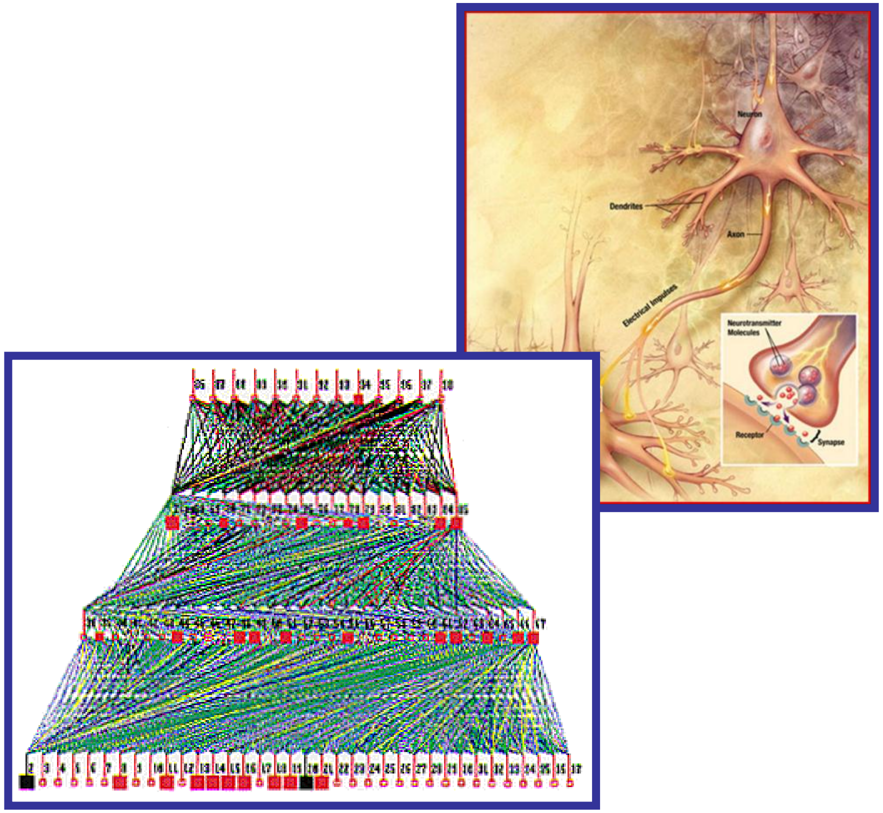
NEURAL NETWORKS

- **Objectives:**
  Feedforward Networks
  Multilayer Networks
  Backpropagation
  Posteriors
  Kernels

- **Resources:**
  DHS: Chapter 6
  AM: Neural Network Tutorial
  NSFC: Introduction to NNs
  GH: Short Courses

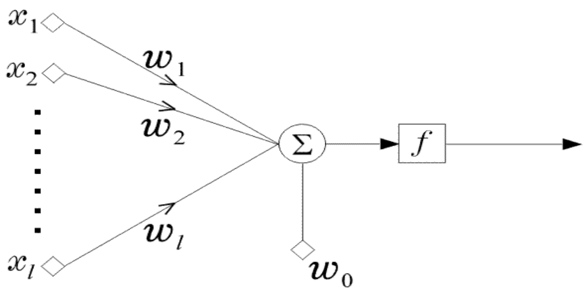


- URL: **.../publications/courses/ece_8443/lectures/current/lecture_17.ppt**

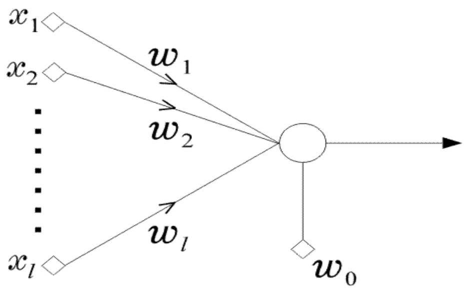
Overview

- **There are many problems for which linear discriminant functions are insufficient for minimum error.**
- **Previous methods, such as Support Vector Machines require judicious choice of a kernel function (though data-driven methods to estimate kernels exist).**
- **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.**
- **There is no automatic method for determining the nonlinearities when no information is provided to the classifier.**
- **Multilayer Neural Networks attempt to learn the form of the nonlinearity from the training data.**
- **These were loosely motivated by attempts to emulate behavior of the human brain, though the individual computation units (e.g., a node) and training procedures (e.g., backpropagation) are not intended to replicate properties of a human brain.**
- **Learning algorithms are generally gradient-descent approaches to minimizing error.**

❖ The perceptron



$$\begin{matrix} > \\ < \end{matrix}\ 0$$



$$\begin{matrix} > \\ < \end{matrix}\ 0$$

w_i's synapses or synaptic weights

w_0 threshold

- The network is called perceptron or neuron
- It is a learning machine that learns from the training vectors via the perceptron algorithm

- The Perceptron Algorithm
  - Assume linearly separable classes, i.e.,

$$\exists \mathbf{w}^* : \mathbf{w}^{*T}\mathbf{x} > 0 \quad \forall \mathbf{x} \in \omega_1$$

$$\mathbf{w}^{*T}\mathbf{x} < 0 \quad \forall \mathbf{x} \in \omega_2$$

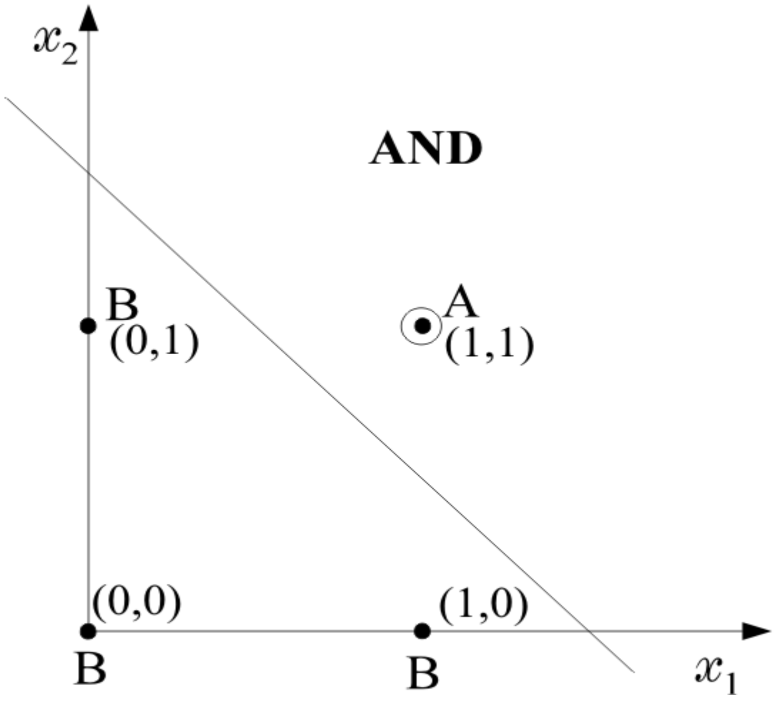
  - The case $\mathbf{w}^{*T}\mathbf{x} + w_0^*$ falls under the above formulation, since
    - $\mathbf{w}' \equiv \begin{bmatrix} w^*_0 \\ \mathbf{w}^* \end{bmatrix}, \quad \mathbf{x}' = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$
    - $\mathbf{w}^{*T}\mathbf{x} + w^*_0 = \mathbf{w}'^T\mathbf{x}' = 0$

