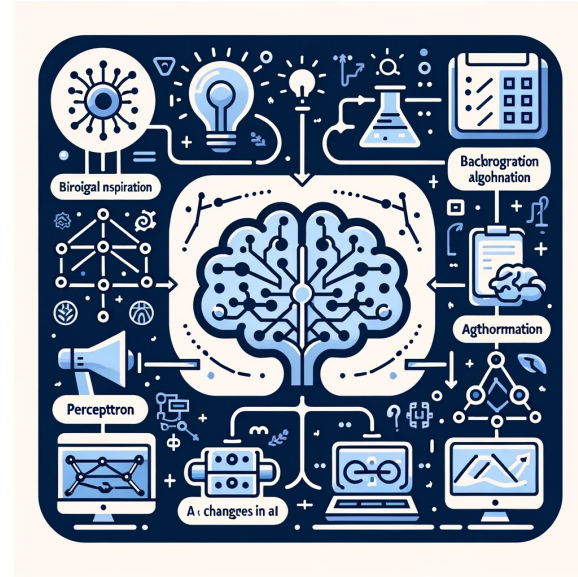


Artificial Neural Network

Johan Hjorth

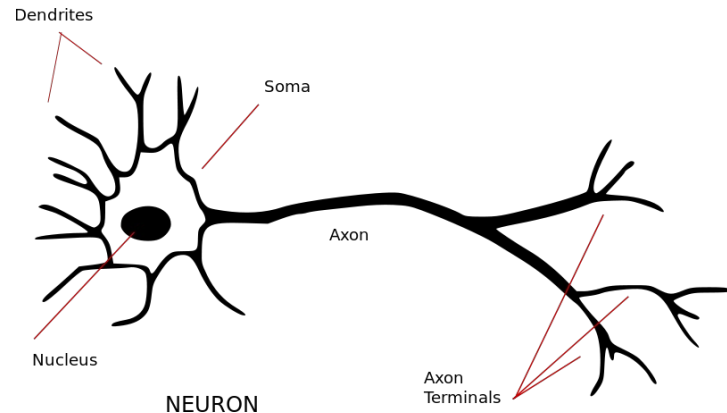
Outline

- Biological Inspiration
- Perceptron
- Multi-layer networks
- Backpropagation algorithm (gradient descent)



Biological Neuron

- A neuron has:
 - Branching **inputs** (dendrites)
 - A branching **output** (the axon)
- The information circulates from the dendrites to the axon via the cell body
- Axon connects to dendrites via synapses (**weights**)
 - Synapses vary in strength
 - Synapses may be excitatory or inhibitory



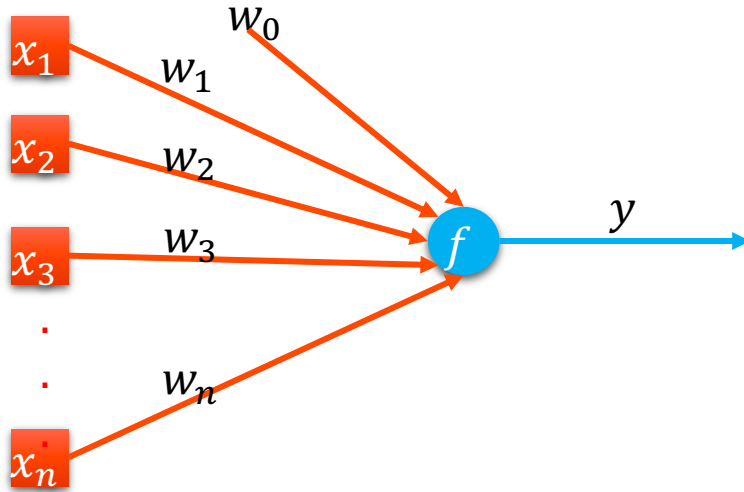
This Photo by Unknown Author is licensed under [CC BY-SA](#)

Biological inspiration

- Some numbers...
 - The human brain contains about 10 billion nerve cells (neurons)
 - Each neuron is connected to other ($10^4 - 10^5$) neurons through synapses
 - Neuron switching time : 10^{-3} secs (much slower than computer switching within 10^{-10} secs)
- Properties of the brain
 - It can **learn** from experience
 - It **adapts** to the environment
 - It has a **high degree of parallel information** processing
 - It is **robust** and z

Artificial neuron

- Non linear function with restricted ourput range:
 - **Inputs:** Analogy to dendrites.
 - **Output:** Analogy to axon.
 - **Weights** associated with inputs: analogy to synapses



Linear Regression (with multiple variables)

$$y = f\left(w_0 + \sum_{i=1}^n w_i x_i\right)$$

Linear regression with multiple variables

We don't β use we use weights!

- The hypothesis is:

$$h_{\beta}(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

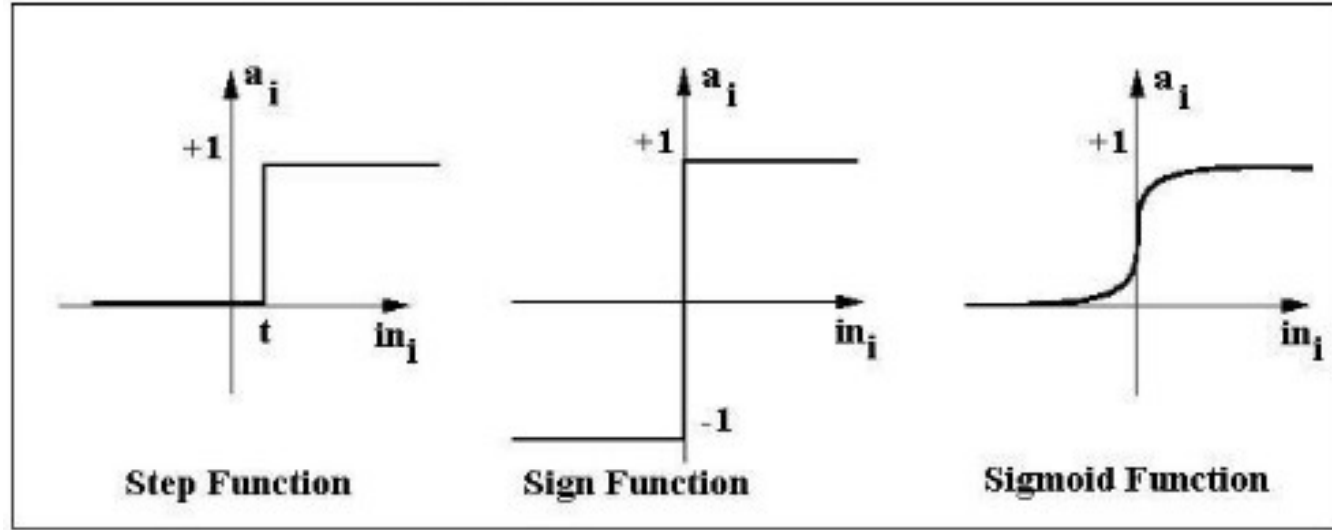
For convenience, we will define $x_0 = 1$. The hypothesis is equal to

$$h_{\beta}(x) = \beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n \text{ where } \mathbf{X} = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \text{ and } \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

In Matlab, python or octave you can do: $\mathbf{h}_{\beta}(x) = \boldsymbol{\beta}^T \mathbf{X}$ where

$$\boldsymbol{\beta}^T \begin{bmatrix} \beta_0 & \beta_1 & \dots & \beta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \mathbf{X}$$

Activation Functions

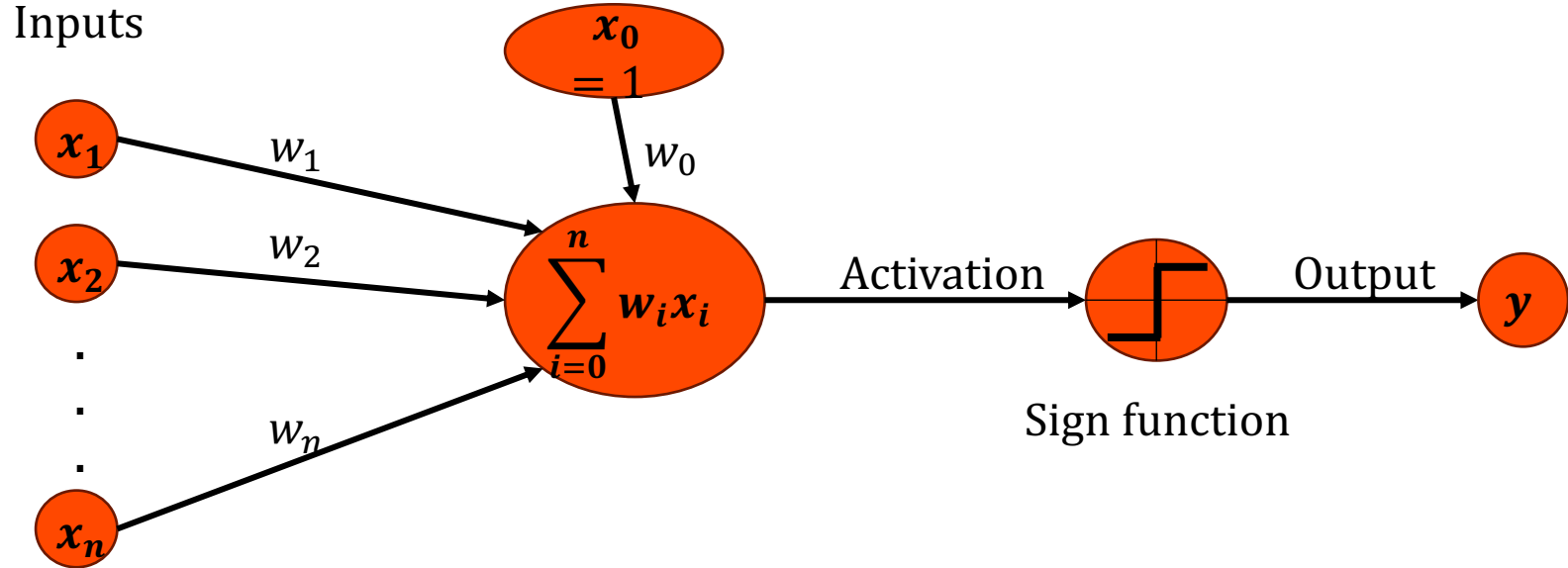


$$step(x) = \begin{cases} 1, & \text{if } x \geq t \\ 0, & \text{if } x < t \end{cases}$$

