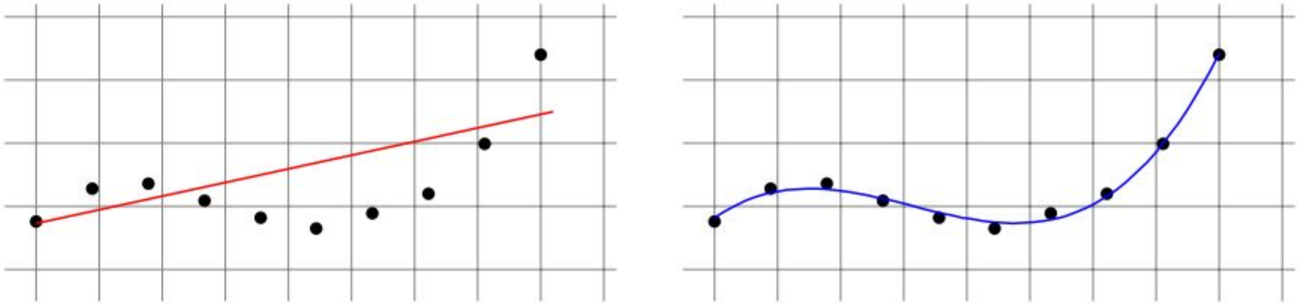
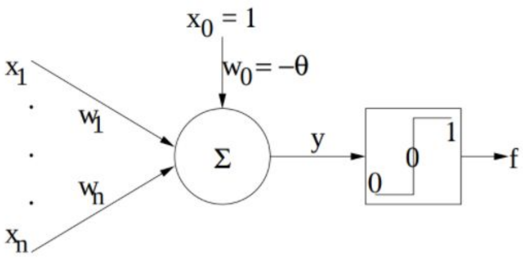
Supervised Learning

Regression / Relation to Perceptron



Perceptron



Regression Problem

- Training data: sample drawn i.i.d. from set X according to some distribution D,

$$S=((x_1,y_1),\ldots,(x_m,y_m))\in X\times Y,$$

with $Y\subseteq\mathbb{R}$ is a measurable subset.

- Loss function: $L\colon Y\times Y\to\mathbb{R}_+$ a measure of closeness, typically $L(y,y')=(y'-y)^2$ or $L(y,y')=|y'-y|^p$ for some $p\geq 1$.

- Problem: find hypothesis $h\colon X\to\mathbb{R}$ in H with small generalization error with respect to target f

$$R_D(h)=\mathop{\mathrm{E}}_{x\sim D}\Big[L\big(h(x),f(x)\big)\Big].$$

Notes

- Empirical error:

$$\widehat{R}_D(h) = \frac{1}{m} \sum_{i=1}^{m} L\big(h(x_i), y_i\big).$$

- In much of what follows:
    - $Y = \mathbb{R}$ or $Y = [-M, M]$ for some $M > 0$.
    - $L(y, y') = (y' - y)^2 \longrightarrow$ mean squared error.

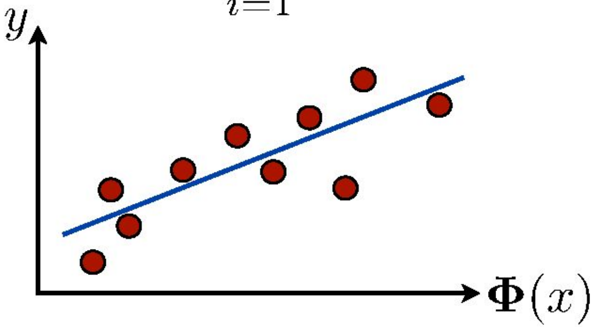
Linear Regression

- Feature mapping $\mathbf{\Phi}: X \to \mathbb{R}^N$.
- Hypothesis set: linear functions.

$$\{x \mapsto \mathbf{w} \cdot \mathbf{\Phi}(x) + b \colon \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}\}.$$

