1.3 Neural Networks

Neural Networks are large structured systems of equations. These systems have many degrees of freedom and are able to adapt to the task they are supposed to do [Gupta].

Two very different types of Neural Networks exist: **Backpropagation Neural Networks** and **Probabilistic Neural Networks**. Backpropagation Neural Networks use equations that are connected using weighting factors. The selection of the weighting factors makes these Neural Nets so powerful. Probabilistic Neural Nets use statistical methods to select the equations within the structure and do not weight these functions differently [Chrekassky].

1.3.1 Backpropagation Neural Networks

1.3.1.1 Structure and Algorithm of a Backpropagation Neural Network

The Backpropagation Neural Networks consist of **neurons** organized in one input layer and one output layer and several hidden layers of neurons (Fig. 1.3-1). Several planes can exist that are generally built up like the first plane [Zurada]. Neurons perform some kind of calculation. They use inputs to compute an output that represents the system. The outputs are given on to a next neuron. An arc indicates to which neurons the output is given. These arcs carry weights.

In (Fig. 1.3-1) the structure of a backpropagation neural network is shown. The Input vector consists of variables that are used to predict the desired variable. The input vector, using an example of a heat exchanger with inputs and outputs. The inputs could consist of the mass flow rate of water, the inlet temperature of water, the velocity of the air flowing across the pipe and the temperature of the air. The output could be the effeciveness of the heat exchanger. This information of the input is given to the input layer of the neural network. In the input layer the data are normalized. A neural network does not work in a unit system. A neural network looks solely at the numbers. It is therefore necessary to normalize the input. This means if the temperature changes over the whole range from *300K-400K* the range of the temperature is normalized to a range of *0-1*. The same normalization happens for each individual input.

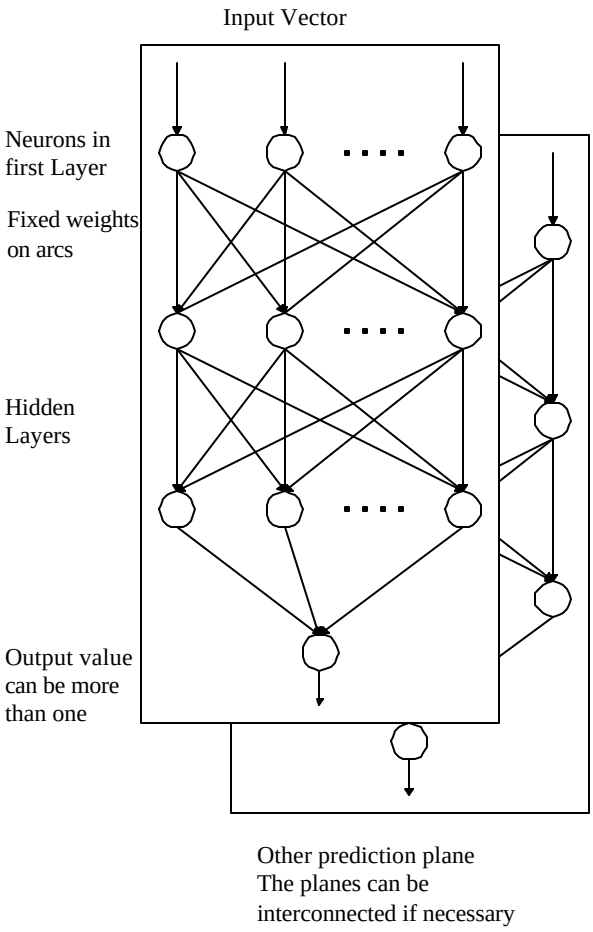

Fig. 1.3-1 Fig organization of neural network

The normalized signal is given on to the next neuron. Each neuron receives signals from any number of other neurons [Zurada]. The architecture of the neural network does not need be as shown in (Fig. 1.3-1) but for now we assume the architecture of the neural network as shown in (Fig. 1.3-1). Each signal is weighted as it is given from the input layer to the first hidden layer. As the new signals reaches a neuron in the hidden layer they all the signals that are received by a neuron are summed up. This process can be looked at as a vector multiplication (Eqn. 1.3-1) of the weight vector $\mathbf{w}$ and the signal vector $\mathbf{y}_{\mathbf{previous}}$ of the previous

layer [Zurada]. In the hidden layer new signals are computed and given on to the next hidden layer. This process continues until the output layer is reached.

$$x = \mathbf{w} * \mathbf{y}_{\mathbf{previous}}$$

Eqn. 1.3-1

The scalar x causes the neuron to react in a certain way. There are two very widely-used functions that are used in backpropagation neural networks: the threshold function (Eqn. 1.3-2) and the sigmoid function (Eqn. 1.3-3).

$$y_{new,i} = \boldsymbol{threshold}(x)$$

Eqn. 1.3-2

$$y_{new,i} = \boldsymbol{sigmoid}(x)$$

Eqn. 1.3-3

The process going on in a neuron is shown in (Fig. 1.3-2). It can be seen that the sigmoid function has a continuous output whereas the output from the threshold function is either *0* or a value, usually one.

