Machine Learning

Artificial Neural Networks

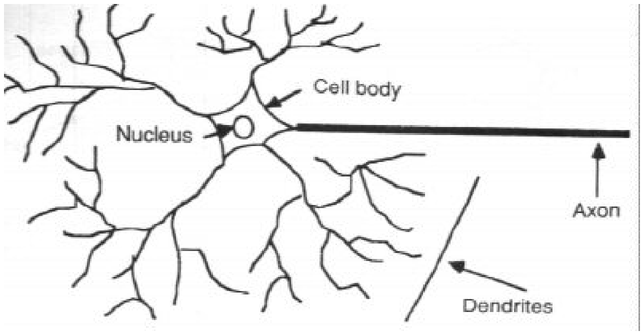
Artificial Intelligence & Computer Vision Lab
School of Computer Science and Engineering
Seoul National University

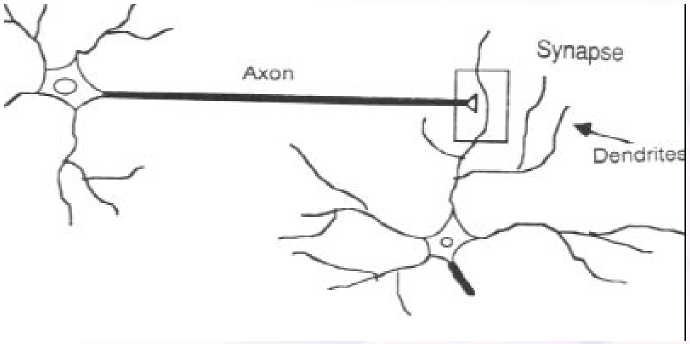
Overview

- Introduction
- Perceptrons
- Multilayer networks and Backpropagation Algorithm
- Remarks on the Backpropagation Algorithm
- An Illustrative Example: Face Recognition
- Advanced Topics in Artificial Neural Networks

Introduction

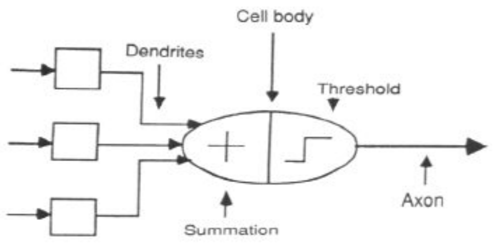
- Human brain
  - densely interconnected network of 10^{11} neurons each connected to 10^{4} others (neuron switching time : approx. 10^{-3} sec.)
- ANN (**A**rtificial **N**eural **N**etwork**)**
  - this kind of highly parallel processes

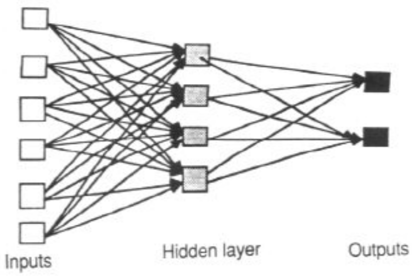




Introduction (cont.)

- Properties of ANNs
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed process

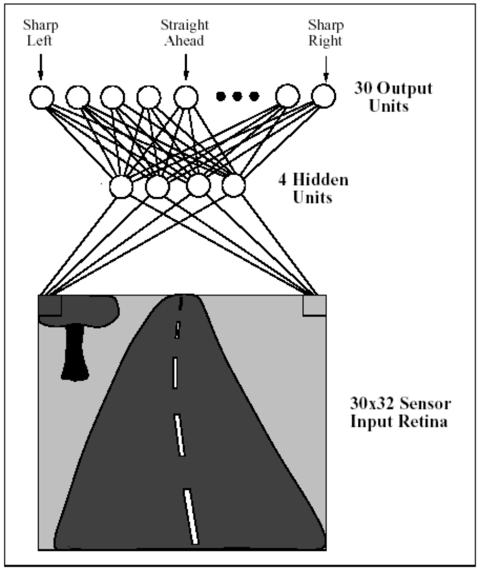


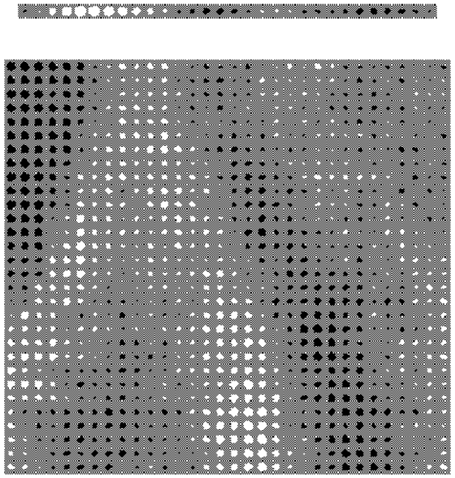


Introduction (cont.)

- Neural network representations

  -ALVINN drives 70 mph on highways





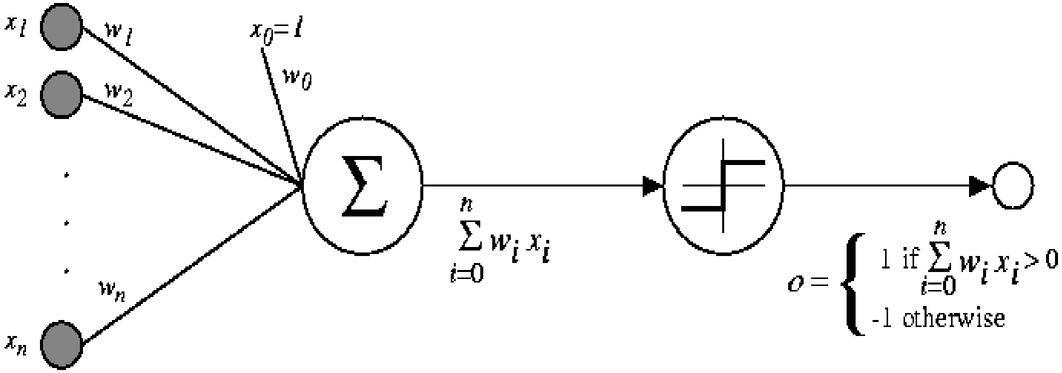
Introduction (cont.)

