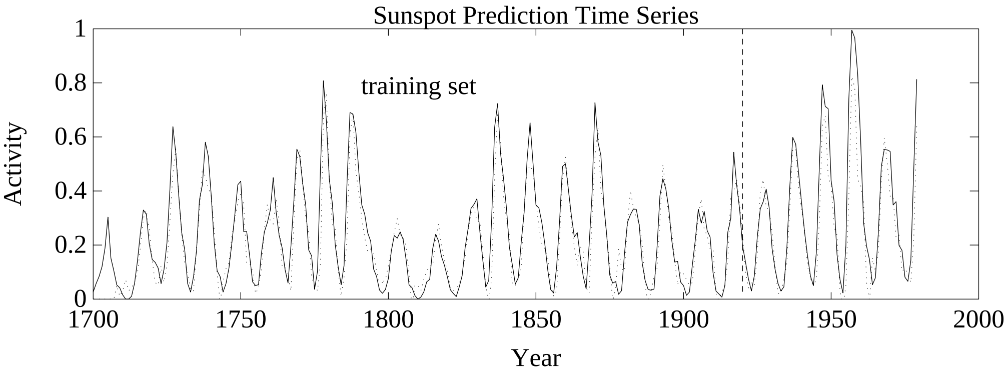


Figure 27^2: Sunspot prediction. *Classic regression problem. The actual sunspot time series (solid) and the sunspot activity as predicted by the hardware ANN (dotted).*

The ANN was trained using a standard chip-in-the-loop back-propagation algorithm. In the calculation of the neuron k transfer function derivative the tanh characteristic was exploited $g'_{k\text{calc}} = \beta_k(1 - (\alpha_k g_k)^2)$, where β_k and α_k are constants and g_k is the actual hardware ANN neuron activation. To ensure $g'_{k\text{calc}} > 0$ (otherwise learning can not take place, cf. the following chapters, *Lehmann* [139, 140]), the neuron activations g_k are scaled such that $\alpha_k g_k < 1$ (to compensate for the neuron output offset- and scale errors). For $\alpha_k \equiv \alpha$ this closely resembles the procedure that we shall apply using learning hardware. Prior to learning, the neuron output ranges are measured to determine the optimal α or α_ks.

The performance of the hardware ANN was compared to that of an "ideal" software ANN with identical architecture but without the non-idealities of the analogue hardware (ie. coarse weight discretization, offset errors, non-linearities, etc.). The *normalized average relative variance* (NARV) of the error on the training set as the training progresses (see appendix B.4) can be seen in figure 28^2. It is noticed that the performance of the hardware ANN is somewhat noisy and slightly worse than that of the software ANN — as would be expected because of the limited accuracy of the analogue hardware. The output accuracy of the hardware ANN is approximately 4–5 bit (see *Lansner* [132, 133]). The NARV of the error on two test sets can be seen in figure 29^2 (the curves with the high NARV are for the atypical test set 1956–1979; the sunspot count in this period does not resemble the rest of the data set very closely). The error will always be lower for the training set than for the test set and the latter exhibits a minimum beyond which further training leads to over-fitting. Notice that the hardware ANN test error is noisy and higher (compared to the training error) than the software test error. This is caused by inabilities of the limited accuracy of analogue VLSI.

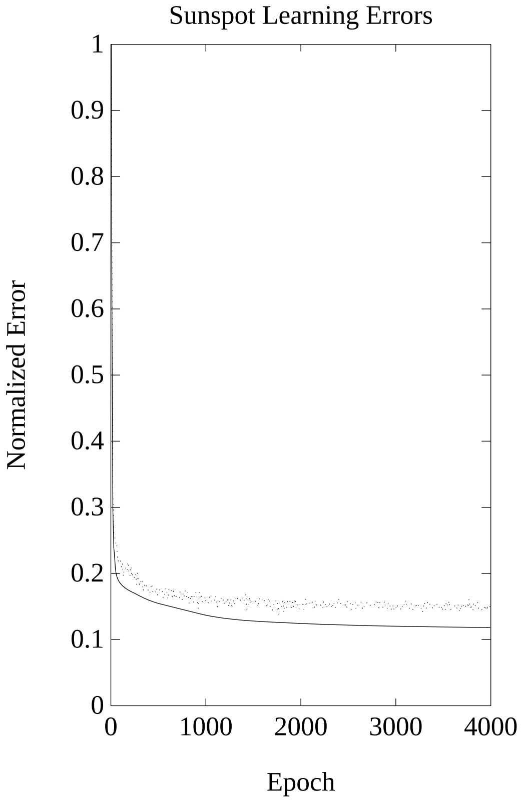


Figure 28[2]: Sunspot learning error. *NARV as function of learning epoch for the hardware ANN (dotted) and an "ideal" software ANN (solid). The error approaches asymptotically a minimum.*

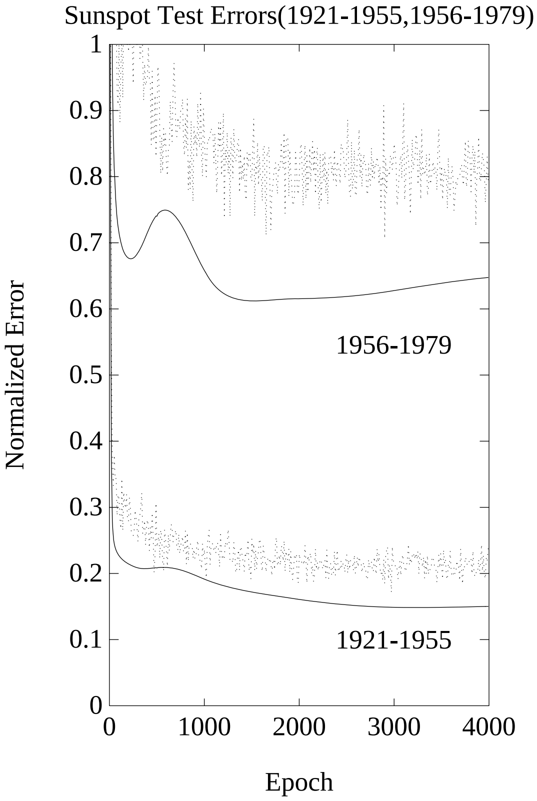


Figure 29[2]: Sunspot prediction error. *NARV as function of learning epoch for the hardware ANN (dotted) and an "ideal" software ANN (solid) for two different test sets. Notice the NARV minima; further training leads to over-fitting.*

The successful system evaluation indicates that implementing hardware ANNs using the proposed architecture is indeed feasible, as claimed above — and as other authors have also noted for different analogue hardware ANNs. The next step is to implement learning hardware for this ANN system.

2.7 Further work

Related to our hardware implementation of the artificial neural network, there are several issues that need to be considered before a volume production would be possible. We shall display some of these in this section.

2.7.1 Process parameter dependency canceling

As noted in section 2.2.5, we must eliminate the undocumented process parameters when using parasitic effects of our semiconductor process — such as the MOSFET operated in the *lateral bipolar mode*. In the case of the hyperbolic tangent neuron, the "undocumented process parameter" in question is the forward emitter-collector current gain, α_{FC} (cf. (4^2)) of the LMB MOST differential pair. In figure 30^2 a simple regulating circuit is seen: A single LBM MOST emulates a differential pair that is driven to saturation (such that $\tanh(.) \equiv 1$). The (non-substrate) collector current in this LBM MOST is thus the *effective bias current* $I_{\mathrm{Beff}} = I_{\mathrm{B}}\alpha_{\mathrm{FC}}$. Subtracting a reference current I_{Bref} from this and dumping the difference into a high impedance node gives a voltage, V_{AB}, with which to drive the bias transistor. V_{AB} is distributed to all the neuron LBM MOST differential pairs which will thus have effective bias currents of approximately I_{Bref}. A loop gain of $A_{\mathrm{L}} = -g_{m\mathrm{B}}\alpha_{\mathrm{FC}}/2g_{\mathrm{dsI}} \approx -100$ is easily implemented using simple MOS transistors as indicated in the figure. This corresponds to an effective bias current error of $(I_{\mathrm{Bref}} - I_{\mathrm{Beff}})/I_{\mathrm{Bref}} = 1/(1 - A_{\mathrm{L}}) \approx 1\%$. As V_{AB} is meant to be distributed to a large number of neurons (ie. bias transistors and differential pairs (α_{FC}s) distributed over a large area are required to match) there is no point in increasing the gain for better accuracy. For a sufficiently high number of attached neurons, the C_{gs}s of the bias transistors will act as compensation capacitance (C_{C}).

Using a simple regulating loop to cancel unknown (global) process parameters is a principle with general applicability. The principle is illustrated in figure 31^2. The references can be on-chip internal references; for instance temperature compensated (which would cancel the temperature dependency of the "unknown" block). An off-chip reference would be used if a signal (input or output) needs to be within some absolute range (eg. critical inter-chip signals in multi-chip systems).

The hyperbolic tangent neuron *output range* transresistance (cf. figure 15^2) is an evident component on which to apply *current factor* dependency canceling. If we wish to calculate the neuron derivative off-chip as $1 - g^2$, we must ensure that the neuron output is in the absolute range $[-1, 1]$; thus the *output reference* would be a "1". The input reference must be the LBM MOST differential pair output current that corresponds to a neuron output of "1", ie. I_{Beff}.

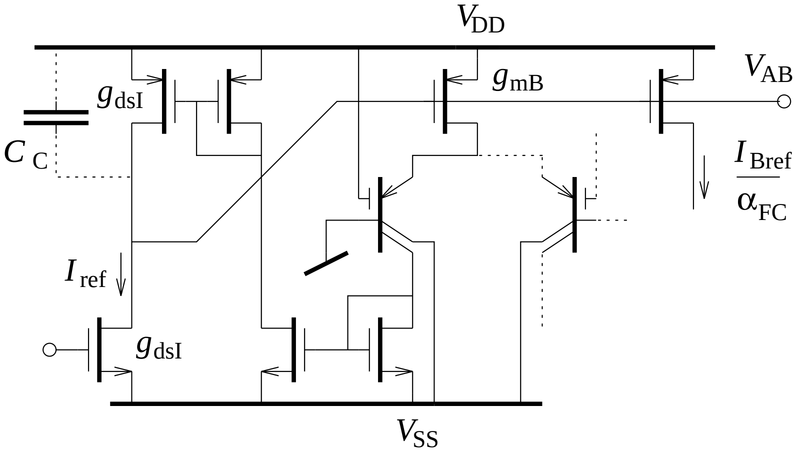


Figure 30[2]: Non unity e-c current gain canceling. *Circuit for LBM MOST differential pair. The saturated BJT differential pair output current (ie. the effective tail current) is compared to a reference current, amplified (via the high impedance "V_{AB}" node) and fed back to the tail current source. A single BJT or a BJT pair with one transistor turned off can be used as reference BJT "pair".*

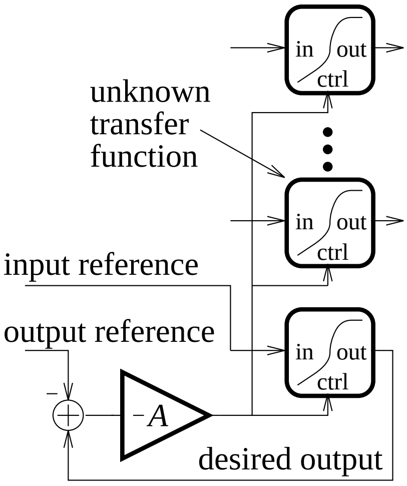


Figure 31[2]: General process parameter canceling circuit. *Principal schematic of the method applied to cancel non-unity e-c current gain. The unknown transfer function blocks must be matched.*

2.7.2 Temperature compensation

The excessive use of transconductances (outside feedback loops) in analogue neural networks (eg. synapse multipliers) makes temperature drift a major concern (this was actually a problem in the experimental ANN system solving the sunspot regression problem). The temperature dependence of the MRC, for instance, is primarily determined by the mobility μ which is proportional to $T^{-3/2}$ for low substrate dopings at room temperature (*Sze* [235]). ($\mu \propto T^{3/2}$ for high substrate dopings at room temperature.) In the temperature range [300 K, 370 K], this corresponds to a synapse strength drift of 27 %. In most real-world applications such a large temperature drift is unacceptable (unless the temperature is known to be constant as in implanted devices) and temperature compensation must be employed.

We can regard the temperature as a time varying "undocumented process parameter". Thus temperature compensation can be implemented as above, provided that a temperature independent reference is available (such as a bandgap reference).

Of primary interest in relation to temperature drift is the effective neuron activation slope (or, equivalently, the effective maximum synapse weight) assuming the output range is well defined. As noted in section 2.3.3, process variation influences on the effective maximum synapse weight can be canceled by the learning algorithm; assuming constant temperature, that is. However, in the important special case of classification, the neurons are usually saturated after the learning phase. In this case it is the relative (rather than the absolute) synapse strengths that describes the network and if we can assume a constant system wide temperature, such a system will function independently of the temperature†. For regression problems, on the other hand, where analogue outputs are required, temperature compensation is unavoidable‡.

2.7.3 Other improvements

In addition to the important process parameter/temperature variance compensation, several issues are subjects for improvement of the developed cascadable ANN chip set. These can be found in appendix D.1.2.

2.8 Summary

In this chapter we designed a cascadable, analogue VLSI artificial neural network. Several network models and topologies from the literature were displayed. Deterministic, first order neurons using continuous valued currents and voltages for signalling was chosen and a cascadable architecture placing neurons on one chip and synapses on another selected for generality.

Essential building block components, memories, multipliers and thresholders, were reviewed next. No good analogue *memories* are presently available. For adaptive systems simple capacitive storage seems the best choice, though weight refresh is a problem. We have chosen to use a digital RAM weight back-up memory — even though this puts severe restrictions on the efficiency of the learning scheme. Another possibility would be to use digital storage in combination with an analogue adjustment. Using four quadrant *multipliers* is not strictly necessary though

† In our system, the temperature dependency of the synapse multipliers cancels out with those of the neuron input scale transresistances. The resulting temperature dependency on the effective neuron activation slope will be that of $V_{\mathrm{t}} = kT/q$ which varies 23 % in the above temperature range.

‡ This is not absolutely true: using *linear* output neurons, and assuming the hidden neurons are saturated — a situation often found in regression networks (see eg. *Krogh et al.* [125], *Svarer et al.* [234], *Pedersen and Hansen* [186]) — temperature dependencies could be canceled.

probably advantageous; we use the very compact MRC. It was noted that, in order to be absolutely general, the dynamic range of the synapse multiplier in a scalable system would have to be infinite; also, the output offset error was important. We propose (though we shall not employ this) to use a highly non-linear multiplier to improve the dynamic range and thereby the relative output offset error. As the *neuron activation function* we chose a hyperbolic tangent function as not to restrict the implementations of learning hardware (the derivative is easily computed).

The design of our ANN chip set was displayed. Further, a "sparse input synapse chip" was proposed; this chip architecture exploits the limited input bandwidth efficiently for problems with a discrete input alphabet. Measurements on a fabricated chip set was displayed, indicating a possible $\sim 3.8\,$GCPS/chip for a full-scale chip set. Offset errors (primarily synapse output- and neuron input-) were large, though tolerable: Training results from a 8–4–4 test perceptron implemented using our test chips and trained via a PC were displayed: the hardware network learning error was slightly worse than that of an ideal software net.

Finally, the importance of process parameter dependency canceling in real systems was stressed (including temperature compensation). A sample circuit for eliminating the unknown forward emitter current gain of LBM MOSTs was displayed.

Chapter 3

Preliminary conceptions on hardware learning

The basic analogue artificial neural network architecture now being defined and tested, we shall turn our attention to the implementation of analogue learning hardware for this ANN. At first some general conceptions on hardware learning will be presented in this chapter: Firstly, we shall consider the tolerable amount of hardware spent on the learning implementation. Secondly, the choices of learning algorithms that we will implement are discussed. Finally, we give some general considerations on the implementation of ANN learning algorithms using analogue VLSI, in relation to the limited precision of this technology.

3.1 Hardware consumption

In chapter 1 we argued that at least two niches for analogue hardware implementations of learning algorithms exist:

- Massively parallel, possibly adaptive, application specific systems having a parallel real-world interface.
- Small, adaptive, low power, application specific systems with a real-world interface.

Now, for both niches the tolerable *amount of hardware* put into the *learning algorithm* is application dependent. At one extreme, when it is crucial to exploit inherent parallelism, when speed rather than cost is important, and when the learning scheme is used excessively, there is no upper bound on the amount learning hardware (other than it must be realistic to implement). A massively parallel implementation of the learning algorithm is the choice. However, a vast amount of learning hardware can severely limit the applicability of the learning scheme for certain applications — the learning hardware lies idle when the system is used in recall mode and it reduces the number of integrated synapses for a given silicon area. So, at the other extreme, when cost or power consumption† rather than speed is important, and when the learning scheme is employed only occasionally, the amount of learning hardware must be kept as small as possible.

In this work we shall implement learning algorithms belonging to both categories: a hardware efficient implementation of back-propagation and a parallel (though not fully parallel) implementation of real-time recurrent learning.

† For a given learning algorithm the amount of *energy* used by the learning hardware to process a single input/output pattern is independent of the parallelism of the implementation (ideally; in reality a serial implementation will most probably consume more energy than a parallel implementation). Thus if power consumption is *the* concern and if the learning algorithm is employed only occasionally, a fully parallel implementation with a power down circuit could be a solution.

3.2 Choice of learning algorithms

The choice of learning algorithm is highly dependent on the application at hand. Different learning algorithms operate on different network architectures using different optimization procedures for different goals. In addition, new learning algorithms arise continuously and it has thus been argued that VLSI learning hardware should be adaptable to changing learning algorithms (cf. eg. *Ramacher* [192]). However, as for the reconfigurability of the analogue ANNs, a high degree of programmability of analogue learning hardware does not comply with the technology: much hardware would lie idle or would be used to configure the algorithm rather than participate in the computations. Neither for massively parallel- nor low power implementations is this acceptable; it would compromise the advantages of the analogue technology (algorithmic *variations* can be included when possible and appropriate, though; cf. the inclusion of both entropic and quadratic cost functions in chapter 5). It is not possible — as it is in principle for the ANN itself — to implement a general purpose, analogue ANN learning machine. Thus, the learning algorithm should be implemented specifically for the application at hand. Ideally.

An application specific learning algorithm does not comply with our *general purpose* building block ANN chip set, though; it would sacrifice the generality of the ANN/learning chip set (for the particular application this would not matter, of course). If we do not have a specific application in mind, we must then choose our learning algorithm carefully in order that it can be applied to a large range of applications (*Murray* [174]). The learning algorithm must possess the same properties as the neural network model: desirably it should be

- General purpose
- Simple
- Suitable for the technology

The simplicity is very important. Partly because a complex learning model requires much hardware but primarily because of the limited precision of the technology: other things being equal, the more hardware participating in the calculations the larger the accumulated errors (the finer points of a complex learning scheme would quickly be insignificant compared to the errors). Needless to say, the learning algorithm must also map in a simple way on silicon (it should rely on local communication and must not consume too much area/memory, etc.).

Choosing a learning algorithm means choosing an application area. Artificial neural networks can be applied to a wide range of applications (see eg. *Sánchez-Sinencio and Lau* [206]). For instance:

- Classification or
- Pattern recognition
- Regression
- Example described problems
- Function approximation
- Associative memories
- Feature Mapping

- Optimization
- Control
- Data compression

In this work we shall predominantly be interested in applications classified as *classification* or *regression* problems. These important classes of problems are often successfully solved using artificial neural networks. For instance the prediction of splice sites in human pre-mRNA molecules (*Brunak et al.* [29]), pig carcase grading (*Thodberg* [239]), implantable heart defibrillators (*Jabri et al.* [108]), and high energy particle-detector track-reconstruction devices (*Masa et al.* [157]). For such problems *supervised learning* is usually employed. Choosing such a learning algorithm for our system will make it applicable to a broad range of applications (though not "general purpose"); we shall thus commit the following text to the implementation of supervised learning algorithms. (The implementation of unsupervised learning is also interesting; however, that is not our story.)

3.2.1 Gradient descent learning

A very simple approach to supervised learning when the ANN has differentiable neuron activation functions is to use *gradient descent* (cf. appendix B.3). Defining a *cost function* $\mathcal{J}$ which measures the cost of the network error, one adjust the free parameters of the ANN (ie. the synapse strengths, neuron thresholds and slopes, etc.) such that $\mathcal{J}$ decreases most rapidly; ie. down the gradient:

$$\Delta w_{kj}(t) = -\eta \frac{\partial \mathcal{J}(t)}{\partial w_{kj}(t)} ,$$

where the *learning rate* η is a small positive constant. Real gradient descent requires the cost function to be a function of all training patterns ($\mathcal{J}_{\mathrm{tot}} = \sum_{\mathrm{ptns}} \mathcal{J}_{\mathrm{ptn}}$; batch learning). Often, though, the weights are changed on the basis of the instantaneous cost function, evaluated for one pattern only (on-line learning). For small learning rates the methods are equivalent (compare to the *Gauss-Seidel* method of solving linear equations numerically (*Press et al.* [189])).

Gradient descent is not a very good optimization technique (cf. *Hertz et al.* [95], among others). Most notably it

- Has a tendency to get stuck in local minima of the cost function
- Converges slowly

For improved conversion time, algorithms as the *conjugate gradient method* or *quasi-Newton* can be employed (see also *Press et al.* [189]). These methods have the advantage that they use the first order cost function derivatives (as gradient descent), which can be computed quite efficiently (cf. below), for computing the weight changes. Using the second order derivatives, the full *Hessian* matrix, or approximations to these can likewise improve learning time significantly (*Hertz et al.* [95], *Buntine and Weigend* [36], *Pedersen and Hansen* [186]). Other methods, such as *simulated annealing*, are also quite interesting. Simulated annealing searches the weight space with occasional uphill moves, hence does not as often as gradient based methods get stuck in a local minimum.

The expense of these algorithms compared to simple gradient descent is increased computational cost.

In spite of the poor performance, gradient descent has had most vigorous interest in the neural network society. It has been thoroughly analysed and tested in connection with artificial neural networks and a wealth of improvements to clean gradient descent has emerged. Further, it is very popular among application people; quite impressive results have been obtained by the method (eg. *Brunak et al.* [29], *Masa et al.* [157], see also *Williams and Zipser* [268], *Wulff* [271]). Gradient descent is central in the present state of neural network art. Finally, the method is quite simple and maps topologically very nicely on VLSI — which suggest that an analogue hardware implementation would be possible.

This is the *motivation for using gradient descent*, which we shall do; we should have a fair chance to do an implementation in our limited precision technology (though cf. section 3.3 and the following chapters) and "users" (application people) would know what to expect from the implementation; they would buy our solution.

It should be noted that a supervised learning system (eg. using gradient descent) can be extended in a straight forward manner to implement *learning with a critic* which can be used for prediction and control.

Other authors have also looked at the implementation of learning in analogue hardware. Many of these use gradient descent based algorithms; though not every one. *Alspector et al.* [9], for instance, use a simulated annealing scheme (Boltzmann machine), *Card* [40] uses Hebbian learning (unsupervised learning) and *Macq et al.* [154] use Kohonen feature mapping.

3.2.2 Error back-propagation

The usual network architecture applied to *classification* and *regression* problems is the *multi layer perceptron* (*MLP*). The error *back-propagation* learning algorithm (*Rumelhart et al.* [199], *Hertz et al.* [95], chapter 4) is a formulation of gradient descent which maps on feed forward ANN architectures (eg. the *multi layer perceptrons*). From an analogue VLSI point of view it has a number of draw backs, though (in addition to drawbacks of gradient descent learning); for example:

- The neuron derivative needs to be computed.
- It is very sensitive to offsets on various signals (most notably the weight changes).
- The learning rate must be rather small for convergence.

It should be noted that these problems are not particular for back-propagation; they apply to many other algorithms as well (see also section 3.3). The problems shall be addressed in the following chapters. From an analogue VLSI point of view back-propagation also has a number of advantages (which also applies to many other algorithms as well); for example:

- Fully parallelizable.
- Uses local signalling.
- Relatively simple.

The implementation of back-propagation in analogue VLSI has been considered by several authors (eg. *Valle et al.* [251], *Wang* [256], *Cho et al.* [46], *Lehmann* [142]). Other authors have chosen to derive new algorithms, having the implementation in analogue VLSI in mind. These alternatives to back-propagation include *weight perturbation* and *virtual targets*:

Weight perturbation By some considered *the* standard algorithm for analogue VLSI (as opposed to *the* standard algorithm, back-propagation, for simulated networks), *weight perturbation* (*Jabri and Flower* [107]) is a difference quotient approximation to gradient descent learning. Weight perturbation is inspired by the fact that back-propagation (i) usually requires three times the synapse hardware of a recall mode system (though cf. chapter 4) and (ii) requires the computation of the neuron activation function derivatives. Given an (instantaneous) cost function $\mathcal{J}(\underline{\underline{w}}, t)$, weight perturbation prescribes the weight changes

$$\Delta w_{kj}(t) = -\eta \frac{\mathcal{J}(\ldots, w_{kj}(t) + \Delta w_{\mathrm{pert}}, \ldots, t) - \mathcal{J}(\ldots, w_{kj}(t), \ldots, t)}{\Delta w_{\mathrm{pert}}} ,$$

where Δw_{pert} is the *weight perturbation constant* (possibly weight dependent).

This very direct way of approximating the cost function derivative has the additional advantage that *any* kind of non-linearity, offset error, etc. in the recall mode network are transparent to the algorithm; their impact on the cost function derivative will be included by the algorithm (in contrast: implementations of back-propagation usually requires reasonably linear synapses as neuron derivatives are used to compute the cost function derivative). Assuming that the weights are externally accessible, very little hardware and no extra signal routing is required for a hardware implementation of weight perturbation. The drawback of the algorithm is that it is computationally expensive ($O(N^4)$ per training pattern or time step) and not fully parallelizable (serial weight update). The latter problem has been addressed by *Flower and Jabri* [72]; in *summed weight neuron perturbation* a speed up of $O(N)$ can be achieved at the expense of a $O(N)$ storage requirement.

Matsumoto and Koga [160] use an algorithm very similar to weight perturbation: Oscillating all the weights concurrently at different frequencies the weight derivatives — and thus the weight changes — can be computed simultaneously. (This procedure introduces new problems related to accurate band pass filtering, intermodulation, and system bandwidth requirements, though.)

It should be noted that weight perturbation can be applied to any network topology; not just MLPs.

Virtual targets While not exactly a gradient descent learning algorithm, the *virtual targets* MLP learning algorithm (*Murray* [174]) uses gradient descent for each layer, locally. The weight change rule is the same as for back-propagation (cf. (9^4)) (assuming quadratic cost function):

$$\Delta w_{kj}^{l}(t) = \eta g'(s_k^l(t))\varepsilon_k^l(t) z_j^l(t) \,.$$

The neuron error $\varepsilon_j^l(t)$ is computed differently from back-propagation: In virtual targets *all* neurons are assigned target values for all training patterns. The targets for the hidden neurons are initialized to random values and are developed during training according to:

$$\Delta d_k^l(t) = \eta_{\mathrm{tgt}} \sum_j w_{jk}^{l+1} \varepsilon_j^{l+1}(t) \,,$$

where η_{tgt} is a *target learning rate*. When a pattern is successfully learned the algorithm must cease to react on this pattern (otherwise the hidden neuron activations would drift, causing the pattern to be unlearned); if the classification of the pattern is forgotten during further training, reaction on the pattern is resumed.

Explicitly using target values for the hidden neurons can improve the learning speed and the algorithm seems to possess an ability to “jump out” of local minima. The disadvantage of the algorithm is the requirements of hidden neuron access and target storage; furthermore, the scheme is more complicated than, for instance, back-propagation or weight perturbation.

Being “central in the present state of neural network art”, the *implementation* of the *gradient descent* back-propagation learning algorithm in analogue VLSI is an important issue of integrated ANN research. A problem which we shall address in the following chapter — with emphasize on the issues of *hardware cost*, *derivative computation*, and *weight updating schemes*.

3.2.3 Real-time recurrent learning

Though very popular for pattern recognition etc., feed-forward ANN architectures have their limitations. A more general set of architectures are *recurrent networks* (*recurrent artificial neural networks*, *RANN*s). If no constraints are put on the connections (as, for instance, symmetric connections found in many architectures) recurrent network architectures are potentially *very* powerful. They have the ability, for example, to deal with

- *Temporal information*
- *Storing of data*
- *Attractor dynamics*

Using supervised learning (read gradient descent) to train recurrent neural networks, these can be taught to recognize sequential structures (eg. Reber grammars, *Smith and Zipser* [225]) to imitate finite automatons (eg. Turing machines, *Williams and Zipser* [268]) or to simulate strange attractors (eg. Mackey-Glass series, *Wulff* [271]). Recurrent neural networks can also be used instead of, say, tapped delay line perceptrons (eg. *Pedersen and Hansen* [186]): If a temporal pattern recognition task is dependent on a few, unknown, temporarily wide distributed input samples, a recurrent network can solve the task using much fewer connections — which can be very important; especially if only a relatively small training data set is available. (This is actually the application driven part of *our* motivation for implementing a recurrent network learning scheme: the prediction of splice sites in pre-mRNA molecules mentioned in section 2.3.3 can be solved using a 10–50 neuron RTRL network (*Brunak and Hansen* [31]).) The trouble with recurrent networks is that they are usually very hard to train.

To make available, to the users of our ANN system, the potent possibilities of recurrent neural networks we shall, in addition to the implementation of back-propagation learning, investigate the implementation of a learning algorithm for our ANN architecture connected in a recurrent way. Several examples of gradient descent like algorithms for recurrent networks exist in the literature (see eg. *Hertz et al.* [95]). As not to compromise the generality of our cascadable ANN architecture (more than absolutely necessary) we must choose an algorithm with a most general applicability. *Real-time recurrent learning* (*RTRL*) (*Williams and Zipser* [267, 268], chapter 5) is a good choice. This algorithm has a number of advantages:

- *General ANN architecture.* The RTRL algorithm is formulated for a completely general ANN architecture: A fully interconnected network. The network will organize itself to reflect the structure of the application during learning. If any structure of the problem to be solved is known a priori (which should then be reflected in the network architecture) the algorithm can as well teach a constrained architecture. For the generality of our ANN/learning chip set, this is very important; it is possible to implement a ANN/learning chip set applicable to any network topology.
- *Application invariant.* When the network topology and size is determined, the learning scheme is determined; independent of the application. The storage requirement for other RANN learning algorithms (as *back-propagation through time* or *time-dependent recurrent back-propagation*) is often proportional to the maximum sequence length (the memory of the system) that needs to be processed. An application independent learning hardware architecture is important to our "general" system.
- *Real-time training.* Or *in-the-flight training.* Unlike many other algorithms, RTRL does not use a training phase and a recall phase; RTRL functions in-the-flight, training the system during use. This is essential to *adaptive systems* but also very important to analogue implementations in general: *storing the training patterns* for batch learning is hostile to an analogue implementation; the storage must (most probably) be in digital RAM and is not in compliance with the "real-world interface" requirement of analogue learning hardware. Of

course, if training patterns are not held in store for teaching, the environment in which the system resides must be able to generate representative learning sequences when the system is to be taught†.

- *Hardware compatible.* The algorithm is parallelizable; fully or partial, and it turns out that the architecture maps very nicely on hardware. Furthermore, it is computationally a fairly simple algorithm which is suitable for analogue hardware.
- *Powerful.* A range of impressive problems have been solved using RTRL (the above cited examples are solved using RTRL).

It also has a number of disadvantages:

- *Computation requirements.* RTRL requires an order $O(N^4)$ computation primitives for each training example; or, as the required number of examples scale as $O(N^2)$, at the least an order $O(N^6)$ computation primitives to train the network. This is the major drawback of RTRL. RTRL requires parallel computing even for relatively small systems.
- *Memory requirements.* RTRL requires memory of an order $O(N^3)$. Even for (semi) parallel implementation this will limit the network size.
- *Trainability.* Recurrent networks are harder to train than feed forward networks (and should thus be used only when that type does not suffice). Some argue (eg. *Tsoi et al.* [248]) that the completely general fully connected architecture is too hard to train and one should select less general recurrent architectures. Note, however, that one must *always use a priori knowledge* of any kind in the problem at hand; thus the general architecture should be selected when nothing is known of the solution to the problem.

In chapter 5 we shall investigate the implementation of the RTRL algorithm — with emphasize on the issues of *hardware cost*, *derivative computation*, and *weight updating schemes* as for the back-propagation implementation.

† Though usually meant to be used off-line, back-propagation employed by example can be used in a similar in-the-flight (on-line) manner — thus, it is not of paramount importance to hold the training patterns in store when using (MLP) back-propagation.

3.3 Hardware considerations

Implementing artificial neural networks using limited precision technologies as analogue VLSI is, by now, considered a fairly straight forward matter. The inherent adaptability of the ANN systems can accommodate for non-idealities. For learning algorithms this is not so. Several authors have noted that learning algorithms presently available to the analogue designer are typically very sensitive to certain kinds of non-idealities displayed by, for instance, analogue VLSI. In this section we shall have a look at the most important ones. The discussion is based on work primarily concerning gradient descent like algorithms; some of the issues are specific to these kind of algorithms (as the derivative computation) while other probably are generally applicable (as the weight change offset).

The allowable non-ideality magnitudes are dependent on each other as well as being application, topology and size dependent. The observations below are only qualitative.

Weight discretization Most ANN implementations use connection strengths with a *discrete number of values*; either as a consequence of a weight refreshing scheme or because of digital, non-volatile weights used in analogue/digital hybrids. A discrete number of weight values limits the problem space solvable using the network, of course; or in other words: this will degrade performance (*Xie and Jabri* [272], *Lehmann* [139, 140]).

Much more important, however, is the fact that the smallest weight change is restricted to one LSB: weight values computed by gradient descent learning algorithms, for instance, are constituted of many small weight changes. Therefore much higher weight resolution is required during learning than in recall mode. To meet this demand (assuming the ANN has weight resolutions tailored to the recall mode), the learning hardware can have access to a high precision version of the synapse strengths on the network (*Hollis et al.* [97], *Asanovic and Morgan* [17], see also section 2.2.4). Another procedure is to use *probabilistic rounding*: For computed weight changes $|\Delta w|$ smaller than 1 LSB, a 1 LSB weight change is carried out with a probability $|\Delta w|/1\,\mathrm{LSB}$ (*Höhfeld and Fahlman* [96]). See also section 2.2.3.

Dynamic range When trained using a gradient descent like algorithm (for instance; eg. on a pattern recognition problem) the synapse strength magnitudes tend to grow with time. This can easily exhaust the limited *dynamic range* of the synapses (see also section 2.2.4). Using logarithmic coded synapse strengths (*Hollis et al.* [99]) can increase the effective dynamic range. Also, weight decay can be employed or the neuron gain can be increased during learning (*Hollis et al.* [97]) to postpone weight exhaustion.

Derivative computation Gradient descent like algorithms often need the *neuron derivatives* to compute the cost function gradient. This is a major concern in many analogue implementations. Several authors have reported that learning can take place even for *very* approximate neuron derivative calculations (eg. *Valle et al.* [251]). (The learning trajectory will not follow the gradient in this case, of course.) One property must the calculated neuron derivative possess, though: it *must have the right sign.* This could typically be a problem for saturated sigmoid neurons for which the actual derivative is close to zero: if the calculated derivative is negative, a gradient descent weight updating rule would result in an *up-hill* cost function climb, possibly irrecoverably bringing the neuron deeper in saturation (*Lehmann* [139, 140], *Krogh et al.* [125], see also *Woodburn et al.* [270]). A small positive offset deliberately introduced to the neuron derivative calculation circuit can prevent this hazard (*Lehmann* [142], *Shima et al.* [220]). This would also enable saturated neurons to be taught (still using a gradient descent like algorithm) which is otherwise prevented by the zero derivatives.

Offset errors Possibly the most problematic issues of analogue VLSI implementations of ANN learning algorithms are the ever present *offset errors.* While offset errors on some signals (for instance the neuron net inputs) are insignificant, they can completely prevent learning when present on other signals. Most sensitive to offset errors are:

Weight change offsets. If the weight change offset error, $\Delta w_{\rm ofs}$, is comparable, in some sense, with a typical weight change, Δw, the weights would develop over time as $w_{kj}(t) = w_{kj}(0) + t\Delta w_{\rm ofs}$ rather than governed by the learning algorithm; for sufficiently large $\Delta w_{\rm ofs}$ and t learning is impossible (*Lehmann* [139, 140], *Montalvo et al.* [168], *Withagen* [269]).

Neuron error offsets. Offsets on the errors of output neurons just displaces the target values (which can be serious enough for analogue outputs). Offsets on the errors of hidden neurons can be more severe, though: such offsets cause learning to take place on the hidden neurons even after the output error is zero; ie. the solution to the training problem is not a stable state of the system (*Lehmann* [139, 140]; see also *Murray* [174]).

Cost function offsets. For weight perturbation, where the learning is controlled by the computation of the cost function, offset errors on this quantity will, as above, cause the solution to the training problem to be an unstable state of the network; which degrades learning (*Montalvo et al.* [168]).

Learning rate For simulated networks the *learning rate* is usually chosen quite small for good learning. Smaller than typically compatible with analogue VLSI: In the presence of weight discretization and weight updating offsets, the learning rate must be so large that these effects are "small' compared to typical weight changes (*Tarassenko et al.* [238]). Also, if a learning scheme is used to refresh a purely capacitive synapse storage, the typical weight change must be large compared to the memory droop rate (see also *Hansen and Salamon* [91], *Lehmann and Hansen*

[145]).

Noise Analogue systems are noisy. While *noise* is beyond doubt a nuisance in many analogue signal processing systems, it can actually be an advantage in ANN learning systems. The limited “resolution” (ie. signal to noise ratio) of analogue systems is not comparable to the limited resolution (in number of signal processing bits) of digital systems. The presence of noise can improve *learning* (make the system “jump” out of local minima because of occasional random movements), improve *generalization ability* (the network is “forced” to locate the underlying structure of the training data in the presence of noise) and improve *fault tolerance* (the information tend to spread more evenly among the synaptic connections) (*Edwards and Murray* [67], *Jim et al.* [113]; see also *Hertz et al.* [95] and *Qian and Sejnowski* [190]).

Chapter 4

Implementation of on-chip back-propagation

The inclusion of back-propagation learning on our ANN chip set using a small amount of additional hardware is the objective of this chapter. The learning algorithm is first described, after which it is shown how it can be mapped on our ANN architecture — and our hardware efficient solution is presented. The design of an experimental VLSI chip set (a synapse- and a neuron chip) and measurements on this are presented next. We also present the design of a complete back-propagation system including the learning hardware not present on the chips themselves. Problems in relation to derivative computation and learning in a system using digital weight backup are discussed. The novel *non-linear back-propagation* learning algorithm is displayed and we show that the algorithm has several nice properties in relation to an analogue hardware implementation; hardware for a *very* low-cost implementation is presented. Reflections on future work are then given: A chopper stabilization technique for elimination of offset errors is proposed and the inclusion of algorithmic variations in our system is outlined. A summary concludes the chapter.

4.1 The back-propagation algorithm

The error *back-propagation learning* (*BPL*) algorithm is a supervised, gradient descent algorithm (cf. appendix B.3). In this section we describe the basic algorithm and display modifications typically applied to it.

4.1.1 Basics

The error back-propagation learning algorithm for a layered feed-forward neural network (*multi-layer perceptron*, MLP, cf. appendix B.1, figure 75^{B}) can be described as follows (*Hertz et al.* [95], *Rumelhart et al.* [199]): Given an *input vector* $\underline{x}(t)$ at time t, we can write the neuron k activation in layer l (32^{B}) as

$$y_k^l(t) = g(s_k^l(t)) = g\left(\sum_j w_{kj}^l(t) z_j^l(t)\right), \tag{6^4}$$

where we assume $g_k^l(.) \equiv g(.)$ and where

$$z_j^l(t) = \begin{cases} x_j(t)\,, & \text{for } l = 1 \\ y_j^{l-1}(t)\,, & \text{for } 1 < l \leq L \end{cases}.$$

The neuron biases are implicitly given as the connection strengths from constant inputs $z_0^l = 1$. Given a set of *target values* d_k for the neurons in the *output layer*, L, we define the *neuron errors* (when using a quadratic cost function (35^{B})) as

$$\varepsilon_k^l(t) = \begin{cases} d_k(t) - y_k^l(t)\,, & \text{for } l = L \\ \sum_j w_{jk}^{l+1}(t)\delta_j^{l+1}(t)\,, & \text{for } 1 \leq l < L \end{cases}, \tag{7^4}$$

where the *weight errors* (or "deltas") are defined as

$$\delta_j^l(t) = g'(s_j^l(t))\varepsilon_j^l(t)\,. \tag{8^4}$$

Using a discrete time *on-line learning* updating scheme, the connection strengths should then be changed according to the *weight updating rule*:

$$w_{kj}^l(t+1) = w_{kj}^l(t) + \Delta w_{kj}^l(t) = w_{kj}^l(t) + \eta\delta_k^l(t)z_j^l(t)\,, \tag{9^4}$$

where η is the *learning rate.*

4.1.2 Variations

Though still often serving as *the* reference for new MLP learning algorithms, the efficiency of the basic back-propagation algorithm has been questioned by several authors: the algorithm is slow and the network often gets stuck in local minima (eg. *Fahlman* [69], *Hertz et al.* [95], *Haykin* [93]). For this reason a wealth of back-propagation-like algorithms (or improvements of the algorithm) has emerged. Many of these improvements do not alter the basic *topology* of the algorithm and are thus easy to incorporate in the VLSI architectures that we shall describe shortly. However, the *cost* of the incorporation is very dependent on the exact implementation (ie. whether digital/analogue weights are used, whether parallel/serial weight update is used, etc.). The most common modifications of the algorithm (which can also be applied to many other learning algorithms) include (*Hertz et al.* [95], *Haykin* [93], *Plaut et al.* [188], *Krogh and Hertz* [124], *Solla et al.* [226], *Fahlman* [69], and others):

- *Weight decay.* Modifying the weight updating rule (9^4) as

$$w^l_{kj}(t+1) = \big(w^l_{kj}(t) + \Delta w^l_{kj}(t)\big)(1-\epsilon_{\text{dec}})\,, \tag{10^4}$$

  where $0 < \epsilon_{\text{dec}} \ll 1$ is the *weight decay parameter*, discourages large weight magnitudes and eliminates small (ie. unnecessary) weights (as OBD). This improves generalization ability. From an analogue VLSI point of view, discouraging large weights is also advantageous as the limited dynamic weight range is less likely to be exceeded.

- *Momentum.* Modifying the *weight change*, implicitly defined by (9^4), as

$$\Delta w^l_{kj}(t) = \alpha_{\text{mtm}}\Delta w^l_{kj}(t-1) + \eta\delta^l_k(t)z^l_j(t)\,, \tag{11^4}$$

  where $0 < \alpha_{\text{mtm}} < 1$ is the *momentum parameter*, averages "random" weight changes and magnifies "consistent" ones. This reduces oscillations often found during learning. The disadvantage of using momentum is the need for additional memory (especially severe for a VLSI implementation).

- *Cost function.* Using the (standard) *quadratic cost function* (cf. appendix B.3, (35^{B})) leads to the weight errors δ^l_j in (8^4). Using a typical sigmoid-like activation function, these weight errors will be close to 0 when the neuron net input s^l_j is numerically large; thus even for large neuron errors ε^l_j, no weight change will take place. This problem can be eliminated by the use of the *entropic cost function* (39^{B}) or the *Fahlmann perturbation*. The latter resulting in the following (heuristically derived) weight error:

$$\delta^l_j(t) = \big(g'(s^l_j(t)) + \gamma_{\text{F}}\big)\varepsilon^l_j(t)\,,$$

  where $\gamma_{\text{F}} > 0$ is the *derivative perturbation*. (For the entropic cost function $\delta^L_j(t) \equiv \varepsilon^L_j(t)$.) Theoretically, the weight errors should only be modified for the output layer ($l = L$). However, in an analogue VLSI implementation using the derivative perturbations on all weight errors can ensure $g'(s^l_j(t)) + \gamma_{\text{F}} > 0$ rather than $g'(s^l_j(t)) \gtrsim 0$ which would probably otherwise be calculated by the non-ideal hardware and which is destructive for the learning process.

  In *adaptive systems* it is especially important that learning can take place even though the neurons are saturated (ie. "very confident" in their decision to fire or not) as the system functionality changes over time. In this case the entropic cost function or the Fahlmann perturbation are superior to the quadratic cost function.

  The incorporation of these alternative cost functions in a VLSI implementation is straight forward.

- *Dynamic learning rate.* The learning rate is a most important parameter: If it is too large, gradient descent leads to oscillations; if it is too small, gradient descent converges *very* slowly. Adapting the learning rate to the increase in the cost function since last set of weight changes, $\Delta\mathcal{J}(t) = \mathcal{J}(t) - \mathcal{J}(t-1)$, can reduce this problem:

$$\eta(t+1) = \begin{cases} \eta(t) + a\,, & \text{for } \Delta\mathcal{J}(t), \Delta\mathcal{J}(t-1), \ldots, \Delta\mathcal{J}(t-T) < 0 \\ (1-b)\eta(t)\,, & \text{for } \Delta\mathcal{J}(t) > 0 \\ \eta(t)\,, & \text{otherwise} \end{cases} \,,$$

where $0 < a, b \ll 1$ and T are constants. Though not impossible to implement in analogue VLSI, a dynamic learning rate scheme is somewhat complex and requires memory ($O(1)$) (in addition, the cost function must be simple in order to be computable). Also, in a hardware implementation one must ensure that the learning rate does not decrease below a critical value η_{crit} where the weight changes are insignificant compared to weight discretization or weight change-offset errors, which would switch off learning.

- *Eta finder.* To avoid the complexity of a dynamic learning rate, one can choose an optimal learning rate. *Reyneri and Filippi* [196] propose (for $y_k \in [-y_{\max}, y_{\max}]$, $z_j \in [-z_{\max}, z_{\max}]$):

$$\eta_k^l = \frac{1}{2y_{\max}^2 z_{\max}^2 \beta_{\text{t}}^2 M_k^l}\,,$$

where $\eta_k^l = \eta^l$ is the layer l learning rate, $M_k^l = M^l$ is the (effective) number of inputs (fan-in) to layer l and β_{t} is the *neuron transfer function steepness.* Others prefer (*Hertz et al.* [95, 94]):

$$\eta_k^l = \eta \Big/ \sqrt{M_k^l}\,,$$

which can be combined with a dynamic learning rate rule. For a non-reconfigurable network, the learning rates can be computed in advance for the current network architecture. For a reconfigurable network, additional hardware at each reconfigurable block is needed if the learning rate computation is to be automated.

- *Batch learning.* Doing *real* gradient descent the weights are updated only after each epoch:

$$w_{kj}^l\big((1+n)T_{\text{epc}}\big) = w_{kj}^l(nT_{\text{epc}}) + \sum_{t=0}^{T_{\text{epc}}-1} \Delta w_{kj}^l(nT_{\text{epc}} + t)\,,$$

where T_{epc} is the epoch length (cf. appendix B.3). Usually on-line learning is considered the faster method†; however, using batch learning it is possible to do the weight updates in larger "chunks" which makes this scheme potentially less sensitive to offset and weight discretization (which is very important for an analogue VLSI implementation) (*Valle et al.* [251]). The disadvantage is that additional memory is needed ($O(N^2)$).

† Note, however, that on-line learning never converges (for constant learning rate); the weights will stir about the optimal solution (*White* [264], *Battiti and Tecchiolli* [19]).

4.2 Mapping the algorithm on VLSI

The MLP recall mode equation (6^4) can be written $\underline{y}^l = \underline{g}(\underline{s}^l)$, $\underline{s}^l = \underline{\underline{w}}^l \underline{z}^l$, as noted in section 2.2 (see *Lehmann and Bruun* [143], *Widrow and Lehr* [265]). Likewise, we can write the neuron error equation (7^4) as $\underline{\varepsilon}^{l-1} = (\underline{\underline{w}}^l)^{\mathrm{T}} \underline{\delta}^l$. We notice two important properties of these equations: (i) The matrix used to calculate the neuron error is the transposed of the one used to calculate the neuron net input. (ii) The signal flow is reversed. For an on-chip implementation of the back-propagation algorithm, this means that we can calculate the neuron errors and let the signals propagate from layer l to layer $l-1$ using a matrix-vector multiplier topologically identical to our recall mode synapse chip but with the positions of the inputs and the outputs exchanged. In other words: we can use an expanded version of the synapses. The neuron chip must in turn be able to calculate $\underline{\delta}^l$ as given in (8^4).

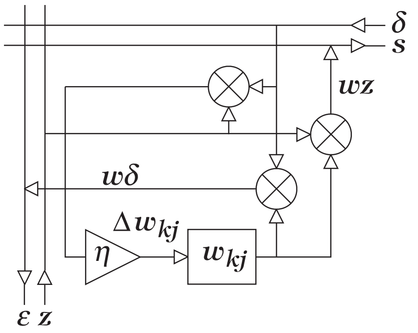


Figure 32^4: Schematic back-propagation synapse. *Two additional (current output) multipliers are basically needed compared to a recall-mode synapse.*

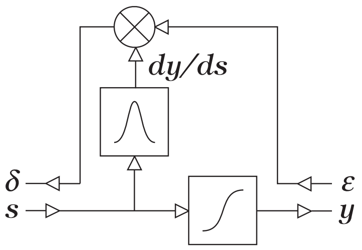


Figure 33^4: Schematic back-propagation neuron. *An additional neuron derivative computing block and multiplier are needed compared to a recall-mode neuron.*

Block diagrams of the expanded synapse and neuron can be seen in figures 32^4 and 33^4 respectively. On the synapse is also included the hardware for calculating the weight change, Δw^l_{kj}, according to (9^4). The expanded synapse has voltage inputs and current outputs just as the original synapse. Mapping the algorithm on silicon like this gives an order $O(N^2)$ improvement in speed, compared to a serial approach, as all ($O(N^2)$) weights can be updated simultaneously. Several back-propagation silicon systems with the above or similar architectures have been reported lately (*Valle et al.* [251], *Wang* [256], *Cho et al.* [46]). By enabling the neuron to route back the "y^l_k" signal on the "δ^l_k" wire, the architecture can also realize Hebbian learning† or a back-propagation/Hebbian hybrid algorithm (*Cho et al.* [46], *Shima et al.* [220]).

Instead of placing the *weight updating hardware* at the synapse sites, it can be placed at the neuron sites — which reduces the amount of weight updating

† Plain Hebbian learning uses the weight changes $\Delta w^l_{kj} = \eta y^l_k z^l_j$ (*Hertz et al.* [95]).

hardware by an order $O(N)$. In this case only weights from one neuron in layer $l-1$ to the neurons in layer l can be updated simultaneously; thus this procedure will give an order $O(N)$ improvement in speed compared to a serial approach. It should be noted that, according to (9^4), $z^l_j(t)$ is needed when calculating the $\Delta w^l_{\star j}(t)$s and must thus be routed to the sites of the weight updating hardware; if it is the $w^l_{\star j}(t+1)$s that is calculated, also the $w^l_{\star j}(t)$s are needed. The efficiency of such a scheme is highly dependent on the chosen weight storage method: Using simple capacitive storage without back-up memory, the weight change would typically be applied by a transconductance amplifier (see *Card* [40], *Wang* [256], *Linares-Barranco et al.* [151], *Woodburn et al.* [270]) which correspond to the "η-amplifier" of figure 32^4. To avoid weight degradation/destruction by charge redistribution on bus lines, this amplifier (or some kind of shielding circuit) would have to be present in any implementation, thus reducing the amount of saved hardware. As *digital memory* usually can be read non-destructively, there is no similar penalties when using capacitive storage with a digital backup memory or digital storage (the objective of the learning algorithm in these cases is to modify the digital memory). Saving weight updating hardware is particularly important when the weight modifications are in the digital domain as an A/D converter is needed at each "modification site". Placing A/D converters with more than one bit precision at each synapse would be unrealistic; the area consumption would be too large (*Shima et al.* [220]). In systems using a digital RAM backup memory, the weight access would (most probably) be serial; thus the weight updating hardware could — without further performance loss in orders of N — be placed off both synapse- and neuron chips in a single (ie. $O(1)$), separate module. In this case the cost of the weight update A/D converter is insignificant but the weight updating would be serial (an order $O(1)$ speed "improvement").

Using an $O(1)$ weight updating module has the advantage of low cost implementations of certain algorithmic improvements (the ones related to the weights rather than the neurons), as no additional synapse hardware is needed for a more complex updating scheme. Implementing momentum, for instance, basically requires a memory and an adder (or a leaky integrator for continuous time) at each synapse site when using a fully parallel ($O(N^2)$) weight updating scheme. Using an $O(1)$ weight updating module, one adder would be required. The weight change matrix could be placed in a standard digital RAM, which would cost far less than additional synapse hardware. If digital weight backup memory (or indeed digital weight memory) was used in the first place, no additional data converters would be needed. The $O(1)$ weight updating scheme implementations also have potentially better accuracy than fully parallel ones, see below.

