A generalized view on learning in feedforward neural networks

Georg Dorffner
Austrian Research Institute for Artificial Intelligence
Neural Network Group
and
Dept. of Medical Cybernetics and Artificial Intelligence
University of Vienna

Abstract

In this paper we introduce a generalized view on feedforward neural networks. In this view, well-known network types like multilayer perceptrons and radial basis function networks are just a few of many possibilities in a virtual space of neural network types, spanned by the three dimensions *propagation rule*, *transfer function*, and *learning rule*. We list several examples of other combinations of values along these dimensions and discuss the advantages of such a view. The goal of depicting neural networks this way is to arrive at strategies to find optimal neural network solutions for given data sets, aided by statistical data analysis to identify the best method.

1 Introduction

Feedforward neural networks are being widely applied to problems in the domain of classification, pattern recognition, approximation, forecasting, and control (see, for instance, [Maren et al. 90, Dagli et al. 92] for overviews). They are characterized by an architecture consisting of multiple layers of units, aligned in a cascade from input to output, with full or partial connections in one direction (input toward output) between adjacent layers. Among the most prominent models with this architecture are multilayer perceptrons (MLP; [Rumelhart et al. 86]), radial basis function networks (RBFN; [Broomhead & Lowe 88]), and competitive learning models employing "winner-take-all"

([Grossberg 76, Kohonen 84, Rumelhart & Zipser 85]).[1] For each of those models, a large number of extensions and variations exist – such as shortcut connections between non-adjacent layers in MLPs, variations in backpropagation learning (usually seen as the learning method of choice for MLPs), variations in the basis function for RBFNs, variations in the initialization scheme for RBFNs (usually seen as the weight setting method of choice for those networks), etc. Which network is applied depends either on rough guesses as to which might be more appropriate or an experimental comparison of several types used side by side (e.g. [Rosenberg et al. 93, Dorffner & Porenta 94]). Hardly ever, MLPs and RBFNs are considered as variations of perhaps a more general underlying network type (for exceptions in literature, see below). In this paper, we want to suggest that appropriate and optimal application of neural networks for given data sets can only come from adopting a more generalized view, which opens up a large number of variations (much larger than two or three), and permits a more dedicated use of network solutions. Before we introduce and discuss such a view, we briefly discuss an important motivation behind this proposal and previous attempts at similar endeavors.

2 The relationship between neural networks and statistics

Recently, more and more arguments and discussions around the relationship between neural networks and statistical methods for classification, approximation, regression, or optimization can be heard. Arguments range from "Neural networks are statistics, but often done badly" ([Sarle 94]), "Neural networks are statistics for amateurs" ([Ripley 92]), and "Neural network researchers constantly reinvent the wheel known to statisticians for decades", one one hand, to "Neural networks are asympotically as powerful as other non-linear statistical techniques, but have a variety of advantages" ([Hutton 92])[2] on the other. At the bottom line there is no doubt that a serious communication between neural network researchers and statisticians must take place. For much too long, neural networks have been applied rather "blindly" by practitioners who have not been aware about the underlying mathematical properties and the applicabilites of certain network types with respect to a given data distribution. Theorems about MLPs and RBFNs as "universal function approximators" ([Hornik et al. 89, Kurkova 92, Leonard et al. 92] have given the

[1] Another common depiction of competitive learning architectures is one with intra-layer connections in the competitive layer – see, for instance, [Dorffner 91]. With those, the networks would not belong to the feedforward class as characterized above. However, common implementations of "winner-take-all" mechanisms do not require cross-connections but assume that the mechanism is applied globally to the whole layer

[2] All these quotes have been paraphrased here and must be taken just by their content.

illusion to many that those methods are something like a "universal solution approach," where one needs no or only little knowledge about the data the network is applied to.

The way we like to put it is the following: Feedforward neural networks, as characterized above, *are* a modern form of non-linear statistics. There is no use in depicting neural networks and statistics as different and thus competitors. As [Sarle 94] has pointed out, many feedforward neural network models can be equated with or directly compared to more classical methods like linear regression, logistic regression, nearest neighbor classification, etc. He also correctly points out (see the quote above) that in spite of this identity or similarity, neural network researchers have largely not made the same careful assumptions about probability distributions of the data, the limits of the method, and other important factors, as statisticians have been used to for decades. We would, however, still claim (along with [Hutton 92]) that neural networks possess a number of advantages that give them flexibility in applications, many of which are not of the "mathematical equivalence of power" type (see our summary section below).

As a result, neural networks must be analysed in the same way classical statistical algorithms are analysed and compared (see, for instance, [Duda & Hart 73]). The view we present in this paper is designed as an important step toward identifying

- which network type can be applied to what kind of data,
- which network type does correspond to an algorithm traditionally known from statistics (in orer to make use of research results in statistical literature),
- what do known neural network types have in common, and what separates them,
- and what are the range and limits of the applicability of each network type.

The suggested view rests on the observation that MLPs and RBFNs have a lot more in common than usually admitted by their depiction in literature. Thus, in the next section, we briefly overview previous attempts of comparing these two architectures.

3 Previous attempts at a more general view

An extension to the prototype networks MLP and RBFN can be achieved by combinations of layers with different units, or by combining units of different

type in one layer. For instance, the viablity of two RBF layers in cascade (replacing the linear associator between hidden and output layer of an RBFN with another Euclidean/Gaussian layer) has been shown ([Robinson et al. 88, Dorffner 92]). Also the combination of MLP and RBF units in one hidden layer can lead to improved results ([Weymaere & Martens 91]). This latter extension is a way of making MLPs and RBFNs complement each other with respect to decision regions, as suggested above. A learning (or "optimization") procedure is introduced that incrementally extends the hidden layer by further MLP or RBF unit candidates until optimum performance with minimum network complexity (defined as number of degrees-of-freedom) is achieved. This procedure can include both bounded and unbounded decision regions and thus be more flexible with regard to input distributions.

