# Applied inductive learning - Lecture 5

Louis Wehenkel

Department of Electrical Engineering and Computer Science
University of Liège

Montefiore - Liège - October 17, 2005

*Find slides:* *http://montefiore.ulg.ac.be/∼lwh/AIA/*

# Batch-mode vs Online-mode Supervised Learning

- Objects (or observations): $LS = \{o_1, \ldots, o_N\}$
- Attribute vector: $\mathbf{a}^i = (a_1(o_i), \ldots, a_n(o_i))^T$, $\forall i = 1, \ldots, N.$
- Outputs: $y^i = y(o_i)$ or $c^i = c(o_i)$, $\forall i = 1, \ldots, N.$
- LS Table

| $o$ | $a_1(o)$ | $a_2(o)$ | $\ldots$ | $a_n(o)$ | $y(o)$ |
|---|---|---|---|---|---|
| 1 | $a_1^1$ | $a_2^1$ | $\ldots$ | $a_n^1$ | $y^1$ |
| 2 | $a_1^2$ | $a_2^2$ | $\ldots$ | $a_n^2$ | $y^2$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $N$ | $a_1^N$ | $a_2^N$ | $\ldots$ | $a_n^N$ | $y^N$ |

Focus for this lecture on numerical inputs, and numerical outputs (classes will be encoded numerically if needed).

# Batch-mode vs online mode learning

- In batch-mode
  - Samples provided and processed together to construct model
  - Need to store samples (not the model)
  - Classical approach for data mining
- In online-mode
  - Samples provided and processed one by one to update model
  - Need to store the model (not the samples)
  - Classical approach for adaptive systems
- But both approaches can be adapted to handle both contexts
  - Samples available together can be exploited one by one
  - Samples provided one by one can be stored and then exploited together

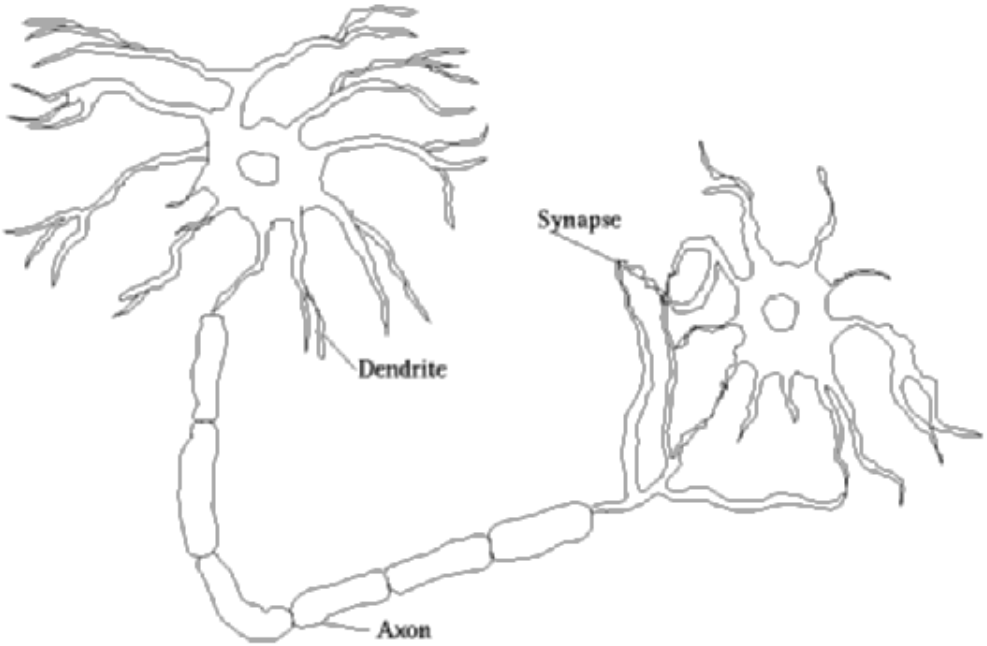
# Motivations for Artificial Neural Networks

Intuition: biological brain can learn, so let's try to be inspired by it to build learning algorithms.

