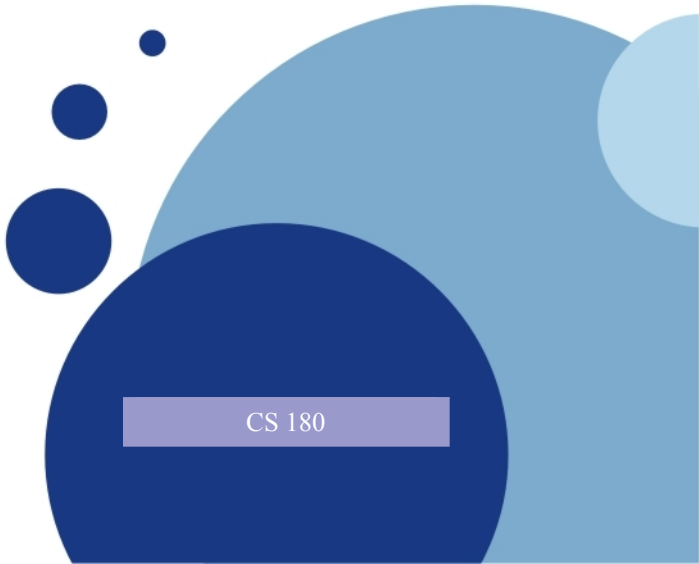
AI

Artificial Neural Networks



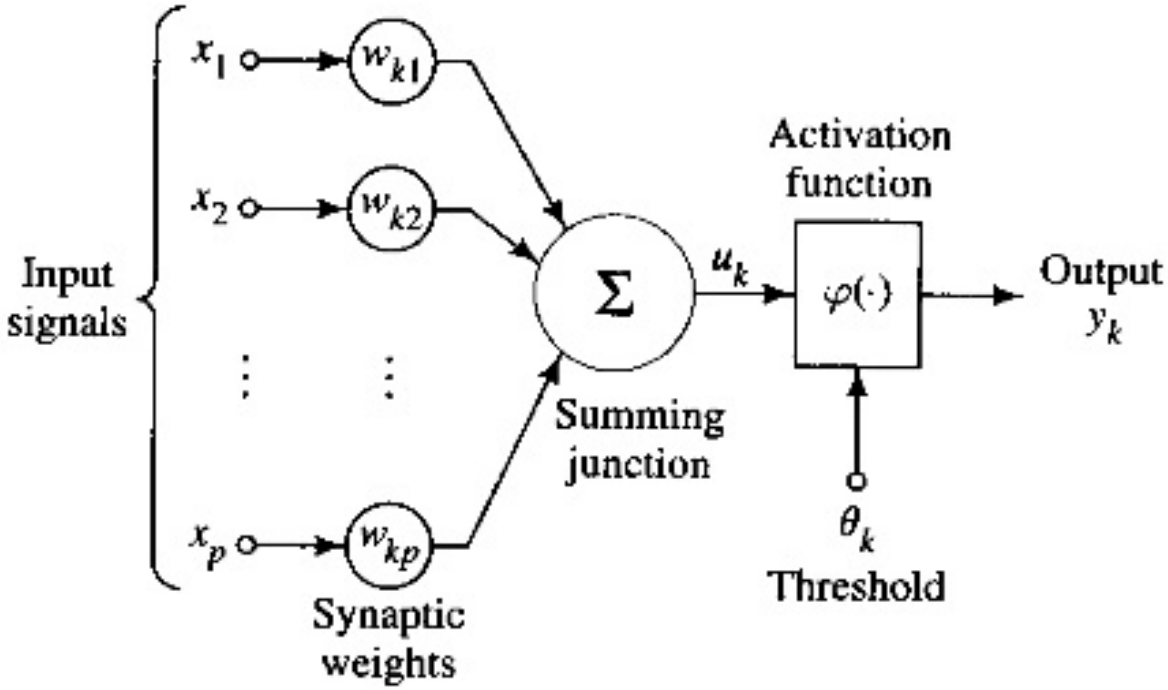
AI

Today: Artificial Neural Networks

- The Perceptron
- Model of a Neuron
- Types of Neural Network Architectures
- The Multilayer Perceptron
- The Backpropagation Algorithm

AI

Model of Neuron



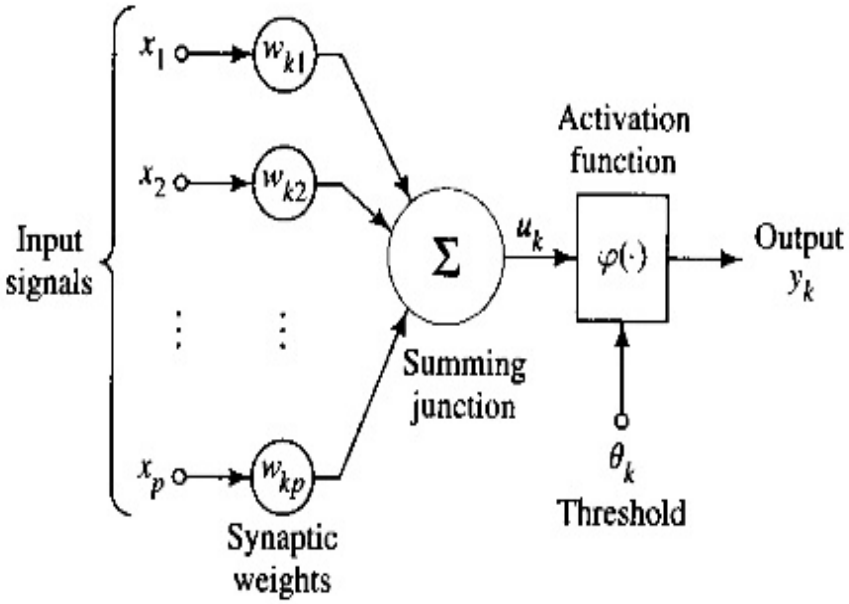
AI

Neuron: Elements

1. Synapses - represented by connecting links, each characterized by a weight

   - excitatory/inhibitory synapse: positive/negative weight values

   - inputs are multiplied by weights to yield weighted inputs



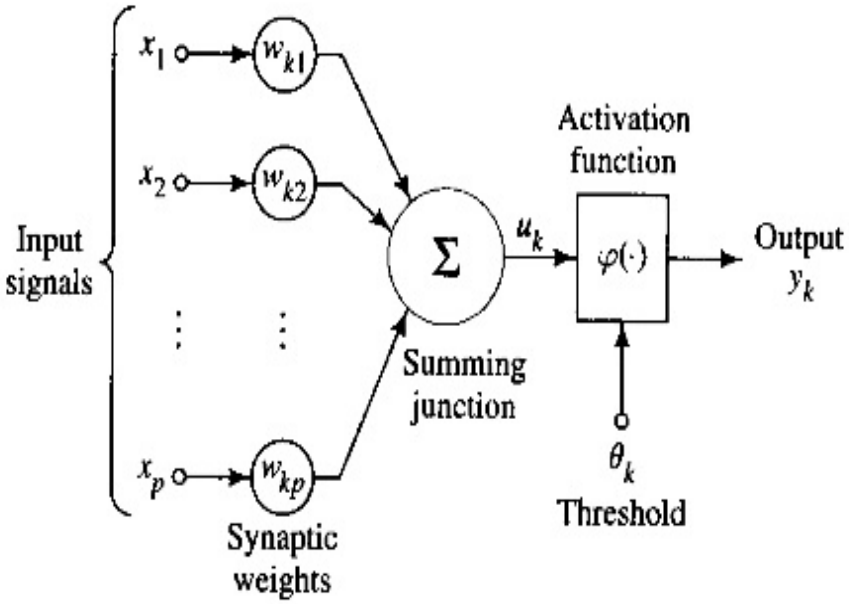
$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