Several people have pointed out prinicpal equivalences of either the propagation rules or the transfer functions of MLPs and RBFNs. [Maruyama et al. 92] and [Denoeux & Lengelle 93] have shown that by writing the Euclidean distance as

$$\sum_{i=1}^{n}(x_i - w_{ij})^2 = ||\vec{x} - \vec{w}_j||^2 = ||\vec{x}||^2 - 2\vec{x}\vec{w}_j + ||\vec{w}_j||^2 \tag{1}$$

and by assuming normalized input ($||\vec{x}|| = 1$) the propagation rule of the RBFN (Euclidean distance) can in principle be mapped onto the propagation rule of an MLP unit (dot product $\vec{x}\vec{w}$ plus a bias, in this case $||\vec{w}_j||^2 + 1$), with the exception of its sign. In [Maruyama et al. 92] it is shown that any MLP unit can implement RBF units this way (in that for any RBFN an MLP with the same number of units can be found that computes the same function), while RBF units can implement MLP units only under certain conditions, mainly due to the missing extra bias (threshold) parameter, for which a dummy extra input has to be introduced. Also in [Maruyama et al. 92], as well as in [Geva & Sitte 92], ways of approximating Gaussians with sigmoids and vice versa are shown. They are based on the observation that the bell shape of a Gaussian (or better still, the positive half of it) resembles an inverted sigmoid (we will come back to that aspect below).

An interesting way of viewing the propagation rules of MLP and RBFN and their decision regions in the same light is pointed out in [Omohundro 89]. He shows that by introducing an additional dimension to the n-dimensional input space, and setting this $n+1$-st unit identical to the sum of the squares of the other n inputs, the resulting MLP units behave like RBF units with localized receptive fields (i.e. decision regions) with respect to the original n dimensions. Thus, an infinite decision region can be mapped to a bounded one.

4 A virtual space of feedforward networks

In this section we introduce the more general view on feedforward neural networks hinted upon above. Based on the work discussed in the previous section the basic idea is that the main dimensions distinguishing between different feedforward network models are *propagation rule* (i.e. the input or "net input" [Rumelhart et al. 86] to a unit), the *transfer function*, and the *learning rule*. These three dimensions can be seen as spanning a virtual 3D space of possible network types, as depicted in fig. 1. If, for the moment, three layer networks (i.e. one input, one hidden, and one output layer) are considered, each network actually consists of a combination of points in two such virtual spaces – one for the input-to-hidden connections, and one for the hidden-to-output connections. A common multilayer perceptron, for instance, corresponds roughly to the two points:

- *(prop. rule = dot product (weighted sum),*
  transfer function = sigmoid,
  learning rule = gradient descent on error in weight space)
  in space I (input-to-hidden)

- *(prop. rule = dot product (weighted sum),*
  transfer function = sigmoid or linear,
  learning rule = gradient descent on error in weight space)
  in space II (hidden-to-output)

A simple type of radial basis function network ([Broomhead & Lowe 88]) roughly corresponds to

- *(prop. rule = Euclidean distance,*
  transfer function = Gaussian,
  learning rule = initialization (weight vector := training sample))
  in space I (input-to-hidden)

- *(prop. rule = dot product (weighted sum),*
  transfer function = linear,
  learning rule = gradient descent on error in weight space (delta rule))
  in space II (hidden-to-output)

We do not want to give the impression that this is all there is to MLPs and RBFNs. Nor do we want to depict this as a formal specification of these network types (see [Dorffner et al. 94a] for general problems with formal neural network specifications). We do, however, want to suggest that these dimensions are crucial aspects in distinguishing the types, in identifying their properties and limits in dealing with certain data distributions, and – above

