Introduction to Machine Learning

A classification and engineering perspective

Aggelos Pikrakis, Eng., Ph.D.
Dept. of Informatics, University of Piraeus, Greece
email: pikrakis@unipi.gr,
web: https://sites.google.com/site/aggelospikrakis/

This presentation was prepared based on material from the books:

- S. Theodorids and K. Koutroumbas, "*Pattern Recognition, 4th Edition*", Academic Press, 2008
- S. Theodoridis, A. Pikrakis, K. Koutroumbas and D. Cavouras, "*Introduction to Pattern Recognition: a Matlab Approach*", Academic Press, 2010

What is Machine Learning?

- A machine learning algorithm is an algorithm that is able to learn from data.

- "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (after Mitchell, 1997)

- The name suggests the use of a machine/computer to learn in analogy to how the brain learns and predicts.

- In some cases, the methods are directly inspired by the way the brain works, as is the case with neural networks.

Well known machine learning tasks

Classification: assign an unknown pattern to one out of a number of classes that are considered to be known.

Regression: predict a numerical value given some input.

Transcription: observe a relatively unstructured representation of some kind of data and transcribe it into discrete, textual form (e.g., OCR).

Machine Translation: input consists is a sequence of symbols in some language, and the algorithm must convert this into a sequence of symbols in another language.

Anomaly Detection: the computer program sifts through a set of events or objects, and flags some of them as being unusual or atypical.

Well known machine learning tasks

Synthesis and Sampling: generate new examples that are similar to those in the training data.

Imputation of Missing values: given a new example with some missing entries, provide a prediction of the values of the missing entries.

Denoising: input is a corrupted example obtained by an unknown corruption process The learner must predict the clean example from its corrupted version.

Density Estimation: learn a function $p_{model} : R^n \rightarrow R$, where $p_{model}(x)$ can be interpreted as a probability density function (if x is continuous) or a probability mass function (if x is discrete) on the space that the examples were drawn from.

Scientific disciplines with a Machine Learning Character

- Statistics and Statistical Learning
- Pattern Recognition
- Signal and Image Processing and Analysis
- Computer Science
- Data Mining
- Machine Vision
- Bioinformatics
- Industrial Automation
- Computer-Aided Medical Diagnosis

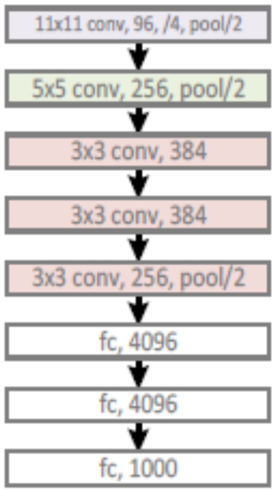
What about deep learning?

Deep Learning is a class of machine learning algorithms that:

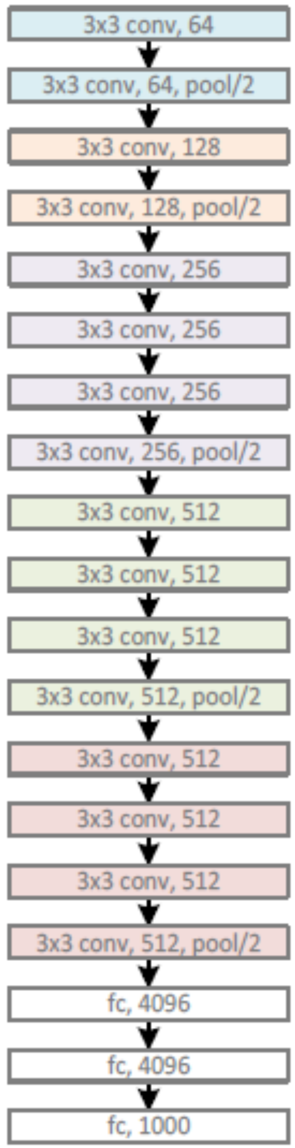
- ❖ Use a cascade of many layers of non-linear signal processing
- ❖ Are part of the broader machine learning field of learning representations of data facilitating end-to-end optimization
- ❖ Learn multiple levels of representations that correspond to hierarchies of concept abstractions. (after Li Deng)
- ❖ Feedforward architectures (including stacks of Restricted Boltzmann Machines, Autoencoders and Convolutional Neural Networks) and Recurrent architectures (including Recurrent Neural Networks (RNNs) and LSTM RNNs)

Depth is of crucial importance

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

ILSVRC (Large Scale Visual Recognition Challenge)

(slide credit: Jian Sun, MSR)

Applications and APIs

Celebrated applications

- Speech Recognition
- Speech-to-Speech Translation (cross-lingual conversations in real-time)
- Object Recognition, Image Auto-tagging, Face recognition
- Semantic modeling, natural language processing, multimodality, reasoning, knowledge representation/management/exploitation, optimal decision making

Popular Application Programming Interfaces (APIs)

✓ Tensorflow (Google, https://www.tensorflow.org)
✓ Torch (http://torch.ch)
✓ The Microsoft Cognitive Toolkit (github.com/Microsoft/CNTK/wiki)
✓ Theano (Academic, http://deeplearning.net/software/theano/)

(*switch to PyCharm for a sec …*)

Frequently used datasets

- **MNIST**: A standard toy data set of handwritten digits.
- **TIMIT**: A standard speech benchmark for clean speech recognition.
- **CIFAR-10** and **CIFAR-100**: Tiny natural images (Krizhevsky, 2009).
- **Street View House Numbers data set (SVHN)**: Images of house numbers collected by Google Street View (Netzer et al., 2011).
- **ImageNet**: A large collection of natural images.
- **Reuters-RCV1**: A collection of Reuters newswire articles.
- **Alternative Splicing data set**: RNA features for predicting alternative gene splicing.

LINEAR CLASSIFIERS

❖ The Problem: Consider a two class task with ω_1, ω_2

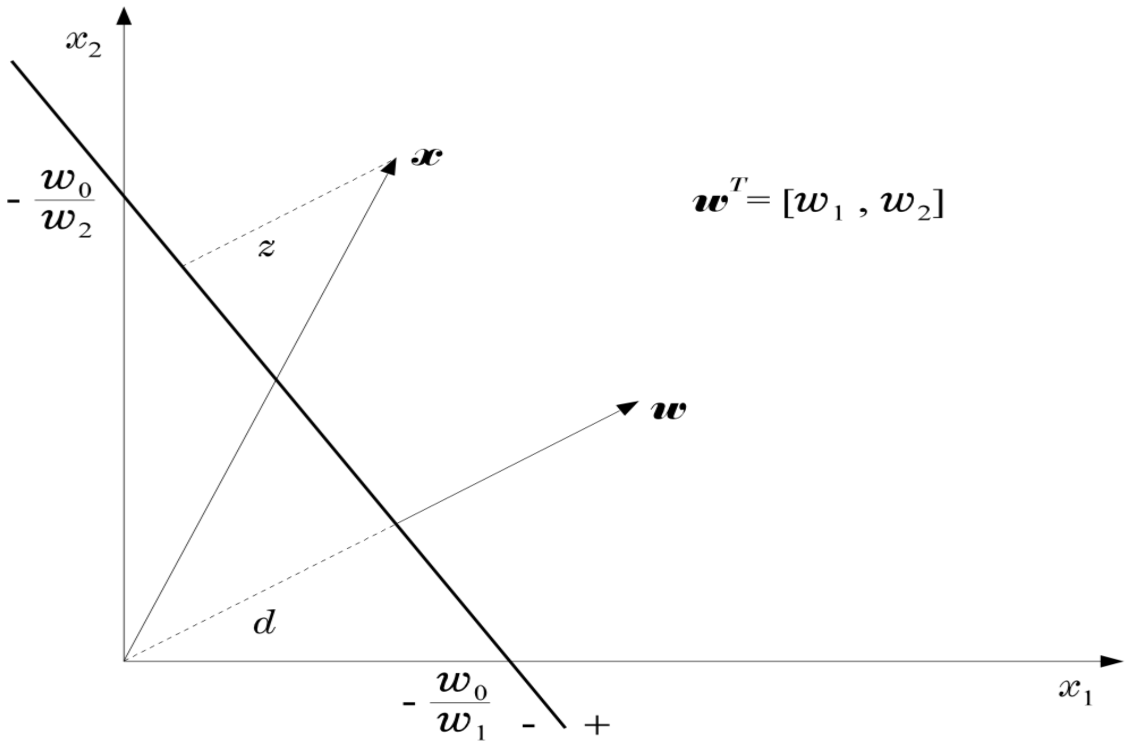
➢ $$g(\underline{x}) = \underline{w}^T \underline{x} + w_0 = 0 =$$
$$w_1 x_1 + w_2 x_2 + \ldots + w_l x_l + w_0$$

➢ Assume $\underline{x}_1, \underline{x}_2$ on the decision hyperplane:
$$0 = \underline{w}^T \underline{x}_1 + w_0 = \underline{w}^T \underline{x}_2 + w_0 \Rightarrow$$
$$\underline{w}^T (\underline{x}_1 - \underline{x}_2) = 0 \;\; \forall \underline{x}_1, \underline{x}_2$$

