Machine Learning Review

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Rodrigo Fernandes de Mello
http://www.icmc.usp.br/~mello
mello@icmc.usp.br

Machine Learning Review

- What is machine learning for you?
- How would you define ML?

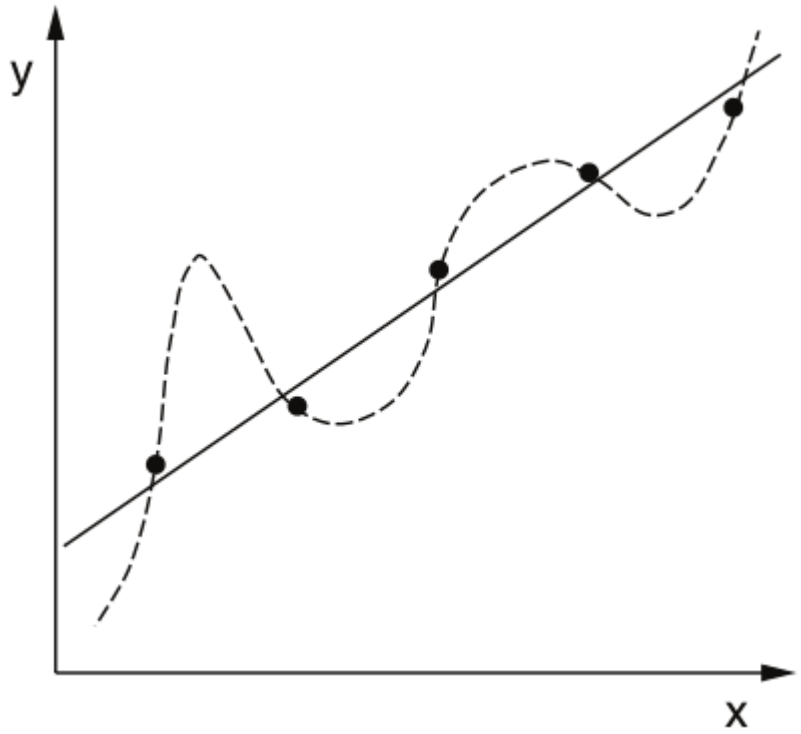
Machine Learning Review

- There are two main types of learning:
  - Supervised
    - Ulrike von Luxburg and Bernhard Schoelkopf, Statistical Learning Theory: Models, Concepts, and Results, Handbook for the History of Logic, Vol. 10: Inductive Logic. Elsevier, 2011
    - D. H. Wolpert and William G. Macready, 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1), 67–82
  - Non-supervised
    - Gunnar Carlsson and Facundo Memoli, Characterization, Stability and Convergence of Hierarchical Clustering Methods, Journal of Machine Learning Research, 2010

Supervised Learning is strongly based on the Bias-Variance Dilemma

Bias-Variance Dilemma

- For instance, consider the examples (points) colected during an experiment. Then, take two different functions to model them



Which function is the best?

To answer it we need to assess their Expected Risks

Figure from Luxburg and Scholkopf, Statistical Learning Theory: Models, Concepts, and Results

Bias-Variance Dilemma

- But how to compute the Expected Risk if we only have one sample?

- **Consider that line f(x) has training error greater than zero**

- **Consider the polinomial function g(x) has training error equals to zero**

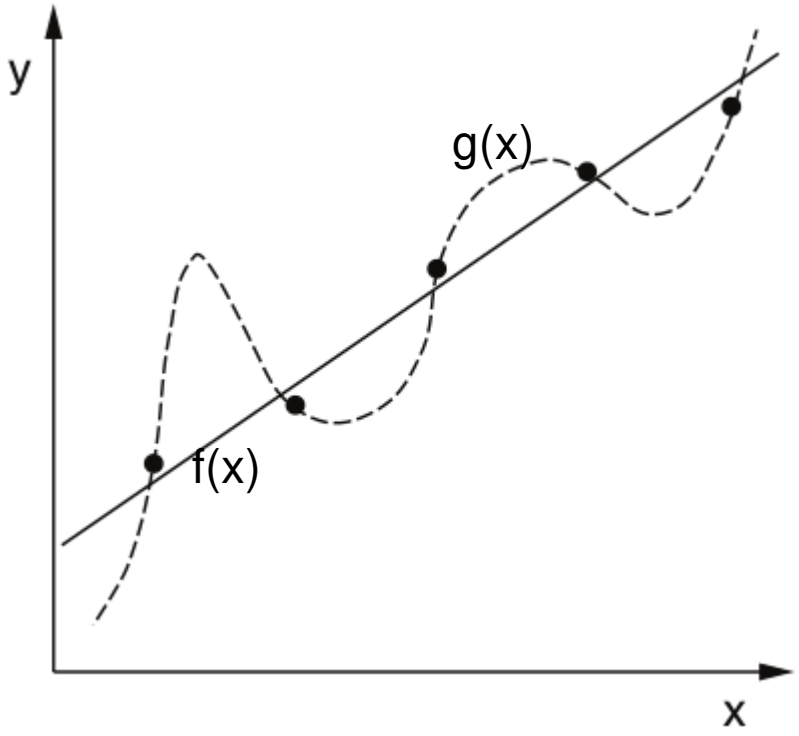


Figure from Luxburg and Scholkopf, Statistical Learning Theory: Models, Concepts, and Results

Bias-Variance Dilemma

- But how to compute the Expected Risk if we only have one sample?

- **Consider that line f(x) has training error greater than zero**
- **Consider the polinomial function g(x) has training error equals to zero**
- **Let g(x) represent an overfitted model**
- **Thus, as unseen examples are received, g(x) moves from no Training error to a great Expected error**

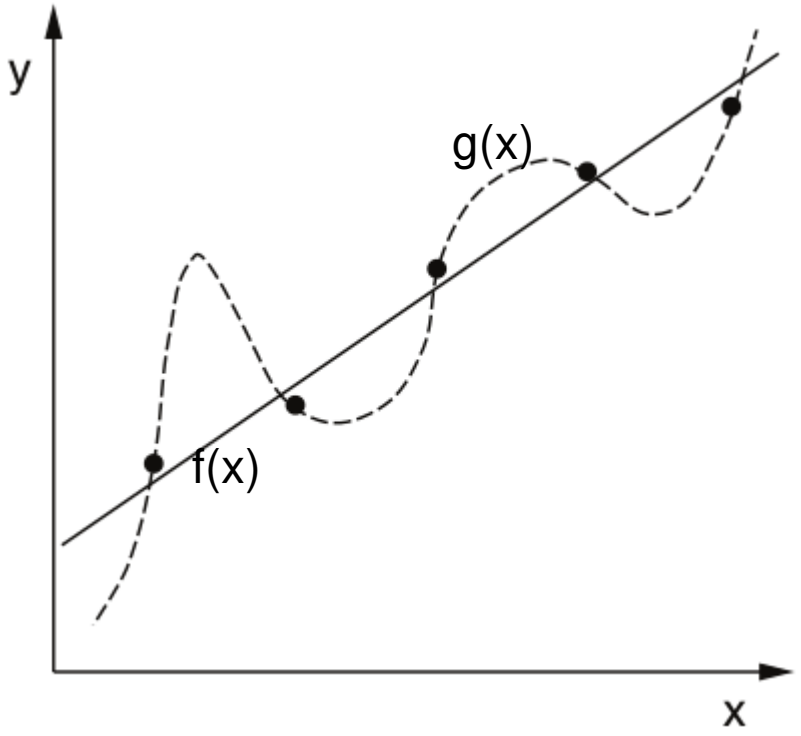


Figure from Luxburg and Scholkopf, Statistical Learning Theory: Models, Concepts, and Results

Bias-Variance Dilemma

- But how to compute the Expected Risk if we only have one sample?

- **Consider that line f(x) has training error greater than zero**
- **Consid g(x) h**
- **Le mo**
- **Thus, received, g Training error to error**

But the opposite could also happen, i.e., g(x) may be the best function!

So, how to select the best one???

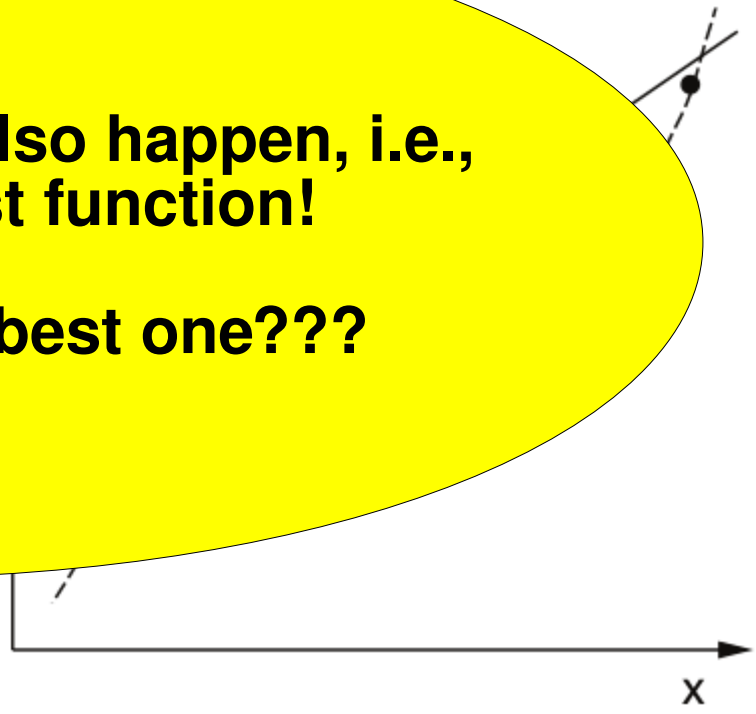


Figure from Luxburg and Scholkopf, Statistical Learning Theory: Models, Concepts, and Results

Bias-Variance Dilemma

- Finally, which classifier should we choose?

1) The one with best fit with training data (the most complex one)?

2) Or the one that has greater Training error, however it was obtained using a simpler class of functions?

In Statistics, this is known as the Bias-Variance Dilemma

This Dilemma is central for supervised learning!

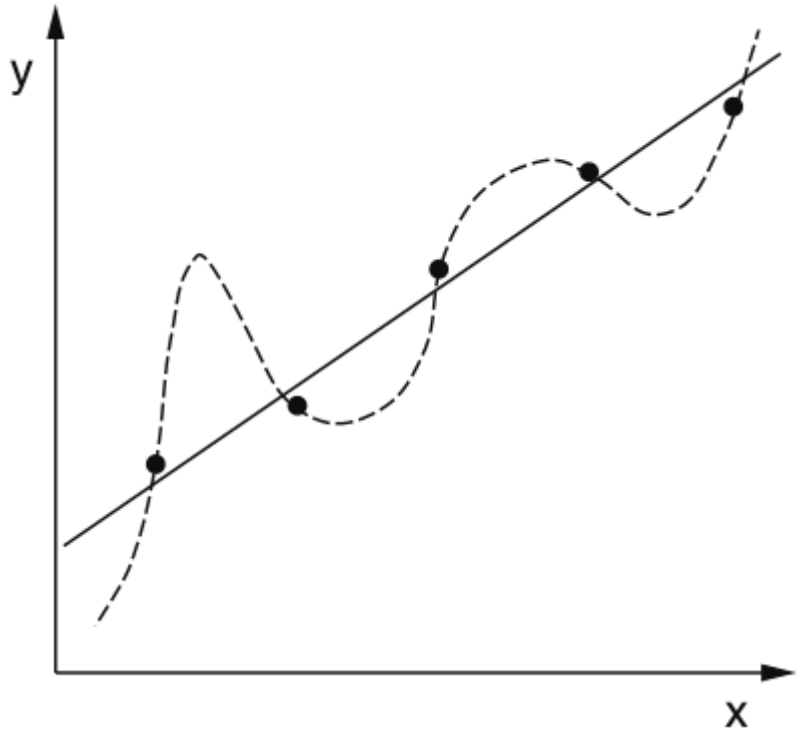


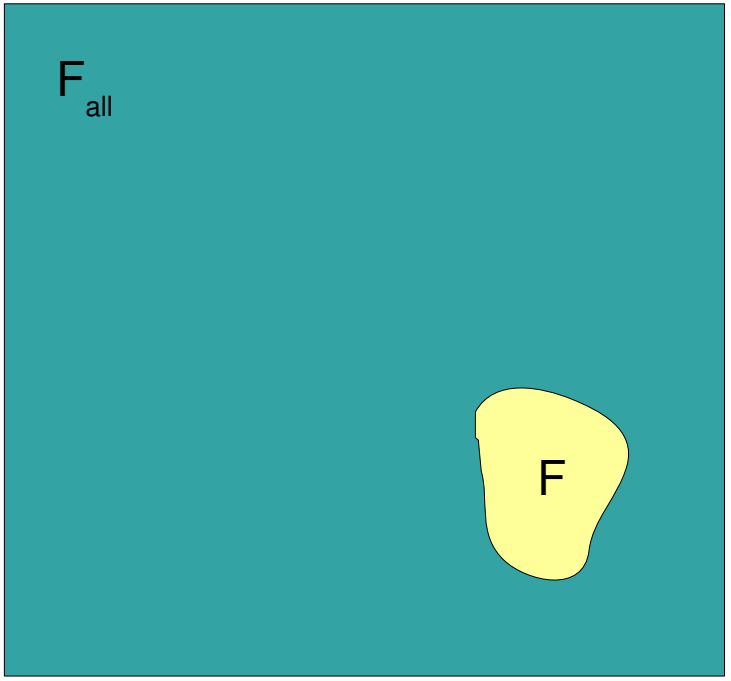
Figure from Luxburg and Scholkopf, Statistical Learning Theory: Models, Concepts, and Results

Bias-Variance Dilemma

- Dilemma:
  - Bias:
    - If we assume a linear fit, only a linear classifier could be obtained
    - i.e., there is a strong bias imposed by ourselves
  - Variance:
    - If we fit a high-order polinomial function over training data, we can always have a perfect classifier for the sample
    - However this classifier is subject to greater fluctuations to unseen data

Bias-Variance Dilemma

- This dichotomy associated to the Bias-Variance Dilemma is obvious when we consider the space of possible functions to build our classifier



