Neurocomputing 2 (1990/1991) 183/197
Elsevier

Multilayer perceptrons for classification and regression

Fionn Murtagh*

*Space Telescope – European Coordinating Facility, European Southern Observatory, Karl-Schwarzschild-Str. 2.
D-8046 Garching, Germany*

Abstract

Murtagh, F., Multilayer perceptrons for classification and regression, Neurocomputing 2 (1990/91) 183–197

We review the theory and practice of the multilayer perceptron. We aim at addressing a range of issues which are important from the point of view of applying this approach to practical problems. A number of examples are given, illustrating how the multilayer perceptron compares to alternative, conventional approaches. The application fields of classification and regression are especially considered. Questions of implementation, i.e. of multilayer perceptron architecture, dynamics, and related aspects, are discussed. Recent studies, which are particularly relevant to the areas of discriminant analysis, and function mapping, are cited.

Keywords. Multilayer perceptron; discriminant analysis; supervised classification; regression; function approximation.

1. Introduction

In this article, we introduce the multilayer perceptron and its application as a supervised classification method. This article critically asseses the multilayer perceptron in relation to statistical and pattern recognition approaches to the problems of classification and regression. The single perceptron is first described, and subsequently the networking of perceptrons as a multilayer perceptron. The influential generalized delta rule, used in training the network, is introduced via the simpler case of the delta rule. Section 5 deals with what design considerations one should consider, for the design of one's network for a specific problem. Section 6 discusses alternative algorithms for determining weights in the multilayer perceptron, which lead to implementations which are more powerful in practice. Section 7 seeks to further elucidate aspects of the multilayer perceptron, which help to answer the question "When should this approach be preferred to alternative pattern recognition methods?" Sections 8 to 10 deal with examples of applications. The balance-sheet, here, is somewhat uneven. In many instances, published papers claiming promising results for the multilayer perceptron scarcely investigate alternative methodologies, if at all. Papers based on various 'toy problems' unfortunately predominate. The multilayer perceptron offers a valid alternative to competing approaches, but this is contingent on

(i) having a good multilayer perceptron algorithm, and
(ii) understanding what characterizes a multilayer perceptron compared with the solutions yielded by other methods.

Where multilayer perceptrons offer an edge

[*] Affiliated to the Astrophysics Div., Space Science Dept., European Space Agency.

0925-2312/91/$03.50

over traditional pattern recognition is in all likelihood due to the nonlinear boundaries or modelling provided. For alternatives to the multilayer perceptron for regression and for supervised classification problems, see Duda and Hart [5], or Hand [15], among other texts.

2. Perceptron learning

Improvement of classification decision boundaries, or assessment of assignment of new cases, have both been implemented using neural networks since the 1950s. The *perceptron* algorithm is due to Rosenblatt in the late 1950s. The perceptron, a simple computing engine which has been dubbed a 'linear machine' for reasons which will become clear below, is best related to supervised classification. The idea of the perceptron has been influential, and generalizations in the form of multilayer networks will be looked at later.

Let $\boldsymbol{x}$ be an input vector of binary values; o an output scalar; and $\boldsymbol{w}$ a vector of weights (or learning coefficients; initially containing arbitrary values). The perceptron calculates $o = \Sigma_j w_j x_j$. Let θ be some threshold.

If $o \geqslant \theta$, when we would have wished $o < \theta$ for the given input, then i is incorrectly categorized. We therefore seek to modify the weights and the threshold.

Set $\theta \leftarrow \theta + 1$ to make it less likely that wrong categorization will take place again.

If $x_j = 0$ then no change is made to w_j. If $x_j = 1$ then $w_j \leftarrow w_j - 1$ to lessen the influence of this weight.

If the output was found to be less than the threshold, when it should have been greater for the given input, then the reciprocal updating schedule is implemented. The updates to weights and thresholds may be denoted as follows:

$$o = \sum_j w_j x_j$$

$\Delta\theta = -(t_p - o_p) = -\delta_p$ (change in threshold for pattern p)

$\Delta w_i = (t_p - o_p)x_{pi} = \delta_p x_{pi}$ (change in weights for pattern p).

If a set of weights exist, then the perceptron algorithm will find them. A counter-example is the exclusive-or (XOR) problem:

Input vector		Desired output
0	0	0
0	1	1
1	0	1
1	1	0

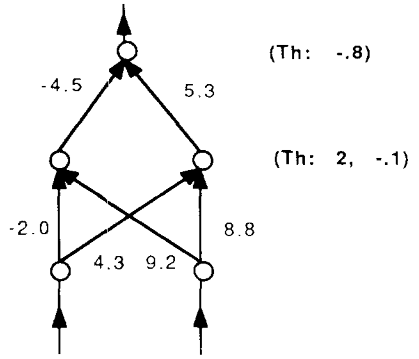
Here it can be quickly verified that no way exists to choose values of w_1 and w_2 to allow discrimination between the first and fourth vectors (on the one hand), and the second and third (on the other).

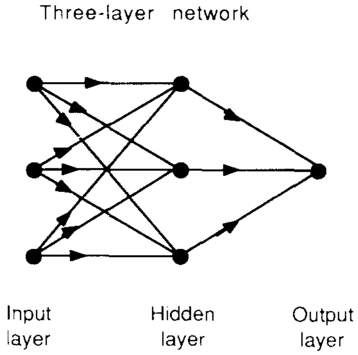
Consider the following sets of four input vectors which we wish to automatically categorize in accordance with the given truth table values. Truth tables, per se, are not at issue here. These sets of vectors, and class assignments, will serve purely as examples. A condition for the applicability of the Perceptron Learning algorithm will become clear, on consideration of graphical representations of these problems. Draw the 2-valued points in the plane. Note the regions spanned by the F-related points versus the T-related points in each case.

