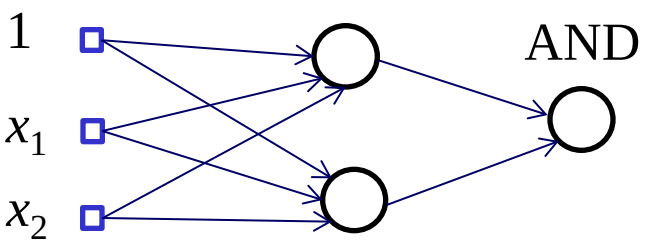
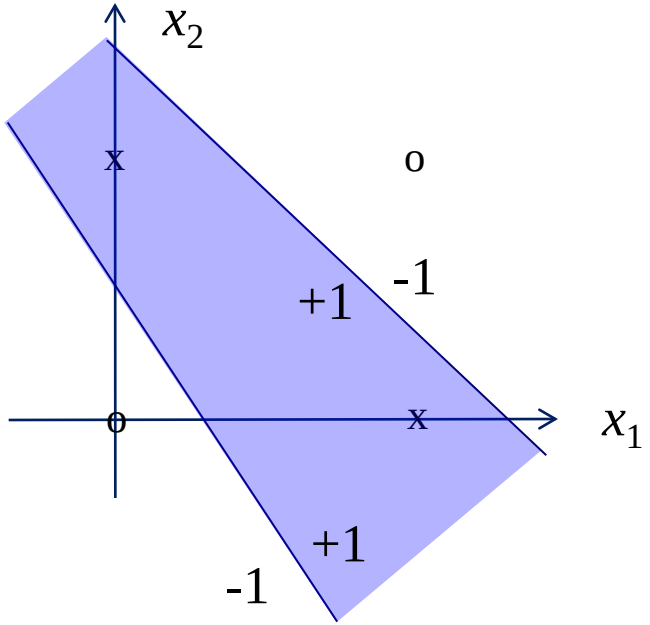
CSE 5526: Introduction to Neural Networks

Multilayer Perceptrons (MLPs)

Motivation

- Multilayer networks are more powerful than single-layer nets
  - Example: XOR problem