$$y_k = \varphi(u_k - \theta_k)$$

AI

Neuron: Elements

2. Adder - sums up the weighted inputs



$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

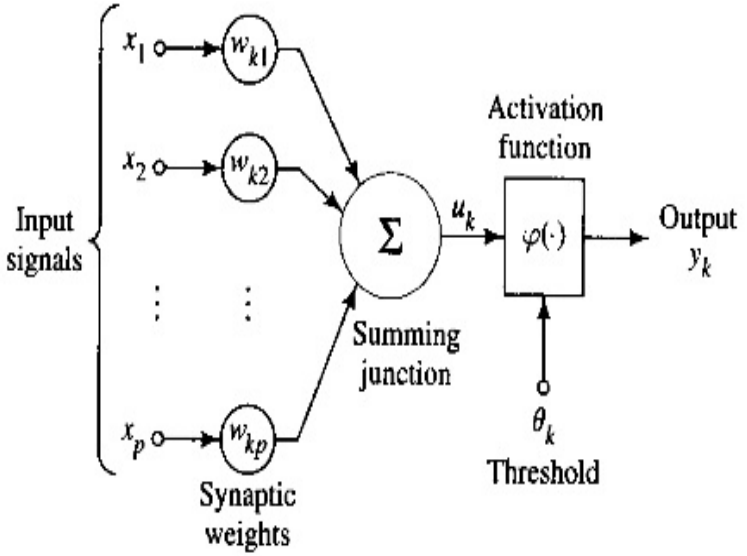
$$y_k = \varphi(u_k - \theta_k)$$

AI

Neuron: Elements

3. Activation function- limits the amplitude of the output to avoid saturation

- also called the “squashing function”
- normalizes the amplitude range of the output to [0, 1] or [-1, 1]
- threshold θ lowers the net input of the activation function

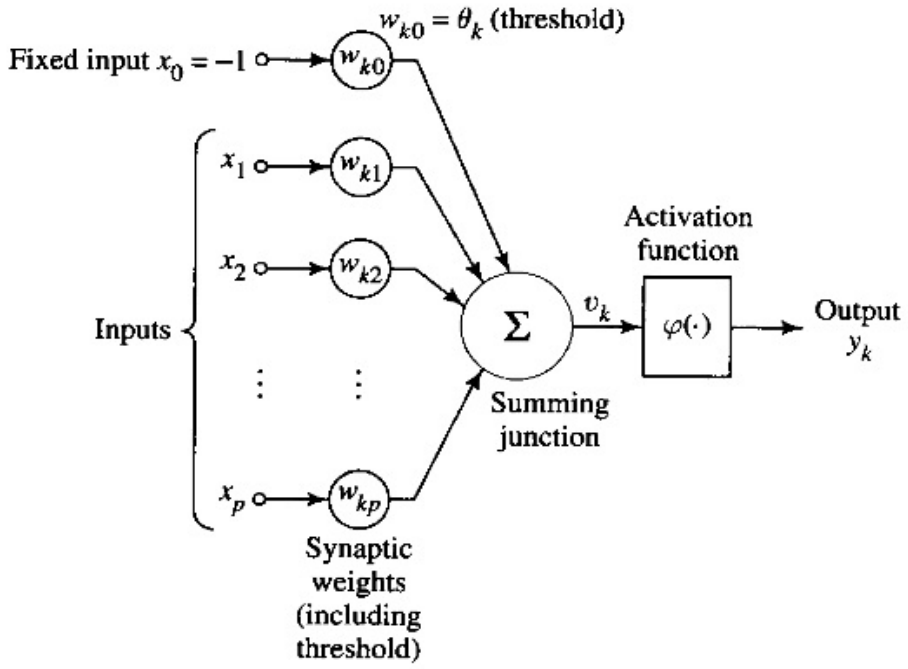


$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

$$y_k = \varphi(u_k - \theta_k)$$

AI

Neuron: Commonly-used model



$$v_k = \sum_{j=0}^{p} w_{kj} x_j$$

$$y_k = \varphi(v_k)$$

AI

Neuron: Elements

- The Activation Function

  The sigmoid function is the most common form of activation function used.

  - logistic or hyperbolic tangent function
  - exhibits nice smoothness (differentiability) and asymptotic properties

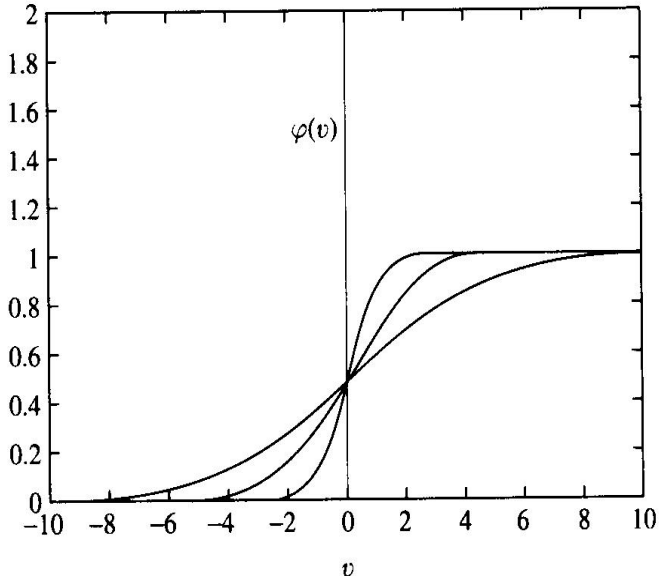
AI

Neuron: Elements

- The Logistic Function

$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

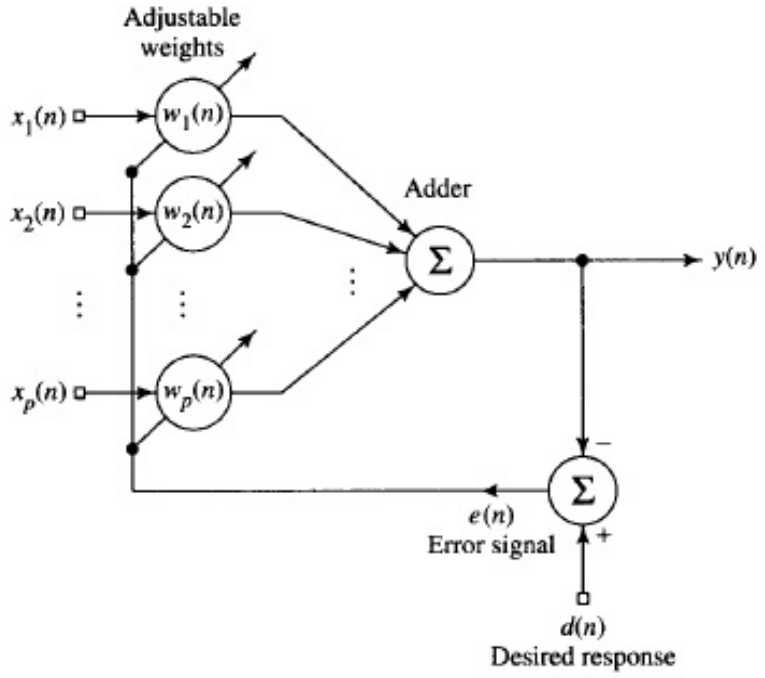
  - where a is the slope parameter.
  - logistic function is continuously differentiable everywhere