$$sign(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$$

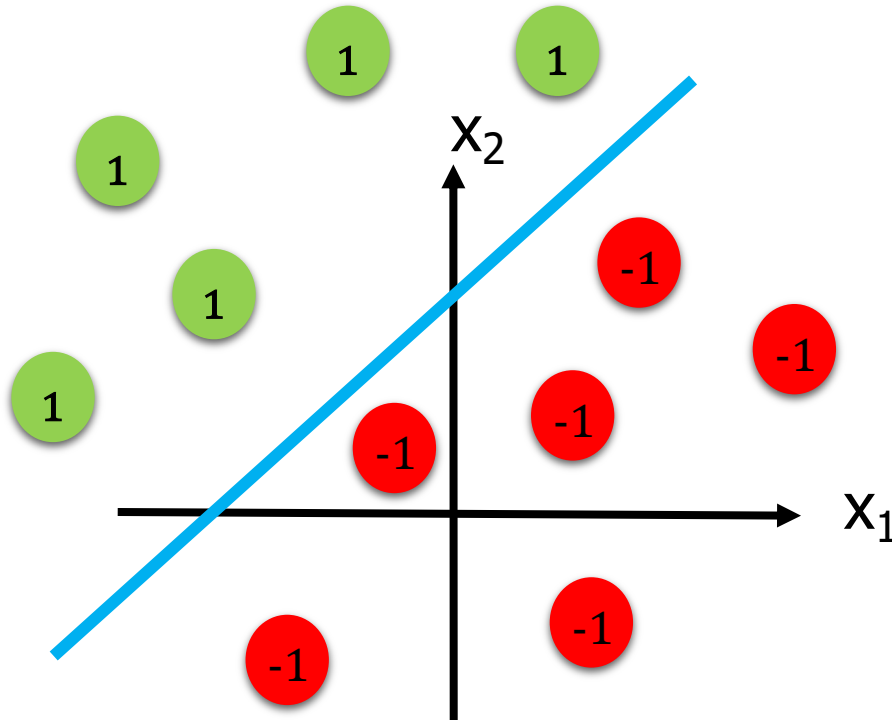
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Perceptron



$$y = \text{sign}(x_0, x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 x_0 + w_1 x_1 + \dots + w_n x_n \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Decision surface of perceptron

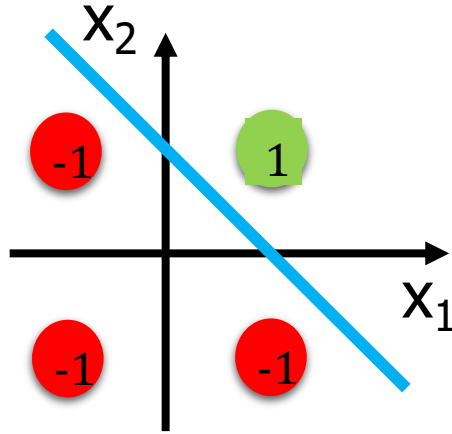


Decision line

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

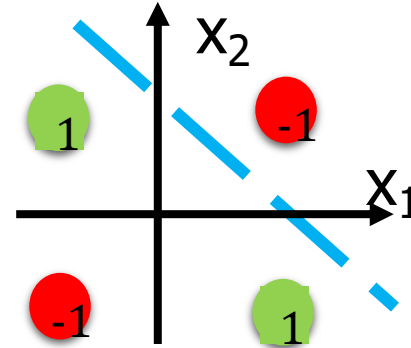
Perceptron represents a hyperplane decision surface in the n-dimensional space of instances

Linear Separability



Logical AND

x_1	x_2	net	y
-1	-1	-1.5	-1
-1	1	-0.5	-1
1	-1	-0.5	-1
1	1	0.5	1



Logical XOR

x_1	x_2	y	
-1	-1	-1	
-1	1	1	
1	-1	1	
1	1	-1	

Perceptron learning rule

Given a training example, revise weights using the following rule:

$$\Delta w_i = \eta(t-o)x_i, \quad \Delta w_0 = \eta(t-o) \bullet 1$$

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

where

- t is the target value specified by the training example
- o is the perceptron output
- x_i is the attribute value in the example
- The parameter η is called the *learning rate*.

- If the output is correct ($t=o$) the weights are not changed ($\Delta w_i=0$).
- If the output is incorrect ($t \neq o$), the weights w_i are changed such that the activation for perceptron is increased or decreased

Perceptron training algorithm

Repeat

For each training example (x, t)

compute the output o using x as the input

if $o \neq t$ then

form a new weight vector w' according to

$$w' = w + \eta (t - o) x$$

else

do nothing

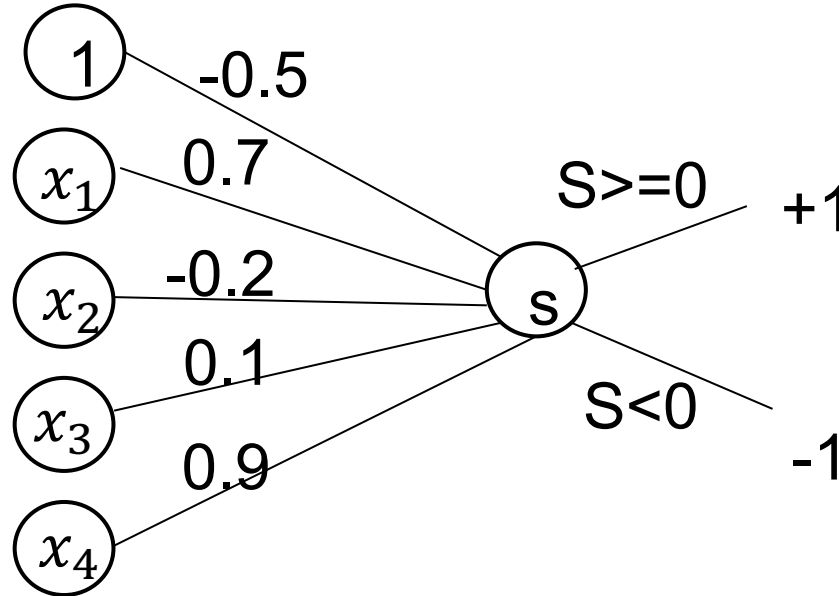
end if

End for (epoch)

Until $o = t$ for all training examples (**Be careful with this**)

Example

- Suppose we have a set of random weights for the perceptron
- Use a learning rate of $\eta = 0.1$



Example

- Suppose a training example, E , as
 - $x_1 = -1$
 - $x_2 = 1$
 - $x_3 = 1$
 - $x_4 = -1$
 - and $t(E) = 1$
- Propagate this information through the network:
 - $S = (-0.5*1) + (0.7*(-1)) + (-0.2*1) + (0.1*1) + (0.9*(-1)) = -2.2$
- Hence the perceptron output $o(E) = -1$
- Next, we use the perceptron rule to update the weights.