- Appropriate problems for neural network learning
  - Input is high-dimensional discrete or real-valued
    (e.g. raw sensor input)
  - Output is discrete or real valued
  - Output is a vector of values
  - Possibly noisy data
  - Long training times accepted
  - Fast evaluation of the learned function required.
  - Not important for humans to understand the weights

- Examples
  - Speech phoneme recognition
  - Image classification
  - Financial prediction

Perceptrons

- Perceptron



  - Input values → Linear weighted sum → Threshold

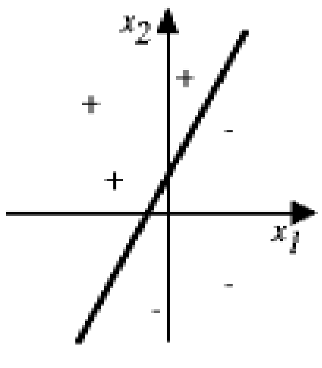
  - Given real-valued inputs x_1 through x_n, the output $o(x_1,\ldots,x_n)$ computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + \ldots + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

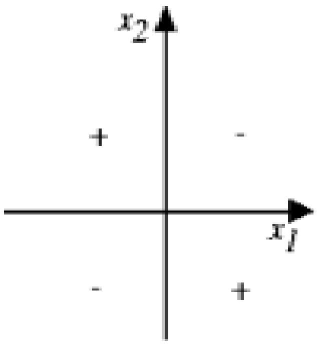
where w_i is a real-valued constant, or weight

Perceptrons (cont.)

- Decision surface of a perceptron: $\vec{w} \cdot \vec{x} = 0$
  - Linearly separable case like (*a*):
    Possible to classify by hyperplane,
  - Linearly inseparable case like (*b*):
    Impossible to classify



(*a*)



(*b*)

Perceptrons (cont.)

- Examples of Boolean functions

  - AND :

    possible to classify when w_0 = -0.8, $w_1 = w_2$ = 0.5

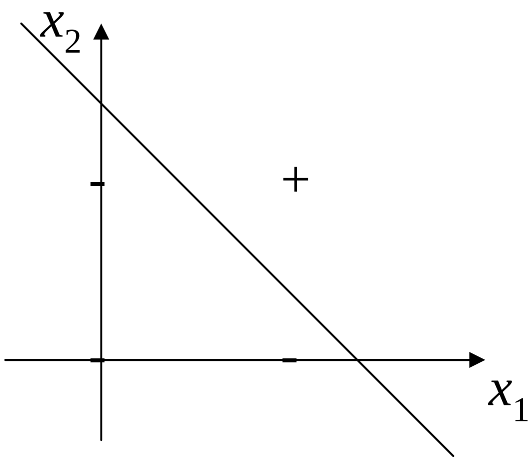
<Training examples>

x_1	x_2	output
0	0	–1
0	1	–1
1	0	–1
1	1	1

Decision hyperplane :

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$-0.8 + 0.5\, x_1 + 0.5\, x_2 = 0$$



$$-0.8 + 0.5\, x_1 + 0.5\, x_2 = 0$$

<Test Results>

x_1	x_2	$\Sigma w_i x_i$	output
0	0	–0.8	–1
0	1	–0.3	–1
1	0	–0.3	–1
1	1	0.2	1

Perceptrons (cont.)

- Examples of Boolean functions

  - OR :

    possible to classify when w_0 = -0.3, $w_1 = w_2$ = 0.5

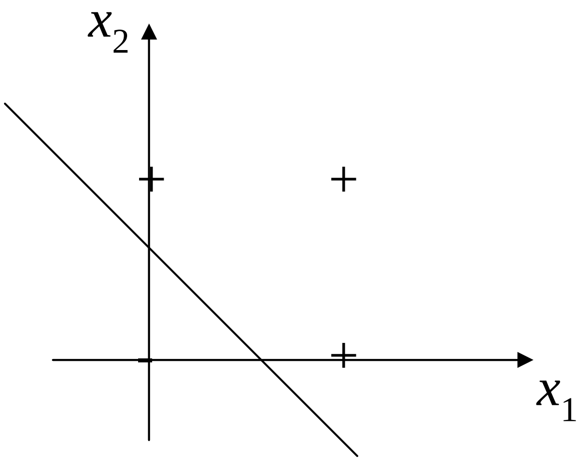
<Training examples>

x_1	x_2	output
0	0	–1
0	1	1
1	0	1
1	1	1

Decision hyperplane :

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$-0.3 + 0.5\,x_1 + 0.5\,x_2 = 0$$



$$-0.3 + 0.5\,x_1 + 0.5\,x_2 = 0$$

<Test Results>

x_1	x_2	$\Sigma\, w_i x_i$	output
0	0	–0.3	–1
0	1	0.2	–1
1	0	0.2	–1
1	1	0.7	1

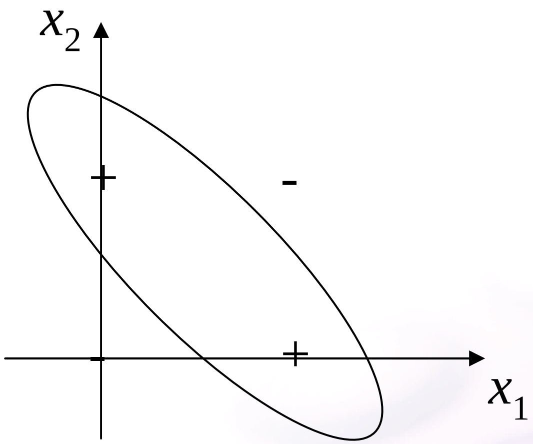
Perceptrons (cont.)

- Examples of Boolean functions

  - XOR :

    impossible to classify because this case is linearly inseparable

<Training examples>		
x_1	x_2	output
0	0	−1
0	1	1
1	0	1
1	1	−1



cf) Two-layer network of perceptrons can represent XOR.

Refer to this equation, $$x_1 \oplus x_2 = x_1\bar{x}_2 + \bar{x}_1 x_2$$

Perceptrons (cont.)

- Perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

$$\text{where} \quad \Delta w_i = \eta\,(t - o)\,x_i$$

Where:

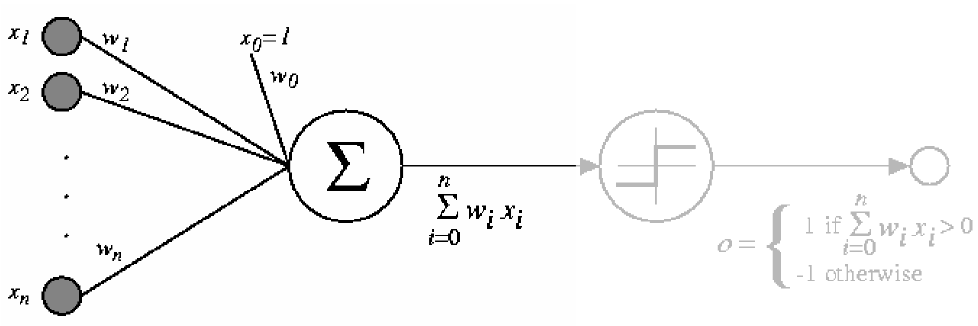
- $t = c(x)$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

Can prove it will converge

- If training data is linearly separable
- and η sufficiently small

Perceptrons (cont.)

- Delta rule



  – Unthresholed, just using *linear unit*, (differentiable)

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

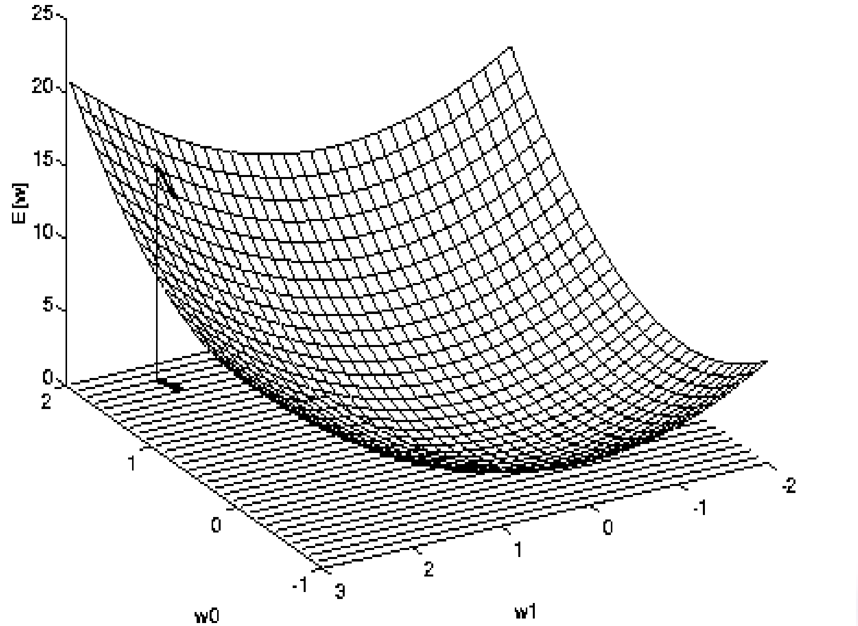
  – Training Error : Learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

  – Where D is set of training examples

Perceptrons (cont.)

- Hypothesis space



- w_o, w_1 plane represents the entire hypothesis space.

- For linear units, this error surface must be parabolic with a single global minimum. And we desire a hypothesis with this minimum.

Perceptrons (cont.)

- Gradient (steepest) descent rule

  - Error (for all Training ex.):

$$E[\vec{w}] \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

  - Gradient of *E* (Partial Differentiating) :

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

  - direction : steepest increase in *E*.

  - Thus, training rule is as follows.

$$\Delta\vec{w} = -\eta\nabla E[\vec{w}] \qquad \Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

(The negative sign : decreases *E*.)

Perceptrons (cont.)

- Derivation of gradient descent

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

where x_{id} denotes the single input components x_i

$$\therefore \quad \Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

- Because the error surface contains only a single global minimum, this algorithm will converge to a weight vector with minimum error, regardless of whether the tr. examples are linearly separable, given a sufficiently small η is used.

Perceptrons (cont.)

- Gradient descent and delta rule
  - Search through the space of possible network weights, iteratively reducing the error *E* between the training example target values and the network outputs

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

Perceptrons (cont.)

- Stochastic approximation to gradient descent

Batch mode Gradient Descent:
Do until satisfied

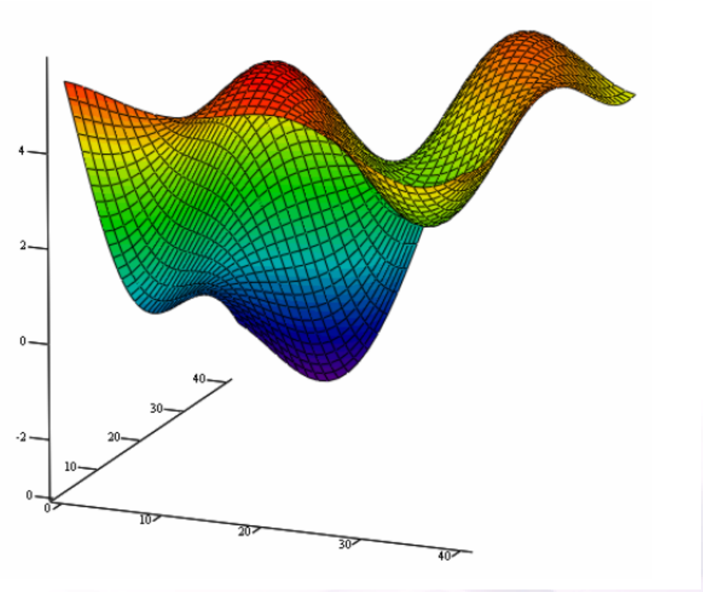
1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode Gradient Descent:
Do until satisfied

- For each training example d in D
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$



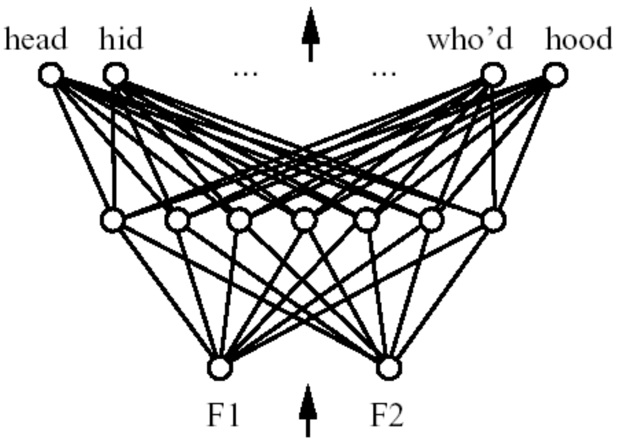
Stochastic gradient descent (i.e. incremental mode) can sometimes avoid falling into local minima because it uses the various gradient of *E* rather than overall gradient of *E*.

