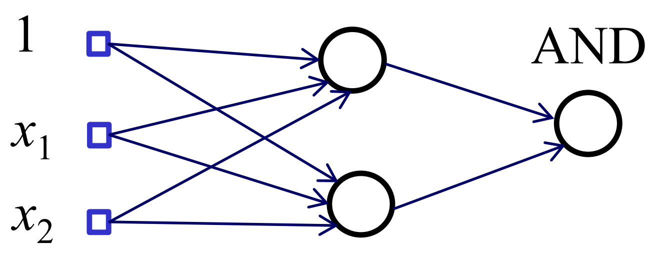
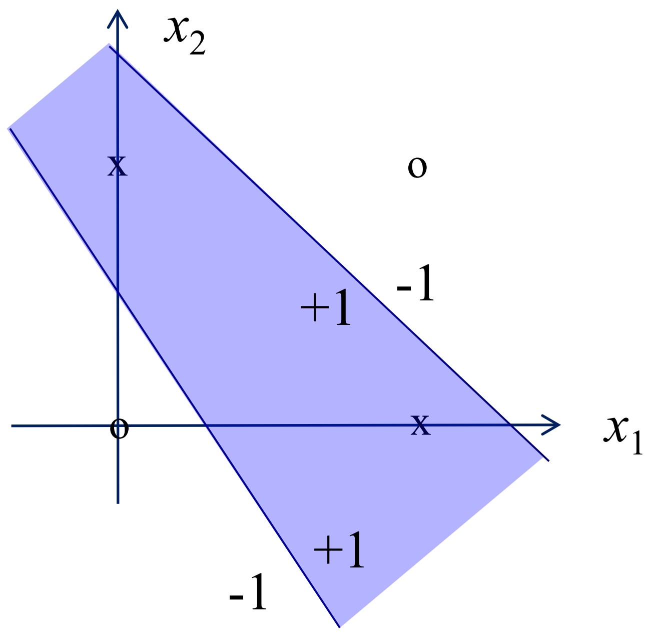
CSE 5526: Introduction to Neural Networks

Multilayer Perceptrons (MLPs)

Motivation

- Multilayer networks are more powerful than single-layer nets
  - Example: XOR problem cannot be solved by a perceptron but is solvable with a 2-layer perceptron

