Neural network hardware performance criteria

van Keulen, Edwin

Award date:
1993

Neural Network Hardware Performance Criteria

Edwin van Keulen

August 1993

Advisors: Dr.ir. J.A. Hegt — Eindhoven University of Technology
Dr. S.B. Colak — Philips Research Laboratories

Abstract

For many real-world problems where neural networks are used, time constraints make hardware implementation indispensable. This is simply due to the fact that only hardware implementations can fully utilize the benefits coming from the inherent parallelism in neural networks. Our study start with a brief overview of the existing important chips for neural networks from major companies in electronic chip manufactoring shows the enormous diversity among them. Both for this fact and the fact that there is little consensus on the term *performance*, comparing these chips and choosing one for an application becomes a very difficult task. This work will contribute to that task by generating a new set of criteria which represents the functionality of a neural network chip in a more accurate fashion than the existing criteria.

In the main part of the present work, first meta-criteria are proposed as guidelines to come up with a set of *good* criteria. These meta-criteria demand both generality and practicality of a hardware performance criterium. Then, several commonly used conventional criteria are evaluated according to the proposed meta-criteria. The ineffectiveness of these existing criteria are revealed, from which it can be concluded that there is need for additional hardware performance criteria to make better comparisons possible. We, next, study theoretical aspects of a few new criteria, which are primarily related to the capacity of a neural network. We find that such criteria are often intractable or too weak to say something about performance of *hardware*, but they can be used as a starting point for more hardware related criteria. Several new hardware performance criteria are proposed, such as a ``Reconfigurability Number`` that says something about the size and reconfigurability of a chip. Also, we propose a new speed criterium which is normalized to the effective accuracy of a connection: ``Effective Connection Primitives Per Second``. Finally, experiments with the Intel ETANN chip are reported, where some of the proposed criteria are put into practice. All the mentioned new criteria, together, provide a set which is much more effective than the existing conventional criteria for evaluation of the performance of a neural network chip.

Table of contents

1. Introduction

Recently, there has been great interest in studying adaptable parallel signal processors called neural networks. Software implementation of neural networks on conventional hardware is far from efficient because it does not make use of the inherent parallelism of the network. So hardware implementation will be needed to fully utilize the benefits of neural networks. Choosing a specific hardware implementation for an apllication implies comparison of those chips. However, only little *good* criteria are present, which makes *fair* comparison difficult. This work will be aimed at the constructing of a better, more complete set of criteria that makes better comparison possible. Therefore, first a short explanation will be given of what exactly a neural network is.

1.1 What is a Neural Network ?

Neural networks are model-free estimators

Globally the world problems can be divided into several categories. One of those is the category of problems that can be explicitly and analytically solved. Often only simple problems, for example those in which linear algebra can be applied, can be solved in that way. For other problems that are too complex analytically or inherently unsolvable, one can often develop rules or heuristics in order to find an optimal solution. One of such problems is the fitting of data to an empirical model. The effort here is to make a proper model of the relation between the input and output data, with a limited number of free parameters. In this case examples of data points are required as instances of the desired optimal solution, which will be used to fill in the values of the free parameters during the *fitting of the data* to the model.

In the fitting procedure the parameters are chosen according to built-in rules or heuristics in a way that a previously defined error-criterium is minimized. By definition the optimal solution will be reached at the point where the values of the parameters are chosen to make the error-criterium minimal.

At one extreme the model can have no free parameters, in fact the model is exact. This coincides with the analytical way of solving problems. At the other extreme the model can have an infinite number of free parameters to be filled in. In this case there is no built-in knowledge of the problem at all, and therefore statisticians call this method *model-free* or *non-parametric* estimation. Neural networks can be viewed from this point of view as tending towards the model-free estimators, which can be explained by looking at the basic architecture of a neural network.

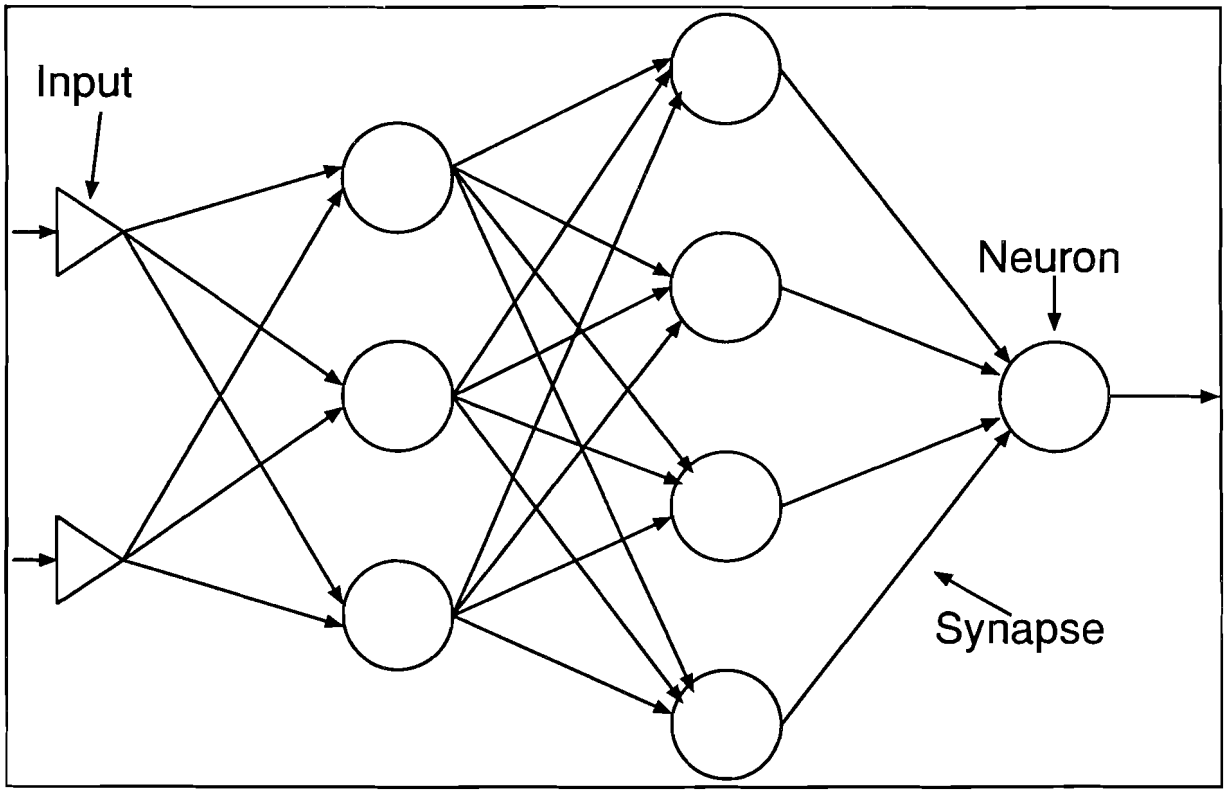

Figure 1.1: The architecture of a three layer feedforward neural network

The basic architecture of a neural network

A neural network is composed of an arbitrary number of *neurons* connected to each other by *synapses*. Often neural networks are subdivided by looking at the way the neurons are connected, the *architecture* or *topology*. If the data is flowing only from input to output, the neural network is called a feedforward neural network. Although in strict sense it is not a requirement, almost all feedforward neural networks can be decomposed in *layers* of several neurons. Also in most cases feedforward neural networks are fully connected in the sense that each neuron in one layer is connected to all of the neurons in the next layer. In this case data is supplied to the input layer and after it has been transferred through all the hidden layers it is available at the neurons of the output layer. Figure 1.1 depicts an ordinary feedforward architecture. Because of the fact that in the input layer no processing is performed, it will not be counted as a real layer. So if a neural network has only an input and an output layer it is called a one-layer neural network. Another kind of neural network is the class of recurrent or feedback neural networks, in which there are some connections in backward direction, from output to input. One example of this is the Hopfield neural network in which all neurons are connected to each other and serve both as input and output neurons.

So what are the functions performed by the neurons and synapses ? A synapse is a device in which an *input* x_i is multiplied by a *weight* w_i. This will be represented by

labelling the connections by their weights. In the neuron the contributions of all synapses are added together forming a weighted sum of the inputs, or in other words the inner product of the input vector $\underline{x}$ and weight vector $\underline{w}$. An example of a simple d-input 1- layer 1-neuron neural network is given in figure 1.2.

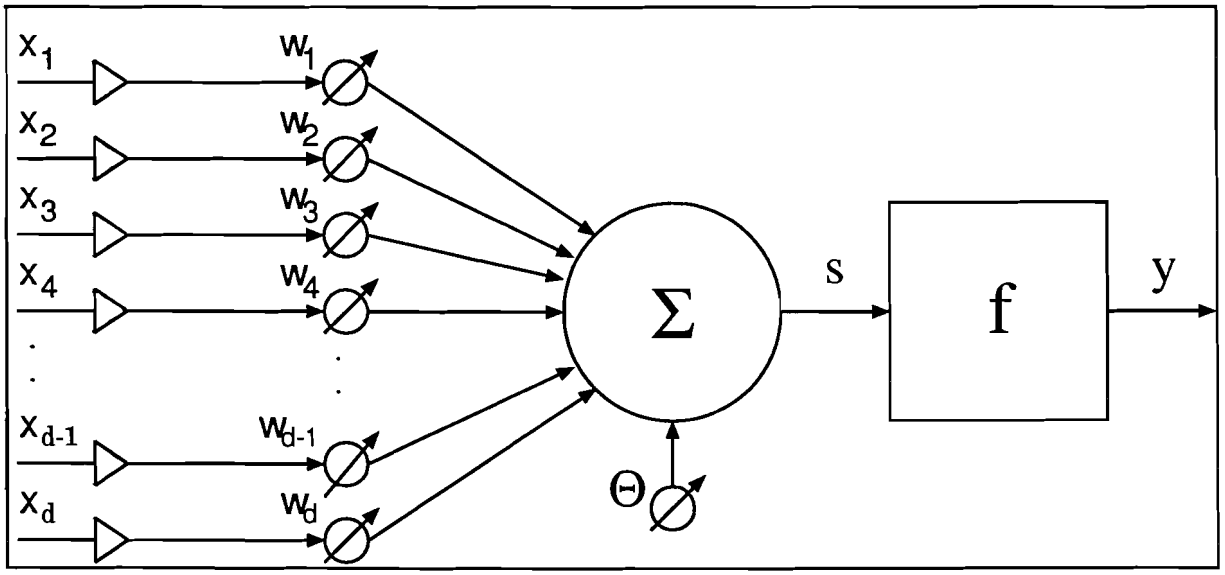

Figure 1.2: The architecture of a d-input 1-output neural network

The various contributions are added and supplied to a non linear saturating function, which is often chosen to be tanh(s), sgn(s) or $1/(1+e^{-s})$, in which s is given by (1).

$$s=\sum_{l=1}^{d} x_l w_l+\theta \qquad (1)$$

This model of neurons and synapses has been inspired by the elements of biological neural networks, such as the brain. Apart from the borrowed terminology there seems to be only little resemblance between biological and these simple artificial neural networks. By adjusting the values of the weights, different non-linear mappings can be made. A learning algorithm is used to adapt the weights of the network in a way that on the average given an input $\underline{x}$, the output $\underline{y}$ of the network is as close as possible to a desired output, or *target* $\underline{t}$. Therefore, the learning algorithm or training algorithm has access to a *training set* T of size M, in which $T = \{(\underline{x}_1,\underline{t}_1),(\underline{x}_2,\underline{t}_2),....,(\underline{x}_M,\underline{t}_M)\}$. Its task is to adapt the weights in order to minimize a predefined error-criterium. Often the error criterium is chosen to be the mean-square-error over the training set T. For a network with N outputs, the mean-square-error is given by (2).

$$MSE_T = \frac{1}{2M}\sum_{l=1}^{M} E(\underline{t}_l, \underline{y}_l) \quad \text{where} \quad E(\underline{t}_l, \underline{y}_l) = \frac{1}{N}\sum_{o=1}^{N} (t_{l,o} - y_{l,o})^2 \qquad (2)$$

Since $\underline{y}$ depends on the value of the weights, the mean-square-error can be minimized by adjusting the weights. Error-backpropagation [58] is a popular learning algorithm that can be used for this purpose. It adjusts the weights in the opposite direction of the gradient of the error surface in weight space. This means that in each iteration the update of the weight w is proportional to $-\partial E/\partial w$, in a way that this derivative tends to zero, yielding a minimum of the error surface in weight space.
When there are no restrictions on the number of hidden neurons, a two layer feedforward neural network can make arbitrary continuous mappings from input space to output space. This argument is often used when using neural networks as model-free estimators, however in practical situations the probability that one can find a proper set of weights that realizes this mapping depends heavily on the learning algorithm that is used, the number of training examples, the size of the network and the complexity of the desired mapping. As can be seen from (2), the mean-square-error also depends on the particular choice of the input-target examples, the training set. In general this is not preferable because good performance is also expected on inputs that were not in the training set. In fact this is one of the main reasons to use a neural network and it is called generalization performance. If generalization performance was not required, storing the training set in a database would give better performance on that training set. So exactly this generalization performance gives the predicting power of a neural network and is the main reason to use one. To measure generalization, the mean-square-error on a so called *test set* is calculated. If the error on the test set does not differ much from the error on the training set, the network is said to generalize well. Better even is to evaluate the learned network with multiple test sets and to calculate the mean and the standard deviation of the mean-square-error over the test sets [1]. Once convinced of using a neural network to solve a problem, the implementation in hardware becomes very beneficial.

1.2 Why hardware implementation ?

As shown if figure 1.2, a neural network performs *connections* or *multiply-and-add operations* that can be done in parallel. If however a neural network is implemented on a Von Neumann computer in software, all these operations are done sequentially, making it a very laborious task. So software neural networks, running on these sequential machines, do not make use of the *inherent parallel architecture* of a neural network. However hardware implementations do, making them orders of magnitude faster than their software equivalents. As for many problems, also for neural networks speed requirements

make hardware implementation a necessity, but especially for neural networks the gain in speed is very high if a hardware implementation is used.
Another argument in favour of hardware implementation of neural networks is their *inherent fault tolerance*. It has always been a nuisance of hardware implementations that no guarantees can be given that the system will be free of fabrication errors, especially for Very Large Scale Integration (VLSI) of hardware. In fact the required testability for VLSI circuits is becoming one of the main limiting factors of the integration density, because often one single error on a chip can make it totally useless. Only if correct functionality of every part of the chip can be established, correct functionality of the whole chip can be established. In the case of neural networks it has been shown [2] that they exhibit *graceful degradation* of the functionality of the whole network, which means that if in a neural network one neuron malfunctions, the functionality of the whole network only degrades partially. This makes the need to test every neuron in the circuit separately less stringent with increasing size of the integrated network, so it is unlikely that testability will become a limiting factor for the density of integration.
A third more practical reason in favour of implementing a neural network in hardware is the possibility of using a relatively cheap hardware neural network in embedded systems as vacuum-cleaners or washing machines.

1.3 How to compare neural network chips

Once chosen for the use of hardware implementation for a neural network in order to gain speed at a cost of software flexibility, the next problem to solve is how to choose among the available chips which is the most suitable one for your application. This question will be addressed in this work. Therefore, here, both commonly used criteria for neural network chips and additional criteria, as needed for further evaluation, will be investigated.

First, neural network chips of major companies in electronics will be examined, introducing the topics under investigation and giving some concrete examples. Next guidelines, which will be called meta-criteria, will be proposed for neural network hardware implementation criteria, trying to answer the question: What is a good hardware performance criterium ? Then some traditional criteria will be studied and judged according to the earlier proposed meta-criteria. In the next chapter some additional criteria will be proposed and explained. Finally, some experiments using the ETANN chip will be recorded of submitting it to some of the proposed new criteria. This work will be concluded by critically questioning how far we have come from having a proper, complete and sufficiently powerful set of criteria to judge hardware implementations of neural networks.

2. Chips for neural networks

2.1 From software to hardware neural networks

A major part of the neural networks running today are implemented in software on a conventional computer. This is due to several reasons. First of all, software gives maximum flexibility. In many cases where neural networks are being used, it is not clear whether the problem can be adequately solved using a neural network, and if it is, often it is not clear what kind of network and what kind of learning algorithm are most favourable. So much research comes down to mere trial and error, for which software is the ultimate tool. In software one has access to all data and any learning algorithm can be tried. Moreover no expensive investments are required to be able to try out a neural network.
The only investment to be made in software neural networks is patience, and for many real world problems (or rather real time problems) this is just what one does not have. It is of little use to have an optical character recognizer for postal sorting, that does one character per minute. A first solution for this is to speed up the software program by writing it for a series of Digital Signal Processors (DSP) connected to the host computer, a so called accelerator-board. This is a good solution if time constraints are not too heavy, because no offer in flexibility is required [3]. However since in many problems time constraints are more severe, dedicated neural hardware implementation becomes the only alternative at the cost of flexibility.
In this chapter most common and advanced chips for neural networks will be reviewed to give concrete examples of chips that have to be compared in order to establish their suitability for a given apllication. These examples then will be used in following chapters where already existing criteria will be evaluated and new will be proposed. Chips will be reviewed from digital DSP like neurochips to analog application dedicated ones.

2.2 Digital chips for neural networks

The **Connected Network of Adaptive ProcessorS** (CNAPS) chip of Adaptive Solutions and Inova [4--8] looks very much like the DSP solution. It is a chip consisting of 64 parallel Processor Nodes (PNs) connected by 8 bit broadcast input and 8 bit broadcast output busses. A 31 bit control bus is shared among the PNs to control their operation. Using broadcast busses the designers circumvented the problem of needing many output lines, at a cost of time. The input and output busses are used in a Time Division Multiplexed (TDM) way, allowing N connections per time step, for in each time step one PN can

broadcast its output to N other PNs. Here N can grow larger than 64 by cascading several CNAPS chips, connecting their busses. An external sequencer chip controls bus arbitrage via the control bus. Each processor node consists of an adder, multiplier, a logical unit, 32 16-bit registers, bus-drivers and 4 kB internal weight memory, making the chip sufficiently flexible to implement any learning algorithm, which is the major strength of this chip. A weakness is that in the case of many chips connected together, PNs spend more time waiting for bus availability then for computing, making the total system prohibitively slow.

The **Lneuro 1.0** chip from Laboratoire d'Electroniques Philips [9--10] is a general purpose LEGO-like building block processor. It has the possibility of performing 16 operations in parallel on 16 Processing Elements (PEs). Because of the 16 bit resolution of these PEs, by using only a part of this for the neuron state, several neurons can be implemented on one PE. Since the sigmoid function is calculated off-chip, effectively, each cycle only one neuron output is available. In total there is 1 kB on chip weight memory. Connecting several Lneuro's on a board controlled by a host-transputer makes a powerful accelerator board for general purpose computers. Here also a trade-off is possible between flexibility and speed. Though strictly there is no learning algorithm on-chip, because of its general-purpose architecture, off-chip learning algorithms can make use of the chips resources.

The **Neural Bit-Slice computing element** (NBS) chip of Micro Devices [11--12] is in many aspects similar to Lneuro, because the same design considerations were made. One NBS chip implements 8 neurons without synaptic weights, so external (read slow) memory is needed to provide the weight data. Again by a broadcast bus multiple NBSs can be connected exchanging speed for size in a TDM way. There is no learning on-chip.