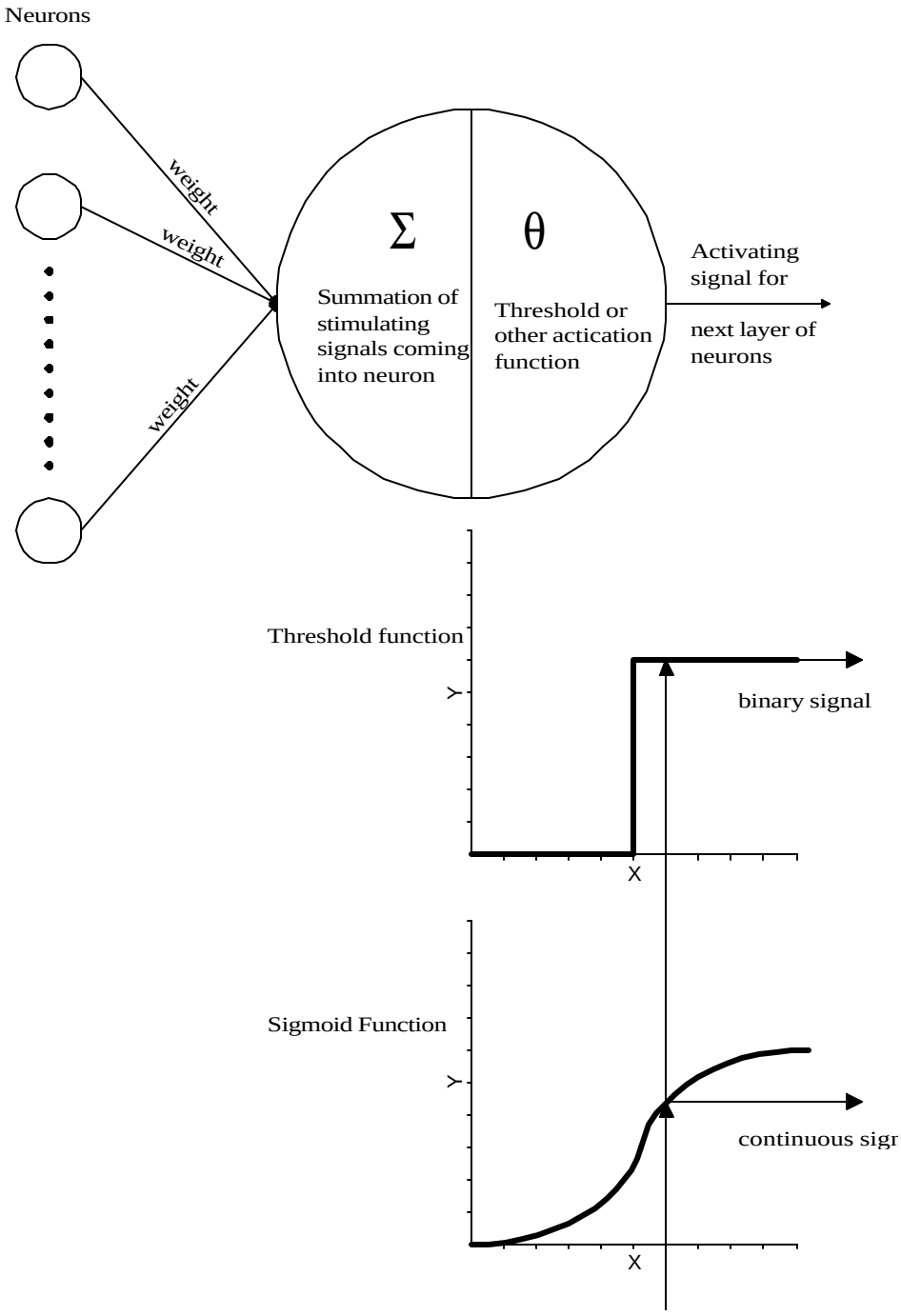

Fig. 1.3-2 Sigmoid function and Threshold function

The following process in the neural network is basically a repetition of the process that was just described. The values of the signals and the weights will be different but the process

itself will continue similarly. The signals are weighted and then summed up, they cause a reaction of the neuron and the reaction, which is the new signal, will be given on to the next neuron.

The last step is to give the signal to the output layer. The output layer can consist of one or more neurons. More neurons would mean that the plane of the neural network has multiple outputs. In the output neuron a calculation is again necessary to yield a value that is not normalized and has again a physical meaning.