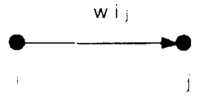
AND			OR			XOR		
0	0	F	0	0	F	0	0	F
0	1	F	0	1	T	0	1	T
1	0	F	1	0	T	1	0	T
1	1	T	1	1	T	1	1	F

The line (hyperplane) separating 'T' from 'F' is defined by $\Sigma_j w_j x_j = \theta$. In the XOR case, linear separability does not hold. Perceptron Learning fails for nonseparable problems.

A multilayer perceptron, which solves the XOR problem, is shown below. Values of weights and thresholds which permit this are shown.







Network designs which are more sophisticated than the simple perceptron can be used to solve this problem. The network shown in the Figure above is a feedforward 3-layer network. This type of network is the most widely used. It has directed links from all nodes at a level to all nodes at the next. Such a network architecture can be related to linear algebra transformations (see remarks at the end of the next section). It has been found to provide a reasonably straightforward architecture which can also carry out learning, or supervised classification, tasks quite well. For further discussion of the perceptron, Gallant [10] may be referred to for various interesting extensions and alternatives. Other architectures are, clearly, always possible: directed links between nodes at a given level; symmetric directed links between all pairs of nodes in the network, as in the Hopfield model; directed links from later levels back to earlier levels, as in recurrent networks; and so on.

3. The delta rule for updating weights in layered networks

A generalization of a perceptron is a set of layers of perceptron, with connections between all neurons in one layer and all neurons in the subsequent layer. This gives rise to what is termed a feedforward multilayer perceptron (MLP).

Initially we consider linear units only, i.e.

$$o_{pj} = \sum_i w_{ij} x_{pi} .$$

The input at the ith neuron, x_{pi}, is occasioned by the pth pattern. The weight connecting the ith neuron in a given layer, to the jth neuron in the subsequent layer, is denoted w_{ij}. Consider an energy term which we seek to minimize at each output neuron j; and let p be the pattern which is currently being presented to the net. Then

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2$$

where o is the output obtained when using the current set of weights. The multiplicative constant of a half is purely for conventional purposes. The target output, t, is what we wish the network to replicate on being presented with the pth pattern. Consider

$$E = \sum_p E_p .$$

We may write the expression for E_p as

$$\frac{1}{2} \sum_j \delta_{pj}^2 .$$

The rate of change of E_p with respect to w_{ij} is given by the chain rule:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ij}} .$$

Now, since $E_p = \frac{1}{2}\Sigma_j(t_{pj} - o_{pj})^2$, the first term here is $-(t_{pj} - o_{pj})$. Given that linear units are being considered, i.e. $o_{pj} = \Sigma_i w_{ij} x_{pi}$, the second term equals x_{pi}. The gradient of E_p is thus

$$\frac{\partial E_p}{\partial w_{ij}} = -\delta_{pj} x_{pi} .$$

If $\partial E / \partial w_{ij} = \Sigma_p \, \partial E_p / \partial w_{ij}$ and if updates do not take place after every training pattern, p, then we may consider the updating of weights as a classical gradient descent minimization of E. Each weight is updated according to

$$\Delta_p w_{ij} = \eta (t_{pj} - o_{pj}) x_{pi} ,$$

where η is a small constant. This corresponds to steepest descent optimization: we vary the weights in accordance with the downwards slope.

Although the algorithm presented for determining optimal weight values through training is valid, it is not difficult to see that we can view what has been done in linear algebra terms. Consider the matrix W_1 as defining the weights between all input layer neurons, i, and all hidden layer neurons, j. Next consider the matrix W_2 as defining the weights between all hidden layer neurons, j, and all output layer neurons, k. For simplicity, we consider the case of the three-layer network but results are immediately generalizable to a network with more layers. Given a vector of inputs, x_p, the values at the hidden layer are given by $x_p W_1$. The values at the output layer are then $x_p W_1 W_2$. Note that we can 'collapse' our network to one layer by seeking the weights matrix, $W = W_1 W_2$. If t_p is the target vector, then we are seeking a solution of the equation $x_p W = t_p$. The backpropagation algorithm described above would not be interesting for such an ordinary problem. Backpropagation assumes greater relevance when nonlinear transfer functions are used at neurons.

4. Generalized delta rule for nonlinear units

Nonlinear transformations are less tractable mathematically, but may offer more sensitive modelling of real data. They provide a more faithful modelling of linear threshold electronic gates or biological neurons. This section updates the previously derived update rules for the case when the transfer function at each neuron is nonlinear.

Consider the accumulation of weighted values of a neuron

$$\text{net}_{pj} = \sum_i w_{ij} o_{pi} ,$$

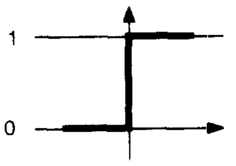
where $o_i = x_i$ if unit i is an input one. This is passed through a differentiable and nondecreasing transformation, f:

$$o_{pj} = f_j(\text{net}_{pj}) .$$

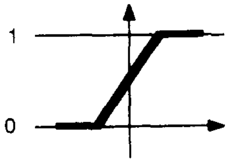
Normally this transfer function is a sigmoidal one. The following figure shows a number of examples (including the linear and binary cases). The sigmoid is defined by $y = (1 + e^{-x})^{-1}$. The quite similar hyperbolic tangent may be determined as follows: for $x > 20.0$, $y = +1.0$; for $x < -20.0$, $y = -1.0$; otherwise $x = (e^x - e^{-x}) / (e^x + e^{-x})$. The binary function is fast and easy to use, especially in hardware. It makes hard decisions. It is not invertible and cannot smoothly imitate arbitrary functions. The linear ramp is relatively easy to implement, and can imitate functions. It is not invertible. The sigmoid, as also the hyperbolic tangent, is invertible and continuously differentiable. Both have semilinear zones, which allow sensitive imitation of input data. Both can make 'soft' or fuzzy decisions. The sigmoid, in particular, is similar to the response curve of the biological neuron.