➢ Hence:

$$\boxed{\underline{w} \perp \text{on the hyperplane}}$$

$$g(\underline{x}) = \underline{w}^T \underline{x} + w_0 = 0$$



$$d = \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}}, \quad z = \frac{|g(\underline{x})|}{\sqrt{w_1^2 + w_2^2}}$$

❖ The Perceptron Algorithm

- Assume linearly separable classes, i.e.,

$$\exists\ \underline{w}^*:\ \underline{w}^{*T}\underline{x} > 0\ \ \forall \underline{x} \in \omega_1$$
$$\underline{w}^{*T}\underline{x} < 0\ \ \forall \underline{x} \in \omega_2$$

- The case $\underline{w}^{*T}\underline{x} + w_0^*$ falls under the above formulation, since

  - $\underline{w}' \equiv \begin{bmatrix} \underline{w}^* \\ w_0^* \end{bmatrix},\ \underline{x}' = \begin{bmatrix} \underline{x} \\ 1 \end{bmatrix}$

  - $\underline{w}^{*T}\underline{x} + w_0^* = \underline{w}'^T\underline{x}' = 0$

- Our goal: Compute a solution, i.e., a hyperplane $\underline{w}$, so that

$$\underline{w}^T \underline{x} (><) 0 \quad \underline{x} \in \left\langle \begin{array}{l} \omega_1 \\ \omega_2 \end{array} \right.$$

  - The steps
    - Define a cost function to be minimized
    - Choose an algorithm to minimize the cost function
    - The minimum corresponds to a solution

➢ The Cost Function

$$J(\underline{w}) = \sum_{\underline{x} \in Y} (\delta_x \underline{w}^T \underline{x})$$

- Where Y is the subset of the vectors wrongly classified by $\underline{w}$. When Y=(empty set) a solution is achieved and

- $J(\underline{w}) = 0$

- $\delta_x = -1$ if $\underline{x} \in Y$ and $\underline{x} \in \omega_1$

  $\delta_x = +1$ if $\underline{x} \in Y$ and $\underline{x} \in \omega_2$

- $J(\underline{w}) \geq 0$

- The Perceptron Algorithm
  - The philosophy of the gradient descent is adopted.

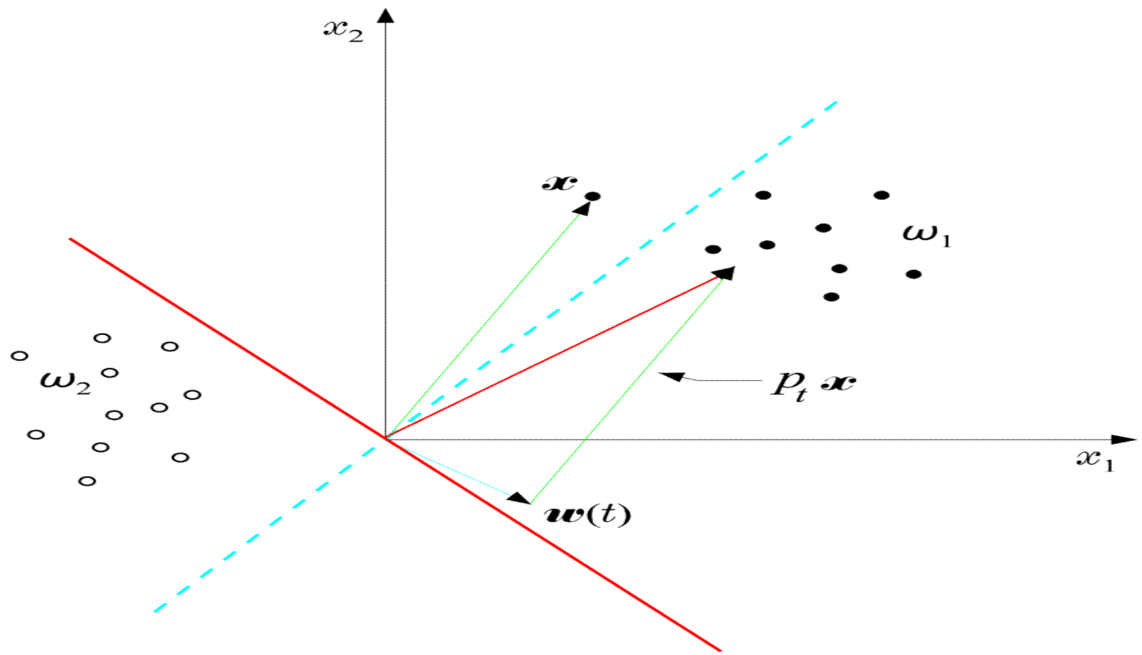
$$\underline{w}(\text{new}) = \underline{w}(\text{old}) + \Delta \underline{w}$$

$$\Delta \underline{w} = -\mu \frac{\partial J(\underline{w})}{\partial \underline{w}} \Big| \underline{w} = \underline{w}(\text{old})$$

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} \left(\sum_{\underline{x} \in Y} \delta_x \underline{w}^T \underline{x} \right) = \sum_{\underline{x} \in Y} \delta_x \underline{x}$$

$$\boxed{\underline{w}(t+1) = \underline{w}(t) - \rho_t \sum_{\underline{x} \in Y} \delta_x \underline{x}}$$

➢ An example:



$$\underline{w}(t+1) = \underline{w}(t) + \rho_t \underline{x}$$
$$= \underline{w}(t) - \rho_t \delta_x \underline{x} \quad (\delta_x = -1)$$

➢ The perceptron algorithm **converges** in a **finite** number of iteration steps to a solution if

$$\lim_{t\to\infty} \sum_{k=0}^{t} \rho_k \to \infty, \qquad \lim_{t\to\infty} \sum_{k=0}^{t} \rho_k^{\ 2} < +\infty$$

$$\text{e.g.,:}\ \ \rho_t = \frac{c}{t}$$

❖ A useful variant of the perceptron algorithm

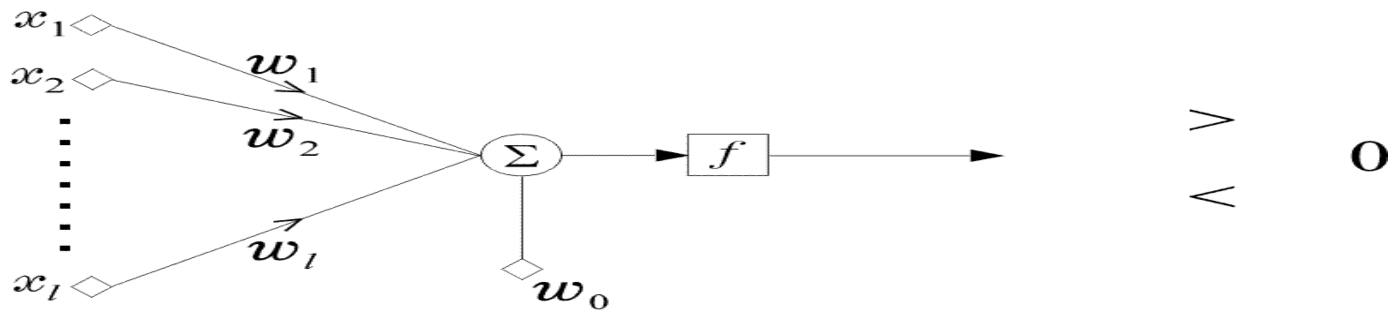
$$\underline{w}(t+1) = \underline{w}(t) + \rho \underline{x}_{(t)}, \quad \begin{array}{c} \underline{w}^T(t)\underline{x}_{(t)} \leq 0 \\ \underline{x}_{(t)} \in \omega_1 \end{array}$$

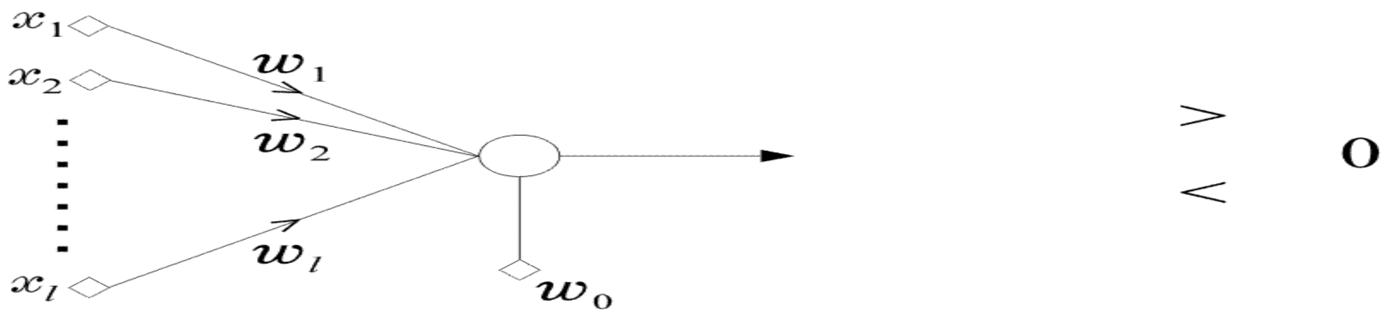
$$\underline{w}(t+1) = \underline{w}(t) - \rho \underline{x}_{(t)}, \quad \begin{array}{c} \underline{w}^T(t)\underline{x}_{(t)} \geq 0 \\ \underline{x}_{(t)} \in \omega_2 \end{array}$$