- Starting point: single neuron models
  - perceptron, LTU and STU for linear supervised learning
  - online (biologically plausible) learning algorithms
- Complexify: multilayer preceptrons
  - flexible models for non-linear supervised learning
  - universal approximation property
  - iterative training algorithms based on non-linear optimization
- ...other neural network models of importance

# Single neuron models

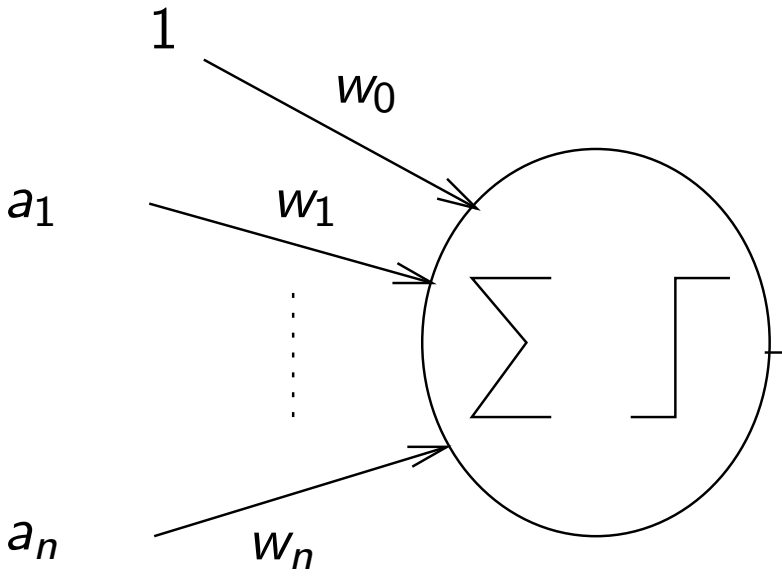
The biological neuron:




Human brain: $10^{11}$ neurons, each with $10^4$ synapses
Memory (knowledge): stored in the synapses