The **Polyhedric Discrimination Neuron** (PDN) neural network from the NTT LSI Laboratories [13], implements a 64-input 13-output architecture, that calculates fully in parallel. Instead of calculating the inner product $\underline{x}\cdot\underline{w}$, the L_1-norm is adopted: $\Sigma_i |x_i-w_i|$, which makes expensive multiplication become superfluous. Another feature of the chip is the so called low-power-chain-reaction architecture, which is done without clock signals and bus-drivers cutting down power consumption. Also calculation is stopped whenever the neuron output is saturated (this can be done because for every input the L_1 norm can only grow) saving power and time. A drawback of course is its fixed architecture and the unconventional way of calculating the output. As a special purpose chip for low power consumption systems this however seems to be a suitable chip. Weight updates must be calculated off-chip.

The **Wafer Scale Integration** (WSI) neural network of Hitachi [14--15] implements 576 digital neurons on a 5 inch wafer, which is much more than the typically few neurons that can be implemented on one digital chip. Although generally chips can be connected to make a large network, the problem of making the appropriate number of interconnections is bigger between chips than within chips because of the limited number of possible pins on a chip. This is exactly why the broadcast approach is so popular, but the disadvantage of this is the fact that speed degrades with increasing size. An answer for this problem came from the Hitachi corporation: Implementing a neural network on a wafer makes more and faster interconnections between chips possible. A wafer consists of 49 neuron chips, of which one is a spare chip. Each chip has 12 neurons making the total of 576 neurons on a wafer. The neurons are interconnected by a hierarchical bus structure. The fault-tolerance argument is used to relieve the fact that within a wafer some chips may be not working. Since each neuron has memory for 64 weights, a maximum of 64 connections per neuron are possible. No learning is on-wafer. Because of the wafer-scale integration, large networks can be made faster than by implementing them on cascaded building block chips.

The newest version of this wafer scale integration neural network includes on-chip back-propagation learning (WSI-BP) [16--17]. Because extra logic and memory were needed in each neuron to be able to implement the backprop algorithm, the number of neurons dropped to 288. However for learning, separate neurons are needed for the feedforward and the back-propagation phase, realizing effectively only 144 neurons on a wafer. Still, learning on-chip is a significant improvement if the neural network has to be able to adapt quickly to a changing environment. Only recently, learning has been implemented on systems large enough to become interesting for real-world problems. This is one of the first serious attempts in this direction.

Some recent publications show that significant progress has been made in the integration of larger digital neural networks. Hitachi's **1.5-V 10^6-Synapse Neural Network** (10^6-S NN) [18] is an example of this. Because pulse-stream techniques use only a very small area for synapses, these are used to attain this large number of synapses. However, the effective speed per synapse is not so high, and there is no learning on-chip. The Intel corporation recently announced to have a digital learning neural network in its test phase calling it **Ni1000** [57]. This chip has 1024 neurons, on-chip Reduced Coulomb Energy learning [57] and 256k 5-bit synapses.

2.3 Analog chips for neural networks

With their **TDM Neurochip** Fujitsu [19] claimed to have the first commercial neurochip. In a mixed Bipolar-CMOS technology, it implements one neuron consisting of a multiplying D/A converter, an adder and a sigmoid generation circuit. Weights have to be supplied from external RAM and get multiplied with analog input by the multiplying D/A converter. The resulting value is stored as charge on an external capacitor. The sigmoid function is provided by six Bipolar differential amplifiers. Useful operation requires multiple chips connected by TDM busses like in many digital chips.

Because of a higher level of integration the **Electrically Trainable Analog Neural Network** (ETANN) of the Intel corporation [20--22] is considered to be the first successful commercial neural network chip. 64 Neurons together with 10,240 synapses are implemented with the possibility to make two layers of size 64x64. Therefore, two synapse-arrays are present, which both consist of 4096 normal weights and 64·16=1024 bias weights. There are 16 bias weights per neuron to give the bias sufficient influence in the sum (1). In two layer operation the outputs of the 64 neurons in the first layer are stored in buffers, after which they are fed through the second synapse-array. Then the same neurons are used again for the second layer. An important characteristic is the fact that it uses EEPROM technology to store its weights. The advantage of this is that unlike many analog chips, ETANN does not need area consuming refresh circuits to retain the weightvalue. Another advantage is the non-volatilily of the weightvalue. A disadvantage is the long weight update time, caused by the Fowler-Nordheim tunnelling process involved. Synapse multiplication is performed by a simplified differential Gilbert-Multiplier, providing four quadrant multiplication. Summing is done by a simple summing wire, collecting the differential output-currents from the multipliers. 64 Sigmoid circuits seperately provide the sigmoid functions. Because of its all-analog operation an Intel Neural Network Training System (iNNTS) has been developed, including control logic, D/A and A/D converters making the chip accessible by a PC-system. Together with software routines to control the iNNTS, Intel provides a whole platform of support necessary for commercial success. It must be noted that because of the EEPROM storage of the weights, the chip is not suitable for applications where it has to be reprogrammed frequently. Not only because of the long programming time that is involved, but also because of the limited number of times that a cell can be reprogrammed before it degenerates. Although no learning is on-chip, ETANN does make the implementation of the Madeline [58] learning algorithms easier because of the easy way to perturb the neuron inputs by drawing some current from the summing wires. The architecture of this particular chip as well as its performance will be investigated further in one of the following chapters, where it will be submitted to some original neural network hardware performance criteria.

The **BiCMOS Analog Neural Network** (BiCMOS ANN) of the Matsushita Electronic Research Laboratory [23] is a fixed architecture 32x16x16 neural network chip. Weight memory is implemented by dynamically refreshed capacitors like in DRAMs. Multiplication output is buffered and added around an operational amplifier, after which a saturating differential amplifier applies the sigmoid operation. With these circuits Matsushita made a chip with rather precise operations preventing it from having a large number of synapses. Learning has to be done off-chip.

The **Reconfigurable Neural Net Chip with 32k Connections** (NET32K) of the AT&T company [24--25] is a collection of 256 'building block' neurons. In one block, 128 1x1-bit multiplications can be performed in parallel. A comparator provides a hard-limiting function. One feature of this chip is that several blocks can be combined into a block of a certain (up to 4 bit) analog depth. This is done by multiplying the currents from each block with different weights (1 1/2 1/4 and 1/8) before adding them together. By setting the references of the comparators at a different value also up to 4 bit analog output can be calculated. Using this approach carry bits are not needed, so the analog multiplication can be done fully in parallel. Configuration registers provide the reconfiguration possibility. Because of the digital access to the chip, digital system integration is straightforward. No learning has been implemented on-chip.

Another chip from AT&T is the **Artificial Neural Network ALU** (ANNA) [26--29]. This chip implements 8 neurons together with 512 physical synapses and 4096 weights. In one clockcycle 8 vectormultipliers can each multiply a 3 bit inputvector and a 6 bit weightvector of dimension 64, after which the value can be multiplexed to one of the eight available neurons. This is done four times in each calculation making a total of 256 inputs per neuron possible. Also many configurations with less inputs per neuron are possible without loss of parallel performance. The outputs of the vectormultipliers can be shifted up to 3 bits to enhance the dynamic range of the neurons. Since the weightvalues are charges on capacitors, they have to be refreshed by on-chip D/A converters. Despite its reconfigurability, in many practical situations the topology of the network implemented often does not make full use of the chips parallelism, effectively making it slower [29]. Also here, digital access makes system integration relatively easy. Weight updates must be calculated off-chip.

The **Neural Network with Branch-Neuron-Unit Architecture** (336-BNU) from the Mitsubishi Electric Corporation [30--31] is a highly scalable Bolzmann Machine. The number of branch-neuron-units, which consist of a few synapses together with a piece of neuron, scales with the number of chips that are connected together, so one neuron unit always

has to drive the same load capacitance making the speed constant in the number of connected chips. Within one chip each of the 336 neurons drives 84 synapses making a total of 28k synapses. Learning is done according to the Bolzmann algorithm, and is performed every time the weightvalues, stored on capacitors, begin to degrade.

In spite of the increased number of synapses (40.000) and neurons (400), the next version of this chip (400-BNU) [32] has refreshable synapses to overcome this relearning necessity. Since all synapses are refreshed in parallel the refreshtime is proportional to the number of memorized patterns, which is proportional to the square-root of the number of synapses. This makes the implementation of large networks advantageous. However, for both chips it can be argued that the implementation of a Bolzmann machine restricts them to only a limited application area.

The **2-Chip Set with on-chip learning** (2-Chip Set) of the Toshiba corporation [33] has on-chip Backpropagation or Hebbian learning. One of the two chips implements 24 neurons, and the other 576 synapses in 9 groups of 64. Each group has its own learning control circuit and the size of these groups was determined by speed/size tradeoffs. Because of the limited drive force of one neuron a maximum of 20 synapse chips can be driven by one neuron chip.

Recently the Korea Telecom Research Centre reported a pulse operation chip with as much as 135.424 synapses called the **Universally Reconstructable Artificial Neural-network** (URAN) [34]. It consists of 16 modules of each 92x92 synapses in pulse-stream mode making asynchronous wired-OR expansion possible. Because of this, expansion can be made without considering timing or load-capacitance constraints.

Apart from all these chips, also much research has been done on analog special-purpose or Application Specific Integrated Circuits (ASIC). The researchers at the Californian Institute of Technology became world leaders in this area, by making silicon cochleas [35] and retinas [36]. Analog technology is very suitable for ASICs because the main drawback of analog circuitry, little flexibility, is not considered to be a nuisance here. Since only performance on a specific application is required for these chips, comparison with other chips is pointless, so they are left out of the scope of this thesis.

3. Hardware performance meta-criteria

In the previous chapter, a versatile range of chips for neural networks have been reviewed. All a product of their designers' compromising considerations between technology, resources and performance demands. A reason for the diversity of chips is the mere ignorance of making such compromising decisions on the *performance* of the chip.
On the one hand, this is because of the fact that the technology of neural network hardware is relatively new. Therefore there is only little experience in making neural network chips. First many possible implementations must be tried out before such experience is gained. On the other hand, there is little consensus on what will be a good definition of performance. Comparing and deciding in favour of one approach therefore becomes a very difficult task. So first there must be some standard, accepted set of *criteria*, along with standardized measurement methods to measure performances in terms of these criteria before *fair* or *unbiased* comparison can be done. This work will contribute mainly to the first stage: the definition of a proper set of criteria. Needed for this is a set of guidelines for those criteria, which will be called *meta-criteria*, to be able to compare the hardware performance criteria.

1) A hardware performance criterium must be sufficiently general

With this it is meant that the criteria must apply to the whole range of possible hardware neural networks; digital, analog, hybrid and preferably also software neural networks to make the range of comparison as broad as possible.

2) A hardware performance criterium must be of sufficiently high conceptual level

A user of neural networks is interested of the performance of the chip on his/her particular problem, in terms of speed, learnability, generalization ability rather than the fact that his/her chip has eight or three bits for its weights.

3) A hardware performance criterium must be problem independent

Since we talk about general purpose neural network chips, they are supposed to perform on many different problems. Good performance on one problem does not imply good performance on the other, making it very difficult to generalize from a criterium that is problem dependent [37--38]. Still benchmarking, as this is called, is a popular way of

testing performances of systems in general. Benchmarking will only be acceptable if the investigated aspects are independent of the particular problem, or at least these dependencies should be known. This does not seem to hold for neural networks, because of the little experience with them. One could construct a *typical set* of problems representative for the whole universe of problems neural networks can be applied to. However this requires recording of all the experiments with neural networks trying to find such typical set, so for the moment it is not clear in what way a benchmark says something about the performance of a chip.

4) A hardware performance criterium must have sufficient distinctive power.

Undoubtedly a criterium must be capable of making distinction between different hardware approaches. If a certain criterium is so weak as to yield the same numbers for many intuitively different chips, it fails to distinguish between them and therefore does not reveal much information about the chips.

5) A hardware performance criterium must be obtainable

This seems a rather trivial demand, but one example of an interesting non-obtainable criterium would be: The averaged mean-square-error over all possible problems that can be mapped on the chip, after learning with all possible learning algorithms with infinite training sets. Although this is a rather extreme example, still, constant care should be taken to be able to satisfy this demand.

Looking at these meta-criteria, they can be divided globally into two contradicting classes. Meta-criteria 1,2 and 3 all propose some kind of *generality*. 1) Demands generality towards the chips that can be compared. 2) Demands a high level of abstraction and 3) demands generality towards the problems. The danger of generality or abstraction is loss of information, which exactly contradicts the demands of the criteria in the second class 4) and 5). 4) Demands that there must be sufficient information left to be able to make distinction between chips. 5) Says that the criterium must not become so general that it becomes intractable. So 4) and 5) demand *practicality*. Clearly these two contradicting classes imply compromising decisions to be taken when coming up with hardware performance criteria. This is also why *good* criteria are so hard to find, because criteria that are good in the sense of 1) 2) and 3) are likely to be bad in the sense of 4) and 5) and vice versa. In the following chapters traditional hardware performance criteria will be evaluated and new criteria will be proposed according to these 5 meta-criteria.

4. Conventional hardware performance criteria

Several aspects play a role in the performance of a neural network chip. First of all, it is important exactly *what* a chip can do in terms of its absolute architectural limitations, and how accurately this can be done. If this is known, the *speed* of operation is very interesting, for this was one of the main reasons to use hardware in the first place. Finally we want to have an indication of the *cost* of using the chip to make sensible cost/performance trade-off possible. For all these areas of interest, several criteria have been proposed in literature. While reviewing them, they will be evaluated according to the meta-criteria of chapter 3, using the chips of chapter 2 as examples. All the numbers of these criteria will be given in appendix 1.

4.1 Criteria for size and accuracy of the chip

A constraint that always must be satisfied when using a neural network chip is that the neural network to be implemented by hardware must be mappable on the chip. Not only *topological constraints* are important here but also whether any analog depth or *accuracy* is demanded and whether *learning* must be on-chip. In previous work similar to this the following criteria have been proposed [39--42].

The **Number of neurons per chip** (N_n) and the **Number of synapses per chip** (N_s) clearly give an idea of the size of the network that can be implemented. The advantages of these criteria are clearly the generality of them and in fact meta-criteria 1), 2) and 3) are well satisfied for these and 5) too. However 4) is not satisfied because these numbers can not distinguish between chips with the same numbers N_n and N_s that have their neurons and synapses connected to each other in a different manner, which is equally important to judge if a given neural network can be mapped on the chip. Also the fact that some chips multiplex their synapses in time can not be reflected by these criteria. To meet this last drawback also the **Number of weights per chip** (N_w) is reported, counting not only the physical synapses (=multipliers) but rather the number of weights that can be implemented. Also the **Number of Connections per Neuron** (CPN) (read weights per neuron) are reported giving a slightly better idea of the possible topologies of the chip. A disadvantage of just counting the number of them is that accurate and non accurate neurons/synapses/weights are counted as equal, while intuitively it is clear that a 8x8-bit synapse does a lot more than a 1x1-bit synapse. Only the **Number of bits per weight** (b_w) and the **Number of bits per input** (b_x) are given which do satisfy 1) 3) 4) 5) , but completely fail to contribute to 2) because the effect of these accuracies are not well known yet.

Moreover this is not sufficient because there are accurate and less accurate 8x8-bit multipliers, so the precision of the multiplication and the precision of the neuron function should also be regarded. To be able to satisfy 2) to a larger extent, the effect of all these kind of errors (on the outputs of the chip) must be known.
For the learning part (if there is one) it might be interesting which **Learning Algorithm** (LA) can be implemented on the chip, again lacking to say anything about the accuracies of the implemented algorithms.

4.2 Criteria for speed of the chip

The most commonly mentioned criterium for hardware performance is the number of **Connections Per Second** (CPS) a chip performs, meaning the number of multiply-and-accumulate operations per second. Although this fits very well to the intuitive idea of speed for a neural network (in fact 1) 2) 3) and 5) are satisfied) it again completely fails to discriminate between chips with high and low accuracy. To compare the speed of the NET32K chip and the CNAPS chip just by their CPS value seems to be biased in favour of the NET32K chip for the following simple reason: CNAPS does a lot more in one connection than NET32K does, so it is not fair to count them as equal. Somehow again, the computational accuracy must weight the extent to which a connection is counted.
Another disadvantage of plain CPS is the fact that it does not say anything about the speed of one particular connection because a chip with many parallel connections reaches a given value of CPS with much slower connections than a chip with a little number of connection does. This can be solved by normalizing the CPS value to the number of weights of the chip yielding **Connections Per Second Per Weight** (CPSPW), which seems better than dividing by the number of synapses, because in this way time division multiplexed synapses can be included. Also this criterium gives an idea of the ratio between number of weights and speed, which should in some way be balanced to the accuracy of the chip, as considered in [42]. Of course this balance is still unclear if the accuracy is not somehow accounted for and precisely this it where these criteria fall short of.

For the learning phase there are **Connection Updates Per Second** (CUPS) and **Connection Updates Per Second Per Weight** (CUPSPW). For these the same arguments hold as for the first criteria: they lack to account for the accuracy of operation thus fail to satisfy 4) to a reasonable extent.

4.3 Criteria for the costs of the chip

Since numbers like dollar per connection are hard to obtain and too much depend on various external condition, costs of hardware are generally given in terms of size of the package, number of pins and power consumption. Because only few of the reviewed chips are in their commercial stage, such numbers are rarely given. In stead related numbers such as the area of a synapse as well as the area of a neuron are mentioned. Because of the fact that generally there are a lot more synapses than neurons on a chip consuming most of the chips die area, sometimes the **Effective Synapse Area** (ESA) of a synapse is given by dividing the total area of the chip by the number of synapses or by the nurnber of weights. This second option includes the time division virtual synapses, and therefore this one will be adopted.
Power Dissipation (PD) is given as the maximum power dissipation at a certain clock frequency. By dividing this figure by the number of connections in one second (CPS) at the same frequency the resulting quantity is **Energy Per Connection** (EPC), giving an idea of the economy of the chips calculations, again this value should somehow be weighted by the accuracy of such connection.
Not always the figures are given for these quantities, because they involve measurement; but as can be seen from appendix 1 there is a link between technology, area and power consumption, so the technology given will be a good direction in what order of magnitude these values will be.

Of course, non of the mentioned criteria are designed to satisfy the mentioned meta-criteria, and moreover no criterium can ever satisfy all to a maximum extent. For example there always will be aspects of a chip that can not be distinguished under a given criterium, so for those aspects 4) will never hold. It can also be argued that the *combination* of the above mentioned criteria *does* satisfy the meta-criteria to a larger extent. Although both arguments are valid, in general one is interested in as little as possible criteria that intuitively fit as well as possible to the interesting characteristics of a chip. So although N_x,N_w and CPN together say something about size and possible ways to connect neurons, integrating them into a number that would immediately express whether a given neural network can be implemented on the chip, both reduces the number of criteria and fits better to the, for a user, interesting aspect of a chip. Also N_x , N_w and CPS together can distinguish between accurate and inaccurate connections in terms of speed, a number that would integrate these figures intuitively would give a much better idea of what that chip really does in one second.

5. New hardware performance criteria

As can be seen from the previous chapter, there is need for some additional criteria to be able to make a better selection among the available chips. Therefore, first, a review will be made of some fundamental neural network performance criteria known from theory. Since for these, several practical requirements are not satisfied, they only will be used as an inspiration to come up with hardware related criteria. Several new proposals will be made, such as a number that expresses the reconfigurability and size of a chip, a definition of capacity and a accuracy normalized speed criterium. Figures, from these newly proposed criteria, are given in appendix 2.

