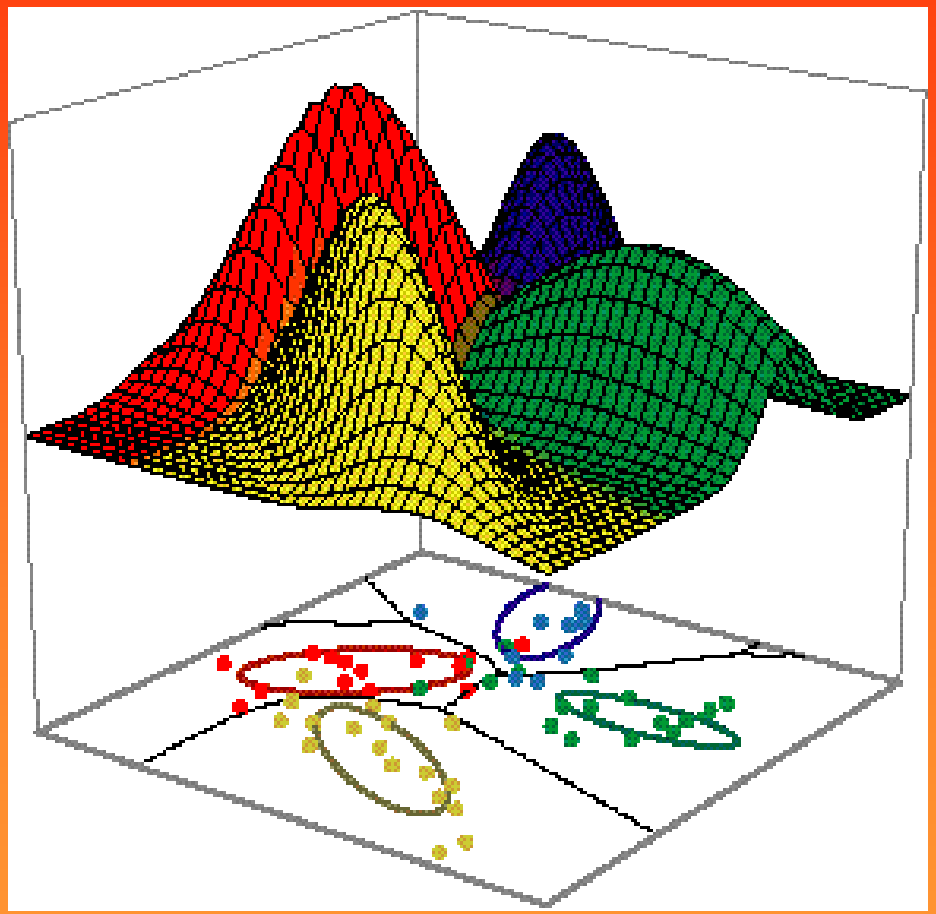
Multilayer Perceptron = FeedForward Neural Network

- History
- Definition
- Classification = feedforward operation
- Learning = "backpropagation"
  = local optimization in the space of weights



Pattern Classification

Figures in these slides were taken from *Pattern Classification (2nd ed) by R. O. Duda, P. E. Hart and D. G. Stork, John Wiley & Sons, 2000*
with the permission of the authors and the publisher

History

- Perceptron (Rosenblatt, 1956) with its simple learning algorithm generated a lot of excitement

- Minsky & Papert (1969) showed that even a simple XOR cannot be learnt by a perceptron, whichvlead to skepticism about their utility

- The problem was solved by “networks” of perceptrons (multi-layer perceptrons, feedforward neural networks)

