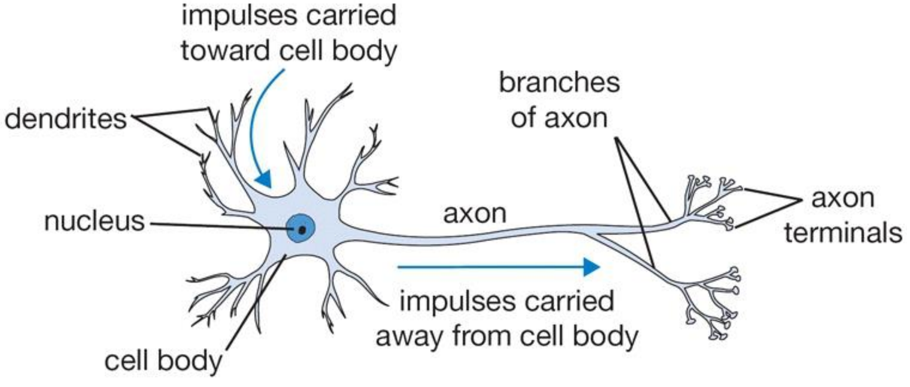
Machine Learning: Neural Networks

Russell Chap. 18.7

Luger Chap. 11

The Brain

- The brain consists of
  - 1 billion ~10 billion neurons
  - 100 ~ 100000 connections / neuron
- Operation of neuron
  - Propagation of electo-chemical signal
  - Operates in milliseconds



Characteristics of the Brain

- Parallel / distributed processing
  - Understanding scene or sentence by computer
    → more than 1 second = 1000 million steps
  - Understanding scene or sentence by brain
    → less than 1 second = 1000 steps
- Robustness
  - Failure of one component in a computer
    → total failure
  - Failure of one component in a brain
    → still performs well

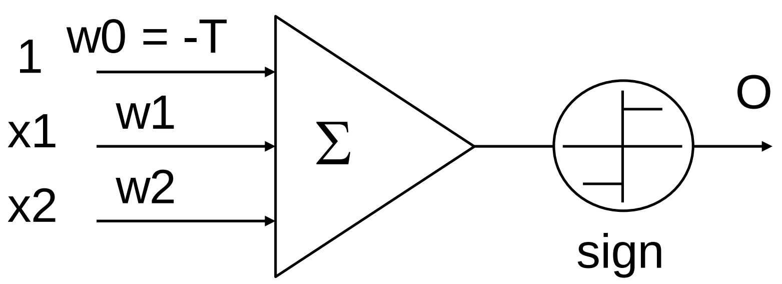
Connectionist Model

- Motivation
  - Brain-like performance
- Features
  - Large number of simple processing elements
  - Large number of weighted connections
  - Parallel, distributed processing
  - Automatic learning
- Parallel Distributed Processing (PDP)
- Neural Networks

Perceptron

- A model of neuron
  - McCulloch and Pitts [1943], Rosenblatt [1958]
- Function
  - Input x_i, weight w_i, threshold T
  - Output = 1 if $\Sigma x_i w_i \geq T$ ($O = \mathrm{sign}(\Sigma x_i w_i)$)

    -1 otherwise

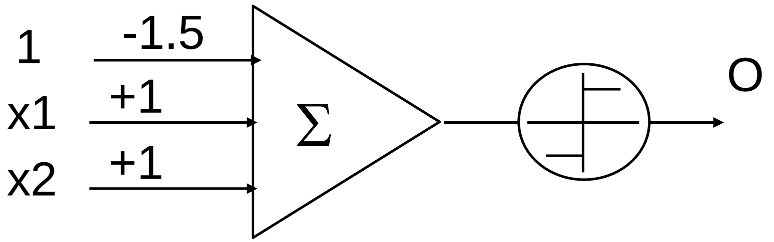


$x1w1 + x2w2 \geq T$

$x1w1 + x2w2 - T \geq 0$

Perceptron

- Example



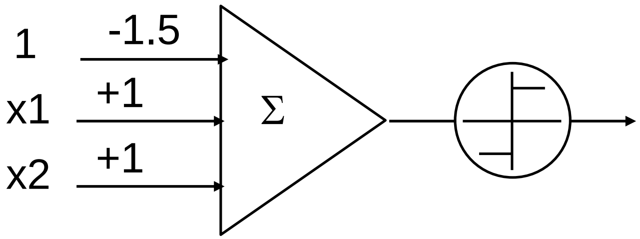
x1	x2	O
0	0	−1
0	1	−1
1	0	−1
1	1	1

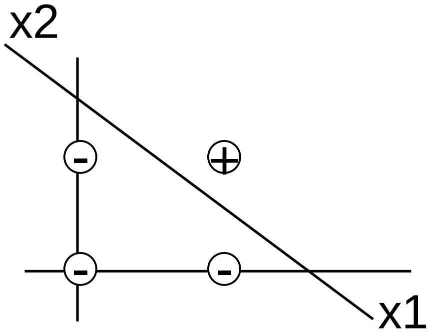
Perceptron

- Represent linearly-separable function
  - Output = 1 for input (x_1, x_2) such that