5.1 Fundamental neural network performance criteria

From a more theoretical, statistical and decision-theoretical point of view, neural networks are often placed into the more general context of statistical estimators or learning machines [1][43]. Within this framework several definitions of the term *capacity* are used to bound the number of training samples needed to give good learning and generalization performance. By looking at these definitions in more detail, it might be possible to extract some useful hints for a good definition of a hardware related form of capacity, as discussed in the following paragraph.

Learning and optimization

Consider the input space **X** and the output space **Y**. An unknown distribution P on $\mathbf{X}\times\mathbf{Y}$ determines the mapping between points $x\in\mathbf{X}$ and $y\in\mathbf{Y}$. Let $\mathcal{F}\subset\mathbf{X}\rightarrow\mathbf{Y}$ be the abstract representation of the set of all possible functions that can be implemented by a given neural network by varying its weights, and let $f_w\in\mathcal{F}$. The goal of learning is to find a point

$$Err(f_w) = E[(y-f_w(x))^2] \tag{3}$$

in weight space **W** and a corresponding f_w that minimizes the error function (3). Here E denotes the expectation over P. For a given x the expectation of this squared error is

$$\begin{aligned} E[(y-f_w(x))^2|x] &= E[((y-E[y|x])+(E[y|x]-f_w(x)))^2|x] \\ &= E[(y-E[y|x])^2|x] + 2E[y-E[y|x]|x]\cdot(E[y|x]-f_w(x)) + (E[y|x]-f_w(x))^2 \\ &= E[(y-E[y|x])^2|x] + 2(E[y|x]-E[y|x])\cdot(E[y|x]-f_w(x)) + (E[y|x]-f_w(x))^2 \\ &= E[(y-E[y|x])^2] + E[(f_w(x)-E[y|x])^2] \end{aligned} \tag{4}$$

known to consist of two terms: *variance* and *bias* as shown in (4). The first term at the right hand side of (4) is just the variance of y given x and is independent of f_w. This is the price we have to pay for trying to estimate the ambiguous P(y|x) by a deterministic function. So the optimal $f_w(x)$ is just the expected value of y given x, known as the *regression*. The second term measures in a natural way how far $f_w(x)$ is from that regression and therefore is a good way to evaluate the performance of $f_w(x)$ as a predictor of P(y|x). In many cases however, as for example in classification there is only one allowed value for y given x and then P is said to be *degenerate*. The variance term then becomes zero and the minimal error also becomes zero. If P would be known, (3) could be calculated as a function of w and then it could be tried to find a minimum. However, P is unknown and therefore we need a learning algorithm that chooses in a suitable way among the available functions in $\mathcal{F}$ to minimize (3).
Since the calculation of (3) requires integration over the whole input and output space, it is not computable and it can only be estimated by the average value over a finite training set as stated in (5).

$$\hat{Err}(f_w)=\frac{1}{M}\sum_{i=1}^{M}(y_i-f_w(x_i))^2 \tag{5}$$

For a fixed f_w the M numbers $(y_i-f_w(x_i))^2$ are identically independently distributed (i.i.d.) realisations of the random variable $(y-f_w(x))^2$. Therefore when M is growing to infinity the Law of Large Numbers (LLN) says that $\hat{Err}(f_w) \rightarrow Err(f_w)$.
Now let $f_{\hat{w}}$ minimize (5) and let f_{w^*} minimize (3) resulting in Err* under the assumption that f_{w^*} is in $\mathcal{F}$. To evaluate $f_{\hat{w}}$ as a minimizer of (3) in stead of f_{w^*}, $Err(f_{\hat{w}})$ must be evaluated.

$$\begin{aligned} Err(f_{\hat{w}}) &= E[(y-f_{\hat{w}}(x))^2] \approx \frac{1}{M}\sum_{i=1}^{M}(y_i-f_{\hat{w}}(x_i))^2 \\ &\leq \frac{1}{M}\sum_{i=1}^{M}(y_i-f_{w^*}(x_i))^2 \approx E[(y-f_{w^*}(x))^2] = Err^* \end{aligned} \tag{6}$$

Since by definition Err* is the minimum of (3), also $Err(f_{\hat{w}})$ can approximate this minimum arbitrarily well by enlarging M. This however does not necessarily mean that the bias term of (3) and (5) both go to zero. This requires that an f_w is included $\mathcal{F}$ that has $f_w(x) = E[y|x]$ for all x, because only then $f_{w^*}(x)$ equals E[y|x], making the bias terms go to zero and both Err* and $Err(f_{\hat{w}})$ go to their absolute minima. So it is important to have $\mathcal{F}$ large enough to include the regression. For a two layer neural network for which the number of hidden neurons can grow arbitrary large, it can be guaranteed that for every continuous P on $\mathbf{X}\times\mathbf{Y}$ the regression is in $\mathcal{F}$ [1].
Another problem comes from using the first time the LLN in (6). It *is* true that $\hat{Err}(f_w) \rightarrow Err(f_w)$ for *fixed* functions f_w as stated, but $f_{\hat{w}}$ depends on all the x_i's and y_i's, because it is

constructed by them and therefore the $(y_i-f_{\hat{w}}(x_i))^2$ are not i.i.d. anymore. Therefore the LLN may not hold anymore. To ensure LLN can be used normally it is explicitly demanded that (7) holds uniformly over all possible probability distributions P.

$$Pr(\exists f_w: |\hat{Err}(f_w) - Err(f_w)|>\epsilon)\leq\delta \qquad (7)$$

This statement is known as the *law of uniform convergence of empirical means to their expectations* and was at first pointed out by Vapnik [44]. Because this law needs to hold for every distribution, it really is some kind of worst case criterium, or *minimax* criterium. This as opposed to the *Bayesian* analysis that depends on specific information about the probability distribution.

Probably Approximately Correct learning

So in one way $\mathcal{F}$ must be large enough to include the regression, and in another way $\mathcal{F}$ must not be too large to be able to satisfy (7). While learning, freedoms in choosing among $\mathcal{F}$ are reduced and therefore the effective size of $\mathcal{F}$ is reduced. The more samples that are used, the less probable it will be that there exists an f_w for which its empirical mean error does not converge sufficiently to its expected value. So the larger $\mathcal{F}$ is the more samples are needed to satisfy (7), but on the other hand the higher the probability that the regression is in it. One learning method that explicitly uses this is the Probably Approximately Correct (PAC) learning method [45]. Within this method (7) is assured, by first beginning with a small $\mathcal{F}$ and then gradually increasing the size to eliminate all bias. Therefore the resulting f_w is probably (with probability 1-δ) approximately (not more than ε from being) correct. Much work has been done to estimate δ as a function of the *size* of the set of functions $\mathcal{F}$ and the number of samples M. It turns out that for a given required probability δ, the sample size needed grows proportionally to the logarithm of the size $|\mathcal{F}|$ [43]. This result is only useful if this size is finite, so instead of size a more general notion is used and this is called the *capacity* of the set of functions $\mathcal{F}$. For the case of finite size, the capacity can be defined to be the size, however for infinite sizes capacity can be defined in several other ways.

Capacity as epsilon-cover

In mathematics, if the size of an infinite but bounded set must be established, the method of making an ε-cover is used. Therefore the infinite set $\mathcal{F}$ is covered by a finite set of functions in $\mathcal{F}$ in a way that all functions in $\mathcal{F}$ have a *distance* less than ε from any of the functions in the cover. Distance between two functions f_{w1} and f_{w2} now is defined as the average according to P of the difference in error when using f_{w1} in stead of f_{w2} (8).
As this expectation is taken over a given probability distribution, a problem independent way of defining capacity is as the supremum of the size of the ε-covers over all possible

$$d(f_{w1},f_{w2}) = E[|(y-f_{w1}(x))^2 - (y-f_{w2}(x))^2|] \quad (8)$$

distributions P. Again δ can be expressed in terms of this capacity, and also for a given δ it has been proven that the number of samples needed, grows proportionally with the logarithm of this capacity [43]. Although there are some constructive results that bound the capacity in the case of neural networks [43], these results are rather weak in the sense that the effect of limitations of hardware such as limited accuracy or others on this capacity are not known theoretically. One result is that the capacity grows exponentially in the number of weights, and this agrees with the rule of thumb that the number of samples needed for proper generalisation of a neural network should be proportional to the number of weights [58].

It can be seen quite easily that measurement of capacity in this sense is intractable. For even if the requirement of including all possible distributions was dropped and exchanged for some statistic, for example the average over a large number of them, the computational complexity of determining a cover grows exponential in the dimensionality of the set. This dimensionality is in the case of a neural network the number of weights. Moreover just to determine one single distance of two functions, again averaging must be done over a large number of input-output samples.

Capacity as Vapnik Chervonenkis-dimension

An alternate definition of capacity that also can be used to bound (7) is the *Vapnik-Chervonenkis dimension* (VC-dimension) of a class of functions [46]. This VC-dimension happens to coincide with the intuitive notion of the capacity of a neural network as some sort of discrimination ability of the network in the case of classification. Although this work is based on binary valued functions, recently generalizations of the VC-dimension have been proposed to address this objection [43].

Let the input space $\mathbf{X} \subset \mathbb{R}^d$ and output space $\mathbf{Y} = \{0,1\}$. $\mathcal{F}$ is the set of functions from $\mathbf{X}$ onto $\mathbf{Y}$ for which the VC-dimension can be defined in the following way. First of all a set of N points in input space are said to be *shattered* by $\mathcal{F}$, if each of the 2^N possible mappings of dividing N points in the two classes {0,1} can be implemented by some function in $\mathcal{F}$. The VC-dimension is the size of the largest set of points in the input space that can be shattered by $\mathcal{F}$.

Knowledge of the VC-dimension makes estimates possible of the number of samples needed to achieve good generalisation [47]. The problem here once again is to establish this VC-dimension. The VC-dimension is known to be d+1 for a simple d-input, 1 output neural network with a hard-limiting function as in figure 1.2 [48]. However for more general architectures of networks this VC-dimension only can be bounded in terms of the number of weights, neurons and layers [60]. Furthermore it is not clear in what sense limitations

of hardware will effect the VC-dimension. For example when looking at the above stated simple case of a d-input 1-output neural network but now with weights restricted to have only b weight bits. The only bound that directly can be given from this is that the VC-dimension will be less than 2^{db}. This number however is in no proportion to the number that is given by the limits of the architecture d+1. On the other hand this does not mean that hardware limitations do not have any effect on the VC-dimension. Because of the limited input-accuracy there is, unlike in theory, always a finite number possible sets of N points that can be shattered. Then it is possible that there is only one set of points that has size N equal to the VC-dimension. The limited weight accuracy now can cause that this particular set can not be shattered anymore, which decreases the effective VC-dimension.

Straightforward measurement of the VC-dimension is intractable because not only an exponentially growing number of possible classifications have to be verified, also until such classification has been found, all possible functions (exponentially in the number of weights) must be tried out. Still there are some results of measurement of the VC-dimension [49--50], but these rely on the fact that for very simple networks the learning curve can be predicted by the VC-dimension. By fitting this experimentally achieved curve to the VC-dimension, this dimension can be measured in very simple cases. However for larger, more complex networks only crude bounds can be given for the mean-square-error [59]. The predictive value of the VC-dimension then becomes too small to be able to fit this curve to a function with the VC-dimension in it.
A classification benchmark that could be extracted from this is to statistically try to establish the percentage of all 2^N classifications that can be learned assuming some learning algorithm, training set size and number of allowable epochs, to be plotted against N. Of course then it remains to be seen whether good performance on random classifications predicts performance on other problems.

5.2 Hardware related performance criteria

Reconfigurability Number
Important for a neural network is that the relation between the input and output data that must be predicted is as good as possible in the class of functions that can be implemented by that neural network. It is the topology of a neural network that determines this class and therefore at this moment much research is done in order to find a suitable topology for a given problem. There are even learning algorithms that automate this process. They alter the topology while exploring the solution space. So although without an application no

topology can be said to be preferable the ability to change the topology makes a neural network suitable for more applications. This ability to change the topology of a neural network is called *reconfigurability.*
Several criteria say something about possible topologies, as the already known maximum number of neurons/weights and the maximum number of weights per neuron. However no number has been proposed to say something about how many *different* topologies can be implemented with a chip. To limit the scope only layered feedforward neural networks are included.
To do this the following will be defined.

- The *topology* of a M-layer feedforward neural network is the following M+1 tuple: $\{L_0,L_1,L_2,...,L_M\}$, in which L_0 denotes the input layer and L_1 t/m L_M the hidden and output layers. $L_0 = \{N_0\}$, where $N_0 \subset \mathbf{N}$ is the set of input buffers and for all $1 \le i \le M$: $L_i = \{N_i,S_i\}$. $N_i \subset \mathbf{N}$ denoting the set of neurons in layer i and $S_i \subset N_{i-1} \times N_i$ denoting the sets of synapses from inputs or neurons in layer i-1 to neurons in layer i. To prevent empty and unconnected layers: $N_0 \neq \emptyset$, and for all $1 \le i \le M$: $N_i \neq \emptyset$ and $S_i \neq \emptyset$. Furthermore since a path from every input to output is required, it will be demanded that for all input buffers $n_x \in N_0$ $path\_to\_output_0(n_x)$ holds.

$$\begin{aligned} path\_to\_output_i(n) &= \exists n' \in N_{i+1}: ((n,n') \in S_{i+1} \wedge path\_to\_output_{i+1}(n')) \\ path\_to\_output_M(n) &= TRUE \end{aligned} \qquad (9)$$

- A topology is *implementable* by a chip if and only if all the necessary calculations in order to calculate the output of the neural network that generates this topology can be performed by the chip under the constraint that the data may not leave the chip before the output is available. So the whole neural network must fit on the chip. This means for example that if a certain topology needs off-chip feedback loops it is not implementable in this sense.

- Two topologies are *equal* if and only if they can be constructed out of eachother by permutating the numbering of the neurons or inputs. Formally this defines an equivalence relationship resulting in partitions of the set of all topologies. Two topologies are *different* if and only if they are not equal.

- The *set of all different implementable topologies* of a chip is constructed out of the set of all topologies generated by any neural network by picking out of each equivalence class one of the topologies that is implementable by that chip , if there is one, and adding it to the set.

- The Feedforward **Reconfigurability Number** (RN) can now be defined as the size of the set of all different implementable topologies.

The difficult part here is to determine this set of all different implementable topologies. According to the definition it comes down to picking a topology and looking if it is implementable. Clearly the maximum number of neurons,weights and layers that can be implemented on the chip make the search space finite. But still the number of possible candidate topologies grows exponential in the number of maximal weights WMAX. For consider this finite search space and consider the maximal connected topology in this space. Then each of these weights has the two possibilities to be included in, or left out a candidate topology resulting in a maximum of 2^{WMAX} candidates. Checking if an equal candidate already has been tried out is $O((\#candidates)^2)$ also exponential in WMAX. Therefore finding the reconfigurability number in this sense becomes intractable. First lets give an example of all of the above. Suppose your chip can implement a feedforward neural network of maximal 1 layer, 2 inputs, 2 neurons and 4 weights. Then all of the possible candidate topologies are shown in figure 5.1.

Now it can be seen that putting the equal ones together yields the following classes of candidates: {{1,2,3,4},{5,6},{7,8},{9,10},{11,12,13,14},{15}}. Since all are implementable in this case RN=6 DiFs (**Different Feedforward networks**).

To severely limit the number of candidate topologies the following is defined:

- A *fully connected topology* is a topology that has the following constraint: For all neurons in the topology there must be a synapse from every neuron or input in the previous layer to that neuron, so formally

$$\forall 1 \leq i \leq M\text{: } \forall n \in N_i, n' \in N_{i-1}\text{: } (n,n') \in S_i \qquad (10)$$

- The fully connected feedforward **Reconfigurability Number** (RN) is the size of the set of all different implementable fully connected topologies.

So in the example the classes {5,6} and {11,12,13,14} are not fully connected so the RN = 4 DiFFs (**Different Fully connected Feedforward networks**). Since there is only one way to fully connect two layers of a given size, the number of candidates drops to order of maximum number of neurons square. To be more specific, a chip that has maxima of $|N_0|$ inputs, M layers of each maximum $|N_i|$ neurons, the maximum number of candidate fully connected topologies is $\Pi_i |N_i|$ for i from 0 to M, since for every layer only the number of neurons can be chosen.

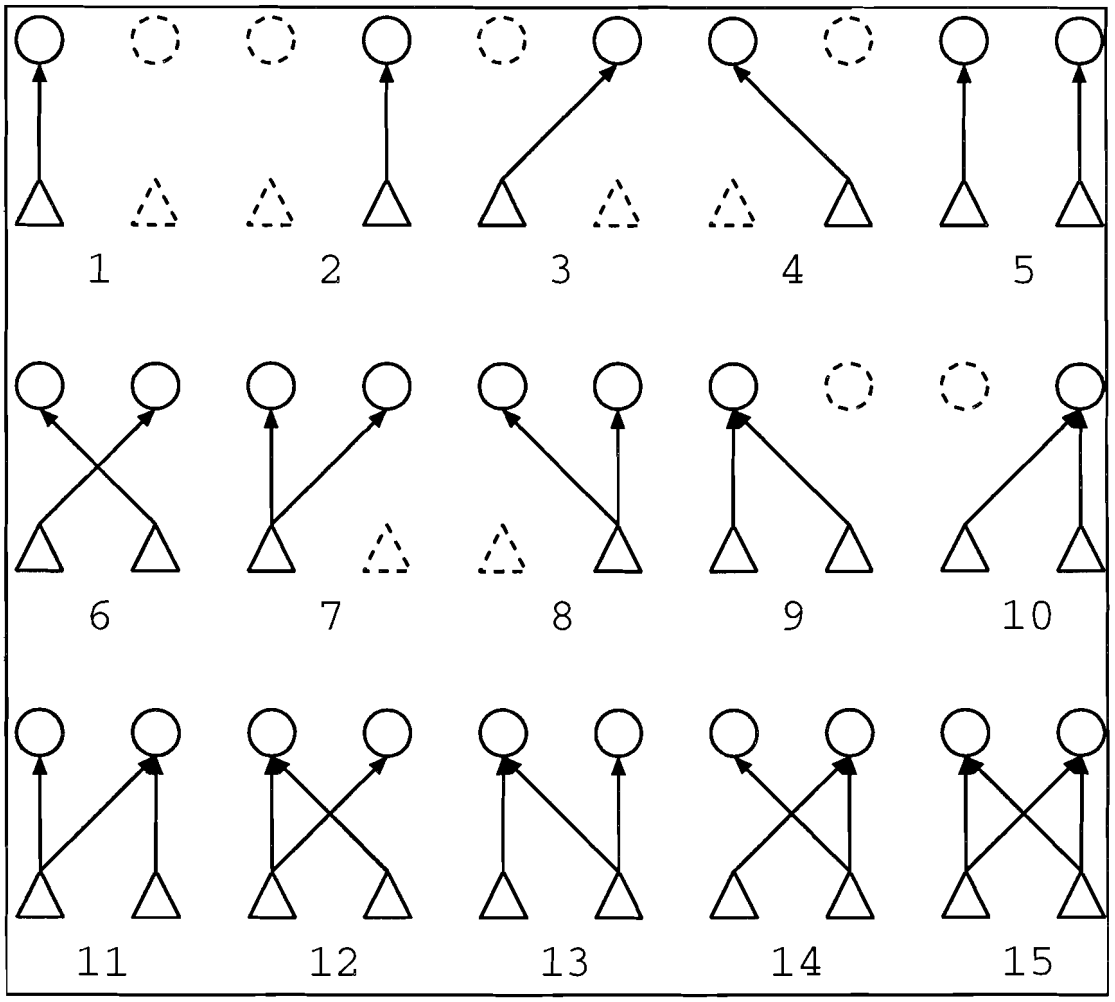

Figure 5.1: Set of all possible candidate topologies

Now lets give some examples of real chips. For the CNAPS chip, there is really only one layer of maximal 64 neurons and a maximum of 4K inputs limited by the 4kB weight memory. All the of the 4096·64=262,144 candidate topologies are implementable, so in 8-bit weight mode, the RN = 256K DiFFs. In 16-bit weight mode the RN = 128K DiFFs. In 1-bit mode things get more complicated. For 1 to 64 neurons, they can be spread over all 64 PNs, giving a maximum of 32K inputs each, for there are a maximum of 32K 1 bit weights per PN. For 65 to 128 neurons only 16K inputs are possible. For 129 to 192, there are (floor of 32K/3) 10922 inputs and so on till we have 449 to 512 neurons for which maximally 4096 inputs. Giving in total a RN of $5.7 \cdot 10^6$ DiFFs.