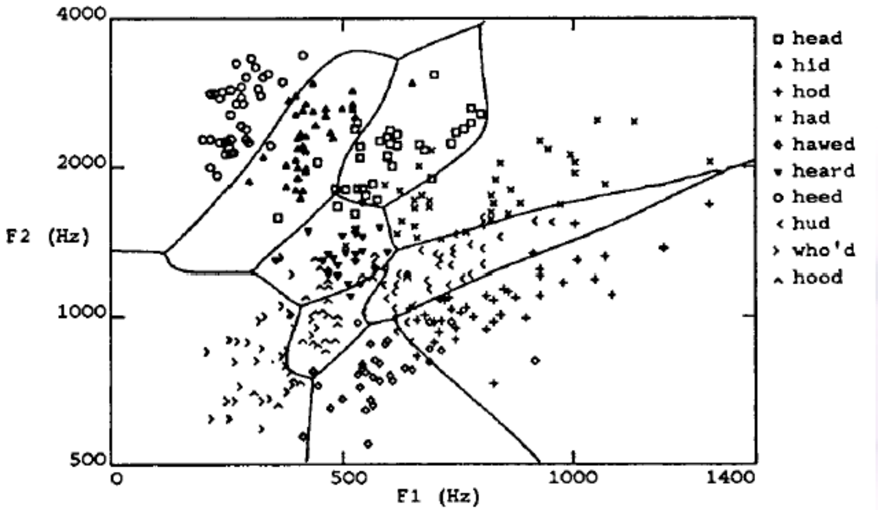
Perceptrons (cont.)

- Summary
  - Perceptron training rule
    - Perfectly classifies training data
    - Converge, provided the training examples are linearly separable
  - Delta Rule using gradient descent
    - Converge asymptotically to min. error hypothesis
    - Converge regardless of whether training data are linearly separable

Multilayer Networks and Backpropagation Algorithm

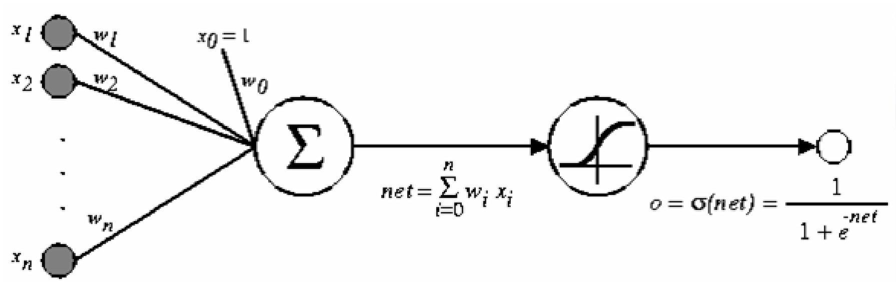
- Speech recognition example of multilayer networks learned by the backpropagation algorithm
- Highly nonlinear decision surfaces





Multilayer Networks and Backpropagation Algorithm (cont.)

- Sigmoid threshold unit
  - What type of unit as the basis for multilayer networks
    - Perceptron : not differentiable -> can’t use gradient descent
    - Linear Unit : multi-layers of linear units -> still produce only linear function
    - Sigmoid Unit : differentiable threshold function



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1+e^{-x}}$$

Multilayer Networks and Backpropagation Algorithm (cont.)

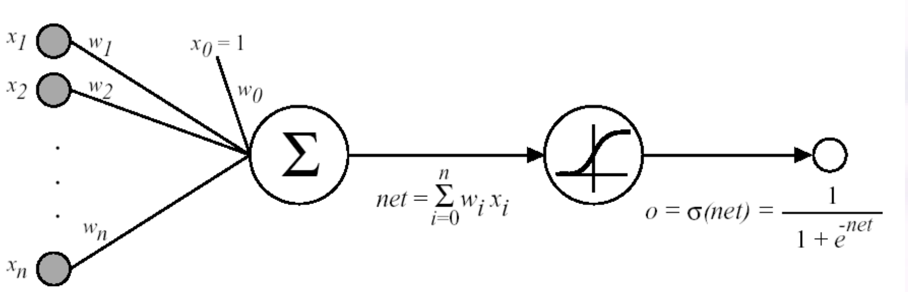
- Sigmoid threshold unit

  - Interesting property: $$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

  - Output ranges between 0 and 1

  - We can derive gradient decent rules to train One sigmoid unit

  - Multilayer networks of sigmoid units $\rightarrow$ Backpropagation

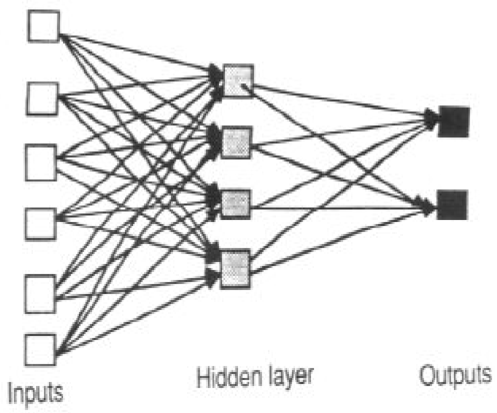


Multilayer Networks and Backpropagation Algorithm (cont.)

- Backpropagation algorithm
  - Two layered feedforward networks

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k
  $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$
  3. For each hidden unit h
  $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$
  4. Update each network weight $w_{i,j}$
  $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$
  where
  $$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$



Multilayer Networks and Backpropagation Algorithm (cont.)