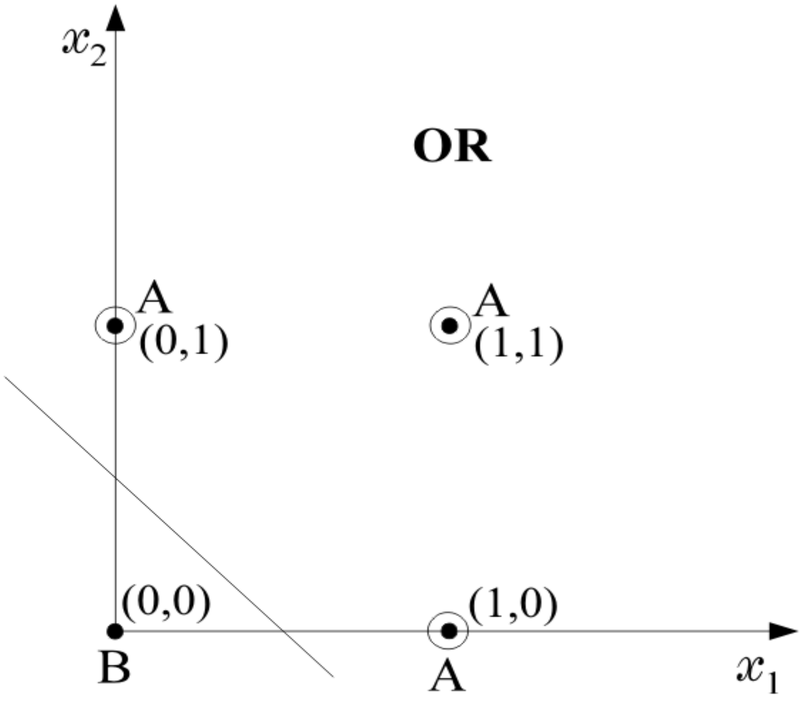
- Our goal: Compute a solution, i.e., a hyperplane $\mathbf{w}$, so that

$$\begin{tikzcd} & \omega_1 \\ \mathbf{w}^T\mathbf{x} > (<) 0 \quad \mathbf{x} \in \arrow[ur] \arrow[dr] & \\ & \omega_2 \end{tikzcd}$$

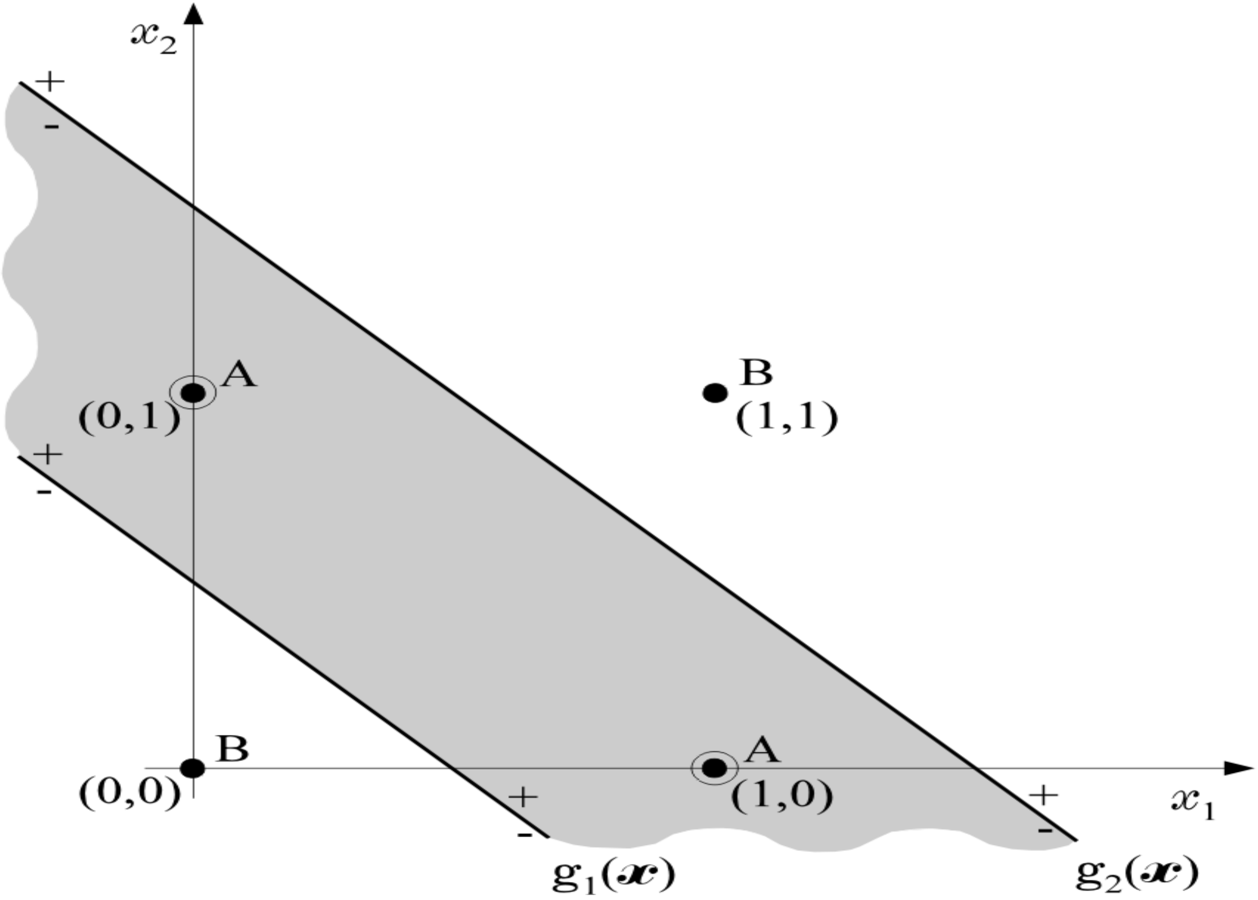
  - The steps
    - Define a cost function to be minimized
    - Choose an algorithm to minimize the cost function
    - The minimum corresponds to a solution

- There is no single line (hyperplane) that separates class A from class B. On the contrary, AND and OR operations are linearly separable problems





- The Two-Layer Perceptron
  - For the XOR problem, draw two, instead, of one lines



- Then class B is located outside the shaded area and class A inside. This is a two-phase design.
  - Phase 1: Draw two lines (hyperplanes)

    $$g_1(\mathbf{x}) = g_2(\mathbf{x}) = 0$$

    Each of them is realized by a perceptron. The outputs of the perceptrons will be

    $$y_i = f(g_i(\mathbf{x})) = \begin{cases} 0 \\ 1 \end{cases} \quad i = 1,\ 2$$

    depending on the position of **x**.

