

Artificial Neural Networks

- **Learning** real-valued, discrete-valued, and vector-valued **functions** from examples.
- **Robust to errors** in training data.
- **Applications**: interpreting visual scenes, speech recognition, robot control strategies

Biological Motivation

- Human brain = 10^{11} neurons
- Each connected to 10^4 others
- Switching time = 10^{-3} seconds
(Computer switching speed = 10^{-10} seconds)
- It requires 10^{-1} seconds to recognize a human face

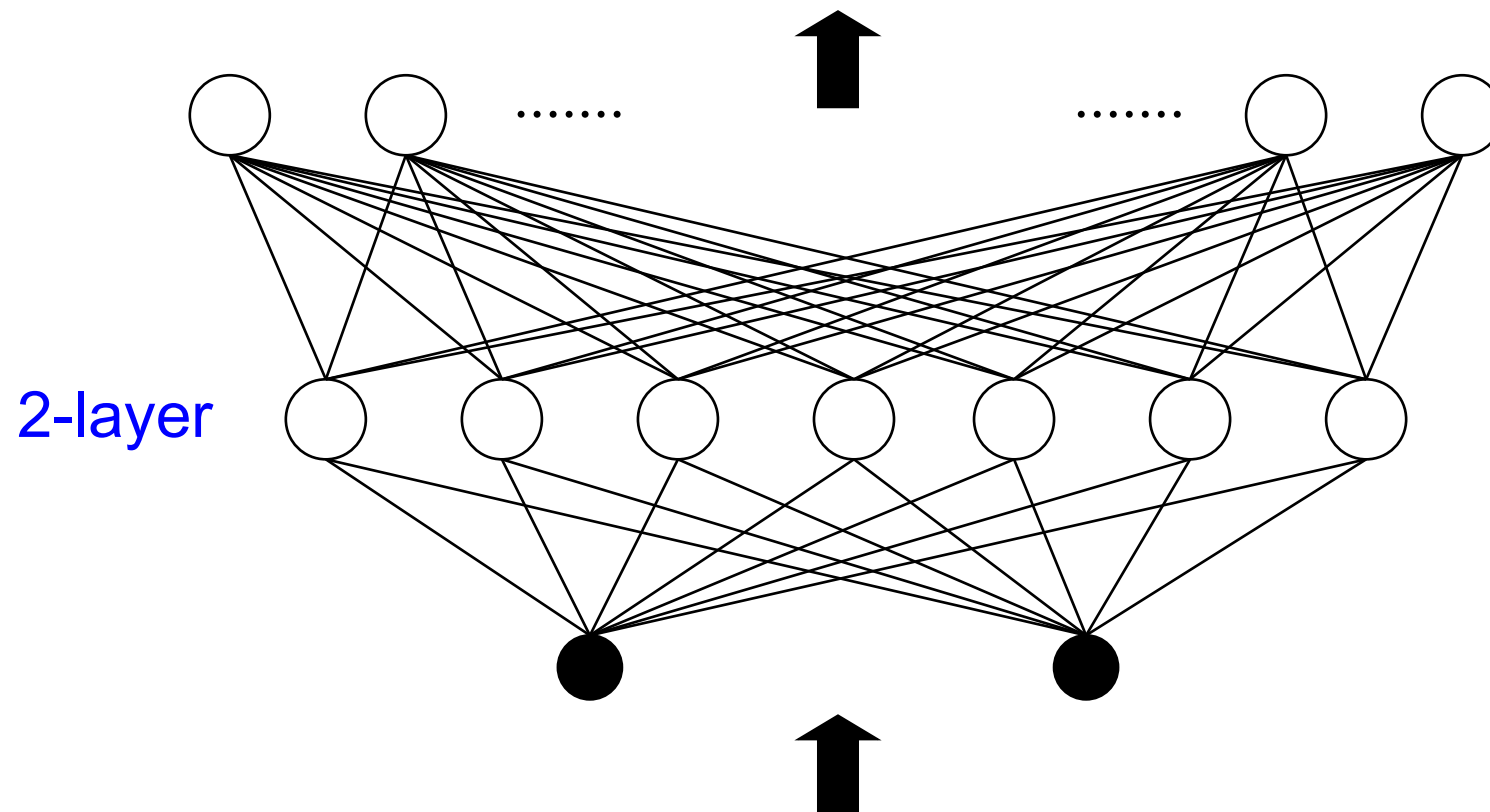
⇒ highly parallel and distributed processes

Biological Motivation

- ANN model is **not the same** as that of biological neural systems
- Using ANNs to study and model **biological learning processes**

