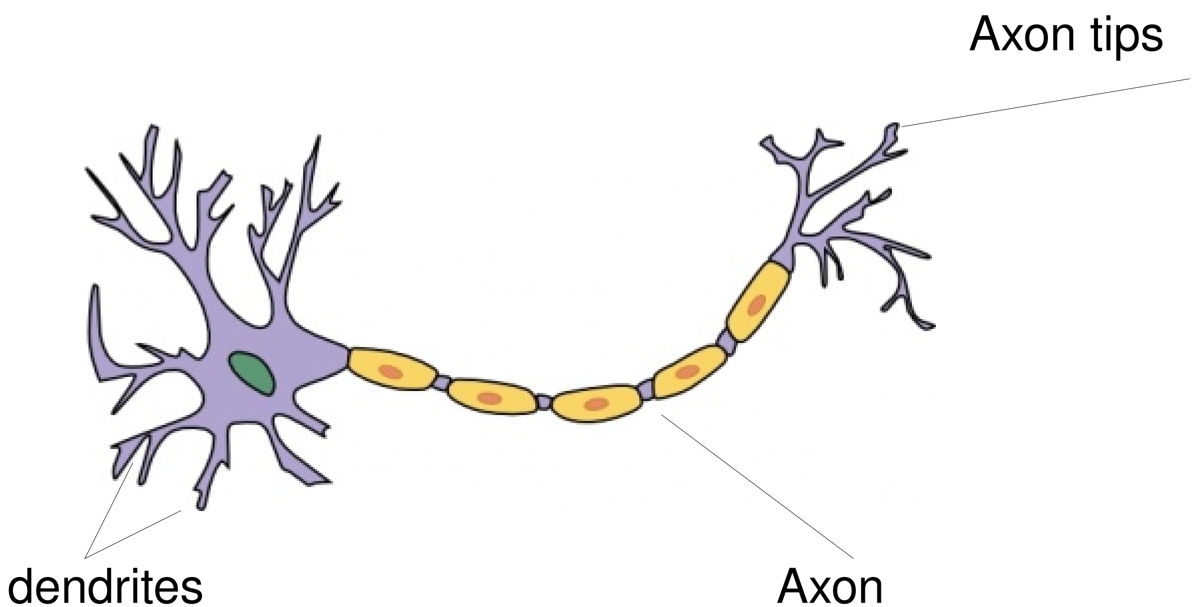
The Perceptron

Rodrigo Fernandes de Mello
Invited Professor at Télécom ParisTech
Associate Professor at Universidade de São Paulo, ICMC, Brazil
http://www.icmc.usp.br/~mello
mello@icmc.usp.br

Artificial Neural Networks

- Conceptually based on biological neurons
- Programs are written to mimic the behavior of biological neurons
- Synaptic connections forward signals from dendrites to the axon tips



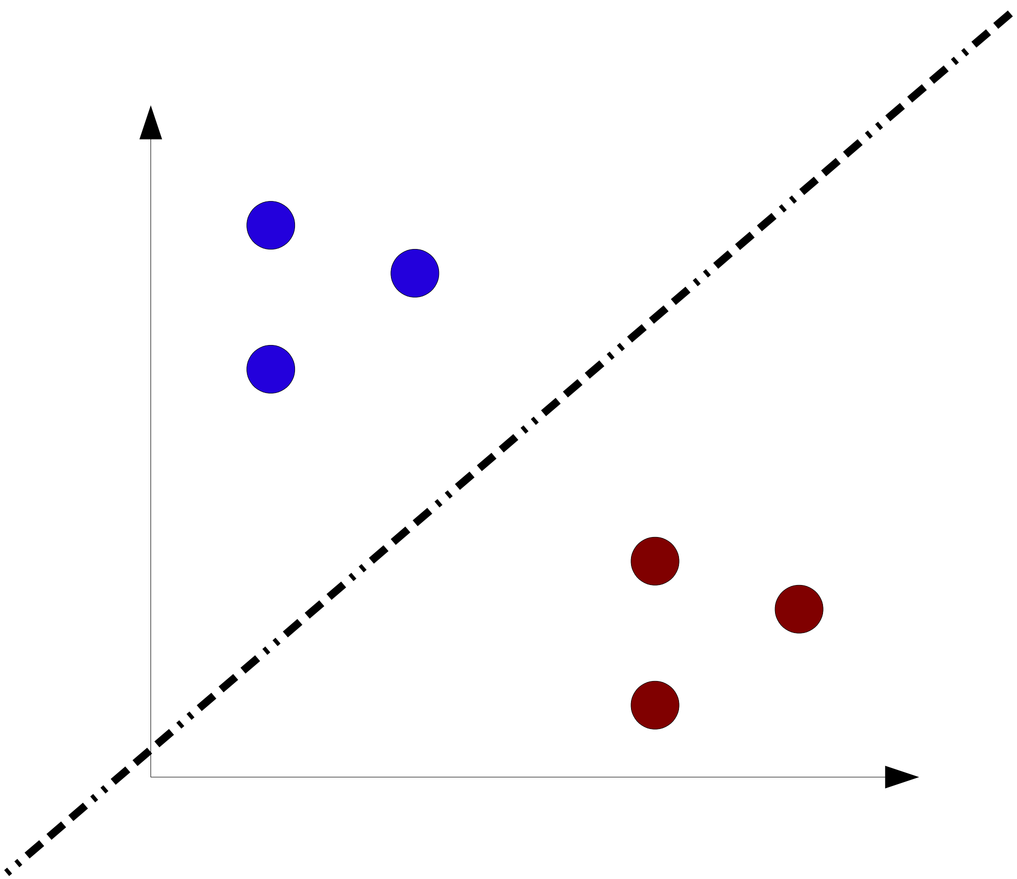
Artificial Neural Networks: History

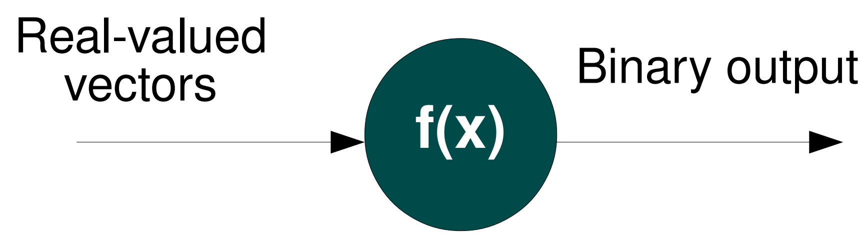
- McCullouch and Pitts (1943) proposed a computational model based on biological neural networks
  - This model was named Threshold logic