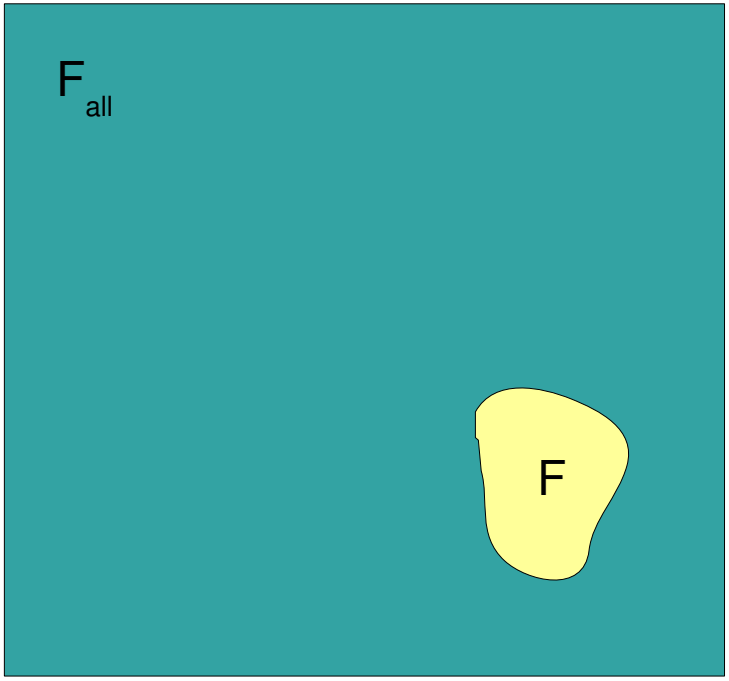
Bias-Variance Dilemma

- This dichotomy associated to the Bias-Variance Dilemma is obvious when we consider the space of possible functions to build our classifier

Let space F_{all} contain all possible classification functions (or regression functions)

We could define a strong bias, i.e., a subspace F which contains only the linear functions to perform regression

This bias reduces the variance, i.e., it reduces the number of possible classifiers we can produce



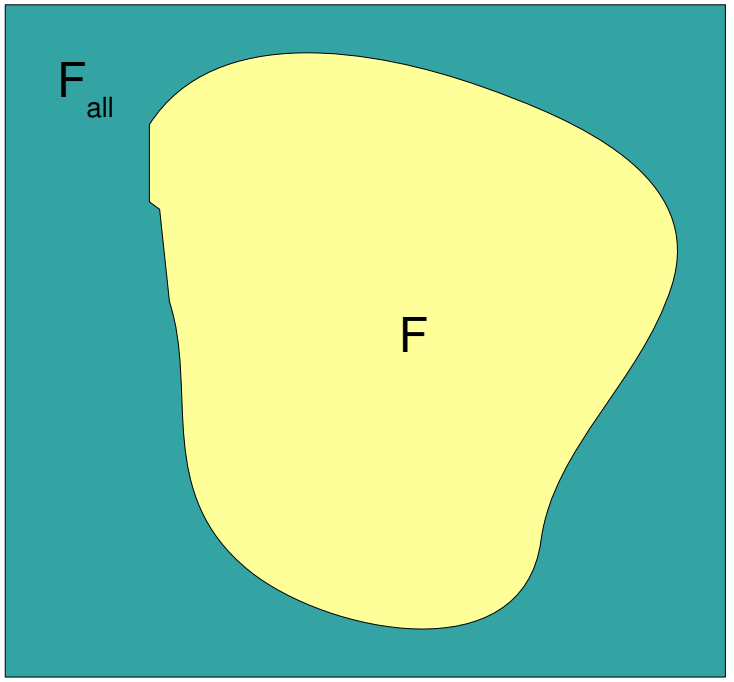
Bias-Variance Dilemma

- This dichotomy associated to the Bias-Variance Dilemma is obvious when we consider the space of possible functions to build our classifier

On the other hand, now see a small bias defined by subspace F, which contains many more functions to produce a classifier

In this case, variance is greater, i.e., the number of possible classifiers to select from and choose is huge!

What if this space contains a memory-based classifier?



To assess the quality of a Classifier,

we need to define a Loss function

Loss function

- Pay attention:
  - We need some function to compute when classifications are missed
  - Every classification technique relies on a loss function:
    - Then they may “walk” on the descent gradient to reduce training error
  - But how can we guarantee a small training error is good?
    - This is the central question for the STL
    - This is basically the Bias-Variance Dilemma

Loss function

- Pay attention:
  - We need some function to compute when classifications are missed
  - Every classification technique relies on a loss function:
    - Then they may “walk” on the descent gradient to reduce training error
  - But how can we guarantee a small training error is good?
    - This is the central question for the STL
    - This is basically the Bias-Variance Dilemma
    - **We should evaluate our classifier in terms of unseen examples**
      - **And compare the errors produced after training and testing!**

Recall some important classification algorithms

Machine Learning Review

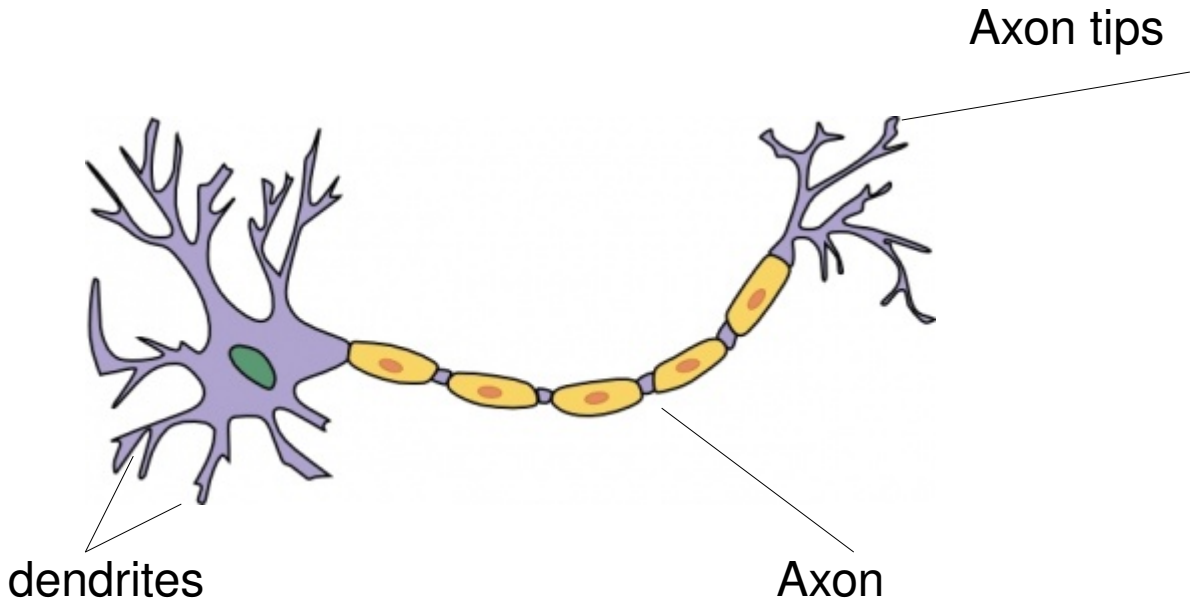
- Recall some important supervised learning algorithms:
  - Perceptron
  - Multilayer Perceptron
  - Naive Bayes

- Do they work? Do they “learn” something?
  - First of all, what is learning for them?
  - Can we obtain some sort of formalization that ensures learning for them?

Concepts on Artificial Neural Networks

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



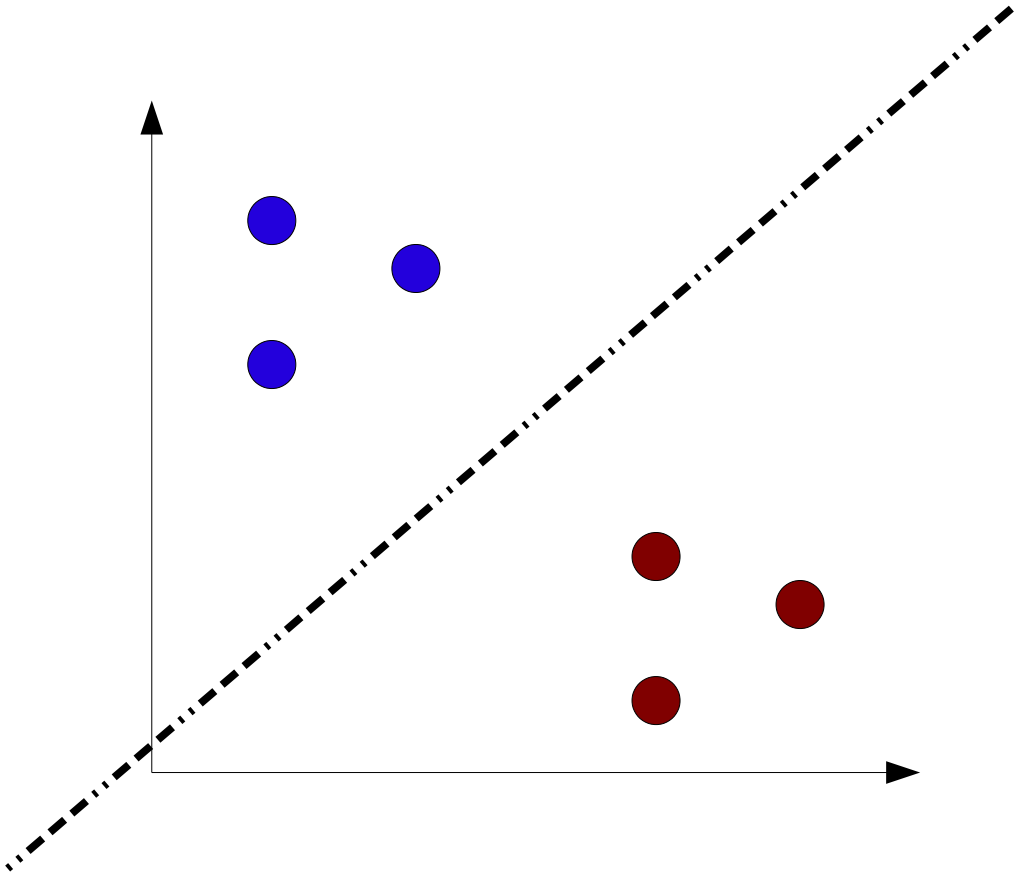
Artificial Neural Networks: History

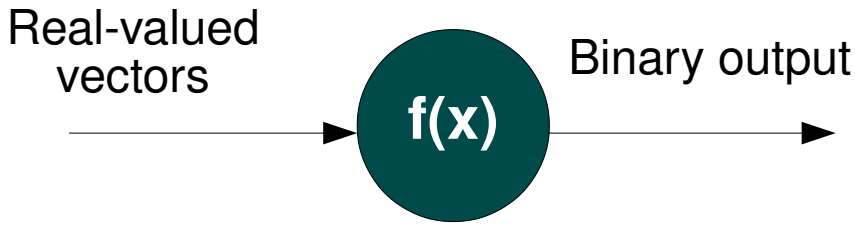
- McCullouch and Pitts (1943) proposed a computational model based on biological neural networks
  - This model was named Threshold logic

- Hebb (década de 1940), 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)