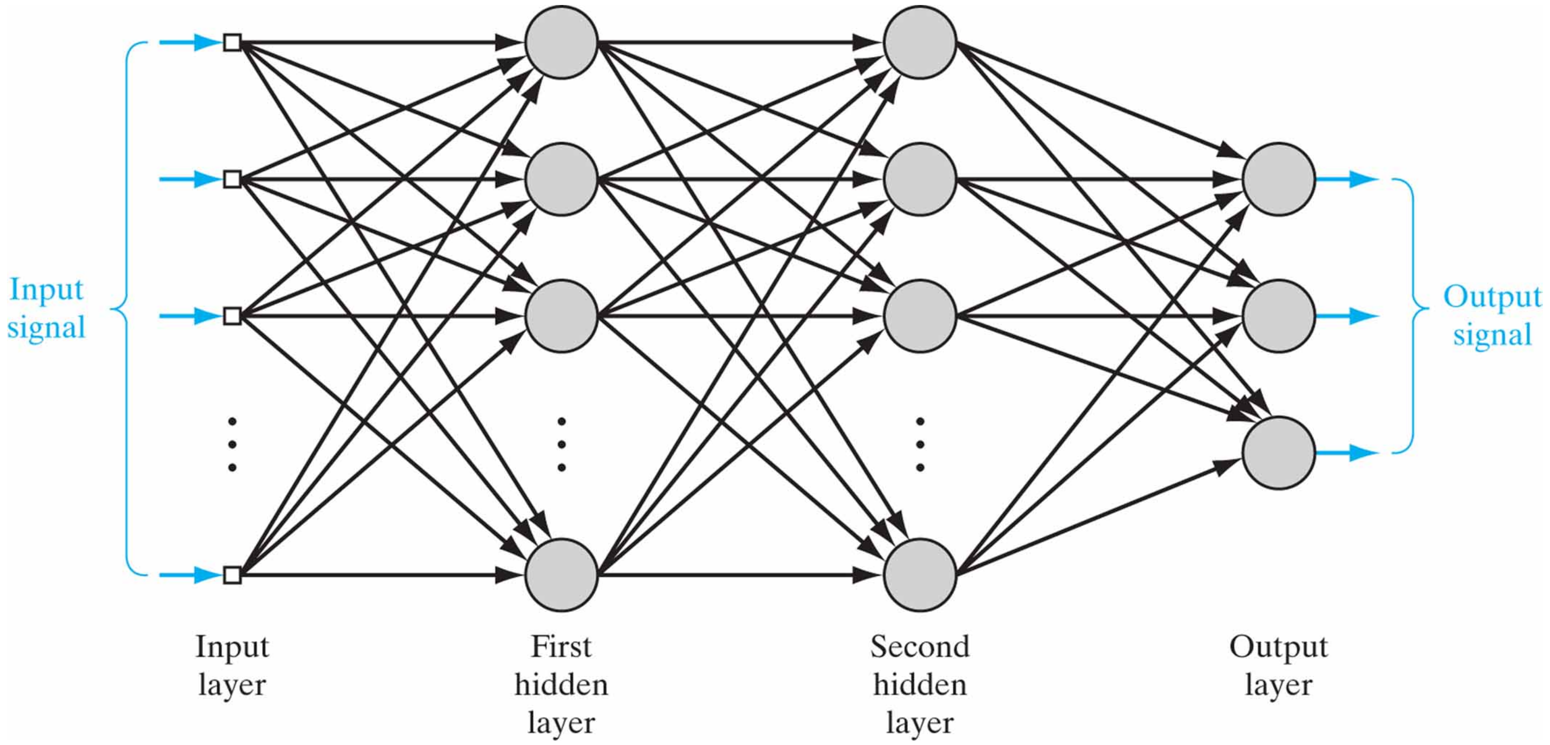




Power of nonlinearity

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

MLP architecture



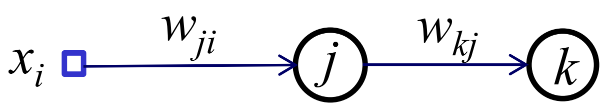
Notations

- Notation for one hidden layer

$$\begin{tikzcd} x_i \ \square \arrow[r, "w_{ji}"] & \textcircled{j} \arrow[r, "w_{kj}"] & \textcircled{k} \end{tikzcd}$$

- The format of presentation to an MLP is the same as to a perceptron

Backpropagation



- Derive the backpropagation (backprop) algorithm by cost minimization via gradient descent

  For the output layer at iteration n, we have

$$E(n) = \frac{1}{2}\sum_k [d_k(n) - y_k(n)]^2$$

$$= \frac{1}{2}\sum_k [d_k(n) - \varphi(\sum_j w_{kj} y_j)]^2$$

Backprop (cont.)

$$\begin{tikzcd} x_i \; \square \arrow[r, "w_{ji}"] & \boxed{j} \arrow[r, "w_{kj}"] & \boxed{k} \end{tikzcd}$$

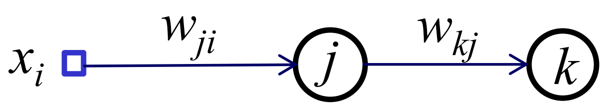
- Then

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial v_k} \frac{\partial v_k}{\partial w_{kj}}$$

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

$$= -e_k \varphi'(v_k) y_j$$

Backprop (cont.)



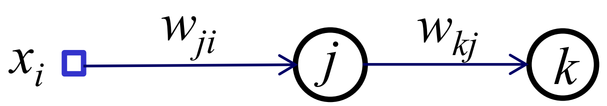
- Hence

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

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

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

Backprop (cont.)



- For the hidden layer at iteration *n*,

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial w_{ji}}$$

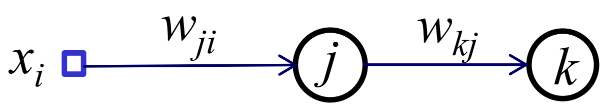
$$= \frac{\partial}{\partial y_j}\{\frac{1}{2}\sum_k [d_k - \varphi(\sum_j w_{kj}y_j)]^2\}\varphi'(v_j)x_i$$

$$= -\sum_k [d_k - \varphi(\sum_j w_{kj}y_j)]\frac{\partial \varphi(\sum_j w_{kj}y_j)}{\partial y_j}\varphi'(v_j)x_i$$

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

$$= -\varphi'(v_j)(\sum_k \delta_k w_{kj})x_i$$

Backprop (cont.)



