CpE 646 Pattern Recognition and Classification

Prof. Hong Man

Department of Electrical and Computer Engineering
Stevens Institute of Technology

STEVENS Institute of Technology

Visual Information Environment Laboratory

Chapter 6: Multilayer Neural Networks

Chapter 6 (Section 6.1 – 6.3, 6.6):

- Introduction
- Feedforward Operation and Classification – classification
- Backpropagation Algorithm – training
- Practical considerations

Introduction

- There are many problems for which linear discriminants are insufficient for minimum error
- As seen in previous chapter, with a clever choice of nonlinear φ function, we can obtain arbitrary decision regions. But the central difficulty was the choice of the appropriate nonlinear functions
- A “brute” approach might be to select a complete basis set such as all polynomials; such a classifier would require too many parameters to be determined from a limited number of training samples
- Prior knowledge may guide us in finding a good φ function, but it is not automatic and not general.

Introduction

- Multilayer Neural Networks or Multilayer Perceptrons:
  - Multilayer Neural networks implement linear discriminants, but in a space where the inputs have been mapped nonlinearly.
  - The parameters governing the nonlinear mapping are learned at the same time as those governing the linear discriminant.
  - The learning algorithms are fairly simple, yet effective in many real-world applications.

Introduction

- One of the most popular methods for training such multilayer networks is based on gradient descent in error – the backpropagation algorithm
  - The conceptual and algorithmic simplicity of backpropagation makes neural networks a flexible heuristic technique for adaptive pattern recognition with complicated models (hundreds of parameters).

Feedforward Operation and Classification

- A three-layer neural network consists of an input layer, a hidden layer and an output layer interconnected by modifiable weights represented by links between layers
- A single bias unit is connected to each unit other than the input units
- The function of units is loosely based on properties of biological neurons, therefore they may be called neurons
- In pattern recognition applications, the input units represent the components of a feature vector, and the signal emitted by the output units are the values of the discriminant functions used for classification

Feedforward Operation and Classification

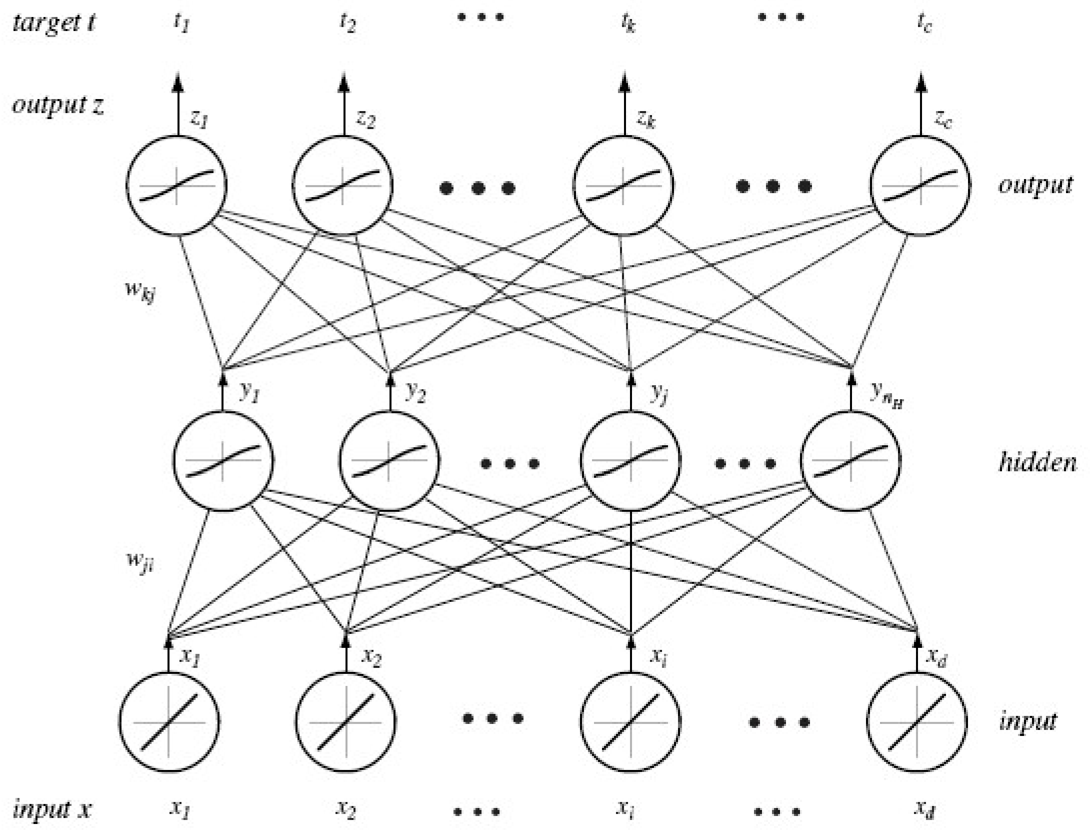

FIGURE 6.4. A d-n_H-c fully connected three-layer network and the notation we shall use. During feedforward operation, a d-dimensional input pattern $\mathbf{x}$ is presented to the input layer; each input unit then emits its corresponding component x_i. Each of the n_H