Example – Calculating the changes

- $\Delta w_0 = \eta(t(E) - o(E))x_0$
 $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$
- $\Delta w_1 = \eta(t(E) - o(E))x_1$
 $= 0.1 * (1 - (-1)) * (-1) = 0.1 * (-2) = -0.2$
- $\Delta w_2 = \eta(t(E) - o(E))x_2$
 $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$
- $\Delta w_3 = \eta(t(E) - o(E))x_3$
 $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$
- $\Delta w_4 = \eta(t(E) - o(E))x_4$
 $= 0.1 * (1 - (-1)) * (-1) = 0.1 * (-2) = -0.2$

Example – Calculating new weights

Initial

Changes from previous slide

• $w'_0 = -0.5 + \Delta w_0 = -0.5 + 0.2 = -0.3$

• $w'_1 = 0.7 + \Delta w_1 = 0.7 - 0.2 = 0.5$

• $w'_2 = -0.2 + \Delta w_2 = -0.2 + 0.2 = 0$

• $w'_3 = 0.1 + \Delta w_3 = 0.1 + 0.2 = 0.3$

• $w'_4 = 0.9 + \Delta w_4 = 0.9 - 0.2 = 0.7$

Example

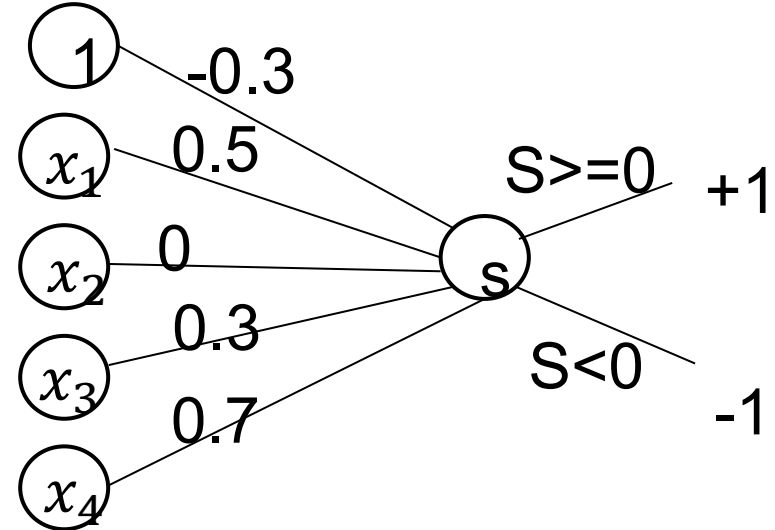
- Calculate for the example, E, again:

- $S = (-0.3 * 1) + (0.5 * (-1)) + (0 * 1) + (0.3 * 1) + (0.7 * (-1)) = -1.2$

- Still gets the wrong categorisation

- But the value is closer to zero (from -2.2 to -1.2)

- In a few epochs time, this example will be correctly classified



Exercise

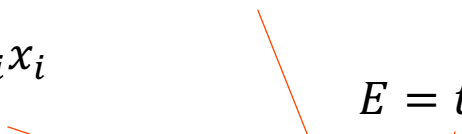
- Solve OR function with the table.
- t: Target value
- net: Value before the activation function.
- o: Value after the activation function
- Activation Function:

$$\text{step}(x) = \begin{cases} 1, & \text{if } x \geq U \\ 0, & \text{if } x < U \end{cases}$$