# Hard threshold unit...

A simple (simplistic) mathematical model of the biological neuron




$$g(\mathbf{a}(o)) = \text{sgn}\left\{w_0 + \mathbf{w}^T\mathbf{a}(o)\right\}$$

$$= \text{sgn}\left\{\mathbf{w}'^T\mathbf{a}'(o)\right\}$$

Parameters to adapt to problem: $\mathbf{w}'$

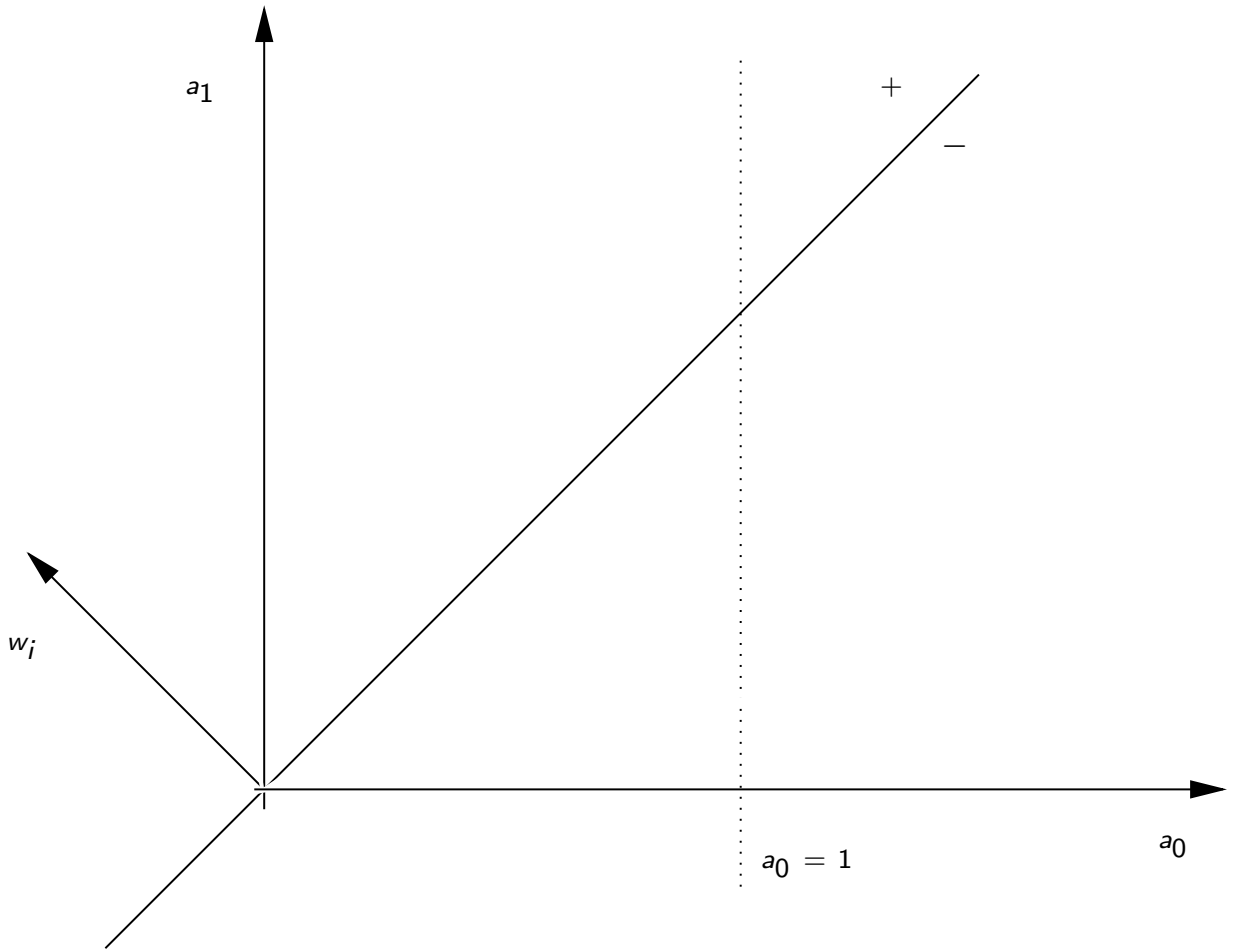
# ...and the perceptron learning algorithm

1. For binary classification: $c(o) = \pm 1$.
2. Start with an arbitrary initial weight vector, e.g. $\mathbf{w}'_0 = \mathbf{0}$.
3. Consider the objects of the $LS$ in a cyclic or random sequence.
4. Let $o_i$ be the object at step $i$, $c(o_i)$ its class and $\mathbf{a}(o_i)$ its attribute vector.
5. Adjust the weight by using the following correction rule,

