5.3 Neural Networks

The neural networks are one of the hottest topics in modern signal processing of these days. They become very popular quickly and full many a neural solutions of different kind of problems have been born. Unfortunately, most of these have been stayed at theoretical level and in many cases there are no practical realizations of them at best some simulations. This is because of not so trivial to find such kind of implementation, which saves the advantages of the neural structure e.g. low level parallelism, massively parallel structure and the advantages come from it (e.g. fault tolerance), adaptivity, speed, etc.

5.3.1 An overview

The term *Neural Networks (NNs)* denotes a lot of different models which have some common points. The main ones are that they are composed by many simple units, the so called *neurons* or *processing units (PU)* generally forming a regular structure and using learning by input-output sample pairs to achieve the desired function. Neural network architectures are based on ideas derived from many years of research into the way in which the human brain functions. The claim is that massively parallel adaptive networks of simple processing units can recall information associatively from incomplete or noisy inputs, and can derive optimal mapping operations solely by generalization from examples. Although, these networks solve very wide and different classes of problems, they have some revealing properties. Hereafter, we recite these and the main network casts to demonstrate the task needed to face during their implementation.

The networks have two working modes:

- the so called *learning phase* when the network adjusts its weights for the sake of achieving the optimal projection.
- and the *recall phase* when the network performs the projection has been learned before.

The learning of the network can be *supervised* or *unsupervised*. Supervised learning means that the ideal output corresponding to the traing sample is compared to the current outputs of the network and the difference between them is computed. After it the weights are modified according to this error. If the network learns only by their input and output themselves then it is the so called *self-organization* or unsupervised learning. The learning and recall phases sometimes can not be detached. If the projection should be realized by the network depends on the input data themselves then the network must be *adapted* in the recall phase also. This means that the learning must go until recall, so not enough to implement only the pre-trained network, but the training algorithm must be implemented also. It arises a basic question of the network implementation namely: have the learning algorithm to be implemented or it is enough to realize a fixed pre-trained network?

Thus, fixed and adaptive networks can be distinguished, the fixed ones are trained off-line while the adaptive ones learn on-line. This is very a important question because the implementation of the learning rule which enables the adaptivity of the network is mostly very expensive. Certainly, a given type network - e.g. a *Multi Layer Perceptron* - can be used as a fixed network in one application and the same type network can be used in another (or possibly the same or very similar) application as an adaptive one.

The structure of a neuron

The neurons are multi-input single-output devices mostly using local memory. The neurons generate the weighted linear combination of their inputs using the weights are stored in their local memory. The output of the linear combiner is connected to a general, mostly strongly nonlinear function the so called *activation* or *squashing* function producing the final output of the neuron (Fig. 5.1). The activation function is generally one of the step, tangent hyperboloid, sigmoid or the simple linear functions. This neuron model is general however, there are exceptions to this in where the neuron is not a simple linear combiner plus an activation function e.g. in the case of *Radial Basis Function* networks (See later in 5.3.2).

Consequently, a certain number of multiplications and additions corresponding to the number of the inputs and a nonlinear function are needed to realize a general neuron.

General Neural Network Topology

The most general network topology is the fully interconnected one where all neurons are connected to all the others as well as themselves. However, the networks are organized into layers in the most of the cases. This means that there are three different kind of layers:

- *input layer* consists of single input, single output neurons which are only buffer the inputs and have no processing role
- *hidden layer* which are connected only to input or output neurons

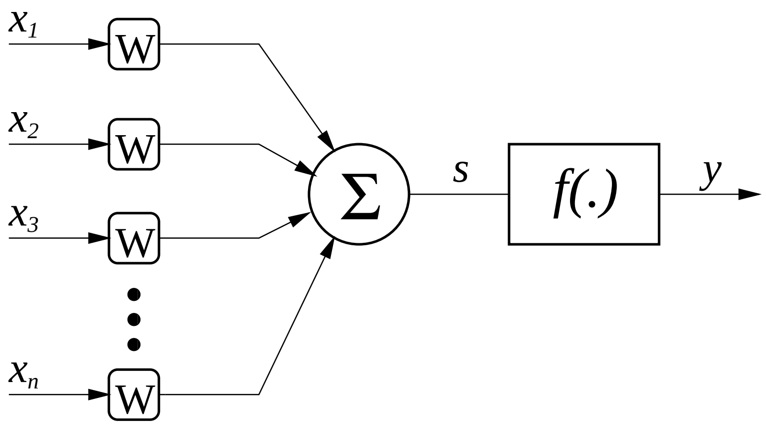

Figure 5.1: Neuron model

- *output layer* consist of neurons, which outputs are the output of the network at the same time.

Fig. 5.2 shows this topology. Nevertheless, two different basic classes of networks can be distinguished, the feedforward and the feedback ones. A feedback can be global if the output of the whole network is fed back to its input. The feedback is local if

- the output of a neuron is fed back to its inputs (*recurrent connection*)
- the output of a neuron is fed back to neurons in the same layer (*lateral* or *intra-layer connection*)
- the output of a neuron in one layer is fed back to the neuron in one of the previous layers (*inter-layer connection*). In this case the graph of the network contains closed loops.

Training strategies

For fixed network realizations this question is not relevant against the case of on-line training. Only the possible learning strategies, which are important in the implementation of on-line learning are discussed here:

- The learning strategies can be *mutual* or *individual*. Mutual training means the training of all the weights is influenced by all the output

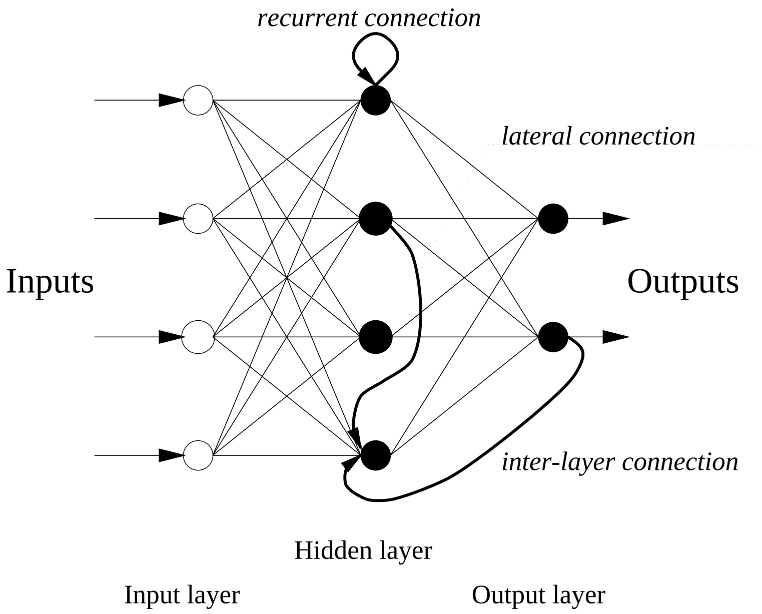

Figure 5.2: General Layered Topology of the Neural Networks

values. In individual training, the training of an individual subnet will not be influenced by the outputs of other subnets. The mutual learning strategy allows two neighboring categories to "negotiate" their mutual decision boundary what leads to more acute boundaries. The drawback is that it may take much more iterations to reach it.

- The networks can be trained *sample by sample* or in *batch mode*. Sample by sample training means that the weights are changed in every iteration step. In batch mode the changes of the weights are computed in every iteration and accumulated through several iterations. After these iterations the weights are modified with these accumulated values. It is also possible not to use all of the input samples - especially in the case of adaptive network, trained by unsupervised way - only train by every n*th* samples. This can be useful when the learning takes much more time than retrieving. This can be combined by batch learning.

- If on-line learning is used in many cases it is not enough to use a constant learning rate. The learning rate can be varied according a predetermined function of the time or a function of some property of the input or output. This must be considered in adaptive application.

5.3.2 Taxonomy of Neural Networks

The general topology of the networks is presented in the previous. Unfortunately, the network for different purposes using different learning rule. Moreover, there are some family of the networks which have different topology than it is detailed before. This is why the most frequently used networks are shortly introduced in the following emphasizing the questions concerning the realization. The networks can be grouped some different aspects:

By training method

- *Supervised learning networks*
- *Unsupervised learning networks*
- *Fixed network*

By topology

- *Feedforward networks*
- *Feedback networks*

By dynamics

- *Static networks*
- *Dynamic networks*; Feedback networks have inherent dynamic operation. This means that they do not provide their output in one shot after the input vector presented but only after a transient period.
- *Temporal networks*; Temporal networks have some different "dynamic" operation. This means that the output of the temporal network depends on not only the current input vector but the whole sequence of the input vectors, too.

A given network can be classified into more classes. Some revealing classifications are given in the followings.

Feedforward Networks Using Supervised Learning

Perceptron

The perceptron is a single layer network. It is built up of linear combiner neurons having a simple step function as an activation function at their outputs. Perceptrons use a simple LMS like learning rule:

$$\underline{w}_{k+1} = \underline{w}_k + \mu \varepsilon_k \underline{x}_k \tag{5.1}$$

where μ is the learning rate and ε is the difference between the current and the desired output value (supervised learning).

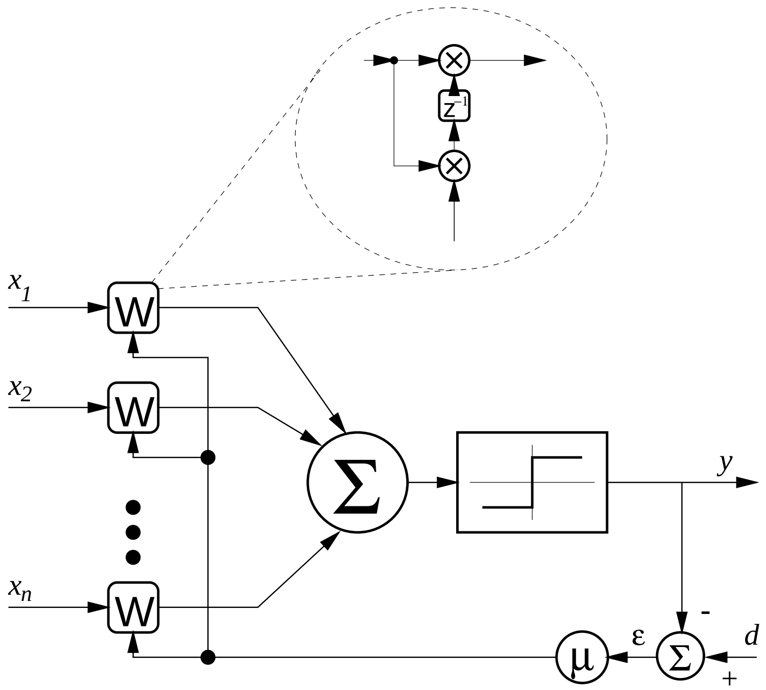

Figure 5.3: Perceptron

The signal flow of the perceptron can be seen in Fig. 5.3 including its learning rule.

Adaline

Adaline is a very similar network to the perceptron, the difference between them are only in that the interpretation of the output error and the type of the activation function. The output error is computed from the output of the linear combiner and not the output of the whole neuron (the output of the activation function). The learning rule is the LMS algorithm, formally same as at the perceptron (Eq. 5.1).

Multi Layer Perceptron, MLP

This topology is on of the most frequently used ones in the practice. Its activation function must be differentiable, typically it is sigmoid. The learning algorithm is the so called *back propagation (BP)* rule, which is similar to the LMS but can be used for multi layer networks because the output error of the hidden layers can be computed by the chain rule of the differentials. The topology of the MLP and the back propagation are in Fig. 5.4.

The Radial Basis Function Network, RBF

The RBF networks are simple three layer networks. The output layer consists of simple linear combiners. The neurons in the hidden layer perform the following projection:

$$y_i(\underline{c}, \underline{x}) = \sqrt{\sum_{j=1}^{n} (x_j - c_{ij})^2} \tag{5.2}$$

This second order function can also be extended to a more general elliptic-basis function. The signal flow of the RBF network is shown on Fig. 5.5

The Cerebeller Model Articulation Controller Network, CMAC

This network origins from perceptron and performs also a two step projection as the RBF network. The first step is to transform the input vectors to a so called association vector. This results that the input vectors are sparsely coded into these association vectors in the way that the association vector contains fixed number of "1"-s and the neighboring inputs are coded to neighboring vectors. Furthermore, the distance between the input samples is greater, the Hamming distance of the association vectors is also greater. Unfortunately, if the input dimension is too high or the quantization of the input is too fine then the association vector can be unrealizable long. This is why some compression of the association vector is used in the practical applications so the transformation is not invertible further more in this case. Hash coding is the most frequently used compression method.

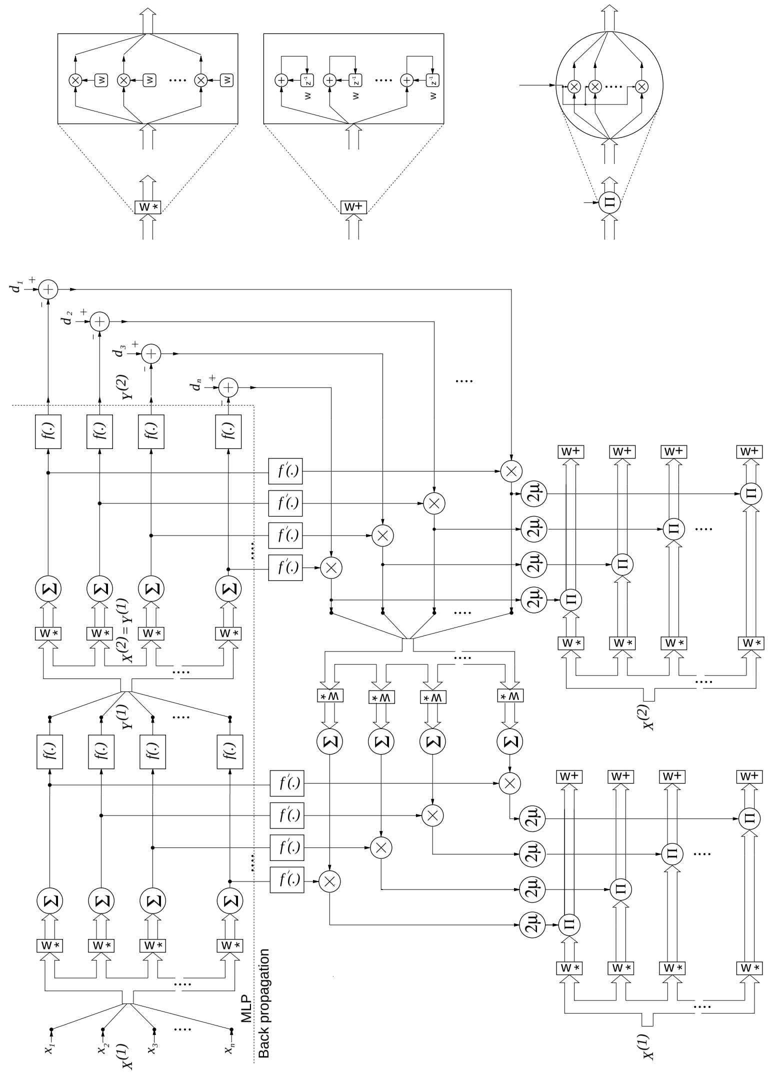

Figure 5.4: MLP with back-propagation

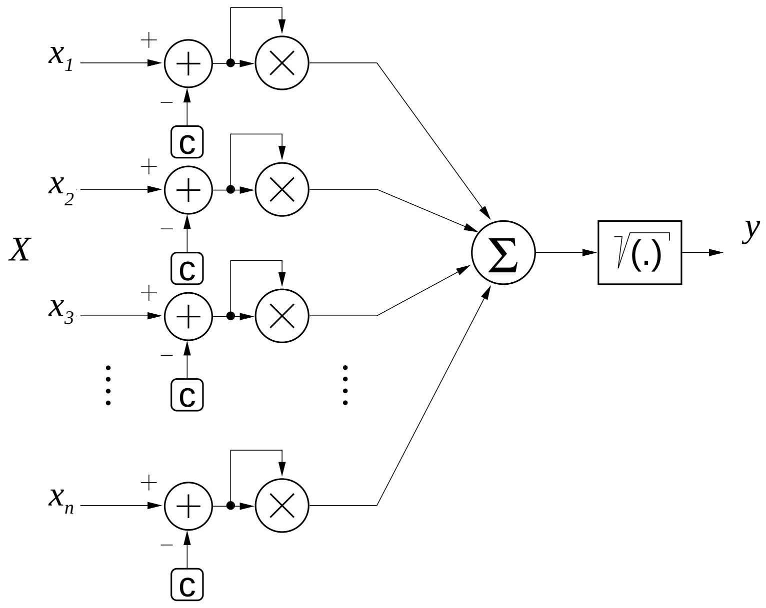

Figure 5.5: Single output RBF network

The output layer of the CMAC consists of simple linear combiners (Fig. 5.6) mostly trained off-line by supervised way. The weights are modified according to the output square error. Being the elements of the association vector are binaries (ones or zeros) the weights of the output neurons are simple selected or not. This makes the digital implementation easy because it does not require any multiplication only the sum of the fixed number of weights.

Unsupervised Learning Networks

Unsupervised learning networks significantly differ from supervised one. They do not use desired output values for their learning. This property makes them very suitable for adaptive applications because of their learning does not require any extra data only the input vectors which are available at recall phase. If the learning algorithm is also implemented and used in the recall phase for maintaining adaptivity, these nets naturally contain feedback themselves.

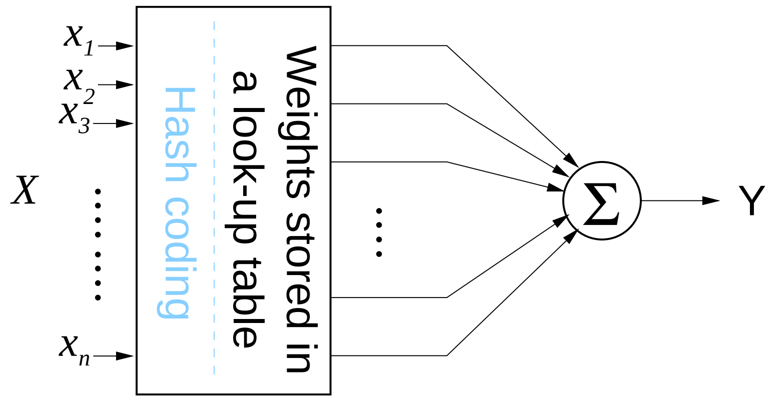

Figure 5.6: CMAC network

Kohonen Self-organizing Map

The Kohonen network consists of linear neurons in a two dimensional array (against the multi-layer structure as discussed before). Every neuron is connected to the every inputs and cover an output of the network. There are feedbacks in the network but only lateral ones in the columns. Kohonen nets use also the Hebbian learning rule, but only the weights of the neuron having the maximal output (the winner) or the first several one are modified (*competitive learning*). This requires the MAXNET circuit or threshold function (the lateral connections can automatically provide winner selection).