4.2.1 Hardware efficient approach

The architecture in figure 32^4 has two major drawbacks: (i) For a given silicon area, the number of synapses is reduced compared to the number of synapses in a recall mode system, as three multipliers are used instead of one. Also, in recall mode most of the synapse hardware lies idle, which is of course undesirable. (ii) The number of wires between the synapse- and neuron chips is doubled compared to a recall mode system. Both disadvantages can severely restrict the applicability of the adaptive neural network, if its physical size is of importance. Fortunately, it is possible to overcome these disadvantages:

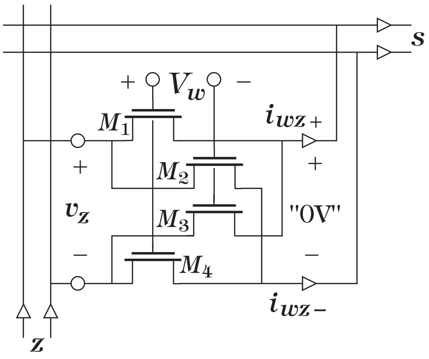


Figure 34^4: MRC operated in forward mode. *This is the standard mode operation as shown in chapter 2.*

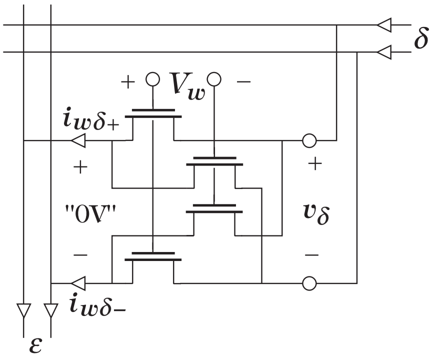


Figure 35^4: MRC operated in reverse mode. *Because of the circuit symmetry no extra synapse hardware is required for alternating forward/reverse mode operation.*

Studying the synapse multiplier used in section 2, which is repeated in figure 34^4 for convenience, we notice that it is perfectly symmetric. Thus, if we apply a differential voltage v_δ at the former output nodes and ensure virtual short-circuit between the former input nodes, the difference current $i_{w\delta}$ injected to the former input nodes would be

$$i_{w\delta} = i_{w\delta+} - i_{w\delta-} = \beta V_w v_\delta = \frac{1}{r_{\mathrm{MRC}}} v_\delta \,,$$

which is illustrated in figure 35^4. This implies that we can enable our original matrix-vector multiplier (cf. section 2.3.2) to perform multiplication with the transposed matrix, ie. to calculate $\underline{\varepsilon}^{l-1} = (\underline{\underline{w}}^l)^{\mathrm{T}}\underline{\delta}^l$, simply by *exchanging* the output *current conveyors* and the input *buffers.* See figure 37^4. When several modified matrix-vector multipliers are cascaded it is still valid that multiplication with the transposed matrix is performed when inputs and outputs are exchanged.

If the weight updating hardware is placed at the neuron sites, we can thus implement the back-propagation learning algorithm without *any* extra hardware at the synapse sites; ie. with a hardware cost of a mere order $O(N)$. Routing $z_j^l(t)$ to the neuron module is possible using the "s/δ" wires and a few $(O(N))$

switches. Thus, also *no* significant extra signal routing is needed†. It should be noted that thus *time multiplexing* the "z/ε" and "s/δ" wires obviously requires a discrete time system. (If a digital weight backup memory is used, the learning algorithm is required to run in discrete time anyway; otherwise this might restrict the applicability of the scheme.) The neurons that corresponds to this modified synapse chip would look quite like the one in figure 33^4; only, the output would have to be sampled (cf. section 4.3.2).

The implementation of such a low hardware cost analogue neural network with on-chip back-propagation can be found in the next section. The *principal operation* of the back-propagation system is illustrated in figure 36^4. During normal operation all chips are in *forward mode* and the response to an input pattern is propagated to the output after a certain delay. When a synapse weight, w^l_{kj} in a layer l is to be updated, all chips in the previous layers operate in *forward mode* and produce the z^l_is. All the chips in the layers following layer l operate in *reverse mode* and produce the ε^l_ks. The synapse chips in layer l operate in *route mode* and route z^l_j to the inputs of the neuron chips in layer l. These in turn operate in *learn mode* and calculate $w^l_{kj}(t+1)$. It will be noticed that the newly updated weights are used when back-propagating the errors in reverse mode. This does not exactly comply with the learning algorithm, though for small learning rates the difference should be indistinguishable. Actually, one would expect faster learning than when using synchronous weight update — this is actually equivalent to the *Gauss-Seidel* method of solving linear equations numerically (*Press et al.* [189]).

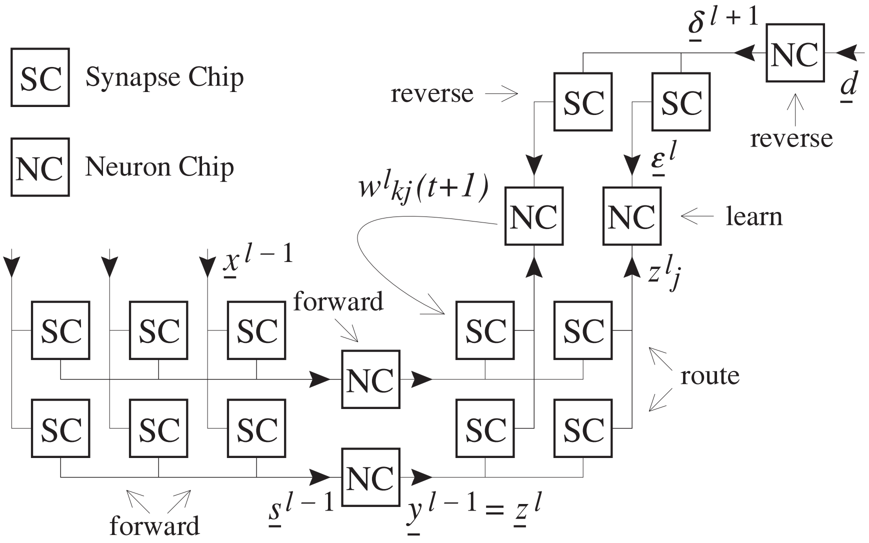


Figure 36^4: Back-propagation system. *The different operation modes of the two back-propagation chip sets are shown for a three layer network when the middle layer is updated.*

† Ie. only an order $O(1)$ control lines, etc. This is assuming a $O(1)$ weight updating module; if a $O(N)$ weight updating module is used, this should be placed at the synapse chip (one segment per row), rather than on the neuron chip, to avoid inter-chip routing of the $\Delta w^l_{\star j}$s or $w^l_{\star j}$s.

Exploiting the bidirectional properties of the MRC synapse multiplier is also possible when placing the weight updating hardware at the synapse sites for maximum speed improvement. According to the weight updating rule (9^4) neither s_k^l nor ε_j^{l-1} are needed when updating the w_{kj}^ls. Thus the z_j^ls and the δ_k^ls can be distributed simultaneously on the synapse chip while ignoring the synapse multipliers — transconductance multipliers placed at the synapse sites will then have access to the appropriate signals for calculating the $\Delta w_{\star\star}^l$s simultaneously for one layer. In other words: at the expense of using discrete time, it is possible to eliminate the extra inter-chip connections and 1/2 of the synapse learning components compared to a straight forward, fully parallel implementation of the back-propagation algorithm on top of a VLSI MLP.

As we shall see in section 4.6, it is possible also to reduce the additional neuron hardware.

An additional advantage of the bidirectional usage of the MRC is process variation insensitivity; the very same transistors are used for the synapse multiplication in recall mode as well as in back-propagation mode. Assuming the input reference voltage is close to the output voltages, M_4 in figures 34^4 and 35^4 will never conduct any current. In forward mode, the differential output current is $i_{\mathrm{D}1} - i_{\mathrm{D}2}$ and in reverse mode it is $i_{\mathrm{D}1} - i_{\mathrm{D}3}$. Thus for matching the forward and reverse currents, it is necessary only to match M_2 and M_3. (As M_4 ideally does not conduct current it could be removed; this, however, requires a very low neuron input impedance (cf. *Flower and Jabri* [71]).)

4.3 Chip design

The basic idea of the ANN chip set with on-chip back-propagation developed at our institute is the bidirectional properties of the MRC. In this section we shall describe this chip set. A description of the chip set was published in *Lehmann* [141, 142].

As noted in chapter 2, it is our intention to add learning hardware to an acting recall mode ANN; ie. the one described in that chapter. Thus, we shall use capacitive storage with a digital RAM backup memory which prevents us from using a parallel weight updating scheme. As the additional hardware cost for the serial weight updating scheme is very small, we can replace the original ANN chip set with one containing on-chip back-propagation; the learning algorithm can be disregarded by the user if so wished. As in the original ANN chip set, we shall use a hyperbolic tangent neuron activation function.

As was the case for the implementation of the original ANN, the back-propagation chip set was designed to test the functionality of the hardware efficient approach. Also, as much layout as possible from the first design was reused (or corrected/improved and reused). Design details can be found in appendix D.2.

4.3.1 The synapse chip

The computing elements of the back-propagation synapse chip in forward mode can be seen in figure 37[4]; this is identical to the computing elements of the second generation synapse chip. Writing on the synapse storage capacitors is done in the same way as on the first generation synapse chip: Pre-charged row- and column selectors and NAND gates at the synapse sites determines on which synapse the globally distributed weight voltage is to be written. The synapse schematic is a little different from the original one: no explicit storage capacitors are used; the synapse multiplier gate-channel capacitances act as memory.

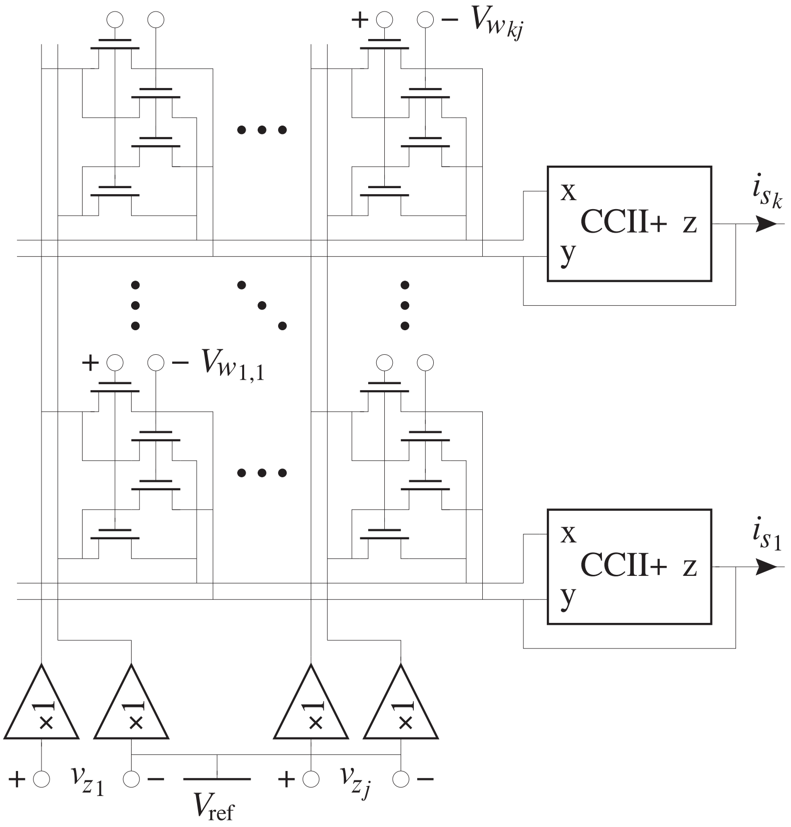


Figure 37[4]: Second generation synapse chip. *For reverse mode back-propagation operation reverse the signal flow and exchange the buffers and current conveyors.*

If the current conveyors are implemented as *supply current sensed op-amps* (*Toumazou et al.* [243]), these op-amps can be used as a voltage buffers in reverse mode. This way the components needed for each row/column on the chip are: two op-amps, two current mirrors and 11 switch transistors (plus row/column decoders, synapses, etc.). Or basically an increase of one op-amp per row and column for the back-propagation implementation. For reasonably sized switches, the voltage drop across these, when dozens of synapses source current through them, is non-negligible (say $\lesssim 200\,\text{mV}$). In order to ensure proper input voltage buffering and virtual synapse output short circuits, it is necessary to put the switches inside high gain loops or ensure zero- or matched switch transistor currents. In the latter case, the transistors need to be matched also†. The switch transistor placement for the row and column elements can be found in appendix D.2.1 (notice that the two elements are identical; only the control signals are permuted).

As no neuron errors are to be computed for the input layer, the input layer synapse chips need not be able to run in reverse mode. For this reason, the hardware efficient architecture is compatible with the *sparse input synapse chip* mentioned in section 2.3.3. Only, the sparse input synapse chip functionality must be extended to include routing of inputs to outputs.

4.3.2 The neuron chip

As stated in section 2.3.3, the output voltage of the second generation synapse chip should be close to the reference voltage V_{ref} to avoid DC common mode currents in the synapses. As the neuron output is referred to this voltage, it is necessary to separate the bipolar pair and the output range MRC with current mirrors to ensure sufficient emitter-collector voltage of the bipolar pair. Otherwise the principal neuron schematic is unaltered. The schematic of a second generation hyperbolic tangent neuron is shown in figure 38[4]. The extra current mirrors will inevitably cause an increased neuron output offset.

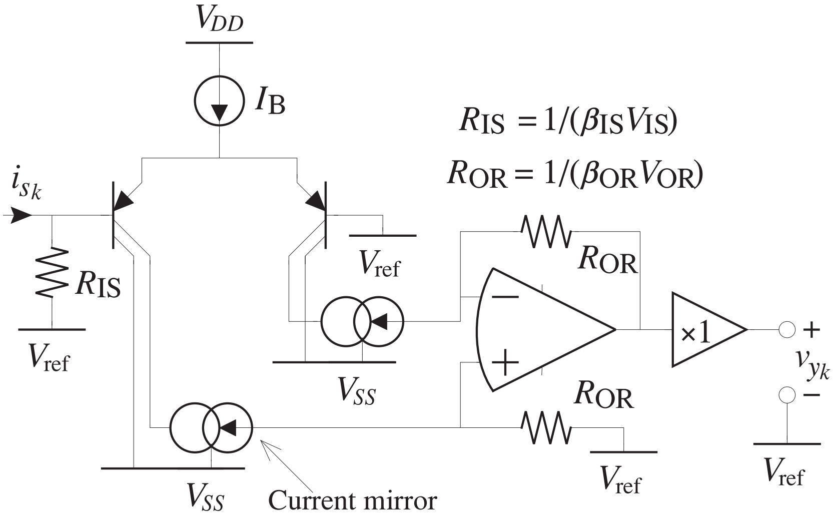


Figure 38[4]: Second generation hyperbolic tangent neuron. *Simplified schematic. The resistors are implemented as MRCs as in the first generation neuron.*

Extending the second generation neuron chip for on-chip back-propagation we must enable the neurons to compute the weight errors δ_k^l. Though serial ($O(1)$) weight updating scheme was selected, we have chosen to place some of the weight

† Unfortunately this was overlooked at design time causing the synapse weight offsets to be larger than necessary and different in forward and reverse modes.

updating hardware on the neuron chip. Strictly speaking, this is unnecessary for all but one of the neuron chips in a system. However, it is convenient to to place the "$\delta_k^l \cdot z_j^l$"-multiplier near the physical location of the "δ_k^l" and "z_j^l" signals. Also, if the back-up RAM is organized in parallel accessible banks (as the *p*-RAM in figure 70^5) this "semi parallelism" can be exploited. A block diagram of the back-propagation neuron is seen in figure 39^4. For a more detailed circuit schematic refer to appendix D.2.2. It is crucial for the functionality of the learning algorithm that the offset errors on the weight change signals are *very* small (cf. section 3.3 and the following). For this reason, hardware for offset compensating the weight change signal is indispensable. As the current system uses a serial weight updating hardware and as the new weights ultimately have to be written in the digital backup memory, there is no strong motivation to keep the weight change calculating hardware analogue†. Indeed, using *digital hardware* in most of the $O(1)$ part of the system, we can reduce the problem with weight change offset; cf. section 4.5.

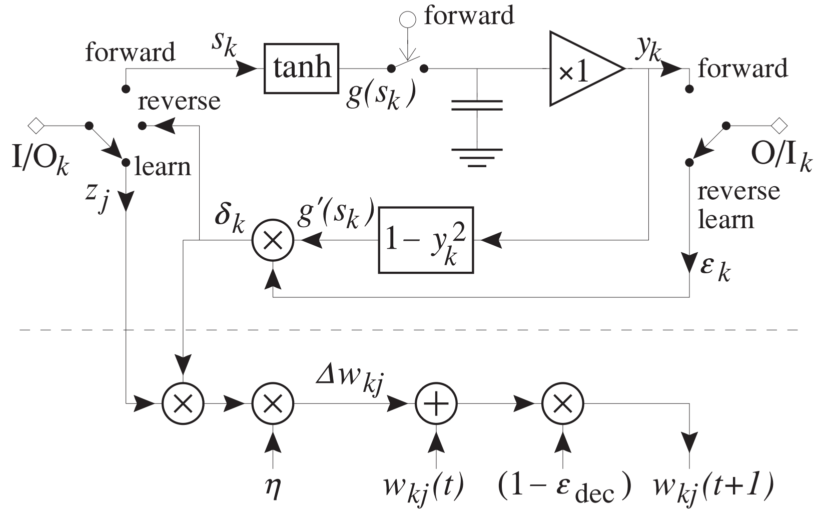


Figure 39^4: Back-propagation neuron. *Block diagram. The positions of the switches in forward, reverse and learn mode are indicated. The elements below the dashed line is the weight updating hardware which is common for all neurons.*

As we use a hyperbolic tangent neuron activation, the derivative of this is calculated as $g'_{\mathrm{calc}} = 1 - g^2$. For this operation a two-dimensional inner product *multiplier based on MRCs* is used (topologically identical to the IPM of figure 16^2,

† Recall that motivations for using analogue hardware for ANNs in the first place included the size advantage of massively parallel system and the power advantage of (especially) small systems. For $O(1)$-scaling hardware, neither of these are very important. Thus, we should not maintain a purely analogue system at *any* cost. Of course, data converters are needed when digital circuitry is included; if the expense of these is too large (eg. in small, low power systems) a fully analogue system is the solution.

p. 31, without the output transconductor). This is a *very* versatile component which is the core of most computing circuitry in this thesis. It has the power to do multiplication, addition, subtraction and division: using tree identical MRCs the output is (independently of process parameters) given by:

$$v_{\mathrm{out}} = v_{s_k} = \frac{(v_{z_1+} - v_{z_1-})(v_{w_{k1}+} - v_{w_{k1}-}) + (v_{z_2+} - v_{z_2-})(v_{w_{k2}+} - v_{w_{k2}-})}{v_{\mathrm{C}+} - v_{\mathrm{C}-}} \, ,$$

where $v_{z_1}\pm$, $v_{w_{k1}}\pm$, $v_{z_2}\pm$, $v_{w_{k2}}\pm$ and $v_{\mathrm{C}}\pm$ are inputs (cf. figure 16^2; $v_\xi \equiv v_{\xi+} - v_{\xi-}$). This is easily configured to compute the desired function (level shifters will have to be inserted at the $v_{w_{k\zeta}}\pm$ and $v_{\mathrm{C}}\pm$ inputs to ensure the transistors operate in the triode region).

The weight errors δ_k^l are computed using a one-dimensional MRC IPM. As the inputs to the synapse chip are buffered, the neuron chip does not have to have a *very* low output impedance (neither in forward nor reverse mode). Thus, the switches that redirect the signal flow in the various operating modes can be simple MOS transistors (a reasonably sized transistor (cf. appendix D.2.2) in the technology used can drive a 50 pF load to 8 bit accuracy in 100 ns — for larger capacitive loads, feedback can be used to lower the switch impedance, cf. the synapse chip row/column elements). When operating in reverse or learn mode, the neuron net input s_k^l is unavailable. Thus the neuron activation has to be sampled in forward mode in order to provide data for the calculation of the weight error.

The synapse chip provides the neuron error ε_k^l as a current. As the "$g'(s_k^l)\cdot\varepsilon_k^l$"-multiplier needs voltage inputs, we need a transimpedance at the neuron error input. This is implemented using a MRC (plus op-amp) as the neuron input scale transimpedance (see figure 15^2). The neuron errors (7^4) are computed differently for the output layer than for the preceding layers. We can accommodate to this by adding a second MRC to the neuron error transimpedance (ie. transforming this to a one-dimensional IPM) and activating this MRC at the output layer only (see figure 97^{D}). When acting as an IPM, the circuit can take the difference of an applied input *voltage* and the on-chip neuron activation. In other words: at the output layer one shall now provide a target value as a voltage rather than a neuron error as a current.

4.4 Chip measurements

A 4 neuron back-propagation neuron chip and an 8 × 4 synapse back-propagation synapse chip has been fabricated in a standard 2.4μm CMOS process. In this section, we shall present measurements on these chips (first published in *Lehmann and Bruun* [143] and *Lehmann* [142]). A table of the most important chip characteristics can be found in appendix D.2.

4.4.1 The synapse chip

The measured *forward mode synapse transfer characteristics* for a single synapse can be seen in figure 40^4; the *reverse mode characteristics* for the same synapse can be seen in figure 41^4. The non-linearity is less than 3.5 % or approximately equal to the first generation synapse non-linearity, as would be expected. The chip output offset current (in both forward and reverse mode) can be quite large; of a magnitude comparable to the maximal synapse output current. Responsible for this output offset is the CCII+ row current differencer — presumably the x-z current buffer. According to appendix C.3 we should expect a large output offset when using current differencing by rows (rather than by synapse). As the output current conveyor is designed to sink current from a large (50–100) number of synapses (and as it is this that gives the output offset), the chip can be scaled (ie. synapses added) without significant increase in output offset current. Comparing this second generation synapse chip output offset with that of the first generation chip we notice that it has been reduced by a factor 4 because of the simpler current differencing circuit. Still, with the reduced synapse maximum output current (compared to the first generation chip) it is necessary to dedicate a synapse per row for offset canceling (which would be an extra bias synapse) in order not to exhaust the neuron bias synapse. For operation in reverse mode, the offset errors are probably larger than tolerable by the back-propagation algorithm, without offset canceling being employed (cf. section 3.3, section 5.2.1).

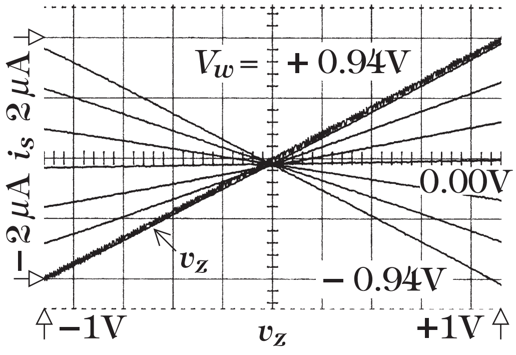


Figure 40^4: Forward mode synapse characteristics. *Measured transfer functions for single back-propagation synapse at different synapse strengths.*

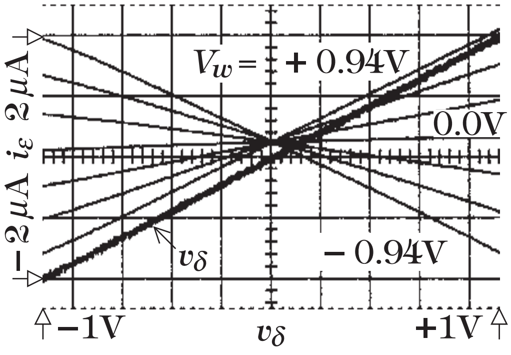


Figure 41^4: Reverse mode synapse characteristics. *Measurements on same synapse in reverse mode. The output offset errors have been almost canceled.*

As it was for the first synapse chip, the weight resolution is at least 10 bit which should be sufficient for a range of applications. The effective weight offset, however, is somewhat higher and strongly correlated with the differencers to which the synapses are connected (cf. figure 42^4 and 43^4). The offsets are caused by the inability of the synapse row/column elements to ensure virtual short circuit at the synapse output (presumably caused by mismatch of the reconfiguring switch transistors). If the systematic offset is canceled, the offset is below that of the first generation synapse chip. This non-ideality of the row/column differencers causes the synapses to have different offset in forward mode and reverse mode. Though undesirable, the magnitude of the offset is most probably tolerable by the learning scheme (the offset error corresponds to a 5 bit recall mode weight accuracy).

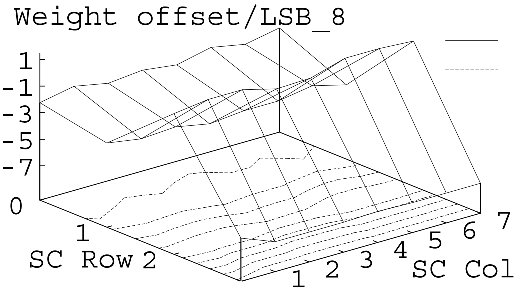


Figure 42^4: Forward mode weight offsets. *Sample chip. Notice the correlation among synapses along a row, which is emphasized by the contour plot (dashed lines).*

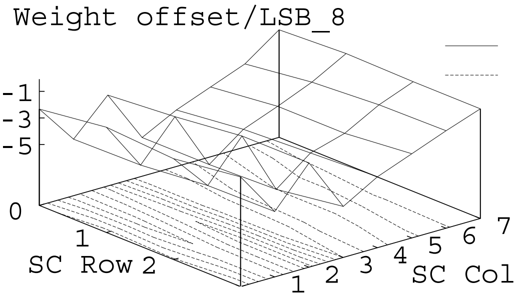


Figure 43^4: Reverse mode weight offsets. *Sample chip. Notice the correlation among synapses along a column (dashed lines: contour plot).*

4.4.2 The neuron chip

The measured *forward mode neuron transfer characteristics* can be seen in figures 44^4 and 46^4; the increased output offset compared to the first generation neuron chip is noticed. The non-linearity is less than 2% (cf. figure 47^4) and the offset errors are of little importance to the recall mode performance (as for the first generation chip set). (The increased output offset compared to the first generation chip is caused by the additional current mirrors; cf. above.) Figure 45^4 shows the *electronically computed neuron derivative* as calculated by the neuron chip (as $dy/ds = 1-y^2$, cf. figure 48^4). The asymmetry is caused by the output offset of the neuron transfer function and the input offset of the derivative computing "$1 - y^2$" block. Note, however, that the neuron input offset does not affect the accuracy of the computed derivative as would have been the case if the derivative was computed on the basis of the neuron input. The non-linearity of the derivative computing block is less than 2 % (cf. figure 49^4). The total non-linearity of the computed derivative is less than 6%. The offset errors related to the derivative computation are quite large, however; causing the computed derivative to be negative in particularly poor specimens of the neuron, which is destructive for the learning process. By including a *derivative perturbation* in the weight error calculation as mentioned above, we hope that the offsets can be tolerated by the learning scheme. This has yet to be experimentally proven, of course. (Note that several authors have employed very coarse derivative approximations and still observed learning progress (eg. *Valle et al.* [251]; see also *Shima et al.* [220], *Cho et al.* [46]).) The derivative perturbation is easily inserted by substituting the "1" by a worst case "$y^2_{\max}$" when calculating the derivative: $dy/ds \approx y^2_{\max} - y^2$. This procedure is the same as one (of several) used on the first generation chip set; the neuron output variation is larger for the second generation chip, though, which will most likely degrade the performance.

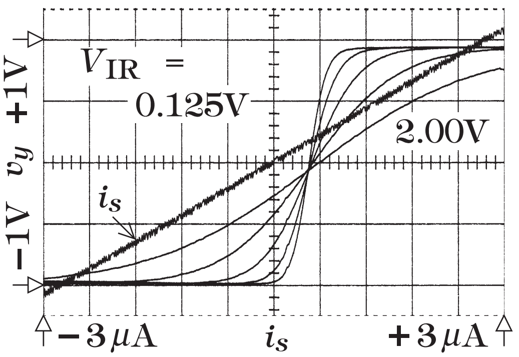


Figure 44[4]: Forward mode neuron characteristics. *Measured transfer characteristics for different input scale voltages V_{IS}.*

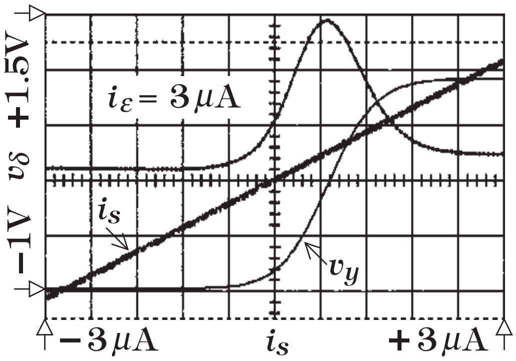


Figure 45[4]: Computed neuron derivative. *Measured derivative as computed by the chip. The asymmetry is caused by the neuron output offset.*

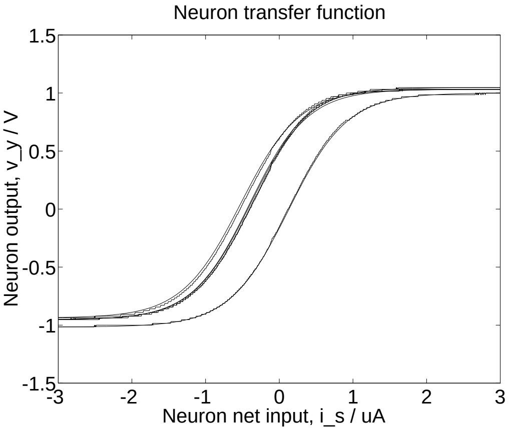


Figure 46[4]: Different neuron transfer functions. *Four measured (stair case) and fitted (smooth) tanh functions (almost indistinguishable).*

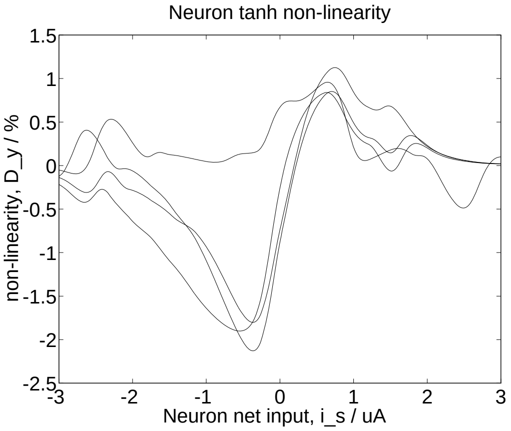


Figure 47[4]: Different neuron non-linearities. *Measured/fitted -tanh relative difference.*

The measurements on the chip set indicate that, with the inclusion of offset canceling on certain signals, the chip set should be able to function as the core of a discrete time, analogue back-propagation neural network: the chip set functions as

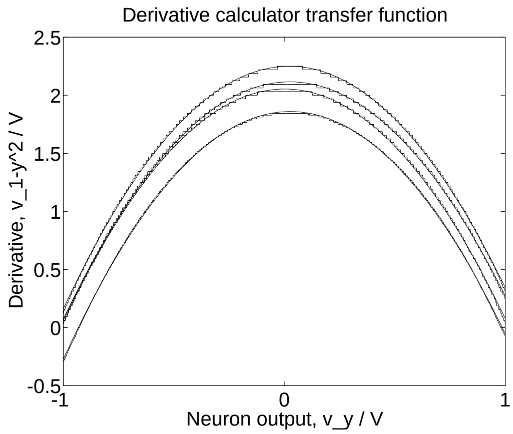


Figure 48^4: Different parabola transfer functions. *Four measured (stair case) and fitted (smooth) derivative computing parabolas.*

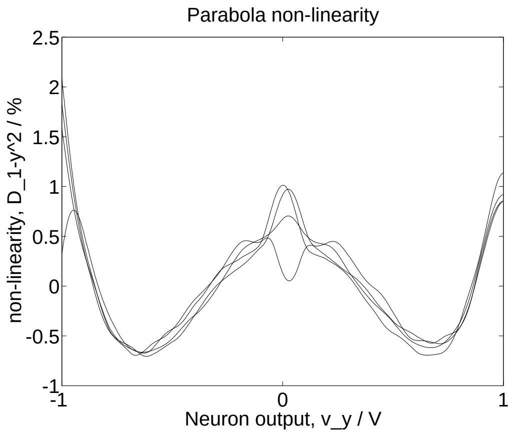


Figure 49^4: Different parabola nonlinearities. *Measured/fitted -parabola relative difference.*

predicted. Variations on transconductances, gains, offset errors (hopefully), etc. should be canceled during the learning process.

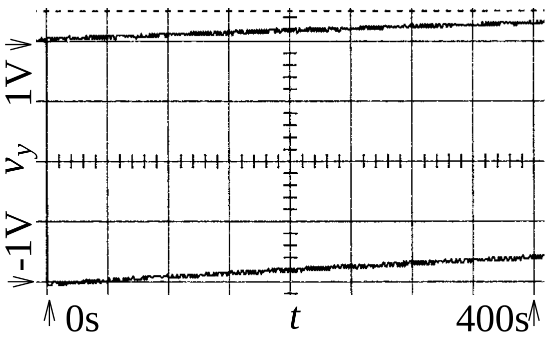


Figure 50^4: Neuron sampler droop rate. *Measured droop rate for two different starting points ($\pm 1\,\mathrm{V}$). Notice the almost linear decay caused by (mismatched) diode reverse currents. The sign and magnitude of the droop rate will vary from neuron to neuron. The synapse strength droop rate have a similar appearance.*

The synapse- and neuron chip propagation delays give a layer propagation delay of $t_{l\mathrm{pd}} < 2.5\,\mu\mathrm{s}$ which gives a *recall mode speed* of 12.8 MCPS per synapse chip in a layer (or 4 GCPS if a full-size 100×100 synapse chip is used). Note that the reconfiguring switches at the neuron output do not degrade the performance (compared to the first generation chip) at the current capacitive load. The neuron chip takes $t_{x\Delta w\mathrm{pd}} \approx 3.6\,\mu\mathrm{s}$ to calculate a new weight giving a *learning speed* of approximately 0.25 MCUPS for the back-propagation system. Given the *droop rate* of the neuron activation sampler (cf. figure 50^4), this will limit the system size to about $2 \cdot 10^6$ connections for an 8 bit accuracy of the neuron activations. (Applications using $0.3 \cdot 10^6$ connections are known; cf. chapter 2 (*Brunak et al.* [29]).) Should this be a problem, digital refresh of the sampled neuron activation can be employed (the neuron sampler actually has an extra input for this very purpose)†.

4.4.3 Improving the derivative computation

Apart from the synapse chip output offsets and weight change offsets — which can be canceled by an auto offset canceling scheme — the most concerning problem of the chip set is the calculation of the neuron derivative (this has also been the concern of other authors; *Jabri and Flower* [107], *Hollis et al.* [99], and others; some authors take advantage of the fact that the derivative need not be computed very accurately for learning to proceed and use very coarse derivative approximations *Valle et al.* [251], *Woodburn et al.* [270]). To ensure a non-negative result of the computation, we could use a derivative perturbation as mentioned above (also employed by *Shima et al.* [220]). Alternatively (or in addition to), we could clamp the output to zero whenever this was negative: Using a two dimensional CCII+ based IPM (as the second generation synapse chip IPM) to calculate "$1-y^2$", the output would be a current. Mirroring this current using a simple current mirror, we would ensure that the output was never negative (some additional hardware would be needed to ensure a reasonably input impedance, speed, etc.). It is also possible (though not necessarily without a considerable amount of additional hardware) to employ auto offset canceling techniques to cancel the offsets.

Other, more elegant, approaches to come around the offset related problems of the derivative calculation are also possible for the back-propagation chip set: It was stated in section 2.2.5 that "we would most probably not have access to the neuron net inputs" when calculating the neuron derivatives for the learning algorithm. Integrating the learning algorithm on-chip — rather than as an add-on module — we actually do have access to the neuron net input. This implies that we can choose our neuron transfer function more freely — as long as the derivative is computable. *Bogason* [26] propose to approximate the derivative by a *differential quotient*:

$$\frac{\partial i_y(v_s)}{\partial v_s} \approx \frac{i_y(v_s+\Delta V) - i_y(v_s-\Delta V)}{2\Delta V},$$

where ΔV is a "small" voltage. A block diagram for such a circuit based on a general *differential voltage in, differential current out* neuron is shown in figure 51^4. The output current is the neuron output when the switches are open (and $\Delta V = 0$) and the derivative approximation when the switches are closed.

The neuron could for instance be a MOST differential pair. In this case the magnitude of a possible negative output when calculating the derivative is determined by the matching of the two tail current sources. A 1 % mismatch of these is realistic (a small derivative perturbation could be added to ensure the derivative is calculated strictly larger than 0, of course). Using a MOST differential pair to implement the neuron transfer function is also an advantage in terms of speed, compared to the hyperbolic tangent neuron. The accuracy of the difference quotient

† In hardware efficient systems using parallel weigh update ($O(N)$ or $O(N^2)$) the neuron activation deterioration would be less problematic, of course.

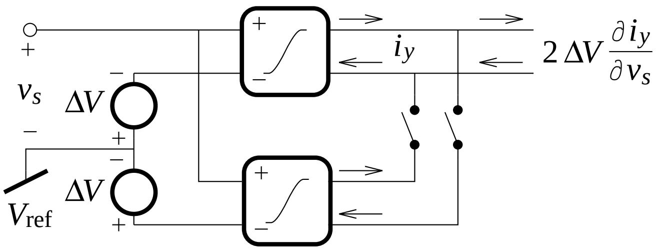


Figure 51[4]: Differential quotient derivative approximation. *Two arbitrary differential voltage in, differential current out transfer function blocks can be used for approximating the derivative (switches closed) of the transfer function (when the switches are open).*

approach can be brought within 10 % using a reasonably large ΔV. Instead of using *two* neuron transfer blocks to calculate the derivative, a single block combined with a switched capacitor (much like that we shall see in section 4.6) can be used. This reduces the transistor count.

Another approximating circuit for calculating the derivative, which inherently has positive output and which uses even fewer transistors, has also been reported lately (*Annema* [12]).

The apparent difficulties in computing the activation function derivatives needed by gradient descent have inspired some authors to do without the derivative information; for instance by substituting a constant† for the derivative or by using completely different optimization techniques. (See eg. *Battiti and Tecchiolli* [19]; also *Krogh et al.* [125]). We shall not elaborate on such solutions, as one of the objectives of this work is to approximate standard algorithms with known properties.

† Assuming a monotonous activation function. Knowledge of the *sign* of the derivative should be sufficient for convergence (using a decreasing learning rate); compare to *stochastic approximation* (*Gelb et al.* [78]).

4.5 System design

Most of the hardware for an ANN with on-chip back-propagation is included on the back-propagation chip set. For a complete system, though, some additional hardware is needed. This is:

- A digital weight backup memory.
- Most of the order $O(1)$ scaling hardware, ie. D/A and A/D converters for accessing the backup memory, and some of the weight updating hardware. Also including:
- A finite automaton to control the system (weight refresh, applying inputs, controlling the learning scheme, etc.).
- An environment in which to place the ANN.

In this section, we shall describe such a complete system.

For ease of test, we embed the ANN in a *digital PC interface* and we use the PC as the master *finite automaton* (or *finite state machine*, *FSM*). This solution does not allow us to test the speed of the system: The PC AT (ISA) bus — and the necessary A/D and D/A converters — is the the bottleneck of the system in terms of speed. Neither the recall mode nnor learning mode speed performance can be tested. However, there is no reason that the system should not run at the speed indicated by the measurements on the individual chips. The circuit level system performance, on the other hand, is most easily tested using a high degree of programmability; thus we use this "artificial" PC environment. (Also: the design time was rather important.) For a real application, the environment would be in the electrical analogue domain and the finite automaton — which is quite simple — would be a digital ASIC, possibly including the D/A and A/D converters.

The system designed is actually an *RTRL/back-propagation learning ANN hybrid*: The RTRL system (cf. chapter 5) shall be based on the second generation (back-propagation) ANN chip set. Thus, back-propagation can be included in this system almost for free, eliminating the cost of the PC interface for a separate back-propagation system. As a consequence, the back-propagation ANN architecture is determined by the RTRL ANN architecture and not designed to a specific application. For test this is acceptable. The complete system schematic can be found in enclosure III(see also appendix D.4).

Scaled back-propagation synapse chip In *back-propagation mode* the system realizes a two layer 28–12–4 perceptron. As this architecture would require 20 8×4 synapse chips it was necessary to have a *scaled back-propagation synapse chip* (16×16) fabricated. Unfortunately, on that particular MPC run the process parameters were outside the specified ranges; the n-channel process transconductance parameter, for instance, being as low as $K'_{\mathrm{N}} = 33\,\mu\mathrm{A/V}^2$ (compared to a nominal value of $K'_{\mathrm{N}} = 57\,\mu\mathrm{A/V}^2$). Presumably, the threshold voltages were numerically large which would explain the reduced dynamic range of components tested in previous MPC runs. The impacts on the chip are primarily reduced input range and large systematic offset errors (see appendix D.2.3). Using a raised reference voltage and external current offset compensation circuits it is our hope that we can make the system work in spite of the poor quality chips.

4.5.1 ASIC interconnection

Using the scaled 16×16 synapse chips the synapse chip count is reduced to three, interconnected (when the system operates in back-propagation mode) as shown in figure 52^4 (the synapse chips are drawn as having the architecture of figure 1^2 for convenience). Four input lines and four output lines in each matrix of synapses are driven by DC voltages, reserving four rows and four columns in each layer for neuron thresholds (forward mode only) and offset compensation as indicated in the figure. (Strictly speaking, reverse mode offset compensation is unnecessary for the input layer.) Prior to learning, the matrix-vector multiplier outputs are measured (in both forward and reverse mode) and the offset compensation synapse strengths are adjusted to minimize the offsets (ie. on the s_k^ls and the ε_k^ls).

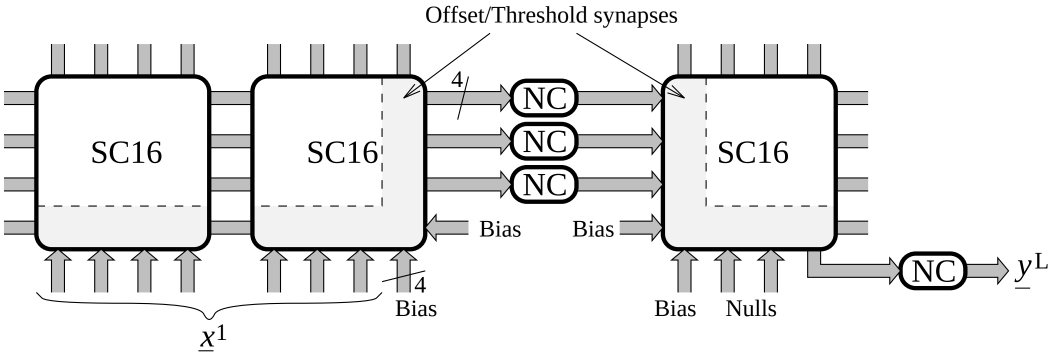


Figure 52^4: Back-propagation ANN architecture. *ANN chip interconnections when the system operates in back-propagation mode. Blocks* `SC16` *and* `NC` *are synapse- and neuron chips respectively. Input (output) lines of the* `SC16` *blocks are accessible at both top and bottom (left and right) in this figure.*

As the *synapse strength backup memory* we use a 16 bit RAM. Several authors have addressed the problem of weight discretization in ANNs (*Hollis et al.* [97], *Tarassenko* [238], *Lehmann* [139, 140], *Lansner* [133], *Brunak and Hansen* [31]; see section 3.3). A 16 bit resolution for a learning system seems to be in compliance with most of the reported simulations. Subsequent to learning, the necessary weight resolution is lower — because it is usually the relative weight magnitudes rather than the exact weight values that determines the system behavior; during learning, however, small weight changes need to be accumulated. When refreshing the synapse strengths on the synapse chip from the RAM, we use a 12 bit DAC. The actual ANN weight discretization is thus reduced to 12 bit; however, we can still accumulate weight changes as small as when using a 16 bit discretization. Actually, the ANN performance when using this scheme will be superior to that of a system where the weight discretization is reduced from 16 bit to 12 bit after learning: We train on the actual final network rather than on a intermediate network with "artificial" high weight resolution (see also *Lansner* [133]).

4.5.2 Weight updating hardware

Offset values and weight resolutions typically found in analogue VLSI systems restrict the learning rate to a range somewhat higher than what is often employed in software simulations (cf. sections 3.3 and 4.3.2, *Tarassenko and Tombs* [237], *Krogh et al.* [125], *Hansen and Salamon* [91], *Hertz et al.* [95, 94] *Weigend et al.* [261]; see also *Williams and Zipser* [268]). As we are using an $O(1)$ weight updating scheme — and as we use high precision weight backup memory — we can reduce the influence of weight change offsets and weight change discretization, and therefore *reduce the minimum learning rate*, by adding the weight change to the old weight in the digital domain. Using a 12 bit† ADC to convert the analogue weight change signal to digital form and adding this (padded with zeroes at the 4 MSBs) to the 16 bit weight, we scale the effective weight change offset by a factor $1/2^4$. This is illustrated in figure 53[4]. The actual schematic is slightly more complex as bipolar weights and weight changes needs to be handled and as overflow in the digital adder must be prevented. Also, a multiplexor is included to allow test of the on-chip analogue weight updating hardware (cf. enclosure III).

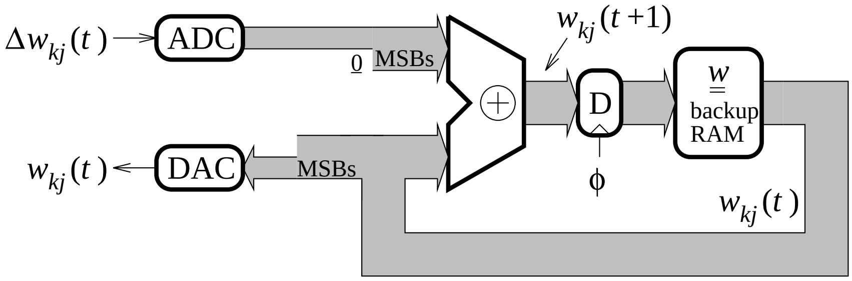


Figure 53[4]: Digital weight updating hardware principle. *The digital part uses higher precision than offered by the data converters, thereby reducing the effective learning rate.*

Studying the back-propagation neuron block diagram in figure 39[4], we see that the Δw_{kj} signal is not directly accessible; applying the $w_{kj}(t)$ and ϵ_{dec} signals on the neuron chips as zeros gives the desired weight change at the output (though inevitable with a larger offset than that of the internal Δw_{kj}). As the synapse chip outputs, the Δw_{kj} output is offset compensated prior to learning (by applying zero inputs in learn mode and adjusting the $w_{kj}(t)$ signal such that the output is also zero).

In addition to the small hardware cost, *the* advantage of using a serial weight updating scheme is this possibility to employ an advanced, accurate (ie. hardware hungry) weight updating scheme without an extreme hardware cost†. Learning algorithms that are inherently using serial weight updating (eg. weight perturbation)

† If the weight change offset is large, we should use a coarser discretization; preferably, the weight change offset should be below 1 LSB.

† All other things being equal, adding additional hardware in the analogue domain for inclusion of a more advanced updating scheme will increase weight updat-

should take advantage of this; for instance by using the weight updating scheme above. (The above updating scheme requires a weight storage with digital weight backup, of course; for large systems this is also a good choice when using a serial weight updating scheme.)

The back-propagation system is, at the time of writing, under construction. Thus, unfortunately, we can not present any system level experiments. Hopefully such will be available in the near future.

4.6 Non-linear back-propagation

As we now have seen, one of the major concerns when implementing gradient descent like learning algorithms in hardware is the computation of the neuron derivatives. Many different approaches to approximate the derivative have been proposed in the literature: difference quotient (locally or globally computed) or other approximating approaches, perturbations for reducing offset related errors, as well as implementations largely ignoring the derivative.

These implementation related difficulties recently motivated the development of a new gradient descent like algorithm, *non-linear back-propagation* (*NLBP*), in which the *derivative computation is avoided* (*Hertz et al.* [94]). In this section we shall display the algorithm and show how to incorporate it in an existing back-propagation architecture.

4.6.1 Derivation of the algorithm

The derivation of non-linear back-propagation in the framework of recurrent back-propagation can be found in *Hertz et al.* [94]. In the feed forward case, we recall the weight updating rule (9^4) which define the weight change

$$\begin{aligned}\Delta w^l_{kj}(t) &= \eta \delta^l_k(t) z^l_j(t) = \eta g'(s^l_k(t)) \varepsilon^l_k(t) z^l_j(t) \\ &= \alpha_{\mathrm{N}} \frac{\eta}{\alpha_{\mathrm{N}}} g'(s^l_k(t)) \varepsilon^l_k(t) z^l_j(t)\end{aligned} \quad ,$$

ing errors. As reducing these are a primary concern in a learning scheme implementation, this advocates for using a *simple weight updating scheme* in the analogue domain. Thus, it could be argued that, presently, the only realistic way to implement advanced updating schemes is to use an order $O(1)$ updating scheme in the digital domain.

where we call α_{N} the *NLBP domain parameter*. Now, the basic idea in non-linear back-propagation is to interpret the above equation as a first order Tailor expansion of the equation

$$\Delta w^l_{kj}(t) \approx \alpha_{\mathrm{N}} \Big[g\big(s^l_k(t) + \frac{\eta}{\alpha_{\mathrm{N}}} \varepsilon^l_k(t)\big) - g\big(s^l_k(t)\big) \Big] z^l_j(t) \,,$$

which is valid for small $\frac{\eta}{\alpha_{\mathrm{N}}} \varepsilon^l_k(t)$. Redefining the weight error definition (8^4) to

$$\delta^l_{\mathrm{N}k}(t) = \frac{\alpha_{\mathrm{N}}}{\eta} \Big[g\big(s^l_k(t) + \frac{\eta}{\alpha_{\mathrm{N}}} \varepsilon^l_k(t)\big) - g\big(s^l_k(t)\big) \Big] \,, \qquad (12^4)$$

where the $\delta^l_{\mathrm{N}k}(t)$s are the *NLBP weight errors*, the *NLBP weight change* equation has the same form as the original back-propagation equation:

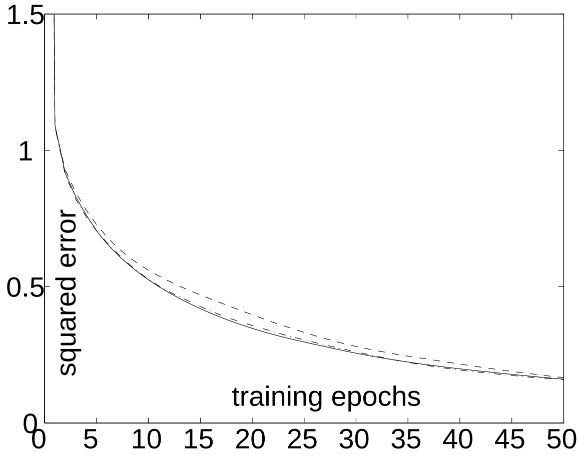
$$\Delta w^l_{kj}(t) = \eta \delta^l_{\mathrm{N}k}(t) z^l_j(t) \,.$$

When the NLBP domain parameter α_{N} is large, the Tailor approximation is good but requires high precision to compute. When α_{N} is small, the algorithm is numerically stable but is taken far from gradient descent behaviour. We think of α_{N} as being in the range $\eta \leq \alpha_{\mathrm{N}} \leq 1$. In the numerically most stable limit — which is most interesting for a VLSI implementation because of the limited precision of this technology — (12^4) takes the simpler form†:

$$\delta^l_{\mathrm{N}k}(t) = g(s^l_k(t) + \varepsilon^l_k(t)) - g(s^l_k(t)), \qquad \text{for } \alpha_{\mathrm{N}} = \eta \,. \qquad (13^4)$$

As for ordinary back-propagation, we can chose $\delta^L_{\mathrm{N}k}(t) = \varepsilon^L_k(t)$ for the output layer if we wish to use the entropic cost function.

Figure 54^4: NLBP training error. *Training error as function of learning epoch for normal back-propagation (solid line) and the non-linear version with $\alpha_{\mathrm{N}} = \eta = 0.05$ (dashed line) and $\alpha_{\mathrm{N}} = 1$ (dot-dashed line).*



† Note that the errors are propagated through the same non-linear units as the neuron activations are in recall mode, rather than through the linearized units used in ordinary back-propagation. Hence the name: *non-linear* back-propagation.

In figure 54[4] the training errors (using the *NETtalk* data set (*Sejnowski and Rosenberg* [214])) for normal back-propagation and NLBP are compared (from *Hertz et al.* [94]). The performance of NLBP is very similar to that of ordinary back-propagation in these simulations (note that the usual algorithm variations as weight decay, momentum, etc. are applicable to NLBP). In a hardware implementation it would be expected to be superior to ordinary back-propagation: Being based on addition and subtraction in addition to the neuron non-linearity, rather than being based on differentiation and multiplication, the calculation of the NLBP weight error is much simpler and thus bound to be more accurate. Also favouring a hardware implementation is the indication that NLBP seems to be superior to ordinary back-propagation for large learning rates (cf. section 3.3).