AI

Error-Correction Learning

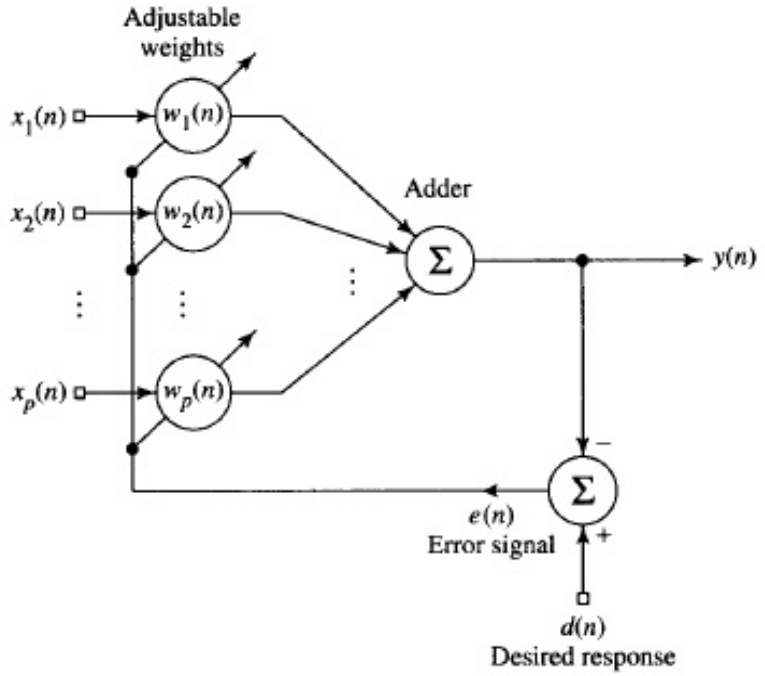
- Let a stimulus input vector *x(n) be applied to the network in which a neuron k is embedded. The neuron will generate an output (called actual response), $y_k(n)$.*



AI

Error-Correction Learning

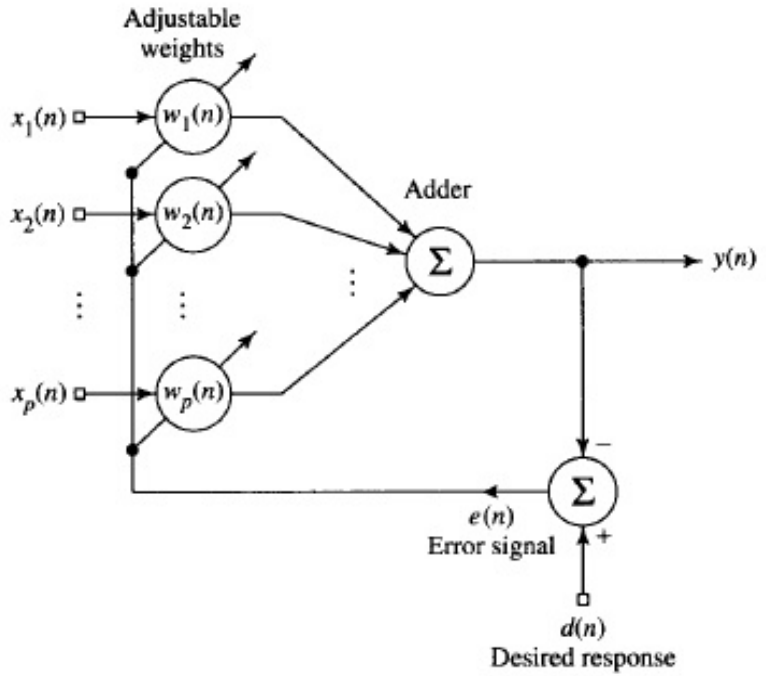
- If $d_k(n)$ *is the desired or target response, the error signal* $e_k(n)$ *is the difference between the target response and the actual response*



AI

Error-Correction Learning

- *The objective is to minimize a cost function based on $e_k(n)$*



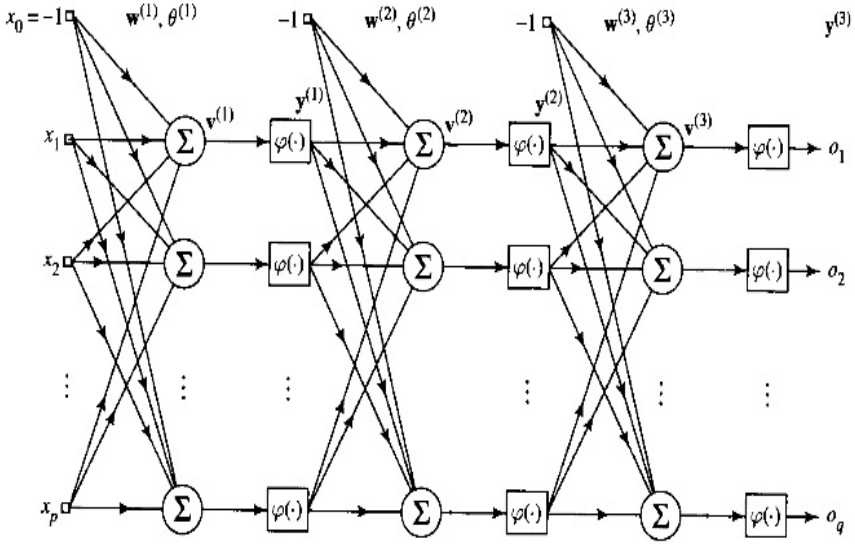
AI

Error-Correction Learning

- *One such cost function is the instantaneous value of the sum of squared errors J(n)*

$$J(n) = \frac{1}{2}\sum_{k} e_k^2(n)$$

- Summation is over all the neurons in the output layer of the network



AI

Error-Correction Learning

- *Learning – finding the set of synaptic weights that will minimize the cost function J(n)*
- The Delta Rule – synaptic weights will converge to their optimal values when the adjustment to the synaptic weight is proportional to the product of the error signal and the input signal

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n)$$

  - where η is the learning rate parameter, a positive constant that determines rate of training

AI

Error-Correction Learning

- if η is too small, convergence is achieved after a long time
- if η is too large, system may not converge, but if it does, learning is fast

AI

The LMS Algorithm

1. Initialization
   Set $\hat{w}_k(1) = 0$ for $k = 1, 2, \ldots, p$.

2. Filtering
   For time $n = 1, 2, \ldots,$ compute

$$y(n) = \sum_{j=1}^{p} \hat{w}_j(n) x_j(n)$$

$$e(n) = d(n) - y(n)$$

$$\hat{w}_k(n+1) = \hat{w}_k(n) + \eta e(n) x_k(n) \qquad k = 1, 2, \ldots, p$$

AI

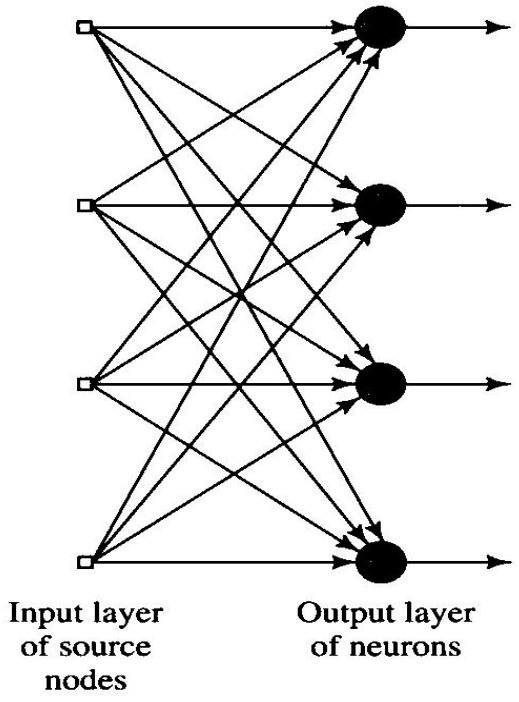
Neural Network Architectures

- Network architecture refers to how the neurons comprising the network are connected to one another
  - single vs multilayer
  - only computational units are counted when counting number of layers
  - feedforward vs feedback