- Optimization problem: empirical risk minimization.

$$\min_{\mathbf{w},b} \; F(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \left(\mathbf{w} \cdot \mathbf{\Phi}(x_i) + b - y_i\right)^2 .$$



Linear Regression - Solution

- Rewrite objective function as $F(\mathbf{W}) = \frac{1}{m}\|\mathbf{X}^\top \mathbf{W} - \mathbf{Y}\|^2$, $\mathbf{X} = \left[\begin{smallmatrix} \Phi(x_1) & \ldots & \Phi(x_m) \\ 1 & \ldots & 1 \end{smallmatrix}\right] \in \mathbb{R}^{(N+1)\times m}$

  with $\mathbf{X}^\top = \begin{bmatrix} \mathbf{\Phi}(x_1)^\top & 1 \\ \vdots & \\ \mathbf{\Phi}(x_m)^\top & 1 \end{bmatrix}$ $\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_N \\ b \end{bmatrix}$ $\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$.

- Convex and differentiable function.

$$\nabla F(\mathbf{W}) = \frac{2}{m}\mathbf{X}(\mathbf{X}^\top \mathbf{W} - \mathbf{Y}).$$

$$\nabla F(\mathbf{W}) = 0 \Leftrightarrow \mathbf{X}(\mathbf{X}^\top \mathbf{W} - \mathbf{Y}) = 0 \Leftrightarrow \mathbf{X}\mathbf{X}^\top \mathbf{W} = \mathbf{X}\mathbf{Y}.$$

Linear Regression - Solution

- Solution:

$$\mathbf{W} = \begin{cases} (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{Y} & \text{if } \mathbf{X}\mathbf{X}^\top \text{ invertible.} \\ (\mathbf{X}\mathbf{X}^\top)^{\dagger}\mathbf{X}\mathbf{Y} & \text{in general.} \end{cases}$$

  - Computational complexity: $O(mN + N^3)$ if matrix inversion in $O(N^3)$.
  - Poor guarantees in general, no regularization.
  - For output labels in $\mathbb{R}^p$, $p > 1$, solve p distinct linear regression problems.

Higher order polynomials

The polynomial regression model

$$y_i \;=\; \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_m x_i^m + \varepsilon_i \;(i = 1, 2, \dots, n)$$

can be expressed in matrix form in terms of a design matrix $\mathbf{X}$, a response vector $\vec{y}$, a parameter vector $\vec{\beta}$, and a vector $\vec{\varepsilon}$ of random errors. The *i*-th row of $\mathbf{X}$ and $\vec{y}$ will contain the *x* and *y* value for the *i*-th data sample. Then the model can be written as a system of linear equations:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix},$$