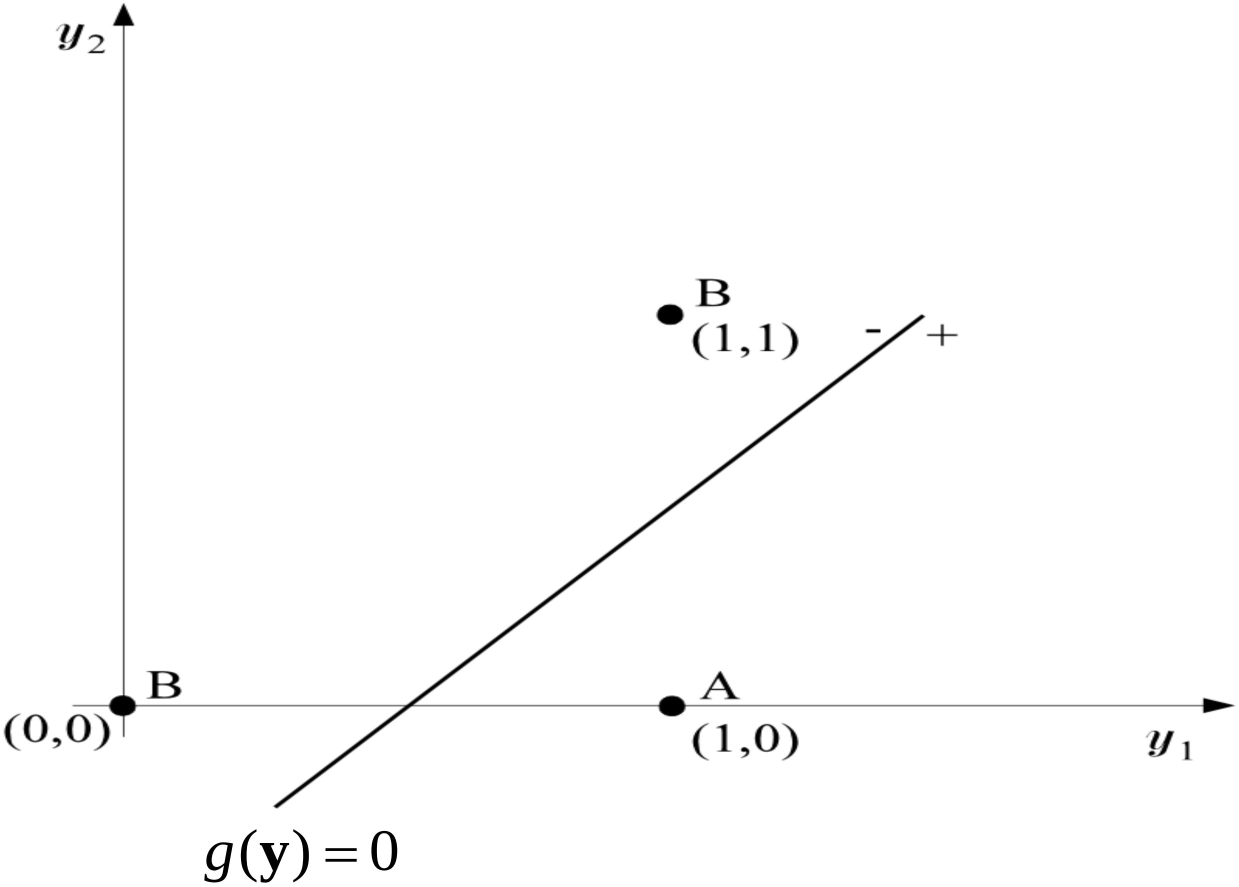
  - Phase 2: Find the position of **x** *w.r.t.* both lines, based on the values of y_1, y_2.

1st phase				**2nd phase**
x_1	x_2	y_1	y_2	
0	0	0(-)	0(-)	B(0)
0	1	1(+)	0(-)	A(1)
1	0	1(+)	0(-)	A(1)
1	1	1(+)	1(+)	B(0)

- Equivalently: The computations of the first phase

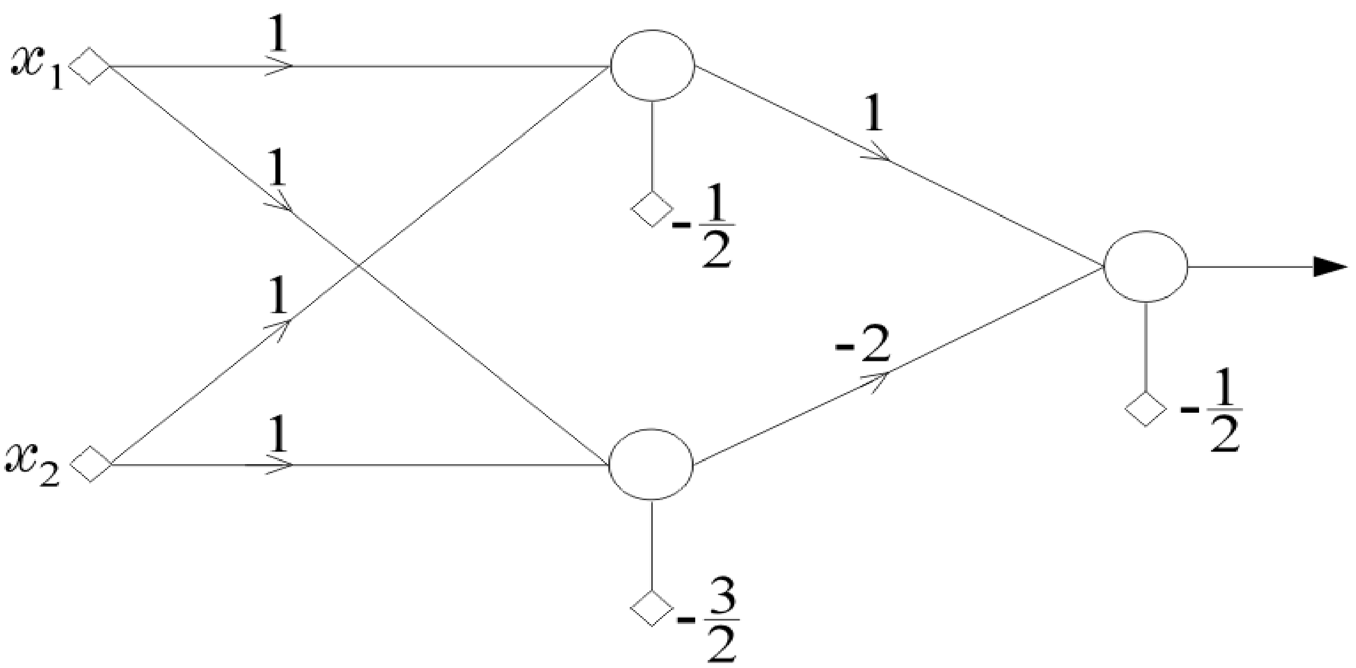
perform a mapping $\mathbf{x} \rightarrow \mathbf{y} = [y_1, \; y_2]^T$