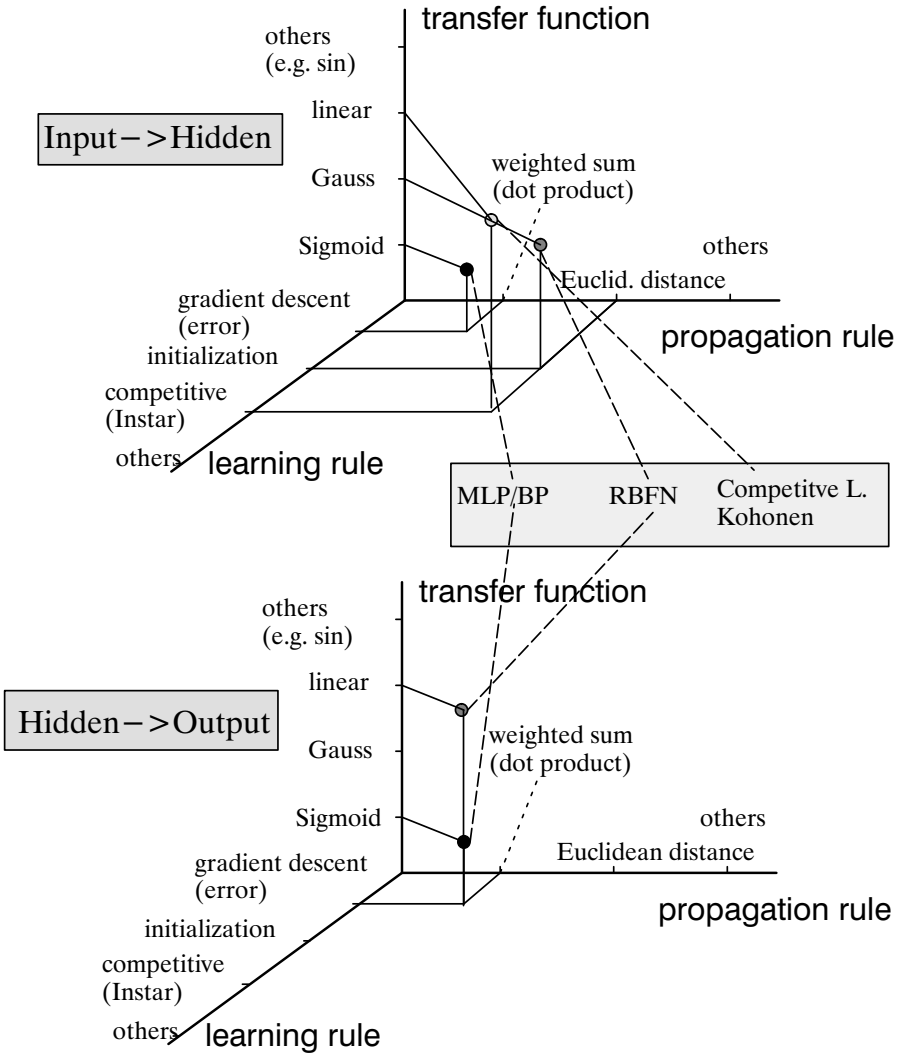


Figure 1: A virtual space of feedforward neural networks, spanned by the dimensions propagation rule, transfer function, and learning rule, for a three-layer network (one hidden layer).

all – in pointing out a variety of other, different, types. The latter is the main topic of this paper.

Therefore, in the next few subsections we discuss other points in the virtual space of feedforward networks. Some of these are known from statistical literature, some have been published before in neural network literature but have gone virtually unnoticed by practitioners, and some appear to be novel.

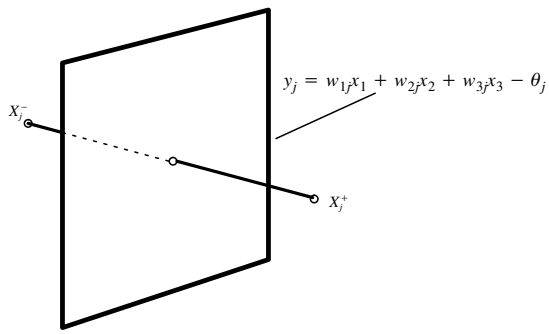
4.1 Alternative weight adaptation in MLPs and RBFNs

An obvious variation of the two "classical" types MLP and RBFN is to exchange their "learning" method in space I.[3] This leads to an MLP the input-to-hidden weights of which are initialized based on training samples, and an RBFN that is fully trained with a gradient descent procedure. The latter has, among others, been suggested by [Robinson et al. 88, Dorffner 92] but seems to be feasible only as a "fine-tuning" method, since the sensible areas of randomly initialized radial hidden units tend to lie too far from the input data (note that using Euclidean distance and Gaussian decision regions are bounded and confined to rather small protions of the input space). The former has been introduced several times independently in neural network literature ([Smyth 92, Weymaere 93]) but has also been known for years in statistics (e.g. [Duda & Hart 73, Bock 74]). The idea is a direct analogy to simple RBFNs (initializing the center of the decision regions by setting it identical to a training sample), namely initializing a hyperplane (the decision boundary of an MLP unit) such as to be a Voronoi tesselation of two training samples, one being a positive and one being a negative exemplar of a given class. This, together with the initialization equations, is depicted in fig. 2. In other words, the hyperplane is set such as to be on the midpoint of the line connecting the two samples, and orthonormal to it. Best results are achieved when cluster centers (from a prior cluster analysis) are taken as exemplars.

It must be noted that this method is valid only for classification problems (as opposed to approximation) and does not take the data distribution into account. However, adaptations to these cases appear straightforward. For instance, if cluster centers (after preceding cluster analysis) are taken as positive and negative samples, the hyperplane can be shifted from midpoint in proportion to the standard deviations within each cluster.

A thus initialized network can subsequently be trained by regular backpropagation ([Smyth 92]). However, this often does not lead to any improvements in performance or training speed, mainly because it can be shown that backpropagation does not always tend to follow the so-called "hyperplane assumption" ([Pratt & Christensen 94]), i.e. the assumption that hyperplanes

[3] I put 'learning' in quotes here, since initialization of input-to-hidden weights can hardly be called such.



$$w_{ij} = x_{ji}^+ - x_{ji}^-$$

$$\theta_j = \frac{1}{2}\sum_{i=1}^{n}\left(x_{ji}^{+\,2} - x_{ji}^{-\,2}\right)$$

$$y_j = \sum_{i=1}^{n} w_{ij}x_i - \theta_j$$

Figure 2: Initialization of MLP units as a Voronoi tesselation of a positive and a negative class exemplar.

are moved in input space such as to actually separate positive from negative samples. This, by the way, points to some possible general problems of backpropagation for classification problems. An alternative would be – in another direct analogy to the RBFN – to train hidden-to-output weights by a simple delta rule, while leaving input-to-hidden weights fixed. Several practical examples (e.g. [Dorffner & Porenta 94, Dorffner et al. 94b]) show that this can lead to equal performance as backpropagation (in some cases even better) but with much smaller learning times. In addition, such a procedure (initialization plus subsequent delta rule) is not prone to be stuck in local minima (as pointed out by [Broomhead & Lowe 88]) and involves much fewer degrees of freedom (the weights) during learning. The latter is very important for applications with a rather limited number of training samples, in order to not under-determine the estimation problem. (A problem is under-determined if it contains more degrees of freedom than training samples. This is an aspect that is very perspicuous in statistical literature, but has often been overlooked by neural network practitioners. Consider, for instance, the often-heard argument "If in doubt take more input variables than fewer, since network learning will find the relevant ones anyway").