- $U = 2$

$$\text{net} = \sum_{i=1}^n w_i x_i$$

$o = \text{step}(\text{net})$

$$E = t - o$$


W1	W2	X1	X2	t	net	o	E
0	0	0	0	0			
		1	0	1			
		0	1	1			
		1	1	1			
		0	0	0			
				

Perceptron convergence theorem

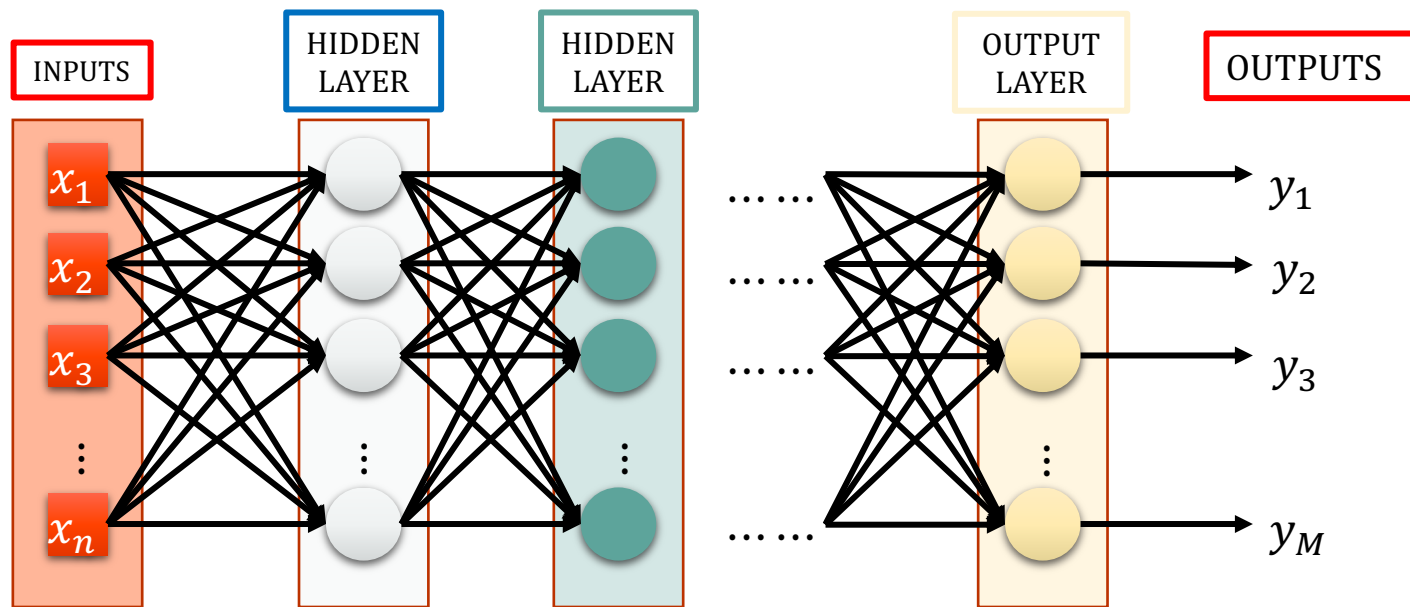
- The perceptron learning procedure can converge within a finite number of applications of the perceptron rule to a weight vector that correctly classified all training examples, provided
 - The training examples are linearly separable (i.e. Such separating linear surfaces exist)
 - A sufficiently small learning rate is used

Artificial neural nets (ANNs)

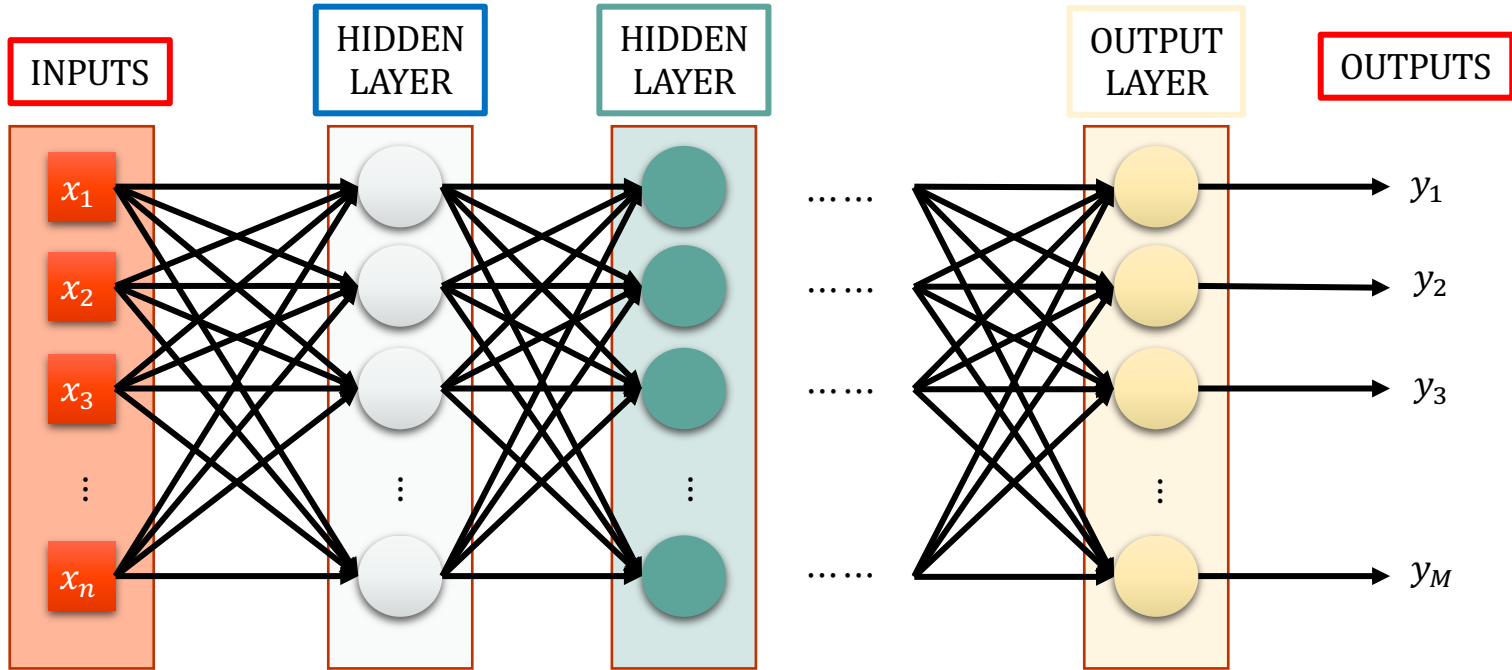
- Inspired by biological systems consisting of interconnected neurons, artificial neural networks (ANNs) are built out of interconnected artificial units, each of which taking a number of real-valued inputs and producing a real valued output.
 - Many simple neuron-like switching units
 - Many weighted interconnections among units
 - Learning by adaptation of the connection weights
 - gross simplification of real neural networks
 - *Human brains: 100,000,000,000 neurons*
 - *ANNs: < 1000 usually*
 - Implement very complex input-output function that approximate any desired function with arbitrary accuracy.

Multilayer network of units

To represent more complex nonlinear decision surfaces, we need a network to combine many units.