- Adding momentum

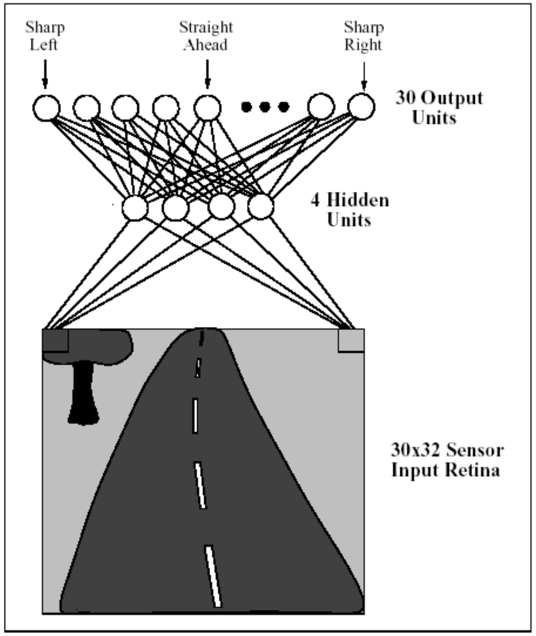
  - Another weight update is possible.

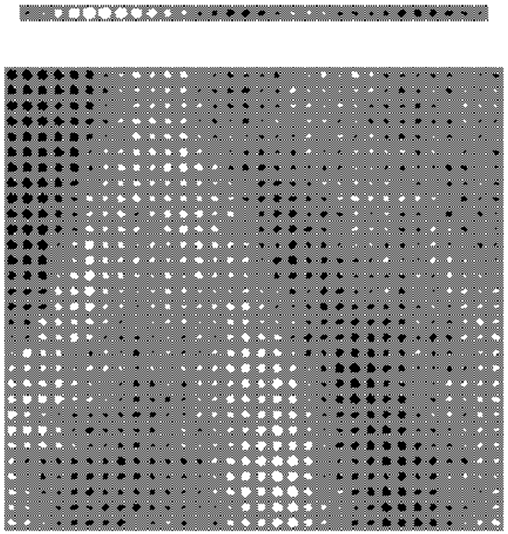
$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

  - n-th iteration update depend on (n-1)th iteration
  - α : constant between 0 and 1 -> momentum
  - Role of momentum term :
    - Keep the ball rolling through small local minima in the error surface.
    - Gradually increase the step size of the search in regions where the gradient is unchanging, thereby speeding convergence

Multilayer Networks and Backpropagation Algorithm (cont.)

- ALVINN (again..)
  - Uses backpropagation algorithm
  - Two layered feedforward network
    - 960 neural network inputs
    - 4 hidden units
    - 30 output units





Remarks on Backpropagation Algorithm

- Convergence and local minima
  - Gradient descent to some local minimum
    - Perhaps not global minimum...
      - Add momentum
      - Stochastic gradient descent
      - Train multiple nets with different initial weights

Remarks on Backpropagation Algorithm (cont.)

- Expressive capabilities of ANNs
  - Boolean functions:
    - Every boolean function can be represented by network with two layers of units where the number of hidden units required grows exponentially.
  - Continuous functions:
    - Every bounded continuous function can be approximated with arbitrarily small error, by network with two layers of units [Cybenko 1989; Hornik et al. 1989]
  - Arbitrary functions:
    - Any function can be approximated to arbitrary accuracy by a network with three layers of units [Cybenko 1988].

Remarks on Backpropagation Algorithm (cont.)

- Hypothesis space search and Inductive bias
  - Hypothesis space search
    - Every possible assignment of network weights represents a syntactically distinct hypothesis.
    - This hypothesis space is continuous in contrast to that of decision tree.
  - Inductive bias
    - One can roughly characterize it as smooth interpolation between data points. (Consider a speech recognition example!)

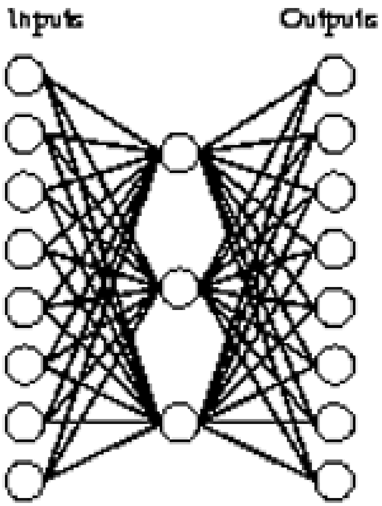
Remarks on Backpropagation Algorithm (cont.)

- Hidden layer representations

  - This 8x3x8 network was trained to learn the identity function.

  - 8 training examples are used.

  - After 5000 training iterations, the three hidden unit values encode the eight distinct inputs using the encoding shown on the right.

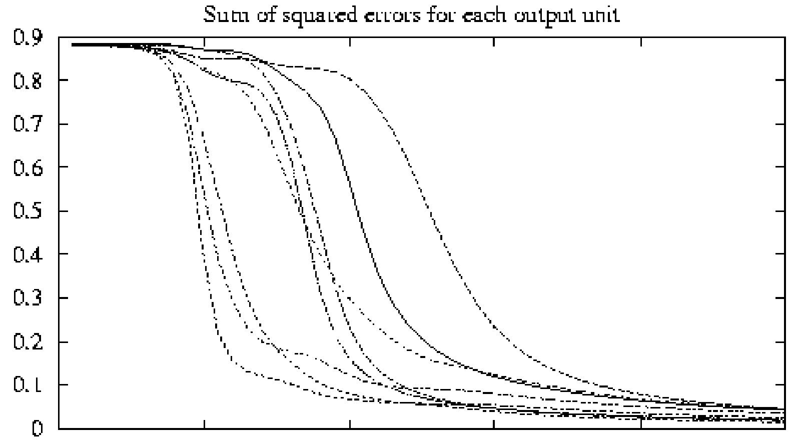


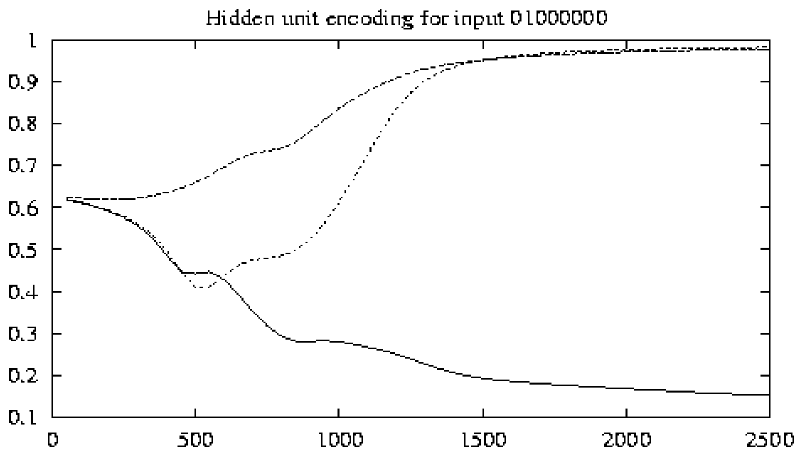
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