Now lets look at another chip: the ETANN, for here there are other complications. First of all it can implement a 1-layer 128x64 network. Also it can implement a 2-layer network of 64x64x64, and moreover 2-layer networks of 128x(64-j)xj for 0<j<64.
To calculate the RN, we can do the following:

In the 128x64 case there are 8K DiFFs
In the 64x64x64 case there are 256K DiFFs.
In the 128x(64-j)xj case for 0<j<64 the additional ways of how many fully connected 2-layer feedforward neural networks can be implement will be counted. This means that care has to be taken not to count topologies that are equal to already counted topologies. For j=63, there are 128-64=64 ways to chose the number of input neurons. One way to chose one hidden neuron and 63 ways to chose a set of 1 to 63 output neurons. For j=62 there is in fact also only one way to chose the number of hidden neurons although there are two of them because the topologies that have one hidden neuron have already been counted in the j=63 case. So every j 0<j<64 contributes 64·j additional topologies giving a total of 64·(1+2+..+63) = 64·63·64/2 = 129,024. In total this gives a RN of $4 \cdot 10^5$ DiFFs.

As these examples point out, however straightforward, the counting can become rather tedious. Therefore since topologies can be described formally very well, the counting can in principle be performed by a computer, giving it a description of the chip. For a few chips with simple architectures the RNs a given in appendix 2.

Scalability
Scalability is the ability for the chips to scale up to make implementation of larger neural networks possible, than those that can be implemented on one chip. In general digital chips have better scalability, not only because of their larger ability to drive load capitances, but also because of better controllability by for example transputers or DSP-chips [9]. One of the largest problems with scalability is to keep up with the growing number of connections that need to be made between chips. The best possible way to solve this is by transmitting over broadcast busses in a time division multiplexed way. Chips like CNAPS, Lneuro, WSI have very high scalability. Also with ETANN, broadcast bus interconnection is possible. In general, with these chips it is not possible to significantly enlarge the number of synapses per neuron, limiting the systems capabilities to non-fully connected neural networks. Therefore in this case to determine a reconfigurability number in the sense previously defined, for systems does not seem suitable.
There are a few chips that are exception for this rule. Those are the chips with growing neurons when more synapses are added, as the Mitsubishi BNU chips. Here every few synapses have their own piece of neuron. By connecting the neurons at their summing wires, the number of synapses can get very large. Another chip that makes use of this technique is the 'reconfigurable neurochip' [51--52], but here it is applied within the chip. Another highly scalable chip is URAN, which allows asynchronous/wired-OR expansion, which means that outputs can be tied together without considering timing or load effects.

In this case it will be the growing probability of pulse overlap that limits the scalability. A suitable figure for comparison would be just to give the number of neurons and weights of the largest neural network that can be implemented on a system consisting of particular neurochips. The only constraint that has to be imposed in this matter is to assure that this largest neural network is globally connected (not necessarily fully) so it can not be decomposed in two or more separate neural networks. Since for none of the reviewed chips this number is really known (there are some estimates) no figures are given for this criterium.

Weight memory capacity

From the theoretical point of view, a neural network consists of weights that can have infinitely many values, although bounded. For this, results are known that bound the capacity of the resulting network [43]. An ideal neural network chip would be capable of implementing all those different weightvalues. Due to practical limitations however, hardware can never be made to make infinitely many values, at least not significantly. In digital systems the weightvalues are often *quantized* to values within a certain range of maximum to minimum value in a way that the resulting values are not more than half a *quantization level* q from the intended unquantized value. This is similar to making an q/2-cover of the infinite set of all possible weightvalues. The size of this cover: the number of quantization levels defines in a straightforward way the capacity of a weight.

Likewise for a whole neural network counting the number of points that cover the whole weight space would be a good definition of capacity for the network. If a weight can be specified by b bits, then 2^b quantization levels are possible. Now if there are N weights, then in total 2^{bN} points can be accessed in weight space, in other words the size of the cover made by the chip is 2^{bN}. Note that the difference metric that is taken here is just the absolute difference between the unquantized and the quantized weightvalues. How this difference translates to differences at the output of a chip is not expressed by this definition, but as stated before, to get a problem independent definition of capacity in that way, averaging would be required over a huge amount of problems and input-output samples.

Since the logarithm of this capacity determines in a more natural way the 'number of freedoms' the neural network has, this also will be included in the following straightforward definition of capacity:

Assume a chip has N_w weights and for each weight w_i , b_i bits to specify its weightvalue. Then the **Weight Memory Capacity** (WMC) of the chip is $\sum_i b_i$.

Clearly this is just the memory contained on the chip to store the weightvalues in, so it is a very natural and easy-to-obtain criterium. In many situations the b_i values are the same for all weights (b_w) resulting in a WMC of $N_w \cdot b_w$. Note that this criterium does not make any distinction between chips that have 100 weights of 10 bits each, or 1000 1 bit weights. Since some problems are better solvable with few accurate weights and other with many less accurate weights, this seems to be a logical consequence of having a problem independent definition of capacity.
At first sight it seems that this definition of capacity is only suitable for digital representations weightvalues. Analog systems have other sources of error such as noise, non-linearities, decaying weights and so on. The effects of these can be totally different than those of quantization. A somewhat crude, but commonly used way to deal with this is to consider the amount of error in the stored weightvalue relatively to the full range of the weights. This percentage is used to argue that significantly a limited number of weightvalues can be distinguished. Taking the $\log_2$ of this number then results in an effective bit accuracy of a weight storing device. So if for example an analog weight storing device has an accuracy of 1% of the full range, a maximum of 100 levels can be distinguished significantly, resulting in a bit-accuracy of $\log_2 100 = 6.6$ bits. Although somewhat dubious, there are no better ways to be able to compare analog and digital weight storing devices in terms of accuracy. By adopting this convention weight memory capacity becomes a very general and easy to obtain criterium, that contributes better to the intuitive and theoretical idea of capacity than just counting the number of weights. Appendix 2 gives the numbers of this criterium for the reviewed chips.

Signal to Noise Ratio

So far, nothing has been said about accuracies of multipliers, adders and sigmoid-functions; the hardware devices that perform these operations. Since all of the inaccuracies of these devices contribute to the error on the output of the chip, they also contribute to the mean-square-error of a given training- or test set, therefore limiting the minimum value of it. Since performance of a trained neural network is most interesting, only errors will be considered that are left *after learning*. The learning experiment will be described in the next paragraph. Given a problem it is interesting to know which part of the mean-square-error, after learning, is due to the limitations of the architecture of the neural network that is implemented, and which part comes from hardware inaccuracies. Since limitations of the chip to implement neural network architectures have already been considered, now the effects of the hardware inaccuracies will be investigated by the following experiment. Take a neural network architecture that can be implemented by a chip under investigation. Now implement this neural network in two ways. First an idealised version in software with very high accuracies for weightvalues, inputvalues,

multiplication, addition and sigmoid generation. This version will be called the *ideal neural network*. Next implement the neural network using the chip with its limitations in accuracy and call this the *chip neural network*. By doing this, errors that occur because of inaccuracies of hardware can be measured. Since it is impossible to measure difference in mean-square-error for all problems, the following method will be adopted.
Inputs and weights are drawn from a suitable distribution and supplied to both the ideal and the chip network. Then the ideal neural network gives an output y and the chip an output $\hat{y}$. The difference of them $\Delta y = \hat{y} - y$ is the error due to hardware inaccuracies. Figure 5.2 shows the outline of the experiment.

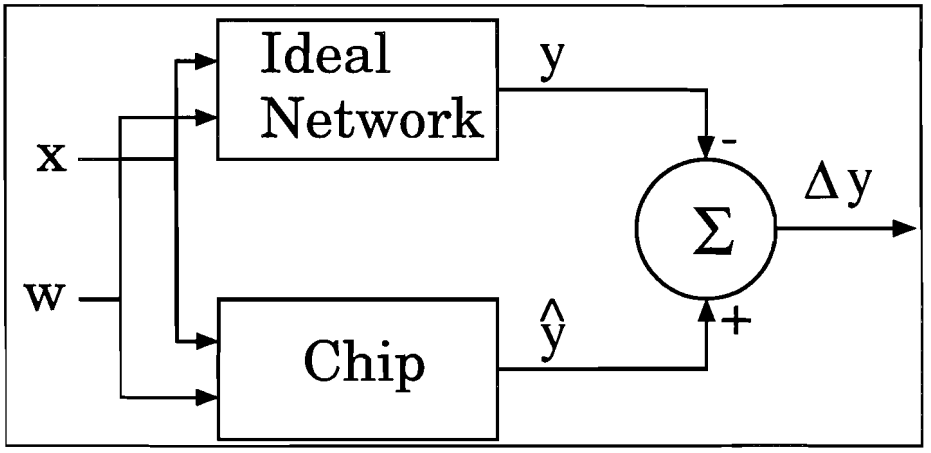

Figure 5.2: Outline of the signal to noise experiment

Both the average value of Δy and the variance of Δy over the chosen input and weight distributions give an idea of the absolute error contribution of the hardware. Often however the ratio between the variance of the signal σ^2_y and the variance of the error $\sigma^2_{\Delta y}$ gives a better idea of the magnitude of the error relative to the magnitude of the signal. This ratio is called the **Signal to Noise Ratio** (SNR) of a chip.
The problem is to determine a suitable distribution of inputs and weights. A good way to choose these distributions would be to record input- and weightvalues for many typical problems the chip is likely to be used for. Since typical distributions for weights and inputs are unknown and since it is hard to get theoretical results from such complex distributions each inputvalue and each weightvalue is considered to occur equally frequent resulting in uniform distributions of input- and weightvalues. Therefore the results that come out of this experiments might be atypical for many problems, but since only for this distribution theoretical results are known and since better, more practical distributions are yet unknown, this is the best that can be done for the moment. Also merely to compare chips with this number, choosing the *same* distributions for each chip is more important than

choosing a typical one. The experiment as such is independent of the distribution, so if and when more typical distributions will be known, then these can be used for the experiment. So the inputs x_i are drawn independently from a uniform distribution between their minimum and maximum value and likewise weights w_i are drawn. For these i.i.d. realisations of inputs and weights theoretical results are known that analytically calculate the SNR [38][53]. Although this experiment can be performed for any size neural network, for simplicity a one neuron network will be considered with N inputs and weights.
Under the assumptions that inputs x_i are i.i.d. realisations of a uniform distribution with zero mean and variance σ^2_x and weights w_i are i.i.d. realisations of a uniform distribution with zero mean and variance σ^2_w and a large number of inputs N, the values of the variable s in (1), i.d. the weighted sum of the inputs, tend to the normal distribution according to the central limit theorem. Therefore the 2N dimensional integral that needs to be solved to calculate σ^2_y can be replaced by a one dimensional integral in s.
Similarly the original 4N dimensional integral that has to be calculated in order to calculate the error variance can be replaced by a two dimensional integral in which the variables are distributed according to the bivariate normal distribution by making use of the central limit theorem. Again i.i.d. realisations of input and weight errors Δx_i and Δw_i are required as well as a large number of inputs.
If furthermore it is assumed that errors are small in order to be allowed to neglect second order errors, the SNR of a single neuron neural network with N inputs and a sigmoid function $\tanh(\lambda s)$ is given by (11) [38].

$$\left(\frac{S}{N}\right)_y = \frac{\sigma^2_y}{\sigma^2_{\Delta y}} = \frac{1}{g(\lambda\sqrt{N}\sigma_x\sigma_w)}\left(\frac{\left(\frac{S}{N}\right)_x \cdot \left(\frac{S}{N}\right)_w}{\left(\frac{S}{N}\right)_x + \left(\frac{S}{N}\right)_w}\right) \qquad (11)$$

The value of the *Noise to Signal gain* (NSR-gain) g depends on the standard deviation σ_s of the variable s, normalised to a sigmoid with steepness λ=1. Since inputs and weights are independently drawn, the value of this σ_s is just the square-root of $(N\cdot\sigma^2_x\cdot\sigma^2_w)$. The value of g is plotted in figure 5.3 as a function of this normalised standard deviation.
For small values of $\lambda\cdot\sigma_s$, the sigmoid appears to be linear, so the signal is amplified by the same amount as the error. For larger values of $\lambda\cdot\sigma_s$, the saturating shape of the sigmoid starts to limit the signal variance, while errors that occur near zero get significantly amplified. Therefore the NSR-gain increases.

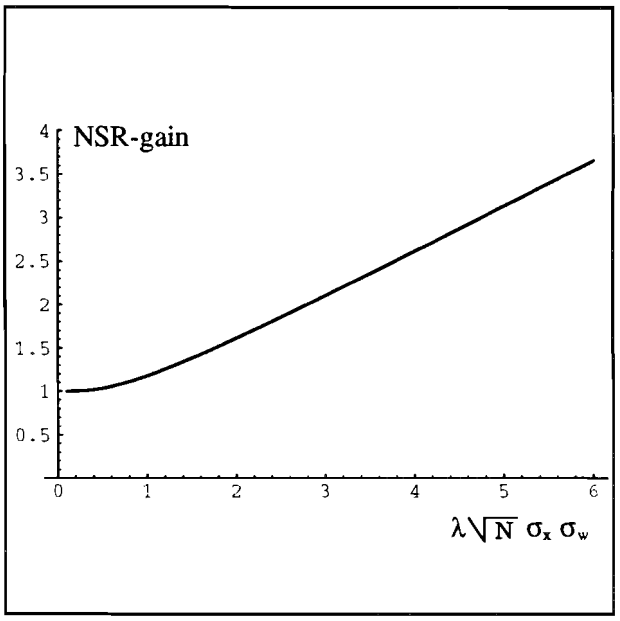

Figure 5.3: The value of the NSR gain for neurons with sigmoids

To give an example of the above, the following experiment has been performed. Both inputs and weights are drawn from a uniform distribution in the range [-r,r]. The chip neural network differs from the ideal only in that it uniformly quantizes weights and inputs to levels represented by b bits. This quantization is modeled by adding errors that are independent of the values and have variance $\sigma^2_{\Delta x}=\sigma^2_{\Delta w}=q^2/12$ [38], in which q is the quantization level. Assuming independency of values and errors only is not allowed in cases of very course quantization. Since in this case $q=2r/2^b$, and furthermore $\sigma^2_x=\sigma^2_w=r^2/3$ the NSR can be expressed in terms of r, b, N and λ.

$$\left(\frac{S}{N}\right)_y = \frac{2^{2b-1}}{g\left(\lambda\sqrt{N}\frac{r^2}{3}\right)} \qquad (12)$$

For r=1, N=64 and λ=1,10 theoretical curves and experimental data is plotted of the SNR in dB against b, the number of input and weight bits. Results are shown in figure 5.4. Also for a hardlimiting function ($\lambda \rightarrow \infty$) this experiment has been performed. In this case the theoretical model of (11) does not hold anymore because errors are not small anymore. For this case it has been derived that the SNR is as in (13). [53]

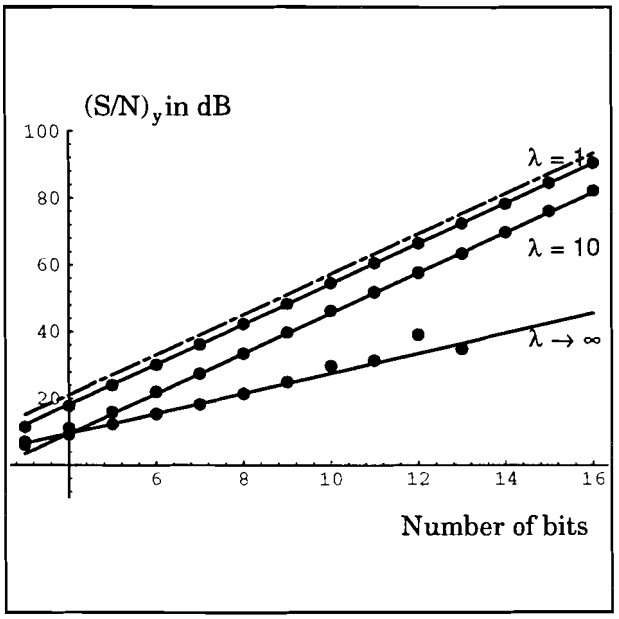

Figure 5.4: Theoretical signal to noise ratios of a neuron with sigmoid functions with varying steepness from 1 to infinity, together with points collected from Monte-Carlo simulations

$$\left(\frac{S}{N}\right)_y = \frac{\pi}{4}\sqrt{\frac{\left(\frac{S}{N}\right)_x \cdot \left(\frac{S}{N}\right)_w}{\left(\frac{S}{N}\right)_x + \left(\frac{S}{N}\right)_w}} \qquad (13)$$

Because of the square root in (13), the SNR now grows half as fast with increasing bit accuracies than in the case of sigmoidal neurons. In this sense hardlimiting neural networks are less robust than neural networks with sigmoids. The fact that in this case theory and practice do not match so well anymore for larger values of b, is due to the fact that straightforward Monte-Carlo simulations, as used in this experiment, are not suitable for small probabilities of large errors. The above example shows that the SNR criterium can be used to model errors caused by hardware on a chip. If the data collected by Monte-Carlo type of measurements matches with the curves predicted by the model, the model can be regarded as faithful.

Apart from using the SNR as a figure to investigate sources of error in a chip, the SNR as such is an interesting number to compare chips for their accuracies, even when no model of the errors is known. The advantage of measuring the signal to noise ratio by this experiment is the fact that it can be performed for digital as well as analog chips.

Learning to deal with systematic errors

A disadvantage of signal to noise ratio is that it only measures error *variance* relative to signal *variance*. The *mean* values of error and signal are left out of its scope. This is mainly because originally SNR has been defined for noise-like errors (therefore its name), which have zero mean. Some examples of not noise-like errors in chips would be offsets of the sigmoid function, errors occuring due to non-linearities in multipliers, the total malfunctioning of one or several neurons/synapses and deviations from the required sigmoid shape. In general, *systematic errors* could be defined as those errors that contribute to the mean error at the output of the neural network. So systematic errors are exactly those errors that are left after averaging over different inputs and weights.

