Perceptron

Thanh-Nghi Do
dtnghi@cit.ctu.edu.vn

Can Tho
Dec. 2019

Content

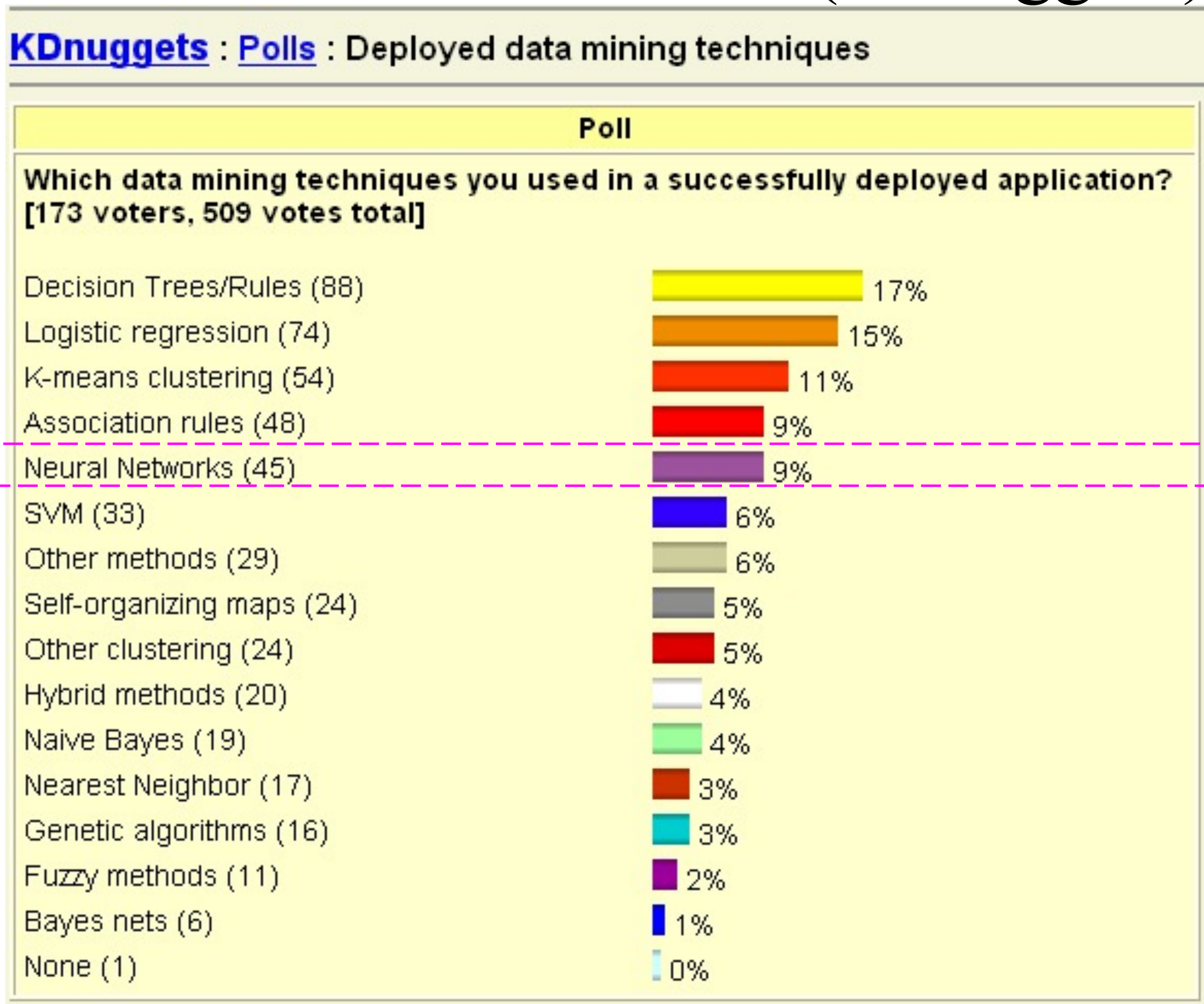
- Introduction
- McCulloch & Pitts model
- Perceptron
- Conclusions

Content

- **<u>Introduction</u>**
- McCulloch & Pitts model
- Perceptron
- Conclusions

Successfully Methods

(Kdnuggets)