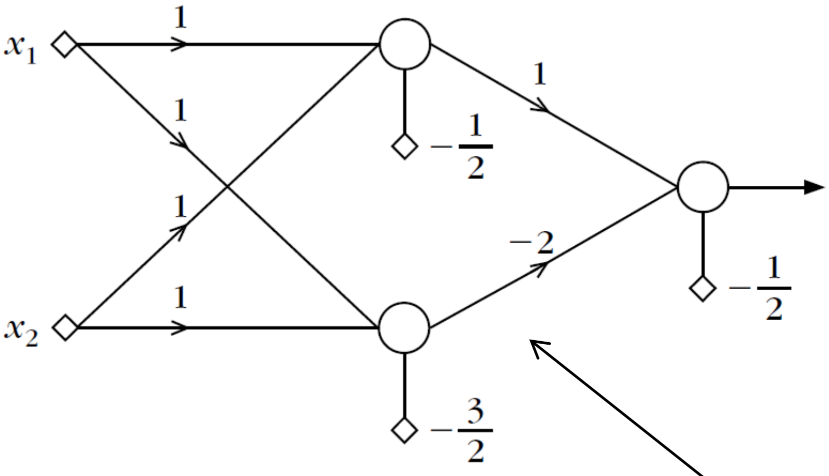
The decision is now performed on the transformed $\mathbf{y}$ data.



This can be performed via a second line, which can also be realized by a perceptron.

- Computations of the first phase perform a mapping that transforms the nonlinearly separable problem to a linearly separable one.
  - The architecture





A two-layer perceptron solving the XOR problem.

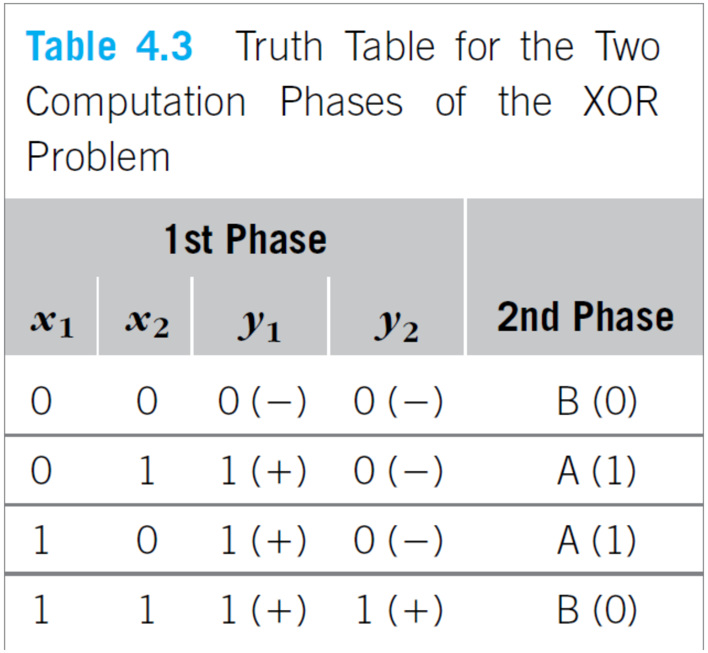
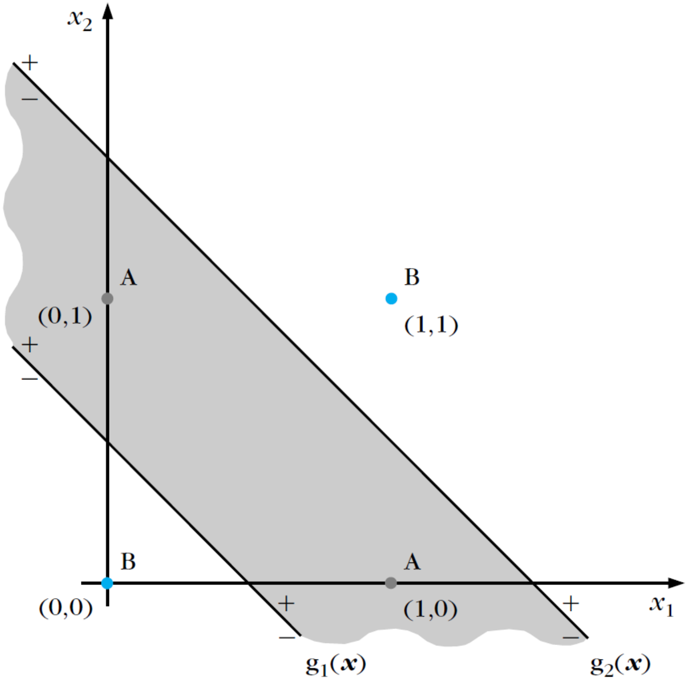
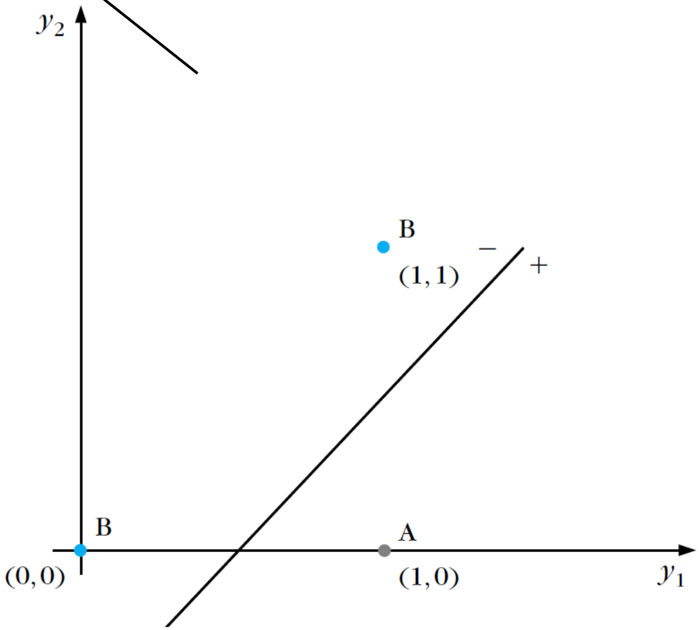


