Nonlinear data mapping by neural networks

R.P.W. Duin *Delft University of Technology, Netherlands*

Abstract

A review is given of the use of neural networks for nonlinear mapping of high dimensional data on lower dimensional structures. Both, unsupervised and supervised techniques are considered.

Keywords: Nonlinear data mapping, neural networks, supervised, unsupervised, feedforward networks, self-organising maps, vector quantization

1 Introduction to nonlinear mapping

The analysis of high-dimensional data has an increasing importance due to the growth of sensor resolution and computer memory capacity. Typical examples are images, speech signals and multi-sensor data. In the analysis of these signals they are represented in their entirety or part by part in a sample space. As an example, a single data point may represent a 16x16 (sub)image. A collection of these images constitutes a cloud of points in a 256-dimensional space.

In most multi-sensor data sets there is a large dependency between the sensors or between the sensor elements. This is certainly true for nearby image pixels. But also more fundamentally, it is not to be expected that any physical experiment contains hundreds of degrees of freedom that are of significant importance. Consequently, multi-dimensional data sets may be represented by lower dimensional descriptions. There are various reasons why such representations may be of interest. They may reveal the structure of the data or the problem, they may be used for relating individual data points to each other (e.g. finding the most similar one in a database) or they may be used for retrieving missing data values. Below this will be defined more formally.

The following representations of a set of k-dimensional data points $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{ij}, ..., x_{ik})$, $\mathbf{x}_i \in R_k$ defined by $X = \{\mathbf{x}_1,\mathbf{x}_2, ..., \mathbf{x}_i, ... \mathbf{x}_m\}$ will be discussed here:

a. A set of prototypes $Z = \{\mathbf{z}_1,\mathbf{z}_2, ..., \mathbf{z}_i, ... \mathbf{z}_n\}$ with $\mathbf{z}_i \in R_k$, i.e. a small number of data points in the same data space, $n < m$, selected or constructed in such a way that each of the original points $\mathbf{x}_i$ is nearby one of the prototypes $\mathbf{z}_i$. So Z is chosen such that:

$$J = \frac{1}{m}\sum_i \min_j \{\|\mathbf{x}_i - \mathbf{z}_j\|\} \qquad (1)$$

is minimised.The dimensionality of data space itself is not reduced. The reduction is realised by assigning new data points $\mathbf{x}$ to one out of a set of known prototypes.

$$\mathbf{y} = \arg\min_{\mathbf{z}_j} \{\|\mathbf{x} - \mathbf{z}_j\|\} \qquad (2)$$

This might be used, for example, in coding applications. Sets of prototypes are therefore also addressed as codebooks.

b. A subspace $\mathbf{y} = g(\mathbf{x},\theta)$ with $\mathbf{y} \in R_k$, $\theta \in R_{k'}$, $k' < k$, i.e. a structure having a lower dimensionality than the original data space. Each of the data points will be close to or in such a subspace. A natural criterion is the mean square error:

$$J = \frac{1}{m}\sum_i \|\mathbf{x}_i - \mathbf{y}_i\|^2 \quad (3)$$

c. A probability density function f(**x**). This gives for each point in the data space the probability density that a data point is found there. If this is done well the densities of the data points in X will be high, as measured by the maximum likelihood criterion:

$$J = \sum_i \log(f(\mathbf{x}_i)) \quad (4)$$

In each of these cases we have a criterion for optimizing a map M, a rule for mapping new data points on M and a distance $D_M(\mathbf{x})$ of that point to the map. A particular way to apply this is the completion of incomplete data. Let a data vector **x** consist out of a known part **x'** and an unknown part **y**, so **x** = **(x',y)**. For a given map M one can now find the vector **y** that minimises $D_M(\mathbf{x})$:

$$\hat{\mathbf{y}} = \mathrm{argmin}_{\mathbf{y}}(D_M(\mathbf{x})) = \mathrm{argmin}_{\mathbf{y}}(D_M(\mathbf{x}', \mathbf{y})) \quad (5)$$

It has to be realised that the above way for optimizing M (e.g. Equation 3) is not focused on minimising the error in $\hat{\mathbf{y}}$ as it uses the distance between the map and the entire vector **x**. In contrast to this *unsupervised training* of maps the technique of *supervised training* can be defined using some error measure $E(\hat{\mathbf{y}}, \mathbf{y})$ between an estimated and a desired vector **y**: Choose M such that for the set of given data vectors $\{(\mathbf{x'}_i, \mathbf{y}_i), i = 1,m\}$

$$J = \sum_i E(\mathrm{argmin}_{\mathbf{y}}(D_M(\mathbf{x}'_i, \mathbf{y}), \mathbf{y}_i)) \quad (6)$$

is minimum.

