

CSE 569 Fundamentals of Statistical Learning & Pattern Recognition

Multilayer Neural Networks

- Introduction
- Back-propagation algorithm
- Back-propagation as feature mapping
 - Convolutional neural networks
- Bayesian interpretation
- Techniques for improving the BP algorithm
- Some other neural networks: RBF networks, Kohonen maps

One Slide from last chapter: A Simple Linear Classifier: Implementation

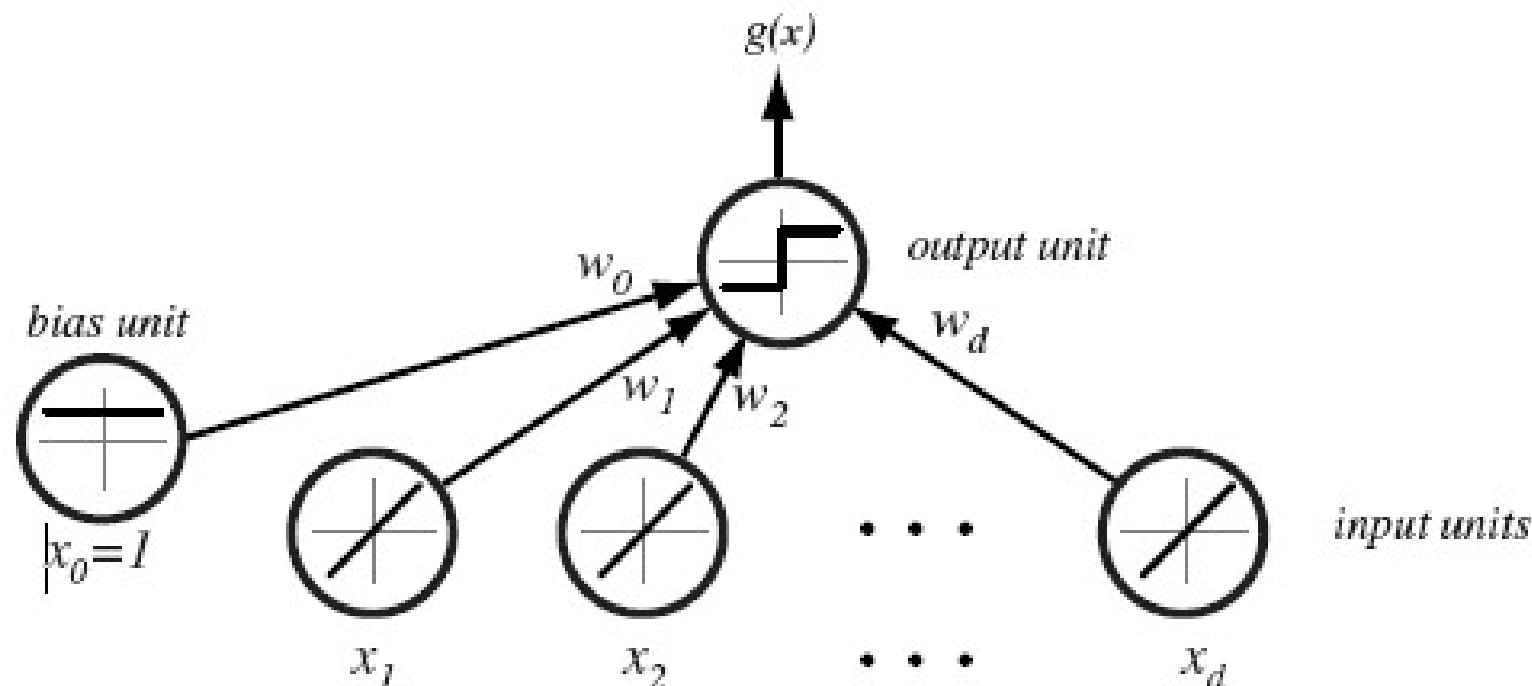


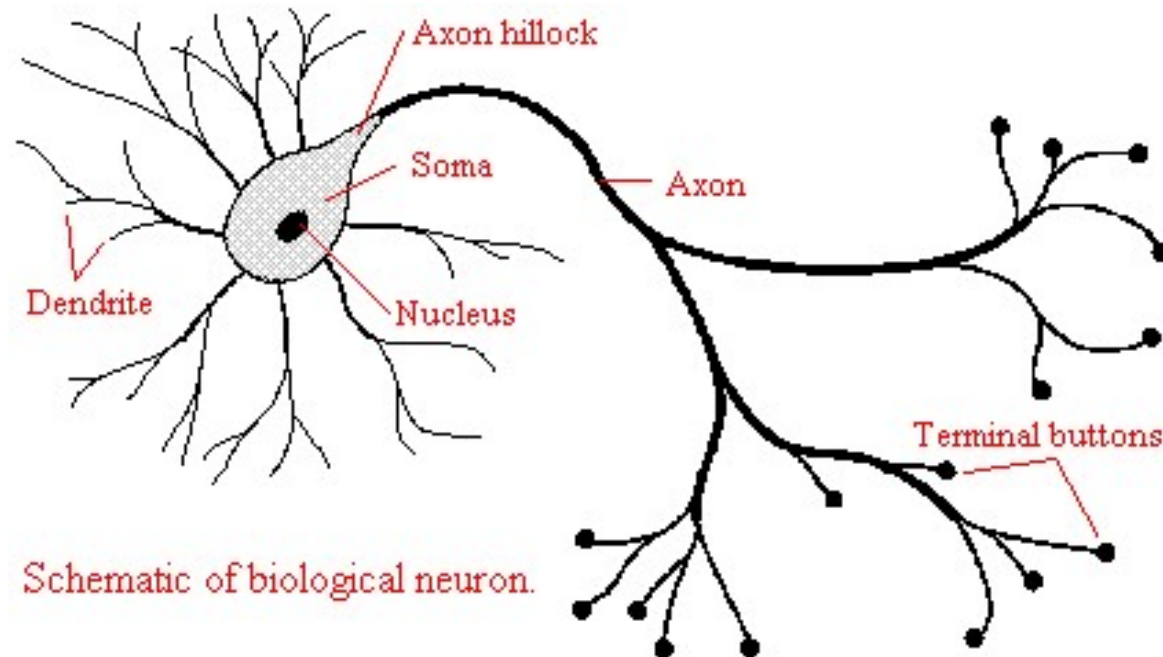
FIGURE 5.1. A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i ; the effective input at the output unit is the sum all these products, $\sum w_i x_i$. We show in each unit its effective input-output function. Thus each of the d input units is linear, emitting exactly the value of its corresponding feature value. The single bias unit unit always emits the constant value 1.0. The single output unit emits a $+1$ if $\mathbf{w}^t \mathbf{x} + w_0 > 0$ or a -1 otherwise. From: Richard O. Duda, Peter E. Hart,

Introduction

- Many problems are not linearly separable.
 - We have seen that using a nonlinear mapping could help (e.g., as in SVM).
 - Need to choose a proper mapping (or, implicitly, a kernel).
- It is desirable if the nonlinear mapping and the linear discriminant functions can both be learned from training data.
 - Want to get this nonlinearity by simply stacking layers of simple algorithms such as the perceptron.
 - Multilayer Perceptrons (MLP) (or multilayer neural network):
We may think them as multiple layers of LMS algorithms.
 - LMS differs from the perceptron and relaxation procedures in that all samples are used in updating, with a target (desired) vector \mathbf{b} .

Biological Analogy

- A typical neuron:



- Neuroscience for kids:
 - <http://faculty.washington.edu/chudler/neurok.html>

MLP: Basic Structure and Notations

- A three-layer MLP
 - Solve the XOR problem:

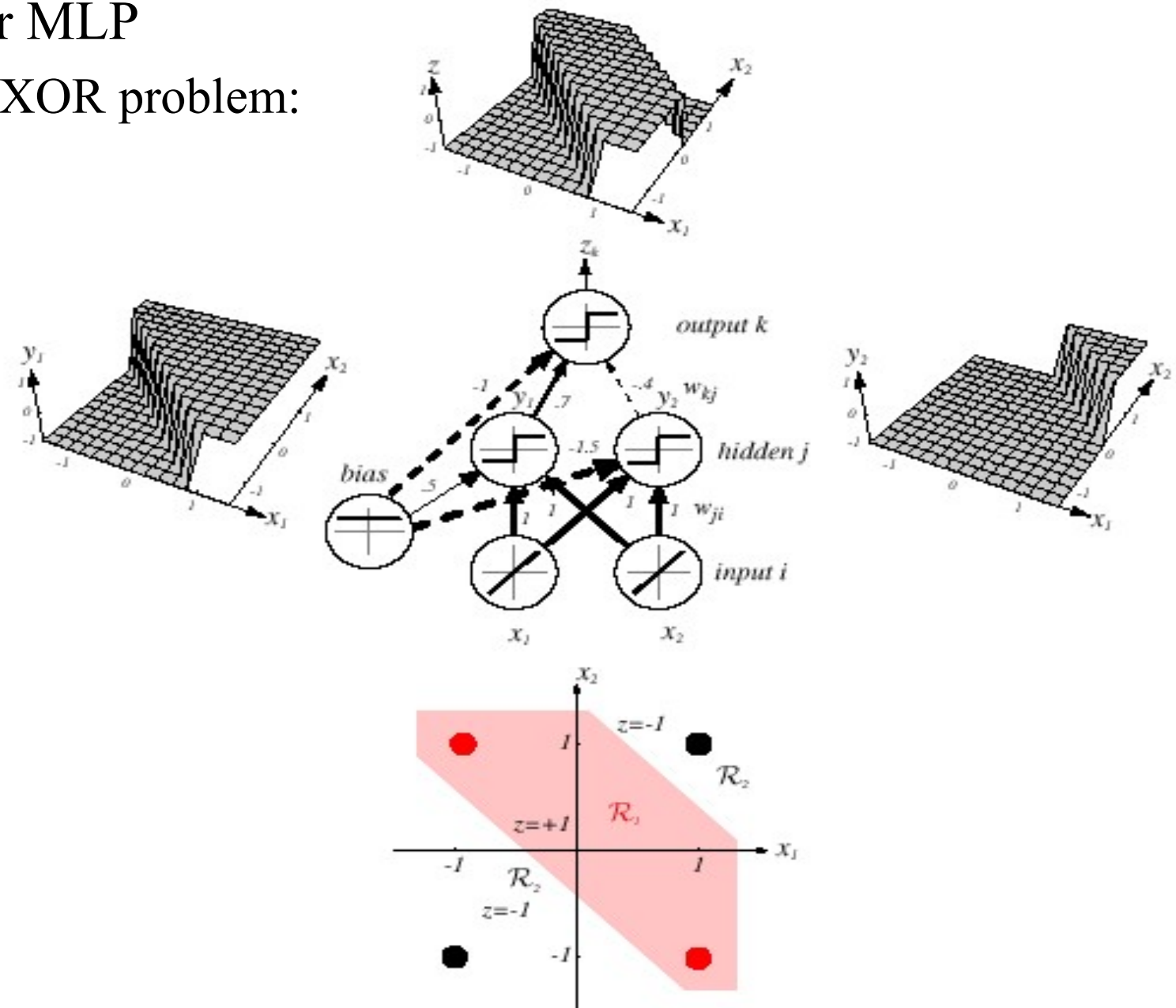


Fig. 6.1

Example: Solving the XOR Problem (Fig. 6.1)

- The hidden unit y_1 computes the boundary:

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

- The hidden unit y_2 computes the boundary:

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

- The final output unit emits $z_1 = f(0.7y_1 - 0.4y_2 - 1)$,

$$\Rightarrow z_1 = +1 \Leftrightarrow y_1 = +1 \text{ and } y_2 = -1.$$

$$\Rightarrow z_k = y_1 \& (\text{not } y_2) = (x_1 \text{ or } x_2) \& (\text{not } (x_1 \text{ and } x_2)) = x_1 \text{ XOR } x_2$$

which provides the nonlinear decision of Fig. 6.1.