    $x_1 w_1 + x_2 w_2 - T \geq 0 \Rightarrow x_2 \geq -(w_1 / w_2) x_1 + (T / w_2)$
- Example
  - $w_1 = 1$, $w_2 = 1$, $T = 1.5$

    : $O = 1$ if $x_2 + x_1 - 1.5 \geq 0 \Rightarrow x_2 \geq -x_1 + 1.5$



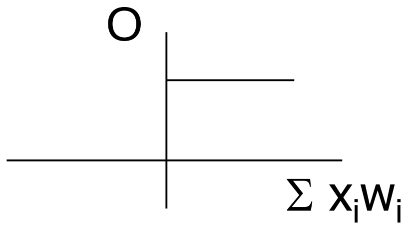


Learning of Perceptron

- Learning
  - Adjusting weights so that it can produce right output for given example <X, d>
    - If d = 1 but O = -1

      → increase weights of + input
    - If d = -1 but O = 1

      → decrease weights of + input



$$\boxed{\Delta w_i = c\,(d - O)\,x_i}$$

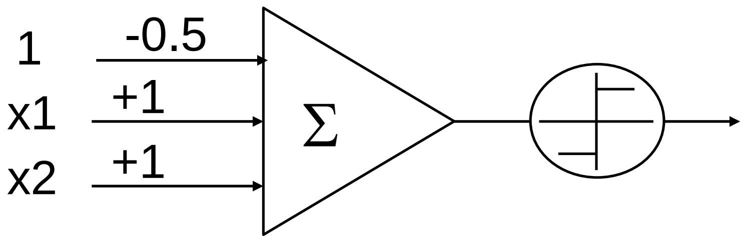
- d: desired output
- (d - O): error
- c: learning rate

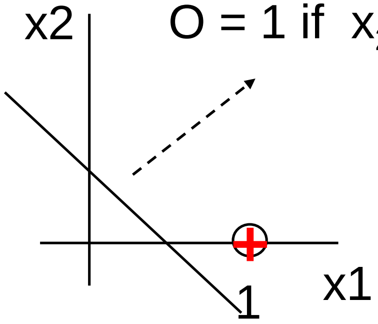
Learning of Perceptron

- Example: <(1, 0), -1> (d = -1)

Before learning:

$O = 1$ if $x_2 \geq -x_1 + 0.5$





- Before learning: $\boxed{O = 1 \quad \text{if} \quad 1 \cdot x_1 + 1 \cdot x_2 - 0.5 \geq 0}$

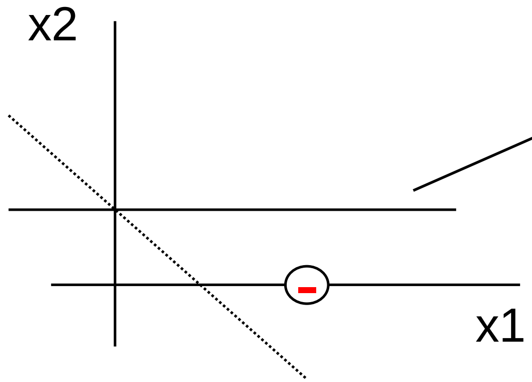
- O for (1, 0): $1 \cdot 1 + 1 \cdot 0 - 0.5 \geq 0$ ($O = 1$) ⇒ Error !

Learning of Perceptron

- $\Delta w_1 = 0.5 \cdot (-1 - 1) \cdot 1 = -1 \rightarrow w_1' = w_1 + \Delta w_1 = 1 - 1 = 0$
  $\Delta w_2 = 0.5 \cdot (-1 - 1) \cdot 0 = 0 \rightarrow w_2' = w_2 + \Delta w_2 = 1 - 0 = 1$

- After learning: $\boxed{O = 1 \text{ if } 0 \cdot x_1 + 1 \cdot x_2 - 0.5 \geq 0}$

- O for (1, 0): $0 \cdot 1 + 1 \cdot 0 - 0.5 \leq 0$ ($O = -1$)

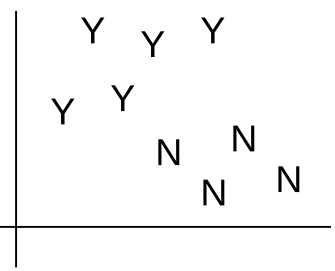


After learning:

$O = 1$ if $x_2 \geq 0.5$

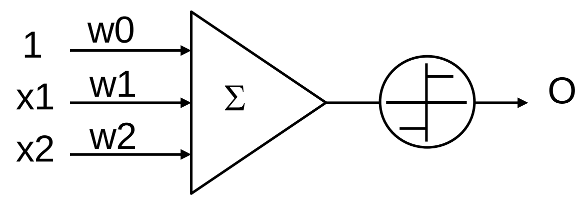
Learning of Perceptron

X		O
1.0	5.2	Y(1)
9.4	1.4	N(-1)
...	...	...

