Other Classification Methods

- **Instance-Based Learning (kNN)**
- **Artificial Neural Networks**

Instance-Based Learning (kNN)

Instance-Based Learning

- Instance-based learning methods simply store the training examples instead of learning explicit description of the target function.
  - Generalizing the examples is postponed until a new instance must be classified.
  - When a new instance is encountered, its relationship to the stored examples is examined in order to assign a target function value for the new instance.
- One of instance-based learning methods is ***k-nearest neighbor*** method.
- Instance-based methods are referred to as **lazy learning methods** because they delay processing until a new instance must be classified.
  - **Eager methods** (decision tree, neural networks, …) generalize the training set to learn a function.
- A key advantage of lazy learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.

k-Nearest Neighbor Learning (Classification)

- **k-Nearest Neighbor Learning** algorithm can be used in the prediction of values of continuous valued functions in addition to the prediction of class values of discrete-valued functions (**classification**)
- **k-Nearest Neighbor Learning** algorithm assumes all instances correspond to points in the n-dimensional space $\Re^n$
- The nearest neighbors of an instance are defined in terms of Euclidean distance.
- Euclidean distance between the instances $x_i = \langle x_{i1},\ldots,x_{in}\rangle$ and $x_j = \langle x_{j1},\ldots,x_{jn}\rangle$ are:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (x_{ir} - x_{jr})^2}$$

- For a given instance x_q, Class(x_q) is computed using the class values of k-nearest neighbors of x_q

k-Nearest Neighbor Classification

- Store all training examples <x_i,Class(x_i)>
- Calculate Class(x_q) for a given instance x_q using its k-nearest neighbors.
- **Nearest neighbor: (k=1)**
  - Locate the nearest traing example x_n, and estimate Class(x_q) as Class(x_n).
- **k-Nearest neighbor:**
  - Locate k nearest traing examples, and estimate Class(x_q) using **majority vote** among class values of k nearest neighbors.
- The test example is classified based on the majority class of its nearest neighbors:

$$\text{Majority Voting: } y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$$

where v is a class label, y_i is the class label for one of the nearest neighbors, and I(.) is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

k-Nearest Neighbor Classification - Example

A	B	Class
1	1	no
2	1	no
3	2	yes
7	7	yes
8	8	yes

3-Nearest Neighbor Classification of instance <3,3>

A	B	Distance of <3,3>
1	1	$\sqrt{8}$
2	1	$\sqrt{5}$
3	2	$\sqrt{1}$
7	7	$\sqrt{32}$
8	8	$\sqrt{50}$

- First three example are 3 Nearest Neighbors of instance <3,3>.
- Two of them is **no** and one of them is **yes**.
- Majority of classes of its neighbors are **no**, the classification of instance <3,3> is **no**.

Distance Weighted kNN Classification

- In the majority voting approach, every neighbor has the same impact on the classification
- We can weight the influence of each nearest neighbor x_i according to its distance to instance x_q.

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

- Using the distance-weighted voting scheme, the class label can be determined:

$$\text{Distance-Weighted Voting: } y' = \underset{v}{\text{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} w_i \times I(v = y_i)$$

Distance Weighted kNN Classification- Example

A	B	Class
1	1	no
2	1	no
3	2	yes
7	7	yes
8	8	yes

Distance Weighted 3-Nearest Neighbor Classification of instance <3,3>

A	B	Distance of <3,3>
1	1	$\sqrt{8}$
2	1	$\sqrt{5}$
3	2	$\sqrt{1}$
7	7	$\sqrt{32}$
8	8	$\sqrt{50}$

- First three example are 3 Nearest Neighbors of instance <3,3>.
- Weight of **no** = 1/8 + 1/5 = 13/40 Weight of **yes** = 1/1 = 1
- Since 1 > 13/40, the classification of instance <3,3> is **yes**.

k-Nearest Neighbor Algorithm

Consider the following set of training examples:

A	B	Target
1	9	yes
4	8	yes
4	7	yes
6	5	no
9	1	no

- Using 3-nearest neighbor algorithm, find the target classification for the instance <A=5,B=5>. Show your work.
- Using distance weighted 3-nearest neighbor algorithm, find the target classification for the instance <A=5,B=5>. Show your work.

k-Nearest Neighbor Algorithm

A	B	Target
1	9	yes
4	8	yes
4	7	yes
6	5	no
9	1	no

Distance of <5,5>

sqrt(32)

sqrt(10) * nearest

sqrt(5) * neighbors

sqrt(1) *

sqrt(32)