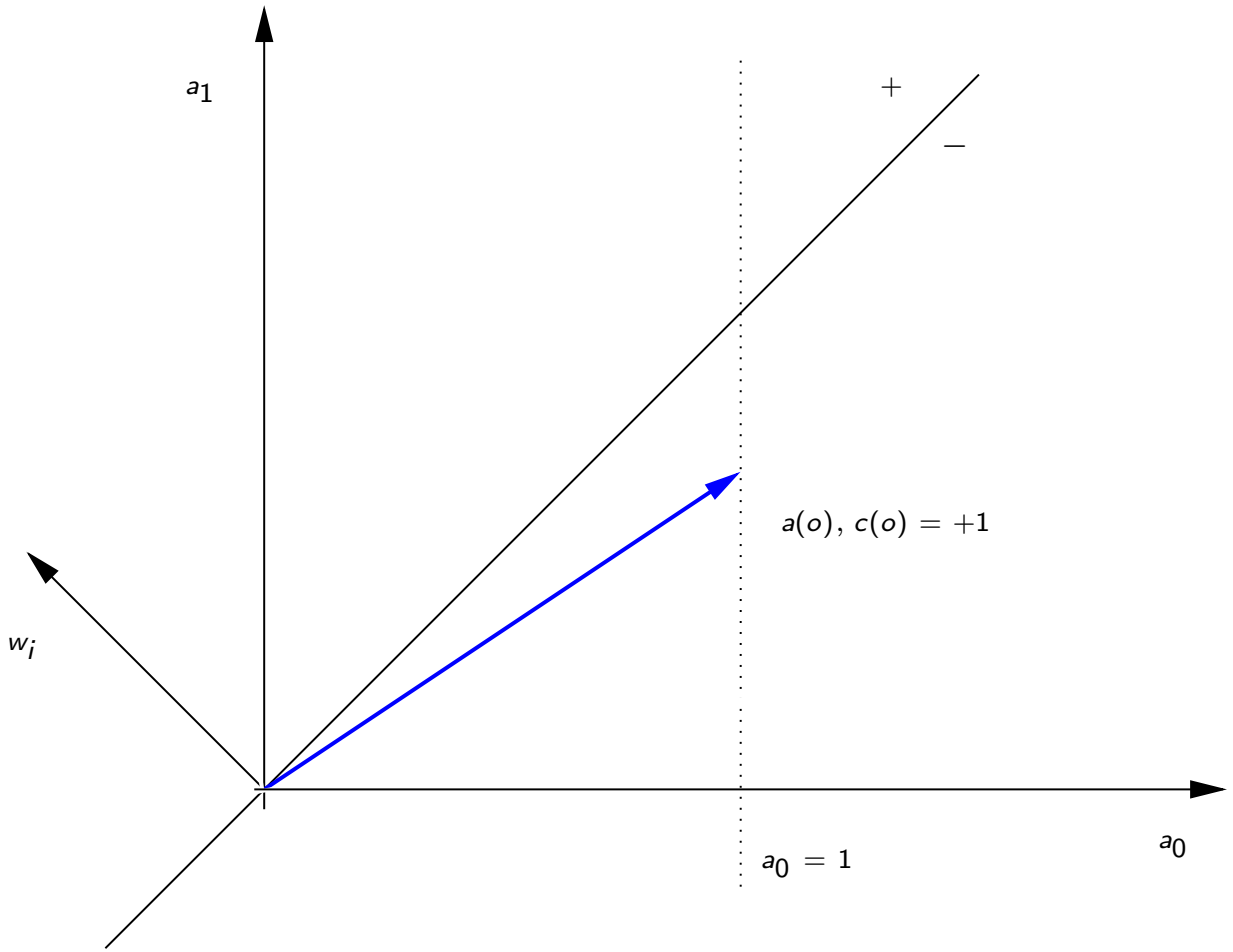
$$\mathbf{w}'_{i+1} = \mathbf{w}'_i + \eta_i \left(c(o_i) - g_i(\mathbf{a}(o_i))\right) \mathbf{a}'(o_i).$$

- $\mathbf{w}'_i$ changes only if $o_i$ is not correctly classified
- it is changed in the right direction ($\eta_i > 0$ is the learning rate)
- at any stage, $\mathbf{w}'_i$ is a linear combination of the $\mathbf{a}(o_i)$ vectors