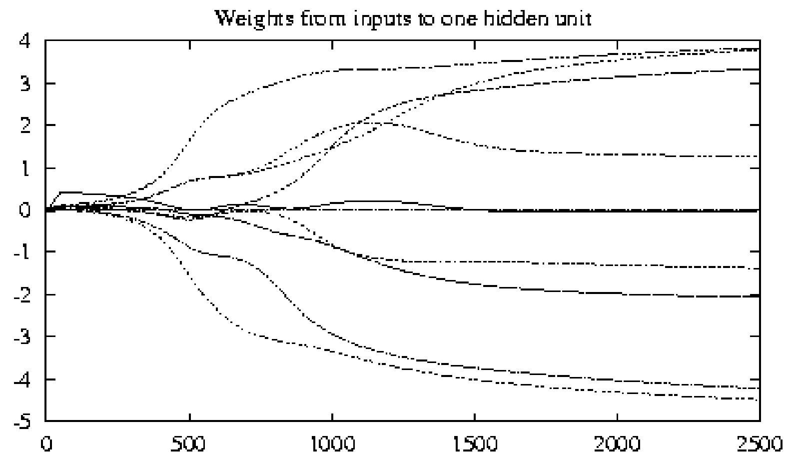
Remarks on Backpropagation Algorithm (cont.)

- Learning the 8x3x8 network

  - Most of the interesting weight changes occurred during the first 2500 iterations.





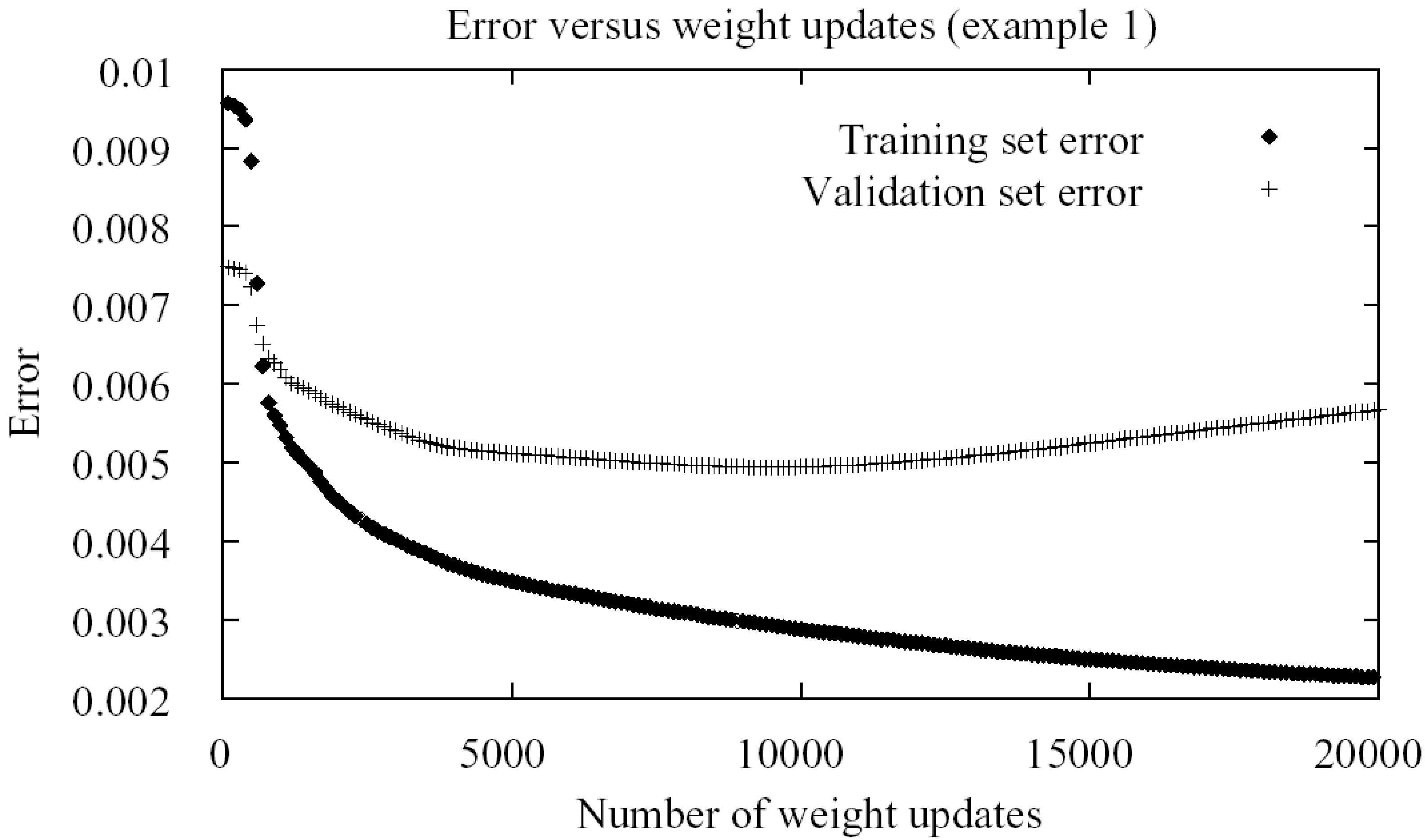


Remarks on Backpropagation Algorithm (cont.)