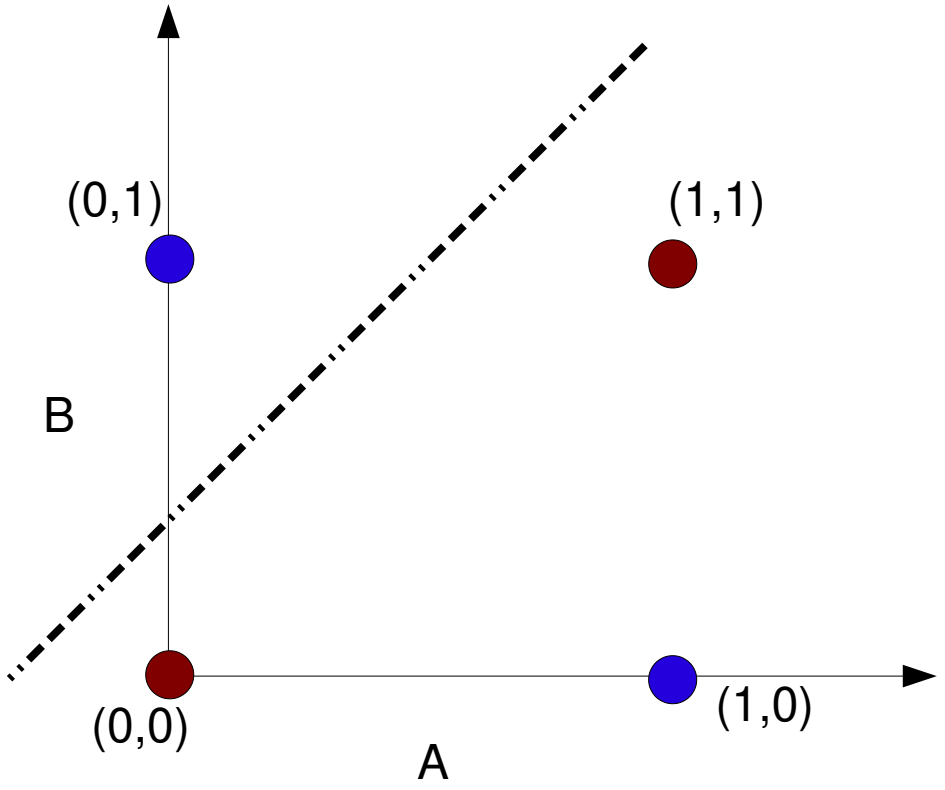
|w| = |x|

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

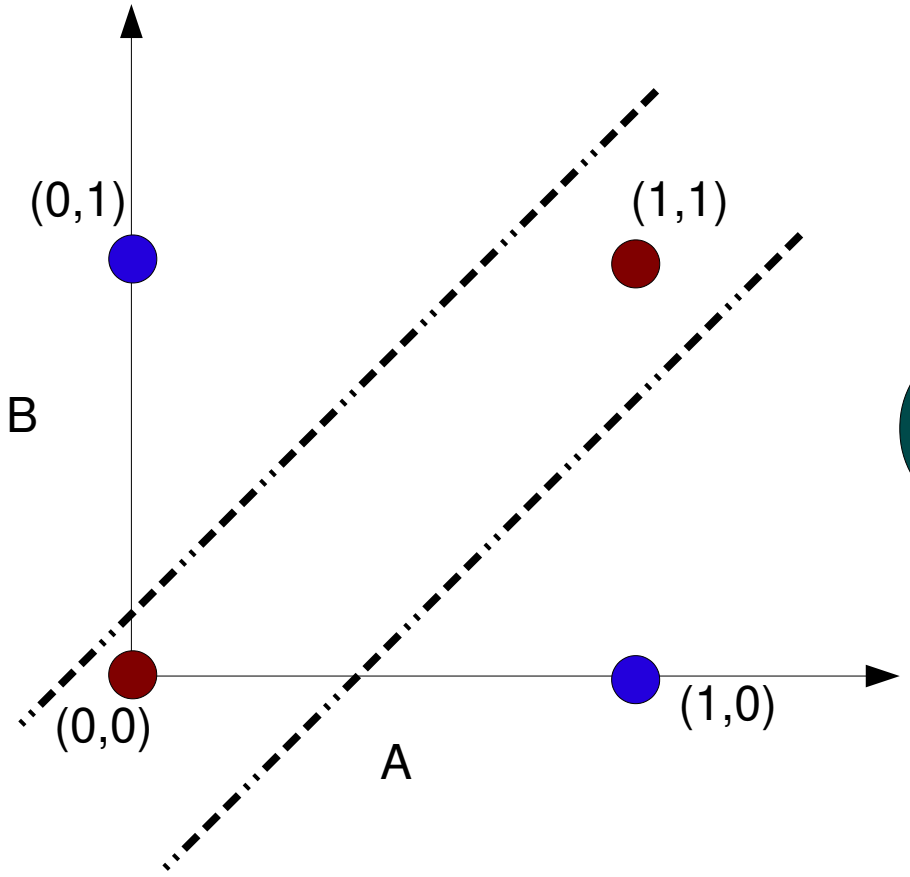


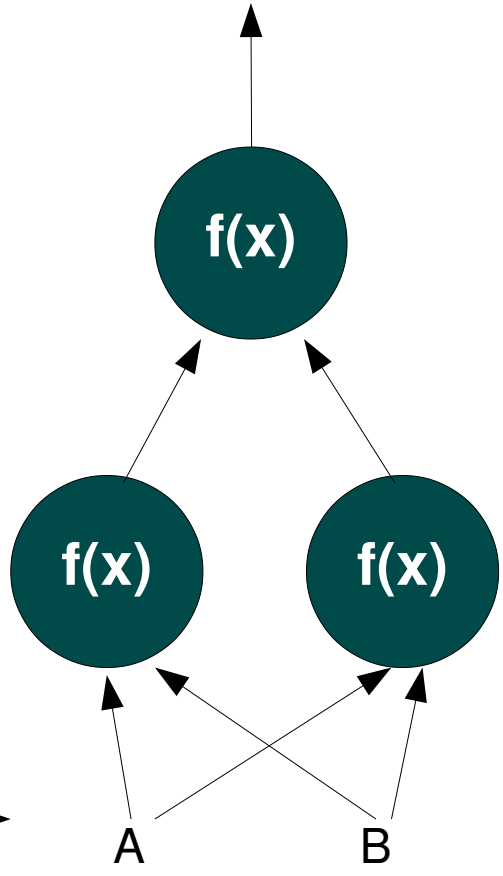
Artificial Neural Networks: History

- 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

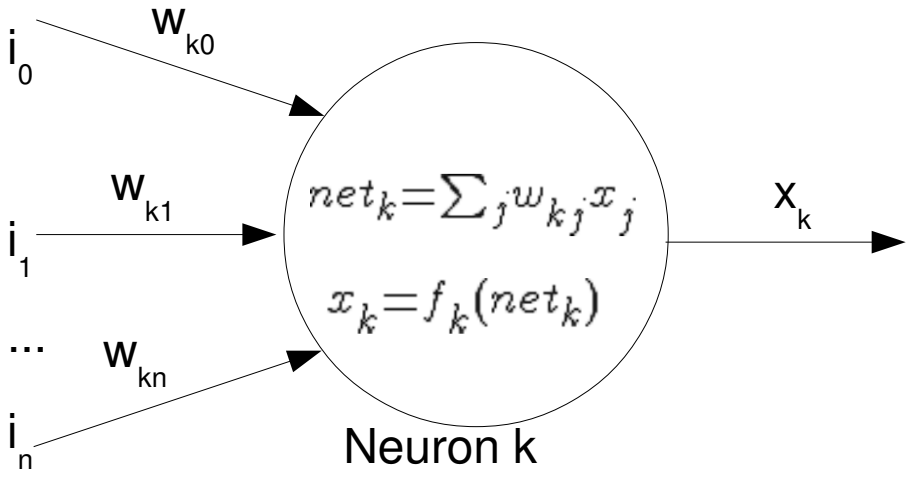
- “Rediscovery” of the Backpropagation algorithm through the paper entitled “Learning Internal Representations by Error Propagation” (1986)
  - This has motivated its adoption and usage

Artificial Neural Networks

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

General Purpose Processing Element

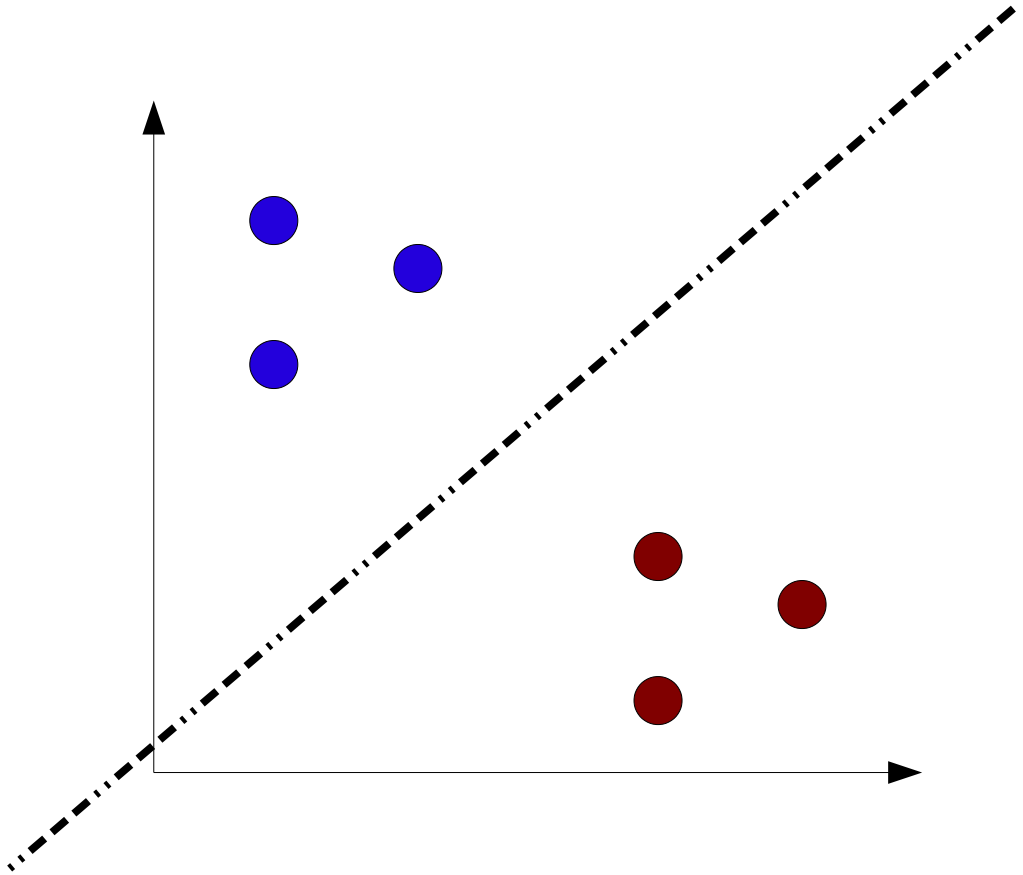
- Artificial neurons:
  - Nodes, units or processing elements
  - They can receive several values as input, but only produces one single output
  - Each connection is associated to a weight w (connection strength)
  - Learning happens by adapting weights w

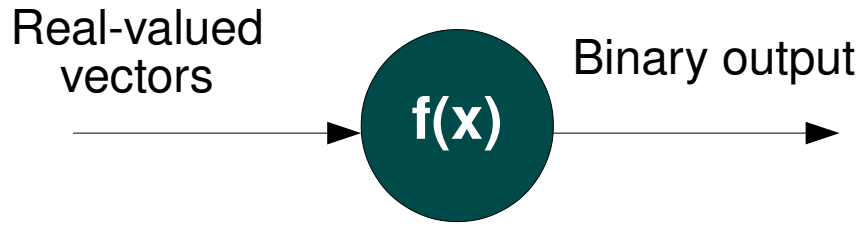


The Perceptron

The Perceptron

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



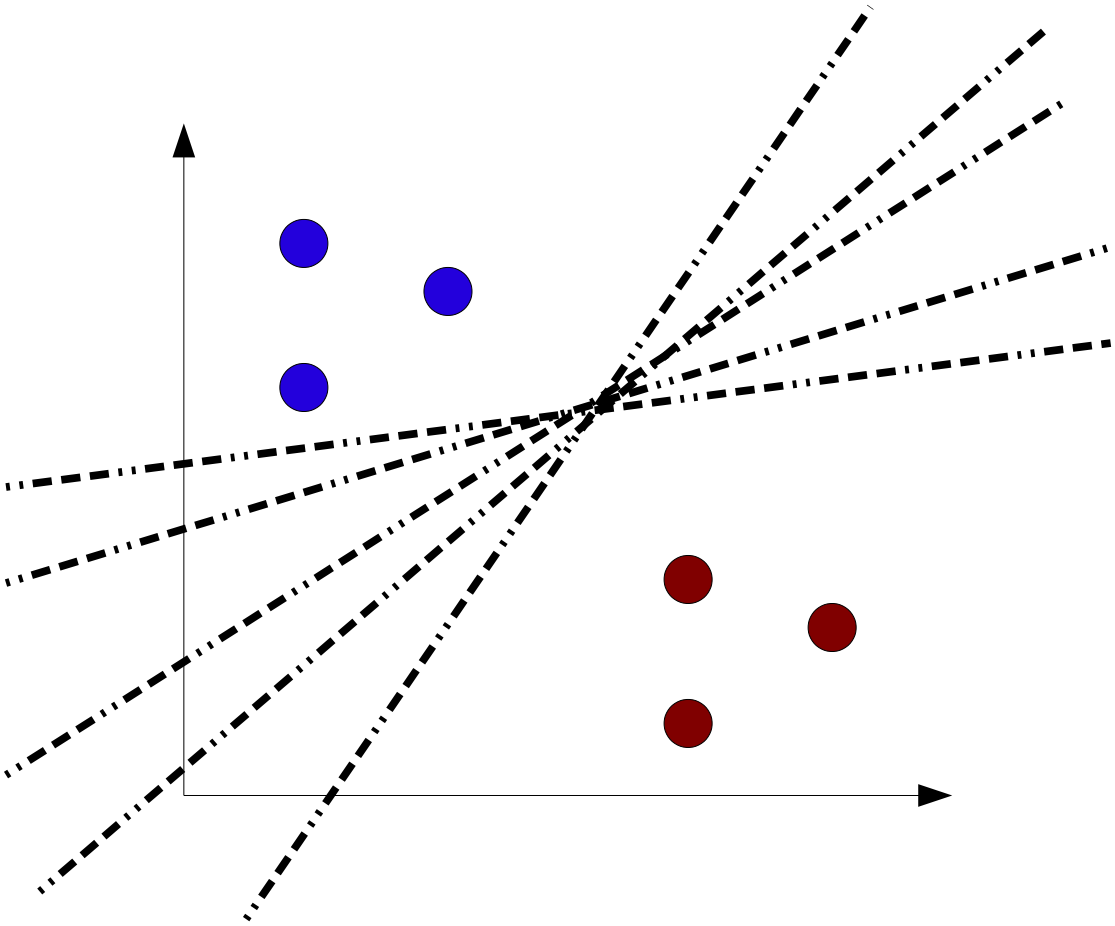


$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

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

|w| = |x|

The Perceptron

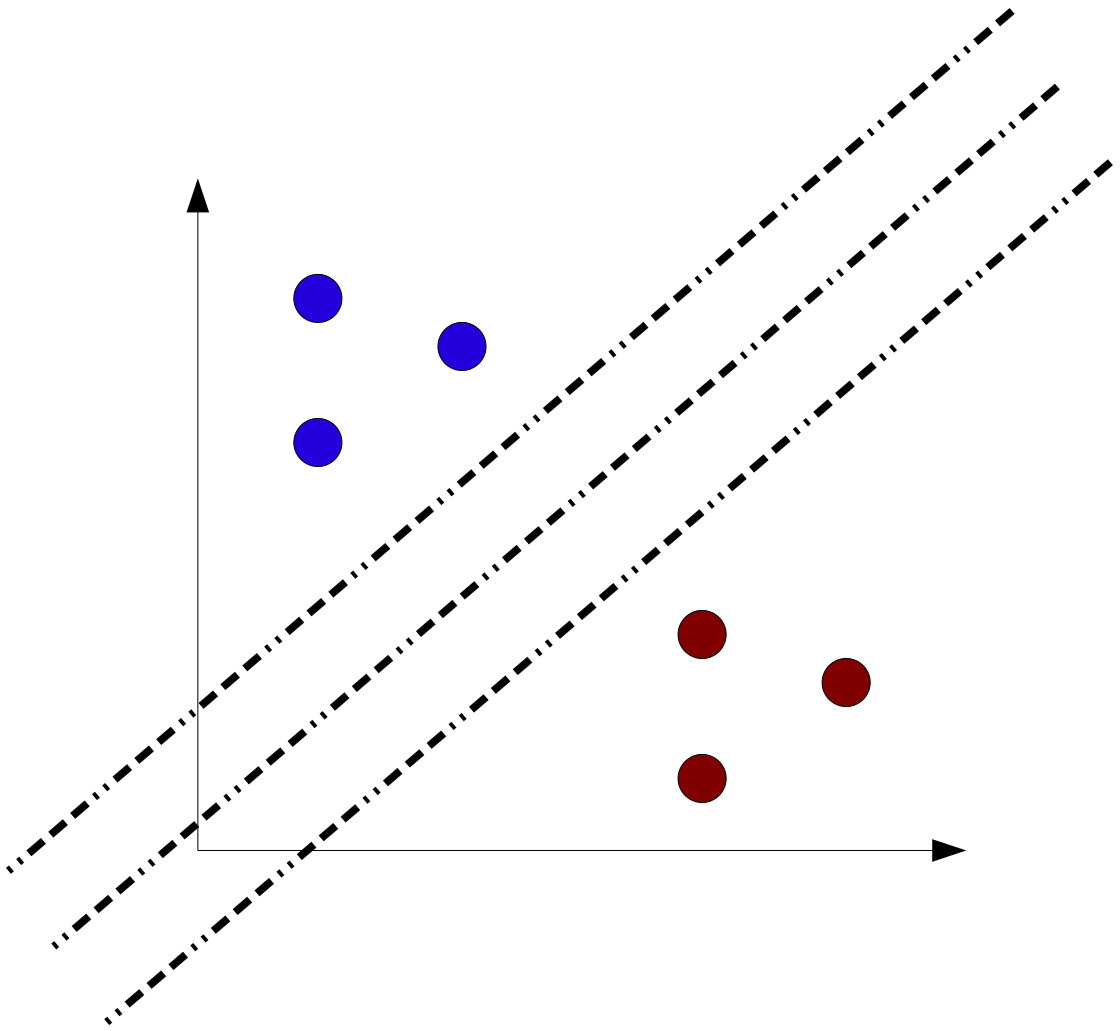


Weights w modify the slope of the line

Try gnuplot using:

pl x, 2*x, 3*x

The Perceptron



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), ..., (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