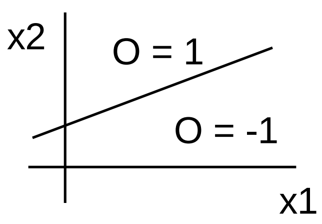


Find
(adjust weights)

X →



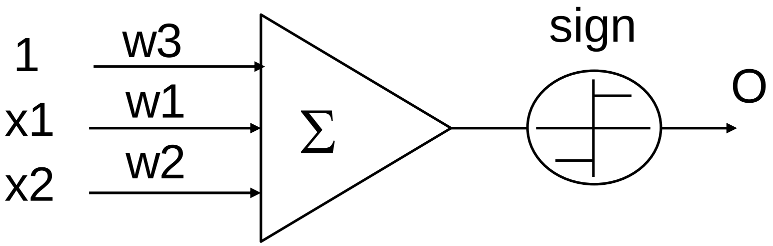
$O = 1$ if $x_1 w_1 + x_2 w_2 + w_0 \geq 0$



→ O
(f(X))

Example

X1	X2	O
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
…	…	…



- Perceptron
  - O = sign (w1·x1 + w2·x2 + w3·1) = sign(W·X)
    (W = (w1, w2, w3), X = (x1, x2, 1))
- Learning
  - W’ = W + 0.2 · (d – sign(W·X)) · X

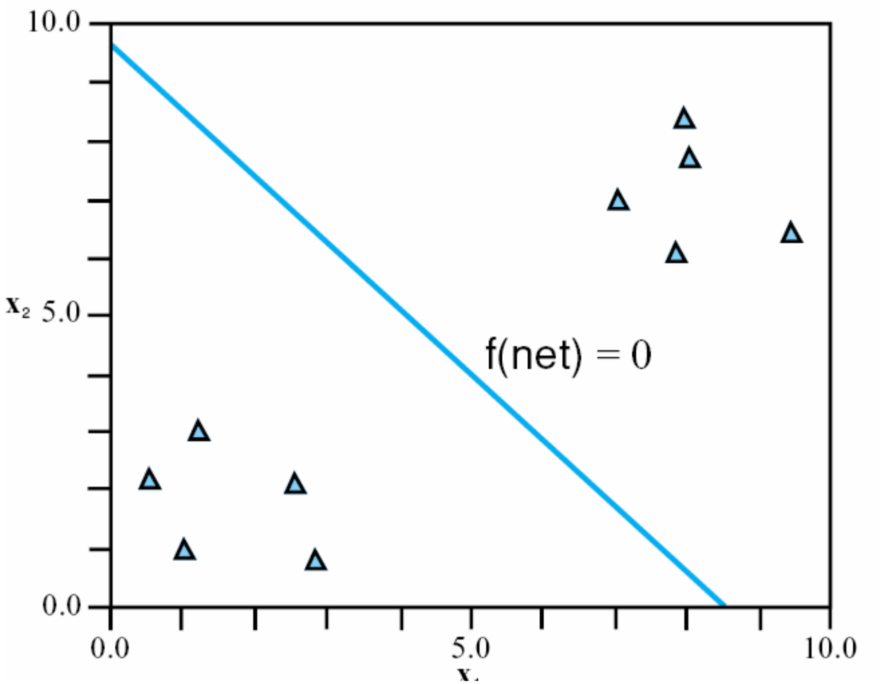
Example

- Random initial W = (0.75, 0.5, -0.6)
- W = (0.75, 0.5, -0.6), Example 1: X = (1.0, 1.0, 1) → d = 1
  - O = sign(W · X) = sign(0.75*1.0 + 0.5*1.0 – 0.6*1.0) = 1
- W = (0.75, 0.5, -0.6), Example 2: X = (9.4, 6.4, -1) → d = -1
  - O = sign(W · X) = sign(0.75*9.4 + 0.5*6.4 – 0.6*1.0) = 1
  - W' = (0.75, 0.5, -0.6) + 0.2*(-1-1)*(9.4, 6.4, 1) = (-3.01, -2.06, -1.0)
- W = (-3.01, -2.06, -1.0), Example 3: X = (2.5, 2.1, 1) → d = 1
  - O = sign(W · X) = sign(-3.01*2.5 – 2.06*2.1 – 1.0*1.0) = -1
  - W' = (-3.01, -2.06, -1.0) + 0.2*(1-(-1))*(2.5, 2.1, 1) = (-2.01, -1.22, -0.6)
- W = (-2.01, -1.22, -0.6), Example 4: …
  - …
- After 500 iteration, W = (-1.3, -1.1, 10.9)