$$\underline{w}(t+1) = \underline{w}(t) \quad \text{otherwise}$$

- It is a **reward and punishment** type of algorithm

❖ The perceptron





$w_i's$ synapses or synaptic weights

w_0 threshold

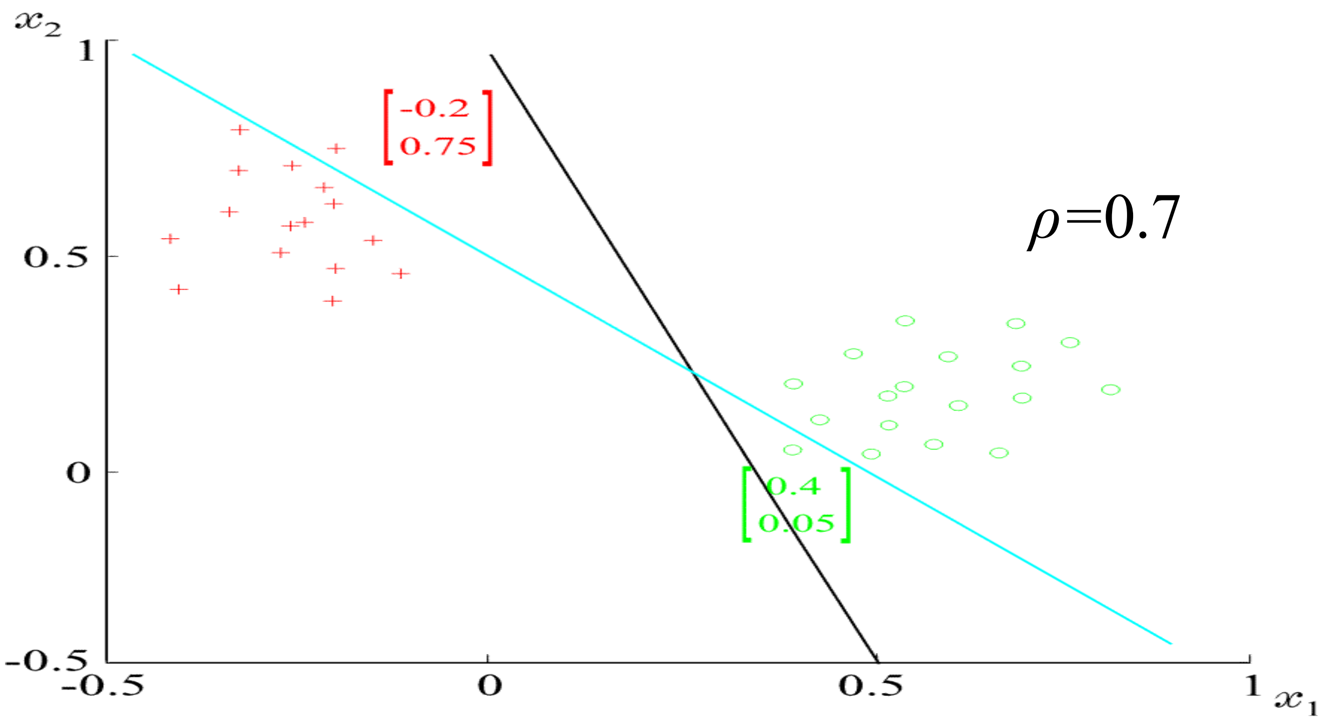
- The network is called perceptron or neuron
- It is a learning machine that learns from the training vectors via the perceptron algorithm

➢ Example: At some stage t the perceptron algorithm results in

$$w_1 = 1,\ w_2 = 1,\ w_0 = -0.5$$

$$x_1 + x_2 - 0.5 = 0$$

The corresponding hyperplane is

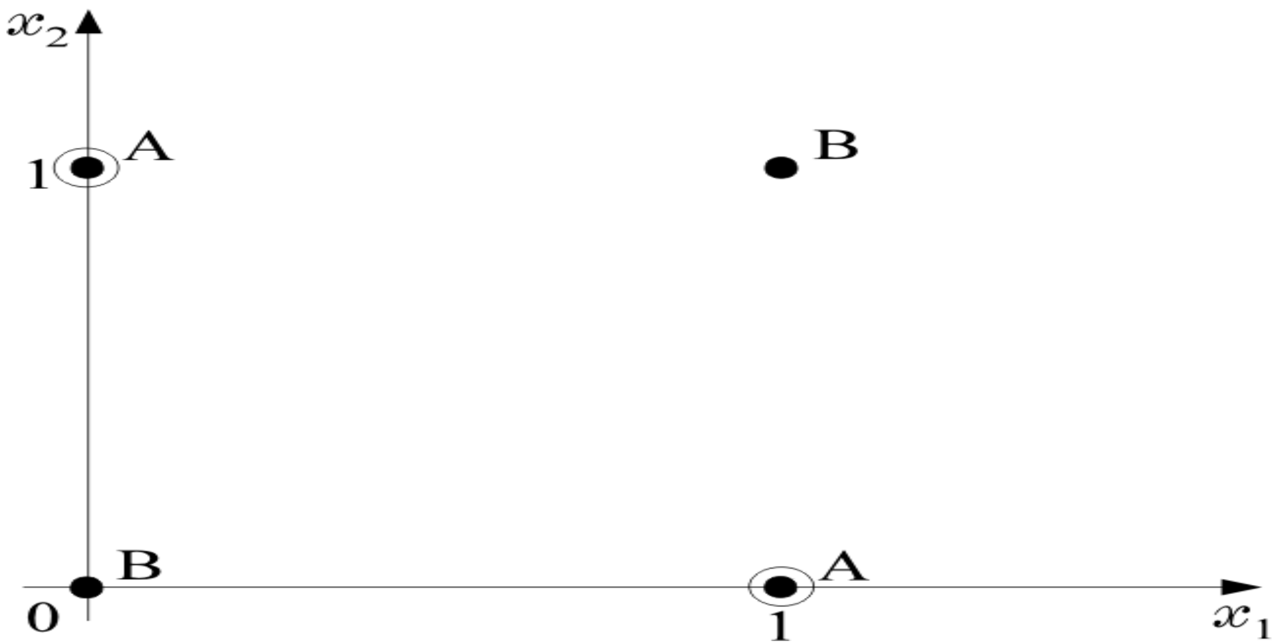


$$\underline{w}(t+1) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix} - 0.7(-1)\begin{bmatrix} 0.4 \\ 0.05 \\ 1 \end{bmatrix} - 0.7(+1)\begin{bmatrix} -0.2 \\ 0.75 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.42 \\ 0.51 \\ -0.5 \end{bmatrix}$$