Feedforward Operation and Classification

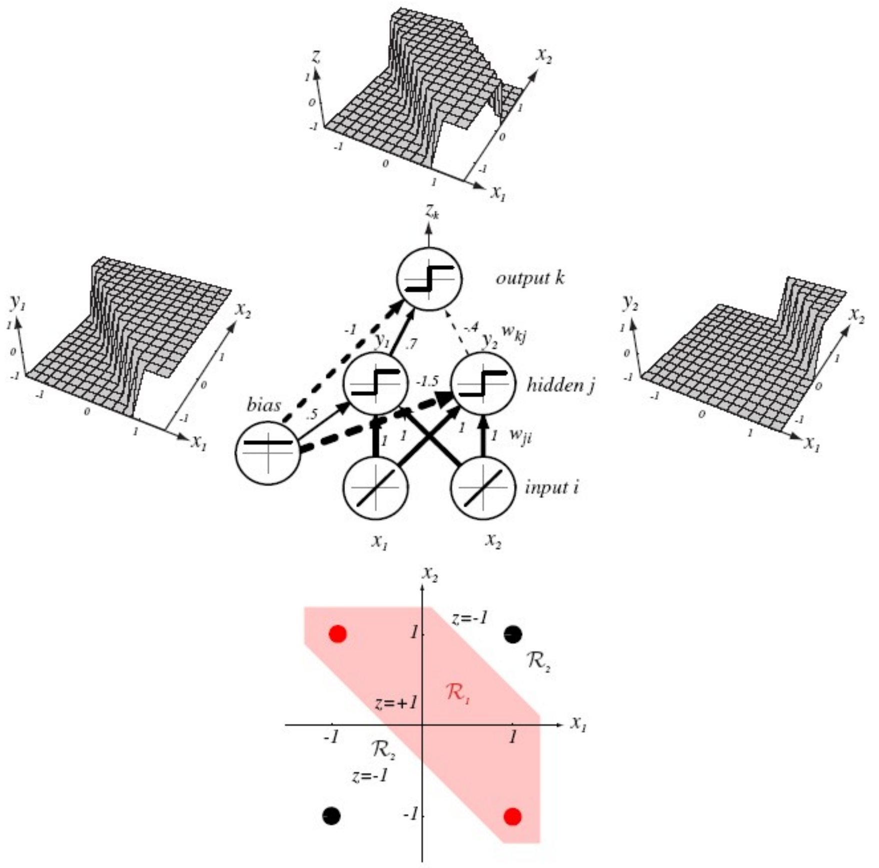

FIGURE 6.1. The two-bit parity or exclusive-OR problem can be solved by a three-layer network. At the bottom is the two-dimensional feature x_1x_2-space, along with the four patterns to be classified. The three-layer network is shown in the middle. The input units are linear and merely distribute their feature values through multiplicative weights

Feedforward Operation and Classification

- Example: The exclusive-OR (XOR) problem
  - Each two-dimensional input vector is presented to the input layer
  - Each hidden unit computes the weighted sum of its inputs from input units and bias to form its scalar net activation, or *net*

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{w}_j^t \mathbf{x}$$

where i is input unit (feature component) index, j is hidden unit index, w_{ji} is the weight (also called synapse) between i-th input unit and j-th hidden unit, w_{j0} is bias and $x_0=1$.

Feedforward Operation and Classification

– Each hidden unit emits an output that is a nonlinear function of its activation,

$$y_j = f(net_j)$$

- In this example the function is a simple threshold or sign function

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

- The function $f(\cdot)$ is also called the activation function or "nonlinearity" of a unit. There are more general activation functions with desirables properties. This function can serve as φ function

Feedforward Operation and Classification

– Each output unit similarly computes its net activation based on the hidden unit signals as:

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{w}_k^t \mathbf{y}$$

where y_j is output of hidden unit j, n_H denotes the number of hidden units, k is output unit index, and w_{k0} is bias and $y_0=1$.

Feedforward Operation and Classification

- An output unit computes the nonlinear function of its net, emitting

$$z_k = f(net_k)$$

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

    $z_k = g_k(y)$ and classify the input $\boldsymbol{y}$ according to the largest discriminant function $g_k(y) \;\; \forall\, k = 1, \ldots, c$

Feedforward Operation and Classification

- The three-layer network with the weights listed in Fig. 6.1 solves the XOR problem
  - The hidden unit y_1 computes the boundary:

$$x_1 + x_2 + 0.5 \begin{cases} \geq 0 \Rightarrow y_1 = +1 \\ < 0 \Rightarrow y_1 = -1 \end{cases}$$

  - The hidden unit y_2 computes the boundary:

$$x_1 + x_2 - 1.5 \begin{cases} \geq 0 \Rightarrow y_2 = +1 \\ < 0 \Rightarrow y_2 = -1 \end{cases}$$

Feedforward Operation and Classification

- The final output unit emits

  $z_1 = +1$ if and only if $y_1 = +1$ AND $y_2 = -1$

  In computer logic

  $z_k = y_1$ AND NOT $y_2 = (x_1$ OR $x_2)$ AND NOT $(x_1$ AND $x_2)$

  $= x_1$ XOR x_2

Feedforward Operation and Classification

- General Feedforward Operation
  - Nonlinear multilayer networks involve input units, hidden units and output units.
  - Hidden units enable us to express more complicated nonlinear functions and thus extend the classification – greater expressive power
  - For *c*-category problem, the discriminant function at each output unit is