Introduction

- Artificial neural networks
  - Inspired by biological brains and neurons
  - Studied since 1943 (McCulloch & Pitts neuron)
  - Perceptron (Rosenblatt, 1958): learning rule for the computational neuron model

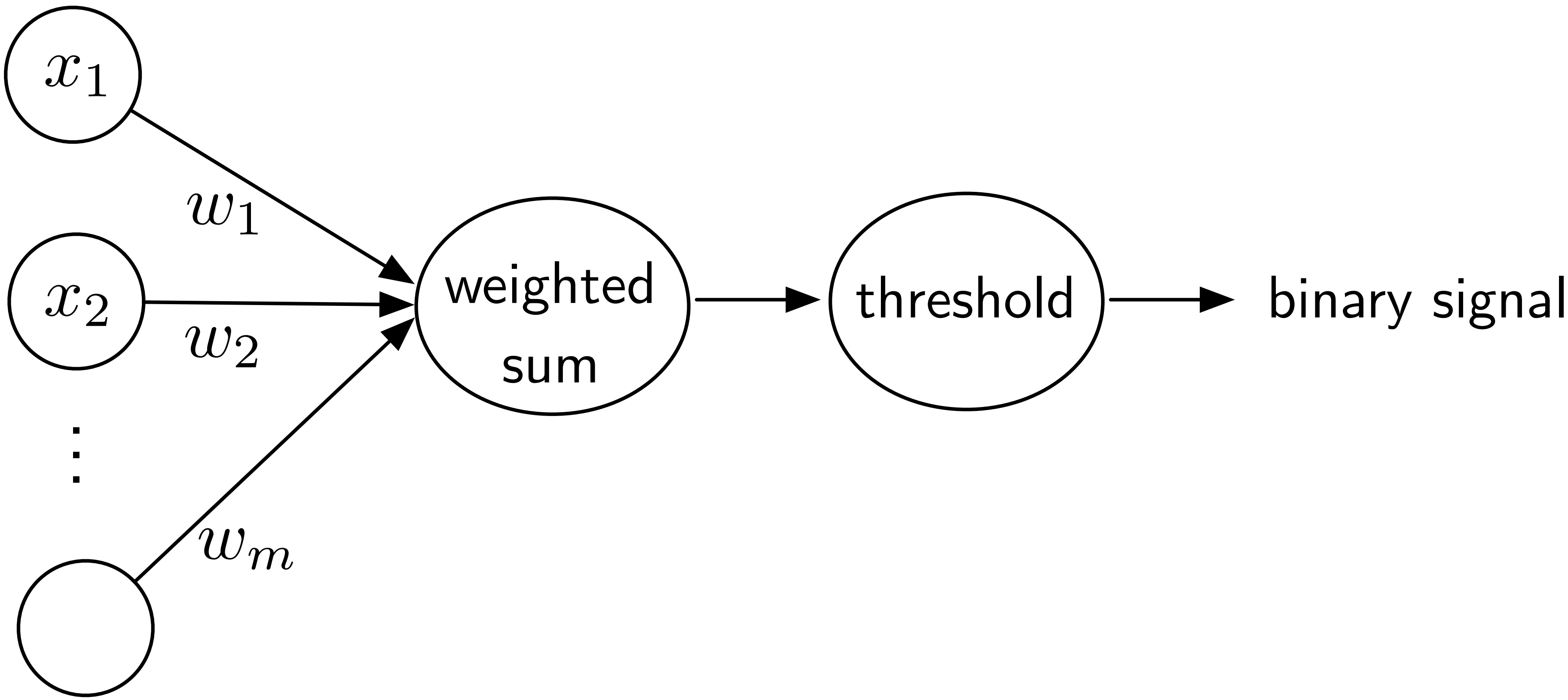
Introduction

- History
  - Mathematical formulation of a biological neuron (McCulloch & Pitts, 1943)
  - Perceptron: a learning algorithm for the neuron model (Rosenblatt, 1958)
  - Adaline: a nicely differentiable neuron model (Widrow & Hoff, 1960)
  - Learning representations by back-propagating errors (Werbos, Lecun, Rumelhart, Hinton, Williams, 1980s)
  - Self-organizing map (Kohonen, 1984)
  - Convolutional neural networks (Hinton, LeCun, Bengio, 1995)
  - Support vector networks (Vapnik, 1995)
  - Deep learning (Hinton, 2006)