The effects of these systematic errors have been investigated in the case of neural networks [54], and often it is argued that neural networks can learn to adapt themselves to these errors. The success in this adaptation does not only depend on the amount and magnitude of systematic errors, but also on the capabilities of the learning algorithm to deal with them. Algorithms that originally are designed for ideal neural networks often do not perform very well on hardware with limited resources. For example for the standard back-propagation algorithm it is known that for digital implementations at least 12 bits weight-values are required to be able to make the little weight updates that are necessary. Also it does not account for the fact that weights are limited in their range. Introducing a weight-decay term in the error function would cause the learning algorithm to find a solution with smallest possible weights, which therefore will not have as much trouble from limited range of weights than algorithms that do not use this. However, in that case, the effects of small signal quantization will probably play a part. Learning algorithms that would be capable of distributing the weightvalues evenly over the whole possible range would compromize between these two effects and therefore in this sense be optimal.

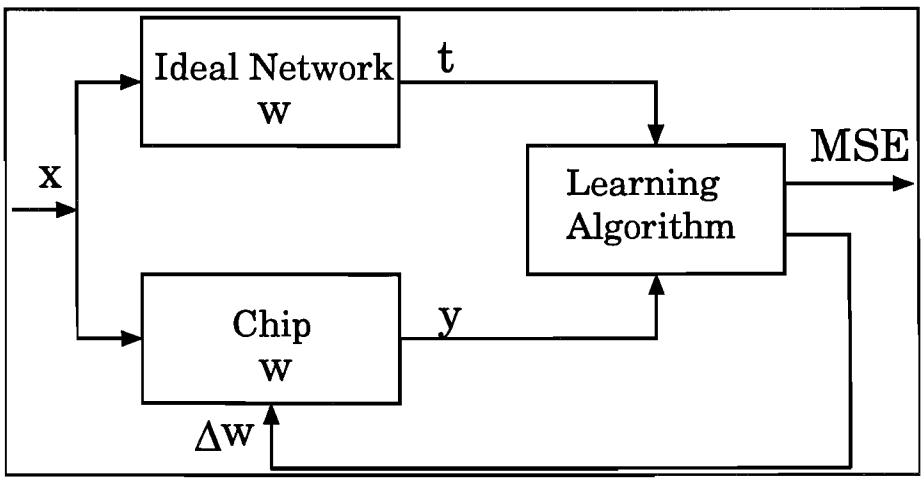

Figure 5.5: Setup of the learning of systematic errors

The experiment shown in figure 5.5 can determine the ability of a learning algorithm to learn to deal with the systematic errors of the chip. Therefore again consider the previously defined ideal neural network and the neural network implemented on the chip. First, the weights of both networks are initialised with equal random values W. Then a training sample ($\underline{x}$,$\underline{t}$) is generated by randomly choosing the input $\underline{x}$ and calculating the output of this input of the ideal neural network, which defines $\underline{t}$. These samples will be generated until a suitable size of the training set has been reached. Next the chip is learned according to a certain learning algorithm and the generated training set. The performance of this learning in terms of number of training samples that were required, residual mean-square-error and the number of epochs needed, in other words the *learning curve* gives an idea of the ability of the learning algorithm to deal with the systematic errors of that chip. This is done for many different weight initializations, for which mean and variance of the learning curves can be plotted to give an indication of the average performance of the learning algorithm to learn to deal with the errors of the chip. The advantage of this experiment is that it can be done no matter which systematic errors a chip has. The success of it will depend on the ability of the learning algorithm to deal with them. Therefore, to really compare chips with this experiment, the same learning algorithm must be adopted in the case that no algorithm is on-chip. If there is an algorithm on-chip, of course, it is most evident and interesting to use that particular algorithm. A further advantage of this experiment is that the effects of errors on the learning part of the chip are measured. If somehow limitations of the chip make learning difficult this results in bad performance on this experiment. Since this experiment not only measures systematic errors of the chip, but also performance of a learning algorithm, it really measures the fitness of a learning algorithm to be used with the chip.

Figure 5.6 shows a typical example of a learning curve generated by the backpropagation algorithm while learning to deal with simulated offset errors of the sigmoid function. Here the chip neural network was a simulated chip with built-in offset errors of the sigmoid function to serve as a systematic error. Measured mean-square-errors are on a test set. This example only shows that learning to deal with systematic errors is possible. A more interesting study is to perform this experiment on a real chip with real systematic errors. This experiment on the ETANN chip will be reported in the next chapter.

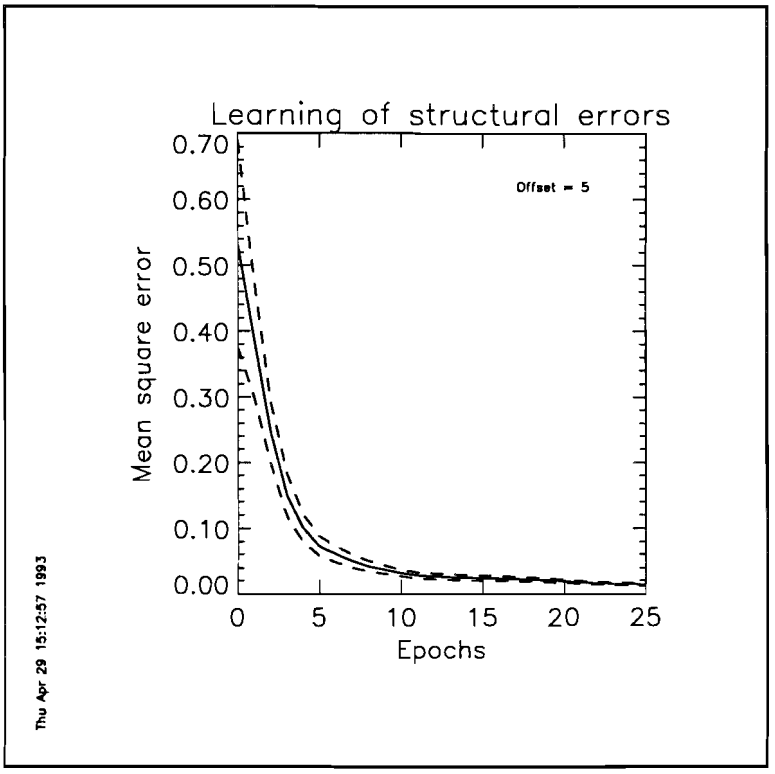

Figure 5.6: A typical learning curve while learning to deal with systematic offset errors

Effective Connection Primitives Per Second

As considered in previous chapters, it is not fair to compare the speed of chips by just counting the number of *connections* per second because clearly accurate devices do more in one connection than inaccurate ones. Somehow the extent to which a connection is counted must be weighted by the accuracy of such a connection.

Since the computational complexity of a connection of b_x input bits times b_w weight bits is $b_x \cdot b_w$, a first attempt in this direction can be made by weighting a connection by this product. So **Connection Primitives Per Second** (CPPS) could be defined as $b_x \cdot b_w \cdot$CPS. Although this already would be far less biased in favour of inaccurate chips, there are two disadvantages of this criterium. First of all this criterium fails to distinguish between chips that make accurate and less accurate $b_x x b_w$-bit connections. The second disadvantage can be explained by the following consideration. Suppose chip 1 has 5x5-bit connections and 2 GCPS, then it has 50 GCPPS according to the definition. Now there is another chip, chip 2, that has the same 50 GCPPS, but made up of 10x5-bit connections with 1 GCPS. Although they have the same number of GCPPS, the second chip is two times as slow, but is *not* two times as accurate, because at the output the accuracy is not very much better than the 5x5-bit connections (as pointed out by (11)). In other words $b_x \cdot b_w$ does

not weight the connection properly by the effective accuracy of such a connection at the outputs of the neural network chip.
Normally accuracy as considered in this sense, can be measured by the SNR and first it must be clear how speed and accuracy in terms of SNR can be traded-off. In the normal definition of SNR as signal variance over *noise* variance, it is assumed that the errors are independent of the outputs. Then by looking several times at the output the observed values can be averaged, and so the noise can be averaged out. To be more specific assume we have T observations of the outputs $\hat{y}_i = y + \Delta y_i$. Now sum these values, calling the signal term y_{sum} and the noise term Δy_{sum}, then the following can be derived if i.i.d. errors are assumed:

$$\hat{y}_{sum} = \sum_{i=1}^{T} \hat{y}_i = Ty + \sum_{i=1}^{T} \Delta y_i = y_{sum} + \Delta y_{sum} \tag{14}$$

$$\mu_{y_{sum}} = E[Ty] = TE[y] = T\mu_y \tag{15}$$

$$\mu_{\Delta y_{sum}} = E[\sum_{i=1}^{T} \Delta y_i] = \sum_{i=1}^{T} E[\Delta y_i] = T\mu_{\Delta y} \tag{16}$$

$$\sigma^2_{y_{sum}} = E[(y_{sum} - \mu_{y_{sum}})^2] = E[(y_{sum})^2] - \mu^2_{y_{sum}} = E[(Ty)^2] - (T\mu_y)^2 = T^2(E[y^2] - \mu^2_y) = T^2\sigma^2_y \tag{17}$$

$$\sigma^2_{\Delta y_{sum}} = E[(\Delta y_{sum} - \mu_{\Delta y_{sum}})^2] = E[(\Delta y_{sum})^2] - \mu^2_{\Delta y_{sum}} = E[(\sum_{i=1}^{T} \Delta y_i)^2] - (T\mu_{\Delta y})^2 \tag{18}$$
$$= \sum_{i=1}^{T} E[(\Delta y_i)^2] - T^2\mu^2_{\Delta y} = \sum_{i=1}^{T} (E[(\Delta y_i)^2] - \mu^2_{\Delta y}) - (T^2 - T)\mu^2_{\Delta y} = T\sigma^2_{\Delta y} - (T^2 - T)\mu^2_{\Delta y}$$

If the mean of the error is zero ($\mu_{\Delta y} = 0$)

$$(\frac{S}{N})_{y_{sum}} = \frac{T^2\sigma^2_y}{T\sigma^2_{\Delta y}} = T\frac{\sigma^2_y}{\sigma^2_{\Delta y}} = T(\frac{S}{N})_y \tag{19}$$

Looking at (19), if the mean error is zero, the SNR can be doubled by looking twice at the output, or in other words by proceeding with half the speed. So the product SNR·CPS at first sight seems to be a good criterium because a chip that has double the speed but half the SNR of another chip can by looking twice at its outputs get the same accuracy at the same effective speed.

However being able to see in what way chips are the same, if they have the same value of a criterium is not enough. If for example chip 1 has four times the value of chip 2 under this criterium, then it also must perform four times as good. For speed this means that the amount of *effort* that has to be done by chip 2 per unit of time must be four times the effort of chip 1. Clearly when using SNR·CPS as a criterium, this is not the case, because as (12) points out, *adding* a bit for inputs and weight would already yield a four times as large value, while it does not require four times as much effort. For chip 2 to make four times as much effort, for example the number of bits four inputs and weights must be doubled. This would give a factor of 2 improvement of the *logarithm* of the SNR. So a good criterium that would say something about the effective effort that a chip makes per unit of time, would scale proportionally to the square of the logarithm of the signal to noise ratio. Another problem is the fact that for digital circuits the error is *not* independent of the signal value, in fact it is totally determined by it. This means that if the output is measured twice, then also the same errors will occur, so averaging will not decrease the error.
To be able to use the SNR in a speed criterium, a little less obvious construction must be made:

- A *bxb-bit connection neural network chip* is an ideal neural network chip except for the fact that it has b bits for inputs and weights, making bxb-bit connections.

- A neural network chip is said to be *equivalent* to another neural network chip if and only if they implement the same neural network topology.

- A neural network chip is said to be *equally accurate* as another neural network chip if and only if they have the same SNRs.

- The **Effective Connection Primitives Per Second** (ECPPS) of a neural network chip is defined as b^2·CPS, where b is from its equivalent equally accurate bxb-bit connection neural network chip.

In this way all the inaccuracies of the chip are accounted for in the speed criterium of the chip. If the SNR and the NSR-gain of the chip are known then determining b is straightforward according to (11) and (12).

$$\left(\frac{S}{N}\right)_{chip} = \left(\frac{S}{N}\right)_{equi} = \frac{1}{g_{equi}} \cdot \left(\frac{\left(\frac{S}{N}\right)_x \cdot \left(\frac{S}{N}\right)_w}{\left(\frac{S}{N}\right)_x + \left(\frac{S}{N}\right)_w}\right) = \frac{2^{2b-1}}{g_{chip}} \qquad (20)$$

$$b = \frac{\log_2\left(\left(\frac{S}{N}\right)_{chip}\right) + \log_2(g_{chip}) + 1}{2} \qquad (21)$$

Note that $g_{equi}=g_{chip}$ because since they are equivalent they implement the same neural network, so the NSR-gains are equal. From (21) it can be seen that ECPPS is proportional to the speed CPS and the square of the logarithm of the SNR as was considered to be a good criterium to measure performance in terms of effort per unit of time. If two chips have the same ECPPS then they effectively process equal number of connection primitives per second.
A disadvantage of this criterium is that it relies upon knowledge of the SNR value of the chip, so measurements have to be done to obtain numbers. Therefore no numbers can be given yet for the reviewed chips of chapter 2. To be able to give some numbers of chips for a related criterium, the ECPPS criterium will be weakened a little.
Because the problem is to have a good value for the SNR of a chip (CPS is often known), it will be *idealized* by assuming that the only sources of error in the chip come from limited accuracies for weights and inputs. Of course this is a little optimistic and also biased in favour of chips with less accurate devices, but by lack of more knowledge of the SNRs of the chips this is the only way to come up with some numbers.

- Now chips are said to be *idealized equally accurate* if and only if they have the same idealized SNRs.

- The **Balanced Connection Primitives Per Second** (BCPPS) of a neural network chip is defined as $b^2 \cdot$CPS, where b is from its equivalent idealized equally accurate bxb-bit connection neural network chip.

Here it is called balanced because the only difference between this and plain CPPS is the fact that here the number of input and weight bits is balanced. The idealized SNR and henceforth b can be calculated from (11)

$$\left(\frac{S}{N}\right)_{chip} = \frac{1}{g_{chip}} \cdot \frac{2^{2b_x} \cdot 2^{2b_w}}{2^{2b_x} + 2^{2b_w}} = \frac{2^{2b-1}}{g_{equi}} = \left(\frac{S}{N}\right)_{equi} \qquad (22)$$

$$b = \frac{1 - \log_2\left(\frac{1}{2^{2b_x}} + \frac{1}{2^{2b_w}}\right)}{2} \qquad (23)$$

As can be seen from (23) if $b_x=b_w$ then $b=b_x=b_w$. Furthermore if $b_x >> b_w$ then $b \approx b_w+½$ and vice versa. In other words, $\min(b_x,b_w) \leq b < \min(b_x,b_w)+½$. BCPPS values for the reviewed chips of chapter 2 are given in appendix 2. As can be seen from the above, a chip is heavily punished under this criterium for not having balanced weight and input accuracies. Also the 320 BCPPS of the NET32K chip does not seem so extreme as under plain CPS. From the numbers of the CNAPS chip it can be clearly seen that balancing weight and input accuracies yield the highest BCPPS rates.

6. Experiments with the ETANN chip

The ETANN chip will be used to measure its Signal to Noise Ratio after learning with back-propagation to deal with its errors. Both of these experiments as explained in the previous chapter. For this, first an overview will be given of ETANNs sources of error. While learning, the major source of error appeared to be the shape of ETANNs sigmoid. Since this error covers in magnitude all others it would be interesting to know what would happen if this shape was adopted by the ideal network. In this way the learnability of other sources of error could be investigated. In order to be able to do this, a model of ETANNs sigmoids will be derived. In 6.3 a fitting procedure will be described where all parameters of the ideal sigmoid and the derived model of the ETANN sigmoid will be obtained. In 6.4 the actual learning experiment is described for both the case where the ideal sigmoid was adopted and the case where the model of ETANNs sigmoid was adopted. In paragraph 6.5 the results of the Signal to Noise Ratio measurements are given.

6.1 The ETANN chip and its sources of error

64 Analog inputs can be clocked in the ETANN chip, and simultaneously transferred to a input synapse array, where they are multiplied by 4096 synapses. For every neuron there are 16 biases (1024 in total), making a total of 5120 synapses.

$$s_j = \sum_{i=1}^{64} w_{ij} x_i + \sum_{b=1}^{16} w_{bj} \quad \text{for } 1 \le j \le 64 \tag{24}$$

Inputs are restricted to values in the range [-1,1] and weights can have values in the range [-2.5,2.5]. After outputs are calculated by the sigmoid generation circuits, they are available at the output pins, but also they can be fed back to the next array of 5120 synapses, making implementation of a second layer possible.

Figure 6.1 shows the path of one single input in a little more detail. Looking at the devices for multiplication addition and sigmoid generation in more detail allows qualitative inventory of the errors that will be likely to occur. To really predict the magnitudes of these errors a more detailed model of the chip will be required. Since this requires knowledge of several specific parameters such as W over L ratios and process variables, no such model is explicitly derived. Instead a global inventory is made and a more complete version of this is given in [55].

The input buffers provide the input for 64 synapse circuits and therefore cause the major part of the total delay of the chip. Physical variations among buffers and within buffers cause common-mode variations at the output of the buffers for different inputs and for different buffers. At extreme inputs non-linear effects begin to appear.

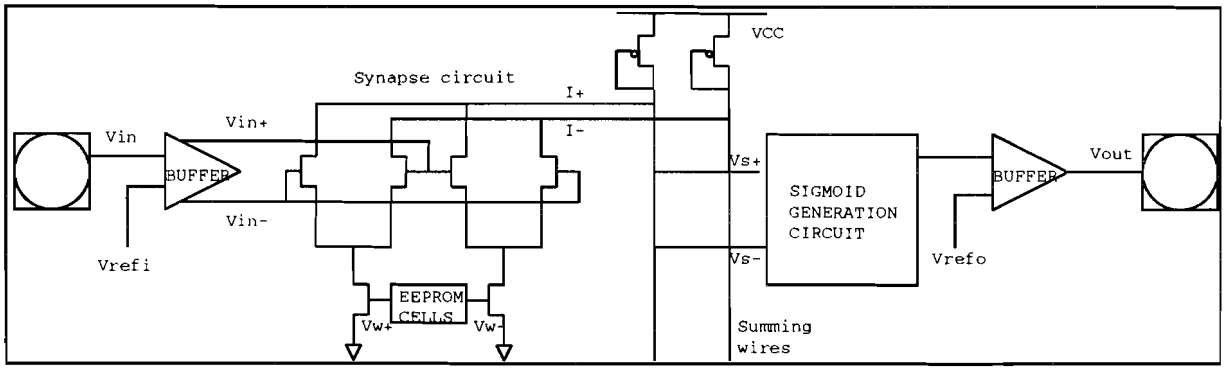

Figure 6.1: ETANNs path from input to output

The same can be said for the simplified Gilbert-Multipliers that implement the synapse circuits. To maintain sufficient voltage-space, the normally present current source at the bottom of the device that sets up a constant current through the device has been removed in the ETANN chip. With $\Delta V_{in} = (V_{in}^{+}-V_{in}^{-})$ the input differential voltage, and $\Delta V_w = (V_w^{+}-V_w^{-})$ the differential voltage over the floating gate cells representing the weight value, the output differential current $\Delta I = (I^{+}-I^{-})$ is proportional to $\Delta V_{in} \cdot \Delta V_w$. The charges on two floating gate cells determine V_w^{+} and V_w^{-} respectively and can be programmed by a Fowler-Nordheim tunnelling process. At extreme values non-linear saturating effects of the synapse circuits limit the differential current ΔI. Common-mode values $(I^{+}+I^{-})/2$ vary with different input and weights.