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

Feedforward Operation and Classification

- The activation function does not have to be a sign function, it is often required to be 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

Feedforward Operation and Classification

- Expressive power of multi-layer networks
  - Any continuous function from input to output (in particular any posterior probability) can be implemented in a three-layer network, given sufficient number of hidden units n_H, proper nonlinearities, and weights.
    - A. Kolmogorov proved that any continuous function $g(x)$ defined on the unit hypercube I^n ($I=[0,1]$, $n\geq 2$) can be represented in the form

$$g(x) = \sum_{j=1}^{2n+1} \Xi_j \left(\sum_{i=1}^{d} \psi_{ij}(x_i) \right)$$

Feedforward Operation and Classification

- Each of the $2n+1$ hidden units takes as input a sum of d nonlinear functions Ψ_{ij}, one for each input feature x_i
- Each hidden unit emits a nonlinear function Ξ_j of its total input
- The output unit emits the sum of the contributions of the hidden units
- Unfortunately Kolmogorov's theorem tells us very little about how to find the nonlinear functions Ξ_j and Ψ_{ij} based on data; this is the central problem in network-based pattern recognition

(Note: in Kolmogorov's theorem, nonlinear functions Ξ_j and Ψ_{ij} are before the summations, in standard 3-layer NN, the nonlinear functions are after the summations.)

Feedforward Operation and Classification

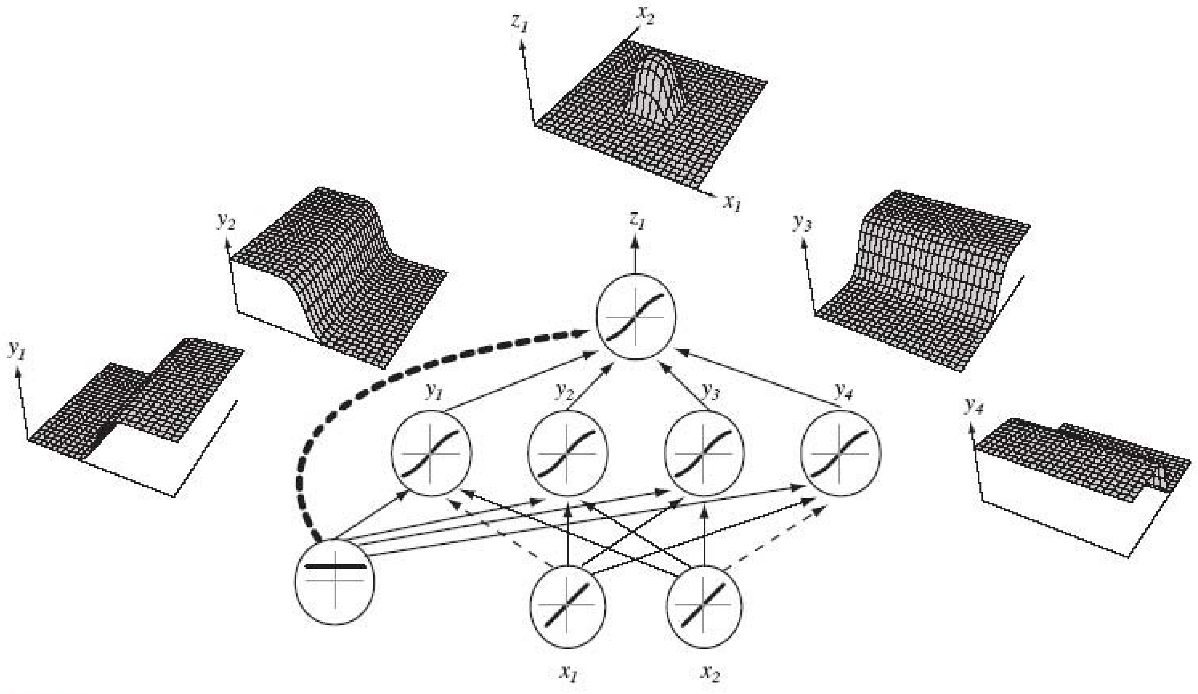

FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large 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.