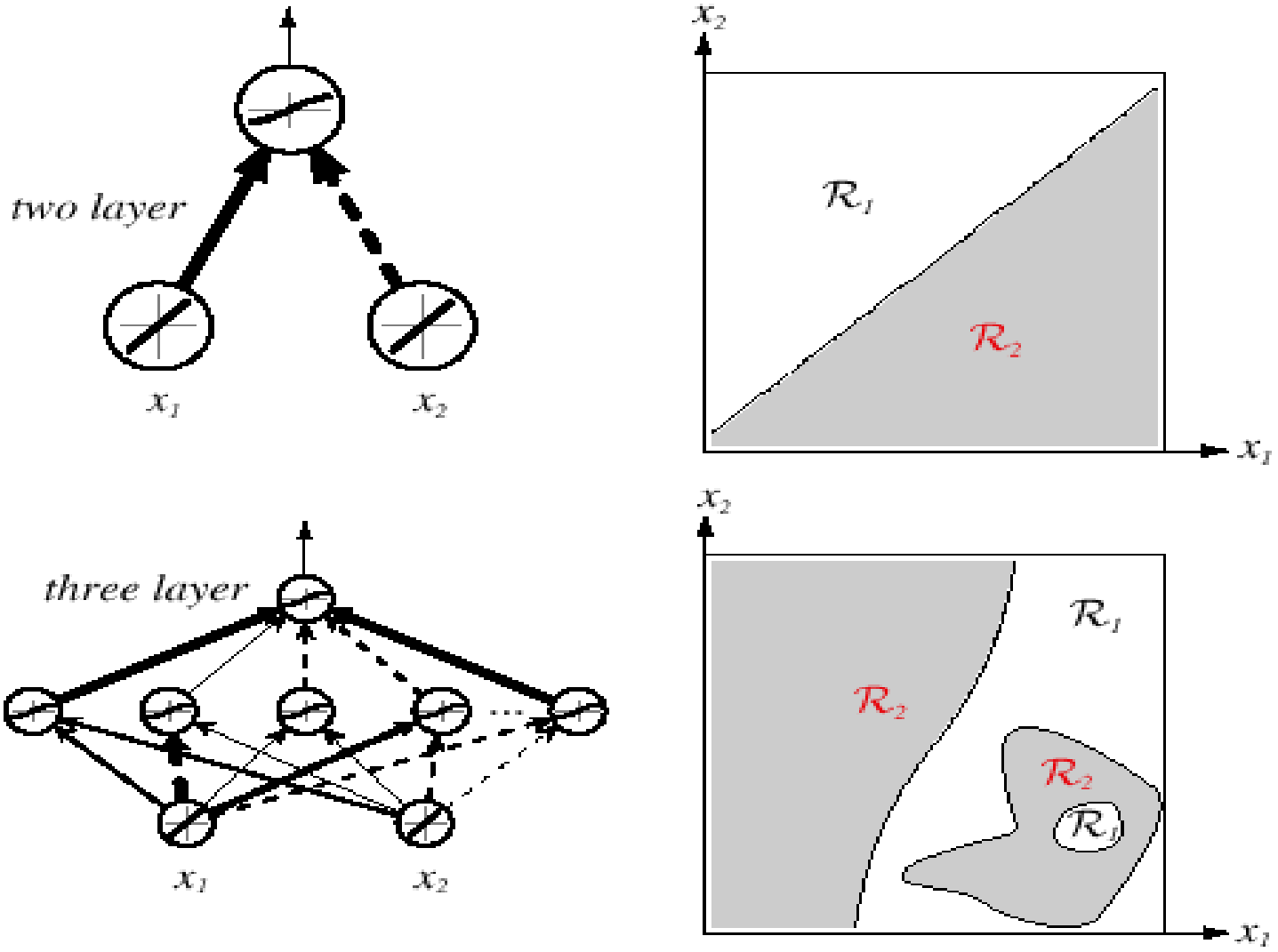
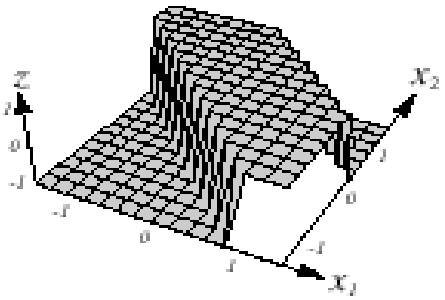
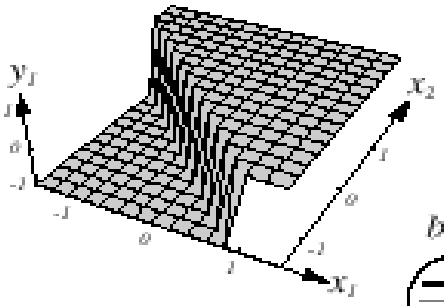
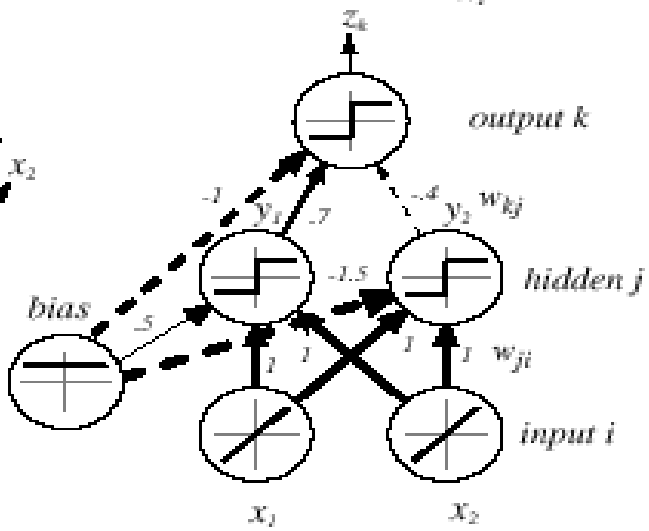