Summing wires add the various currents for each of the 80 synapse circuits. Simple non-linear load transistors are used to make voltages V_s^{+} and V_s^{-} as inputs for the sigmoid generation circuits for two reasons. First of all, when only few synapses are used, the common-mode current will be little resulting in a higher impedance of the loads, causing a larger difference in voltages V_s^{+} and V_s^{-}. So this results in automatic gain adjustment of the neuron when few or many inputs are used. A second advantage is that for a fixed number of inputs common-mode variations of the summing currents, that can vary as much as 680µA to 1mA in total, get less amplified than with linear loads making the demands for the common-mode rejection ratio of the sigmoid generation circuit less severe.

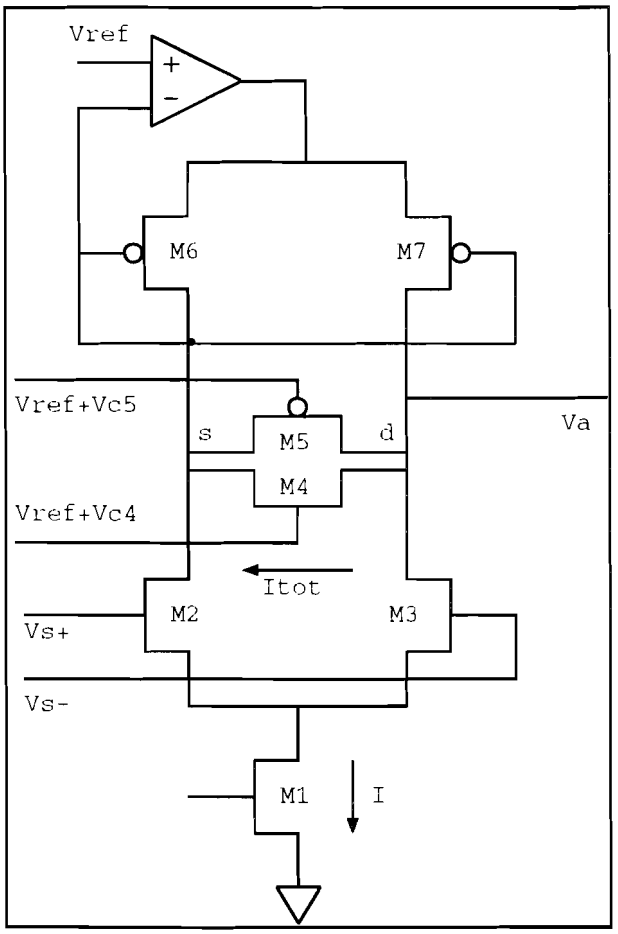

Figure 6.2: The sigmoid generation circuit of the ETANN chip

6.2 The sigmoid generation circuit of the ETANN chip

The operation of the sigmoid generation circuit as shown in figure 6.2 will be investigated in more detail. When $\Delta V_s = (V_s^+ - V_s^-)$ is zero, equal currents flow through all the transistor pairs resulting in a completely balanced circuit and therefore then $V_A = V_{REF}$. Assuming that both M_6 and M_7 are in saturation they form a current mirror. Therefore, when ΔV_s is different from zero, difference in currents through M_2 and M_3 cause the flow of an effective current $I_{tot} = Ids_4 + Ids_5$ through M_4 and M_5. This current determines the voltage difference $V_A - V_{REF}$.

At the gate of M_4 a constant voltage $V_{REF}+Vc_4$ is supplied and similarly the voltage $V_{REF}+Vc_5$ to the gate of M_5. Since the voltages Vc_4-Vt_4 and Vc_5-Vc_5 determine the operation area of the transistors, the shape of the sigmoid will be derived as a function of these values. For simplicity it is assumed that $Vc_4-Vt_4=-(Vc_5-Vt_5)$, which is correct apart from

mismatches in transistors that make these values. For the relation between Ids and Vds of a N-channel MOST formula (25) will be used. For a P-channel MOST (26) will be used.

$$Ids = K((Vgs - Vt)_+^2 - (Vgd - Vt)_+^2) \quad (25)$$

$$Ids = -K((Vgs - Vt)_-^2 - (Vgd - Vt)_-^2) \quad (26)$$

Here $(.)_+$ denotes that the term is only included if it is larger than zero and $(.)_-$ denotes that the term is only included when it is less than zero. The labels d and s in figure 6.2 have no strict relation to the sources and drains of transistors M_4 and M_5, they are just labels. The following two cases are distinguished:

Case 1: $Vc_4 > Vt_4$ and $Vc_5 < Vt_5$

For current Ids_4 the following holds:

$Vgs_4 = V_{REF} + Vc_4 - V_{REF} = Vc_4 > Vt_4$ so the first term in (25) is included.
$Vgd_4 = Vgs_4 - Vds_4 = Vc_4 - Vds_4$.

So if $Vds_4 < Vc_4 - Vt_4$ the second term is also included and M_4 is in its linear region yielding for Ids_4

$$Ids_4 = K_4(2(Vc_4 - Vt_4)Vds_4 - Vds_4^2) \quad (27)$$

If $Vds_4 > Vc_4 - Vt_4$ then only the first term is included and M_4 is in saturation yielding for Ids_4

$$Ids_4 = K_4(Vc_4 - Vt_4)^2 \quad (28)$$

For current Ids_5 in a similar way:

$Vgs_5 = V_{REF} + Vc_5 - V_{REF} = Vc_5 < Vt_5$ so the first term of (26) is included.
$Vgd_5 = Vgs_5 - Vds_5 = Vc_5 - Vds_5$.

So if $Vds_5 > Vc_5 - Vt_5$ the second term is also included and M_5 is in its linear region yielding for Ids_5

$$Ids_5 = -K_5(2(Vc_5 - Vt_5)Vds_5 - Vds_5^2) \quad (29)$$

If $Vds_5 < Vc_5 - Vt_5$ then only the first term is included and M_5 is in saturation yielding for Ids_5

$$Ids_5 = -K_5(Vc_5-Vt_5)^2 \tag{30}$$

In total $I_{tot} = Ids_4+Ids_5$ and because $Vds = Vds_4 = Vds_5$, I_{tot} consists of the following three parts:

For $Vds > Vc_4-Vt_4$ clearly also $Vds > Vc_5-Vt_5$ so formula (28) and (29) hold so

$$I_{tot} = K_4(Vc_4-Vt_4)^2-K_5(2(Vc_5-Vt_5)Vds-Vds^2) \tag{31}$$

For $Vc_5-Vt_5 < Vds < Vc_4-Vt_4$ (27) and (29) hold so

$$\begin{aligned} I_{tot} &= K_4(2(Vc_4-Vt_4)Vds-Vds^2) - K_5(2(Vc_5-Vt_5)Vds-Vds^2) \\ &= 2(K_4(Vc_4-Vt_4)-K_5(Vc_5-Vt_5))Vds - (K_4-K_5)Vds^2 \end{aligned} \tag{32}$$

Finally for $Vds < Vc_5-Vt_5$ also $Vds < Vc_4-Vt_4$ so formula (27) and (30) hold so

$$I_{tot} = K_4(2(Vc_4-Vt_4)Vds-Vds^2) - K_5(Vc_5-Vt_5)^2 \tag{33}$$

Now if $K_4 = K_5$ then the quadratic term of (32) cancels out resulting in a linear relation between Vds and I_{tot}, with gain adjustable by Vc. Note that due to mismatches of the P and N channel MOST, K_4 and K_5 typically are not exactly equal, resulting in a not exact linear relation of formula (32). Also in that case the slopes of the sigmoids will not be equal anymore.

Case 2: $Vc_4 < Vt_4$ and $Vc_5 > Vt_5$

In this case for both transistors the first terms of (25) and (26) will not be included. So $Ids_4 = 0$ for $Vds > Vc_4-Vt_4$ and for $Vds < Vc_4-Vt_4$ (34) holds:

$$Ids_4 = -K_4(Vds-(Vc_4-Vt_4))^2 \tag{34}$$

Also $Ids_5=0$ for $Vds < Vc_5-Vt_5$ and for $Vds > Vc_5-Vt_5$ (35) holds:

$$Ids_5 = K_5(Vds-(Vc_5-Vt_5))^2 \tag{35}$$

The sum I_{tot} is now just given by formula (35) for $Vds > Vc_5-Vt_5$, formula (34) for $Vds < Vc_4-Vt_4$ and zero for values inbetween. As can be seen from (34) and (35) mismatches of transistors will lead to different slopes for the roll-off sections of the sigmoid.

The steepness of the resulting sigmoids of course is not infinite now, but rather determined by the open loop gain of the resulting transconductance amplifier when M_4 and M_5 are cut away. So this also will result in a sigmoid of three sections: Two square-root curves that provide the roll-off sections of the sigmoid and a linear region in between. From $V_A = V_{REF}+Vds$ it can be seen that the sigmoid shape is formed around V_{REF}. However, at the output it must be formed around V_{refo}, so after the sigmoid generation circuit, first $V_{REF}-V_{refo}$ is subtracted from V_A. After that it is amplified 5x around an inverting amplifier. Together this would give the sigmoids as shown in figure 6.3. However, the opamps that perform these operations clip at a given point to their extreme values, limiting the sigmoids. Therefore, at the output a sigmoid consists of the following 5 sections: 1) When inputs ΔVs are large in negative, the sigmoid is clipped to a minimum value. 2) For smaller negative values of ΔVs, a square rooted part provides the negative roll-off section of the sigmoid. 3) For small absolute values of ΔVs, the sigmoid is formed by a straight line. 4) For larger positive values again a square rooted curve determines the positive roll-off section of the sigmoid until a point where ΔVs becomes so large that the sigmoid clips to its maximum value. It is this 5 section model that will be used to fit the datapoints to that are collected at the output of the ETANN chip and the input of the sigmoid generation circuit. First, however, this data will be fitted to an ideal sigmoid curve.

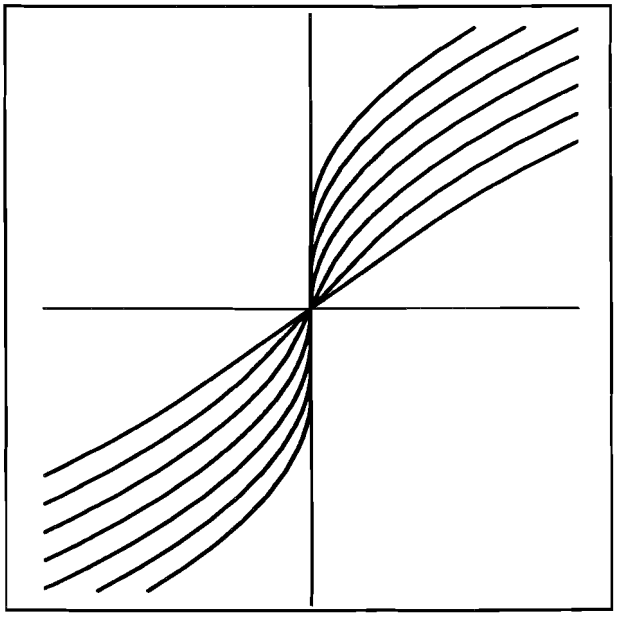

Figure 6.3: Typical sigmoid characteristics of the ETANN chip for different values of Vc_4 and Vc_5, as calculated from (31)--(35)

6.3 Fitting of ETANNs sigmoids

In order to perform the proposed experiment of letting ETANN learn to deal with its errors not only the chip will be needed, but also a good definition of the ideal neural network that is involved. Clearly the ideal neural network must have the same architecture as the ETANN chip: 64 weights and 16 bias weights to each of the 64 neurons. Also it must have the same range for allowable inputvalues [-1,1] and weightvalues [-2.5,2.5], because it must be possible to load the ETANN chip with the same weights as the ideal neural network and also it must be possible to present the same inputs to the ETANN.
Another issue is what sigmoid has to be used to provide the ideal network with. Of course just $1/(1+e^{-s})$ could be adopted, but this would mean that errors would occur just for the fact that ETANNs sigmoid gain is different from 1. It is better, because having a gain different from 1 is not considered to be an error, to use $1/(1+e^{-\lambda s})$ where λ is the same as in the ETANN chip. Also, because the extreme values of ETANN are not exactly 0 and 1 severe errors would occur if the ideal network has those values as extrema for the sigmoid. Since also it will not be considered an error that ETANN does not exactly have these extreme values of the sigmoid, the ideal network will have to adopt the same extrema as the ETANN chip. In other words, all the characteristics of the chip that are not considered to be errors must be adopted by the ideal network, so the ideal network must have the same architecture, input and weight ranges and steepness and extreme values of the sigmoid as the ETANN chip. Using this approach, errors that occur due to any difference in the ideal network and ETANN chip can be investigated. Examples of these are limited accuracies for weights and inputs, errors due to non-linearities of multipliers, offset of the sigmoid and deviations from the sigmoid shape.
So in order to properly make an ideal neural network the sigmoid gain and the extreme values of ETANNs sigmoid must be known. To estimate these, an experiment has been performed, called the *fitting of ETANNs sigmoids*. For one single neuron with 64 inputs and 16 biases, 100 times, different random weight sets were chosen, in which each weight was drawn from a uniform distribution on [-2.5,2.5]. For each of these weight instances a set of 100 different random inputs, drawn uniformly from [-1,1], was fed through the chip, giving a total of 10,000 output values. These data points could have been fitted together with the estimated values s of (24), but then errors occurring because of non ideal calculation of s should be included in the model of the sigmoid in order to make a good fit possible. The ETANN chip, however, has the possibility to measure the voltages Vs^+ and Vs^- that represent these values. It is assumed here that the sigmoid generation circuit has a sufficient common mode rejection ratio to be allowed to consider only differences ΔV_s. These measured values, which henceforth will be called *net*, are used as input datapoints for the fitting procedure. The outputs (y) serve as output datapoints and their

relation will be fitted to a model of an ideal sigmoid and to the model of paragraph 6.2, using the nonlinear fitting method of Levenberg-Marquardt [56]. Figure A3.1 in appendix 3 shows the results of this fit to the two different sigmoid models. The dotted curve shows the fit to the following model:

$$y = \frac{a - b}{1 + e^{-c(net-d)}} + b \qquad (36)$$

The solid curve shows a fit to the 5 section model presented in paragraph 6.2, with the clipping of the opamps included. Clearly this fit is far better than the fit with model (36), because the model is more accurate. However, the first model is the one that will be used by the ideal network and getting the parameters of this model is of first interest. Later, to investigate the learnability of other sources of error, the fit to the in paragraph 6.2 derived model will be used. By performing the first fit a,b,c and d have been estimated so the extremes of the sigmoid a and b were known. However, not yet a proper value of the gain of the sigmoid, because value c in this model is the gain of the sigmoid from net values to output values and not from s values to output values. For the ideal neural network the parameters of the following model must be known:

$$y = \frac{a - b}{1 + e^{-\lambda \cdot s}} + b \qquad (37)$$

Where it is assumed that the relation between net and s can be modeled by:

$$net = e \cdot s + f \qquad (38)$$

Demanding that (37) with s given by (38) with f=0 equals (36) with d=0, gives the required value $\lambda = c \cdot e$. The only reason that d and f were included was to make a better fit possible. The values of e and f are estimated by fitting s and net datapoints to model (38) as shown in figure A3.2. Finally in figure A3.3 a plot is given from s to y, with (37) as the dotted curve and the solid curve is the same as in A3.1 but then with transformation (38) in it. As ideal sigmoid (37) will be adopted. The values of the mentioned parameters are given in table 6.1 for a single neuron of the ETANN chip.

6.4 Learning ETANN to deal with its systematic errors

Fitting of ETANNs sigmoids has given all the parameters needed to construct the ideal network so all the prerequisites have been made to perform the experiment of learning to deal with the errors of the ETANN chip by the back-propagation algorithm.

Table 6.1: Estimated values of the fitting parameters

a	b	c	d	e	f	λ
0.87	-0.94	122	0.0052	0.0022	0.013	0.26

First of all, a random weight initialization was generated, where again, weights were drawn independently from a uniform distribution over [-2.5,2.5]. This set of weights was downloaded into the ideal network and the ETANN chip. Next a training set of 100 inputs and targets was generated by feeding the randomly drawn inputs through the ideal network. This training set was used to train the ETANN chip with the standard back-propagation algorithm without momentum and with constant learning speed. After having learned one epoch, i.d. one time the whole training set, a test set was generated on which mean-square-error, mean error and error variance were calculated. This test set was used as training set for the next epoch, so one sample was only used one time. Because of the abundancy of samples this was possible. 25 Epochs were needed to detect no further improvement in the magnitude of the errors. This learning procedure was repeated for 100 different weight initializations. A flow diagram of the learning procedure is given in figure 6.4.

Results of the learning procedure are given in appendix 4 figure A4.1. The solid curves of this figure depict mean learning curves over different weight initializations, while the dotted curves show the variances of these curves. As can be seen from these curves, though the main part of the residual mean-square-error is contributed by error variance, the contribution of the mean of the error is not negligible. Still these mean errors are much better learned than error variances because before learning, they contributes most to the mean-square-error. This supports the notion that systematic errors can be learned better than errors that variate over the inputs and weights. While before learning these errors contribute most to the mean-square-error, after learning the error variance dominates the mean-square-error.

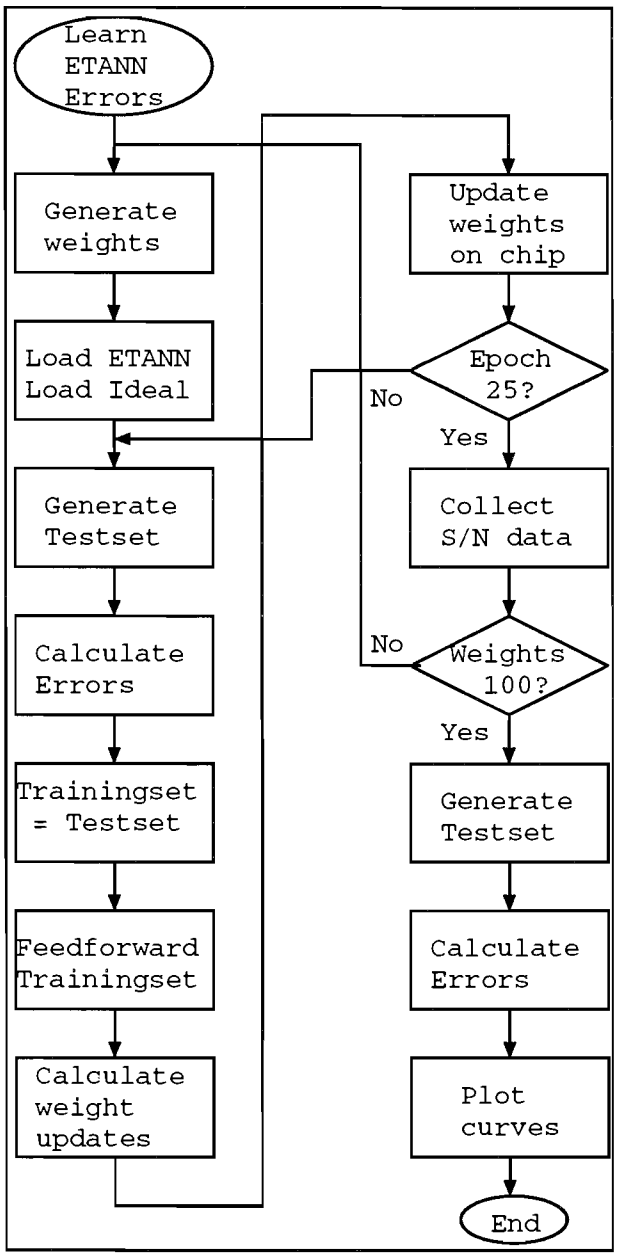

Figure 6.4: Flow diagram of the learning procedure

To be able to qualitatively judge learning performance, datapoints s against y are plotted after learning in figure A4.2, together with the ideal learning curve (35). When comparing this to figure A3.3, it can be seen that mainly errors in the calculation of s are corrected by learning. Of course this is hardly suprising, because only this value can be manipulated by reprogramming the weights. From figures A4.3 and A4.4 that depict s against net and net against y respectively, it can be concluded that learning caused an additional offset on the sigmoid of the ETANN. This was done in order to keep all the datapoints of figure

A4.2 as close as possible to the ideal learning curve, because the mean-square-error of these deviations is what was minimized by the learning algorithm. If no such offset was introduced the resulting datapoints in A4.2 would have been far further away from the ideal curve. Figure A4.3 also shows the effect that the variance in net has been reduced which means that errors due to non linearities of the multipliers are rather well learned. From all the figures in appendix 4 it can be seen that the main part of the error is caused by the shape of the sigmoid that ETANN provides. The problem with this is that ETANN can never learn to adapt to this source of error because it has no way of reshaping its sigmoids else than changing the gain and the extreme values a little. The best that can be done is to use a learning algorithm that also automatically adjusts these parameters. Much improvement does not have to be expected from this however.