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

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

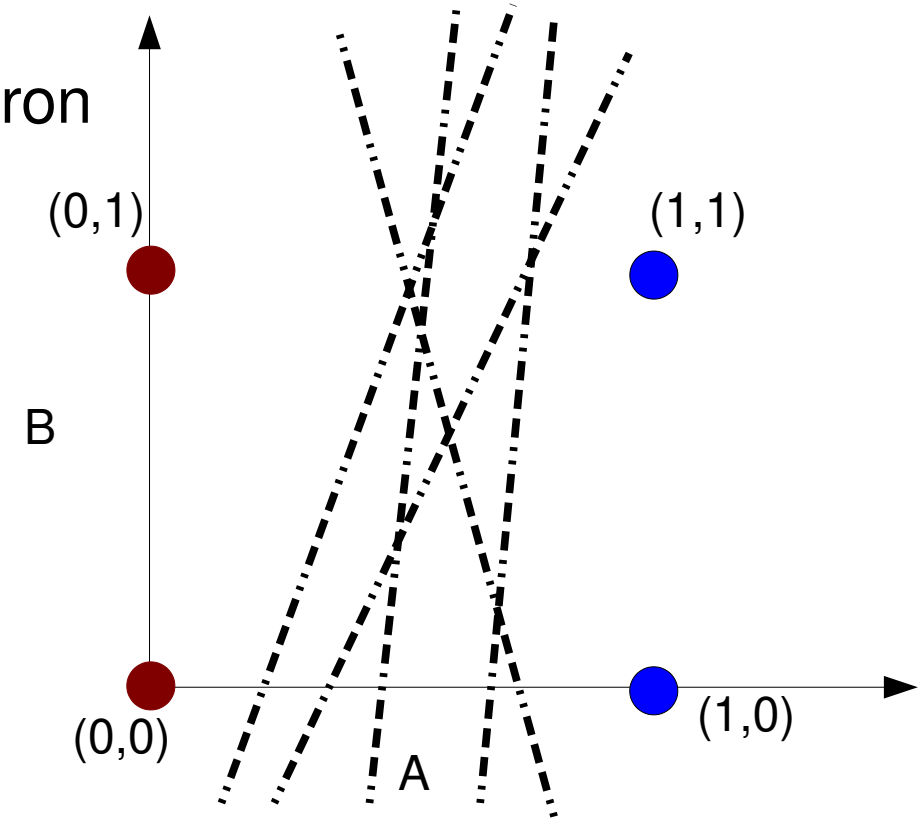
  - Execute until the error is less than a given threshold or for a number of iterations

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

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

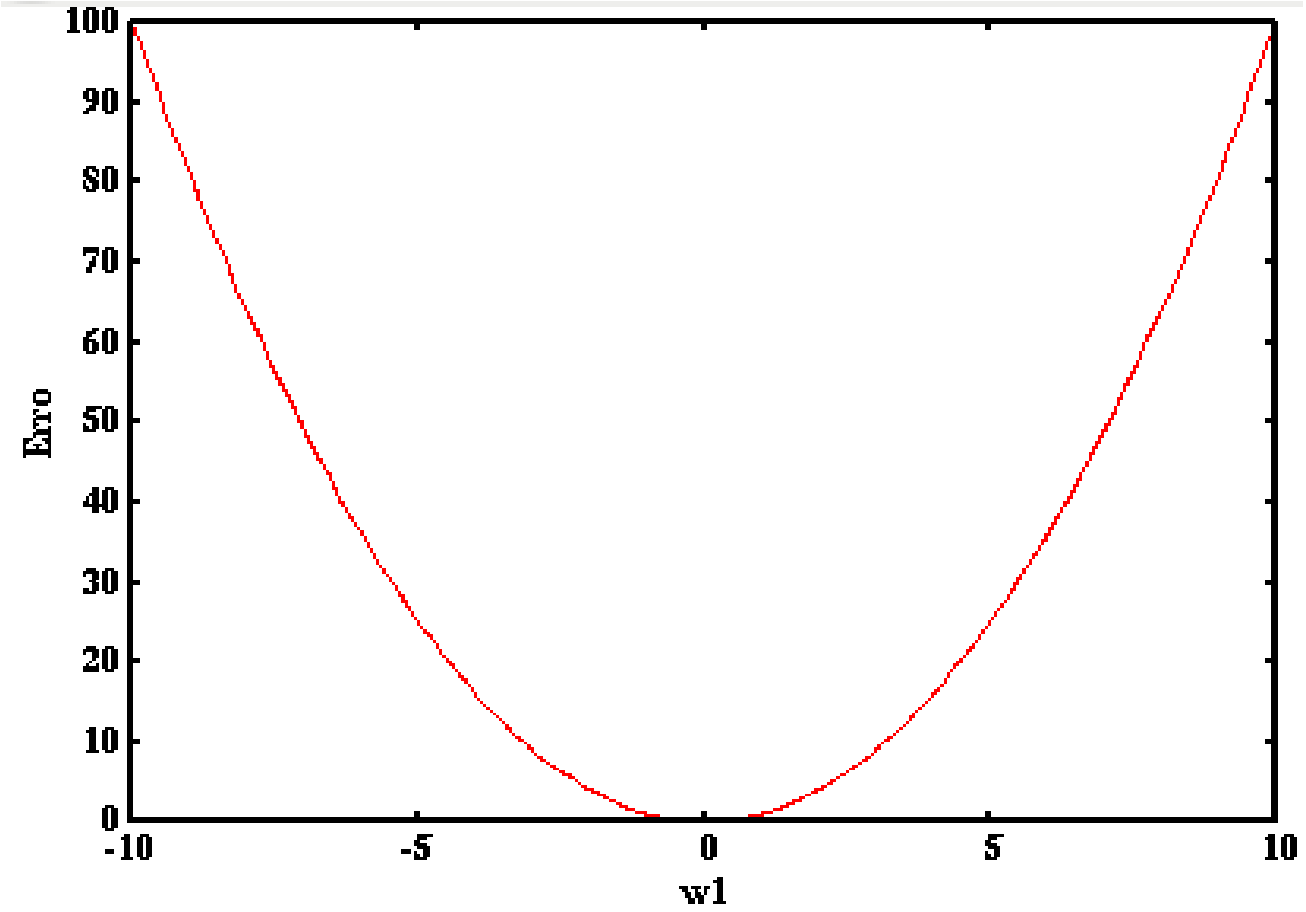
```
gnuplot> set border 4095 front linetype -1 linewidth 1.000
gnuplot> set view map
gnuplot> set isosamples 100, 100
gnuplot> unset surface
gnuplot> set style data pm3d
gnuplot> set style function pm3d
gnuplot> set ticslevel 0
gnuplot> set title "gray map"
gnuplot> set xlabel "x"
gnuplot> set xrange [ -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
```

The Perceptron

- More about the Gradient descendent method

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

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



The Perceptron

- 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

The Perceptron

- Formalize the adaptative equation

The Perceptron

- How do we get this adaptive equation?

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

  - Consider an input vector **x**
  - Consider a training set {$\mathbf{x}_0$, $\mathbf{x}_1$, ..., $\mathbf{x}_L$}
  - Consider that each **x** must produce an output value **d**
    - Thus we get {$\mathbf{d}_0$, $\mathbf{d}_1$, ..., $\mathbf{d}_L$}
  - 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:

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

The Perceptron

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

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

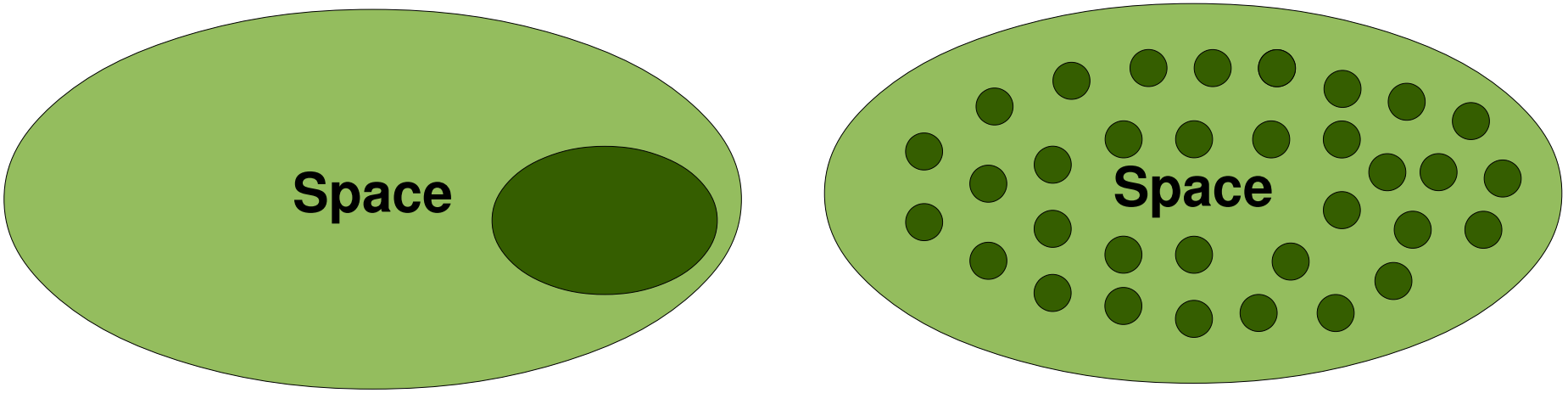
- In our scenario, we model as:

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

- 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
    - 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

The Perceptron

- 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

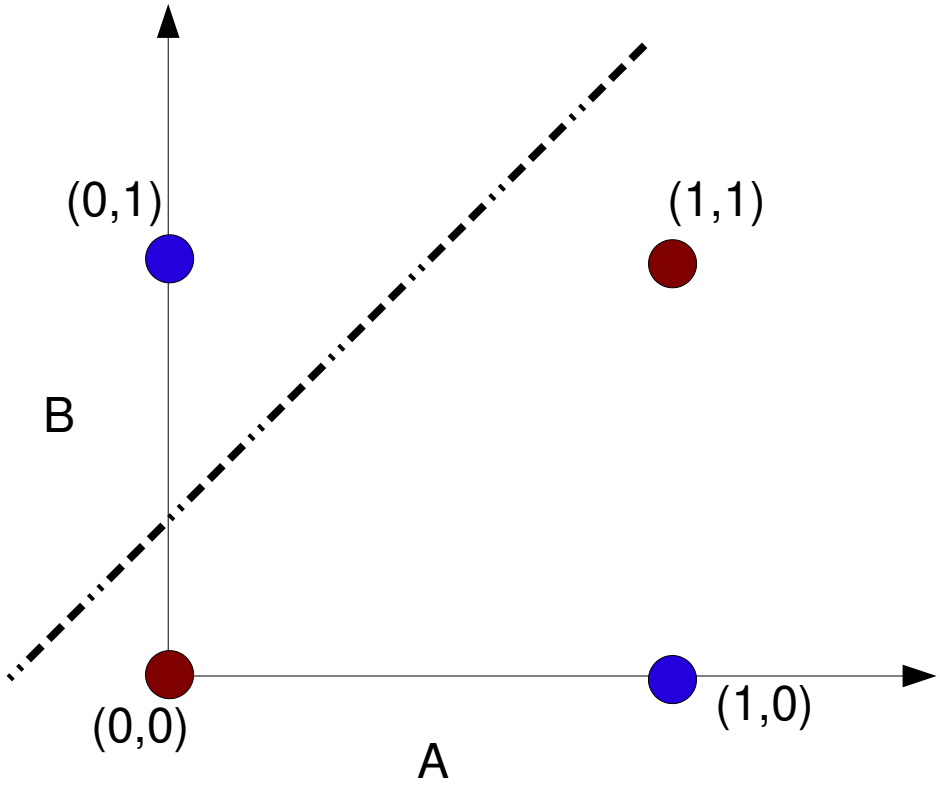
- Minsky and Papert (1969) wrote the book entitled “Perceptrons: An Introduction to Computational Geometry”, MIT
  - They demonstrated that a perceptron linearly separates classes
  - However, several problems (e.g., XOR) are not linearly separable
  - The way they wrote this book seems to question this 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

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

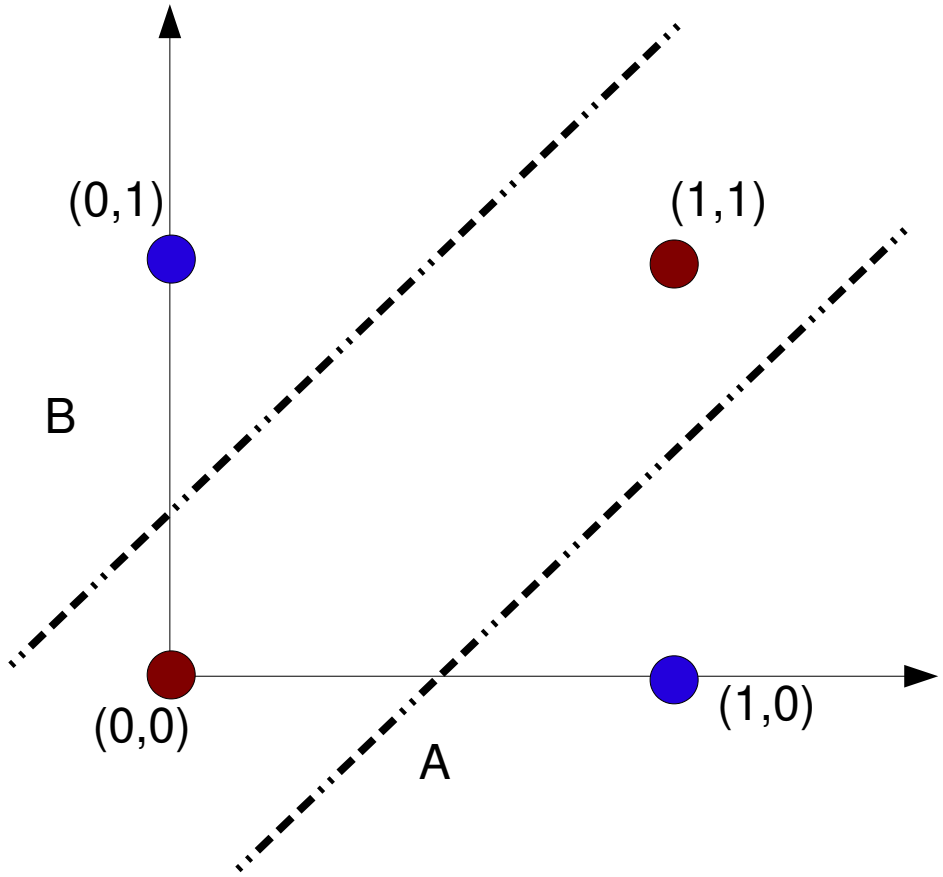
- Thus, it can only linearly separate classes
- In linearly separable problems:
  - This equation builds a hyperplane
  - Hyperplanes are (n-1)-dimensional objects used to separate n-dimensional hyperspaces in two regions