Table 4.3 Truth Table for the Two Computation Phases of the XOR Problem

1st Phase				2nd Phase
x_1	x_2	y_1	y_2	
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)





Definitions

- **A single "bias unit" is connected to each unit other than the input units.**
- **Net activation:** $$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} = \mathbf{w}_{\mathrm{j}}^{\mathrm{t}} \mathbf{x}$$
  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.

- **Each hidden unit emits an output that is a nonlinear function of its activation:** $y_j = f(net_j)$

- **Even though the individual computational units are simple (e.g., a simple threshold), a collection of large numbers of simple nonlinear units can result in a powerful learning machine (similar to the human brain).**

- **Each output unit similarly computes its net activation based on the hidden unit signals as:**
  $$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{w}_{\mathrm{k}}^{\mathrm{t}} \mathbf{y}$$
  where the subscript k indexes units in the output layer and n_H denotes the number of hidden units.

- z_k **will represent the output for systems with more than one output node. An output unit computes** $z_k = f(net_k)$.

Computations

- **The hidden unit** y_1 **computes the boundary:**
  - $\geq 0 \Rightarrow y_1 = +1$

$x_1 + x_2 + 0.5 = 0$

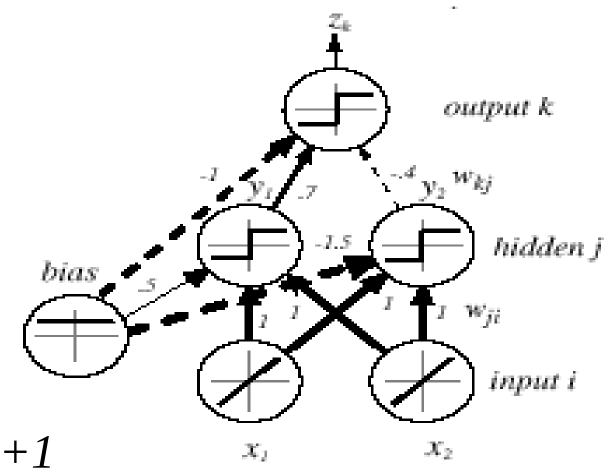
  - $< 0 \Rightarrow y_1 = -1$

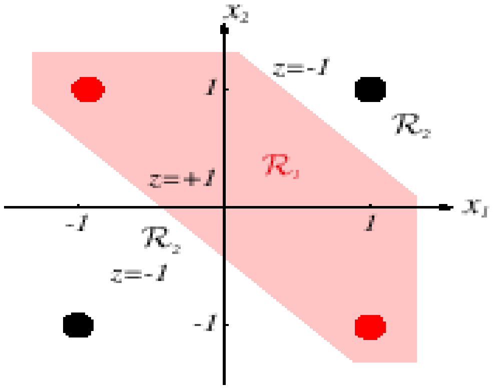
- **The hidden unit** y_2 **computes the boundary:**
  - $\leq 0 \Rightarrow y_2 = +1$
- $x_1 + x_2 - 1.5 = 0$
  - $< 0 \Rightarrow y_2 = -1$
- **The final output unit emits** $z_1 = +1 \Leftrightarrow y_1 = +1$ ***and*** $y_2 = +1$

$z_k = y_1$ **AND NOT** y_2

$= (x_1 \text{ OR } x_2)$ **AND** $\text{NOT } (x_1 \text{ AND } x_2)$

$= x_1 \text{ XOR } x_2$





General Feedforward Operation

- For c output units:

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

- 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.
- 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.
- Can every decision be implemented by a three-layer network?
- 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.”

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

for properly chosen functions δ_j and β_{ij}

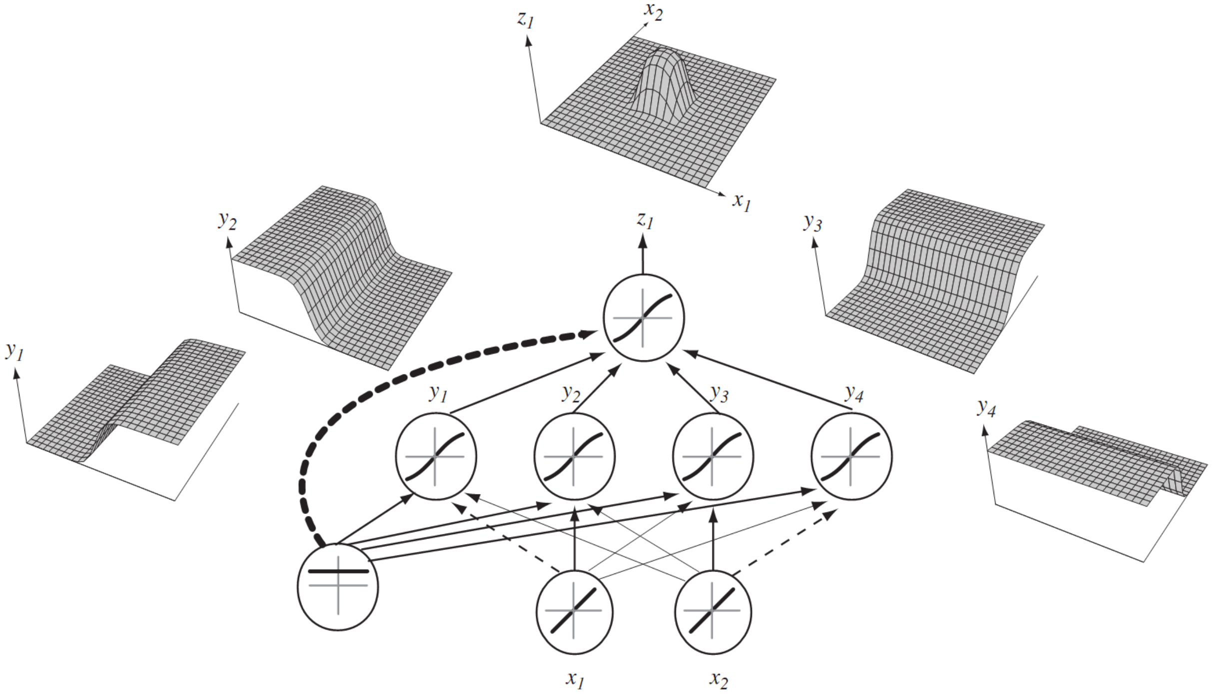
General Feedforward Operation (Cont.)