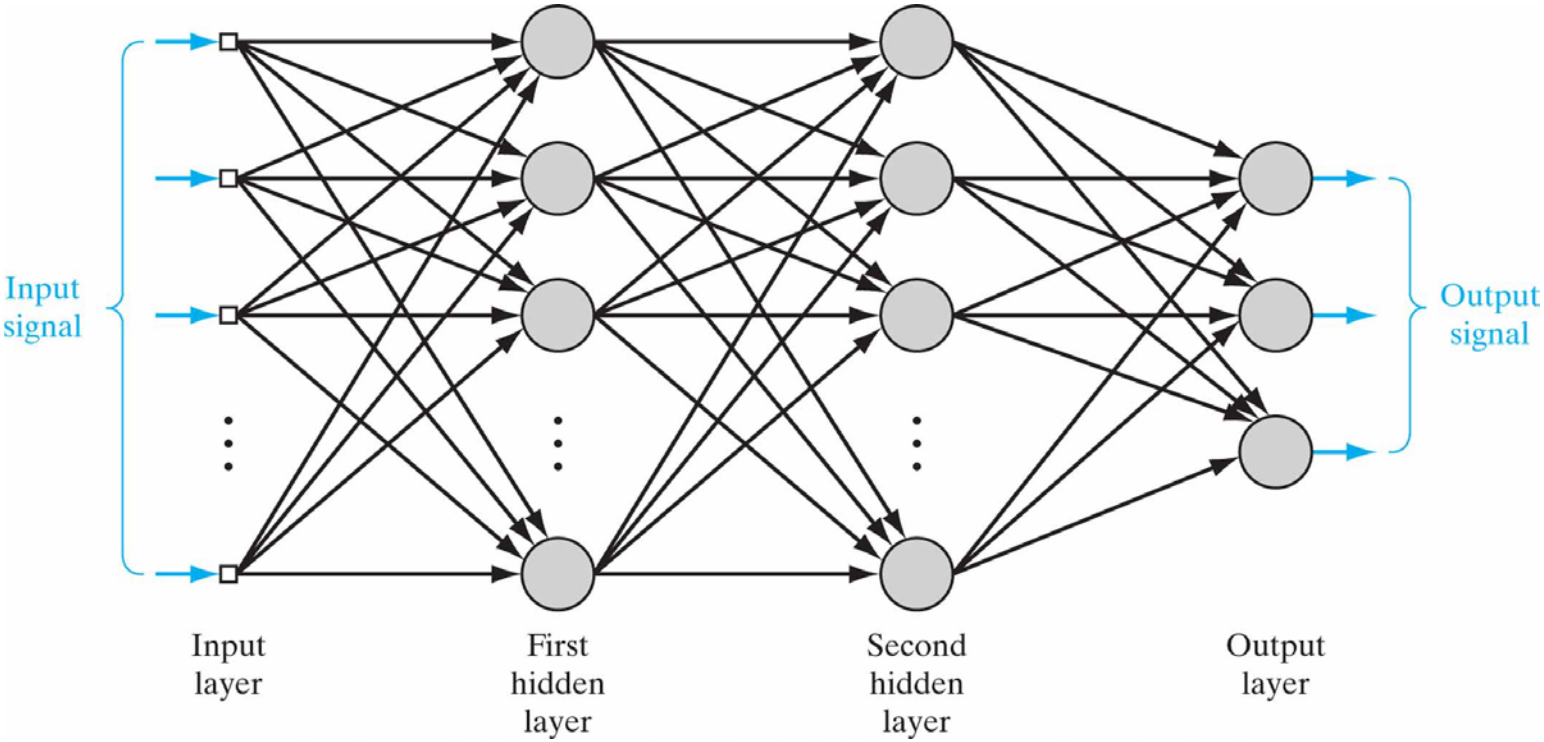




Power of nonlinearity

- For linear neurons, a multilayer net is equivalent to a single-layer net. This is not the case for nonlinear neurons
  - Why?

MLP architecture



Multi-layer perceptron

- Think of an MLP as a complicated, non-linear function of its input parametrized by $\boldsymbol{w}$:

$$\mathbf{y} = F(\boldsymbol{x}; \boldsymbol{w})$$

- Note that “Multi-layer perceptron” is a bit of a misnomer because they use a continuous activation function

MLP Training

- Given a set of training data $\{\boldsymbol{x}_p, \boldsymbol{d}_p\}$ can we adjust $\boldsymbol{w}$ so that the network is optimal?
- Optimal with respect to what criterion?
  - Must define error criterion btwn $\boldsymbol{y}_p = F(\boldsymbol{x}_p; \boldsymbol{w})$ and $\boldsymbol{d}_p$
  - We will use the mean square error for now, but others are possible (and often preferable)
- Goal find $\boldsymbol{w}$ that minimizes

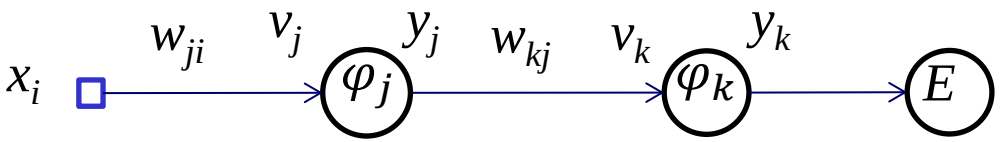
$$\bar{E}(\boldsymbol{w}) = \sum_p E_p(\boldsymbol{w}) = \sum_p \frac{1}{2} \left\| \boldsymbol{d}_p - F\left(\boldsymbol{x}_p; \boldsymbol{w}\right) \right\|^2$$

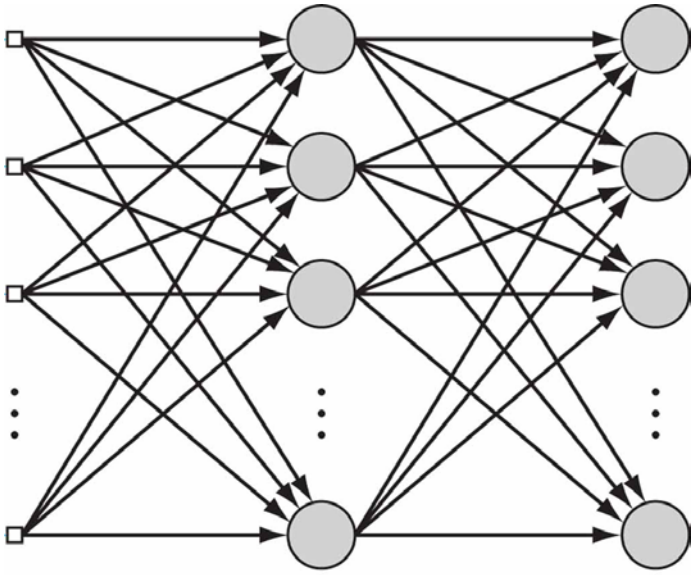
Backpropagation

- Because $\bar{E}(\boldsymbol{w})$ is still a complication non-linear function, we will optimize it using gradient descent
- Because of the structure of MLPs, we can compute the gradient of $E_p(\boldsymbol{w})$ very efficiently using the **backpropagation** algorithm
- Backpropagation computes the gradient of each layer recursively based on subsequent layers
- Because this is $E_p(\boldsymbol{w})$ and not $\bar{E}(\boldsymbol{w})$, we will be using **stochastic** gradient descent

Notation

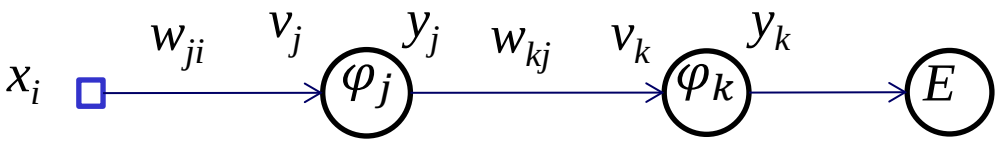
- Notation for one hidden layer (drop p for now)





Notation

- Notation for one hidden layer (drop p for now)

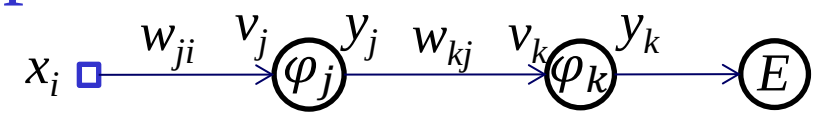


$$y_k = \varphi\left(\sum_j w_{kj}\,\varphi\left(\sum_i w_{ji}x_i\right)_j\right)_k$$

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_k (d_k - y_k)^2$$

- Keep in mind during the derivation:
  - How would changing $E_p(\boldsymbol{w})$ affect the derivation?
  - How would changing $\varphi(\boldsymbol{v})$ affect the derivation?