MLP: Basic Structure and Notations

- The net activation for each neuron in the first hidden layer:

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

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 . (In neurobiology, such weights or connections are called “synapses”).

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

$$y_j = f(net_j)$$

- This is where the nonlinearity comes from; Often called activation function.

MLP: Basic Structure and Notations

- Similar computation for other hidden layers or the output layer
 - The outputs of the output nodes are denoted z_k . 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
 - $z_k = g_k(x)$ and classify the input \mathbf{x} according to the largest discriminant function $g_k(x) \quad \forall k = 1, \dots, C$

For A 3-layer MLP ...

- The general expression for the output can be written as (assuming C classes)

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

- Hidden units enable us to express more complicated nonlinear functions and thus extend the classification.
- The activation function does not have to be a sign function, it is often required to be continuous and differentiable.
- The activation function can be different for the output layer and for the hidden layer or even have different activation for each individual unit.
 - We assume that they are the same in following discussion.

Expressive Power of MLP

- Question: Can any decision (boundary) be implemented by a three-layer network described by (*) ?

Answer: (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.”

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

for properly chosen functions Ξ_j and ψ_{ij} .

Unfortunately: Kolmogorov’s theorem tells us very little about how to find the nonlinear functions based on data -- the central problem in network-based pattern recognition.

- There are different views arguing this, e.g., see Girosi & Poggio, “Representation Properties of Networks: Kolmogorov’s Theorem Is Irrelevant”, Neural Computation, 1, 465-469, 1989.

Illustration

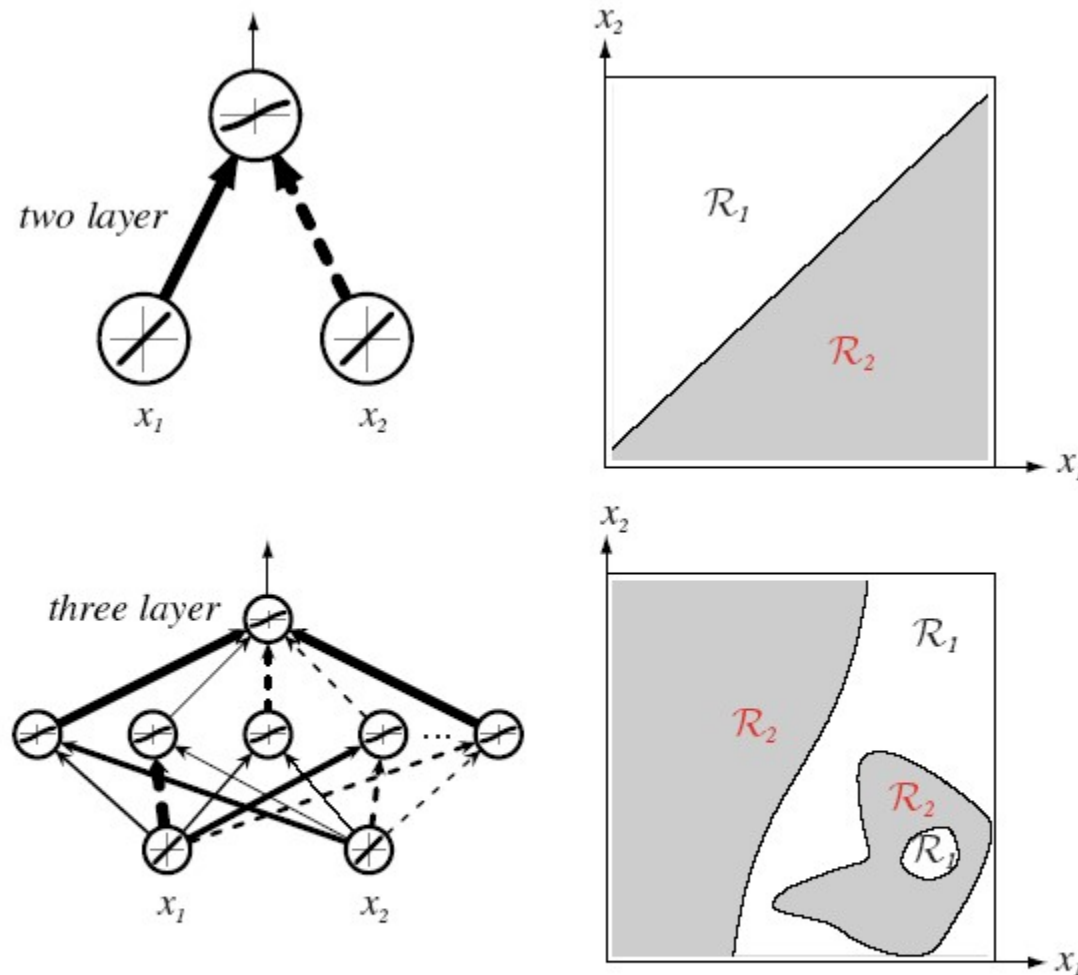


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O.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

Another View of the Expressive Power

Caveat: Not necessarily how the training works.

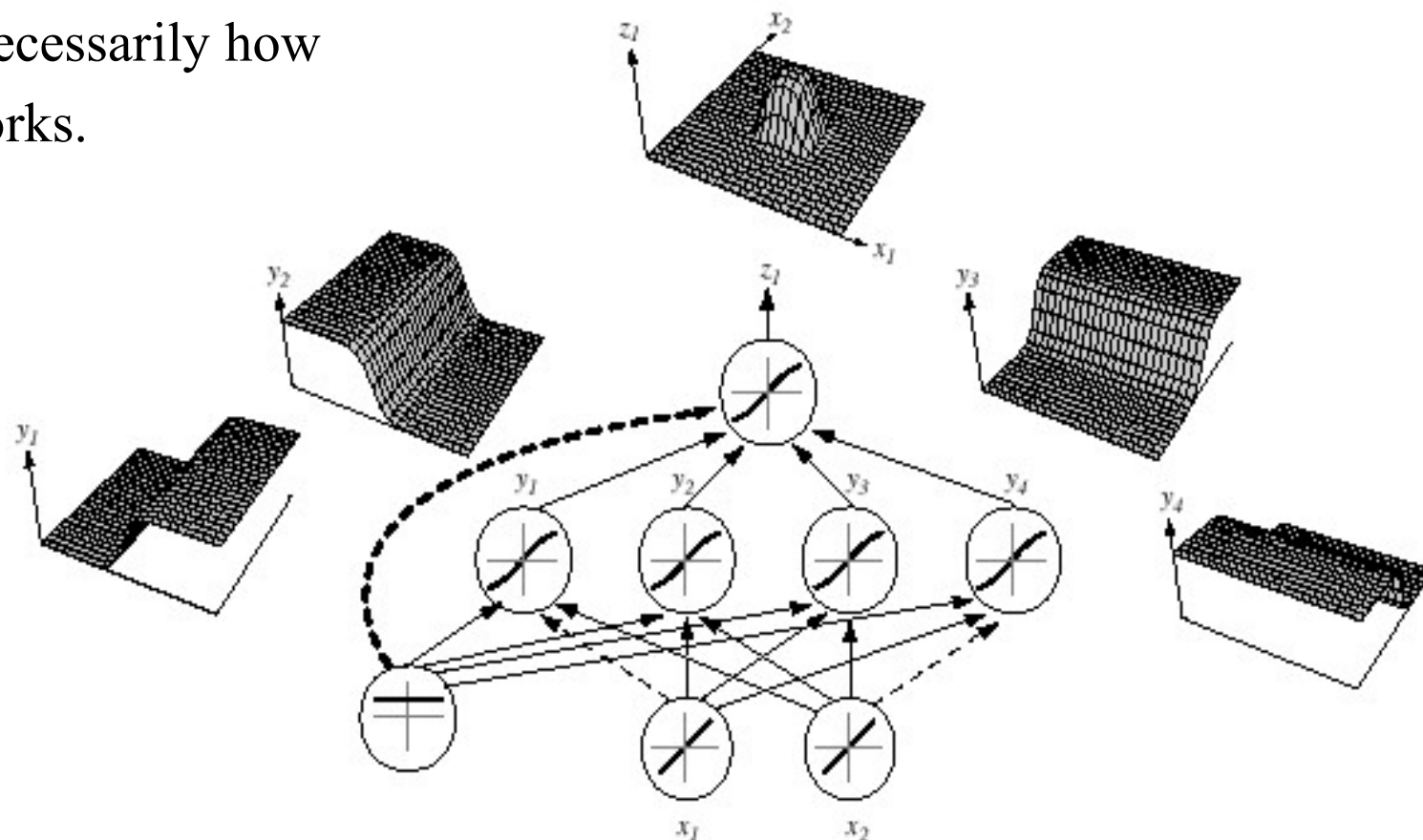


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a “bump” at the output unit. Given a sufficiently large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

Backpropagation Algorithm

- In practice, how can we get the nonlinear functions and the weights?
- The goal: to learn the weights based on the training patterns and the desired outputs (different sets of weights result in different nonlinear mappings).
- In a three-layer MLP, it is straightforward to understand how the output, and thus the error, depend on the hidden-to-output layer weights.
 - Hence we can imagine that, for example, the LMS algorithm can update the weights between the output and the hidden layers.
 - But no “desired outputs” for the hidden nodes are given → Cannot compute error directly.
 - Compute “effective” errors by back-propagating the errors from the output layer.