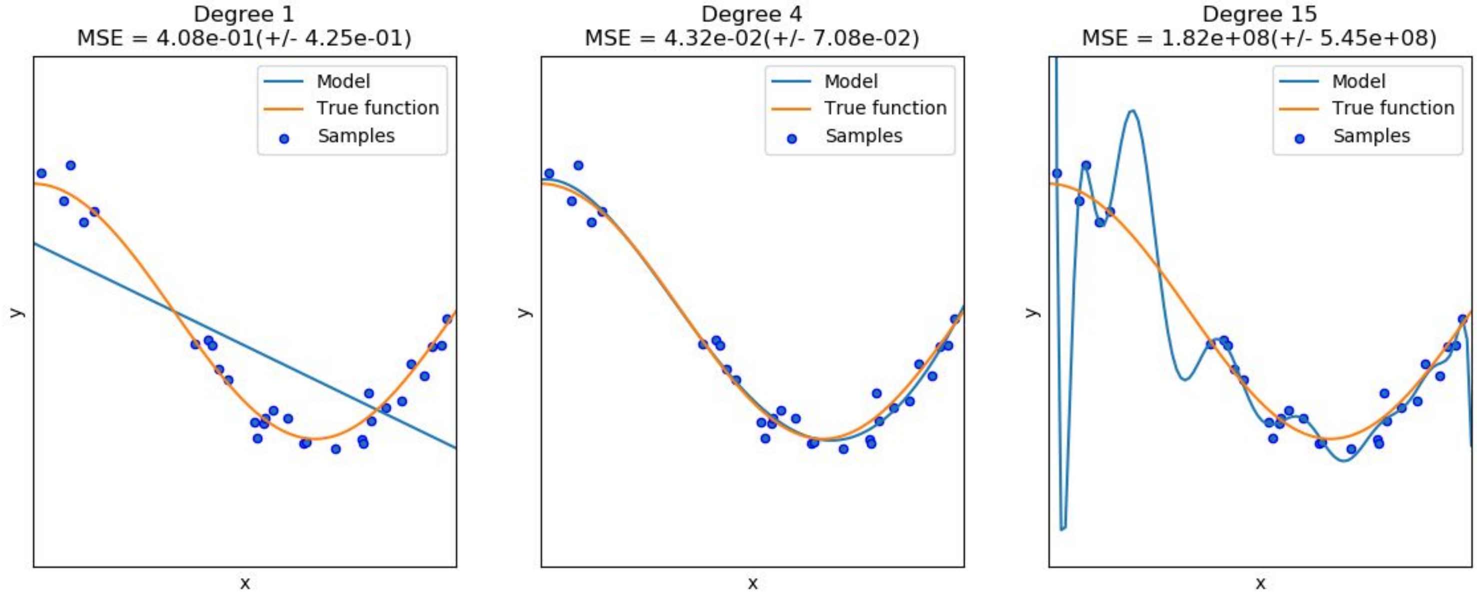
which when using pure matrix notation is written as

$$\vec{y} = \mathbf{X}\vec{\beta} + \vec{\varepsilon}.$$

The vector of estimated polynomial regression coefficients (using ordinary least squares estimation) is

$$\widehat{\vec{\beta}} = (\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\;\mathbf{X}^{\mathsf{T}}\vec{y},$$

Higher order polynomials - Overfitting



Ridge Regression

(Hoerl and Kennard, 1970)

- Optimization problem:

$$\min_{\mathbf{w}} F(\mathbf{w}, b) = \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^{m} \left(\mathbf{w} \cdot \mathbf{\Phi}(x_i) + b - y_i\right)^2,$$

where $\lambda \geq 0$ is a (regularization) parameter.

  - directly based on generalization bound.
  - generalization of linear regression.
  - closed-form solution.
  - can be used with kernels.

LASSO

(Tibshirani, 1996)

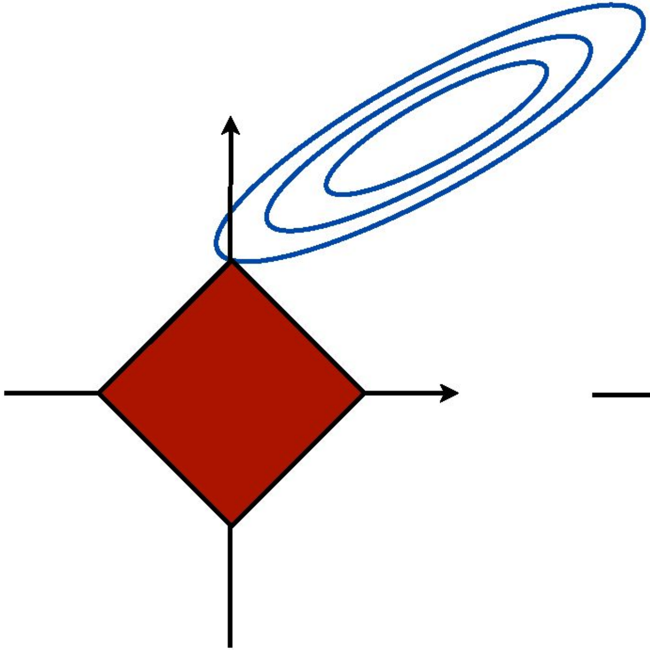
- Optimization problem: ‘least absolute shrinkage and selection operator’.