- Therefore,

$$w_{ji}(n+1) = w_{ji}(n) + \eta\varphi'(v_j(n))[\sum_k \delta_k(n) w_{kj}(n)] x_i(n)$$
$$= w_{ji}(n) + \eta\delta_j(n) x_i(n)$$

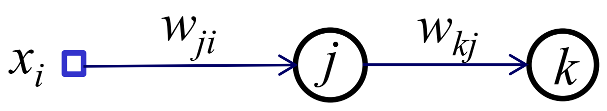
where

$$\delta_j(n) = \varphi'(v_j(n)) \sum_k w_{kj}(n)\delta_k(n)$$

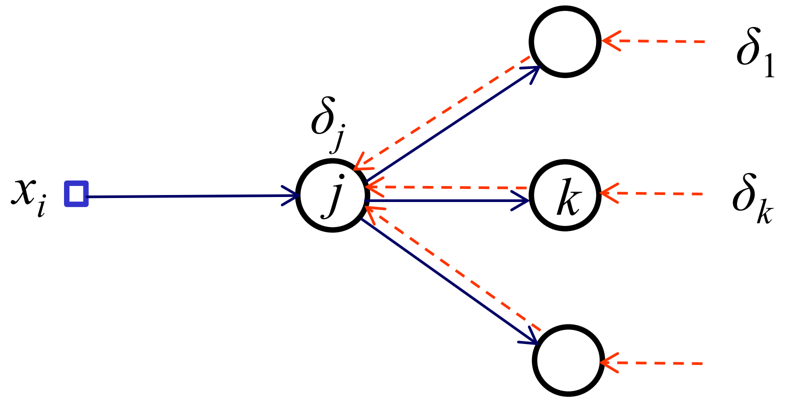
The above is called the generalized δ rule

- textbook correction

Backprop (cont.)



- Illustration of the generalized δ rule,



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

Backprop (cont.)

$$\begin{tikzcd} x_i \arrow[r, "w_{ji}"] & j \arrow[r, "w_{kj}"] & k \end{tikzcd}$$

- For the logistic sigmoid activation, we have

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

hence

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

$$= ay_k(n)[1-y_k(n)][d_k(n)-y_k(n)]$$

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

Backprop (cont.)

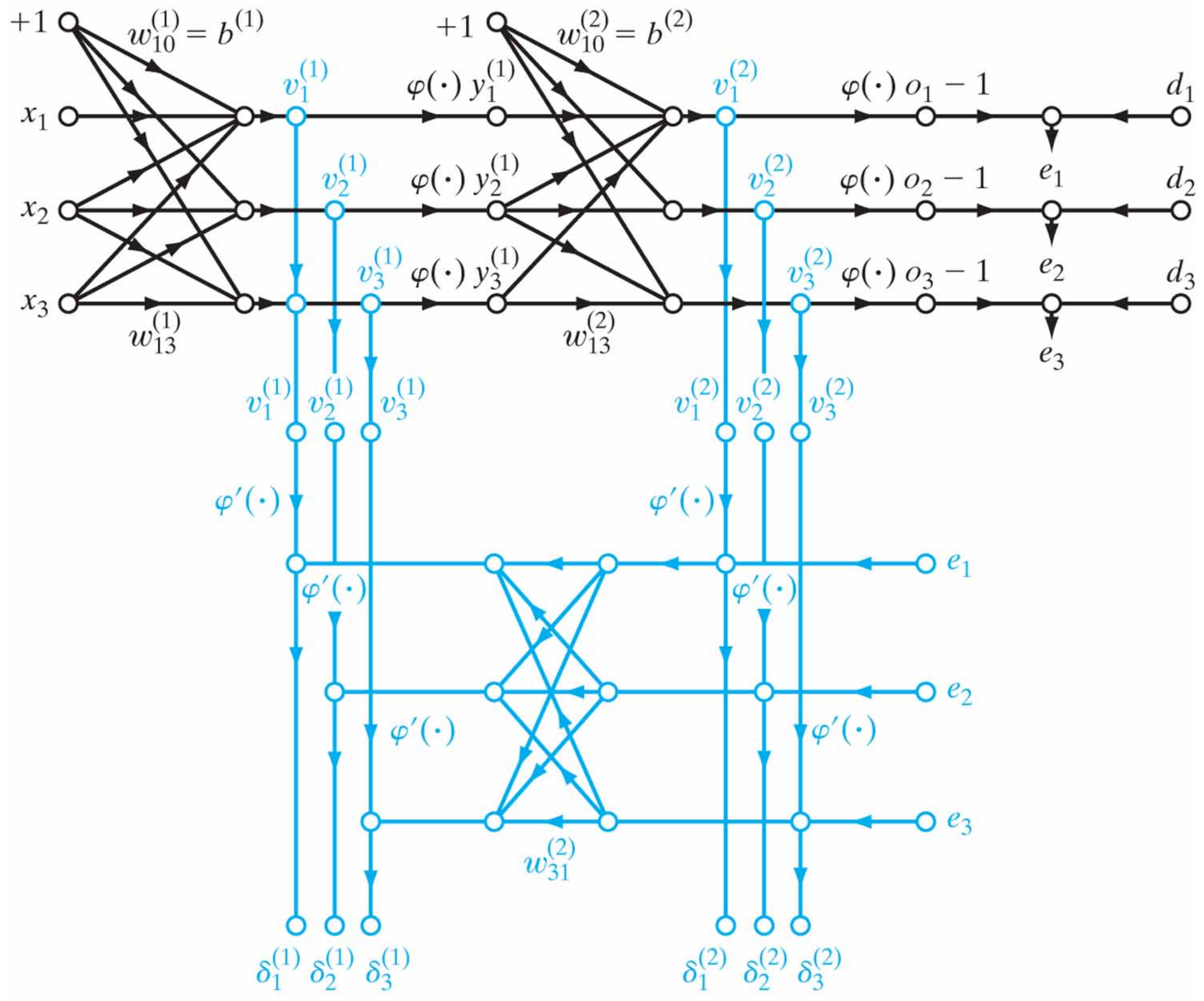
- Extension to more hidden layers is straightforward. In general we have