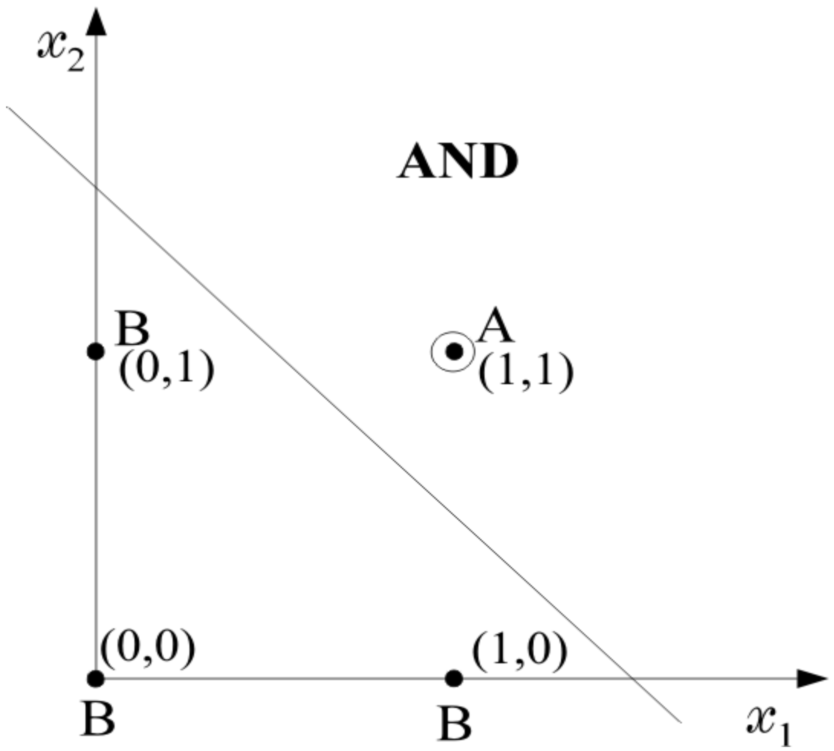
Non Linear Classifiers

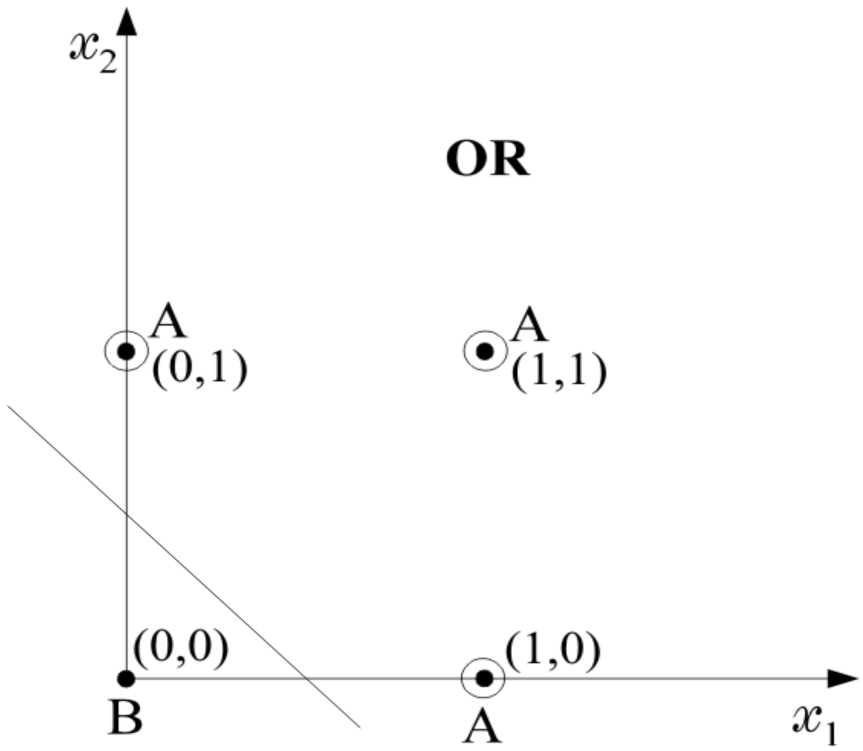
❖ The XOR problem

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B



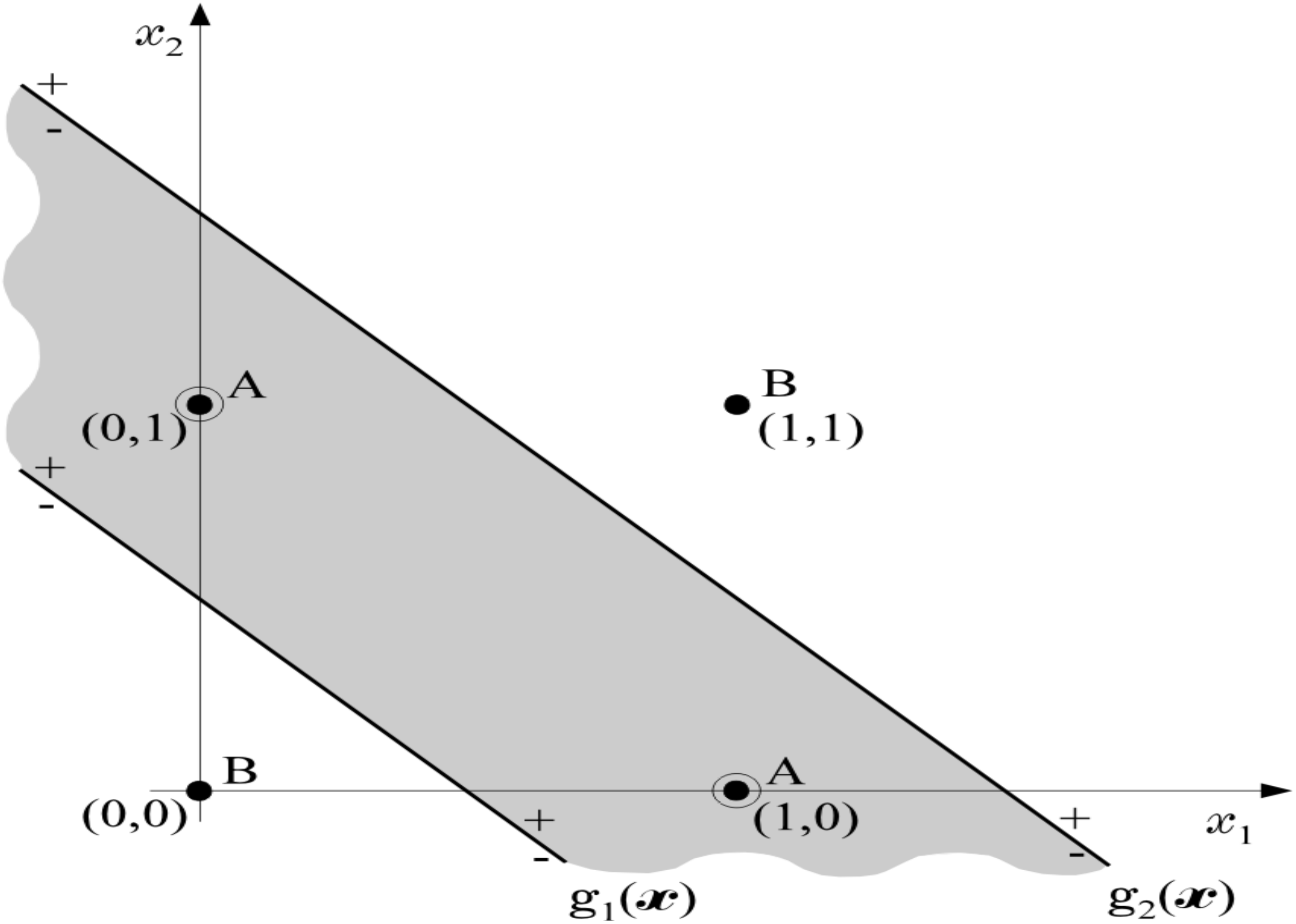
❖ There is no single line (hyperplane) that separates class A from class B. On the contrary, AND and OR operations are linearly separable problems





❖ The Two-Layer Perceptron

➢ For the XOR problem, draw two, instead, of one lines



- Then class B is located outside the shaded area and class A inside. This is a two-phase design.
  - Phase 1: Draw two lines (hyperplanes)

    $$g_1(\underline{x}) = g_2(\underline{x}) = 0$$

    Each of them is realized by a perceptron. The outputs of the perceptrons will be

    $$y_i = f(g_i(\underline{x})) = \begin{cases} 0 \\ 1 \end{cases} \quad i = 1, 2$$

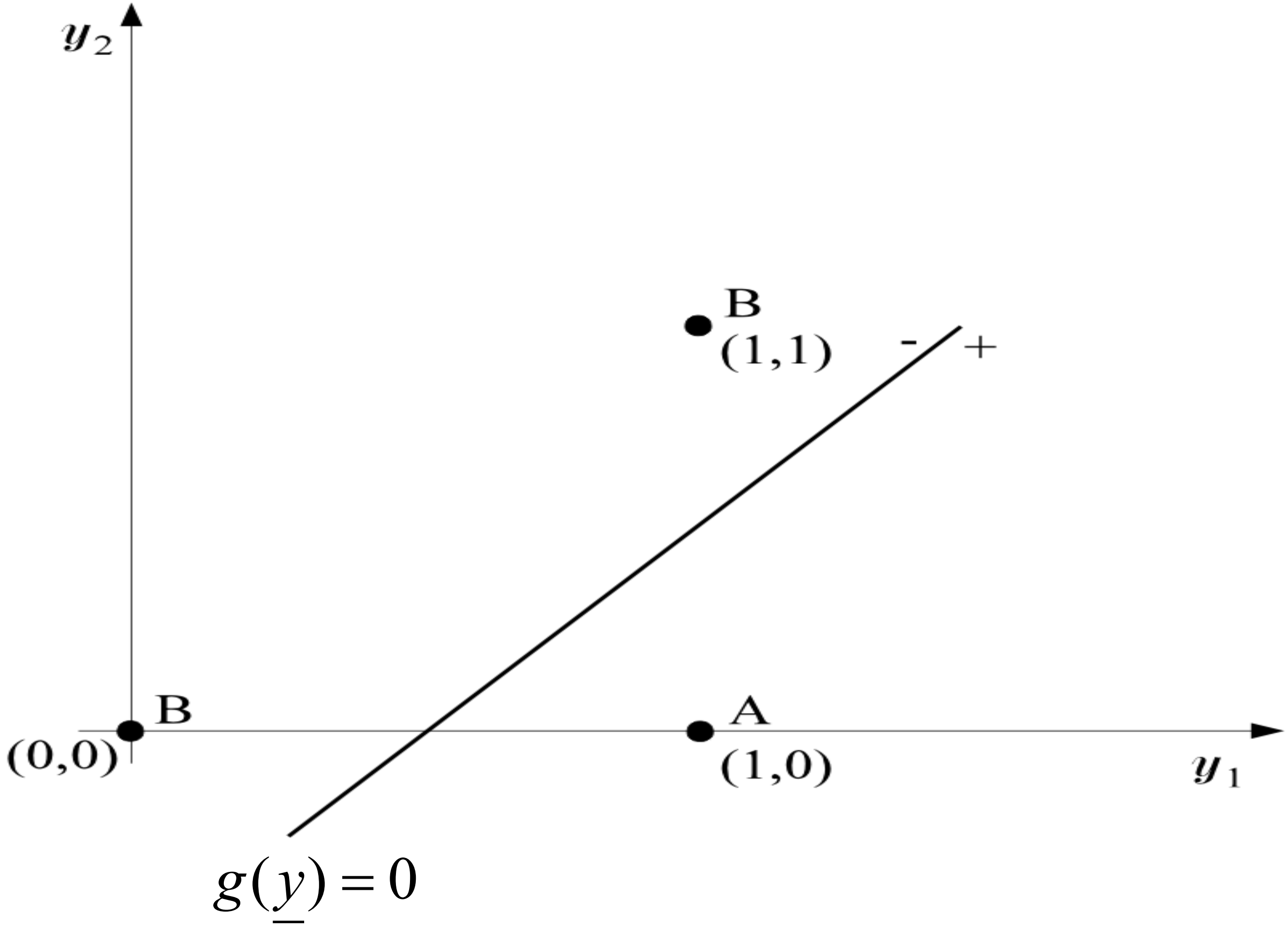
    depending on the position of $\underline{x}$.