AI

Neural Network Architectures

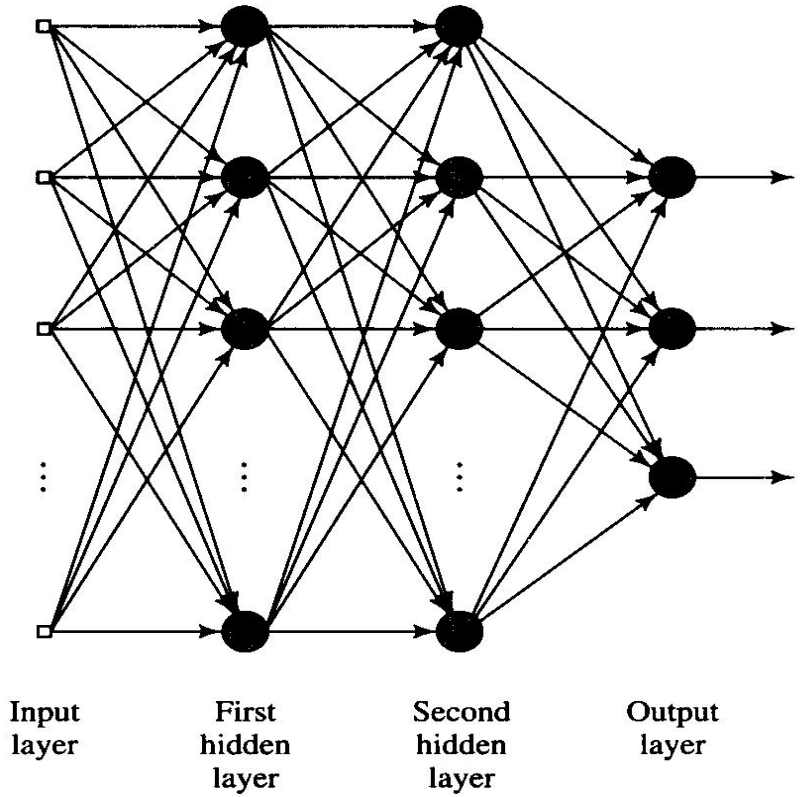
- Single Layer Feedforward Networks
- Linear associative network
  - signals flow forward
  - no feedback



AI

Neural Network Architectures

The Multilayer Perceptron



AI

Neural Network Architectures

- The Multilayer Perceptron
  - has one or more hidden layers of neurons (hidden = not connected to the ``outside world'' through the neuron's output)
  - fully connected network - one in which every node in each layer is connected to every other node in the adjacent forward layer.
  - example: 5-4-2 multilayer feedforward network
- There are also partially connected multilayer feedforward networks

AI

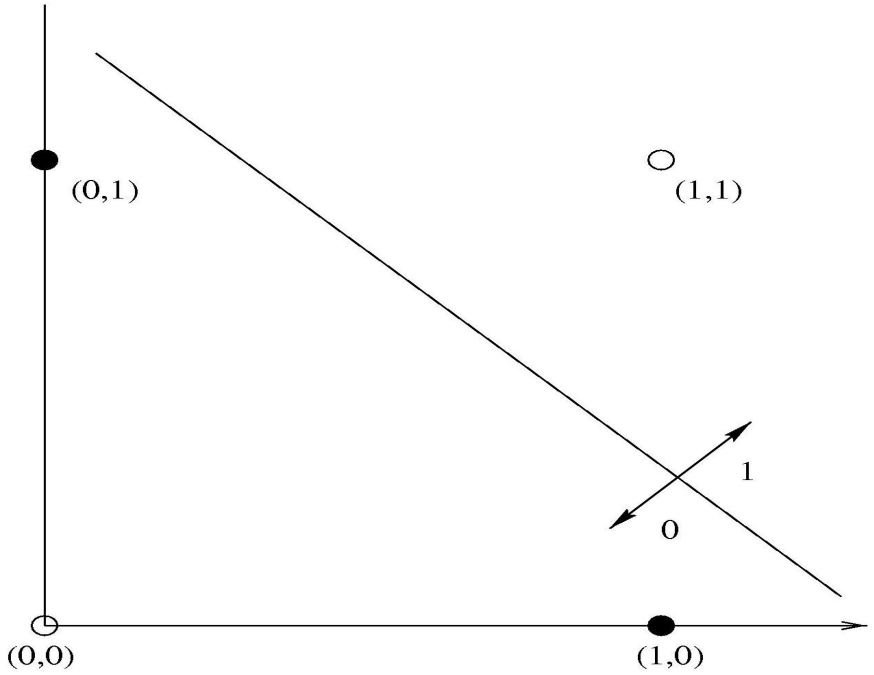
The Single Layer Perceptron

- **The Single Perceptron**
- can perform pattern classification for two sets of linearly separable patterns
- Perceptron Convergence Algorithm can be used to compute for the separating hyperplane in a finite number of steps.
- cannot handle more than two classes
- cannot solve the XOR problem

AI

Neural Network Architectures

- The Perceptron and the XOR Problem (Minsky and Papert)



AI

Multilayer Perceptron

- General Form: fully connected feedforward network
- Can deal with nonlinearly separable patterns (e.g. can solve XOR problem)
- can handle multiple classes

AI

Multilayer Perceptron

- Training algorithm: Backpropagation Algorithm
  - a generalization of the LMS Algorithm which is based on the error-correction learning paradigm
- training algorithm can find nonlinear separating hyperplanes

AI

Multilayer Perceptron

The MLP as a Nonlinear Input-Output Mapping

network's continuous nonlinear activation functions produce continuous nonlinear output functions endowing it the capacity to perform interpolation and therefore generalize from training inputs to unknown ones.

AI

Multilayer Perceptron

The MLP as a Nonlinear Input-Output Mapping

- training involves searching over a large weight space corresponding to the large set of possible functions
- training therefore is equivalent to looking for the function that will fit a given training data set.

AI

Multilayer Perceptron

Limitations of MLP

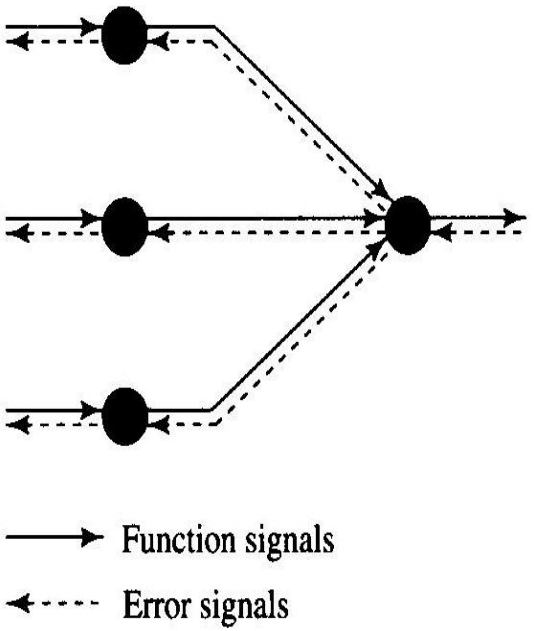
- training is time-consuming
- backpropagation algorithm does not always find the function that gives good generalization ability (weight may land in a local minimum)
- tendency to overfit: high accuracy on training set but low on test set; poor generalization

AI

The Backpropagation Algorithm

Two types of signal flows in a MLP:

- 1. Forward propagation of function signals
  - function signals come in at the network inputs, get modified as they pass through the network, and emerge as output signals.