4.6.2 Hardware implementation

As the only difference between ordinary back-propagation and non-linear back-propagation is the way the weight strength errors are computed — and as these are computed locally — NLBP maps topologically on hardware in exactly the same way as ordinary back-propagation; only the neuron implementation differ. In this section we shall show the core of two neuron implementations for an ANN with on-chip NLBP learning (first reported in *Hertz et al.* [94]).

Continuous time NLBP neuron Taking the BJT differential pair of our original neuron as a starting point for implementing NLBP with a hyperbolic tangent neuron activation function, leads to the schematic in figure 55[4]. For simplicity, the differential pairs are shown implemented with npn BJTs; in an actual CMOS implementation, lateral bipolar mode p-channel MOSTs would be used as in the previous designs, of course. As the actual neuron activation function is unimportant for NLBP, MOST differential pairs could be used instead (as in *Bogason* [26]) which would favour speed. The use of LBM MOST differential pairs, probably favour accuracy (*Vittoz* [253], *Salama et al.* [205]). One will notice that the circuit requires application of the negated neuron error, $-\varepsilon_k$. Thus the synapse chip would have to compute this in reverse mode, rather than ε_k (requiring a simple modification).

It is interesting to notice that the circuit structure is identical to the one used by *Bogason* [26] (cf. figure 51[4]) to compute the neuron activation function derivative: Substituting $-v_{\varepsilon_k}$ by a small, constant voltage $-\Delta V$, the i_{δ_k} output will approximate $\Delta V \cdot \partial i_{y_k}/\partial v_{s_k}$. One can interpret NLBP as a way to exploit this implicit multiplication of the difference (13[4]) which eliminates the "$g'(s_k) \cdot \varepsilon_k$"-multiplier — and hence a source of errors. Further, v_{ε_k} is not "a small voltage" (as opposed to ΔV) which makes inherent inaccuracies less significant, relatively. Consequently, using the circuit for NLBP gives better accuracy than when using it for ordinary back-propagation.

The accuracy of the circuit (ie. on the weight error calculation) is determined by the matching of the two differential pairs and their tail currents. This can be in the order of 1 % of the output current magnitude (see eg. *O'Leary* [182]; though see

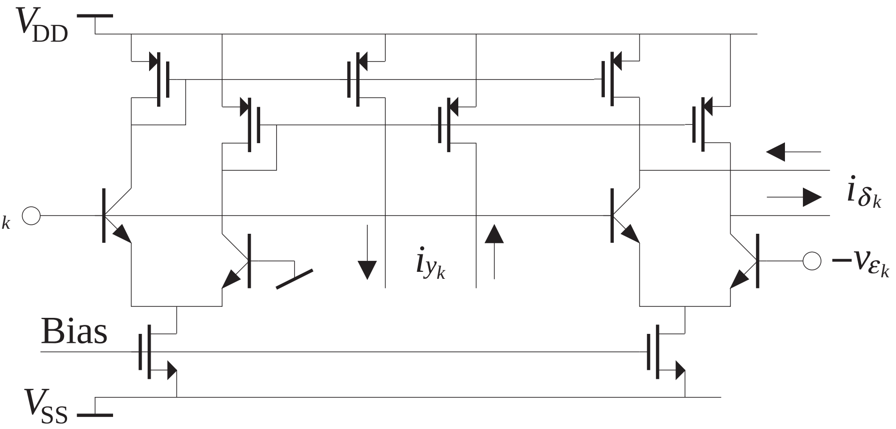


Figure 55[4]: Continuous time non-linear back-propagation neuron. *Using a MOST differential pair is also possible, though the activation function will be different. The precision is determined by the matching of the two differential pairs.*

also *Salama et al.* [205]) — which is better by far than the accuracy of our present chips and will probably enhance the performance significantly. Still, though, linear transresistances are needed at the inputs and outputs for compatibility with the synapse chip. This will degrade performance.

The circuit as presented functions in continuous time and can substitute the schematic back-propagation neuron of figure 33[4] in a system that uses fully parallel weight updating.

Discrete time NLBP neuron As the actual shape of the neuron activation function is irrelevant to non-linear back-propagation implementation† there is no need to base the implementation on differential pairs. A far better approach is to use circuits that inherently have the current inputs and voltage outputs which are needed. Also, as the same *function* is used to calculate the neuron activations and the weight errors, it would be preferable to use the same *hardware* for these calculations as this eliminates the need for matched components‡. This is possible if the system is not required to function in continuous time, though the output would have to be sampled (which introduces errors).

Shown in figure 56[4] is the simplified schematic of such a discrete time neuron which reuses the activation function block and which has current inputs/voltage outputs. During the ϕ_1 clock phase, v_{y_k} is available at the output and is sampled at the capacitor. During the ϕ_2 clock phase, v_{δ_k} is available at the output. It is the switched capacitor that computes the difference in (13[4]) and which determines the

† Indeed, we could implement a "bump" function (or *radial basis function*) which is popular among certain authors (see eg. *Hertz et al.* [95], *Sánchez-Sinencio and Lau* [206]) or any other function as long as the implementation is time invariant and possibly also reproducible.

‡ Also, if a very complex activation function is used, the extra hardware consumption might prohibit the implementation of two activation blocks per neuron.

accuracy of the circuit. Note that the output buffer needs to be linear but its offset error is canceled by the switched capacitor. Also, the neuron transfer function block (the six MOSTs) can be an arbitrary current in/voltage out circuit; static errors (as input current/output voltage offsets) in this block are irrelevant. Using design techniques to reduce charge- injection and redistribution (*Robert and Deval* [198], *Gatti et al.* [75], *Signell and Mossberg* [222], *Kerth et al.* [121]), the accuracy can be brought within 0.1 % of the output voltage range.

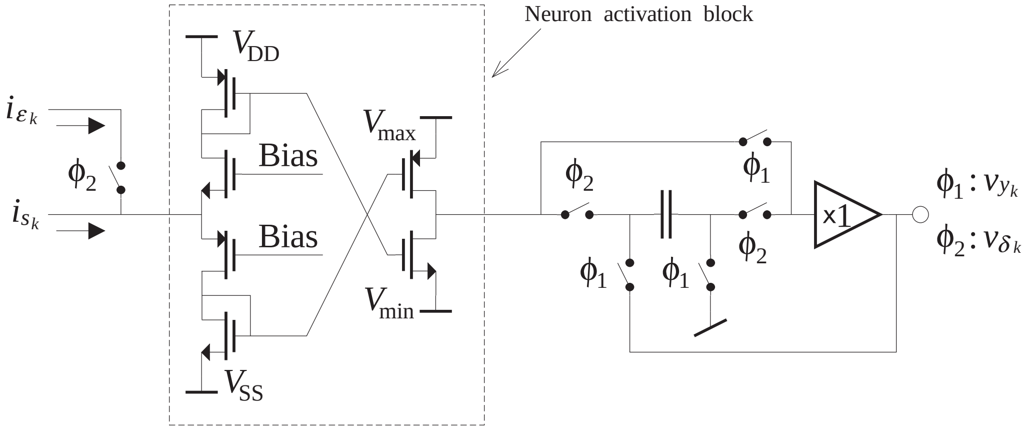


Figure 56[4]: Discrete time non-linear back-propagation neuron. *Simplified schematic. Time invariant inaccuracies will not affect the performance of this circuit. The precision is determined by the switched capacitor.*

This discrete time NLBP neuron can directly replace the computing elements of our original back-propagation neuron in figure 39[4]. Thus, assuming that the voltage buffer would be needed in a recall mode version of the neuron, the *NLBP hardware overhead* is potentially extremely small: consisting of only a switched capacitor at the neuron sites in addition to the very modest hardware increase of our original "hardware efficient back-propagation synapse chip" and the order $O(1)$ weight updating hardware and finite automaton algorithm controller.

Operating the output transistors of the transfer function block in the triode mode the circuit output voltage will exhibit a reasonably smooth transition from V_{max} to V_{min} when the input current is increased, giving an S-shaped transfer function. The circuit, however, has a *very* poor power supply rejection ratio (*PSRR*). A more realistic circuit is shown in figure 57[4]. The insertion of current mirrors in the signal paths gives a much better PSRR and the possibility of a lower input impedance. To avoid drawing current from the output range references (which would compromise their rigidity) simple amplifiers buffer these (NLBP does not require the neuron output range to be *very* well defined, thus the large input offset ($\gtrsim 2V_{\text{T}}$) of the amplifiers need not match very well). The neuron steepness is controlled by the input stage bias current, I_{B}. Transfer function simulation for different bias currents can be seen in figure 58[4].

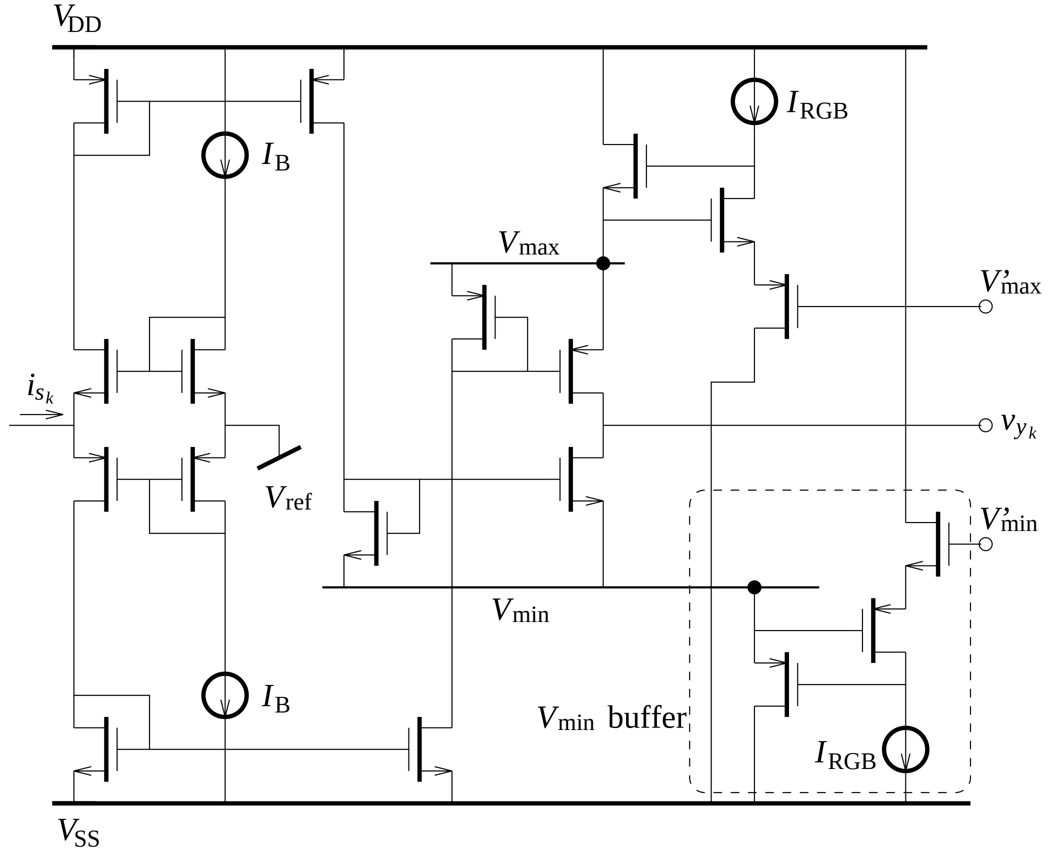


Figure 57[4]: Neuron activation block schematic. *Improved PSRR. The outputs from a standard source follower current input stage drives the transfer function shaping output transistors via current mirrors. The output level reference buffers can be thought of as degenerated regulated cascodes.*

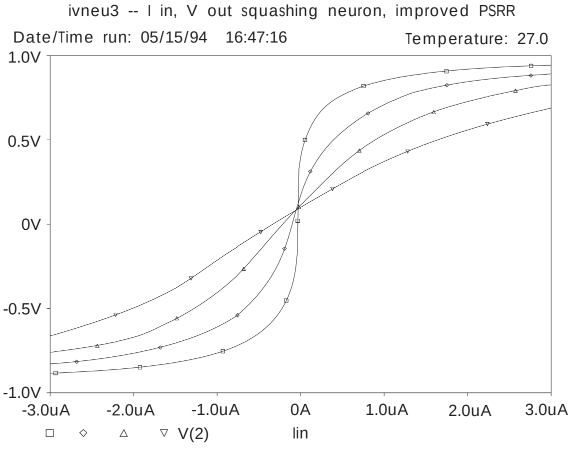


Figure 58[4]: Simulatedi neuron transfer function. *Transfer function at* ±2.5 V *power supply for several bias currents.*

4.7 Further work

Obviously, system level experiments need to be carried out in order to evaluate the applicability of our proposed chip set (especially with respect to derivative computation and weight change offsets). This work is presently being carried out at our institute. Also, the implementation of the proposed discrete time NLBP neuron chip (which could be pin compatible with our present back-propagation neuron chip and thus substitute this in the back-propagation system) is an evident future design task.

As for the recall mode neural network design of chapter 2, several design issues need consideration prior to "a volume production". The considerations on *process parameter dependency canceling* and *temperature compensation* mentioned in section 2.7 also apply for the back-propagation chip set. In addition, the implementation of the *high accuracy calculations* (or rather low offset ones) needed by the learning scheme requires further investigation. In this section we shall discuss a few approaches to reduce the influence of offset on critical signals and to the inclusion of algorithmic variations.

4.7.1 Chopper stabilizing

One of the more critical signals in an ANN learning scheme with respect to offset is the weight change. A straightforward solution, mentioned in section 4.5, to this problem is to measure the offset during an auto zeroing phase and subsequently subtract this offset. This solution has two major drawbacks: it requires (i) memory and (ii) an offset free comparator (though see chapter 5.3.2). Especially for systems with a fully parallel weight updating scheme, these drawbacks are quite severe. Another way to eliminate the weight change offset is to apply a *chopper stabilizing* technique known from operational amplifier offset cancellation (*Hsieh et al.* [104], *Allen and Holberg* [7], *Coln* [52]): The polarity of the inputs and outputs of a differential op-amp is synchronously and periodically (at f_{chp}) reversed which moves the offset error (and low frequency noise) to the odd harmonics of the chopping frequency f_{chp}. In a similar way, we can periodically permute the inputs/output polarities of the "$\delta_k \cdot z_j$"-multiplier used to compute the weight changes. This is illustrated in figure 59^4, for the case where the weight updating multipliers are placed at the synapse sites for parallel weight updating†.

Assume that the weight updating multiplier (which should be a differential inputs, differential output multiplier; eg. the Gilbert multiplier of figure 7^2) computes:

$$\Delta w_{kj\text{mul}} = \eta(\delta_k + \delta_{kj\text{ofs}})(z_j + z_{kj\text{ofs}}) + \eta_{kj\text{ofs}} \, .$$

Inserting four switch transistors at each input and at the output, which reverses the polarity when the corresponding control signal ϕ_ξ is high, and doing four successive weight updates using

$$\begin{bmatrix} \phi_\eta \\ \phi_z \\ \phi_\delta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},$$

† Pulse width modulation (by gating the ϕ_η and $\overline{\phi}_\eta$ signals) of the contacts connected to the storage capacitor could be used to adjust the learning rate; this is not shown in the figure. A multiplier bias current could also be used to control the learning rate.

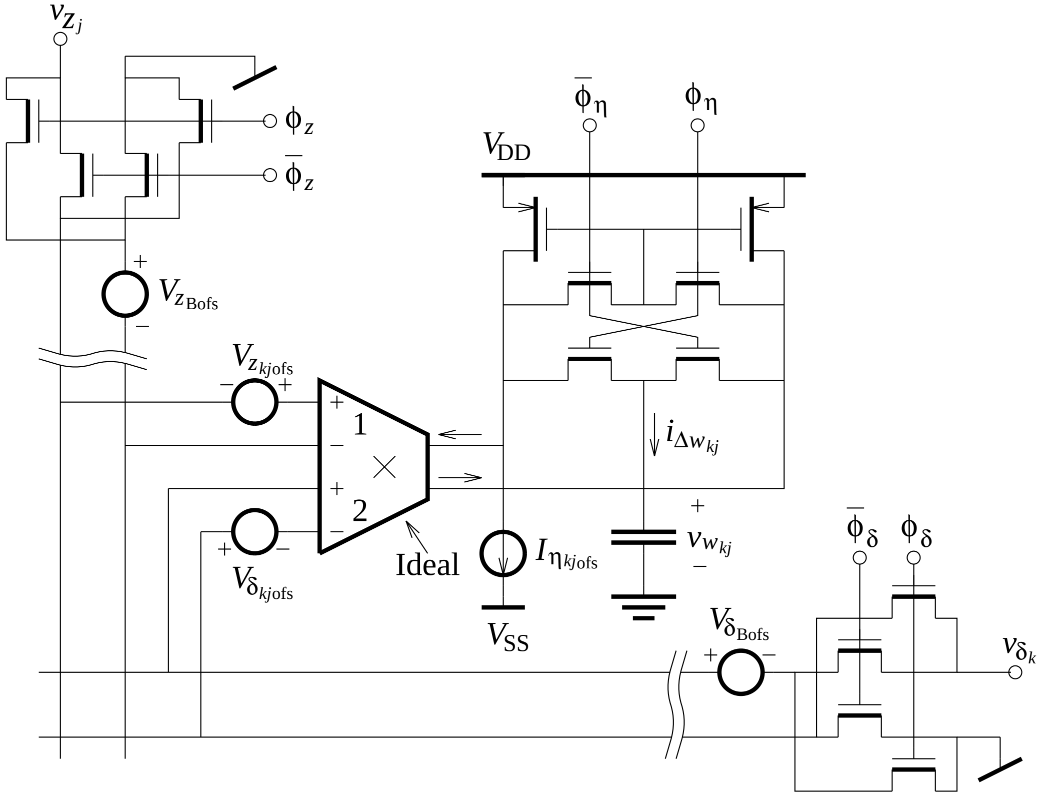


Figure 59[4]: Chopper stabilized weight updating. *Principal schematic. For parallel weight updating as indicated, only four extra minimum switches are needed at the synapse sites.*

gives the following resulting weight change:

$$\begin{aligned}\Delta w_{kj} &= \sum_{\phi_\eta,\phi_z,\phi_\delta} \eta\big((1-2\phi_\delta)\delta_k + \delta_{kj\mathrm{ofs}}\big)\big((1-2\phi_z)z_j + z_{kj\mathrm{ofs}}\big) + (1-2\phi_\eta)\eta_{kj\mathrm{ofs}} \\ &= 4\eta\delta_k z_j\end{aligned} .$$

We call this *multidimensional chopper stabilization.* Note that the output switches are placed in such a way that, to a first order approximation, offset errors related to the (switched) differencing current mirror are also canceled. Placing the switch transistors as indicated on the figure, we see that only four minimum switch transistors (in addition to the weight updating multiplier) is necessary at each synapse site. The other switch transistors can be common to a row or column of synapses. This has the additional advantage that possible offsets in input buffers (as indicated on the figure) also will be canceled. If we were to add chopper stabilizing of the weight change signal in our present back-propagation system, only one stabilizer (or actually one per neuron chip) would be required, of course.

For exact offset cancellation, the data frequencies should be lower than half the smallest of the chopper frequencies. In a discrete time system, however, one would probably apply a new data set for each of the four ϕ_ξ triples; offset cancellation would still be expected.

The chopper stabilizing technique can be used at other signals as well. Most probably at the back-propagated error signals. Chopper stabilizing this signal on our present back-propagation synapse chip is, unfortunately, somewhat difficult because of the current conveyor based differencing technique. Basing the synapse differencer on a current mode operational amplifier instead, the stabilizing might be possible using a few more switch transistors in the row/column elements.

If signals, used concurrently with the stabilized ones above, are to be chopper stabilized (say, one would stabilize the δ_ks computed at the neuron chip), more chopper frequencies need to be introduced. One must ensure that all permutations of chopper phases ϕ_ξ, giving a constant resulting sign of the weight change signal, are present in a complete cycle.

4.7.2 Including algorithmic improvements

As mentioned in section 4.5, an advantage of using a serial weight updating scheme is that advanced procedures can inexpensively be employed. Thus, improvements of the system by including relevant algorithm variations displayed in section 4.1 (or other variations) are important to consider.

Weight change threshold In addition to offset compensation and chopper stabilizing, the problem of offsets on the weight changes can be solved by introducing a *weight change threshold* Δw_{min} as proposed by *Montalvo et al.* [168]. The influence of a weight change offset is most severe when the ideal weight change is close to zero; in this case it is the offset rather than the desired weight change that determines the actual applied weight change. The effect is that the weight space state will always drift away from a cost function minimum (a solution to the problem at hand) where the ideal weight changes are zero. Taking the consequences of this is to introduce a weight change threshold below which weight changes are ignored; ie. substituting Δw^l_{kj} in (9^4) by:

$$\Delta w^l_{kj}(t) = \begin{cases} 0\,, & \text{for } |\eta\delta^l_k(t)z^l_j(t)| < \Delta w_{\mathrm{min}} \\ \eta\delta^l_k(t)z^l_j(t)\,, & \text{otherwise} \end{cases} .$$

This is quite easily incorporated in our digital weight updating scheme without introducing errors.

Momentum One of the more popular improvements of back-propagation is *momentum*. In the analogue domain (assuming parallel weight updating), momentum is included by adding a leaky integrator at the output of each "$\delta_k \cdot z_j$"-multiplier. However, using a momentum parameter α_{mtm} (cf. (11^4); typically in the order of 0.9) offset errors associated with the multiplier (and the integrator) is increased by a factor $1/(1-\alpha_{\mathrm{mtm}})$. When the weight changes are small this is a severe problem which might very well prohibit the inclusion of momentum in pure analogue systems. In the digital weight updating scheme used in our back-propagation system (figure 53^4), however, the effective weight change offset was reduced compared to the analogue approach. Thus, in this system we could hope that the increased effective offset error would be acceptable; especially if a weight change threshold is included.

Choosing the momentum parameter $\alpha_{\mathrm{mtm}} = \frac{1}{2}$ enables a very easy implementation of momentum in our back-propagation system. The only hardware needed is a simple digital adder and a digital RAM ($B_{\Delta w} = 12\,\mathrm{bit}$ wide and the same number of words as the weight backup RAM). Using a $B_{\Delta w}$ bit discretization of the weight change signal, the *memory* in the resulting weight change signal is only $B_{\Delta w}$ training samples. This is smaller than desired for most applications. Using $\alpha_{mtm} = 0.9$ gives a 6.6 times longer memory; however, multiplying by 0.9 using digital hardware is *very* inconvenient. Another way to lengthen the weight change memory is to apply the α_{mtm} factor only every A_{mtm}th sample (we call this *degenerated momentum*); ie.:

$$\Delta w_{kj}^{l}(t) = \begin{cases} \alpha_{\mathrm{mtm}} \Delta w_{kj}^{l}(t-1) + \eta \delta_k^l(t) z_j^l(t)\,, & \text{for } t = 0, A_{\mathrm{mtm}}, 2A_{\mathrm{mtm}}, \ldots \\ \Delta w_{kj}^{l}(t-1) + \eta \delta_k^l(t) z_j^l(t)\,, & \text{otherwise} \end{cases} .$$

This increases the memory by a factor A_{mtm}. The resulting weight change corresponding to the training data applied at time t is approximately $A_{\mathrm{mtm}}/(1-\alpha_{\mathrm{mtm}}) \cdot \eta \delta_k^l(t) z_j^l(t)$. The hardware implementation of such a scheme is more complicated than the simple choice of $\alpha_{\mathrm{mtm}} = \frac{1}{2}$. It requires the addition of overflow control on the adder and the insertion of "arithmetic shift left" hardware for the selective multiplication.

Weight decay Implementing *weight decay* in a system with simple capacitive storage (and a parallel weight updating scheme) is, in principle, just a matter of making the storage capacitor leaky; ie. placing a resistor from the capacitor to a "zero weight" reference voltage. The weight decay, however, must be small compared to typical weight changes in order not to prohibit learning. *Krogh and Hertz* [124], for example, use a very small weight decay parameter- learning rate ratio of $\epsilon_{\mathrm{dec}}/\eta = 1 \cdot 10^{-4}$, which would probably be insignificant compared to typical weight change offsets. If one could accept a weight decay that was large compared to the weight change offsets, the influence of these could probably be reduced (in addition to the other advantages of weight decay). Whether this is possible, experiments would have to show.

In our back-propagation system with the order $O(1)$ digital weight updating hardware, the situation is different. Though using $\epsilon_{\mathrm{dec}}/\eta = 1 \cdot 10^{-4}$ at a learning rate $\eta = 0.1$ gives $\epsilon_{\mathrm{dec}} = 2^{-17}$ (which is negligible when using a weight discretization of 16 bit) the weight backup memory precision could easily be enhanced (to 24 bit, say) to accommodate such small weight changes. (If the weight change offset is larger than one LSB of the weight change ADC, a weight change threshold would be needed.) The inclusion of weight decay in our system requires only a digital adder.

4.7.3 Other improvements

As was the case for the recall mode ANN chip set, several improvements of the back-propagation chip set are possible. A list of the more obvious can be found in appendix D.2.4.

4.8 Summary

In this chapter we designed a variation of our cascadable ANN chip set, including on-chip error back-propagation learning. The basic learning algorithm was displayed and the applicability of common algorithmic variations for the implementation in analogue VLSI was discussed. It was shown that a fully parallel implementation would give an order $O(N^2)$ improvement in speed compared to a serial solution. It was also shown that, exploiting the symmetry of the MRC, it was possible to implement back-propagation with no extra hardware at the synapse sites and no extra inter-chip connections (at the cost of an order $O(N)$ in speed); this is our solution. Using digital RAM weight back-up, weight access restrictions reduce the learning speed to $O(1)$ compared to a serial solution.

The design of our back-propagation chip set was displayed. An improved, CCII+ based, current differencer is used on the synapse chip and the neuron chip is approximately twice as complex as the first generation recall-mode one. MRCs are used excessively for the extra computing circuitry on the neuron chip. Measurements on the chip set were displayed indicating a 0.25 MCUPS learning speed. The measurements suggest that a range of offset errors (especially on the weight change signal) would have to be canceled. Also, the neuron derivative computation seems to be problematic when haunted by certain offset errors. Several improvements to this circuit were proposed.

A complete back-propagation system design based on our chip set was displayed. As the weights ultimately have to be placed in a digital RAM, most of the weight change hardware not present on the neuron chips are implemented using discrete digital hardware. We elaborated on the virtues of such a solution: eg. reduced minimum effective learning rate and weight change offset, and easy, reliable implementation of weight decay and momentum. The system is presently under construction, thus no experimental results were presented.

The novel non-linear back-propagation learning algorithm was displayed. Not needing the neuron derivative, the neuron circuitry for this algorithm is superior to that of the original algorithm (the exact neuron transfer function is irrelevant; we can focus the design effort on the electrical characteristics of the neuron). Several possible back-propagation neurons were proposed; one for continuous time non-linear back-propagation, and one compatible with our discrete time system. Using the latter solution virtually no extra hardware is needed for the learning algorithm, compared to the recall-mode system.

A chopper stabilization technique for reducing offset errors were proposed. A sample implementation for reducing offset errors in weight change signals computed by local (synapse multiplier) circuitry was given.

Finally, the inclusion of weight change thresholds, momentum, and weight decay in our back-propagation system was outlined.

Chapter 5

Implementation of RTRL hardware

The implementation of add-on real-time recurrent learning hardware for our ANN chip set is the objective of this chapter. The learning algorithm is first briefly described after which it is shown how it can be mapped on hardware compatible with our ANN architecture using a realistic amount of hardware. Results from simulations modeling analogue VLSI non-idealities in the architecture are displayed. The design of an experimental VLSI chip implementing most of the learning hardware is next presented; including an offset canceling scheme for the critical weight change signal. Chip measurements are also presented. The design of a complete RTRL system is done and it is shown how we can apply algorithmic variations using the system. We derive a *non-linear* version of the *RTRL* algorithm and we argue that this algorithm has the same virtues as non-linear back-propagation. Reflections on future work are then given: A continuous time RTRL system is considered. A summary concludes the chapter.

5.1 The RTRL algorithm

The *real-time recurrent learning* (*RTRL*) algorithm is a supervised, gradient descent algorithm (cf. appendix B.3) for general recurrent artificial neural network architectures. In this section we shall describe the basic algorithm and display modifications typically applied to it.

5.1.1 Basics

The RTRL algorithm for an artificial neural network with a discrete time feedback (discrete time *recurrent artificial neural network*, *RANN*, cf. appendix B.1, figure 74^{B}, figure 60^{5}) can be described as follows (*Williams and Zipser* [267, 268]): Given an *input vector* $\underline{x}(t)$ at time t, we can write the neuron k activation (30^{B}) as

$$y_k(t) = g_k(s_k(t)) = g_k\left(\sum_{j\in I\cup U} w_{kj}(t)z_j(t)\right),$$

where

$$z_j(t) = \begin{cases} x_j(t)\,, & \text{for } j \in I \\ y_j(t-1)\,, & \text{for } j \in U \end{cases} . \tag{14^5}$$

I is the set of *input indices* and U is the set of *neuron indices.* The neuron biases are implicitly given as the connection strengths from a constant input $z_0 = 1$. Note that, because of the discrete time feedback, the z_js dependency on the time is slightly different from the definition in chapter 4. Doing usual gradient descent on-line learning, we use a weight updating rule of the form

$$w_{ij}(t+1) = w_{ij}(t) - \eta\frac{\partial\mathcal{J}(t)}{\partial w_{ij}(t)} \overset{\text{often}}{=} w_{ij}(t) - \eta\sum_{k\in U}\frac{\partial\mathcal{J}_k(t)}{\partial y_k(t)}\frac{\partial y_k(t)}{\partial w_{ij}(t)}\,, \tag{15^5}$$

where $\mathcal{J}(t)$ is the instantaneous cost function (cf. appendix B.3.1). Now, the idea of RTRL is that the neuron activation derivatives can be shown to equal to

$$\frac{\partial y_k(t)}{\partial w_{ij}(t)} = p_{ij}^k(t)\,,$$

where the *neuron derivative variables* p_{ij}^k are computed as†

$$\begin{aligned} p_{ij}^k(0) &= 0\,, \\ p_{ij}^k(t) &= g_k'(s_k(t))\sigma_{ij}^k(t)\,, && (16^5) \\ \sigma_{ij}^k(t) &= \sum_{l\in U} w_{kl}(t)p_{ij}^l(t-1) + \delta_{ik}z_j(t)\,. && (17^5) \end{aligned}$$

We have assumed that teaching starts at time $t = 1$ and we have introduced the *neuron net input derivative variables* σ_{ij}^k. Using the *quadratic cost function*, the resulting weight change is equal to

$$\Delta w_{ij}(t) = \eta\sum_{k\in U}\varepsilon_k(t)p_{ij}^k(t)\,. \tag{18^5}$$

† As before, δ_{ik} denotes Kronecker's delta.

The *neuron error* $\varepsilon_k(t)$ is defined as

$$\varepsilon_k(t) = \begin{cases} d_k(t) - y_k(t)\,, & \text{for } k \in T(t) \\ 0\,, & \text{otherwise} \end{cases}\,, \tag{19^5}$$

where $d_k(t)$ is the neuron k target value at time t and $T(t)$ is the set of neurons for which targets exist at time t (the *target indices*). Using the *entropic cost function* and *hyperbolic tangent activation functions*, it can be shown that the weight change is equal to

$$\Delta w_{ij}(t) = \eta \sum_{k \in U} \varepsilon_k(t) \sigma_{ij}^k(t)\,. \tag{20^5}$$

A new $\{\underline{x}, \underline{d}\}$ pair is applied at time $t+1$. We call the completion of the computations for a given t a *learning cycle* (or *time step*).

5.1.2 Variations

As the back-propagation algorithm, RTRL can be varied in numerous ways. Most of these variations does not alter the topology of the algorithm and can easily be applied to a VLSI implementation. In addition to the use of different cost functions as shown above, variations include (*Williams and Zipser* [267, 268], *Smith and Zipser* [225], *Catfolis* [43], *Hertz et al.* [95], *Brunak and Hansen* [31] and others):

- *Teacher forcing.* When the network is to be taught such that the dynamic behaviour is altered in a qualitative manner, *teacher forcing* can be employed (for instance when the network is taught to oscillate). Here the target values (when they exist) rather than the network outputs are fed back:

  $$z_j^{\mathrm{TF}}(t) = \begin{cases} x_j(t)\,, & \text{for } j \in I \\ d_j(t-1)\,, & \text{for } j \in T(t-1) \\ y_j(t-1)\,, & \text{for } j \in U \setminus T(t-1) \end{cases}\,.$$

  For correct derivative calculation in this case, the sum in (17^5) should be taken for $l \in U \setminus T(t-1)$. The incorporation of teacher forcing in a VLSI RTRL implementation is straight forward.

- *Relaxation* and *Pipelining.* The RTRL network architecture (cf. figure 74^{B}) implies that a delay of a number of time steps will be present from an input is applied to the corresponding output will be seen. If, for instance, the network is to implement an XOR function it must organize itself as a two-layer (at the least) perceptron; ie. it can compute $y(t) = x_1(t-1) \mathbin{\forall\!\!\!\frown} x_2(t-1)$. When applying target values to the network, one is implicitly constraining the network architecture by choosing the input-output delay (in terms of time steps). The delay must be just long enough to enable a sufficiently complex network organization without unnecessary delay insertions (which degrades learning). At least two procedures are possible when introducing the input-output delay: One is to *relax* the network for a number of time steps when an input has been applied; ie. set $\underline{x}(nT_{\mathrm{PD}}+1) = \underline{x}(nT_{\mathrm{PD}}+2) = \cdots \underline{x}((n+1)T_{\mathrm{PD}})$ and apply the corresponding target only at time $(n+1)T_{\mathrm{PD}}$ ($n \in \mathbb{N}_0$ and $T_{\mathrm{PD}} \in \mathbb{N}$ is the desired *propagation delay*). The other is to exploit the *pipelined* nature of the network architecture; ie. use a new input $\underline{x}(t)$ at each time step and apply the corresponding target at time $t + T_{\mathrm{PD}} - 1$. Clearly the latest method gives the highest throughput. It might be harder to train, though, as delays for synchronizing may have to be inserted by the learning algorithm. The choice of relaxation/pipelining only effects the algorithm control mechanism of a VLSI implementation. This is also true for:

- *Learning by subsequence.* Sometimes it is interesting to apply different, independent sequences to the network rather than regarding the inputs as a continuous stream of data. To avoid false correlations between different sequences, the neuron derivative variables p_{ij}^k (and the neuron states) are reset between each sequence (eg. at $t = nT_{\mathrm{seq}}$ if all sequences have the length T_{seq}). Related to this is the application of a priori temporal knowledge: If the output is known to be dependent of the latest T_{mem} input vectors at most, the p_{ij}^ks can be reset at $t = nT_{\mathrm{mem}}$ to enforce this limited memory. (Note that in both cases a tapped delay line feed forward ANN can solve the task. However, in some cases (especially when $T_{\mathrm{seq}}/T_{\mathrm{mem}}$ is large) a recurrent net needs *much* fewer processing elements.)
- *Random initial state.* As an alternative to setting $p_{ij}^k(0) = 0$ (or $p_{ij}^k(nT_{\mathrm{seq}}) = 0$) the initial neuron derivative variables can be set to small random numbers:

  $$p_{ij}^k(0) = n_{ij}^k(0)\,,$$

  where $n_{ij}^k(t)$ are uncorrelated noise sources (eg. white, Gaussian). This have the tendency to speed up the learning of small sequences.
- *Momentum, weight decay, etc.* The standard learning algorithm variations mentioned in section 4.1 are also applicable to RTRL. Also, the notes on applicability for hardware realizations do apply.

It should be noted that a *continuous time* formulation of the algorithm is also possible (cf. section 5.7.1).

5.2 Mapping the algorithm on VLSI

The topological mapping of the RTRL algorithm on analogue VLSI was first published in *Lehmann* [139, 140]. As mentioned in chapter 2 it is our aim to implement learning algorithms for the analogue ANN topology described in chapter 2. By adding a sample-and-hold circuit as feedback in a one layer system based on our recall mode chip set we arrive at the discrete time recurrent network topology for which RTRL was developed. This network architecture is shown in figure 60[5].

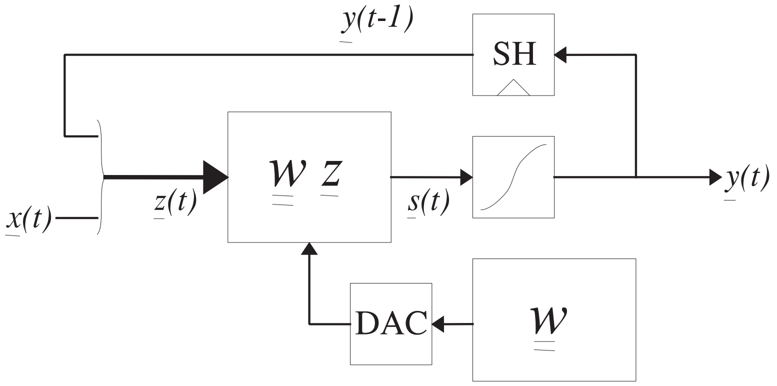


Figure 60^5: The discrete time RANN system. *Block diagram. The system is composed of a collection of synapse- and neuron chips forming a one-layer ANN and a sample-and-hold circuit as feedback.*

The calculations for a training example needed by the RTRL algorithm can be performed fully in parallel. It is, however, unrealistic to construct a system that grows as $O(N^4)$ when N is large. (Say we can have 100^2 multipliers on a chip. For $N = 100$ — a network that could fit on a few synapse- and neuron chips — we need more than 10000 multiplier chips! Also, a fully parallel weight update is incompatible with our ANN system as we have serial access to the weight backup RAM.) Now, studying the basic equations above we notice that

$$\underline{p}^{\star}_{ij}(t) = \big(\underline{1} - \underline{y}(t) \otimes \underline{y}(t)\big) \otimes \underline{\sigma}^{\star}_{ij}(t)\,, \qquad (21^5)$$

$$\underline{\sigma}^{\star}_{ij}(t) = \underline{\underline{w}}_{\square}(t)\underline{p}^{\star}_{ij}(t-1) + \underline{\delta}_{i\star} z_j(t)\,, \qquad (22^5)$$

where $\underline{\underline{w}}_{\square}$ is the synapse weight matrix $\underline{\underline{w}}$ without the columns corresponding to the inputs, $\underline{1} \equiv [1, 1, \ldots, 1]^{\mathrm{T}}$ and "$\otimes$" denotes vector multiplication by coordinates. (In (21^5) we have assumed $g_k(.) \equiv \tanh(.)$.) The weight change equations can be written as

$$\Delta w_{ij}(t) = \eta \underline{\varepsilon}(t) \cdot \underline{p}^{\star}_{ij}(t)\,,$$

$$\Delta w_{ij}(t) = \eta \underline{\varepsilon}(t) \cdot \underline{\sigma}^{\star}_{ij}(t)\,,$$

for the *quadratic* and *entropic* cost function respectively. Implementing the above equations in parallel divides the $O(N^4)$ operations between the space domain and the time domain as $O(N^2)$ to $O(N^2)$. This division has several advantages:

- The area of the computing parts of the learning hardware does not grow faster with N than does the ANN†.
- Most of the calculations (the order determining $\underline{\underline{w}}_{\square}\underline{p}^{\star}_{ij}$) can be performed by a matrix-vector multiplier (almost) identical to the synapse matrix-vector multiplier of the ANN.
- Most of the additional hardware can be implemented in cascadable "*signal slices*".
- The system is *cascadable*; ie. can (in principle) be expanded to arbitrary size.

† Still, the area of the derivative variable memory grows as $O(N^3)$, of course.

- No signal paths need more lines than $O(N)$.
- Weight updating is serial which gives the advantages mentioned in chapter 4.

The *disadvantage* of the $O(N^2)/O(N^2)$ space/time division is of course that the system will be an order $O(N^2)$ slower than a fully parallel implementation.

A block diagram of the proposed RTRL system can be seen in figure 61^5. The two matrix-vector multipliers, the synapse weight memory and the derivative variables memory can easily be identified. The adders "$\oplus$" and multipliers "$\otimes$" are working by coordinates, the select block "`SEL`" chooses the outputs that have target values, the multiplexor-demultiplexor pair computes $\underline{\delta}_{i\star}z_j$, and "$\boxdot$" is a vector inner product multiplier. The dash-dotted signal path is to be used for the entropic cost function. Controlled by a digital finite automaton, the system operation is as follows: At the end of a learning cycle, $\underline{y}(t-1)$ is sampled. Then $\underline{x}(t)$ and $\underline{d}(t)$ are applied and $\underline{y}(t)$, $\underline{g}'(\underline{s}(t))$, etc. are computed asynchronously. Now, for each $\{i,j\}$ $\underline{p}^\star_{ij}(t-1)$ is read from the RAM after which $\underline{p}^\star_{ij}(t)$ and $w_{ij}(t+1)$ are computed and stored in the respective RAMs.

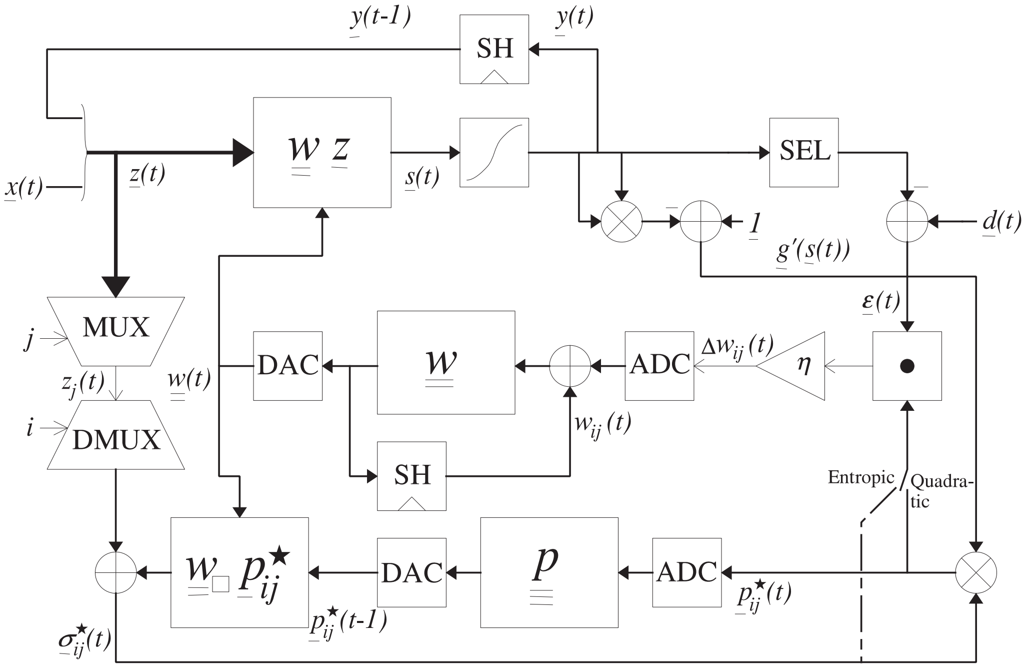


Figure 61^5: The discrete time RTRL system. *Block diagram. The lines carry analogue signal vectors of width* 1, N *or* $N+M$ *as indicated by their thickness or digital signals.*

Comments on the topology As indicated on the figure, the derivative variables are meant to be placed in a digital RAM. Digital RAM has been selected as it is physically small, cheap and reliable. As the storage requirement grows as $O(N^3)$, this is the *size limiting* factor of the system. (For small systems it might be feasible to use analogue storage.) Large RAMs have serial word access. Thus to achieve the required $O(N)$ parallel signal for the $\underline{p}^\star_{ij}$s it will be necessary to multiplex the RAM access. This is the *speed limiting* factor of the system (more precisely it will most probably be the ADCs connected to the multiplexors that limit the speed; this would be a reason to use analogue memory in small systems).

To follow the learning algorithm strictly, we should update the analogue weight storage on the matrix-vector multipliers only after a full learning cycle has been completed. For very large systems the on-chip weight storage might be degraded too much during the time of a learning cycle, though. To come around this problem, two RAM banks would have to be used; one for refresh and one for the new weights. If the weight changes are small, however, there is no reason to believe that a periodical weight refresh (using the weights in the partially updated weight backup memory) should prohibit learning (cf. section 4.2.1).

All elements in figure 61[5] except the multiplexor, the digital weight updating hardware, the RAMs, and the matrix-vector multipliers operate by coordinates on vectors of width N. These elements can thus be placed on a cascadable "*width N data path module*". The inner product multiplier would be distributed among each signal slice of the module and must have current output to ensure cascadability. Thus a system with the architecture in the figure will be comprised of the following components:

- A number of synapse chips doing the $\underline{\underline{w}}\,\underline{z}$ multiplication.
- A number of neuron chips applying the tanh nonlinearities. The two set of chips act as the one layer core neural network.
- An $N + M$ way multiplexor; the *width* $N + M$ *data path module.*
- Two digital RAMs with corresponding A/D and D/A converters. The data converters can be off-chip or on-chip components.
- A *width 1 data path module* for the weight updating hardware. On this module the digital finite automaton for controlling the learning scheme would also be placed.
- A number of *width N data path modules* (with a total of N signal slices) performing the rest of the calculations.

If we are to add the learning hardware to an existing ANN, we must control the ANN neuron output range as we compute the neuron derivative as $1 - y^2$ (cf. previous chapters). Also, we are implicitly requiring the two matrix-multipliers (and the width N signal slices) to match (inter chip matching!). It turns out, though, that this matching is not very important (cf. below).

It should be noted that the *sparse input synapse chip* mentioned in section 2.3.3 can be used to process the network inputs ($\underline{x}$) — if the z_j-multiplexor is also made capable of handling binary coded inputs.

5.2.1 System simulations

In *Lehmann* [139, 140] *simulations* were done on the influence of various *non-idealities* in the system. Non-linearities, offset errors, and quantizations on selected signals were investigated. The restrictions found are in compliance with what other authors have found for other learning algorithms (eg. back-propagation and weight perturbation) (*Montalvo et al.* [168], *Hollis et al.* [97], see also *Tarassenko et al.* [238], *Withagen* [269], and others). See also section 3.3. The qualitative conclusions of these simulations were that:

- The *neuron output* must not be larger than 1. Because of the way the neuron (tanh) derivative is computed this is a very strict requirement. (If $y_k > 1$ the computed derivative can have the wrong sign.) Non-linearities of the transfer function can be tolerated to some extent. (In the simulations non-linearities in the range $-20\,\% \lesssim D_y \lesssim 0\,\%$ were acceptable† (for the quadratic cost function; for the entropic the acceptable range is generally smaller). However, the exact tolerable range will depend on the *problem*, on the *network size*, on the *number of training cycles*, on the *learning rate* and on other *non-idealities*. Thus the qualitative conclusion is more interesting than the actual figures.)
- The *synapse strength discretization* must be sufficiently fine. (In the simulations at least 8 bit.)
- The *weight change offset* must be very small. (In the simulations at most $0.8 \cdot 10^{-3}$.)
- The *neuron error offsets* must be small. This applies to the non-targeted neurons. For the output neurons a neuron error offset will merely displace the network output by the offset. (In the simulations the error offset had to be at most $1 \cdot 10^{-2}$.)

In addition to these constraints, the simulations showed that

- The *neuron derivative variable discretization* can be rather coarse. (In the simulations 3 bit, though even three levels $(-1, 0, 1)$ seems to be adequate in some situations.)
- The *non-linearities* in general can be large. (Up to at least 40 % in the simulations.)
- The *offset errors* in general can be large. (For instance up to about 10 % for the derivative variables.)

These very soft requirements on the computing accuracy indicates that, for instance, inter chip matching is not very important. Offset cancellation on various signals, on the other hand, will be necessary.

† The lower bound was determined by the target values; obviously these must be within the neuron range.

5.3 Chip design

The recall mode ANN for which we shall design the RTRL hardware is based on the back-propagation chip set of chapter 4. We will disregard the back-propagation operation modes for this implementation. An important design strategy for the RTRL system, as for the other systems in the thesis, is to reuse hardware. The "$\underline{\underline{w}}_{\Box}\, \underline{p}^{\star}_{ij}$"-multiplier is of course implemented using synapse chips, but also at transistor level for the *width N data path module* shall we reuse hardware. Most of the layout on this module actually originates from the back-propagation chip set. As the other chips, the RTRL hardware chips was designed to validate the system topology; ie. as little hardware as possible was put on silicon.

In this section we shall present the design of the *width N data path module*. Design details can be found in appendix D.3. The rest of the RTRL system will be implemented using discrete components and will be presented in section 5.5.

5.3.1 The width N data path module signal slice

Compiling the learning components on figure 61[5] that operate in data paths of with N, we arrive at the block diagram in figure 62[5] for a single signal slice. The *width N data path RTRL module* (the *RTRL chip*) basically consists of a number of these signal slices. For a more detailed circuit schematic refer to appendix D.3. As on the other chips designed so far, the MRC is used extensively for the analogue computing components.

To avoid oscillations (or "race-around") when the neuron activations are sampled, the y_k-sample-and-hold circuit must be edge trigged. The sampler is implemented using two successive simple track-and-hold samplers, the first using the inverted clock signal of the second.

The "$1 - y_k^2$"-block calculating the neuron derivative is identical to the one on the back-propagation neuron chip. It is implemented using a two dimensional MRC based IPM. Likewise, the "$\delta_k - y_k$"-subtractor is implemented using a one dimensional IPM as on the back-propagation neuron chip (for the output layer).

A two way multiplexor controlled by $T_k(t)$ is used to select whether neuron k has a target at time t. If it has not, a zero is given as the neuron error signal. This implementation ensures a negligible error offset (originating from this circuit) on the neurons without target values which, according to the simulations, is essential for learning. The input offset on the succeeding inner product multiplier will, of course, cause an offset; this is unavoidable. One must ensure that the IPM (dimension number k) input with inherently lowest offset is used for the error signal. Controlled by a multiplexor, the other (dimension number k) input to the IPM can be either p_{ij}^k or σ_{ij}^k depending on which cost function one chooses.

The inner product multiplier that calculates the weight change is distributed among the signal slices in exactly the same way that the IPMs on the back-propagation synapse chip are distributed among the matrix columns: One MRC multiplier is placed in each signal slice and the differential current outputs from

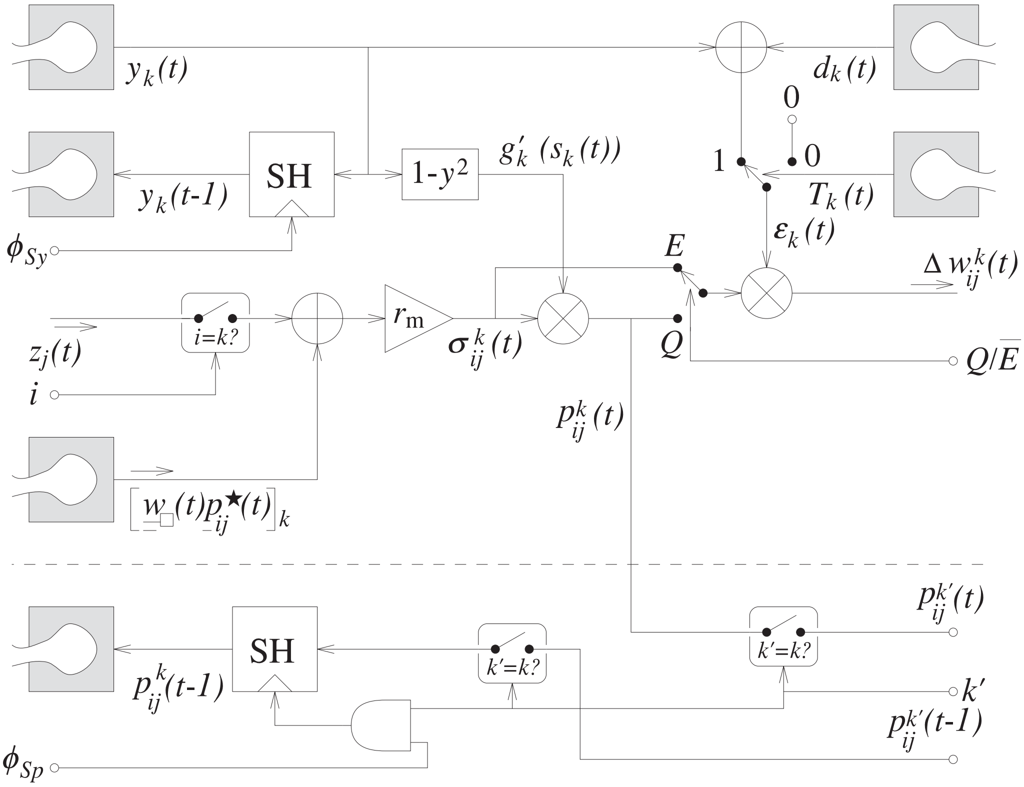


Figure 62^5: Order N signal slice. *Block diagram of signal slice in width N data path module. The "$\xi = k$?"-switches are parts of distributed (de)multiplexors. Chip I/O nodes are indicated by the "bonding pads". The elements below the dashed line are p^k_{ij} access circuitry.*

these are computed by a single CCII+ giving the chip as a whole the desired Δw current output.