A 3-layer MLP and Notations

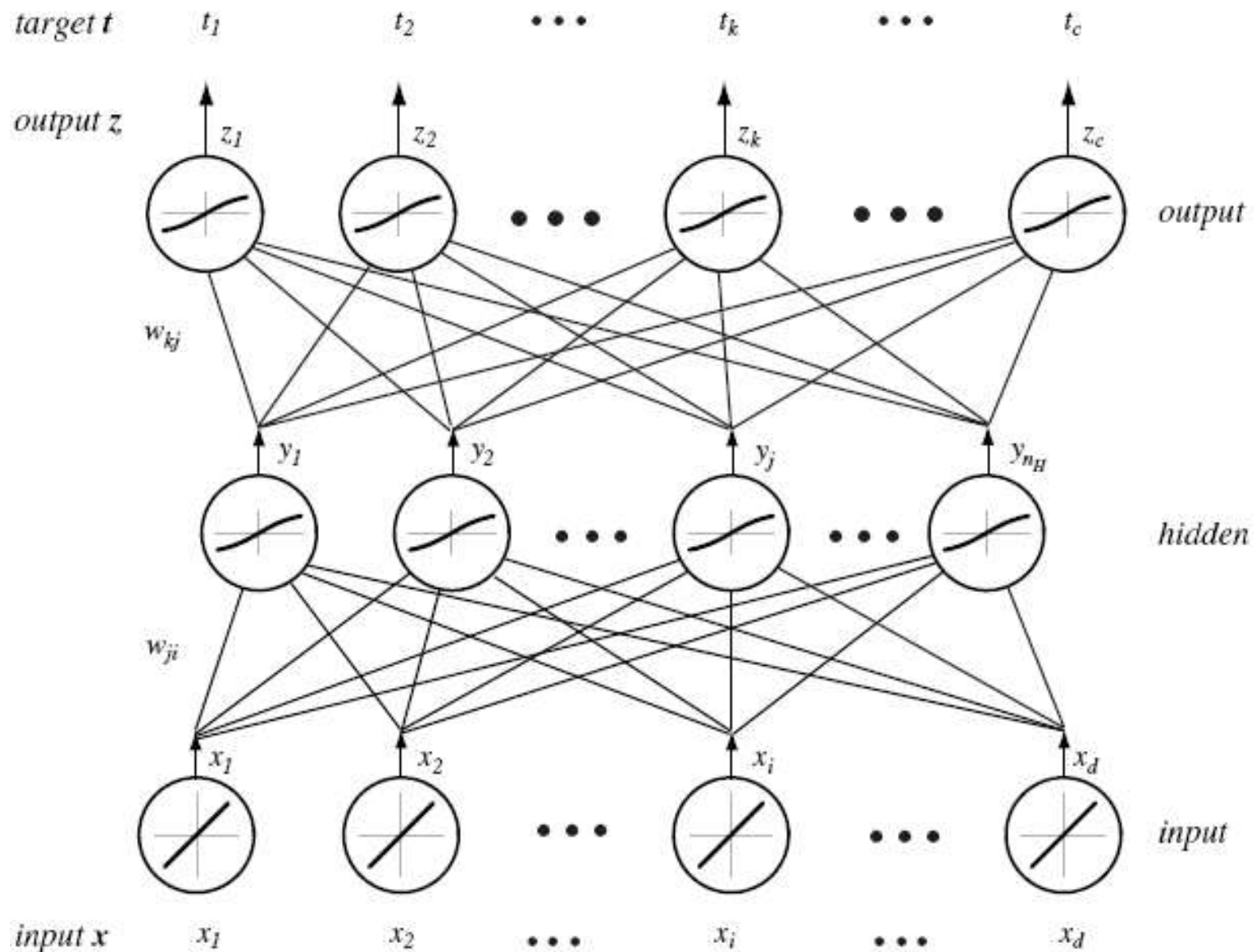


FIGURE 6.4. A d - n_H - c fully connected three-layer network and the notation

Network Learning Based on BP (1)

- Let t_k be the k -th target (or desired) output and z_k be the k -th computed output with $k = 1, \dots, C$
 - The training error: $J(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^C (t_k - z_k)^2 = \frac{1}{2} \|\mathbf{t} - \mathbf{z}\|^2$
 - The backpropagation learning rule is based on gradient descent
 - The weights are initialized with random values and are changed in a direction that will reduce the error:

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

where η is the learning rate, and the weight update is done as

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

for the m -th pattern presented

Network Learning Based on BP (2)

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

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

- Conclusion: The learning rule for the input-to-hidden weights is:

$$\Delta w_{ji} = \eta x_i \delta_j = \eta \underbrace{\left[\sum_{k=1}^C w_{kj} \delta_k \right]}_{\delta_j} f'(net_j) x_i$$

Network Learning Based on BP (3)

- Stochastic v.s. Batch BP
 - A stochastic BP algorithm

```
Begin initialize  $n_H$ ;  $\mathbf{w}$ , criterion  $\theta$ ,  $\eta$ ,  $m \leftarrow 0$   
  do  $m \leftarrow m + 1$   
     $\mathbf{x}^m \leftarrow$  randomly chosen pattern  
     $w_{ji} \leftarrow w_{ji} + \eta \delta_j x_i$ ;  $w_{kj} \leftarrow w_{kj} + \eta \delta_k y_j$   
  until  $\|\nabla J(\mathbf{w})\| < \theta$   
  return  $\mathbf{w}$   
End
```

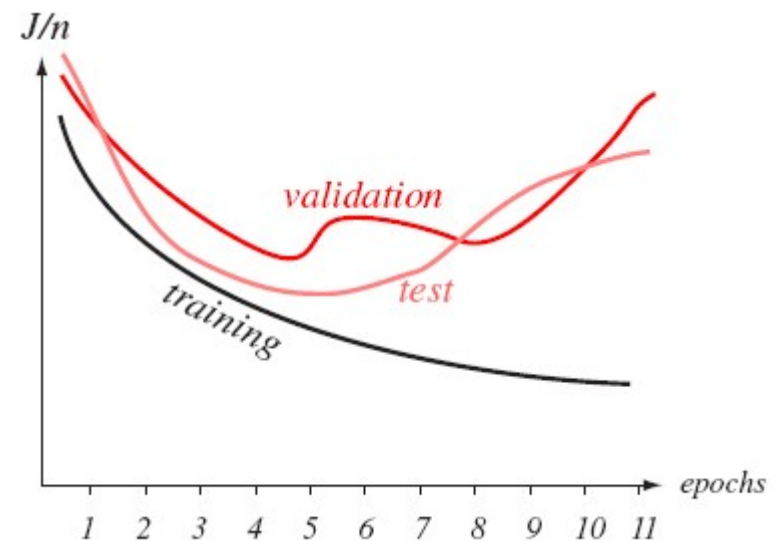
Network Learning Based on BP (4)

- 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 → Could use batch algorithm, but single sample algorithm should work if it converges
 - The total training error is the sum over the errors of n individual patterns

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

Network Learning Based on BP (5)

- 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 in general higher than on the training set, and it can decrease as well as increase.
 - A validation set is used in order to decide when to stop training



BP as Feature Mapping

- Consider a 3-layer MLP
 - The hidden-to-output layer leads to a linear discriminant
 - The nonlinearity in the hidden layer is critical
 - Typical activation functions

Step function $f(a) = \begin{cases} +1 & \text{if } a \geq u \\ -1 & \text{if } a < u \end{cases}$

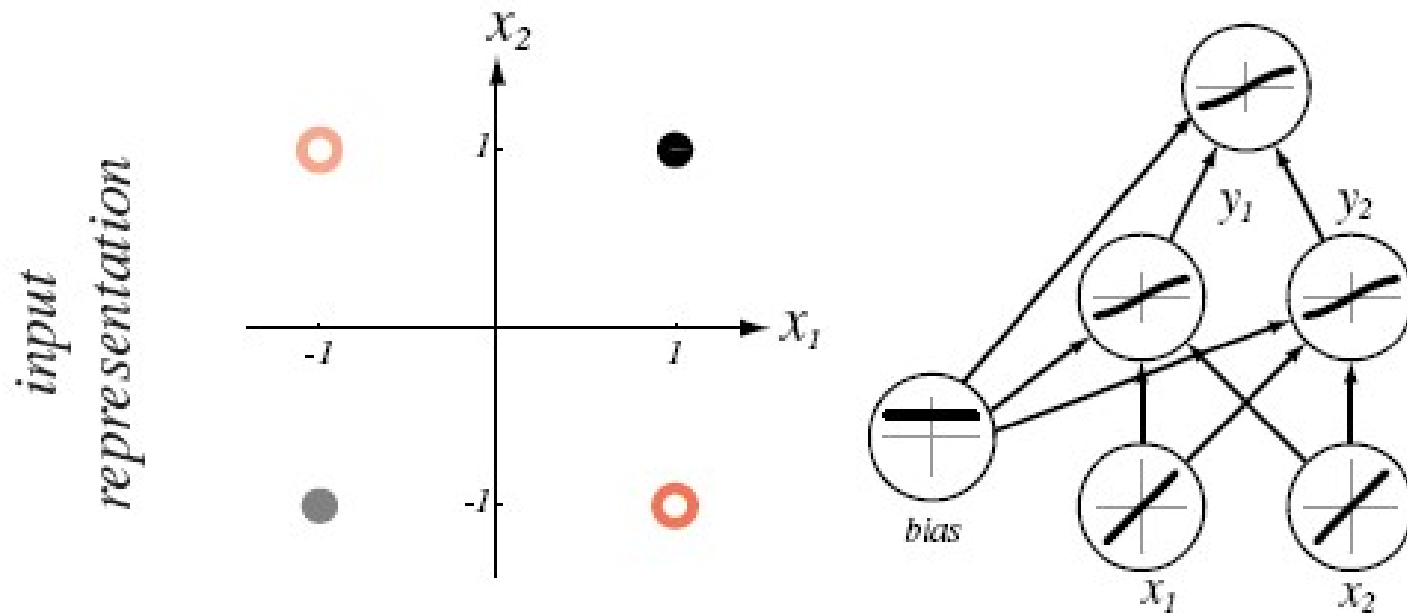
Truncated linear function $f(a) = \begin{cases} +1 & \text{if } a \geq u \\ a & \text{if } -u \leq a < u \\ -1 & \text{if } a < -u \end{cases}$