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

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 backprop (error is back propagated)

Backprop illustration



Learning rate control: momentum

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

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

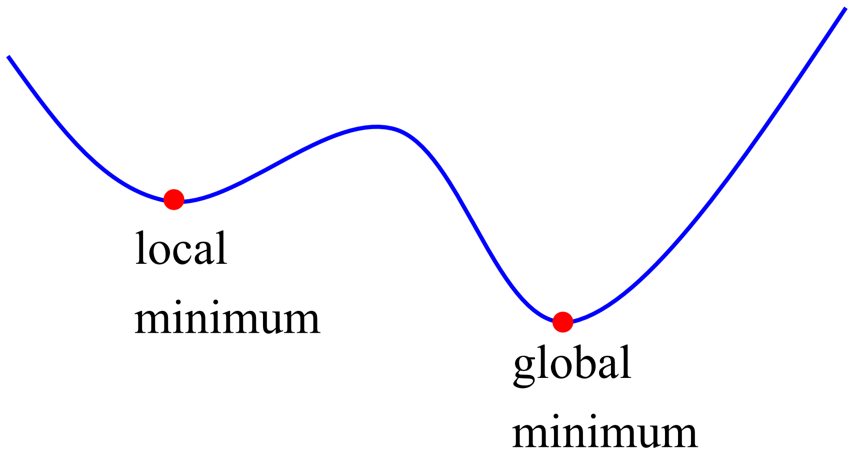
  - In the downhill situation, $\Delta w_{ji}(n) \approx \frac{\eta}{1-\alpha} \delta_j(n) y_i(n)$

    thus accelerating learning by a factor of $1/(1 - \alpha)$

  - In the oscillating situation, it smoothes weight change, thus stabilizing oscillations

Remarks on backprop

- Backprop learning is local, concerning "presynaptic" and "postsynaptic" neurons only
- Local minima are possible due to nonlinearity, but infrequent due to online update and the practice of randomizing the order of presentations in different epochs



Remarks (cont.)

- MLP 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
- Gradient descent is a simple method for cost optimization. More advanced methods exist, such as conjugate gradient and quasi-Newton methods (see textbook)
- Much design is still needed, such as the number of hidden layers, number of neurons for each layer, initial weights, learning rate, stopping criteria, etc.