The weights on the connections are not chosen arbitrarily, but result from training. Several training algorithms exist. The well known backpropagation training, that actually gave the name for the backpropagation neural network, is discussed in Section 2.3.1.2.

1.3.1.2 Training of a Backpropagation Neural Network

Initially the network predicts an output for one input vector for which the true output is known. This combination of known output and input vector is called **training sample.** The predicted output is compared to the known value. The weights on the arcs are adjusted depending on how the prediction of the actual result. In (Fig. 1.3-3) the general process in a backpropagation neural network is shown again. This figure is used for nomenclature in the process shown in (Fig. 1.3-4) for the training.

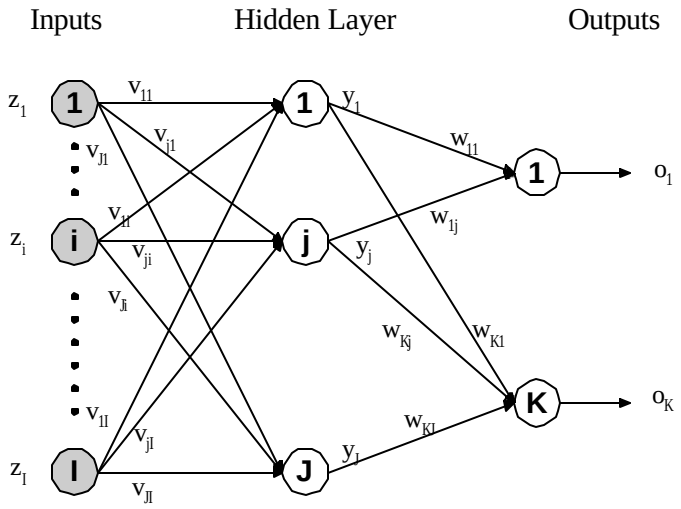

Fig. 1.3-3 Backpropagation neural network, nomenclature used for learning description

In (Fig. 1.3-4) the first step in the training is to assume initial weights. With these initial weights the first output for one training sample, ***o***, is calculated. The third step in the training is to calculate the cycle error made in the prediction. The cycle error is as shown in (Fig. 1.3-4) the sum of the squared differences between the known result the neural network should predict, ***d***, and the result, ***o***, the neural network does predict. The cycle error is computed for each run through the training data. The cycle error is used in each cycle of the training to compare to the maximal allowable cycle error in step eight of (Fig. 1.3-4). In the fourth step, the training signals are calculated. The weights between the different layers are changed according to these training signals, that are calculated in step four in (Fig. 1.3-4). The process is repeated until no more training samples are available. If there are no more training samples available the cycle error then is calculated and compared to the maximal allowable error. If the cycle error exceeds the maximal allowable cycle error, then the training has to start with a new

cycle. This is repeated until the cycle error is small enough. The training of Backpropagation Neural Networks is a very time consuming process [Zurada]. Note that the error signal vector must be determined in the output layer first and then it is propagated back toward the network input nodes.