Logistic Sigmoid $f(a) = \frac{1}{1 + e^{-ha}}$

Gaussian $f(a) = e^{-\frac{|a-u|}{2\sigma^2}}$

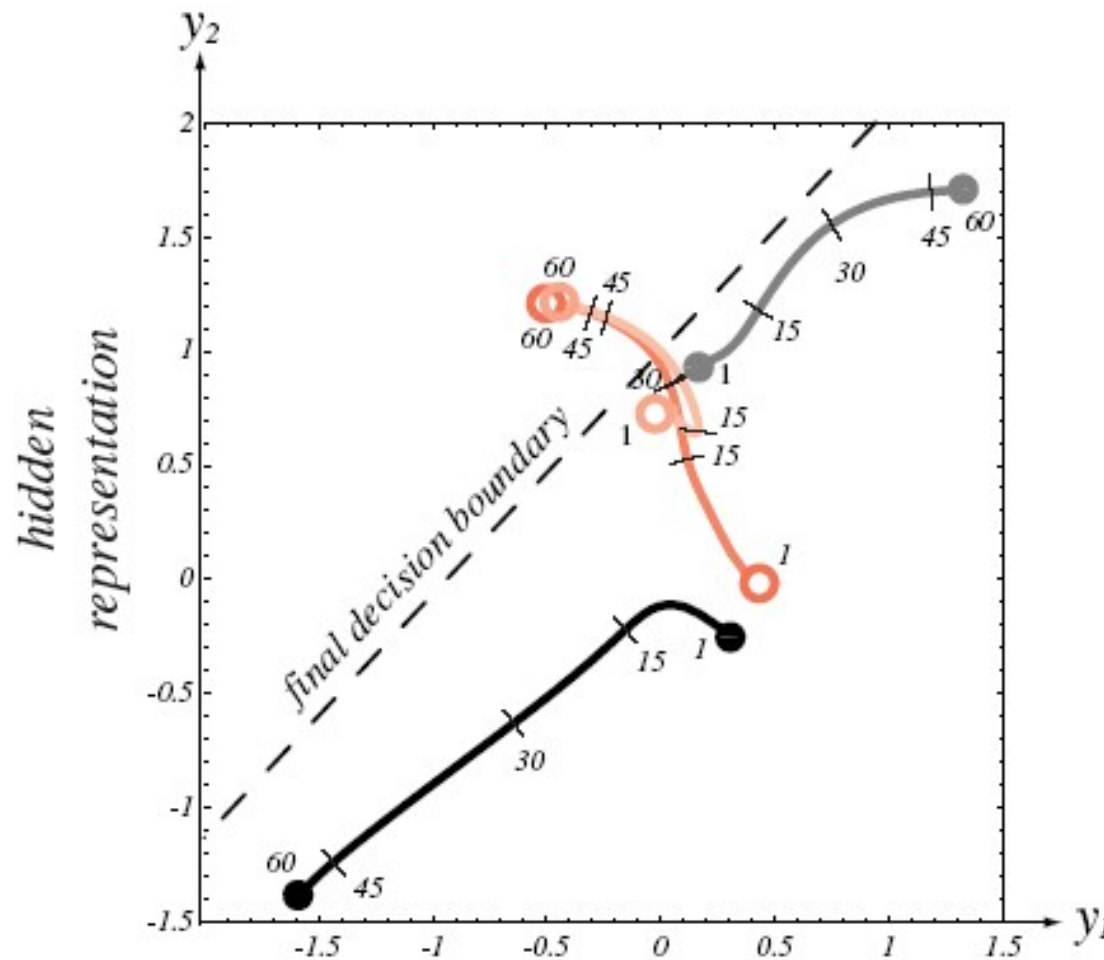
XOR Problem Again

- What does the input-to-hidden layer do to solve e.g. the XOR problem?
 - During training, how does the internal representation (y_1, y_2) evolve until we can linearly separate the two classes?



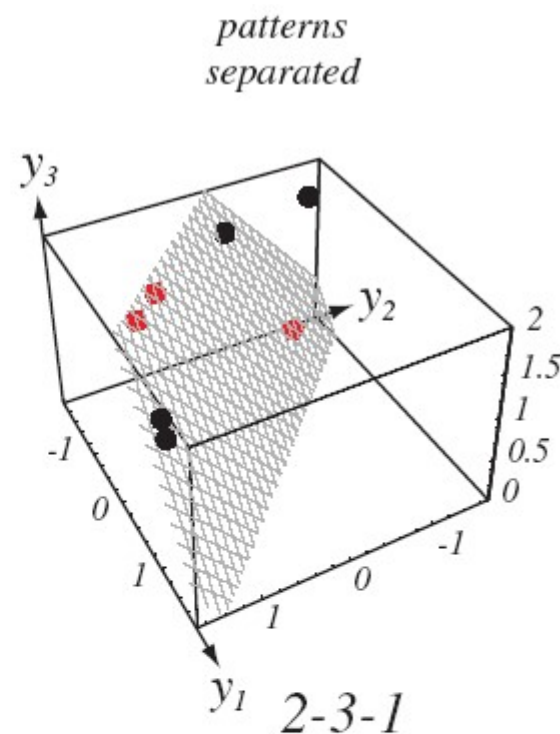
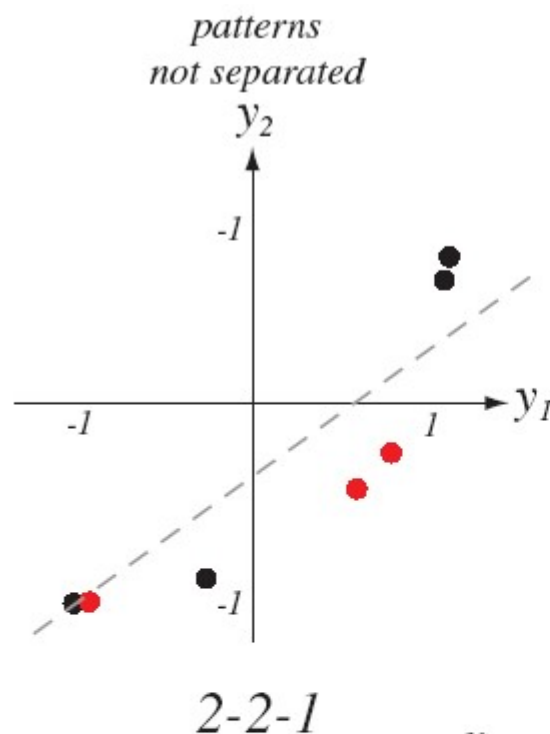
Trajectories of the mapped features

- Initial random small weights \rightarrow Not linearly separable.
 - Assuming a sigmoid activation function, then for small weights, the network is roughly linear.
- Final weights map the point into separable configuration.



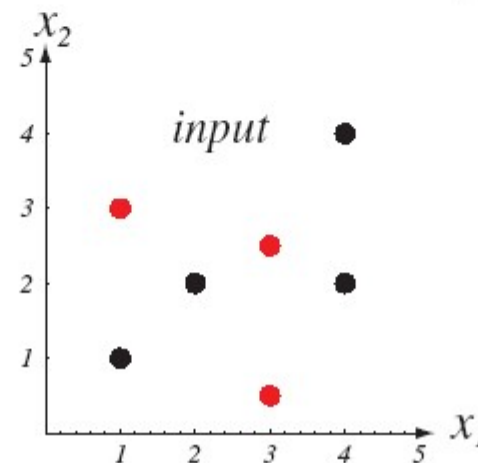
A More Complex Example

- A 2-2-1 MLP cannot separate the data completely.



- A 2-3-1 MLP solves the problem
 - More hidden nodes \rightarrow Higher dimension for the internal representation in the hidden layer.

Question: If the input is already high-dimensional such as an image?



Another Example

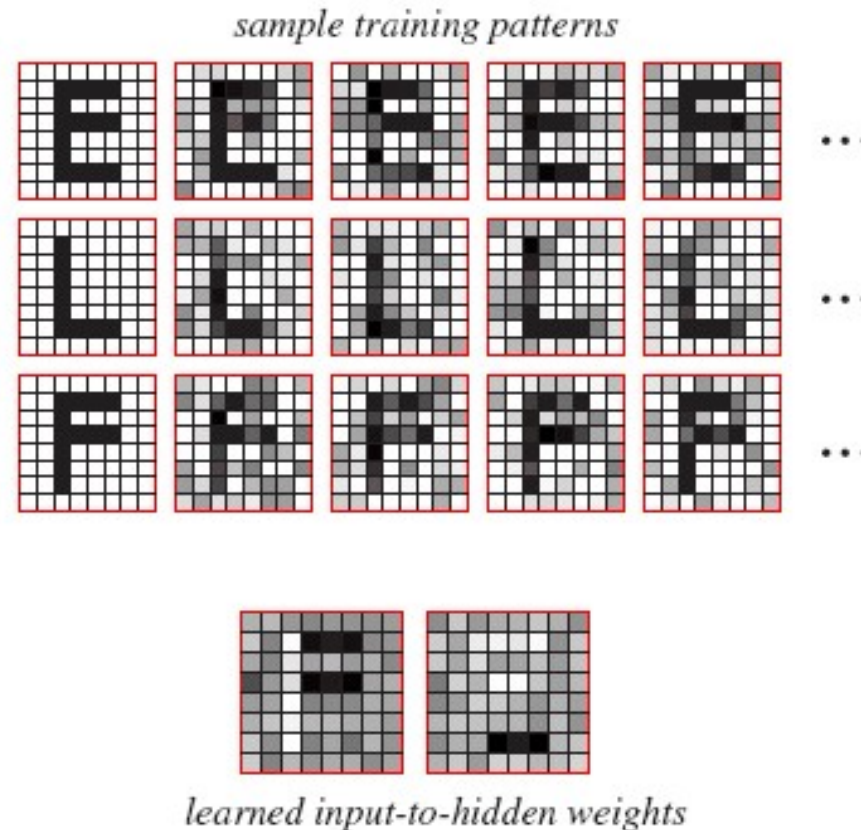
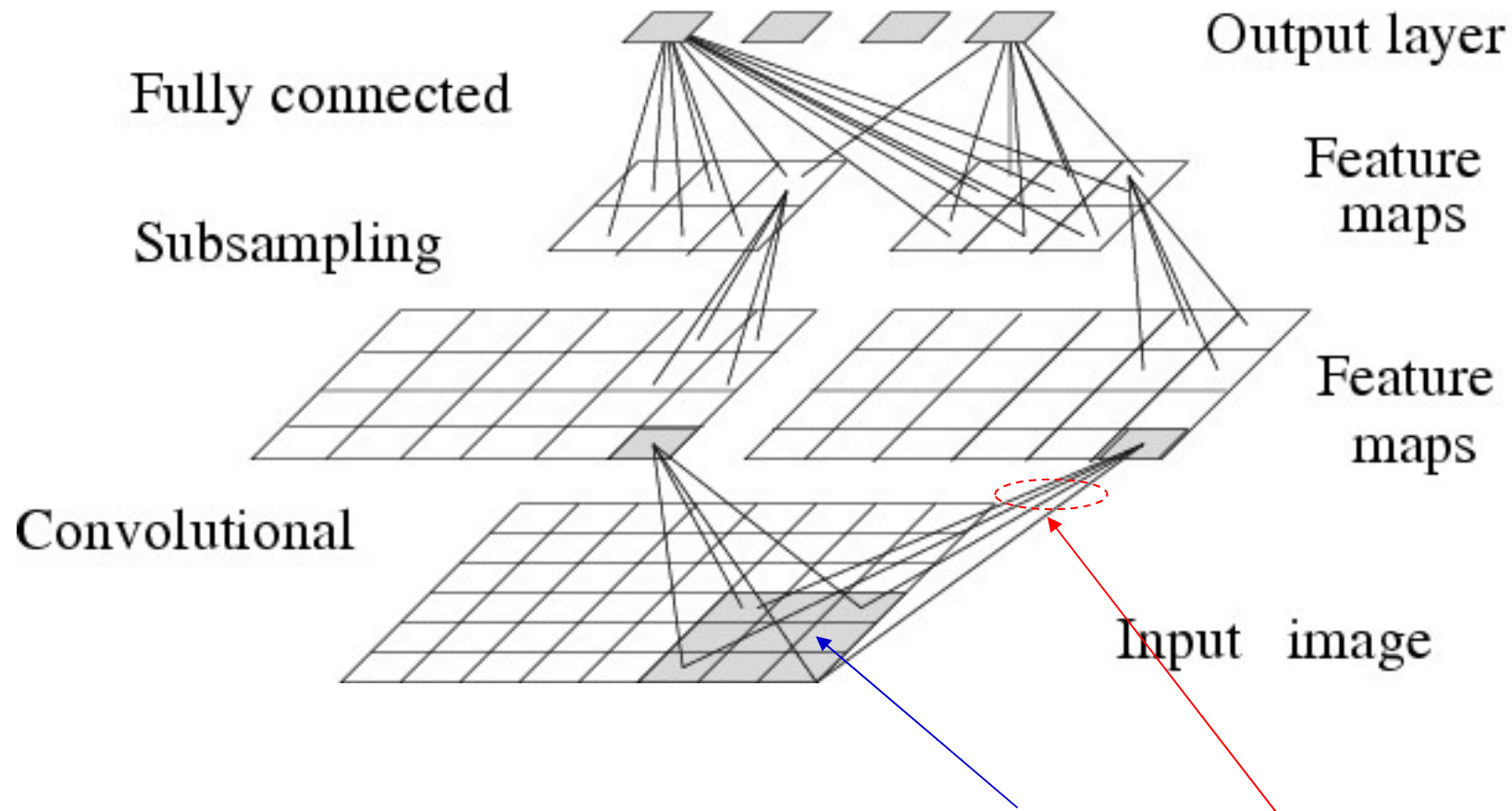