Backprop



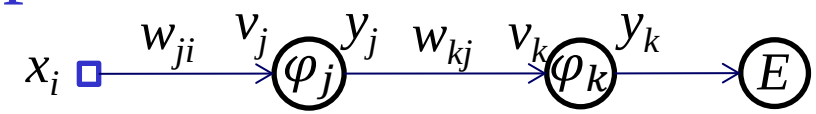
$$E(\boldsymbol{w}) = \frac{1}{2}\sum_k (d_k - y_k)^2$$

$$= \frac{1}{2}\sum_k \left(d_k - \underbrace{\varphi\left(\overbrace{\sum_j w_{kj} y_j}^{v_k} \right)_k}_{y_k} \right)^2$$

- Then, to adjust the hidden-output weights

$$\frac{\partial}{\partial w_{kj}} E(\boldsymbol{w}) = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial v_k} \frac{\partial v_k}{\partial w_{kj}}$$

Backprop



$$\frac{\partial E}{\partial y_k} = -(d_k - y_k) = -e_k$$

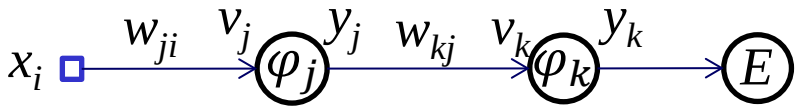
$$\frac{\partial y_k}{\partial v_k} = \frac{\partial}{\partial v_k}\varphi(v_k) = \varphi'(v_k)$$

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

So

$$\frac{\partial}{\partial w_{kj}} E(\boldsymbol{w}) = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial w_{kj}} = -e_k \varphi'(v_k) y_j$$

Backprop



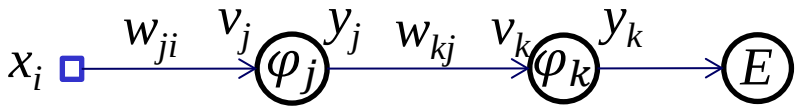
- Hence, to update the hidden-output weights

$$w_{kj}(n+1) = w_{kj}(n) - \eta \frac{\partial E}{\partial w_{kj}}$$

$$= w_{kj}(n) + \eta \underbrace{e_k \varphi'(v_k)}_{\delta_k} y_j$$

$$= w_{kj}(n) + \eta \delta_k y_j \qquad \text{(δ rule)}$$

Backprop



- For the input-hidden weights,

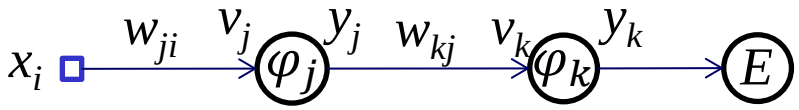
$$\frac{\partial}{\partial w_{ji}} E(\boldsymbol{w}) = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}}$$

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j} \frac{1}{2} \sum_k \left(d_k - \varphi \left(\overbrace{\sum_j w_{kj} y_j}^{v_k} \right)_k \right)^2$$

$$= -\sum_k (d_k - y_k) \varphi'(v_k) w_{kj}$$

$$\frac{\partial y_j}{\partial v_j} = \varphi'(v_j) \qquad \frac{\partial v_j}{\partial w_{ji}} = x_i$$

Backprop



- So

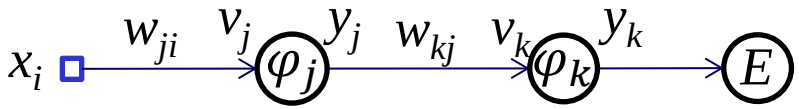
$$\frac{\partial}{\partial w_{ji}} E(\boldsymbol{w}) = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}}$$

$$= -\sum_k \underbrace{(d_k - y_k)\varphi'(v_k)}_{\delta_k} w_{kj} \varphi'(v_j) x_i$$

$$= -\left(\underbrace{\sum_k \delta_k w_{kj}}_{e_j} \right) \varphi'(v_j) x_i$$

$$= -e_j \varphi'(v_j) x_i$$

Backprop



- Hence, to update the input-hidden weights

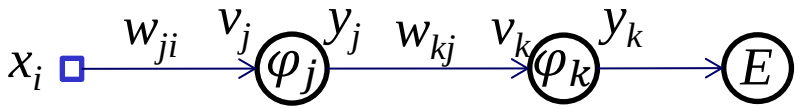
$$w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial E}{\partial w_{ji}}$$