- **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.**

- **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 based on data; this is the central problem in network-based pattern recognition.**



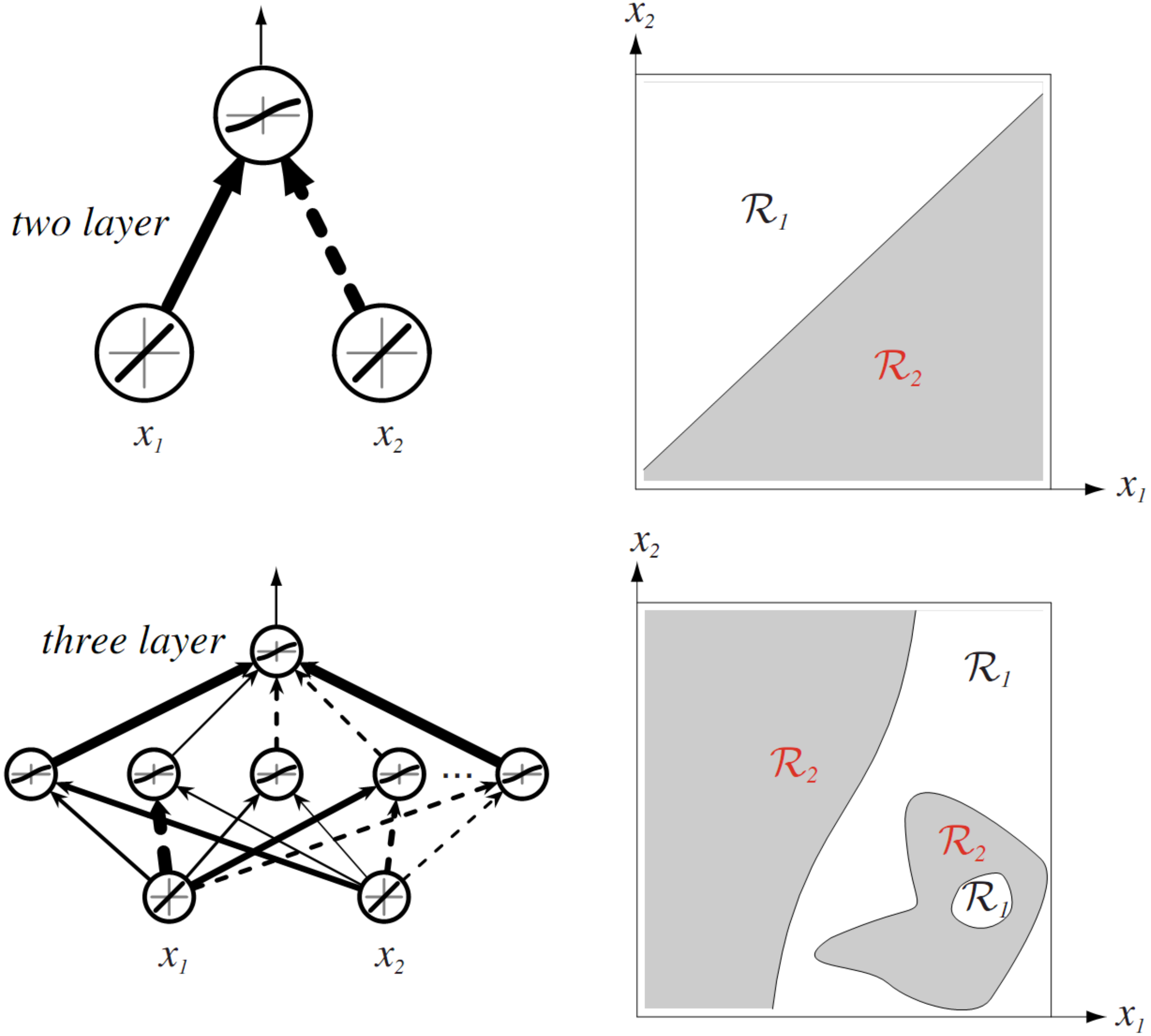
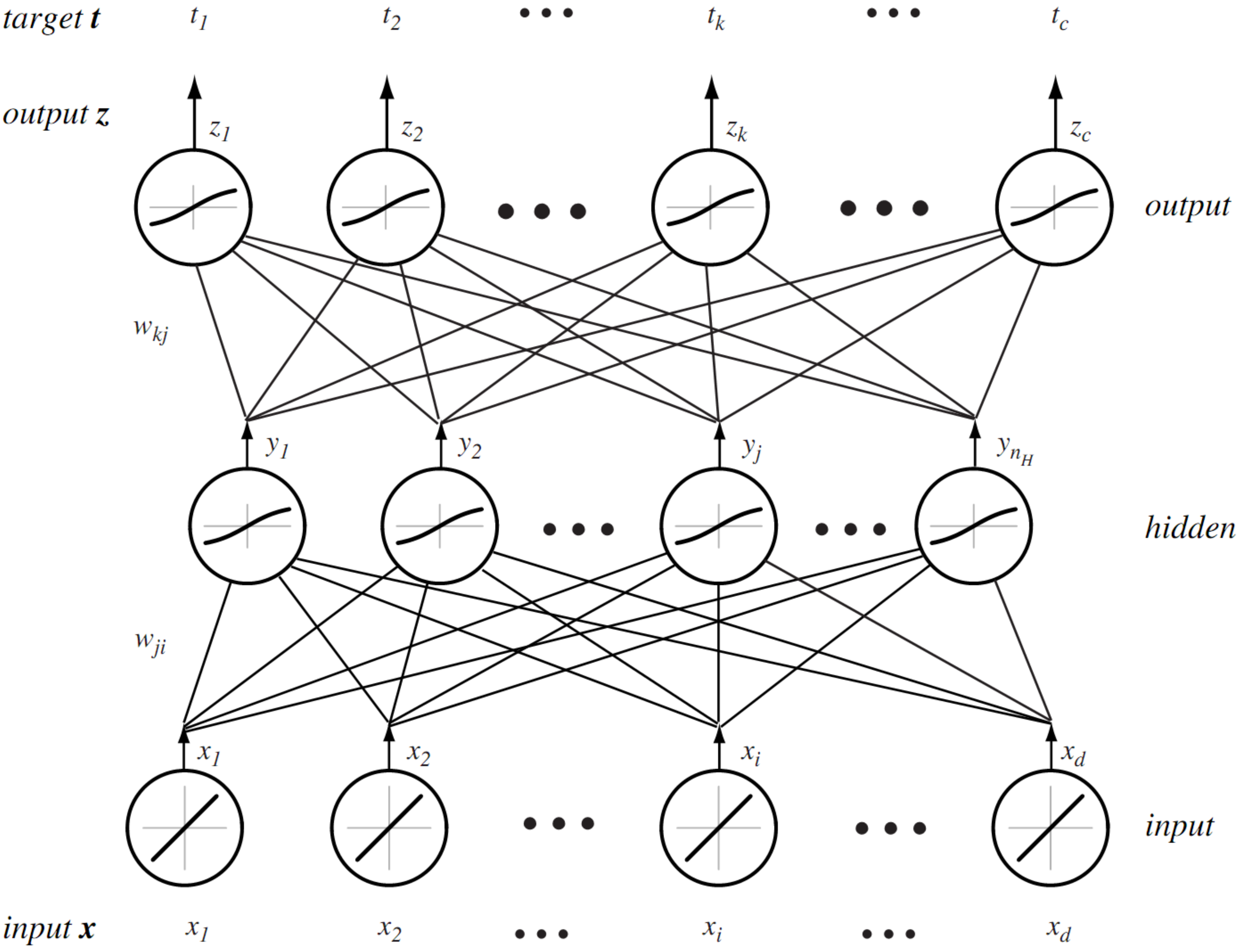