When using the ETANN chip in a completely different way, for example for classification, the mapping that has to be learned will drive the inputs to the sigmoids of the output neurons to large values because the extreme values of the sigmoids have to be learned. In that case the deviations of the ideal sigmoid function, which mainly occur in the area where the sigmoid function has intermediate values, will have not nearly as much inpact as in the above experiment. So because the above experiment measures in the area where the sigmoid transfers from extreme values, the outcomes of the experiment are only significant in cases where the chip is used that way. If the probability distributions of inputs and weights were chosen to yield large input values for the sigmoids, then the experiment would say something about the ability to learn to deal with errors in classification problems.

Because now this error masks in magnitude all the other errors that occur on the chip, no conclusions about other errors can be made. Therefore it is interesting to know what would be the next source of error apart from sigmoid shape. To investigate this the model of ETANNs sigmoids can now be used in the ideal chip, thereby declaring it to be no error anymore. While the results of this experiment can not be used anymore to compare ETANN to other chips because a major source of error is eliminated, they do give an indication of the amount of error caused by other non-idealities of the chip. Results of this experiment are shown in appendix 5. From figure A5.1 it can be seen that there is improvement is mean-square-error of approximately factor 3. Almost all mean error is eliminated, leaving only error variance as contribution to the mean-square-error. Still this is far from the theoretical value that one would expect if there were only errors of limited input and weight accuracies as was used in experiments in the previous chapter. Of course this is because there are a lot more sources of error that occur in the ETANN chip, as mentioned in paragraph 6.1, and they all should be included in a model that would give some theoretical expectation of the achievable minimum error.

Figure A5.2 show almost the same datapoints as A4.2, except for a change in offset. Since now these points are compared to the proper model of ETANNs sigmoids, they give rise to much less error. Also, now, no introdution of a change in offset is required to correct for these errors, which explains the difference between the results shown in figures A5.3 and A4.3. From all the figures in appendix 5 it can be seen that most of the error now comes from variations in net values over different input and weight values. Again these are difficult to learn errors because they variate over inputs and weights.
Note that by incorporating each time an error in the ideal neural network, the difference in mean-square-error that results from doing this determines the learnability of that error under a given learning algorithm. In this way the learnability of all errors of a chip can be determined. Meanwhile this results in a proper model for the chip and its errors.

6.5 The Signal to Noise Ratio of the ETANN chip

While learning the errors of the chip, each time after 25 epochs when the learning had finished, data was collected to determine the Signal to Noise Ratio of the ETANN chip. Results of these measurements are given in table 6.2.

Table 6.2: Results of the Signal to Noise measurements of the ETANN chip

	μ_y	$\mu_{\Delta y}$	σ_y^2	$\sigma_{\Delta y}^2$	SNR	b	ECPPS
Ideal sigmoid	-0.04	+0.04	0.37	0.021	17.5	3	18G
ETANN sigmoid	-0.3	+0.006	0.37	0.0077	48,7	-	-

The effective bit accuracy b as introduced in the previous chapter has been calculated according to (21). The NSR-gain value that was used for this calculation can only be determined in the case of the ideal sigmoid shape, because only here the model is valid. The following values were used: λ=0.26 (see table 6.1), σ_x= 1/$\sqrt{3}$ for inputs and σ_x= 1 for biases, σ_w= 2.5/$\sqrt{3}$ and N=80.
The fact that the ETANN chip is after learning only as accurate as a 3x3-bit neural network, seems to be bad, but care has to be taken to conclude anything from this

because totally no data of other chips is available from these experiments. It does show that merely taking input and weight accuracies as main sources of error to estimate the SNR in this case clearly is too optimistic as can be seen from the difference in this ECPPS value and the BCPPS value of the ETANN chip in appendix 2.

7. Conclusions and Recommendations

The success of neural networks will, to a large extent, depend on the ability of them to deal with interesting real world problems. Since often these kind of problems impose severe time constraints, hardware implementation becomes indispensable.
A brief review has been given of popular neural network chips of major companies in electronic chip manufactoring. Chips for neural networks vary widely from straightforward digital, DSP-like implementations, which are often very general purpose but typically only have little integration density, to analog ICs, that implement larger networks but suffer from inaccuracies due to noise and other physical short-comings. Due to this enormous diversity, which is mainly the product of lack of experience in making them and little consensus on what makes a good chip, comparing their *performance* becomes a difficult task.
The first step for the solution of this problem has been to define a proper set of hardware performance criteria to which the chips can be subjected. In order to be able to evaluate hardware performance criteria, first, meta-criteria are proposed. These meta-criteria demand both generality and practicality of a hardware performance criterium. These two requirements tend to conflict and therefore a good hardware performance criterium is subjected to a compromize between them.
Commonly used criteria for size, accuracy, speed and cost of a neural network chip, often fail to satisfy at least one of the proposed meta-criteria sufficiently. Counting just the number of neurons and weights that can be implemented does not necessarily answer the question whether a given neural network topology can be mapped on a chip. Also something must be said about the possible ways of interconnecting those neurons and weights: the reconfigurability of the chip. Also counting numbers of bits for the accuracy of inputs and weights does not give the complete picture of the overall accuracy of the chip, because there might be many more sources of error. Finally, measuring speed in ``Connections Per Second`` (CPS) clearly is biased too much in favour of chips that make inaccurate connections.
Several newly proposed criteria, which are defined in this thesis, address these objections. First of all, criteria coming from neural network theory have been reviewed. For example the term *capacity* of a neural network can be defined in several ways, such as the size of the set of all possible functions from input to output, which can be obtained by counting those functions if this size is finite or by counting the size of its ε-cover if it is infinite. Also the popular VC-dimension is a definition of capacity for a neural network. Knowledge of any of these capacities would make estimations possible of the number of samples needed in a training set to guarantee performance on learning and generalization.

However interesting and promising, either to get numbers from these theoretical criteria is intractable or they are to weak too say something about performance of *hardware*. In other words, their practicality seems to be insufficient. Despite this fact, they do give hints about what would be a good definition of a hardware related performance criterium.
The first newly proposed hardware related performance criterium is the ``Reconfigurability Number``, which counts the number of possible different (fully connected) feedforward networks that can be mapped on the chip and therefore expresses in one number something about the size and the reconfigurability of a neural network chip. Next ``Weight Memory Capacity`` integrates the number of weights and their bit-accuracy and is a straightforward, easy-to-obtain definition of capacity inspired from neural network theory. The ``Signal to Noise Ratio`` (SNR) of a chip measures the inaccuracies of the chip independently of their sources by comparing outputs of the chip to an idealized equivalent of it. The same kind of setup can be used to measure the effect of systematic errors in the chip by trying to compensate for them according to a given learning algorithm. This experiment establishes also the fitness of a learning algorithm to be used with a chip, because not only the amount or magnitude of the systematic errors is important, but also the ability of the learning algorithm to deal with them. Finally, a new speed criterium has been proposed, normalized to the effective accuracy of a connection: The ``Effective Connection Primitives Per Second`` (ECPPS), in which the SNR and the CPS criteria are integrated. This criterium counts the number of connection primitives that a suitably defined ``equally accurate equivalent`` of the neural network chip makes in a second.
Some of the proposed criteria involve measurement, so experiments have been performed on the ETANN chip from the Intel corporation to get figures of the Signal to Noise Ratio of that chip. Therefore, first its systematic errors were learned by the back-propagation algorithm. From this experiment it could be concluded that ETANNs main source of error is the shape of its sigmoid. Eliminating this as a source of error gives a factor 3 improvement on the residual error after learning. The SNR and ECPPS of the ETANN chip have been established, but no conclusions can be drawn from these figures until the same experiments have been performed on several other chips.

The contribution of this work is mainly towards the definition stage of good criteria to evaluate the performance of neural network hardware implementations. All the proposed criteria deserve and need full attention on their own to be able to establish their strengths, weaknesses and practical importance. Therefore measurement procedures must be standardized in every detail to eliminate biases due to differences in (details of) measurements. Next, actual chips must be subjected to them, and from the numbers that are obtained, then, conclusions can be made on if a certain criterium fits well to the intuitive idea of performance and if the resulting numbers are really unbiased. Before this

can happen however, it is only after sufficient experience in applying them that a sensible conclusion can be given on the completeness of a set of criteria for the judgement of neural network hardware performance. Until this moment, criteria have to be proposed, rejected, standardized and above all used. Once commercial interest in neural networks really begins to rise, no doubt, all this effort will be made with substantial speed and care, for what is a better way to advertize a neural network chip other than by the providing numerical figures that show good performance on an accepted set of hardware performance criteria.

Appendices

Appendix 1: Tables of conventional criteria

Table A1.1: Conventional criteria for size and accuracy

Chips	Size and Accuracy						
	N_n	N_s	N_w	CPN	b_x	b_w	LA
CNAPS	64	64	128K 256K 2M	2K 4K 32K	8	16 8 1	-
Lneuro	1	16	512 1K	16 .. 256	16 .. 1	16 8	-
NBS	8	8	-	-	8	16	-
PDN	13	832	832	64	8	8	-
WSI	576	576	36K	64	9	8	-
WSI-BP	144	144	9K	64	9	8	BP
10^6-S NN	256	10^6	10^6	1024		8	-
Ni1000	1024		256K	256		5	RCE PNN
TDM NN	1	1	-	-		16	-
ETANN	64	10240	10240	64/128	6	6	-
BiCMOS	64	768	768	32/16			-
NET32K	256	32K	32K .. 8K	128	1 .. 4	1 .. 4	-
ANNA	8	512	4096	16 .. 256	3	6	-
336 BNU	336	28K	28K	168	1	8	BZ
400 BNU	400	40K	40K	200	1	8	BZ
2-Chip	24	576	576	24		8	BP HEB
URAN	-	135K	135K	-		8	-

Table A1.2: Conventional criteria for speed

Chips	Speed			
	CPS	CPSPW	CUPS	CUPSPW
CNAPS	0.8G 1.6G 12.8G	6.1K	-	-
Lneuro	0.1G	97.6K	-	-
NBS	10M	-	-	-
PDN	8.0G	9.6M	-	-
WSI	1.24G	33.6K	-	-
WSI-BP			0.28G	30K
10^6-S NN	1.37G	1.37K	-	-
Ni1000	10G(?)	0.38M(?)		
TDM NN			-	-
ETANN	2.0G	0.2M	-	-
BiCMOS	0.1G	0.13M	-	-
NET32K	320G	9.8M	-	-
ANNA	10G	2.4M	-	-
336 BNU	1T	35M	28G	1M
400 BNU	2T	50M	80G	2M
2-Chip	90M	0.15M		
URAN	200G	1.5G	-	-

Table A1.3: Conventional criteria for cost

Chips	Cost			
	ESA (μm^2)	PD (Watt)	EPC (Joule)	EPCU (Joule)
CNAPS		4.0	5.0m 2.5n 0.3n	-
Lneuro				-
NBS		0.2	20	-
PDN	100K	54m	6.8p	-
WSI	410K			-
WSI-BP				
10^6-S NN	273	75m	54p	-
Ni1000				
TDM NN				-
ETANN	2000	2.25	1.1n	-
BiCMOS				-
NET32K	960			-
ANNA				-
336 BNU	4900	3.0	3.0p	107p
400 BNU	3025	4.5	2.3p	56p
2-Chip	80K	0.25	28n	
URAN	1124			-

Appendix 2: Table of new criteria

Table A2.1: New criteria

Chips	New criteria			
	RN	WMC	b	BCPPS
CNAPS	128K 256K $5.7 \cdot 10^6$	256 kB	8.5 8 1.5	57.8G 102.4G 28.8G
Lneuro		1 kB		
NBS		0	8.5	0.72G
PDN	832	832 B	8	512G
WSI		36 kB	8.3	85.5G
WSI-BP		9 kB	8.3	
10^6-S NN		10^6 B		
Ni1000		160 kB		
TDM NN		0		
ETANN	$4 \cdot 10^6$	7.5 kB	6	72G
BiCMOS	768			
NET32K		4 kB	1	320G
ANNA		3 kB	3.5	122G
336 BNU		28 kB	1.5	2.25T
400 BNU		40 kB	1.5	4.5T
2-Chip		576 B		
URAN		135 kB		

Appendix 3: Results of the fitting of ETANNs sigmoids

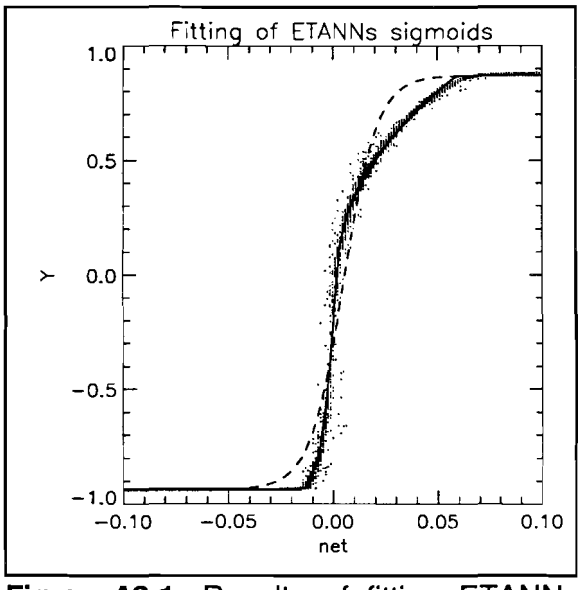

Figure A3.1: Results of fitting ETANNs sigmoids, net against y

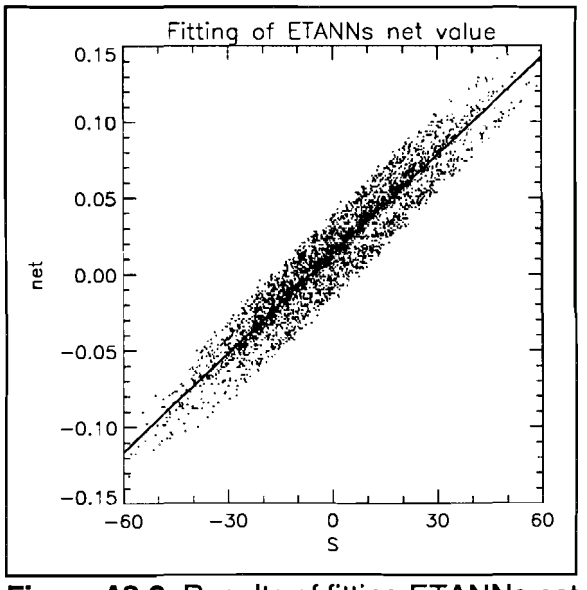

Figure A3.2: Results of fitting ETANNs net value

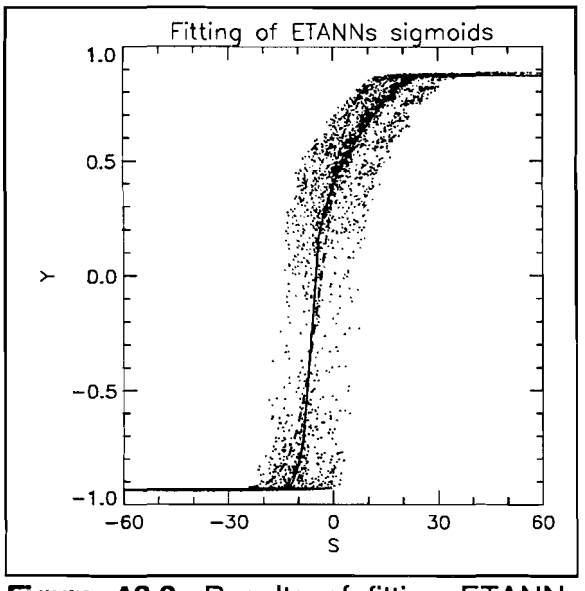

Figure A3.3: Results of fitting ETANNs sigmoids, s against y

Appendix 4: Results of learning with ideal sigmoid

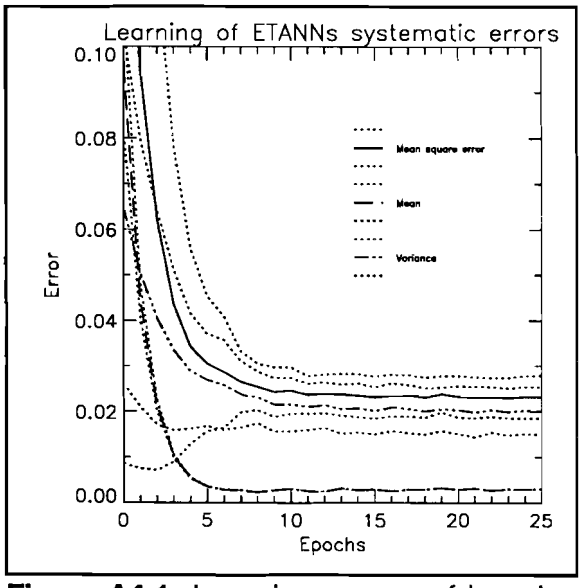

Figure A4.1: Learning curves of learning systematic errors with the ideal sigmoid