As before, the change in weights will be defined to be proportional to the energy slope, and the chain rule yields:

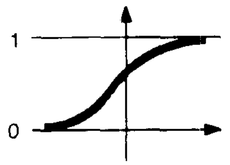
$$\Delta_p w_{ij} \propto -\frac{\partial E_p}{\partial w_{ij}} = -\frac{\partial E_p}{\partial \text{net}_{pj}} \frac{\partial \text{net}_{pj}}{\partial w_{ij}} .$$



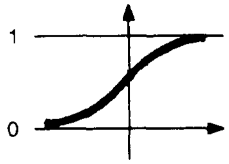
Binary



Linear



Sigmoid



Hyperbolic Tangent

From the definition of net_{pj}, the last term is o_{pi}. Let $\delta_{pj} = -\partial E_p / \partial \text{net}_{pj}$. Hence

$$-\frac{\partial E_p}{\partial w_{ij}} = \delta_{pj} o_{pi}$$

or

$$\Delta_p w_{ij} = \eta \delta_{pj} o_{pi} ,$$

where η is the learning constant, usually a small fraction which prevents rebounding from side to side in ravines in the energy surface.

Parenthetically, note that for a linear output unit, by definition $o_{pj} = \text{net}_{pj}$, and so we have $-\partial E_p / \partial o_{pj} = \delta_{pj}$ as was seen above when considering such units.

It must now be determined how to define δ_{pj}. We have

$$\delta_{pj} = -\frac{\partial E_p}{\partial \text{net}_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial \text{net}_{pj}} ,$$

and the last term is equal to $f_j'(\text{net}_{pj})$, i.e. the derivative of the transfer function. Two cases will be distinguished, depending on whether the unit is an output one or a hidden layer one.

Case I. Unit j is an output one, and it is found that

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(\text{net}_{pj}) .$$

Case II. Unit j is a hidden layer one, and it is found that

$$\delta_{pj} = f_j'(\text{net}_{pj}) \sum_k \delta_{pk} w_{kj} .$$

Hence the deltas at an internal node can be derived from the deltas at a subsequent (closer to output) layer.

The overall algorithm is as follows: present pattern; feed forward, through successive layers; backpropagate – updating weights; repeat.

An alternative is to determine the changes in weights as a result of presenting all patterns: $\Delta w_{ij} = \Sigma_p \Delta_p w_{ij}$. This so-called 'off-line' updating of weights is computationally more efficient but loses out on the adaptivity of the overall approach.

Some further notes on the multilayer perceptron, using the generalized delta rule, follow.

A local optimum set of weights may be arrived at. There is no guarantee of arriving at a global optimum in weight space.

For the logistic activation function defined by

$$f_i(\text{net}_{pj}) = 1/(1 + \exp^{-\text{net}_{pj}}) ,$$

where

$$\text{net}_{pj} = \sum_i w_{ij} o_{pi} + \theta_j .$$

and the last term is a bias (or threshold), we have the following result:

$$f_j'(\text{net}_{pj}) = o_{pj}(1 - o_{pj}) .$$

For this function:

$$\delta_{pj} = (t_{pj} - o_{pj}) o_{pj} (1 - o_{pj}) \quad \text{for an output unit}$$

$$\delta_{pi} = o_{pj}(1 - o_{pj}) \sum_k \delta_{pk} w_{ki}$$

for a hidden layer unit.

In practice one must use approximations to hard-limiting values of 0, 1; e.g. 0.1, 0.9 can be used. Otherwise, infinite weight values would be needed in the above expression for $f_j(\text{net}_{pj})$.

It is found that symmetry breaking is necessary in order to get the backpropagation algorithm started. This involves randomizing the initial arbitrary weight values.

The presence of an additional momentum term: $\Delta w_{ij}^{(n+1)} = \eta\delta_{pj}o_{pi} + \alpha\Delta w_{ij}^{(n)}$ often helps the convergence properties of the steepest descent iterative optimization. This term takes the effect of previous steps into account. If the change in the previous step was large, the momentum tends to continue to keep the delta large.

The learning rate, η, should be small (0.7 or smaller). A larger learning rate implies quicker learning, but possible oscillations. The additional momentum term should have momentum rate α close to 1 (e.g. 0.9). On the one hand, these parameters should be set such that convergence is attained; on the other hand, they must allow some change to take place. It may be desired to allow α to be small initially, and to increase it in the course of convergence. In sum, the most appropriate choice of these parameters depends on the training data and the network architecture.

The MLP architecture, using the generalized delta rule, can be slow for reasons such as the following. The compounding effect of sigmoids at successive levels can lead to a very flat energy surface. Hence movement towards fixed points is not easy. Such a flat energy landscape has been well described in Lapedes and Farber [20]. A second reason for slowness when using the generalized delta rule is that weights tend to try together to attain the same value. This is unhelpful – it would be better for weights to prioritize themselves when changing value. This is what has been done in the algorithm of Fahlman and Lebiere [7] and constitutes a promising current research direction.

5. Multilayer perceptron architectures

The number of hidden layers in a multilayer perceptron, and the number of nodes in each layer, can vary for a given problem. In general more nodes offer greater sensitivity to the problem being solved, but also the risk of overfitting (cf. the discussion on regression in Section 7). The following are a number of relevant references on this topic..

Lippmann [22] justifies three layers as being adequate on the basis of the dichotomies which can be carried out by each neuron/perceptron. Lapedes and Farber [20] discuss the energy surface in multilayer networks, with sigmoid transfer functions, and favour the use of two hidden layers for approximating an unknown function. Without empirical results, Chester [4] similarly argues in favour of more than one hidden layer.