Obtaining highly **effective machine learning algorithms**

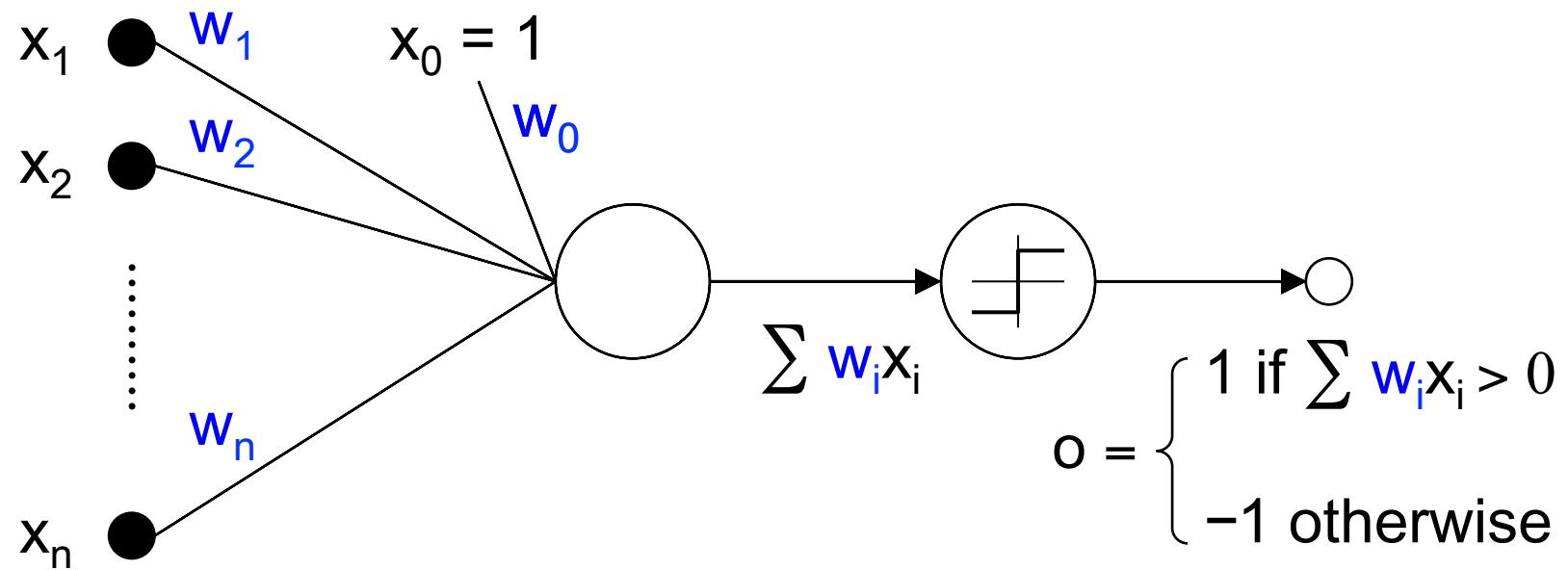
ANN Representation



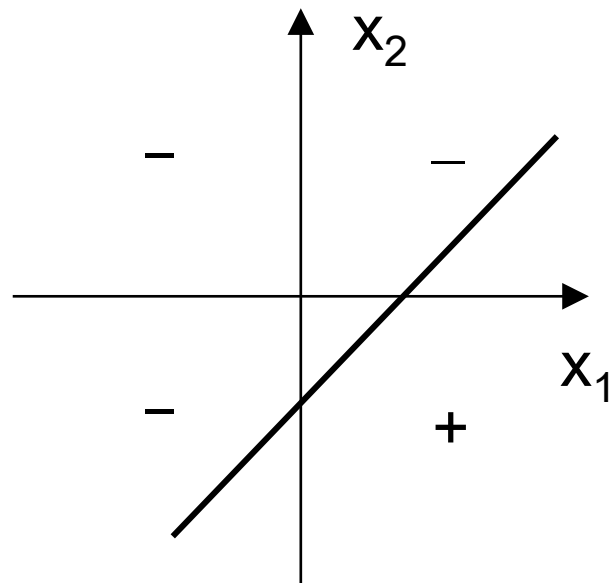
Appropriate Problems for ANNs

- Instances are represented by many attribute-value pairs
- Target function output may be discrete-valued, real-valued, vector-valued
- Training examples can contain errors
- Long training time is acceptable
- Fast evaluation of the learned target function may be required
- Understanding the learned target concept is not important

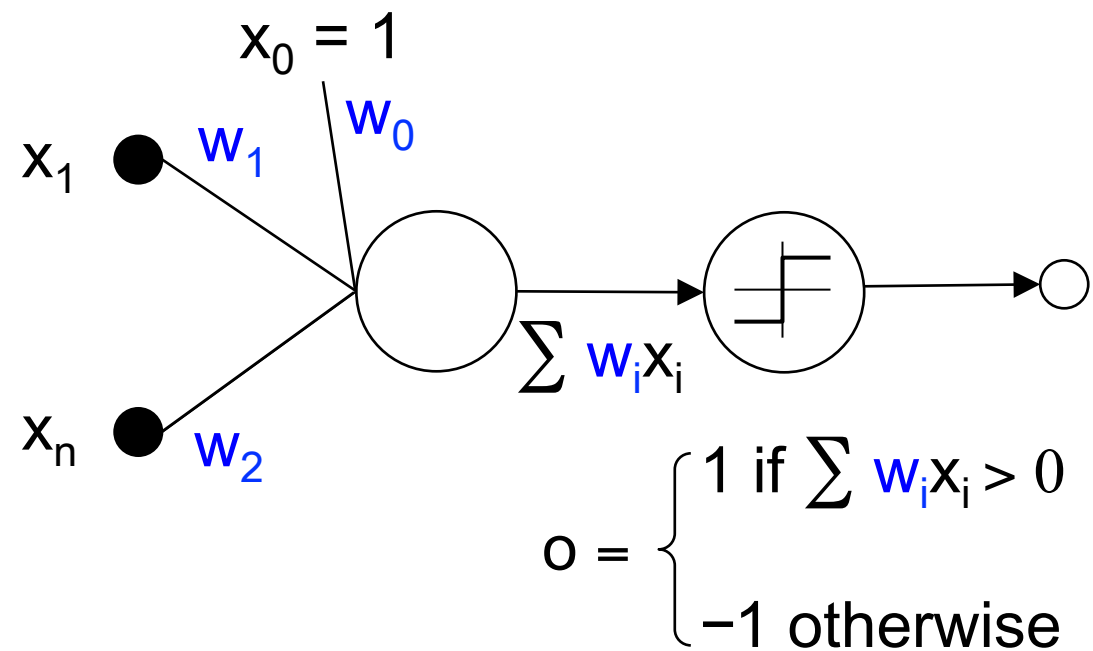
Perceptrons



Perceptrons



$A \wedge \neg B$



Perceptron Training Rule

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

$$\Delta w_i = \eta(t - o)x_i$$

- t target output of the current training example
- o the **thresholded** output generated by the perceptron
- η learning rate (positive constant)