- Generalization, overfitting, and stopping criterion
  - Termination condition
    - Until the error E falls below some predetermined threshold (overfitting problem)
  - Techniques to address the overfitting problem
    - Weight decay : Decrease each weight by some small factor during each iteration.
    - Cross-validation
    - k-fold cross-validation (small training set)

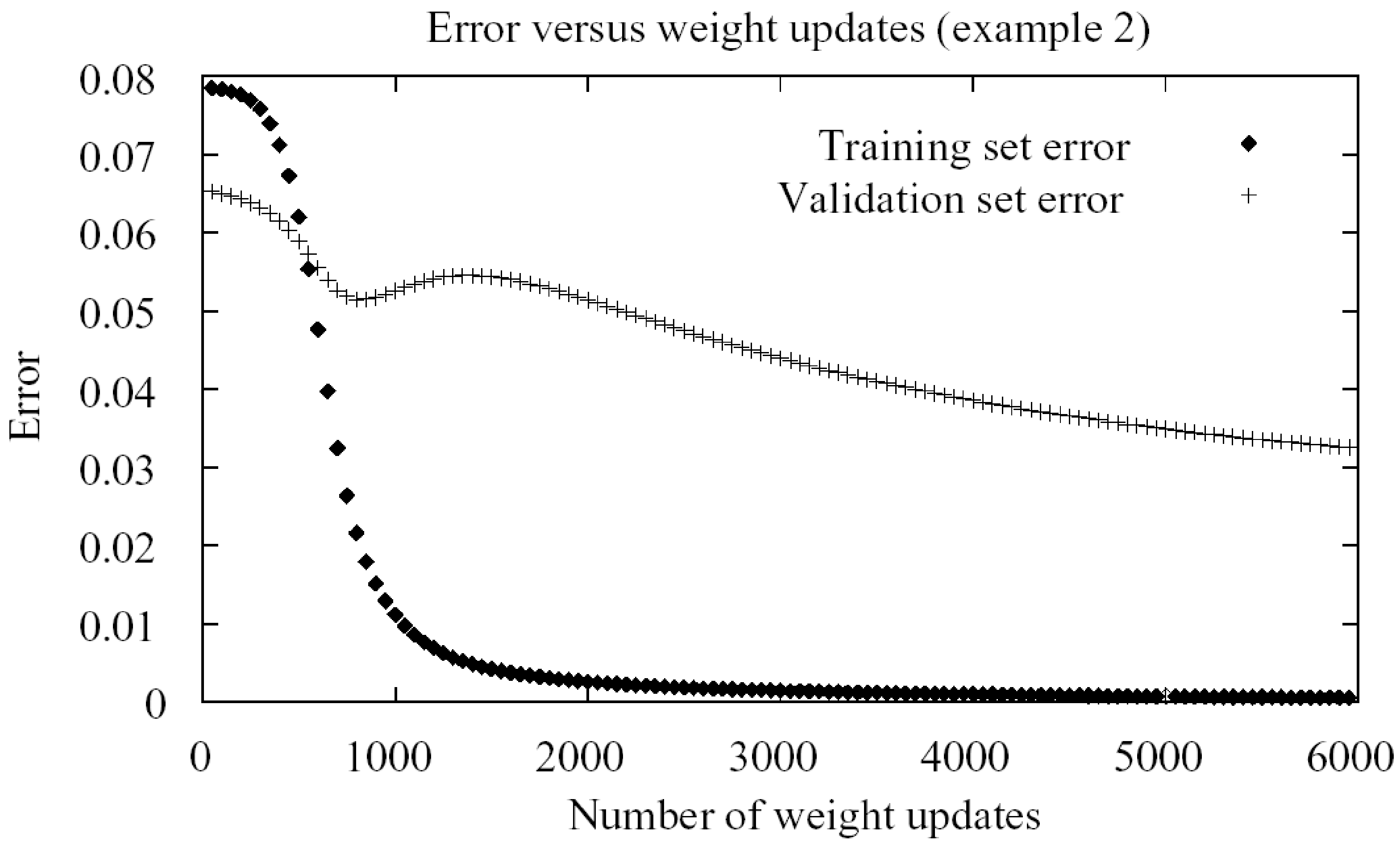
Remarks on Backpropagation Algorithm (cont.)

- Overfitting in ANNs



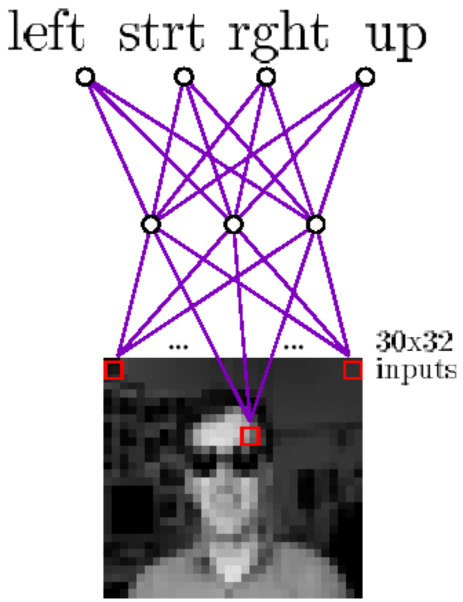
Remarks on Backpropagation Algorithm (cont.)

- Overfitting in ANNs



An Illustrative Example: Face Recognition

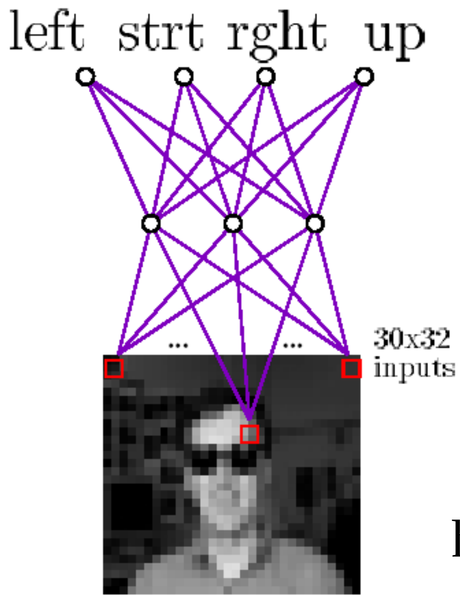
- Neural nets for face recognition
  - Training images : 20 different persons with 32 images per person.
  - (120x128 resolution → 30x32 pixel image)
  - After 260 training images, the network achieves an accuracy of 90% over a separate test set.
  - Algorithm parameters : η =0.3, α =0.3