MLP Design

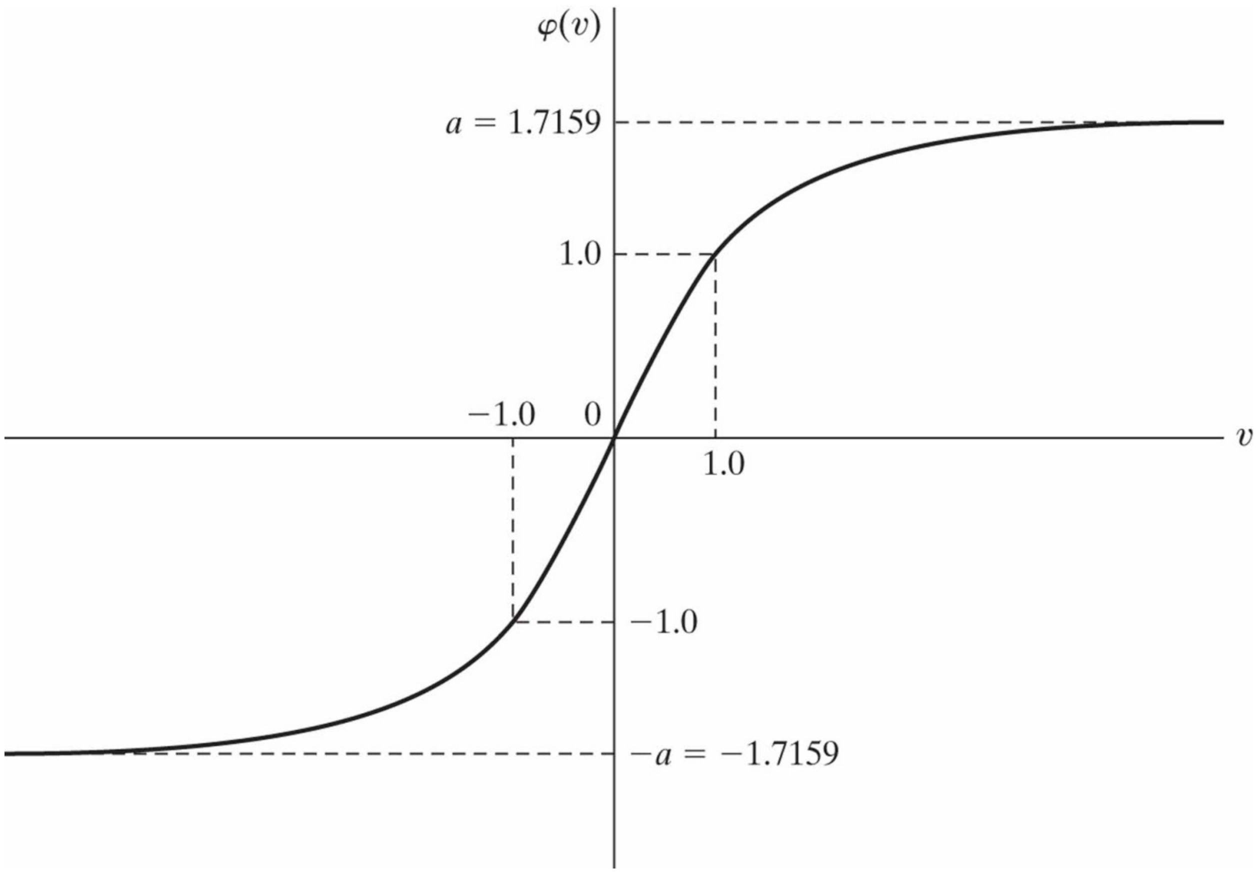
- Besides the use of momentum, learning rate annealing can be applied as for the LMS algorithm

- Batch update versus online update
  - For batch update, weight adjustment occurs after one presentation of all training patterns (one epoch)
  - For online update, weight adjustment occurs after the presentation of each pattern
  - For backprop, online learning is commonly used

MLP Design (cont.)