The z_j demultiplexor is also distributed among the signal slices. The output of the preceding (off-chip) multiplexor will be a voltage (which is also necessary for distributing the z_j signal to several *width N data path* chips). On-chip, z_j is transformed to a current which can easily be demultiplexed (as in figure 20^2). The output k from the demultiplexor is added to the (current) output k of the "$\underline{\underline{w}}_{\square}\,\underline{p}^{\star}_{ij}$"-matrix vector multiplier; the resulting current being transferred to a voltage. The transresistance used for this is equivalent to the input transresistance of the back-propagation neuron chip (it is an MRC plus an op-amp); the input voltage must likewise be close to V_{ref} to avoid DC common mode currents in the "$\underline{\underline{w}}_{\square}\,\underline{p}^{\star}_{ij}$"-matrix vector multiplier. The transresistance value is chosen such that the maximum effective synapse strengths are in the range $0.3 \lesssim |w_{ij}|_{\max} \leq 10$. The output level is large to ensure good accuracy at small inputs (nominally 1 V for $z_j = 1$) — at the expense of a quickly saturated output.

Finally, the "$g'_k \cdot \sigma^k_{ij}$"-multiplier is a one dimensional IPM. This gives a total of six operational amplifiers for the computing hardware in a signal slice.

Also included in the *width N data path* chip is a demultiplexor, sample-and-hold circuits and a multiplexor for accessing the derivative variable RAM. These components are drawn below the dashed line in the figure. The multiplexor and demultiplexor are distributed among the signal slices as the z_j demultiplexor. The sample-and-hold circuits (one per signal slice) are simple track-and-hold implementations. Placing the derivative variable access circuitry on the *width N data path* chips as thus indicated means that one D/A converter, one A/D converter and one derivative variable RAM bank must be present per *width N data path* chip. This is convenient, but the system speed will be low if many signal slices are implemented on a single chip (cf. *Lehmann* [139, 140]). The chip is supplied with two input channels for the sampling of the $p_{ij}^k(t-1)$s: one channel meant for reading the RAM and another meant for initialization of the $p_{ij}^k(t-1)$s (to be connected to zero or a noise generator dependent on which variation of the algorithm one uses).

5.3.2 Auto offset compensation

Repeatedly noted in this text is the necessity of a low weight change offset; ie. the weight change output has to be offset compensated. The weight change offset compensation circuitry can be placed on the *width 1 data path module*. However, as the offset standard deviation will be proportional to the square root of the number of cascaded *width N data path modules* (assuming uncorrelated individual module offsets), the dynamic range of such an offset compensation circuit must be very large (in principle infinite for an arbitrary ANN size). For this reason the compensation circuitry should be placed on the *width N data path module* instead (or perhaps in addition to)†.

The principal schematic of the *width N data path module* offset compensation circuit is shown in figure 63[5]. During an auto-offset phase the Δw-signal is disconnected from the output pin and lead into a current comparator instead while the inputs to the Δw computing IPM are brought in a state resulting in an ideal zero Δw current. Now, the *offset compensation current* controlled by the *successive approximation register* (*SAR*) is adjusted to give a zero comparator input current.

The D/A converter To avoid the problems with charge injection and weight drift of analogue storage, we have chosen to store the measured offset in a digital way as indicated on the figure. A D/A converter is therefore needed to subtract the measured offset. The summed weight change currents of the *width N data path* chips must be converted to a digital signal by the *width 1 data path module*. Now the effective weight change offset must clearly be less than, say, $\frac{1}{2}\,\mathrm{LSB}_{\Delta w}$ of this converter if no measures against offset errors are taken on the *width 1 data path*

† Placing the compensation circuitry on the *width N data path module* is in compliance with the observations of analogue computing accuracy of appendix C.3: Quadrupling the number of connected Δw current outputs doubles the resulting expected offset error. To bring the offset error below a certain value, the offset canceling circuit precision must then be doubled which requires quadrupling the area.

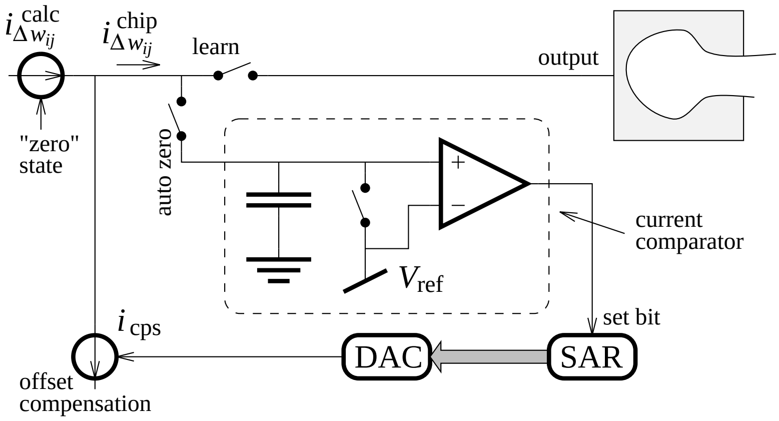


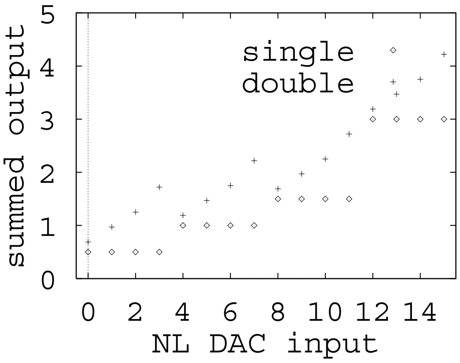
Figure 63[5]: Current auto zeroing principle. *During auto-zeroing an ideally zero output current $i^{\text{chip}}_{\Delta w_{ij}}$ is directed to the current comparator instead of the chip output pin. The offset is stored in the SAR and subtracted (i_{cps}) from the output.*

module. Assuming the input range of the converter corresponds to the maximum synapse weight, the allowable offset (relative to a unit output current) is

$$\Delta w_{\text{ofs}} < \frac{|w_{ij}|_{\max}}{\eta} \cdot 1\,\text{LSB}_{\Delta w}\,.$$

If the offset is known to be less than two MRC maximal output currents†, and if we are using a $B_{\Delta_w} = 12\,\text{bit}$ data converter, a maximum synapse weight of $|w_{ij}|_{\max} = 10$ and a learning rate of $\eta = 1$, we need a 10 bit current offset canceling D/A converter. While not impossible to implement in a standard CMOS process (*Salama et al.* [205]), a 10 bit DAC is quite area consuming. By sacrificing monotonicity, fortunately, we can instead cascade two (or more) lower precision DACs:

Figure 64[5]: Double resolution D/A conversion. *Deliberately introducing nonmonotonousness of weighted sums of D/A outputs increases the over all resolution.*



† Apart from a slightly scaled MRC unit cell, the weight change computing IPM is identical to a row on the back-propagation synapse chip; thus the offset measurements on these chips give a good estimate of what to expect on the RTRL chip. Hence this value.

Assume we have two (B_α and B_β bit) D/A converters, α and β, with ideal output ranges $[0,(1-1\,\mathrm{LSB}_\alpha)\alpha_{\max}]$ and $[0,(1-1\,\mathrm{LSB}_\beta)\beta_{\max}]$ and maximum (relative differential and -integral) nonlinearities D_α and D_β. Taking the weighted sum, γ, of the outputs, α and β, in the following way

$$\gamma = \alpha + \frac{1\,\mathrm{LSB}_\alpha + D_\alpha}{\beta_{\max}/\alpha_{\max}}\beta\,, \tag{23^5}$$

γ will have a resolution of

$$\gamma_{\mathrm{res}} \leq (1\,\mathrm{LSB}_\alpha + D_\alpha)(1\,\mathrm{LSB}_\beta + D_\beta)\alpha_{\max}\,.$$

(For $D_\alpha \leq 1\,\mathrm{LSB}_\alpha$, $D_\beta \leq 1\,\mathrm{LSB}_\beta$ this corresponds to $B_\alpha + B_\beta - 2$ bits.) This is illustrated in figure 64^5 where a sample γ is shown for $B_\alpha = B_\beta = 4$ and $D_\alpha = D_\beta = \frac{1}{2}\,\mathrm{LSB}_\alpha$. In practice, the γ resolution will be somewhat coarser, because one can not make the accurate scaling needed in (23^5); one must make sure that β is scaled with a factor at least as large as the ideal one.

One is, with relative ease, convinced that the successive approximation register on figure 63^5 works with this resulting non-linear D/A converter. Such analogue offset canceling using a deliberately non-linear DAC has also been proposed by *Kaulberg and Bogason* [118].

On the *width N data path* chip we have used two eight bit standard cell DACs for the current offset canceling D/A converter. The voltage outputs of these control two (mutually scaled) MRCs connected to the input of the weight change computing IPM. A resolution of about 14 bit could be expected.

The current comparator For high precision offset canceling, it is of paramount importance that the current comparator in figure 63^5 is offset free. This is accomplished by using a very high input impedance voltage comparator as indicated in the figure (see also *Domínguez-Castro et al.* [61]). During offset canceling the chip output current, $i^{\mathrm{chip}}_{\Delta w_{ij}}$ (the offset encumbered "zero" output current, $i^{\mathrm{calc}}_{\Delta w_{ij}}$ minus the current value of the offset canceling current i_{cps}), will be integrated on the input capacitance, C_{cmp}, of the comparator (or actually the node capacitance; especially for small comparator input capacitances the current source output capacitances will be significant — this is actually the case for our RTRL chip). Given enough time, the voltage on the capacitor will eventually exceed any comparator offset error and saturate the comparator output at the desired value. Given a comparator gain A_{cmp}, input offset V_{ofs}, and minimum high output V_{high} and a maximum comparison time T_{cmp} the input current resolution is

$$i_{\mathrm{cmp\,res}} = C_{\mathrm{cmp}}\frac{V_{\mathrm{ofs}} + V_{\mathrm{high}}/A_{\mathrm{cmp}}}{T_{\mathrm{cmp}}} \approx C_{\mathrm{cmp}}\frac{V_{\mathrm{ofs}}}{T_{\mathrm{cmp}}}\,.$$

For $C_{\mathrm{cmp}} = 500\,\mathrm{fF}$, $V_{\mathrm{ofs}} = 10\,\mathrm{mV}$ and $T_{\mathrm{cmp}} = 1\,\mu\mathrm{s}$ we have $i_{\mathrm{cmp\,res}} \approx 5\,\mathrm{nA}$. To reduce the comparison time for a given input resolution, source followers can be placed at the comparator inputs (though the offset error will increase, the input capacitance can be reduced much).

During normal operation, the weight change output voltage will be close to the reference voltage V_{ref} (cf. above). As the weight change offset would be expected to be dependent on the output voltage we use a comparator reference level of V_{ref} (which ensures that the weight change output voltage is close to V_{ref} during offset canceling). Prior to each comparison the comparator input voltage is reset to V_{ref} to ensure fast comparison. By adding a capacitor and one or more switches, the comparator offset error can be reduced by standard auto offset canceling techniques (*Geiger et al.* [77]), thus improving the comparison time.

The SAR The *successive approximation register* can be implemented as bit slices in a straight forward manner with CMOS multiplexors and dynamic delay elements. A bit slice of such a SAR is shown in figure 65^5. For the sake of clarity, the switches are shown as single transistor switches though CMOS switches are actually used. The circuit needs a two phase non-overlapping clock and a *start conversion* signal, `sc`, which must be high during the ϕ_1 phase prior to conversion (for the generation of the clock signals and start signal, refer to appendix D.3). The current comparator must be reset at the ϕ_2 clock phase and active during the ϕ_1 phase. The SAR consists basically of a shift register that shifts a 1 from MSB to LSB during conversion and a memory register. The bit-under-test will apply a 1 to the DAC while the current comparator output (*set bit*, `sb`) is stored in the register. The other bit slices apply the stored bit to the DAC.

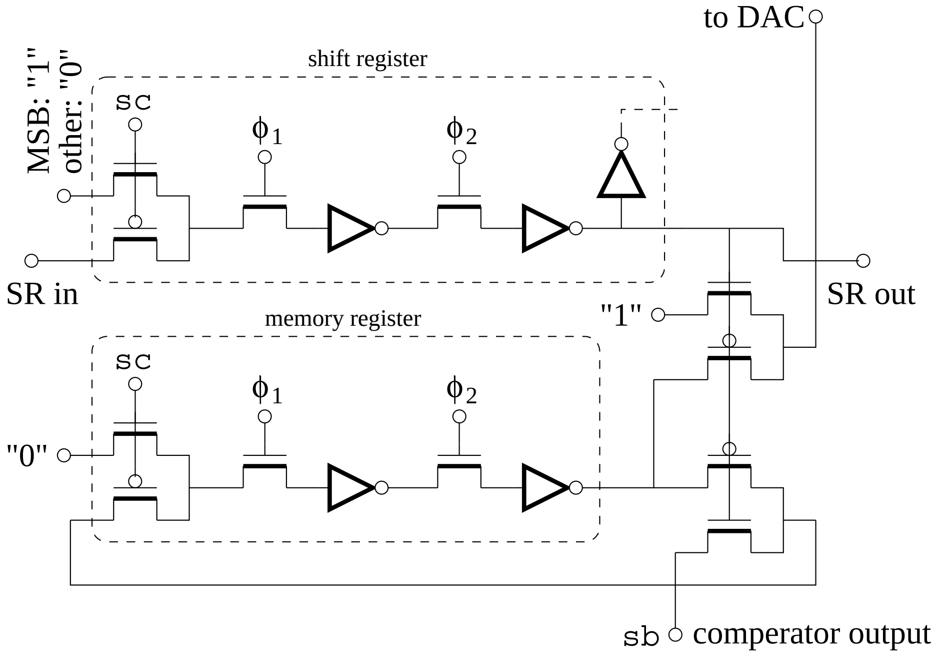


Figure 65^5: SAR bit slice. *Single transistor switches are shown for simplicity. Bit slices are cascaded by connecting "SR in" to "SR out" of the just more significant slice.*

5.4 Chip measurements

A 4 signal slices *width N data path* module RTRL chip was fabricated in the $2.4\,\mu$m CMOS process. Unfortunately the process parameters of this particular batch (the same MPC run as the one the scaled back-propagation synapse chip was a part of) were outside the specified ranges. This has a severe influence on the chip performance; especially signal ranges and speed. It is our hope that we can implement a working system in spite of the poor chip performance (we must raise the reference voltage to accommodate to the reduced input voltage range of, in particular, the current conveyor), thus we shall present some measurement results in this section. Note that the chip functions from an architectural point of view, indicating correct on-chip block interconnection and possibly applicability. A table of chip characteristics can be found in appendix D.3.

Non-linearities and offset errors are comparable to the ones found on the back-propagation chip set, when the reduced signal range is taken into consideration. A typical multiplier characteristic (from the weight change computing IPM) can be seen in figure 66[5]. If the input is large, $v_y \gtrsim 0.8\,\mathrm{V}$, the non-linearity is $D_{\sigma\varepsilon} \lesssim 2\,\%$; for full scale range inputs $D_{\sigma\varepsilon} \lesssim 5\,\%$. Other non-linearities, as the derivative computing "$1 - y^2$"-block which characteristics are shown in figure 67[5] (compare to figure 48[4]), are typically in the order of $D \lesssim 3\,\%$; a magnitude that should not cause any trouble for the learning process. In both figures the neuron activation input v_y has been varied for different values of the multiplexed neuron-input input v_z. A considerable offset on the last input is observed: $|V_{z\mathrm{ofs}}| \lesssim 300\,\mathrm{mV}$. Whether this is acceptable will have to be experimentally investigated; it is probably at the upper limit for an acceptable offset. The large offset is caused by the output offset of the current conveyor used to transform the v_z input to a current. This current conveyor is the same as the one used for the multidimensional IPMs and is thus designed to source a much larger current than necessary for the v_z transresistance; it should be redesigned.

The other possibly problematic offset error is the output offset error of the "$1 - y^2$"-block that computes the neuron derivative. This was also the case for the back-propagation neuron chip; the offset error is of the same magnitude as for this chip and must be dealt with in the same way (cf. section 4.4). Other non-idealities on the chip are acceptable for the learning process.

The input and output for an edge trigged neuron activation sampler is shown in figure 68[5] which illustrates its applicability to prohibit race around in the feedback neural network: The sample time is in the order of 100 ns; the output settling time seen in the figure is approximately $1\,\mu$s. The charge injection is acceptable.

On none of the chips tested, the weight change output auto offset compensation scheme worked. Note that according to SPICE simulations of the circuit, as shown in figure 69[5], the auto zeroing circuit topology is valid. But whether the circuit malfunction is caused by a layout error or by the out-of-specified-range process parameters we have not been able to determine (no layout errors have been found, though) because of the lack of internal test points. Measurements seem to indicate that the current comparator does not work, though.

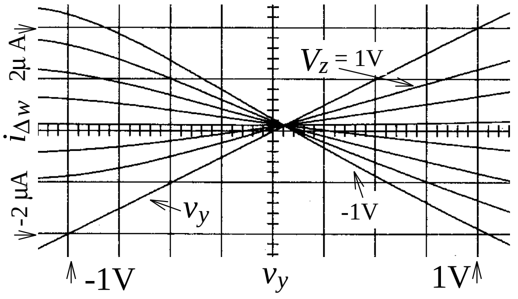


Figure 66[5]: Weight change IPM element characteristics. *Measured weight change current as function of neuron activation input for different network inputs (IPM single element characteristics). Notice the reduced input range and the v_z offset.*

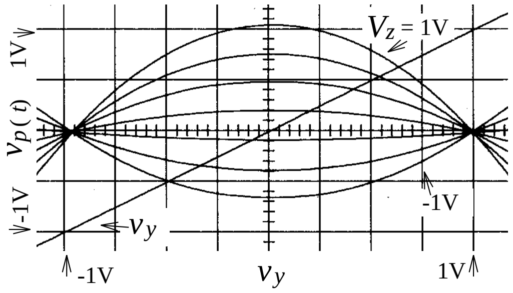


Figure 67[5]: Tanh derivative computing block characteristics. *Measured neuron derivative output as function of neuron activation input for different network inputs (parabola block characteristics). Notice again the v_z offset.*

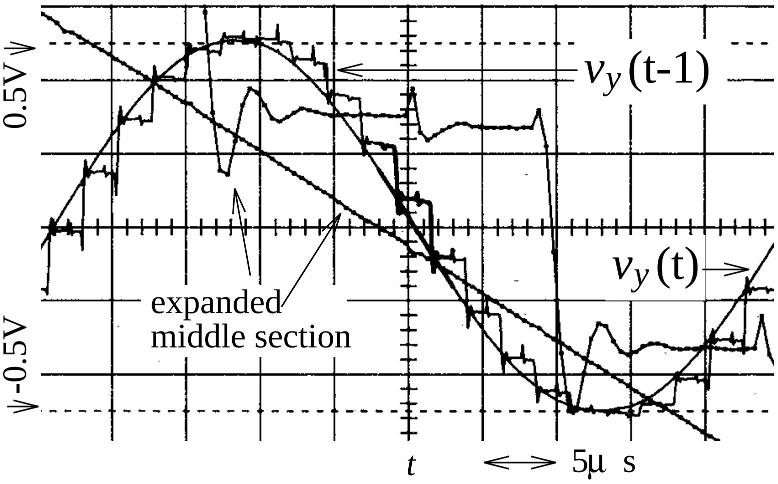


Figure 68[5]: Edge trigged sampler sampling. *Measured (half scale range) input and sampled output. The effect of using two cascaded samplers is clearly seen: Charge injection occurs both at the sample time and half way through the hold period (at both clock edges).*

5.5 System design

For the completion of the RTRL ANN system, we need some hardware in addition to the synapse- and neuron chips and the *width N data path* RTRL chips. As mentioned in section 4.5, we design a RTRL/back-propagation hybrid system to save hardware (and design time). Thus, much of this hardware is basically already present in our system:

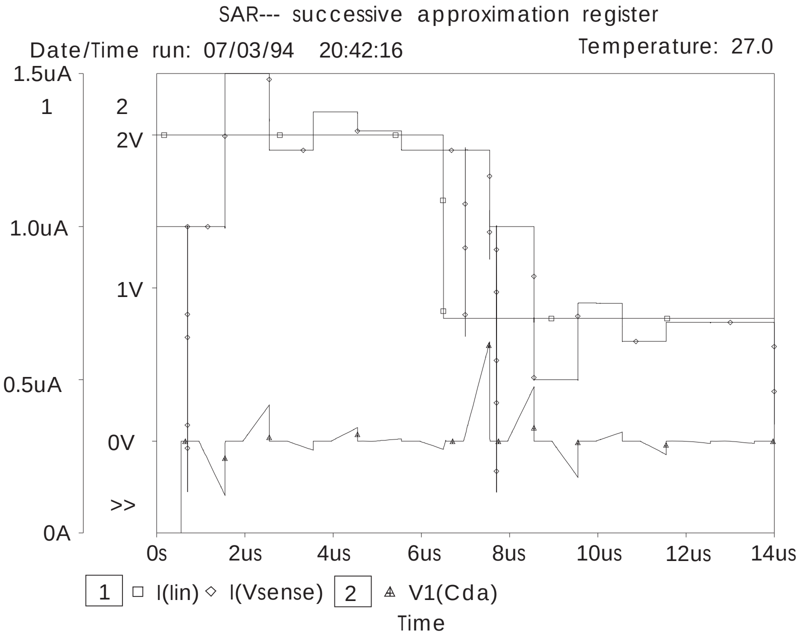


Figure 69[5]: Auto zeroing simulation. *SPICE using simple DAC and opamp models. Top: input (□) and output (◇) currents. Bottom: current comparator input voltage (△). Notice how the output current increasingly accurately approximates the input current (two trials). The input voltage indicates whether the output current is too large or too small.*

- The digital weight backup memory.
- The digital, serial weight updating hardware including A/D and D/A converters for interfacing (the *width 1 data path module* when including:)
- The finite automaton for system control.
- The ANN environment.

To complete the RTRL system we also need:

- The multiplexor (the *width $N + M$ data path module*).
- A derivative variable RAM including A/D and D/A converter for each RTRL chip in the system.

In this section we shall describe the complete system. For a complete system schematic refer to enclosure III (see also appendix D.4).

5.5.1 ASIC interconnection

The application on which we want to apply the RTRL system is the prediction of *splice sites* in pre-mRNA molecules (cf. section 2.3.3). Using a recurrent network we will need at least ten neurons (of which one is an output neuron) and tree letter inputs (corresponding to one amino acid code) taken from a 4 letter alphabet (*Brunak and Hansen* [31]). Using unary coding this gives twelve inputs (plus biases)†. Mapping this topology on our chip set requires two synapse chips and three neuron chips for the ANN. Assigning two inputs for offset compensation (the expected total MVM output offset is $\sqrt{2} \cdot \sigma_{I_{wz}}$ where $\sigma_{I_{wz}}$ is the single chip output offset standard deviation) and two inputs for the thresholds‡, this gives a *RTRL system topology* of 16 inputs and 12 neurons♮.

The custom chip interconnections when the system operates in RTRL mode can be seen in figure 70^5 (assuming the neuron- and RTRL chips have the same number of neurons/signal slices). The $z_j(t)$ multiplexor (`MUX`) is implemented using a standard cascadable analogue multiplexor. The derivative variables RAMs (p`RAM`) are implemented as 8 bit wide static RAMs accessed via 8 bit data converters. The A/D converters are fast flash converters as they are the bottle neck in the system (in terms of speed). For clarity, the target values ($\underline{d}(t)$) and target indicators ($\underline{T}(t)$) are not explicitly shown. These signals are, among various control signals, supplied by the "environment/serial weight updating hardware" block.

† This particular application does not fit the description "massively parallel, possibly adaptive, application specific system having a real-world interface" for analogue VLSI learning systems of chapter 1. This is a toy network, however; the application would benefit from an additional 40 neurons or so. If enhanced performance using network ensembles is employed (cf. section 6.3.1) several hundreds of neurons could be exploited — such a system would be massively parallel. The input(output), on the other hand, would still consist of only 6(1) bit and can thus easily be supplied by (say) a standard AT bus harddisc (which is how the input data is available). A real-world interface is unnecessary.

‡ Connecting ξ synapse chips together the expected total output offset is $\sqrt{\xi} \cdot \sigma_{I_{wz}}$ where $\sigma_{I_{wz}}$ is the single chip output offset standard deviation; thus we must commit ξ synapses per row to offset compensation (cf. section 5.3.2). We use two threshold synapses per row because it is often seen that the neuron thresholds have a larger magnitude than typical synapse strengths.

♮ The system thus has four extra inputs which must be tied to zero. This illustrates a typical problem when using "standard" *building block components* to implement an application specific system: If the application does not exactly match the topology of the building block components, hardware is wasted. A solution to this problem is to make available a range of synapse- and neuron chips (say 8 × 8, 32 × 32, and 128 × 128 synapses and 8, 32, and 128 neurons).

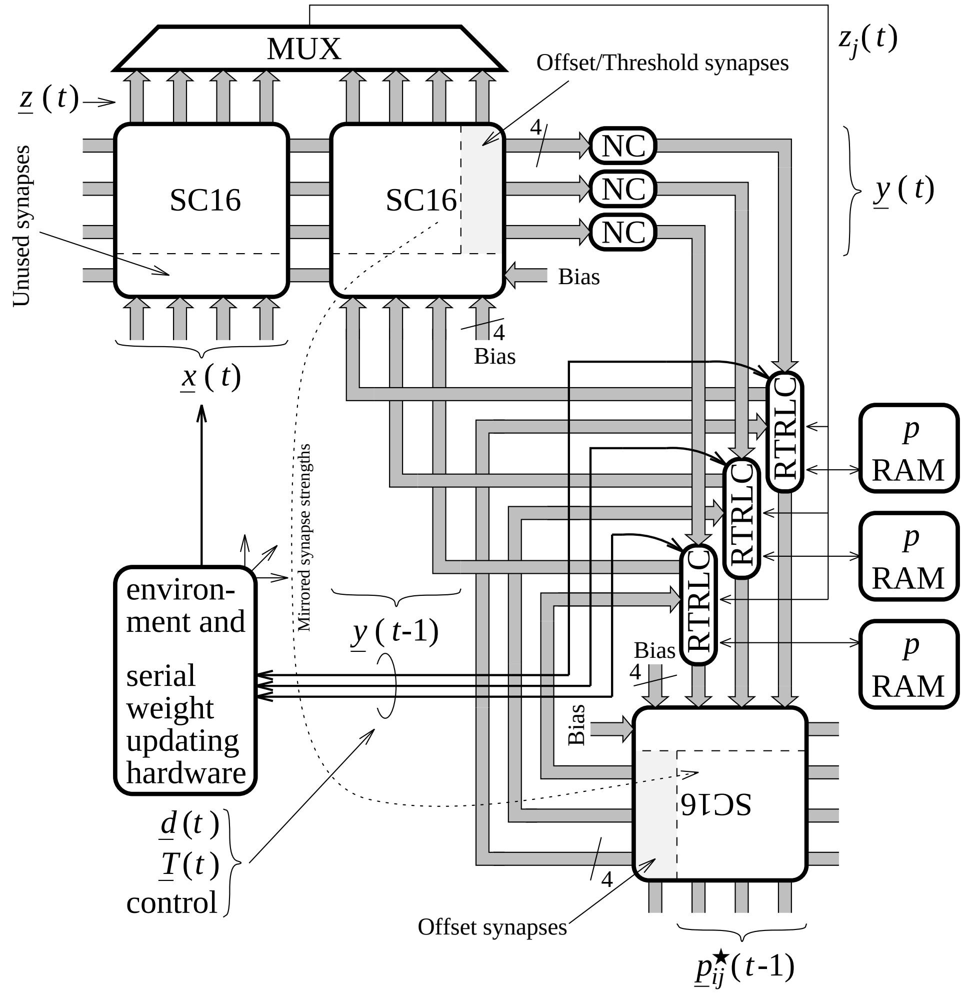


Figure 70^{5}: RTRL ANN basic architecture. *ANN and RTRL chip interconnections when the system operates in RTRL mode. The blocks* RTRLC *are RTRL chips. Input (output) lines of the* SC16 *blocks are accessible at both top and bottom (left and right) in this figure.*

5.5.2 The width 1 data path module

Apart from the weight strength backup memory and the system environment, the components not explicitly shown in figure 70^{5} are part of the *width 1 data path module* of the RTRL system implementation. Though basically identical to the back-propagation learning hardware, some minor modifications are necessary:

All synapses in the systems are given a unique address $\{i, j, l\}$ reflecting the back-propagation topology w^l_{ij}. Also the $\underline{\underline{w}}_\square \underline{p}^\star_{ij}$ MVM for the RTRL algorithm is given an address in this space. However, except for the offset compensation synapses the weight on this chip must be the same as the ANN synapse chip with the $y(t-1)$ inputs (cf. the figure). For this reason the weight backup RAM use a filtered version of $\{i, j, l\}$ as address bus which mirrors the relevant part of the ANN MVM on the $\underline{\underline{w}}_\square \underline{p}^\star_{ij}$ MVM. Doing, for instance, a weight refresh — which is governed by the $\overline{\text{PC}}$ — one would simply run through all the $\{i, j, l\}$s and all synapse sites would get the correct weight.

The high precision digital weight updating hardware used in back-propagation mode (cf. figure 53^4) is also used in RTRL mode; with the addition of a transresistance at the joined RTRL chip current weight change output for compatibility. (The comments in section 4.5 on the virtues of this architecture apply for the RTRL mode also, of course.)

On the RTRL chip the digital i signals for addressing the distributed demultiplexor and the digital k' signals used to access the derivative variables RAM was erroneously combined to save pins on the chip. The error has been fixed by the addition of a digital "$\{i, k'\}$ to i/k' converter" (see enclosure III). The error is still unfortunate, though: the lengthy analogue calculation of the weight change signal can not take place concurrently with the derivative variable RAM updating; this degrades performance.

The whole system is controlled via a large number of digital *handles*. For instance the "sample the neuron activation" signal $\phi_{\mathrm{S}y}$, the "initiate auto zeroing" signal `sc`, or the "current synapse" signal $\{i, j, l\}$ above. The synapse- and neuron chips also needs signals as the `forward`, `reverse` and `learn` signals. In all, some 60 control signals are needed, including the $T_k(t)$s. (Many of these can be combined if the mutual timing requirements are known; for the prototype chip set they were not at the system design time, however. Letting the system controlling finite automaton control the internal timing is a safe (and fast designed) choice. As noted in section 4.5, only the low frequency circuit level system performance (ie. not the speed) can be tested with this system.) The system controlling finite automaton must supply all these 60 control signals. These are placed in registers accessible to a host PC which, as said, is used as this automaton.

5.5.3 The interface

The host PC is interfaced via a standard 16 bit I/O channel at the PC AT (ISA) bus (*Eggebrecht* [68]). In addition to the 60 odd digital handles that controls the RTRL/back-propagation system, the host PC also must provide various analogue control signals that are likely to be changed by the user. For instance learning rate- and neuron activation steepness control signals. Most of these are placed in 8 bit serial DACs; signals for offset compensation are driven by 12 bit DACs.

To monitor how learning progresses, the PC has access to the weight backup RAM and the derivative variables RAMs (which requires a substantial amount of extra hardware). Also, all neuron outputs as well as the output layer reverse mode synapse chip currents and the $\underline{\underline{w}}_{\square}\underline{p}^{\star}_{ij}$ MVM current outputs are available to the PC via 12 bit ADCs. (These are used to offset compensate the synapse chips, hence the high precision.) Having all neuron outputs available also includes the possibility of using the PC to refresh sampled neuron activations in case the on-chip analogue samplers should unexpectedly prove inadequate.

As well as acting as the master finite automaton, the PC provides the environment in which the ANN is placed: It supplies the input signals and target values via 12 bit DACs and reads the network outputs via 12 bit ADCs. For our application we only need binary inputs (and targets); and a 8 bit sampling of the neuron activation would most probably be sufficient for the gradient descent algorithm. As this is a test system, however, we will not prevent ourself from using analogue input/output data as such data set can give additional information on the system performance. For this reason high precision data converters are used.

5.5.4 Algorithm variations

Most of the RTRL algorithm variations listed in section 5.1.2 affect only the algorithm control mechanism (ie. the finite automaton) and are thus easily incorporated in our system:

- *Teacher forcing.* Ignoring the neuron activation samplers at the RTRL chips and configuring the ANN MVM inputs as in back-propagation mode (cf. figure 52[4]) the PC can supply the z_j^{TF} ANN MVM inputs used for teacher forcing. The *initialization channel* of the derivative variable RAM access circuit is connected to zero in our system. Thus when reading the $p_{ij}^k(t-1)$s prior to a weight calculation, the initialization channels rather than the RAM channels should be used when sampling $p_{ij}^k(t-1)$s for which $k \in T(t-1)$ to ensure the sum in (17[5]) is taken over $l \in U \setminus T(t-1)$. (This is assuming target neurons on all neuron chips. If this is not the case one can explicitly write zeros in the derivative RAMs instead.) This approach is somewhat inelegant; *designing* the system to include teacher forcing would require $2N$ two way analogue multiplexors.
- *Relaxation* and *pipelining.* The system is designed to use pipelining. Using the relaxation scheme is just a matter of updating the neuron activations and neuron derivative variables a couple (T_{PD}) of times before updating the weights. Note that, in general, whenever the *neuron activation target set* is known to be *empty* one should omit the weight updating step — partly because of speed but primarily to avoid unnecessary influence of weight updating offsets.
- *Learning by subsequence.* Resetting the neuron activation variables between each subsequence is trivial; one just uses the initialization channels rather than the RAM channels when sampling the $p_{ij}^k(t-1)$s.
- *Random initial states.* Though our system not designed for this, random initial derivative variables are obtained by letting the PC write random numbers in the derivative RAMs (instead of using the initialization channel for initialization). Including the option in the system *design* involves the placement of digital pseudo random generators that can override the derivative variable RAM outputs.
- *Momentum, weight decay, etc.* The observations on the implementation of weight decay, momentum, etc. in chapter 4 also apply for the system in RTRL mode. Note, however, that though the system is not designed for it, it is possible to read the weight changes $\Delta w_{ij}(t)$ from the PC, which actually enables weight decay and momentum among other things in the present system; in a

somewhat laborious way, though.

The real-time recurrent learning system is, at the time of writing, under construction. Thus, unfortunately, we can not present any system level experiments. Hopefully such will be available in the near future.

5.6 Non-linear RTRL

As for the back-propagation system (and other gradient descent learning systems) the hardware implementation of the $\partial g(s)/\partial s$ neuron activation derivative computation is a primary error source in the RTRL system. Inspired by non-linear back-propagation we shall now derive and show how to implement an analogous version of the RTRL algorithm in which the *derivative computation is avoided*: *non-linear real-time recurrent learning* (*NLRTRL*). Actually this *non-linear principle* for approximating a derivative is applicable to any gradient descent like learning algorithm that uses derivatives (RTRL, virtual targets, back-propagation and variations, etc.).

5.6.1 Derivation of the algorithm

Taking the derivative variable definition (16^5) as our point of departure:

$$p^k_{ij}(t) = g'_k(s_k(t))\sigma^k_{ij}(t)\,,$$

we interpret this as a first order Tailor expansion of the equation

$$p^k_{\mathrm{N}ij}(t) = g_k\big(s_k(t) + \sigma^k_{ij}(t)\big) - g_k(s_k(t))\,, \qquad (24^5)$$

which defines the *NLRTRL neuron derivative variables.* As for NLBP we could scale $\sigma^k_{ij}(t)$ in this equation by a factor η/α_N for a more accurate theoretic Tailor expansion when the *NLRTRL domain parameter* α_N is large. However, we are interested in numeric accuracy rather than theoretic accuracy which is why we choose the domain parameter small, $\alpha_\mathrm{N} = \eta$, as we did in the NLBP case. The NLRTRL algorithm (and variations of it) now simply arrives by substituting $p^k_{\mathrm{N}ij}$ for p^k_{ij} in the equations in section 5.1.1.

Simulations using the NLRTRL algorithm have not yet been performed; and an experimental verification of the algorithm functionality must be done before an implementation, of course. However, as both RTRL and back-propagation are gradient descent algorithms — the RTRL p^k_{ij}s and σ^k_{ij}s plays a role very similar to the δ^l_{kj}s and ε^l_{kj}s respectively of back-propagation — and as NLRTRL and NLBP are derived in complete analogous ways, we should expect the simulated performance of NLRTRL to be very similar to that of RTRL. The performance of a NLRTRL hardware implementation is, of course, expected to be superior to that of RTRL as the derivative computation is omitted and as a signal slice multiplier is saved.

5.6.2 Hardware implementation

When mapping the NLRTRL algorithm on hardware it turns out advantageous to use a time offset weight updating scheme

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t-1)\,,$$

rather than (15^5)†. Now, assuming we use the *discrete time NLBP neurons* of figure 56^4 and 57^4 (possibly without the extra error input i_{ε_k}), and assuming we are using the quadratic cost function, the discrete time system of figure 61^5 takes the form shown in figure 71^5.

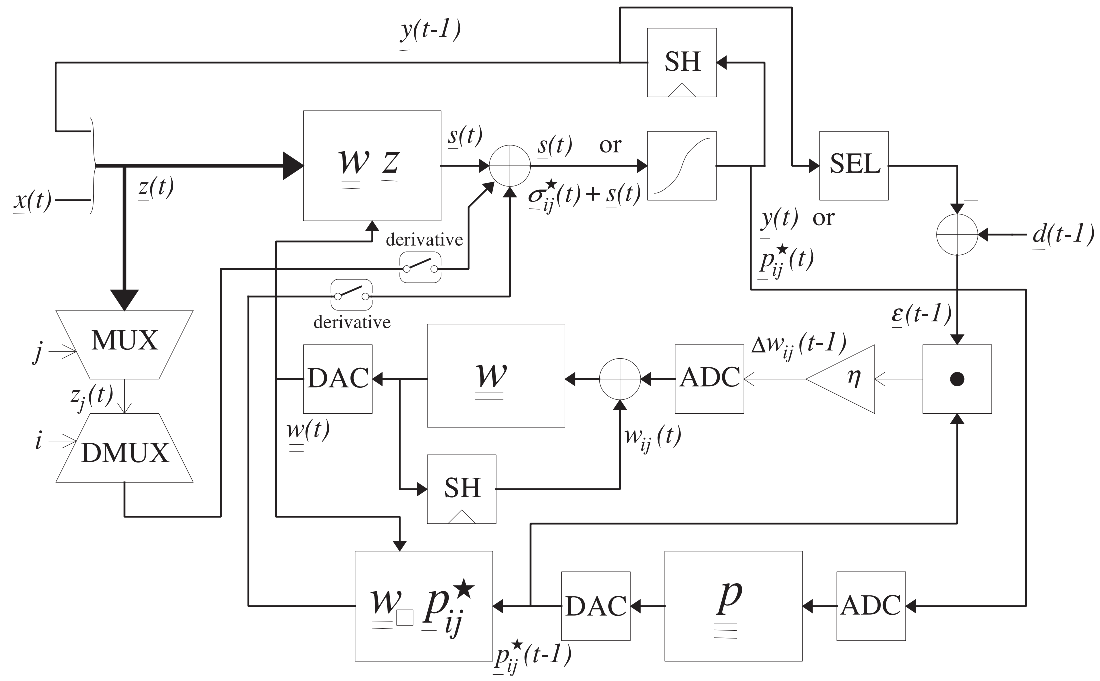


Figure 71^5: Non-linear RTRL system. *Block diagram. This topology considerably reduces the order N data path hardware compared to the original RTRL system and is most probably more accurate. The system uses delayed targets.*

† This is actually the weight updating scheme originally proposed by *Williams and Zipser* [267, 268].

The system is operated as follows: At the end of a learning cycle, $\underline{y}(t-1)$ is sampled. Then $\underline{x}(t)$ and $\underline{d}(t-1)$ are applied and $\underline{y}(t)$ is computed asynchronously. Now, unlike the original system, the neuron chip samples $\underline{y}(t)$. For each $\{i,j\}$ $\underline{p}^{\star}_{ij}(t-1)$ is read from the RAM after which the $\underline{\underline{w}}_{\square}\underline{p}^{\star}_{ij}$ multiplier and the demultiplexor adds $\underline{\sigma}^{\star}_{ij}(t)$ to the ANN MVM output, forcing the neuron chip to calculate $\underline{p}^{\star}_{ij}(t)$. This, as well as the concurrently computed $w_{ij}(t+1)$ is stored in the respective RAMs.

A few notes on the architecture: As the demultiplexor and both MVMs have current outputs, adding $\underline{\sigma}^{\star}_{ij}(t)$ to $\underline{s}(t)$ is trivial. However, doing this $\underline{\sigma}^{\star}_{ij}(t)$ is not explicitly present in the system and we must thus refrain from using the entropic cost function. As the neuron activations are sampled by the neurons for the p^{k}_{ij} computation, the neuron activation sampler on the NLRTRL chip need not be edge trigged (ie. one op-amp can be saved; in all three op-amps (or almost 50% of the computing hardware) are saved per NLRTRL signal slice, not counting the derivative variable RAM access circuit).

5.7 Further work

As for the back-propagation system, we need to carry out system level experiments in order to evaluate the applicability of our proposed chip set. These experiments could suitably include "simulated" (ie. using the PC to perform the digital, fixed point computations) weight change threshold, momentum, weight decay etc. to verify the applicability of these algorithm variations (these simulations can also be performed using the system in back-propagation mode). These experiments are presently carried out at our institute. Other obvious future tasks include a remanufacturing (and possibly redesign) of the weight change offset canceling circuit as well as simulation on and implementation of the proposed NLRTRL algorithm.

Considerations on *process parameter dependency canceling* and *temperature compensation* are necessary before "a volume production" (cf. section 2.7) just as was the case for the other chips designed so far. Also, the considerations on the implementation of *high accuracy calculations* mentioned in section 4.7 (as chopper stabilizing and weight change threshold) apply to the RTRL chip.

In addition to these tasks further work on VLSI implementations of the RTRL (or NLRTRL) algorithm could include a continuous time system version:

5.7.1 Continuous time RTRL system

One of the very nice features of analogue signal processing is the inherently asynchronous functionality. In our systems so far we have ignored this and dealt with sampled time systems. The back-propagation algorithm can fairly easily be implemented in an asynchronous way because of the feed forward architecture. This is not so easy for the RTRL architecture, though it would be very attractive to do so. In this section we shall roughly outline how a continuous time (or asynchronous) version of the (NL)RTRL algorithm could be implemented in analogue hardware.

Now, instead of using sample-and-hold circuits as feedback in the ANN (cf. figure 60^5; see also figure 74^{B}) we use continuous time low pass filters having transfer functions $H_{y_k}(s)$. Note that the low-pass filter can only hinder parasitic oscillations caused by non-ideal electrical components. As the signs and magnitudes of the connection strengths in the synapse matrix are unknown, these can cause oscillation. It is the learning scheme that must adjust the weights to prevent oscillation — or indeed to cause it. (This applies also to the discrete time system, for that matter.) Using a continuous time feedback the neuron input definition (14^5) becomes

$$z_j(t) = \begin{cases} x_j(t)\,, & \text{for } j \in I \\ h_{y_j}(t) * y_j(t)\,, & \text{for } j \in U \end{cases},$$

where $h_{y_j}(t)$ is the impulse response of the $H_{y_k}(s)$ filter†. For input signal frequencies much lower than the filter cutoff frequency f_{0y} (use for instance $H_{y_k}(s) = (1 - s/2\pi f_{0y})^{-1}$) the network will work as a *relaxation network* (though a "relaxed state" can be an oscillation). For input frequencies approaching the cutoff frequency the network will function somewhat like a *pipelined network* (we can ascribe the filter a delay of, say, $\ln 2/2\pi f_{0y}$)‡.

Choosing, for example, the *NLRTRL neuron derivative variable* definition

$$p^k_{\mathrm{N}ij}(t) = g_k\big(s_k(s) + \sigma^k_{ij}(t)\big) - g_k(s_k(t))\,,$$

we now propose to compute the *neuron net input derivative variables* as

$$\sigma^k_{ij}(t) = \sum_{l \in U} w_{kl}(t) \left[h_{p^l_{ij}}(t) * p^l_{\mathrm{N}ij}(t)\right] + \delta_{ik} z_j(t)\,,$$

where we, as for the ANN itself, have substituted low-pass filters $H_{p^l_{ij}}(s)$ for the delay elements. To ensure stability the low-pass filters are needed also in this equation. Now, if we select

$$H_{p^k_{ij}}(s) \equiv H_{y_k}(s)\,,$$

it still holds that $\partial y_k(t)/\partial w_{ij}(t) = p^k_{ij}(t)$ (or $\partial y_k(t)/\partial w_{ij}(t) \approx p^k_{\mathrm{N}ij}(t)$) as for the original algorithm.

† "$*$" denotes *convolution*: $h(t) * y(t) \overset{\mathrm{def}}{=} \int_0^t h(\tau)y(t-\tau)\,d\tau$.

‡ This is of course vastly simplified. See eg. *Gabel and Roberts* [74] for details on linear systems.

Finally, we must formulate the weight updating rule in continuous time. Quite trivially (15^5) generalizes to an integration:

$$
\begin{aligned}
w_{ij}(t) &= w_{ij}(0) - \eta \int_0^t \frac{\partial \mathcal{J}(\tau)}{\partial w_{ij}(\tau)} \, d\tau \\
&= \begin{cases} w_{ij}(0) + \eta \displaystyle\int_0^t \sum_{k \in U} \varepsilon_k(\tau) p^k_{\mathrm{Nij}}(\tau) \, d\tau & \text{Quadratic} \\ w_{ij}(0) + \eta \displaystyle\int_0^t \sum_{k \in U} \varepsilon_k(\tau) \sigma^k_{ij}(\tau) \, d\tau & \text{Entropic} \end{cases} ,
\end{aligned}
$$

where we have expanded the equation for the cases of the quadratic- and the entropic cost functions.

Thus implementing the NLRTRL algorithm has one unsaid disadvantage: it is *necessary* to do a fully parallel (ie. $O(N^4)$ area!) implementation thus limiting this approach to fairly small systems. Needlessly to say, continuous time NLRTRL is incompatible with our present ANN architecture which uses serial, discrete time weight access. Further research is needed before a continuous time recurrent learning system can be implemented; among other things the simpler discrete time version of the algorithm should be proven operational. (For continuous time recurrent learning systems see also *Ramacher and Schildberg* [194].)

5.7.2 Other improvements

Essentially composed by components also found on the back-propagation chip set, several issues of the RTRL chip are subject for improvements. A list of these can be found in appendix D.3.1.

5.8 Summary

In this chapter add-on hardware for applying real-time recurrent learning to our cascadable ANN chip set was designed. The basic learning algorithm was displayed and the applicability of common algorithmic variations for the implementation in analogue VLSI was discussed. It was shown that doing an $O(N^2)/O(N^2)$ space/time division of the $O(N^4)$ computational primitives per time step was a good choice with respect to scalability, hardware cost, ANN system compatibility and implementation ease. A system level implementation in which most of the computations (the order $O(N^2)$ part) were performed by a synapse multiplier was presented. Results from simulations modeling analogue VLSI non-idealities in this architecture were found in compliance with like simulations by others: The system is generally tolerant to non-idealities with the exception of the weight change offsets, hidden neuron error offsets and neuron derivative computation.

The design of an RTRL chip for computing the $O(N)$ part of the computational primitives was displayed. This chip was implemented using almost exclusively components from the back-propagation chip set. A weight change offset compensation circuit based on a DAC resolution enhancement technique was included on the chip. Unfortunately this chip was malmanufactured which resulted in a very poor computation speed and a reduced signal range (possibly also the malfunction of the offset canceling circuit which did only function in simulations).

A complete RTRL system design based on our various chips was displayed. A 16 inputs, 12 neurons test system for pre-mRNA splice sites prediction was chosen. The learning hardware not implemented on the ASICs — basically the ($O(1)$) weight updating hardware — is the same as used in the back-propagation system (the possibility of using mostly digital weight updating hardware is adding to the virtues of the proposed silicon mapping). For ease of test, the system is controlled by a PC. Various algorithmic variations can be simulated via this interface. The system is presently under construction, thus no experimental results were presented.

A non-linear version of real-time recurrent learning was proposed. A system level implementation of this algorithm was shown and we saw that the implementation was superior to the RTRL implementation in terms of both hardware cost and computational accuracy. We argued that the applicability of this algorithm would be similar to that of non-linear back-propagation.

Finally, we proposed a continuous time version of the non-linear real-time recurrent learning algorithm for analogue VLSI.

Chapter 6

Thoughts on future analogue VLSI neural networks

In this chapter some odds and ends on future analogue VLSI neural network design — which did not fit into the other chapters — are collected. Firstly, some thoughts on gradient descent learning using analogue VLSI are presented. Secondly, we propose an ANN architecture for massively parallel systems that maps well on hardware. Thirdly, we point out that using *analogue VLSI neural network ensembles* must be a future trend and we propose a weight refreshing scheme based on such ensembles. Finally, we reflect on combining read-only- and plastic- synapses in analogue VLSI computational neural networks.

6.1 Gradient descent learning?

In this work we have investigated the possibility of implementing supervised learning with a teacher — or more precisely: gradient descent learning — in analogue VLSI neural networks. Being wiser from our experiments carried out so far, we can ask the question: does gradient descent learning in analogue VLSI have a future? Perhaps. Though unsupervised learning or learning with a critic is possibly better suited for the technology (because of the lack of a good analogue memory) the need for massively parallel pattern recognition engines is evident (eg. *Hertz et al.* [95], *Sánchez-Sinencio and Lau* [206], *Ramacher and Rückert* [193]) which strongly encourage the development of efficient learning-with-a-teacher algorithms.

Even should our RTRL/back-propagation system prove able to solve classification/regression tasks (which is indeed our expectation) this small scale system does not *prove* the applicability of the learning schemes for massively parallel implementations. And, though we believe it demonstrate some noteworthy points, it is surely not the ideal solution. From an analogue VLSI point of view, present learning-with-a-teacher schemes have some serious draw-backs. Most notably in relation to offset errors (and for very large systems probably also in relation to the dynamic range of the synapse strengths). Eliminating offset errors (by offset canceling or chopper stabilizing) as proposed in this thesis is a solution probably ensuring the applicability of gradient descent learning. However, it is not a *good* solution: It requires precision circuitry and is thus not "neural" of nature. A "neural" way to deal with weight change offsets could be use of a weight change threshold; perhaps induced via a highly non-linear weight change multiplier. Other (or additional) procedures to deal with weight change offsets could be to somehow (i) increase the "*learning loop gain*" (corresponding to a very large learning rate) or (ii) *AC-couple* the *learning hardware* to remove the dependency of DC offsets.

At any rate, it is our profound belief that the ultimate implementation of hardware learning with a teacher requires *research in learning algorithms* as well as research in the electronic implementations. Such research must, of course, focus on the limitations of analogue VLSI, and the resulting learning algorithms should resemble popular simulated algorithms in order to attract application people. The elegant solutions, found in non-linear back-propagation or weight perturbation, to come around the problem of computing neuron activation derivatives are good examples of how to bend the algorithm to meet the technology requirements. The human brain is obviously capable of doing reliable learning using an inaccurate technology†. Perhaps we should seek further inspiration from neuro-biology?

† The computational part of the brain being totally embedded in sensors and actuators can not possible use "learning with a teacher" as we know it. This does not mean that such learning algorithms are somehow inferior; we must *always* use all possible information when solving a problem (ie. we must use learning with a teacher when we can).

6.2 Neuron clustering

Our present scalable ANN architecture which, in principle, can implement any ANN topology is well suited for many present applications. For the implementation of *huge, massively parallel systems*, however, the scalable principle will not hold. The implementation of neurons with, say, millions of synaptic inputs is beyond the capabilities of the technology. The required dynamic range of the synapses and neuron ability to sink current put a bound, somewhere, on the neuron fan-in; to say nothing of the impact of offset errors and of electrical parasitics degrading the speed of such an architecture. The chip input/output bottle neck and inter-chip routing would also be problematic. One could imagine that learning in such a system using conventional algorithms would be difficult (*Krogh et al.* [125], *Benedict* [21], *Haykin* [93], *Houk* [102]) — even using an "ideal" learning machine; using the limited precision technology of analogue VLSI, learning would (almost surely) prove very hard (increased fan-in requires increased precision, cf. section 4.1, section 3.3, section 2.3.1).