FIGURE 6.13. The top images represent patterns from a large training set used to train a 64-2-3 sigmoidal network for classifying three characters. The bottom figures show the input-to-hidden weights, represented as patterns, at the two hidden units after training. Note that these learned weights indeed describe feature groupings useful for the classification task. In large networks, such patterns of learned weights may be difficult to interpret in this way. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Convolutional Neural Network

- Recall that a general pattern recognition task involves *feature extraction* and *classification in the feature space*.
- When no good features are known a priori, can we let an MLP do even the feature extraction?
 - The objective in convolutional neural networks (CNNs) for image-based pattern recognition.
- CNNs
 - Have no explicit feature extraction.
 - Operate directly on image.
- **On your own: Check out recent updates in the name of “deep learning”.**

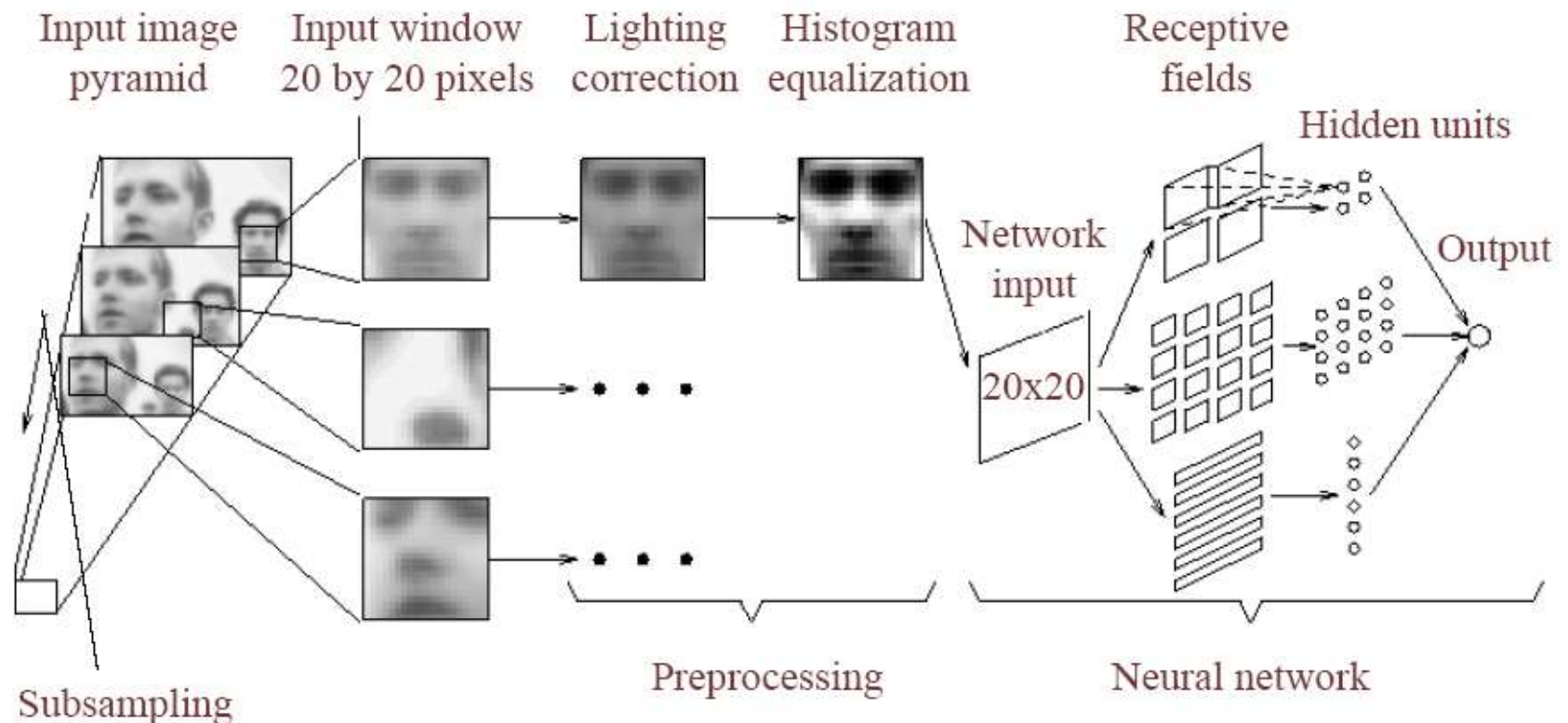
CNN: An Illustration



Key concepts: Feature map, receptive field, kernel, weight sharing.

CNN Applications

- Proposed in the context of OCR (Fukushima, Y. LeCun, *et al.*)
- Also successful for face detection (similar idea): **Neural Network-Based Face Detection**, [H. Rowley](#), [S. Baluja](#), and [T. Kanade](#), *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 1, January, 1998, pp. 23-38.



Face Detection Example



CNN for Document Segmentation

- Let's look at another example: document segmentation (into graphical and non-graphical regions): A 2-class problem.
- The experiments:
 - Collect some data with manual segmentation.
 - Define an input window size, e.g., 16x16 (the resolution to do segmentation).
 - 16x16 blocks with labels given by the manual segmentation forms the training set.
 - We could synthesize more training samples by randomly positioning the blocks of the same label and then re-sample (Caveat: this introduces blocking artifacts in the training samples).
 - Train a CNN, with different receptive fields for different feature maps.
 - Test on a new document

CNN for Document Segmentation (cont'd)

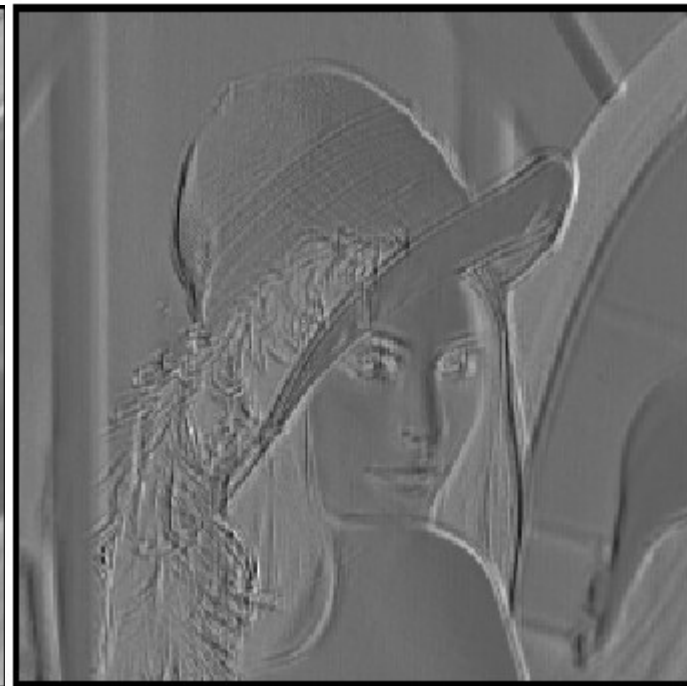
- Interesting results:
 - Let's applied the learned kernel onto an irrelevant image.



Input



Kernel of size 3x5



Kernel of size 7x5

MLP: Bayesian Interpretation

- Consider an MLP for a C class problem.
 - Let $g_k(\mathbf{x}; \mathbf{w})$ be the output of the k -th output node
 - Let the target signal be

$$t_k(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \omega_k \\ 0 & \text{otherwise} \end{cases}$$

- For a given training set, the training error at the k -th output may be computed as

$$J_k(\mathbf{w}) = \sum_{\mathbf{x}} (g_k(\mathbf{x}; \mathbf{w}) - t_k)^2$$

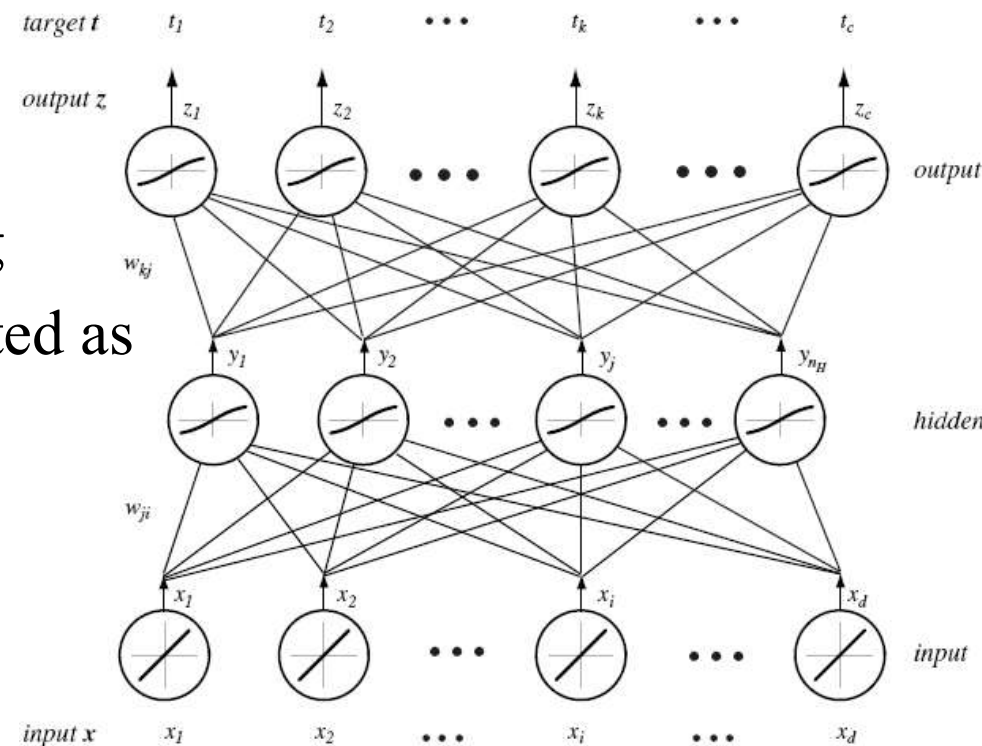


FIGURE 6.4. A d - n_H - c fully connected three-layer network and the notation

Bayesian Interpretation (cont'd)