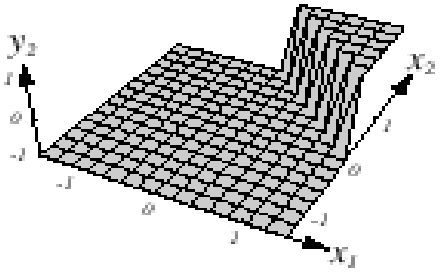
FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

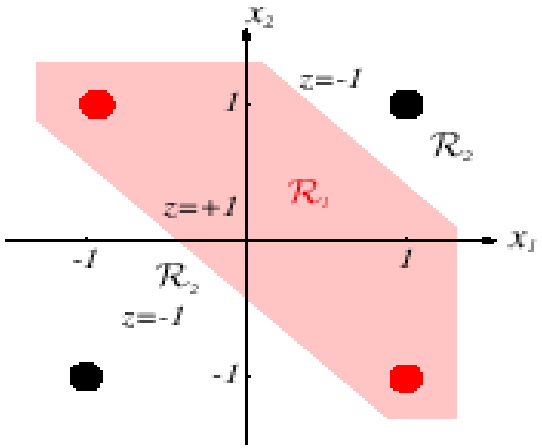
A three layer perceptron network implementing the XOR











Multilayer Perceptron, terminology

- Net activation:

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv w_j^t . x,$$

  where the subscript i indexes units in the input layer, *j* in the hidden; w_{ji} denotes the input-to-hidden layer weights at the hidden unit *j*.

- A single "bias unit" is connected to each unit other than the input units
- Each hidden unit emits an output that is a nonlinear function of its activation, that is: $y_j = f(net_j)$
- Each output unit similarly computes its net activation based on the hidden unit signals as:

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = w_k^t . y,$$

where the subscript *k* indexes units in the ouput layer and n_H denotes the number of hidden units

- The function *f(.),* the activation function, introduces nonlinearity the a unit. The first activation function considered was the sign (as in perceptron):

$$f(net) = sgn(net) \equiv \begin{cases} 1 & if \ net \geq 0 \\ -1 & if \ net < 0 \end{cases}$$

- Unfortunately, a network with such ***f*** is difficult to train

Multilayer Perceptron = FF Neural net

- Is a function of the following form

$$g_k(x) \equiv z_k = f\left(\sum_{j=1}^{n_H} w_{kj}\, f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}\right) \qquad (1)$$

$$(k = 1,\ldots,c)$$

- Popularized in 1986, backpropagation learning bacame possible for f continuous and differentiable

- We can allow the activation in the output layer to be different from the activation function in the hidden layer or have different activation for each individual unit