An example from [Dorffner et al. 94b] is illustrated in fig. 3. In the application of neural networks in controling so-called rotary blood pumps, the task was to predict the state of "danger of suction" in the ventricle, based on thirteen physiological measurements. A simpel prinicpal component analysis was applied to the thirteen-dimensional data, revealing the fact that the first three components covered about 85 % of the variance. These three components

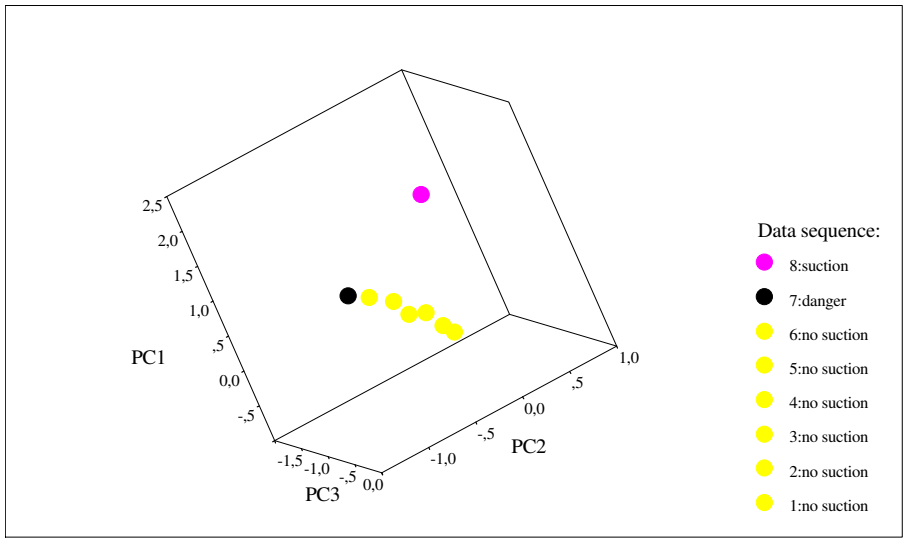


Figure 3: A single data sequence in a control application (rotary blood pumps) depicting states going from no suction to suction. Visualization was made possible through principal component analysis.

were used to visualize the data, as depicted for a particular time series of system states going from "no danger" via "danger of suction" (black dot in fig. 3) to "suction". This visualization permitted easy initialization of hyperplanes by using the "danger" point and the point one time step before as positive and negative class exemplars. Again, only hidden-to-output weights were trained (by a delta rule), leading to equal performance than a tedious backpropagation procedure.

4.2 Hyperplane fitting with feedforward networks

Another possible point in virtual network space is this:

- *(prop. rule = dot product (weighted sum),*
  transfer function = Gaussian,
  learning rule = competitive (winner-take-all + anti-Hebbian))
  for space I (input-to-hidden)

This combination appears rather absurd at first sight. The combination of the dot product with the Gaussian function leads to "ridges" as decision regions, which are unbounded in one direction, and bounded in others. In [Dawson & Schopflocher 92] this combination was introduced, together with a gradient descent learning rule, but without any real motivation of why it should be used and why it can perform better than the original MLP.

Here is a quick demonstration that the above network type (with a rather simplified hidden-to-output layer) implements a classification technique known in statistics.

It is well-known in statistics that classification cannot only be achieved by hyperplanes separating the classes, but also through hyperplanes *representing* the classes (or sub-classes), pretty much like in regression ([Bock 74, Mosteller & Tukey 77]). A hyperplane representing a sub-class of data points is defined as the hyperplane for which the sum of the quadratic orthogonal distances of all points is at a minimum. In the two-dimensional case this is identical to the regression line approximating that sub-class. Classification is then achieved by (a) assigning a class label to each hyperplane, and (b) for each data point choosing the closest hyperplane. This is in contrast to other ways of classification such as variations of nearest-neighbor or Voronoi tesselations, where a single point represents such sub-classes. On the other hand, the case of representing hyperplanes could be considered a more general case, if one permits hyperplanes of lesser dimensionality, i.e. defined through fewer degrees of freedom (e.g. lines in three-dimensional space), which would include single points, as well. As is pointed out in [Bock 74], in this more general case the decision boundaries between the sub-classes (i.e., the boundaries where two hyperplanes are equally far in terms of orthogonal distance) are not necessarily hyperplanes again, but more general quadratic surfaces such as cones. Thus, this method of classification is not directly equivalent to classification through separating hyperplanes.

These observations led us to define representing hyperplanes in a neural network unit to construct such a classifier from a layered network. The difficult problem in the above approach is to automatically find a sub-classification into clusters of data points and their appropriate representing hyperplanes. Viewing the method in the context of neural networks, especially through applying a gradient descent rule similar to backpropagation ([Werbos 74, Rumelhart et al. 86]), finding hyperplanes can be done through a learning algorithm. The crucial observation is that the net input y of a multilayer perceptron unit (i.e., the result of the propagation rule, which is the weighted sum of input values) is proportional to the orthogonal distance of the data point to the hyperplane defined by the weights and the bias of that unit. Thus, choosing the closest hyperplane can be done by choosing the unit with smallest absolute value of the net input, i.e., the smallest weighted sum. This somewhat reverses the usual way of picking a "winner" in a neural network layer, which is based on looking for the largest dotted product of the weight and input vectors ([Grossberg 76]) or the smallest Euclidean distance ([Kohonen 84]). To turn this into a "real" winner-take-all procedure, a Gaussian-type of transfer function can be chosen which assign an activation of 1 to a net input of 0, and asymptotically goes to 0 for increasing absolute values of the net input.