- Hebb (1940s), psychologist, proposed the learning hypothesis based on the neural plasticity mechanism:
  - Neural plasticity
    - Ability the brain has to remodel itself based on life experiences
    - Definition of connections based on needs and environmental factors
  - It originated the Hebbian Learning (employed in Computer Science since 1948)

Artificial Neural Networks: History

- Rosenblatt (1958) proposed the Perceptron model
  - A linear and binary classifier





$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

w – weights
x – input vector
b – bias (correction term)

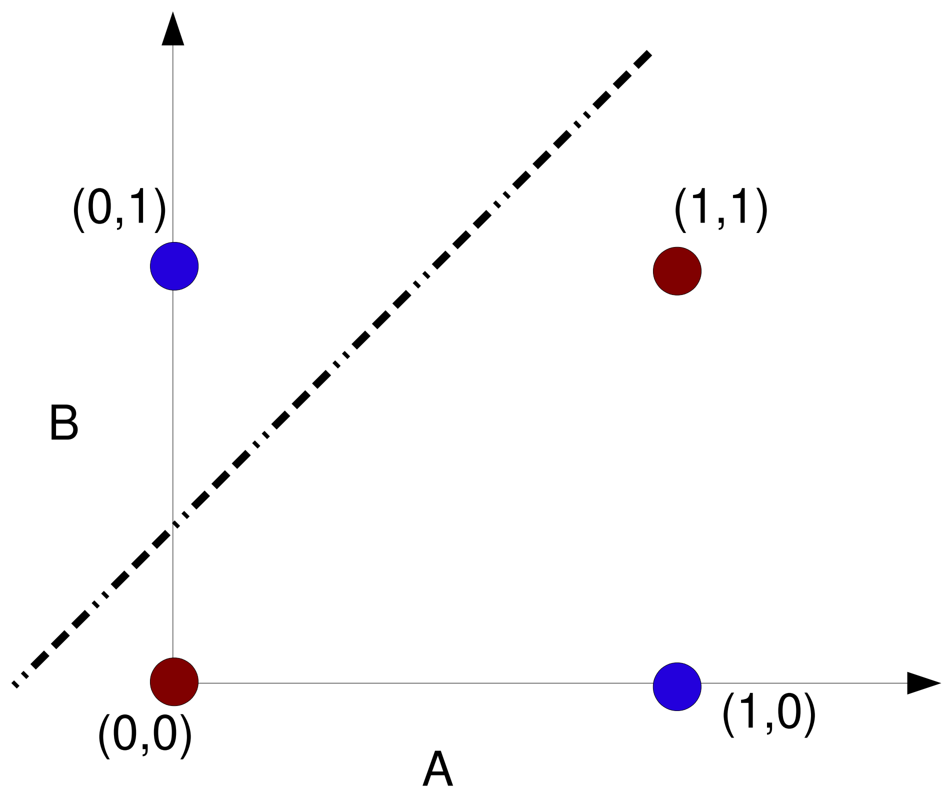
|w| = |x|

Artificial Neural Networks: History

- After the publication by Minsky and Papert (1969), this area got stuck, because they found out:
  - That problems such as the Exclusive-Or could not be solved using the Perceptron
  - Computers did not have enough capacity to process large-scale artificial neural networks