  - Phase 2: Find the position of $\underline{x}$ *w.r.t.* both lines, based on the values of y_1, y_2.

1st phase				2nd phase
x_1	x_2	y_1	y_2	
0	0	0(-)	0(-)	B(0)
0	1	1(+)	0(-)	A(1)
1	0	1(+)	0(-)	A(1)
1	1	1(+)	1(+)	B(0)

- Equivalently: The computations of the first phase perform a mapping $\underline{x} \rightarrow \underline{y} = [y_1, y_2]^T$

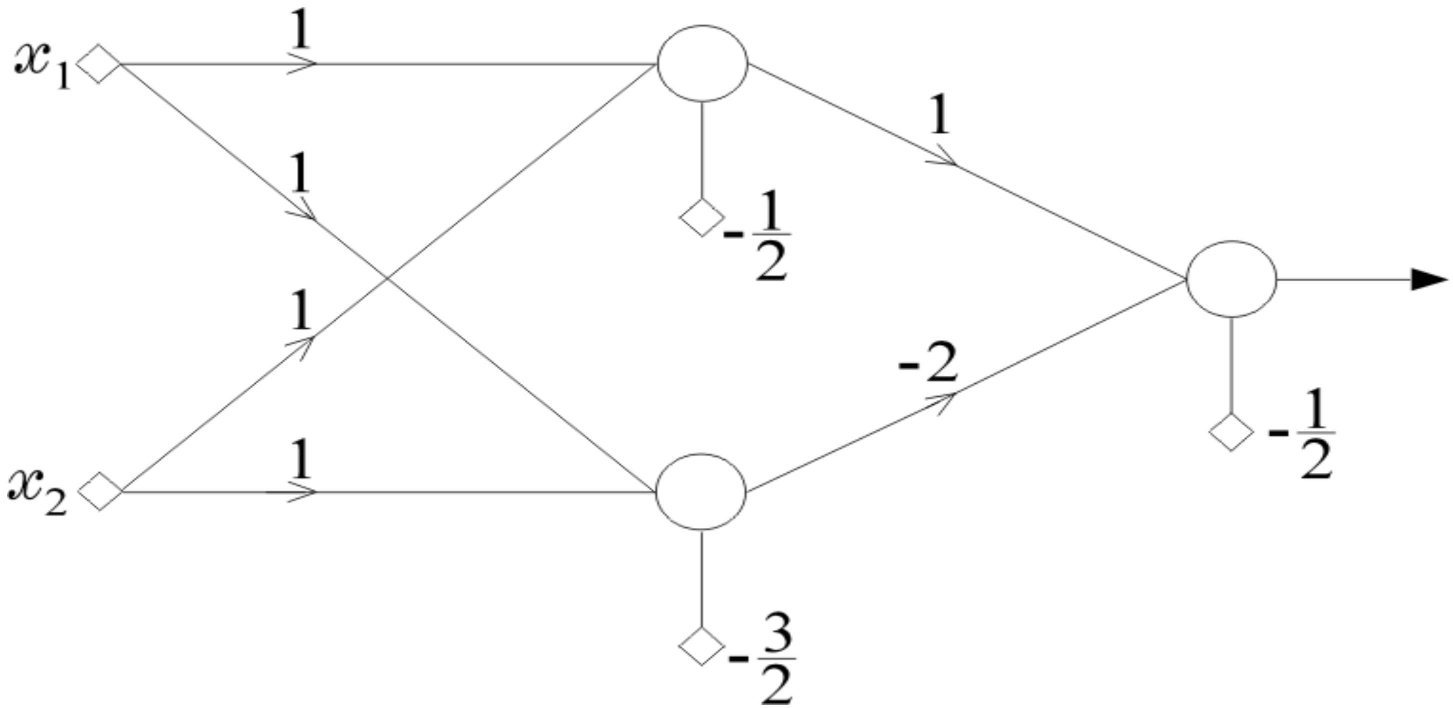
The decision is now performed on the transformed $\underline{y}$ data.



This can be performed via a second line, which can also be realized by a perceptron.

- Computations of the first phase perform a mapping that transforms the nonlinearly separable problem to a linearly separable one.

- The architecture



- This is known as the two layer perceptron with one hidden and one output layer. The activation functions are

$$f(.) = \begin{cases} 0 \\ 1 \end{cases}$$

- The neurons (nodes) of the figure realize the following lines (hyperplanes)

$$g_1(\underline{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

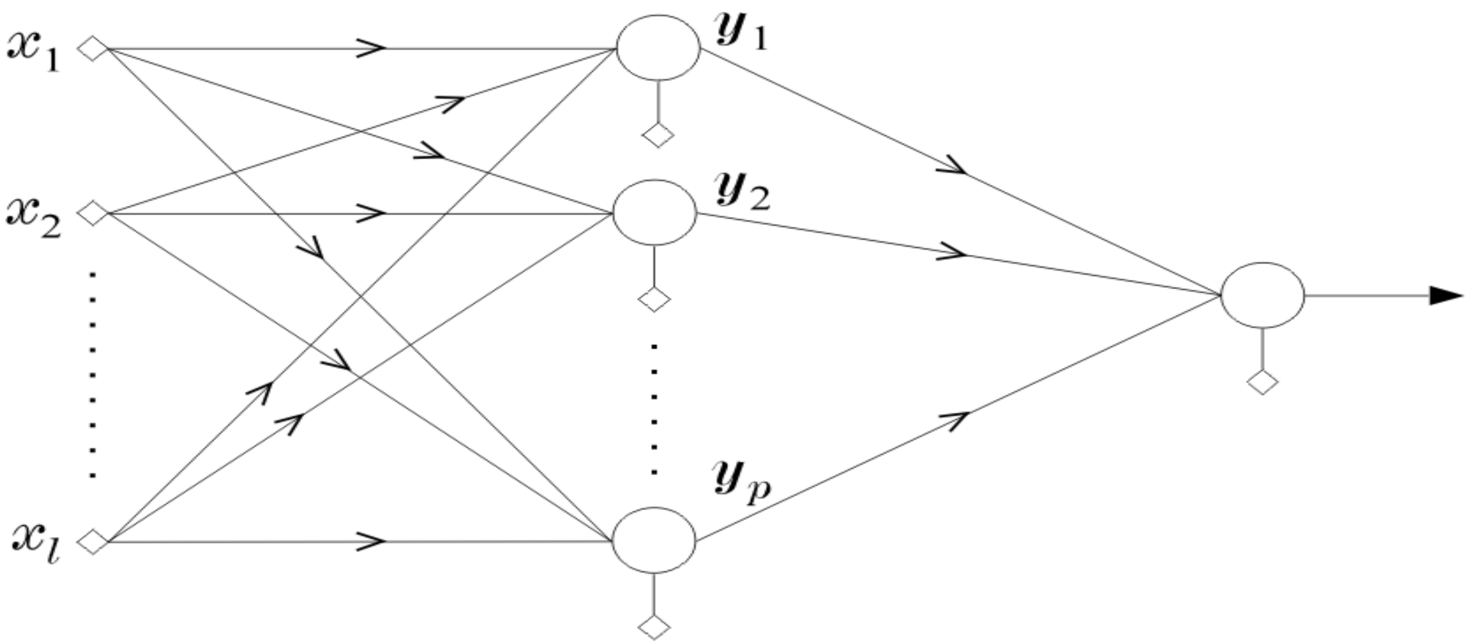
$$g_2(\underline{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

$$g(\underline{y}) = y_1 - 2y_2 - \frac{1}{2} = 0$$

❖ Classification capabilities of the two-layer perceptron

➢ The mapping performed by the first layer neurons is onto the vertices of the unit side square, e.g., (0, 0), (0, 1), (1, 0), (1, 1).