The MAXNET circuit is a special sorting circuit selecting the maximal output value. It can be digitally implemented using comparators or in analog way also using lateral connections as it is in Fig. 5.7. Assume that $\varepsilon > 0$, and the neurons have nonlinear activation functions with linear region (e.g. saturated ramp function) and we operate in this linear region in the beginning. If the initial values of all neurons are positive and the slope of the linear part of the nonlinearity is 1, than the system will force its all outputs to zero, except the unit that initialy has the largest value. If the slope is larger than 1, the final vector will have more than one non zero elements. If the threshold is absent, the network is unstable.

The modified Hebb rule is used to avoid the weights to increase over by weight normalization:

$$\Delta \underline{w}_i = \mu(\underline{x} - \underline{w}_i) \tag{5.3}$$

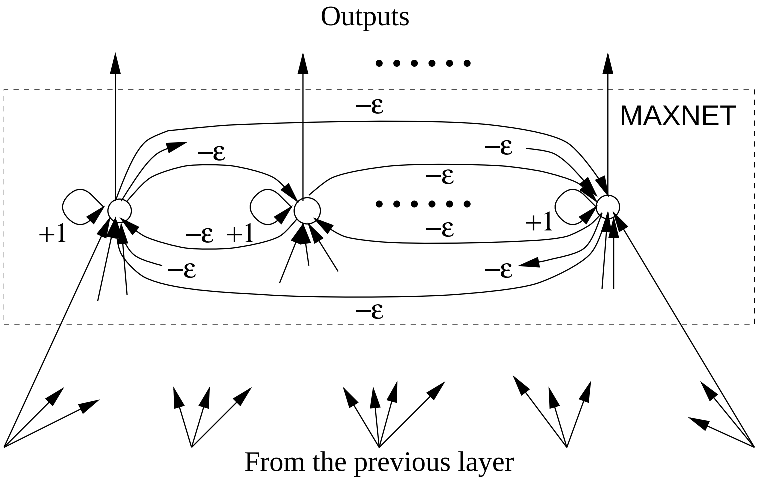

Figure 5.7: MAXNET circuit

There is a modification of the Kohonen net, which uses supervised learning. The so called *Learning Vector Quantization, LVQ* algorithm modifies the weights of the winner neuron by the Kohonen rule (Eq. 5.3) if it is the right winner and by the negalt one if it is not the excepted neuron.

Principal Component Analysis Networks, PCA

This family of the networks are particularly useful for data compression and feature extraction. The PCA net is a single layer linear net without feedback using the Hebbian-type learning (the so called *Oja subspace rule* (Eq. 5.4) or its modification).

$$\Delta \underline{w}_i = \mu(y_1 \underline{x} - y_1^2 \underline{w}_i) \tag{5.4}$$

This nets perform the *Karhunen-Loéve Transformation (KLT)* or similar orthogonal subspace transformation. There are many extensions of the PCA nets but two of them are emphasized here. One of them is the so called *Adaptive Principal-component Extractor (APEX)* which uses lateral connections for orthogonalization. The other is the non-linear version of the PCA nets *(NPCA)* which has non-linear activation functions and performs non-linear transformation (higher order statistics). The basic single output PCA net is on Fig. 5.8.

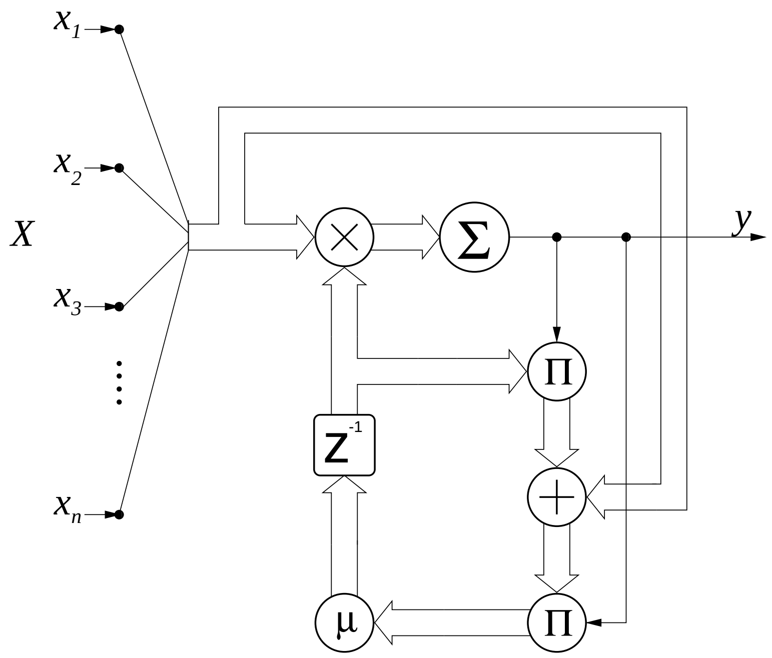

Figure 5.8: Single output PCA network

Neocognitron

The neocognitron is suitable for image recognition regardless of position and distortion. It has hierarchical multi-layer structure which processes the patterns hierarchically stage by stage from the input to the output layer. The level of representation in the layers increase towards the output layer. Each layer consists of sub-layers. Two types of the layers exist: one of them consists of simple feature-extracting cells, and there is another more complex position-readjusting layer type. The feature extractor layers use self organizing networks, the position readjusting cells have fixed connections. These two layers are successively follows each other in the hierarchical structure of the neocognitron (Fig. 5.9).

Fixed Feedforward Networks

The most typical members of this and the next (Fixed Feedback Networks) classes of the networks are the so called *Associative Memory, (AM)* net-

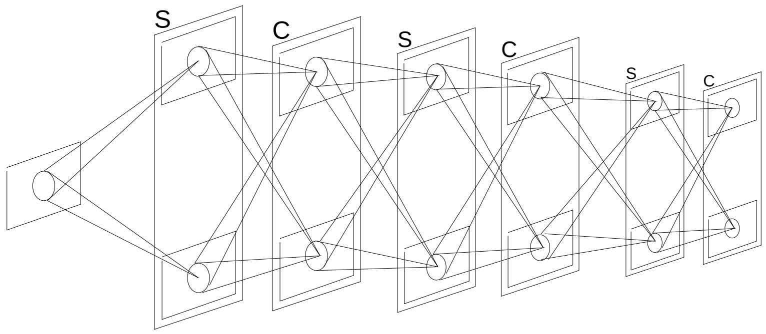

Figure 5.9: Neocognitron

works. These are off-line trained typically using the so called *Hebbian* learning rule or analytically trained (by computing an energy function) networks. Typical energy function minimizing methods are the *Stochastic Simulated Annealing (SSA)* and the *Genetic Algorithm (GA)*.

Linear Associative Memory, LAM

This fixed linear network retrieves the stored patterns in one shot. The LAMs are single layer networks consist of simple linear-output neurons performing a transformation from an input space to the output one. Its objective is to recover the output pattern based on the full or partial information of the input pattern. The AMs can be used for *auto-associative* or *hetero-associative* function. Auto-association means mapping to the same output pattern as the input one against hetero-association where the associated pattern is different from the input one. In practical application the LAM is rarely used because of the sensitivity to the input noise.

Nonlinear Associative Memory, NAM

Application of a nonlinear processing unit will eliminate unwanted perturbation caused by input noise. Two types of nonlinearity can be added to the LAM, a step (or threshold) function or the so called MAXNET (discussed above) circuit.

Hamming Networks

Hamming network can be used when the inputs are binary valued. It can be proved that in this case either the inner product measure or the Hamming-distance measure will yield the same effect. The MAXNET can be adopted to select the winner from the stored patterns, which has the least Hamming distance from the input vector. It is very suitable for digital implementation.

Fixed Feedback Networks

Hopfield Networks

The Hopfield network is a single layer feedback network with binary outputs (Fig. 5.10). The feedback network starts from the input pattern as an initial state and after many iterations will converge to a solution that minimizes an energy function pre-trained to the network.

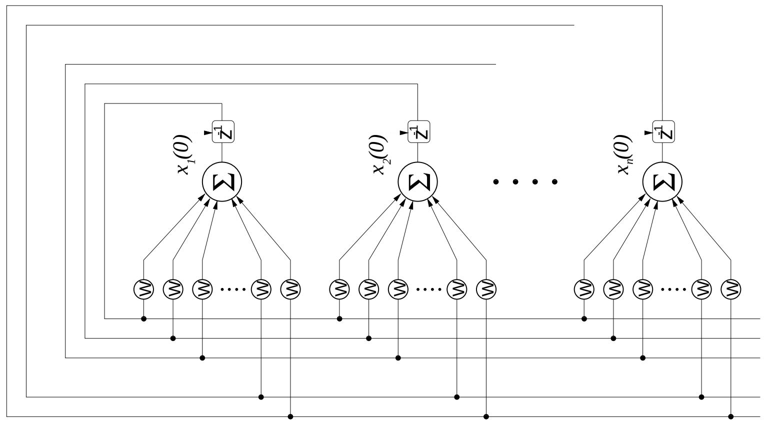

Figure 5.10: Hoppfield network

This networks operates in two different ways: the *sequential (asynchronous)* or the *parallel (synchronous)* mode. In synchronous mode every neurons change their states at the same time in every iteration step. This mode allows to exploit the potential of parallel processing. If the asynchronous update is used, only one neuron is updated in every iteration. The two modes are basically different. The system can change its state through successive neighboring states in sequential mode against parallel mode where the system can reach any arbitrary state in every step.

Boltzmann Machines

The Hopfield network can be stabilized in a false steady state at a local minimum of the energy function. Especially the sequential version of the Hopfield network is disposed to do this. One way to avoid the network to stuck to a local minimum making the learning stochastical. This means adding noise to the learning algorithm which helps the system to get out from local minima. This method is called as *Stochastic Simulated Annealing, (SSA)* in the literature. This extension of the Hopfield net is the Boltzmann Machine.

Cellular Neural Networks, CNN

The structure of the CNN is somewhat differ from that discussed before. The neurons are in a two- or three dimensional mesh and all of them are fully interconnected to the others in a circle with radius r. Interaction between these cells are limited to these finite local neighborhoods of each cell. The weights are analytically determined and they are identical for every neurons. Thus, the CNN network can be described by an r by r weight matrix the so called *template* which is space invariant. These networks work similarly to the Hopfield nets, the input pattern should be processed can be loaded to the inner state-variables or mapped to the inputs and after a transient the network serves the result at its output. CNN is typically implemented in analog way and used for fast binary image processing.

Temporal Networks

The networks treated before perform static projections. Temporal networks work on *temporal patterns*. The operation of these time dependent networks depends on not only the current input pattern but the previous ones and the sequence of the input vectors also. The most important classes of the temporal networks are the *Time-Delay Neural Networks (TDNN)* and the *Recurrent Neural Networks (RNN)*. Temporal networks mostly originate from static ones using the following extensions.

Time-Delay Neural Networks

This family of the networks apply *Tapped Delay Lines (TDL)* or *FIR filters* to show dynamic properties. The delay can be at the inputs or between the layers of a static feedforward network (Fig. 5.11). TDNNs do not contain feedbacks. The learning of the TDNNs typically can be derived from the back-propagation algorithm.

Recurrent Neural Networks

The RNNs achieve their dynamic nature by feedback which can be global or local. The feedback is global if the delayed output of the network is connected to its own inputs (as it is shown in section 5.3.1). The graph of an RNN is in Fig. 5.12. Their learning algorithm can be derived as an extension of the learning algorithm used for the conventional recursive filters.

5.3.3 Implementation Issues

Taxonomy of Implementations

Neural implementations can be classified in many ways. Some classes are given in the next by different respects. Certainly, a given implementation can belong to more than one of the following classes.

Application point of view

- **Simulators.** These softwares and hardwares generally provide flexible tools for developing new neural applications, algorithms and structures. They are suitable to train the weights for a fixed network realization. Mostly they have good, typically graphic user interfaces. Some of them are supported by special *hardware accelerator*s because of the high computational burden needed by these neural network computations.

- **"Real-world" applications.** These are the applications, that are not for new algorithm developing purposes but used in real-time real-world systems.

By measure of parallelism

- **Sequential implementations.** The fully sequential implementation is typically used by software simulators running on a general single processor hardware. The main disadvantages of the sequential implementations is the loss of almost everything why neural networks are used (e.g. speed, fault tolerance). Certainly, many level of parallelism can be distinguished between the fully sequential and fully parallel implementations. These will be detailed in the 5.3.4 and 5.3.5 sections.

- **Parallel implementations.** Typical parallel implementations are the softwares run on parallel systems (e.g. transputers) and generally the dedicated hardware solutions whether it is analog or digital. The level and type of parallelism are discussed below.

By tools

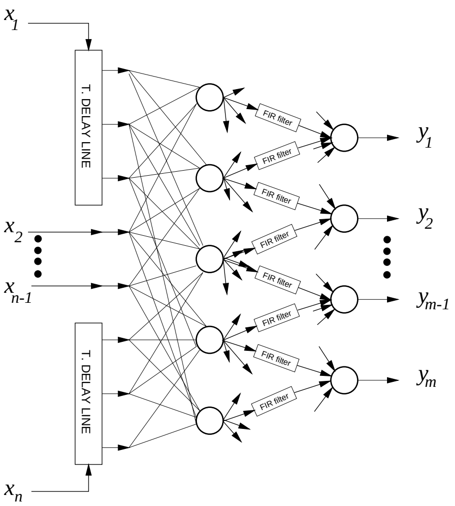

Figure 5.11: Time-Delay Neural Network

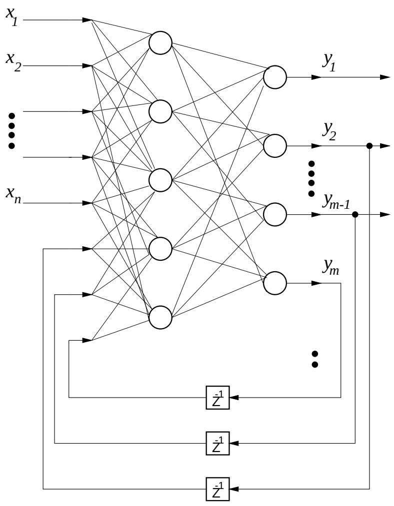

Figure 5.12: Recurrent neural network

- **Software tools.** The properties of a software solution mostly depends on what kind of hardware will it be run on. The typical software tools are the simulators that can be run on a general single processor computer or on some parallel architecture. To increase the speed single processor computers are frequently supported by dedicated hardware (e.g. coprocessor or accelerator board). Real applications are mostly use some parallel and/or dedicated hardware. This is why the real advantage of the massively parallel nature of the nets (e.g computing speed, fault tolerance) can be achieved using this way.

- **Hardware tools.** Hardware tools will mean dedicated neural network hardwares in the sequel. There are a large variety of very different approaches have been developed during the last years. These range from one of the first (perhaps the first of all) learning machine built of artificial elements in 1951 by Edmonds and Minsky to special analog or digital neuro chips. The machine of Edmonds and Minsky consisted of tubes, potentiometers, motors and electric clutches. However, the whole machine was working even if some tubes or wires were wrong, showing the advantage of neural networks, namely their inherent fault tolerance.

Simulators

Simulators have great significance in development of new neural applications and new neural paradigms. They help the translation of the problem into neural network representation, which includes deciding upon the architecture and the learning algorithm. The validity of these decisions can be tested by simulators. They can be used for analyzing how the network works, how the results are achieved, how well the system generalize, and how sensitive it is to perturbations. Simulators usually have many statistical tools, and diagnostic facilities to do this. Data preparation for learning and test is supported by pattern editor and simulation script facility. The field of neuro-simulators grows up very fast. There are many commercial simulators but some others can be obtained freely from academic institutes.

Software simulators can be ranked into three classes:

- template
- non-template
- network specific

Template simulators

Template simulator allows ease of use at the expense of flexibility. They provide rapid construction of a net from a given set of the network families and easy modification of the model parameters. Defined nets can usually be called from C allowing nets to be incorporated into layer application. Usually they have a good user interface.

Non-template simulators

Such a simulator allows the user to define new ANN models or implement major modification to existing ones. A feature of most of these simulators is the provision of a well-developed graphical user interface. This enables the user to observe and understand unexpected features of the run-time behavior of a new net.

Network specific simulators

They are specific to a particular network type. Thus, the application area is also limited and determined by the given network type. Such a simulator especially can be used for developing real-world applications to determine the weight set of a fixed network. Usually they have simple interface.

Sequential Implementations

Sequential implementations means software solutions. Every neural algorithms can be implemented on von Neumann machines, but these implementations are mostly not enough efficient and fast. This is why sequential implementations are used mostly only for simulations. The sequential implementation of network algorithms makes it necessary to chose data structures to represent network states. This is why, the algorithms consists of states and operations corresponding to the learning rule, which must be stored in the main memory together. The general data structure is arrays of unit states together with weight matrix. If the weight matrix is large and sparse than this approach is unattractive. Fortunately, generally the weight matrix is regular so the calculation can be performed by simple array processing routines.

Implementation of regular nets

The network algorithms consist of a few matrix by vector, or vector by vector operations. Thus, the main efforts must be spent to optimize these operations. (For example, efficient implementations place the vector data "near" to the processor such as in cache memory or registers because a matrix by vector operation requires the vector data many times within an operation.)

Implementation of irregular nets

In general, the connection matrix for a neural net can be very large and sparse. One solution to this problem is the use of pointers and linked lists. Such a data structure was used in Rochester Connectionist Simulator. In RCS every node was represented by a data structure. Hanging off each node is a linked list of structures representing weights into the node. Each weight structure has a pointer to its source node. The RCS supports nets with more than one type of connection. Simulation using the RCS thus requires much pointer following.

Parallel Implementations

Parallelism is one of the most important inherent property of the neural networks. The implementations which want to exploit the potentials of the neural networks such as computing speed, fault tolerance have to be parallel. Certainly, these parallelism can be on different level and have different nature according to the type of implementations (e.g. analog or digital hardware, software, etc.). The common attributes of the different parallelisms are discussed here, the specific ones in the corresponding sections (5.3.5).