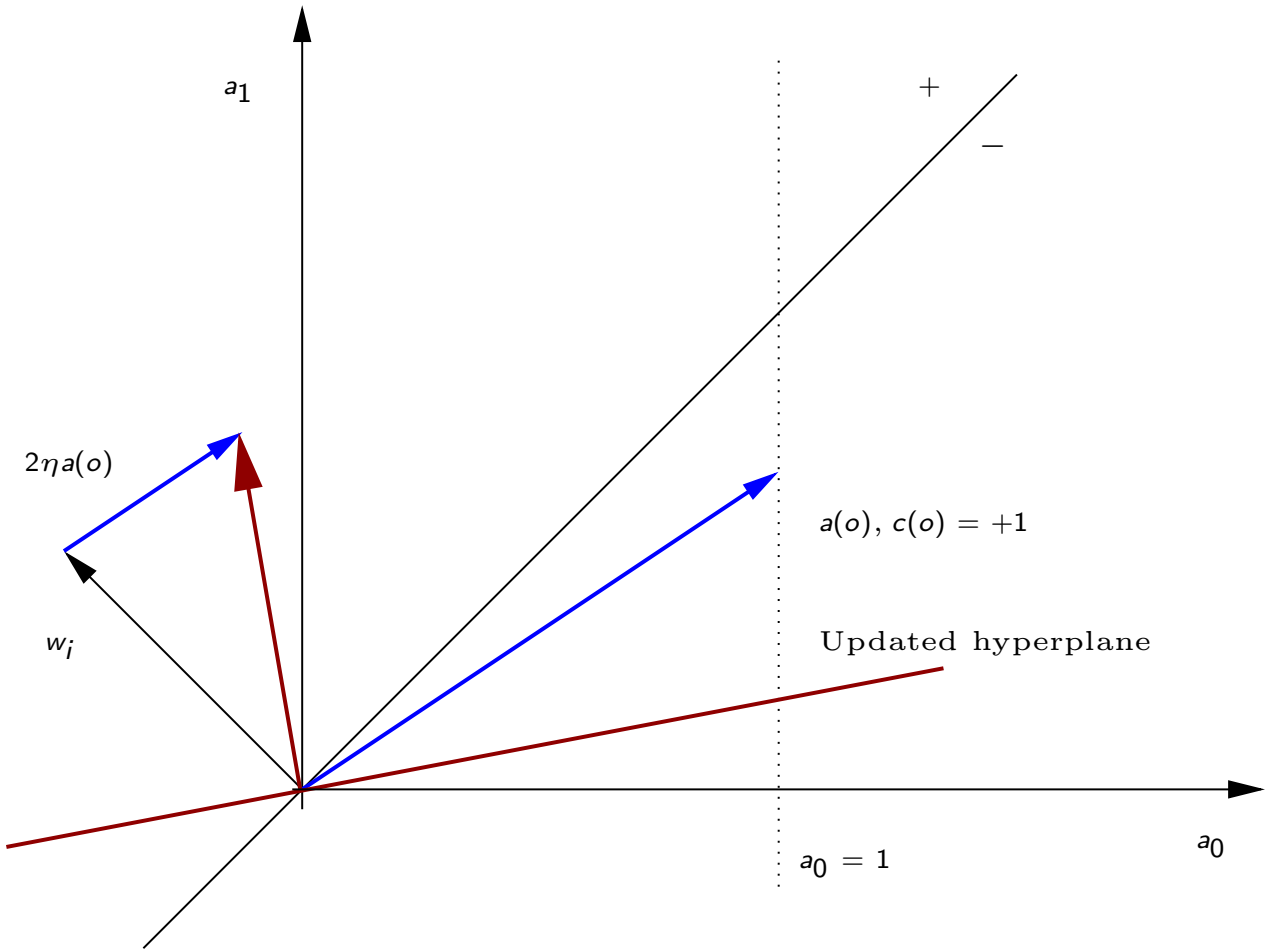
# Geometrical view of update equation



# Geometrical view of update equation



# Geometrical view of update equation



# Soft threshold units...

The input/output function $g(\mathbf{a})$ of such a device is computed by

$$g(\mathbf{a}(o)) \stackrel{\triangle}{=} f(w_0 + \mathbf{w}^T \mathbf{a}(o)) = f(\mathbf{w}'^T \mathbf{a}'(o))$$

where the *activation* function $f(\cdot)$ is assumed to be differentiable. Classical examples of activation functions are the sigmoid

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)},$$

and the hyperbolic tangent

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)},$$

# ... and gradient descent

Find vector $\mathbf{w}'^T = (w_0, \mathbf{w}^T)$ minimizing the square error (TSE)

$$TSE(LS, \mathbf{w}') = \sum_{o \in LS} (g(\mathbf{a}(o)) - y(o))^2 = \sum_{o \in LS} \left( f(\mathbf{w}'^T \mathbf{a}'(o)) - y(o) \right)^2 .$$