Content

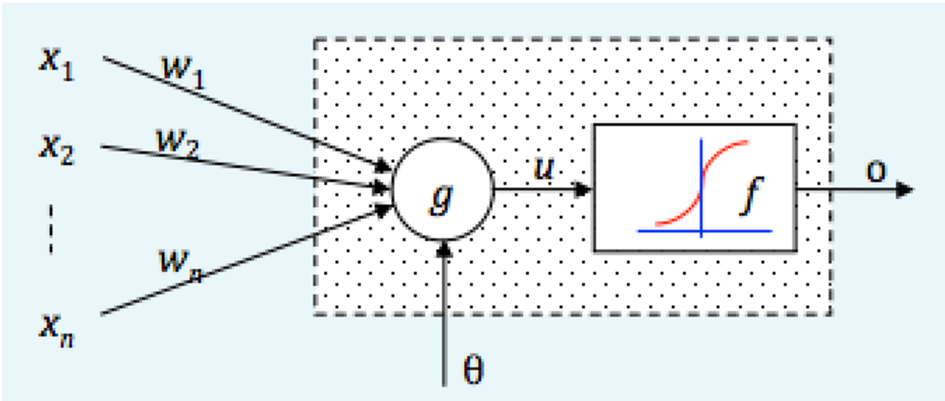
- Introduction
- **McCulloch & Pitts model**
- Perceptron
- Conclusions

McCulloch & Pitts neuron



McCulloch & Pitts neuron

- Neuron: basic computing unit
  - n inputs, 1 threshold (θ) and 1 output (o)
  - w_i: weights
  - g: combination function
  - f: activation function



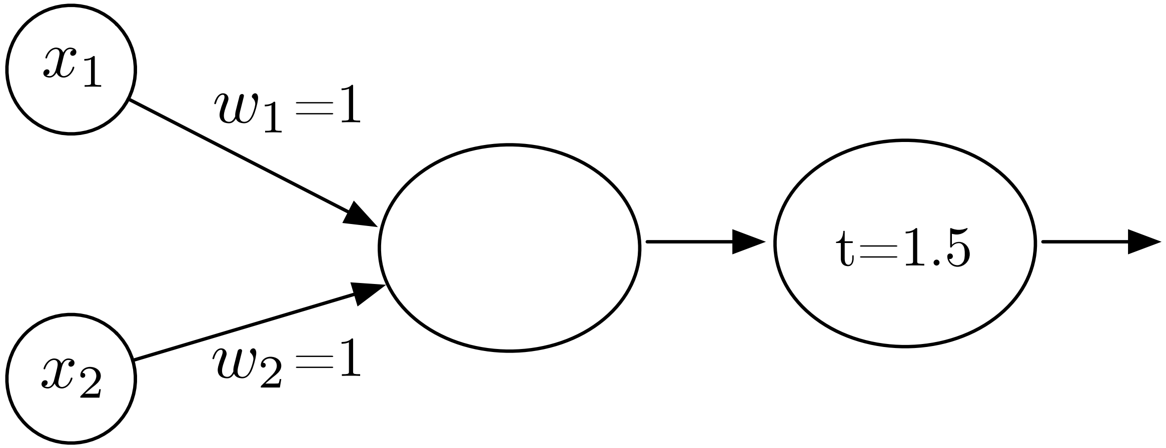
$$u = g(x) = \sum_{i=1} w_i x_i + \theta$$