Hecht-Nielsen [17] discusses the possibility of a 3-layer perceptron approximating arbitrary functions, given appropriate squashing functions at the neurons. He bases this result on work by Kolmogorov which stated that any function could be exactly modelled by a multilayer network. Kolmogorov's result was premised on appropriate transfer functions – not necessarily the same and also not necessarily smooth – at the nodes of the network. Hecht–Nielsen's study was critically dicussed by Girosi and Poggio [12].

Funahashi [9] show that the multilayer perceptron allows approximation of any continuous mapping. Hornik et al. [18] find in favour of 3-layer networks for approximating arbitrary Borel-measurable functions, provided sufficient hidden layer neurons are present.

Although the theory presented in these studies helps to justify the use of multilayer perceptrons, two issues are still open. Firstly, the network architecture which should be used in any particular instance cannot be specified in advance. In many instances, a one hidden-layer network with $2n + 1$ neurons, where n is the number of inputs, can be recommended, but this is based on empirical grounds more than anything else. The second issue which is still open probably casts light on why a precise network architecture cannot in fact be specified in advance. If a multilayer perceptron can be useful for representing a function, it is clear that the network architeture should be chosen in accordance with the function to hand. Singling out functions, or categorizing them in a non-trivial manner, is probably not possible in general.

6. Alternative MLP algorithms

A multilayer perceptron is defined by:

(i) its configuration or architecture – number of layers, number of neurons per layer, etc.;

(ii) activation methods – the transfer functions used at various neurons;

(iii) the specification of the learning method – gradient descent using the generalized delta rule, conjugate gradient, or other method; and

(iv) specification of events – whether weight updating is carried out in 'real-time' (i.e. following each training pattern) or 'off-line' (at each epoch, following presentation of all training patterns).

If we choose (i) and (ii) here, options for (iii) and (iv) can be varied. In fact this distinction between specifying the net architecture, and the training-related aspects, is reminiscent of the distinction between a general method and an algorithm for its implementation. In this article, we use the term 'multilayer perceptron' for the general method or class of architectures; and, for learning, we consider backpropagation or the generalized delta rule; Quickprop; conjugate gradient optimization; and so on.

An efficient algorithm for the MLP is important for practical problems. As has been seen, optimization of the weights in a multilayer perceptron may be carried out by the backpropagation algorithm. Additionally, the incorporation of a 'momentum' term has been found to improve the convergence properties.

Another alternative to improve speed is to use information from the previous iteration in the form of the previous slope and the previous step-size. This is what Fahlman's [6] Quickprop algorithm does. As in standard backpropagation, Quickprop determines $\partial E/\partial w$ but a second-order method is used instead of gradient descent. Updating involves a function of $\partial E/\partial w_{t-1}$, $\partial E/\partial w_t$ and Δw_{t-1}.

Conjugate gradient (Press et al., [24]) is an approach to multidimensional minimization. It does not require the storage of Hessian matrices as do quasi-Newton or variable metric methods. The Hessian is the matrix of second order partial derivatives of the error function. In the second order methods mentioned, either the inverse of the Hessian, or an approximation of this inverse, is required. The conjugate gradient approach on the other hand requires the current and previous gradient and search vectors in addition to the weights. A conjugate gradient algorithm is described by Barnard and Cole [2], and a comprehensive review of this approach is available in Johansson et al. [19]. The conjugate gradient algorithm updates weights only after all inputs have been processed, and their contributions to the changes in weights assessed. Hence this procedure operates in an 'off-line' mode. On-line updating of weights would be advantageous when extremely large datasets are being handled. The conjugate gradient approach is similar to backpropagation insofar as the same error term is to be minimized. It aims at a closed-form minimization, which is strictly valid in the case of the error being a quadratic function of the weights. As steps downwards are taken in the error surface, 'conjugate' directions are used for each iteration. Conjugate directions are appropriate orthogonal directions (see Johansson et al., [19]), and allow the minimum to be determined in precisely m steps. This value m is the dimensionality of the error space, i.e. the total number of weights to be determined, and thus is network architecture-dependent. In practice, the error surface would rarely be quadratic. Assuming that the highly nonlinear error function is locally quadratic, the approach can still be used. Now, a fixed number of m steps in the minimization no longer is valid, and instead iterative convergence is used. Within this general perspective there is still latitude for different implementations of the conjugate gradient approach.

Although no doubt open to debate (and to further convincing research results), we feel that a conjugate gradient-based MLP is a necessary condition for use of the MLP in practice, assuming that specialized hardware is not being availed of.

7. Just what does the MLP do?

The following general objectives in training a multilayer perceptron might be of relevance:

- *Regression*. An input vector is to be mapped onto some given set of values by the network. In many cases the output will consist of one value. In this case, the network regresses the independent or carrier variables, provided by the inputs, onto the dependent variable.

  It will be seen how the multilayer perceptron allows nonlinear regression to be carried out. It will also be seen below how such a regression may be used for prediction or forecasting.

- *Supervised classification or discriminant analysis*. An input vector or pattern is to be trained to produce a classification vector at the output of the network. Thus, for example, a classification vector of values $(0, 0, 1)$ could be used to code a three-class problem, where it is wanted to map the input pattern onto class number 3.

  It will be seen how the multilayer perceptron allows piecewise linear separation boundaries to be carried out.

Consider, by way of example, a general regression problem. This example will additionally show one approach to forecasting using the multilayer perceptron. The net's architecture consists of 3 input layer neurons, 5 hidden layer neurons, and 1 output layer node. The logistic function $1/(1+e^{-x})$ is used as the transfer function at the hidden units, and the identity function was used at the output units. Using the identity function at the output of the network is often done so that real-valued outputs are obtained. If the logistic function, for instance, were employed then the outputs would necessarily be squashed into the range [0, 1].