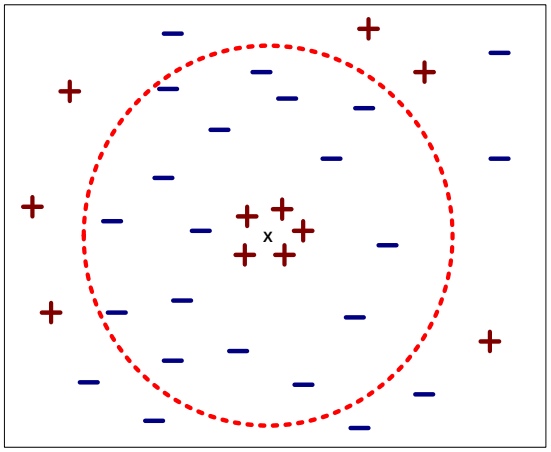
3-nearest neighbor: 2 yes, 1 no ➔ YES

Weighted 3-nearest neighbor:
Weight of yes: 1/10 + 1/5 = 3/10
Weight of no : 1/1 = 1

Since 1 > 3/10 ➔ NO

k-Nearest Neighbor Classification - Issues

- **Choosing the value of k:**
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes.



- **Scaling issues:**
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes

Artificial Neural Networks

Artificial Neural Networks

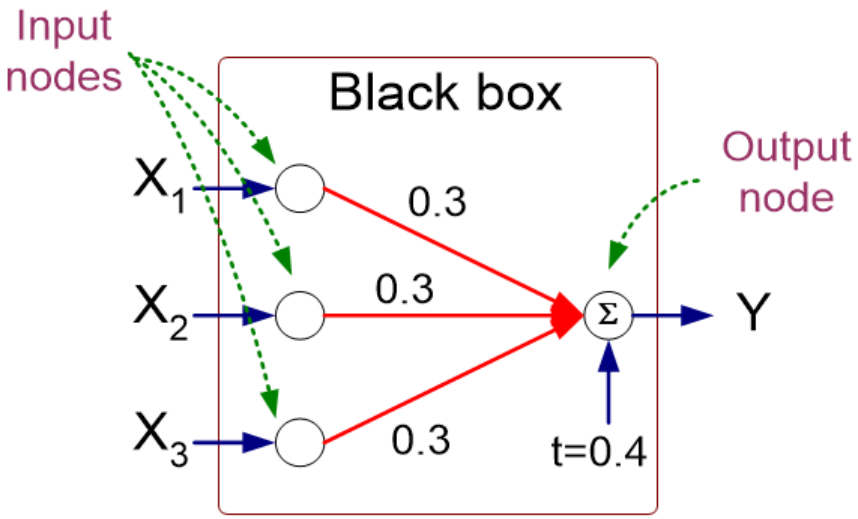
- Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples.
- Algorithms such as BACKPROPAGATION gradient descent to tune network parameters to best fit a training set of input-output pairs.
- The study of artificial neural networks (ANNs) has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons.
- Artificial neural networks are built out of a densely interconnected set of simple units, where each unit takes a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output (which may become the input to many other units).

Properties of Artificial Neural Networks

- A large number of very simple, neuron-like processing elements called **units**,
- A large number of weighted, directed connections between pairs of units
  - Weights may be positive or negative real values
- Local processing in that each unit computes a function based on the outputs of a limited number of other units in the network
- Each unit computes a simple function of its input values, which are the weighted outputs from other units.
  - If there are n inputs to a unit, then the unit's output, or activation is defined by a = g((w1 * x1) + (w2 * x2) + ... + (wn * xn)).
  - Each unit computes a (simple) function g of the linear combination of its inputs.
- Learning by tuning the connection weights