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

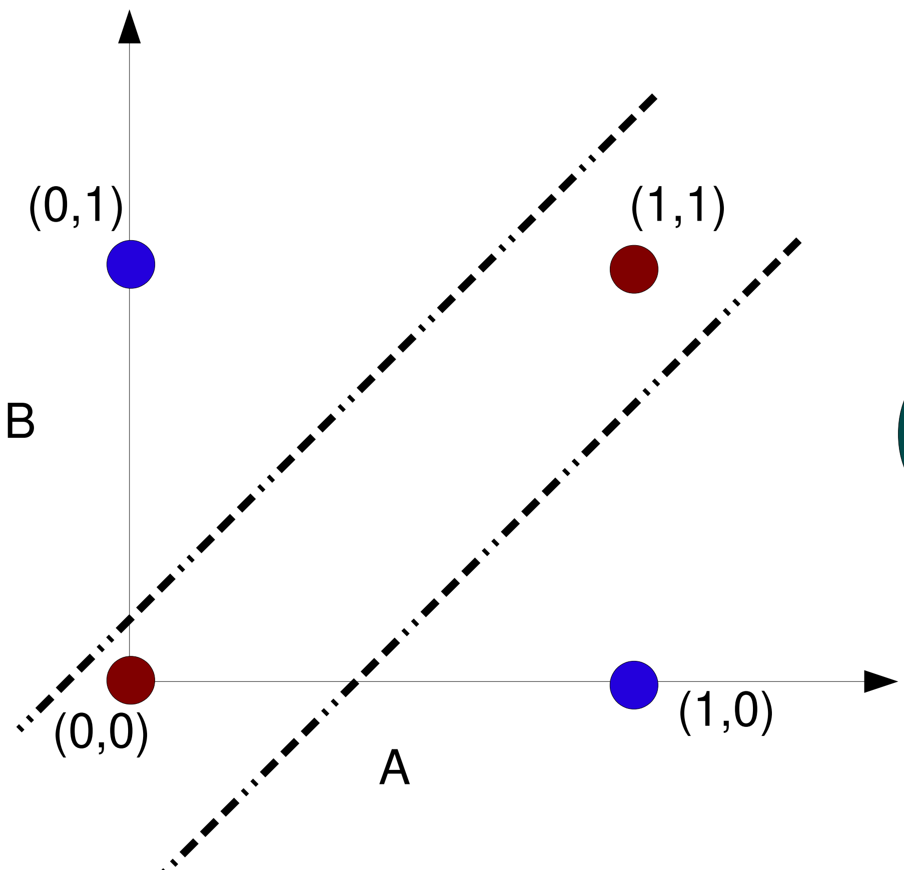


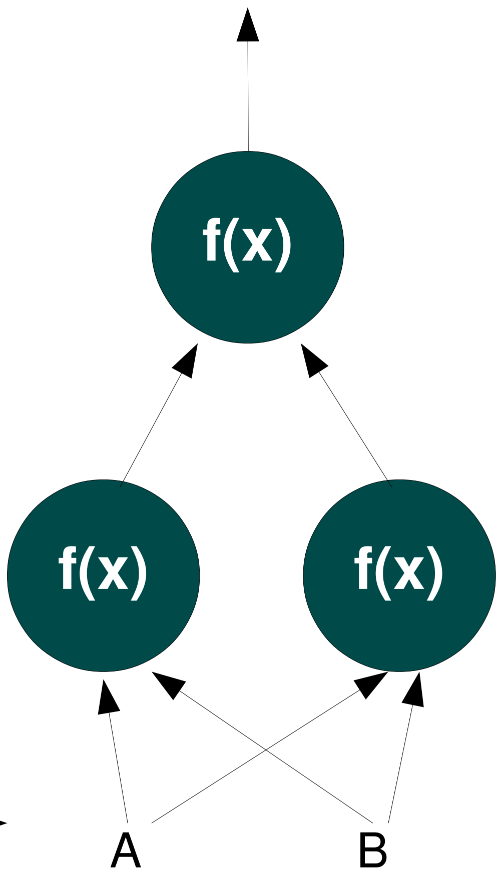
Artificial Neural Networks: History

- Investigations got back after the Backpropagation algorithm (Webos 1975)
  - It solved the Exclusive-Or problem

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0





Artificial Neural Networks

- In 1980's, the distributed and parallel processing area emerges using the name **conexionism**
  - Due to its usage to implement Artificial Neural Networks

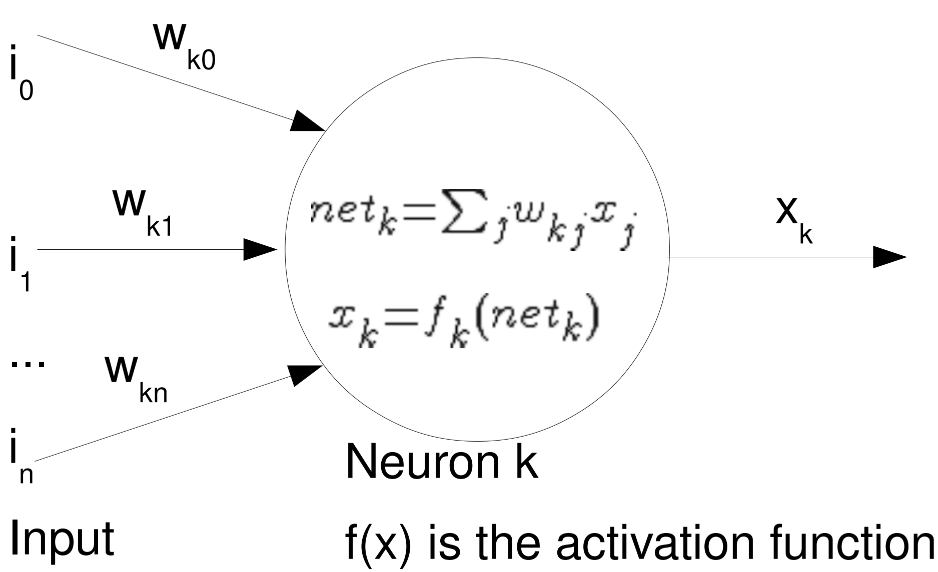
- “Rediscovery” of the Backpropagation algorithm through the paper entitled “Learning Internal Representations by Error Propagation” (1986)
  - This has motivated its adoption and usage

Artificial Neural Networks

- Applications:
  - Speech recognition
  - Image classification
  - Identification of health issues
    - AML, ALL, etc.
  - Software agents
    - Games
    - Autonomic robots

General Purpose Processing Element

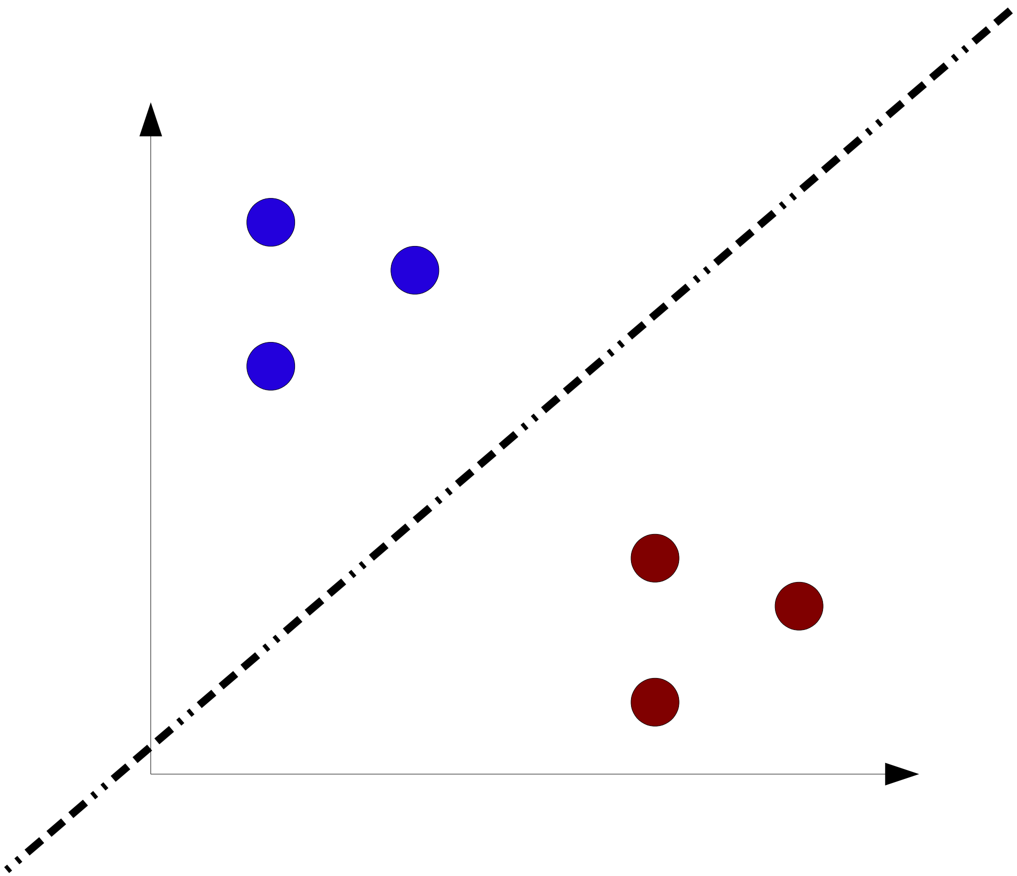
- Artificial neurons:
  - Nodes, units or processing elements
  - They can receive several values as input, but only produces one single output
  - Each connection is associated to a weight w (connection strength)
  - Learning happens by adapting weights w