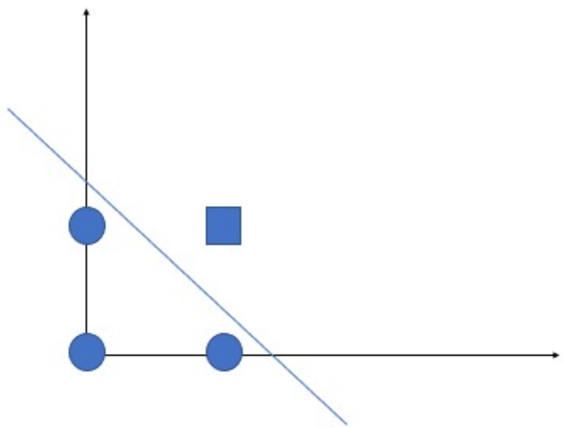
$$o = f(u) = \frac{1}{1 + e^{-u/T}}$$

Sigmoid function

McCulloch & Pitts neuron

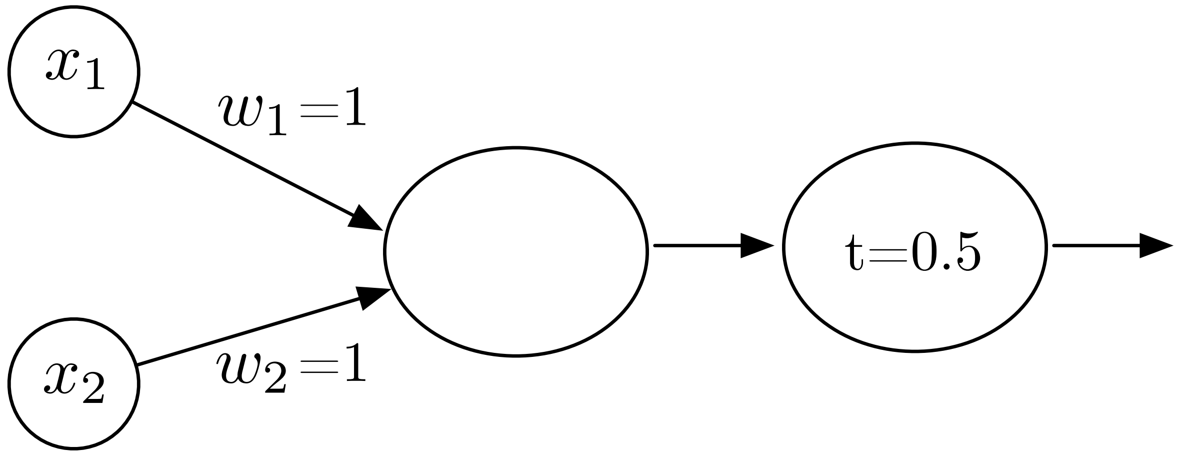
x_1	x_2	Out
0	0	0
0	1	0
1	0	0
1	1	1

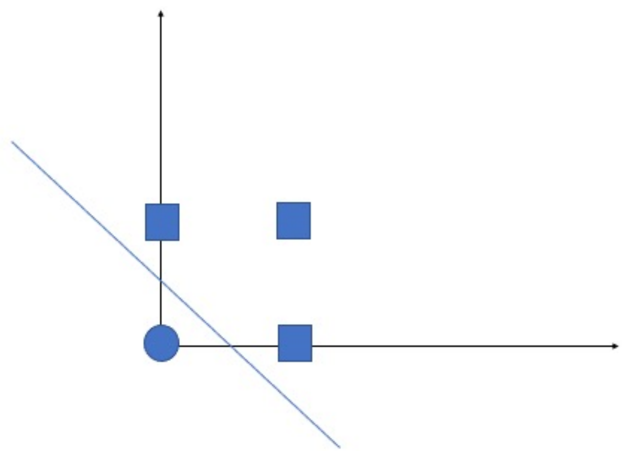




McCulloch & Pitts neuron

x_1	x_2	Out
0	0	0
0	1	1
1	0	1
1	1	1





Content

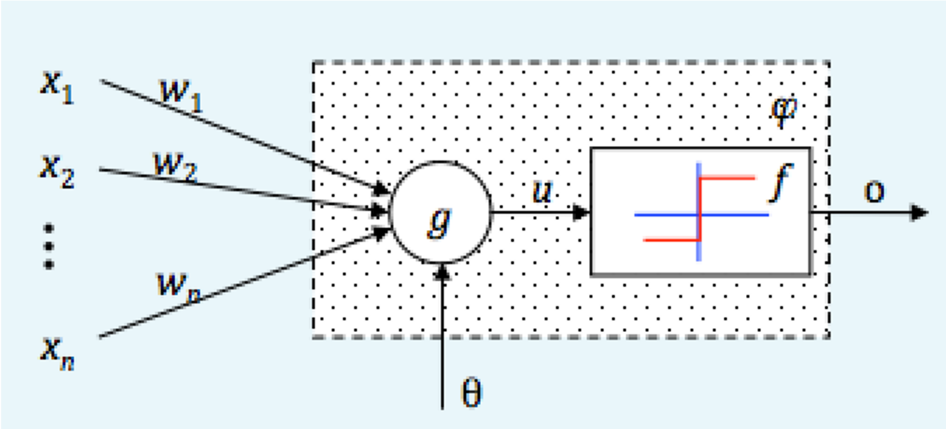
- Introduction
- McCulloch & Pitts model
- **<u>Perceptron</u>**
- Conclusions