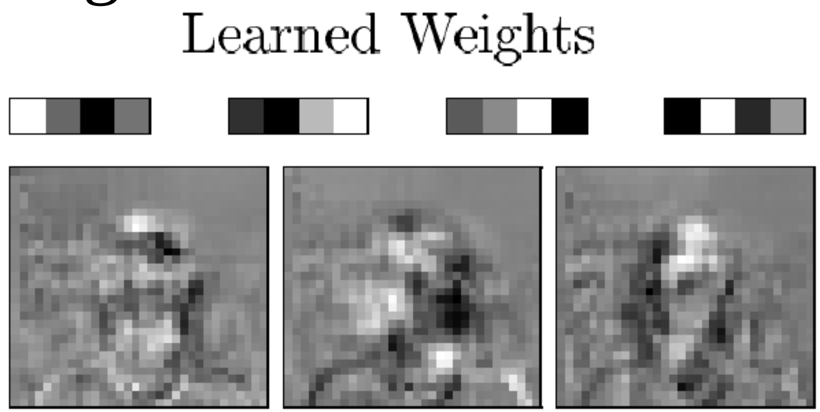


Typical input images

An Illustrative Example: Face Recognition (cont.)

- Learned hidden unit weights





http://www.cs.cmu.edu/tom/faces.html

Typical input images

- Alternative error functions
  - Penalize large weights: (weight decay) : Reducing the risk of overfitting

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

  - Train on target slopes as well as values:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} \left[(t_{kd} - o_{kd})^2 + \mu \sum_{j \in inputs} \left(\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j} \right)^2 \right]$$

  - Minimizing the cross entropy : Learning a probabilistic output function (chapter 6)

$$-\sum_{d \in D} t_d \log o_d + (1 - t_d) \log(1 - o_d)$$

where target value, $t_d \in \{0,1\}$ and o_d is the probabilistic output from the learning system, approximating the target function, $f'(x_d) = p(f(x_d) = t_d = 1)$ where $d = \langle x_d, t_d \rangle$, $d \in D$

Advanced Topics in Artificial Neural Networks (cont.)

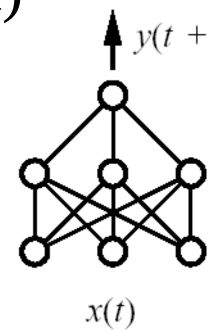
- Alternative error minimization procedures
  - Weight-update method
    - Direction : choosing a direction in which to alter the current weight vector (ex: the negation of the gradient in Backpropagation)
    - Distance : choosing a distance to move (ex: the learning ratio η)

Ex : Line search method, Conjugate gradient method

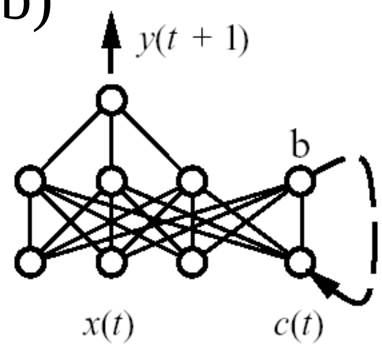
Advanced Topics in Artificial Neural Networks (cont.)

- Recurrent networks

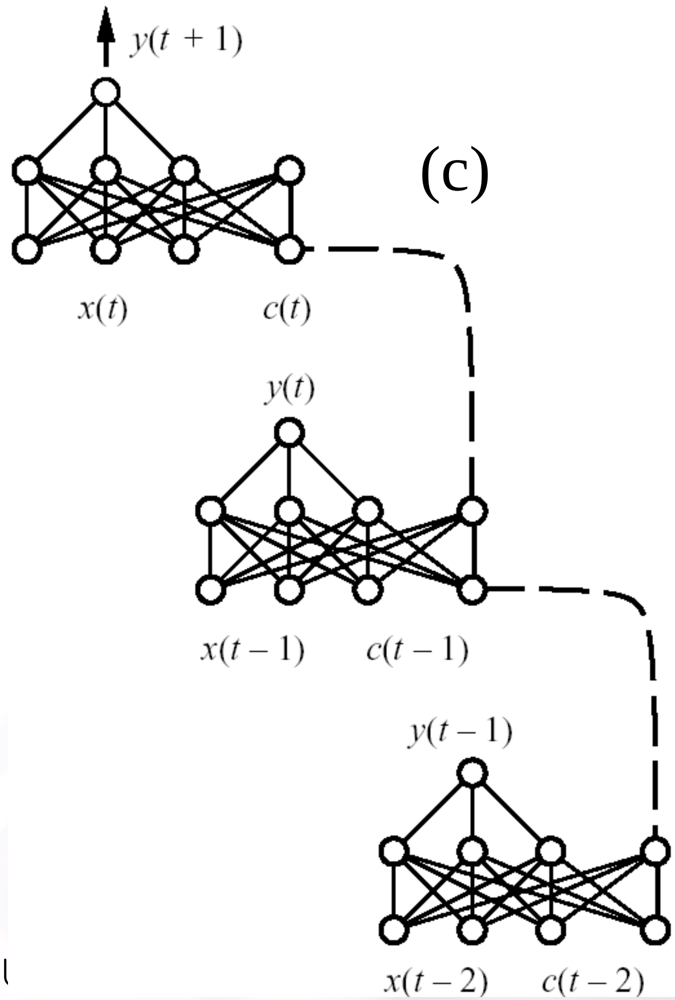
(a)



(b)



(c)



(a) Feedforward network
(b) Recurrent network
(c) Recurrent network unfolded in time

Advanced Topics in Artificial Neural Networks (cont.)

- Dynamically modifying network structure
  - To improve generalization accuracy and training efficiency
  - Cascade-Correlation algorithm (Fahlman and Lebiere 1990)
    - Start with the simplest possible network and add complexity
  - Optimal brain damage (Lecun *et al*. 1990)
    - Start with the complex network and prune it as we find that certain connectives are inessential.