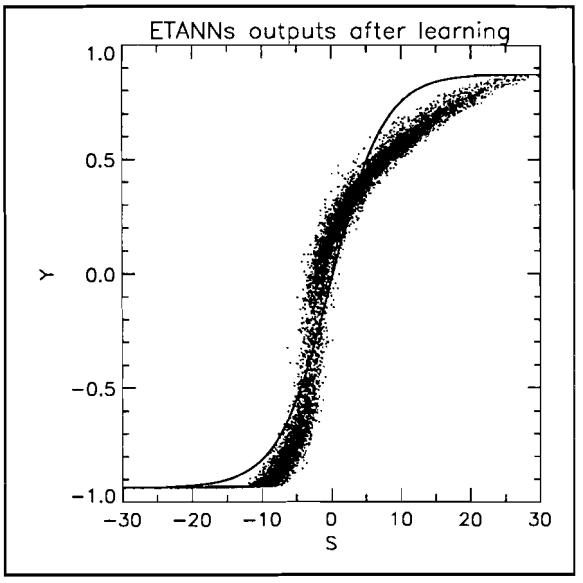

Figure A4.2: ETANNs outputs after learning with the ideal sigmoid,s against y

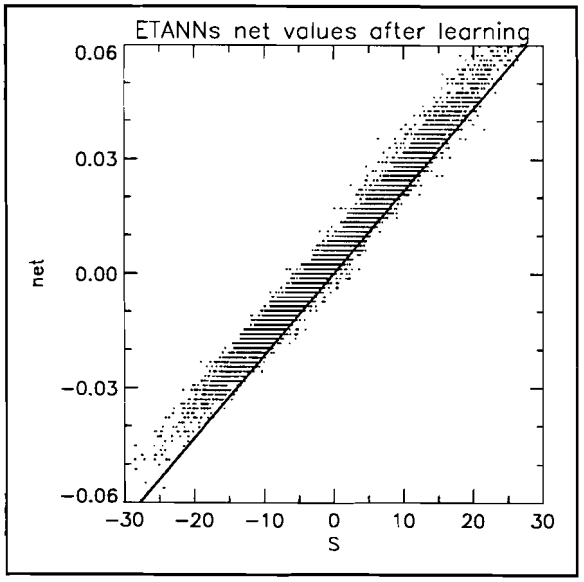

Figure A4.3: ETANNs net values after learning with the ideal sigmoid

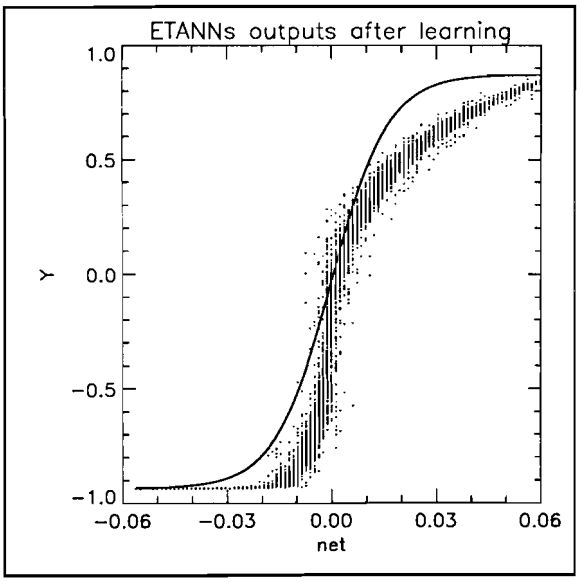

Figure A4.4: ETANNs outputs after learning with ideal sigmoid, net against y

Appendix 5: Results of learning with ETANNs sigmoid

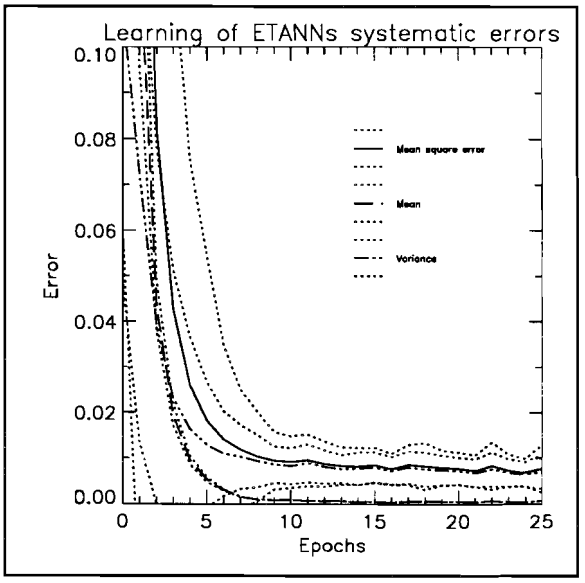

Figure A5.1: Learning curves of learning systematic errors with ETANNs sigmoid

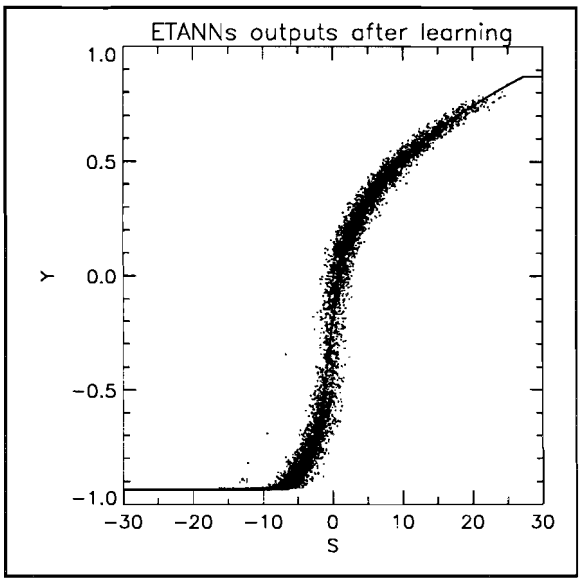

Figure A5.2: ETANNs outputs after learning with ETANNs sigmoid, s against y

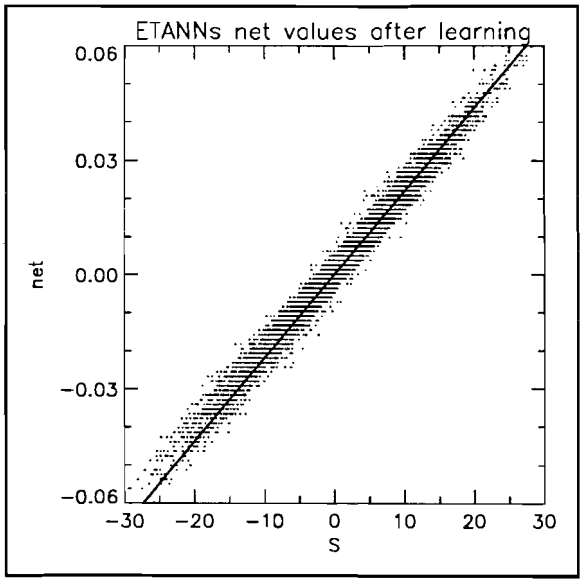

Figure A5.3: ETANNs net values after learning with ETANNs sigmoid

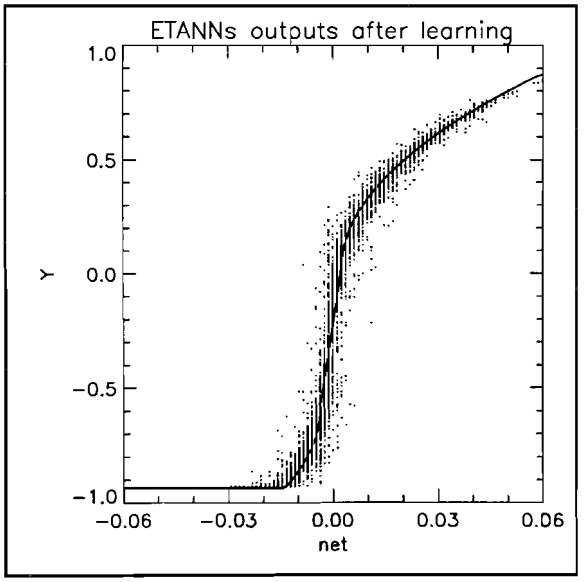

Figure A5.4: ETANNs outputs after learning with ETANNs sigmoid, net against y

References

[1] Geman S. and Bienenstock E., Doursat R., "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.

[2] Bolt G. R., *Fault Tolerance in Artificial Neural Networks.* PhD thesis, University of York, November 1992.

[3] Jackson D. and Hammerstrom D., "Distributing back propagation networks over the intel iPSC/860 hypercube," in *1991 International Joint Conference on Neural Networks* [67], pp. I,569–574.

[4] Hammerstrom D., "A VLSI architecture for high-performance, low-cost, on-chip learning," in *1990 International Joint Conference on Neural Networks* [68], pp. II,537–544.

[5] Means E. and Hammerstrom D., "Piriform model execution on a neurocomputer," in *1991 International Joint Conference on Neural Networks* [67], pp. I,575–580.

[6] McCartor H., "Back propagation implementation on the adaptive solutions CNAPS neurocomputer chip," in Lippman *et al.* [64], pp. 1028–1031.

[7] Griffin M. and Tahara G., Knorpp K., Pinkham R., Riley B., "An 11-million transistor neural network execution engine," in *1991 International Solid-State Circuits Conference* [69], pp. 180–181.

[8] Hammerstrom D. and Nguyen N., "An implementation of Kohonen's self- organizing map on the Adaptive Solutions neurocomputer," in Kohonen *et al.* [65], pp. 715–720.

[9] Manduit N. and Duranton M., Gobert J., Sirat J.-A., "Lneuro 1.0: A piece of hardware LEGO for building neural network systems," *IEEE Transactions on Neural Networks*, vol. 3, pp. 414–421, May 1992.

[10] Aglan F. and Bru B., Duranton M., Manduit N., Frydlender H., "Lneuro boards: Implementing some applications," in *1991 Second International Conference on Microelectronics for Neural Networks* [70], pp. 447–453.

[11] Yestrebsky J. and Paul B., Jerry R., "Neural bit-slice computing element," Tech. Rep. TP102600, Micro Devices, 1990.

[12] Micro Devices, *MD1220 Neural Bit Slice, Data Sheet*, March 1990.

[13] Uchimura K. and Saito O., Amemiya Y., "An 8G connections- per-second 54mW digital neural network chip with low power chain-reaction architecture," in *1992 International Solid-State Circuits Conference* [71], pp. 134–135.

[14] Yasunaga M. and Masuda N., Asai M., Yamada M., Masaki A., Hirai Y., "A wafer scale integration neural network utilizing completely digital circuits," in *1989 International Joint Conference on Neural Networks* [72], pp. II, 213–217.

[15] Yasunaga M. and Masuada N., Yagyu M., Asai M., Yamada M., Masaki A., "Design, fabrication and evaluation of a 5-inch wafer scale neural network LSI composed of 576 digital neurons," in *1990 International Joint Conference on Neural Networks* [68], pp. II,527–535.

[16] Yasunaga M. and Masuda N., Yagyu M., Asai M., Shibata K., Ooyama M., Yamada M., Sakaguchi T., Hashimoto M., "A self- learning neural network composed of 1152 digital neurons in wafer-scale LSIs," in *1991 International Joint Conference on Neural Networks* [62], pp. 1844–1849.

[17] Yasunaga M. and Masuda N., "A self-learning digital neural network using wafer-scale LSI," *IEEE Journal of Solid State Circuits*, vol. 28, no. 2, pp. 106–114, 1993.

[18] Watanabe T. and Kimura K., Aoki M., Kakata T., Itoh K., "A single 1.5V digital chip for a 10 6- synapse neural network," in *1992 International Joint Conference on Neural Networks* [61], pp. II,7–12.

[19] Tsuchiya C. and Sugiura Y., Iwamoto H., "The first commercial neurochip," tech. rep., Fujitsu Ltd., June 1989.

[20] Holler M. and Tam S., Castro H., Benson R., "An electrically trainable analog neural network (ETANN) with 10240 'floating gate' synapses," in *1989 International Joint Conference on Neural Networks* [72], pp. 191–196.

[21] Intel Corporation, Santa Clara, *80170NX Electrically Trainable Analog Neural Network, Experimental Data Sheet*, June 1991.

[22] Holler M., ed., *80170NX Neural Network Technology and Applications.* Santa Clara: Intel Corporation, 1992.

[23] Morishita T. and Youichi T., Otsuki T., "A BiCMOS analog neural network with dynamically updated weights," in *1990 International Solid-State Circuits Conference* [73], pp. 142–143.

[24] Graf H. P. and Janow R., Henderson D., Lee R., "Reconfigurable neural net chip with 32K connections," in Lippman *et al.* [64], pp. 1032–1038.

[25] Graf H. P. and Henderson D., "A reconfigurable CMOS neural network," in *1990 International Solid-State Circuits Conference* [73], pp. 144–145.

[26] Boser B. E. and Sackinger E., "An analog neural network processor with programmable network topology," in *1991 International Solid-State Circuits Conference* [69], pp. 184–185.

[27] Boser B. E. and Sackinger E., Bromley J., leCun Y., Howard R. E., Jackel L. D., "An analog neural network processor and its application to high-speed character recognition," in *1991 International Joint Conference on Neural Networks* [67], pp. I,415–420.

[28] Boser B. E. and Sackinger E., Bromley J., leCun Y., Jackel L. D., "Hardware requirements for neural network pattern classifiers," *IEEE Micro*, pp. 32–39, February 1992.

[29] Sackinger E. and Boser B. E., Bromley J., LeCunn Y., Jackel L. D., "Application of the ANNA neural network chip to high-speed character recognition," *IEEE Transactions on Neural Networks*, vol. 3, pp. 498–505, May 1992.

[30] Arima Y. and Mashiko K., Okada K., Yamada T., Maeda A., Notani H., Kondoh H., Kayano S., "A 336-neuron 28k-synapse self-learning neural network chip with branch-neuron-unit architecture," in *1991 International Solid-State Circuits Conference* [69], pp. 182–183.

[31] Mashiko K. and Arima Y., Okada K., Murasaki M., Kayano S., "Silicon implementation of self-learning neural networks," in *Proceedings of the 1991 IEEE International Symposium on Circuits and Systems*, (Singapore), pp. 1279–1282, June 1991.

[32] Arima Y. and Murasaki M., Yamada T., Maeda A., Shinohara H., "A refreshable analog VLSI neural network chip with 400 neurons and 40k synapses," in *1992 International Solid-State Circuits Conference* [71], pp. 132–133.

[33] Shima T. and Kimura T., Kamatani Y., Itakura T., Fujita Y., Iida T., "Neuro chips with on- chip backprop and/or hebbian learning," in *1992 International Solid-State Circuits Conference* [71], pp. 138–139.

[34] Han I. and K.H. A., "Neural network chip implementation of analog-digital mixed operation for more than 100,000 connections," in *1993 Third International Conference on Microelectronics for Neural Networks* [66], pp. 159–162.

[35] Watts L. and Kerns D. A., Lyon R. F., Mead C. A., "Inproved implementation of the silicon cochlea," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 692–700, May 1992.

[36] Maher M.-A. C. and DeWeerth S. P., Mahowald M. A., Mead C. A., "Implementing neural architectures using analog VLSI circuits," in Sanches-Sinencio and Lau [63], pp. 209–217.

[37] Achyuthan A. and Elmarsy M., "A methodology for selection of analog hardware for back-propagation algorithm implementation," in *1992 International Joint Conference on Neural Networks* [61], pp. II,672–677.

[38] Piche S., *Selection of Weight Accuracies for Neural Networks.* PhD thesis, Stanford University, May 1992.

[39] Zazula R., "Hardware implementations of artificial neural networks," tech. rep., January 1992.

[40] Graf H. P. and Sackinger E., Boser B., Jackel L. D., "Recent developments of electronic neural nets in the US and Canada," in *1991 Second International Conference on Microelectronics for Neural Networks* [70], pp. 471–488.

[41] Hirai Y., "Hardware implementation of neural networks in japan," in *1991 Second International Conference on Microelectronics for Neural Networks* [70], pp. 435– 446.

[42] Holler M. A., "VLSI implementations of learning and memory systems: A review," in Lippman *et al.* [64], pp. 993–1000.

[43] Haussler D., "Decision theoretic generalizations of the PAC model for neural net and other learning applications," *Information and Computation*, vol. 100, no. 1, pp. 78–150, 1992.

[44] Vapnik V., *Estimation of Dependences Based on Empirical Data.* Springer Series in Statistics, New York: Springer-Verlag, 1982.

[45] Valiant L., "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[46] Vapnik V. N. and Chervonenkis A. Y., "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and its Applications*, vol. XVI, no. 2, pp. 264–280, 1971.

[47] Baum E. B. and Haussler D., "What size net gives valid generalization?," *Neural computation*, vol. 1, no. 1, pp. 151–160, 1989.

[48] Cover T. M., "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Transactions on Electronic Computers*, vol. EC-14, pp. 326–334, 1965.

[49] Vapnik V., "Measuring the capacity of a learning machine (I)," in *Snowbird Conference*, 1992.

[50] Levin E. and Le Cun Y., Vapnik V., "Measuring the capacity of a learning machine (II)," in *Snowbird Conference*, 1992.

[51] Satyanarayana S., *Analog VLSI Implementation of Reconfigurable Neural Networks.* PhD thesis, Columbia University, New York, NY, 1991.

[52] Satyanarayana S. and Tsividis Y. P., Graf H. P., "A reconfigurable VLSI neural network," *IEEE Journal of Solid- State Circuits*, vol. 27, pp. 67–81, January 1992.

[53] Stevenson M. and Winter R., Widrow B., "Sensitivity of feedforward neural networks to weight errors," *IEEE Transactions on Neural Networks*, vol. 1, pp. 71–80, March 1990.

[54] Frye R. C. and Rietman E. A., Wong C. C., "Back-propagation learning and nonidealities in analog neural network hardware," *IEEE Transactions on Neural Networks*, vol. 2, pp. 110–117, January 1991.

[55] Castro H. A. and Tam S. M., Holler M., "Implementation of an analog non- volatile neural network," in Holler [22], pp. 1–28.

[56] Press W. and Flannery B., Teukolsky S., Vetterling W., *Numerical Recipes*. New York: Cambridge University Press, 1986.

[57] Diamond , J. p. c., "Intel and nestor deliver second-generation neural network cip to DARPA," Feb 1993. Press-Release.

[58] Widrow B. and Lehr M. A., "30 years of adaptive neural networks: Perceptron, madaline and backpropagation," *Proceedings of the IEEE*, vol. 78, pp. 1415–1442, September 1990.

[59] Cohn D. and Tesauro G., "How tight are the vapnik-chervonenkis bounds?," *Neural Computation*, vol. 4, no. 2, pp. 249–269, 1992.

[60] Bartlett P. L., "Vapnik-chervoninkis dimension bounds for two- and three-layer networks," *Neural Computation*, vol. 5, pp. 371–373, 1993.

[61] IEEE ans INNS, *International Joint Conference on Neural Networks*, (Baltimore), June 1992.

[62] IEEE and INNS, *International Joint Conference on Neural Networks*, (Singapore), Oktober 1991.

[63] Sanches-Sinencio E. and Lau C., eds., *Artificial Neural Networks*. IEEE Press, 1992.

[64] Lippman R. P. and Moody J. E., Touretzky D. S., eds., *Advances in Neural Information Processing Systems 3*. San Mateo, CA: Morgan Kaufmann Publishers, 1991.

[65] Kohonen T. and Makisara K., Simula O., Kangas J., eds., *Artificial Neural Networks*. North- Holland: Elsevier Science Publishers B.V., 1991.

[66] University of Edinburgh, *Third International Conference on Microelectronics for Neural Networks*, (Edinburgh, Scotland), May 1993.

[67] IEEE and INNS, *International Joint Conference on Neural Networks*, (Seattle), July 1991.

[68] IEEE and INNS, *International Joint Conference on Neural Networks*, (San Diego), June 1990.

[69] IEEE, *International Solid-State Circuits Conference*, (San Francisco), February 1991.

[70] Technical University of Munich, *Second International Conference on Microelectronics for Neural Networks*, (Munich), October 1991.

[71] IEEE, *International Solid-State Circuits Conference*, (San Francisco), February 1992.

[72] IEEE and INNS, *International Joint Conference on Neural Networks*, (Washington DC), June 1989.

[73] IEEE, *International Solid-State Circuits Conference*, (San Francisco), February 1990.