Perceptron

- Perceptron: 1 neuron
  - Proposed by (Rosenblatt, 1958)
  - McCulloch & Pitts model
  - Perceptron with threshold θ
    - n inputs, 1 threshold & 1 output
    - activation function: threshold function

$$u = g(x) = \sum_{i=1} w_i x_i + \theta$$

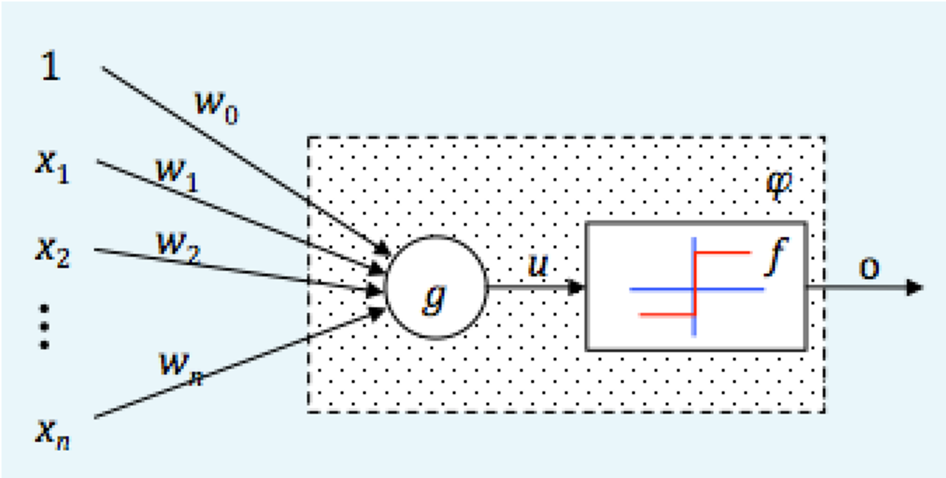
$$o = f(u) = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0 \end{cases}$$



Perceptron

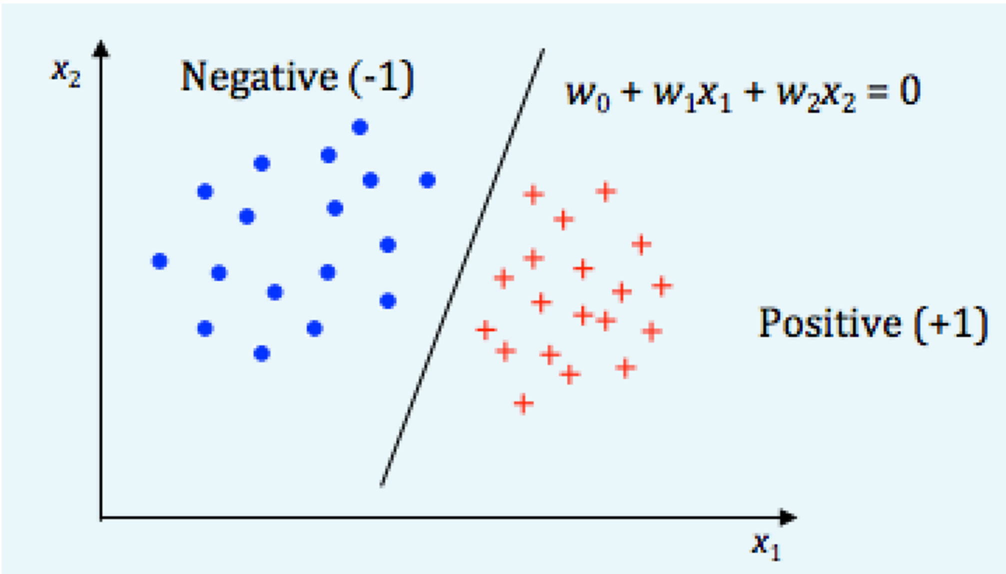
- Perceptron with threshold θ
  - threshold θ is considered as w_0 associated with *pseudo input* $x_0 = 1$
  - (n + 1) inputs and 1 output