- This can be further written as

$$\begin{aligned} J_k(\mathbf{w}) &= \sum_{\mathbf{x}} (g_k(\mathbf{x}; \mathbf{w}) - t_k)^2 \\ &= \sum_{\mathbf{x} \in \omega_k} (g_k(\mathbf{x}; \mathbf{w}) - 1)^2 + \sum_{\mathbf{x} \notin \omega_k} (g_k(\mathbf{x}; \mathbf{w}) - 0)^2 \\ &= n \left[\frac{n_k}{n} \frac{1}{n_k} \sum_{\mathbf{x} \in \omega_k} (g_k(\mathbf{x}; \mathbf{w}) - 1)^2 + \frac{n - n_k}{n} \frac{1}{n - n_k} \sum_{\mathbf{x} \notin \omega_k} (g_k(\mathbf{x}; \mathbf{w}) - 0)^2 \right] \end{aligned}$$

n_k : # of training samples in ω_k

n : # of total training samples

Bayesian Interpretation (cont'd)

- After some manipulation, we can have

$$\lim_{n \rightarrow \infty} \frac{1}{n} J_k(\mathbf{w}) = \int [g_k(\mathbf{x}; \mathbf{w}) - P(\omega_k | \mathbf{x})]^2 p(\mathbf{x}) d\mathbf{x} \\ + \int P(\omega_k | \mathbf{x}) P(\omega_{i \neq k} | \mathbf{x}) p(\mathbf{x}) d\mathbf{x}$$

- The training process minimizes $J_k(\mathbf{w})$. This is equivalent to minimizing (since the second term is independent of \mathbf{w})

$$\int [g_k(\mathbf{x}; \mathbf{w}) - P(\omega_k | \mathbf{x})]^2 p(\mathbf{x}) d\mathbf{x}$$

Bayesian Interpretation (cont'd)

- Above is true for all C classes ω_k , $k=1,\dots,C$. Equivalently, the training process minimize

$$\sum_{k=1}^C \int [g_k(\mathbf{x}; \mathbf{w}) - P(\omega_k | \mathbf{x})]^2 p(\mathbf{x}) d\mathbf{x}$$

- Asymptotically, the outputs of the trained network approximate the a posteriori probabilities in a least-square sense,

$$g_k(\mathbf{x}; \mathbf{w}) \approx P(\omega_k | \mathbf{x})$$

- See 6.6.2 for caveats of this interpretation.

Outputs as Probabilities

- We saw that, with 0-1 target values, the C outputs would have the desired limiting property of approaching the posterior probabilities.
 - With limited data, this may not be attained.
 - In fact, the outputs may not even sum to 1.0.
 - If this happens for too much of the input space, it may be an indication that the network is not accurately modeling the posteriors.
- To facilitate modeling probabilities, let the output nonlinearity be:

$$f(net_k) \propto e^{net_k}$$

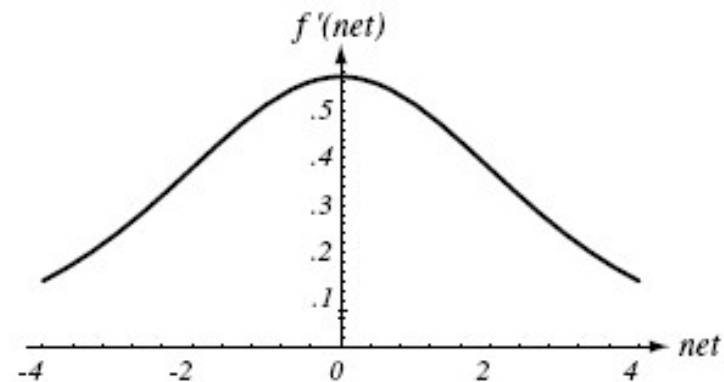
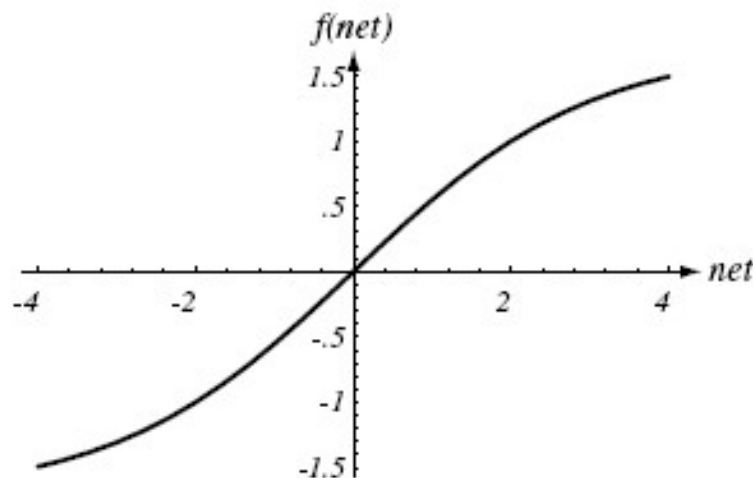
and for each pattern, normalize the outputs by
$$z_k = \frac{e^{net_k}}{\sum_{i=1}^C e^{net_i}}$$

➔ the *softmax* method (as opposed to the simple *max* or winner-take-all)

Techniques for Improving BP

- Activation function
 - Basic requirement: non-linear, “saturated”, continuous and smooth
 - Desired properties: Monotonicity, linearity for small net activation
 - Sigmoid is an example having all these properties
 - Anti-symmetric sigmoid is also commonly used

$$f(x) = a \left[\frac{e^{bx} - b^{-bx}}{e^{bx} + b^{-bx}} \right]$$



Techniques for Improving BP

- Scaling the input: “standardization”.
- Target values: should be smaller than the saturate value of the sigmoid function.
- Expanding the training set by adding noise.
- Expanding the training set by construction: e.g., apply the likely transformation group.
- Number of hidden nodes (see an example on the next slide)
 - Weight pruning.
- Number of hidden layers.

Techniques for Improving BP

- Number of hidden nodes:

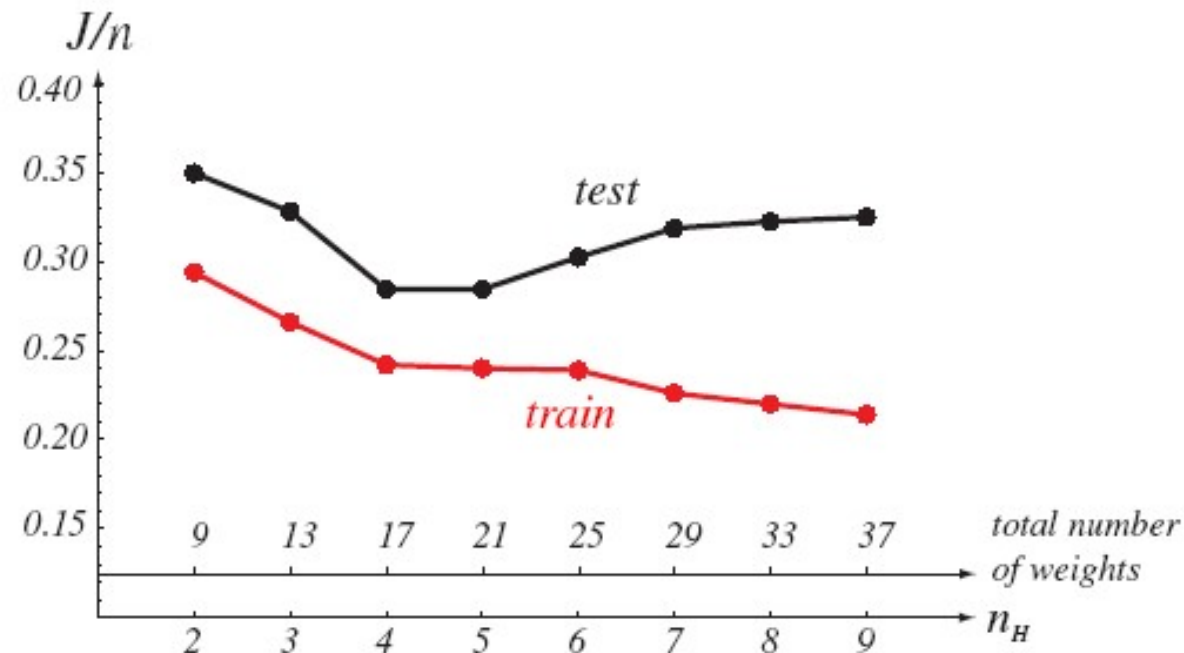


FIGURE 6.15. The error per pattern for networks fully trained but differing in the numbers of hidden units, n_H . Each $2 - n_H - 1$ network with bias was trained with 90 two-dimensional patterns from each of two categories, sampled from a mixture of three Gaussians, and thus $n = 180$. The minimum of 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.

Techniques for Improving BP

- Weight initialization: setting them to small random numbers; not all the same; not all 0s.
- Learning rates.

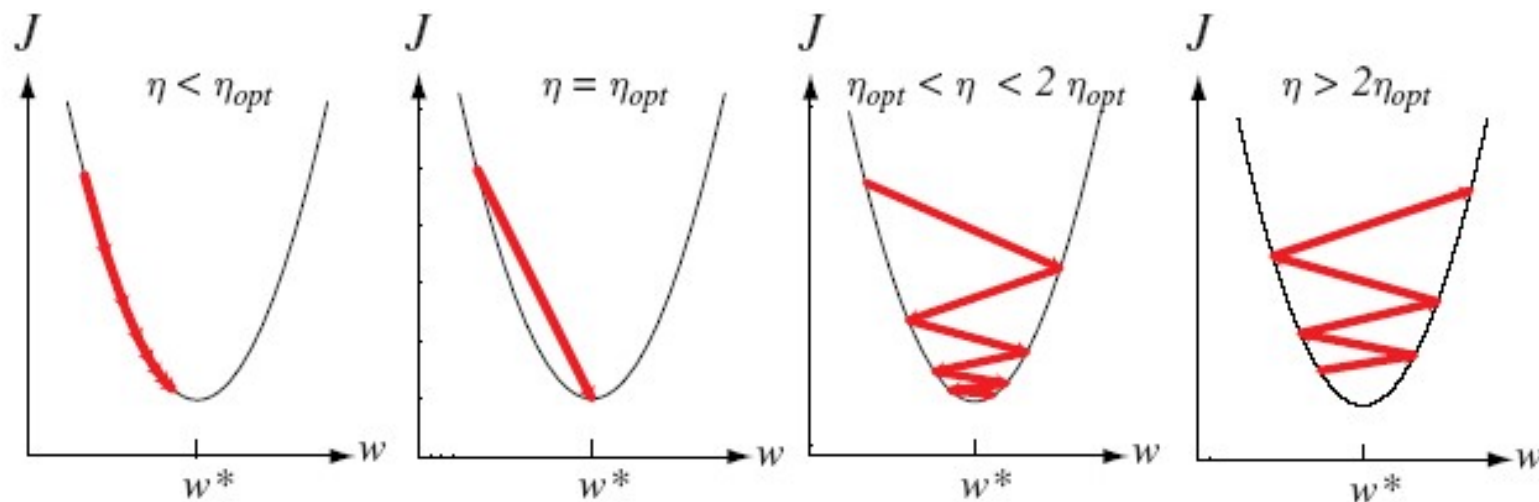


FIGURE 6.16. Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, 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.