Learning can then be done in a way very similar to learned vector quantization (or LVQ; [Kohonen 84]). The hidden layer of units comprising the hyperplanes is divided into as many clusters as there are classes to be recognized, and each of them is assigned to one of the classes. For each point in the training set the cluster corresponding to its class is considered, and winner-take-all is performed in the above manner. Then a learning rule (to be described below) is applied to the weights which turns and shifts the corresponding hyperplane such as to minimize the summed squared distance of all points leading to the same winner. For this to work, learning must proceed in the "batch" mode, i.e., weight increments are summed up and added to the weights after an entire epoch. Otherwise, minimization would always be done with respect to a single point, ignoring the structure of the sub-class to be fitted. After learning, novel input points are presented to the network and winner-take-all is performed on the whole hidden layer. The input is then assigned the class attached to the cluster the winner lies in. From this we can see that an output layer degenerates to fusing the activations of all units of a cluster, i.e. by introducing one unit for each class with positive weights to its corresponding cluster. Of course, winner-take-all can also be omitted leading to distributed responses. In this case, hidden-to-output weights can, again, be trained by a simple delta rule.

The update and learning rules An $n-1$-dimensional hyperplane H_j, defined by a unit j (using the weighted sum as propagation rule, as in the multilayer perceptron) is defined in n-dimensional space through the equation

$$\sum_{i=1}^{n} w_{ij} x_i + c_j = 0 \tag{2}$$

where c_j is a constant and the vector $(w_{ij}), i = 1..n$ is orthognal to the hyperplane. It can be shown easily that the net input y of this unit

$$y_j = \sum_{i=1}^{n} w_{ij} x_i - \theta_j \tag{3}$$

with bias $\theta_j = -c_j$ is proportional to the orthogonal distance $d(H_j, \vec{x})$ of any point $\vec{x}$ to the hyperplane. To choose a hyperplane H_j to represent (or characterize – [Bock 74]) a sub-class $A_{kl} \subseteq C_k$ (where C_k is one of m classes of the classification problem) one picks the one with the following property

$$\sum_{\{i|\vec{x}_i \in A_{kl}\}} d(\vec{x}_i, H_j)^2 \rightarrow \min \tag{4}$$

i.e., one that forms a multivariate regression of the sub-class. With the help of hyperplanes representing sub-classes, classification can be done for any arbitrary point $\vec{x}_i$ by choosing the hyperplane H_j with smallest absolute value

of its orthogonal distance $d(\vec{x}_i, H_j)$ and returning the class k to which sub-class A_{kl} has been assigned to. Assuming a Gaussian transfer function for all hyperplane units, e.g.

$$x_j = e^{-\frac{y_j^2}{\sigma^2}} \tag{5}$$

with a freely chosen σ for all units, finding the closest hyperplane is identical to winner-take-all among the hyperplane units using activations x_j. To permit several sub-classes A_{kl} per class C_k, the layer of hyperplane units must be divided into m clusters.

The crucial point now is to find appropriate hyperplanes through learning. For this we assume a gradient descent rule which minimizes the squared orthogonal distances of all points I it represents, as in eq. 4. This is identical to minimizing the sum of squared net inputs for all those points. Similar to [Rumelhart et al. 86] we can thus define the weight increment for learning as

$$\begin{aligned}
\Delta w_{ij} &\propto -\frac{\partial}{\partial w_{ij}} \left(\sum_{\{I|\vec{x}_i^{(I)} \in A_{kl}\}} \sum_{i=1}^{n} w_{ij} x_i^{(I)} - \theta_j \right)^2 && (6) \\
&= -\eta \left(\sum_{\{I|\vec{x}_i^{(I)} \in C_k\}} \sum_{i=1}^{n} w_{ij} x_i^{(I)} - \theta_j \right) x_i && (7) \\
&= -\eta y_j x_i && (8)
\end{aligned}$$

with a learning rate η (subsuming the factor 2 from the derivative). What results is a kind of anti-Hebbian rule using the net input of the postsynaptic unit and the activation of the pre-synaptic one. Similarly, it can be derived that

$$\Delta \theta_j = \eta y_j \tag{9}$$

These rules must be applied to the winner l in the k-th cluster representing sub-class A_{kl}. The assumption is that each hyperplane is changed so as to form a regression of all the points that are currently closest to it. This is a crucial assumption which might lead to non-optimal results if, through unfortunate initial weights, the hyperplane attempts to fit points that should rather be fitted by a different plane. This appears to be akin to local minima problems in regular backpropagation. In any case, learning must be done in the "batch" mode adapting weights only after computing increments for an entire epoch. Since the size of the weight vector influences the proportional dependency between orthogonal distances and the net input, normalization of the weight vectors after each epoch is advisable.

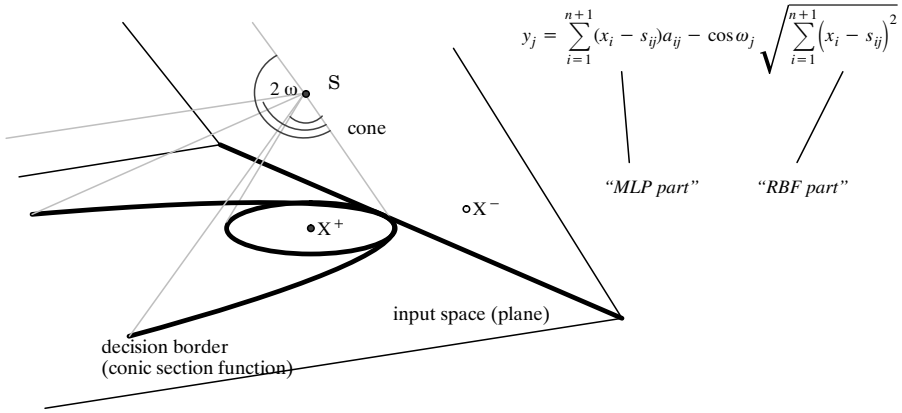


Figure 4: A conic section function network. A cone intersects the input space by different functions dependening on the opening angle ω, with hyperplane (MLP) and hypersphere (RBFN) as extremes.