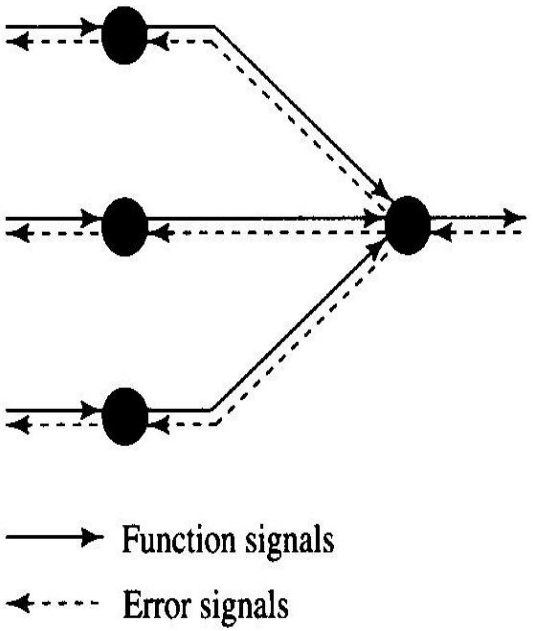


AI

The Backpropagation Algorithm

Two types of signal flows in a MLP:

- 2. backward propagation of error signals
  - error signals come from the output neurons and propagate backward through the network.
  - error signals are used to adjust the weights of the individual neurons



AI

The Backpropagation Algorithm

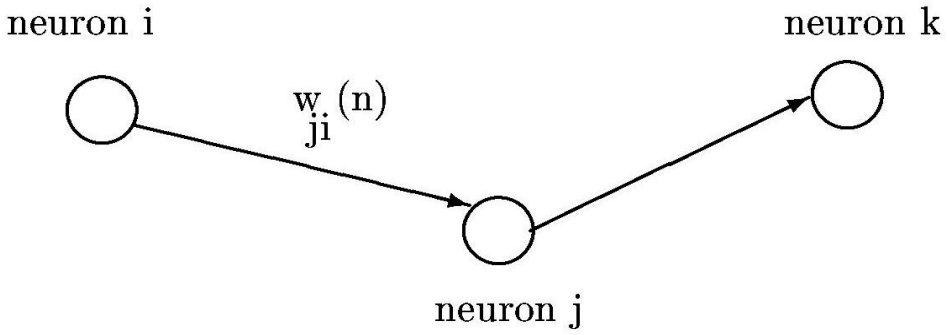
- MLP uses a smooth (differentiable) activation function.
- activation function must be nonlinear, otherwise MLP is just equivalent to a single-layer perceptron.

AI

The Backpropagation Algorithm

Conventions

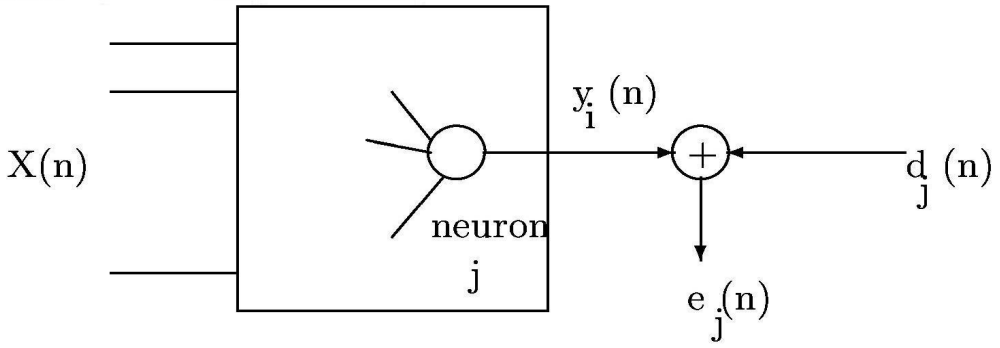
1. Function signals move from left to right and error signals from right to left.
2. Most of the time we will be talking about the jth neuron.
   Neuron i lies to the left of neuron j and is connected to neuron j.
   Neuron k lies to the right of neuron j and is connected to neuron j.



3. Iteration n refers to the n-th training pattern presented to the network.
4. The synaptic weight connecting the output of neuron i to the input of neuron j at iteration n is denoted by $w_{ji}(n)$.
5. The ith element of the input vector is denoted by $x_i(n)$.
6. The kth element of the overall (network) output vector is denoted by $o_k(n)$

AI

The Backpropagation Algorithm



$$e_j(n) = d_j(n) - y_i(n)$$

- We assume that the network is in the training phase.
- Start by presenting the n-th training input pattern to the network.
  - $e_j(n)$ is the error signal for neuron j
  - $d_j(n)$ is the desired response for neuron j
  - $y_i(n)$ is the output signal for neuron j

AI

The Backpropagation Algorithm

- Conventions

Let us define the instantaneous value of the square error for neuron j as $\frac{1}{2}e_j^2(n)$.

The instantaneous sum of square errors $\xi(n)$ of the network is

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

where set C is the set of all output neurons of the network.

If there are N total input patterns in the training set, the average squared error for the entire training set is

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(n)$$

The quantity ξ_{av} is the cost function which we want to minimize. The objective of the learning process is to adjust the weights so as to minimize ξ_{av}.

AI

The Backpropagation Algorithm

- Epoch - one complete presentation of the training set during training
- Two Ways of Adjusting Weights:
  - on a pattern-by-pattern basis - weights are updated after each pattern
  - on a per epoch basis - weights are updated at the end of the epoch

AI

The Backpropagation Algorithm

The Delta Rule

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

where

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n)) \qquad \text{if neuron } j \text{ is an output neuron}$$

and

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \qquad \text{if neuron } j \text{ is a hidden neuron}$$

AI

The Backpropagation Algorithm

The Deltas

Logistic Function Derivative:

$$\varphi_j(v_j(n)) = \frac{1}{1 + exp(-v_j(n))}$$

$$\varphi'_j(v_j(n)) = y_j(n)[1 - y_j(n)]$$

Delta of the Output Neuron:

$$\begin{aligned} \delta_j(n) &= e_j(n)\varphi'_j(v_j(n)) \\ &= [d_j(n) - o_j(n)]o_j(n)[1 - o_j(n)] \end{aligned}$$

Delta of the Hidden Neuron:

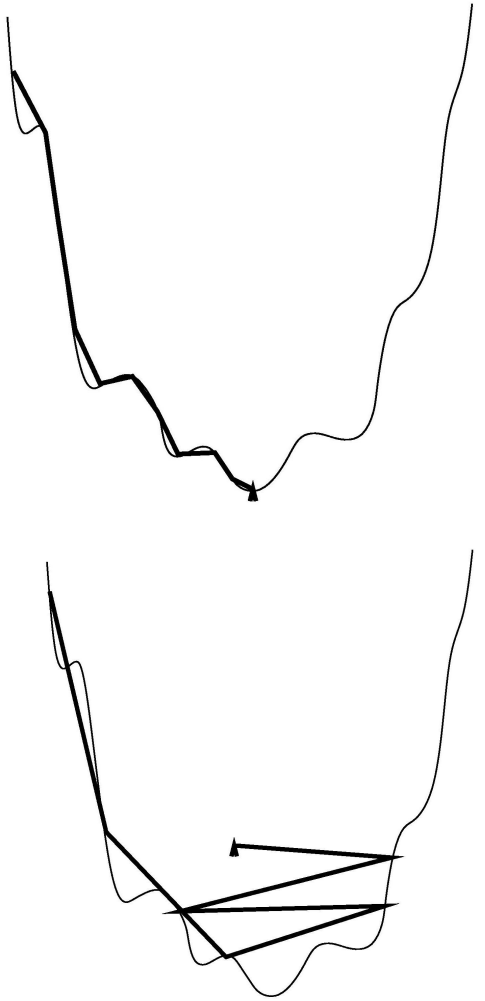
$$\begin{aligned} \delta_j(n) &= \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \\ &= y_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n) \end{aligned}$$

AI

The Backpropagation Algorithm

- Small values of the learning rate parameter η make the trajectory in weight space smooth.
  - learning is slow (long training time)
- Too large a value of η might make the network unstable because of oscillations.

Speed-up in learning can be achieved without sacrificing stability by including a momentum term.



AI

The Backpropagation Algorithm

- The generalized delta rule

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_j(n)$$

- where α is a positive value called the momentum constant.

- large changes in w_{ji} cause instability because it is based on the local gradient.

Generalized Delta Rule takes into account also the value and direction of the previous weight update (and therefore the previous local gradient value).

AI

Pattern and Batch Training Modes

- During training, the order of presentation of training examples must be randomized from one epoch to the next.