For applications using huge networks (this could be in robotics, for instance) an alternative network topology must be found. From an analogue VLSI point of view some kind of *neuron clustering* would be advantageous: we propose an architecture consisting of sparsely interconnected modules of densely connected neurons, see figure 72[6]. The individual clusters would solve different, reasonably complex subproblems (possibly the same problems as other clusters to ensure fault tolerance at a module level (see also the section 6.3.1)); ie., the global problem will be solved in a divide-and-conquer manner. See also *Jacobs et al.* [112], *Haykin* [93], *Houk* [102]. Note that the input data structure of some problems (eg. that of visual systems) *do* need large fan-in input layer neurons (see eg. *Sánchez-Sinencio and Lau* [206], *Masa et al.* [157]); so, we would still need large fan-in (cascadable) modules dedicated for such peripheral tasks. For high level data processing, however, the need for large fan-in is not so evident (compare to models of the human brain, *Rumelhart et al.* [200], *Miles and Rogers* [166], *Joublin et al.* [116], *Mountcastle* [172]); this *neuron cluster* ANN topology might be applicable for *general, high level computational ANNs.*

The *cluster size* and *topology* will obviously have a tremendous impact on system functionality; implicit constraints are put on the problem architecture. Thus, for an efficient system, we need to do excessive research in the areas of cluster size and topology and cluster interconnection architectures. Also, the problem of teaching such a system needs to be addressed. Questions such as "do we need more than one type (size, topology) of clusters?", "must the cluster modules be reconfigurable?", and "should the modules be cascadable?" would be asked. The learning scheme should probably involve both supervised and unsupervised learning. Now, it would be *most* convenient if a single cluster of neurons would fit on a chip. In this case the problem with the chip input/output bottleneck would be reduced and inter module communication could efficiently take place using robust neuron activation signals (eg. pulse frequency modulation). On-chip communication could also be optimized (with respect to speed, power consumption, area, etc.) when the

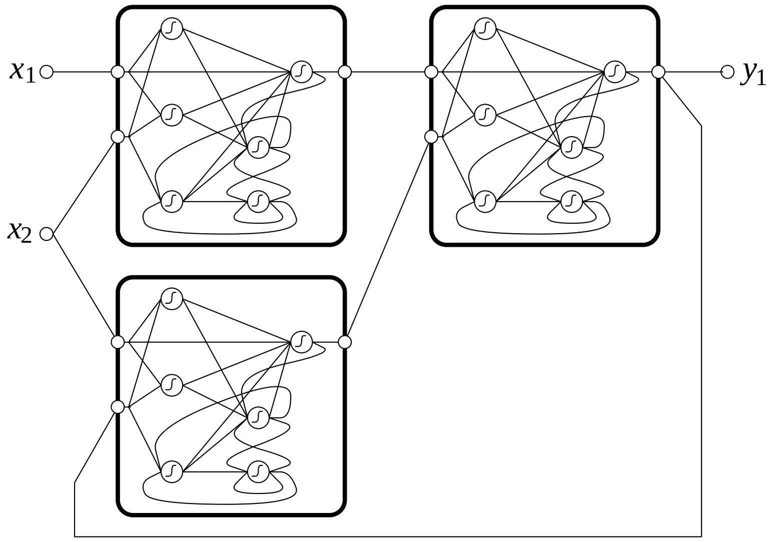


Figure 72[6]: Neuron clustering. *From a VLSI point of view this is a very attractive network architecture. The different blocks could be individual chips. Possibly some kind of restricted cascadability/reconfigurability would be needed.*

local ANN topology is known a priori and no off-chip communication of "intermediate signals" (as synapse outputs) takes place. Using CMOS VLSI technologies of today, the integration of about 100 neurons and 10000 synapses (including learning hardware) would be realistic.

The high level data processing part of the human brain (the *cerebral cortex*) is organized in 4–6 vertical layers. Communication within the cerebral cortex is mostly between such layers and is distinctively local of nature; there are only few long-distance connections. Further, the cerebral cortex seems to be organized in (more or less) disjoint patches (*Rumelhart et al.* [200], see also *Miles and Rogers* [166]). This has inspired some authors to propose a modular model of the cerebral cortex, arranging the neurons in disjoint, densely connected "columns" mutually sparse interconnected (*Mountcastle* [172]; see also *Joublin et al.* [116]); very like our proposed cluster architecture for analogue VLSI implementations. We believe that this topology would be a good starting point for investigations on ANNs with clustered neurons. Others have also proposed such neuron clustering architectures inspired by the hardware friendly sparse connectivity (*Joublin et al.* [116]).

It should be noted that many reported chip architectures actually resemble the proposed neuron clustering architecture in the sense that a densely interconnected, fixed (or reconfigurable to some extend) architecture ANN is integrated on-chip in a non-expandable way, using neuron activation for inter-chip communication (eg. *Graf and Henderson* [84], *Valle et al.* [251], *Castro et al.* [42], *Hamilton et al.* [88], *Serrano et al.* [216]). However, to the best of our knowledge, an exhaustive investigation of chips architectures vs. system generality has yet to be carried out (though cf. *Johansen and Foss* [114] on the problem of modeling complex systems with ensembles of simple models).

6.3 Self refreshing system

As mentioned in section 2.2.3 one of the major concerns of analogue VLSI ANN research is the issue of synapse strength storage — especially in connection with on-chip learning: The only true long-term analogue memories (as floating gate devices) are tedious to program, probably expensive, and not well suited for adaptive systems. The area penalty of high resolution on-chip digital storage or some kind of *quantize-regenerate* scheme compromises the advantages of analogue VLSI. Finally, for systems where the use of external components is unacceptable or where a parallel weight updating is necessary, using capacitive storage with a RAM back-up, as in our work, is inapplicable.

Now (still referring to section 2.2.3), for certain applications using an *unsupervised learning* algorithm or *learning with a critic* we can, in an elegant way, apply *refresh by relearning* on systems using simple capacitive storage, using pure analogue signal processing circuitry. The question that now arises is this: Is it possible, without storing training patterns, to apply a like *refresh by relearning* scheme in application areas using *learning with a teacher* as pattern recognition and related tasks? Using the *self-repair* properties of *neural network ensembles*, this is the case for certain applications.

6.3.1 Neural network ensembles

A *neural network ensemble* (*Hansen and Salamon* [90], *Hansen et al.* [89]) is a collection of neural networks (often topologically identical) trained, using different initial states, to solve the same problem. (The training algorithm applied could for instance be back-propagation.) Applying the ensemble to a given problem, the solution given by this is a *consensus decision*, based on the individual network outputs. This could be the *majority decision* for binary outputs or a *weighted sum decision* (as "stacked regression", *LeBlanc and Tibshirani* [136]) for analogue outputs. Now, providing that

- the individual networks perform reasonably well and
- the errors of the different networks are to some degree independent

the consensus decision will be superior to that of the individual networks. More specifically: for a classification task where the probability of doing a misclassification is p for each individual network and providing the network errors are independent, the error probability for an ensemble of N_{E} networks is

$$p_{\mathrm{E}} = \sum_{k>N_{\mathrm{E}}/2}^{N_{\mathrm{E}}} \binom{N_{\mathrm{E}}}{k} p^k (1-p)^{N_{\mathrm{E}}-k} \, .$$

For $p < 1/2$ we have $p_{\mathrm{E}} \to 0$ for $N_{\mathrm{E}} \to \infty$. Using ensembles of $N_{\mathrm{E}} = 7$ networks, for instance, *Hansen et al.* [89] have reported a 20%–25% improvement of the individual network performance on a handwritten digit recognition problem.

While improving performance is always important — sometimes even at a very high cost — the properties of neural network *ensembles* are particularly important in relation to *analogue VLSI ANN implementations*:

- *Fault tolerance.* While the potential fault tolerance of neural networks is repeatedly emphasized in the literature, networks trained using standard approaches (as back-propagation) do not exhibit a very high degree of fault tolerance (*Šerbedžija and Kock* [215], *Woodburn et al.* [270], *Neti et al.* [181]): Though insensitive to small weight perturbations (recall that the gradient descent solution is given by $\partial J/\partial w_{kj} = 0$) the network is *not* insensitive to the complete loss of a synapse as the network architecture is kept as simple as possible to ensure good generalization abilities. Using neural network ensembles provides a simple way to introduce fault tolerance. From an analogue VLSI point of view this fault tolerance is important because analogue signal processing requires better components than digital signal processing (and are thus more sensitive to processing errors). For a fault tolerant *system* the consensus decider must also be implemented using redundant hardware.
- *Enhanced performance.* Implementing ANNs in the limited accuracy technology of analogue VLSI, the performance of our ANN solutions is bound to be inferior to that of high precision simulated networks (see eg. section 2.6, *Tarassenko et al.* [238], *Lansner* [133], *Ramacher and Rückert* [193]). Whether this is acceptable or not is application dependent; if it is not, neural network ensembles provides a simple way to enhance the analogue ANN performance.

  Regularly it happens that an ANN trained using gradient descent gets stuck in a local minimum of the cost function (ie. the network does not solve the task at hand after learning). For recall mode systems this is not a fatal incident; one can just repeat the training phase with other initial weights. For an *adaptive system* trained on-line this is not so. There will be only one chance to learn the task. The enhanced performance offered by ANN ensembles might well prove crucial in such systems.

The implementation of a neural network ensemble is very simple; requiring only the design of the *consensus decider* (assuming we have a collection of acting neural networks). As mentioned in the previous chapters such simplicity is important to analogue VLSI design. Further, duplicating a whole ANN, say, $N_{\mathrm{E}} = 7$ times for the implementation of an ensemble is, computationally, a very expensive procedure. Thus, even for moderate size networks, parallel computations might be necessary; for a dedicated hardware implementation, ensemble methods are hardware hungry — thus the potentially small sizes of analogue computing elements makes *analogue VLSI an ideal technology for neural network ensembles. And vice versa.*

The consensus decision of a neural network ensemble being superior in performance to the individual networks provides a way to do *self-repair* in a system haunted by degrading weights (*Hansen and Salamon* [91]): Using the consensus decision as *target values* for all the networks, we simply apply a supervised, on-line learning algorithm to make weight updates after each presentation of input patterns while the system is running in recall mode (we call this a *consensus trainer*). Under certain conditions this scheme can keep weight deterioration in check; this can be exploited for *weight refresh* in analogue VLSI neural systems using simple capacitive storage and on-chip learning with a teacher, see figure 73[6].

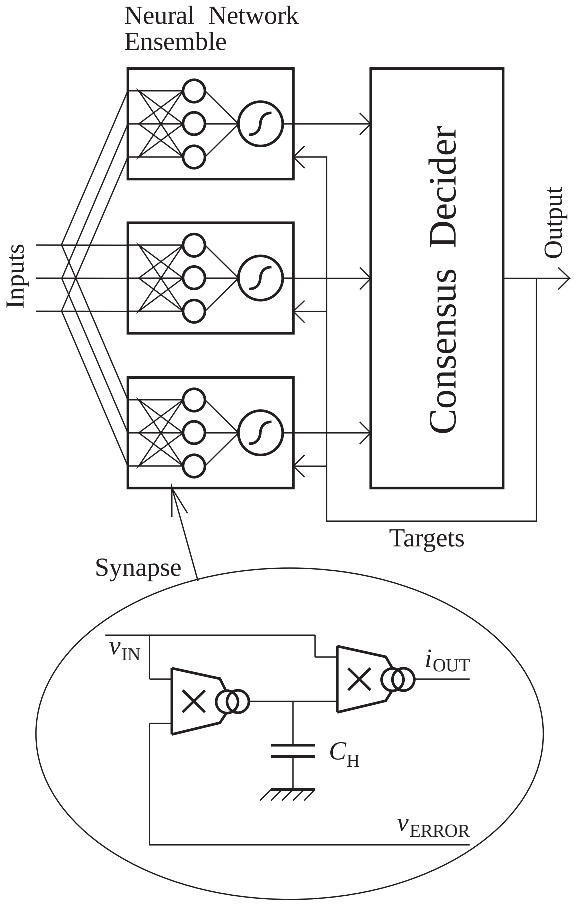


Figure 73[6]: Self refreshing ANN system. *The self-repair properties of a neural network ensemble is used to retain the weights stored on the pure capacitive analogue synapse storage. Using occasional external target values (read adaptive systems) will prolong the time to memory exhaustion.*

Now, assume a weight perturbation gives a proportional network error probability change (whether caused by the learning algorithm or the weight deterioration mechanism)† and assume we can model the discrete time weight change as

$$\Delta w_{kj}(t) = \underbrace{-\eta \frac{\partial \mathcal{J}^{\mathrm{C}}(t)}{\partial w_{kj}(t)}}_{\text{learning scheme}} + \underbrace{(\delta + n(t)) T_{\mathrm{cyc}}}_{\text{weight deterioration}} \ ,$$

where δ is a constant (worst case) weight droop rate and $n(t)$ is a noise term; and where T_{cyc} is the time needed to do a full weight matrix update (*learning cycle*) and $\mathcal{J}^{\mathrm{C}}$ is a cost function computed using the consensus decision as target values (the *consensus cost function*). (Note that the δT_{cyc} product is equivalent to a weight change offset.) In this case it turns out that when the weight *restoration efficiency* $\Lambda \equiv \eta/(\delta T_{\mathrm{cyc}})$ is larger than a critical value, Λ_{crit}, the system performance will remain largely unchanged over a period of time that depends on the noise level. For $\Lambda \leq \Lambda_{\mathrm{crit}}$ there is an abrupt transition to a regime where the system performance degrades rapidly (*Hansen and Salamon* [91]).

Though an ANN ensemble using a consensus trainer can sustain a weight deterioration which would rapidly destroy a non-retrained ensemble (to say nothing of a single network), the life time is finite: Patterns once misclassified by the consensus decider are never relearned as all the networks are trained to imitate the misclassification (the probability of doing new misclassifications is finite; partly because of noise, *Lehmann and Hansen* [145]). As for other *refresh by retraining* schemes, the system would tend to forget the classification of scarcely occurring input data. If it is possible occasionally to apply an external teacher, which would be the case in adaptive systems, the life time might be profoundly improved. Actually, in such systems the inherent forgetfulness of the systems might be an advantage; if old memories are considered irrelevant.

Intuitively, the critical *restoration efficiency* would be expected to increase as the system size increases, thus limiting the network size to which we can apply a consensus trainer for weight refresh in an analogue system. Further, for very large systems the extra hardware used for the ensemble might be a hindrance for implementation; for small systems, on the other hand, the reduction in hardware cost for a more complex refreshing scheme might easily accommodate the extra hardware for consensus refresh — especially if the improved performance of the ensemble is taken into consideration.

In all: We propose to use a *consensus trainer* to do *refresh by relearning* in small, adaptive, analogue ANN systems with on-chip *learning with a teacher*, that use simple capacitive weight storage and function in an ever changing, hostile environment. The applicability of the scheme is a subject for on-going research

† Because of the pronounced non-linearity of ANNs this is a highly inaccurate — though conservative — approximation. Assume we use gradient descent learning with small learning rate and a quadratic cost function, and approximate the neuron activation functions by $g(s) = s$ for $|s| < 1$ and $g(s) = \mathrm{sign}(s)$ otherwise: For saturated neurons a small weight change does not alter the error probability. For non-saturated neurons the output error change is linear in the weight change.

presently carried out at our institute (*Lehmann and Hansen* [145]). Of course, memory destruction at power loss will, for most artificial neural systems, be unacceptable. We will need means to read the volatile synapse strengths for backup purposes and for replication. Or, alternatively, we could use a combination of read-only and dynamic synapse memories:

6.4 Hard/soft hybrid synapses

Even if we could, in a convenient manner, read the synapse strengths of an ANN using simple capacitive storage, the memory restoration after a power loss or the synapse strength down loading for volume manufacturing might well prove tedious. In both cases retraining would be necessary and an in-system back-up memory for in-site memory restoration is not necessarily compatible with an analogue ANN system. If only *short term adaptations* are needed a solution could be the use of "*hard/soft hybrid synapses*" consisting of a pre-programmed non-volatile (possibly read-only) part and a perturbation stored in a volatile capacitive memory. The idea is to (i) use a pre-determined template (for instance obtained via simulations as in *Masa et al.* [157]) to generate the non-volatile (hard) parts of the synapse strengths during manufacturing (implemented for instance as scaled transistors or floating gate devices) and to (ii) use an on-chip learning algorithm for adapting the system via the volatile (soft) part of the synapse strengths. The hard memory part would reflect the *behavioural model* of the system and the soft memory part would reflect the current *working conditions*.

Consider for instance a robot walking on sand/snow/earth/pavement/mud/-pebbles. The basic locomotive behaviour could be pre-programmed ("place one leg in front of the other", "keep the balance", etc.) while temporary adaptations to the current ground cover would be determined using real-time learning. In such applications adaptive, analogue systems would be tremendous powerful.

Chapter 7

Conclusions

In this thesis the implementation of three analogue VLSI neural systems was presented: (i) a cascadable chip set for recall mode neural networks, (ii) a variation of this chip set including on-chip error back-propagation learning in a hardware efficient way, and (iii) add-on hardware for doing real-time recurrent learning on the cascadable chip set using a realistic amount of hardware and time. The recall mode system was tested experimentally both at a chip (electrical) level and at a system level. The two learning system were tested at a chip level; the learning systems are, at the time of writing†, under construction.

During our recall mode chip development we reviewed different chip- and network- architectures as well as different basic building block components as memories, multipliers and thresholding functions. We settled on a two chip cascadable system (a "synapse chip" and a "neuron chip" using analogue voltage/current signalling) capable of — in principle — implementing any ANN topology using first order deterministic neurons. Also, we chose to use simple capacitive synapse strengths storage with a digital RAM backup, a MOS resistive circuit based synapse multiplier, and a hyperbolic tangent neuron activation function based on parasitic bipolar transistors. In addition to the basic synapse chip, we showed how a special "sparse input synapse chip" could efficiently exploit the chip input bandwidth when unary coded network inputs were used.

A 4×4 synapses synapse chip and a 4 neurons neuron chip were fabricated in a standard $2.4\,\mu$m CMOS process. Our measurements on these chips showed a $\gtrsim$ 10 bit synapse resolution, non-linearities below 16% (on most quantities below 3%) and offset errors below 10% (on most quantities below 5%); magnitudes compatible

† September 1994.

with a large range of applications. A system using full size (100 × 100) synapse chips was estimated to do 3.8 GCPS per synapse chip. Measurements on a 8–4–1 experimental perceptron (solving the sunspot prediction problem) based on the chip have shown a learning error slightly worse than that of an "ideal" simulated network.

For classification- and regression tasks multi layer perceptrons trained by error back-propagation are often employed. A fully parallel VLSI implementation of this algorithm gives a $O(N^2)$ improvement in speed compared to a serial implementation†; usually at the cost of 3 times the synapse hardware and twice the inter module wiring compared to a recall mode system. If the physical size of the system is important or if the learning scheme is employed only occasionally the additional hardware can severely restrict the system applicability. We showed how to implement back-propagation without *any* extra synapse hardware or inter module wiring, using the MOS resistive synapse multiplier in a novel configuration which exploits its bidirectional properties — at the cost of discrete time and at most a $O(N)$ improvement in speed compared to a serial approach.

A 8 × 4 synapses back-propagation synapse chip and a 4 neurons back-propagation neuron chip were fabricated in a standard 2.4 μm CMOS process. Our measurements on these chips showed a $\gtrsim$ 10 bit synapse resolution, non-linearities below 6% (on most quantities below 3.5%) and offset errors below 75% (on most quantities below 12%); magnitudes compatible with a range of applications if offset canceling is applied to critical signals. A system using full size (100 × 100) synapse chips was estimated to do 4 GCPS per synapse chip and 0.25 MCUPS (serial weight update as the digital weight back-up RAM puts this restriction on the system). We showed how to implement a back-propagation system based on the chip set. In addition to the chip set this was basically a finite automaton controlling the system and the weight updating hardware implemented using digital components.

The very powerful recurrent neural networks can solve a larger class of problems than perceptrons. Real-time recurrent learning can train a completely general network architecture and have several nice properties with respect to a VLSI implementation. It does, however, require a massive order $O(N^4)$ computational primitives per training example. We showed how dividing the computational primitives between the space and the time domain as $O(N^2)$ to $O(N^2)$ was a good choice with respect to scalability, hardware cost, speed, implementation ease and compatibility with our acting recall mode neural system. We showed how we could perform learning in our recall mode system by adding a synapse multiplier and a scalable "RTRL chip" consisting of an order $O(N)$ "signal slices"; plus weight updating hardware of order $O(1)$ and a finite automaton controlling the system.

A 4 signal slices RTRL chip was fabricated in a standard 2.4 μm CMOS process; unfortunately this chip was malmanufactured resulting in reduced signal range and speed. Our measurements showed a topological functionality, non-linearities below 5% and offset errors below 64% (on most quantities below 10%); magnitudes

† In a system with N neurons

possibly compatible with the algorithm if offset canceling is applied to critical signals. Our implementation of a 12 neurons, 16 inputs system was displayed.

⋆ ⋆ ⋆

Using a digital RAM as back-up for the synapse strengths obviously restricts the learning scheme efficiency: only serial weight change is possible and D/A and A/D converters are needed. However, we showed that implementing (most of) the $O(1)$ weight updating hardware using digital electronics exhibits a range of virtues: The possibility of implementing high accuracy circuits using digital components enables the applicability of advanced weight updating schemes; eg. momentum, which is not likely to function in a simple analogue system as weight change offsets are amplified. Also, it turns out that the scheme can reduce the minimum effective learning rate and the weight change offset, which are of major concern in analogue implementations of learning algorithms. Other algorithmic variations as weight decay and weight change threshold are also readily and accurately implemented in the digital domain. Note, that since the RAM weight updating hardware scale as $O(1)$ the hardware cost is not very important. Finally, placing the synapse strengths in digital RAM is convenient for back-up purposes which is important for real applications.

We noted that, though applications exist that can tolerate parameter variances induced by temperature drift, analogue neural networks must in general be temperature compensated. Some implementations (as the back-propagation system above) also need process parameter variation canceling in various subcircuits; a scheme for this was outlined.

In relation to the limited accuracy of analogue computing systems, we noted that two problems of teaching artificial neural networks using gradient descent based algorithms were especially severe: Offset errors (primarily on the weight change signal) and neuron derivative calculation.

We presented several procedures to reduce offset errors: One based on a DAC resolution enhancement technique and another utilizing chopper stabilizing.

To ensure the correct sign of the computed neuron derivative (which is of profound importance for convergence) we introduced a deliberate offset in the computing hardware. We also suggested that "clipping" the computed quantity was a possibility. However, attacking the problem from an algorithmic starting point was more attractive by far: We proposed to use "non-linear gradient descent" for analogue VLSI:

The novel non-linear back-propagation learning algorithm was displayed. This algorithm has two important properties for a hardware implementation: (i) no activation function derivatives need to be computed and (ii) the back-propagation of errors is through the same non-linear network as the forward propagation. We showed that this implies that an electronic implementation can model the algorithm much more accurately than is possible for ordinary back-propagation; design efforts can be put into the electrical properties of the system components. We proposed hardware implementations of both a continuous time version and a discrete time version of the algorithm; combining the latter with our hardware efficient back-propagation synapse chip, we saw that the implementation of non-linear back-propagation was possible using *virtually no extra hardware* compared with the recall-mode system. Simulations using the non-linear back-propagation for learning the NETtalk problem have shown a performance very similar to ordinary back-propagation.

We derived a non-linear version of the real-time recurrent learning algorithm and argued that this (compared to ordinary real-time recurrent learning) would have properties very similar to non-linear back-propagation; both at an application level (performance) and with respect to hardware implementations. Our system level implementation of the algorithm showed that half the hardware on the "RTRL chip" could be saved. We also proposed a continuous time version of the non-linear real-time recurrent learning for exploitation of the asynchronous properties of analogue VLSI.

We saw the implementation of *analogue neural network ensembles* as a future trend of the field: Using ensembles introduce the much glorified but seldom implemented fault tolerance in analogue neural systems. More importantly, though, using ensembles enhance performance: Because of the limited accuracy of the technology, analogue VLSI neural systems are bound to be inferior in performance compared to "ideal" simulated networks. For adaptive systems this is particular severe as training data is not necessarily reproducible; faulty learning is intolerable. Neural network ensembles are expensive in terms of hardware; thus analogue VLSI is an ideal technology for neural network ensembles. And vice versa.

While the cascadable solution of our various chips set functions for a limited number of cascaded devices, we argued that the generalization to a truely arbitrary size for huge neural networks is not in compliance with the analogue technology: such systems requires infinite dynamic range of, for instance, the synapse strengths. Using highly non-linear synapse multipliers the cascadability can be improved (though it will still be limited). We also proposed to use a network topology with clusters of neurons (sparsely interconnected to other clusters) to come around the cascadability problem — though research in cluster topology vs. system generality must be carried out.

In addition to the problem with limited accuracy of analogue VLSI, the issue of weight storage in such systems is a major concern: no good analogue, electronic memory is presently available.

To eliminate the need for RAM back-up in systems using simple capacitive storage, we proposed to use the "self repair" properties of neural network ensembles to do auto refresh in systems with an on-chip supervised learning algorithm. This can efficiently prolong the time-to-exhaustion of the weights. In adaptive systems this might prove sufficient; especially if the synapse strengths are a combination of read-only (behavioral) and plastic (adaptive) memories.

If the use of digital synapse memory is acceptable one can, in a similar manner, combine this with an "analogue adjustment" enabling an analogue learning algorithm to train a system with coarse discretized synapse strengths.

The *bottom line*: Though implementations of analogue neural network learning systems have begun to emerge in the literature — including the present thesis — excessive research in this field is still needed before the field is mature. Research in learning algorithms are needed both at an algorithmic and at an implementation level. Problems that need to be addressed include insensitivity to weight change offset, analogue memories and enhancement of system performance reliability. It is our believe that such research will prove fruitful for the future VLSI implemenations of supervised (and other) learning algorithms.

Bibliography

The references of this work are logically placed in five categories:

- *On-chip learning* [6, 9, 10, 14, 16, 22, 37, 39, 40, 44, 46, 50, 58, 60, 63, 67, 72, 85, 96, 97, 99, 100, 107, 109, 117, 139, 140, 141, 142, 143, 144, 145, 146, 151, 154, 160, 167, 168, 170, 174, 177, 194, 196, 210, 216, 220, 223, 237, 238, 251, 256, 269, 270].
- *Analogue neural networks* [5, 12, 13, 23, 24, 25, 26, 32, 38, 42, 51, 53, 59, 65, 66, 71, 82, 83, 84, 88, 98, 101, 110, 111, 115, 122, 123, 126, 130, 131, 132, 133, 137, 147, 149, 150, 155, 156, 157, 158, 162, 163, 169, 171, 173, 175, 176, 180, 185, 191, 192, 193, 195, 197, 206, 207, 211, 228, 246, 254].
- *Artificial neural networks* [1, 4, 15, 17, 18, 19, 20, 21, 29, 30, 31, 36, 43, 47, 48, 55, 56, 62, 69, 70, 79, 80, 86, 87, 89, 90, 91, 93, 94, 95, 103, 105, 108, 112, 113, 116, 124, 125, 127, 128, 134, 135, 152, 159, 166, 178, 179, 181, 183, 184, 186, 188, 190, 199, 200, 203, 209, 214, 215, 218, 224, 225, 226, 230, 234, 236, 239, 248, 258, 259, 261, 264, 265, 267, 268, 271, 272].
- *Integrated circuits* [3, 7, 11, 28, 33, 34, 35, 41, 45, 49, 52, 54, 57, 61, 64, 73, 75, 76, 77, 92, 102, 104, 106, 118, 120, 121, 129, 138, 161, 164, 165, 182, 187, 198, 201, 202, 204, 205, 208, 212, 213, 217, 219, 221, 222, 227, 229, 232, 233, 240, 241, 242, 243, 244, 245, 247, 252, 253, 255, 257, 263, 260, 262, 266, 274].
- *Miscellaneous references* [2, 8, 27, 68, 74, 78, 81, 114, 119, 136, 148, 153, 172, 189, 231, 235, 250, 249, 273].

A few of the references are not cited in the thesis but have been included for a more thorough referring of the field. Most probably many authors have unjustly been excluded from this list; I most sincerely apology for this. Following are the references listed in alphabetical order:

[1] "The Connectionist Mailing List", 1993-1994, restricted Internet mailing list, Connectionists-Request@cs.cmu.edu.

[2] "HP Direct", Hewlett Packard, no. 1, 1994, Series 700 Workstations.

[3] "NEAR Workshop on European Analog Research", September: 1992, post conference workshop at ESSCIRC*92, Copenhagen.

[4] "On Cytological Screening using Perceptrons", December: 1991, talk at 4th Neural Information Processing Systems Conference, Denver.

[5] Aanen Abusland and Tor S. Lande, "Local Generation and Storage of Reference Voltages in CMOS Technology," in *Proc. 11'th European Conference on Circuit Theory and Design*, pp. 281–286, 1993.

[6] P. Y. Alla, G. Dreyfus, J. D. Gascuel, A. Johannet, L. Personnaz, J. Roman and M. Weinfeld, "Silicon Integration of Learning Algorithms and Other Auto-Adaptive Properties in Digital Feedback Neural Networks," in *VLSI Design of Neural Networks*, Ulrich Ramacher and Ulrich Rückert, Eds., Norwell: Kluwer Academic Publishers, 1991, pp. 170–186.

[7] Phillip E. Allen and Douglas R. Holberg, *CMOS Analog Circuit Design*, Fort Worth: Holt, Rinehart and Winston Inc., 1987.

[8] George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, Redwood City: Benjamin/Cummings Publishing Company Inc., 1989.

[9] Joshua Alspector, Anthony Jayakumar and Stephan Luna, "Experimental Evaluation of Learning in a Neural Microsystem," in *Proc. 4'th Neural Information Processing Systems Conference*, pp. 871–878, 1992.

[10] Preben Alstrøm, "On VLSI Implementaion of Reinforcement Learning," private communication, The Niels Bohr Institute, 1994.

[11] Per Andersson, "Portable CMOS design rules for the Swedish Universities", Lund: Wallin & Dalholm Boktryckeri AB, 1990.

[12] A. J. Annema, "Hardware realisation of a neuron transfer function and its derivative," *Electronics Letters*, vol. 30, no. 7, pp. 576–577, 1994.

[13] Anne-Johan Annema, "Analysis, Modelling and Implementation of Analog Integrated Neural Networks," Ph.D. thesis, University of Twente, The Netherlands, 1994.

[14] A. J. Annema, K. Hoen and H. Wallinga, "Precision Requirements for Single-layer Feed-forward Neural Networks," in *Proc. 4'th International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, Turin, pp. 145–151, 1994.

[15] Anne-Johan Annema, Klaas Hoen and Hans Wallinga, "Learning Behaviour and Temporary Minima of Two-layer Neural Networks," *Neural Networks*, vol. 7, no. ?, pp. +18, 1994.

[16] Yutaka Arima, Misuhiro Murasaki, Tsuyoshi Yamada, Atsushi Maeda and Hirofumi Shinohara, "A Refreshable Analog VLSI Neural Network Chip with 400 Neurons and 40K Synapses," *IEEE Journal of Solid-State Circuits*, vol. SC-27, no. 12, pp. 1854–1867, 1992.

[17] Krste Asanovic and Nelson Morgan, "Experimental Determination of Precision Requirements for Back-propagation Training of Artificial Netral Networks," in *Proc. 2'nd International Conference on Microelectronics for Neural Networks*, pp. 9–15, 1991.

[18] Les E. Atlas and Yoshitake Suzuki, "Digital Systems for Artificial Neural Networks," *IEEE Circuits and Devices Magazine*, vol. 5, no. 11, pp. 20–24, 1989.

[19] Roberto Battiti and Giampietro Tecchiolli, "Learning with first, second, and no derivatives: a case study in High Energy Physics," *Neurocomputing*, vol. 6, pp. 181-206, 1994.

[20] Randall D. Beer, Hillel J. Chiel and Leon S. Sterling, "A Biological Perspective on Autonomous Agent Design," *Robotics and Autonomous Systems*, vol. 6, pp. 169–186, 1990.

[21] Kiet hA. Benedict, "Learning in the Multilayer Perceptron," *Journal of Physics A: Math. Gen.*, vol. 21, pp. 2643–2650, 1988.

[22] Ronald G. Benson and Douglas A. Kerns, "UV-Activated Conductances Allow For Multiple Time Scale Learning," *IEEE Transaction on Neural Networks*, vol. 4, no. 3, pp. 434–440, May 1993.

[23] Steven Bibyk and Mohammed Ismail, "Issues in Analog VLSI and MOS Techniques for Neural Computing," in *Analog VLSI Implementation of Neural Systems*, Carver Mead and Mohammed Ismail, Eds., Norwell: Kluwer Academic Publishers, 1989, pp. 103–133.

[24] Steven Bibyk and Mohammed Ismail, "Neural Network Building Blocks for Analog MOS VLSI," in *Analogue IC design: the current-mode approach*, C. Toumazou, F. J. Lidgey and D. G. Haigh, Eds., IEE Circuits and Systems (2) Series, London: Peter Peregrinus Ltd, 1991, pp. 597–615.

[25] Christian Björk and Sven Mattisson, "Multivalued memory in standard CMOS for weight storing in Neural Networks," in *Proc. 10'th European Conference on Circuit Theory and Design*, vol. 2, pp. 461–468, 1991.

[26] Gudmundur Bogason, "Generation of a Neuron Transfer Function and its Derivative," *Electronics Letters*, vol. 29, no. 21, pp. 1867–1869, 1993.

[27] E. J. Borowski and J. M. Borwein, *Dictionary of Mathematics*, Glasgow: Collins Reference, 1989.

[28] T. Botha, "CMOS Analogue Current-Steering Multiplier," *Electronics Letters*, vol. 28, no. 6, pp. 525–526, 1992.

[29] Søren Brunak, Jacob Engelbrecht and Steen Knudsen, "Prediction of Human mRNA Donor and Acceptor Sites from the DNA Sequence," *Journal of Molecular Biology*, vol. 220, pp. 49–65, 1991.

[30] Søren Brunak and Benny Lautrup, "Linjedeling med et neuralt netværk," *Skrifter for anvendt og matematisk lingvistik*, vol. -, pp. +20, 1989.

[31] Søren Brunak and Hans Hansen, "On Predicting Splice Sites with RTRL," private communication, Technical University of Denmark, 1991–1994.

[32] Erik Bruun, John A. Lansner and Torsten Lehmann, "Analog VLSI Architectures for Computational Neural Networks," in *Proc. 10'th NORCHIP Seminar*, pp. 59–68, 1992.

[33] Erik Bruun, "Bandwith Limitations in Current Mode and Voltage Mode Integrated Feedback Amplifiers," EI preprint., Technical University of Denmark, 1994.

[34] Erik Bruun, "Analogue Signal Processing: Collected Papers 1991–93," Electronics Institute, Technical University of Denmark, Lyngby, 1994.

[35] Erik Bruun, Gudmundur Bogason, Thomas Kaulberg, John Lansner and Peter Shah, "On Analogue VLSI," private communication, Technical University of Denmark, 1991–1994.

[36] Wray L. Buntine and Andreas S. Weigend, "Computing Second Derivatives in Feed-forward Networks: a Review," *IEEE Transactions on Neural Networks*, vol. NN-4, pp. +17, 1993.

[37] Graham Cairns and Lionel Tarassenko, "Learning with Analogue VLSI MLPs," in *Proc. 4'th International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, Turin, pp. 67–76, 1994.

[38] Yong Cao, Sven Mattisson and Christian Björk, "SeeHear System: A New Implementation," in *Proc. 18'th European Solid State Circuits Conference*, pp. 199–202, 1992.

[39] Howard Card, "Relaxation Networks: Recent Examples of Analog Circuits from the U. S. and Canada," in *Proc. 3'rd International Conference on Microelectronics for Neural Networks*, pp. 265–296, 1993.

[40] Howard Card, "Analog Circuits for Relaxation Networks," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 359–379, 1993.

[41] L. Richard Carley, "Trimming Analog Circuits Using Floating-Gate Analog MOS Memory," *IEEE Journal of Solid-State Circuits*, vol. SC-24, no. 6, pp. 1569–1575, 1989.

[42] Hernan A. Castro, Simon M. Tam and Mark A. Holler, "Implementation and Performance on an Analog Nonvolatiole Neural Network," *Analog Integrated Circuits and Signal Processing*, vol. 4, pp. 97–113, 1993.

[43] Thierry Catfolis, "A Method for Improving the Real-Time Recurrent Learning Algorithm," *Neural Networks*, vol. 6, no. 6, pp. 807–821, 1993.

[44] Daniele D. Caviglia, Maurizio Valle and Giacomo M. Bisio, "Effects of Weight Discretization on the Back Propagation Learning Method: Algorithm Design and Hardware Realization," in *Proc. IEEE International Joint Conference on Neural Networks*, pp. II-631–II-637, 1990.

[45] Tsin-Yuan Chang, Cheng-Chi Wang and Jain-Bean Hsu, "Two Schemes for Detecting CMOS Analog Faults," *IEEE Journal of Solid-State Circuits*, vol. SC-27, no. 2, pp. 229–233, 1992.

[46] Jungwook Cho, Yoon Kyung Choi and Soo-Young Lee, "Modular Analog Neuro-chip Set with On-chip Learning by Error Back-propagation and/or Hebbian Rules," in *Proc. International Conference on Artificial Neural Networks*

'94, Sorrento, vol. 2, pp. 1343–1346, 1994.

[47] Leon O. Chua and Lin Yang, "Cellular Neural Networks: Applications," *IEEE Transactions on Circits and Systems*, vol. CAS-35, no. 10, pp. 1273–1290, 1988.

[48] Leon O. Chua and Lin Yang, "Cellular Neural Networks: Theory," *IEEE Transactions on Circits and Systems*, vol. CAS-35, no. 10, pp. 1257–1272, 1988.

[49] A. L. Coban and Pe. E. Allen, "Low-voltage, Four-quadrant, analogue CMOS Multiplier," *Electronics Letters*, vol. 30, no. 13, pp. 1044-1045, 1994.

[50] M. H. Cohen and A. C. Andreou, "MOS Circuit for Nonlinear Hebbian Learning," *Electronics Letters*, vol. 28, no. 6, pp. 591–593, 1992.

[51] Dean R. Collins and P. Andrew Penz, "Considerations for Neural Network Hardware Implementations," in *Proc. IEEE International Symposium on Circuits and Systems*, pp. 834–836, 1989.

[52] Michael C. W. Coln, "Chopper Stabilization of MOS Operational Amplifiers Using Feed-Forward Techniques," *IEEE Journal of Solid-State Circuits*, vol. SC-16, no. 6, pp. 745–748, 1981.

[53] D. Del Corso, F. Gregoretti and L. M. Reyneri, "An Artificial Neural System Using Coherent Pulse Witdh and Edge Modulation," in *Proc. 3'rd International Conference on Microelectronics for Neural Networks*, pp. 105–114, 1993.

[54] P. J. Crawley and G. W. Roberts, "High-Swing MOS Current Mirror with Arbitrarily High Output Resistance," *Electronics Letters*, vol. 28, no. 4, pp. 361–363, 1992.

[55] Yann Le Cun, John S. Denker and Sara A. Solla, "Optimal Brain Damage," in *Proc. Neural Information Processing Systems Conference '89*, San Mateo, pp. 598–605, 1990.

[56] Yann Le Cun, Ido Kanter and Sara A. Solla, "Second Order Properties of Error Surfaces: Learning Time and Generalization," in *Proc. Neural Information Processing Systems Conference '90*, Denver, pp. 918–924, 1991.

[57] Zdzislaw Czarnul, "Novel MOS Resistive Circuit for Synthesis of Fully Integrated Continous-Time Filters," *IEEE Transactions on Circuits and Systems*, vol. CAS-33, no. 7, pp. 718–721, 1986.

[58] M. van Daalen, J. Zhao and J. Shawe-Taylor, "Real Time Output Derivatives for On Chip Learning using Digital Stochastic Bit Stream Neurons," *Electronics Letters*, vol. 30, no. 21, pp. 1775–1777, 1994.

[59] Casper Dietrich, "Analog VLSI — kontruktion af matrix-vektor multiplikator med digitalt lagrede vægte," M.Sc. thesis, Elektronisk Institut, Danmarks Tekniske Højskole, Lyngby, 1994.

[60] B. K. Dolenko and H. C. Card, "Neural Learning in Analogue Hardware: Effects of Component Variation from Fabrication and from Noise," *Electronics Letters*, vol. 29, no. 8, pp. 693–694, 1993.

[61] R. Domínguez-Castro, A. Rodríguez-Vázquez, F. Medeiro and J. L. Huertas, "High Resolution CMOS Current Comparators," in *Proc. 18'th European Solid State Circuits Conference*, pp. 242–245, 1992.

[62] Kenji Doya and Shuji Yoshizawa, "Adaptive Neural Oscilator Using Continuous-Time Back-Propagation Learning," *Neural Networks*, vol. 2, pp. 375–385, 1989.

[63] T. Duong, S. P. Eberhardt, M. Tran, T. Duad, and A. P. Thakoor, "Learning and Optimization with Cascaded VLSI Neural Network Building-block Chips," in *Proc. IEEE International Joint Conference on Neural Networks*, pp. I-184–I-189, June 1992.

[64] Scott T. Dupuie and Mohammed Ismail, "High Frequency CMOS Transconductors," in *Analogue IC design: the current-mode approach*, C. Toumazou, F. J. Lidgey & D. G. Haigh, Eds., IEE Circuits and Systems (2) Series, London: Peter Peregrinus Ltd., 1990, pp. 181–238.

[65] Silvio Eberhardt, Tuan Duong and Anil Thakoor, "Design of Parallel Hardware Neural Network Systems from Custom Analog VLSI 'Building Block' Chips," in *Proc. IEEE International Joint Conference on Neural Networks*, pp. II-183–II-190, 1989.

[66] Silvio Eberhardt, Alex Moonpenn and Anil Thakoor, "Considerations for Hardware Implementations of Neural Networks," in *Proc. 22nd Asilomar Conference on Signals, Systems and Computers*, pp. 649–653, 1988.

[67] Peter J. Edwards and Alan F. Murray, "Analogue Synaptic Noise — Implications and Learning Improvements," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 427–433, 1993.

[68] Lewis C. Eggebrecht, *Interfacing to the IBM Personal Computer*, 2nd ed., Indianapolis: Sams, 1990.

[69] S. E. Fahlman, "Fast-Learning Variations on Back-propagation: An Empirical Study," in *Proc. Connectionist Models Summer School '88*, Pittsburgh, D. Touretzky, G. Hinton and T Sejnowski, Eds., Morgan Kaufmann, pp. 38–51, 1989.

[70] Nabil H. Farhat, "Optoelectronic Neural Networks and Learning Machines," *IEEE Circuits and Devices Magazine*, vol. 5, no. 9, pp. 32–41, 1989.

[71] Barry Flower and Marwan Jabri, "The Implementation of Single and Dual Transistor VLSI Analogue Synapses," in *Proc. 3'rd International Conference on Microelectronics for Neural Networks*, pp. 1–10, 1993.

[72] Barry Flower and Marwan Jabri, "Summed Weight Neuron Perturbation: An $O(N)$ Improvement over Weight Perturbation," in *Proc. Neural Information Processing Systems Conference 5 '92*, San Mateo, pp. +7, 1993.

[73] Thaddeus J. Gabara, Gregory J. Cyr and Charles E. Stroud, "Metastability of CMOS Master/Slave Flip-Flops," *IEEE Transactions on Circuits and Systems, Pt. II*, vol. CAS-39, no. 10, pp. 734–740, 1993.

[74] Robert A. Gabel and Richard A. Roberts, *Signals and Linear Systems*, 3rd ed., New York: John Wiley and Sons, Inc., 1987.

[75] Umberto Gatti, Franco Maloberti and Valentino Liberali, "Full Stacked Layout of Analogue Cells," in *Proc. IEEE International Symposium on Circuits and Systems*, pp. 1123–1126, 1989.

[76] U. Gatti, F. Maloberti and G. Palmisano, "An Accurate CMOS Sample-and-Hold Circuit," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 120–122, 1992.

[77] Randall L. Geiger, Phillip E. Allen and Noel R. Strader, *VLSI Design Techniques for Analog and Digital Circuits*, Singapore: McGraw-Hill Publishing Company, 1990.

[78] Arthur Gelb, Joseph F. Kasper Jr., Raymond A. Nash Jr., Charles F. Price, Arthur A. Sutherland Jr. and the Analythic Science Corporation, *Applied Optimal Estimation*, Cambridge: MIT Press, 1974.

[79] C. L. Giles, D. Chen, C. B. Miller, H. H. Chen, G. Z. Sun and Y. C. Lee, "Grammatical Inference Using Second-Order Recurrent Neural Networks," in *Proc. IEEE International Joint Conference on Neural Networks*, pp. +8, 1991.

[80] Shelly D. D. Goggin, Karl E. Gustafson and Kristina M. Johnson, "Connectionist Nonlinear Over-Relaxation," in *Proc. IEEE International Joint Conference on Neural Networks*, pp. III-179–III-184, 1990.

[81] Malcolm S. Gordon, *Animal Physiology: Principals and adaptions*, 2nd ed., New York: Macmillan Publishing Co. Inc., 1972, pp. 369–413.

[82] Hans P. Graf and Lawrence D. Jackel, "Analog Electronic Neural Network Circuits," *IEEE Circuits and Devices Magazine*, vol. 5, no. 7, pp. 44–49, 1989.

[83] H. P. Graf, "Analog Electronic Neural Networks," in *Proc. 18'th European Solid State Circuits Conference*, pp. 57–60, 1992.

[84] Hans Peter Graf and Don Henderson, "A Reconfigurable CMOS Neural Network," in *Artificial Neural Networks*, Edgar Sánchez-Sinencio and Clifford Lau, Eds., New York: IEEE Press, 1992, pp. 260–262.

[85] David Grant, John Taylor and Paul Houselander, "Design, Implementation and Evaluation of a High-Speed Integrated Hamming Neural Classifier," *IEEE Journal of Solid-State Circuits*, vol. SC-29, no. 9, pp. 1154–1157, 1994.

[86] Sten Grillner, Peter Wallén, Lennart Brodin and Anders Lansner, "Neuronal Network Generating Locomotor Behavior in Lamprey," *Annual Reviews on Neuroscience*, vol. 14, pp. 169–199, 1991.

[87] Heng Guo and Saul B. Gelfand, "Analysis of Gradient Decent Learning Algorithms for Multilayer Feedforward Neural Networks," *IEEE Transactions on Circuits and Systems*, vol. CAS-38, no. 8, pp. 883–894, 1991.

[88] Alister Hamilton, Stephen Churcher, Peter J. Edwards, Geoffrey B. Jackson, Alan F. Murray and H. Martin Reekie, "Pulse Stream VLSI Circuits and Systems: The Epsilon Neural Network Chipset," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 395–405, 1993.

[89] Lars Kai Hansen, Christian Liisberg and Peter Salamon, "Ensemble Methods

for Handwritten Digit Recognition," in *Proc. The 1992 IEEE Workshop on Neural Networks for Signal Processing*, pp. -, 1992.

[90] Lars Kai Hansen and Peter Salamon, "Neural Network Ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-12, no. 10, pp. 993–1001, 1990.

[91] Lars Kai Hansen and Peter Salamon, "Self-repair in Neural Network Ensembles," AMSE Conference on Neural Networks, San Diego, 1991.

[92] Ole Hansen, "On VLSI Devices," private communication, Technical University of Denmark, 1991–1994.

[93] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, New York: Macmillan Collage Publishing Company, Inc., 1994.

[94] John Hertz, Anders Krogh, Benny Lautrup and Torsten Lehmann, "Nonlinear Back-propagation: Doing Back-propagation without Derivatives of the Activation Function," Neuroprose preprint, Niels Bohr Institute, Copenhagen, 1994.

[95] John Hertz, Anders Krogh and Richard G. Palmer, *Introduction to the Theory of Neural Computation*, Redwood City: Addison-Wesley Publishing Company, 1991.

[96] Marcus Höhfeld and Scott E. Fahlman, "Probabilistic Rounding in Neural Network Learning with Limited Precision," in *Proc. 2'nd International Conference on Microelectronics for Neural Networks*, pp. 1–8, 1991.

[97] Hollis, Harper and Paulos, "The Effects of Precision Constraints in a Backpropagation Learning Network," *Neural Computation*, vol. 2, no. 3, pp. 363–373, 1990.

[98] Paul W. Hollis and John J. Paulos, "Artificial Neural Networks Using MOS Analog Multipliers," *IEEE Journal of Solid-State Circuits*, vol. SC-25, no. 3, pp. 849–855, 1990.

[99] Paul W. Hollis, John J. Paulos and Christopher J. D'Costa, "An Optimized Learning Algorithm for VLSI Implementation," in *Proc. 2'nd International Conference on Microelectronics for Neural Networks*, pp. 121–126, 1991.

[100] Paul W. Hollis and John J. Paulos, "A Neural Network Learning Algorithm Tailored for VLSI Implementation," *IEEE Transactions on Neural Networks*, vol. NN-5, no. 5, pp. 784–791, 1994.

[101] Yoshihiko Horio and Shogo Nakamura, "Analog Memories for VLSI Neurocomputing," in *Artificial Neural Networks*, Edgar Sánchez-Sinencio and Clifford Lau, Eds., New York: IEEE Press, 1992, pp. 244–363.

[102] J. C. Houk, "Learning in Modular Networks," in *Proc. 7th Yale Workshop on Adaptive and Learning Systems*, pp. 80–84, 1992.

[103] Keun-Rong Hsieh and Wen-Tsuen Chen, "A Neural Network Model which Combines Unsupervised and Supervised Learning," *IEEE Transactions on Neural Networks*, vol. NN-4, no. 2, pp. 357–360, 1993.

[104] Kou-Chiang Hsieh, Paul R. Gray, Daniel Senderowicz and David G. Messer-

schmitt, “A Low-Noise Chopper-Stabilized Differential Switched-Capacitor Filtering Technique,” *IEEE Journal of Solid-State Circuits*, vol. SC-16, no. 6, pp. 708–715, 1981.

[105] John F. Hurdle, Erik L. Brunvand and Lüli Josephson, “Asynchronous VLSI Design for Neural System Implementation,” in *Proc. 3'rd International Workshop on VLSI for Neural Netwoks and Artificial Intelligence*, pp. -, 1992.

[106] Mohammed Ismail and Terri Fiez, *Analog VLSI Signal and Information Processing*, Electrical and Computer Engineering Series, New York: McGraw-Hill, 1994.

[107] Marwan Jabri and Barry Flower, “Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks,” *IEEE Transactions on Neural Networks*, vol. NN-3, no. 1, pp. 154–157, 1992.

[108] M. Jabri, S. Pickard, P. Leong, Z. Chi, B. Flower and Y. Xie, “ANN Based Classification for Heart Defibrillators,” in *Proc. Neural Information Processing Systems Conference '91*, Denver, pp. 637–644, 1992.

[109] Marwan A. Jabri, “Practical Performance and Credit Assignment Efficiency of Analog Multi-layer Perceptron Perturbation Based Training Algorithms,” System Engineering and Design Automation Laboratory, Sydney University Electrical Engineering, SEDAL tech. rep. 1-7-94, 1994.

[110] Lawrence D. Jackel, “Practical Issues for Electronic Neural-Net Hardware,” tutorial notes at the 4'th Neural Information Processing Systems Conference, 1991.

[111] Geoffrey Jackson, Alister Hamilton and Alan Murray, “Pulse Stream VLSI Neural Systems: Into Robotics,” in *Proc. IEEE International Symposium on Circuits and Systems*, London, vol. 6, pp. 375–378, 1994.

[112] Robert A. Jacobs, Michael I. Jordan and Andrew G. Barto, “Task Decomposition through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks,” *Cognitive Science*, vol. 15, pp. 219–250, 1991.

[113] Kam Jim, C. Lee Giles and Bill G. Horne, “Synaptic Noise in Dynamically-driven Recurrent Neural Networks: Convergence and Generalization,” Institute for Advanced Computer Studies, University of Maryland, UMIACS-TR-94-89 and CS-TR-3322, 1994.

[114] Tor A. Johansen and Bjarne A. Foss, “Constructing NARMAX Models using ARMAX Models,” *International Journal of Control*, vol. 58, no. 5, pp. 1125–1153, 1992.

[115] D. E. Johnson, J. S. Marsland and W. Eccleston, “Neural Network Implementation using a Single MOST per Synapse,” to appear in IEEE Transactions on Neural Networks, 1994.

[116] F. Joublin, M. Lemesle, S. Wacquant and R. Debrie, “Proposed Hardware Implementation of Massively Parallel Cortical Automation Networks,” *Electronics Letters*, vol. 28, no. 18, pp. 1711–1712, 1992.

[117] Yaron Kanshai and Yair Be'ery, “Back Propagation and Distrubuted Data

Architectures," in *Proc. 3'rd International Conference on Microelectronics for Neural Networks*, pp. 143–150, 1991.

[118] Thomas Kaulberg and Gudmundur Bogason, "An Angle Detector Based on Magnetic Sensing," in *Proc. IEEE International Symposium on Circuits and Systems*, London, vol. 5, pp. 329–332, 1994.

[119] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, 2nd ed., Englewood Cliffs: Prentice Hall, 1988.

[120] Douglas A. Kerns, "Experiments in Very Large-Scale Analog Computation," Ph.D. thesis, California Institute of Technology, Pasadena, Caifornia, 1993.

[121] Donald A. Kerth, Navdeep S. Sooch and Eric A. Swanson, "A 12-bit 1MHz Two-Step Flash ADC," *IEEE Journal of Solid-State Circuits*, vol. SC-24, no. 2, pp. 250–255, 1989.

[122] Edwin van Keulen, Sel Colak, Heini Withagen and Hans Hegt, "Neural Networ Hardware Performance Criteria," private communication, Eindhoven University of Technology, 1993.