Feedforward Operation and Classification

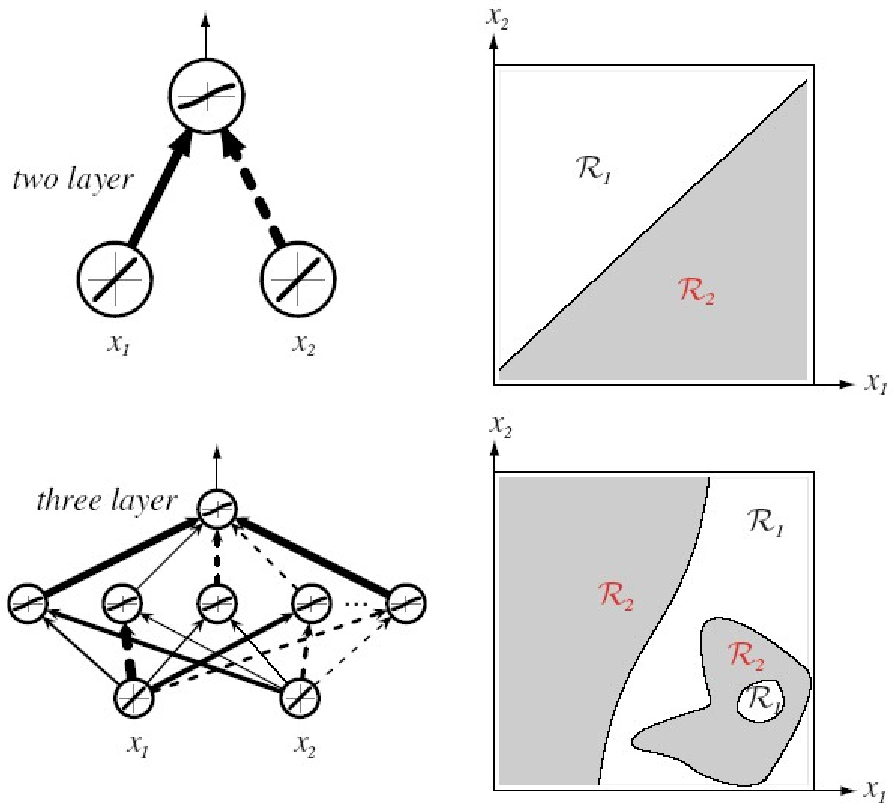

FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O.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

Backpropagation Algorithm

- Any function from input to output can be implemented as a three-layer neural network
- The goal now is to set the interconnection weights based on the training patterns and the desired outputs
- In a three-layer network, it is a straightforward matter to understand how the output, and thus the error, depend on the hidden-to-output layer weights (Chapter 5, Figure 5.1)
- The power of backpropagation is that it enables us to compute an effective error for each hidden unit, and thus derive a learning rule for the input-to-hidden weights, this is known as the credit assignment problem

Backpropagation Algorithm

- Network have two modes of 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) – classification (pp. 6-19)
  - 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 or target value

Backpropagation Algorithm

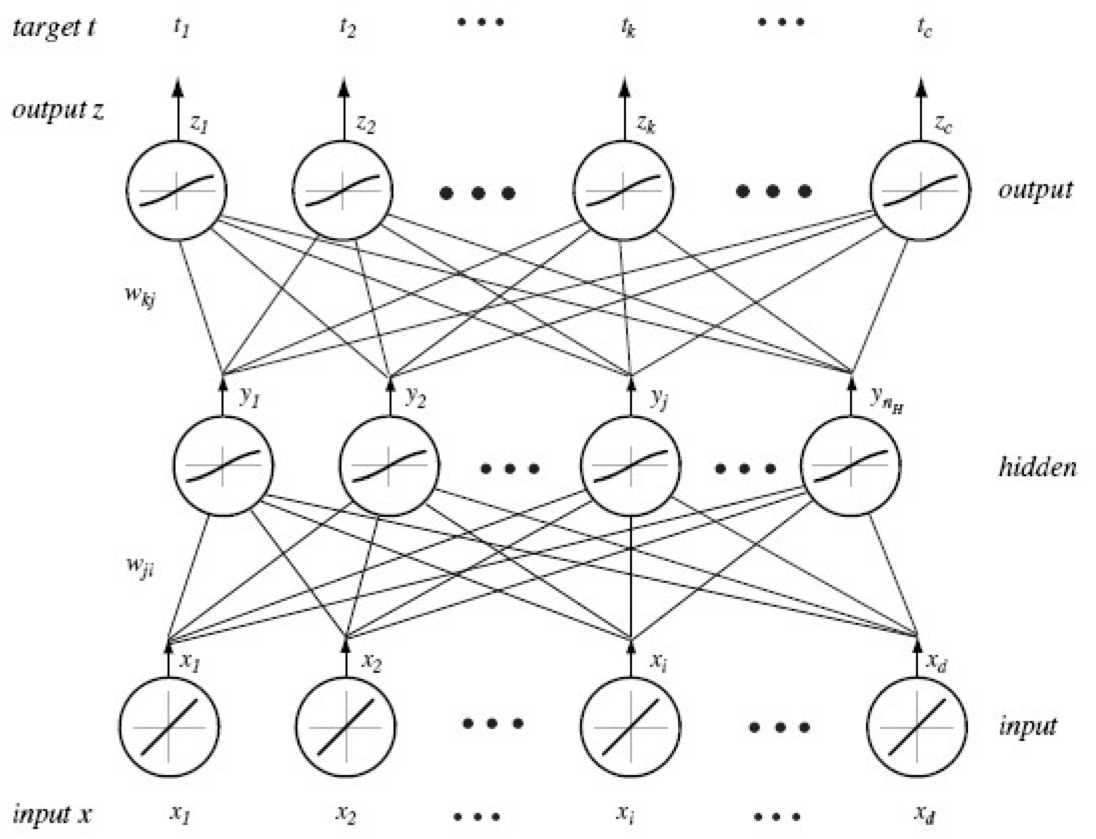

FIGURE 6.4. A d-n_H-c fully connected three-layer network and the notation we shall use. During feedforward operation, a d-dimensional input pattern $\mathbf{x}$ is presented to the input layer; each input unit then emits its corresponding component x_i. Each of the n_H