The forecasting of an unknown value of a time series was the objective. From the time series for which the multilayer perceptron was to function as a forecasting engine, a large number of pairs $\{y_{t-1}, y_{t-2}\}$ were randomly drawn with replacement. The inputs were: $\{1, y_{t-1}, y_{t-2}\}$. One of the input units is often clamped to a unit value, as here, in order to produce the same effect as having a threshold at the next layer's neurons. The architecture and optimand are discussed in the box.

> Prediction as nonlinear regression.
>
> The objective is to have the net learn to output the value y_t when presented with values y_{t-1} and y_{t-2}. Hence we are also regressing y_t on the values y_{t-1}, y_{t-2}.
>
> Consider $\boldsymbol{x}_t = (1, y_{t-1}, y_{t-2})$. The input value 1 is used to provide a threshold.
>
> The net architecture will be 3 input nodes (to be associated with the above three inputs); 5 hidden layer nodes; and one output node.
>
> Let ξ_t be the target value. Let U be the weights on the links connecting the input to the hidden layers; U is a 3×5 matrix.
> Let W be the weights on the links connecting the hidden to the output layers; W is a 5×1 matrix.

The weights on entry to the hidden layer are given by the product $x'_t U$ (where $'$ represents transpose).

On exit from the hidden layer, we have the following: $1/(1+e^{-x'_t U})$.

For given values of ξ_t and x_{tj}, we seek values of w and of u satisfying:

$$\xi_t = w_0 + \sum_{k=1}^{5} w_k \left(\frac{1}{1 + \exp\left(\sum_{j=1}^{3} -x_{tj} u_{jk} \right)} \right)$$

Note that a linear transfer function is used here for output neurons, and a sigmoid transfer function at the hidden layer neurons.

An MLP solves the above equation. A complete example, very similar to the above, is discussed in the next section.

In Lapedes and Farber [21], the network additionally (and unusually) had links between the input and output layers. The output was, in this instance, given by the following equation (a few values have been rounded). Note where the weights are located in *Fig. 1*.

$$\begin{aligned} y_t = -0.64 \quad & g(-1.11 y_{t-1} - 0.26) \\ -1.3 \quad & g(2.22 y_{t-1} - 1.71) \\ -2.3 \quad & g(3.91 y_{t-1} + 4.82) \\ -3.9 \quad & g(2.46 y_{t-1} - 3.05) \\ +6.0 \quad & g(1.68 y_{t-1} + 0.60) \\ & +(0.31 y_{t-1} - 2.0) \end{aligned}$$

with $g(x) = \frac{1}{2}(1 + \tanh x)$.

The data on which this was carried out was the logistic map, $y_t = 4y_{t-1}\ (1 - y_{t-1})$, which is a simple example of a series exhibiting chaotic behaviour. A plot of y_t against y_{t-1} is a parabola. So also is the more unwieldy equation defined above which was found by the neural net. Inputs and outputs to the net were scaled to take values in [0, 1]. In this region the two equations are the same.

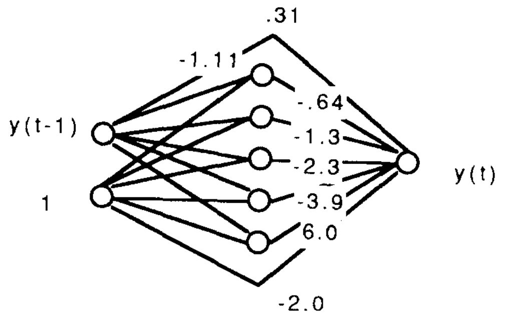


Fig. 1. Architecture and (some) final weights for a forecasting problem.

In the supervised classification or discriminant analysis problem, we seek a mapping of each input vector onto a target indicator vector. If, for example, it is decided that there are 3 classes, then the target vector could be (1.0, 0, 0) for class 1, (0, 1.0, 0) for class 2, and (0, 0, 1.0) for class 3. This implies the presence of 3 neurons in the output layer. If a sigmoid or other squashing transfer function is used in the output layer, infinite weight values would be required to provide zero/one output values. For this reason, an output of (0.2, 0.8, 0.2) would be considered, by convention, acceptably close to (0, 1.0, 0).

If we went further and accepted anything over 0.5 as being 1, and anything under 0.5 as being 0, we could view output layer neurons in the network as producing two-way splits of the data. It is legitimate to think of what the network does in these terms: each neuron brings about a two-way split of the data. Hence the outputs are the compound effect of many two-way splits in the network. Note that any two-way division of the data, in a space of arbitrary dimensionality m, means drawing an $m-1$-dimensional curve (an $m-1$-dimensional hypersurface) between two parts of the data. Hence the compound effect of two-way splits is the drawing of piecewise linear divisions between parts of the data.

8. Example: forecasting sunspots

Monthly means of daily relative sunspot readings, based on counts of spots and groups of spots, are available from the beginning of the 18th century onwards. This and other sunspot data sets have been used as benchmarks for nonlinear and other prediction methods. The data set used is available as a test data set in the S-Plus statistical environment package. It is clear that sunspots are due to physical processes, but what these processes are is not known. Hence the interest in being able to make sense of these data purely by numerical means. Here are a few arbitrary years' values:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1749:	58.0	62.6	70.0	55.7	85.0	83.5	94.8	66.3	75.9	75.5	158.6	85.2
1833:	11.3	14.9	11.8	2.8	12.9	1.0	7.0	5.7	11.6	7.5	5.9	9.9
1905:	54.8	85.8	56.5	39.3	48.0	49.0	73.0	58.8	55.0	78.7	107.2	55.5
1976:	8.1	4.3	21.9	18.8	12.4	12.2	1.9	16.4	13.5	20.6	5.2	15.3

We took values in the period 1937 to 1977, inclusive. We took two successive values, with the aim of predicting the following value. That is, the values of May and June 1963 (for instance) were used to predict the value of July 1963. Without loss of generality, the data were normalized so that all values were comprised between 0 and 1. This was necessitated by the multilayer perceptron program used, and was motivated by the desire to avoid possible arithmetic overflow errors. We treated this problem as a classification problem, in the following way: the value to be predicted was discretized into the range 1 to 10 in a straightforward manner. Then training and test sets were chosen from the triples, i.e. category to be predicted, followed by the two months' values. Every even-numbered triple was chosen as a training pattern – in all 241 triples. Every triple with odd sequence number was chosen as a test pattern – in all 240 triples.