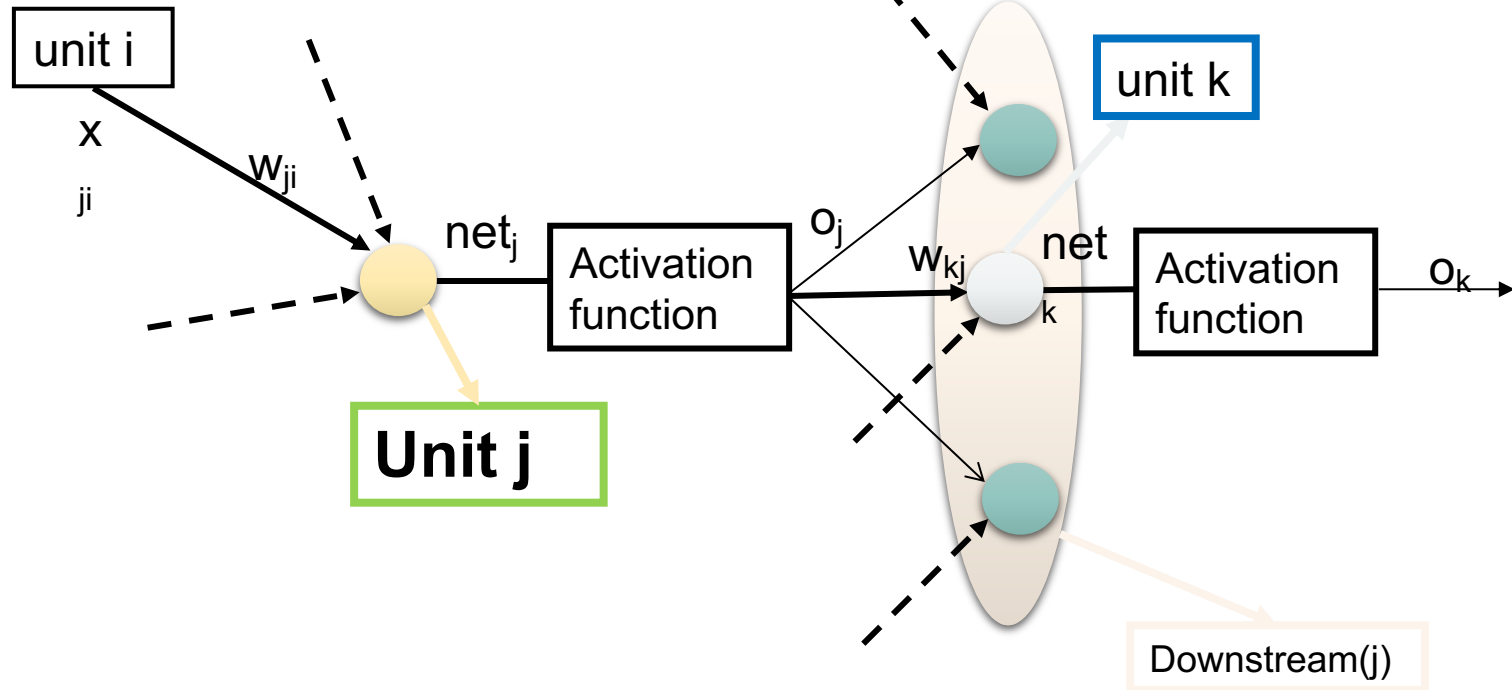
BACKPROPAGATION



Forward: Propagate information from input to output layer

Backward: Propagate error from output to hidden layer

Multilayer network notation



Multilayer network notation (2)

o_j : the output computed by unit j

t_j : the target output for unit j (only for output layer)

net_j : the weighted sum of inputs at unit j

x_{ji} : the input from unit i to unit j , w_{ji} denotes the corresponding weight

$\text{Downstream}(j)$: the set of units whose immediate inputs include the output of unit j .

outputs: the set of units in the final layer of the network

- The cost function in Linear regression is equal to

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^m (h_{\beta}(x^{(i)}) - y^{(i)})^2$$

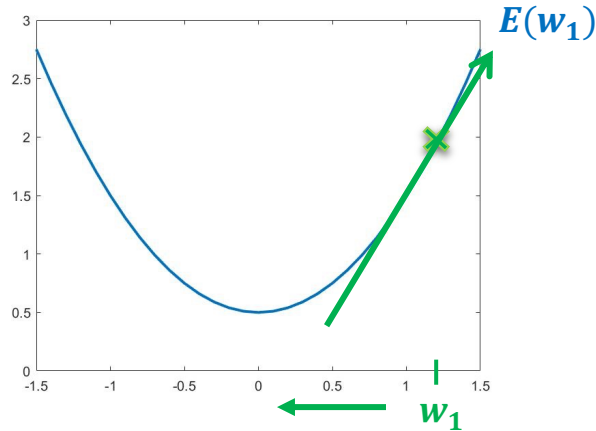
where β are the parameters, $y^{(i)}$ is the target and $h_{\beta}(x^{(i)})$ is the prediction.

- In Neural Network, we will define the error (similar to cost) of an example d as

$$E_d(\vec{w}) = \sum_{k \in \text{outputs}} (t_k - o_k)^2$$

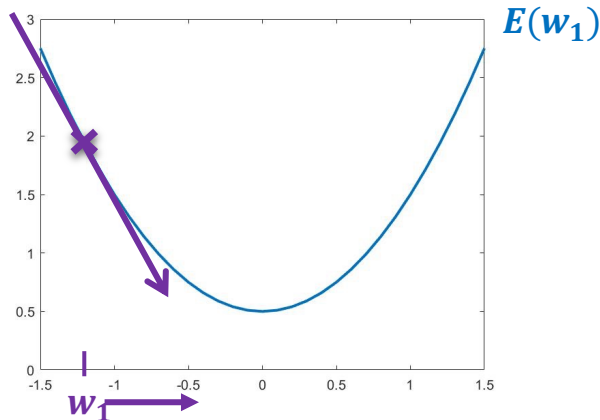
where t_k is the target of the k -th class and o_k is the output value of the k -th output neuron.