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

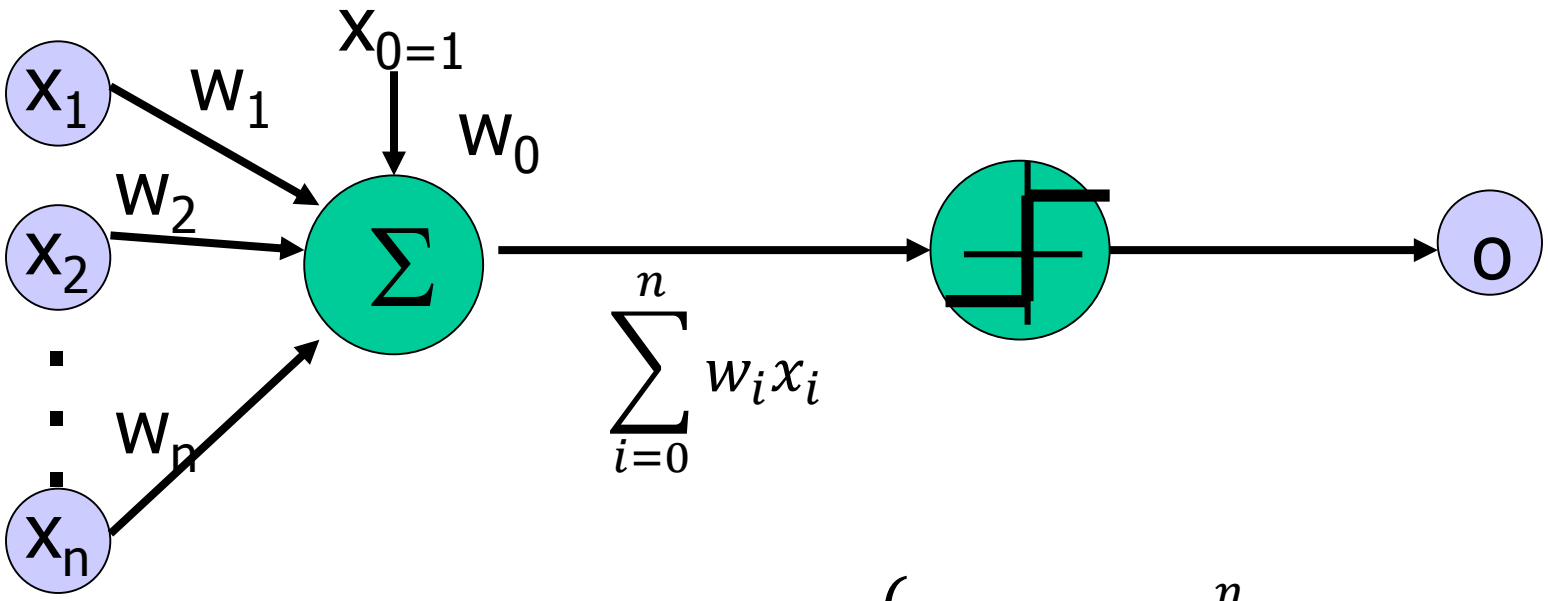
Perceptron Model

$$Y = I(\sum_i w_i X_i - t) \quad \text{or}$$

$$Y = sign(\sum_i w_i X_i - t)$$

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t

Perceptron



$$\sum_{i=0}^{n} w_i x_i$$

$$o(x_0, \ldots, xn) = \begin{cases} 1 & if \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & otherwise \end{cases}$$

Perceptron

- **Perceptron** is a **Linear Threshold Unit (LTU).**
- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs 1 if the result is greater than some threshold and -1 otherwise.
- Given inputs x_1 through x_n, the output $o(x_1, \ldots, x_n)$ computed by the perceptron is:

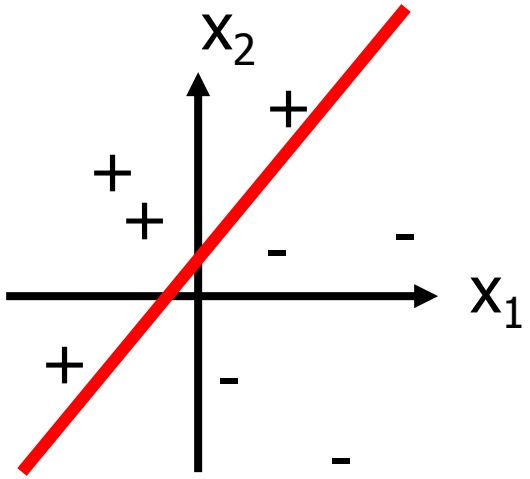
$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

  each w_i is a real-valued constant, or weight, that determines the contribution of input x_i to the perceptron output.
- The quantity $(-w_0)$ is a **threshold** that the weighted combination of inputs must surpass in order for the perceptron to output 1.
  - To simplify notation, we imagine an additional constant input $x_0 = 1$

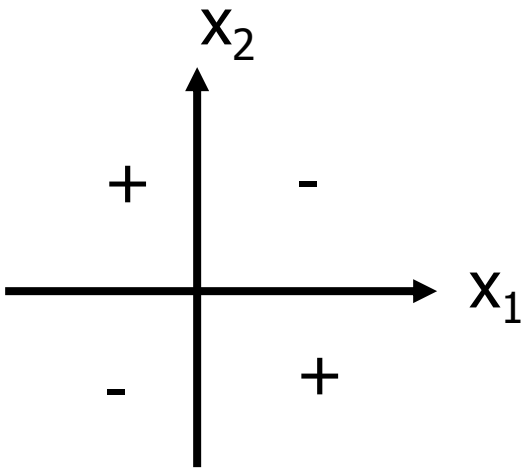
Perceptron

- **Learning a perceptron** involves choosing values for weights $w_0, \ldots, w_n$.
- A perceptron represents a hyperplane decision surface in the n-dimensional space of instances.
- The perceptron outputs 1 for instances lying on one side of the hyperplane and outputs -1 for instances lying on the other side.
- Some sets of positive and negative examples cannot be separated by any hyperplane.
  - Those that can be separated are called **linearly separable** sets of examples.
- A single perceptron can be used to represent many boolean functions.
  - AND, OR, NAND, NOR are representable by a perceptron
  - XOR cannot be representable by a perceptron.