➢ The more general case,



$$\underline{x} \in R^l$$

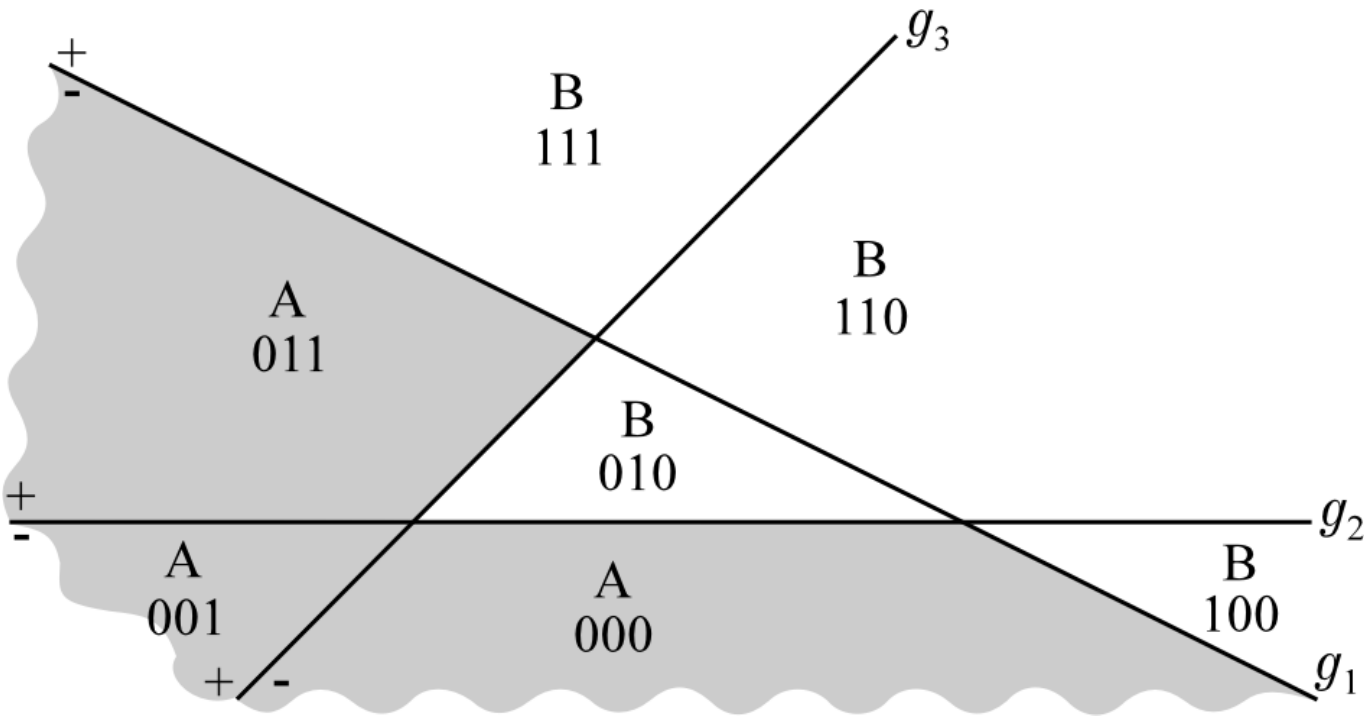
$$\underline{x} \rightarrow \underline{y} = [y_1, \ldots y_p]^T,\ y_i \in \{0, 1\}\ \ i = 1, 2, \ldots p$$

performs a mapping of a vector

onto the vertices of the unit side H_p hypercube

- The mapping is achieved with p neurons each realizing a hyperplane. The output of each of these neurons is 0 or 1 depending on the relative position of $\underline{x}$ w.r.t. the hyperplane.

➢ Intersections of these hyperplanes form regions in the l-dimensional space. Each region corresponds to a vertex of the H_p unit hypercube.

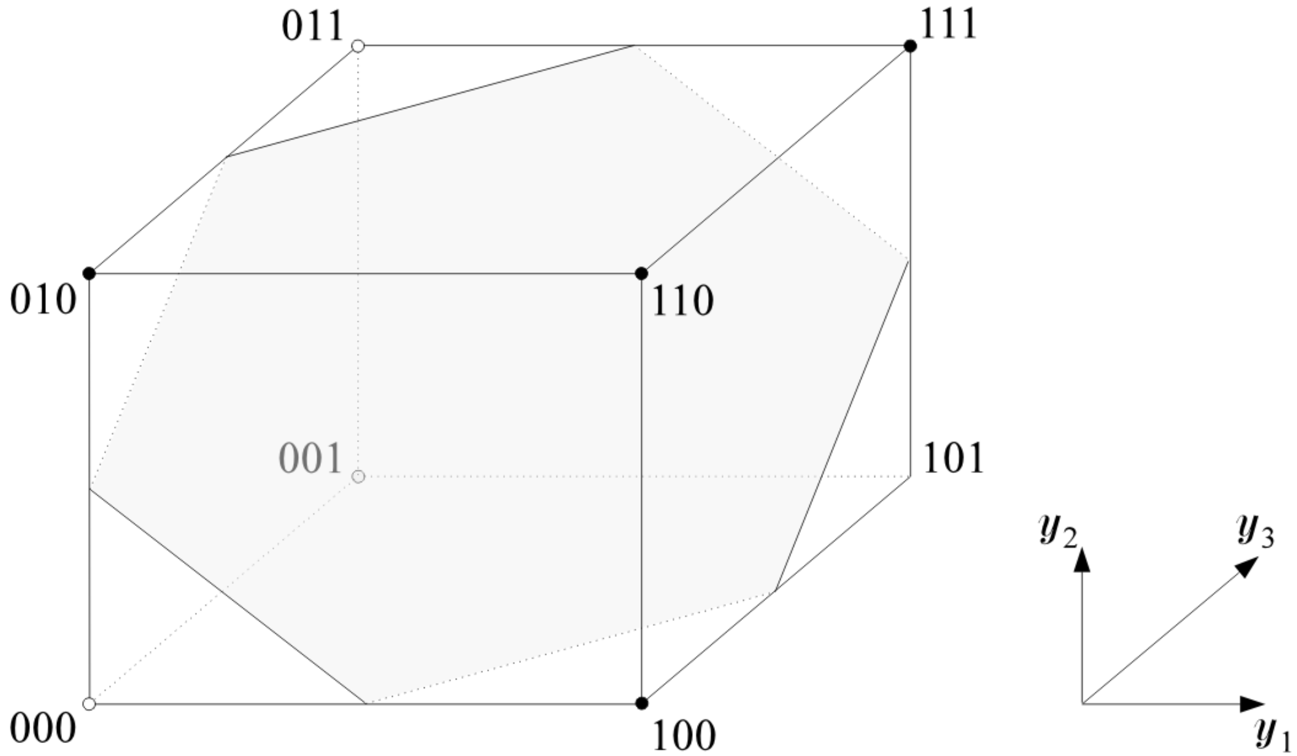


For example, the 001 vertex corresponds to the region which is located

to the (-) side of $g_1(\underline{x})=0$

to the (-) side of $g_2(\underline{x})=0$

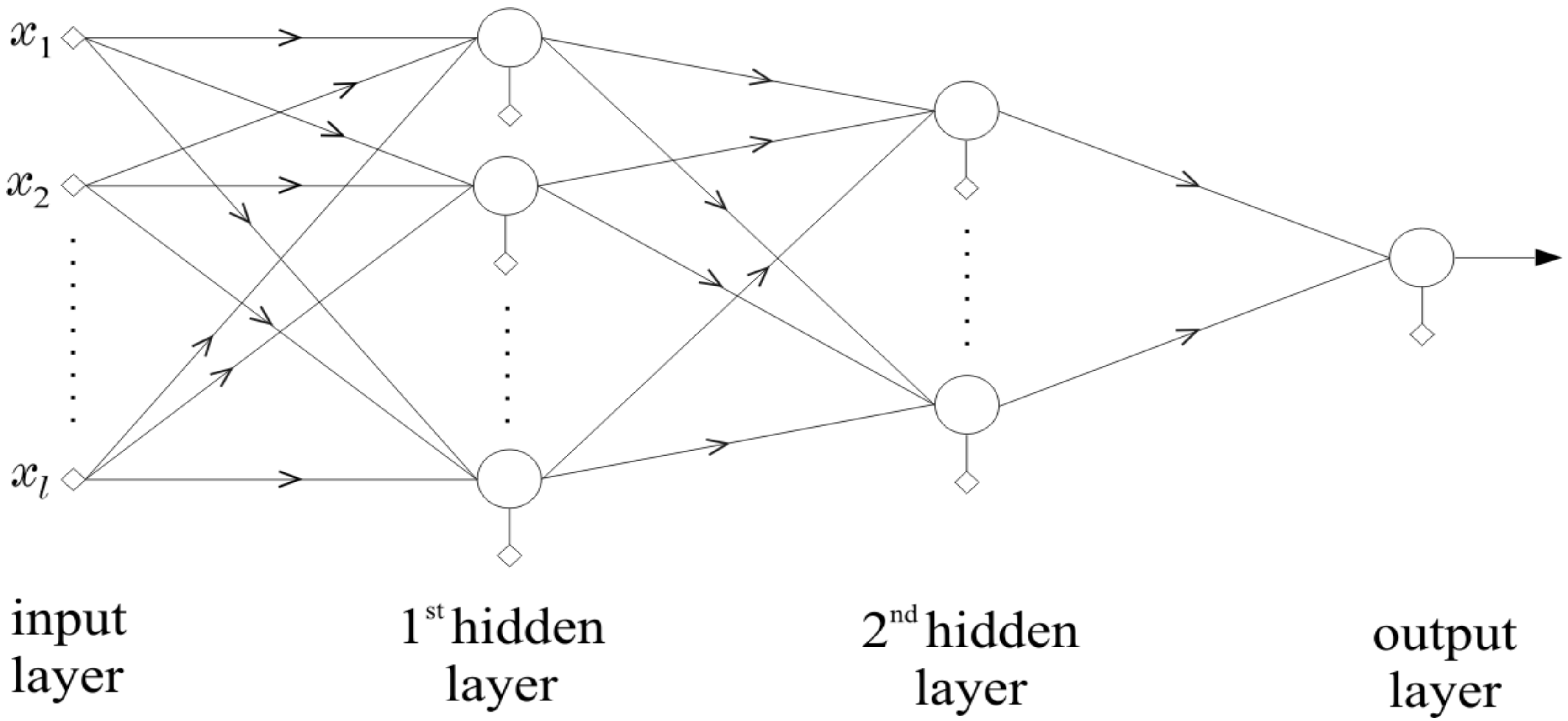
to the (+) side of $g_3(\underline{x})=0$