Example

- $O = \text{sign}(-1.3 \cdot x1 - 1.1 \cdot x2 + 10.9)$

  $= 1 \text{ if } x2 < -1.18 \cdot x1 + 9.9$

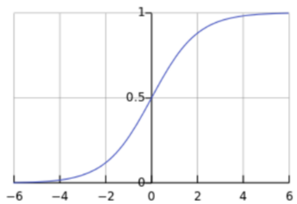


Multi-layer Neural Network

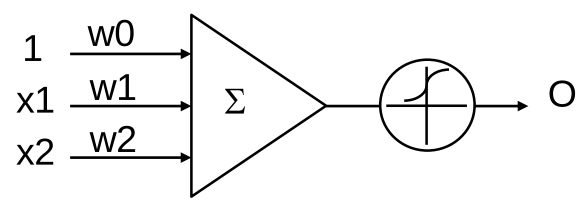
- Use sigmoid for output(activation) function

$$O = f(S) = \frac{1}{(1+e^{-S})}, \quad S = \sum x_i \cdot w_i$$

f(S)



S ($\Sigma\, x_i\, w_i$)

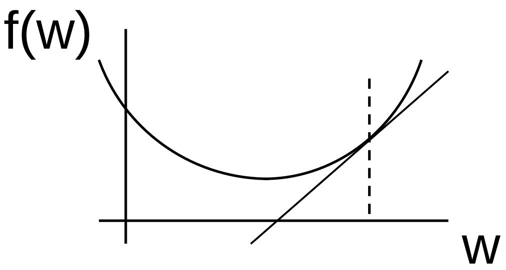


- Adjust weight
  - Error is a function of weights
  - E = (d – O) = (d - 1 / (1+e$^{-\Sigma\, xi\, wi}$)) = f (w_1, w_2, …)
  - Update W = (w_1, w_2, … w_n) to the direction which
    most rapidly reduce the error

Multi-layer Neural Network

- For $E = f(w)$
  - If $f'(w) = \frac{dE}{dw} > 0$ → decrease w
  - If $f'(w) = \frac{dE}{dw} < 0$ → increase w

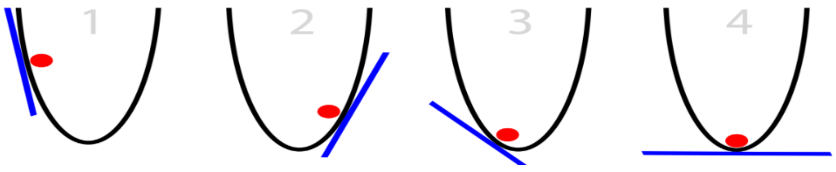
  $\therefore \quad \Delta w = -c \cdot \frac{dE}{dw}$



- For $E = f(W) = f(w_1, w_2, \dots, w_n)$

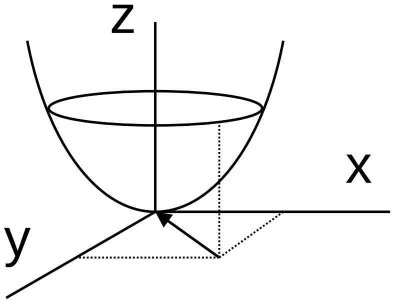
  $\therefore \quad \Delta W = -c \cdot \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_n} \right)$

Gradient descent:



Multi-layer Neural Network

- Example
  - $z = f(x, y) = x^2 + y^2$



  - $\Delta x = -\frac{\partial z}{\partial x} = -2x$, $\Delta y = -\frac{\partial z}{\partial y} = -2y$

    ∴ At (1, 1) the direction that most rapidly reduce z is (-2, -2)

Multi-layer Neural Network

- Minimize E

$$E = \frac{1}{2}(d - O)^2 \qquad O = \frac{1}{(1 + e^{-S})} \qquad S = \sum_i x_i w_i$$

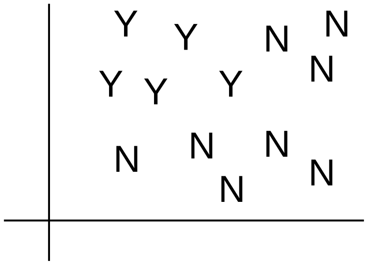
$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial O} \cdot \frac{\partial O}{\partial S} \cdot \frac{\partial S}{\partial w_i} = -(d - O) \cdot O(1 - O) \cdot x_i$$