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected.

Backpropagation

- **Any function from input to output can be implemented as a three-layer neural network.**
- **These results are of greater theoretical interest than practical, since the construction of such a network requires the nonlinear functions and the weight values which are unknown!**
- **Our 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.**
- **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."**
- **Networks have two modes of operation:**
  - **Feedforward: consists of presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units.**
  - **Learning: 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.**

Backpropagation (Cont.)



Network Learning

- Let t_k be the 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.
- Training error: $$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$$
- 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: $$\Delta\mathbf{w} = -\eta\frac{\partial J}{\partial w}$$

    where η is the learning rate which indicates the relative size of the change in weights.
  - The weight are updated using: $\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m)$.
- 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 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: $$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}.\frac{\partial z_k}{\partial net_k} = (t_k - z_k)f'(net_k)$$

Network Learning (Cont.)

- **Since** net_k **=** $\mathbf{w_k^t.y}$**:** $\frac{\partial net_k}{\partial w_{kj}} = y_j$

- **Therefore, the weight update (or learning rule) for the hidden-to-output weights is:** $\Delta w_{kj} = \eta\delta_k y_j = \eta(t_k - z_k) f'(net_k) y_j$

- **The error on the input-to-hidden units is:** $$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\cdot\overset{f'(net_j)}{\boxed{\frac{\partial y_j}{\partial net_j}}}\cdot\boxed{\frac{\partial net_j}{\partial w_{ji}}} \rightarrow x_i$$

- **The first term is given by:** $$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$ $$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\cdot\frac{\partial net_k}{\partial y_j} = \underbrace{-\sum_{k=1}^{c}(t_k - z_k) f'(net_k) w_{kj}}$$

- **We define the sensitivity for a hidden unit:** $$\delta_j \equiv f'(net_j)\boxed{\sum_{k=1}^{c} w_{kj}\delta_k}$$