Backpropagation Algorithm

- Network learning
  - Procedure:
    - Start with an untrained network
    - Present a training pattern to the input layer
    - Pass signal through the net and determine the output
    - The output is compared with the target value, the difference is error. The error or criterion function is some scalar function of weights.
    - The weights are adjusted to minimize this error

Backpropagation Algorithm

- Let t_k be the k-th target (or desired) output and z_k be the k-th computed output, $\boldsymbol{t}=\{t_k\}$ and $\boldsymbol{z}=\{z_k\}$ with $k = 1, \ldots, c$, and $\boldsymbol{w}$ represents all the weights of the network
- The training error:

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

(Note: we use k as index for output units, j as index for hidden units and i as index for input units.)

Backpropagation Algorithm

- The backpropagation learning rule is based on gradient descent
  - The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

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

where η is the learning rate which indicates the relative size of the change in weights

$$\boldsymbol{w}(m+1) = \boldsymbol{w}(m) + \Delta\boldsymbol{w}(m)$$

where m corresponds to the m-th pattern presented

Backpropagation Algorithm

– Error on the hidden–to-output weights

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

where the sensitivity of output 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*. This is related to the non-linear function $f(\cdot)$

Backpropagation Algorithm

- Since $z_k = f(net_k)$, assume $f(\cdot)$ is differentiable

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

$$= (t_k - z_k) f'(net_k)$$

- Since $net_k = \mathbf{w}_{\mathbf{k}}^t\mathbf{y}$ therefore: $\dfrac{\partial net_k}{\partial w_{kj}} = y_j$

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

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

Backpropagation Algorithm

– Error on the input-to-hidden units

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

- The first term on the right hand side

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

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

Backpropagation Algorithm

- The second term, since $y_j = f(net_j)$, $\frac{\partial y_j}{\partial net_j} = f'(net_j)$
- The third term, since $net_j = \mathbf{w_j}^t \mathbf{x}$, $\frac{\partial net_j}{\partial w_{ji}} = x_i$
- We have

$$\Delta w_{ji} = -\eta \frac{\partial J}{\partial w_{ji}} = -\eta \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

$$= \eta \left(\sum_{k=1}^{c} (t_k - z_k) f'(net_k) w_{kj} \right) f'(net_j) x_i$$

$$= \eta \left(\sum_{k=1}^{c} w_{kj} \delta_k \right) f'(net_j) x_i$$

Backpropagation Algorithm

- 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 multiplied by $f'(net_j)$ of the hidden unit.

- Therefore the learning rule for the input-to-hidden weights is:

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

Backpropagation Algorithm

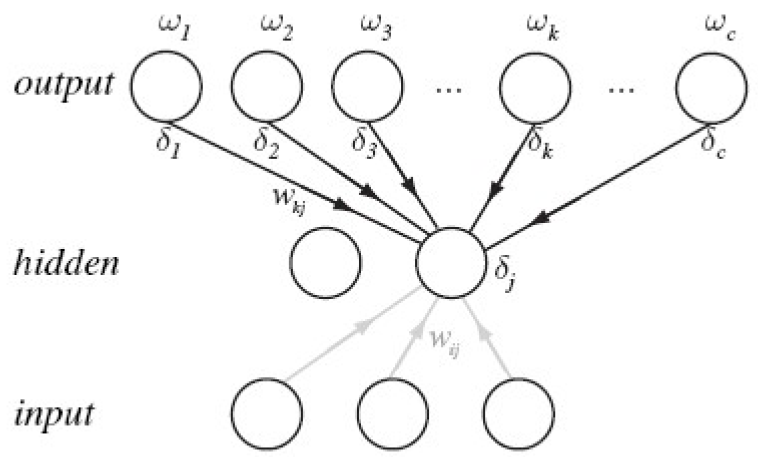

FIGURE 6.5. The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units: $\delta_j = f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$. The output unit sensitivities are thus propagated "back" to the hidden units. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Backpropagation Algorithm

- Training protocols
  - Supervised training consists in presenting to the network those patterns whose category label are known, i.e. the training set
  - Training process is to find the output of the net and adjust the weights so as to make the actual output more like the desired target value

Backpropagation Algorithm

- Three most useful training protocols:
  - Stochastic training: patterns are chosen randomly from the training set, and the weights are updated for each pattern. Patterns can be repeatedly used
  - Batch training: all (or many) patterns are presented to the network before learning take place
  - Online training: each pattern is presented once and only once, and there is no use of memory for storing the patterns
- The overall amount of pattern presentation is described in epoch, one epoch corresponds to a single presentations of all training patterns.