Representational Power of Perceptrons



Representable by a perceptron



NOT representable by a perceptron

Perceptron - Example

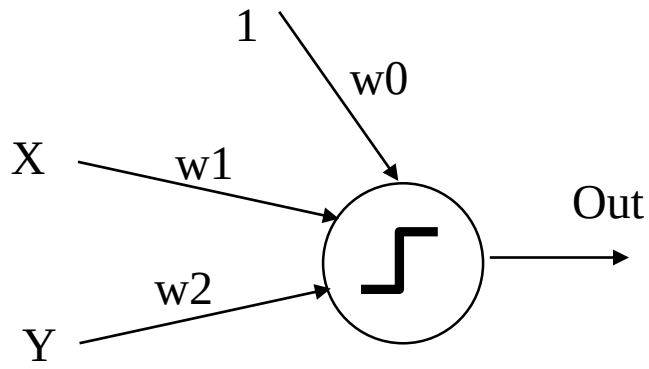
X	Y	Out
0	0	0
0	1	1
1	0	1
1	1	1

Give a perceptron to represent OR function.

Perceptron - Example

X	Y	Out
0	0	0
0	1	1
1	0	1
1	1	1

Give a perceptron to represent OR function.



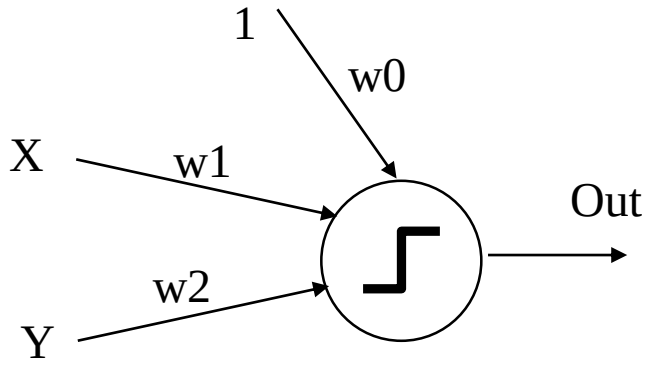
Out is 1 if w0+w1*X+w2*Y > 0 ; 0 otherwise

What will be the weights?

Perceptron - Example

X	Y	Out
0	0	0
0	1	1
1	0	1
1	1	1

Give a perceptron to represent OR function.



Out is 1 if w0+w1*X+w2*Y > 0 ; 0 otherwise

w0 = -0.5 w1 = 0.7 w2 = 0.7

Perceptron - Example

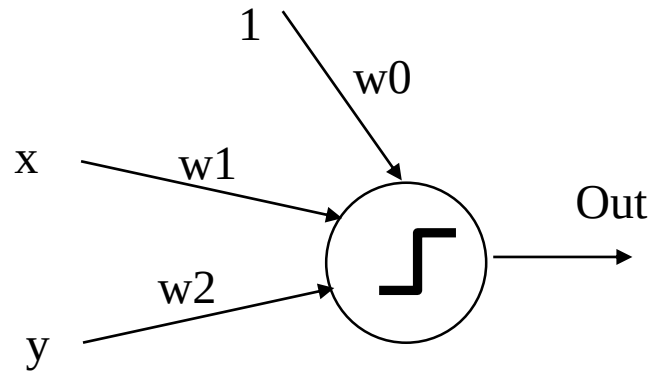
Attributes x	Attributes y	Classification
0	0	1
0	1	1
1	0	1
1	1	0

Give a linear threshold unit (a perceptron) that implements the following function by giving its weight values.

Perceptron - Example

Attributes x	Attributes y	Classification
0	0	1
0	1	1
1	0	1
1	1	0

Give a linear threshold unit (a perceptron) that implements the following function by giving its weight values.



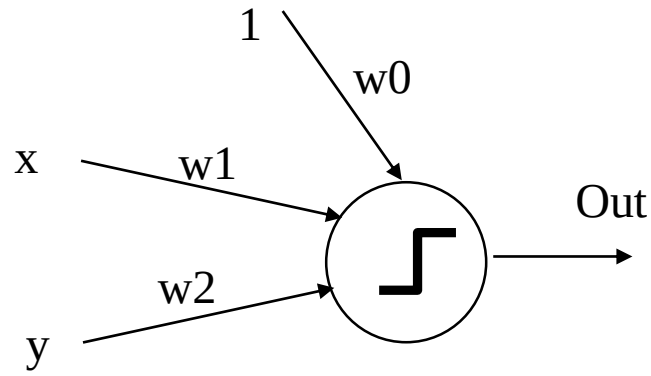
Out is 1 if w0+w1*x+w2*y > 0 ; 0 otherwise

What will be the weights?