$$= w_{ji}(n) + \eta \underbrace{e_j \varphi'(v_j)}_{\delta_j} x_i$$

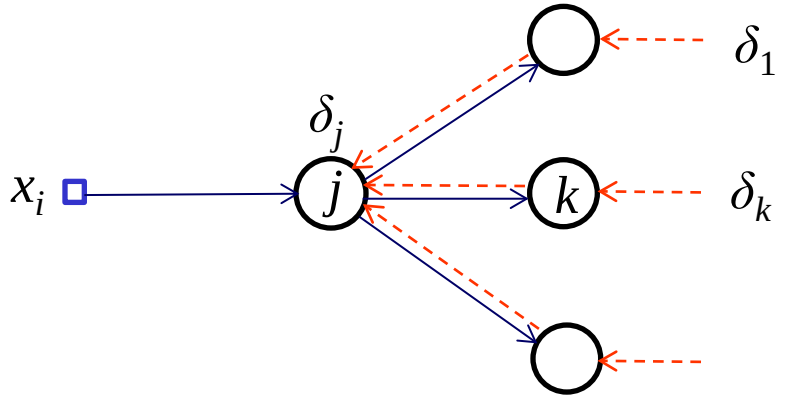
$$= w_{ji}(n) + \eta \delta_j x_i$$

- The above is called the generalized δ rule

Backprop

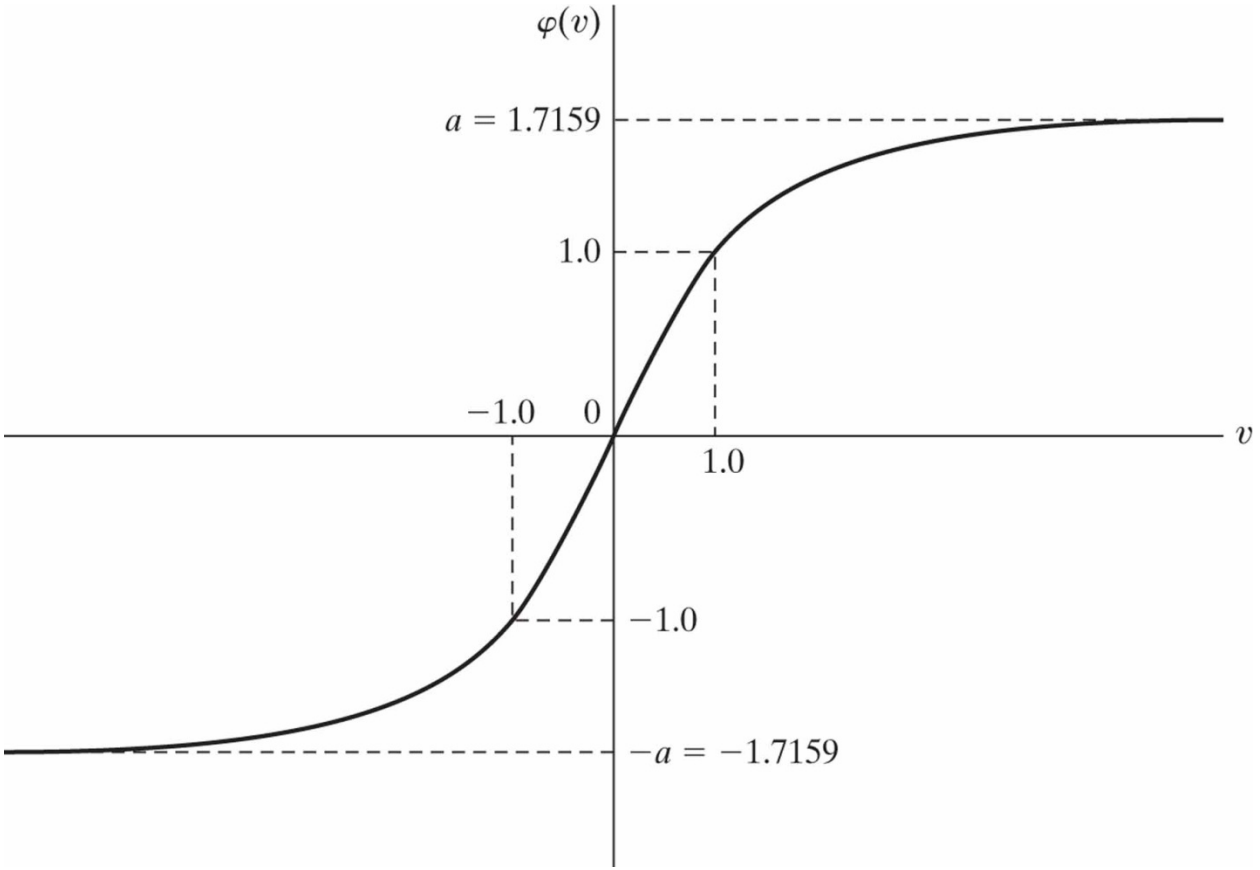


- Illustration of the generalized δ rule,

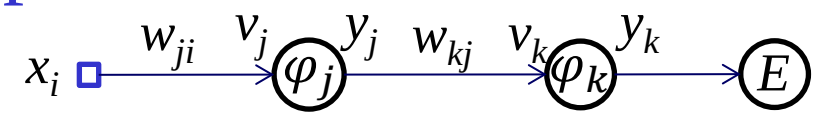


  - The generalized δ rule gives a solution to the credit (blame) assignment problem