4.3 Conic section function networks (CSFN)

In another paper ([Dorffner 94]), we have demonstrated that the virtual space given above can even be continuous along parts of its axes. In particular, we introduced a network type that realizes a continuum between the propagation rules and thus decision borders of MLPs (hyperplanes) and simple RBFNs (hyperspheres). The idea is based on the observation that the two known decision borders are special cases of general (hyper-) conic section functions. Fig. 4 depicts a two-dimensional input space with positive and negative training samples, and a cone in three-dimensional space, the opening angle ω of which can be changed such as to intersect the input plane by a line, a circle, or a conic section function in between. The propagation rule (rather complicated on first sight) is also depicted, consisting of an "MLP part" (a weighted sum with weight parameters a_{ij}) and an "RBF part" (a Euclidean distance with separate parameters s_{ij}). The suggested strategy for using this network type is to (a) initialize input-to-hidden weights as an MLP, like above; (b) adapt one parameter per hidden unit (the opening angle) to adjust decision borders; and (c) train hidden-to-output weights by a simple delta rule. For more details and examples, see [Dorffner 94, Dorffner & Porenta 94].

This network type fulfills several criteria considered important in statistics. The first could be dubbed "localization principle" (e.g. [Omohundro 89]), i.e. the aspect that each sub-space in data space should be covered only by one or a few "basis functions" (one or a few hidden units). This permits easier adaptation and transfer of solutions. Secondly, the CSFN implements

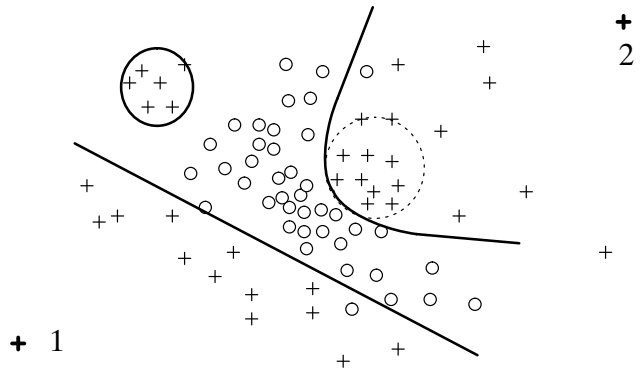


Figure 5: Different types of decision regions in a data space and their ability to extrapolate. Hyperplanes tend to extrapolate to novel points (such as 1) without being able to give an estimation of the error if that point is not close to any training point. CSFNs, by using their RBFN part (hypersphere) can extrapolate (e.g. to point 2), while estimating the possible error of this extrapolation (distance to the hypersphere).

"marginalization" as would be required by classification techniques applied to unknown data ([Roberts 94]): One of the disadvantages of MLPs is that they cannot distinguish between a correct output and a classification that is highly likely to be incorrect since the data point is far away from any seen training sample (see fig. 5). In other words, MLPs tend to extrapolate to infinite portions of data space and do not possess the "competence" to know about their own error probability. RBFNs, on the other hand, do not extrapolate as easily, since their unit decision regions are bounded. But it also means that they hardly ever extrapolate at all. A CSFN can do both. It is able to extrapolate, but since a bounded region is contained in its propagation rule (the "RBF part" of the rule) that part can be used to estimate the likely error, by computing the distance to that part.

5 A systematic strategy for choosing neural network solutions

The view on feedforward neural networks given above can be considered a first step toward more dedicated and pin-pointed applications of neural network to given data. Although many combinations (points in virtual network space) are still unexplored, the examples above already shed light on how each network type behaves and how different types compare to each other.

5.1 Propagation rules

The propagation rule mainly determines the decision boundary of single hidden units. We have seen that most common network types use one special case of conic section function, which can vary between bounded and unbounded decision regions. We have demonstrated that the continuum between extreme cases can be exploited as well. Bounded regions appear appropriate when data is confined to subspaces of the possible space, and when error estimation for outliers is crucial. Unbounded regions, besides the fact that they correspond to linear propagation rules, seem appropriate when at least some kind of extrapolation onto subspaces outside the training set is warranted, or when large (e.g. approximately hyperbolic) data regions exist, where approximation with small bounded regions would be too demanding on unit resources.

5.2 Transfer functions

In literature, much emphasis has been put on transfer functions, almost overshadowing the role of propagation rules. For instance, talk about "Gaussians vs. sigmoids" can be found frequently, meaning the comparison between RBFNs and MLPs. The above view suggests a somewhat reduced role of transfer functions. For many problems the exact shape of the function is not so crucial (compare [Hornik et al. 89], where the universal approximation theorem is proved for a more general class of transfer functions). What matters for particular neural network types, are the function's

- sign, i.e. whether the function is increasing or decreasing with increasing net input
- non-linearity (e.g. for the theoretical power of MLPs)
- continuity and differentiability (e.g. for gradient descent rules)
- saturation toward minimum and maximum values
- gradient 0 at the origin (this is important to make many gradient descent rules stable)

The positive half of a Gaussian[4], for instance, resembles an inverted sigmoid in its general characteristic. Examples are [Geva & Sitte 92, Maruyama et al. 92] – where this property is exploited to show possible equivalences between MLPs and RBFNs –, or the CSFN, where one transfer function – a sigmoid – can handle both MLP and RBFN parts (since the sign for the RBFN part is already inverted by the particular propagation rule). One should also not forget that it is the Euclidean (or other) distance that makes an RBFN "radial," and not the Gaussian.

[4]For RBFNs only the positive half is relevant, since distances cannot be negative.

5.3 Learning rules

We have seen two major classes of learning rules: One that directly initializes weights based on training samples, and one that fine-tunes solutions by gradient descent over some criterion (summed squared error of network, or summed squared distance from a regression hyperplane). A more unified view on different learning rules, including Hebbian rules and instar/outstar rules, would still have to derived. (We only saw how an anti-Hebbian rule can arise from a kind of gradient descent rule.) It is clear, however, that for reasons of theoretical learnability (compare [Baum & Haussler 89, Maass 94]) initialization methods should be an integral part of setting weights, wherever possible.