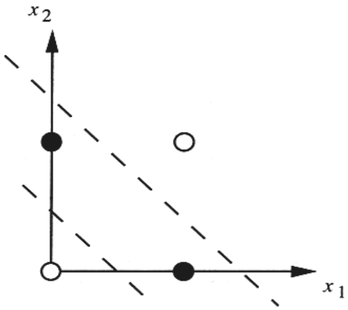
$$\left(\frac{\partial O}{\partial S} = \frac{-(-e^{-S})}{(1 + e^{-S})^2} = \frac{1 \cdot (1 + e^{-S} - 1)}{(1 + e^{-S}) \cdot (1 + e^{-S})} = O \cdot (1 - O) \right)$$

$$\boxed{\Delta w_i = -c \cdot \frac{\partial E}{\partial w_i} = c \cdot (d - O) \cdot O(1 - O) \cdot x_i}$$

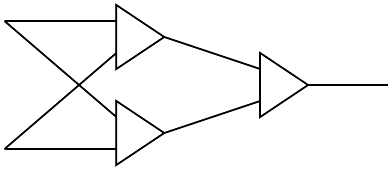
Multi-layer Neural Network

- Limitations of single-layer perceptron
  - Only applicable to linearly separable problems

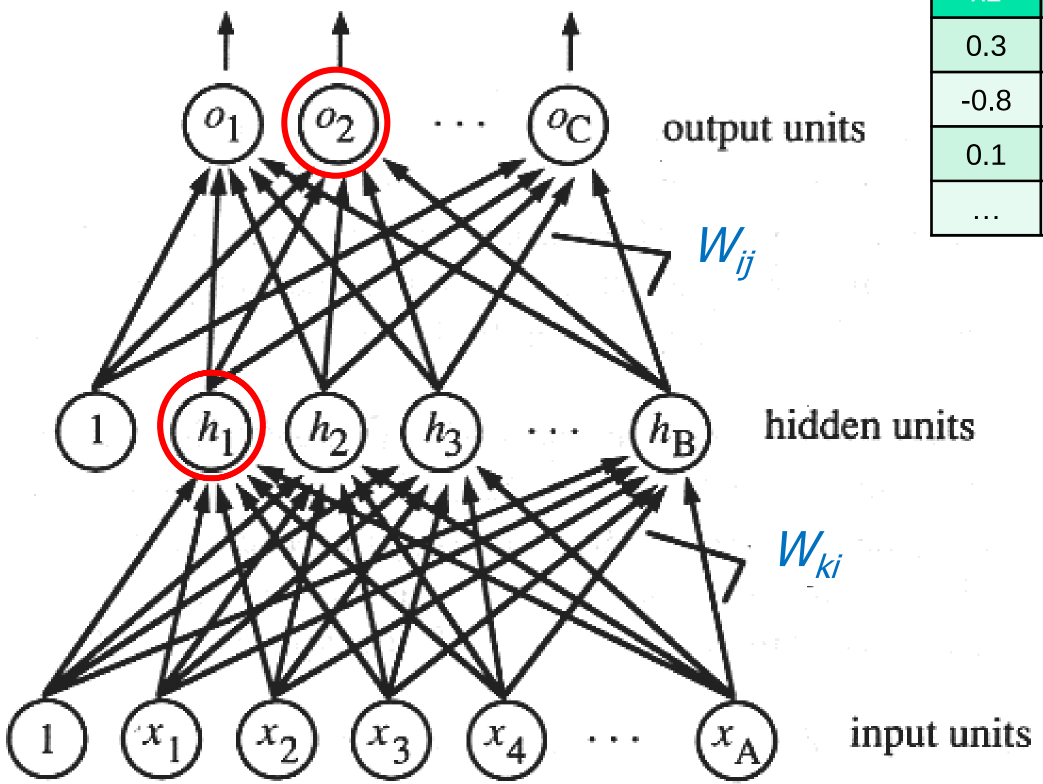




Use multi-layer network

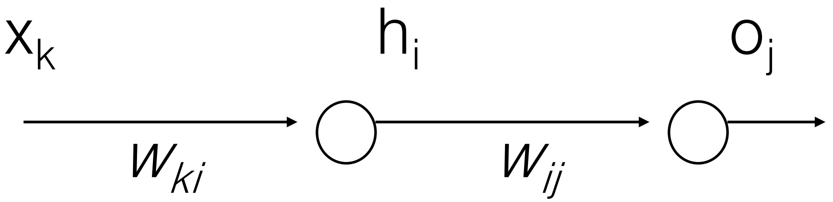


Multi-layer Neural Network



x1	x2	x3	d1	d2	d3
0.3	0.9	0.4	1	0	0
-0.8	-0.1	0.6	0	0	1
0.1	-0.5	-0.2	0	1	0
…	…	…	…	…	…