2 Neural network mapping

Nonlinear maps can be represented by neural networks. These are collections of almost identical simple operators (neurons) each contributing partially to the map, e.g.:

$$\mathbf{y} = (N_j(\mathbf{x}), j = 1, n) \quad (7)$$

in which $N_j(\mathbf{x})$ is a single neuron computing one component of **y.** Such a set of parallel neurons may be nested:

$$\mathbf{y} = (N_{j_2}(N_{j_1}(\mathbf{x}), j = 1, n_1), j = 1, n_2) \quad (8)$$

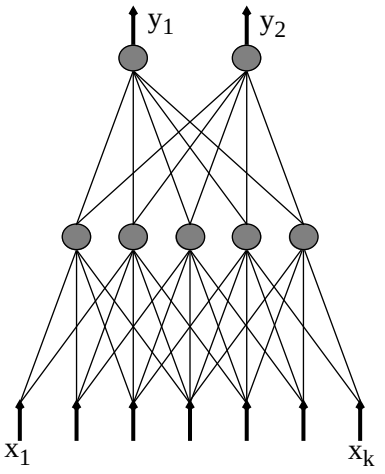

Figure 1: 2-layer neural network

In this case the original data vectors **x** are mapped on a first layer of n_1 input neurons. Their outputs are mapped on a second layer of n_2 output neurons. Thereby **y** has n_2 components. This can be represented in a graphical way like given in Figure 1. The neurons in the first (input) layer have k inputs (here k = 7). The neurons in the second (output) layer have 5 inputs. All neurons have just a single output value. So this network maps a 7-dimensional space first on a 5-dimensional space and then on 2-dimensional space.

In each neuron a different set of weights may be stored that makes their operation different from the other neurons. If a neuron has k inputs the size of this set is usually k+1: one weight for each input, constituting a k-dimensional vector **w** and an additional constant w_0, often called bias. Two very common types of neurons are the the *spatial neurons* and the *directional neurons*.

The spatial neurons can be thought to be located at a particular point in their input space. Their weights represent this location. The bias may represent a size parameter determining the size of some influence sphere. Such a neuron may thereby be defined as:

$$N(\mathbf{x}|\mathbf{w}, w_0) = \exp\{-\|\mathbf{x} - \mathbf{w}\| / w_0\} \quad (9)$$

This neuron yields a high output value for input data vectors that are close to their location (relative to w_0). This neuron transfer function is called a radial basis function.

The directional neurons can be thought to represent a direction in their input space. Their outputs are related to the angles between the input data vectors and their weight vectors, e.g.:

$$N(\mathbf{x}|\mathbf{w}, w_0) = \frac{1}{1 + \exp\{-\mathbf{w} \bullet \mathbf{x} - w_0\}} \quad (10)$$

This neuron transfer function is called a sigmoidal function or just sigmoid. Note that the transfer functions as defined by Equation 9 and Equation 10 yield values between 0 and 1. It is very common the scale the values within neural network layers in this way. Sometimes they are scaled between -1 and +1. In order to be able to produce arbitrary output values in the output layer linear transfer function have to be used:

$$N(\mathbf{x}|\mathbf{w}, w_0) = \mathbf{w} \bullet \mathbf{x} + w_0 \quad (11)$$

The feed-forward network as given in Figure 1 can be used for all three types of mappings given in the first section. Prototypes can be represented by radial basis functions in the input layer. The distance between an input vector and the set of prototypes is defined in the output layer. If sigmoidal outputs are used the network represents some probability density function if the outputs are correctly normalised. A feed-forward networks with just sigmoidal transfer function represents a mapping by nonlinear subspaces. Another common type of neural networks is the self-organising map (SOM) or Kohonen network as shown in Figure 2. In this network of spatial neurons, the neurons are not connected. They thereby represent a set of prototypes. During the optimization of the network some connectivity is assumed. This will further be discussed below.

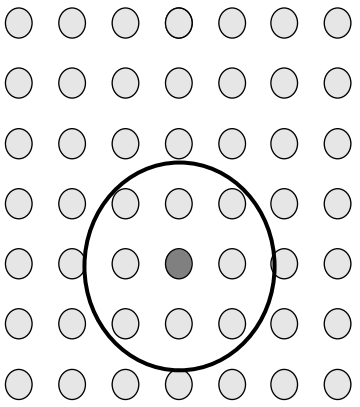

Figure 2: Architecture of a 2D self-organising map.

3 Unsupervised techniques

We will now give some examples on how neural networks can be trained in an unsupervised way. The three mapping types given in the introduction will be treated separately.

3.1 Prototypes by vector quantization

Suppose we have an initial set of prototypes (e.g. a random selection of the original data set) $Z = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_i, ... \mathbf{z}_n\}$. A very simple way of updating this set is by determining the subsets $X_i \subseteq X$, i=1,n such that X_i contains all vectors in X that have $\mathbf{z}_i$ as their nearest prototype. Now all prototypes are replaced by the means of the data vectors in the corresponding X_i. This is reiterated until stability. This procedure is called the *k-means* method. A sequential procedure can be defined by:

$\mathbf{z}_j = \mathbf{z}_j + \alpha(t)[\mathbf{x}_i - \mathbf{z}_j]$ if $\mathbf{z}_j$ is the nearest prototype of $\mathbf{x}_i$ (12)

repeat for i = 1,m

iteratively repeat with decreasing $\alpha(t)$.