- We assume for now that all activation functions to be identical

- The vector of outputs is denoted z_k.

$$z_k = f(net_k)$$

- In the case of c outputs (classes), we can view the network as computing c discriminants functions

  $z_k = g_k(x)$ and classify the input x according to the largest discriminant function $g_k(x) \quad \forall\, k = 1, \ldots, c$

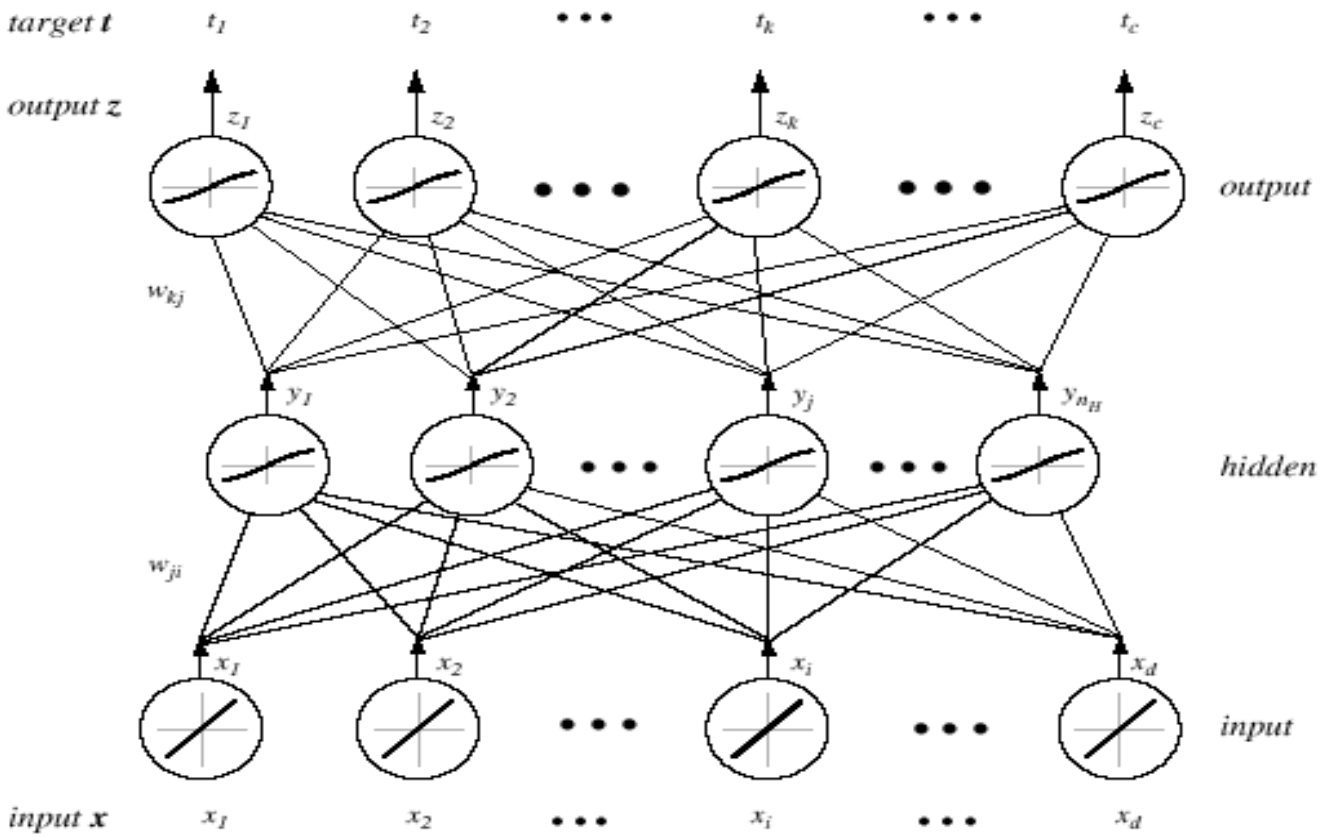


FIGURE 6.4. A d-n_H-c fully connected three-layer network and the notation we shall use. During feedforward operation, a d-dimensional input pattern $\mathbf{x}$ is presented to the input layer; each input unit then emits its corresponding component x_i. Each of the n_H hidden units computes its net activation, net_j, as the inner product of the input layer signals with weights w_{ji} at the hidden unit. The hidden unit emits $y_j = f(net_j)$, where $f(\cdot)$ is the nonlinear activation function, shown here as a sigmoid. Each of the c output units functions in the same manner as the hidden units do, computing net_k as the inner product of the hidden unit signals and weights at the output unit. The final signals emitted by the network, $z_k = f(net_k)$, are used as discriminant functions for classification. During network training, these output signals are compared with a teaching or target vector $\mathbf{t}$, and any difference is used in training the weights throughout the network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Expressive Power of multi-layer Networks

Question: Can every decision be implemented by a three-layer network described by equation (1) ?

Answer: Yes (due to A. Kolmogorov)

"Any continuous function from input to output can be implemented in a three-layer net, given sufficient number of hidden units n_H, proper nonlinearities, and weights."

$$g(x) = \sum_{j=1}^{2n+1} \delta_j \left(\Sigma \beta_{ij} (x_i) \right) \quad \forall x \in I^n (I = [0,1]; n \geq 2)$$

for properly chosen functions δ_j and β_{ij}

Expressive Power of multi-layer Networks

- Each of the *2n+1* hidden units δ_j takes as input a sum of *d* nonlinear functions, one for each input feature x_i
- Each hidden unit emits a nonlinear function δ_j of its total input
- Unfortunately: Kolmogorov's theorem is not constructive; tells us very little about how to find the nonlinear functions based on data; this is the central problem in network-based pattern recognition
- Remarkably: Kolmogorov's theorem proves that a continuous function in dimension n can be represented as a function of a sufficient number of one-dimensional functions