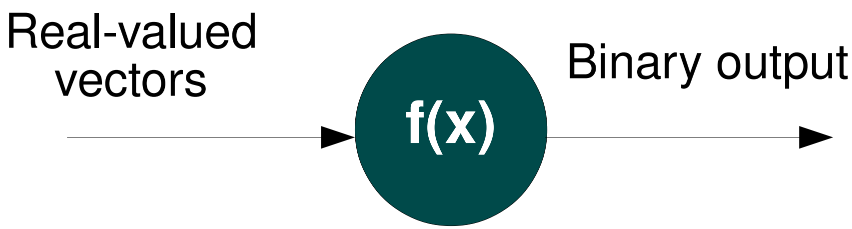


The Perceptron

The Perceptron

- Rosenblatt (1958) proposed the Perceptron model
  - A linear and binary classifier

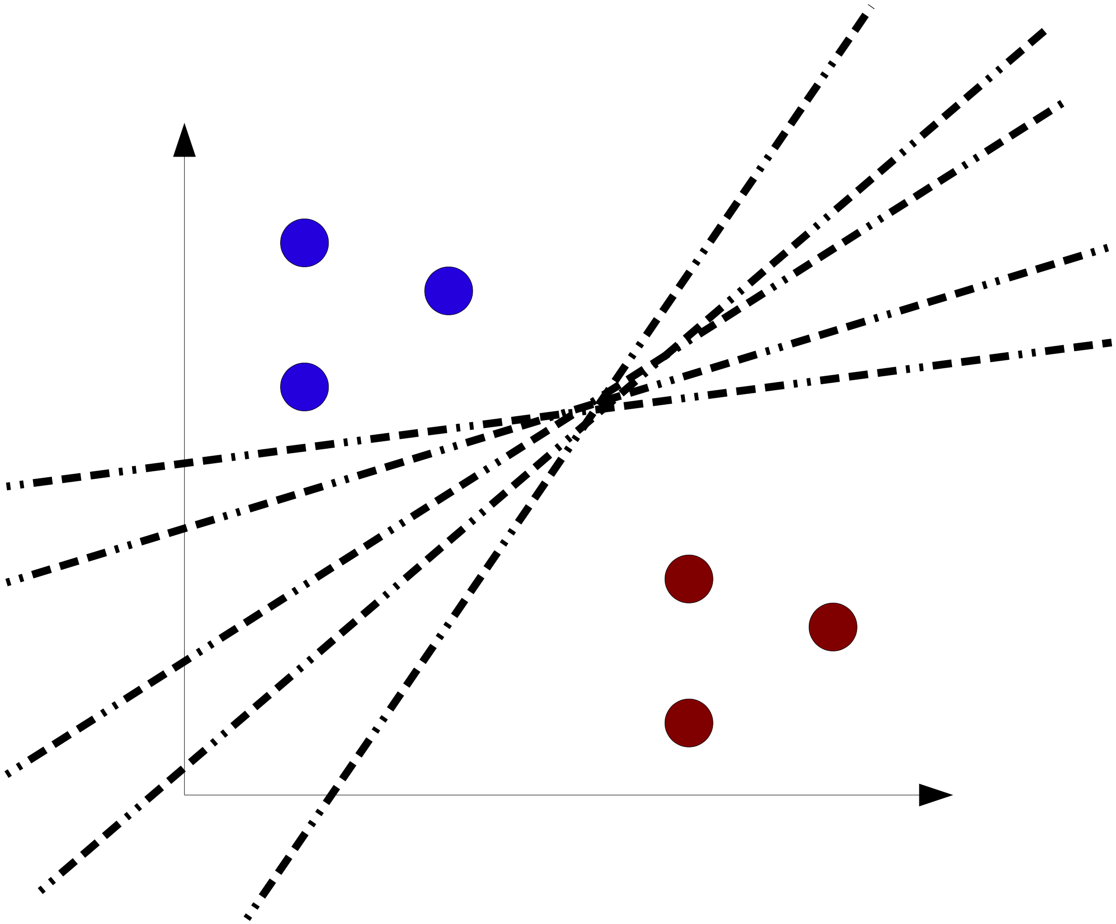




$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

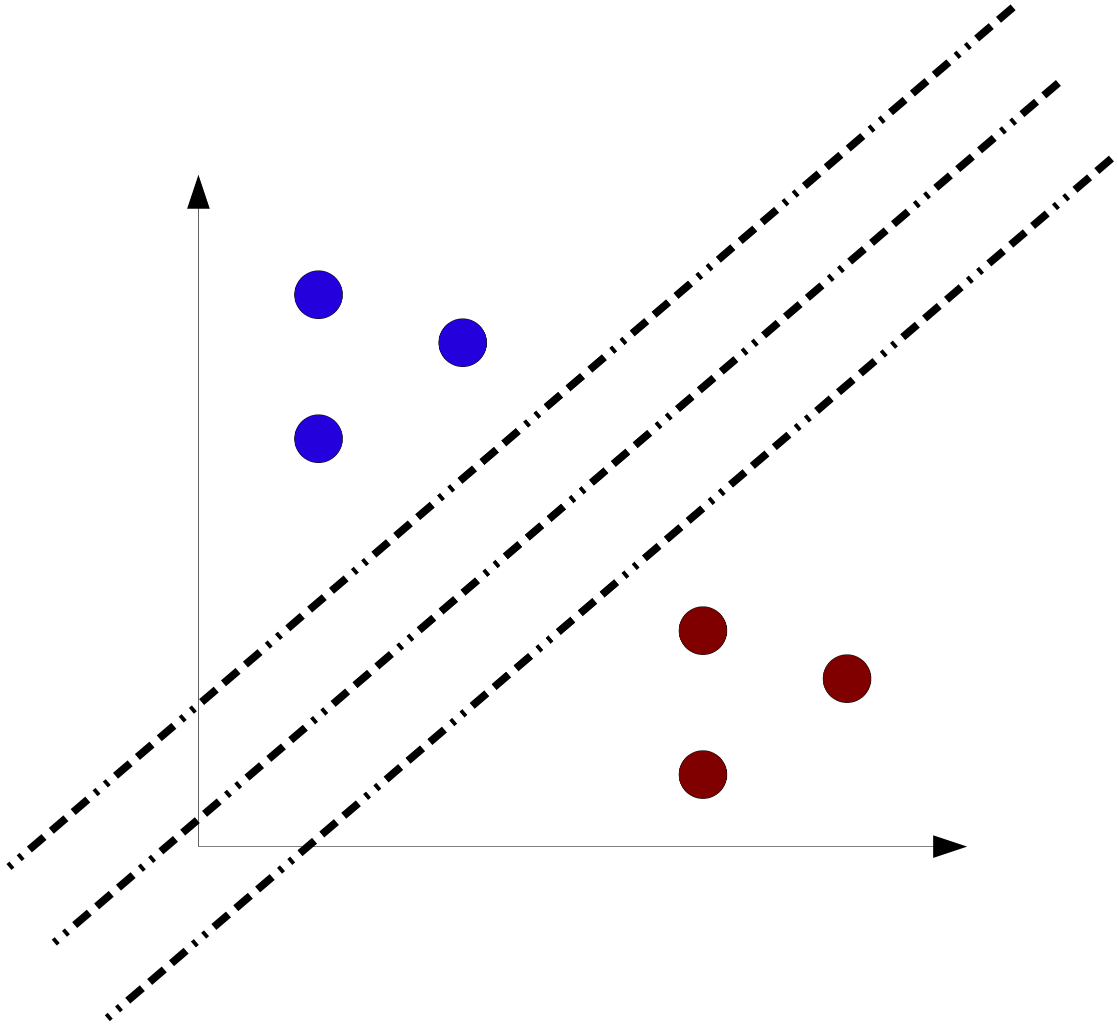
w – weights
x – input vector
b – bias (correction term)

|w| = |x|



Weights w modify the slope of the line

Try gnuplot using:
pl x, 2*x, 3*x



Bias b only modifies the position in relation to y axis

Try gnuplot using:
pl x, x+2, x+3

The Perceptron

- Perceptron learning algorithm does not converge when data is not linearly separable
- Algorithm parameters:
  - $y = f(i)$ is the perceptron output for an input vector i
  - b is the bias
  - $D = \{(x_1, d_1), \ldots, (x_s, d_s)\}$ corresponds to the training set with s examples, in which:
    - X_1 is the input vector with n dimensions
    - d_1 is the expected output
  - $x_{j,i}$ is the value for neuron i given an input vector j