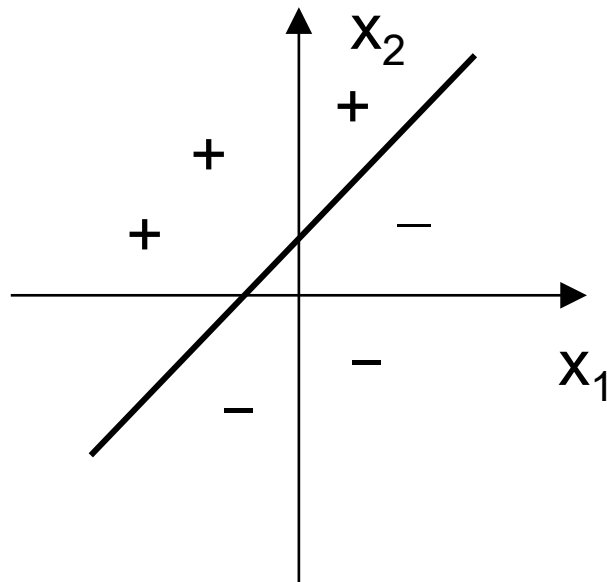
Perceptron Training Rule

w_0	w_1	w_2	x_1	x_2	$t = x_1 \wedge \neg x_2$	$o = /w_0 + w_1x_1 + w_2x_2/$	Δw_0	Δw_1	Δw_2
1	0	1	1	1	-1	1	-2	-2	-2
-1	-2	-1	1	-1	1	-1	2	2	-2
1	0	-3	-1	1	-1	-1	0	0	0
1	0	-3	-1	-1	-1	1	-2	2	2
-1	2	-1	1	1	-1	-1	0	0	0
-1	2	-1	1	-1	1	1	0	0	0
-1	2	-1	-1	1	-1	-1	0	0	0
-1	2	-1	-1	-1	-1	-1	0	0	0

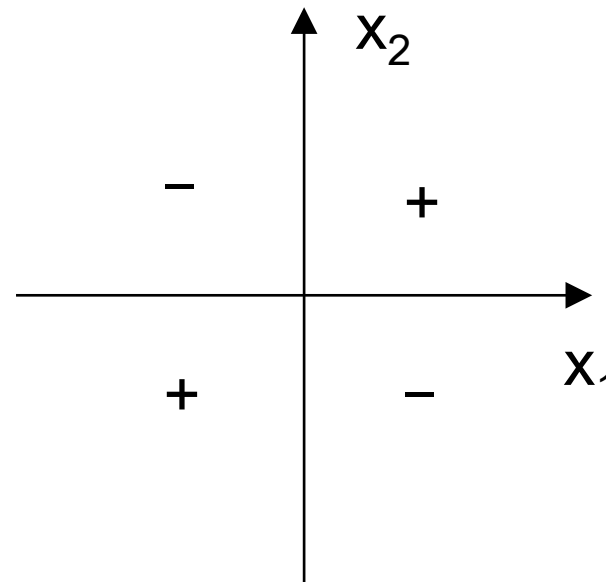
$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = (t - o)x_i$$