The data dependencies of neural network algorithms permit two main kinds of parallelism, namely *network parallelism* and *training parallelism.* Network parallelism occurs in the algorithms that implement a network's state or learning rules. Training parallelism is obtained by concurrently calculating the weight updates caused by different members of the training set. These independently computed weight updates are then combined to form the update of the network state. This typically occurs in the case of off-line learning. The training parallelism speeds up the learning but has no effect to the speed of the network in the recall phase.

Network Parallelism

- **layer parallelism.** Concurrent execution of layers within a network. Typically it means that one processor implements one layer.
- **node parallelism.** Parallel execution of nodes for a single input. It means that every node are computed by separate processors.
- **weight parallelism.** Simultaneous weighted summation within a node. This means that all of the arithmetic operations (weighting and summation) are computed parallel. This is the full parallel implementation that can use different number of processors for the same neuron according to the complexity of a processor. For example a

multiple input sum can be computed by a single processing unit, which has the corresponding number of inputs, or using several binary adders forming an adder tree. Moreover, these processors are not necessarily uniforms. The fully parallel computation is revealing to the dedicated hardware implementations, in which a node consists of multipliers, adders and a nonlinear element.

The granularity goes finer from the layer parallelism to the weight parallelism. The mentioned categorization of training and network parallelisms is valid for both the software and hardware implementations. Although, they are not equally typicals. The training parallelism is revealing to software simulators against the weight parallelism which is rather common in dedicated hardware systems. There is another kind of parallelism according to the calculating way of the digital arithmetics (local parallelism) that is discussed later (in section 5.3.5.

5.3.4 Software Implementations

Parallelism in Software Implementations

The problems of parallel programming that can prevent the linear speed up of parallel algorithms by increasing the number of processors are: the need for processor synchronization, limited algorithmic parallelism, runtime system overhead, data connection, and I/O bottleneck. These points are addressed in this section.

Whichever approach is taken, the training, the network parallelism or both, the resulting parallel algorithm can be made to have the following four properties:

- *Compile-time decomposition.* The decomposition of the algorithm into processes and the data flow between them can be determined at compile time. With network parallelism this occurs because networks have a fixed structure and a fixed method of operation. With training parallelism, each process corresponds to a portion of the whole training set and data flow corresponds to the exchange of weight update information. Compile time decomposition allows the planing of data flows and processor loads. Moreover, data can be sent to a processor without the processor's needing to request it. Thus, no overhead for runtime load balancing and no extra communication needed.

- *Regular decomposition.* The processes produced and the data flow is regular, in addition to being known at compile time. With network parallelism it occurs because networks have regular connection structure and a few unit types. With training parallelism, each process is doing the same work. The regular decomposition of an algorithm greatly simplifies programming. In addition, the regular decomposition may be modular and thus generalize to varying number of processors, depending on the number of processors is available.

- *Possibility of purely local data flow.* In both forms of parallelism, the process decomposition can be arranged so that each process communicates with a number of local processes. This does not mean the absolute lack of global communication. For example, in a ring structure all processes can access all others, but only local communication is used. Global data flow is difficult to implement on distributed memory systems and systems with only local interprocessor links. However on systems with broadcast communications of global fan-in, this requirement is less important.

- *Reuse of data.* An *I/O bound* computation is one where each data item takes part in one computation. A *compute bound* computation has data items that occur in many computation. In the case of compute bound computation it is possible to increase execution rate while keeping the same I/O bandwidth by increasing the number of processors. With I/O bound computation, it is not. In network parallelism, unit activations are shared, that is, the vector in a matrix calculation is reused. With training parallelism, the network weights are shared between all processes. Data reuse prevents a computation from being I/O bound, that is the limiting factor the rate at which the system operates.

Parallel Software Techniques

There is no consensus how to implement neural networks in software and how to take into account the architectural characteristics of available hardware because of requirement of the speed, the high number of parallel nodes and interconnections. Communication demands are very high because of the network's massive interconnections. Fast communication must be provided for efficient realization. The most frequently used communication schemas are broadcast, circulation, and general routing (message passing). Broadcast seems to be natural for neural processing, but there are architectures that provide faster data exchange using circulation or general routing. The communication strategy together with hardware characteristics and parallelization techniques determine the overall performance of the given implementation.

Depending on the presence or absence of central control, parallel computers can be divided into two categories: *data parallel* and *control parallel*. Data parallel architectures simultaneously process large data sets using centralized (typically SIMD) or regular (for example, pipelined) control flow. Control parallel architectures performs decentralized processing (MIMD). These two types of architecture require different programming approaches.

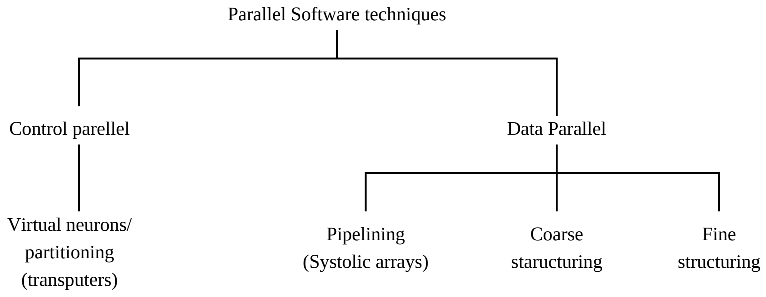

Figure 5.13: Taxonomy of parallel software techniques

Data parallel machines

A large amount of data is processed by many processors simultaneously in data parallel machines in a synchronous or regular fashion. All processors execute the same program (SIMD or SPMD *Single Program, Multiple Data*). They use boardcast or circular communication and centralized synchronization. The typical parallelization techniques are:

- **Coarse structuring**, providing training-example and node-per-layer parallelization. Node-per-layer parallelism means mapping a vertical set of neurons onto a processor in the way that at most one neuron belongs to a processor from each layer (Fig. 5.14). Thus, the total number of processors is equal to the number of nodes in the largest layer. This method yields mapping onto 1D (linear) processor array. Training parallelism can be used for 2D processor arrays.

- **Fine structuring**, providing single-node and weight parallelization. This means every nodes or weights are computed by different processors (Fig. 5.15.

- **Pipelining**, provides layer (and sometimes node) parallelization. Pipeline structures are mostly, but not necessary implemented on systolic arrays (general purpose parallel machines also can work in pipelined

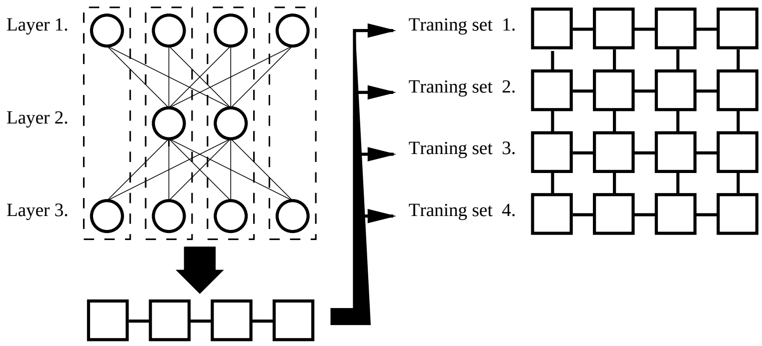

Figure 5.14: Mapping by coarse structuring

manner with rhythmical processing). This means numerous processors in an array performing simple operations in the way that a processor does the same partial operation all the time and passes the result to the following one (Fig. 5.15).

The most popular data parallel machines are the Connection Machine, MasPar, Warp, and systolic arrays. These are described in the 5.3.7 section.

Control parallel machines

Decentralized control, decentralized data distribution, and MIMD model characterize the control parallel machines. The parallel program is divided to processes and the processes are placed on different processors. Thus, the processors execute different programs on different data (MIMD). The communication schema is usually general routing which means message passing. The most popular control parallel machines are the transputers. The problem is how to distribute numerous neurons onto several processors. A possible solution is dividing the task to subtasks, for instance, vertically like in the case of coarse structuring, and mapping these subtasks onto the transputers. In this case the layers can be serially computed while a master transputer synchronizes computation.

As it is shown by Table 5.1 the control parallel implementations are outperformed by data parallel ones. However, the advantage of the coarse structuring approaches is surprising. One to one mapping (node and weight parallelism) has poor performance because of the high communication requirements and the due to simplicity of a neuron and the complexity of a processor.

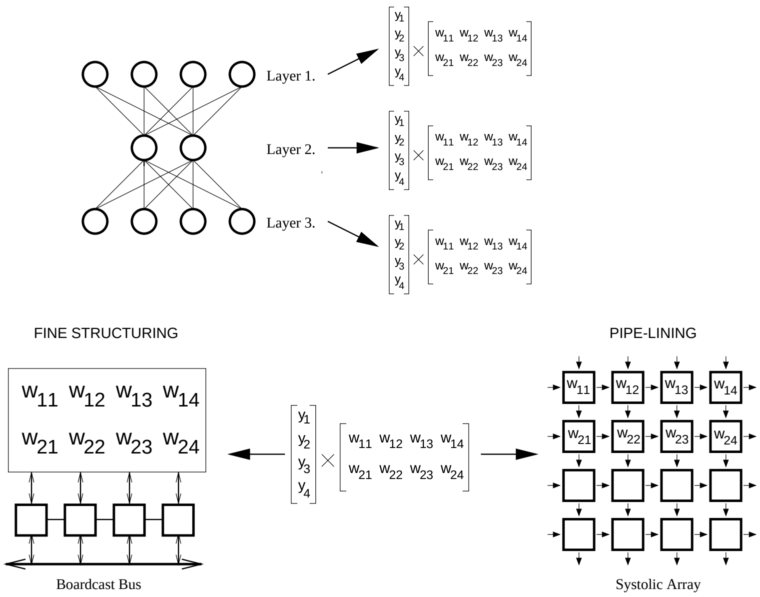

Figure 5.15: Mapping by fine structuring and pipelining

Mapping neural networks onto parallel architectures

Beyond the most frequently used mapping techniques detailed above there are some other approaches. The theoretical basis of mapping is based on deriving Dependence Graph (DG) from the neural algorithm, what is detailed for example in the book of S.Y. Kung (Digital Neural Networks). In DGs the *nodes* represent computation and *arcs* specify the data dependencies between computations. The processor assignment and schedule assignment can be derived from DG.

Hardwares for Software Implementations

The simulators and real-world applications, which are implemented in software, require hardware to run on it. The possible hardwares are categorized in this section.

The real-world applications are implemented rather on neurocomputers or maybe on general purpose parallel machines than accelerator boards or sequential computers. These latters and the general purpose parallel

Structuring technique	Machine	**No. of proc.**	Benchmark	**Performance**
Coarse: training, node per layer	Connection Machine	64k	NetTalk	180M CPS, 38M CUPS
Coarse: training, node per layer	MasPar	16k	NetTalk	176M CPS, 42M CUPS
Fine: node, weight	Connection Machine	64k	NetTalk	13M CUPS
Pipeline: training, layer	Warp	10	NetTalk	17M CUPS
Pipeline: layer, node	Systolic array	13k	NetTalk	248M CUPS
Control parallel	Transputer	6		207k CUPS

Table 5.1: Parallel implementations on general purpose machines

machines also generally used by simulators.

Computers for neuro-implementations should have some properties, like:

1. considerable **computational power**
2. sufficient **capacity** for large scale networks
3. sufficient **flexibility** for various network type
4. potential to implement new models

Sequential computers

Sequential computers are mostly used for neural network simulators. This is why they are quite slow and ineffective and can not exploit the inherent parallelism and features origin from parallelism of the neural nets. These machines are built with either CISC, RISC, DSP, integer or floating point processors with or without coprocessor.

Boards from commercial processors

General purpose processors have a great advantage, namely they are cheaper than the processors specially developed for neuro-hardware. Usually they provide much more programmability than it is required by neuro-implementations. Obviously the solutions using processors developed for

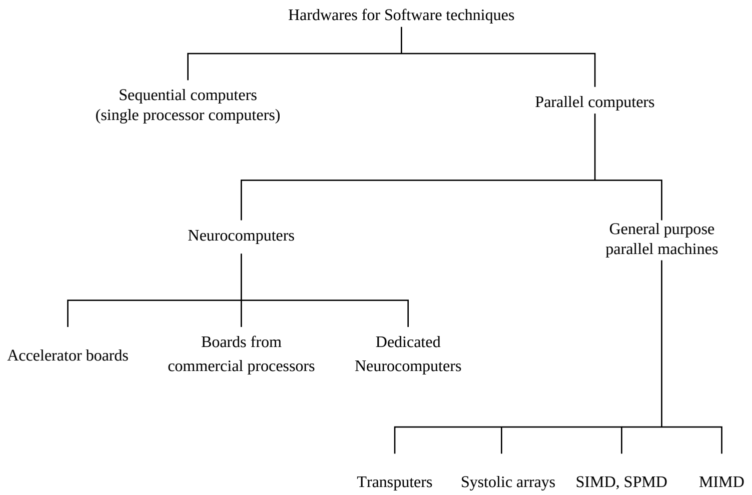

Figure 5.16: Classification of hardwares for software implementations

single or parallel systems can provide less efficiency than neurocomputers built with neurochips.

Accelerator boards

Accelerator boards are the most frequently used neural commercial hardware means especially for simulators, because they are relatively cheap, widely available, simple to connect to a PC or workstation, typically provided with user friendly software tools which have been already known by user in some cases, because they had used them in their non-accelerated standard systems. The available speed up to sequential implementations is about one order of magnitude. Usually these systems supports many different neural paradigms, but mostly they are not enough flexible to allow developing novel ones.

Dedicated Neurocomputers

These machines are specially developed for neural network implementations using dedicated neurochips. They speed up the neural implementations about 2 or more order of magnitude compared to other solutions. The implementation techniques of neurochips differ, they can be analog, digital, or hybrid as it is detailed in section 5.3.5. The main points in the design of neurochips and neurocomputers are the measure of parallelism versus speed, required chip surface and precision.

General purpose parallel machines

These machines were not specially designed for neuro-implementations, but many examples are in the literature for developing neural network implementations on general purpose parallel machines. This is because these machines are simple to achieve, and provide high performance especially for network simulations. Real applications are not so often implemented on general purpose machines because usually they are relatively expensive, big, and provide a lot of features which are non usable in neuro-implementations.

5.3.5 Dedicated Hardware Implementations

The two different implementation approaches, the software and the hardware follow two different attitudes. The software approach looks for an optimal model for neural processing, the hardware approach mostly tries to mimicry the biological evidence at different levels using the best of silicon technology.

Parallelism in Hardware Implementations

Hardware implementations significantly differs from the software ones thus the questions of parallelism arise in different aspects. In the hardware case two main categories can be distinguished: the global and the local parallelism. Global parallelism is similar to the parallelism in software. Local parallelism means low level parallelism in arithmetic units and data paths.

Global parallelism

Network as well as training parallelism theoretically can be used in hardware implementations, although training parallelism is not a convenient method in this case. This is why the parallel implementation of the network and the training algorithm has many difficulties in itself. Usually the technical limits of the realizations do not allow fully parallel implementations. The main efforts are for how can the given network implementation be as parallel as possible. This is why the training parallelism has less importance in this case. The network parallelism can be measured in four levels:

- **WP/WS** Weight Parallel or Weight Serial implementation according to the computation of the partial products (parallel or serial) of the inner product in the neuron.

- **NP/NS** Neuron Parallel or Neuron Serial implementations according to the computation of the outputs of the neuron in a given layer (parallel or successively computed).
- **LP/LS** Layer Parallel of Layer Serial in accordance with the parallel or serial computation of the layers (in the multi layer network).
- **SNS**, SubNetwork Serial. This kind of parallelism can be used only if the network can be "well" partitioned into similar subnetworks corresponding to the point of view of hardware implementation.

All of the possible combinations of the above can be realized from the **WSNSLS** (sequential implementation) to the whole parallel **WPNPLP** implementations. The hardware cost (chip area) and the speed of these solutions are very different so it should be considered before deciding for a given implementation.

Local parallelism

Local parallelism arises only in the case of digital implementations where the arithmetics and the corresponding data paths can be bit serial, digit serial or bit parallel. The experiences in the literature show that the bit serial or digit serial (with low digit size) arithmetics with global parallelism have great importance in digital signal processing because of the necessary chip surface and connections. The global parallelism can compensate the speed loss of these locally serial implementations. This is even more in neural implementations which have inherently large number of communication channels and arithmetic units.

Dedicated Neural Hardwares, Neurochips

Dedicated neural network hardware solutions can be sorted as it is shown in Fig. 5.17. The two main competitive trends are the analog and the digital implementations. All of them has its own advantages and disadvantages as it is detailed here and in section 5.3.5. Hybrid systems try to combine these advantages. While optical technology is in the early inchoative stage yet with few practical experiences.

Analog hardware

Many application examples show that the analog VLSI is an excellent medium to build artificial neural networks inspired from biology, like perception or recognition problems (audio or early vision functions). The analog computation of the weighted sum and the nonlinear function are

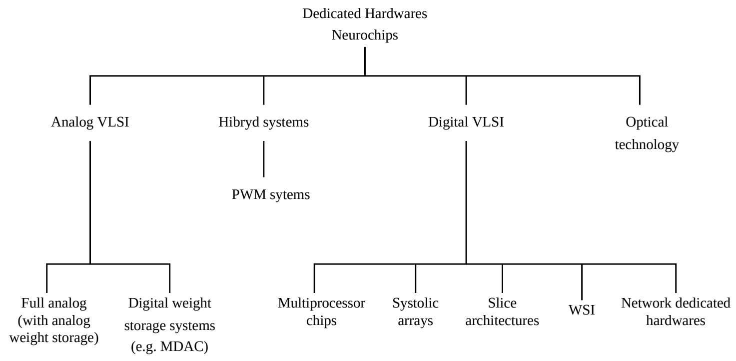

Figure 5.17: Classification of dedicated hardware implementations

revealing signs of the analog realizations. This follows that realizations in which the weight storage is digital but the multiplication is analog can be considered as an analog implementation. Certainly, the analog implementations in the literature are differs. The typical implementations ways are the followings:

- The analog neuron is basically a differential amplifier with synaptic weights implemented by resistors. Although there are experiments with bismuth-oxygen resistors which resistance is "programmable" by electric pulses, but this analog solution is mainly suitable for fixed (pre-trained) network realizations. The main problem is with the realization of the efficient resistor with standard silicon process.