  - w_i is the weight i to be multiplied by the i-th value of the input vector
  - α is the learning rate which is typically in range (0,1]
  - Greater learning rates make the perceptron oscillate around the solution

摆动

The Perceptron

- Algorithm
  - Initialize weights w using random values
  - For every pair j in training set D
    - Compute the output

$$y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j] = f[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}]$$

    - Adapt weights

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$, for all nodes $0 \leq i \leq n$.

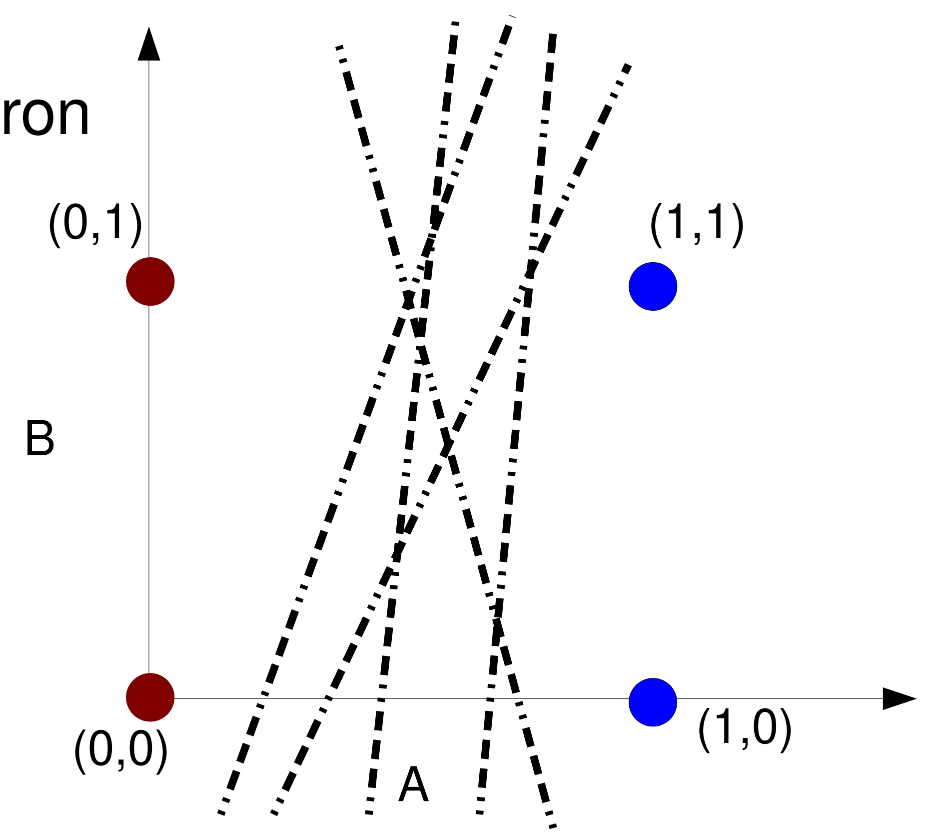
  - Execute until the error is less than a given threshold or for a number of iterations

$$d_j - y_j(t) < \gamma$$

The Perceptron

- Activation (or transference) function for the Perceptron
  - Step function
  - Try on gnuplot using:
    - f(x)=(x>0.5) ? 1 : 0
    - pl f(x)
- Implementation
  - Solve NAND using the Perceptron