Techniques for Improving BP

- Learning with momentum: to cope with potential plateaus in the error surface.

$$\mathbf{w}(m+1) = \mathbf{w}(m) + (1 - \alpha) \Delta \mathbf{w}_{bp}(m) + \alpha (\mathbf{w}(m) - \mathbf{w}(m-1))$$

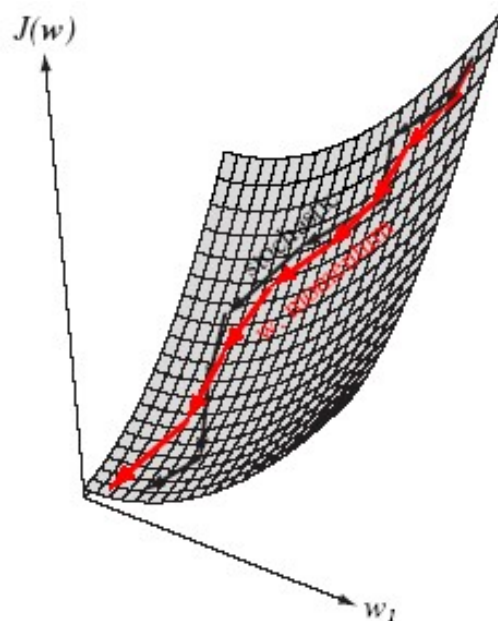


FIGURE 6.18. The incorporation of momentum into stochastic gradient descent by Eq. 37 (red arrows) reduces the variation in overall gradient directions and speeds learning. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.


Radial Basis Function Networks

- The interpolation problem: Given input samples $\{\mathbf{x}_i\}$ and corresponding target output $\{t_i\}$, find a function $F: R^d \rightarrow R$ such that

$$F(\mathbf{x}_i) = t_i, \text{ for all } i$$

- **RBF** technique (Powell, 1988)

some non-linear
function

$$F(\mathbf{x}) = \sum_{i=1}^n w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|)$$


$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1n} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{n1} & \varphi_{n2} & \cdots & \varphi_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}$$

Radial Basis Function Networks (cont'd)

- Generalize: only use n_H basis functions: each hidden unit should represent a *cluster* in the input space

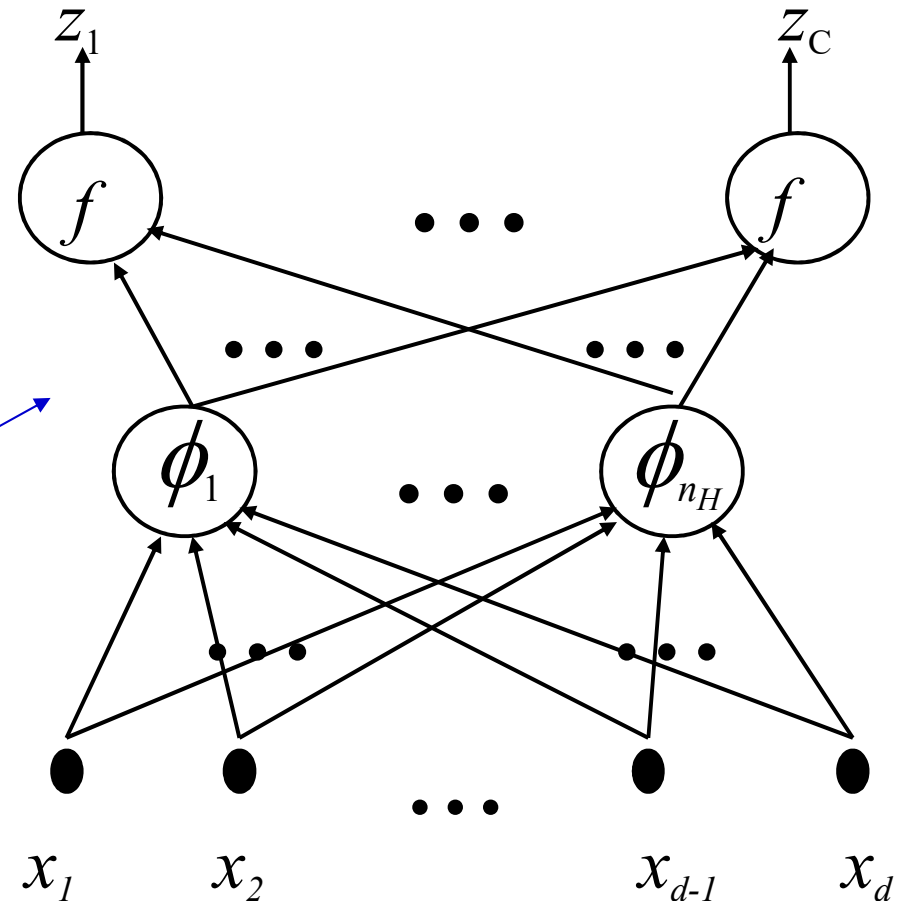
$$F(\mathbf{x}) = \sum_{i=1}^{n_H} w_i \phi(\|\mathbf{x} - \mathbf{c}_i\|)$$

– Typically

$$- n_H \ll n$$

$$- \phi(\|\mathbf{x} - \mathbf{c}_i\|) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{2\sigma_i^2}\right)$$

- With multiple outputs, we have
- If $f(\mathbf{x})$ is again a nonlinear function of the net activation, we could use BP to learn the weights.



RBFNs and MLPs

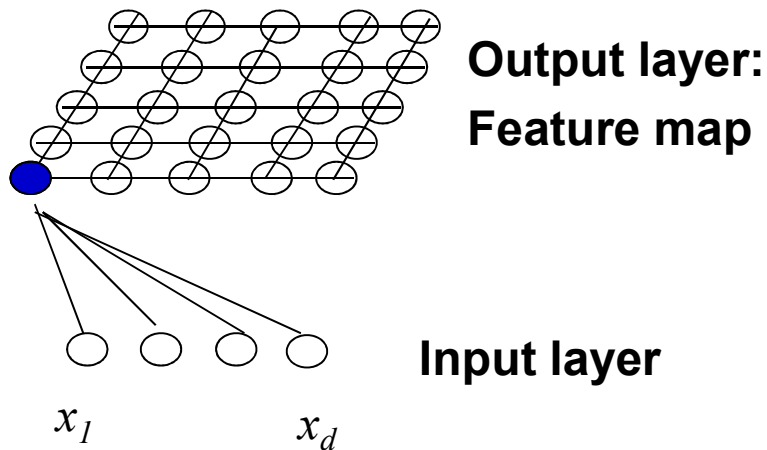
- **Locality.** In RBFNs only a small fraction of ϕ_j is active for each input vector \Rightarrow more efficient training algorithms
- **Separation surfaces.** MLP produces open separation surfaces vs. RBFNs closed separation surfaces
- **Approximation capability.** The universality property still holds for RBFNs if a sufficient number of ϕ is given.
- **Interpretation:** RBFNs are easier to interpret than MLPs. ϕ_j can be interpreted as $p(\text{cluster } j | \mathbf{x})$ and w_{kj} as $p(C_k | \text{cluster } j)$

Kohonen Self-Organizing Feature Maps

- An unsupervised learning method.
 - Proposed in earlier 1980s by Kohonen
- The basic method is useful for finding clusters and their relationships in input data
 - Can identify local clusters in the data and organize them globally in an unsupervised fashion
- Not intended for optimal classification
 - Learning vector quantization does classification better

Basics of SOMs

- Consider an SOM viewed as the following 2-layer structure (the 2-D rectangular grid of nodes forms the feature map)



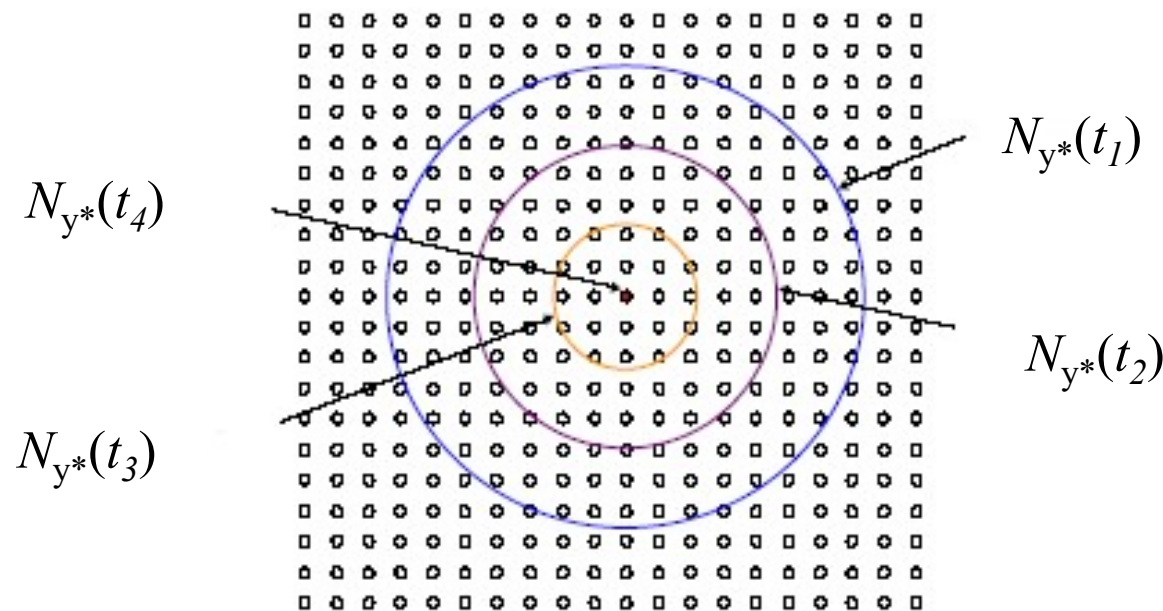
Every node in the output layer is connected to the input layer via d weights (shown only for one node)

Each output node (defined by its weights) can be viewed as a vector in the d -dimensional feature space

The output layer could be of other forms (e.g., 1-D array, 2-D hexagonal lattice, 3-D grid, etc.)