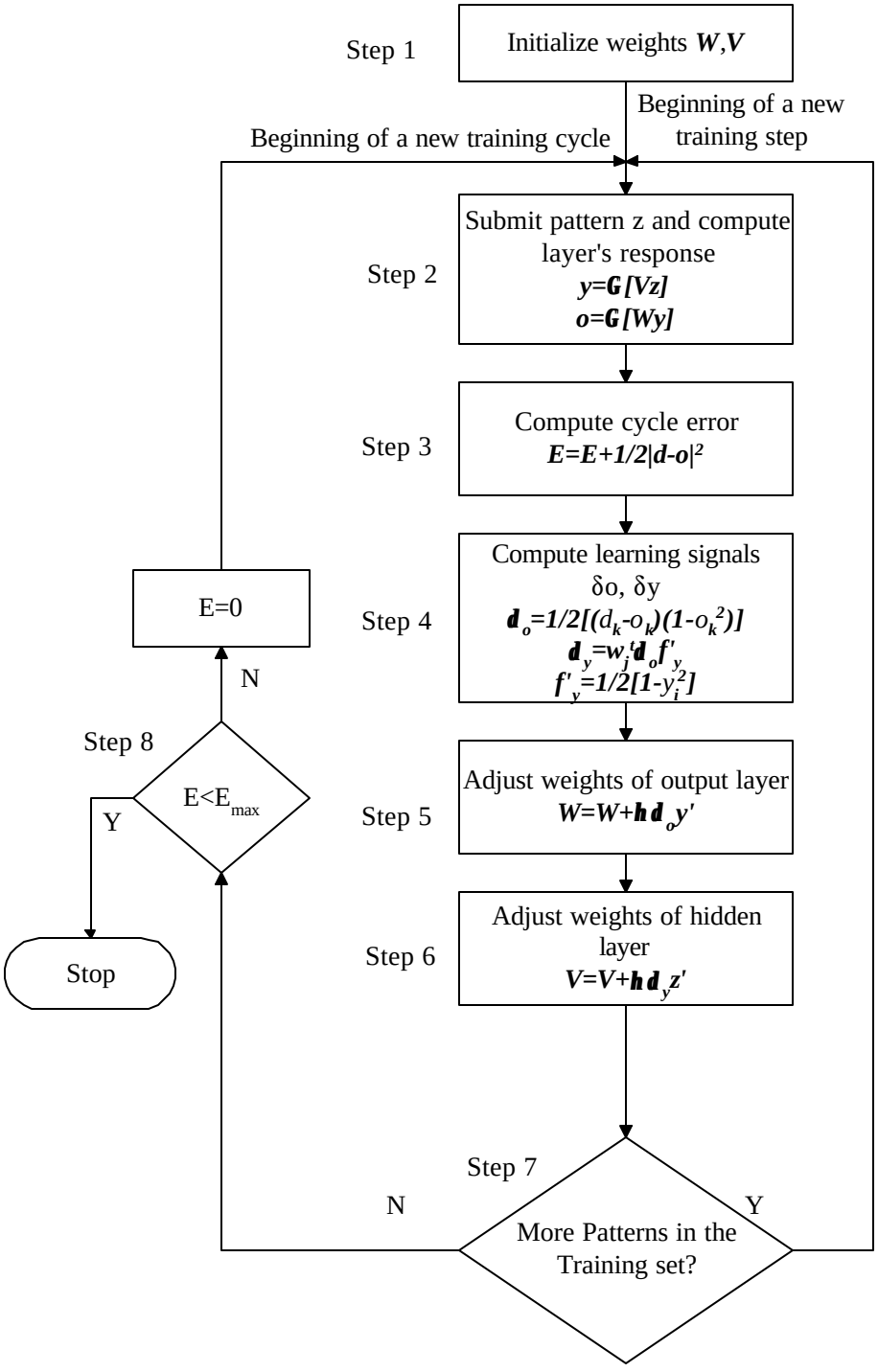

Fig. 1.3-4 Block Diagram for Backpropagation Learning Algorithm [Zurada]

Training of Backpropagation Neural Nets depends on the order in which the training samples are used [Zurada]. The dependence on order is a very critical point. It is not known how the order influences the selection of the weights. A frequently used method is to randomly choose the order of the samples and repeat it several times to see how the weights change depending on the order of the training samples [Gupta].

1.3.1.3 Pros and Cons of Backpropagation Neural Nets

Backpropagation Neural Networks have the very desirable characteristic of being very flexible. They can be used for pattern recognition as well as for decision making problems. Another advantage is like for every other neural network that the process is highly parallel and therefore the use of parallel processors is possible and cuts down the necessary time for calculations [Specht 91, Gupta, Cherkassky].

Backpropagation Neural Networks have negative characteristics. The training of the network can need a substantial amount of time [Gupta]. As soon as the training is done though the network performs very fast. In some respect a disqualifying aspect, is that the size of the training data for Backpropagation Neural Nets has to be very large. Often it is almost impossible to provide enough training samples [Zurada]. Examples would be when training samples result out of very expensive experiments. Or when the data are observations of nature and these observations can only occur very rarely.

1.3.2 Probabilistic Neural Network

Probabilistic Neural Nets use a statistical approach in their prediction algorithm. The bases for the statistical approach is given in the "The Bayes Strategy for Pattern Classification" [Specht 90].

1.3.2.1 The Bayes Strategy for Pattern Classification

Bayes strategies are strategies used to classify patterns. These strategies can be applied to problems containing any number of categories[Parzen, Specht 90]. Consider a situation for which the state of nature θ is known to be either $\theta_{\mathbf{A}}$ and $\theta_{\mathbf{B}}$. It has to be decided in which of these two states θ is. The available information is a p-dimensional input vector $\mathbf{X}^{\mathbf{T}} = \left[X_1, \ldots X_i, \ldots X_p \right]$. The Bayes decision rule becomes

$$\begin{aligned} \boldsymbol{d}(\mathbf{X}) &= \boldsymbol{q}_{\mathbf{A}} \quad \text{if} \quad \boldsymbol{h}_{\boldsymbol{A}} \boldsymbol{I}_{\boldsymbol{A}} \boldsymbol{f}_{\boldsymbol{A}}(\mathbf{X}) > \boldsymbol{h}_{\boldsymbol{B}} \boldsymbol{I}_{\boldsymbol{B}} \boldsymbol{f}_{\boldsymbol{B}}(\mathbf{X}) \\ \boldsymbol{d}(\mathbf{X}) &= \boldsymbol{q}_{\mathbf{B}} \quad \text{if} \quad \boldsymbol{h}_{\boldsymbol{A}} \boldsymbol{I}_{\boldsymbol{A}} \boldsymbol{f}_{\boldsymbol{A}}(\mathbf{X}) < \boldsymbol{h}_{\boldsymbol{B}} \boldsymbol{I}_{\boldsymbol{B}} \boldsymbol{f}_{\boldsymbol{B}}(\mathbf{X}) \end{aligned}$$