Multi-layer Neural Network



- Activation of hidden layer

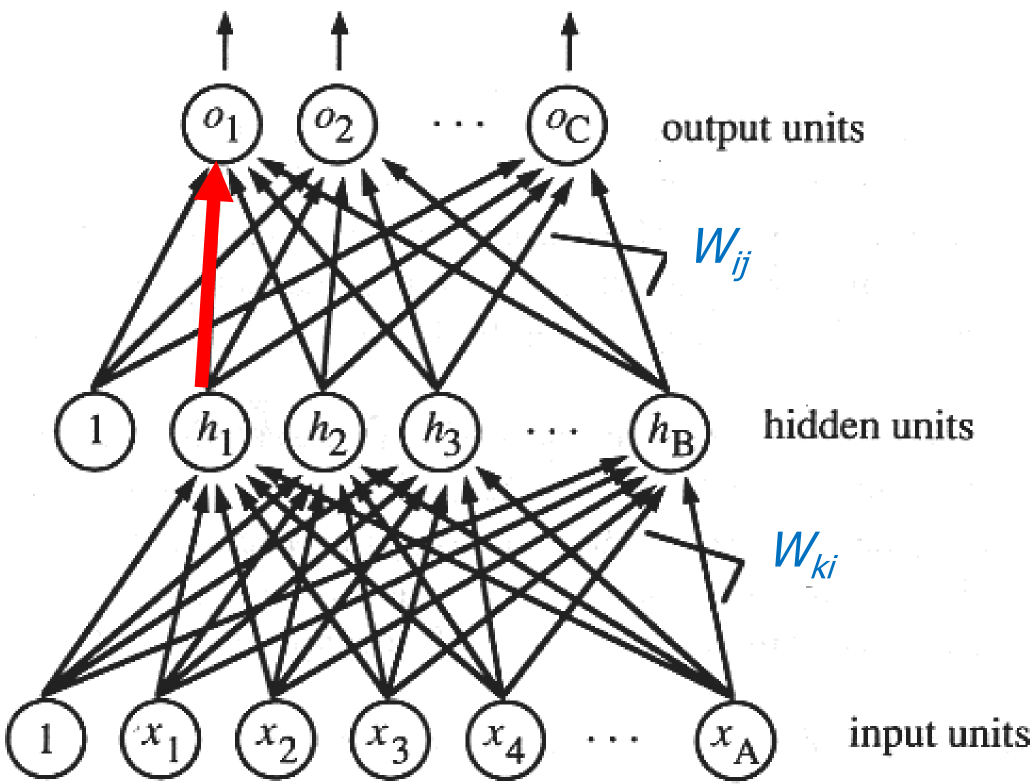
$$h_i = \frac{1}{(1+e^{-S_x})}, \quad S_x = \sum x_k \cdot w_{ki}$$

- Activation of output layer

$$o_j = \frac{1}{(1+e^{-S_h})}, \quad S_h = \sum h_i \cdot w_{ij}$$

Back Propagation Learning

$E = E_1 = ½(d_1 - O_1)^2$



Back Propagation Learning

- Learning
  - Let $$E = \frac{1}{2}\sum_j (d_j - O_j)^2 \qquad E_j = \frac{1}{2}(d_j - O_j)^2$$
  - Adjusting weights to output layers