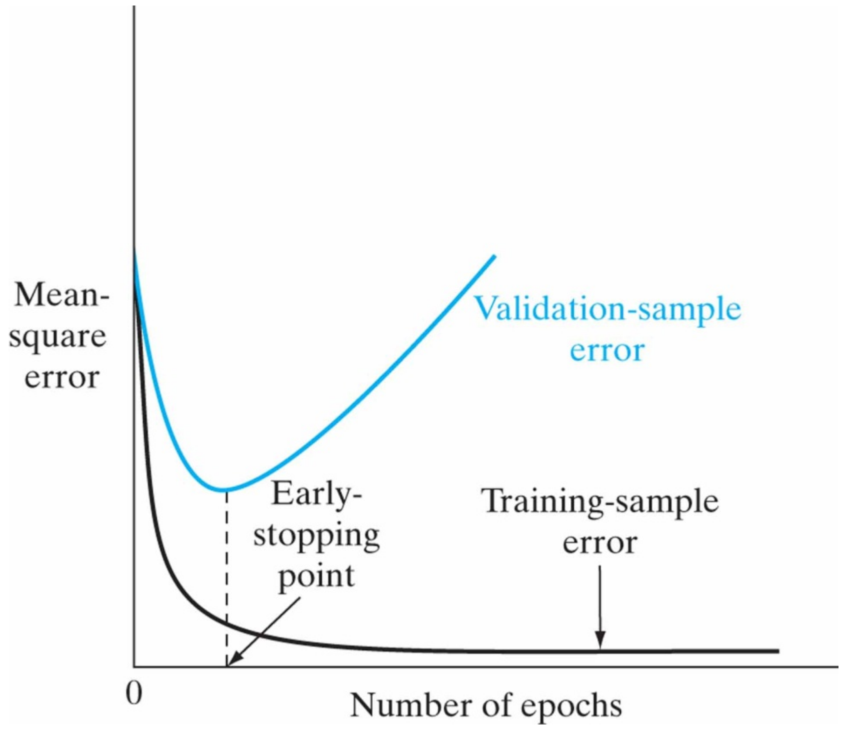
- Weight initialization: To prevent saturating neurons and break symmetry that can stall learning, initial weights (including biases) are typically randomized to produce zero mean and activation potentials away from saturation parts of the activation function
    - For the hyperbolic tangent activation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

Hyperbolic tangent function



When to stop learning

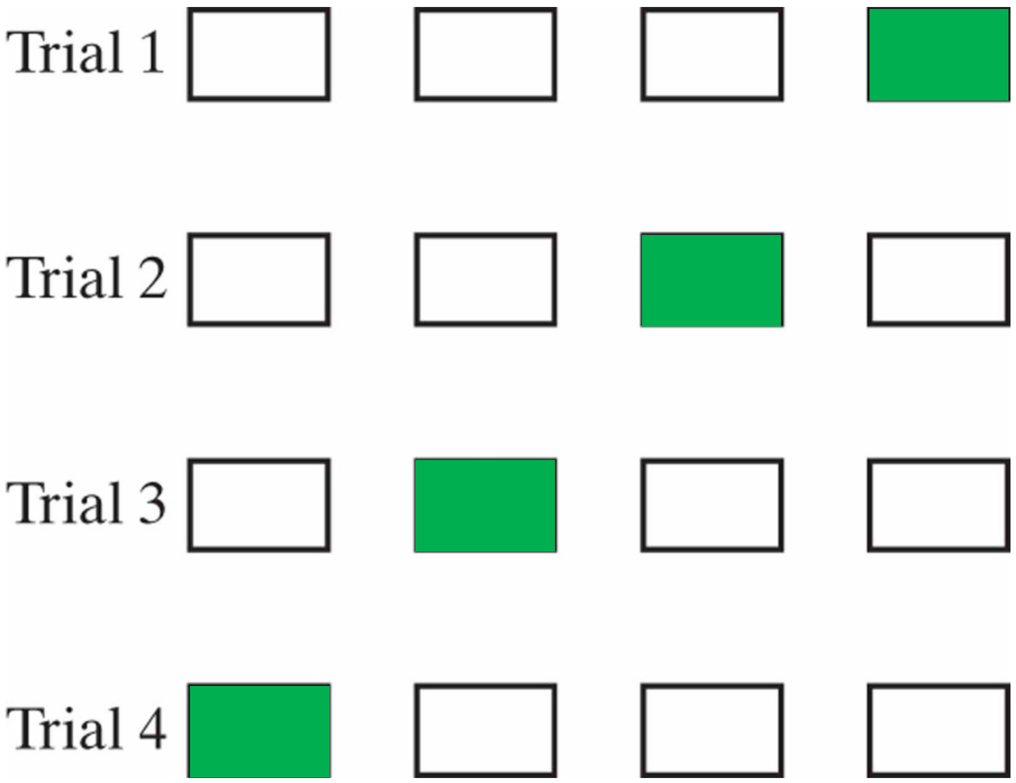
- One could stop after a predetermined number of epochs or when the MSE decrease is below a given criterion
- Early stopping with cross validation: keep part of the training set, called validation subset, as a test for generalization performance