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



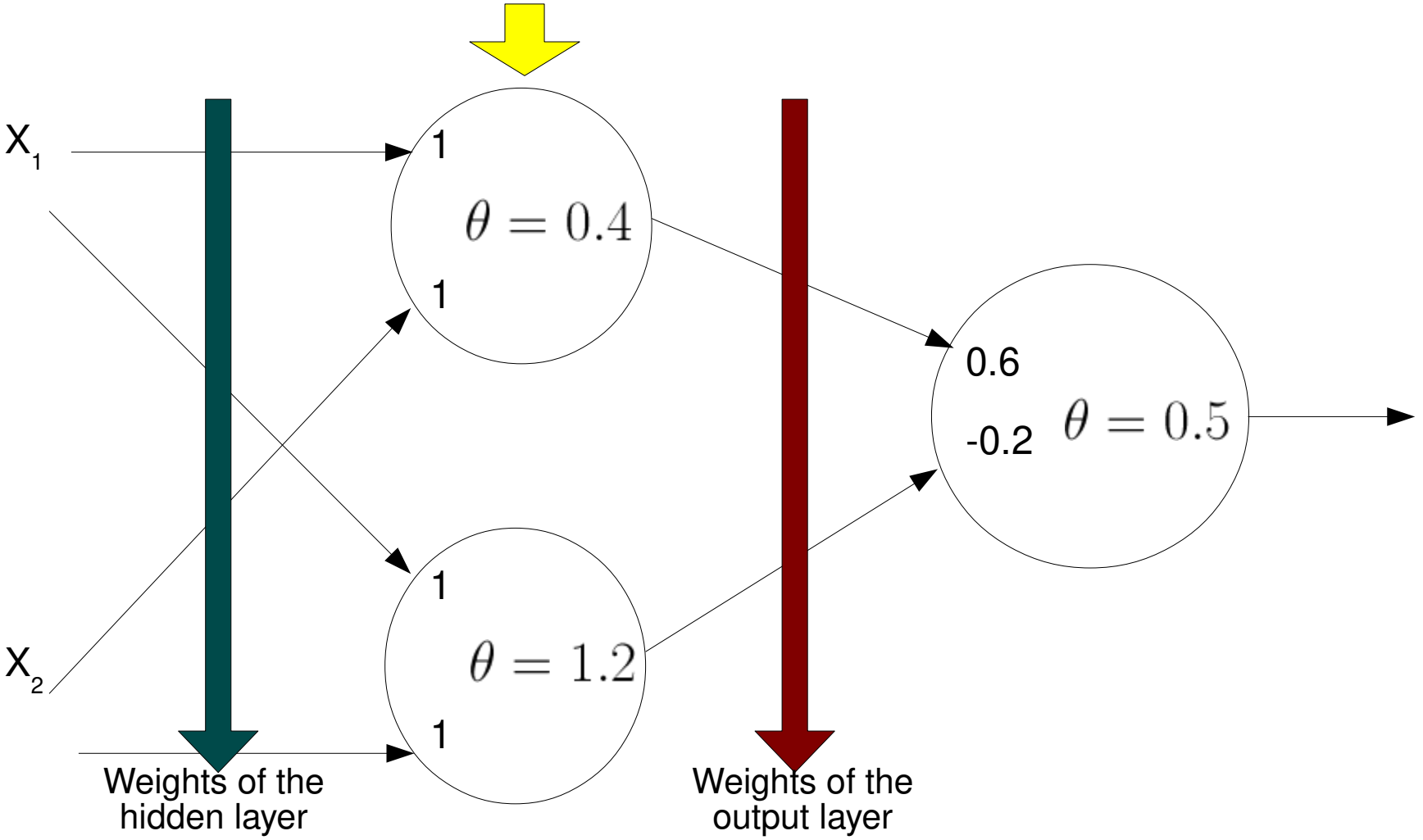
Perceptron: Solving XOR

- 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

The Multilayer Perceptron

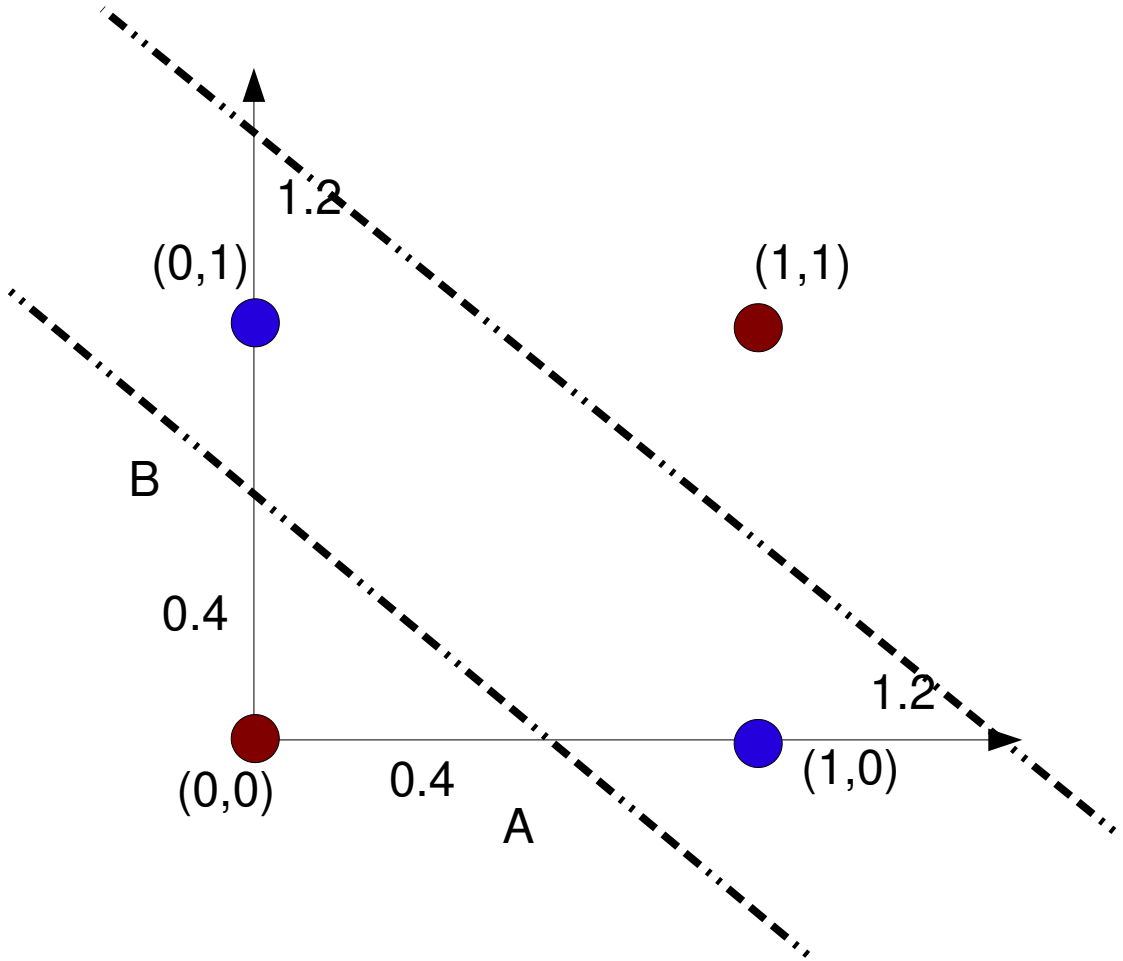
Multilayer Perceptron: Solving XOR

- Implementing XOR
  - Additional layer also called hidden layer → Multilayer Perceptron (MLP)



Multilayer Perceptron: Solving XOR

- Implementing XOR
  - Additional layer also called hidden layer
  - This result was produced by the parameters in the previous slide

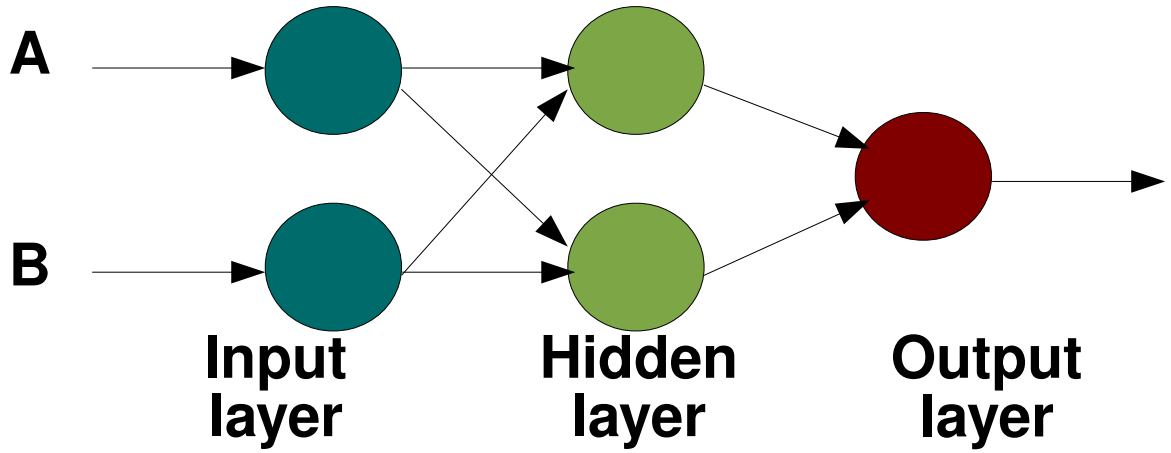


Multilayer Perceptron: Solving XOR

- Implementing XOR
  - However there is a new problem
  - How can we train this network?
    - Training means to update weights to represent input vectors and expected outputs
    - We should find a way to train weights for both layers: hidden and output ones

Multilayer Perceptron: Solving XOR

- Implementing XOR
  - Topology
    - Input length: 2 bits
    - Output length: 1 bit
    - Number of neurons in the output layer: 1
    - Number of neurons in the input layer = Input length, i.e., 2
    - Number of neurons in the hidden layer may vary



XOR Truth Table

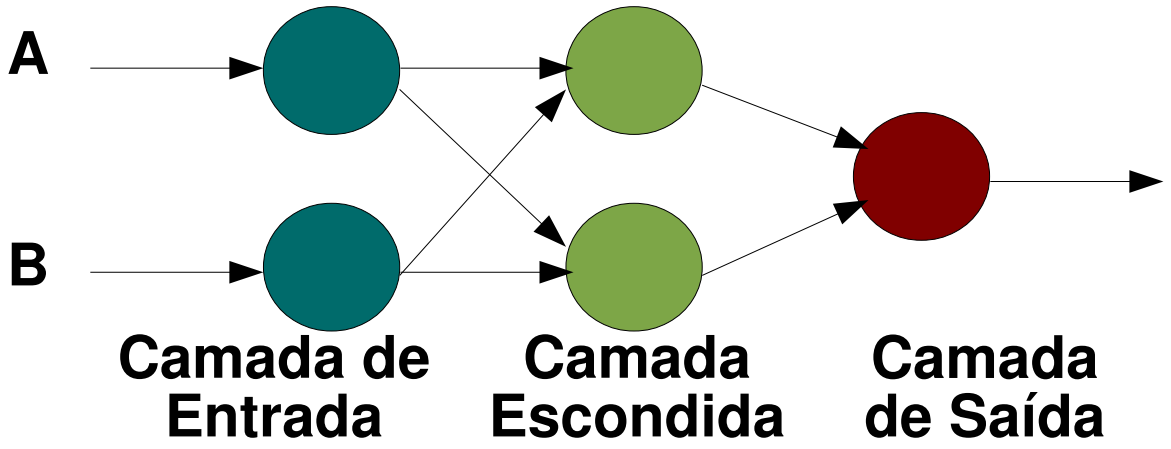
Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Multilayer Perceptron: Solving XOR

- Feedforward network
  - Inputs produce outputs
  - There is no recurrence such as for BAM and Hopfield neural networks

XOR Truth Table

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



- So we can use the Backpropagation algorithm to train MLP
  - Error is propagated from the last layer in the direction of the first and used to update weights

Multilayer Perceptron: Generalized Delta Rule

- Training (weight adaptation) occurs using the error measured at the output layer
- Learning follows the Generalized Delta Rule (GDR)
  - It is a generalization of LMS (Least Mean Squares), seen previously
  - LMS is used for linear regression (separates the space using linear functions)
    - This GDR allows non-linear regression
- Suppose:
  - Pairs of vectors (input, expected output):

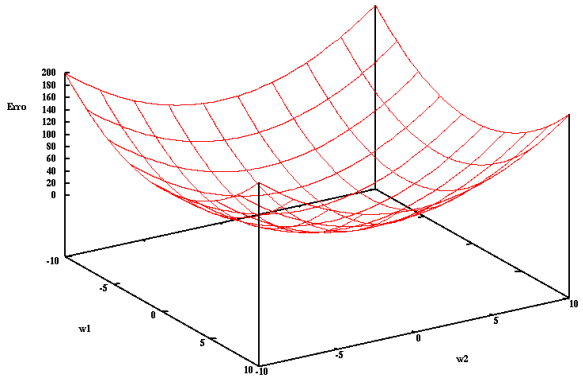
$$(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_p, \mathbf{y}_p)$$

  - Given:

$$\mathbf{y} = \phi(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^N, \mathbf{y} \in \mathbb{R}^M$$