Two Ways of Training MLP:

1. Pattern Mode - update weights after presentation of each training example.
   - requires less storage
2. Batch Mode - update weights after presentation of all training examples (i.e. at the end of each epoch)
   - parallelizable

AI

Backpropagation Summary

The backpropagation algorithm operating in the pattern mode cycles through the training data $\{[\vec{x}(n), \vec{d}(n)]; n = 1, 2, \ldots, N\}$ in the following manner:

I. Initialization

1) Configure the network (determine the number of hidden layers and the number of nodes in each hidden and output layers).
2) Initialize the synaptic weights and thresholds to small uniformly distributed random numbers.

II. Presentation of Training Examples

Present the network with an epoch of training examples.
For each training example, $[\vec{x}(n), \vec{d}(n)]$, perform steps 3 to 10:
3) Apply the input vector $\vec{x}(n)$ to the input layer.

AI

Backpropagation Summary

III. Forward Computation

4) Calculate the activation potentials of the neurons in the layer l:

$$v_j^{(l)}(n) = \sum_{i=0}^{p} w_{ji}^{(l)}(n) y_i^{(l-1)}(n)$$

- for $i = 0$, we have $y_0^{(l-1)}(n) = -1$ and $w_{j0}^{(l)}(n)$ is treated just like the other weights.
- for layer 1, $y_j^{(0)}(n) = x_j(n)$, where $x_j(n)$ is the jth element of the input vector $\vec{x}(n)$

5) Calculate the outputs for the layer l:

$$y_j^{(l)}(n) = \varphi(v_j^{(l)}(n))$$

If we are using the logistic activation function, we have:

$$y_j^{(l)}(n) = \frac{1}{1 + e^{-v_j^{(l)}(n)}}$$

6) Repeat 4 and 5 for all layers including the output layer.
The output of the network is just the outputs of the neurons in the output layer ($l = L$):

$$o_j(n) = y_j^{(L)}(n)$$

AI

Backpropagation Summary

IV. Error Computation and Weight Update

7) Compute the Error Signals:

$$e_j(n) = d_j(n) - o_j(n)$$

where $d_j(n)$ is the jth element of the desired response vector $\vec{d}(n)$.
8) Compute the sum of squared errors of the network:

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

9) Calculate the local gradient δ_j, of the network by proceeding backward layer by layer:

$$\delta_j^{(L)} = e_j^{(L)}(n) o_j(n)[1 - o_j(n)] \qquad \text{for neurons in the output layer } L$$

or

$$\delta_j^{(l)} = y_j^{(l)}(n)[1 - y_j^{(l)}(n)] \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) \qquad \text{for neurons in the hidden layer } l$$

10) Adjust the synaptic weights of the network according to the Generalized Delta Rule:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[w_{ji}^{(l)}(n) - w_{ji}^{(l)}(n-1)] + \eta \delta_j^{(l)}(n) y_i^{(l-1)}(n)$$

- Note: compute first the δ's of layer $(l-1)$ before updating the weights in layer l. The order of weight updates is not important.

AI

Backpropagation Summary

V. Iteration

11) Repeat steps 3 to 10 for all training examples in the training set. Compute the average squared error for the epoch:

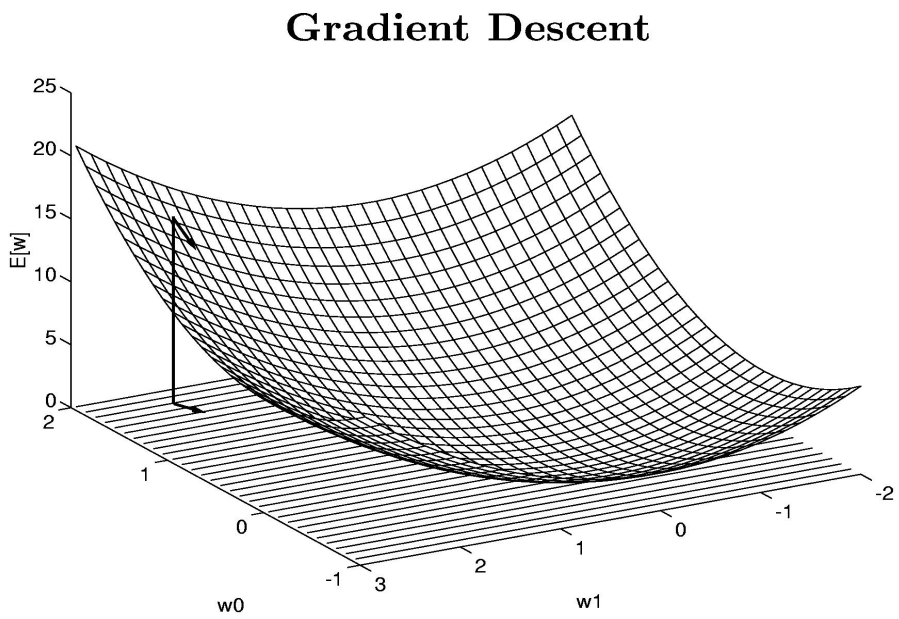
$$\xi_{av} = \frac{1}{N} \sum_{n=1}^{N} \xi(n)$$

where N is the number of training examples.
You must randomize the order of presentation of training examples from epoch to epoch. Stop training when the selected stopping criterion is satisfied.

AI

More on Backpropagation

- Gradient descent over the entire network weight vector



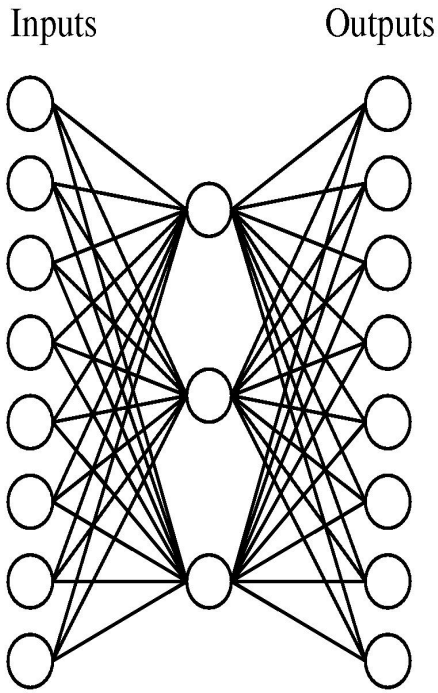
AI

More on Backpropagation

- Will find a local, not necessarily global error minimum
- Training can take thousands of iterations (slow)
- Using network after training is very fast

AI

Learning Hidden Layer Representations



- Task: Learn the simple target function f(x) = x where x is a vector containing seven 0's and a single 1
- The network must learn to reproduce the eight inputs at the corresponding eight output units

AI

Learning Hidden Layer Representations

- Can this be learned?
- A target function:

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

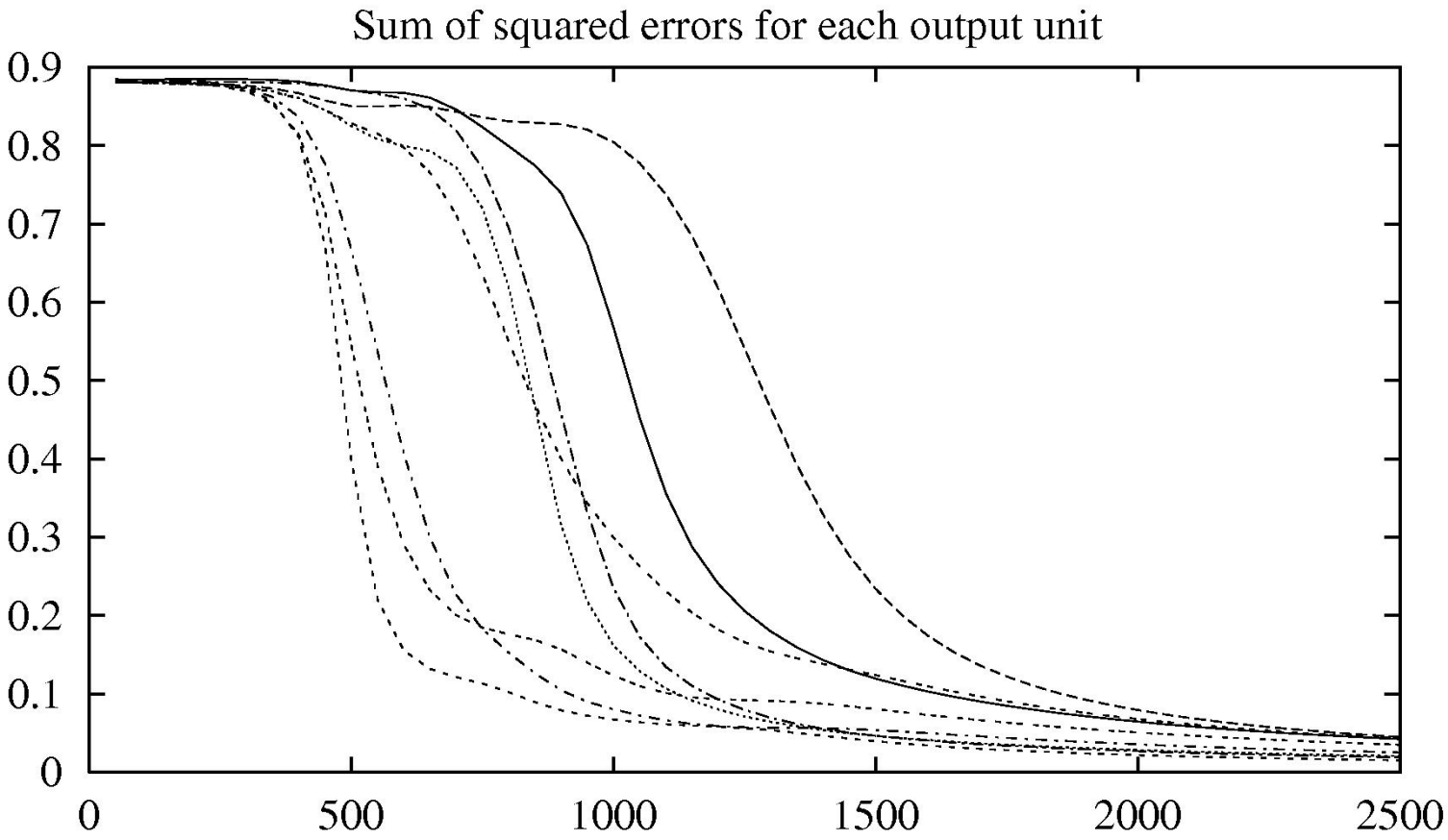
AI

Learning Hidden Layer Representations

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

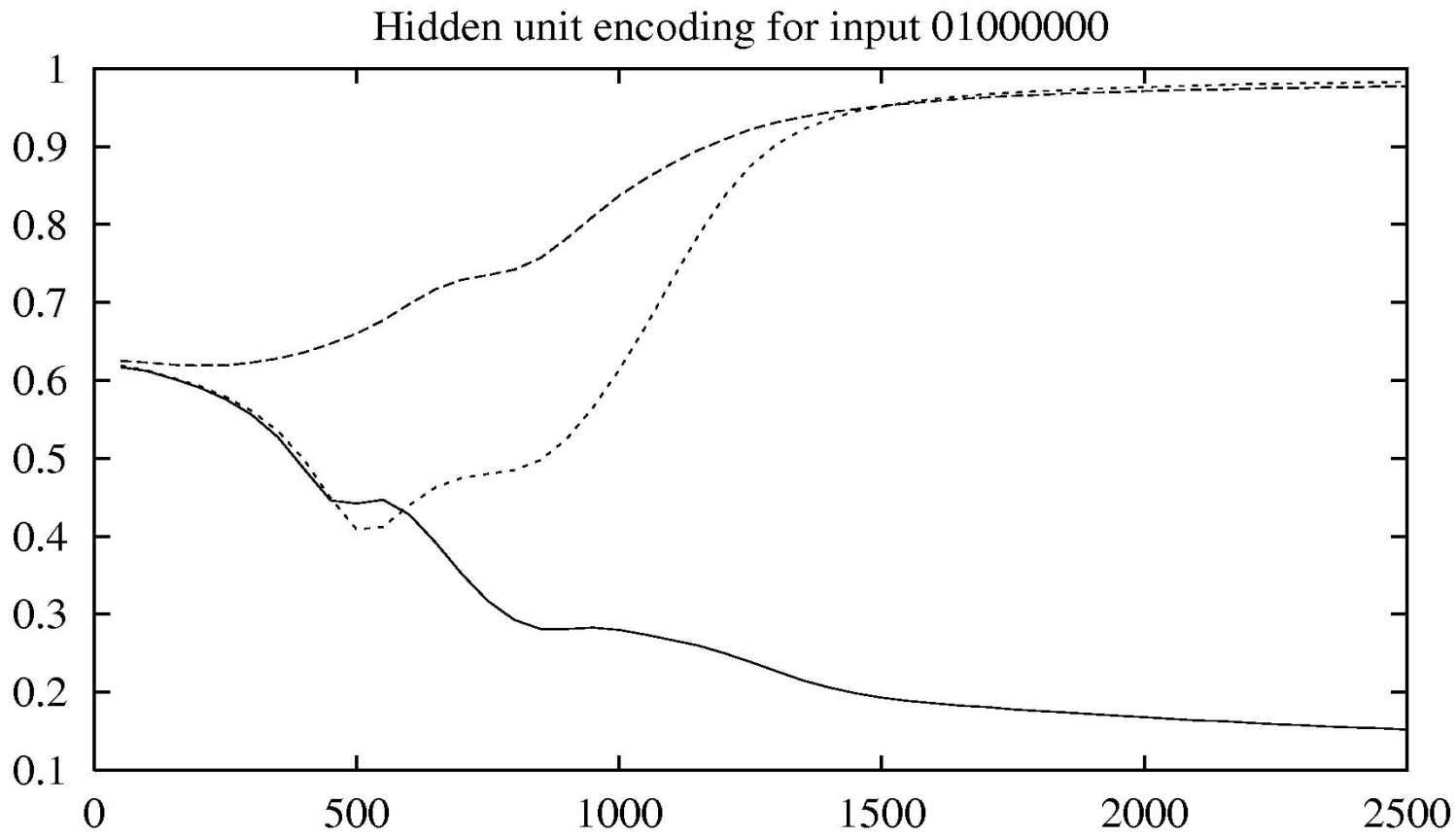
AI

Training


Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500

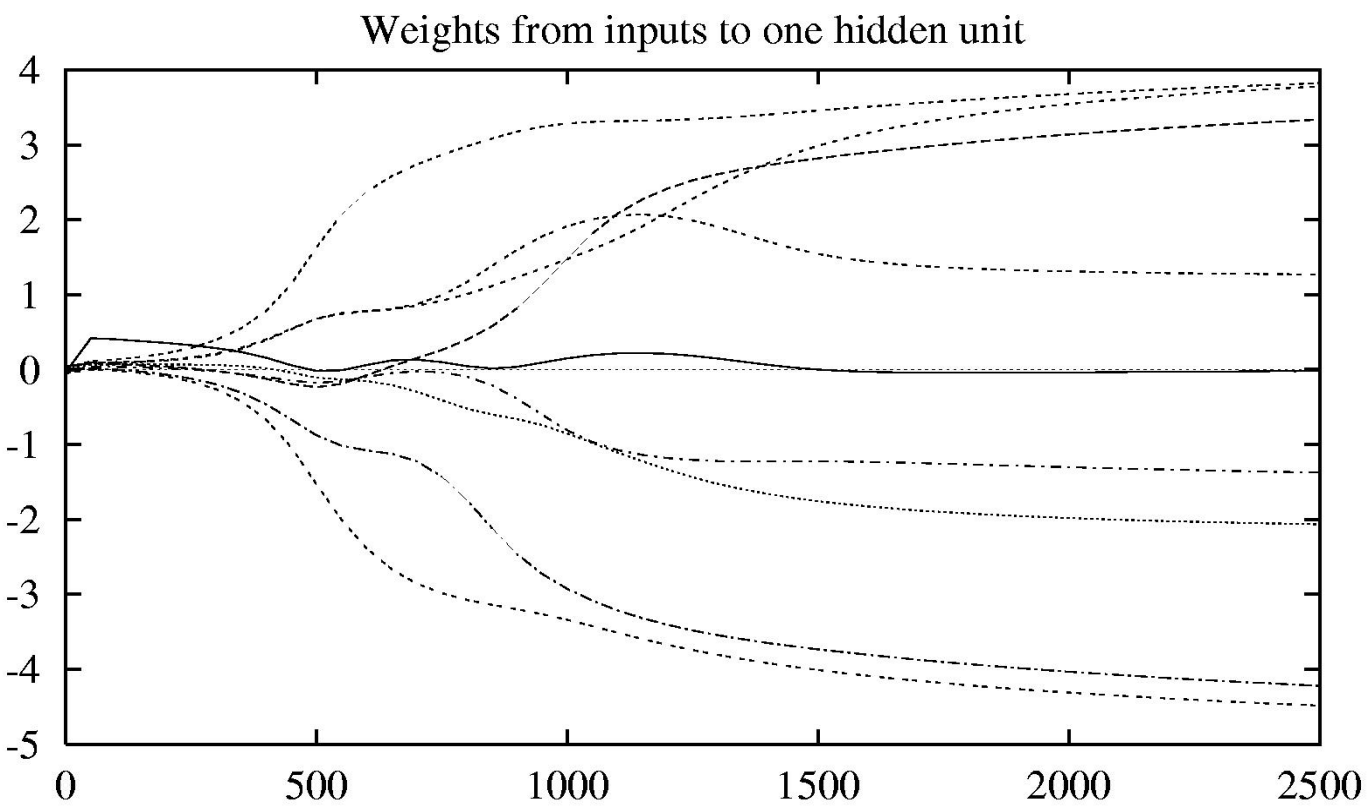
AI

Training


Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500

AI

Training


Weights from inputs to one hidden unit
4
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500

AI

Convergence of Backpropagation

- Gradient descent to some local minimum
  - Perhaps not global minimum
  - Add momentum
  - Train multiple networks with different initial weights
- Nature of convergence
  - Initialize weights near zero
  - Therefore initial networks near linear
  - Increasingly nonlinear functions possible as training progresses

AI

Expressiveness of Neural Networks

- Boolean functions:
  - Every Boolean function can be represented by network with single hidden layer
  - But might require exponential (in number of inputs) hidden units

AI

Expressiveness of Neural Networks

- Continuous functions:
  - Every bounded continuous function can be approximated with arbitrarily small errors, by network with one hidden layer
  - Any function can be approximated to arbitrary accuracy by a network of two hidden layers

AI

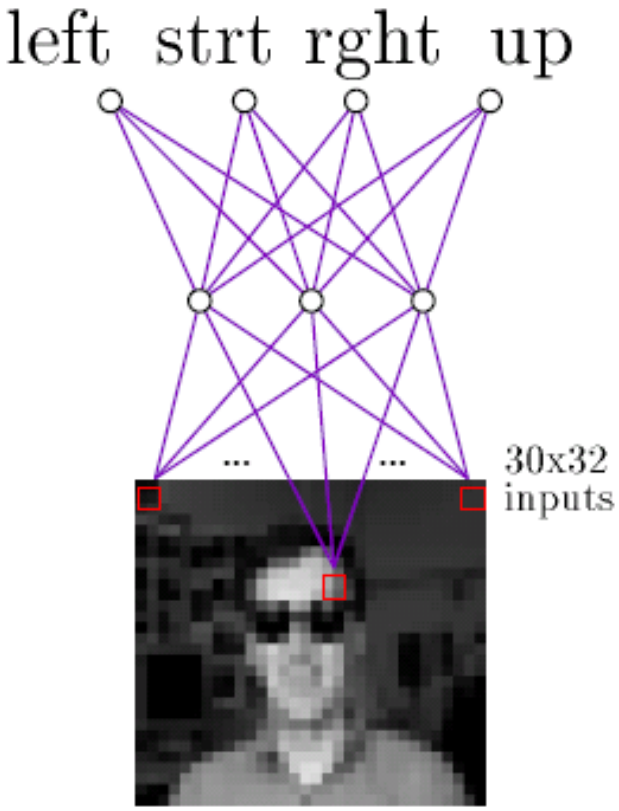
ApplicationFace Recognition Data

Images of 20 different people were collected, including approximately 32 images per person, varying the person's expression (happy, sad, angry, neutral), the direction in which they were looking(left, right, straight ahead, up) and whether or not they were wearing sunglasses

AI

Learning an artificial neural network to recognize face pose

A 960 x 3 x 4 network is trained on grey-level images of faces to predict whether a person is looking to their left, right, ahead or up



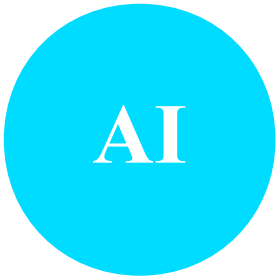
AI

Design Choices: Input Encoding

Encode the image as a fixed set of 30x32 pixel intensity values, with one network input per pixel

The pixel intensity values ranging 0 to 255 were linearly scaled to range from 0 to 1 so that network inputs would have values in the same interval as the hidden and output units