Eqn. 1.3-4

where $\boldsymbol{f}_{\boldsymbol{A}}(\mathbf{X})$ and $\boldsymbol{f}_{\boldsymbol{B}}(\mathbf{X})$ are probability density functions for categories *A* and *B*. $\boldsymbol{I}_{\boldsymbol{A}}$ is the loss function associated with the decision $\boldsymbol{d}(\mathbf{X}) = \boldsymbol{q}_{\mathbf{A}}$ when $\boldsymbol{q} = \boldsymbol{q}_{\mathbf{B}}$. $\boldsymbol{I}_{\boldsymbol{B}}$ is the loss function for the decision $\boldsymbol{d}(\mathbf{X}) = \boldsymbol{q}_{\mathbf{B}}$ when $\boldsymbol{q} = \boldsymbol{q}_{\mathbf{A}}$, respectively. The losses for the correct decision are equal to zero. The ratio of the loss functions in (Eqn. 1.3-6) can usually be set to -1 (an inverter) if there is no reason for biasing the decision. Otherwise it would mean that the decision $\boldsymbol{d}(\mathbf{X}) = \boldsymbol{q}_{\mathbf{A}}$ for example cannot be trusted as much as the decision $\boldsymbol{d}(\mathbf{X}) = \boldsymbol{q}_{\mathbf{B}}$. $\boldsymbol{h}_{\boldsymbol{A}}$

is the prior probability of occurrence of patterns of the category A. $\boldsymbol{h_B} = 1 - \boldsymbol{h_A}$ is the prior probability of $\boldsymbol{q} = \boldsymbol{q_A}$ [Parzen].

The two category decision surface according to (Eqn. 1.3-4) therefore is

$$\boldsymbol{f_A}(\mathbf{X}) = \mathbf{K}\, \boldsymbol{f_B}(\mathbf{X})$$

Eqn. 1.3-5

with

$$\mathbf{K} = \frac{\boldsymbol{h_B I_B}}{\boldsymbol{h_A I_A}}.$$

Eqn. 1.3-6

A similar decision rule can be stated for multi-category problems [Parzen, Specht 90]. A decision surface separating two states of nature is shown in (Fig. 1.3-5). The decision surface can be arbitrarily complex since there are no limitations on the densities except the restrictions on the probability density functions. Namely

- non negative
- integrable
- integral over space equals unity.

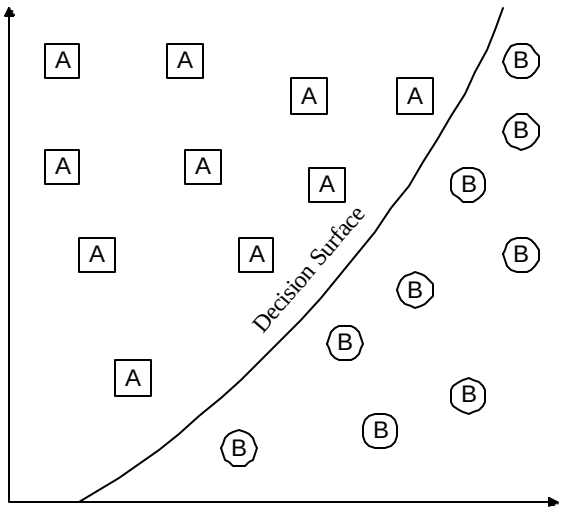

Fig. 1.3-5 Decision surface between states of nature A and B

To be able to use (Eqn. 1.3-5) it is necessary to estimate the probability density function accurately. The only available information to estimate the density functions are the training samples. The loss functions require subjective evaluation if the decision is biased. A loss function is a function that subtracts a value of the overall value for a wrong decision [Specht 90].

According to [Parzen] a class of probability density function estimates asymptotically approaches the underlying parent density function. The only requirement is that the density function is continuous.