Gradient descent intuition



$$w_1 \leftarrow w_1 - \eta \frac{\partial}{\partial w_1} E(w_1) \geq 0$$

$w_1 \leftarrow w_1 - \eta(\text{positive number})$



$$w_1 \leftarrow w_1 - \eta \frac{\partial}{\partial w_1} E(w_1) \leq 0$$

$w_1 \leftarrow w_1 - \eta(\text{negative number})$

Gradient descent

- We incrementally update the weights in terms of individual instances.
- The idea is to modify the weights according to the negative of gradient of the error function.

$$\Delta w_{i,j} = -\eta \frac{\partial E_d}{\partial w_{i,j}}$$

Learning rate

- Then, the weights are updated as

$$w_{ij} = w_{i,j} + \Delta w_{i,j}$$

How much the
weights change

Updating the weights

- The update for the weight can be rewritten as $\Delta w_{ji} = \eta \left(-\frac{\partial E_d}{\partial net_j} \right) x_{ji}$
- Let $\delta_j = -\frac{\partial E_d}{\partial net_j}$ be the error term of unit j
- Then, the weight update rule is formulated as $\Delta w_{j,i} = \eta \delta_j x_{j,i}$
- δ_j is calculated differently depending if j belongs to an output layer or to a hidden layer.

Error term

- The error term at the output layer is calculated as follow

$$\delta_k = (t_k - o_k) * f'(o_k)$$

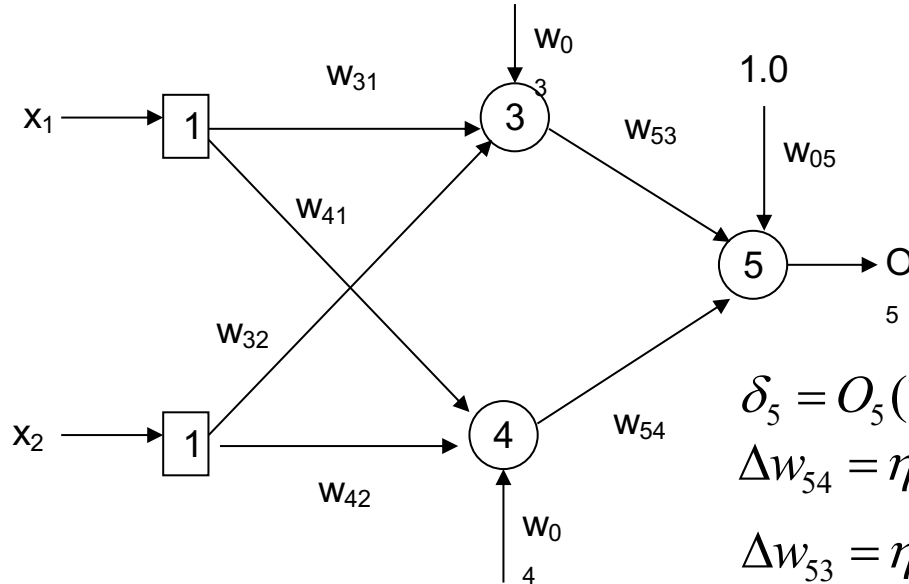
where k stands for one of the output neurons and f' will be dependant on the activation function used.

- The error term in the hidden layer is calculated as follow

$$\delta_j = f'(o_j) * \sum_{k \in \text{downstream}(j)} \delta_k w_{kj}$$

where k stands for the neurons of the next layer.

Example of Backpropagation



$$\delta_5 = O_5(1 - O_5)(t_5 - O_5) = -0.1274$$

$$\Delta w_{54} = \eta \cdot \delta_5 \cdot O_4 = 0.1 \cdot (-0.1274) \cdot 0.8808 = -0.0112$$