- There are two main methods for the implementation of trainable or variable weight networks. One is when the weight storage element is an analog device like floating gate MOS transistor or capacitor. Floating gate devices employ an extra gate between the silicon and the standard MOS gate. This gate is completely isolated, and allows storing charge which affects channel conductance. Several kinds of floating gate device exist. They can store information in analog way over several years or ten years with 3...5 bits accuracy. This is many order of magnitude longer as it is achievable by simple capacitors. In these implementations the multiplication is performed by analog multipliers. The so called Gilbert cell is a typical analog multiplier which allows four-quadrant multiplication. Gilbert cell is the basis for multipliers used in analog implementations.

- The analog weight storage solutions are not accurate enough (approximately 3...6 bits) for many applications. This is why digital weight storage systems are used. These realizations can be considered as hybrid hardware implementations but because only the storage is digital and the whole computation is analog they are often ranked to analog. Thereafter, the weights are stored in digital registers, these informations must be converted to analog signals and multiplied by the corresponding synaptic data. There are also two wide-spread methods to perform this. Digital to analog converter, for instance R-2R resistor ladder network, can be used for conversion and the Gilbert multiplier for multiplication. On the other hand the so called MDAC (Multiplying D/A Converter) can perform the conversion and the multiplication in one step. MDAC is a special D/A converter which uses an analog input voltage instead of reference voltage. The output of the MDAC is an analog signal which is proportional to the product of the analog and digital inputs.

The realizations mentioned in the first two items can be named as fully analog solutions and the systems in the last item as digital weight storage systems. Usually, the summation is automatically fulfills by the current summation (Kirchoff's low) in such an analog system. The nonlinear effects of the devices inherently allows realization of sigmoid-type functions.

Most of the analog implementations are on CMOS VLSI devices using mainly custom standard cell VLSIs (ASICs Application Specific Integrated Circuits) which shorten development time and decrease costs.

Digital hardware

Digital hardware implementations are motivated by connectionist-type networks rather than the one based on biological evidence. The connection networks are used to perform high level neural functions such as classification. Digital technology offers arbitrarily high precision and great flexibility.

Digital implementation means discrete time and discrete value implementation. Certainly, the hardware that runs neural software can be considered as a digital neural hardware. However, generally the processors or hardware realizations, which specially designed for neuro-implementations and allow the exploition of the advantages of the neural networks can be rated to dedicated hardwares. These processors can be categorized in several ways. Hereafter the classes are distinguished by their architecture and "working philosophy".

Multiprocessor chips

These chips contain several processing units which can work parallel in one chip. These processors can work in different modes (as it it described in section **??**) according to the partial processors do the same operation or not.

Slice architectures

To cut the network into identical elements which are easy to cascade is a well known method in digital VLSI development. They are the so called bit slice architectures. This method has several advantages like easy to design, high flexibility, low design cost, etc. Similar idea can be followed in neural computing.

Wafer-scale LSI

Digital circuits and Wafer-Scale Integration (WSI) provides high-speed, high-accuracy neuro-implementation. WSI means integrated circuit which use the whole wafer. It is very difficult to use WSI for ordinary computers because of the defects on the wafer. Only if one defect is occur on a wafer the ordinary chip won't be work. Nevertheless, neural networks have many redundant neurons that compensate faulty ones, thus these networks are highly fault tolerant and can be constructed using WSI.

Network dedicated chips

There are also some chips developed for a specific application or for a specific type of network. These chips can be efficient because the lack of generality allows a lot of special solutions. Specialized units can be designed instead of universal ones, for instance fixed connection scheme can be used instead of general routing. This means that some elements must not be implemented or it is enough to implement a much simpler one (e.g. complicated instruction decoder and execution controller are unnecessary, a simple controller is enough) in many cases. Moreover, the neural algorithm (the learning) can be modified in a way that it would be easy to implement.

Hybrid hardware

There are a wide variety of hybrid design. In these designs some mixture of the analog, digital, optical, and CCD (Charge Coupled Device) subsystems is used. Typical couplings are the analog+CCD, optical+analog, MDAC+CCD, analog+MDAC, analog+AD/DA, PWM.

It is a very difficult task to classify hybrid solutions because of their high diversity. However, some revealing classes can be distinguished.

Some of the hybrid solutions use CCD (Charge Coupled Devices). This technology is widely used in video processing and thus it is obvious to use them in optical input systems like vision related processing.

On the other hand, there are some analog-optical systems where Liquid Crystal Display (LCD) modules with optical sensor matrix are used for perform the weighted summation in parallel. However these systems have less practical importance.

The most promising mixture of the digital and analog technology is the so called PWM (Pulse With Modulation) circuits. These systems are built from digital elements. The signals are pulse streams with variable width which average (the integrate) carries the information (the value of the signal).

Pulse-coded Neurons

The pulse stream neuro-implementations want to combine the advantages of the analog and digital implementations. This means implementation of pseudo-analog chip with cheap and common digital CMOS processes. Pulse stream techniques are inspired by biological neural systems in which the neurons communicate to each other with "pulses", the so called action potentials. The Pulse Width Modulation (PWM) and Pulse Code Modulation (PCM) techniques have some advantages:

- The effects of fabrication process variations are minimized
- The noise immunity is much higher than that of the analog chips
- The chip surface requirement of a pulse coded neuron is less than a digital one, although its operating speed does not achieve the speed of the analog implementations

Two basic pulse coded approaches have been developed sofar. The first is when pulse width modulated signals control switches through which the output of the preceding neuron is passed to the summation element (leaky integrator). The pulse modulated signal is turned out by a comparator whose inputs are a reference triangle signal and the analog voltage which represents the corresponding weight. This method makes possible four quadrant multiplication. The output of the neuron is the output voltage of the integrator. (Figure 5.18)

The second, more known method performs multiplication by gating signals whit chopping clocks (**??**). This method is very suitable for digital weight storage. The synaptic binary weight is represented by different "chopping clocks" (figure 5.19). This clocks have 1:1, 1;2, 1:4, etc. mark-space ratio representing the 1/2, 1/4, 1/8 bits in the binary weight. The

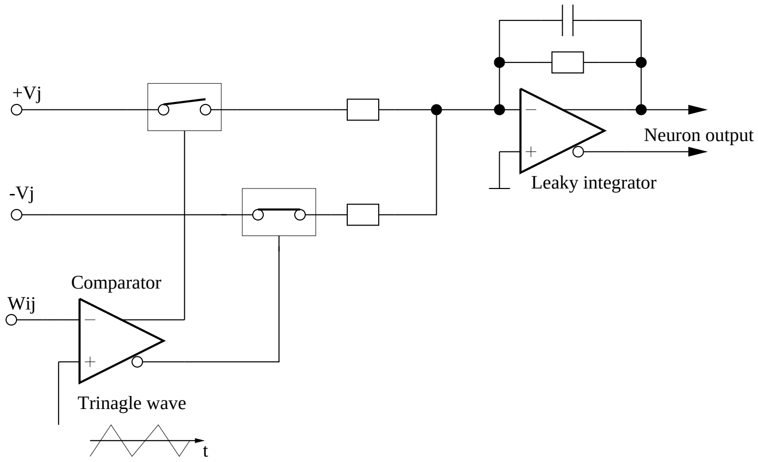

Figure 5.18: Pulse width modulation multiplier

clocks are synchronous to one other but asynchronous to the neuron activities. Each cycle of a chopping clock defines a time interval for gating the pulse modulated output signal of the preceding neuron. If the weight bit is zero, the respective chopping clock becomes zero, too. The sign bit controls whether the resulting signal is transfered to the inhibitory or the excitatory line.

The addition of two pulse coded signals can be also performed by a simple OR-gate if the two pulse streams are uncorrelated. In this case the probability that the pulsed signal is logical one represents its "analog" value. The overlaps between the ones of the signals distort the result. To avoid this (decrease the effect of it), the mark-space ratio of the pulsed signals must be chosen according to the number of signals to be added. If the number of the synapses is high the current pulse addition mentioned above (with integrator) should be used.

Optical hardware

Optical systems have potential for high speed processing, massive storage and massive interconnectivity, but provides less precision, programmability and flexibility than digital implementations. Therefore, it represents only a complementary approach to VLSI.

The information carrier is light beam in the optical systems which makes

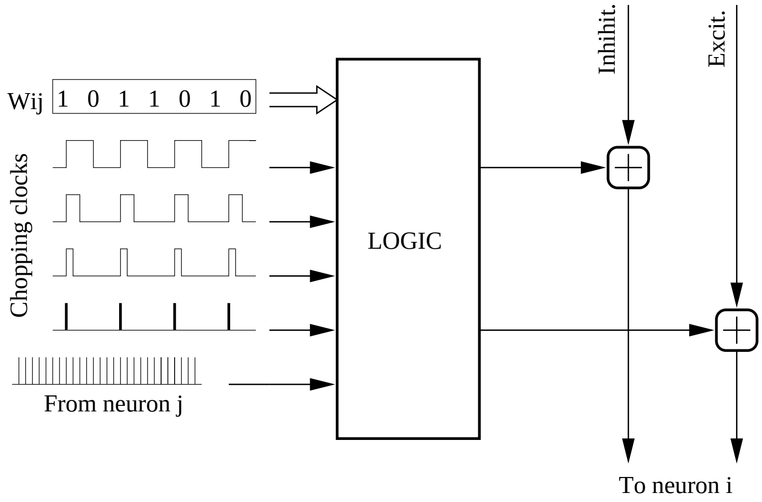

Figure 5.19: Pulse width modulation multiplier by chopping clocks

possible to achieve the theoretically maximum operating speed. Optical devices (lenses) can be designed to solve complex mathematical functions. The holography allows 3D, high density information storage. The main disadvantage is the poor programmability. The synaptic weights implemented with space-variant holographic elements and LCD (Liquid Crystal Display) spatial light modulators in existing experiments. Integration of the optical implementation into systems also has not been solved satisfactorily yet. The existing opto-electric connection devices are slow thus the advantages of the high-speed optical computation vanishes. However, the combined optical-CCD-analog systems have potential for vision related neural networks.

Analog Contra Digital Implementation

The main measures of a given hardware implementation are the required chip surface and the achieved speed or the ratio defined by $speed/(powerconsumption * chiparea)$.

Advantages of analog implementation:

- **High density.** The analog neurons are much simpler than digital ones. They consist less transistors, and require less chip surface. This allows the implementation of large number of neurons on fair chip size (a whole network on a chip) and makes possible whole parallel operation. Fair chip size results good turnout from a waffer (low spoilage ratio).

- **Very high operating speed.** While the amplitude of the electricity directly represents variables in continuous time, the operating speed can be extremely high.

Disadvantages of analog implementation:

- **Low precision.** The achievable low computation precision has three main causes. The analog weight storages are based on charge storage. This charge is being continuously discharged by leakage currents. The stored charge can not be arbitrary because of the implementable capacitors (a fair size is $30\mu m * 30\mu$m, about 1pF on which 1pA current cause 1V/s variation). The second cause is by the multipliers. At low current level the matched transistors in the multipliers operate in weak inversion which cause linearity problems. Conversely, higher currents mean higher chip surface, power consumption and heat the chip which issues in thermal drift. The third is the reproduction uncertainty. The element of the analog chip (resistors, matched transistors) can be manufactured with only quite high parameter variance. The precision is usually about 5-6 bits but limited to no more than 8 bits.

- **Difficult implementation of variable weights.** There is no satisfactory method to implement variable, high precision synaptic weights. There are some experiments with switched capacitor networks and floating gate devices (Intel ETANN chip, CHMOS-III EEPROM technology). These devices are not stable and precise enough in long time. Although, for instance the ETANN chip can be reprogrammed but programming is not easy, takes a long time, requires external hardware and difficult programming algorithm. The fixed or mask programmable resistor networks or current generators are lack of programmability and thus the possibility of on line learning.

- **Lack of adaptivity.** This is why the weights are hard to vary. On the other hand the continuous time analog networks work asynchronously which make the application of the convenient learning algorithms hard.

- **Hard to cascade chips.** Although quite a large network can be implemented on a single chip, the cascadability of the chips is difficult.

- **Susceptibility.** The analog circuits are more sensitive to noise, crosstalk, power supply variation, temperature effects and fabrication parameter variation.
- **Expensive fabrication.** The fabrication parameters must be strictly held. The post fabrication test of the analog chips are also difficult. Although self-test circuit could help this but they requires large chip surface and high efforts from the designer.
- **Tricky to design.** An analog circuit design requires large experience, extensive simulations on the physical behavior level which is time consuming.

Advantages of digital implementation:

- **Arbitrary precision and dynamic range.** The precision and the dynamic range of the computation theoretically can be arbitrarily chosen. In practice they are limited by computation time and/or chip size (complexity of the arithmetics). However, both parameter can be much better than in case of analog or other implementations. The connectionist approach exploits these properties.
- **Flexibility.** Digital designs give much more flexibility than analog ones. Well developed CAD technologies help modular system development and quick and relatively cheap prototype construction. There are many designs methods which can be overtaken from DSP and multiprocessor system design.
- **Stability.** The stability issues of a digital design are much more easy to handle because of its synchronous operation and insensitivity of parameter drift.
- **Insensitivity.** The digital realizations are less sensitive to electronic noise, crosstalk, power supply variation and thermal drift.
- **Relative small design experience is necessary** comparing to the case of the analog counterpart.

Disadvantages of digital implementation:

- **Large required chip surface.** Digital arithmetic functions require usually much more chip area than analog ones. The number of connectivity is also higher than in analog case (especially with bit parallel arithmetics) which is the main bottleneck in many cases.

- **Lower operation speed.** The lower operation speed of the digital arithmetics sometimes makes the digital realizations unacceptable (e.g. some high energy physics applications).

General remarks

In addition to the problems mentioned above generally two main problems occur. Both are because of the high connectivity. The many inputs, outputs and the cascadability of the chips require a lot of pins which cause package problems. On the other hand, the two-dimensional chip allows less connections. Recently some three dimensional implementations were developed to solve this problem (e.g. as it is described in section 5.3.7). This is why the bit serial communication has advantage in digital implementations.

5.3.6 Benchmarking, Performance Criteria

The performance analysis is one of the main point deciding about neural network implementations. The usual standard measures like FLOPS (Floating point Operations Per Second), IPS (Instruction Per Second), OPS (Operation Per Second) or clock frequency are mostly not revealing well the neuro-machines, but sometimes they are used. A good measurement method must satisfy some fundamental criteria:

- A performance criteria must be *sufficiently general*. It should be applicable to the whole range of possible implementations.
- It must be of sufficiently *high conceptual level*. The user is not interested in details but the performance of the implementation in terms of speed, learning ability, etc.
- It must be *problem independent*. General purpose implementations can be measured only by general way.
- It must have sufficient *distinctive power*. If a criteria is so weak as to yield the same number for different neuro-implementations, it can not be used to qualify them.
- It must be *obtainable* in order to receive measures within an acceptable time period.

Speed criteria

There are some commonly accepted units to perform the speed of neuro-implementations:

- **CPS, Connections Per Second.** It measures the rate of Multiply-and-ACCumulate (MACC) operations during recall phase.
- **CUPS, Connection Update Per Second.** It measures how fast the network learn. It is the number of weight changes per second during learning phase.
- **CPSPW, CPS Per Weight.** Normalizing the CPS value by the number of the weights gives a better performance measure of the chip.
- **CUPSPW, CUPS Per Weight.** Similarly, it is the CUPS normalized by the number of weights.
- **CPPS, Connection Primitives Per Second.** This value includes the computation precision into the processing performance. $CPPS = b_{in} * b_w * CPS$, where b_in and b_w are the bit numbers representing the input and weight values.
- **CPPSPW, CPPS Per Weight.** It is the normalized CPPS by the number of weights.
- **ECPPS, Effective Connection Primitives Per Second.** Using the effective bit accuracy (it is defined bellow) for the connections CPPS can be modified as $ECPPS = b^2 * CPS$.

Similarly **CPUPS**, **CPUPSPW**, etc. can be defined but these are not so rife in the practice. There are no widely accepted benchmarks for neuro-implementations, although the learning time for NetTalk is often given (NetTalk is a back-propagation network that translates a text into phonemes).

Accuracy criteria

Expressing the accuracy in terms of number of bits for inputs b_{in} and number of bits for weights b_w neglects all the other error sources, such as offset error, nonlinearity of multipliers, etc. This is why the **effective number of connection bits** b are defined the following way. An error can be defined as the difference between the current real implementation (with input bit numbers, b_{in} and weight bit numbers, b_w) and its idealized version using infinite precision (Fig. 5.20). A *Signal to Noise Ratio, SNR* can be defined as the signal variance per this error variance. The effective number of connection bits, b is a value that yields this SNR.

Both the ideal and real networks are initialized with the same randomly but uniformly chosen weight values and inputs. In this measurement not

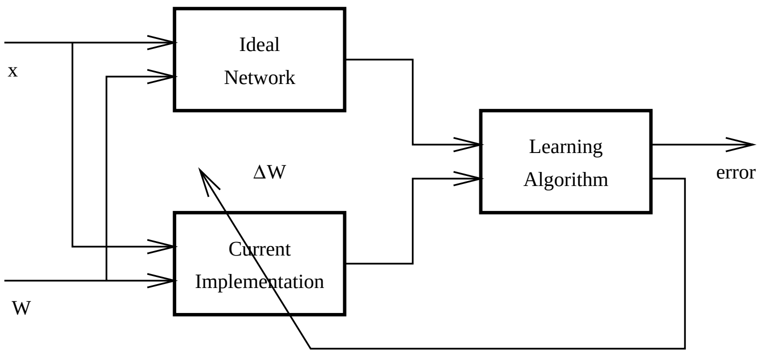

Figure 5.20: Measuring SNR after learning

only the simple amount of the error is important, but the ability of the learning algorithm which compensates for this error. If an error can be neutralized simple by altering the weights, this error will be of little effect at the output after learning. This is why it is unfair to measure it before the chip has had the possibility to deal with it. Therefore if no learning algorithm is implemented in the real neuro-implementation, a standard learning algorithm -for example back-propagation- must be used to train (off-line training) the network. Certainly, this off-line training must be performed with the same conditions (e.g. word-length of the arithmetics) as the target implementation. This allows that the training compensates the errors.

Criteria for size and reconfigurability

If the desired neural network has more neurons, weights and synapses that are connected to a single neuron than the **Number of neurons per chip** N_n, the **Number of weights per chip** N_w and the maximum **Connection Per Neuron CPN**, there is no guarantee that the network can be mapped to the chip. **Reconfigurability** says something about the probability that a given topology can be mapped onto that chip. Certainly, any kind of network in any size can be implemented in software but at worst it will be prohibitively slow.