  - The learning objective is to obtain an approximation:

$$\bar{\mathbf{y}} = \bar{\phi}(\mathbf{x})$$



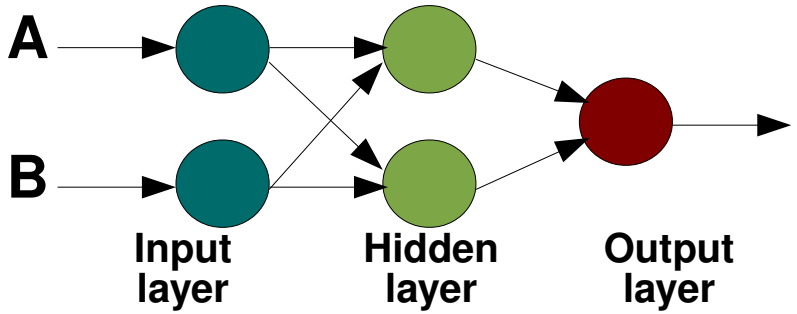
Multilayer Perceptron: Generalized Delta Rule

- The input layer is simple:
  - Neurons only forward values to the hidden layer
- The hidden layer computes:

$$\mathbf{net}_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_{j}^{h}$$

- The output layer computes:

$$\mathbf{net}_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o} i_{pj} + \theta_{k}^{o}$$

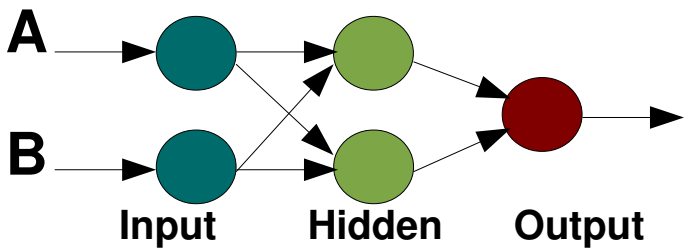


- In which:

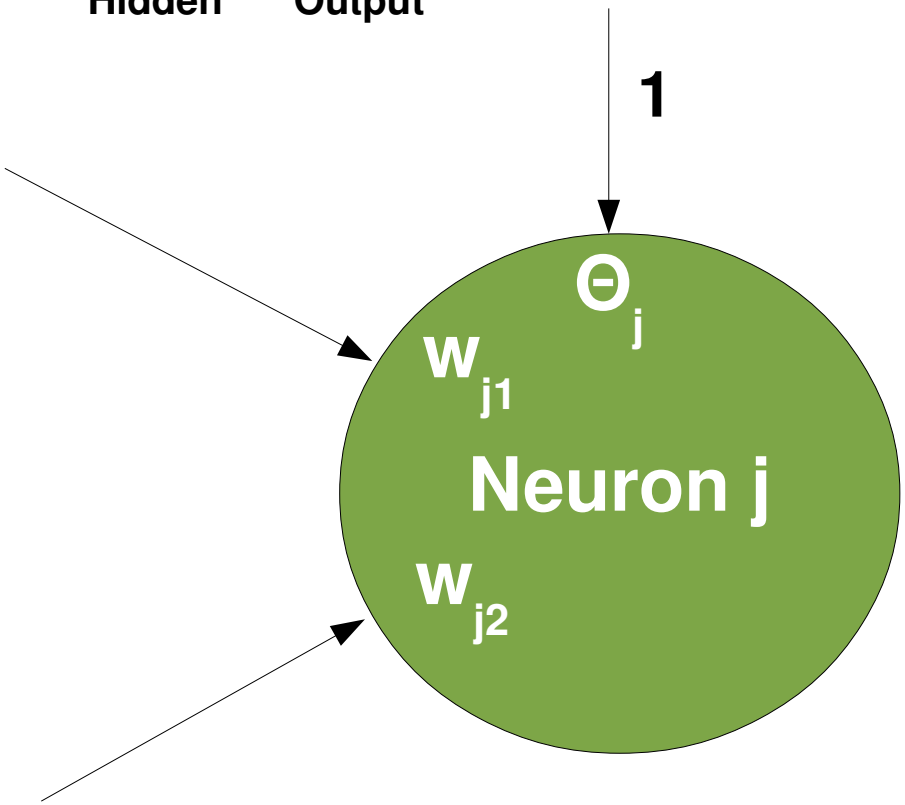
w_{ji}^{h} é o peso da conexão com o neurônio de entrada i
w_{kj}^{o} é o peso da conexão com o neurônio j da camada escondida
θ_{j}^{h} e θ_{k}^{o} são os bias

Multilayer Perceptron: Generalized Delta Rule

- For example:
  - Consider one neuron at the hidden layer



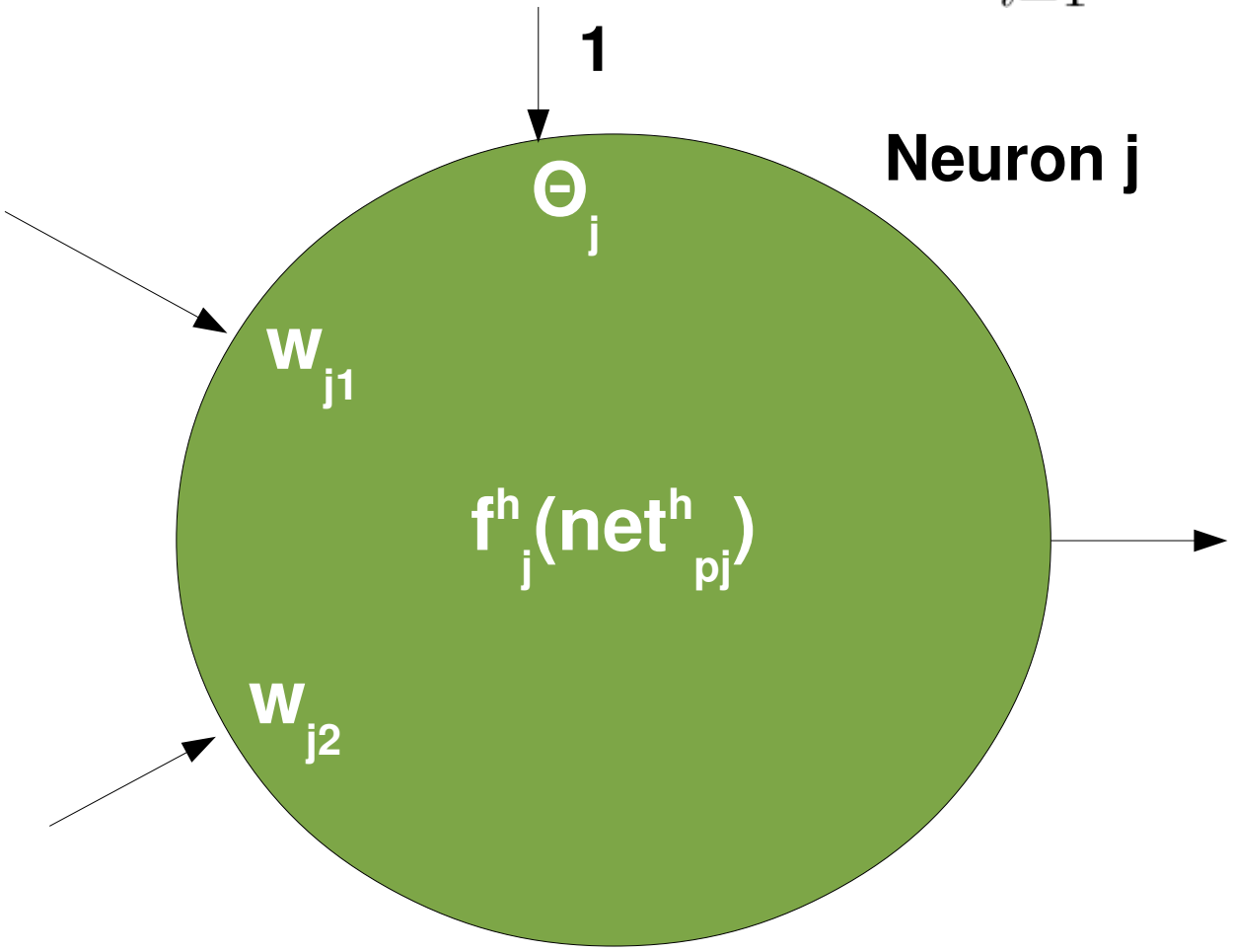
$$\mathbf{net}_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_{j}^{h}$$



Multilayer Perceptron: Generalized Delta Rule

- More details...

$$\mathbf{net}_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_{j}^{h}$$



Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- **Updating weights at the output layer**
- The output layer may contain several neurons
  - The error for a given neuron at this layer is given by:

$$\delta_{pk} = (y_{pk} - o_{pk})$$

  - Having:

y_{pk} saída esperada do neurônio k para vetor de entrada p
o_{pk} saída produzida pelo neurônio k para vetor de entrada p
p identifica o vetor de entrada usado no treinamento
k indica o neurônio da camada de saída

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- The objective is to minimize the squared sum of errors to all output units, considering an input p

$$\mathbf{E}_p = \frac{1}{2} \sum_{k=1}^{M} \delta_{pk}^2$$

  - The factor ½ is added to simplify the derivative
    - As we will have another constant to adapt weights, this term ½ does not change anything in terms of concepts, only the step
  - **M** indicates the number of neurons in the output layer
  - Important:
    - The error associated to each input is squared

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Our objective is to “walk” in the direction to reduce the error, which varies according to weights w

$$\mathbf{E}_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}^2$$

$$\delta_{pk} = (y_{pk} - o_{pk})$$

$$\mathbf{E}_p = \frac{1}{2}\sum_{k=1}^{M} (y_{pk} - o_{pk})^2$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Deriving the error in the direction of what can be changed (weights w), for that we have (according to the chain rule):

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x) \text{ ou } \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

- Simplifying:

$$E_{pk} = \frac{1}{2}(y_{pk} - o_{pk})^2 \text{ em que:}$$
$$f(g(x)) = \frac{1}{2}(y_{pk} - o_{pk})^2$$
$$g(x) = y_{pk} - o_{pk}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Thus:

$$f'(g(x)) = 2 \cdot \frac{1}{2}(y_{pk} - o_{pk})$$
$$g'(x) = 0 - o'_{pk}$$
$$\text{em que } y_{pk} \text{ é uma constante (saída esperada)}$$

- in which:

$$o_{pk} = f_k^o(\mathbf{net}_{pk}^o)$$

- Therefore the derivative will be (also following the chain rule):

$$o'_{pk} = \frac{\partial f_k^o}{\partial \mathbf{net}_{pk}^o} \cdot \frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Unifying:

$$f'(g(x))g'(x) = \left(2 \cdot \frac{1}{2}(y_{pk} - o_{pk})\right) \cdot \left(0 - \frac{\partial f_k^o}{\partial \mathbf{net}_{pk}^o} \frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o}\right)$$

- We have:

$$\frac{\partial E_{pk}}{\partial w_{kj}^o} = -(y_{pk} - o_{pk}) \frac{\partial f_k^o}{\partial \mathbf{net}_{pk}^o} \frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o}$$

- Solving the last term we find:

$$\mathbf{net} = \sum_{j=1}^{L} w_{kj}^o i_{pj} + \theta_k^o$$

$$\frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o} = \frac{\partial}{\partial w_{kj}^o} \left(\sum_{j=1}^{L} w_{kj}^o i_{pj} + \theta_k^o\right) = i_{pj}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Substituting:

$$\frac{\partial E_{pk}}{\partial w^o_{pj}} = -(y_{pk} - o_{pk}) \frac{\delta f^o_k}{\partial \mathbf{net}^o_{pk}} i_{pj}$$

- We still have to differentiate the activation function
  - So this function MUST be differentiable
  - This avoids the usage of the step function (Perceptron)
- Examples of activation functions:

1) se $f^o_k(\mathbf{net}^o_k) = \mathbf{net}^o_k$ então $f'^o_k(\mathbf{net}^o_k) = 1$
assim como $f(x) = x$ temos $f'(x) = 1$ **Linear Function**

2) se $f^o_k(\mathbf{net}^o_k) = (1 + e^{-\mathbf{net}^o_{jk}})^{-1}$ então
$f'^o_k(\mathbf{net}^o_k) = f^o_k(1 - f^o_k)$ **Sigmoid Function**

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Considering the two possibilities for the activation function, we have the weight adaptation as follows:
  - For the **linear function**:

$$f_k^o(\mathbf{net}_{jk}^o) = \mathbf{net}_{jk}^o$$

  - We have:

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})i_{pj}$$

  - For the **sigmoid function**:
  - We have:

$$f_k^o(\mathbf{net}_{jk}^o) = \frac{1}{1 + e^{-\mathbf{net}_{jk}^o}}$$

$$f_k'^o(\mathbf{net}_k^o) = f_k^o(1 - f_k^o) \text{ logo, neste cenário } f_k'^o(\mathbf{net}_k^o) = o_{pk}(1 - o_{pk})$$

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})o_{pk}(1 - o_{pk})i_{pj}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- We can define the adaptation term in a generic way, i.e., for any activation function:

$$\delta_{pk}^{o} = (y_{pk} - o_{pk}) f_k^{o\prime}(\mathbf{net}_{pk}^{o})$$

- And generalize (**Generalized Delta Rule**) the weight adaptation for any activation function, as follows:

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta \delta_{pk}^{o} i_{pj}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- **Updating hidden layer weights**
  - How can we know the expected output for each neuron at the hidden layer?
    - At the output layer we know what is expected!
  - In some way, error E_p measured at the output layer must influence in the hidden layer weights

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- The error measured at the output layer is given by:

$$
\begin{aligned}
E_p &= \frac{1}{2}\sum_k (y_{pk} - o_{pk})^2 \\
&= \frac{1}{2}\sum_k (y_{pk} - f_k^o(\mathbf{net}_{pk}^o))^2 \\
&= \frac{1}{2}\sum_k \left(y_{pk} - f_k^o\left(\sum_j w_{kj}^o i_{pj} + \theta_k^o\right)\right)^2
\end{aligned}
$$

- Term i_{pj} refers to the values produced by the previous (hidden) layer
  - So we can explore this fact to build equations to adapt hidden layer weights

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- In this manner, we define the error variation in terms of hidden layer weights:

$$\frac{\partial E_p}{\partial w_{ji}^h} = \frac{1}{2}\sum_k \frac{\partial}{\partial w_{ji}^h}(y_{pk} - o_{pk})^2$$

$$= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \mathbf{net}_{pk}^o} \frac{\partial \mathbf{net}_{pk}^o}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial \mathbf{net}_{pj}^h} \frac{\partial \mathbf{net}_{pj}^h}{\partial w_{ji}^h}$$