Perceptron - Example

Attributes x	Attributes y	Classification
0	0	1
0	1	1
1	0	1
1	1	0

Give a linear threshold unit (a perceptron) that implements the following function by giving its weight values.



Out is 1 if w0+w1*x+w2*y > 0 ; 0 otherwise

w0 = 1 w1 = -0.6 w2 = -0.6

Perceptron - Example

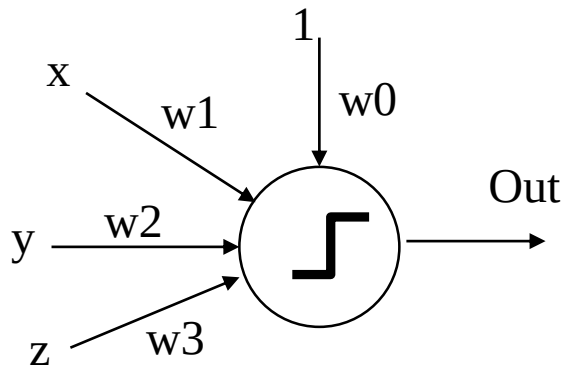
Attributes			Classification
x	y	z	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Give a linear threshold unit (a perceptron) that implements the following function by giving its weight values.

Perceptron - Example

Attributes			Classification
x	y	z	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Give a linear threshold unit (a perceptron) that implements the following function by giving its weight values.



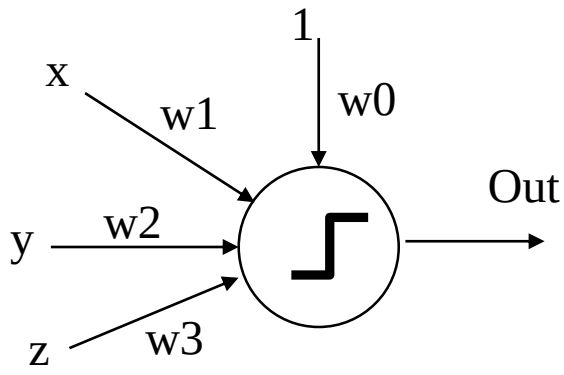
Out is 1 if $w0+w1*x+w2*y+w3*z > 0$; 0 otherwise

What will be the weights?

Perceptron - Example

Attributes x	y	z	Classification
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Give a linear threshold unit (a perceptron) that implements the following function by giving its weight values.



Out is 1 if w0+w1*x+w2*y +w3*z > 0 ; 0 otherwise

w0 = -0.5 w1 = 0.6 w2 = 0.3 w3=0.3

Perceptron Training Rule

- To learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
  - If the training example classifies correctly, weights are not updated.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
  - Each pass through all of the training examples is called one **epoch**
- Weights are modified at each step according to **perceptron training rule**