[123] Nabil I. Khachab and Mohammed Ismail, "A Nonlinear CMOS Analog Cell for VLSI Signal and Information Processing," *IEEE Journal of Solid-State Circuits*, vol. SC-26, no. 11, pp. 1689–1699, 1991.

[124] Anders Krogh and John A. Hertz, "A Simple Weight Decay Can Improve Generalization," in *Proc. Neural Information Processing Systems Conference '91*, Denver, pp. 950–957, 1992.

[125] Anders Krogh, Lars Kai Hansen and Jan Larsen, "On Neural Networks," private communication, Technical University of Denmark, 1991–1994.

[126] Francis J. Kub, Keith K. Moon, Ingham A. Mack and Francis M. Long, "Programmable Analog Vector-Matrix Multipliers," *IEEE Journal of Solid-State Circuits*, vol. SC-25, no. 1, pp. 207–214, 1990.

[127] Hon Keung Kwan, "Systolic Architectures for Hopfield Network, BAM and Multi-Layer Feed-Forward Networks," in *Proc. IEEE International Symposium on Circuits and Systems*, pp. 790–793, 1989.

[128] H. K. Kwan and C. Z. Tang, "Designing Multilayer Feedforward Neural Networks Using Simplified Sigmoid Activation Functions and One-Powers-of-Two Weights," *Electronics Letters*, vol. 28, no. 25, pp. 2342–2345, 1992.

[129] Kadaba R. Lakshmikumar, Robert A. Hadaway and Miles A. Copeland, "Charactiration and Modeling of Mismatch in MOS Transistors for Precision Analog Design," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 6, pp. 1057–1066, 1986.

[130] John A. Lansner and Torsten Lehmann, "A Neuron- and a Synapse Chip for Artificial Neural Networks," in *Proc. 18'th European Solid State Circuits Conference*, pp. 213–216, 1992.

[131] John A. Lansner and Torsten Lehmann, "An Analog CMOS Chip Set for Neural Networks with Arbitrary Topologies," *IEEE Transaction on Neural Networks*, vol. 4, no. 3, pp. 441–444, May 1993.

[132] John A. Lansner, "An Experimental Hardware Neural Network using a Cascadable, Analog Chip Set," to appear in International Journal of Electronics, Technical University of Denmark, 1994.

[133] John A. Lansner, "Analogue VLSI Implementation of Artificial Neural Networks," Ph.D. thesis, Electronics Institute, Technical University of Denmark, Lyngby, 1994.

[134] Alan Lapedes and Robert Farber, "How Neural Nets Work," in *Proc. Neural Information Processing Systems Conference '87*, D. Z. Anderson, Eds., New York: American Institute of Physics, pp. 442–456, 1988.

[135] Jan Larsen, "Design of Neural Network Filters," Ph.D. thesis, Electronics Institute, Technical University of Denmark, Lyngby, 1993.

[136] Michael LeBlanc and Robert Tibshirani, "Combining Estimates in Regression and Classification," preprint, University of Toronto, 1993.

[137] Bang W. Lee, Bing J. Sheu and Han Yang, "Analog Floating-Gate Synapses for General-Purpose VLSI Neural Computation," *IEEE Transactions on Circuits and Systems*, vol. CAS-38, no. 6, pp. 654–658, 1991.

[138] Hae-Seung Lee, David A. Hodges and Paul R. Gray, "A Self-Calibrating 15 Bit CMOS A/D Converter," *IEEE Journal of Solid-State Circuits*, vol. SC-19, no. 6, pp. 813–819, 1984.

[139] Torsten Lehmann, "A Hardware Implementation of the Real-Time Recurrent Learning Algorithm," in *Proc. 10'th European Conference on Circuit Theory and Design*, vol. 2, pp. 431–440, 1991.

[140] Torsten Lehmann, "Neurale Netværk i VLSI Teknologi," M.Sc. thesis, Elektronisk Institut, Danmarks Tekniske Højskole, Lyngby, 1991.

[141] Torsten Lehmann, "A Cascadable Chip Set for ANN's with On-chip Back-propagation," in *Proc. 3'rd International Conference on Microelectronics for Neural Networks*, pp. 149–158, 1993.

[142] Torsten Lehmann, "A Hardware Efficient Cascadable Chip Set for ANN's with On-chip Back-propagation," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 351–358, 1993.

[143] Torsten Lehmann and Erik Bruun, "Analogue VLSI Implementation of Back-propagation Learning in Artificial Neural Networks," in *Proc. 11'th European Conference on Circuit Theory and Design*, pp. 491–496, 1993.

[144] Torsten Lehmann, "Implementation Issues for Back-propagation Learning in Analog VLSI Neural Networks," in preparation, Technical University of Denmark, 1995.

[145] Torsten Lehmann and Lars Kai Hansen, "Analogue VLSI Neural Network Ensemble Issues," in preparation, Technical University of Denmark, 1994.

[146] Torsten Lehmann, Erik Bruun and Casper Dietrich, "Analogue/Digital Hybrid VLSI Synapses for Recall- and Learning Mode Neural Networks," in *Proc. 12th NORCHIP seminar*, Gothenburg, Sweeden, pp. 31–38, 1994.

[147] Torsten Lehmann, Erik Bruun and Casper Dietrich, "Mixed Analogue/Digital

Matrix/Vector Multiplier for Neural Network Synapses," in preparation, Technical University of Denmark, 1995.

[148] F. Thomson Leighton, *Introduction to Parallel Algorithms and Architechrures: Arrays, Trees, Hypercubes*, San Mateo: Morgan Kaufmann Publishers, 1992.

[149] Phillip H. W. Leong and Marwan A. Jabri, "Kakadu — A Low Power Analogue Neural Network," in *Proc. 3'rd International Conference on Microelectronics for Neural Networks*, pp. 207–216, 1993.

[150] Phillip H. W. Leong and Marwan A. Jabri, "Kakadu — A Low Power Analogue Neural Network Classifier," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 381–394, 1993.

[151] Bernabé Linares-Barranco, Edgar Sánchez-Sinencio, Angel Rodríguez-Vázquez and José L. Huertas, "A CMOS Analog Adaptive BAM with On-Chip Learning and Weight Refreshing," *IEEE Transaction on Neural Networks*, vol. 4, no. 3, pp. 445–455, May 1993.

[152] Richard P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, vol. -, no. 4, pp. 4–22, 1987.

[153] Ronald J. MacGregor, *Neural and Brain Modeling*, San Diego: Academic Press Inc., 1987.

[154] Damien Macq, Michel Verleysen, Paul Jespers and Jean-Didier Legat, "Analog Implementation of a Kohonen Map with On-chip Laearning," *IEEE Transaction on Neural Networks*, vol. 4, no. 3, pp. 456–461, May 1993.

[155] Kurosh Madani, Ghislain de Tremiolles and Ion Berechet, "Temperature Effects Modelling and Compensation Analysis in Analogue Implementation of Stochastic Artificial Neural Networks," in *Proc. 4'th International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, Turin, pp. 170–177, 1994.

[156] Jim Mann, Richard Lippmann, Bob Berger and Jack Raffel, "A Self-Organizing Neural Net Chip," in *Proc. IEEE Custom Integrated Circuits Conference*, pp. 10.3.1–10.3.5, 1988.

[157] P. Masa, K. Hoen and H. Wallinga, "20 Million Patterns Per Second Analog CMOS Neural Network Pattern Classifier," in *Proc. 11'th European Conference on Circuit Theory and Design*, pp. 497–502, 1993.

[158] L. W. Massengill, "A Dynamic CMOS Multiplier for Analog Neural Network Cells," in *Proc. IEEE Custom Integrated Circuits Conference*, pp. 26.4.1–26.4.4, 1990.

[159] Ofer Matan, Christopher J. C. Burges, Yann Le Cun and John S. Denker, "Multi-digit Recognition Using a Space Displacement Neural Network," in *Proc. 4'th Neural Information Processing Systems Conference*, pp. 488–495, 1992.

[160] Takao Matsumoto and Masafumi Koga, "A High-Speed Learning Method for Analog Neural Networks," in *Proc. IEEE International Joint Conference on Neural Networks*, pp. II-71–II-76, 1990.

[161] M. J. McNutt, S. LeMarquis and J. L. Dunkley, "Systematic Capacitance

Matching Errors and Corrective Layout Procedures," *IEEE Journal of Solid-State Circuits*, vol. SC-29, no. 5, pp. 611–616, 1994.

[162] Carver Mead, *Analog VLSI & Neural Systems*, Reading: Addison-Wesley Publishing Company, 1989.

[163] Carver Mead and Mohammed Ismail, *Analog VLSI Implementation of Neural Systems*, Norwell: Kluwer Academic Publishers, 1989.

[164] Christopher Michael and Mohammed Ismail, "Statistical Modeling of Device Mismatch for Analog MOS Integrated Circuits," *IEEE Journal of Solid-State Circuits*, vol. SC-27, no. 2, pp. 154–166, 1992.

[165] Jean-Yves Michel, "High-Performance Analog Cells in Mixed-Signal VLSI: Problems and Practical Solutions," *Analog Integrated Circuits and Signal Processing*, vol. 1, pp. 171–182, 1991.

[166] Coe F. Miles and C. David Rogers, "The Microcircuit Associative Memory: A Bilogically Motivated Memory Architecture," *IEEE Transactions on Neural Networks*, vol. NN-5, no. 3, pp. 424–435, 1994.

[167] Antonio J. Montalvo, Ronald S. Gyurcsik and John J. Paulos, "Building Blocks for a Temperature-Compensated Analog VLSI Neural Network with On-Chip Learning," in *Proc. IEEE International Symposium on Circuits and Systems*, London, vol. 6, pp. 329–332, 1994.

[168] Antonio J. Montalvo, Paul W. Hollis and John J. Paulos, "On-Chip Learning in the Analog Domain with Limited Precision Circuits," in *Proc. International Symposium on Circuits and Systems*, pp. I-196–I-201, 1992.

[169] Keith K. Moon, Francis J. Kub and Ingham A. Mack, "Random Address 32X32 Programmable Analog Vector-Matrix Multiplier for Artificial Neural Netwoks," in *Proc. IEEE Custom Integrated Circuits Conference*, pp. 26.7.1–26.7.4, 1990.

[170] Takashi Morie and Yoshihito Amemiya, "An All-Analog Expandable Neural Network LSI with On-Chip Backpropagation Learning," *IEEE Journal of Solid-State Circuits*, vol. SC-29, no. 9, pp. 1086–1093, 1994.

[171] Alessandro Mortara and Eric A. Vittoz, "A Communication Architecture Tailored for Analog VLSI Artificial Neural Networks: Intrinsic Performance and Limititions," *IEEE Transactions on Neural Networks*, vol. NN-5, no. 3, pp. 459–466, 1994.

[172] V. B. Mountcastle, "An Organizing Principle for Cerebral Function: The Unit Module and the Distributed System," in *The Mindful Brain*, G. M. Edelman and V. B. Mountcastle, Eds., Cambidge: MIT Press, 1978, pp. 7–50.

[173] Paul Mueller, Jan van der Spiegel, David Blackman, Timothy Chiu, Thomas Clare, Christopher Donham, Tzu Pu Hsieh and Marc Loinaz, "Design and Fabrication of VLSI Components for a General Purpose Analog Neural Computer," in *Analog VLSI Implementation of Neural Systems*, Carver Mead and Mohammed Ismail, Eds., Norwell: Kluwer Academic Publishers, 1989, pp. 135–169.

[174] Alan F. Murray, "Multilayer Perceptron Learning Optimized for On-Chip

Implementation: A Noise-Robust System," *Neural Computation*, vol. 4, no. 3, pp. 366–381, 1992.

[175] Alan F. Murray, Dante Del Corso and Lionel Tarassenko, "Pulse-Stream VLSI Neural Networks Mixing Analog and Digital Techniques," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 193–204, March 1991.

[176] A. F. Murray, L. Tarassenko, H. M. Reekie, A. Hamilton, M. Brownlow, S. Churcher and D. J. Baxter, "Pulsed Silicon Neural Networks — Following the Biological Leader," in *VLSI Design of Neural Networks*, Ulrich Ramacher and Ulrich Rückert, Eds., Norwell: Kluwer Academic Publishers, 1991, pp. 104–123.

[177] Alan F. Murray and Peter J. Edwards, "Analogue Synaptic Noise — A Hardware Nuisance, or an Aid to Learning?," in *Proc. 3'rd International Conference on Microelectronics for Neural Networks*, pp. 121–129, 1993.

[178] Alan F. Murray and Peter J. Edwards, "Enhanced MLP Performance and Fault Tolerance Resulting from Synaptic Weight Noise During Training," *IEEE Transactions on Neural Networks*, vol. NN-5, no. 5, pp. 792–802, 1994.

[179] O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz and G. Dryfus, "Training Recurrent Neural Networks: Why and How? An Illustration in Dynamical Process Modeling," *IEEE Transactions on Neural Networks*, vol. NN-5, no. 2, pp. 178–184, 1994.

[180] Charles F. Neugebauer and Amnon Yariv, "A Parallel Analog CCD/CMOS Signal Processor," in *Proc. Neural Information Processing Systems Comference '91*, pp. 748–755, 1992.

[181] Chalapathy Neti, Michael H. Schneider and Eric D. Young, "Maximally Fault Toerant Neural Networks," *IEEE Transactions on Neural Networks*, vol. NN-3, no. 1, pp. 14–23, 1992.

[182] Paul O'Leary, "Practical Aspects of Mixed Analogue and Digital Design," in *Analogue Digital ASIC's — Circuit Techniques, Design Tools and Applications*, R. S. Soin, F. Maloberti and J. Franca, Eds., IEE Circuits and Systems (3) Series, London: Peter Peregrinus Ltd., 1991, pp. 213–238.

[183] O. Osowski, "New Approach to Selection of Initial Values of Weights in Neural Function Approximation," *Electronics Letters*, vol. 29, no. 3, pp. 313–315, 1993.

[184] G. Palm, K. Goser, U. Rückert and A. Ultsch, "Knowledge Processing in Neural Architecture," in *Proc. 3'rd International Workshop on VLSI for Neural Netwoks and Artificial Intelligence*, pp. -, 1992.

[185] Joshua C. Park, Christopher Abel and Mohamed Ismail, "Design of a Silicon Cochlea Using MOS Switched-current Techniques," in *Proc. 11'th European Conference on Circuit Theory and Design*, pp. 269–274, 1993.

[186] Morten With Pedersen and Lars Kai Hansen, "Recurrent Networks: Second Order Properties and Pruning," EI preprint, 1994.

[187] Marcel J. M. Pelgrom, Aad C. J. Duinmaijer and Anton P. G. Welbers, "Matching Properties of MOS Transistors," *IEEE Journal of Solid-State*

Circuits, vol. SC-24, no. 5, pp. 1433–1440, 1989.

[188] D. Plaut, S. Nowlan and G Hinton, "Experiments on Learning by Backpropagation," Department of Computer Science, Carnegie Mellon University, Pittsburgh, tech. Rep. CMU-CS-86-126, 1986.

[189] William H. Press, Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, *Numerical Recipes in C*, Cambridge: Cambridge University Press, 1988.

[190] Ning Qian and Terrence J. Sejnowski, "Predicting the Secondary Structure of Globular Proteins Using Neural Network Models," *Journal of Molecular Biology*, vol. -, no. 202, pp. 865–884, 1988.

[191] Jack I. Raffel, "Electronic Implementation of Neuromorphic Systems," in *Proc. IEEE Custom Integrated Circuits Conference*, pp. 10.1.1–10.1.7, 1988.

[192] Ulrich Ramacher, "Guide Lines to VLSI Design of Neural Nets," in *VLSI Design of Neural Networks*, Ulrich Ramacher and Ulrich Rückert, Eds., Norwell: Kluwer Academic Publishers, 1991, pp. 1–17.

[193] Ulrich Ramacher and Ulrich Rückert, *VLSI Design of Neural Networks*, Norwell: Kluwer Academic Publishers, 1991.

[194] Ulrich Ramacher and Peter Schildberg, "Recent Developments in Neurodynamics and their Impact on the Design of Neuro-chips," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 309–316, 1993.

[195] A. A. Reeder, I. P. Thomas, C. Smith, J. Wittgreffe, D. Godfrey, J. Hajto, A. Owen, A. J. Snell, A. F. Murray, M. Rose and P. G. LeComber, "Application of Analogue Amorphous Silicon Memory Devices to Resistive Synapses for Neural Networks," in *Proc. 2'nd International Conference on Microelectronics for Neural Networks*, Munich, pp. 253–260, 1991.

[196] Leonardo M. Reyneri and Enrica Filippi, "An Analysis on the Performance of Silicon Implementations of Backpropagation Algorithms for Artificial Neural Networks," *IEEE Transactions on Computers*, vol. C-40, no. 12, pp. 1380–1389, 1991.

[197] L. M. Reyneri, M. Chiaberge and D. del Corso, "Using Coherent Pulse Width and Edge Modulations in Artificial Neural Systems," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 407–418, 1993.

[198] Jacques Robert and Philippe Deval, "A Second-Order High-Resolotion Incremental A/D Converter with Offset and Charge Injection Compensation," *IEEE Journal of Solid-State Circuits*, vol. SC-23, no. 3, pp. 736–741, 1988.

[199] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Internal Repressentations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, D. E. Rumelhart, J. L. McClelland and the PDP Reserch Group, Eds., Cambridge: MIT Press, 1986, chap. 8.

[200] D. E. Rumelhart, J. L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge: MIT Press, 1986.

[201] Eduard Säckinger and Linda Fornera, "On the Placement of Critical Devives in Analog Integrated Circuits," *IEEE Journal of Solid-State Circuits*, vol. SC-37, no. 8, pp. 1052–1057, 1990.

[202] Eduard Säckinger and Walter Guggenbühl, "A High-Swing, High-Impedance MOS Cascode Circuit," *IEEE Journal of Solid-State Circuits*, vol. SC-25, no. 1, pp. 289–298, 1990.

[203] Shigeo Sakaue, Toshiyuki Koda, Hiroshi Yamamoto, Susumu Maruno and Yasuharu Shimeki, "Reduction of Required Precision Bits for Back-Propagation Applied to Pattern Recognition," *IEEE Transactions on Neural Networks*, vol. NN-4, no. 2, pp. 270–275, 1993.

[204] S. Sakurai and M. Ismail, "High Frequency Wide Range CMOS Analogue Multiplier," *Electronics Letters*, vol. 28, no. 24, pp. 2228–2229, 1992.

[205] C. Andre T. Salama, David G. Nairn and Henry W. Singor, "Current Mode A/D and D/A Converters," in *Analogue IC design: the current-mode approach*, C. Toumazou, F. J. Lidgey & D. G. Haigh, Eds., IEE Circuits and Systems (2) Series, London: Peter Peregrinus Ltd., 1990, pp. 491–514.

[206] Edgar Sánchez-Sinencio and Clifford Lau, *Artificial Neural Networks*, New York: IEEE Press, 1992.

[207] Srinagesh Satyanarayana, Yannis P. Tsividis and Hans Peter Graf, "A Reconfigurable VLSI Neural Network," *IEEE Journal of Solid-State Circuits*, vol. SC-27, no. 1, pp. 67–81, 1992.

[208] Navin Saxena and James J. Clark, "A Four-quadrant CMOS Analog Multiplier for Analog Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. SC-29, no. 6, pp. 746–749, 1994.

[209] Jürgen Schmidhuber, "An $O(n^3)$ Learning Algorithm for Fully Recurrent Networks," Institut für Informatik, Technische Universität München, München, 1991.

[210] Christian Schneider and Howard Card, "Analog CMOS Synaptic Learning Circuits Adapted from Invertebrate Biology," *IEEE Transactions on Circuits and Systems*, vol. CAS-38, no. 12, pp. 1430–1438, 1991.

[211] Jesper S. Schultz, "Neurale Netværk i VLSI Teknologi — med sparse digitale inputs," M.Sc. thesis, Elektronisk Institut, Danmarks Tekniske Højskole, Lyngby, 1993.

[212] Evert Seevinck, "Analog Interface Circuits for VLSI," in *Analogue IC design: the current-mode approach*, C. Toumazou, F. J. Lidgey and D. G. Haigh, Eds., IEE Circuits and Systems (2) Series, London: Peter Peregrinus Ltd., 1990, pp. 451–489.

[213] Charles L. Seitz, "System Timing," in *Introduction to VLSI Systems*, Carver Mead and Lynn Conway, Eds., Reading: Addison-Wesley Publishing Company, 1980, pp. 218–262.

[214] Terrance J. Sejnowski and Charles R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, vol. 1, pp. 145–168, 1987.

[215] Nikola B. Šerbedžija and Gerd Kock, "Fault-Tolerant Neuro-Computing," in *Proc. International Conference on Artificial Neural Networks '94*, Sorrento, vol. 2, pp. 1404–1407, 1994.

[216] T. Serrano, B. Linares-Barranco, and J. L. Huertas, "A CMOS VLSI Analog Current-Mode High-Speed ART1 Chip," in *Proc. IEEE International Conference on Neural Networks*, Orlando, vol. 3, pp. 1912-1916, 1994.

[217] Peter Shah, "A Short Term Analogue Memory," in *Proc. 18'th European Solid State Circuits Conference*, pp. 127–130, September 1992.

[218] Samir Shah and Francesco Palmieri, "MEKA — A Fast, Local Algorithm for Training Feedforward Neural Networks," in *Proc. IEEE International Joint Conference on Neural Networks*, pp. III-41–III-46, 1990.

[219] Je-Hurn Shieh, Mahesh Patil and Bing J. Sheu, "Measurement and Analysis of Charge Injection in MOS Analog Swithces," *IEEE Journal of Solid-State Circuits*, vol. SC-22, no. 2, pp. 277–281, 1987.

[220] Takeshi Shima, Tomohisa Kimura, Yukio Kamatani, Tetsuro Itakura, Yasuhiko Fujita and Tetsuya Iida, "Neuro Chips with On-chip Back-propagation and/or Hebbian Learning," *IEEE Journal of Solid-State Circuits*, vol. SC-27, no. 12, pp. 1868–1876, 1992.

[221] Masakazu Shoji, "Elimination of Process-Dependent Clock Skew in CMOS VLSI," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 5, pp. 875–880, 1986.

[222] Svante Signell and Kåre Mossberg, "Offset-Compensation of Two-Phase Switched-Capacitor Filters," *IEEE Journal of Solid-State Circuits*, vol. SC-36, no. 1, pp. 31–41, 1989.

[223] Roy Ludvig Sigvartsen, Yngvar Berg and Tor Sverre Lande, "An Analog Neural Network with On-chip Back-propagation Learning," in *Proc. 12th NORCHIP seminar*, Gothenburg, Sweeden, pp. 169–176, 1994.

[224] Patrick K. Simpson, "Foundations of Neural Networks," in *Artificial Neural Networks*, Edgar Sánchez-Sinencio and Clifford Lau, Eds., New York: IEEE Press, 1992, pp. 3–24.

[225] Anthony W. Smith and David Zipser, "Learning Sequential Structure with the Real-time Recurrent Learning Algorithm," *International Journal of Neural Systems*, vol. 1, no. 2, pp. 125–131, 1989.

[226] S. A. Solla, E. Levin and M. Fleisher, "Accelerated Learning in Layered Neural Networks," *Complex Systems*, vol. 2, pp. 625–639, 1988.

[227] Jens Sparsø, Christian D. Nielsen, Lars S. Nielsen and Jørgen Staunstrup, "Design of Self-timed Multipliers: A Comparison," in *Proc. IFIP TC10/WG-10.5 Working Conference on Asynchronous Design Methodologies*, Manchester, pp. 165–179, 1993.

[228] Jan Van der Spiegel, Paul Mueller, David Blackmann Peter Chance, Cristopher Donham, Ralph Etienne-Cummungs and Peter Kinget, "An Analogue Neural Computer with Modular Architecture for Real-Time Dynamic Computations," *IEEE Journal of Solid-State Circuits*, vol. SC-27, no. 1, pp. 82–92,

1992.

[229] Balsha R. Stanisic, Nishath K. Verghese, Rob A. Rutenbar, L. Richard Carley and David J. Allstot, "Addressing Substrate Coupling in Mixed-Mode IC's: Simulation and Power Distribution Synthesis," *IEEE Journal of Solid-State Circuits*, vol. SC-29, no. 3, pp. 226–238, 1994.

[230] W. Scott Stornetta and B. A. Huberman, "An Improved Three-layer Back Propagation Algorithm," in *Proc. IEEE International Conference on Neural Networks*, vol. 2, pp. 637–643, 1987.

[231] Lubert Stryer, *Biochemistry*, 3rd ed., New York: W. H. Freeman and Company, 1988.

[232] Sun, Moll, Berger and Alders, "Break-down Mechanism in Short Channel MOS Transistors," in *Proc. IEEE Technical Digest, International Electron Device Meeting*, Wasington DC, pp. 478, 1978.

[233] Ivan E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

[234] C. Svarer, L. K. Hansen and J. Larsen, "On Design and Evaluation of Tapped-Delay Neural Network Architectures," in *Proc. IEEE International Conference on Neural Networks*, vol. 1, pp. 46–51, 1993.

[235] S. M. Sze, *Semiconductor Devices, Physics and Technology*, New York: John Wiley & Sons, 1986.

[236] Janos Sztipanovits, "Dynamic Backpropagation Algorithm for Neural Network Controlled Resonator-Bank Architecture," *IEEE Transactions on Circuits and Systems II*, vol. CAS-39, no. 2, pp. 99–108, 1992.

[237] Lionel Tarassenko and Jon Tombs, "On-chip Learning with Analogue VLSI Neural Networks," in *Proc. 3'rd International Conference on Microelectronics for Neural Networks*, pp. 163–174, 1993.

[238] Lionel Tarassenko, Jon Tombs and Graham Cairns, "On-chip Learning with Analogue VLSI Neural Networks," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 419–426, 1993.

[239] Hans Henrik Thodberg, "The Neural Information Processing System used for pig carcase grading in Danish Slaughterhouses," Danish Meat Research Institute preprint, No. 989 E, Roskilde, 1991.

[240] Axel Thomsen and Martin A. Brooke, "A Floating Gate CMOS Signal Conditioning Circuit for Nonlinearity Correction," *Analog Integrated Circuits and Signal Processing*, vol. 4, pp. 21–29, 1993.

[241] Cris Toumazou and John Lidgey, "Universal Current-Mode Analogue Amplifiers," in *Analogue IC Design: The Current-Mode Approach*, C. Toumazou, F. J. Lidgey and D. G. Haigh, Eds., IEE Circuits and Systems (2) Series, London: Peter Peregrinus Ltd., 1990, pp. 127–180.

[242] C. Toumazou, F. J. Lidgey and D. G. Haigh, *Analogue IC Design: The Current-Mode Approach*, IEE Circuits and Systems (2) Series, London: Peter Peregrinus Ltd., 1990.

[243] C. Toumazou, F. J. Lidgey and C. A. Makris, "Extending Voltage-mode Op Amps to Current-mode Performance," *IEE Proceedings, Pt. G*, vol. 137, no. 2, pp. 116–130, 1990.

[244] C. Toumazou, A. Payne and J. Lidgey, "Current-Feedback Versus Voltage-Feedback Amplifiers: History, Insight and Relationships," in *Proc. IEEE International Symposium on Circuits and Systems*, pp. 1046–1049, 1993.

[245] Yannis Tsividis, Mihai Banu and John Khoury, "Continuous-Time MOSFET-C Filters in VLSI," *IEEE Transactions on Circuits and Systems*, vol. CAS-33, no. 2, pp. 125–140, 1986.

[246] Y. Tsividis and S. Satyanarayana, "Analogue Circuits for Variable-Synapse Electronic Neural Networks," *Electronics Letters*, vol. 23, no. 24, pp. 1313–1314, 1987.

[247] Yannis P. Tsividis, "R&D in Analog Circuits: Possibilities and Needed Support," in *Proc. 18'th European Solid State Circuits Conference*, pp. 1–15, 1992.

[248] A. C. Tsoi, C. N. W. Tan and S. Lawrence, "Financial Time Series Forcasting: Application of Artificial Neural Network Techniques," preprint, Department of Electrical Engeneering and Computer Engineering, University of Queensland St. Lucia, Australia, 1994.

[249] Paul W. Tuinenga, *SPICE: A Guide to Circuit Simulation & Analysis Using PSpice*, Englewood Cliffs: Prentice Hall, 1988.

[250] Dogan A. Tugal and Osman Tugal, *Data Transmission: Analysis, Design, Applications*, New York: McGraw-Hill, 1982.

[251] Maurizio Valle, Daniele D. Caviglia and Giacomo M. Bisio, "An Experimental Analog VLSI Neural Chip with On-Chip Back-Propagation Learning," in *Proc. 18'th European Solid-State Circuits Conference*, pp. 203–206, 1992.

[252] S. R. Vemuru, "Layout Comparison of MOSFETs with Large W/L Ratios," *Electronics Letters*, vol. 28, no. 25, pp. 2327–2329, 1992.

[253] Eric A. Vittoz, "MOS Transistors Operated in the Lateral Bipolar Mode and Their Application in CMOS Technology," *IEEE Journal of Solid-State Circuits*, vol. SC-18, no. 2, pp. 273–279, 1983.

[254] E. Vittoz, H. Oguey, M. A. Maher, O. Nys, E. Dukstra and M. Chevroulet, "Analog Storage of Adjustable Synaptic Weights," in *VLSI Design of Neural Networks*, Ulrich Ramacher and Ulrich Rückert, Eds., Norwell: Kluwer Academic Publishers, 1991, pp. 48–63.

[255] Fong-Jim Wang and Gabor C. Temes, "A Fast Offset-Free Sample-and-Hold Circuit," *IEEE Journal of Solid-State Circuits*, vol. SC-23, no. 5, pp. 1270–1272, 1988.

[256] Yiwen Wang, "A Modular Analog CMOS LSI for Feedforward Neural Networks with On-Chip BEP Learning," in *Proc. 1993 IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 2744–2747, 1993.

[257] Zhenhua Wang, "A CMOS Four-Quadrant Analog Multiplier with Single-Ended Voltage Output and Improved Temprature Performance," *IEEE Jour-*

nal of Solid-state Circuits, vol. SC-26, no. 9, pp. 1293–1301, 1991.

[258] Timothy L. H. Watkin, Albrecht Rau and Michael Biehl, "The Statistical Mechanics of Learning a Rule," *Moder Physics Review*, vol. -, pp. +45, 1992.

[259] R. B. Webb, "Optoelectronic Implementation of Neural Networks," *International Journal of Neural Systems*, vol. 4, no. 4, pp. 435–444, 1993.

[260] George Wegmann, Eric A. Vittoz and Fouad Rahali, "Charge Injection in Analog MOS Switches," *IEEE Journal of Solid-State Circuits*, vol. SC-22, no. 6, pp. 1091–1097, 1987.

[261] S. Andreas Weigend, Bernardo A. Huberman and David E. Rumelhart, "Predicting the Future: a Connectionist Approach," *International Journal of Neural Systems*, vol. 1, no. 3, pp. 193–209, 1990.

[262] Neil H. E. Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Reading: Addison-Wesley Publishing Company, 1985.

[263] Chin-Long Wey, Benlu Jiang and Gregory M. Wierzba, "Build-In Self-Test (BIST) Design of Large-Scale Analog Circuit Networks," in *Proc. IEEE International Symposium on Circuits and Systems*, pp. 1123–1126, 1989.

[264] Halbert White, "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, vol. 1, pp. 425–464, 1989.

[265] Bernard Widrow and Michael A. Lehr, "30 Years of Adaptive Neural Networks: Perceptrons, Madaline, and Backpropagation," *IEEE Proceedings*, vol. 78, no. 9, pp. 1415–1442, 1990.

[266] Remco J. Wiegerink, Evert Seevinck and Wim de Jager, "Offset Cancelling Circuit," *IEEE Journal of Solid-State Circuits*, vol. SC-24, no. 3, pp. 651–658, 1989.

[267] Ronald J. Williams and David Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[268] Ronald J. Williams and David Zipser, "Experimental Analysis of the Realtime Recurrent Learning Algorithm," *Connection Science*, vol. 1, no. 1, pp. 87–111, 1989.

[269] Heini Withagen, "Implementing Backpropagation with Analog Hardware?," in *Proc. International Conference on Neural Networks*, Orlando, pp. -, 1994.

[270] Robin Woodburn, H. Martin Reekie and Alan F. Murray, "Pulse-stream Circuits for On-chip Learning in Analogue VLSI Neural Networks," in *Proc. IEEE International Symposium on Circuits and Systems*, London, vol. 4, pp. 103–106, 1994.

[271] Niels Holger Wulff, "Learning Dynamics with Recurrent Networks," M.Sc. thesis, NORDITA, Nordisk Institut for Teoretisk Fysik, København, 1992.

[272] Yun Xie and Marwan Jabri, "Analysis of the Effects of Quantization in Multilayer Neural Networks using a Statistical Model," *IEEE Transactions on Neural Networks*, vol. NN-3, no. 2, pp. 334–338, 1992.

[273] Moritoshi Yasunaga, Noboru Masuda, Masayoshi Yagyu, Mitsuo Asai, Minoru

Yamada and Akira Masaki, “Design, Fabrication and Evaluation of a 5-inch Wafer Scale Neural Network LSI Composed of 576 Digital Neurons,” in *Artificial Neural Networks*, Edgar Sánchez-Sinencio and Clifford Lau, Eds., New York: IEEE Press, 1992, pp. 235–243.

[274] Chong-Gun Yu and Randall L. Geiger, “An Automatic Offset Compensation Scheme with Ping-pong Control for CMOS Operational Amplifiers,” *IEEE Journal of Solid-State Circuits*, vol. SC-29, no. 5, pp. 601–610, 1994.

Index

ELECTRONICS
INSTITUTE

Hardware Learning in Analogue VLSI Neural Networks

A thesis by
Torsten Lehmann

Appendices

and

Enclosures

September 1994

Appendix A

Definitions

In this appendix, *definitions* are given of concepts that are not otherwise well-defined.

Accuracy

The *accuracy* of a quantity ξ is defined as its maximal deviation from ideality:

$$D_{\mathrm{A}\,\xi} \overset{\mathrm{def}}{=} \frac{\max_{\underline{x}} |\xi(\underline{x}) - \xi_{\mathrm{ideal}}(\underline{x})|}{\xi_{\max} - \xi_{\min}} \,. \qquad (25^{\mathrm{A}})$$

The normalization can also be with respect to the ideal range $\xi_{\mathrm{ideal,max}} - \xi_{\mathrm{ideal,min}}$.

Bit relative measure

The *bit relative measure* of a quantity $\Delta\xi$ is defined as the number of *least significant bits* LSB_B of $\Delta\xi$ given an B bit discretization of the quantity ξ to which $\Delta\xi$ is related:

$$\left(\frac{\Delta\xi}{\xi_{\max} - \xi_{\min}}\right) \Big/ 1\,\mathrm{LSB}_B \equiv \frac{\Delta\xi}{\xi_{\max} - \xi_{\min}} \times 2^B \,. \qquad (26^{\mathrm{A}})$$

(Or $1\,\mathrm{LSB}_B \overset{\mathrm{def}}{=} 2^{-B}$.) Sometimes we use the LSB_B measure, somewhat imprecisely, in a non unit-less way; in this case $1\,\mathrm{LSB}_B \sim 2^{-B}(\xi_{\max} - \xi_{\min})$. We call this the *bit absolute measure.*

Connections per second

The standard speed measure for neural networks is the *Connections Per Second* measure, *CPS*, which counts the number of synaptic connections (multiply-adds, that is) the network does per second. (Thus the work involved in computing the activation function is ignored.) This measure has been questioned by several individuals, and other measures have been proposed. The *Effective Connection Primitives Per Second* (*Keulen et al.* [122]), for instance, seems a good candidate for the future standard.

Connection updates per second

The standard speed measure for teaching neural networks is the *Connection Updates Per Second* measure, *CUPS*, which counts the number of updates on the synaptic connection strengths the learning algorithm does per second.

Non-linearity

The *non-linearity* (or *integral non-linearity*) of a quantity ξ is defined as its maximal deviation from ideality when the offset error has been canceled:

$$D_\xi \overset{\text{def}}{=} \frac{\max_{\underline{x}} |\xi(\underline{x}) - \xi_{\text{ofs}} - \xi_{\text{ideal}}(\underline{x})|}{\xi_{\max} - \xi_{\min}} \,. \tag{27$^{\text{A}}$}$$

The normalization can also be with respect to the ideal range $\xi_{\text{ideal,max}} - \xi_{\text{ideal,min}}$.

Offset error

The *offset error* of a quantity ξ is defined as its deviation from ideality at ideal zero value:

$$\xi_{\text{ofs}} \overset{\text{def}}{=} \xi(\underline{x}(\xi_{\text{ideal}} = 0)) \,. \tag{28$^{\text{A}}$}$$

It is presumed that non-linearities and offset errors that are related to $\underline{x}$ has been canceled to make this definition well-defined. (If this is not possible, the offset error must be defined for a specific $\underline{x}$; preferably a non-biased one like $\underline{x} = \underline{0}$.)

Resolution

The *resolution* of a quantity ξ is defined as the smallest change of this quantity that can be distinguished at some appropriate output $f(\xi)$:

$$\xi_{\text{res}} \overset{\text{def}}{=} \min_{|f(\xi+\Delta\xi)-f(\xi)|>\varepsilon} \Delta\xi \quad , \tag{29$^{\text{A}}$}$$

where ε is smallest distinguishable output change.

Appendix B

Artificial neural networks

This appendix briefly describes the concepts of neural networks. We present the most popular models and display typical application areas and motivations for using artificial neural networks. We briefly touch the concepts of learning, with emphasis on gradient descent. A performance evaluation measure is also given.

An *artificial neural network* or *ANN* is a type of computer, with a topology inspired by the human brain: it consists of a large number of simple calculating units, or *neurons*, which are interconnected in massive parallelism. In a typical biological neuron, each connection, or *synapse*, has an associated *connection strength*, and the neuron integrates the thus weighted outputs from other neurons over time. If this integral reach a certain threshold, the output of the neuron is pulsed high: the neuron *fires*. The neuron firing rate will be in the range zero to the inverse of the pulse time. Very simplified, the result is this: the *firing rate* of a neuron is a non-linear function of a weighted sum of its inputs firing rates (*Gordon* [81], *Rumelhart et al.* [200], *MacGregor* [153]).

B.1 The ANN model

In the standard model of an artificial neural network, it is this firing rate, or *neuron activation*, relation that is modeled (*Hertz et al.* [95], *Rumelhart et al.* [200], *Haykin* [93]): A neuron k calculates as its output y_k an (often nonlinear) function g_k of the weighted sum of its inputs z_j:

$$y_k = g_k(s_k), \qquad \text{where} \quad s_k = \sum_j w_{kj} z_j - \Theta_k \,. \tag{30^{\mathrm{B}}}$$

Here, s_k is called the *net input*, Θ_k is called the *neuron threshold* value (also, $-\Theta_k$ is called the *neuron bias*), $g_k(.)$ is called the *transfer function*, *squashing function* or *activation function*, and w_{kj} is the connection strength, or *weight*.† Θ_k is often neglected as it can be modeled as the connection strength from an input with the constant value of -1.

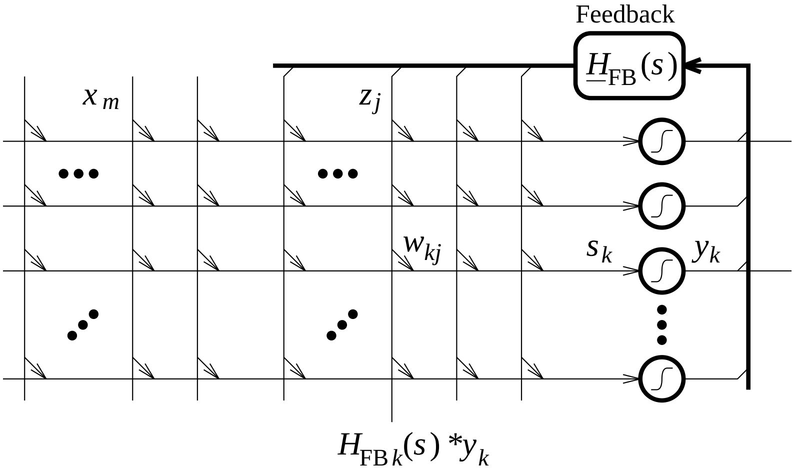


Figure 74$^{\mathrm{B}}$: General neural network model. *The feedback can be either a continuous time or a discrete time one. The arrows represent synaptic connections.*

Interconnecting these artificial neurons gives the artificial neural network. A general model of such a fully interconnected network can be seen in figure 74$^{\mathrm{B}}$. A neuron input can be either an output from another neuron or an input x_m to the network. Letting the M inputs have indices $m_\xi \in I$ and the N neurons have indices $k_\xi \in U$, we have:

$$\underline{x} = \begin{bmatrix} x_{m_1} \\ x_{m_2} \\ \vdots \\ x_{m_M} \end{bmatrix}, \quad \underline{y} = \begin{bmatrix} y_{k_1} \\ y_{k_2} \\ \vdots \\ y_{k_N} \end{bmatrix}, \quad \underline{z} = \begin{bmatrix} z_{j_1} \\ z_{j_2} \\ \vdots \\ z_{j_{M+N}} \end{bmatrix},$$

† This notation is based on a paper by *Williams and Zipser* [268], though somewhat modified to be consistent with notation used in *Hertz et al.* [95].

where

$$z_j = \begin{cases} x_j, & \text{for } j \in I \\ y_j, & \text{for } j \in U \end{cases} . \tag{31^B}$$

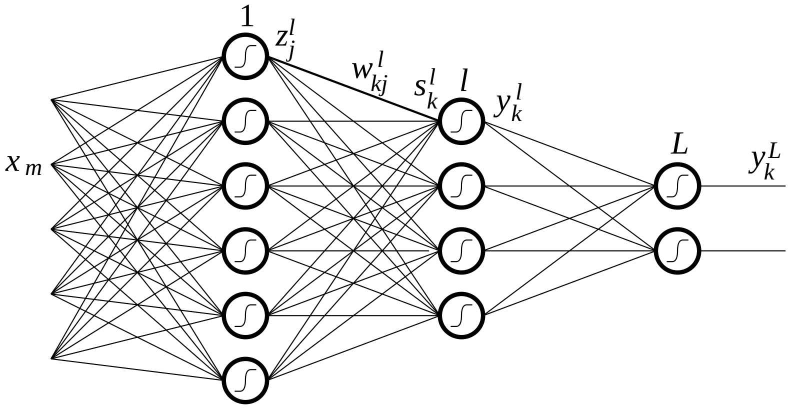


Figure 75$^\mathrm{B}$: Layered feed-forward neural network. *Also called a perceptron. This is a special, very popular, version of the general neural network suitable for a large range of classification/regression/etc. tasks.*

Often, a network is constructed as a layered feed-forward network, or a *perceptron* (also *multi-layer perceptron, MLP*); see figure 75$^\mathrm{B}$. In this case, the synapses and neurons usually bear a layer index l in addition to the ones above. Thus:

$$y_k^l = g_k^l(s_k^l), \qquad s_k^l = \sum_j w_{kj}^l z_j^l \quad , \tag{32^B}$$

where

$$z_j^l = \begin{cases} x_j, & \text{for } l = 1 \\ y_j^{l-1}, & \text{for } 1 < l \leq L \end{cases} , \tag{33^B}$$

and L is the number of layers.

In this thesis we shall be concerned with both types of networks.

B.2 Applications and motivations

Inspired by the human brain, artificial neural networks would be expected to be good at solving problems that the human brain solves efficiently. Indeed, this is so:

- *Example described problems* (as opposed to problems with an algorithmic solution) is where ANNs have the advantage over traditional methods. Problems that typically fall in this category are:
- *Associative memories*, that are the "Bohr atoms" of neural networks; closely related to
- *Classification* and
- *Regression*.

All applications where one typically has large set of data describing the problem but no (obvious) algorithm for the solution. *Recognition of handwritten characters* (eg. *Matan et al.* [159]) is a good example. Perhaps less obvious, the example described problems are also found in areas as:

- *Time series analysis*,
- *Control systems* and
- *Data compression*.

Today, ANNs are applied to a wide range of applications; often performing an order of magnitude better than previous solutions. One can mention the prediction of splice sites in human pre-mRNA (*Brunak et al.* [29]), pig carcase grading in Danish slaughterhouses (*Thodberg* [239]) and cytological screening (*NIPS* [4]).

In addition to the superior performance in certain application areas, neural networks offer several nice properties that further motivate their use:

- *Fault tolerant*. The distributed data processing of ANNs makes it very easy to include the necessary redundancy to implement a fault (error, noise, etc.) tolerant system.
- *Parallel*. The ANN equations can be totally parallelized (which is also true for many associated *learning algorithms*) implying that ANNs can be *very* fast.
- *Regular*. Most ANNs are composed of few different elements that are interconnected in a regular way. This regularity makes a hardware (eg. VLSI) implementation cheap.
- *Adaptive*. Programmed by way of examples, ANNs are easily adapted to new working conditions. This is a very powerful property, hardly challenged by any conventional method.
- *Asynchronous*. Most neural networks can (or do) function in an asynchronous way (including the human brain). This is advantageous when implementing electrical circuits, because problems with *spikes on supply currents* and *worst case timing design* are eliminated.

Notice that most of the above properties are in favour of hardware implementations of ANNs.

After the reviving of artificial neural network research in the early 1980es, ANNs got the reputation of being a "magic" tool that would give impressive results when applied to anything. This is obviously not true, and ANNs were labeled frivolous in certain circles. Today, this label is unjustified: neural network theory is advancing every day and is well founded; ANN limitations are known (see *Hertz et al.* [95], *Sánchez-Sinencio and Lau* [206], *Haykin* [93]).

To cite John Denker: "neural networks are the second best way of doing just about anything." The best way always being that of applying an algorithm — if such can be found. As the literature shows, this is often not possible; especially if an adaptive solution is sought.

B.3 Teaching ANNs

The process of determining the free parameters of an ANN such that it solves a given task is called *learning*; or *teaching* of the ANN. Learning algorithms are usually classified as one of the following (*Hertz et al.* [95]):

- *Supervised learning* using a *teacher*, eg. *back-propagation* or *real-time recurrent learning*. These algorithms are used when target values can be defined for the ANN outputs.
- *Supervised learning* using *critic*, typically derived from an algorithm using a teacher. These algorithms are used when it is possible only to tell if the ANN output is good or not. The algorithms are less efficient than the ones using a teacher.
- *Unsupervised learning*, eg. a *Hebb rule*. These algorithms are used when nothing is known about the desired ANN output; the network is supposed to find structures in the input data. All the learning algorithms rely on the ability to find structure in the input data, thus the unsupervised learning algorithms can also be used to aid one of the other types of algorithms.

Many applications are using supervised learning algorithms with a teacher; such learning algorithms are of primary interest in this text.

B.3.1 Gradient descent algorithms

Supervised learning algorithms are very often based on *gradient descent*: Some kind of *error measure* is defined for the network, and the object of the algorithm is to minimize a *cost function* defined on this error measure. This is done by successively finding the gradient of the cost function with respect to the free parameters of the system and changing the parameters a fraction in the opposite direction (corresponding to the steepest downhill climb in the cost function landscape (*Borowski and Borwein* [27])). More precisely (see *Lehmann* [139], *Williams and Zipser* [268] and *Hertz et al.* [95]):

For each network output $y_k(t)$ at time t, we define the *output error*:

$$\varepsilon_k(t) = \begin{cases} d_k(t) - y_k(t), & \text{for } k \in T(t) \\ 0, & \text{for } k \in U \setminus T(t) \end{cases} , \qquad (34^{\mathrm{B}})$$

where $T(t)$ is the set of outputs that has a desired value, or *target value*, $d_k(\mathrm{t})$ at time t. (Combined with the corresponding inputs, these are the *training data*; defined on $t_1 < t \leq t_2$ which is called an *epoch* (also, $T_{\mathrm{epc}} = t_2 - t_1$ is the *epoch length*). The time is often completely arbitrary but it is convenient to use.) The *total cost function* is the *instant cost function* accumulated over time:

$$\mathcal{J}_{\mathrm{tot}} = \sum_{t_1 < t \leq t_2} \mathcal{J}(t) \overset{\text{usually}}{=} \sum_{t_1 < t \leq t_2} \sum_{k \in U} \mathcal{J}_k(t)\,,$$

or using continuous time:

$$\mathcal{J}_{\mathrm{tot}} = \int_{t=t_1}^{t_2} \mathcal{J}(t)\, dt\,.$$

For the instant cost function, a popular choice is the *quadratic cost function*:

$$\mathcal{J}_{\mathrm{Q}}(t) \overset{\mathrm{def}}{=} \tfrac{1}{2} \sum_{k \in U} \varepsilon_k^2(t) = \tfrac{1}{2} \sum_{k \in T(t)} \left(d_k(t) - y_k(t)\right)^2\,. \qquad (35^{\mathrm{B}})$$

Letting $\underline{w}_{\star\star}$ denote the free parameters of the network, we must change these according to $\Delta\underline{w}_{\star\star} = -\eta \nabla_w \mathcal{J}_{\mathrm{tot}}$, where η is a small positive number called the *learning rate*. Expressed in coordinates this is:

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{J}_{\mathrm{tot}}}{\partial w_{ij}}\,. \qquad (36^{\mathrm{B}})$$

After this change, the total cost function is calculated once again etc., until equilibrium is reached. This is hopefully the minimum of the cost function (actually, what is reached is most likely a local minimum and this is one of the major objects of the ongoing neural network theory research).

Rather than changing the weights after each epoch (*learning by epoch* or *batch learning*), the weights are often changed continuously (*learning by example* or *online learning*) according to $\Delta\underline{w}_{\star\star}(t) = -\eta \nabla_w \mathcal{J}(t)$ or

$$\Delta w_{ij}(t) = -\eta \frac{\partial \mathcal{J}(t)}{\partial w_{ij}(t)}\,. \qquad (37^{\mathrm{B}})$$

If η is small, this will, apart from a constant factor, sum up to (36^{B}) approximately. This resembles the Gauss-Seidel method of solving linear equations numerically (*Press et al.* [189], *Goggin et al.* [80]) in the sense that iterative changes in the unknown vector $\underline{w}_{\star\star}$ are applied as quickly as possible.

When using a gradient descent algorithm, the neuron squashing function must be differentiable as

$$\frac{\partial \mathcal{J}_k}{\partial w_{ij}} = \frac{\partial \mathcal{J}_k}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_{ij}} = \frac{\partial \mathcal{J}_k}{\partial y_k} \cdot \frac{\partial y_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{ij}}\,. \qquad (38^{\mathrm{B}})$$

Often used is a function as the hyperbolic tangent: $g_k(s_k) \equiv \tanh(\beta_\mathrm{t} s_k)$, or the *sigmoid function*: $g_k(s_k) \equiv 1/(1+e^{-2\beta_\mathrm{t} s_k})$, where $\beta_\mathrm{t} \overset{\text{often}}{=} 1$ is the neuron *squashing function steepness* at $s_k = 0$.

Using the quadratic cost function above imposes a problem: When an output y_k is close to ±1, $\partial y_k/\partial s_k$ and thus $\partial \mathcal{J}/\partial w_{ij}$ will be close to zero, regardless of the value of d_k.

To come around this problem, one can use a cost function that diverges when d_k and y_k approaches different extreme values; for instance the *entropic cost function* which measures the relative entropy of d_k and y_k (*Hertz et al.* [95]):

$$\mathcal{J}_\mathrm{E}(t) \overset{\text{def}}{=} \sum_{k\in T(t)} \left[\tfrac{1}{2}(1+d_k(t))\ln\frac{1+d_k(t)}{1+y_k(t)} + \tfrac{1}{2}(1-d_k(t))\ln\frac{1-d_k(t)}{1-y_k(t)}\right] . \qquad (39^\mathrm{B})$$

Alternatively, one can use a *Fahlman perturbation* which substitutes $\partial y_k/\partial s_k + \gamma_\mathrm{F}$ for $\partial y_k/\partial s_k$ in (38^B), where γ_F is a small positive number which we also call the *derivative perturbation*.

Gradient descent type learning algorithms exist for both recurrent and feed-forward networks and many perturbations to the true gradient descent have been proposed to improve the algorithms (in terms of learning speed, generalization ability, etc.). Many other types of learning algorithms also exist but we are primarily interested in the gradient descent types in this work. See also chapter 3.

B.4 Performance evaluation

When applying an artificial neural network (using teacher supervised learning) to a problem, this is usually defined as set of input/output examples. To evaluate the performance of the network, the data set is split in a *training set* and a *test set*. Using the time-notation above for instance:

$$\begin{array}{rll} \text{trainig set} & \{x_m(t), d_k(t)\}, & t_1 < t \le t_2 \\ \text{test set} & \{x_m(t), d_k(t)\}, & t_2 < t \le t_3 \end{array} .$$

To avoid fitting the noise in the input data, it is of paramount importance that the system is over determined (compare to curve fitting (*Press et al.* [189])), ie. that there are more input/output examples than degrees of freedom in the system — as a rule of thumb (*Hertz et al.* [95], *Krogh et al.* [125]) 2–5 times more.