Backpropagation Algorithm

- All training starts with a pseudo-random weight configuration
- The stochastic backpropagation algorithm can be written as:

```
Begin initialize $n_H$, $\mathbf{w}$, criterion $\theta$, $\eta$, $m \leftarrow 0$
          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 $\mathbf{w}$
End
```

Backpropagation Algorithm

– In the on-line backpropagation algorithm, line 3

  ($x^m \leftarrow$ randomly chosen pattern)

  is replaced by

  ($x^m \leftarrow$ sequentially chosen pattern)

Backpropagation Algorithm

– The batch backpropagation algorithm can be expressed as

```
Begin initialize $n_H$, $\mathbf{w}$, criterion $\theta$, $\eta$, $r \leftarrow 0$
    do $r \leftarrow r + 1$ (increment epoch)
        $m \leftarrow 0$;  $\Delta w_{ji} \leftarrow 0$; $\Delta w_{kj} \leftarrow 0$;
            do $m \leftarrow m + 1$
                $x^m \leftarrow$ select pattern
                $\Delta w_{ji} \leftarrow \Delta w_{ji} + \eta\delta_j x_i$; $\Delta w_{kj} \leftarrow \Delta w_{kj} + \eta\delta_k y_j$
            until $m=n$ (complete an epoch)
            $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$ ; $w_{kj} \leftarrow w_{kj} + \Delta w_{kj}$ ;
    until $\|\nabla J(w)\| < \theta$
  return $\mathbf{w}$
End
```

Backpropagation Algorithm

– Stopping criterion

- The algorithm terminates when the change in the criterion function $J(\mathbf{w})$ is smaller than some preset value θ
- There are other stopping criteria that lead to better performance than this one
- So far, we have considered the error on a single pattern, but we want to consider an error defined over the entirety of patterns in the training set
- The total training error is the sum over the errors of n individual patterns $J = \sum_{p=1}^{n} J_p$

Backpropagation Algorithm

- A weight update may reduce the error on the single pattern being presented but can increase the error on the full training set
- However, given a large number of such individual updates, the total error of equation decreases

Backpropagation Algorithm

- Learning Curves
  - Before training starts, the error on the training set is high; through the learning process, the error becomes smaller
  - The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network
  - The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase

Backpropagation Algorithm

- To state the performance of the fielded network, we use the test set
- To decide when to stop training, we use a validation set
  - we do not want to overfit the network and decrease the power of the classifier generalization
  - we stop training at a minimum of the error on the validation set

Backpropagation Algorithm

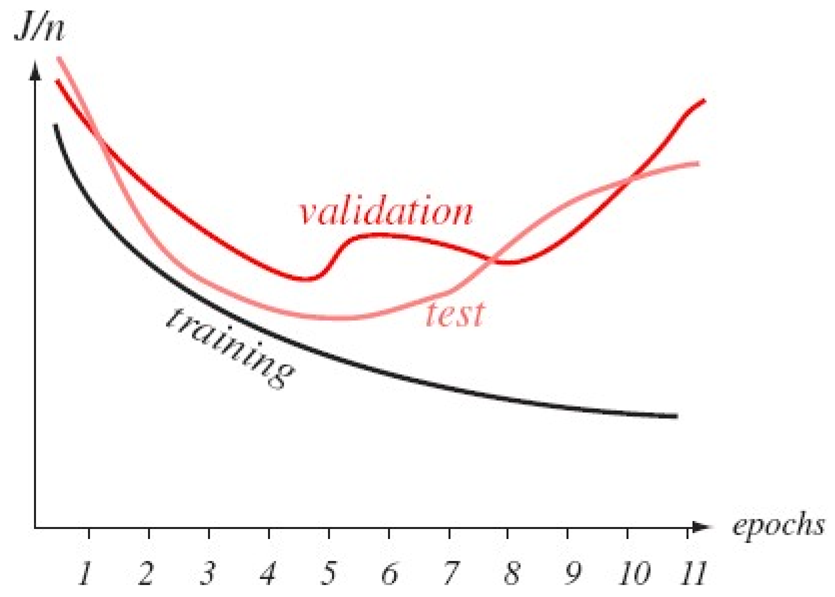

FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n\sum_{p=1}^{n} J_p$. The validation error and the test or generalization 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.

Practical Considerations

- Activation function
  - Backpropagation will work with virtually any activation function given some continuity conditions.
  - However specific problem may select particular function,
    - e.g. a Gaussian activation function may be a good choice for classification of Gaussian mixture models.

Practical Considerations

- Desirable properties of activation function
  - $f(\cdot)$ should be nonlinear.
    - If it is linear, it becomes linear discriminant classifier
  - $f(\cdot)$ should saturate, i.e. the function has finite maximum and minimum output vales.
    - This keeps weights and activation values bounded.
  - $f(\cdot)$ and its derivative $f'(\cdot)$ should be continuous.
    - The backpropagation algorithm relies on it.
  - Monotonicity is non-essential but convenient
    - $f'(\cdot)$ will have the same sign for all input values
    - Single maxima and minima

Practical Considerations

- A popular activation functions
  - One class of function that has all the properties is the sigmoid, such as a hyperbolic tangent.
    - The sigmoid is smooth, differentiable, nonlinear, and saturating.
    - Its derivate $f'(\cdot)$ can be expressed in terms of $f(\cdot)$.
    - A hidden layer of sigmoidal units affords a distributed representation, i.e. a particular input $\boldsymbol{x}$ is likely to yield activity through several hidden units.
      - If some activation function only has significant response for inputs within a small range, fewer hidden units will be active for an input $\boldsymbol{x}$ – local representation.