5.4 Heuristics for network design

In summary, we can easily see that these analyses suggest a data-specific design of network architectures. The design should always be preceded by a careful data analysis revealing the basic structure and distribution of the data. If that is not possible (e.g. due to their high dimensionality), distributions and characteristics will have to be estimated. We see that this points toward one assumption generally held in neural network literatur as being partly false. Even though many neural network methods can be seen as non-parametric (compare [Sarle 94]) they *can or should not be applied without prior assumptions about the data* (e.g. whether bounded or unbounded decision regions appear more appropriate, whether the data can be clustered, etc.). The important aspect of most neural network applications is *generalization* to novel data, and to optimize that goal such assumptions are crucial. In the above examples we have hinted upon a few tools to aid in determining the underlying data characteristics:

- cluster analysis to determine whether training data can be grouped into distinct clusters, which can be exploited in initialization and in devising decision regions,
- principal component analysis or related techniques to reduce the dimensionality of the data in order to analyse their complexity,
- projection pursuit to determine the ability of classes to be regressed by hyperplanes.

Much more research remains to be done in order to extend this list and to arrive at truly helpful heuristics for network design.

6 Summary and conclusion

In this paper we have argued for more data-specific applications of feedforward neural networks, instead of "blind" uses of one or two common techniques like multilayer perceptrons or radial basis function networks. We have presented a general view of feedforward neural networks, depicted in a virtual space of neural networks spanned by the dimensions *propagation rule*, *transfer function*, and *learning rule*. With a few examples we have demonstrated that this space contains many more network types than are usually known or accessible to the practitioner. Finally we have argued that this view is a first important step toward a more pinpointed use of neural networks, and have given a few first heuristics as to how and when to best apply which network type.

The discussion in this paper has touched upon the important issue of the relationship between statistics and neural networks in at least two ways:

- It has pointed out important parallels or equivalences between classical statistical methods and neural networks.
- It has pointed out the importance of taking assumptions about data and the limit of the methods into account, as it is prevalent in traditional statistics.

Both ways suggest that neural network researchers should get acquainted with traditional statistics much more than they have in the past. A cooperation between the two fields, instead of a heated discussion, can help guarantee optimum engineering results in the future. If, as we suggested above, neural networks are but non-linear statistics, one might still ask what the use of neural networks as such will actually be. For this question we can give the following arguments speaking in favor of realizing a statistical method (traditional or novel) as a neurally inspired network architecture:

- The architecture gives the method a great flexibility in devising pinpointed solutions, in fusing different methods, and in making the results somewhat accessible. (Despite the fact that neural networks generally do not contain easy "explanation capabilities," they provide the practitioner with an important model and metaphor to reason about the underlying information processing – much more than is possible with algorithms represented as closed formulae or software packages.)
- Also due to their network architecture, insertion of rule-based and other knowledge is easy and straight-forward to achieve (compare, e.g., [Towell et al. 90]). This is again very important in the context of learnability and the complexity of learning problems.

- Neural networks possess the capability to remain *adaptive* in on-line use, since their learning methods are mostly incremental. This property, although obvious, is not trivial to achieve and has only rarely been investigated, but is nevertheless a major potential of neural networks.

So, as much as neural network researchers can learn form statistics, statisticians can gain much from viewing some of their algorithms as neural networks.

Acknowledgments

I particularly thank the organizers of the Cowan workshop in Cottbus, Germany, where this paper was presented, for their invitation, inspiration, and patience in waiting for the manuscript. I further thank numerous listeners to further talks I have given on this topic for their valuable feedback. I also thank Prof. Robert Trappl for his continuing support for neural network research at the Austrian Research Institute for Artificial Intelligence. The Austrian Research Institute is supported by the Austrian Federal Ministry of Science and Research.

References

[Baum & Haussler 89] Baum E.B., Haussler D.: What Size Net Gives Valid Generalization?, in Touretzky D.(ed.), *Advances in Neural Information Processing Systems*, Morgan Kaufmann, Los Altos, CA, pp.81-90, 1989.

[Bock 74] Bock H.H.: *Automatische Klassifikation*, Vandenhoeck & Ruprecht, Goettingen, 1974.

[Broomhead & Lowe 88] Broomhead D.S., Lowe D.: Multivariable Functional Interpolation and Adaptive Networks, *Complex Systems*, 2,321-355, 1988.

[Dagli et al. 92] Dagli C.H., Burke L.I., Shin Y.C.(eds.): *Intelligent Engineering Systems through Artificial Neural Networks*, Volume 2, ASME Press, New York, 1992.

[Dawson & Schopflocher 92] Dawson M.R.W., Schopflocher D.P.: Modifying the Generalized Delta Rule to Train Networks of Non-monotonic Processors for Pattern Classification, *Connection Science*, 4(1)19-32, 1992.

[Denoeux & Lengelle 93] Denoeux T., Lengelle R.: Initializing Backpropagation Networks with Prototypes, *Neural Networks*, 6(3)pp.351-364, 1993.

[Dorffner 91] Dorffner G.: *Konnektionismus*, Teubner, Stuttgart, 1991.

[Dorffner 92] Dorffner G.: EuclidNet - A Multilayer Neural Network using the Euclidian Distance as Propagation Rule, in Aleksander I. & Taylor J.(eds.), *Artificial Neural Networks*, 2, North-Holland, Amsterdam, pp.1633-1636, 1992.

[Dorffner 94] Dorffner G.: A Unified Framework for of MLPs and RBFNs: Introducing Conic Section Function Networks, *Cybernetics and Systems*, 25(4), 1994.

[Dorffner & Porenta 94] Dorffner G., Porenta G.: On Using Feedforward Neural Networks for Clinical Diagnostic Tasks, *Artificial Intelligence in Medicine*, 6(5), special issue on Neurocomputing in Medicine, 1994.