Perceptron Training Rule



linearly separable

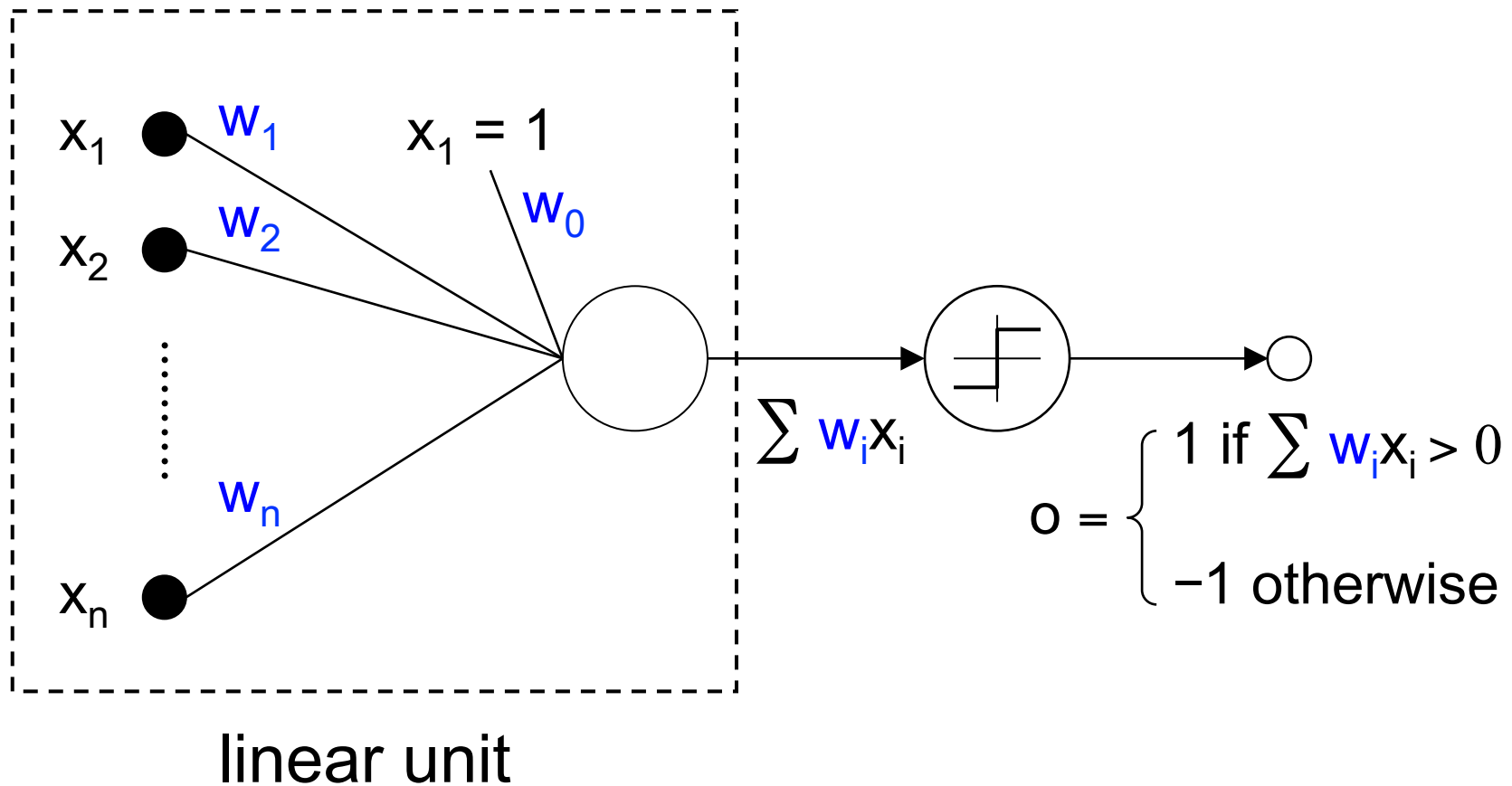


non linearly separable

Perceptron Training Rule

- The learning procedure converges to a weight vector that correctly classifies all **linearly separable** training examples
- Non-differentiable

Delta Rule (using gradient descent)



Delta Rule

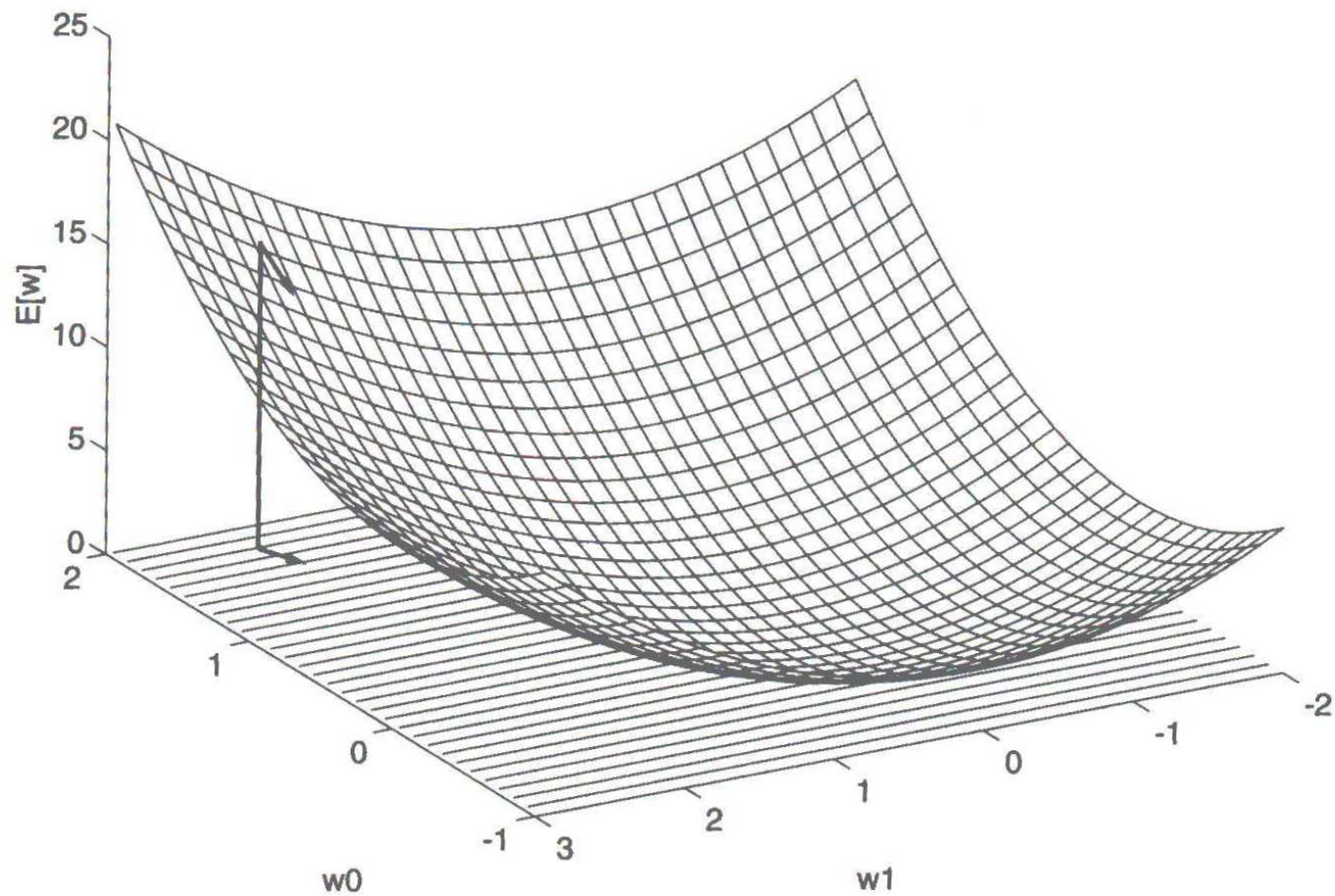
Training error (for the **linear unit**):