Practical Considerations

- Parameters for the Sigmoid
  - A good sigmoid function is defined as

$$f(net) = a \cdot \tanh(b \cdot net) = a \left[\frac{e^{+b \cdot net} - e^{-b \cdot net}}{e^{+b \cdot net} + e^{-b \cdot net}} \right]$$

  - The overall range and slope are not important, because it is their relationship to learning rate, input magnitude and targets that affect learning
  - For convenience we choose a=1.716, b=2/3,
    - $f'(0) \approx 0.5$
    - $f(net)$ is close to linear in $-1 < net < +1$
    - extrema of $f''(net)$ occur roughly at $net \approx \pm 2$.

Practical Considerations

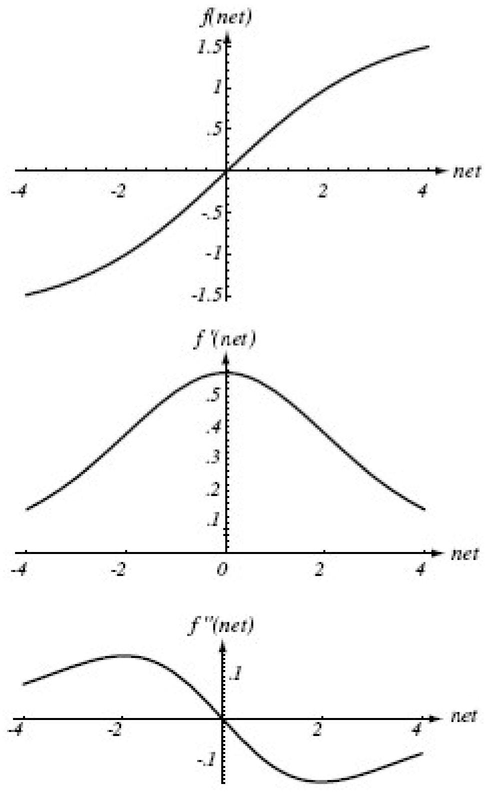

FIGURE 6.14. A useful activation function $f(net)$ is an anti-symmetric sigmoid. For the parameters given in the text, $f(net)$ is nearly linear in the range $-1 < net < +1$ and its second derivative, $f''(net)$, has extrema near $net \simeq \pm 2$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Practical Considerations

- Scaling input
  - When the values of the feature vector components are significantly different, the larger values will be dominant the weight updates.
  - The input vectors should be standarized before the training of the network
    - Over all training vectors, each feature, (a component of the feature vector), should be normalized with zero-mean and same variance, which is usually 1.0.
  - Standarization can only be done for stochastic and batch learning protocols, it is not possible for on-line learning protocol (full data set is not available).

Practical Considerations

- Target values
  - The target value should not be set at saturate value, i.e. ±1.716. This will drive the net and the weights to ∞
  - The target value can be set to +1 for the target category and -1 for non-target categories.
    - For example, in a 4-category problem, if a pattern is in category ω_3 , the target vector should be $\boldsymbol{t}$=[-1, -1, +1, -1]^t.
    - The outputs here do not represent *a posterior* probabilities.

Practical Considerations

- Number of hidden units
  - The number if hidden units, n_H, governs the expressive power of the network, and the complexity of the decision boundary
    - If the patterns are well separated or linearly separable, few hidden nodes are needed.
    - If the densities are highly interspersed, more hidden nodes are needed.
    - Without data information, there is no foolproof method for setting the number of hidden nodes before training

Practical Considerations

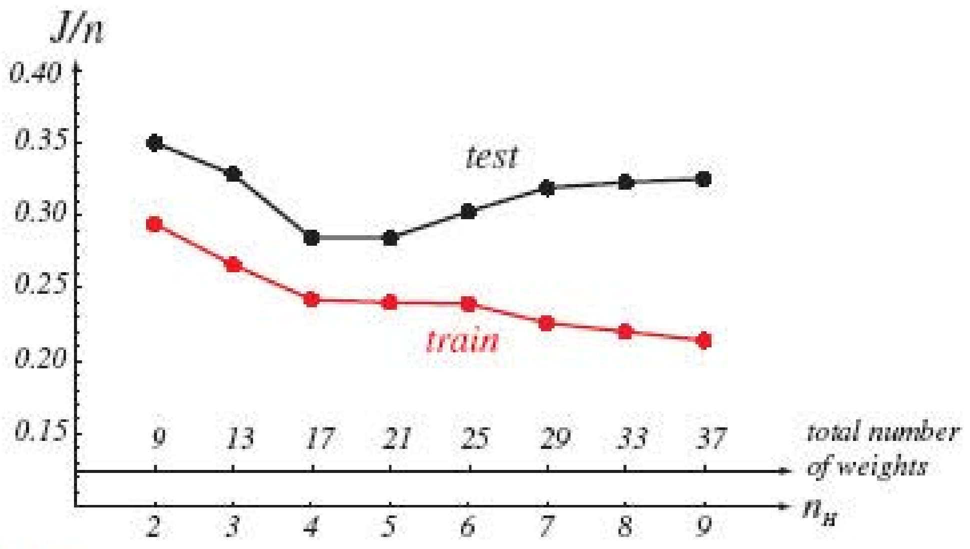

FIGURE 6.15. The error per pattern for networks fully trained but differing in the numbers of hidden units, n_H. Each $2 - n_H - 1$ network with bias was trained with 90 two-dimensional patterns from each of two categories, sampled from a mixture of three Gaussians, and thus $n = 180$. The minimum of the test error occurs for networks in the range $4 \leq n_H \leq 5$, i.e., the range of weights 17 to 21. This illustrates the rule of thumb that choosing networks with roughly $n/10$ weights often gives low test error. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Practical Considerations

- Extreme cases
  - Too few hidden nodes: the network does not have enough free parameter to fit the training data, the test error will be high
  - Too many hidden nodes: the network fits too well with the training data, and may not perform well with test data – overfitting.