The gradient with respect to $\mathbf{w}'$ is computed by

$$\nabla_{\mathbf{w}'} TSE(LS, \mathbf{w}') = 2 \sum_{o \in LS} (g(\mathbf{a}(o)) - y(o))\, f'(\mathbf{w}'^T \mathbf{a}'(o)) \mathbf{a}'(o),$$

where $f'(\cdot)$ denotes the derivative of the activation function $f(\cdot)$.

The gradient descent method works by iteratively changing the weight vector by a term proportional to $-\nabla_{\mathbf{w}'} TSE(LS, \mathbf{w}')$.

# ... and stochastic online gradient descent

*Fixed* step gradient descent in online-mode:

1. For binary classification: $c(o) = \pm 1$.
2. Start with an arbitrary initial weight vector, e.g. $\mathbf{w}'_0 = \mathbf{0}$.
3. Consider the objects of the $LS$ in a cyclic or random sequence.
4. Let $o_i$ be the object at step $i$, $c(o_i)$ its class and $\mathbf{a}(o_i)$ its attribute vector.
5. Adjust the weight by using the following correction rule,

$$\begin{aligned} \mathbf{w}'_{i+1} &= \mathbf{w}'_i - \eta_i \nabla_{\mathbf{w}'} SE(o_i, \mathbf{w}'_i) \\ &= \mathbf{w}'_i + 2\eta_i \left[c(o_i) - g_i(\mathbf{a}(o_i))\right] f'(\mathbf{w}'^T_i \mathbf{a}'(o_i))\mathbf{a}'(o), \end{aligned}$$

($SE(o, \mathbf{w}')$ is the contribution of object $o$ in $TSE(LS, \mathbf{w}')$.)

# Theoretical properties

- Convergence of the perceptron learning algorithm
  - If $LS$ is linearly separable: converges in a finite number of steps.
  - Otherwise: converges with infinite number of steps, if $\eta_i \to 0$.
- Convergence of the online or batch gradient descent algorithm
  - if $\eta_i \to 0$ (slowly), and infinite number of steps, same solution
  - if $f(\cdot)$ linear, finds same solution as linear regression

  NB: slow $\eta_i \to 0$ means
  - $\lim_{m\to\infty} \sum_{i=1}^{m} \eta_i = +\infty$
  - $\lim_{m\to\infty} \sum_{i=1}^{m} \eta_i^2 < +\infty$

# Single layer models

- NB: the number of layers, refers to the number of connection layers.
- A single layer model is composed of several perceptrons.
- These work completely independently
- They can be trained independently
- Interpretations:
  - hyperplanes in Euclidean space
  - boolean operators (e.g. Logical AND, OR etc.)

# Representation capacity of multilayer perceptrons

- Classification:
  - geometrical insight in the representation capacity
  - threelayer hard threshold units
    - First layer: can define a collection of hyperplanes/semiplanes
    - Second layer: can define arbitrary intersections of semiplanes
    - Third layer: can define abitrary union of intersections of semi-planes
    - Conclusion: with a sufficient number of units, very complex regions can be described
- Regression:
  - function approximation insight
  - twolayer soft threshold units
    - One-dimensional input space illustration
    - First layer, defines $K$ offset and scale parameters $\alpha_i, \beta_i, i = 1 \ldots K$: responses $f(\alpha_i x + \beta_i)$
    - Second (linear layer): $\hat{y}(x) = b_0 + \sum_{i=1}^{K} b_i f(\alpha_i x + \beta_i)$

# Learning algorithms for multilayer perceptrons

First assume that only the last layer has to be learned
Then assumme also the other layers need to be learned

# Backpropagation of derivatives

# Second order optimization methods

# Radial basis functions etc.

# Relation with tree-based methods, linear models and KNN

# Overfitting in multilayer perceptrons