- The output neuron realizes a hyperplane in the transformed $\underline{y}$ space, that separates some of the vertices from the others. Thus, the two layer perceptron has the capability to classify vectors into **classes that consist of unions of polyhedral regions**. But **NOT ANY** union. It depends on the relative position of the corresponding vertices.

❖ Three layer-perceptrons

- The architecture



- This is capable to classify vectors into classes consisting of **ANY** union of polyhedral regions.

- The idea is similar to the XOR problem. It realizes more than one planes in the $\underline{y} \in R^p$ space.

- The reasoning
  - For each vertex, corresponding to class, say A, construct a hyperplane which leaves THIS vertex on one side (+) and ALL the others to the other side (-).
  - The output neuron realizes an OR gate

- Overall:

> The first layer of the network forms the hyperplanes, the second layer forms the regions and the output neuron forms the classes.

- Designing Multilayer Perceptrons
  - One direction is to adopt the above rationale and develop a structure that classifies **correctly all** the training patterns.
  - The other direction is to choose a structure and compute the synaptic weights to optimize a cost function.

❖ The Backpropagation Algorithm

- This is an algorithmic procedure that computes the synaptic weights iteratively, so that an adopted cost function is minimized (optimized)

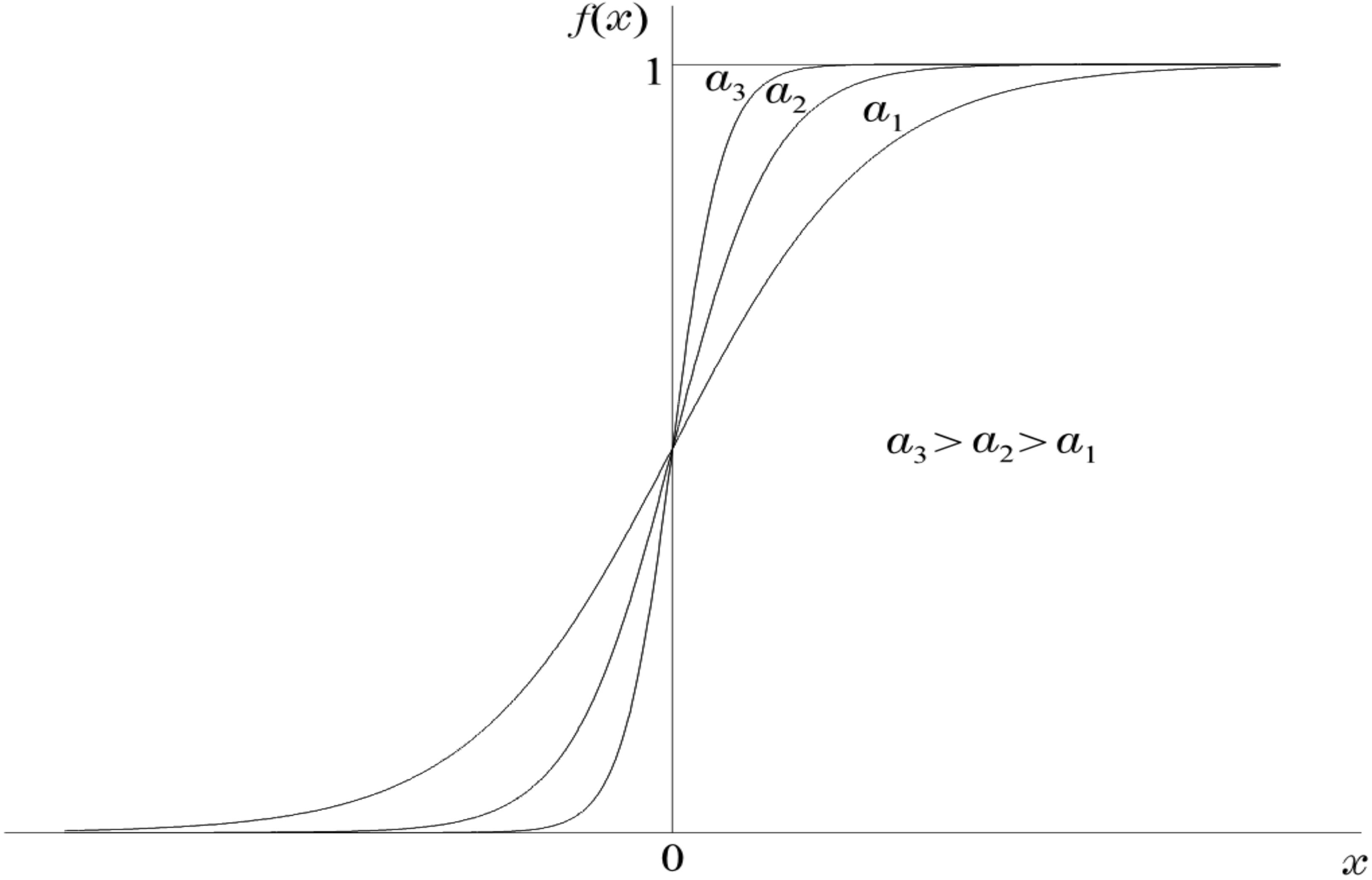
- In a large number of optimizing procedures, computation of derivatives are involved. Hence, discontinuous activation functions pose a problem, i.e.,

$$\cancel{f(x)=\begin{cases}1 & x>0\\ 0 & x<0\end{cases}}$$

- There is always an escape path!!! The logistic function

$$f(x)=\frac{1}{1+\exp(-ax)}$$

is an example. Other functions are also possible and in some cases more desirable.



$f(x)$
1
a_3
a_2
a_1
$a_3 > a_2 > a_1$
0
x

- The steps:
  - Adopt an optimizing cost function, e.g.,
    - Least Squares Error
    - Relative Entropy

    between desired responses and actual responses of the network for the available training patterns. That is, from now on we have to live with errors. We only try to minimize them, using certain criteria.

  - Adopt an algorithmic procedure for the optimization of the cost function with respect to the synaptic weights
    e.g., Gradient descent