A good error measure based on the average relative variance (*Weigend et al.* [261]) can be found in *Svarer et al.* [234], the *normalized average relative variance*, *NARV* (for the continuous time version, replace the summations by integrations):

$$E_{\mathrm{NARV}}[t_\alpha, t_\beta] \overset{\mathrm{def}}{=} \frac{1}{\mathrm{Var}_t\{d_k(t)\}(t_\beta - t_\alpha)} \sum_{t_\alpha < t \le t_\beta} (d_k(t) - y_k(t))^2 \quad , \qquad (40^{\mathrm{B}})$$

where the variance of $d_k(t)$ is taken over the complete data set. For the data set above, this is equivalent to (when using dimension-less time):

$$E_{\mathrm{NARV}}[t_\alpha, t_\beta] = \frac{\dfrac{1}{t_\beta - t_\alpha} \displaystyle\sum_{t_\alpha < t \le t_\beta} (d_k(t) - y_k(t))^2}{\dfrac{1}{t_3 - t_1} \displaystyle\sum_{t_1 < t \le t_2} \left(d_k(t) - \underset{t_1 < \tau \le t_3}{\mathrm{mean}} \{d_k(\tau)\} \right)^2} .$$

A normalized average relative variance of 1 corresponds to the output being identical to the mean of the target values.

Monitoring the average relative variance as the learning progresses, one will typically see the training error asymptotically decreasing and the test error (always larger than the training error) reaching a minimum where the network starts to fit the noise in the input data (see section 2.6, *Qian and Sejnowski* [190], *Watkin et al.* [258]). If the test error is close to the training error we say that the network has a good *generalization ability* (more precisely: the generalization ability is the probability that the network gives the correct answer to an arbitrary input (*Hertz et al.* [95])).

Appendix C

Integrated circuit issues

In this appendix, various issues of *integrated circuits* (also, somewhat inaccurately, termed *VLSI* (*very large scale integration*) circuits) are displayed. Most will probably be known to the reader but will serve to define various symbols used in the thesis. We display the standard models for MOS- and bipolar transistors and the issue of accuracy in analogue computing hardware is touched — in relation to both component sizing and layout.

C.1 MOS transistors

In this thesis we use the Shichman-Hodges model for the *MOSFET*s (Metal Oxide Semiconductor Field Effect Transistors; also *MOST*) in strong inversion (see *Geiger et al.* [77]). The model is suited only for hand calculations — for more accurate models refer to the literature (especially for the subthreshold-saturation region). The model:

The *drain current* i_D (cf. figure 76^C) for a N-channel MOST is given by:

$$i_\mathrm{D} = \begin{cases} 0\,, & \text{cut-off, } v_\mathrm{GS} - V_\mathrm{T} \le 0 \\ \frac{1}{2}\beta(v_\mathrm{GS} - V_\mathrm{T})^2(1 + \lambda v_\mathrm{DS})\,, & \text{saturation, } 0 < v_\mathrm{GS} - V_\mathrm{T} \le v_\mathrm{DS} \\ \beta(v_\mathrm{GS} - V_\mathrm{T} - \frac{1}{2}v_\mathrm{DS})v_\mathrm{DS}(1 + \lambda v_\mathrm{DS})\,, & \text{triode, } v_\mathrm{DS} < v_\mathrm{GS} - V_\mathrm{T} \end{cases}\,, \qquad (41^\mathrm{C})$$

where β is the *transconductance parameter*, V_T is the *threshold voltage*, λ is the *channel length modulation parameter*, W is the *channel width*, and L is the *channel length* (cf. figure 77^C). In reality the transistor is never completely cut-off; near the

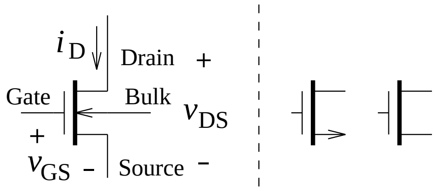


Figure 76$^{\rm C}$: N-channel MOS transistor symbols. *Mostly the bulk terminal is connected to the supply voltage and we omit it in the schematic as shown to the right. The P-channel MOST symbol has the arrows pointing in the opposite directions.*

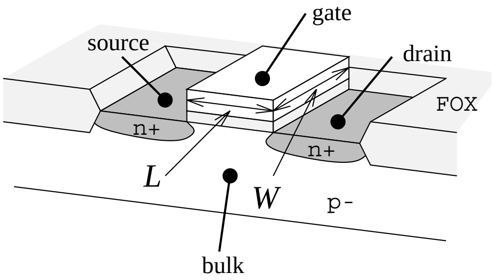


Figure 77$^{\rm C}$: N-channel MOS transistor. *Schematic drawing of physical substrate MOST. The primary design parameters, width W and length L, are shown. Notice the component simplicity.*

threshold voltage the device enters the subthreshold region where the drain current is approximately given by

$$i_{\rm D} = \tfrac{W}{L} I_{\rm D0,st} e^{(v_{\rm GS}-V_{\rm T})/nV_{\rm t}} , \quad \text{subthreshold, } v_{\rm GS} - V_{\rm T} \le nV_{\rm t} \tag{42$^{\rm C}$}$$

where $V_{\rm t} = kT/q$ is the *thermal voltage*†, n is the *subthreshold slope*, and $I_{\rm D0,st}$ is a process related parameter which is dependent on $v_{\rm DS}$ among other quantities. The parameters in the two strong inversion equations (saturation and triode) are:

$$\begin{aligned} \beta &= K'\frac{W}{L} = \mu C_{\rm ox}\frac{W}{L} \\ V_{\rm T} &= V_{\rm T0} + \gamma(\sqrt{2|\phi_{\rm F}| - v_{\rm BS}} - \sqrt{2|\phi_{\rm F}|}) \ , \\ \lambda &\approx \frac{k_\lambda}{L} \end{aligned} \tag{43$^{\rm C}$}$$

where K' is the *process transconductance parameter*, $C_{\rm ox}$ is the *gate oxide capacitance* per unit area, μ is the *surface mobility*, $V_{\rm T0}$ is the *zero bias threshold voltage*, γ is the *bulk threshold parameter*, $2|\phi_{\rm F}|$ is the *strong inversion surface potential*, and we call k_λ the *channel length modulation constant.*

The *transconductance*, $g_{\rm m}$, and the *output conductance*, $g_{\rm ds}$, in saturation is often conveniently expressed by the *quiescent drain current*, $I_{\rm D}$:

$$\begin{aligned} g_{\rm m} &= \frac{\partial i_{\rm D}}{\partial v_{\rm GS}} \approx \sqrt{2I_{\rm D}\beta} \\ g_{\rm ds} &= \frac{\partial i_{\rm D}}{\partial v_{\rm DS}} \approx \lambda I_{\rm D} \end{aligned} \ .$$

† Where k is the *Boltzman constant*, T is the *absolute temperature* and q is the *elementary charge.*

The P-channel MOST equations are defined in a similar way.

In some MOS processes a phenomenon called *snap-back* occurs in short channel devices (*Sun et al.* [232], *Hansen* [92]): For high drain-source voltages the strong electrical field near the drain junction will cause a device break-down and inject a current into the bulk of the device. This current turns on the drain-bulk-source parasitic bipolar transistor (cf. next section) and causes the drain-source current to increase enormously, see figure 78^{C}. The phenomenon wears the device but is not destructive (as latchup is typically). The critical drain-source voltage at which snap-back occurs increase with increased channel length.

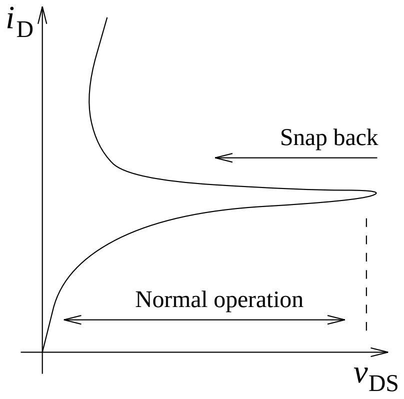


Figure 78^{C}: Short channel snap-back. *Snap-back occurs if the parasitic drain-bulk-source bipolar (NPN for an N-channel MOST) transistor is accidently turned on.*

Three different *MOS transistor symbols* are shown in figure 76^{C}. In this work we *usually* use the simple middle one with the identifiable source terminal; We assume bulk is connected to the supply voltage. For transistors with no obvious source terminal (as switches) and for digitally operated transistors we use the right symbol. The left symbol is used mainly in circuits where the bulk terminal connection is of particular importance.

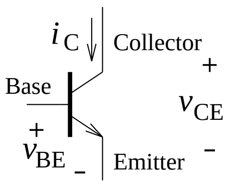


Figure 79$^{\mathrm{C}}$: NPN bipolar transistor symbol. *The PNP BJT symbol has the arrow pointing in the opposite direction.*

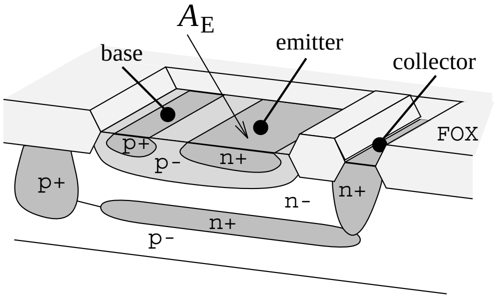


Figure 80$^{\mathrm{C}}$: NPN bipolar transistor. *Schematic drawing of simple, vertical, physical NPN BJT. The primary design parameter, the emitter area A_{E}, is shown. Even for this simple BJT, the minimum layout area is larger than that of a MOST.*

C.2 Bipolar transistors

In this thesis we use the (simplified) Ebers-Moll model for the *BJT*s (Bipolar Junction Transistors) (see *Geiger et al.* [77]):

The *collector current* i_{C} (cf. figure 79$^{\mathrm{C}}$) for a NPN BJT in forward saturation mode is given by:

$$i_{\mathrm{C}} = -\alpha_{\mathrm{F}} i_{\mathrm{E}} = \alpha_{\mathrm{F}} A_{\mathrm{E}} J_{\mathrm{S}} (e^{v_{\mathrm{BE}}/V_{\mathrm{t}}} - 1)(1 + v_{\mathrm{CE}}/V_{\mathrm{AF}}) \,, \qquad (44^{\mathrm{C}})$$

where α_{F} is the *forward emitter-collector current gain*, A_{E} is the *emitter area*, J_{S} is the *transport saturation current density*, $V_{\mathrm{t}} = kT/q$ is the *thermal voltage*, and V_{AF} is the *forward Early voltage*. The PNP BJT equations are defined in a similar way.

Usually, bipolar transistors are not meant to be available in MOS processes. However, parasitic devices are always present and in a typical well CMOS process one type of the transistors can be operated in the *lateral bipolar mode* (*LBM MOSFET*) which gives access to a bipolar device (*Vittoz* [253]). This is illustrated in figure 82$^{\mathrm{C}}$ for an N-well process. The base-emitter (or "bulk-source") diode is biased in the forward direction, which turns on the bipolar device. The BJT has two collectors: one connected to the substrate (conducting a waste current, $i_{\mathrm{S}} = \alpha_{\mathrm{FS}} i_{\mathrm{E}}$) and one connected to the "drain" of the MOS device (conducting the desired collector current, $i_{\mathrm{C}} = \alpha_{\mathrm{FC}} i_{\mathrm{E}}$). To get a reasonably high efficiency of this device, it is important that (i) the area under the emitter diffusion is as small as possible compared to the side-wall emitter diffusion area towards the collector diffusion and (ii) the emitter-collector base width is as small as possible compared to the emitter-substrate base width. Thus proper layout of the LBM MOSFET is a minimum size emitter junction encircled by a minimum length MOS gate; cf. figure 85$^{\mathrm{C}}$. For good bipolar operation, the MOSFET gate area must be biased in strong accumulation to turn off the MOST operation. A LBM MOST device symbol is shown in figure 81$^{\mathrm{C}}$.

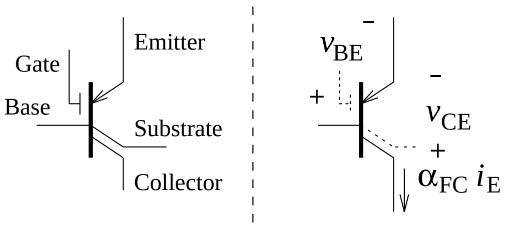


Figure 81^{C}: Lateral bipolar mode MOSFET symbol. *Connecting the gate to V_{DD} we can replace the LBM MOST by a BJT with $\alpha_{FC} \approx 1/2$ (right).*

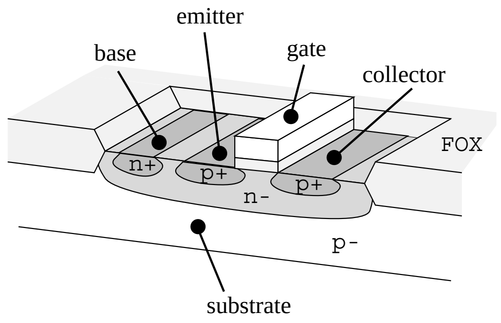


Figure 82^{C}: Lateral bipolar mode MOSFET. *Schematic drawing of physical N-well (P-channel) LBM MOSFET. The effective emitter area is the emitter junction side-wall area towards the collector junction.*

As a current is deliberately injected into the substrate, care should be taken to efficiently guard the LBM MOST device to reduce the risk of latch-up (cf. figure 85^{C}).

C.3 Analogue computing accuracy

The *computing accuracy* (especially in relation to *offset*) of analogue signal processing elements are often limited by *mismatch* of the analogue components. To enlighten the influence of mismatch, we shall do a case study of subtraction using simple MOS current mirrors.

The outputs from analogue synapses are often in the form of differential currents. The subtractions of these can be performed per synapse (figure 83^{C}) or per row of connected synapses (figure 84^{C}), the latter giving poorer accuracy unless a transistor area equal to that of the former is applied:

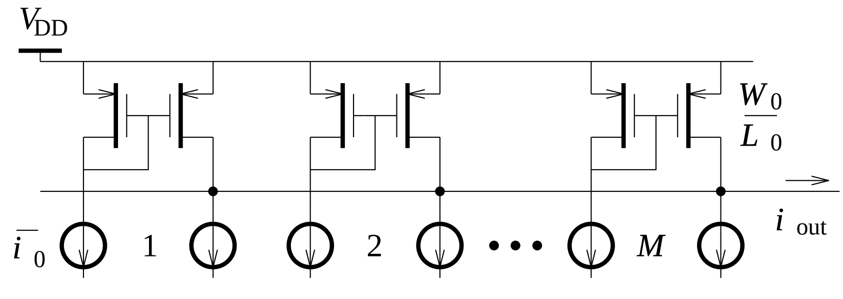


Figure 83^{C}: Current subtraction by synapse. *For simplicity, all M pairs of current sources lead the same current. All drain voltages are imagined large and identical for optimal matching.*

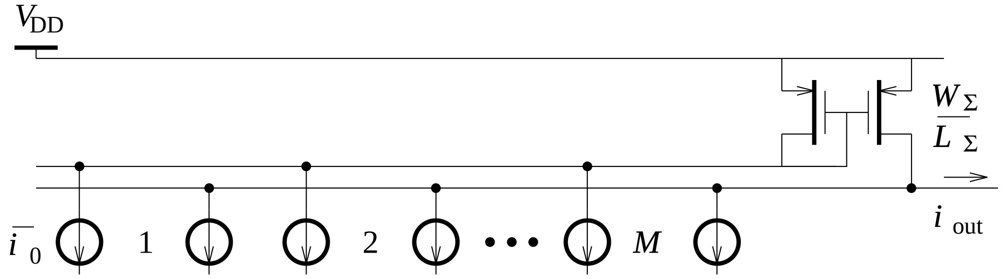


Figure 84$^{\mathrm{C}}$: Current subtraction by row. *Using a single current mirror, this have to be M times as wide to be able to sink the increased current. This very wide current mirror would be implemented as many small in parallel.*

Now, given a *transistor parameter* P, the variance of this (with respect to some "identical" reference device) can be modeled as (cf. *Pelgrom et al.* [187], *Lakshmikumar et al.* [129], *Michael and Ismail* [164], see also *Ismail and Fiez* [106]):

$$\sigma_P^2 \approx A_P^2 G_P + S_P^2 D_P^2 \,,$$

where A_P and S_P are process dependent constants, G_P is a function of the device geometry and D_P is a function of the device layout. For many MOS transistor parameters $G_P = 1/WL$†. D_P would be a (probably highly non-linear) function of *device distance*, D, device orientation, device context, wafer center distance and other layout specific quantities. Usually we set $D_P = D$ for simplicity; this assumes that careful, symmetric layout is used (see below). Assuming a simple quadratic law for a saturated MOSFET, the relative *drain current variance* is (first order approximation):

$$\frac{\sigma_{i_{\mathrm{D}}}^2}{\overline{i_{\mathrm{D}}}^2} \approx \frac{\sigma_W^2}{\overline{W}^2} + \frac{\sigma_L^2}{\overline{L}^2} + \frac{\sigma_{K'}^2}{\overline{K'}^2} + 4\frac{\sigma_{V_{\mathrm{T}}}^2}{(v_{\mathrm{GS}} - \overline{V_{\mathrm{T}}})^2} \,,$$

where $\overline{i_{\mathrm{D}}}$, $\overline{V_{\mathrm{T}}}$, $\overline{K'}$ $\overline{W}$ and $\overline{L}$ are the average drain current, threshold voltage, process transconductance parameter, channel width and channel length respectively, and the σ_ξ^2s are parameter variances. Qualitatively this equates (assuming $G_P = 1/WL$ and $D_P = D$):

$$\frac{\sigma_{i_{\mathrm{D}}}^2}{\overline{i_{\mathrm{D}}}^2} \propto \left(1 + \frac{\xi_1}{(v_{\mathrm{GS}} - V_{\mathrm{T}})^2}\right)\left(\frac{\xi_2}{WL} + D^2\right) \,, \qquad (45^{\mathrm{C}})$$

where the ξ_ζs are constants.

Returning to our example, assume a *current mismatch* $\sqrt{2}\cdot\sigma_{i_0}$ when mirroring a current $\overline{i_0}$ using a current mirror with transistor dimensions W_0 and L_0 and current standard deviation σ_{i_0}. The sum of M such currents (figure 83$^{\mathrm{C}}$) yields a mismatch in the accumulated current, i_{out}, of $\sigma_{\Sigma i_0} = \sqrt{2M}\cdot\sigma_{i_0}$, for independent error sources.

† This holds quite accurately for K' and V_{T}. For the transistor width and length, however, it would be reasonable to assume $G_W = 1$ and $G_L = 1/W$.

Were we to use a single current mirror (figure 84^C), the W/L ratio would have to be scaled to accommodate for the increased current: $W_\Sigma/L_\Sigma = MW_0/L_0$ (for an unchanged set of terminal voltages which is assumed chosen optimally for matching). Ignoring the device distance for the moment, (45^C) reduces to $\sigma_{i_\mathrm{D}}^2 \propto \overline{i_\mathrm{D}}^2/WL$. Thus for an unchanged accuracy, $\sigma_{i_\Sigma} = \sqrt{2M}\cdot\sigma_{i_0}$, we would need $W_\Sigma L_\Sigma = MW_0L_0$ — or an unchanged total transistor area (and $W_\Sigma = MW_0$, $L_\Sigma = L_0$). Ignoring the device distance is not a good approximation. However, for reasonably large devices, efficient layout techniques (inter-digited common centroid, etc.) can be employed in which case the result holds.

Because of the tremendous inter-wafer process variations, the accuracy of a single MOS transistor is not usually interesting. Rather the matching of two or more devices are important (as above). In this connection, the physical device placement — the layout — is very important; global process variations and process gradients must be taken into consideration. Especially if the devices to be matched occupy a very large area as these would be exposed to larger absolute variations than devices concentrated on a small area. In our example, this means that differencing by synapse possibly gives better accuracy (for a given area) than differencing by row as the first solution require only matching locally while the other require matching of all the transistors in the large current mirror — though process gradients having a linear influence on the drain current can be canceled out.

C.4 Integrated circuit layout

The physical layout of integrated circuit components strongly influence the matching properties. Problems that need to be considered during the layout include (see *Sze* [235], *O'Leary* [182], *Michael and Ismail* [164], *Säckinger and Fornera* [201], *McNutt et al.* [161]):

- *Local process variations.* Small random variations on all parameters are inevitable. One can only reduce their influence by device scaling as demonstrated in the previous section.
- *Global process variations.* Most importantly causing a constant device size offset Δl_ofs; eg. caused by over/under etching, or over/under exposure of photoresist. Usually this problem is dealt with by using only "*unit size devices*" which are replicated to implement, say, wider transistors (cf. previous section). Only rational device ratios are possible in this case. This procedure also has the advantage that all unit devices can be placed in identical surrounding, reducing errors due to boundary effects. Also, channel width modulation would be the same for parallel coupled unit MOS transistors.
- *Process gradients.* In addition to random process variations, parameters are subjects to low spatial frequency, systematic variations which can be quite large. Eg. oxide thickness can vary uniformly over the wafer, or device features can be scaled differently at the wafer center and wafer edge. To minimize

such variations on matched devices, one must obviously place such as close together as possible. Also, large devices should be interweaved to ensure the same average parameters on all devices. Process gradients that causes the drain current (or capacitance for capacitors, etc.) to vary linearly with the device position can be canceled by employment of *common centroid layout*, where the devices are placed in such a way that the centers of "mass" (the centroid) of the distributed devices are common (cf. figure 85^{C}).

- *Device orientation.* As process gradients will be different in different directions, it is important that matched devices are placed symmetrical with respect to the gradients to ensure alike influence on the devices (in practice the gradient orientation is unknown and the devices are placed symmetrical with respect to the vertical and horizontal axis; this is also symmetrical with respect to process gradients that varies linear in space). As the temperature has a strong influence on most electronic device parameters, *temperature gradients* are as important as process gradients. Matched devices must be placed symmetrical with respect to known heat sources (eg. output drivers).
- *Boundary effects.* A most prominent influence on device mismatch is due to inaccuracies on the boundary of the devices. Therefore it is important to minimize these inaccuracies by ensuring identical boundary conditions for all devices.
- *Noise.* Especially in mixed analogue-digital circuits, the noise coupling can severely degrade performance. In addition to standard practice of separating power supplies and analogue/digital circuit blocks, and of using ground wire shielding when signals have to be mixed, guard bars must be placed around critical devices to reduce noise coupled via the substrate (which is usually common for all devices).

An example of matched transistor layout is given in figure 85^{C}.

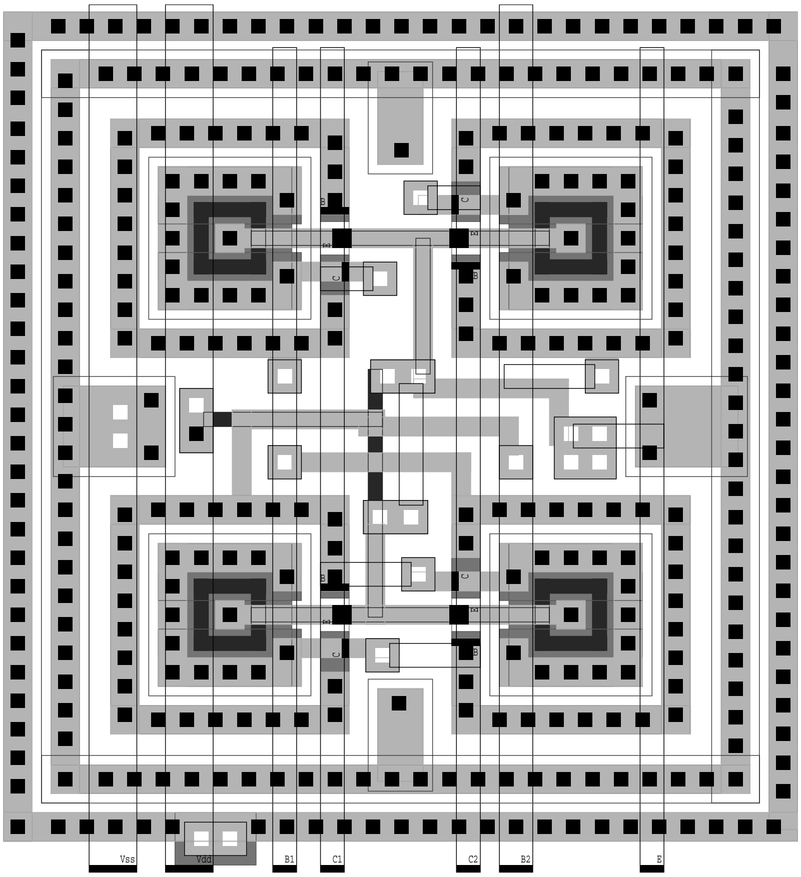


Figure 85^{C}: Layout of matched transistors. *These transistors are operated in the lateral bipolar mode which explains their large distance (necessary well-well spacing), and the heavy guarding that prevents latch-up. The common centroid layout, symmetrical device orientation and almost identical boundary conditions are noticed.*

Appendix D

System design aspects

In this appendix various aspects of the *chip/system designs*, too detailed to put in the main body of the thesis, are displayed. In section D.1, some general design considerations are given. Also, the synapse chip design is discussed and a table of measurements on the first generation chip set is given as well as proposed chip set improvements. Discussions in this chapter apply to most of the fabricated chips. In section D.2, a complete schematic and descriptions of the most important parts of the back-propagation neuron- and synapse chip are given. A table of measurements on the chip set as well as proposed chip set improvements likewise. In section D.3, a schematic, descriptions, a table of measurements and proposed chip improvements of the RTRL chip are given.

D.1 The scalable ANN chip set

A few general system aspect considerations are needed for the design of the scalable ANN chip set. These also apply to the other chips:

- *Voltage levels.* We shall use n-channel MOS resistive circuits rigorously. As these must be biased in the triode region, this determines the voltage levels we can use. For good dynamic range, both inputs to the MRC must be chosen as large as possible. Choosing the gate voltages (their difference: v_w in figure 8^2) in the range
$$v_{\mathrm{MRCG}} \in [1\,\mathrm{V}, 3\,\mathrm{V}]\,,$$
and the source/drain voltages (their difference: output voltage or v_z in the figure) in the range
$$v_{\mathrm{MRCS}}, v_{\mathrm{MRCD}} \in [-3\,\mathrm{V}, -1\,\mathrm{V}]\,,$$
gives a reasonably safe margin in our process ($V_{\mathrm{T}} < 1.8\,\mathrm{V}$ for $v_{\mathrm{BS}} = -4\,\mathrm{V}$). Further, both gate and source voltages are kept well within the power supplies, ensuring that both can be driven easily by a non rail-to-rail op-amp.
- *Current levels.* We are not primarily interested in low-power circuits. Thus, as for the voltage range, we select as high a current level as we can justify, to assure high dynamic range. Further, an increased current range allows us to reduce node impedance levels which improves the speed. The maximum current is determined by the current sinking capabilities of the synapse row differencer (cf. later). For 50–100 synapses connected to this, the full scale differential output current of the MRC (given the voltages above) must be limited to about
$$i_{\mathrm{MRC}} \in [-3\,\mu\mathrm{A}, 3\,\mu\mathrm{A}]\,.$$
(For the first synapse chip, this was set approximately one order of magnitude higher.)
- *Single ended signalling.* Though the components we use are mostly working on differential signal, we have chosen single ended signalling between the chips. The reason for this is to reduce the pin count which is > 200 for a 100×100 synapse chip with single ended signalling. The cost is a 1 bit reduction in resolution and increased noise sensitivity. We choose a *voltage reference level* at $V_{\mathrm{ref}} = -2\,\mathrm{V}$ to be compatible with the chosen MRC voltage ranges. For signals applied to the gates of the MRC we chose the reference level to $V_{w\mathrm{ref}} = 2\,\mathrm{V}$.

For easy digital chip control all digital inputs are TTL compatible. We use the *TTL level* (low $\leq 0.8\,\mathrm{V}$, high $\geq 2.0\,\mathrm{V}$) to *internal logic level* (low $-5\,\mathrm{V}$, high $+5\,\mathrm{V}$) converter in figure 86^{D}; the inset displays the level-shifter symbol. The level-shifter reference $V_{\mathrm{Dref}} \approx -2\,\mathrm{V} \to -3\,\mathrm{V}$ must be capable of sinking current. It can be generated off-chip by a zener diode. In addition to level-shifting the circuit act as a driver for internal capacitive loads.

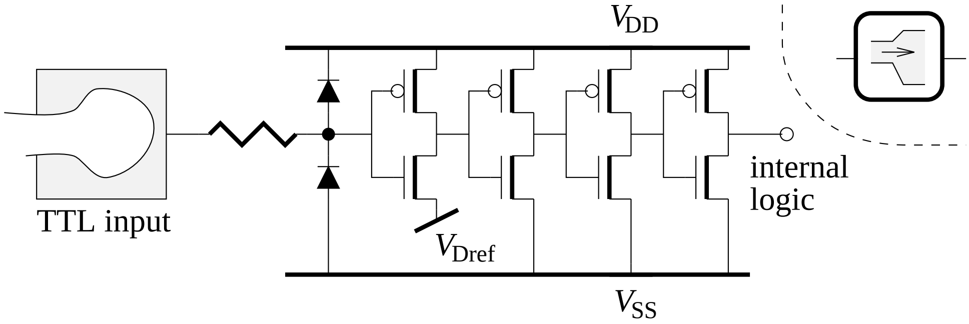


Figure 86^{D}: Digital level shifter. *TTL level to internal logic level converter. The resistor-diode gate protection circuit found on all high impedance chip inputs is also shown. The inset displays a level-shifter symbol.*

D.1.1 The synapse chips

The first generation synapse dimensions were determined to be compatible with the op-amp dimensions (the chip layout of the op-amp and the synapse matrix was done simultaneously by different individuals). As measurements have shown, the weight storage capacitance can easily be reduced. Also, the errors associated with the synapse chips were predominantly determined by the current differencers. Thus the synapse size can easily be reduced compared to the manufactured synapses. The layout of a reduced size synapse is shown in figure 87^{D}. One will notice that non-minimum lengths transistors are used for the NAND gate. The reason is that otherwise snap-back would occur in the CMOS process used. To avoid snap-back (and to reduce power consumption) the logical *high* voltage level is usually reduced with respect to V_{DD}. Unfortunately, this is impossible in the present circuit as the p-channel switches must have a $v_{GS} - V_{T} \gtrsim 0.5\,\mathrm{V}$ to reduce subthreshold leakage currents sufficiently; or (worst case) $v_{G} \gtrsim V_{w\max}$, where $V_{w\max}$ is the maximum allowable gate voltage at the computing transistors. The guard bar around the computing transistors is supposed to reduce noise coupled from the digital circuitry nearby (the highly interweaved placement of analogue and digital circuits can not be avoided, unfortunately). Likewise, several shielding power lines, that protect the analogue signals from the row- and column select signal, can be seen on the figure. Even using this shielding, noise is coupled to the analogue outputs; current spikes in the order of hundreds of nA can be detected.

In enclosure II a photomicrograph of the synapse chip can be seen. It is noticed that the row- and column decoders for writing on the synapse weight matrix are placed to the left and top of the synapse matrix. Digital circuitry is placed to the left and top of these again, to reduce interference with the sensitive analogue components. Analogue components are kept to the right and bottom of the matrix. The row- and column decoders are precharged AND gates; each having a line driving inverter at the output capable of driving 100 synapses. The column decoder is considerably faster than the row decoder, ensuring that the selected sampling switches will close (sample) at the falling edge of the column signal. The analogue weight refresh signal (that is sampled) is then distributed along each synapse row

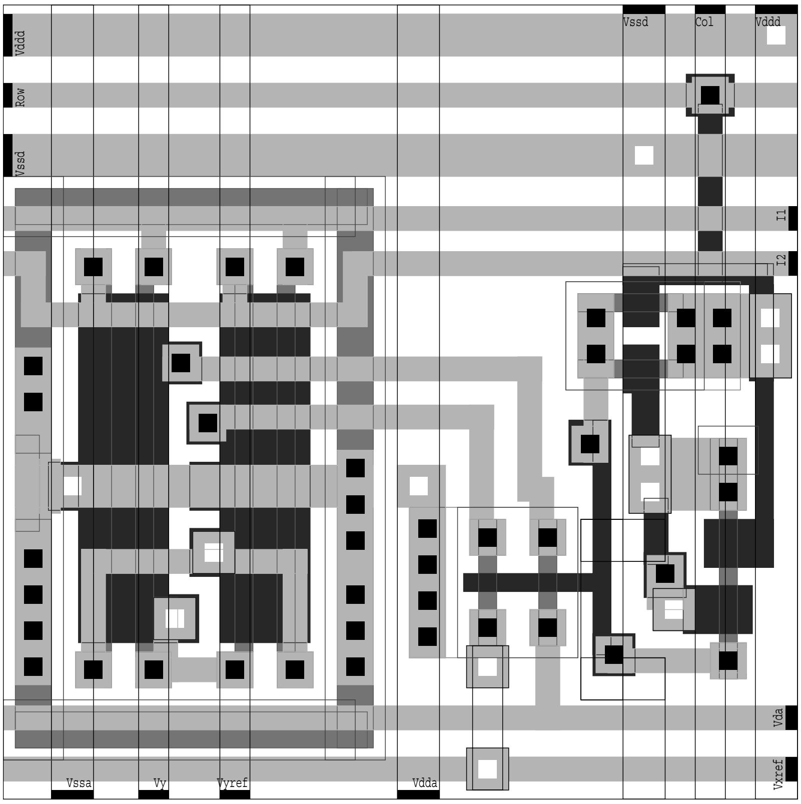


Figure 87^{D}: Synapse layout. *Reduced are. To the left, inside the guard bar, are the four computing synapse transistors. The two minimum size sampling transistors are placed in the middle and the* NAND *gate to the right.*

to reduce coupling from the sampling column signal.

Some of the most important characteristics of the first generation neuron- and synapse chips are shown in figure 88^{D}. The characteristics were measured using standard methods and equipment (oscilloscopes, signal generators, etc.) and custom PCBs for applying various bias/test signals. The synapse chip test PCB includes connection strength back-up RAM and a parallel port PC interface for accessing this. The full schematic of this PCB can be found in enclosure III.

Property	**Value**	**Bits**	**Notes**
Neuron size	$A_{\text{neu}} = 379309\,\mu\text{m}^2$		
Neuron non-linearity	$D_g \lesssim 2\%$	$6\,\text{LSB}_8$	
Neuron derivative non-linearity	$D_{dg} \lesssim 10\%$	$26\,\text{LSB}_8$	
Neuron input offset	$\lvert I_{s\text{Iofs}}\rvert \lesssim 10\,\mu\text{A}$	$26\,\text{LSB}_8$	
Neuron output offset	$\lvert V_{g\text{ofs}}\rvert \lesssim 5\,\text{mV}$	$13\,\text{LSB}_8$	
Neuron propagation delay†	$t_{g\text{pd}} \lesssim 1.8\,\mu\text{s}$	$\frac{1}{2}\,\text{LSB}_8$	$C_\text{L} \approx 16\,\text{pF}$
	$t_{g\text{pd}} \lesssim 0.8\,\mu\text{s}$	$\frac{1}{2}\,\text{LSB}_1$	$\approx 8C_\text{in}^\text{synchip}$
LPNP e/c current gain	$\alpha_\text{FC} \approx 0.55$		
Synapse size	$A_\text{syn} = 33280\,\mu\text{m}^2$		Reducible
Matrix offset	$\lvert V_{w\text{ofs}}\rvert \lesssim 16\,\text{mV}$	$2\,\text{LSB}_8$	
Matrix resolution	$V_{w\text{res}} \lesssim 2\,\text{mV}$	$\frac{1}{4}\,\text{LSB}_8$	
Synapse non-linearity	$D_{wz} \lesssim 16\%$	$21\,\text{LSB}_8$	
	$(D_{wz} \lesssim 3\%)$	$(4\,\text{LSB}_8)$	Estimated
Synapse output offset	$\lvert I_{s\text{Oofs}}\rvert \lesssim 14\,\mu\text{A}$	$14\,\text{LSB}_8$	
Synapse input offset	$\lvert V_{z\text{ofs}}\rvert \lesssim 6\,\text{mV}$	$1\,\text{LSB}_8$	$\left\{ R_\text{L} \approx 10\,\text{k}\Omega \right.$
Synapse propagation delay†	$t_{s\text{pd}} \lesssim 2.0\,\mu\text{s}$	$\frac{1}{2}\,\text{LSB}_8$	$C_\text{L} = 16\,\text{pF}$
	$t_{s\text{pd}} \lesssim 0.4\,\mu\text{s}$	$\frac{1}{2}\,\text{LSB}_1$	$\approx 8C_\text{out}^\text{synchip}$
Matrix write time‡	$t_{w\text{wr}} \lesssim 150\,\text{ns}$	$\frac{1}{8}\,\text{LSB}_8$	
Matrix (weight) drift	$\lvert\delta_w\rvert \lesssim 0.5\,\text{mV/s}$	$0.07\,\text{LSB}_8/\text{s}$	$C_{w\xi} = 1\,\text{pF}$
Weight range	$\lvert\underline{w}_{kj}\rvert_\text{max} \in [0.4, 40]$		for $y_k = \tanh(s_k)$
Layer propagation delay†	$t_{l\text{pd}} \lesssim 2.6\,\mu\text{s}$	$\frac{1}{2}\,\text{LSB}_8$	$C_\text{L} \approx 16\,\text{pF}$
	$t_{l\text{pd}} \lesssim 1.1\,\mu\text{s}$	$\frac{1}{2}\,\text{LSB}_1$	$C_\text{L} \approx 16\,\text{pF}$

† Time from input change to output has settled within $\frac{1}{2}$ LSB.

‡ Necessary length of write pulse that ensures the output will settle within $\frac{1}{8}\,\text{LSB}_8$.

Figure 88$^\text{D}$: Table of ANN chip set characteristics. *The column "Bits" is the equivalent in least significant bits of the property value given an* 8 bit *(or otherwise indicated) resolution. Note that the estimated synapse non-linearity is in compliance with the measurements on the back-propagation chip set.*

D.1.2 Chip set improvements

In addition to the important process parameter/temperature variance compensation, the following issues are subjects for improvement of the developed cascadable ANN chip set:

- Reducing the synapse area. The synapse area can be reduced to about $100\,\mu\text{m}\times 100\,\mu\text{m}$ (cf. above). Though such a reduction will increase the synapse output offset current, the synapse size *must* be small to allow massively parallel computations. If another CMOS process is used, the synapse size can be further reduced.
- Reducing the power supply voltage. Redesigning the circuit for compatibility with a 5 V (or even 3.3 V) process is not a trivial matter. The voltage range on the MRC, for instance, must be reduced to about 1 V in a 5 V process.

- Neuron output level shifting. Referring the inter-chip voltages (as the neuron output) to $-2\,\mathrm{V}$ was the cause of numerous interface problems. Level shifters should be introduced to allow a $0\,\mathrm{V}$ reference.
- Improving the op-amps: reducing area, reducing offset, increasing gain and increasing the output voltage range. Objectives of any op-amp, probably. However the *regulated gain cascodes* in the op-amp did not have the expected performance and the output voltage range must be (almost) rail-to-rail for a future implementation in a digital CMOS process. Further, the layout can be improved.
- Improving the current conveyor: as the op-amp and also improving the accuracy of the x-z current mirroring.
- Implementing on-chip bias circuits. Most bias circuits need not be very accurate and can be generated on-chip. A few external references (such as the neuron output range) are necessary, though.
- Implementing on-chip TTL level-shifter reference; and reducing the power consumption of this circuit.
- Introducing a current op-amp based synapse differencer for automatic process parameter/temperature dependency canceling.
- Placing synapse column drivers on the synapse chip instead of on the neuron chip. This improves the cascadability of the chip set.
- Reducing the neuron size. The neuron size is predominantly determined by the op-amp. Reducing the size of this will reduce the neuron size.
- Exploring other neuron topologies. If we do not need to calculate the neuron derivative as a function of the neuron output (the derivative calculation circuit could be placed on the neuron chip) other neuron circuit topologies are possible. This is discussed in section 4.6.
- Introducing offset canceling. For very large synapse chips, canceling of the output current offset may be inevitable. Different offset canceling techniques must be considered. This is discussed in chapter 4ff.
- Implementing a full-size chip set. Though the layer propagation time is not very dependent on the synapse chip size, it should be verified that a 3.8 GCPS per chip system is indeed feasible using the our $2.4\,\mu\mathrm{m}$ technology. Also, new problems might arise when scaling the system; these should be explored.

D.2 The on-chip back-propagation chip set

The on-chip back-propagation chip set is designed reusing as much layout of the scalable recall mode chip set as possible.

The back-propagation synapse chip layout closely resembles that of the second generation recall mode synapse chip. The synapse layout is identical and the row/column decoders for weight access have also been reused. The op-amp and the current conveyor for synapse output differencing are also largely unchanged; though switch transistors have been added for correct row/column element operation.

The back-propagation neuron chip, on the other hand, is drawn almost from scratch; only the op-amp layout is reused.

D.2.1 The back-propagation synapse chips

The column/row element of the back-propagation synapse chip, which is used to take the accumulated synapse output difference and to drive vectors of synapses, is shown in figure 90^{D}. The “route signal” is distributed to all rows and columns and is used to route the previous layer neuron activation when the chip operates in *route* mode. The six control signals (`A` through `F`) are operated differently for the row and the column elements as shown in figure 89^{D}.

Ctrl	**Driving signal**	
signal	**Row elements**	**Column elements**
`A`	`reverse`	$\overline{\texttt{reverse}}$
`B`	$\overline{\texttt{reverse}}$	`reverse`
`C`	`forward`	`reverse`
`D`	$\overline{\texttt{forward}}$	$\overline{\texttt{reverse}}$
`E`	`route`	`route`
`F`	`row` $= k$	`column` $= j$

Figure 89^{D}: Table of row/column element control. *The three control signals* `forward`, `reverse` *and* `route` *corresponds to the equivalent operational modes.*

The row and column select signals needed for the route mode of the chip is taken directly from the corresponding synapse matrix access (or write) signals. This means that the connection strength at location $\{k, j\}$ is lost when input j is routed to output k. After the learning hardware has computed new weight for a layer l the connection strengths in that layer will be lost and must be rewritten before the computation of new weights in the preceding layer is resumed.

The principal schematic of the row- and column elements in the different operational modes are shown in figures 91^{D} through 96^{D}.

In enclosure II a photomicrograph of the chip can be seen. The synapse matrix and the groups of row- (4) and column- (8) elements are easily identifiable. A layout error — a V_{DD} V_{SS} short circuit in some non-essential test pad cells — necessitated micro-surgery on the chip to isolate the faulty pad cells. Fortunately

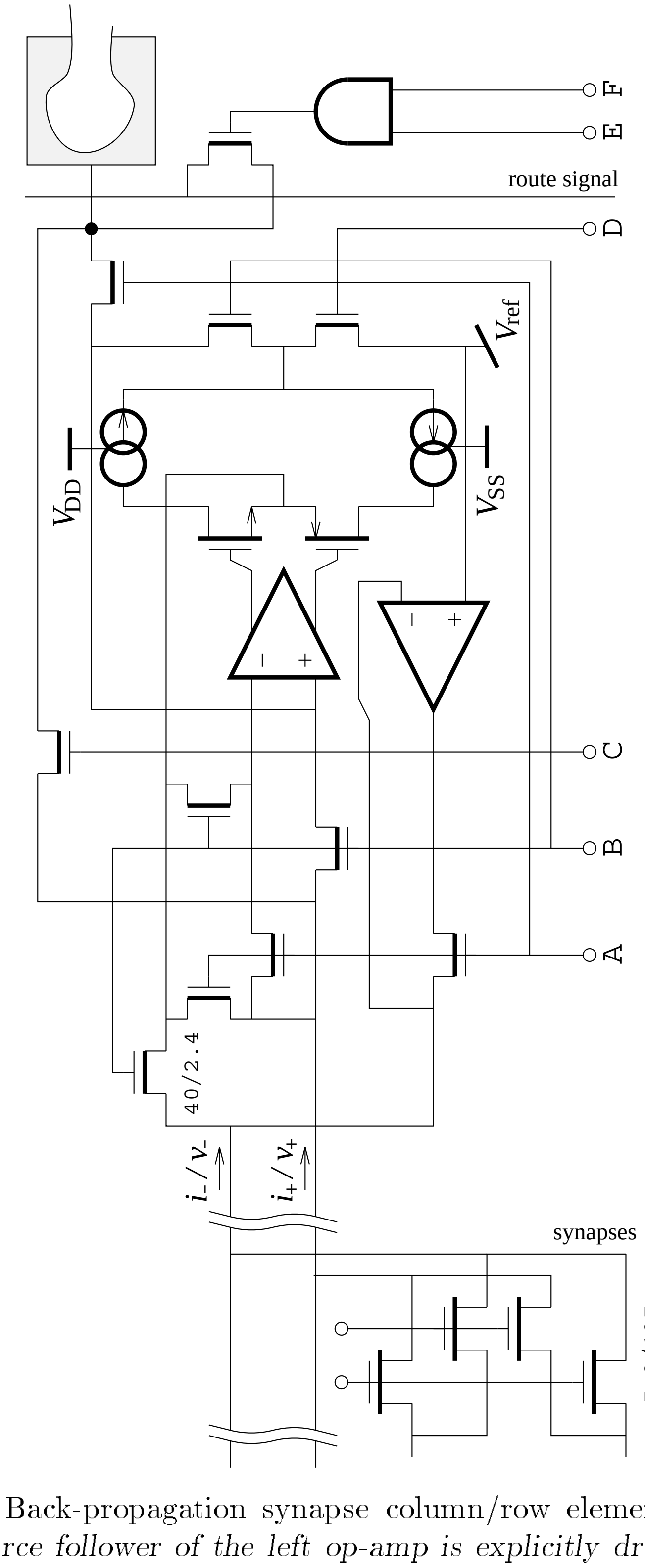


Figure 90$^{\mathrm{D}}$: Back-propagation synapse column/row element. *The output push-pull source follower of the left op-amp is explicitly drawn to show the dual functionality of this op-amp: voltage follower or current conveyor. The connection of a single synapse is also shown. Transistor W/L ratios are given for the synapse and a sample switch.*

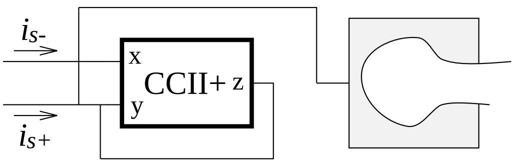


Figure 91$^{\mathrm{D}}$: Forward mode BPL synapse row element. *The CCII+ act as current differencer.*

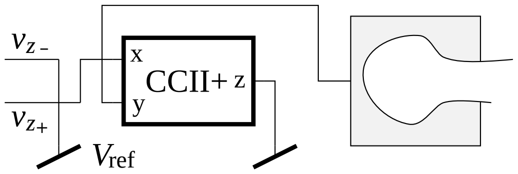


Figure 92$^{\mathrm{D}}$: Forward mode BPL synapse column element. *The CCII+ act as voltage buffer.*

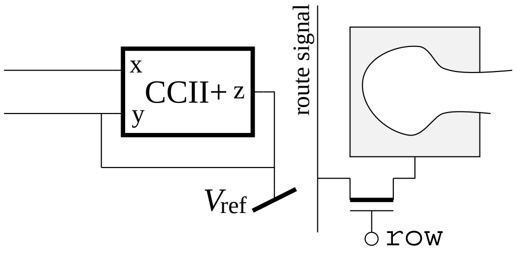


Figure 93$^{\mathrm{D}}$: Route mode BPL synapse row element. *The route signal is applied to row* `row`.

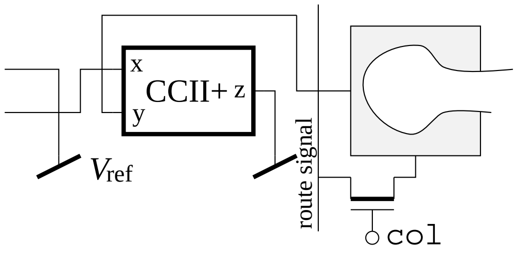


Figure 94$^{\mathrm{D}}$: Route mode BPL synapse column element. *Column* `col` *drives the route signal.*

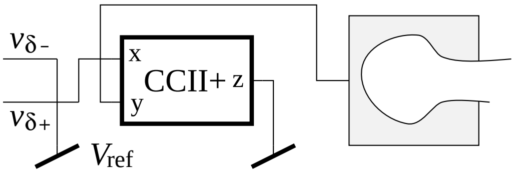


Figure 95$^{\mathrm{D}}$: Reverse mode BPL synapse row element. *The CCII+ act as voltage buffer.*

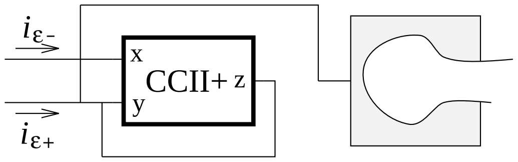


Figure 96$^{\mathrm{D}}$: Reverse mode BPL synapse column element. *The CCII+ act as current differencer.*

this was possible without damaging other parts of the chips for a reasonably large number of devices.

D.2.2 The back-propagation neuron chips

The schematic of a back-propagation neuron can be seen in figure 97$^{\mathrm{D}}$. Note the excessive use of the MRC (see also the following schematics); several of these are preceded by level shifters on the gate inputs (for a sample layout cf. appendix E.3). Various intermediate signal names are indicated in the figure. The different blocks of the neuron are identified as follows:

- `TransRFwd` is the neuron forward mode transresistance (R_{IS}) which is controlled by the external voltage V_{CG} (also V_{IR}) for adjusting the neuron slope.
- `Tanh` is the hyperbolic tangent neuron. The external voltage V_{CT} (also V_{OR}) controls the neuron output range.
- `Sampler2I` is the neuron activation sampler which is controlled by the $\overline{\texttt{holdsq}}$ signal. The differential mode sampling scheme reduces the effect of charge in-

jection and leakage currents. The sampler has an additional (global, external) input `neuact` which can be used to refresh the sampled neuron activation. This input is gated by the `refresh` signal and a "neuron k select" signal $\texttt{neusel}_k$; the latter being generated by a precharged column selector like the one used on the synapse chip.

- `TransRRew` is the reverse mode transresistance/neuron error computer. The module is active when the chip is not in forward mode. If `lastlayer` $= 1$ it computes $d_k - y_y$. Otherwise the input current is converted to a voltage; controlled by V_{CGR}. The transistor dimensions are chosen such that setting $V_{\mathrm{CG}} = V_{\mathrm{CGR}}$ is equivalent to a neuron input scaling of "1" at room temperature.
- `SqrSqr` computes the neuron derivative. For an unscaled result set the external control voltage $V_{\mathrm{C}g'} - V_{w\mathrm{ref}} = 1\,\mathrm{V}$. The maximal neuron output voltage $V_{y\mathrm{max}}$ must also be applied externally.
- `Mul1D` computes the weight strength error. For an unscaled value the external control voltage $V_{\mathrm{C}\delta} - V_{w\mathrm{ref}} = 1\,\mathrm{V}$.
- `actbus` and `deltabus` are used to distribute the neuron activation and the weight strength error respectively to the weight updating hardware.

The schematic of the back-propagation neuron chip weight updating hardware is seen in figure 98^{D}. It consist of two modules:

- `Mul1D` scales the neuron activation with the learning rate; controlled by the external signals V_η and $V_{\mathrm{C}\eta}$.
- `Mul3D` multiply the scaled neuron activation and the weight strength error. To this value the old weight (externally supplied by $V_{w_{kj}}^{\mathrm{old}}$; V_{unit} must be set to "unity", ie. $1\,\mathrm{V}$) and an offset compensation term (externally supplied via the $V_{x\mathrm{ofs}}$ and $V_{y\mathrm{ofs}}$ signals). This sum is divided by the external voltage $V_{1/(1-\varepsilon_{\mathrm{dec}})}$ giving an optional weight decay ($V_{1/(1-\varepsilon_{\mathrm{dec}})} = 1\,\mathrm{V}$ gives no decay).
- `OT1`, `OT2` and `OT3` determines the output type. This can be a "raw" one, a "level shifted" one or a "level shifted and buffered" one. The output is gated by the `learn` control signal and the chip select signal `cs`.

In enclosure II a photomicrograph of the chip can be seen. The neurons are the four long, low horizontal strips.