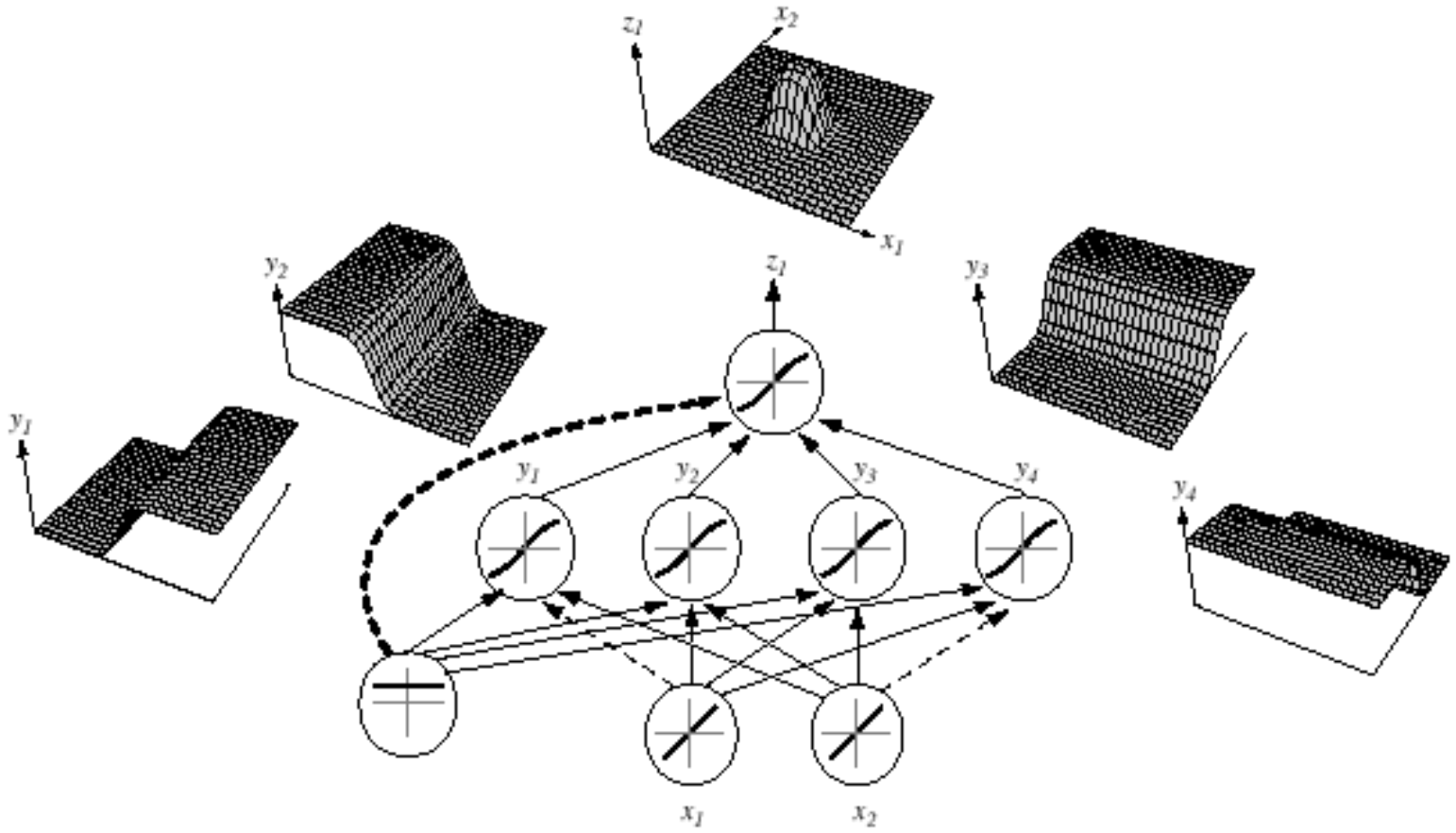


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Modes of operation of MLP

- Network have two modes of operation:

  - **Classfication = Feedforward operation**

    The feedforward operations consists of presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units (no cycles!)

  - **Learning**

    The supervised learning consists of presenting an input pattern and modifying the network parameters (weights) to reduce distances between the computed output and the desired output

Learning: Backpropagation Cost function

- is a gradient descent method. The cost function is, for a single training pattern:

$$J(w) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$$

- where t_k ***is*** k-th target (or desired) output and z_k be the k-th computed output
- with $k = 1, \ldots, c$ and w represents all the weights of the network
- Class k is represented as vector (0,0, ..1, ..., 0), where 1 is in the k-th position.
- For a "pure" output of the newural net, z=(0,..,1..,0), $J(w)$ is 0 or 1.

Learning: Update

- in gradient descent, the update of weights is in the direction of the gradient

$$\Delta w = -\eta \frac{\partial J}{\partial w}$$

- the step size is found e.g. by line search (a search for a minimum in 1D).
- $w_{t+1} = w_t + \Delta w$

Learning: Calculating the gradient

- Error on the hidden–to-output weights

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

where the sensitivity of unit k is defined as:

$$\delta_k = -\frac{\partial J}{\partial net_k}$$

and describes how the overall error changes with the activation of the unit's net

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

Learning: Calculating the gradient

Since $net_k = w_k^t.y$ therefore:

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

Conclusion: the weight update (or learning rule) for the hidden-to-output weights is:

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k)\, f'(net_k) y_j$$

- Error on the input-to-hidden units

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

Learning: Calculating the gradient

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\cdot\frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)\, f'(net_k)\, w_{kj}$$

Similarly as in the preceding case, we define the sensitivity for a hidden unit:

$$\delta_j \equiv f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

which means that: "The sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the hidden-to-output weights w_{kj}; all multipled by $f'(net_j)$" The learning rule for the input-to-hidden weights is:

$$\Delta w_{ji} = \eta x_i \delta_j = \eta \underbrace{\left[\Sigma w_{kj}\delta_k\right] f'(net_j)}_{\delta_j} x_i$$

Learning: Calculating the gradient

- Calculating the gradient for the full training set is a sum of gradients, since the cost function is additive

$$J = \sum_{p=1}^{n} J_p \qquad (1)$$

- Stopping criterion:
  - e.g. The algorithm terminates when the change in the criterion function J(w) is smaller than some preset value θ
  - when cross-validation error goes up