By way of example, here are the first 10 training set patterns (value to be predicted, followed by two months' values):

4	0.5220646	0.5063041
5	0.3305753	0.4306541
6	0.4598109	0.5133964
4	0.5717101	0.5425532
3	0.3967691	0.4921198
4	0.2931442	0.3498818
4	0.3877068	0.4696611
6	0.3408195	0.3979511
7	0.5019701	0.3841608
4	0.6513003	0.4558707

The following are the first 10 test set patterns (known value to be correctly predicted by the system, followed by two test inputs to the system):

5	0.5063041	0.3305753
6	0.4306541	0.4598109
6	0.5133964	0.5717101
5	0.5425532	0.3967691
4	0.4921198	0.2931442
5	0.3498818	0.3877068
4	0.4696611	0.3408195
4	0.3979511	0.5019701
5	0.3841608	0.6513003
4	0.4558707	0.3530339

Clearly the decision surface is quite intricate. The output neurons were expanded in a disjunctive coding manner to ten values, i.e. category 4 had a 'one' value in position 4 and 'zero' values elsewhere. The network architecture therefore had 2 neurons as inputs; 10 neurons as outputs; and a range of possibilities was investigated for the appropriate number of hidden layer neurons. Our final choice was 5 hidden layer neurons. In line with a fairly regular practice, an additional input layer neuron had its value always clamped to 1: this provided a threshold. Hence this architecture was 3–5–10. The conjugate gradient program described in Barnard and Cole [2] was used.

The following is the set of network weights on completion of training:

```
Weights:
*** layer 1 to layer 2 ***
to neuron 1:         -1.937920          -1.737793          -4.493082
to neuron 2:         -0.902301          -0.103767           5.103320
to neuron 3:          1.798849          10.164346          -4.055804
to neuron 4:         -0.974593         -10.609833           0.578527
to neuron 5:          1.603968           5.668143          -4.863671

*** layer 2 to layer 3 ***
to neuron 1:    -3.294010   -0.735511   -2.653013    2.871830    2.465173
to neuron 2:    -1.325523    2.332112   -8.393360   -5.614480   -2.738902
to neuron 3:    -0.816895    0.823868   -1.405683   -4.681433   -2.865742
to neuron 4:     1.522225   -0.127896    0.751299   -3.323257   -4.154643
to neuron 5:     0.427933   -1.229853    2.594134   -2.522885   -5.145156
to neuron 6:     1.986840   -1.699099    1.887604   -1.224791   -1.318854
to neuron 7:     1.279756   -1.714017    1.130694   -2.436572    0.542844
to neuron 8:     1.789475   -2.349161    1.498176   -1.556752    0.892129
to neuron 9:    -0.855488   -2.457291    0.567503    0.006136    1.995623
to neuron 10:    0.507262   -2.658463    1.883208   -1.809968    0.017136

Value of criterion function: 10.417791
```

Testing was then carried out on the 240 patterns of the test set. The following table gives the category obtained relative to that wanted.

Class	Frequencies of assignments of test patterns									
label	1	2	3	4	5	6	7	8	9	10
1:	59	1	0	0	0	0	0	0	0	0
2:	18	21	5	0	0	0	0	0	0	0
3:	1	12	15	2	1	0	0	0	0	0
4:	0	3	7	13	11	0	0	0	0	0
5:	0	0	3	8	21	3	1	0	0	0
6:	0	0	0	0	5	3	2	1	0	0
7:	0	0	0	0	1	2	8	1	0	0
8:	0	0	0	1	1	1	7	0	0	0
9:	0	0	0	0	0	0	0	0	0	0
10:	0	0	0	0	0	0	0	2	0	0

For 200 iterations, this took about 20 seconds CPU time on a Sun SPARCstation IPC. The percentage correct is 58.33%. If an 'approximately correct' result is acceptable, in the sense that a category above or below the desired category was deemed acceptable, then the percentage correct is 93.75%.

Our prediction of sunspots in the period considered is (well...) passable. However if an approximate forecast is sufficient, then the multilayer perceptron has done very well.

How does this compare with alternative approaches? There are many approaches which can be tried, including nonlinear time series, and this data has been used for example by Priestley [25] and by Weigend et al. [30]. While we have treated the problem as a classification one, Tong [28] deals thoroughly with nonlinear time series analysis approaches. Given the clearly complicated decision surface of this data, we used the same test and training sets with K-nearest neighbors discriminant analysis. Values of K = 1 and K = 2 were used. The NEIGHBOR routine in the widely-used SAS (Statistical Analysis System) package was employed. Not surprisingly, K = 1 performed better than K = 2. We found, for K = 1, a percentage correct result of 49.58%, and an approximately correct result (in the sense explained above) of 85.83%. The MLP performed better in both cases.

We have shown that the multilayer perceptron performed well, and did better than K-nearest neighbors classification in the case considered. We cannot, of course, exclude from consideration that some other neural network design could do better again, nor – equally – that some other non-MLP approach could improve on the results obtained.

9. Example: classification of Fisher's iris data

The data initially used by Fisher in the 1930s, in proposing what became known as Fisher's linear discriminant analysis, have been very often used as a test data set in the intervening years. The data consist of measurements made on 150 iris flowers. The data set was published in Fisher's original article (Fisher, [8]) and is also available from many other sources. The measurements are four in number (petal and sepal width and breadth). The 150 flowers come from three known groups, each consisting of 50 samples. The first such group is well-separated from the other two. However groups 2 and 3 overlap – they are not linearly separable – and many classification and discrimination methods only come close to distinguishing them.