$$E(\vec{w}) = \sum_{d \in D} (t_d - o_d)^2 / 2$$

t_d target output of training example d

o_d the **unthresholded** output for d ($= \vec{w} \cdot \vec{x}_d$)

Delta Rule



Delta Rule

Gradient of E (steepest **increase** direction):

$$\nabla E(\vec{w}) = [\partial E / \partial w_0, \partial E / \partial w_1, \dots, \partial E / \partial w_n]$$

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

$$\vec{w} \leftarrow \vec{w} - \eta \nabla E(\vec{w})$$

Delta Rule

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

$$\Delta w_i = -\eta \partial E / \partial w_i$$

$$\partial E / \partial w_i = -\sum_{d \in D} (t_d - o_d) x_{id}$$

$$\Delta w_i = \eta \underbrace{\sum_{d \in D} (t_d - o_d)}_{\delta} x_{id}$$

Delta Rule

- Converging to a local minimum can be quite slow
- No guarantee to converge to the global minimum

Stochastic Approximation

Delta rule:

$$E(\vec{w}) = (t_d - o_d)^2/2$$

$$\Delta w_i = \eta \underbrace{(t_d - o_d)}_{\delta} x_{id}$$

Stochastic Approximation

- Weights are updated upon **examining each training example**
- **Less computation** per weight update step is required
- Falling into **local minima** can be **avoided**

Stochastic Approximation

- The delta rule converges towards a **best-fit approximation** to the target concept, **regardless** of whether the training data are **linearly separable**