Practical Considerations

- The number of hidden units determines the number of weights in the network, and indicates the degrees of freedom of the network
  - We should not have more weights than the number of training points, n.
  - A simple rule, n_H should be set such that the number of weights is roughly $n/10$.
  - Many successful systems employ more than this number.
  - A more principled method is to start with large n_H, and eventually prune, or eliminate, weights.

Practical Considerations

- Initializing weights
  - Given a fixed network topology, we seek weight initialization that provide fast and uniform learning, i.e. all weights reaches their final equilibrium values at about the same time.
  - In each given layer, we can choose weights randomly from a single distribution to help ensure uniform learning
  - Because of data standarization, the weights can also equally be positive and negative

Practical Considerations

– We can choose weights from a uniform distribution

$$-\tilde{\mathbf{w}} < \mathbf{w} < +\tilde{\mathbf{w}}$$

- If $\tilde{\mathbf{w}}$ is too small, the *net* values will be small and *f*(*net*) will operate at linear range
- If $\tilde{\mathbf{w}}$ is too big, the *net* values will be big and *f*(*net*) will become saturate easily.
- We may select $\tilde{\mathbf{w}}$ such that the net activation at a hidden unit is in the range -1 < *net* < +1. This yields $-1/\sqrt{d} < \mathbf{w}_{ji} < +1/\sqrt{d}, \quad -1/\sqrt{n_H} < \mathbf{w}_{kj} < +1/\sqrt{n_H}$ where d is the dimension of the input feature vector

Practical Considerations

- Learning rate
  - Theoretically, a sufficiently small learning rate is able to ensure convergence.
  - Practically networks are rarely trained fully to a training error minimum, learning rate will affect the performance of the final network

Practical Considerations

– The optimal learning rate is the one that can reach the local minimum in one step
  - If the criterion function can be approximated by a quadratic function, i.e.

$$\frac{\partial^2 J}{\partial w^2}\Delta w = \frac{\partial J}{\partial w}$$

Then the optimal rate is

$$\eta_{opt} = \left(\frac{\partial^2 J}{\partial w^2}\right)^{-1}$$

Practical Considerations

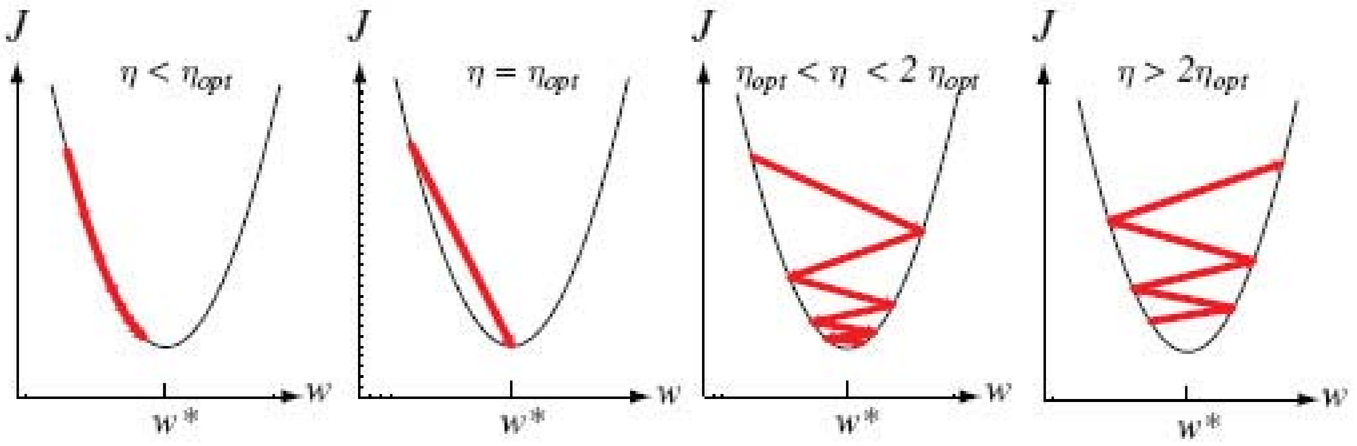

FIGURE 6.16. Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Practical Considerations

- To achieve rapid and uniform learning, we should calculate the second derivative of the criterion function with respect to each weight and set the optimal learning rate separately for each weight.
- The maximum learning rate that will lead to convergence is $\eta_{max} = 2\eta_{opt}$.
- A rate in the range $\eta_{opt} < \eta < 2\eta_{opt}$ will lead to slow convergence.
- For typical problems with sigmoidal networks and parameters discussed previously, $\eta \approx 0.1$ is often a first choice.