$$u = g(x) = \sum_{i=0}^{n} w_i x_i$$



Perceptron

- Geometrical interpretation
  - Given vector x = (x_1, x_2, …, x_n), Perceptron computes the output o which is 0 or 1 => binary classification
  - Separating hyper-plane



$$u = g(x) = \sum_{i=0}^{n} w_i x_i = 0$$

Perceptron

- Learning algorithm
  - Training dataset: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
  - Try to find weights w_i so that the perceptron model predicts the output $o = y^{(i)}$ of a example $x^{(i)}$ for $i = 1, 2, \ldots, m$.
- Geometrical interpretation
  - Try to find a hyper-plane for separating examples so that each class is one side of the hyper-plane.

Perceptron

- Input
  - Training dataset: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
  - Learning rate η
- Learning algorithm for linear separation
  - Randomly initializing weights w_j
  - For each example $(x^{(i)}, y^{(i)})$, perceptron predicts the output o_i of the example $x^{(i)}$

    if o_i != $y^{(i)}$ then updating w_j

$$w_j = w_j + \eta \cdot (y_i - o_i) \cdot x_{ij}, \forall j = 0..n$$

Perceptron

- Learning algorithm for linear non-separation
  - Try to find a good separation hyper-plan as possible
    - Minimizing classification errors (loss)
  - Loss function

$$E(w) \equiv E(w_0, w_1, \ldots, w_n) = \frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - g(x^{(i)})\right)^2$$

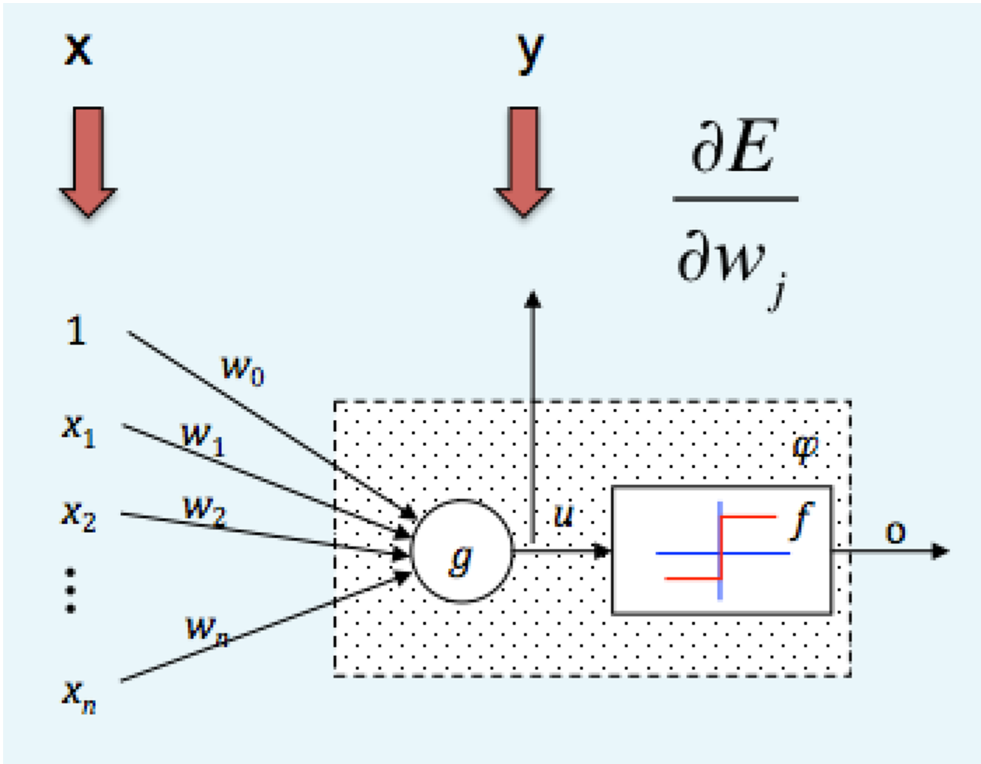
  - Try to find w which minimizes E

Perceptron

- Learning algorithm for linear non-separation
  - Gradient descent (standard)
    - Randomly initializing weights w_j
    - Updating w_j:

$$w_j = w_j - \eta \frac{\partial E}{\partial w_j}$$

$$\frac{\partial E}{\partial w_j} = -\sum_{i=1}^{m} \left(y^{(i)} - g(x^{(i)})\right) x_j^{(i)}$$