The **Reconfigurability Number** RN can be defined as the number of different fully connected feedforward neural networks that can be mapped on the chip regardless of the number of layers involved. Topologies that are mappable on a given chip grows exponentially in the number of weights N_w

on that chip. The restriction feedforward and fully connected are imposed to keep the RN intractable.

The qualification of the implementations by measures requires circumspection because of the different precisions, sizes and learning properties of the different learning algorithms. For example such a Boltzmann learning may only need a few passes trough the training set as compared to perhaps thousands of epoch for back-propagation.

5.3.7 Typical Implementation Examples

Analog Neurochips

NeuroClassifier

A typical analog fixed network implementation example is the NeuroClassifier chip developed by MESA Research Institute. This chip was designed for HEP (High Energy Physics) pattern recognition applications, in which the very high operating speed is a basic requirement. This chip processes two layers in 20ns. It has 70 analog inputs, 6 hidden neurons and one output. The six neurons of the hidden layer computes the weighted linear combination of the 70 inputs and have sigmoid activation function. The output neurons are simple linear combiners. Its block diagram is in Fig. 5.21.

The 5 bits (4 bits + sign) weights of the network are stored in static RAM. The multiplication is performed by MDACs. The 426 analog synapses (multipliers) work in fully parallel. The weight memory of the chip can be digitally programmed, the programming time is 5ms. Its computation speed is 20G CPS, the total processing delay is 20 ns, the equivalent input bandwidth is 4G Byte/s. The size of on chip SRAM is 3750 bits. The chip is realized on 10mm * 9mm chip surface with 1.5μm DLM CMOS, ES2 technology (40,000 transistors). It is mounted into a 144 pins PGA package, the power dissipation is 0.6W . It was shown, that NeuroClassifier is also capable of performing real-time image processing tasks such as convolution with up to 64 element kernel, up to 1000 frames per second on 512x512 pixel images.

Intel ETANN chip

The Intel Electronically Trainable Analog Neural Network (ETANN) chips was one of the first commercially available neural network chip. This chip contains 64 neurons, 10240 synapses. Figure 5.23 shows its block diagram. ETANN works on analog signal, but there are also some digital control signals which helps the configuration, training and cascadability.

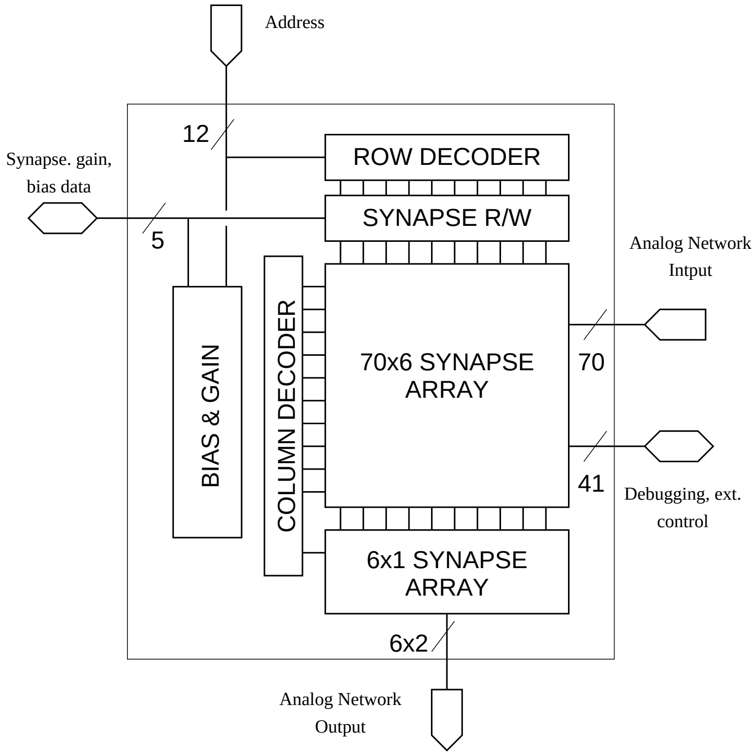

Figure 5.21: NeuroClassifier

Inputs are organized in two 80x64 synapse arrays (64 inputs and 16 fixed biases per neuron), the external and the feedback arrays. This makes possible many types of network implementations, for instance wide variety of the feedforward networks, Hopfield network, etc. The chip basically supports a single-layer network configuration. However, it can be configured for 2-layer processing. The ETANN chip also can be cascaded in two basic multi-chip configurations: direct pin interconnection and bus interconnection. The direct pin to pin connection is the more straightforward way since any given output is connected directly to an input of another chip. No multiplexing is required in this case. The bus oriented interconnection scheme provides more flexibility but has some drawbacks like: data can be moved only in complete 64-neuron blocks, multiplexing is slower and external hardware is

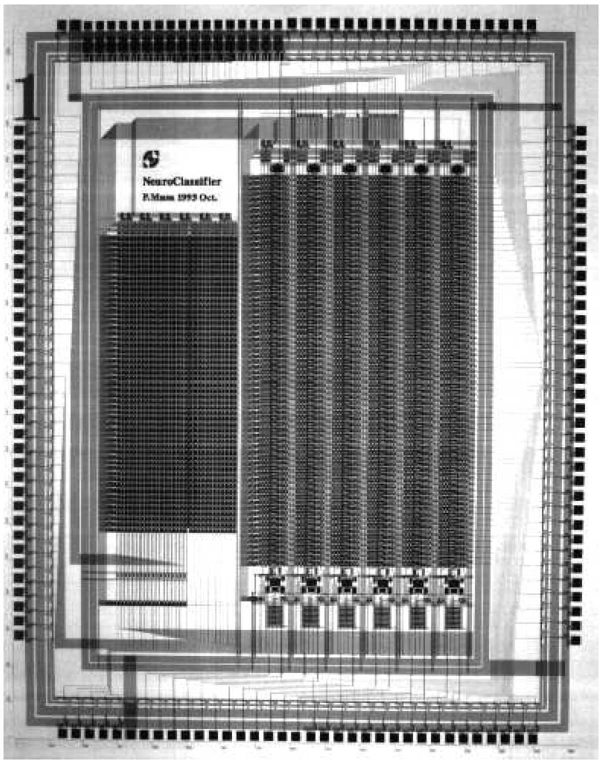

Figure 5.22: Photo of the NeuroClassifier chip

required.

The inner products of the input vector and the stored weight vectors are computed by four quadrant Gilbert multipliers, The outputs of the 64 neurons passed through identical sigmoid functions to the outputs of the chip. The gain of the sigmoid functions can be externally controlled. The weight vectors are stored in floating gate EEPROM cells (using Intel's advanced CHMOS III EEPROM technology). Two EEPROM cells correspond to a weight value as it is in Figure 5.24. The difference in the two floating gate thresholds represents the weight value. It can be programmed by multiple pulses iteratively while the weight does not achieve the required value. Width and amplitude of the pulse is between $10\mu s$ to 1ms and 12V to 20V according to the necessary charge movement on the floating gates. The

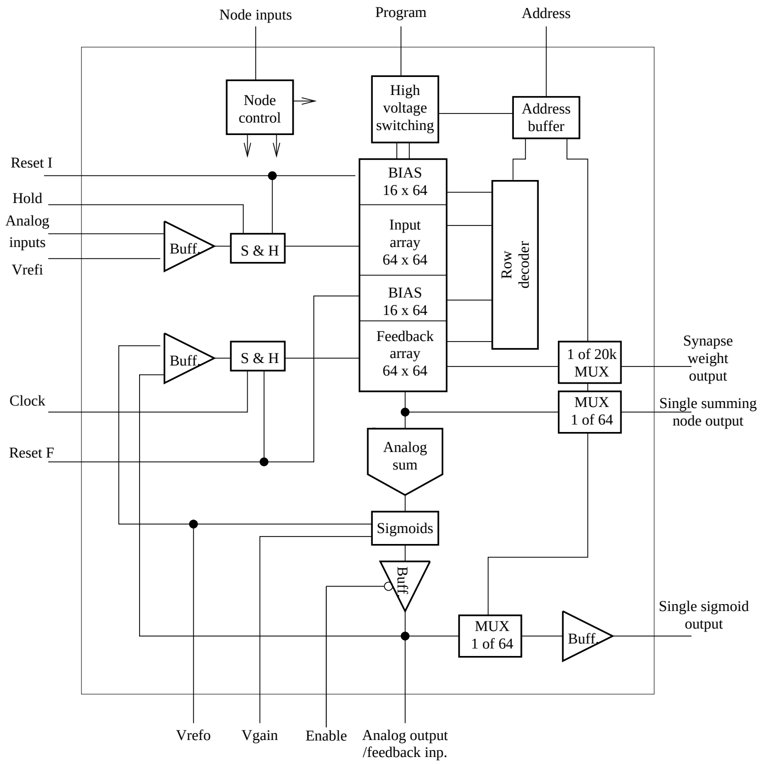

Figure 5.23: Block diagram of the ETANN chip

effective weight representation accuracy is 6-8 bits with the so called Bake re-training, which means that a trained chip is exposed to high temperature (250 $°C$) to accelerate charge movements in the cells and afterwards is re-trained again to restore original charges. The minimum data retention lifetime is 10 years.

The chip contains no on-chip learning algorithm. Intel offers the Intel Neural Network Training System (iNNTS) which provides *chip in the loop* training. This means that the chip can be trained in a special configuration. The feedforward calculation is performed by the given ETANN chip, the training algorithm runs on a host computer and a special adaptor provides the link between the ETANN chip and the host computer. This method allows more precise training than the training on simulators because of the variation of the analog circuitry. However, there are simulators which include ETANN-specific model, for instance California Scientific Software's BrainMaker program.

ETANN chip has 2G CPS performance in recall phase while the learning is quite slow (pulse width is up to 1ms).

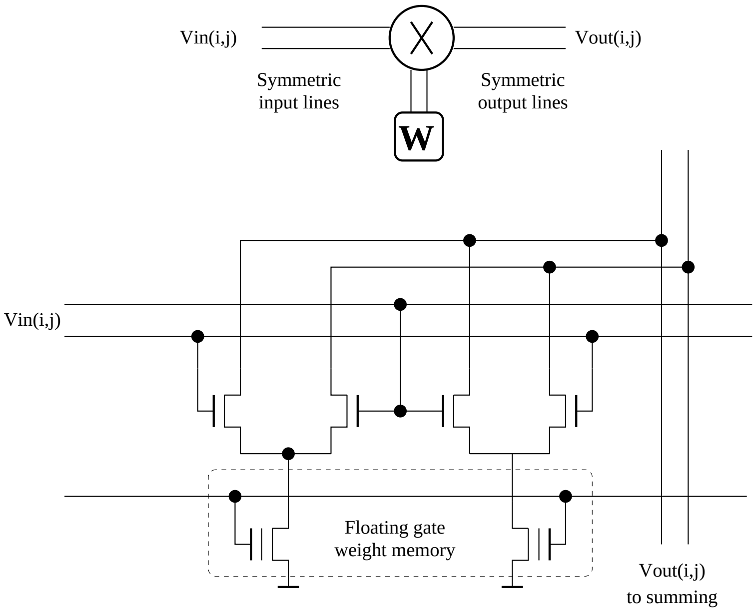

Figure 5.24: ETANN synapse implementation

CNN chip

There are several attempts to implement Cellular Neural Networks using analog technology. The most promising chip is proposed by Centro Nacional de Microelectrónica, Sevilla.

The CNN is composed of identical cells which are placed on a mesh. Usually 3x3 templates are used. The circuit diagram of a cell and the 2-D mesh with nearest neighbor interconnections are shown in figure 5.25.

/it i_{in} and i_{out} are implemented by simple linear voltage controlled current sources. The controlling voltage are the output voltage, and input voltage, of the neighboring cells, respectively. The **A** and **B** templates controls

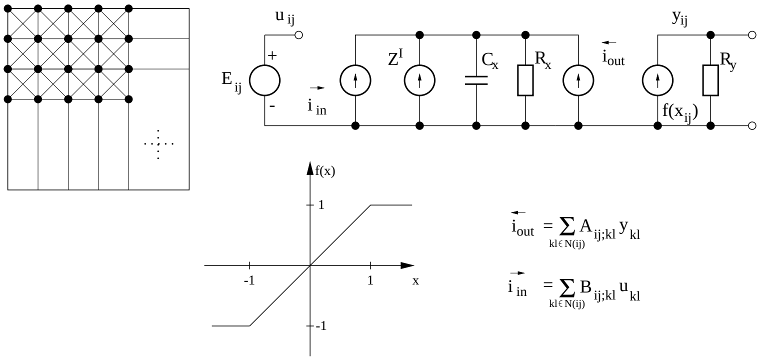

Figure 5.25: 2-D CNN lattice and the circuit diagram of a simple CNN cell

the gain of these controlled sources which are implemented as transconductance amplifier. Z is a bias term. The NxN CNN array has

The given chip has 20x22 cell array with optical black and white (binary) inputs and binary outputs. The chip contains four image and eigth digital template memories. ***ami most jon nem tudom hogy erre a chipre is igaz-e?*** The described CNN nucleous are extended by adding a Local Analog Memory (LAM), a Local Logic Memory (LLM), a Local Logic Unit (LLU), a Local Analog Output Unit (LAOU), and a Local Communication and Control Unit (LCCU). This detailed model is in figure 5.26.

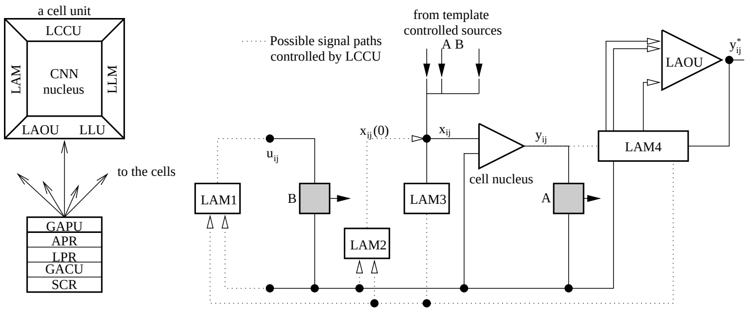

Figure 5.26: Architecture of the CNN universal machine

Elementary logic operations are implemented in LLU which operates data stored in LLM. LAOU can calculate some nonlinear function of the output with a few values stored in LAM. The LCCU provides communication to the central part of the architecture (GAPU). The Global Analogic Program Unit (GAPU) selects the program of the chip from the Analog Program Register (APR), and sends to the cells. It also selects tha logic operations executed by LLUs from the Logic Program Register (LPR). The configuration of the cells is determined by the Switch Configuration Register (SCR). The Global Analog Control Unit (GACU) controls the operation of the whole CNN chip including the I/O operations.

The CNN development system consists of a C-like high level analogic programming language, a Texas DSP based PC board to help programming CNN chips (CNN Operating System), and a CNN platform which interfaces chips to the prototyping board.

3-D IC stack analog neuroprocessor

A three dimensional analog IC cube was proposed by the Center for Space Microelectronincs Technology Jet Propulsion Laboratory, California Institute of Technology for fast pattern recognition. This IC cube consists of 64 thinned VLSI chips as it is shown in figure /refF3dcube. An image sensor array is mated to the IC stack so that each row in the image sensor array is directly connected to the input of one IC. The size of the cube is 1 by 1 by 1 inch. Meta bus lines which runs across side planes provide communication between the 64 chips.

Neural network input could be controlled by a sequencer unit, that controls signal flow along the 64 common bus lines. The sequencer has the following functions:

- selects a desired window (e.g. 8x8) from the image sensor
- converts the window data into a vector
- shuffles the individual vector elements to obtain them in the right sequence
- provides this vector to the input of one or more ICs in the stack

The ICs in the cube are identical, but can be programmed with different weights and/or different input windows. This allows high flexibility.

The feed-forward network has 7-bit weights realized by MDACs, variable gain sigmoid activation functions and control and addressing circuitry which enable a variety of neural architectures and algorithms such as multilayer perceptron, cascade backpropagation, and inner-product based network with winner-take-all algorithm. Fig. 5.27 shows the block diagram of the chip. The so called Cascade BackPropagation (CBP) algorithm is used to train the network because of its good training properties for networks with limited synaptic weight precision (7 bit). The proposed application is neural image processing in missile seeker which requires extremly high speed. The chips are opereted at 4MHz which provide 1000 frame per second data rate. This means 10T CPS that is 250 ns signal processing speed. The simultaneous performances are: extrem operation temperature (90 $^{\circ}K$), and low power consumption (2W).

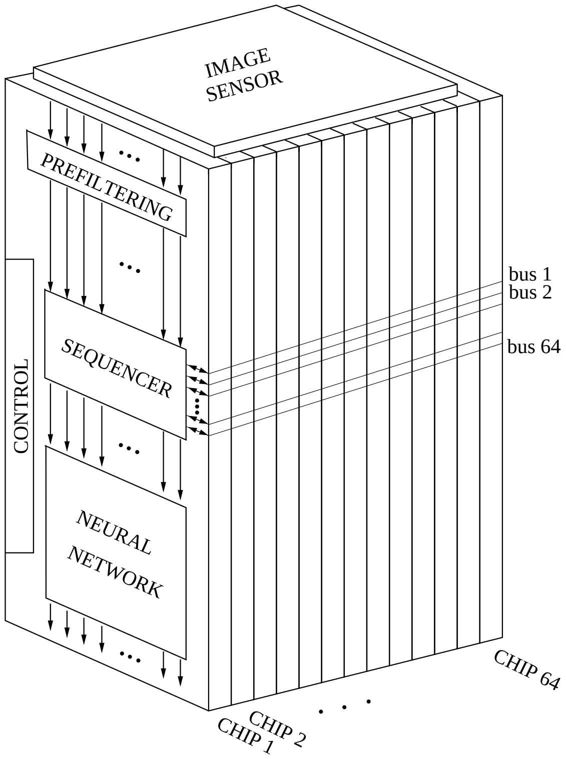

Figure 5.27: 3-D IC stack analog neuroprocessor

This implementation example enlightens one of the main problem of the implementation of artifical neural networks compared to the biological "implementations", namely the three dimensional placement. Biological model has not only more neurons but usually much more connections which connections are allowed by the 3-D placement. This makes possible so effective "implementation" in a compact way.

Digital Neurochips

CNAPS (X1) SIMD multiprocessor chip

Adaptive Solutions's X1 Digital Parallel Processor chip has 64 subprocessors operating in SIMD mode. There is a whole system called Connected Network of Adaptive ProcessorS (CNAPS) built from this chip (see in 5.3.7 section). The chip was developed for neural and image processing applications. The chip supports efficient on-chip learning.