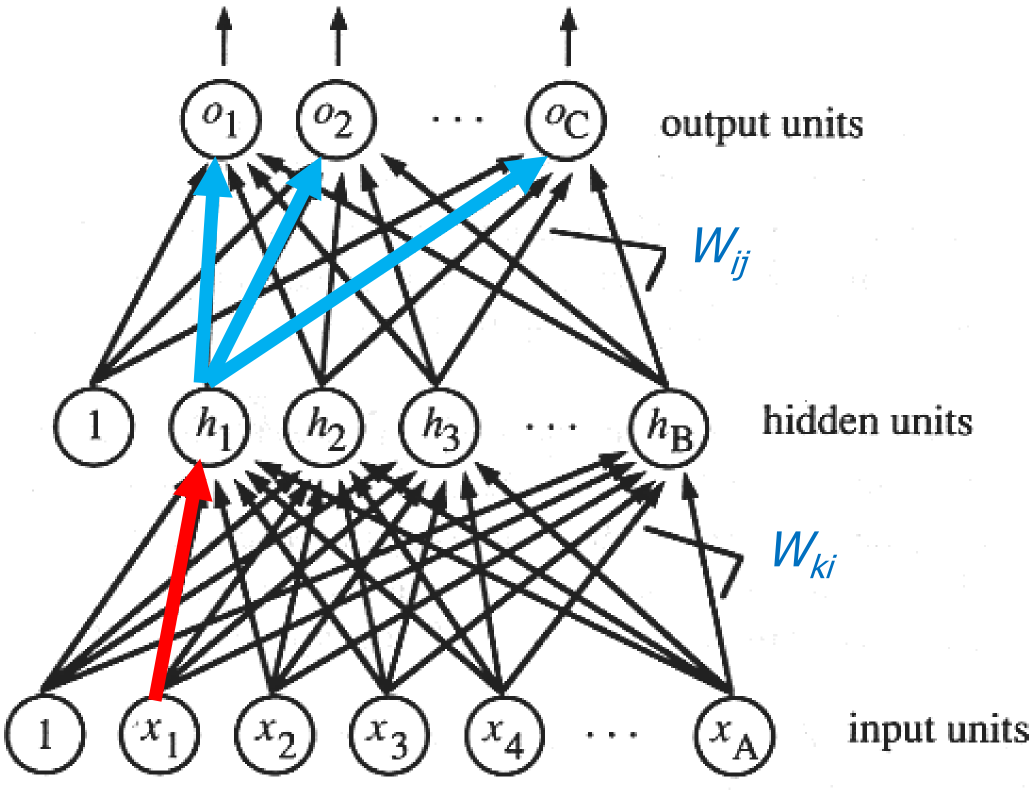
$$\Delta w_{ij} = -c \cdot \frac{\partial E}{\partial w_{ij}} = -c \cdot \frac{\partial E_j}{\partial w_{ij}}$$

$$= -c \cdot \frac{\partial E_j}{\partial O_j} \cdot \frac{\partial O_j}{\partial S_h} \cdot \frac{\partial S_h}{\partial w_{ij}}$$

$$\boxed{\Delta w_{ij} = c \cdot (d_j - O_j) \cdot O_j(1 - O_j) \cdot h_i}$$

Back Propagation Learning

$E = \sum E_j = ½\sum(d_j - O_j)^2$



Back Propagation Learning

- Adjusting weights to hidden layers

$$\Delta w_{ki} = -c \cdot \frac{\partial E}{\partial w_{ki}}$$

$$= -c \cdot \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial S_x} \cdot \frac{\partial S_x}{\partial w_{ki}} = -c \cdot \frac{\partial E}{\partial h_i} \cdot h_i(1-h_i) \cdot x_k$$

$$= -c \cdot \sum_j \frac{\partial E_j}{\partial h_i} \cdot h_i(1-h_i) \cdot x_k$$

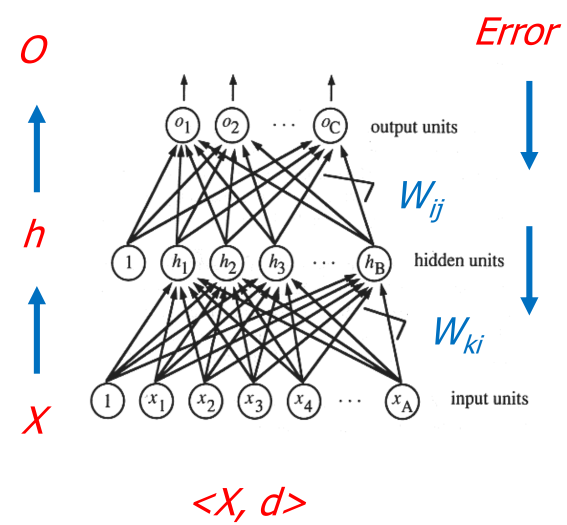
$$= -c \cdot \sum_j \left(\frac{\partial E_j}{\partial O_j} \cdot \frac{\partial O_j}{\partial S_h} \cdot \frac{\partial S_h}{\partial h_i}\right) \cdot h_i(1-h_i) \cdot x_k$$

$$\boxed{\Delta w_{ki} = c \cdot \left(\sum_j ((d_j - O_j) \cdot O_j(1-O_j) \cdot w_{ij}) \right) \cdot h_i(1-h_i) \cdot x_k}$$

Back Propagation Algorithm

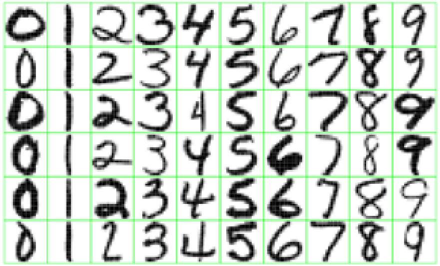
Define network
Initialize all weights
Until satisfied, Do
  For each training example <X, d>
    compute O = f(X)
    For each output unit j
      $\delta_j = (d_j - o_j)\, o_j\, (1 - o_j)$
    For each hidden unit i
      $\delta_i = (\Sigma\, w_{ij}\delta_j)\, h_i\, (1 - h_i)$
    Update each network weight
      $w_{ij} = w_{ij} + c\, \delta_j\, h_i$
      $w_{ki} = w_{ki} + c\, \delta_i\, x_k$