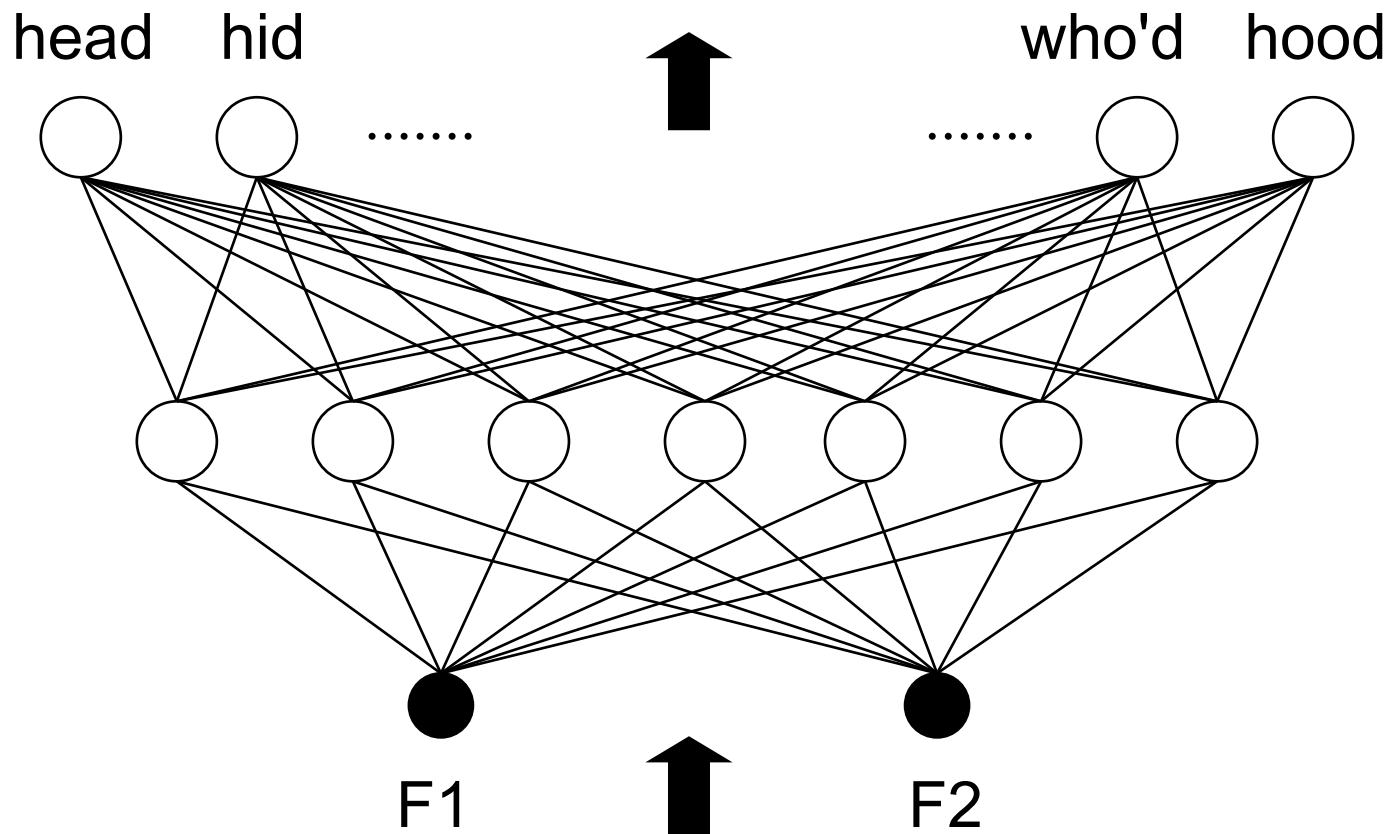
Single Perceptrons

- Single perceptrons can express only **linear** decision surfaces
- Minsky, M. & Papert, S. (1969). **Perceptrons**. MIT Press.