Formally this procedure has not much relation with neural networks. It is, however, for historical reasons treated in connection with these.

3.2 Subspaces by neural networks

3.2.1 Subspaces by self-organising maps (SOMs)

Subspaces can be found in relation with the above treated vector quantization method by conditioning the set of prototypes in a grid of the desired subspace dimensionality. Most popular is the use of a n x n grid of n^2 neurons representing n^2 prototypes. This structure is emphasized and preserved by replacing the above update rule by:

$\mathbf{z}_j = \mathbf{z}_j + \alpha_{jk}(t)[\mathbf{x}_i - \mathbf{z}_j]$ if $\mathbf{z}_j$ is the nearest prototype of $\mathbf{x}_i$ and $\mathbf{z}_k$ is within some predefined grid neighbourhood of $\mathbf{z}_j$. $\alpha_{jk}(t)$ is a weighting function that decreases with the number of iterations and with the grid distance between $\mathbf{z}_j$ and $\mathbf{z}_k$.

This procedure may be followed for structures of any dimension. In practice, however, just one-, two-, and sometimes three-dimensional grids are used. For more dimensions too many neurons will be needed.

3.2.2 Subspaces by diabolo feedforward networks

Feedforward networks as given in Figure 1 represent in each layer a subspace. If it is possible to map the input data on a particular layer a reconstruction by following layers should be possible, see Figure 3. The 3rd and the 4th layer restore the original data vectors after they are mapped on the 2nd layer. Training should be done by minimizing the difference between inputs and outputs of the entire network. In fact this is a supervised procedure, see below. The data set itself, however, does not contain any target values. Thereby this is a supervised solution of a nonsupervised problem.

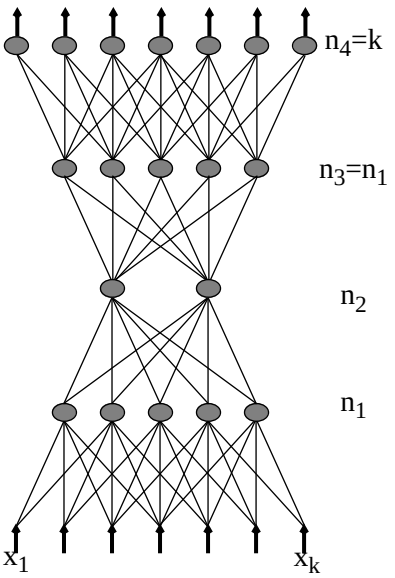

Figure 3: 4-layer diabolo network

3.3 Probability density functions

In this case the neural network outputs are considered as probability densities. For that reason the output layer consist of a single neuron and has a linear transfer function. Usually the first layer has radial basis transfer functions. A good network yields high outputs for the training data vectors. The maximum likelihood criterion is thereby often chosen for optimizing the network:

$$J = \sum_i \log(y_i) \qquad (13)$$

4 Supervised techniques

We will now give some examples on how neural networks can be trained in an supervised way. Recall that in this case the given data set X consists of vectors $\mathbf{x} = (\mathbf{x}', \mathbf{y})$ and that networks have to be found that estimate $\mathbf{y}$ as correctly as possible if the part $\mathbf{x}'$ is given.

4.1 Prototypes by learning vector quantization

The quality of prototypes has to be judged on the basis of the distance between the desired and the estimated **y**-values (targets). Equation 12 might therefore be replaced by something like

$$\mathbf{z}_j = \mathbf{z}_j + A(t)[\mathbf{x}_i - \mathbf{z}_j] \text{ if } \mathbf{z}_j \text{ is the nearest prototype of } \mathbf{x}_i \qquad (14)$$

in which the matrix A(t) has larger values for the target components.

4.2 Subspaces by neural networks

This problem is the traditional use of the feedforward network, Figure 1. The network outputs are the target values. Again linear output transfer functions might be needed. A diabolo network is not necessary now. A wide range of network optimization techniques is available. Traditionally backpropagation is used, an iterative gradient descent technique. Faster techniques, especially for smaller networks, might be preferred.

4.3 Probability density functions

The unsupervised solution can be applied also here. If the joint probability density function for $\mathbf{x} = [\mathbf{x}', \mathbf{y}]$ is estimated by $f(\mathbf{x}', \mathbf{y})$, the **y**-value for a given **x** might be found by

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}}(f(\mathbf{x}', \mathbf{y})) \qquad (15)$$

References

1 J.C. Mao and A.K. Jain, Artificial neural networks for feature extraction and multivariate data projection, IEEE Transactions on Neural Networks, vol. 6, no. 2, 1995, 296-317.

2 A.K. Jain, J. Mao, and K.M. Mohiuddin, “Artificial Neural Networks: A Tutorial,” Computer, vol. 29, no. 3, pp. 31-44, 1996.

3 T. Kohonen, Self-organizing maps, Springer, Berlin, 1995

4 C.M. Bishop, Neural networks for pattern recognition, Clarendon Press, Oxford, 1995

5 B.D. Ripley, Pattern recognition and neural networks, Cambridge University Press, Cambridge, 1996.