Learning in SOMs

- Step 1. A winner node y^* is determined (e.g., based on the distance between the input vector \mathbf{x}_i and the output nodes).
- Step 2. A neighborhood $N_{y^*}(t)$ of y^* is defined
 - The size of the neighborhood is a function of iteration (decreasing)



Learning in SOMs (cont'd)

- Step 3. Nodes in $N_{y^*}(t)$ are updated
- Step 4. Get the next input sample and go to Step 1.
- In the textbook (Sect. 10.14.1), Steps 2 & 3 are sort of combined into the following update rule

$$w_{ki}(t+1) = w_{ki}(t) + \eta(t)\Lambda(|y - y^*|)(x_i - w_{ki}(t))$$

with Λ illustrated as

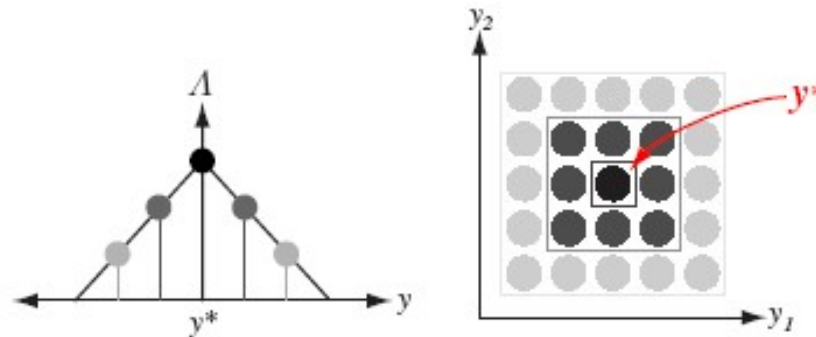


FIGURE 10.29. Typical window functions for self-organizing maps for target spaces in one dimension (left) and two dimensions (right). In each case, the weights at the maximally active unit, y^* , in the target space get the largest weight update while units more distant get smaller update. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

An Illustration: 1-D Output Layer

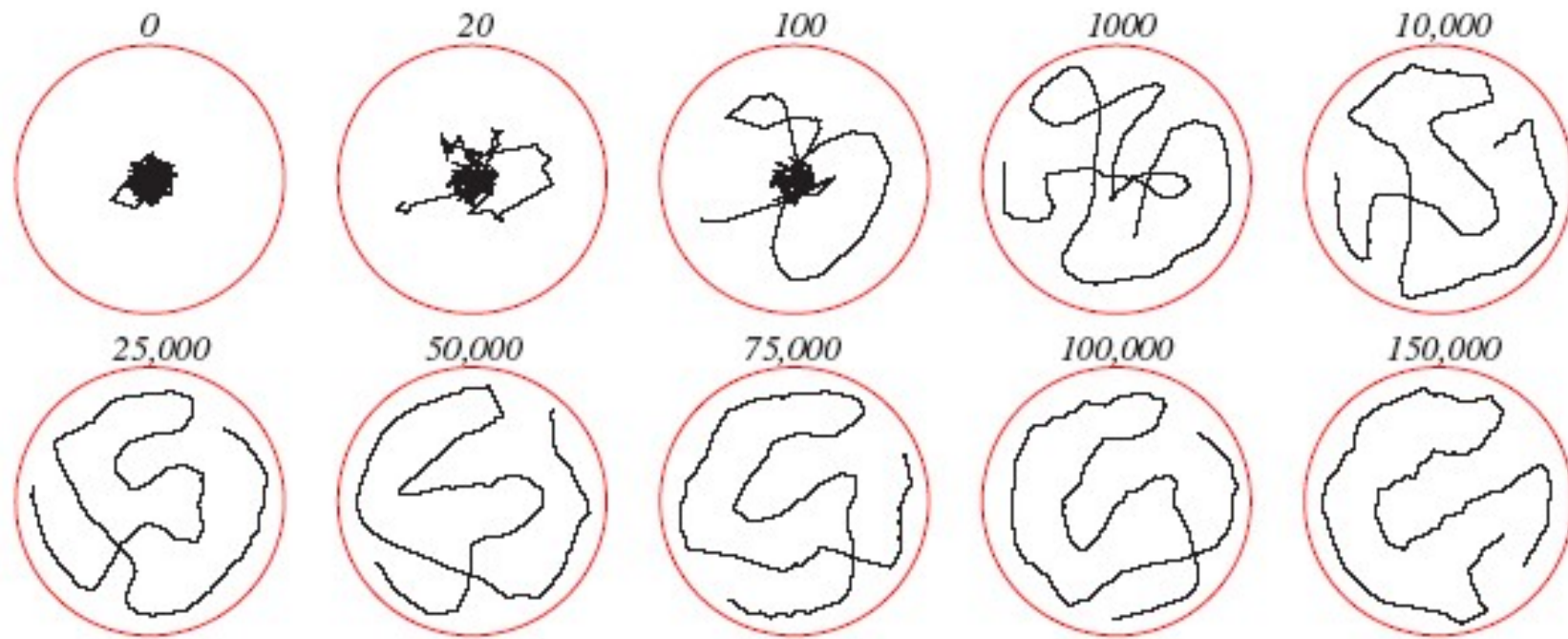


FIGURE 10.30. If a large number of pattern presentations are made using the setup of Fig. 10.28, a topologically ordered map develops. The number of pattern presentations is listed. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

An Illustration: 2-D Output Layer

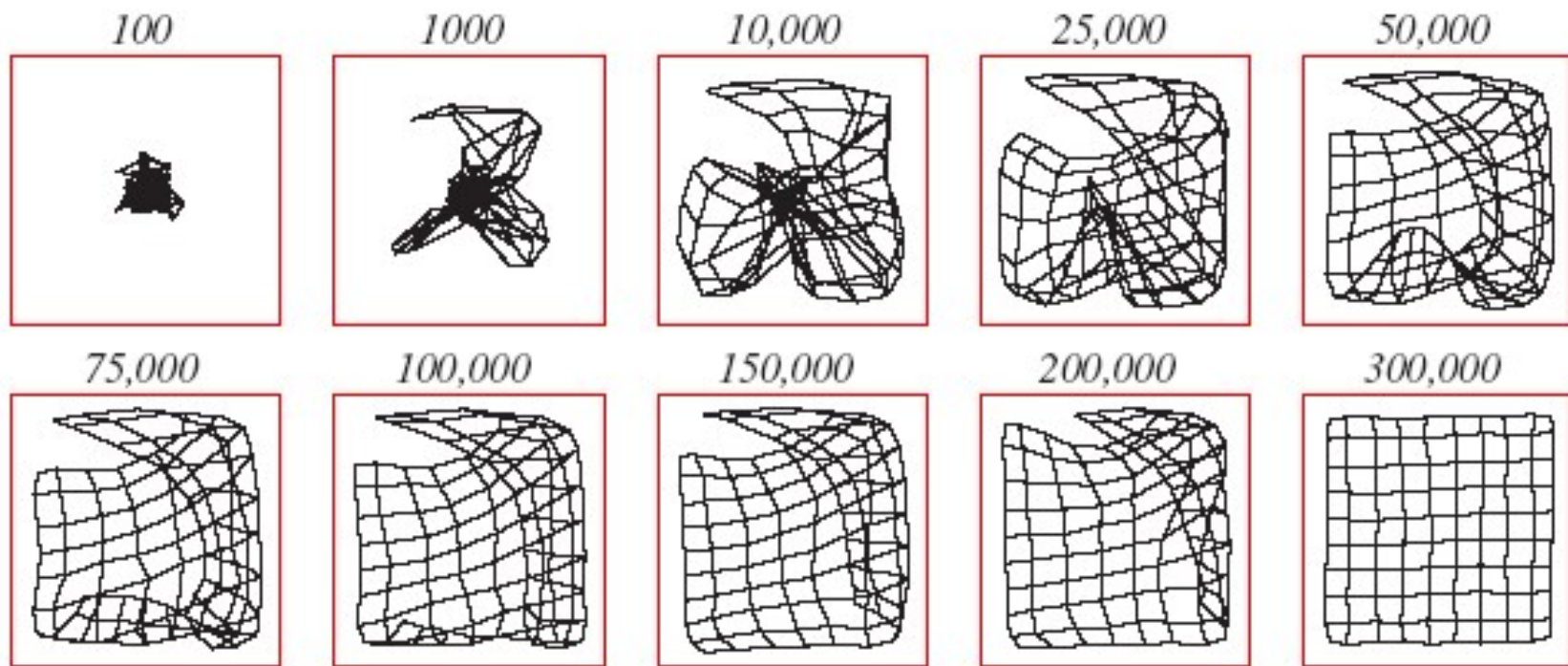


FIGURE 10.31. A self-organizing feature map from a square source space to a square (grid) target space. As in Fig. 10.28, each grid point of the target space is shown atop the point in the source space that leads maximally excites that target point. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

An Illustration: non-uniform input

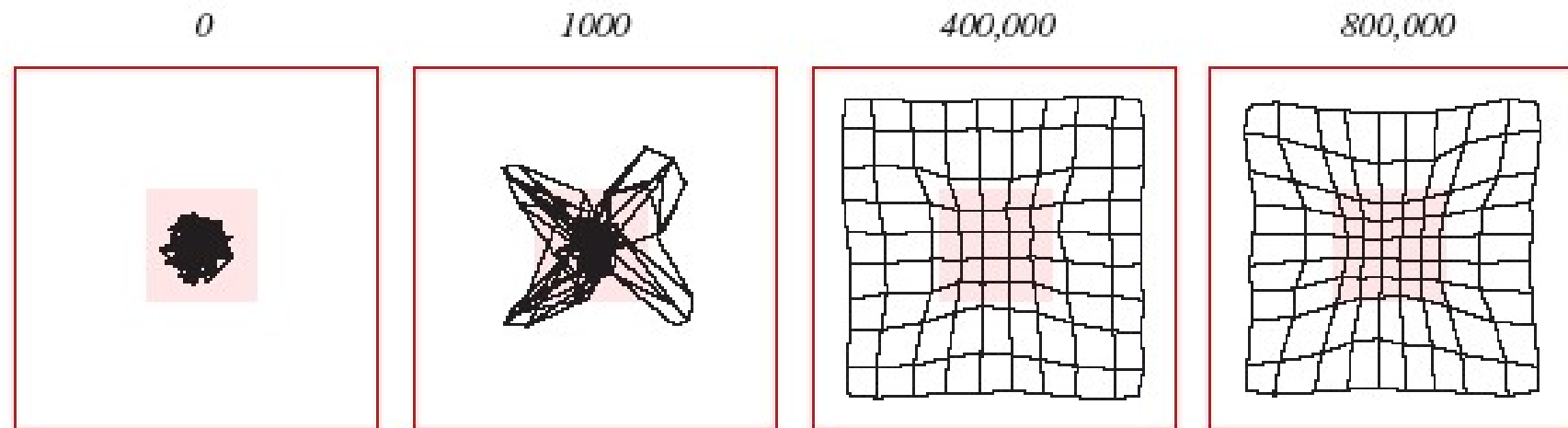


FIGURE 10.33. As in Fig. 10.31 except that the sampling of the input space was not uniform. In particular, the probability density for sampling a point in the central square region (pink) was 20 times greater than elsewhere. Notice that the final map devotes more nodes to this center region than in Fig. 10.31. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.