Perceptron Training Rule

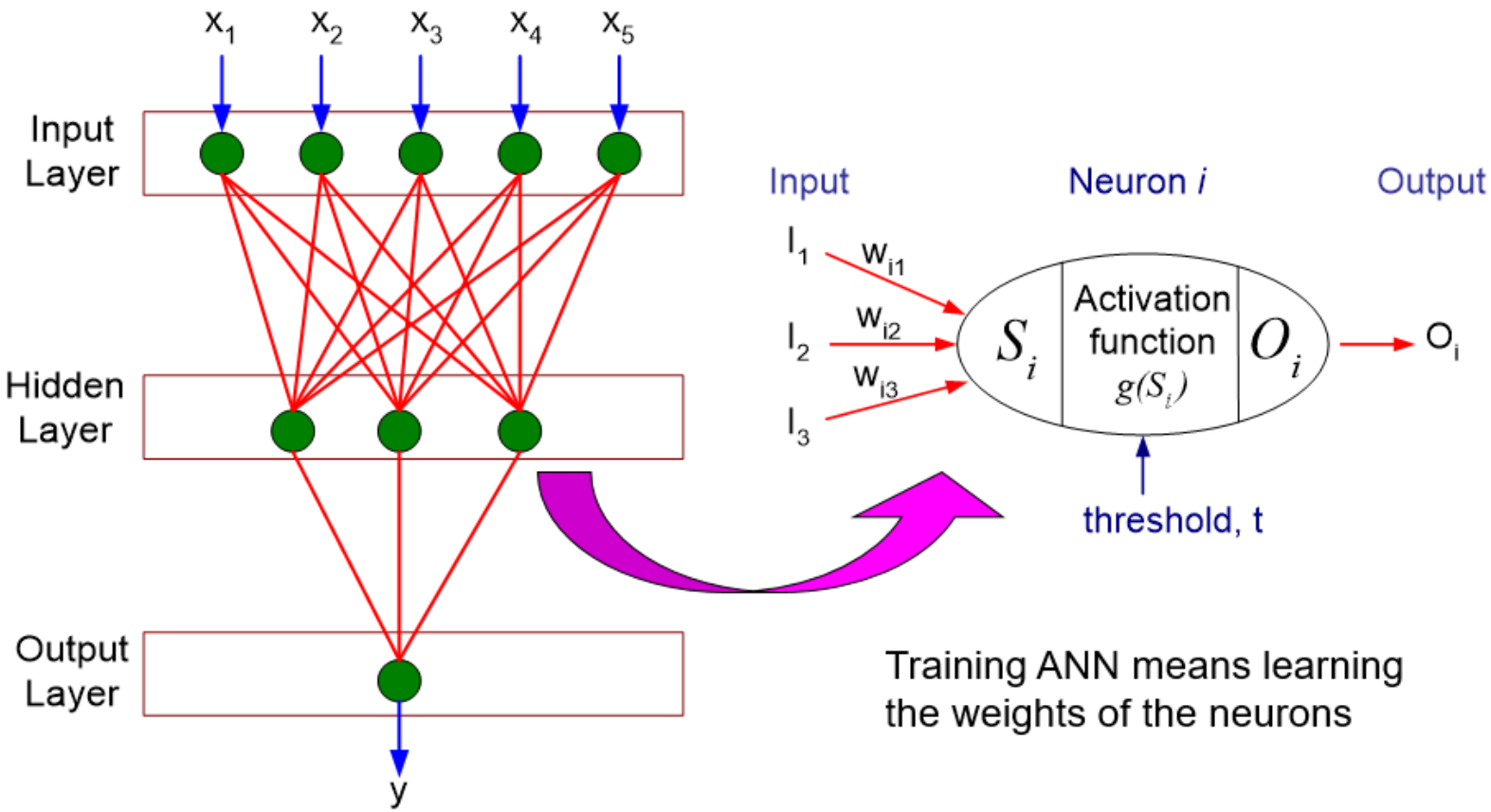
- Weights are modified at each step according to **perceptron training rule.**

$$\mathbf{w_i = w_i + \Delta w_i}$$
$$\mathbf{\Delta w_i = \eta\,(t - o)\,x_i}$$

t is the target value
o is the perceptron output
η is a small constant (e.g. 0.1) called ***learning rate***

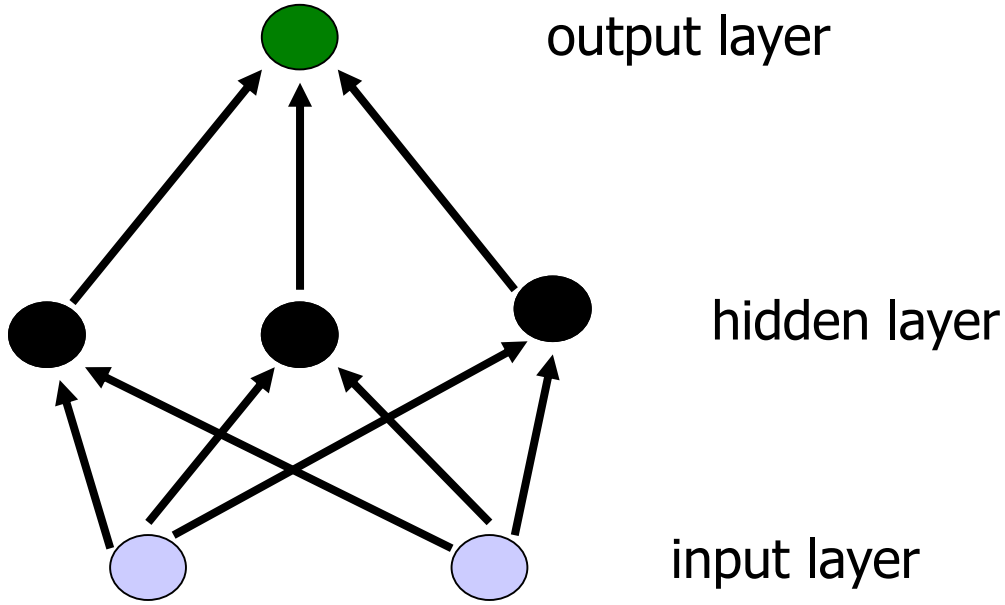
- If the output is correct (t=o) the weights w_i are not changed
- If the output is incorrect (t≠o) the weights w_i are changed such that the output of the perceptron for the new weights is *closer* to t.
- The algorithm converges to the correct classification
  - if the training data is linearly separable
  - and η is sufficiently small