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

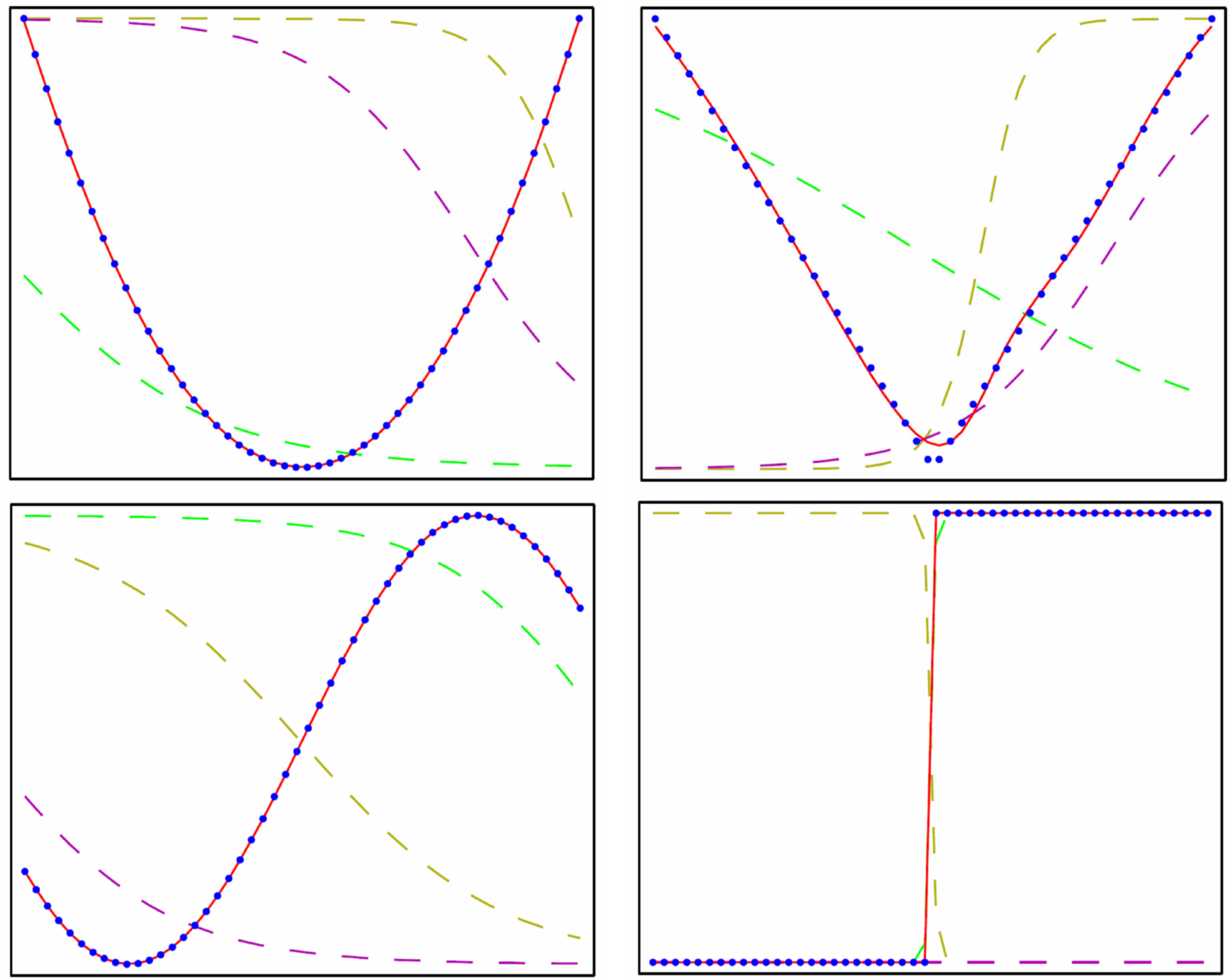
Stochastic Back Propagation

- **Starting 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 $m \leftarrow m + 1$
                  $x^{\mathrm{m}} \leftarrow$ randomly chosen pattern
                  $w_{\mathrm{ji}} \leftarrow w_{\mathrm{ji}} + \eta\delta_{\mathrm{j}}x_{\mathrm{i}}$; $w_{\mathrm{kj}} \leftarrow w_{\mathrm{kj}} + \eta\delta_{\mathrm{k}}y_{\mathrm{j}}$
              until $\|\nabla J(\mathbf{w})\| < \theta$
  return w
End
```

Algorithm 2 (Batch backpropagation)

```
begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, r \leftarrow 0$
    do $r \leftarrow r + 1$ (increment epoch)
        $m \leftarrow 0;\ \Delta w_{ij} \leftarrow 0;\ \Delta w_{jk} \leftarrow 0$
        do $m \leftarrow m + 1$
            $\mathbf{x}^m \leftarrow$ select pattern
            $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i;\ \ \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$
        until $m = n$
        $w_{ij} \leftarrow w_{ij} + \Delta w_{ij};\ \ w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$
    until $\nabla J(\mathbf{w}) < \theta$
return $\mathbf{w}$
end
```



- NNs can learn any (or at least: many) functions
- 50 points sampled (blue dots) from 4 different input functions
- Two layer network, tanh activation, linear output, 3 hidden units, output hidden units shown dashed
- Output network in red

Stopping Criterion

- One example of a stopping algorithm is to terminate the algorithm 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. Most gradient descent approaches can be applied.
- 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: $J = \sum_{p=1}^{n} J_p$
- 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 decreases.

Learning Curves

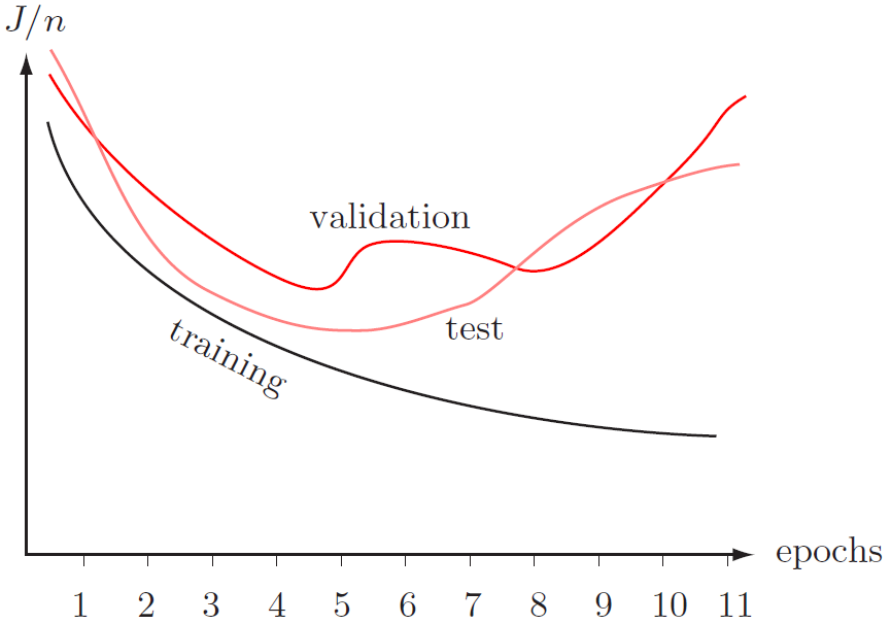
- **Before training starts, the error on the training set is high; through the learning process, the error becomes smaller.**

- **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.**

- **A validation set is used in order to decide when to stop training; 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”



Convergence Issues

- A neural network may converge to a bad solution
  - Train several neural networks from different initial conditions
- The convergence is slow
  - Practical techniques
  - Variations of basic backpropagation algorithms

Practical Techniques for Improving BP

- Transfer functions
  - Prior information to choose appropriate transfer functions
  - Parameters for the sigmoid function
- Scaling input
  - We can standardize each feature component to have zero mean and the same variance
- Target values
  - For pattern recognition applications, use 1 for the target category and -1 for non-target category
- Training with noise

Practical Techniques for Improving BP

- Manufacturing data
  - – If we have knowledge about the sources of variation among the inputs, we can manufacture training data
    - For face detection, we can rotate and enlarge / shrink the training images
- Initializing weights
  - – If we use standardized data, we want positive and negative weights as well from a uniform distribution
    - Uniform learning

Practical Techniques for Improving BP

- Training protocols
  - Epoch corresponds to a single presentation of all the patterns in the training set
  - Stochastic training
    - Training samples are chosen randomly from the set and the weights are updated after each sample
  - Batch training
    - All the training samples are presented to the network before weights are updated
  - On-line training
    - Each training sample is presented once and only once
    - There is no memory for storing training samples

Speeding up Convergence

- Heuristics
  - Momentum
  - Variable learning rate
  - delta-delta rule and delta-bar-delta rule
- Conjugate gradient
- Quickprop
- Second-order methods
  - Newton’s method
  - Levenberg-Marquardt algorithm

The Cost Function Choice

- The least squares optimal estimate of the posterior probability
- The cross-entropy cost function
- minimizing the classification error
- deterministic annealing procedure
- …
- Radial basis function networks (RBF)
- Special bases
- Time delayneural networks (TDNN)
- Recurrent networks
- Counterpropagation
- Cascade-Correlation

Summary

- **Introduced the concept of a feedforward neural network.**
- **Described the basic computational structure.**
- **Described how to train this network using backpropagation.**
- **Discussed stopping criterion.**
- **Described the problems associated with learning, notably overfitting.**
- **What we didn't discuss:**
  - **Many, many forms of neural networks. Three important classes to consider:**
    - **Basis functions:** $z_k = \sum_{j=0}^{n_H} w_{kj}\phi_j(\mathbf{x})$
    - **Boltzmann machines: a type of simulated annealing stochastic recurrent neural network.**
    - **Recurrent networks: used extensively in time series analysis.**
  - **Posterior estimation: in the limit of infinite data the outputs approximate a true a posteriori probability in the least squares sense.**
  - **Alternative training strategies and learning rules.**