Hyperbolic tangent function



Backprop



- For the logistic sigmoid activation, we have

$$\varphi'(v) = a\varphi(v)[1-\varphi(v)]$$

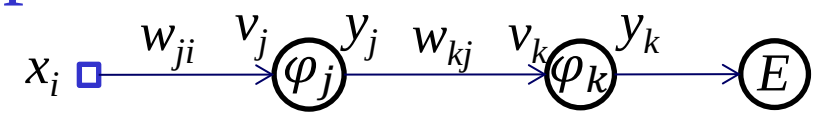
  - hence

$$\delta_k = e_k[ay_k(1-y_k)]$$

$$= ay_k[1-y_k][d_k - y_k]$$

$$\delta_j = ay_j[1-y_j]\sum_k w_{kj}\delta_k$$

Backprop



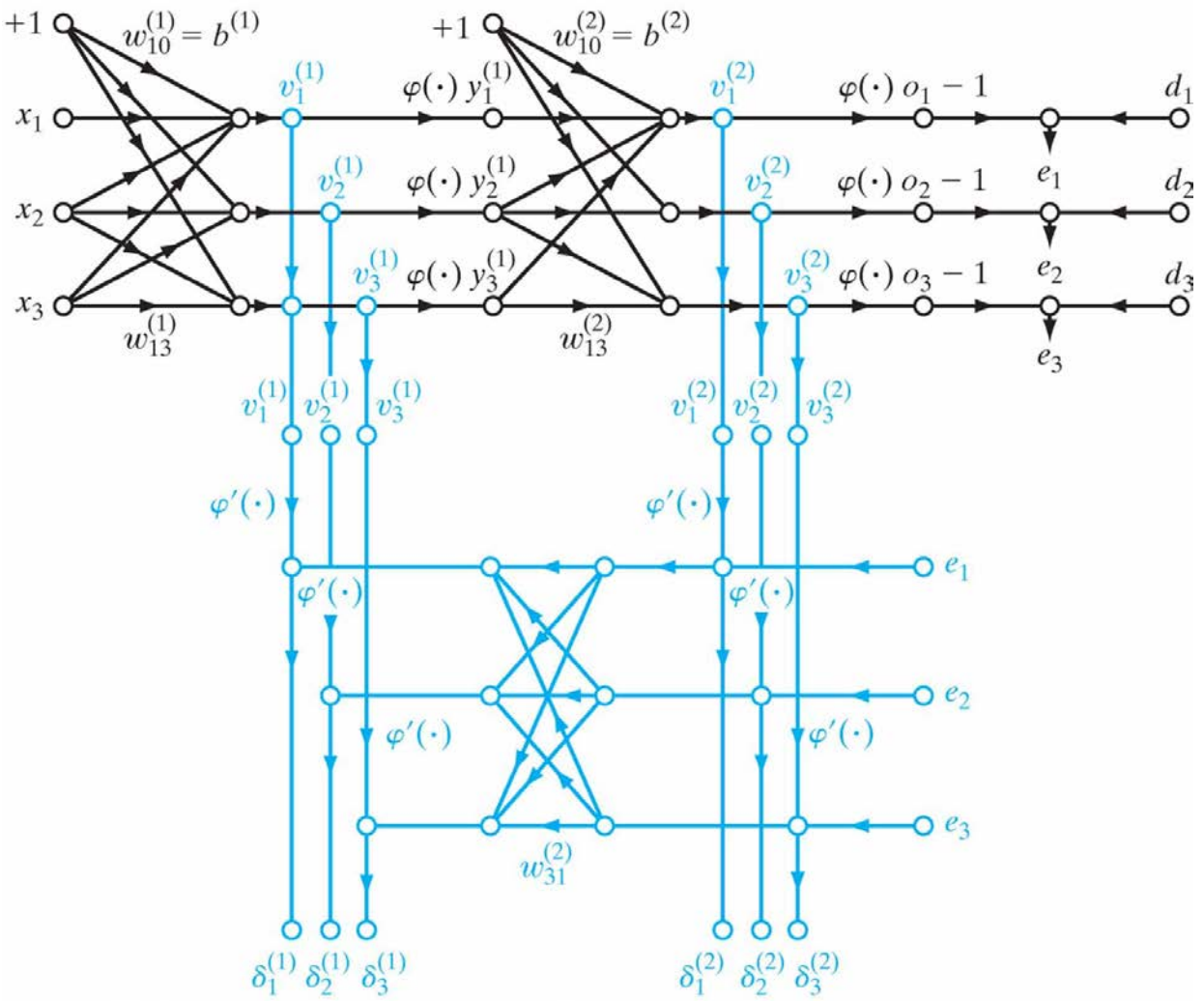
In summary:

$$\frac{\partial}{\partial w_{kj}} E(\boldsymbol{w}) = -e_k \varphi'(v_k) y_j$$

$$\frac{\partial}{\partial w_{ji}} E(\boldsymbol{w}) = -e_j \varphi'(v_j) x_i$$

- Backprop learning is local, concerning "presynaptic" and "postsynaptic" neurons only
- How would changing $E(\boldsymbol{w})$ affect the derivation?
- How would changing $\varphi(\boldsymbol{v})$ affect the derivation?