Parzen [Parzen] showed how to construct a family of estimates of the density function

$$f_{\mathbf{n}}(\mathbf{X}) = \frac{1}{\mathbf{n}\, l} \sum_{\mathbf{i}=1}^{\mathbf{n}} W\left(\frac{\mathbf{X} - \mathbf{X_{Ai}}}{l} \right)$$

Eqn. 1.3-7

which is consistent at all points X at which the probability density function is continuous. Parzen imposes on the weighting function $W(\mathbf{y})$ which are

$$\sup_{-\infty<\mathbf{y}<\infty} \left| W(\mathbf{y}) \right| < \infty$$

Eqn. 1.3-8

where ***sup*** indicates the supremum. (Eqn. 1.3-8) says that the weights are not unbound and cannot reach infinity. Where

$$\int_{-\infty}^{\infty} \left| W(\mathbf{y}) \right| d\mathbf{y} < \infty$$

Eqn. 1.3-9

$$\lim_{\mathbf{y}\to\infty} \left| \mathbf{y}\, W(\mathbf{y}) \right| = 0$$

Eqn. 1.3-10

and

$$\int_{-\infty}^{\infty} W(\mathbf{y})\,d\mathbf{y} = 1.$$

Eqn. 1.3-11

In (Eqn. 1.3-7) λ is chosen as a function of **n** such that

$$\lim_{\mathbf{n}\to\infty} l(\mathbf{n}) = 0$$

Eqn. 1.3-12

and

$$\lim_{\mathbf{n}\to\infty} \mathbf{n}\, l(\mathbf{n}) = \infty$$

Eqn. 1.3-13

Parzen further proved in [Parzen] that the estimate of the density function is consistent in quadratic mean in the sense that the expected error goes to zero with the number of training samples going to infinity

$$E\left|f_n(\mathbf{X}) - f(\mathbf{X})\right|^2 \to 0 \quad \text{as} \quad \mathbf{n}\to\infty.$$

Eqn. 1.3-14

This definition of consistency says that the expected error gets smaller as the number of training samples gets bigger. This consistency is very important since it means that the prediction will converge towards the underlying function with more given training samples.

Murthy [Murthy] and Cacoullos [Cacoullos] relaxed and extended Parzen's results. Using Gausian distribution for the weight function, the density function can be expressed as

$$f_A(\mathbf{X}) = \frac{1}{(2\pi)^{p/2}\sigma^p}\frac{1}{m}\sum_{i=1}^{m} exp\left[\frac{(\mathbf{X}-\mathbf{X_{Ai}})^T(\mathbf{X}-\mathbf{X_{Ai}})}{2\sigma^2}\right].$$

Eqn. 1.3-15

The density function here is simply the sum of Gaussian distributions centered at each training sample. Several more distributions can be used [Specht 90].

The smoothing parameter σ has a very important influence on the approximation. The influence will be shown later. But it is stated already here that this parameter needs special attention in the further use of Gaussian Probabilistic Neural Networks.

The structure of the calculations for the Probability density function has striking similarities to a feed-forward neural network. The structure of the probabilistic approach is very parallel, like a feed-forward neural network.

A probabilistic neural network has certain differences compared to a backpropagation neural network the approach using. The biggest advantage is the fact that the probabilistic approach works with a one-step-only learning. The learning of Backpropagation Nerual Networks can be described as trial and error. This is no longer the case for the probabilistic neural network. The experience is learned not by trial but by experience others made for the neural network [Specht 90, Gupta].

1.3.2.2 Probabilistic Neural Network

Probabilistic Neural networks are frequently used to classify patterns based on learning from examples. Probabilistic neural nets base the algorithm on “The Bayes Strategy for Pattern Classification”. Different rules determine pattern statistics from the training samples. Backpropagation uses methods as previously described that are not based on statistical methods. Backpropagation needs a long time and many iterations and feedbacks until it gradually approaches the underlying function [Specht 91]. It would be desirable to approach the parameters in a one-step-only approach. “The Bayes Strategy for Pattern Classification” extracts characteristics from the training samples to come to knowledge about the underlying function.