- From that we obtain:

$$\frac{\partial E_p}{\partial w_{ji}^h} = -\sum_k (y_{pk} - o_{pk}) f_k^{o\prime}(\mathbf{net}_{pk}^o) w_{kj}^o f_j^{h\prime}(\mathbf{net}_{pj}^h) x_{pi}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- In this manner, we define the error variation in terms of hidden layer weights:

$$\frac{\partial E_p}{\partial w_{ji}^h} = \frac{1}{2}\sum_k \frac{\partial}{\partial w_{ji}^h}(y_{pk} - o_{pk})^2$$

$$= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \mathbf{net}_{pk}^o} \frac{\partial \mathbf{net}_{pk}^o}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial \mathbf{net}_{pj}^h} \frac{\partial \mathbf{net}_{pj}^h}{\partial w_{ji}^h}$$

Derivative of the activation function at the output layer in the direction of net

Derivative of the activation function at the hidden layer in the direction of net

- From that we obtain:

$$\frac{\partial E_p}{\partial w_{ji}^h} = -\sum_k (y_{pk} - o_{pk}) f_k^{o\prime}(\mathbf{net}_{pk}^o) w_{kj}^o f_j^{h\prime}(\mathbf{net}_{pj}^h) x_{pi}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- Having:

$$\mathbf{net}^o_{pk} = \sum_{j=1}^{L} w^o_{kj} i_{pj} + \theta^o_k$$

$$\mathbf{net}^h_{pi} = \sum_{j=1}^{L} w^h_{kj} x_{pi} + \theta^h_k$$

$$\frac{\partial E_p}{\partial w^h_{ji}} = \frac{1}{2} \sum_k \frac{\partial}{\partial w^h_{ji}} (y_{pk} - o_{pk})^2$$

$$= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \mathbf{net}^o_{pk}} \frac{\partial \mathbf{net}^o_{pk}}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial \mathbf{net}^h_{pj}} \frac{\partial \mathbf{net}^h_{pj}}{\partial w^h_{ji}}$$

$$\frac{\partial E_p}{\partial w^h_{ji}} = -\sum_k (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk}) w^o_{kj} f^{h\prime}_j(\mathbf{net}^h_{pj}) x_{pi}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- So we can compute the weight adaptation for the hidden layer in form:

$$\triangle_p w^h_{ji} = \eta f^{h\prime}_j(\mathbf{net}^h_{pj}) x_{pi} \sum_k (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk}) w^o_{kj}$$

$$\triangle_p w^h_{ji} = \eta f^{h\prime}_j(\mathbf{net}^h_{pj}) x_{pi} \sum_k \delta^o_{pk} w^o_{kj}$$

- In this manner, the weight adaptation at the hidden layer depends on the error at the output layer
  - The term **Backpropagation** comes from this notion of dependency on the error at the output layer

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- So we can compute the term delta for the hidden layer in the same way we did for the output layer:

$$\delta_{pj}^{h} = f_{j}^{h\prime}(\mathbf{net}_{pj}^{h}) \sum_{k} \delta_{pk}^{o} w_{kj}^{o}$$

- In this way, the weight adaptation is given by:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta \delta_{pj}^{h} x_{i}$$

- Finally, we can implement the MLP learning algorithm

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

- Output layer:

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- Hidden layer:

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

This is very important!!!

First, we MUST compute all deltas so then we update weights!!!

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

- Output layer:

k identifies the neuron

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- Hidden layer:

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

- Output layer:

This refers to the layer (hidden or output)

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- Hidden layer:

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

- Output layer:

p identifies the input vector

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- Hidden layer:

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

Multilayer Perceptron

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

- Output layer:

Input values produced by the hidden layer

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- Hidden layer:

Input values produced by the input layer

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

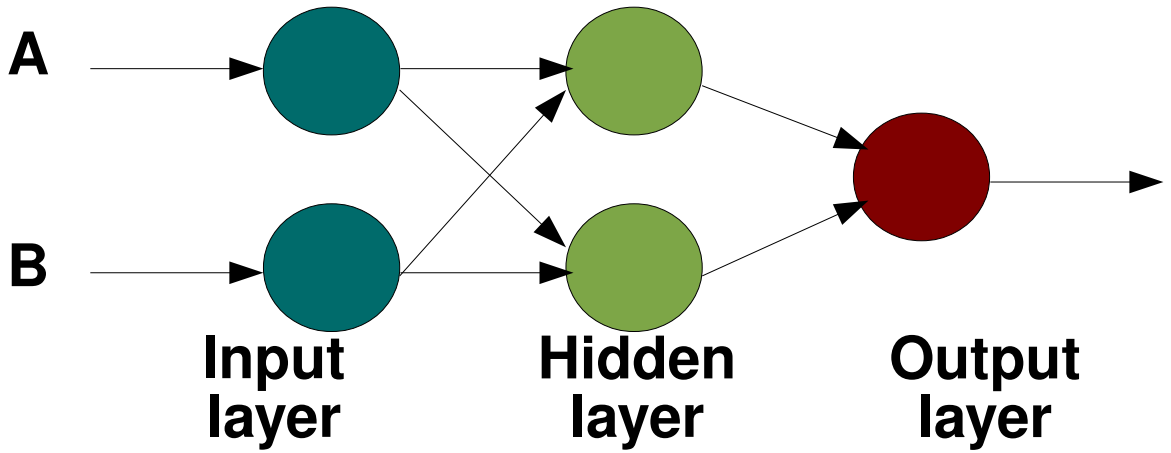
$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

Multilayer Perceptron

- Implementing
  - XOR

XOR Truth Table

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

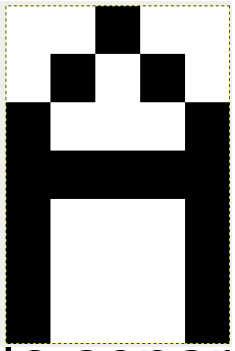


Multilayer Perceptron

- Implementing
  - OCR (Optical Character Recognition)
  - Solution:
    - We can build a Hash Table (Bits → ASCII Code)
      - Problem: when one or more bits are random or present noise
        - How can we deal with noise in such scenario?
    - MLP
      - Capable of learning and generalize knowledge
        - Even in the presence of noise, it will produce good results

Multilayer Perceptron

- Implementing
  - OCR (Optical Character Recognition)
  - Input is given by a 7x5 matrix
    - Example (separate letter A from letter B)



  - Extend this example to separate any character
  - Extend this example for any classification problem

Multilayer Perceptron

- Implementing
  - Face recognition
  - Song classification according to genre

- Consider other datasets:
  - UCI (http://archive.ics.uci.edu/ml/datasets.html):
    - Iris
    - Reuters-21578 Text Categorization Collection
    - CMU Face Images
  - Million Song Dataset
    - http://labrosa.ee.columbia.edu/millionsong/pages/getting-dataset

The Bayesian Learning

First step

- What is **Conditional Probability?**
  - The probability an event A occurs given event B has happened

$$P(A|B)$$

  - We represent it in form:
  - Or:

$$P(A \mid B) = \frac{P(B \cap A)}{P(B)}$$

$$P(B \cap A) = P(A \mid B)P(B)$$

For example

- Conditional Probability:
  - When there is dependency:
    - In Bayesian Learning, we assume that one attribute depends on the values of another (or others)
  - For example:

Day	Outlook	Temperature	Moisture	Wind	Play Tennis
D1	Sunny	Warm	High	Weak	No
D2	Sunny	Warm	High	Strong	No
D3	Cloudy	Warm	High	Weak	Yes
D4	Rainy	Pleasant	High	Weak	Yes
D5	Rainy	Cold	Normal	Weak	Yes
D6	Rainy	Cold	Normal	Strong	No
D7	Cloudy	Cold	Normal	Strong	Yes
D8	Sunny	Pleasant	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rainy	Pleasant	Normal	Weak	Yes
D11	Sunny	Pleasant	Normal	Strong	Yes
D12	Cloudy	Pleasant	High	Strong	Yes
D13	Cloudy	Warm	Normal	Weak	Yes
D14	Rainy	Pleasant	High	Strong	No

For example

- Conditional Probability:
  - What is the probability that an event A occurs given B?

$$P(A \mid B) = \frac{P(B \cap A)}{P(B)}$$

  - Assume B equals to Moisture = High

Day	Outlook	Temperature	Moisture	Wind	Play Tennis
D1	Sunny	Warm	High	Weak	No
D2	Sunny	Warm	High	Strong	No
D3	Cloudy	Warm	High	Weak	Yes
D4	Rainy	Pleasant	High	Weak	Yes
D5	Rainy	Cold	Normal	Weak	Yes
D6	Rainy	Cold	Normal	Strong	No
D7	Cloudy	Cold	Normal	Strong	Yes
D8	Sunny	Pleasant	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rainy	Pleasant	Normal	Weak	Yes
D11	Sunny	Pleasant	Normal	Strong	Yes
D12	Cloudy	Pleasant	High	Strong	Yes
D13	Cloudy	Warm	Normal	Weak	Yes
D14	Rainy	Pleasant	High	Strong	No

For example

- Conditional Probability:
  - Two possible values can be computed for A:
    - Play Tennis = Yes
    - Play Tennis = No
  - Therefore we have:

$$P(A \mid B) = \frac{P(B \cap A)}{P(B)}$$

$$P(Jogar\ Tênis = Sim | Umidade = Alta) = \frac{P(Jogar\ Tênis = Sim \cap Umidade = Alta)}{P(Umidade = Alta)}$$

$$P(Jogar\ Tênis = Não | Umidade = Alta) = \frac{P(Jogar\ Tênis = Não \cap Umidade = Alta)}{P(Umidade = Alta)}$$

For example

- Conditional Probability:
  - What are the probabilities?

$$P(Umidade = Alta) = \frac{7}{14} = 0.5$$

$$P(Jogar\ Tênis = Sim \cap Umidade = Alta) = \frac{3}{14} = 0.214$$

$$P(Jogar\ Tênis = Não \cap Umidade = Alta) = \frac{4}{14} = 0.286$$

  - Thus:

$$P(Jogar\ Tênis = Sim | Umidade = Alta) = \frac{\frac{3}{14}}{\frac{7}{14}} = 0.428$$

$$P(Jogar\ Tênis = Não | Umidade = Alta) = \frac{\frac{4}{14}}{\frac{7}{14}} = 0.571$$

For example

- Conditional Probability:
  - Conclusion:
    - Knowing the moisture is high, we can infer the probabilities:
      - Play Tennis = Yes is equal to 42.8%
      - Play Tennis = No is equal to 57.1%

The Bayes Theorem

Bayes Theorem

- Bayes Theorem:

$$P(A|B) = \frac{P(B \cap A)}{P(B)}$$

Assim: $P(B \cap A) = P(A|B) \cdot P(B)$

Como $P(B \cap A) = P(A \cap B)$ logo:

$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$$

e chegamos ao Teorema de Bayes: $P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$

Bayes Theorem

- In machine learning, we wish:
  - The best hipothesis h_{MAP} contained in space H given we have an observable training set D
    - Thus proceeding with the substitution in:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

    - We have (for a hipothesis h in H):

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Bayes Theorem

- However, to obtain h_{MAP} we need to compute:

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

- We refer to h_{MAP} as the hipothesis with the Maximum A Posteriori Probability (MAP)
  - i.e., the hypothesis that produces best results to unseen examples given the training on set D

Bayes Theorem: Example

- Consider the following medical diagnosis in which there are two alternatives:
  - Patient has the disease W
  - Patient does not have the disease W
- Consider the patient was blood tested

Bayes Theorem: Example

- Consider the following medical diagnosis in which there are two alternatives:
  - Patient has the disease W
  - Patient does not have the disease W
- Consider the patient was blood tested
- And that we know:
  - In the World population, the probability one has to develop this disease is 0.008
  - Consider this blood test:
    - Presents a true positive for this disease in 98% of situations
    - And a true negative in 97% of situations

Bayes Theorem: Example

- We can summarize this scenario as follows:

$$P(W) = 0.008 \qquad P(\neg W) = 0.992$$

$$P(\oplus|W) = 0.98 \qquad P(\ominus|W) = 0.02$$

$$P(\oplus|\neg W) = 0.03 \qquad P(\ominus|\neg W) = 0.97$$

- Suppose the test was positive for a patient:
  - Should we diagnose him/her as having the disease W?

Bayes Theorem: Example

- Our hipothesis in this case is:
  - “Is the patient a carrier of disease W?”
- We then look for the **maximum a posteriori probability**
  - i.e., what is the maximum probability to answer such hipothesis using the data we have, and considering data from this patient we have never seen?
- We know:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

$$P(W) = 0.008 \quad P(\neg W) = 0.992$$
$$P(\oplus|W) = 0.98 \quad P(\ominus|W) = 0.02$$
$$P(\oplus|\neg W) = 0.03 \quad P(\ominus|\neg W) = 0.97$$

Bayes Theorem: Example

- From:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- We compute:

$$P(W|\oplus) = \frac{P(\oplus|W)P(W)}{P(\oplus)}$$

Is the patient a carrier knowing the test was positive?

$$P(\neg W|\oplus) = \frac{P(\oplus|\neg W)P(\neg W)}{P(\oplus)}$$

Is not the patient a carrier knowing the test was positive?

Bayes Theorem: Example

- The hipothesis in this case is:
  - “Is the patient a carrier of disease W?”
- We then look for the maximum a posteriori probability
  - i.e., what is the maximum probability to answer such hipothesis using the data we have, and considering data from this patient we have never seen?
- We know:

$$\begin{array}{ll} P(W) = 0.008 & P(\neg W) = 0.992 \\ P(\oplus|W) = 0.98 & P(\ominus|W) = 0.02 \\ P(\oplus|\neg W) = 0.03 & P(\ominus|\neg W) = 0.97 \end{array}$$

- Thus:

$$P(\oplus|W)P(W) = 0.98 \cdot 0.008 = 0.0078$$
$$P(\oplus|\neg W)P(\neg W) = 0.03 \cdot 0.992 = 0.0298$$

Bayes Theorem: Example

- We do not have access to $P(\oplus)$, but as the probability of both sum up to 1 (and there are only two possibilities), we can normalize as follows:

$$P(\oplus|W)P(W) = 0.98 \cdot 0.008 = 0.0078$$
$$P(\oplus|\neg W)P(\neg W) = 0.03 \cdot 0.992 = 0.0298$$

- And find:

$$P(W|\oplus) = \frac{0.0078}{0.0078 + 0.0298} = 0.207$$
$$P(\neg W|\oplus) = \frac{0.0298}{0.0078 + 0.0298} = 0.793$$

The maximum probability is that the patient is not a carrier of W, besides the result of the blood test is positive!