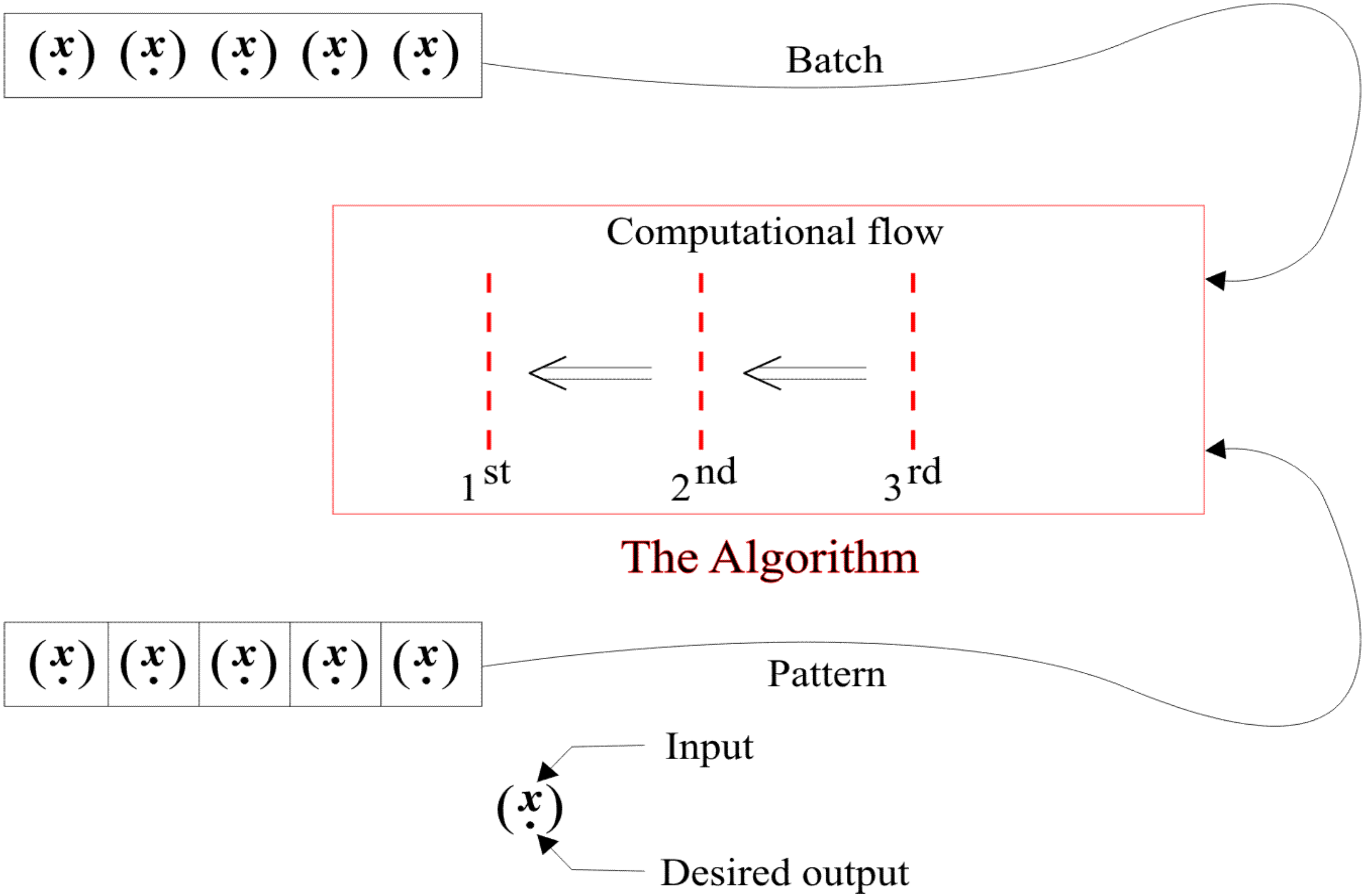
- The task is a **nonlinear** optimization one. For the gradient descent method

$$\underline{w}_1^r(\text{new}) = \underline{w}_1^r(\text{old}) + \Delta\underline{w}_1^r$$

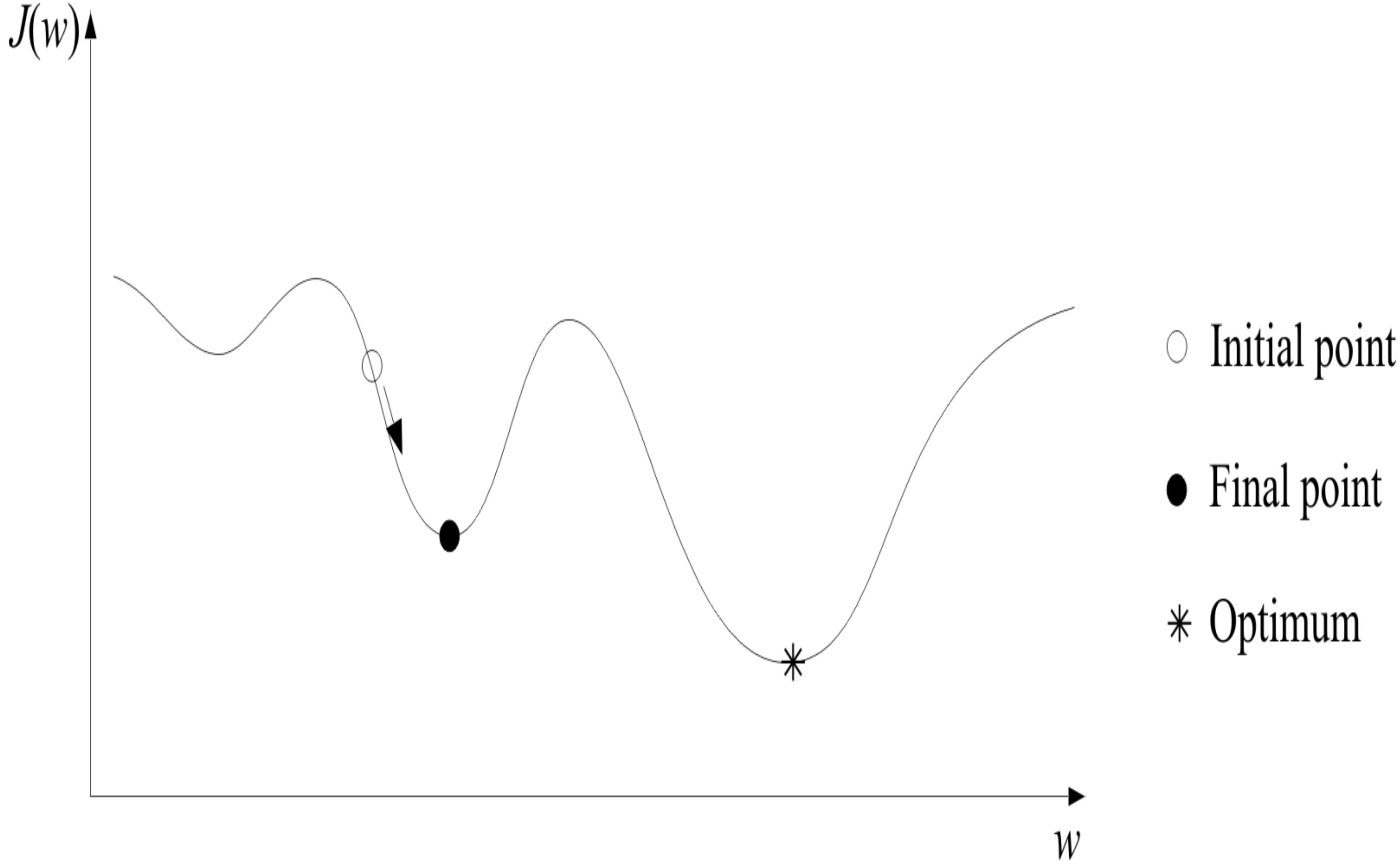
$$\Delta\underline{w}_1^r = -\mu \frac{\partial J}{\partial w_1^r}$$

- The Procedure:
  - Initialize unknown weights randomly with small values.
  - Compute the gradient terms backwards, starting with the weights of the last (3^{rd}) layer and then moving towards the first
  - Update the weights
  - Repeat the procedure until a termination procedure is met

- Two major philosophies:
  - Batch mode: The gradients of the last layer are computed once ALL training data have appeared to the algorithm, i.e., by summing up all error terms.
  - Pattern mode: The gradients are computed every time a new training data pair appears. Thus gradients are based on successive individual errors.


Batch
Computational flow
1st
2nd
3rd
The Algorithm
Pattern
Input
Desired output

➢ A major problem: The algorithm may converge to a local minimum



➢ The Cost function choice

Examples:

- The Least Squares

$$J = \sum_{i=1}^{N} E(i)$$

$$E(i) = \sum_{m=1}^{k} e_m^2(i) = \sum_{m=1}^{k} (y_m(i) - \hat{y}_m(i))^2$$

$$i = 1,2,\ldots,N$$

$y_m(i) \rightarrow$ Desired response of the m^{th} output neuron (1 or 0) for $\underline{x}(i)$

$\hat{y}_m(i) \rightarrow$ Actual response of the m^{th} output neuron, in the interval [0, 1], for input $\underline{x}(i)$

- The cross-entropy

$$J = \sum_{i=1}^{N} E(i)$$

$$E(i) = \sum_{m=1}^{k} \left\{ y_m(i) \ln \hat{y}_m(i) + (1 - y_m(i)) \ln(1 - \hat{y}_m(i)) \right\}$$

This presupposes an interpretation of y and $\hat{y}$ as **probabilities**

- Classification error rate. This is also known as discriminative learning. Most of these techniques use a smoothed version of the classification error.

- Remark 1: A common feature of all the above is the danger of local minimum convergence. "**Well formed**" cost functions guarantee convergence to a "good" solution, that is one that classifies correctly ALL training patterns, provided such a solution exists. The cross-entropy cost function **is** a well formed one. The Least Squares **is not**.

- Remark 2: Both, the Least Squares and the cross entropy lead to output values $\hat{y}_m(i)$ that approximate optimally class a-posteriori probabilities!!!

$$\hat{y}_m(i) \cong P(\omega_m | \underline{x}(i))$$

That is, the probability of class ω_m given $\underline{x}(i)$.

This is a very interesting result. It **does not** depend on the underlying distributions. It is a characteristic of **certain** cost functions. How good or bad is the approximation, depends on the underlying model. Furthermore, it is only valid at the global minimum.