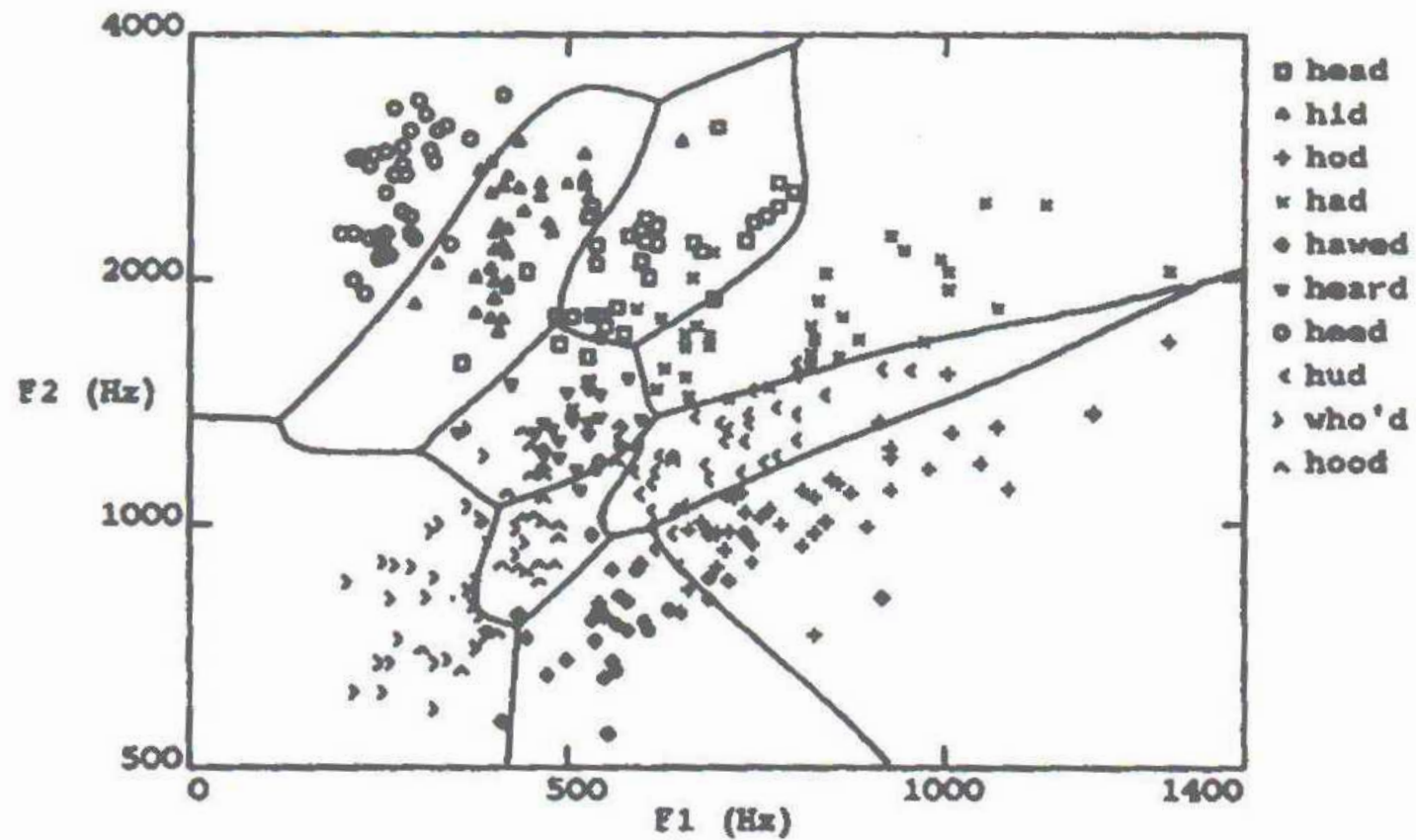
Multilayer Networks

- A multilayer network can represent highly **nonlinear** decision surfaces

Multilayer Networks



Multilayer Networks

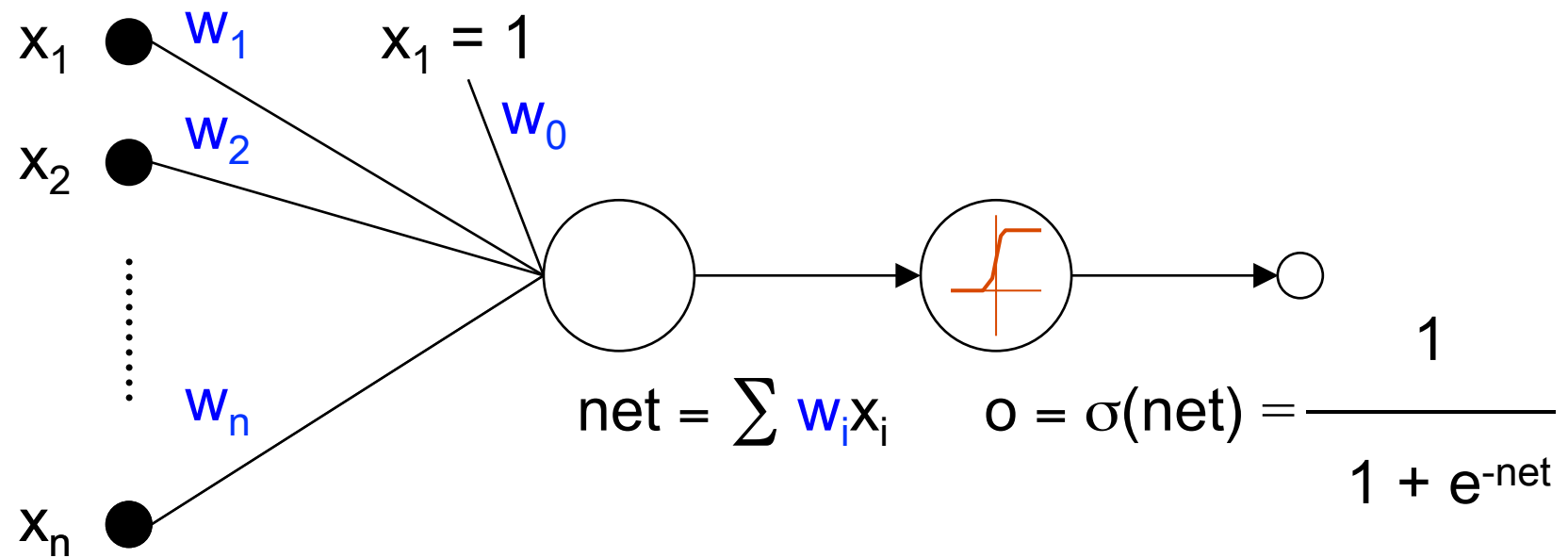


Multilayer Networks

What type of unit ?

- Perceptrons: non-differentiable
- Linear units: only linear functions
-

Multilayer Networks



sigmoid unit

Multilayer Networks

Sigmoid unit:

$$\partial\sigma(y)/\partial y = \sigma(y).(1 - \sigma(y)) = o(1 - o)$$

Backpropagation Algorithm

Training error:

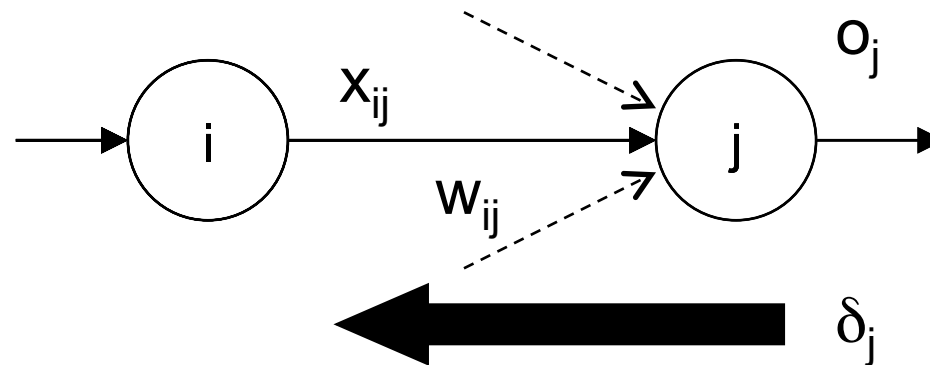
$$E(\vec{w}) = \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2 / 2$$

or stochastic approximation:

$$E(\vec{w}) = \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2 / 2$$

Backpropagation Algorithm

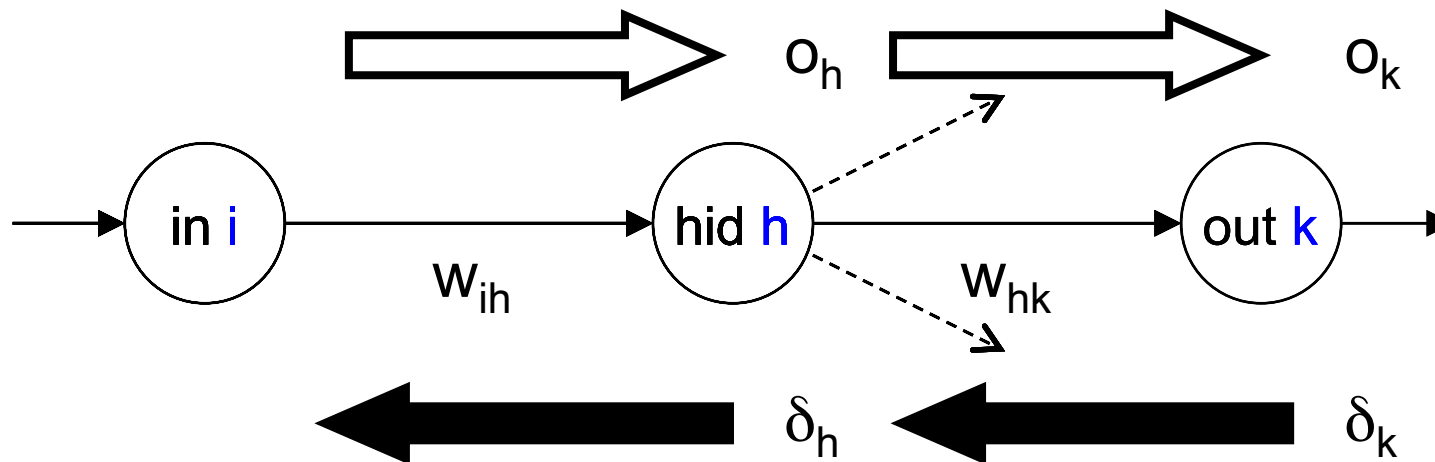
Weight update:



$$\Delta w_{ij} = -\eta \partial E / \partial w_{ij} = \eta \delta_j x_{ij}$$

$$\delta_j = o_j(1 - o_j)(t_j - o_j)$$

Backpropagation Algorithm



$$\delta_h = o_h(1 - o_h) \sum_k w_{hk} \delta_k$$

$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$

Backpropagation Algorithm

Adding momentum:

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

\downarrow \downarrow

iteration momentum ($0 \leq \alpha \leq 1$)

- Keeping the search direction \Rightarrow passing small local minima
- Increasing the search step size \Rightarrow speeding convergence

Backpropagation Algorithm

Convergence and local minima:

- Not guaranteed to converge towards the global minimum error, but highly **effective in practice**
- Approximately linear when the weights are close to 0 (due to sigmoid), hence **avoiding local minima** of non-linear functions

Backpropagation Algorithm

Heuristics to alleviate the local minima problem:

- Add a **momentum** term to the weight-update rule
- Use **stochastic gradient descent** rather than true gradient descent
- Train multiple networks using the same data, but **initializing** each network with **different random weights**

Backpropagation Algorithm

Representation power of feedforward networks:

- **Boolean functions**: any one, using **2-layer** (1 hidden + 1 output) networks
- **Continuous functions**: any bounded one with approximation, using **2-layer** networks
- **Arbitrary functions**: any one with approximation, using **3-layer** networks

Backpropagation Algorithm

Hypothesis space and inductive bias:

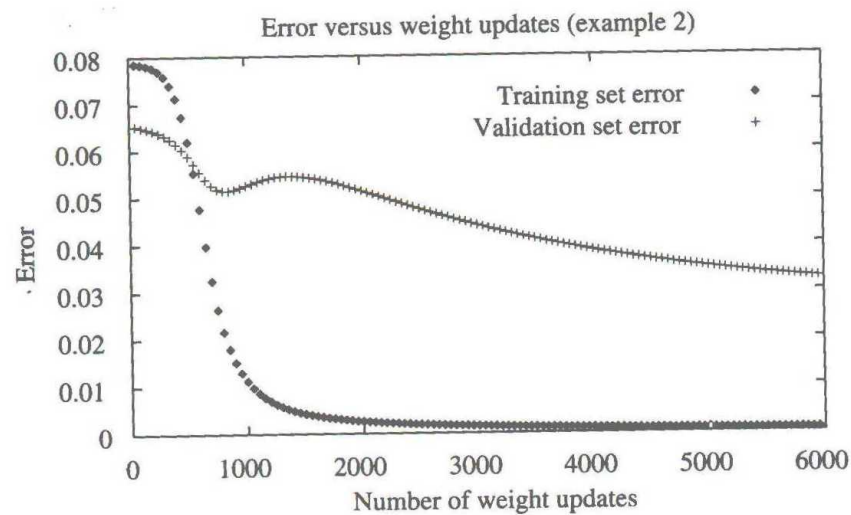
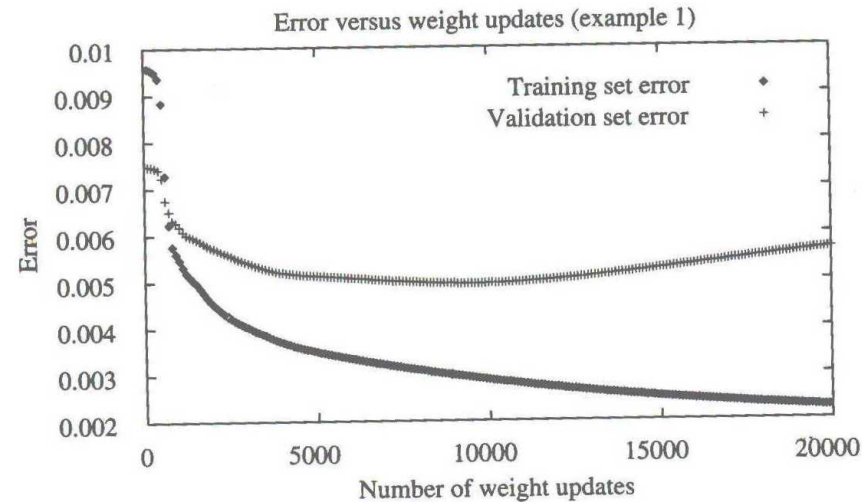
- **Hypothesis**: every possible assignment of network weights
- **Inductive bias**: smooth interpolation between data points

Backpropagation Algorithm

Stopping criterion and overfitting:

- Number of iterations
- Limit of training errors

Backpropagation Algorithm



Applications

To recognize face pose:

- 30 × 32 resolution input images
- 4 directions: left, straight, right, up

⇒ 960 × 3 × 4 network

Exercises

- In Mitchell's ML (Chapter 4): 4.1 to 4.10