The subprocessors (processing nodes, PNs) are configured in a linear array. Figure 5.28 shows the architecture of the SIMD array. CNAPS array has two boardcast buses, the INbus which provides input data for each PNs, and the CMDbus which is the processor node command bus. There is also a local Inter-PN bus which connects the neighbouring PNs together. The PNs can access the OUTbus (one PN at a time). This connection scheme is very suitable for fully interconnected networks. The control of the PNs is provided by another sequencer chip.

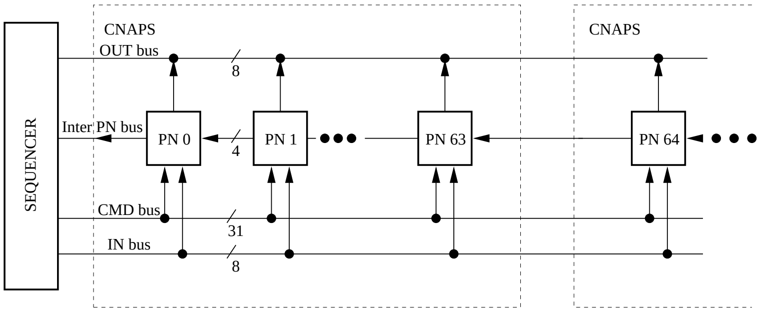

Figure 5.28: X1 chip array

The structure of the PNs is depicted in Figure 5.29. Each PN has two 16 bit internal buses. The input and output units interfaces these internal buses to the Inbus and OUTbus. The input unit is responsible for decoding processor commands. Conditional instructions can be executed using the Inter-PN bus. A 9x16 bit two's complement multiplier, a 32 bit adder and a shifter-logic unit corresponds to each PNs. To store information the PN has 32 16-bit registers and 4k Bytes local memory. The weight can be 1-bit, 8-bit and 16-bit wide.

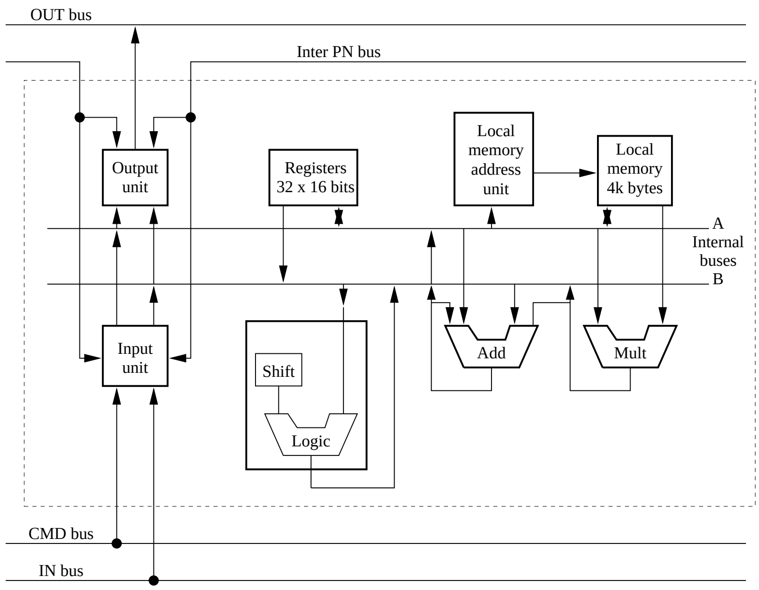

Figure 5.29: X1 chip

The chip is fabricated by Inova Microelectronics Corp. on N64000 name. To implement large PNs, like these, quite a large silicon area is needed. This large die size results low yield. However, this architecture allows redundant chip fabrication which means that more PNs implemented on a die and the non functioning PNs are neutralized afterwards. N64000's die size is 1 square inch and 80 PNs are fabricated on it. The final yield is 90% which means 90% of the dies has more than 64 functioning PNs.

A single 64 PN 128k synapse chip has 870M CPS and 220M CUPS performance.

Neural RISC MIMD multiprocessor chip

A neural RISC processor based MIMD architecture was developed at University College London. The Neural RISC processor chip contains several Processing Units (PUs). Each PU has a 16 bit RISC processor core (arithmetic unit and processor controller), a communication unit and one kword (16-bit wide) local memory (see Figure 5.30). The PUs have only 16 different instructions. All instruction has fixed format: 4 bit opcode plus 12 bit immediate value or address. To reduce the pin number of the processor chip the PUs are connected to two 16-bit buses, an input and an output one. Each PU has its own Name/Layer registers and communication units which provides bidirectional point-to-point links between processors and supports logical broadcasting and ruoting data packets. The 16-bit links are controlled by handshaking lines (three per bus). A message exchanged by processors consists three words:

1. Neuron or layer name to define the destination processor or processor group.

2. The "name" of the input.

3. The value of the input.

The prototype chip was packaged into a 84-pin pakage. It consist two PUs and runs at 5MHz clock.

MA16 systolic array chip

MA16 is a systolic array neural signal processor chip developed by Siemens. The neural signal processor is dedicated to perform matrix-matrix multiplication addition or substraction. These operations are done by elementary Processor Modules (PMs) which consist of a four element systolic chain. Four multipliers and four adders compose an elementary systolic chain (Figure 5.31). This multiplier-adder chain can perform the following operations:

- $s_{ik} = \alpha_i \sum_j x_{ij} y_{jk}$
- $s_{ij} = \pm x_{ij} \pm y_{ij}$
- $s_i = \sum_j (x_{ij} - y_{ij})^2$
- $s_i = \sum_j \mid x_{ij} - y_{ij} \mid$
- $s_i = \sum_j \mid x_{ij} \mid$
- $s_i = \sum_j \mid y_{ij} \mid$
- $s_i = \sum_j x_{ij}^2$
- $s_i = \sum_j y_{ij}^2$

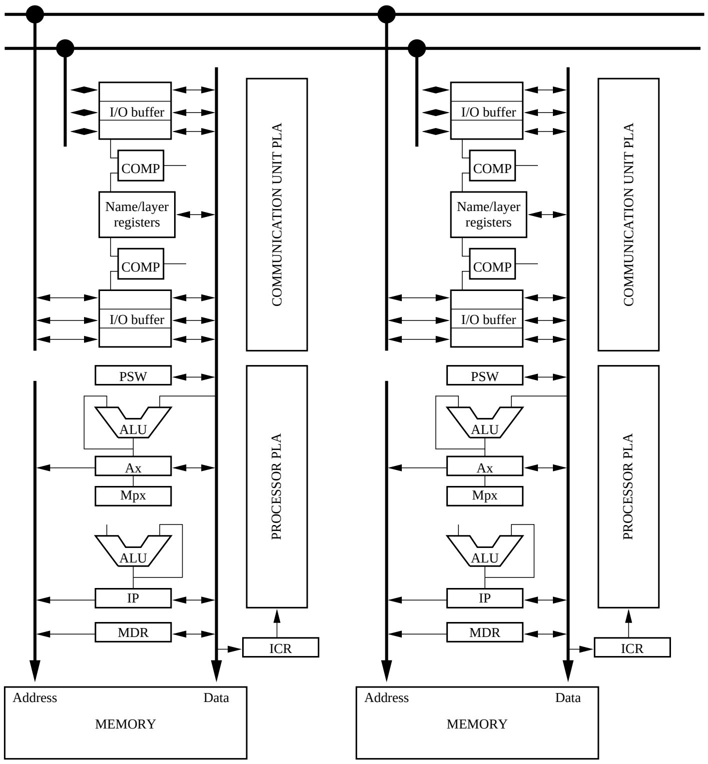

Figure 5.30: The Neural RISC architecture

Each PM has a second unit which realizes some additional neural operations and a control unit. A MA16 chip contains four PMs as it is shown on Figure 5.32. The weights are stored and loaded from off-chip memory. No special support for learning algorithms was implemented on the chip. The outcome of this is a computing performance of 800M CPS with 16x16 bit multiplication at 50MHz clock rate.

SYNAPSE-1 is a general neurocomputer system built from MA16s (see later in 5.3.7).

Lneuro transputer chips

Lneuro 1.0 is developed by Philips. The chip was designed to ease to interface to Transputer host processors. Actually, Lneuro 1.0 is a parallel

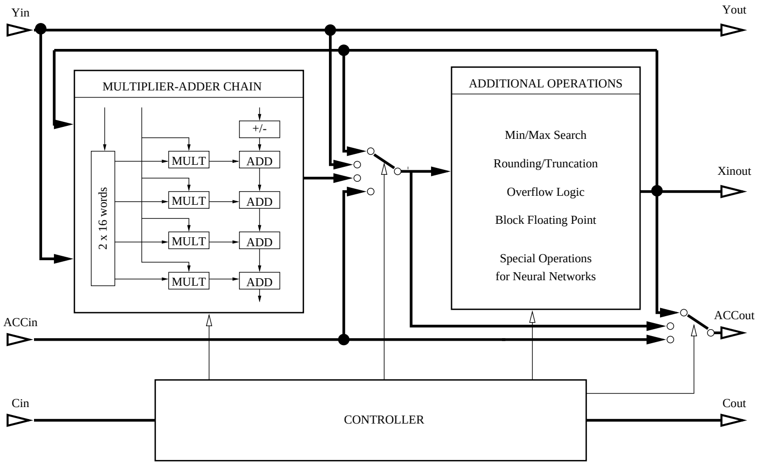

Figure 5.31: Block diagarm of a processor module

coprocessor for a 25MHz INMOS Transpute. Figure 5.33 shows the block diagram of the chip. The data flow in recall phase goes from the synaptic RAM, learning registers, multipliers, adder tree, and accumulation. In the learning phase the synaptic weights are loaded from the RAM, updated inside the learning registers according to the learning function latches, and eventually stored back in the RAM. The main bus provides all I/O and controls the operations. The synaptic RAM is 32-bit wide, thus the unit is organized in four modules. A single Lneuro chip consists of four such unit. This means that there are 16 PEs in every chip. The multiplications in the modules are bit-serialized. This allows some different configurations: 16 1-bit neurons, 4 4-bit, 2 8-bit, or a 16-bit neuron. This results in for instance 256 1-bit neurons per chip. The synaptic memory is 1k Byte. A SIMD type control is added.

The performance of a single chip is 100M CPS and 160M CUPS for 1-bit mode and 26M CPS and 32M CUPS for 8-bit mode. There is an improved version of the chip called Lneuro 2.3 which provides 720M CPS for 8-bit mode.

MD1220 Neural Bit Slice chip

Micro Devices proposed a Neural Bit Slice (NBS) processor (MD1220) which was the first commercial neurochip. An MD1220 chip consists 8

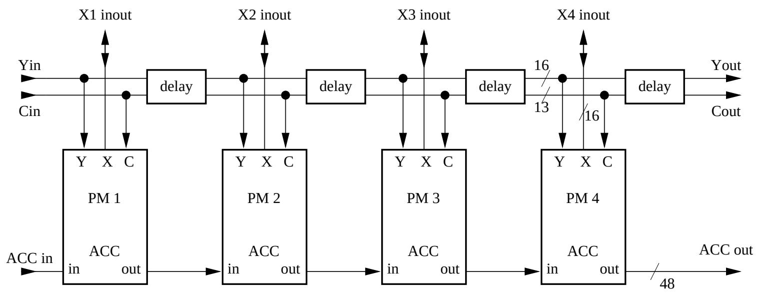

Figure 5.32: A systolic array of four PMs forms an MA16

identical processing slices (Fig. 5.34)and a control logic for the communication and for controlling the external weight memory. Each neural bit slice consists a bit-serial multiplier stage (AND gate) connected to a 16 bits accummulator. The outputs of the accumulators are connected to a data bus (CPU data bus) or a built-in treshold function (hard limiter). The input of a slice can also be connected to a specific data bus (Synapse data bus) via multiplexers. The chip can be configured to several mode by the multiplexers from 1-bit by 16-bit bit-serial multiplication to arbitrary bit size by 16-bit multiplication. The built in input multiplexors are 8 to 1, thus a 8-synaptic weight weight serial, neuron parallel (WSNP) locally-bit-serial processing can be done without external hardware. The number of the synaptic weights per neuron can be extended using external multiplexers, but the 16-bit accummulator can be overflow with increasing probability which causes precision loss. The NBS chip can also be used in a mode where different inputs are processed by the neurons. Unfortunately, no learning is supported by the chip. The learning can be done only by the control of an external CPU. However, the NBS chip can work without external control in the retrieving phase.

The chip is packaged in a 68 pin PLCC and runs at 20 MHz clock frequency which yields 55 MIPS.

Hitachi WSI chip

Hitachi proposed a 144-neuron digital WSI chip. The chip has dual network architecture which makes possible the on-chip high-speed backpropagation learning. This means that a chip contains in fact 288 neurons which are partitioned into two subnetworks. A forward network performs the retrival phase and a backward one is used for learning. The neurons use

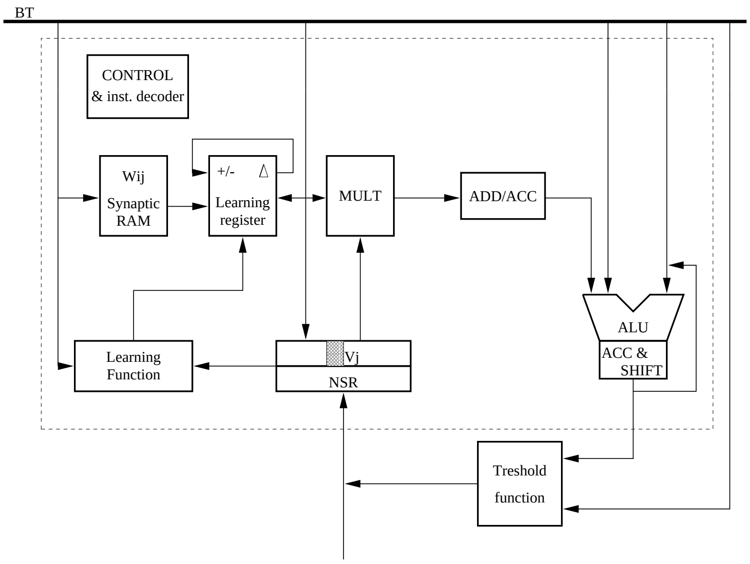

Figure 5.33: Block diagram of Lneuro 1.0

the so-called time-shared bus architecture. In this architecture, each neuron has only one synapse (WSNP operation). The sender neuron use broadcast messages. The backward network is necessary for fast learning in this architecture because of the output error can not be directly broadcasted, so the "inverse" network must be implemented, too. (See Fig. 5.4.)

The simplified block diagram of one neuron is on Fig. 5.35. The whole wafer is composed by 48 neuron chips each of which contains six neurons, neuron control chips, and bus and power lines. The bus and power lines are not fault tolerant. This is why the bus circuits are designed to have triple redundancy and the power lines designed to have large width and spaces to avoid defect. The occurrent defects are detected and eliminated by using microlaser cutter in a final control phase of the fabrication. The fault tolerance of the neural networks was demonstrated with the exclusive-or example. Originally only five neurons are needed to contsruct this network but it is quite difficult (many epochs are necessary) to learn the problem. A three-layer network (2 input, 30 hidden and 1 output neurons) was used with 2 defective neurons (with stuck-at errors) in the hidden layer. The self-organization capability of the network eliminates the defective neurons

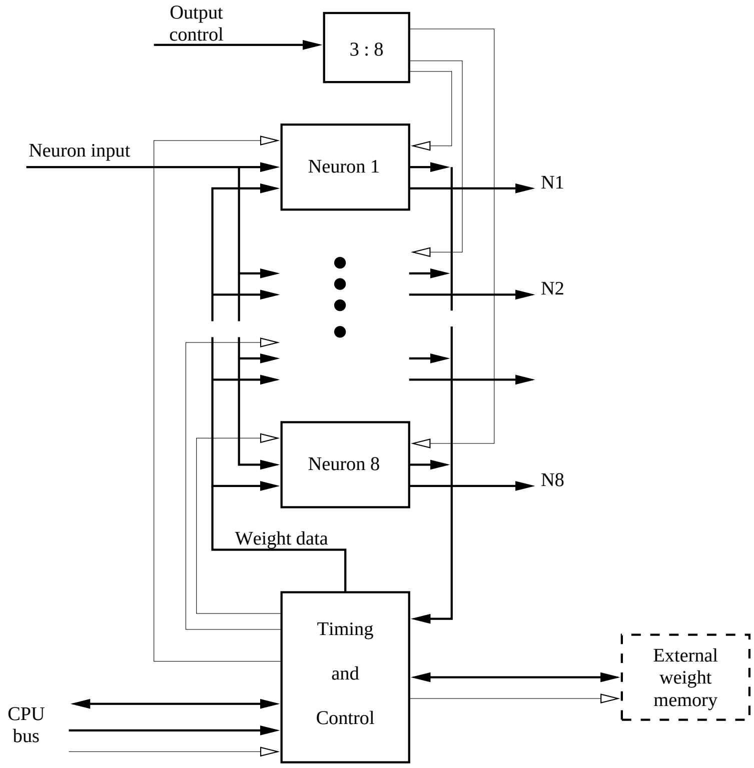

Figure 5.34: MD1220 architecture

(by setting the corresponding weight values to almost zero thus isolating the faulty neurons) by the end of the learning.

An 1000 neurons experimental system was built using 8 wafers with more than 250 live neurons/wafer.

ZISC RBF network dedicated chip

ZISC (Zero Instruction Set Computer) is a digital chip from IBM and Nestor Inc. It has 64 8-bit inputs and 36 radial basis function neurons. It was designed for recognition (speech and visual), image and signal processing application using RBF networks. Multiple chips can be easily cascaded together to create networks of arbitrary size. Input vector V is compared to stored a prototype vector P for each neuron. Figure 5.36 shows

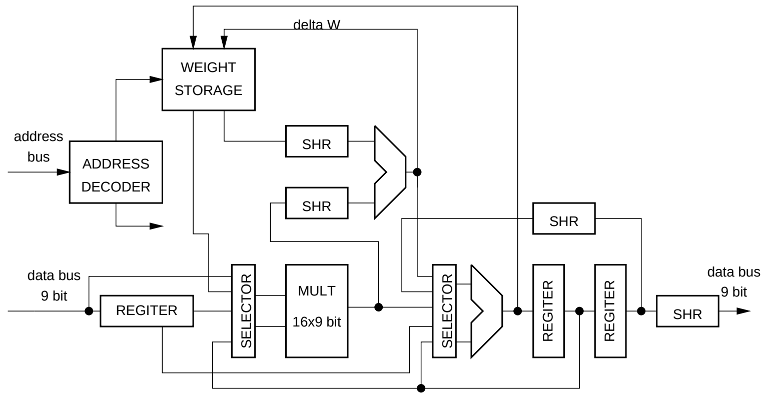

Figure 5.35: Block diagram of a neuron

the block diagram of the chip and the structure of a neuron is shown in figure 5.37 (from **??**). The 14-bit neuron output (for 16K possible categories) is a distance value calculated according to 2 selectable norms: (1) $dist = \sum_{i=1}^{6} 4abs(Vi - Pi)$; (2) $dist = \max_i((Vi - Pi))$. Input is in 8-bit serial format. It takes 3.5μs to load the 64 elements and another 0.5μs for the classification signal to appear. On-chip learning, i.e. storage of training vectors as prototypes of a vector takes about another 2μs beyond the 4μs for loading and evaluation.