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0



The Perceptron

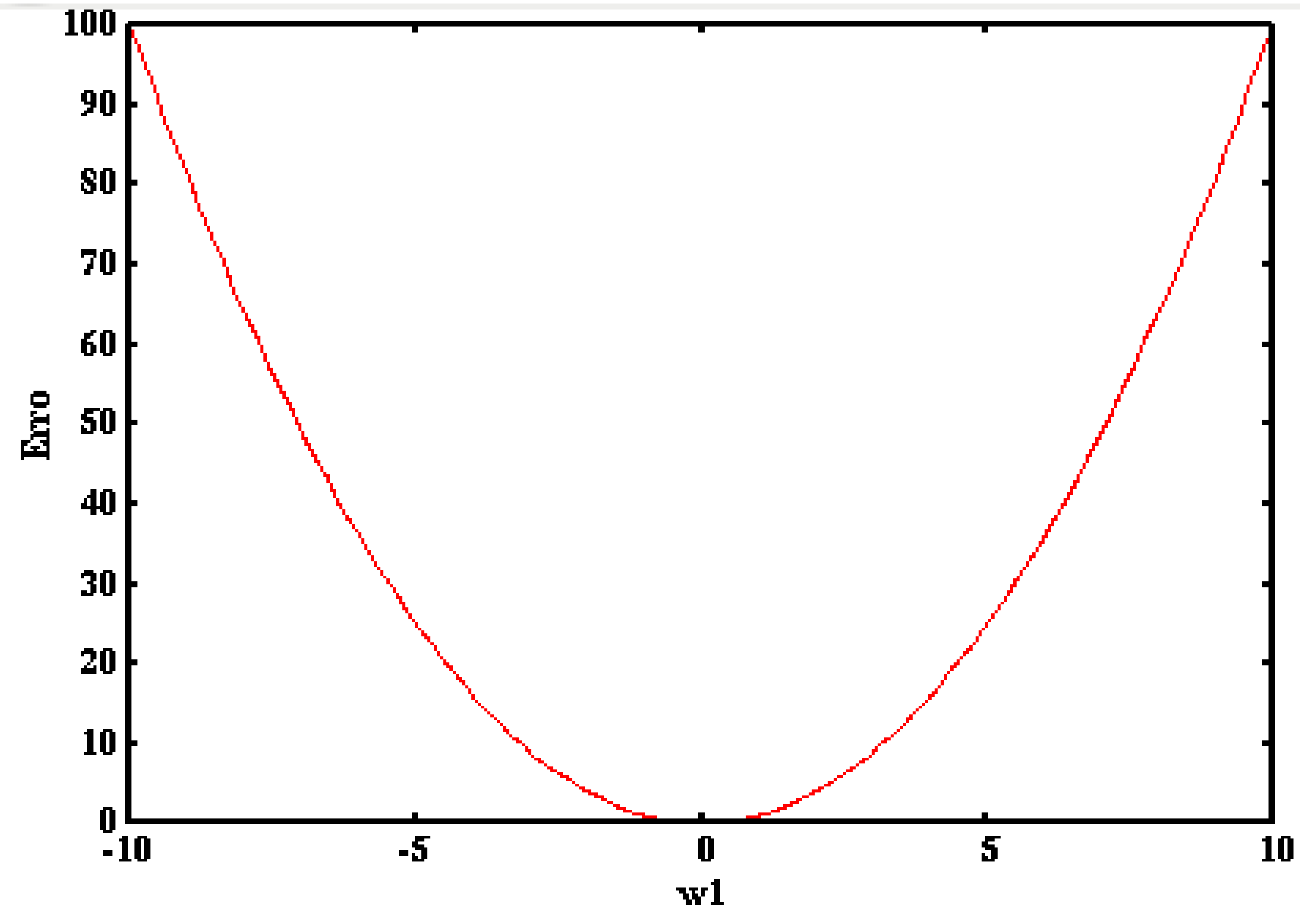
- Implementation
  - NAND
    - Verify weights and plot them using Gnuplot
    - As we have two input dimensions, we must plot it using command “spl”
      - Plot the hyperplane using the final weights

```
gnuplot> set border 4095 front linetype -1 linewidth 1.000
gnuplot> set view map
gnuplot> set isosamples 100, 100
gnuplot> unset surface
gnuplot> set style data pm3d
gnuplot> set style function pm3d
gnuplot> set ticslevel 0
gnuplot> set title "gray map"
gnuplot> set xlabel "x"
gnuplot> set xrange [ -15.0000 : 15.0000 ] noreverse nowriteback
gnuplot> set ylabel "y"
gnuplot> set yrange [ -15.0000 : 15.0000 ] noreverse nowriteback
gnuplot> set zrange [ -0.250000 : 1.00000 ] noreverse nowriteback
gnuplot> set pm3d implicit at b
gnuplot> set palette positive nops_allcF maxcolors 0 gamma 1.5 gray
gnuplot> set xr [0:1]
gnuplot> set yr [0:1]
gnuplot> spl 1.0290568822825088+-0.15481468877189009*x+-0.46986458608516524*y
```

- More about the Gradient descendent method

- What happens with the weight adaptation?
  - Consider the Error versus weight w_1

$$f(x) = x^2$$



- To find the minima we must:
  - Find the derivative in the direction of the weight

$$\frac{\partial f(x)}{\partial x}$$

  - To reach the minima we must, for a given weight w_1, adapt the weight in small steps
    - If we use large steps, the perceptron “swings” around the minimum

$$x(t+1) = x(t) - \mu \frac{\partial f(x(t))}{\partial x}$$

  - If we change to the plus sign, we go in the direction to the function maxima

Implementation

```
x_old = 0
x_new = 6 # initial value to be applied in the function
eps = 0.01 # step
Precision = 0.00001

double derivative(double x) { return 2 * x; }

while (fabs(x_new - x_old) > precision) {
   x_old = x_new
   x_new = x_old - eps * derivative(x_new)
   printf("Local minimum occurs at %f \n", x_new);
}
```

*Test with different values for the step

*Verify the sign change for eps

- Formalize the adaptative equation

The Perceptron

- How do we get this adaptive equation?