Pao [23] provides C code for multilayer perceptron based on the generalized delta rule. This is fairly close to a gradient descent optimization to determine weights on edges in the network, so that the latter can perform a discriminant analysis task. In applying this to Fisher's iris data, the aim was just to establish acceptable class boundaries, so the entire set of 150 species, with 3 classes of 50 species, was used as a training set. A few different network architectures were tried, concentrating on 4–9–3 neurons in the input, hidden, and output layers respectively. The target values comprised a set of three values $(1, 0, 0$ for group 1; $0, 1, 0$ for group 2; and $0, 0, 1$ for group 3). The following values were used: η (learning constant) 0.9 and α (momentum constant) 0.7; maximum total error 0.0001, and maximum individual error 0.00001; the input data were rescaled so that all values were bounded by 0 and 1. 4000 iterations were carried out. At completion, there were four misclassifications. This is comparable or marginally worse than traditional linear methods. However, although the algorithm was performing adequately, there was no convergence after 28 minutes CPU time on a VAX 8600 machine. Such an amount of computation time was extremely large when contrasted with, for instance, Fisher's original method (Fisher [8]).

Alternative multilayer perceptron algorithms are more recommendable in practice. The conjugate gradient approach [2] worked well on

Fisher's iris data. This algorithm for the multilayer perceptron was used for instance in Smyth and Mellstrom [27] where it is also mentioned in very favourable terms. The discrimination between the 3 iris classes was achieved with a 4–15–3 net, with one wrong assignment, within seconds on a Sun SPARCstation 1.

Fahlman's [6] Quickprop also worked well: this is an enhancement of the generalized delta rule whereby information related not just to the current slope but also to the previous slope and to the size of the last jump is taken into account. A 4–15–3 net took a few minutes elapsed time on a SPARCstation. Finally – as far as this experimentation on Fisher's iris data is concerned – the cascade-correlation algorithm of Fahlman and Lebiere [7] worked well. This method creates a multilayer perceptron from scratch, only bringing a node in the network into being when needed. Hidden units are added one by one, in an architecture where every hidden unit h is connected to all inputs, and all outputs; and where hidden unit h is connected to $h + i$ for all i. The input side of each hidden unit is set, once the weights have been determined, and does not change as new hidden units are introduced on the output side. Training takes place in a similar manner to backpropagation, and the speed-up approach used in Quickprop can also be incorporated. The cascade-correlation algorithm performed slowly on Fisher's iris data, but the ability of a network to establish its own structure is certainly a beneficial property. Both Quickprop and the cascade-correlation algorithm achieved 100% correct class separation.

10. Further examples

A number of other references, with critical remarks, follow.

Huang and Lippmann [16] found MLP to be better than or equal to, K-nearest neighbour (K-NN) and quadratic Gaussian (or Bayesian maximum likelihood) classifiers. Speech data was used as input, and backpropagation for training the network. One- and two-valued inputs, only, appear to have been used. Although the results obtained were very good relative to more well-established methods, the scaling problem was not explicitly investigated, in particular as regards high-dimensional inputs. The problem of pathological cases was only briefly looked at. If there are difficult disjoint regions for the MLP, then what about non-disjoint regions which are quite likely to arise in practical problems?

Gish and Blanz [13] compare a MLP with quadratic Gaussian and first, second and third degree polynomial classifiers. Backpropagation is used. Input data comprised: two-category, two-feature, separable distributions; two-category, overlapping distributions of up to 3 features; and a 3-category, 10-feature image data set. All classifiers performed well on these problems.

Barnard and Casasent [1] and Casasent and Barnard [3] compare a MLP using backpropagation to a number of other methods including quadratic Gaussian and a least mean squares linear classifier. Also proposed is an adaptive-clustering neural classifier, which combines a linear classifier with a multilayer perceptron approach. A nearest-neighbour-based approach is used to obtain cluster prototypes which in turn are associated with the hidden layer neurons and provide initial estimates of the weights in the input layer/hidden layer part of the network. Simulated 2-dimensional data and 3-dimensional aircraft shapes are used in experiments.

Shadmehr and D'Argenio [26] compare a multilayer perceptron to Bayesian MAP (maximum a posteriori) and maximum likelihood estimates in the framework of examining chemical drug data. They find the MLP classifier to perform as well as the optimal Bayes estimator.

Gorman and Sejnowski [14], in a thorough study of 2-category sonar response to underwater metal cylinder and rock, found that a multilayer perceptron performed better than a nearest neighbour classifier (as we did also in section 8 of this paper). In various experiments, 60 inputs and up to 24 hidden layer neurons were used, with backpropagation used for training the network.

Gallinari et al. [11] and Webb and Lowe [29] point out how a multilayer perceptron, with appropriate linear transfer functions, can perform a multiple discriminant analysis. The latter is also known as canonical discriminant analysis, or discriminant factor analysis. It could be said to be a principal components analysis of the known classes, where in addition the generalized Euclidean (or Mahalanobis) metric replaces the usual Euclidean one. In experimentation they compare results obtained with nonlinear transfer functions to multiple discriminant analysis, which is premised on best-fitting linear combinations.

11. Conclusion

In this article we have pointed to the need for efficient training of a (software-implemented) MLP. It is clear that any optimization technique requires careful consideration of implementation issues.

By way of examples of where the multilayer perceptron can be fruitfully used, we have concentrated on classification and general regression. There are many preliminary studies which do not perform well when assessed in comparison with traditional procedures, but there are also some very interesting and convincing results.