Bayes Theorem: Example

- Now we can compute $P(\oplus)$:

$$P(W|\oplus) = \frac{P(\oplus|W)P(W)}{P(\oplus)}$$

$$P(\oplus) = \frac{P(\oplus|W)P(W)}{P(W|\oplus)}$$

  - Which summarizes the probability of the collected data is positive over all the data set

$$P(\oplus) = \frac{P(\oplus|W)P(W)}{P(W|\oplus)} = \frac{0.98 \cdot 0.008}{0.207} = \frac{0.03 \cdot 0.992}{0.793} = 0.038$$

Bayes Theorem: Example

- Conclusions:
  - This disease is so rare that we need a better test (with greatest values for the true positive and negative), for example:
    - Consider another test that has a true positive rate of 99.5%
    - Consider this test that has the false positive rate of 99.9%
  - So, we would have:

$$P(W) = 0.008 \quad P(\neg W) = 0.992$$
$$P(\oplus|W) = 0.995 \quad P(\ominus|W) = 0.005$$
$$P(\oplus|\neg W) = 0.001 \quad P(\ominus|\neg W) = 0.999$$

Now yes!

$$P(\oplus|W)P(W) = 0.995 \cdot 0.008 = 0.00796$$
$$P(\oplus|\neg W)P(\neg W) = 0.001 \cdot 0.992 = 0.000992$$

$$P(W|\oplus) = 0.89$$
$$P(\neg W|\oplus) = 0.11$$

The Bayes Optimal Classifier

Bayes Optimal Classifier

- Consider a hypothesis space H containing only h_1, h_2 and h_3
  - Let the a posteriori probability of those hipothesis be equal to:
    - $P(h_1|D) = 0.4$
    - $P(h_2|D) = 0.3$
    - $P(h_3|D) = 0.3$
  - In this scenario, $h_{MAP} = h_1$
    - i.e., this is the hipothesis with the maximum a posteriori probability
    - If we have the chance of choosing only one, this is the best!

Bayes Optimal Classifier

- Now consider that a new example was classified as positive by h_1 and negative for both h_2 and h_3

  - So, if we consider all hipotheses, we have:
    - Probability this new example to be defined as positive is 0.4
    - Probability this new example to be defined as negative is 0.6

  - So the classification given by the MAP hypothesis is different from the one given by all hypotheses together
    - So, h_{MAP} **DOES NOT** answer correctly every time!
    - So hypotheses voting tends to be a better option
      - Ensembles of classifiers
        - But what about the biases of classification algorithms?

Bayes Optimal Classifier

- So, we have the **Bayes Optimal Classification** which takes the most common opinion of all classifiers into account:
  - It is optimal because it considers ALL hypotheses in space H

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

  - But what if H contains infinite hypotheses?

Bayes Optimal Classifier

- Let a new example classified as positive or negative according to one of the hypotheses:

$$P(h_1|D) = 0.4, \boxed{P(\ominus|h_1) = 0, P(\oplus|h_1) = 1}$$
$$P(h_2|D) = 0.3, \boxed{P(\ominus|h_2) = 1, P(\oplus|h_2) = 0}$$
$$P(h_3|D) = 0.3, \boxed{P(\ominus|h_3) = 1, P(\oplus|h_3) = 0}$$

Classification Results

Bayes Optimal Classifier

- Let a new example classified as positive or negative according to one of the hypotheses:

$$P(h_1|D) = 0.4, P(\ominus|h_1) = 0, P(\oplus|h_1) = 1$$
$$P(h_2|D) = 0.3, P(\ominus|h_2) = 1, P(\oplus|h_2) = 0$$
$$P(h_3|D) = 0.3, P(\ominus|h_3) = 1, P(\oplus|h_3) = 0$$

Classified as positive by hypothesis h1

Classified as negative by hypotheses h2 and h3

$$\sum_{h_i \in H} P(\oplus|h_i)P(h_i|D) = 0.4$$

$$\sum_{h_i \in H} P(\ominus|h_i)P(h_i|D) = 0.6$$

$$\arg \max_{v_j \in \{\oplus, \ominus\}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = \ominus$$

Bayes Optimal Classifier

- Any system that classifies examples using equation

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

Is referred to as Bayes Optimal Classifier

- No classifier, using the same space H, can outperform, in average, the Bayes Optimal Classifier
  - It works as a voting strategy taking **all hypotheses** into account!!!

Bayes Optimal Classifier

- We can go back to the blood test problem and conclude that:
  - We could have several tests instead of only one
    - Each test would correspond to a different hypothesis
    - Then we may have better results when diagnosing a disease

The Naive Bayes Classifier

Naive Bayes Classifier

- It is an alternative to the Bayes Optimal Classifier
  - The Optimal one is computionally intensive when the number of hypothesis in H is too great (enumerable)
  - Or even impossible when the number of hypothesis is infinite (non-enumerable)
    - Because it considers all hipotheses!!!

- Naive Bayes Classifier is useful when:
  - We have a representative set of attributes
  - Each example has a given class or label
  - This classifier predicts the class of unseen examples by computing probabilities based on the training set

Naive Bayes Classifier

- According to the Bayes Theorem:
  - We attempt to classify an unseen example according to the most probable class, given a set of attributes <a_1, a_2, ..., a_n>:

$$v_{MAP} = \arg\max_{v_j \in V} \frac{P(a_1, a_2, \ldots, a_n | v_j) P(v_j)}{P(a_1, a_2, \ldots, a_n)}$$

- Using the training set, we need to estimate:
  - Probability $P(v_j)$, which is simple to be estimated
  - However, assuming the training set has a limited size:
    - It becomes difficult to estimate $P(a_1, a_2, \ldots, a_n | v_j)$, because there is possibly few or no identical occurrence in the training set (due to its size, i.e., numbers of examples)
    - This second probability could be estimated if and only if the training set were huge!

Naive Bayes Classifier

- The Naive Bayes Classifier simplifies this process:
  - It assumes that attributes are independent on each other
  - In other words, the probability of observing $a_1, a_2, \ldots, a_n$ is given by the product of the individual probabilities of attributes:

$$P(a_1, a_2, \ldots, a_n | v_j) = \prod_i P(a_i | v_j)$$

  - Thus, Naive Bayes simplifies the classification process as follows:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

    - Using Naive Bayes, we observe there is no explicit search for a hypothesis
      - The hypothesis is always formulated by counting frequencies according to the query example (unseen example)

Naive Bayes Classifier: Example

- Consider the training set:

Day	Outlook	Temperature	Moisture	Wind	Play Tennis
D1	Sunny	Warm	High	Weak	No
D2	Sunny	Warm	High	Strong	No
D3	Cloudy	Warm	High	Weak	Yes
D4	Rainy	Pleasant	High	Weak	Yes
D5	Rainy	Cold	Normal	Weak	Yes
D6	Rainy	Cold	Normal	Strong	No
D7	Cloudy	Cold	Normal	Strong	Yes
D8	Sunny	Pleasant	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rainy	Pleasant	Normal	Weak	Yes
D11	Sunny	Pleasant	Normal	Strong	Yes
D12	Cloudy	Pleasant	High	Strong	Yes
D13	Cloudy	Warm	Normal	Weak	Yes
D14	Rainy	Pleasant	High	Strong	No

- Suppose the unseen example:

<Outlook=Sunny, Temperature=Cold, Moisture=High, Wind=Strong>

- Our task is to predict Yes or No to the concept “Play Tennis”

Naive Bayes Classifier: Example

- In this case we have:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j)$$

- Thus:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j)$$
$$P(Panorama = Ensolarado|v_j)P(Temperatura = Fria|v_j)$$
$$P(Umidade = Alta|v_j)P(Vento = Forte|v_j)$$

- Computing:

$$P(Vento = Forte|Jogar\ Tênis = Sim) = 3/9$$
$$P(Vento = Forte|Jogar\ Tênis = Não) = 3/5$$
$$P(Umidade = Alta|Jogar\ Tênis = Sim) = 3/9$$
$$P(Umidade = Alta|Jogar\ Tênis = Não) = 4/5$$
$$P(Temperatura = Fria|Jogar\ Tênis = Sim) = 3/9$$
$$P(Temperatura = Fria|Jogar\ Tênis = Não) = 1/5$$
$$P(Panorama = Ensolarado|Jogar\ Tênis = Sim) = 2/9$$
$$P(Panorama = Ensolarado|Jogar\ Tênis = Não) = 3/5$$

Naive Bayes Classifier: Example

- In which:

$$P(v_j = Sim) = 9/14$$
$$P(v_j = N\tilde{a}o) = 5/14$$

- Thus:

$$P(Sim)P(Ensolarado|Sim)P(Fria|Sim)P(Alta|Sim)P(Forte|Sim) = 0.0053$$
$$P(N\tilde{a}o)P(Ensolarado|N\tilde{a}o)P(Fria|N\tilde{a}o)P(Alta|N\tilde{a}o)P(Forte|N\tilde{a}o) = 0.0206$$

- Normalizing that, we have the probability for “Play Tennis”=No is 0.795, i.e., there is a 79.5% chance there will be no game
  - Observe Naive Bayes works on discrete data!!!

Naive Bayes Classifier: Example

- Let's implement Naive Bayes...

Naive Bayes Classifier: Probability Estimation

- The probability estimation using training data is not a simple task
  - For example, what happens if we need to make estimations using a small training set with n=5 examples:

$$P(Vento = Forte | Jogar\ Tênis = Não)$$

  - Now suppose we know that:

$$P(Vento = Forte | Jogar\ Tênis = Não) = 0.08$$

  - Thus, having only 5 training examples, we have 0.08 * 5 = 0.4 occurrences in the training set?
    - i.e., there is no example representing this situation
    - How can we solve this issue?

Naive Bayes Classifier: Probability Estimation

- We can use a m-estimative in form:

$$\frac{n_c + mp}{n + m}$$

- i.e., previously we estimated the probability of events by counting, for instance, for the training set with 14 examples we have:

$$P(Vento = Forte | Jogar\ Tênis = Não) = \frac{n_c}{n} = \frac{3}{5}$$

Naive Bayes Classifier: Probability Estimation

- We can use a m-estimative in form:

$$\frac{n_c + mp}{n + m}$$

  - In which p is a estimative of the probability and m is a constant equivalent to the training set
  - A way of choosing p is to assume a Uniform distribution for all k possible values, thus:

$$p = \frac{1}{k}$$

  - For example, the attribute Wind can assume two values, therefore p=0.5
  - Observe that if m=0 then:

$$\frac{n_c + mp}{n + m} = \frac{n_c}{n}$$

Naive Bayes Classifier: Classifying Documents

- Consider we have a collection of documents and we wish to retrieve data according to user queries

- So we have:
  - A collection X of documents
  - Each document contains words of different lengths
  - A training set is available so we know the class or topic associated to each document
  - V is a finite set that contains all possible classes or topics for this collection

Naive Bayes Classifier: Classifying Documents

- To simplify, take into account that:
  - Each document is classified into only two classes for a given user:
    - Class: interesting
    - Class: uninteresting

- So we first need to organize all documents to use them to learn the concept interesting/uninteresting

Naive Bayes Classifier: Classifying Documents

- Suppose we have:
  - A training set with 700 documents classified as interesting and 300 as uninteresting

- When using Naive Bayes we notice that:
  - It does not take the position of attributes into account, i.e., instead of:

$$v_{MAP} = \arg\max_{v_j \in V} P(a_1, a_2, \ldots, a_n | v_j) P(v_j)$$

  - It considers:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

Naive Bayes Classifier: Classifying Documents

- There is an issue with this estimation:
  - If a word occurs 10 times in a document with 100 words, its probability is 10/100 = 0.1
- This happens because there are other words that could happen in a document, so, instead of estimating probabilities using:
- We will use:

$$\frac{n_c}{n}$$

$$\frac{n_c + mp}{n + m}$$

  - In which m represents the number of possible words and p is the probability for each word
  - Taking English as the language, we can assume m = 50,000 and p=1/50,000

Naive Bayes Classifier: Classifying Documents

- In this manner, we estimate the probabilities as follows:

$$\frac{n_c + 1}{n + 50.000}$$

- Therefore we have a better idea about the probability a word happens in a document
  - Once we consider a universe of possible values or words that could occur inside the same text
- Instead of assuming 50,000 as the size of our vocabulary, we can better estimate it by counting all different words occuring in all documents of collection X
  - Therefore we will have a more precise estimation of probabilities to this problem we are approaching

Naive Bayes Classifier: Classifying Documents

- Our learning algorithm is:

Input:

Collection of documents X
V is the set of possible classes

Output:

Probabilities

Learning Algorithm:

1. Collect all words occuring in all documents in X
   Vocabulary ← set of all different words
2. Compute the probabilities $P(v_j)$ and $P(w_k|v_j)$
   For each value v_j in V do:
   $docs_j$ ← subset of documents labeled as v_j
   $P(v_j)$ ← $|docs_j| / |X|$
   $Document_j$ ← Unique document that contains all elements in $docs_j$
   n ← total number of different words in $Document_j$
   For each word w_k in Vocabulary
   n_k ← number of times word w_k occurs in $Document_j$
   $P(w_k|v_j)$ ← $(n_k + 1) / (n + |Vocabulary|)$

Naive Bayes Classifier: Classifying Documents

- So, the classification algorithm is given by:

Input:
A query document
Output:
The class estimated to the query document

Classification Algorithm:
attributes ← list of words found in the unseen document
Return:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_{a_i \in atributos} P(a_i|v_j)$$

Naive Bayes Classifier: Classifying Documents

- Implementing
  - Implement the previous algorithm
  - Test the document classification using the following datasets:
    - 20 Newsgroups Dataset
      - http://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups
    - Reuters-21578 Dataset
      - http://www.inf.ed.ac.uk/teaching/courses/dme/html/datasets0405.html

Other Issues

- There are situations in which attributes have some level of dependency on each other
  - For example, in a document about “machine learning” the probability of happening the word “learning” after “machine” may be greater
  - For that, we can use Bayesian Belief Networks:
    - It takes those dependencies into account

Suggestions

- Install the R Statistical Software:
  - Install packages tm and Snowball
  - Implement Naive Bayes to classify the Reuters-21578 Dataset
  - Important Functions:
    - stopwords()
    - removeNumbers()
    - removePunctuation()
    - removeWords()
    - StemDocument()
    - stripWhitespace()
    - termFreq()
    - PlainTextDocument()

References

- Stephen Marsland, Machine Learning: An Algorithmic Perspective, Chapman and Hall/CRC, 2011
- Freeman and Skapura, Neural Networks: Algorithms, Applications and Programming Techniques, Addison-Wesley, 1991
- C. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag New York, 2006
- S. O. Haykin, Neural Networks and Learning Machines, Prentice Hall, 2008
- W. C. Shefler, Statistics: Concepts and Applications, The Benjamin/Cummings, 1988
- Tom Mitchell, Machine Learning, 1994