Other chips

Some other examples are in the Table 5.2.

Neurocomputers built from neurochips

CNAPS

Adaptive Solution has a general purpose neurocomputer system called CNAPS (Connected Network of Adaptive ProcessorS) built from their N64000 chips (described in 5.3.7 section). This neurocomputer was developed for neural and image processing, pattern recognition and other signal processing applications and provides two order of magnitude faster operation than CRAY-2 supercomputer (see in 5.3.7 section).The CNAPS system consists of several N64000 processors and a sequencer unit (fig. 5.38). The sequencer broadcasts input data in parallel to each processing node (PN) on the input

Name	**Type**	**Performance**	**Notes**
Analog chips			
Synaptics Silicon Retina	Neuromorphic, analog	na	
Digital chips			
Neuralogix NLX-420	FdFwd, ML	300CPS	1-16bit, off-chip synapses
HNC 100-NAP	SIMD	250MCPS, 64MCUPS	100PE, 512k off-chip synapses
Inova N64000	SIMD	870MCPS, 220MCUPS	64PE, 128k synapses, 1-16bit
Nestor/Intel NI1000	RBF	40k pat./s, 27G CPS	1PE, 5bit
Accurate Aut. Corp. NNP	MIMD	140MCP	16bit
INCO SNAD/1	FdFwd, RBF	200MCPS	16bit mult., 40 bit add.
MT19003	FdFwd, ML	32MCPS	1x16bit, 1PE
TOTEM	SIMD	1G CPS	8x16bit, 32 MACC processors
Hybrid chips			
AT&T ANNA	FdFwd, ML	2.1GCPS	16-256 neurons, 3x6bit, 4k synapses
Bellcore CLNN-32	Boltzmann	100MCPS, 100MCUPS	6x5bit, 992 synapses
Ricoh RN-200	FdFwd, ML, BP	3GCPS	16 neurons, 256 synapses

Table 5.2: Comparisation table of some other neurochips

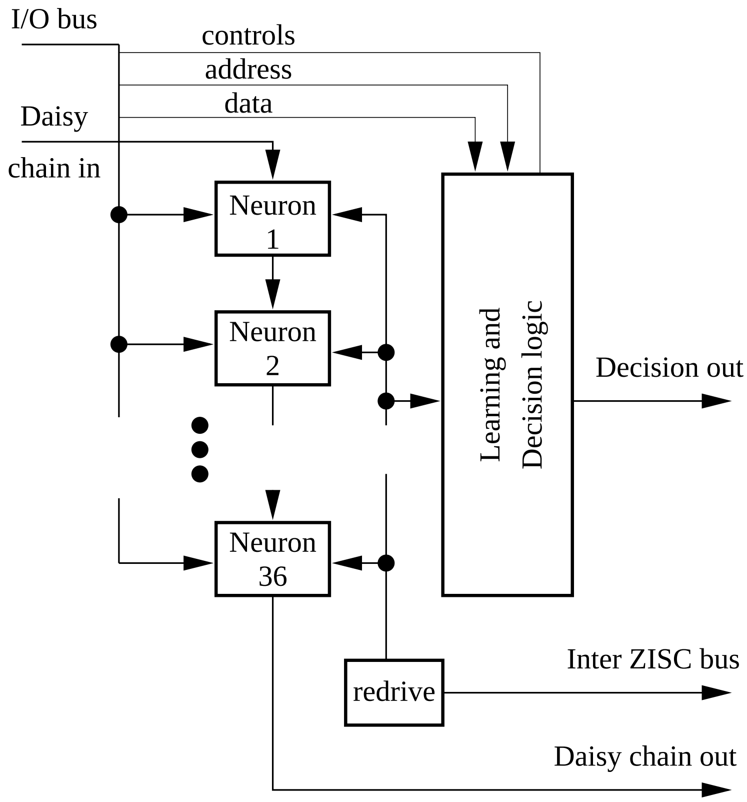

Figure 5.36: Block diagram of the ZISC036

bus. The commands are broadcasted on the command bus. The SIMD processors execute the same commands using different weights stored in local memory. The sequencer reads the result from each PN on the output bus, but deals with one PN at a time in this step. The CNAPS works in conjunction with a UNIX host, such as Sun SPARCstation. A Motorola 68030 based controller interfaces between the host and the CNAPS server. The control software stored in the local memory of the controller (4M byte). CNAPS server has flexible configuration with different number of chips which means different PNs. Some aviable configurations have 64, 128, 256, 512 PNs with 16M or 32M bytes RAM for quick acces to resident data files. Their performances are in table 5.3.

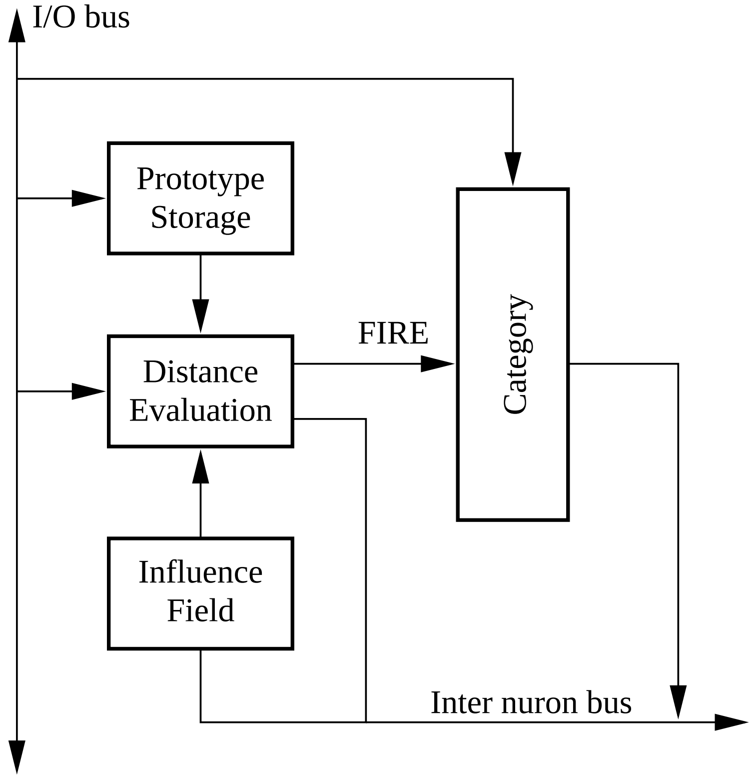

Figure 5.37: Block diagram of a neuron

Some software packages provides easy acces to the CNAPS capabilities:

- *BuildNet* is an experimentation and application development package that offers some precoded neural algorithms
- *CodeNet* is an application development package to support custom neural algorithm development in CNAPS assembly
- *CNAPS-C* is a specialized parallel C compiler for CNAPS
- *CNAPS Quicklib* is precoded function library for achiving maximum performance on CNAPS

	CNAPS64	CNAPS128	CNAPS256	CNAPS512
Number of PNs	**64**	**128**	**256**	**512**
Total PN Memory (Each PN has 4k bytes)				
1 Bit	2M Bits	4M Bits	8M Bits	16M Bits
8 Bit	256k bytes	512k bytes	1M bytes	2M bytes
16 Bit	128k bytes	256k bytes	512k bytes	1M bytes
Peak Performance				
MACC	960M	1.92G	3.84G	7.68G
BP recall, CPS	871M	1.7G	3.16G	5.70G
BP learning, CUPS	220M	429M	813M	1.46G

Table 5.3: Performance of CNAPS systems

SYNAPSE-1

Siemens developed a general purpose neurocomputer called SYNAPSE-1 using the previously mentioned (section 5.3.7) MA16 chips. SYNAPSE-1 (SYnthesis of Neural Algorithms on a Parallel Systolic Engine) is a very flexible neurocomputing system. A wide variety of different neural network algorithms can be simulated on it. The architecture of the system is on figure 5.39.

Four main components form a neurocomputer: the MA16 array board which consists of 8 MA16 processors, the weight memory which is typically 64M (16M to 512M) byte DRAM, the data unit with Motorola 68040 processor which computes the non computation-intensive parts of the algorithm, and the control unit which is also based on M 68040. The idea to compute the computationally intensive and less intensive tasks separately is very suitable to neural networks. The data and control unit offers universal programmability for these tasks thanks to the applied standard type processors. The MA16 array executes only the basic neural operations and the rest is for the Data unit. The data run between these two boards via a ring. See figure 5.40 for the processor memory architecture and data pathes.

A Y memory for input patterns and status values of the neurons is included in the loop. The W and Z memories are assigned to the MA16 array. W memory is for the weights while Z is buffering data for the MA16s. There are two memories for the Data unit, too. The DU Program memory stores the procedures and the C memory is a general purpose storage for the Data

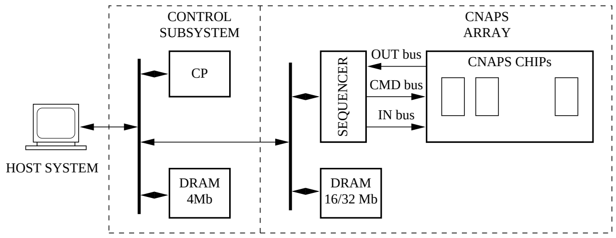

Figure 5.38: Block diagram of a neuron

unit. The DU Program memory hand by hand with CU Program memory contain operating system routines to control the whole system. The Control unit also has a microprogram sequencer which controls directly the MA16 array. The microcode of this sequencer has to be changed according to the given application. The whole system is connected to a host machine (e.g. SparcStation) via the VME bus.

SYNAPSE-1 is equipped with user friendly graphical interface and sophisticated development enviroment. The nAPL (neural Algorithms Programming Language) provides a high level language mapping to the multiprocessor and multimemory architecture. nAPL is embedded in C++ and implemented as a class library. There is also a graphical input and visualization system, the SENN++ (Simulation Enviroment for Neural Networks) implemented on the host computer.

Neurocomputers built from of the shelf elements

Ganglion

GANGLION is a fast digital implementation of a classifier using Xilinx FPGAs (Field Programmable Gate Arrays). The network is implemented on a VME card using some programmable chips. The classifier has twelve inputs, fourteen neurons in the hidden layer and four ones in the output layer. The weights and the inputs are also 8-bit wide while the sums (accumulator) are represented by 20 bits, scaled to 11 bits at the end of the operation and passed through an arbitrary activation function implemented by a 2kbyte (8-bit output) look-up table. The 8-bit by 8-bit multiplications

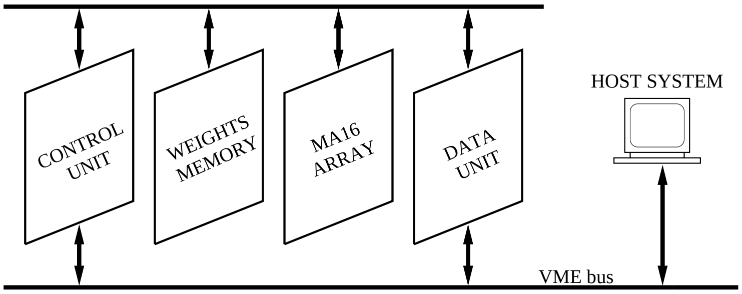

Figure 5.39: Block diagram of SYNAPSE-1

are performed by breaking them into two 8 bit by 4 bit multiplications and an addition. The adder trees are realized by three to two reduction adders and carry save adders. Three to two reduction adders have 3 inputs and 2 outputs.

The classifier performs 20 million decision per second which means 4.48G CPS.

KOKOS and KOBOLD

KOKOS (Univ. of Tübingen, Germany) is a scalable coprocessor for Kohonen's Self Organizing Map (SOM). It is implemented on off-the shelf elements and Xilinx FPGAs. A prototype consists of 8 Memory and Arithmetic Boards (MABs), an adder tree and a controller. The off-the shelf elements are used to build fast arithmetic like multiplier and the adder tree. Irregular structures like substructor with different datapaths for calculating the euclidian distance, controler, etc. are implemented on FPGAs. Performance of KOKOS with 4 MABs is 16M CUPS at 10MHz clock rate.

KOBOLD, a coprocessor for back-propagation consists of SubProcessors (SPs). Each SP represents a column of the network (one layer is one row, one column is a node from every layer). SPs organized into ring structure and additionally a local bus connects each SP with its direct neighbours. The SP has pipelined fixed-point arithmetic plus look-up table for sigmoid. KOBOLD similarly to the KOKOS is implemented with FPGAs and off-the-shelf elements. Its performance is 122M CUPS in worst case for 128 SPs.

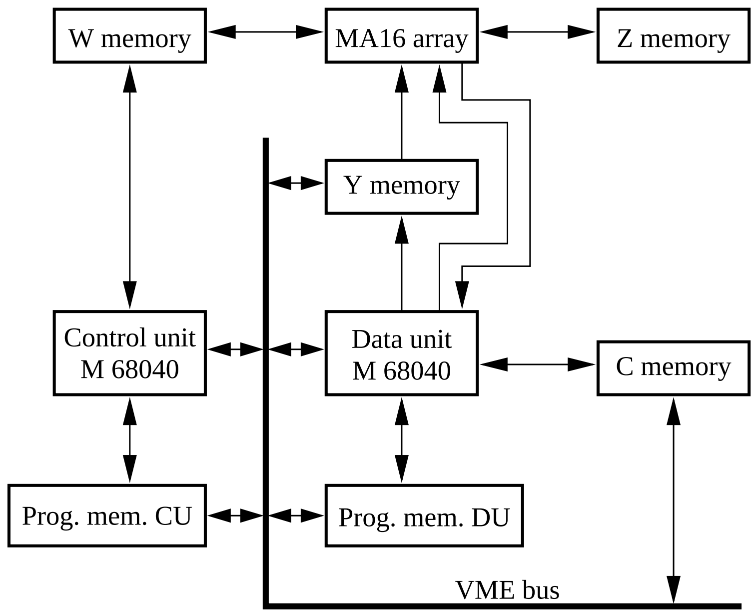

Figure 5.40: Processor memory architecture and data pathes

Neurocomputer built from commercial processors

BSP 400

BSP400 (Brain-Style Processor) is built at Leiden University. It consists of 25 modules (boards) each containing 16 simple 8-bit Motorola MC68701 single-chip computers. The 400 processing elements have 32000 connections and perform 6.4M CPS. Each processor has on-chip EPROM and RAM. The BSP400 system use one to one projection of the nodes to the processors. This means that the EPROM of the processors contains the node software, including activation function. The weight memory is implemented in the on-chip RAM. One node can handle 80 synapses with 8-bit weights because of the limited size of the RAM.

The overall architecture of BSP400 system is in figure 5.41. Each unit on figure 5.41 represents a board with processors or an I/O board which interfaces to sensors, activators, etc. This makes the system suitable for some real-word applications. A network controller controls the units, broad-

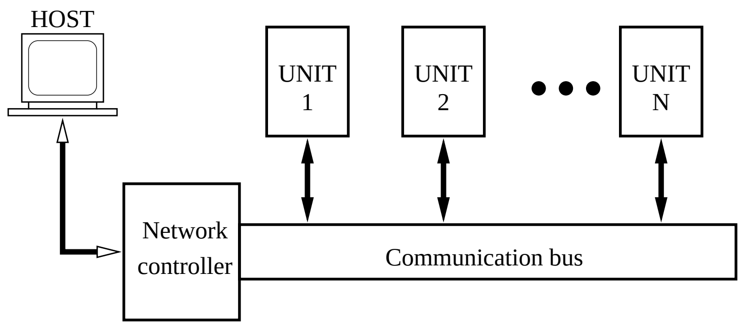

Figure 5.41: Block diagram of BSP400

casts the commands to them and interfaces a host machine to the BSP400 system. The Module and I/O boards are transparent in point of view of network controller.

Accelerator boards

There are a wide variety of accelerator boards in commerce and at research groups, too. These boards provide hardware support for simulators. They can be built from commercial processors, DSP processors or neurochips. Some example are shortly introduced in this section.

- **BrainMaker Professional** (California Scientific Software)

  Description: Two ISA-bus 20MHZ DSP boards on 4 Texas DSPs each. It includes 5MB RAM (for 600 k-connections and 250 k-data points), expandable to 32MB (3.6 million connections and 4 million data points).

  Performance: 40MCPS

- **BrainMaker Professional SNAPS System** (California Scientific Software)

  Description: The CNAPS/PC accelerator board comes with 64 or 128 processors (Adaptive Solution CNAPS) in PCI or ISA-bus version. Both come with 4 Kbytes of on-chip memory per processor, and 4 Mbytes data storage expandable to 36 Mbytes via a SIMM. The trained network's connections (the weight matrix) must fit into board processor memory. The training data does not have to fit onto board data storage since Windows will page out to disk if there is not enough memory for all the facts on board (virtual memory management). Their performance for back-propagation are 1.16G CPS and 293M CUPS for 64 processor card and 2.27G CPS and 573M CUPS for 128 processor card.

- **ZISC/ISA 576 Accelerator for PC** (IBM)

  Description: A card with 16 ZISC036 chips has been built for the ISA bus. The chips appear as one ZISC with 576 prototype vectors. The card comes with a C language library for accessing all the card, training, recall, etc. The Neural Network can be further extended by adding cards in the PC : One card is configured as a Master while other cards are slave. The communication between the cards is done through a ZISC Extension parallel port, and a daisy chain. Configuration with 5 cards (2,880 neurons) has been tested.

- **SYNAPSE2*PC** (Siemens)

  Description: Using one of Siemen's MA-16 matrix processor chips, the SYNAPSE2*PC board (the little brother of the SYNAPSE1) is a general purpose PC coprocessor that can accelerate neural network as well as other vector-matrix intensive algorithms, e.g. image processing. The card fits into the PCI bus and can receive data at 30MBytes/sec rate. Available software includes a C++ library to interface to the board, an image processing C++ library, and a graphical application development package. The host system must use Windows NT and have 32Mbytes RAM.