- The 30x32 pixel image is a coarse resolution summary of the original 120x128 captured image



Design Choices: Output Encoding

The ANN must output one of four values indicating the direction in which the person is looking (left, right, up or straight)

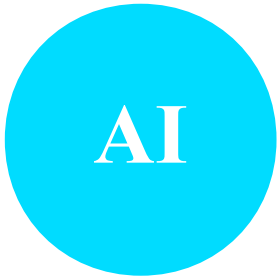
We could encode this four-way classification using a single output unit, assigning outputs, say, 0.2, 0.4, 0.6 and 0.8 to encode these four possible values

Instead, four distinct output units were used, each representing one of the four possible face directions, with the highest valued output taken as the network prediction

AI

Design Choices: Output Encoding

- This is often called a 1-of-n output encoding
- There are two motivations for choosing this over the single unit option
- First, it provides more degrees of freedom to the network for representing the target function (I.e. there are n times as many weights available in the output layer of units)
- Second, the difference between the highest valued output and second highest can be used as a measure of the confidence in the network prediction (ambiguous classifications may result in near or exact ties



Design Choices: Target Values

One obvious choice wold be to use the four target values <1,0,0,0> to encode a face looking to the left <0,1,0,0> to encode a face looking straight etc.

Instead of 0 and 1 values, we use values of 0.1 and 0.9, so that <0.9,0.1, 0.1,0.1> is the target vector for a face looking to the left

AI

Design Choices: Target Values

The reason for avoiding target values of 0 and 1 is that sigmoid units cannot produce these output values given finite weights

If we attempt to train the network to fit target values of exactly 0 and 1, gradient descent will force the weights to grow without bound

On the other hand, values of 0.1 and 0.9 are achievable using a sigmoid unit with finite weights

AI

Design Choices: Network Structure

- Used a layered feedforward network with one hidden layer
- Only 3 hidden units were used yielding a test set accuracy of 90%
- In other experiments, 30 hidden units were used yielding a test set accuracy one to two percent higher
- Although the generalization accuracy varied only a small amount between these two experiments, the second experiment required a significantly more training time (1 hour)

AI

Learned Weights