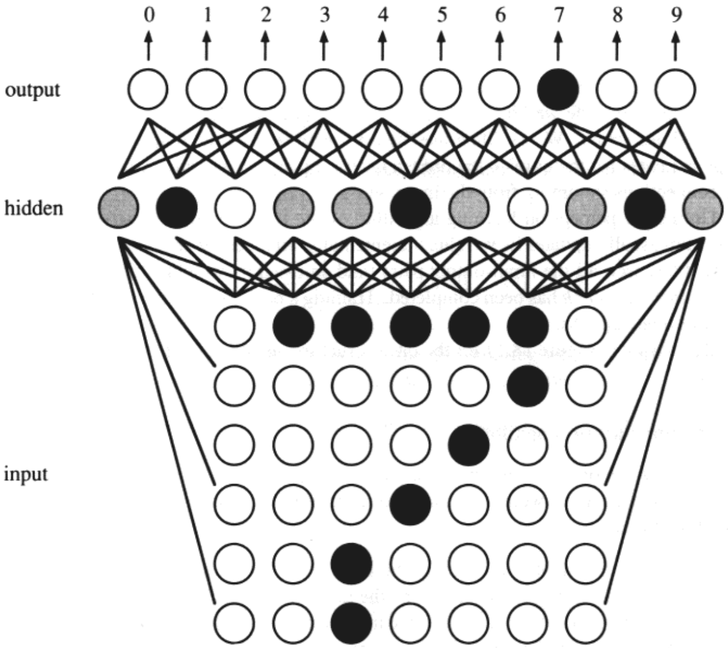
x1	x2	x3	d1	d2	d3
0.3	0.9	0.4	1	0	0
-0.8	-0.1	0.6	0	0	1
0.1	-0.5	-0.2	0	1	0
...	...	...	...	...	...



Applying Neural Network

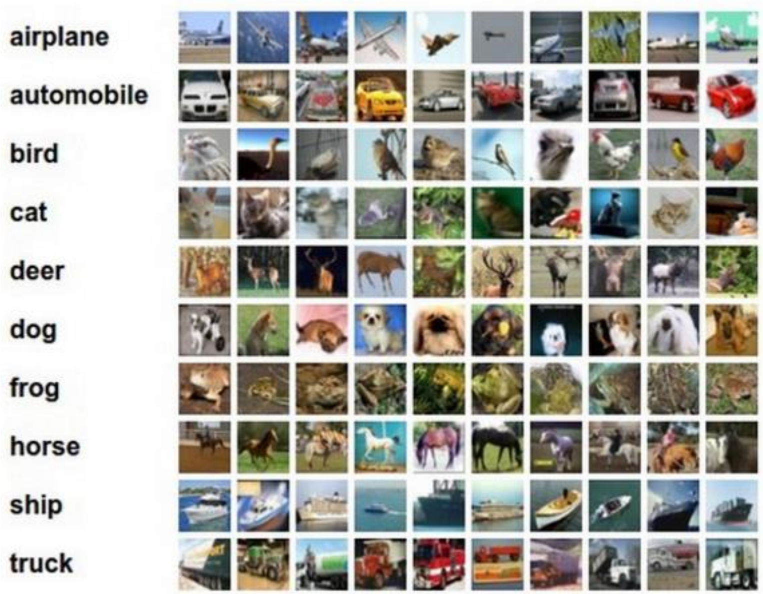
- Hand-written Character recognition
  - Input: 2 dimensional image array
  - Output: classify into a character

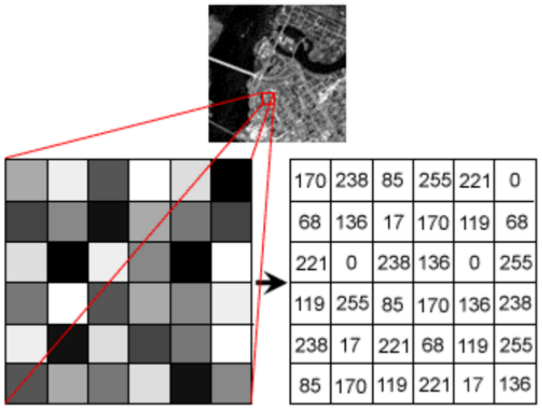




Applying Neural Network

- Image recognition





→ airplane/bird/cat/dog … ?