Backprop illustration



Backprop

- Extension to more hidden layers is straightforward. In general we have

$$\Delta w_{ji}(n) = \eta \delta_j y_i$$

  - The δ rule applies to the output layer and the generalized δ rule applies to hidden layers, layer by layer from the output end.
  - The entire procedure is called backpropagation (error is back propagated from the outputs to the inputs)

MLP design parameters

- Several parameters to choose when designing an MLP (best to evaluate empirically)
- Number of hidden layers
- Number of units in each hidden layer
- Activation function
- Error function

Universal approximation theorem

- MLPs can learn to approximate any function, given sufficient layers and neurons (an existence proof)
- At most two hidden layers are sufficient to approximate any function. One hidden layer is sufficient for any continuous function

Optimization tricks

- For a given network, local minima of the cost function are possible
- Many tricks exist to try to find better local minima
  - Momentum: mix in gradient from step $n - 1$
  - Weight initialization: small random values
  - Stopping criterion: early stopping
  - Learning rate annealing: start with large η, slowly shrink
  - Second order methods: use a separate η for each parameter or pair of parameters based on local curvature
  - Randomization of training example order
  - Regularization, i.e., terms in $E(\boldsymbol{w})$ that only depend on $\boldsymbol{w}$

Learning rate control: momentum

- To ease oscillating weights due to large η, some inertia (momentum) of weight update is added

$$\Delta w_{ji}(n) = \eta \delta_j y_i + \alpha \Delta w_{ji}(n-1), \qquad 0 < \alpha < 1$$

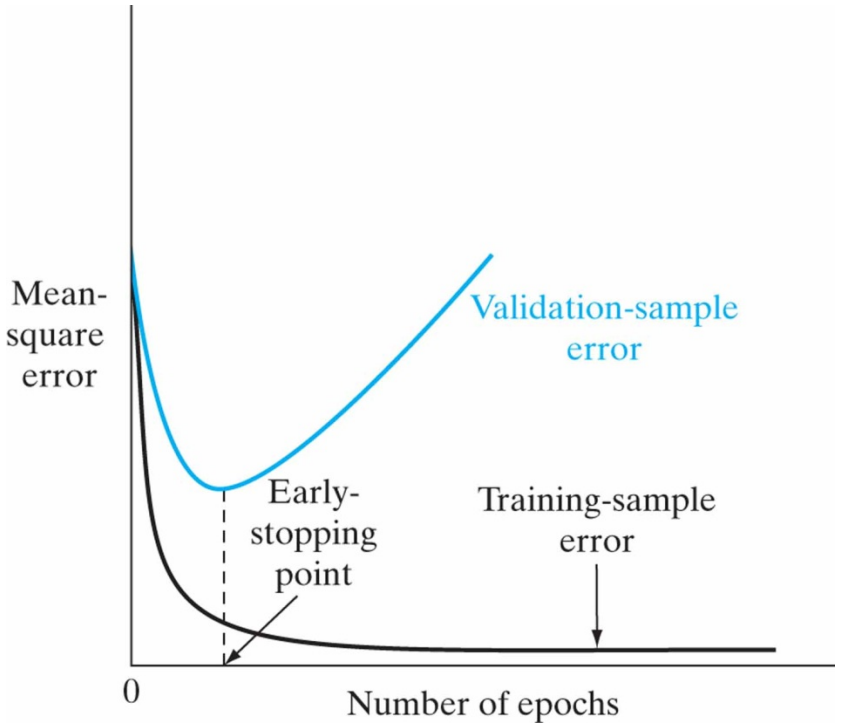
  - In the downhill situation, $\Delta w_{ji}(n) \approx \frac{\eta}{1-\alpha} \delta_j y_i$
    - thus accelerating learning by a factor of $1/(1-\alpha)$
  - In the oscillating situation, it smooths weight change, thus stabilizing oscillations

Weight initialization

- To prevent saturating neurons and break symmetry that can stall learning, initial weights (including biases) are typically randomized to produce zero mean and activation potentials away from saturation parts of the activation function
  - For the hyperbolic tangent activation function, avoiding saturation can be achieved by initializing weights so that the variance equals the reciprocal of the number of weights of a neuron