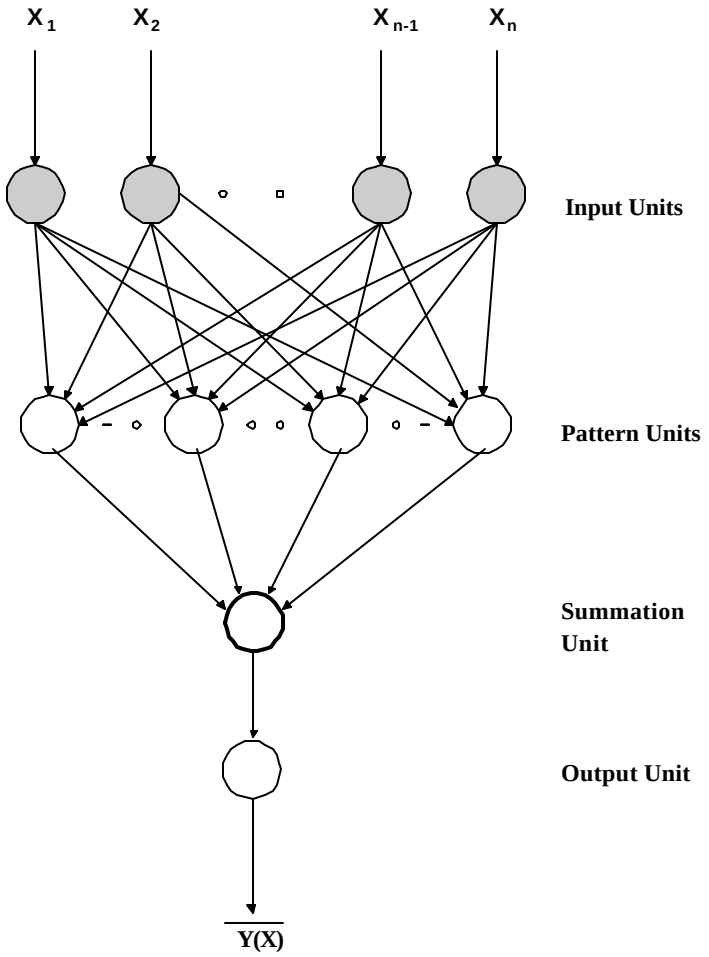

Fig. 1.3-6 Block diagram of a Probabilistic Neural Network

The general structure of a probabilistic neural network (Fig. 1.3-6) is the same as discussed in Section 1.3.1.1. A Probabilistic Neural Network consists of one input layer, and two hidden layers. The first hidden layer contains the pattern units. The pattern units contain the important functional form that was previously discussed. The calculations in the pattern unit are shown in (Fig. 1.3-7). Each pattern unit represents information on one training sample. Each pattern unit calculates the probability on how well the input vector fits into the pattern unit. In the second hidden layer there is only one summation unit. Here it is decided upon the

individual results of each pattern unit in which pattern the input vector finally belongs. The output unit performs again a calculation to give the output a physical meaning. For a probabilistic neural network it is not always possible to have multiple outputs. A futher big difference that exists between a backpropagation neural network and a probabilistic neural network is the difference in the process inside the neurons. A probabilistic neural network uses functions that are based on knowledge resulting from “The Bayes Strategy for Pattern Classification”. The strength of a probabilistic neural network therefore is not the selection of weights that fit the data in the best way. The strength of a probabilistic neural network lies in the function that is used inside the neuron.