General Structure of ANN



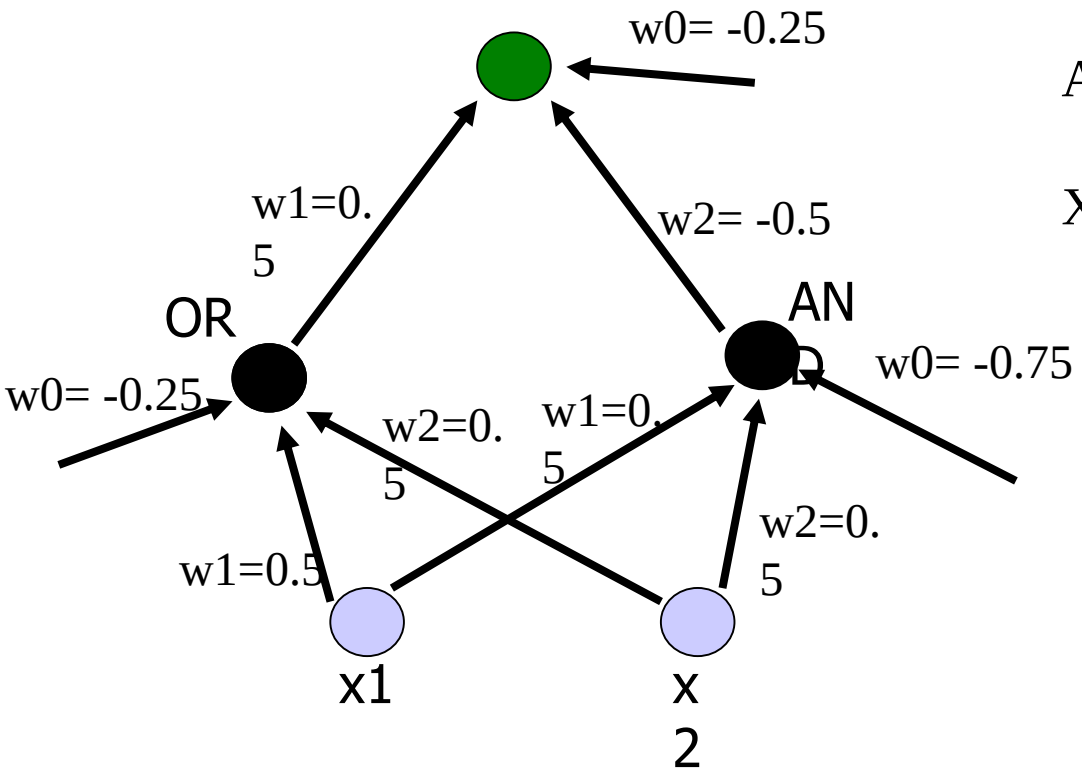
Multi-Layer Networks

- Single perceptron can only express linear decision surfaces.
- Multilayer networks are capable of expressing a rich variety of nonlinear decision surfaces.



Multi-Layer Networks with Linear Units
Ex. XOR

- Multiple layers of cascaded linear units still produce only linear functions.