Clearly, hardware implementations could broaden further the potential of the multilayer perceptron. Even when software implementations alone are at issue, the multilayer perceptron is an approach which should be kept in mind for practical problems.

References

[1] E. Barnard and D. Casasent, A comparison between criterion functions for linear classifiers, with an application to neural nets, *IEEE Trans. Systems Man Cybernet.* 19 (1989) 1030–1041.

[2] E. Barnard and R. Cole, A neural net training program based on conjugate-gradient optimization, Oregon Graduate Center Technical Report No. CSE 89-014, Beaverton, Oregon, 1989.

[3] D.P. Casasent and E. Barnard, Adaptive-clustering optical neural net, *Appl. Optics* 29 (1990) 2603–2615.

[4] D.L. Chester, Why two hidden layers are better than one, *Proc. Internat. Joint Conf. on Neural Networks*, Washington, DC (June 1990) I-265-268.

[5] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis* (Wiley, New York, 1973).

[6] S.E. Fahlman, Faster-learning variations on back-propagation: an empirical study, in: *Proc. 1988 Connectionist Models Summer School* (Morgan Kaufmann, Los Altos, CA, 1988).

[7] S.E. Fahlman and C. Lebiere, The cascade-correlation learning architecture, Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1990.

[8] R.A. Fisher, The use of multiple measurements in taxonomic problems, *Ann. Eugenics* 7 (1936) 179–188.

[9] K.-I. Funahashi, On the approximate realization of continuous mappings by neural networks, *Neural Networks* 2 (1989) 183–192.

[10] S.I. Gallant, Perceptron-based learning algorithms, *IEEE Trans. Neural Networks* 1 (1990) 179–191.

[11] P. Gallinari, S. Thiria and F. Fogelman Soulié, Multilayer perceptrons and data analysis, *IEEE Annual Conf. on Neural Networks*, IEEE, London (1988) I-391-399.

[12] F. Girosi and T. Poggio, Representation properties of networks: Kolmogorov's theorem is irrelevant, *Neural Computation* 1 (1989) 465–469.

[13] S.L. Gish and W.E. Blanz, Comparing a connectionist trainable classifier with classical statistical decision analysis methods, Research Report RJ 6891 (65717), IBM Research, Almaden Research Center, San Jose, California, 1989.

[14] R.P. Gorman an T.J. Sejnowski, Analysis of hidden units in a layered network trained to classify sonar targets, *Neural Networks* 1 (1988) 75–89.

[15] D.J. Hand, *Discrimination and Classification* (Wiley, New York, 1981).

[16] W.Y. Huang and R.P. Lippmann, Comparisons between neural net and conventional classifers, *IEEE First Internat. Conf. on Neural Networks*, IEEE, London (1987) IV-485–493.

[17] R. Hecht-Nielsen, Kolmogorov's mapping neural network existence theorem, *Proc. IEEE First Internat. Conf. Neural Networks*, III, San Diego (1987) 11–14.

[18] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* 2 (1989) 359–366.

[19] E.M. Johansson, F.U. Dowla and D.M. Goodman, Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method, UCRL-JC-104850 preprint, Lawrence Livermore National Laboratory, Sept. 1990.

[20] A. Lapedes and R. Farber, How neural networks work, Los Alamos National Laboratory report LA-UR-88-418. Also in: *Proc. IEEE Denver Conf. Neural Nets*, D.Z. Anderson, ed. (American Institute of Physics, 1988).

[21] A. Lapedes and R. Farber, Nonlinear signal prediction using neural networks: prediction and system modelling, Preprint, Los Alamos National Laboratory LA-UR-87-2662, 1987.

[22] R.P. Lippmann, An introduction to computing with neural nets, *IEEE Acoust. Speech Signal Process. Mag.* (1987) 4–22.

[23] Y.-H. Pao, *Adaptive Pattern Recognition and Neural Networks* (Addison-Wesley, Reading, MA, 1989).

[24] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in C* (Cambridge University Press, 1988) Chapter 10: "Minimization or maximization of functions".

[25] M.B. Priestley, *Non-Linear and Non-Stationary Time Series Analysis* (Academic Press, New York, 1988).

[26] R. Shadmehr and D.Z. D'Argenio, A comparison of a neural network based estimator and two statistical estimators in a sparse and noisy data environment, *Proc. IJCNN*, Washington DC (June 1990) I-289–292.

[27] P. Smyth and J. Mellstrom, Initial results on fault diagnosis of DSN antenna control assemblies using pattern recognition techniques, TDA Progress Report 42-101, Jet Propulsion Laboratory, Pasadena, California, 1990.

[28] H. Tong, *Non-Linear Time Series. A Dynamical System Approach* (Oxford Science Publ., Oxford, 1990).

[29] A.R. Webb, and D. Lowe, A theorem connecting adaptive feed-forward layered networks and nonlinear discriminant analysis, Royal Signals and Radar Establishment Memorandum 4209, August 1988, 26 pp.

[30] A.S. Weigend, B.A. Huberman and D.E. Rumelhart, Predicting the future: A connectionist approach, Stanford PDP Research Group technical report Stanford-PDP-90-01. *Internat. J. Neural Syst.*, in press.

Fionn Murtagh's Engineering Science, Computer Science and Mathematical Statistics degrees are from Dublin and Paris. He has worked in Italy and Ireland for the European Commission and as computer science faculty before his present position with the European Space Agency in Germany. Dr. Murtagh is a member of the Editorial Boards of the *Journal of Classification*, the *Classification Literature Automated Search Service* and *Les Cahiers de l'Analyse des Données*. He has published about 70 papers in journals and conference proceedings, and 6 books including *Knowledge-Based Systems in Astronomy* (edited with A. Heck, Springer-Verlag, 1989), Multivariate Data Analysis (with A. Heck, Kluwer, 1987) and *Multidimensional Clustering Algorithms* (Physica-Verlag, 1985).