Stopping criterion

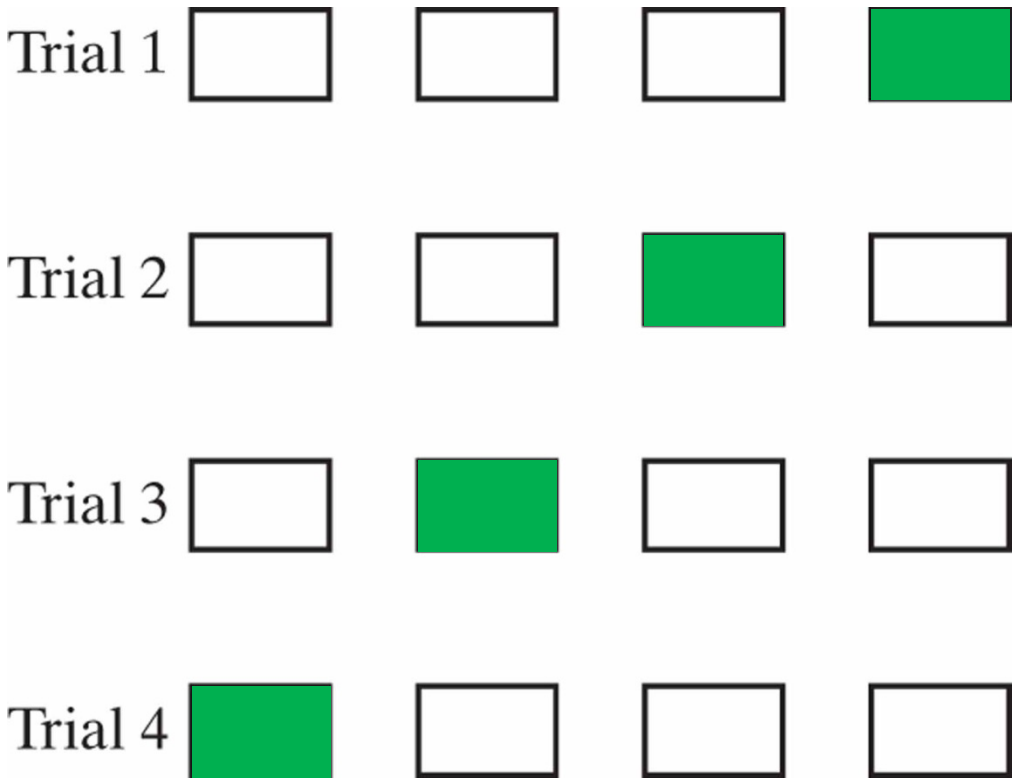
- One could stop after a predetermined number of epochs or when the MSE decrease is below a given criterion
- Early stopping with cross validation: keep part of the training set, called validation subset, as a test for generalization performance



Selecting model parameters: (cross-)validation

- Must have separate training, validation, and test datasets to avoid over-confidence, over-fitting
- When lots of data is available, have dedicated sets
- When data is scarce, use cross-validation
  - Divide the entire training sample into an estimation subset and a validation subset (e.g. 80/20 split)
  - Rotate through 80/20 splits so that every point is tested on once

Cross validation illustration



MLP applications

- Task: Handwritten zipcode recognition (1989)
- Network description
  - Input: binary pixels for each digit
  - Output: 10 digits
  - Architecture: 4 layers (16x16–12x8x8–12x4x4–30–10)
- Each feature detector encodes only one feature within a local input region. Different detectors in the same module respond to the same feature at different locations through weight sharing. Such a layout is called a convolutional net