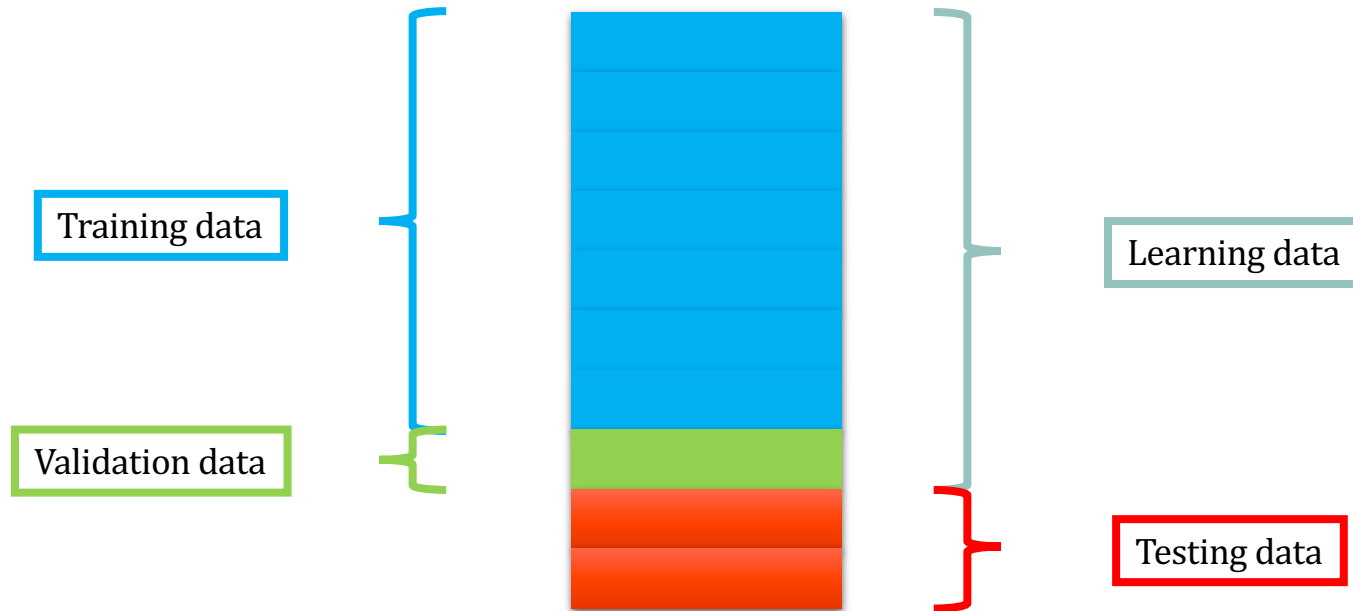
$$\Delta w_{53} = \eta \cdot \delta_5 \cdot O_3 = 0.1 \cdot (-0.1274) \cdot 0.5250 = -0.0067$$

$$\Delta w_{05} = \eta \cdot \delta_5 \cdot 1 = 0.1 \cdot (-0.1274) \cdot 1 = 0.0127$$

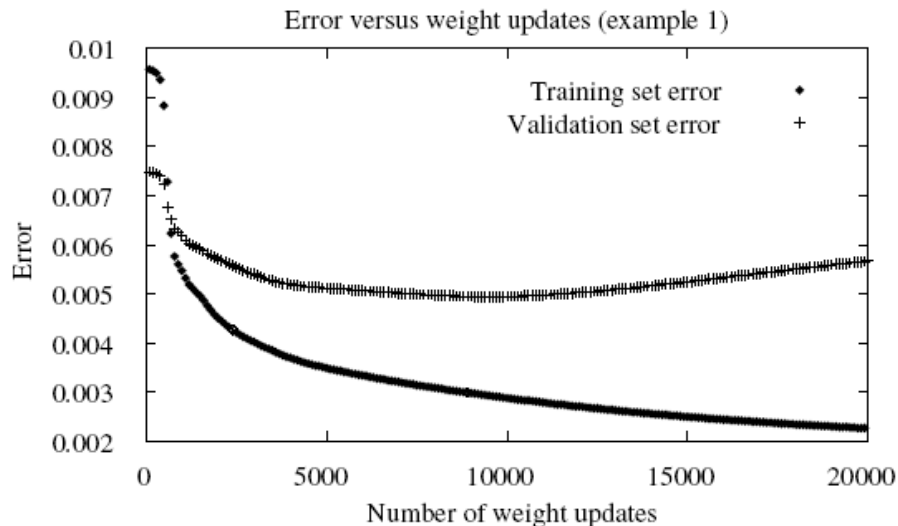
Backpropagation Algorithm

- Initialize weights w_{ij} with a small random values
 - Repeat
 - For each training example $\{(x_1, \dots, x_n)^p, (t_1, \dots, t_m)^p\}$ do
 1. *Present $(x_1, \dots, x_n)^p$ to the network and compute the outputs (forward step)*
 2. *Compute the error terms in the output layer*
 3. *Compute the error terms in the hidden layer (backwards)*
 4. *Update the weights for all units using error terms*
- until overall error E becomes acceptably low

Division of the data



Overfitting the Training Data



Learning coincidence in training data, not representative

As learning proceeds (weights increased), decision surface progressively nonlinear and complex, more chances to learn the coincidence, more risks of overfitting.

Ways to Avoid Overfitting

1. **Weight decay:** decrease each weight by a small factor during each iteration. The aim is to keep weights small to bias learning against too complex decision surfaces.
2. **Separate validation data:** monitor learning on the validation set, terminate the search when accuracy on validation data gets worse
3. **K-fold-cross-validation:** divide the training data into k parts and perform k -cross-validation. Estimate the number of iterations h_i for best performance on validation set in each of the k trials. The mean h' of these estimates for h_i is then calculated, and final BACKPROPAGATION algorithm is performed on the whole training data for h' iterations without validation set.

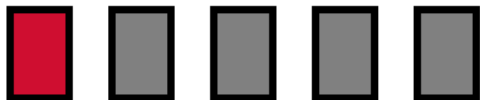
K-Fold-Cross Validation

Training data



Validation

Learning data



Best number of iterations: h_1

Learning data

Validation



Best number of iterations: h_5

Best number of iterations: $h' = (h_1 + h_2 + h_3 + h_4 + h_5)/5$ for the whole training data

Some videos

- SNAKE: https://www.youtube.com/watch?v=l2_CPB0uBkc
- MARIO BROSS:
<https://www.youtube.com/watch?v=05rEefXlmhI>
- FISHES:
<https://www.youtube.com/watch?v=YFZwgxvRbNM>

Extra reading

- Norvig, Peter, and Russell, Stuart Jonathan, *Artificial intelligence: A modern approach*, Pearson Education, 2010 - ISBN: 9780132071482
 - 18 Learning from Examples----- Pag. 693
 - 18.6 Regression and Classification with Linear Methods ----- Pag. 717
 - 18.7 Artificial Neural Networks ----- Pag. 727
- You can find many videos and tutorials on internet.