Some of the most important characteristics of the back-propagation neuron- and synapse chips are shown in figure 99^{D}. Again, the characteristics was measured using standard methods and equipment and custom PCBs for applying various bias/test signals. The test PCBs are similar to the ones used for the recall mode ANN chips — though the bidirectional operation requires a somewhat more flexible

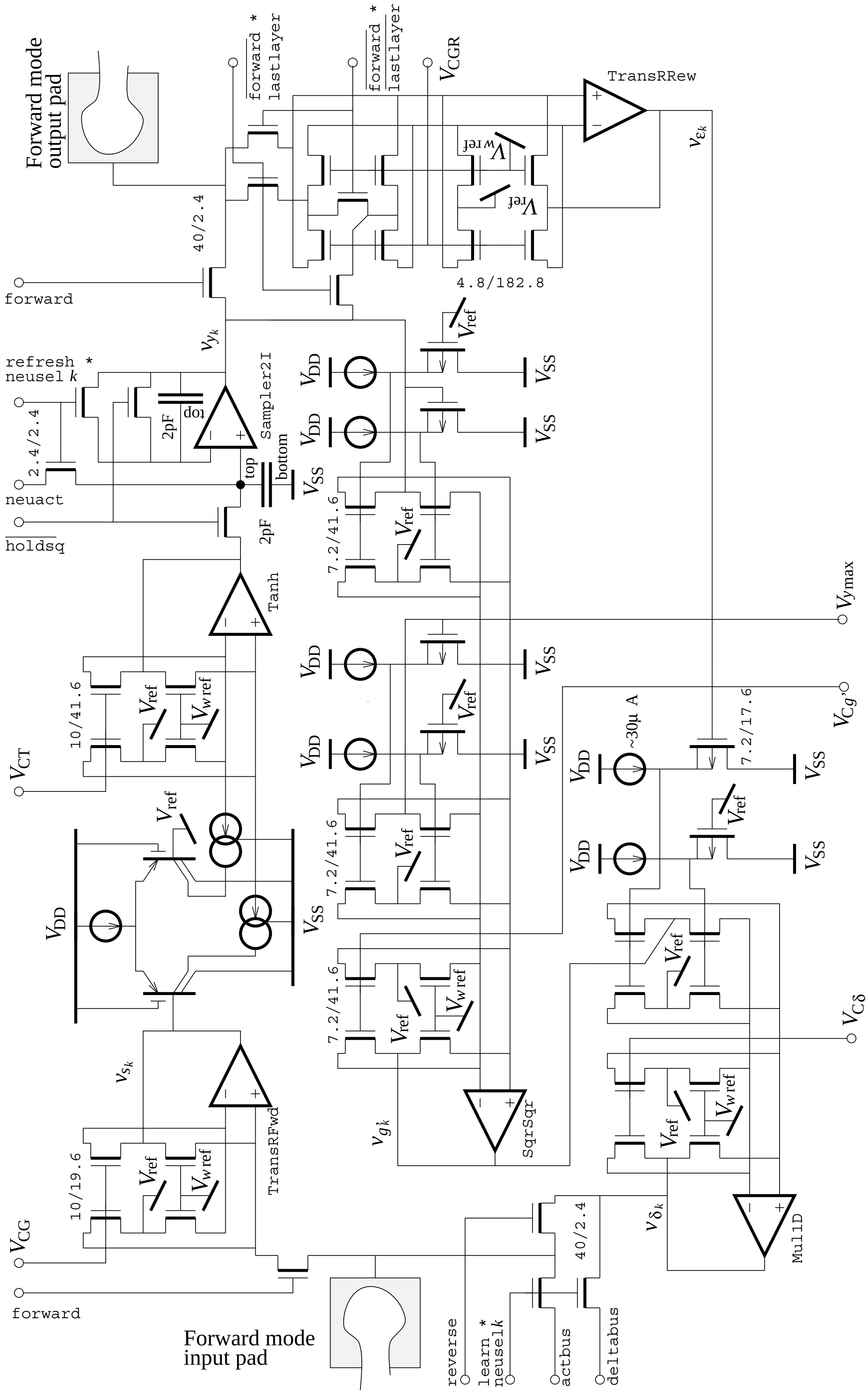


Figure 97^{D}: Back-propagation neuron schematic. *Notice the excessive use of MRCs. W/L ratios are indicated for a representative sample of transistors.*

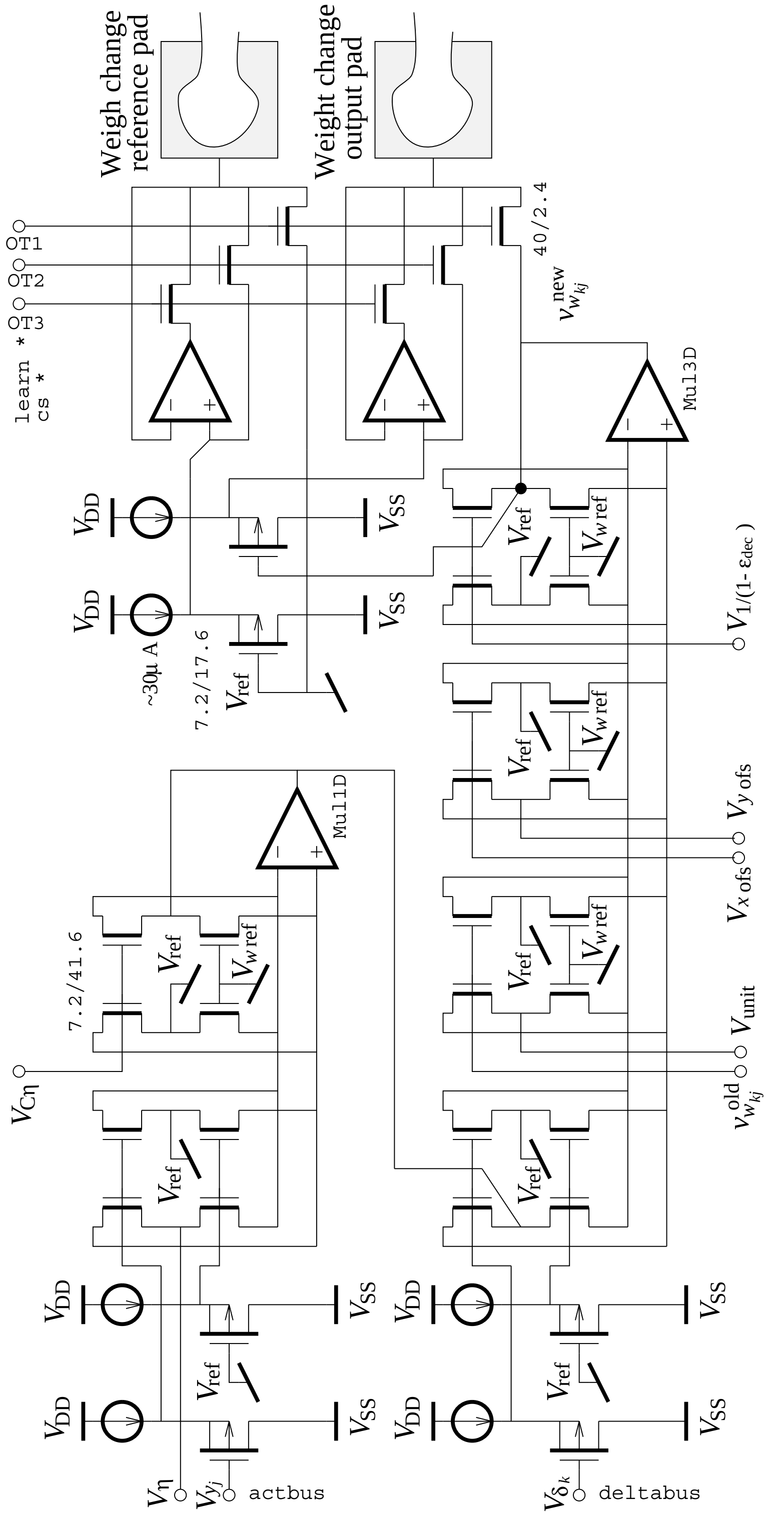


Figure 98$^{\text{D}}$: Back-propagation weight update schematic. *One instance per neuron chip. Three output types are possible (for test purposes), "raw", "level shifted" and "buffered level shifted". The reference output is needed if a level shifted output is used.*

test bed, of course. The full schematic of the back-propagation synapse- and neuron chip test PCBs can be found in enclosure III.

	Property	**Value**	**Bits**	**Notes**
Synapse	non-linearity	$D_{wz} \lesssim 3.5\,\%$	$9\,\mathrm{LSB}_8$	
	transconduc. variation†	$\Delta G_{\mathrm{syn}} \lesssim 6\,\%$	$15\,\mathrm{LSB}_8$	
	chip input offset	$\lvert V_{z\mathrm{ofs}}\rvert \lesssim 12\,\mathrm{mV}$	$2\,\mathrm{LSB}_8$	
	chip output offset	$\lvert I_{s\mathrm{Oofs}}\rvert \lesssim 3\,\mu\mathrm{A}$	$192\,\mathrm{LSB}_8$‡	
	weight offset	$\lvert V_{w\mathrm{ofs}}\rvert \lesssim 55\,\mathrm{mV}$	$7\,\mathrm{LSB}_8$	
	weight resolution	$V_{w\mathrm{res}} \lesssim 2\,\mathrm{mV}$	$1/4\,\mathrm{LSB}_8$	
Neuron	(tanh) non-linearity	$D_g \lesssim 2\,\%$	$5\,\mathrm{LSB}_8$	
	input transres. variation†	$\Delta R_{\mathrm{IS}} \lesssim 2\,\%$	$5\,\mathrm{LSB}_8$	
	output transres. variation†	$\Delta R_{\mathrm{OR}} \lesssim 2\,\%$	$5\,\mathrm{LSB}_8$	
	input offset	$\lvert I_{s\mathrm{Iofs}}\rvert \lesssim 0.8\,\mu\mathrm{A}$	$51\,\mathrm{LSB}_8$	
	output offset	$\lvert V_{y\mathrm{Oofs}}\rvert \lesssim 41\,\mathrm{mV}$	$5\,\mathrm{LSB}_8$	
	parabola non-linearity	$D_{1-y^2} \lesssim 2\,\%$	$5\,\mathrm{LSB}_8$	
	parabola gain variation	$\Delta A_{\mathrm{PG}} \lesssim 2\,\%$	$5\,\mathrm{LSB}_8$	
	parabola input offset	$\lvert V_{y\mathrm{Iofs}}\rvert \lesssim 27\,\mathrm{mV}$	$3\,\mathrm{LSB}_8$	
	parabola output offset	$\lvert V_{\delta\mathrm{ofs}}\rvert \lesssim 244\,\mathrm{mV}$	$31\,\mathrm{LSB}_8$	
	derivative non-linearity	$D_{g'} \lesssim 6\,\%$	$15\,\mathrm{LSB}_8$	
	derivative input offset	$\lvert V_{g'\mathrm{Iofs}}\rvert \lesssim 68\,\mathrm{mV}$	$9\,\mathrm{LSB}_8$	
	derivative output offset	$\lvert V_{g'\mathrm{Oofs}}\rvert \lesssim 244\,\mathrm{mV}$	$31\,\mathrm{LSB}_8$	↓ $C_{\mathrm{L}} \gtrsim 16\,\mathrm{pF}$
	Syn. chip propagation delay♮	$t_{z s\mathrm{pd}} \lesssim 0.9\,\mu\mathrm{s}$	$1/2\,\mathrm{LSB}_8$	$R_{\mathrm{L}} \approx 3\,\mathrm{k}\Omega$
	Syn. chip propagation delay♮	$t_{z s\mathrm{pd}} \lesssim 15\,\mu\mathrm{s}$	$1/2\,\mathrm{LSB}_8$	$R_{\mathrm{L}} \approx 100\,\mathrm{k}\Omega$
	Neu. squashing prop. delay♮	$t_{s y\mathrm{pd}} \lesssim 1.6\,\mu\mathrm{s}$	$1/2\,\mathrm{LSB}_8$	$C_{\mathrm{L}} \gtrsim 16\,\mathrm{pF}$
	Neu. chip weight calc. time♮	$t_{z\Delta w\mathrm{pd}} \lesssim 3.6\,\mu\mathrm{s}$	$1/2\,\mathrm{LSB}_8$	$C_{\mathrm{L}} \gtrsim 16\,\mathrm{pF}$
	Synapse weight drift	$\lvert\delta_w\rvert \lesssim 0.2\,\mathrm{mV/s}$	$0.03\,\mathrm{LSB}_8/\mathrm{s}$	$C_{\mathrm{H}} \approx 2 \times 1.6\mathrm{pF}$
	Neuron activation drift	$\lvert\delta_y\rvert \lesssim 0.5\,\mathrm{mV/s}$	$0.06\,\mathrm{LSB}_8/\mathrm{s}$	$C_{\mathrm{H}} \approx 2 \times 2\,\mathrm{pF}$

† The variations are for a single chip.

‡ Equivalent LSB of a single synapse.

♮ The delays are for the signals to settle within the given precision.

Figure 99$^{\mathrm{D}}$: Table of Back-propagation chip set characteristics. *Apart from the reduced synaptic output current level and the increased neuron output offset, the chip performances are similar to the recall mode chip set.*

D.2.3 The scaled back-propagation synapse chips

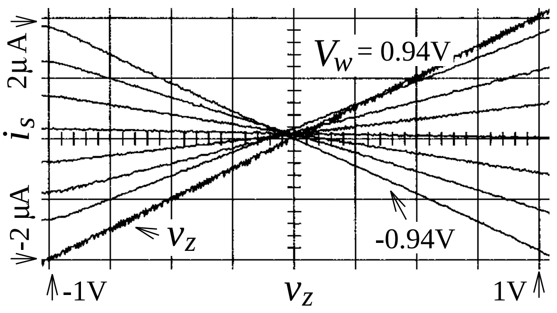
The scaled back-propagation synapse chip was implemented simply by adding rows and columns to the back-propagation synapse chip. In enclosure II a photomicrograph of the chip can be seen. It is seen that now, for this matrix size (16×16), the major part of area is clearly taken up by synapses. The control signals for the scaled back-propagation synapse chip being identical to the ones of the first back-propagation synapse chip, the test PCB was constructed as a "piggyback" PCB that would fit into the synapse chip socket of the original back-propagation synapse chip test PCB. The piggyback PCB splits the address space of the original chip in two: one part is mapped on a relocatable part of the scaled chip address space while the other is replicated in the remaining address space. In this way it is possible for any synapse strength to be set independently of the other synapse strengths, using the limited address space of the original back-propagation synapse chip test PCB. The full schematic of the scaled back-propagation synapse chip test piggyback PCB can be found in enclosure III.

The scaled synapse chip being malmanufactured, measured chip properties (for instance propagation delays, offset errors and non-linearities) will not characterize the chip well. Measurements done on the original back-propagation synapse chip will probably, for most parts, give a better performance estimate of properly manufactured chips. As we shall try to use the chip in our RTRL/back-propagation system in spite of the poor performance, a few measurement results are given in figure 100^{D}. Notice the very large systematic offset errors. The reference voltage was raised to $V_{\mathrm{ref}} = -1.5\,\mathrm{V}$ to accommodate to the reduced input range of the current conveyors. A typical synapse transfer characteristic is shown in figure 101^{D}.

Property	**Value**	**Bits: stoch.**	**syst.**
Matrix offset	$\lvert V_{w\mathrm{ofs}} + 23\,\mathrm{mV}\rvert \lesssim 16\,\mathrm{mV}$	$2\,\mathrm{LSB}_8$	$3\,\mathrm{LSB}_8$
Synapse non-linearity	$D_{wz} \lesssim 3\%$	$8\,\mathrm{LSB}_8$	
Chip output offset	$\lvert I_{s\mathrm{Oofs}} - 9.3\,\mu\mathrm{A}\rvert \lesssim 2.5\,\mu\mathrm{A}$	$213\,\mathrm{LSB}_8$	$794\,\mathrm{LSB}_8$
Synapse input offset	$\lvert V_{z\mathrm{ofs}} - 30\,\mathrm{mV}\rvert \lesssim 6\,\mathrm{mV}$	$1\,\mathrm{LSB}_8$	$4\,\mathrm{LSB}_8$
Synapse input range	$v_z + V_{\mathrm{ref}} \in [-2.5\,\mathrm{V}, 0.0\,\mathrm{V}]$		
Synapse output range	$\lvert i_{wz}\rvert \lesssim 1.8\,\mu\mathrm{A}$	$\lvert v_z\rvert, \lvert V_w\rvert \leq 1\,\mathrm{V}$	

Figure 100^{D}: Table of scaled BPL synapse chip characteristics. *Malfabricated chip; notice the large systematic offsets. See also the characteristics of the non-scaled chip.*

Figure 101^{D}: Scaled synapse chip characteristics. *Measurements on a single synapse (the output offset has been canceled). Notice the restricted input range (and the fairly low transconductance); presumably caused by a low process transconductance parameter and raised threshold voltages.*



D.2.4 Back-propagation chip set improvements

Being composed primarily of components found on the recall mode ANN chip set, most of the improvements mentioned in section D.1.2 apply also to the back-propagation chip set (eg. reducing the power supply, improving the op-amps and current conveyors and implementing on-chip bias circuits and temperature compensation). A few additional issues are subjects for improvement of the developed cascadable back-propagation learning ANN chip set:

- Redirection switch matching. The weight offsets are primarily determined by mismatch in the synapse chip redirection switches. These should be matched.
- Auto offset compensation. Many offset errors being destructive to learning processes, auto offset compensation circuits should be included on the chips. This could be chopper stabilizing circuits or circuits like the one mentioned in section 5.3.2. Several signals are subjects for offset compensation:
  - The synapse chip forward mode output current.
  - The synapse chip reverse mode output current.
  - The neuron chip weight change output voltage.
  - The neuron chip neuron activation output voltage range.
  - The neuron chip weight change error voltage.

  Except for the weight change output, the wire count of these signals grows as $O(N)$; thus a fairly simple (cheap) offset compensation scheme must be employed.
- Improved neuron derivative computation. Several choices for improvement is possible; perhaps the simplest would be to clipping negative outputs from this circuit.
- Scaled synapse chip remanufacturing.
- Non-linear back-propagation extensions. The synapse chip is compatible with non-linear back-propagation. The neuron chip can in a simple way be expanded to include non-linear back-propagation.

D.3 The RTRL chip

The width N data path RTRL chip consists almost entirely of layout taken from the back-propagation chip set. The chip development has thus consisted mainly of a rearrangement (and rerouting) of building blocks; plus the layout of a few digital components.

The schematic of a width N data path RTRL signal slice can be seen in figure 102^{D}. Various intermediate signal names are indicated in the figure. The different blocks of the neuron are identified as follows:

- `ETSampler` is the edge trigged neuron activation sampler. The input signal $v_{y_k}(t)$ is sampled at the falling edge of the $\phi_{\mathrm{S}y}$ clock signal.
- `Diff` computes the neuron error signal $v_{\varepsilon k}(t) = v_{d_k}(t) - v_{y_k}(t)$, where $v_{d_k}(t)$ is the externally applied target value. The output is gated via the target select input $T_k(t)$.
- `SqrSqr` computes the neuron derivative $v_{g'_k}(t)$ (assuming a hyperbolic tangent activation function). The maximal neuron output voltage $V_{y\mathrm{max}}$ must be supplied externally.
- `TransR` computes the net input derivative variable $v_{\sigma^k_{ij}}(t)$. It is composed of a transresistor (transresistance controlled by the externally applied voltage $V_{R\sigma}$) and the signal slice k part of the distributed i_{z_j} demultiplexor (the tree transistors to the left controlled by bits i_0 and i_1 (plus chip select i_{CS} for cascading) of the i variable). The external current input $i_{wp_k}(t)$ is added to the demultiplexed i_{z_j} signal.
- `Mul1D` computes the neuron derivative variable $v_{p^k_{ij}}(t)$. This output is routed to the (chip) global $v_{p^{k'}_{ij}}(t)$ chip output when selected by the k' (bits k'_0, k'_1 and chip select k'_{CS}) neuron derivative variable access input. This input also controls the:
- `Sampler` is the signal slice k $v_{p^k_{ij}}(t-1)$ sampler. The input is taken from the (chip) global $v_{p^{k'}_{ij}}(t-1)$ input and is sample at the falling edge of the $\phi_{\mathrm{S}p}$ clock input when selected by k' (using a precharged address decoder as shown). Actually, two $v_{p^{k'}_{ij}}(t-1)$ input channels are provided; one is meant for initializing the p^k_{ij}s and the other is meant for normal operation.
- `VMulElm` is the signal slice k part of the distributed inner product multiplier computing the weight changes. The outputs are connected to the shared current conveyor. The inputs are the $v_{g'_k}(t)$ and either $v_{p^k_{ij}}(t)$ or $v_{\sigma^k_{ij}}(t)$ (for using the quadratic and entropic cost function respectively) controlled by the $\mathtt{Q}/\overline{\mathtt{E}}$ and $\overline{\mathtt{Q}/\overline{\mathtt{E}}}$ inputs.

The schematic of the order N signal path RTRL chip weight updating hardware is seen in figure 103^{D}. It consists basically of the CCII+ current conveyor that takes the difference of the distributed MRC inner product weight change multiplier (the CCII+X and CCII+YZ lines); connected to the input of the CCII+ are four MRCs used for offset compensation. The first is used for external compensation; a (differential mode) current proportional to the V_{ofs} input is added to the weight change current. The three other MRCs are used for the automatic offset compensation circuit. One is used to ensure a positive output current offset, and the other two are connected to the two 8 bit offset compensating voltage output DACs (vendor standard cell components). The V_{AZMM} and V_{AZML} inputs are used to control the transconductance of the MRCs connected to the most significant and least significant DAC respectively (the MRCs are scaled such that $V_{\mathrm{AZMM}} \approx 10V_{\mathrm{AZML}}$ should be chosen).

When the `azero` auto-zeroing signal is high, the CCII+ output is directed to the current comparator rather than the chip weight change current output $i^{\mathrm{chip}}_{\Delta w_{ij}}$. Also shown on the figure is the v_{z_j} transconductor. The transconductance is controlled by the V_{Gz} input. In enclosure II a photomicrograph of the chip can be seen. Note that the D/A converters (lower left) and the SAR (lower right) take up a considerably amount of area. The signal slices are the four long, low rows (top).

A "start conversion", `sc`, signal is needed for the successive approximation register for initiating an auto offset cancellation phase. This is generated by the circuit in figure 104$^{\mathrm{D}}$: when `azero` signal goes high, the `sc` signal will be high in the following ϕ_1 clock phase. (The CMOS transmission gate symbol is defined in figure 105$^{\mathrm{D}}$. It must be driven by a complementary signal on the gates; in the `sc` generating circuit only non-inverting input signals are shown for simplicity.)

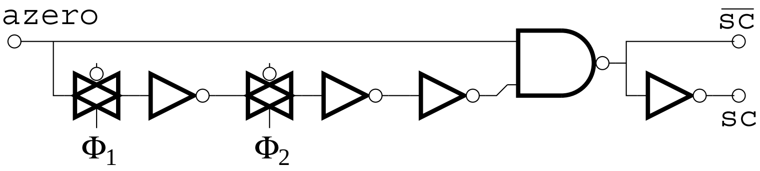
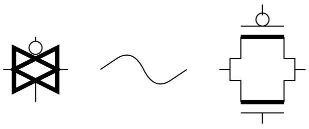


Figure 104$^{\mathrm{D}}$: SAR start signal gating. *Generation of one-clock-period start signal for successive approximation register. The two successive inverters can be removed; they are present for design ease.*

Figure 105$^{\mathrm{D}}$: Transmission gate symbol. *Commonly used symbol for CMOS transmission gate controlled by complementary signals (top and bottom). Sometimes one one of the control gates are explicitly shown driven.*



Standard CMOS digital logic usually requires (as our circuit) a *two phase non-overlapping clock*; usually (as in our case) available in both true and complimentary forms. A circuit that generates the four clock-phases ϕ_1, $\overline{\phi_1}$, ϕ_2, and $\overline{\phi_2}$ from a single input clock signal ϕ is shown in figure 106$^{\mathrm{D}}$. It is based on a cross coupled NOR gate pair for generating the two non-overlapping clock phases (see eg. *Geiger et al.* [77]) and on two $n/n+1$ inverter chain pairs for the generation of skew free inverted clock signals (*Shoji* [221]). Additional capacitive loads can be placed at the outputs of the NOR gates (when `IncICD` $= 1$) to increase the inter-clock delay (the time period when both $\phi_1 = 0$ and $\phi_2 = 0$).

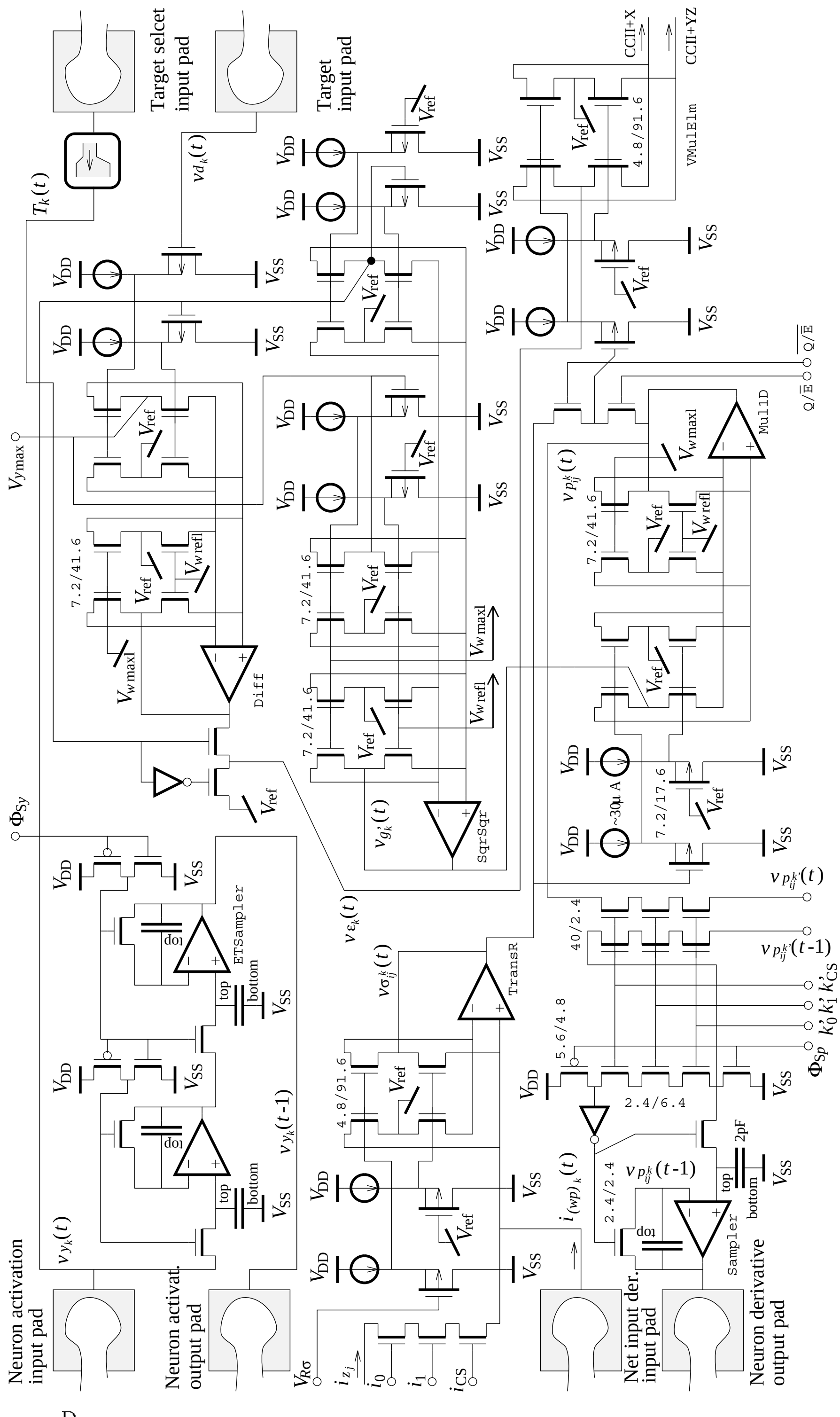


Figure 102$^{\mathrm{D}}$: RTRL signal slice schematic. *The width N data path signal slice is composed mostly of components also found on the back-propagation chip set. Locally generated references ($V_{w\mathrm{refl}}$ and $V_{w\mathrm{maxl}}$) are used out of convenience.*

For skew free clock inversion, one must ensure that the sum of rise times (t_{R}

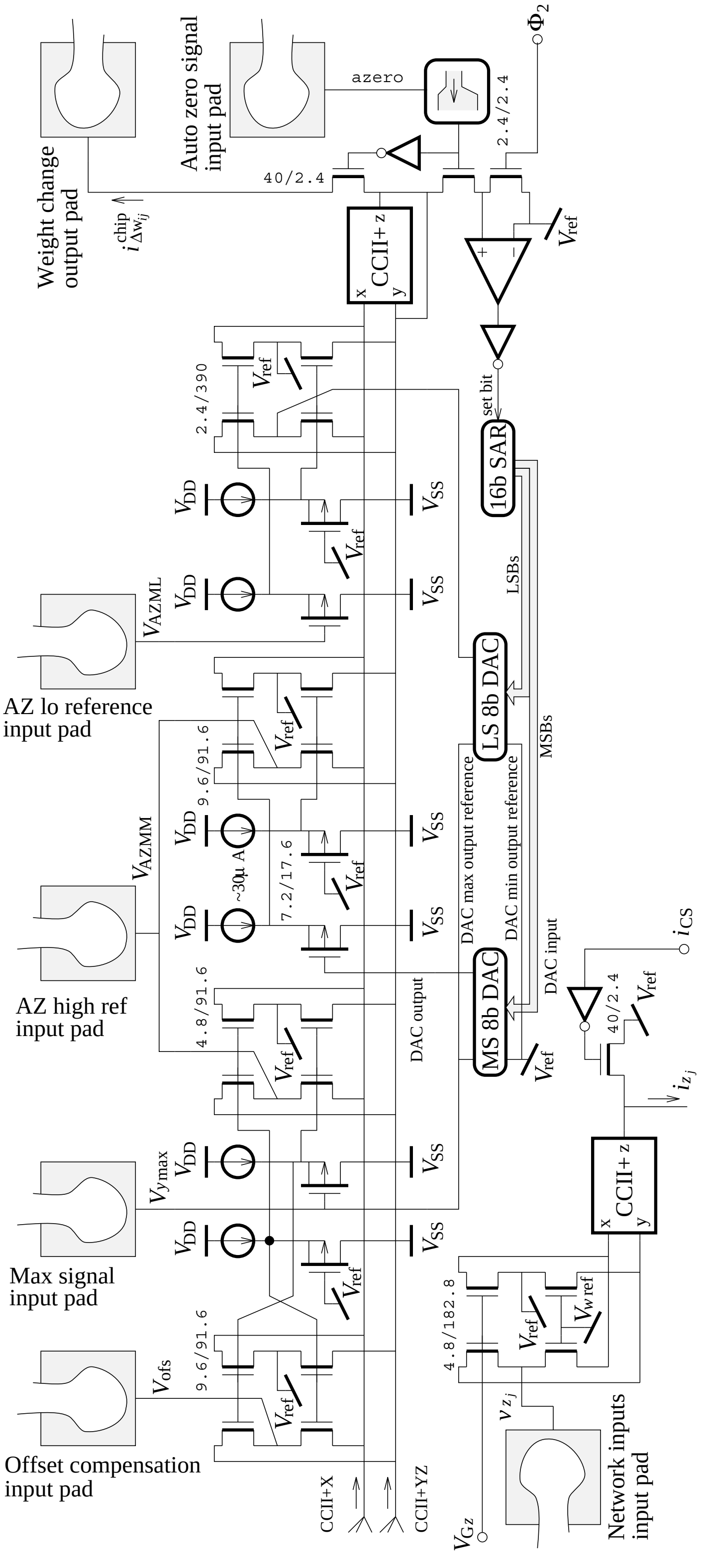


Figure 103$^{\text{D}}$: RTRL weight change schematic. *One instance per RTRL chip. Most of the auto zeroing circuit is shown only as a block schematic. The network input signal transconductor is also shown in the figure.*

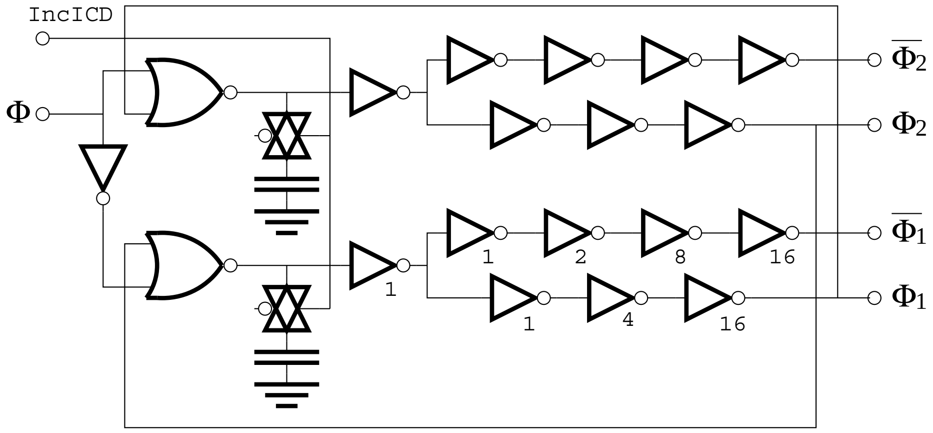


Figure 106$^{\mathrm{D}}$: Clock generator. *Two-phase non-overlapping skew-free clock generating circuit with large capacitive load driving capability. The inter-clock delay can be increased by adding capacitance at* NOR *gate outputs. ϕ_1 is a delayed version of the input ϕ.*

or t_{pdLH}) in the inverting (3 inverters) and the non-inverting (4 inverters) inverter chains are equal for both falling and raising input signals. Likewise for the sum of fall times (t_{F} or t_{pdHL}). Assume valid the simple CMOS inverter *rise/fall time* relations (see eg. *Weste and Eshraghian* [262]):

$$t_{\mathrm{R}} \approx \frac{4C_{\mathrm{L}}}{(V_{\mathrm{DD}} - V_{\mathrm{SS}})K'_{\mathrm{P}}W_{\mathrm{P}}/L_{\mathrm{P}}}, \qquad t_{\mathrm{F}} \approx \frac{4C_{\mathrm{L}}}{(V_{\mathrm{DD}} - V_{\mathrm{SS}})K'_{\mathrm{N}}W_{\mathrm{N}}/L_{\mathrm{N}}}, \qquad (46^{\mathrm{D}})$$

where C_{L} is the inverter load capacitance, and assume

$$C_{\mathrm{L}\xi} \approx C_{\mathrm{ox}}(W_{\mathrm{N}\xi+1}L_{\mathrm{N}\xi+1} + W_{\mathrm{P}\xi+1}L_{\mathrm{P}\xi+1}),$$

where the indices (ξ and $\xi + 1$) refer to inverter number counted from the left. In this case using the relative transistor widths indicated in figure 106$^{\mathrm{D}}$ (equal lengths) gives skew free inversion at the output of the third inverter in the upper chain compared to the second inverter in the lower chain. If the final load is unknown, this is the best we can do; the output inverters are designed to drive a large capacitive load with equal rise and fall times for typical process parameters.

The RTRL chip test PCB schematic is shown in enclosure III. Chip measurements was done in the standard way; a table of chip characteristics is found in figure 107$^{\mathrm{D}}$. The chip — as the scaled back-propagation synapse chip — being malmanufactured, chip properties (as propagation delays, offset errors and signal ranges) will not be characteristic for a properly manufactured chip. As we shall try to utilize the chip anyway, we have included a selection of characteristics. The reference voltage was changed to $V_{\mathrm{ref}} = -1.5\,\mathrm{V}$ to accommodate to the reduced input voltage range of the current conveyor (as it was for the scaled synapse chip).

Property	**Value**	**Bits**	**Notes**
Error input range	$v_d - v_y \in [-0.5, 0.6]\,\mathrm{V}$†		
Sampler input range‡	$v_\xi + V_{\mathrm{ref}} \in [-4.0, 3.2]\,\mathrm{V}$		
Neuron input input range	$v_z + V_{\mathrm{ref}} \in [-2.8, -0.3]\,\mathrm{V}$		
Neuron input offset	$\lvert V_{z\mathrm{ofs}} \rvert \lesssim 300\,\mathrm{mV}$	$30\,\mathrm{LSB}_8$	$\lvert w_{ij} \rvert_{\max} = 1$
Weight change offset	$\lvert I_{\Delta w\mathrm{ofs}} \rvert \lesssim 2\,\mu\mathrm{A}$	$165\,\mathrm{LSB}_8$♮	
Error input offset	$\lvert V_{\varepsilon\mathrm{ofs}} \rvert \lesssim 50\,\mathrm{mV}$		Targeted
Net input derivative offset	$\lvert I_{wp\mathrm{ofs}} \rvert \lesssim 54\,\mathrm{nA}$	$4\,\mathrm{LSB}_8$♮	
Sampler output offset	$\lvert V_{\xi\mathrm{ofs}} \rvert \lesssim 10\,\mathrm{mV}$	$1\,\mathrm{LSB}_8$	at $v_{\mathrm{in}} = 0\,\mathrm{V}$
	$\lvert V_{\xi\mathrm{err}} \rvert \lesssim 20\,\mathrm{mV}$	$3\,\mathrm{LSB}_8$	$\lvert v_{\mathrm{in}} \rvert \leq 1\,\mathrm{V}$
Derivative output offset	$\lvert V_{p(t)\mathrm{ofs}} \rvert \lesssim 3\,\mathrm{mV}$	$0.4\,\mathrm{LSB}_8$	
Parabola input offset	$\lvert V_{g'\mathrm{Iofs}} \rvert \lesssim 33\,\mathrm{mV}$	$4\,\mathrm{LSB}_8$	
Parabola output offset	$\lvert V_{g'\mathrm{Oofs}} \rvert \lesssim 200\,\mathrm{mV}$	$26\,\mathrm{LSB}_8$	
IPM element non-linearity	$D_{\sigma\varepsilon} \lesssim 5\,\%$	$13\,\mathrm{LSB}_8$♮	
Multiplier non-linearity	$D_{\mathrm{mul}} \lesssim 3\,\%$	$8\,\mathrm{LSB}_8$	
Parabola non-linearity	$D_{1-y^2} \lesssim 3\,\%$	$8\,\mathrm{LSB}_8$	
Propagation delays♯	$t_{\xi\mathrm{pd}} \sim 10\,\mu\mathrm{s}$		
Sampler decay rate	$\lvert \delta_\xi \rvert \lesssim 0.6\,\mathrm{mV/s}$	$0.08\,\mathrm{LSB}_8/\mathrm{s}$	

† For $v_d \in [-1, 1]\,\mathrm{V}$ at $V_{\mathrm{ref}} = -1.5\,\mathrm{V}$. $v_d - v_y \in [-1, 1]\,\mathrm{V}$ is possible, though the difference will be non-linear.

‡ Both the edge trigged ($v_{y(t)}$) and the transparent ($v_{p(t)}$) sampler.

♮ Relative to single multiplier element.

♯ Typically; in the order of.

Figure 107$^{\mathrm{D}}$: Table of RTRL chip characteristics. *Malfabricated chip; primarily resulting in a reduced dynamic range. The large $V_{z\mathrm{ofs}}$ is caused by a design flaw. See also the back-propagation chip set characteristics.*

D.3.1 RTRL chip improvements

Being composed primarily of components found on the back-propagation chip set, most of the improvements mentioned in the previous sections apply also to the RTRL chip. A few additional issues are subjects for improvement of the developed real-time recurrent learning chip:

- Making auto offset compensation work. Because the weight change offset is of paramount importance, one of the primary tasks of future research is to implement acting auto offset compensation hardware. A chip remanufacturing might solve the problem but one must make certain that this is probable before doing so.
- Reduce v_z input offset. This can be done in a straight-forward manner: just redesigning the output current mirrors of the current conveyor to the correct current range.
- Improve neuron derivative computation. Just as was the case for the back-propagation neuron chip (see above); though recall that only the neuron activation (and not the neuron net input) will be available for the calculation in

this case.

- Chip remanufacturing.
- Non-linear RTRL extensions. Equivalently to the back-propagation neuron chip, the *with N data path* RTRL module can in a simple way be expanded to include non-linear real-time recurrent learning.

D.4 The RTRL/back-propagation system

A complete schematic (without decoupling capacitors) of the RTRL/back-propagation system can be found in enclosure III. It is currently under construction thus we can not present any measurements based on it. Note that the external synapse chip output offset compensation have yet to be included on the board. The board basically consists of a large number of D/A and A/D converters and digital latches, interfaced to a standard PC AT (ISA) bus, for controlling the custom ASICs in our learning system. Various control signals used on the board — the low-level programmers interface — are described in enclosure IV.

Appendix E

Building block components

In this appendix various *building block components* used on the chips are briefly described: The *regulated gain cascode* based *operational amplifier*, *current conveyor* and *transconductor* designed by Thomas Kaulberg. Also the layout of typical *MOS resistive circuits* is shown.

E.1 The op-amp and the CCII+

The *operational amplifier* and the current conveyor used on the chips was designed by Thomas Kaulberg. The op-amp schematic is shown in figure 108^{E}. It is a two stage cascode amplifier; the gain originating from the input stage differential current being dumped into the very high impedance node where the compensation capacitor is connected. The bulks of the p-channel transistors of the output push-pull source follower are connected to the source terminals to lower the minimal output voltage. The the p-channel transistor placed in series with the p-channel current mirror is present for avoiding snap-back.

For high gain, *regulated gain cascodes* (*RGC*) (*Säckinger and Guggenbühl* [202]) have been used for the current mirrors. A p-type RGC is shown in figure 109^{E}: The drain of main transistor (the transistor connected to the "gate" terminal) is kept at a constant potential using a cascode transistor and a simple inverting amplifier (the two left most transistors). An n-type RGC current mirror is seen in figure 110^{E}.

Doing *supply current sensing* on the output transistors of the op-amp; and connecting the op-amp as a voltage follower, we have a *CCII+* type *current con-*

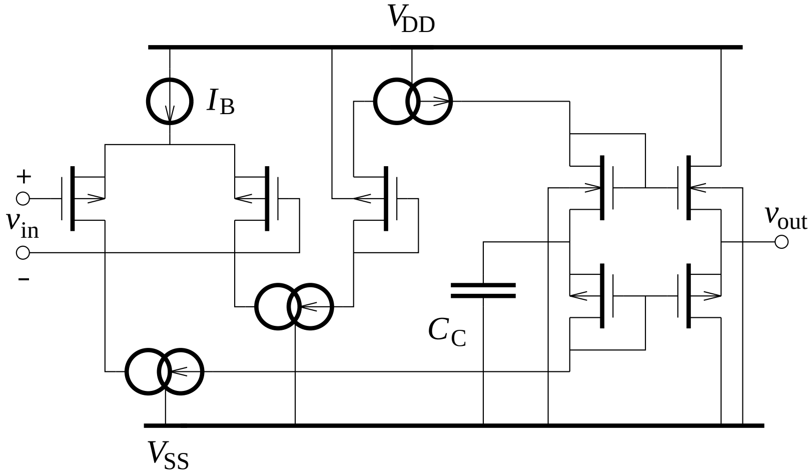


Figure 108^E: The operational amplifier. *Regulated gain cascode opamp with push-pull source follower output stage. Only a single very high impedance node contributes to the gain.*

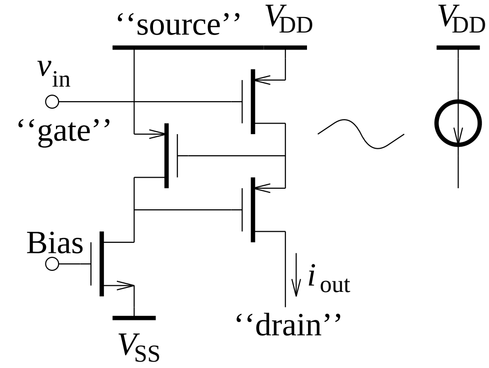


Figure 109^E: Regulated gain cascode. *P-type RGC. The local feedback provides a very high output impedance at the "drain" terminal.*

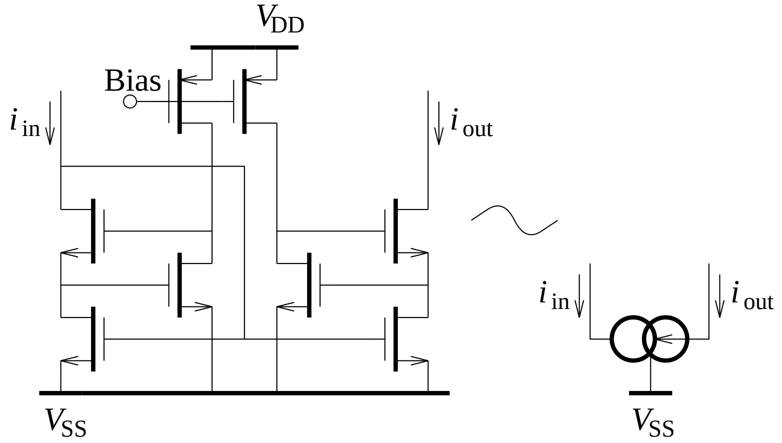


Figure 110^E: RGC current mirror. *Accurate, high output impedance, N-type mirror composed of two regulated gain cascodes. One must ensure (by transistor sizing) that the transistors are saturated in the relevant input current range.*

veyor (*Toumazou et al.* [241, 242, 244]). This is shown in figure 111^{E}(the CCII+ symbol is given in figure 18^2). The CCII± voltage/current relations are:

$$\begin{bmatrix} i_{\mathrm{Y}} \\ v_{\mathrm{X}} \\ i_{\mathrm{Z}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & \pm 1 & 0 \end{bmatrix} \begin{bmatrix} v_{\mathrm{Y}} \\ i_{\mathrm{X}} \\ v_{\mathrm{Z}} \end{bmatrix} .$$

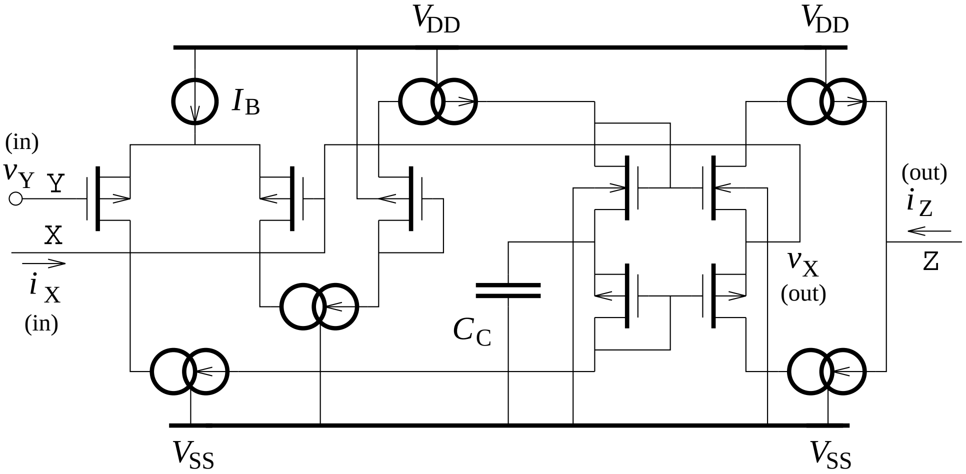


Figure 111^{E}: The current conveyor. *CCII+ implemented by current supply sensing the output transistors of the unity gain coupled opamp. Removing the feedback, we have a very versatile four terminal operational component.*

If we omit the feedback, the resulting four terminal device is a very versatile component; this is used on the back-propagation synapse chip. Using $2\,\mu$A RGC bias currents and $60\,\mu$A differential pair tail currents, the typical open loop gain is as shown in figure 112^{E}. Other typical characteristics are (at a $10\,\mathrm{k}\Omega \| 16\,\mathrm{pF}$ load):

- Input voltage offset $V_{\mathrm{Yofs}} \lesssim 5\,\mathrm{mV}$.
- Output current offset $I_{\mathrm{Zofs}} \lesssim 3\,\mu\mathrm{A}$.
- Voltage follower slew rate $\mathrm{SR} \gtrsim 5\,\mathrm{V}/\mu\mathrm{s}$.
- Current output range $|i_{\mathrm{Z}}|_{\mathrm{max}} \gtrsim 300\,\mu\mathrm{A}$.

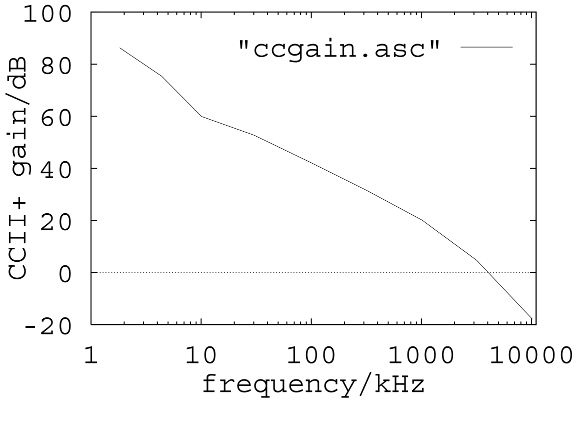


Figure 112^{E}: Op-amp frequency response. *Measured open loop gain vs. frequency for sample CCII+/op-amp (positive input to voltage output).*

E.2 The transconductor

The *transconductor* used on the first generation synapse chip was designed by Thomas Kaulberg. The principal schematic is shown in figure 113^{E}. The resistor is implemented as an n-well resistor.

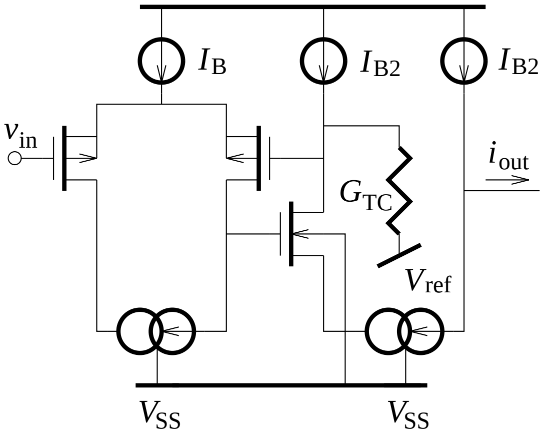


Figure 113^{E}: The transconductor. *Basically a CCII+ with a (N-well) resistor connected to the x-terminal. The CCII+ is based on (single ended) supply current sensing of a two-stage unity-gain coupled opamp.*

E.3 MOS resistive circuit

The *MOS resistive circuit* is used excessively in this work; often with level shifters at the gate inputs. A typical MRC layout is shown in figure 114^{E} (taken from the back-propagation neuron chip). Also shown is a pair (or rater $1\frac{1}{2}$ pair) of gate input level shifters; p-channel source followers (with bulk connected to source for precise buffering) driven by RGC current sources. All MRC gate input level shifters occur in pairs to ensure matching of the voltage by which the level shifters raise the input voltages.

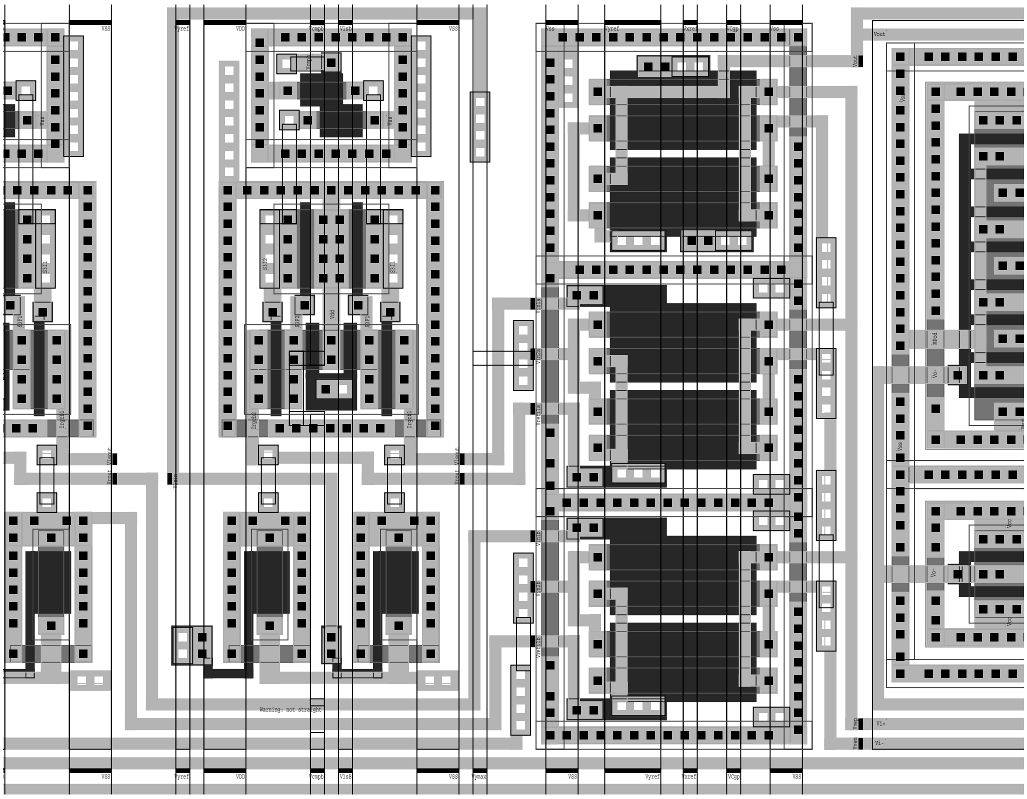


Figure 114^{E}: Typical MRC layout. *The layout of three MRCs connected to an op-amp (right). Two MRCs are preceded by level shifters (left): The P-channel MOSTs source followers and the corresponding regulated cascode current sources can be identified. The layout is taken from a back-propagation neuron.*