OR: 0.5*x1 + 0.5*x2 – 0.25 > 0

AND: 0.5*x1 + 0.5*x2 – 0.75 > 0

XOR: 0.5*x1 - 0.5*x2 – 0.25 > 0

Multi-Layer Networks with Linear Units - Example

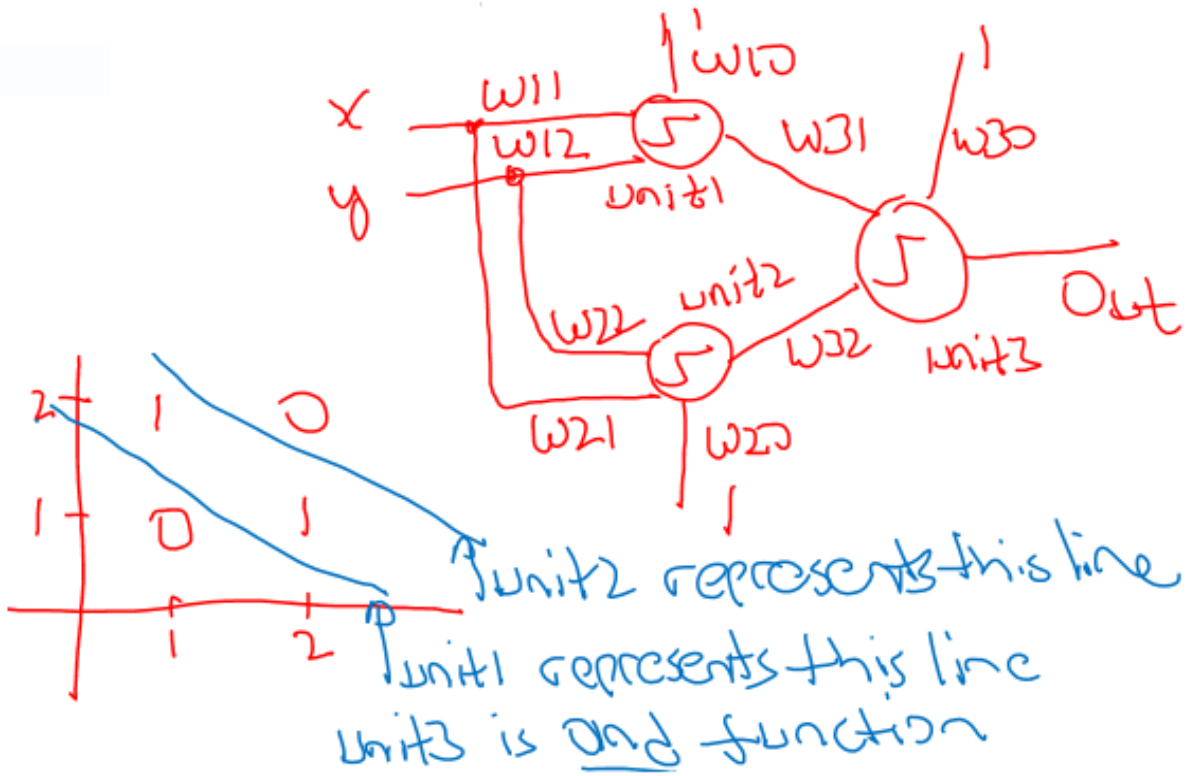
- Give an artificial neural network for the following function. Make clear the structure of your ANN and the used weights.

Attributes		Classification
x	y	
1	1	0
1	2	1
2	1	1
2	2	0

Multi-Layer Networks with Linear Units - Example

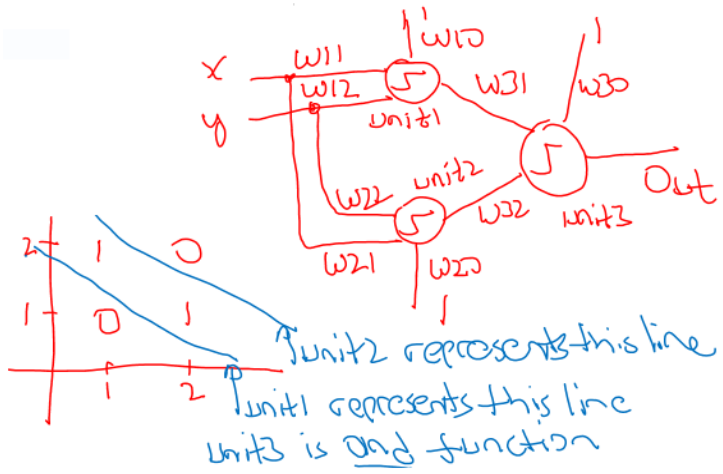
Give an artificial neural network for the following function. Make clear the structure of your ANN and the used weights.

Attributes		Classification
x	y	
1	1	0
1	2	1
2	1	1
2	2	0



Multi-Layer Networks with Linear Units - Example

Attributes x	Attributes y	Classification
1	1	0
1	2	1
2	1	1
2	2	0



Unit1: w10 = -1 w12 = 0.4 w12 = 0.4
Unit2: w20 = 1 w21 = -0.3 w22 = -0.3
Unit3: w30 = - 0.6 w31 = 0.5 w32 = 0.5 (and function)

x	y	unit1	unit2	unit3	Classification
1	1	0	1	0	0
1	2	1	1	1	1
2	1	1	1	1	1
2	2	1	0	0	0

Multi-Layer Networks with Non-Linear Units

- Multiple layers of cascaded linear units still produce only linear functions.
- We prefer networks capable of representing highly nonlinear functions.
- What we need is a unit whose output is a nonlinear function of its inputs, but whose output is also a differentiable function of its inputs.
- One solution is the **sigmoid unit**, a unit very much like a perceptron, but based on a smoothed, differentiable threshold function.

Algorithm for Learning ANN

- Initialize the weights (w_0, w_1, …, w_k)
- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples
  - Objective function: $E = \sum_i [Y_i - f(w_i, X_i)]^2$
  - Find the weights w_i's that minimize the above objective function using **backpropagation algorithm**

Units of ANN

- ***Perceptron (Linear Threshold Unit)***
- ***Linear Unit*** produces continuous output o (not just –1,1)

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

- ***Sigmoid Unit***