Cross validation

- To do it systematically
  - Divide the entire training sample into an estimation subset and a validation subset (e.g. 80-20 split)
  - Every few epochs (e.g. 5), check validation error on the validation subset
  - Stop training when validation error starts to increase
- To avoid systematic bias, validation error can be measured repeatedly for different validation subsets, taken as the average
  - See next slide

Cross validation illustration



Cross validation (cont.)

- Cross validation is widely used as a general approach for model selection. Hence it can also be used to decide on the number of hidden layers and the number of neurons for a given hidden layer

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 (to be further discussed later)

Zipcode recognizer architecture

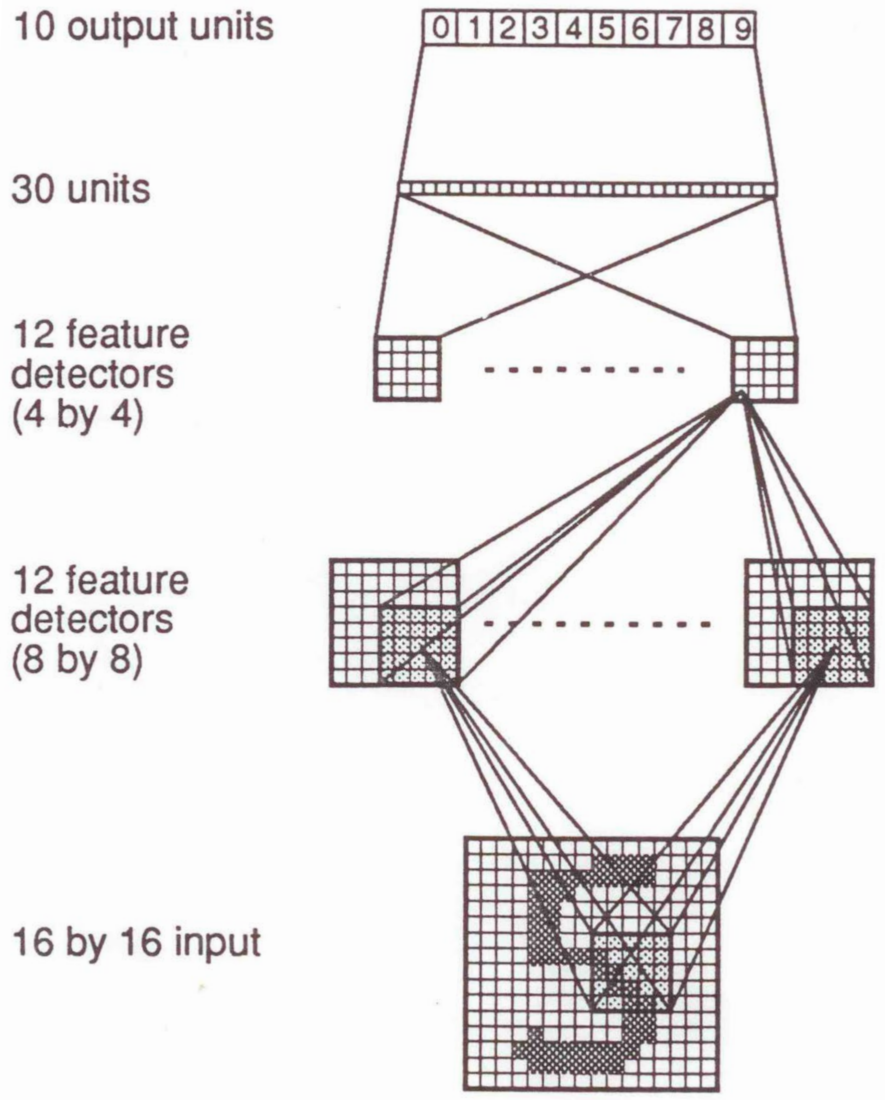


FIGURE 6.10 Archi
of the ZIP-code re
network.

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

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 Chellenge in 2005)

Other MLP applications

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