Zipcode recognizer architecture

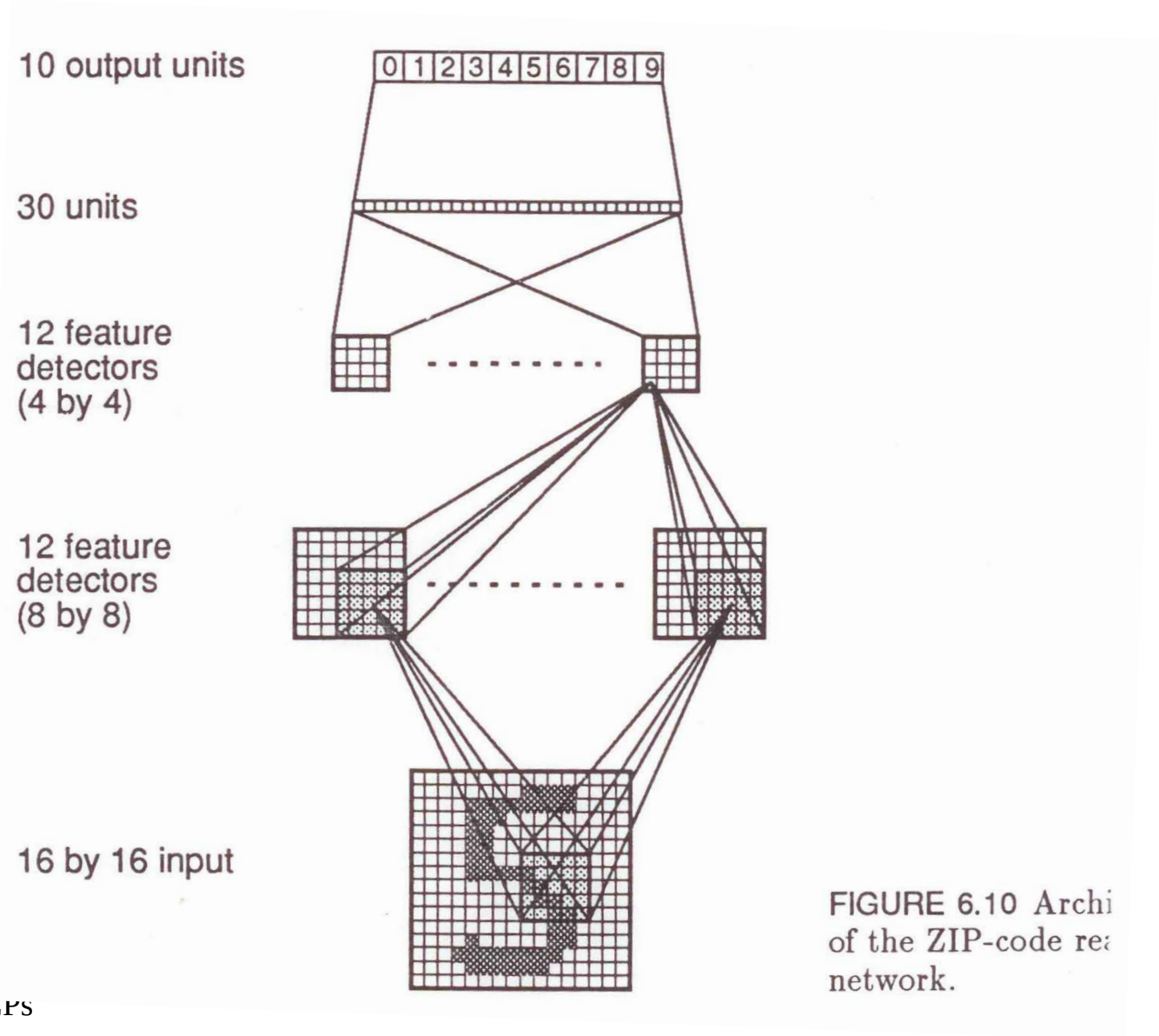


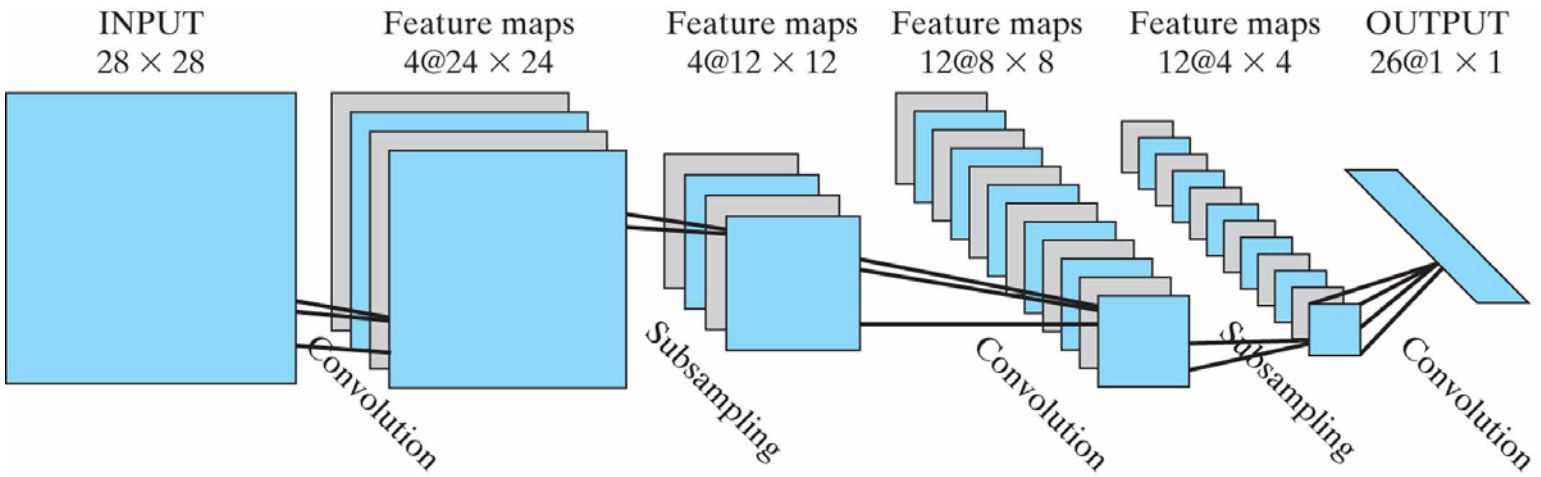
FIGURE 6.10 Archi
of the ZIP-code re
network.

Zipcode recognition (cont.)

- Performance: trained on 7300 digits and tested on 2000 new ones
  - Achieved 1% error on the training set and 5% error on the test set
  - If allowing rejection (no decision), 1% error on the test set
  - The task is not easy (see a handwriting example)
- Remark: constraining network design is a way of incorporating prior knowledge about a specific problem
  - Backprop applies whether or not the network is constrained

Letter recognition example

- The convolutional net has been subsequently applied to a number of pattern recognition tasks with state-of-the-art results
  - Handwritten letter recognition



Automatic driving

- ALVINN (automatic land vehicle in a neural network)

  - One hidden layer, one output layer
  - Five hidden nodes, 32 output nodes (steer left – steer right)
  - 960 inputs (30 x 32 image intensity array)
  - 5000 trainable weights
- Later success of Stanley (won $2M DARPA Grand Challenge in 2005)

Other MLP applications

- NETtalk, a speech synthesizer
- GloveTalk, which converts hand gestures to speech