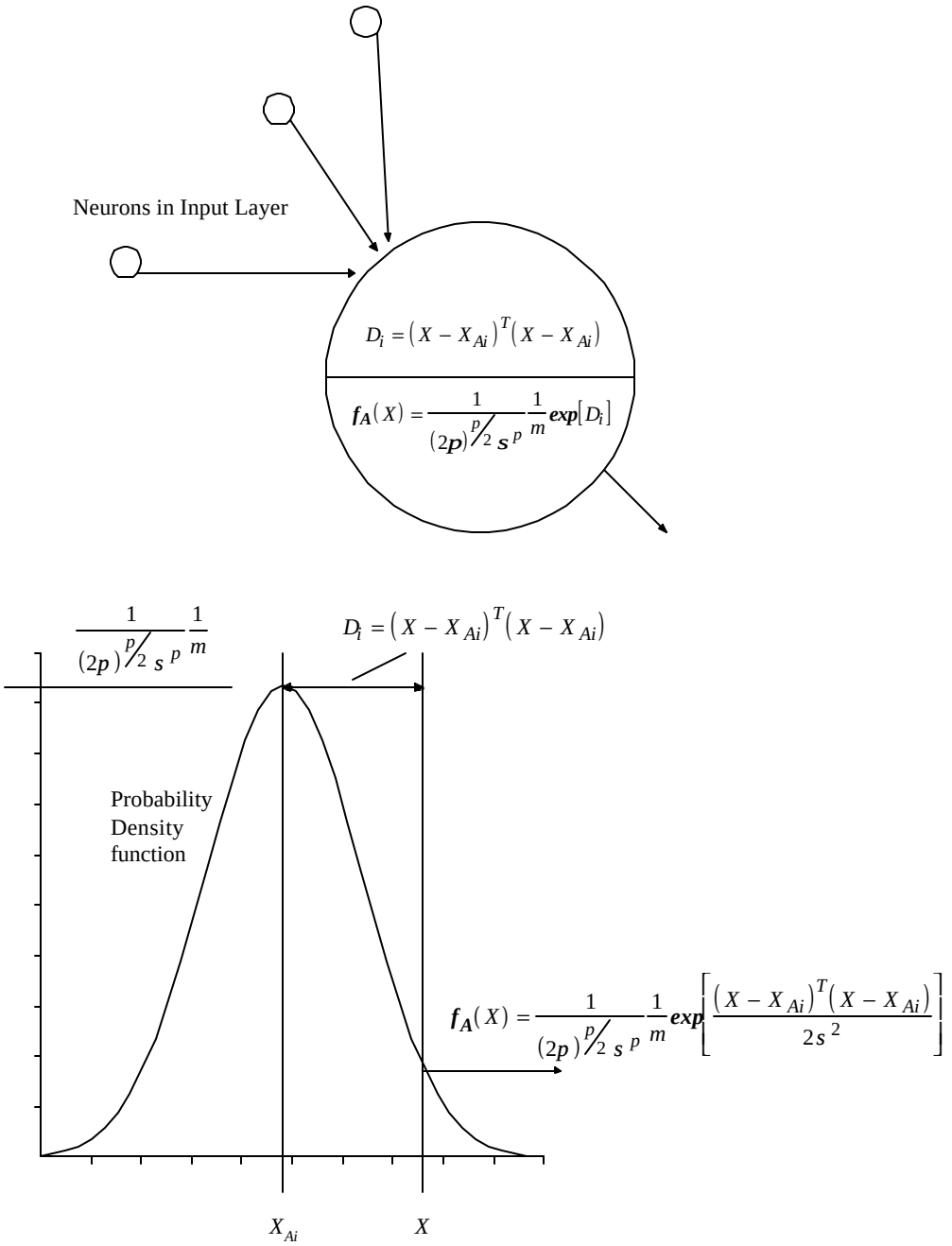

Fig. 1.3-7 Process in a Pattern Unit

The function used in a neuron of a pattern unit is a probability density function. The probability density function needs the distance between the sample point and the position

where the prediction should take place to calculate the output (Fig. 1.3-8). The output of each pattern unit is summed up and in the summation unit and then converted to a result with physical meaning.

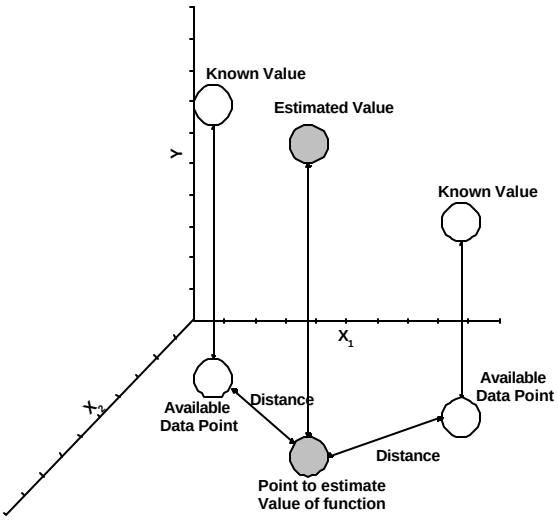

Fig. 1.3-8 Distance between the training sample and the point of prediction

"The Bayes Strategy for Pattern Classification" is valid as well for continuous results [Parzen62]. It is therefore possible to predict outputs that are continuous.

1.3.3 Why Probabilistic Neural Networks and not Backpropagation Neural Networks?

As Cherkassky, Friedman and Wechsler wrote in [Cherkassky94] there is tension between neural network researchers and statisticians because they have different backgrounds and different objectives in designing algorithms or analytical methods. Statistical methods tend

to put more emphasis on the structure of the data. For Neural network methods the structure of the data is secondary. Therefore the amount of data needed for statistical methods is a lot smaller than the amount of data needed for artificial neural network approaches.

Most methods are asymptotically good [Cherkassky94] but most of them have severe drawbacks as well. Backpropagation neural networks need a very large number of training samples and need a lot of time to gradually approach good values of the weights. Adding of new information requires retraining and this is computationally very expensive for Backpropagation Neural Nets but not for Probabilistic Neural Nets. Probabilistic Neural Nets have the big advantage that the prediction algorithm works with only few training samples. The other very big advantage is that they are very flexible and new informations can be added immediately with almost no retraining.

1.4 Chapter Summary

Different approaches to model a system with data available have been shown. Each one of them has its very own qualities and therefore advantages.

The parameters in curve fitting can have physical meaning if a physical model is available and can be used for extrapolation. On the other hand if no physical relation is being used the function chosen to use for the fitting needs some more attention, especially if the problem is multi-dimensional.

Ordinary Splines and B-Splines are very flexible tools to perform curve fitting. Problems that occur can be severe. Ordinary spline functions deviate from the expected very easily as soon as the available data is not without error and would need interpolation. B-Splines do not have this kind of problem but a minimization problem has to be performed instead.

Probabilistic Neural Networks do not need very much additional input but the results are not interpretable. Probabilistic Neural Nets have a very simple structure and are therefore very stable procedures. Probabilistic Neural Networks perform very well for only few available training samples, the quality increases as the number of training samples increases.