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}, \text{ for all nodes } 0 \le i \le n.$$

  - Consider an input vector **x**
  - Consider a training set $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_L\}$
  - Consider that each **x** must produce an output value **d**
    - Thus we get $\{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_L\}$
  - Consider that each **x** produced, in fact, an output **y**
  - The problem consists in finding a weight vector **w*** that satisfies this relation of inputs and expected outputs
    - Or that produces the small error as possible, i.e., to better represent this relation

The Perceptron

- Consider the difference between the expected output and the produced output for an input vector **x** as follows:

$$\epsilon_k = d_k - y_k$$

- Thus the **average squared error** for all input vectors in the training set is given by:

$$< \epsilon^2 >= \frac{1}{L} \sum_{k=1}^{L} \epsilon_k^2$$

*** Why squared?**

- Disconsidering the step function, we have:

$$y = \mathbf{w}^t \mathbf{x}$$

- Thus we can assume the average error for a vector $\mathbf{x}_k$ is:

$$< \epsilon_k^2 >=< (d_k - \mathbf{w}^t \mathbf{x_k})^2 >$$

The Perceptron

- Iterative solution:
  - We estimate the ideal value of:

$$< \epsilon_k^2 >=< (d_k - \mathbf{w}^t \mathbf{x_k})^2 >$$

  - Using the instantaneous value (based on the input vector):

$$\epsilon_i^2(t) = (d_i - \mathbf{w}^t(t)\mathbf{x}_i)^2$$

  - Having ε_i as the error for an input vector $\mathbf{x}_i$ and the expected output $\mathbf{d}_i$

- In that situation, we derive the **squared error function** in the direction of weights, so we can adapt them:

$$\nabla \epsilon_i^2(t) \approx \nabla < \epsilon_i^2 >$$
$$\nabla \epsilon_i^2(t) = -2\epsilon_i(t)\mathbf{x}_i$$

- Steps:

$$\epsilon_i^2(t) = (d_i - \mathbf{w}^t(t)\mathbf{x}_i)^2$$

Pela regra da cadeia, temos:

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$$

Logo:

$$\frac{d}{d\mathbf{w}} \epsilon_i^2(t) = 2 \cdot (d_i - \mathbf{w}^t(t)\mathbf{x}_i) \cdot -\mathbf{x}_i$$

Ou seja:

$$\frac{d}{d\mathbf{w}} \epsilon_i^2(t) = -2 \cdot \epsilon_i(t) \cdot \mathbf{x}_i$$

- As previously seen, the descent gradient is given by:

$$x(t+1) = x(t) - \mu \frac{\partial f(x(t))}{\partial x}$$

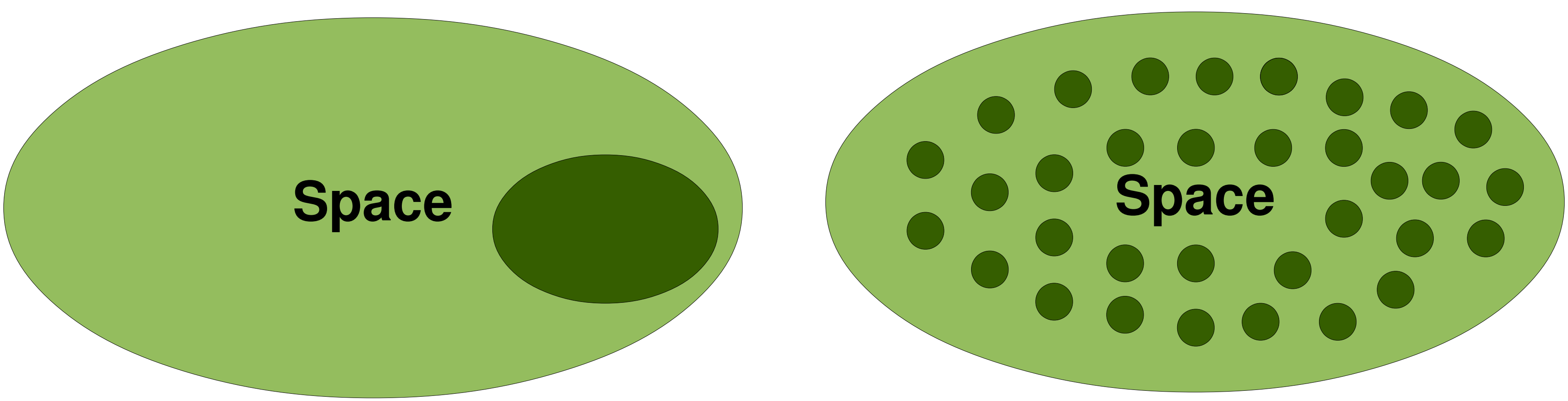
- In our scenario, we model as:

$$\begin{aligned}
&\mathbf{w}(t+1) = \mathbf{w}(t) - \mu \nabla \varepsilon(\mathbf{w}(t)))\\
&\text{Sendo: } \nabla \varepsilon(\mathbf{w}(t)) = \nabla \epsilon_i^2(t) \text{ logo:}\\
&\mathbf{w}(t+1) = \mathbf{w}(t) + 2\mu\epsilon_i \mathbf{x}_i
\end{aligned}$$

- In which μ is the learning rate typically in range (0,1]

The Perceptron

- Observations:
  - Training set must be representative to adapt weights
    - Set must contain diversity 多样化
    - It must contain examples that represent all possibilities for the classification problem
  - Otherwise tests will not produce the expected results



The same amount of examples, however the first is less representative than the second

- Implementation
  - XOR
    - Verify the source code of the Perceptron for the XOR problem