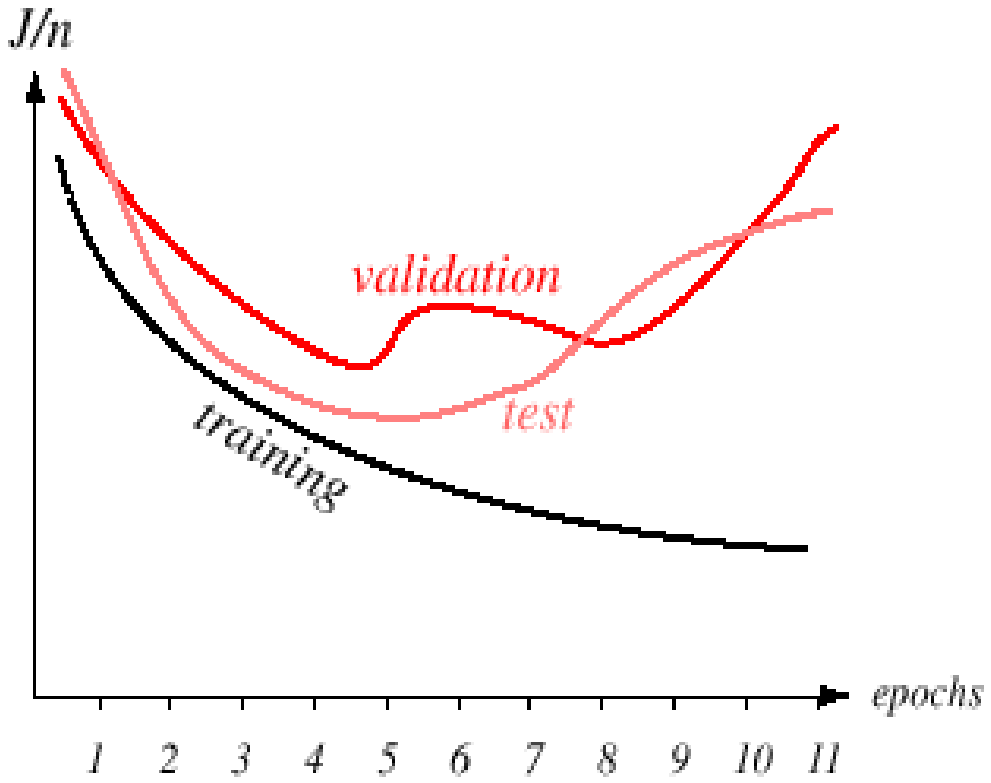


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^{n} J_p$. The validation error and the test or generalization error per pattern are virtually always higher than the training error. In some protocols, training is stopped at the first minimum of the validation set. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Learning: Calculating the gradient

- Starting with a pseudo-random weight configuration, the stochastic backpropagation algorithm can be written as shown in the box.
- The evalution of J is stochastic to speed up the process

Begin **initialize** n_H; w, criterion θ, η, m $\leftarrow$ 0

do m $\leftarrow$ m + 1

$x^m \leftarrow$ randomly chosen pattern

$w_{ji} \leftarrow w_{ji} + \eta\delta_j x_i$; $w_{kj} \leftarrow w_{kj} + \eta\delta_k y_j$

until $||\nabla J(w)|| < \theta$

return w

End

Disadvantages of MLP

- learning by local optimization: global minimum not guaranteed
- time-consuming learning. Many nested loops:
  - repeat for different number of hidden units
  - random restarts to deal with local minima
  - iterations of the backpropagation algorithm
- many parameters, not easy to set for a non-experienced users
- a cost-function (quadratic loss) which is convenient, but not the classification error

Advantages of MLP

- handles well problems with multiple classes
- handles naturally both classification and regression problems (estimating real values)
- after normalization, the output may be interpreted as aposteriori probability. We know the class with maximum response, but also the reliability of it.

Thank you