Perceptron

- Learning algorithm for linear non-separation
  - Stochastic gradient descent
    1. Randomly pick an example $x^{(i)}$ and update w_j

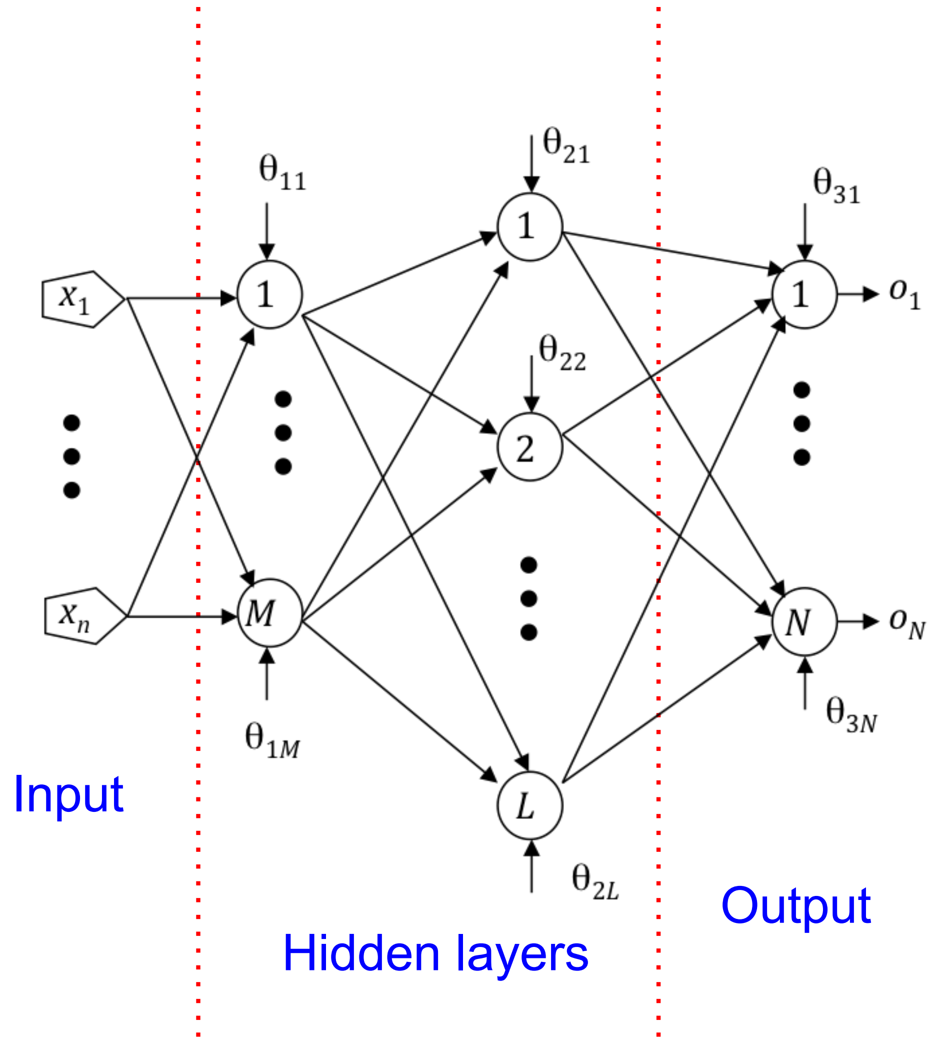
$$w_j = w_j - \eta \frac{\partial E}{\partial w_j}$$

$$\frac{\partial E}{\partial w_j} = -\left(y^{(i)} - g(x^{(i)})\right).x_j^{(i)}$$