  Learning: Can be programmed for various neural network learning algorithms.

  Performance: The MA-16 executes 1.280 billion OPS at a clock rate of 40MHz.

- **Model 1000 NeuroEngine** (Telebyte Technology Inc.)

  Description: The Model 1000 NeuroEngine is a coprocessing board which plugs into a PC and brings high speed neural network processing to the desktop. The board is based upon unique parallel processing concepts developed by AAC. The unit is a multiple instruction multiple data (MIMD) stream coprocessor for the PC. The unique MIMD parallel processing architecture takes advantage of sparse topology to obtain high computational efficiency. This single card fits into a PC and operates under DOS. The board is capable of running 140M CPS. Up to 10 PC Neurocomputers can be configured as a parallel processor to execute 1.4G CPS. Multiple interconnected neural

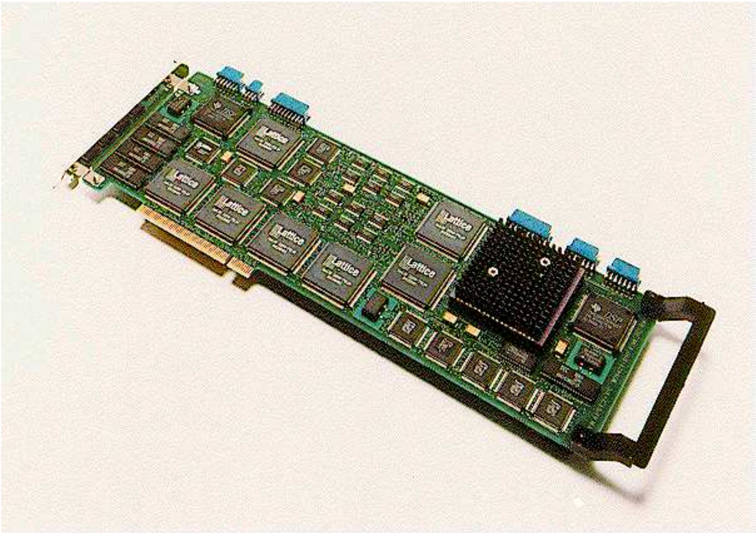

Figure 5.42: Photo of the SYNAPSE2*PC board

networks of different architectures can be implemented simultaneously with one or several neurocomputers. It uses 16 bit, 2's complement arithmetic to realize up to 8,000 neurons and 32,000 connection weights in each PC Neurocomputer.

Other neurocomputers

Some other examples can be seen in the Table 5.4 and Table 5.5.

General purpose parallel machines

Many parallel neuroimplementations were made on existeing general purpose parallel machines simple because neural hardwares were not available. The Connection machine, Warp, Maspar, Huges, GF11, AAP-2, CRAY YM-P supercomputer ware the most popular parallel machines used for neurosimulations. Some of them are shortly introduced in this section.

Warp is a systolic computer consists of three parts: the Warp processor array, a host machine, and an interface between them (figure 5.43. The system consists of 10 or more Warp cells in a linear chain. Each Warp cell can be used independently controlled by a user programmable microengine.

Name	Chip	Performance	Notes
Bellcore VME cascadable card	3 ETANN		
ETANN card, Bologna	ETANN, M68070		
RAP, Berkley	4-40 TMS320C40	57M CPS, 13.2M CUPS per board	VME bus
Texas, NETSIM	80188	450M CPS, 90M CUPS	15 cards, 256 neuron with 256 synapses each per board
REMAP, Luleåa Univ.	FPGAs	100-1G CPS	bit serial SIMD

Table 5.4: Comparisation table of some other neurocomputers

Name	Chip	Performance	Notes
Current Tech. MM32k	spec. SIMD	4.9M CPS, 2.5M CUPS	
HNC Balbo 860	i860	80MFLOPS	
Neural Tech. NT6000	TMS320C20, MT19003	2M CPS	
NeurodynamX XR50	i860	45M CPS	
Sundance, SMT306	2 MT19003	2x 40M CPS	

Table 5.5: Comparisation table of some other accelerator boards

This allows systolic or MIMD shape operations. A Warp cell has 32k word local memory, a 10 MIPS integer arithmetic unit, and 5 MFLOPS floating multiplier and adder.

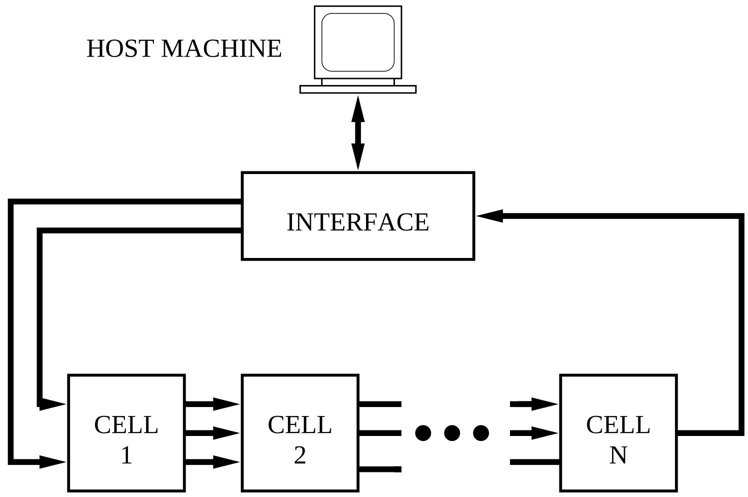

Figure 5.43: The Warp system

MasPar, massively data-parallel computer, is a SIMD array machine, too. MasPar computer family has members with 1024 to 16,384 processing elements. MasPar systems built from chips each consists of 32 processing elements which are small RISC processors with 48 32-bit registers, external memory management unit (16k bytes memory are avaiable for a processing element), and arithmetic unit. It can perform 8, 16, 32, or 64-bit floating point operations. These 32 PE processor chips are ordered in a two-dimensional array. A 4x4 PE matrix forms a cluster. The edges of the clusters are connected in a torus. The communication between these clusters is supported by an X-Net mesh interconnect which provides easy access to the eight nearest neighbors in the array. Global communication is realized by crossbar switches. The system performance is 1600 MIPS, 82 MFLOPS (32-bit) or 36 MFLOP (64-bit) with 1k PEs and 26,000 MIPS, 1300 MFLOPS (32-bit) or 580 MFLOP (64-bit) with 16k PEs. An experimental 900 input units, 20 hidden units and 17 output units network with backpropagation performs 306k CUPS on a 4k PE machine.

Software Tools

There are a lots of simulators available. Some of them are free- or shareware and there are many commercial ones too. Here only some foretastes are given.

Template simulators

- **SNNS** (IPVR at the Univ. of Stuttgart)

  SNNS (Stuttgart Neural Network Simulator) is a software simulator for neural networks on Unix workstations. The SNNS simulator consists of two main components:

  1. simultor kernel written in C
  2. graphical user interface under X11R4 or X11R5

  The simulator kernel operates on the internal network data structures of the neural nets and performs all operations of learning and recall. It can also be used without the other parts as a C program embedded in custom applications. It supports arbitrary network topologies. SNNS can be extended by the user with user defined activation functions, output functions, site functions and learning procedures, which are written as simple C programs and linked to the simulator kernel. Currently the following network architectures and learning procedures are included: Backpropagation (BP) for feedforward networks (vanilla (online) BP, BP with momentum term and flat spot elimination, batch BP), Counterpropagation, Quickprop, Backpercolation 1, RProp, Generalized radial basis functions (RBF), ART1, ART2, ARTMAP, Cascade Correlation, Recurrent Cascade Correlation, Dynamic LVQ, Backpropagation through time (for recurrent networks), Quickprop through time (for recurrent networks), Self-organizing maps (Kohonen maps), TDNN (time-delay networks) with Backpropagation, Jordan networks, Elman networks and extended hierarchical Elman networks, Associative Memory.

  The graphical user interface XGUI (X Graphical User Interface), built on top of the kernel, gives a 2D and a 3D graphical representation of the neural networks and controls the kernel during the simulation run. In addition, the 2D user interface has an integrated network editor which can be used to directly create, manipulate and visualize neural nets in various ways.

- **BrainMaker** (CNS)

BrainMaker is one of the most popular commercial neural simulator software. It can be used in business, finance, medical, manufacturing, etc. applications. BrainMaker has more analysis features for both raw data and the trained neural network's solution. Raw data analysis: data correlator, cyclic analysis, column histograms, financial indicators, graphs. Network analysis: sensitivity analysis, all inputs, all facts, what-if scenarios, etc. It reads Excel, Lotus, dBASE, ASCII text, and binary files. Professional also reads Computrac, Metastock, and CSI files. BrainMaker Professional is available with hardware accelerator boards for the PC.

- **NEU*frame* Professional v3.0**

  NEUframe Professional v3.0 Features :

  - Intelligence Technologies

    Supervised Learning Neural Networks

    Unsupervised Learning (Kohonen) Neural Networks

    Neuro-fuzzy logic
  - Object Oriented Design

    Manipulate data using powerful object based analytical tools

    Create a visual model of your problem
  - Data Flow Architecture
  - Ease of Use

    GUI Development environment available now for Windows 95, Windows NT

    Templates - pre-formed workspaces, simply data in and press 'Play'

    Report Window - context sensitive help, identifies on line method errors

- **Trajan 2.1 Neural Network Simulator** (Trajan Software Ltd.)

  Implemented architectures in Trajan 2.1: Multilayer Perceptron, Radial Basis Function, Kohonen, Probabilistic and Generalised Regression Neural Networks, Linear, Hybrids. Training algorithms: Levenburg-Marquardt, Conjugate Gradient Descent, Back Propagation, Quick Propagation, Delta-Bar-Delta, Kohonen, Singular Value Decomposition, K-Means, K-Nearest, Principal Components Analysis, Automatic Network Design, Neuro-Genetic Input Selection, Weigend Weight Regularisation.

  The simulator has well developed user interface.

Special features:

- *Automatic Design.* Trajan includes two unique state-of-the-art algorithms to simplify the Neural Network design process:

  Automatic Network Designer. One of the most difficult problems is choosing a good architecture (type and size of network) for the problem at hand. Trajan's Automatic Network Designer uses sophisticated optimisation techniques including Simulated Annealing to automatically search for a suitable architecture.

  Neuro-Genetic Input Selection. Another key problem is identifying the important inputs for a particular problem. Possible correlations and inter-dependencies between inputs make this a highly complex problem. Trajan's Neuro-Genetic Input Selection algorithm combines a Genetic Algorithm with the extremely rapid Bayesian (PNN and GRNN) networks to automatically select an appropriate input mask.
- *Custom Architectures.* Trajan allows selecting special Activation functions and Error functions; for example, to use Softmax and Cross-entropy for Probability Estimation, or City-Block Error function for reduced outlier-sensitivity. There are also facilities to "splice" networks together and to delete layers from networks, allowing you to rapidly create pre- and post-processing networks, including Autoassociative Dimensionality Reduction networks, or Loss Matrices.

Non-template simulators

- **PDP++** (CMU, CNBC, NPC)

  Software Features:

  - Full Graphical User Interface (GUI) based on the InterViews toolkit. Allows user-selected "look and feel".
  - Network Viewer shows network architecture and processing in real- time, allows network to be constructed with simple point-and-click actions.
  - Training and testing data can be graphed on-line and network state can be displayed over time numerically or using a wide range of color or size-based graphical representations.
  - Environment Viewer shows training patterns using color or size-based graphical representations.

  - Flexible object-oriented design allows mix-and-match simulation construction and easy extension by deriving new object types from existing ones.
  - Built-in 'CSS' scripting language uses C++ syntax, allows full access to simulation object data and functions. Transition between script code and compiled code is simplified since both are C++. Script has command-line completion, source-level debugger, and provides standard C/C++ library functions.
  - Scripts can control processing, generate training and testing patterns, automate routine tasks, etc.
  - Scripts can be generated from GUI actions, and the user can create GUI interfaces from script objects to extend and customize the simulation environment.

  Supported Algorithms Include:

  - Feedforward and recurrent error backpropagation. Recurrent BP includes continuous, real-time models, and Almeida-Pineda.
  - Constraint satisfaction algorithms and associated learning algorithms including Boltzmann Machine, Hopfield models, mean-field networks (DBM), Interactive Activation and Competition (IAC), and continuous stochastic networks.
  - Self-organizing learning including Competitive Learning, Soft Competitive Learning, simple Hebbian, and Kohonen's Self-organizing Maps.
  - Mixtures-of-experts using backpropagation experts, EM updating, and a SoftMax gating module.

Network specific simulators

- **Mathematica neural network libraries** (Wolfram Research)

  Mathematica is a general purpose math software with a lots of useful math tools. It has a library which contains programs for neural network simulation. Actually, each program is a network specific simulator. The currently available programs are: Adaline.m, Art.m, Backpropagation.m, Bam.m, Elman.m, FunctionalLink.m, GeneticAlgorithms.m, Jordan.m, ProbabilisticNets.m, TravelingSalesperson.m.

- **NNSYSID Toolbox for Matlab** (MathWorks Inc.)

MATLAB is also a general prupose math software with special toolboxes. The NNSYSID toolbox is a set of MATLAB tools for neural network based identification of nonlinear dynamic systems. The toolbox contains a number of programs for training and evaluation of multilayer perceptron type neural networks within the MATLAB environment. There are functions for training ordinary feedforward networks as well as for identification of nonlinear dynamic systems and for time-series analysis.

5.3.8 References and further readings

I.F. Croall, J.P. Mason, “ Industrial Applications of Neural Networks”, ESPRIT Research Reports, Springer-Verlag, Brussels-Luxembourg, 1992.

S.Y. Kung , “ Digital Neural Networks”, Pentice Hall, Englewood Cliffs, New Jersey, 1993.

N. B. Šerbedžija, “ Simulating Artifical Neural Networks on Parallel Architectures ”, Computer IEEE 1996 March

E. van Keulen, S. Colak, H. Withagen, H Hegt, ” Neural Network Hardware Performance Criteria”, IEEE Int. conf. on Neural Networks, Vol. 3 pp. 1885-1893, 1994

C. S. Lindsey, T. Lindblad, ”Review of Hardware Neural Networks: A User’s Perspective”, TRITA FYS 9012 KTH-Frescati, 1994 It is also aviable at http://msia02.msi.se/ lindsey/elba2html/elbaFramed.html

C. S. Lindsey, B. Denby, T. Lindblad, ”Neural Network Hardware”, http://www1.cern.ch/NeuralNets/nnwInHepHard.html rev.: 28 May 1997

J. N. H. Heemskerk, ”Overview of Neural Hardware”, Draft version of the Chapter 3 in Heemskerk, J.N.H (1995). *Neurocomputers for Brain Style Processing, Design, Implementation and Application.* PhD thesis, Unit of Experimental and Theoretical Psychology Leiden University, The Netherlands, (ftp.mrc-apu.cam.ac.uk/pub/nn), 1995

H. Speckmann, P. Thole, M. Bogdan, W. Rosenstiel, ”Coprocessors for special neural networks, KOKOS and KOBOLD”, IEEE Int. conf. on Neural Networks, Vol. 3 pp. 1959-1962, 1994

Y. Fujimoto, N. Fukuda, T. Akabane, "Massively Parallel Architectures for Large Scale Neural Network Simulations", IEEE Tr. on Neural Networks, Vol. 3, No. 6, pp. 876-887, November 1992

J.M.J. Murre, "Neurosimulators", draft version of a chapter from "Handbook of Brain Research and Neural Networks", M.A. Arib (ed.), MIT Press, in preparation for 1995, ftp-able from ftp://ftp.mrc-apu.cam.ac.uk/pub/nn/murre/neurosim1.ps, MRC APU, Cambridge, 1993

M. Misra, "Parallel Enviroments for Implementing Neural Networks", Neural Computing Surveys, vol. 1. 48-60, 1997, http://www.icsi.berkeley.edu/ jagota/NCS

X. Arreguit, E.A. Vittoz, "Analog VLSI Hardware: What is Missing for Industrial Realisations?", IEEE Int. conf. on Neural Networks, Vol. 3 pp. 1883-1884, 1994

M. Yasunaga, N. Masuda, M. Yagyu, M. Asai, K. Shibata, M. Ooyama, M. Yamada, T. Sakaguci, M. Hashimoto, "A Self-Learning Digital Neural Network Using Wafer-Scale LSI", IEEE Journal of Solid-state Circuits, Vol. 28, No. 2, February 1993

M. Glesner, W. Pöchmüller, "Neurocomputers", Chapman & Hall, Cambridge, 1994

Intel Corporation's data sheet, "80170NW Electronically Trainable Analog Neural Network", Intel˘C Corporation, 1990 May

Adaptive Solution Inc.'s data sheets

U. Ramacher, W. Raab, J. Anlauf, U. Hachmann, M. Wesseling, "SYNAPSE-1 - A General-Purpose Neurocomputer" Siemens AG. 1993

A.F. Murray, D. Del Corso, L. Tarassenko, "Pulse-Stream VLSI Neural Networks Mixing Analog and Digital Techniques", IEEE Trans. on Neural Networks, Vol. 2, No. 2, March 1993

N. Maudit, M. Duranton, J. Gobert, J.A. Sirat, "Lneuro 1.0: A Piece of Hardware LEGO for Building Neural Network Systems", IEEE Trans. on Neural Networks, Vol. 3, No. 3, May 1992

"ZISC 036 Neurons, User's Manual", IBM Component Development Lab, Microelectronics Division, Jan. 1997

C.E. Cox, W. E. Blanz, "GANGLION - A Fast Hardware Implementation of a Connectionist Classifier" IEEE Custom Integrated Circuit Conference, CH2994-2/91/0000/0023, 1991

T. Duong, S. Kemeny, T. Daud, A. Thakoor, C. Saunders, J. Carson, "Analog 3-D Neuroprocessor for Fast Frame Focal Plane Image Processing", Simulation, pp. 11-25, July 1995

L.O. Chua, T. Roska, T. Kozek, Á. Zarándy, "CNN Universal Chips Crank Up the Computing Power", IEEE Circuits & Devices, pp. 18-28, July 1996

J.N.H. Heemskerk, J. Hoekstra, J.M.J. Murre, L.H.J.G. Kemna, P.T.W. Hudson, "The BSP400: A Modular Neurocomputer", to appear in Microprocessors and Microsystems, March 1994, from ftp.mrc-apu.cam.ac.uk/pub/nn/murre