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Perceptron: Solving XOR

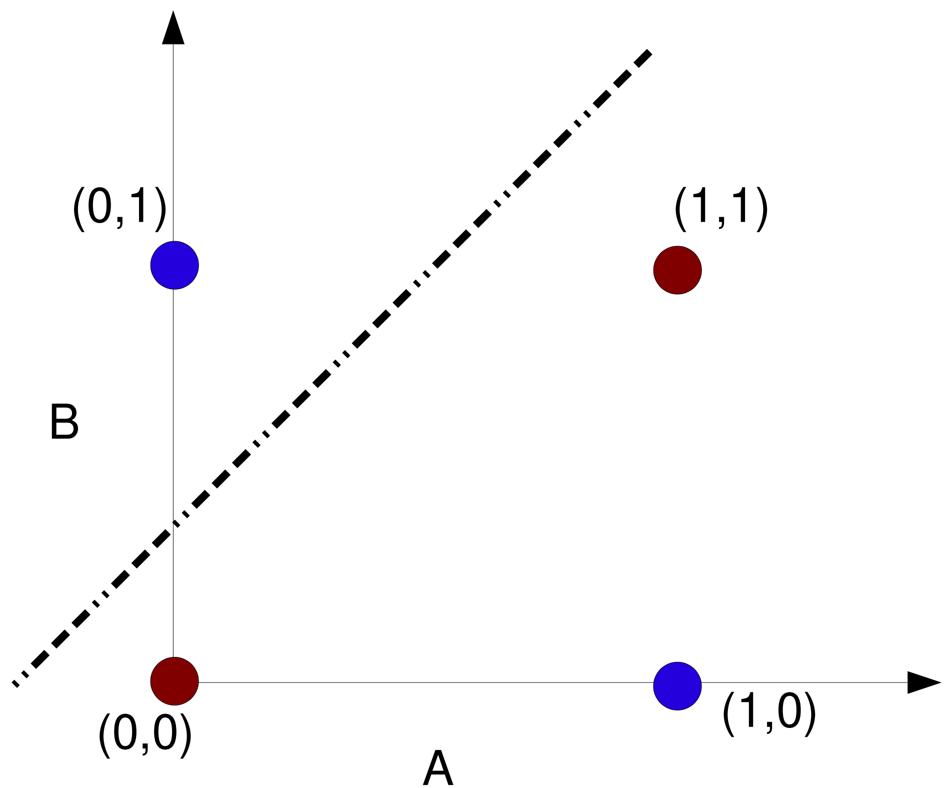
- Minsky and Papert (1969) wrote the book entitled “Perceptrons: An Introduction to Computational Geometry”, MIT
  - They demonstrated that a perceptron linearly separates classes
  - However, several problems (e.g., XOR) are not linearly separable
  - The way they wrote this book seems to question this area
    - As, in that period, the perceptron was significative for the area, then, several researchers believed artificial neural networks, and even AI, were not useful to tackle real-world problems

Perceptron: Solving XOR

- How to separate classes?
  - Which weights we should use? Which bias?

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron: Solving XOR

- Observe the following equation is linear

$$net_i = w_1 x_1 + w_2 x_2$$

- The result of this equation is applied in the activation function

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

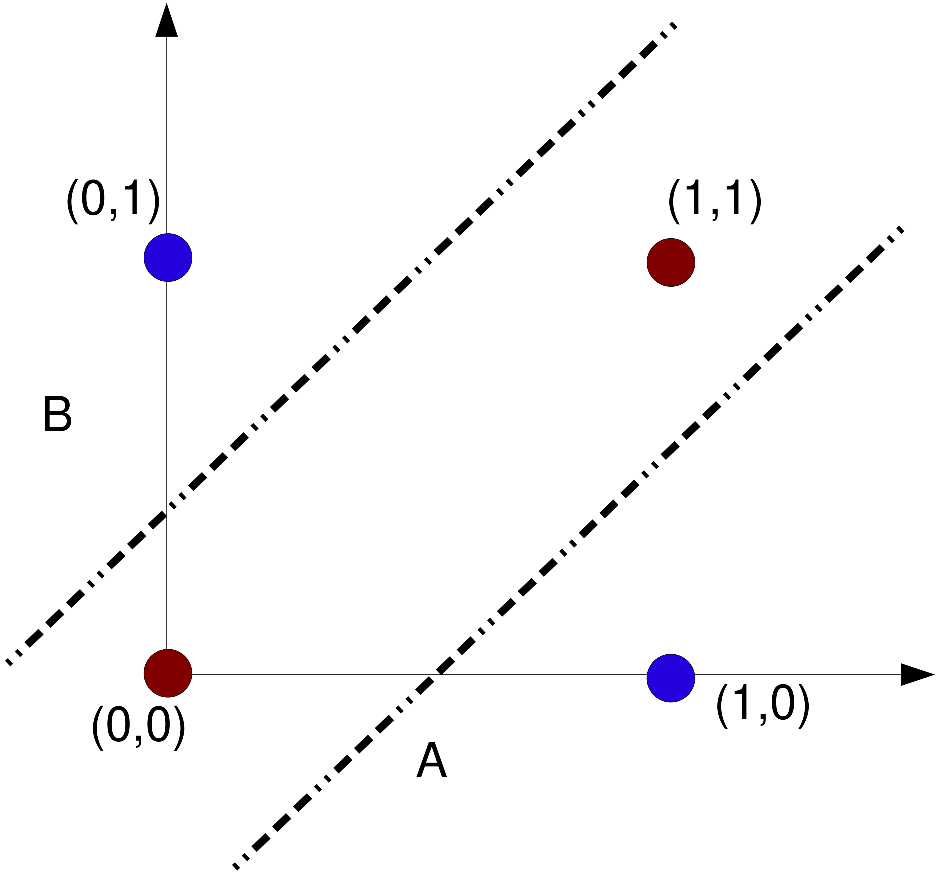
- Thus, it can only linearly separate classes
- In linearly separable problems:
  - This equation builds a hyperplane
  - Hyperplanes are (n-1)-dimensional objects used to separate n-dimensional hyperspaces in two regions

Perceptron: Solving XOR

- We could use two hyperplanes
  - Disjoint regions can be put together to represent the same class

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron: Solving XOR

- This fact does not avoid some problems discussed by Minsky and Papert
  - They still questioned the scalability of artificial neural networks
    - As we approach a large-scale problem, there are undesirable effects:
      - Training is slower
      - Many neurons make learning slower or difficult convergence
      - More hyperplanes favour overfitting
    - Some researchers state that one can combine small scale networks to address such issue

Summary

- Did you understand something?
- Should we get back to some point?