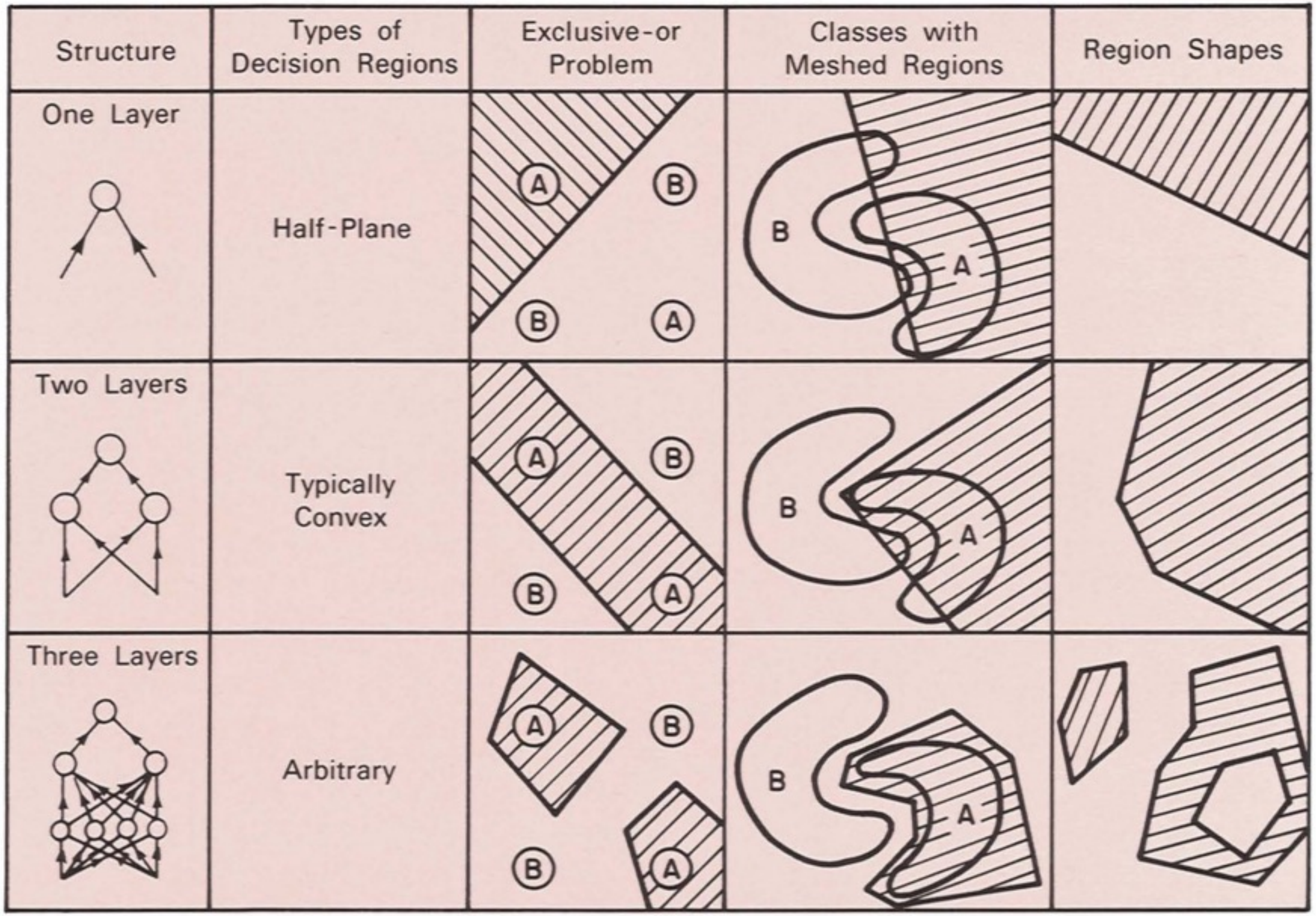
    2. Repeat step 1 until convergence

Multi-layer perceptron

- Neural networks for non-linear classification
  - Layers are usually full connected
  - Number of hidden layers tuned by hand
  - Error back-propagation
  - Stochastic gradient descent



Network structure



Content

- Introduction
- McCulloch & Pitts model
- Perceptron
- **Conclusions**

Conclusions

- McCulloch & Pitts neuron model
- Perceptron: single layer, multi-layer
- Stochastic gradient descent
- Error back-propagation
- Applications: pattern recognition, text classification, bioinformatics, natural language processing
- Deep learning: hot topic!

Merci !