[Dorffner et al. 94a] Dorffner G., Wiklicky H., Prem E.: Formal Neural Network Specification and its Implications on Standardization, *Computer Standards & Interfaces*, 16, spec. issue on Artificial Neural Network Standards, pp.205-219, 1994.

[Dorffner et al. 94b] Dorffner G., Stoecklmayer C., Schmidt C., Schima H.: Synergies between Statistical Data Analysis and Neural Networks in the Control of Rotary Blood Pumps, Austrian Research Institute for Artificial Intelligence, Vienna, TR-94-22, 1994.

[Duda & Hart 73] Duda R.O., Hart P.E.: *Pattern Classification and Scene Analysis*, John Wiley & Sons, N.Y., 1973.

[Geva & Sitte 92] Geva S., Sitte J.: A Constructive Method for Multivariate Function Approximation by Multilayer Perceptrons, *IEEE Transactions on Neural Networks*, 3(4)621-624, 1992.

[Grossberg 76] Grossberg S.: Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors, *Biological Cybernetics*, 21, 145-159, 1976.

[Hornik et al. 89] Hornik K., Stinchcombe M., White H.: Multi-layer Feedforward Networks are Universal Approximators, *Neural Networks* 2, 359-366,1989.

[Hutton 92] Hutton L.V.: Using Statistics to Assess the Performance of Neural Network Classifiers, *Johns Hopkins APL Technical Digest*, 13(2), pp.291-299, 1992.

[Kohonen 84] Kohonen T.: *Self-Organization and Associative Memory*, Springer, Berlin, 1984.

[Kurkova 92] Kurkova V.: Universal Approximation Using Feedforward Neural Networks with Gaussian Bar Units, in Neumann B.(ed.), *Proceedings of the Tenth European Conference on Artificial Intelligence* (ECAI92), Wiley, Chichester, UK, pp.193-197, 1992.

[Leonard et al. 92] Leonard J.A., Kramer M.A., Ungar L.H.: Using Radial Basis Functions to Approximate a Function and Its Error Bounds, *IEEE Transactions on Neural Networks*, 3(4)624-627, 1992.

[Maass 94] Maass W.: Perspectives of Current Research about the Complexity of Learning on Neural Nets, in Roychowdhury V.P., Siu K.Y., Orlitsky A.(eds.): *Theoretical Advances in Neural Computation and Learning.* Kluwer, (to appear), 1994.

[Maren et al. 90] Maren A.J., Harston C., Pap R.M.(eds.): *Handbook of Neural Computing Applications*, Academic Press, New York, 1990.

[Maruyama et al. 92] Maruyama M., Girosi F., Poggio T.: A Connection between GRBF and MLP, Massachusetts Institute of Technology, Cambridge, MA, AI Memo No. 1291, 1992.

[Mosteller & Tukey 77] Mosteller F., Tukey J.W.: *Data Analysis and Regression - a second course in statistics*, Addison-Wesley, Reading, MA, 1977.

[Omohundro 89] Omohundro S.: Geometric Learning Algorithms, International Computer Science Institute (ICSI), report, 1989.

[Pratt & Christensen 94] Pratt L.Y., Christensen A.N.: Relaxing the Hyperplane Assumption in the Analysis and Modification of Back-propagation Neural Networks, in Trappl R.(ed.), *Cybernetics and Systems '94*, World Scientific Publishing, Singapore, pp.1711-1718, 1994.

[Ripley 92] Ripley B.D.: Statistical Aspects of Neural Networks, Department of Statistics, University of Oxford, 1992.

[Roberts 94] Roberts S.: Statistics and Neural Networks, presentation given at a meeting on EEG processing with neural networks in Vienna, Sep. 94; Imperial College, 1994.

[Robinson et al. 88] Robinson A.J., Niranjan M., Fallside F.: Generalising the Nodes of the Error Propagation Networks, Cambridge University Engineering Department, Cambridge, UK, CUED/F-INFENG/TR 25, 1988.

[Rosenberg et al. 93] Rosenberg C., Erel J., Atlan H.: A Neural Network that Learns to Interpret Myocardial Planar Thallium Scintigrams, in Hanson S.J., et al.(eds.), *Advances in Neural Information Processing Systems* 5, Morgan Kaufmann, San Mateo, CA, pp.755-764, 1993.

[Rumelhart & Zipser 85] Rumelhart D.E., Zipser D.: Feature Discovery by Competitive Learning, *Cognitive Science*, 9(1), 75-112, 1985.

[Rumelhart et al. 86] Rumelhart D.E., Hinton G.E., Williams R.J.: Learning Internal Representations by Error Propagation, in Rumelhart D.E., McClelland J.L.(eds.), *Parallel Distributed Processing*, MIT Press, Cambridge, MA, 1986.

[Sarle 94] Sarle W.S.: Neural Networks and Statistical Models, Proceedings of the Nineteenth Annual SAS Users Group International Conference, 1994.

[Smyth 92] Smyth S.G.: Designing Multilayer Perceptrons from Nearest-Neighbor Systems, *IEEE Transactions on Neural Networks*, 3(2)329-333, 1992.

[Towell et al. 90] Towell G.G., Shavlik J.W., Noordewier M.O.: Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks, in Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), AAAI Press/MIT Press, Menlo Park, CA, pp.861-866, 1990.

[Werbos 74] Werbos P.: Beyond regression: New tools for prediction and analysis in the behavioral sciences, Harvard University, Ph.D. Dissertation, 1974.

[Weymaere & Martens 91] Weymaere N., Martens J.-P.: A Fast and Robust Learning Algorithm for Feedforward Neural Networks, *Neural Networks*, 4,361-369, 1991.

[Weymaere 93] Weymaere N.: Optimizing the structure of a multilayer perceptron, *CC-AI*, 10(1- 2) pp. 79-93, 1993.