$$\min_{\mathbf{w}} F(\mathbf{w}, b) = \lambda \|\mathbf{w}\|_1 + \sum_{i=1}^{m} (\mathbf{w} \cdot \mathbf{x}_i + b - y_i)^2 ,$$

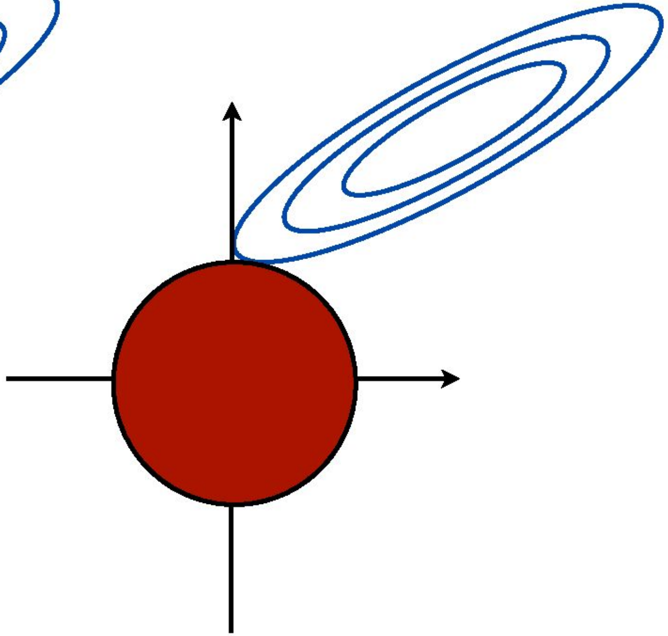
  where $\lambda \geq 0$ is a (regularization) parameter.

- Solution: equiv. convex quadratic program (QP).
  - general: standard QP solvers.
  - specific algorithm: LARS (least angle regression procedure), entire path of solutions.

Sparsity of L1 regularization



L1 regularization



L2 regularization

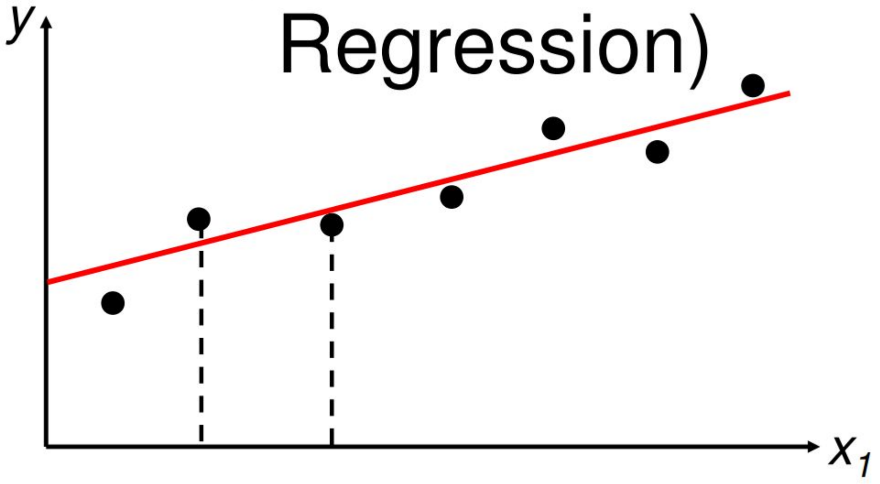
Notes

- Advantages:
  - theoretical guarantees.
  - sparse solution.
  - feature selection.
- Drawbacks:
  - no natural use of kernels.
  - no closed-form solution (not necessary, but can be convenient for theoretical analysis).

Regression

- Kernel-based methods (in Foundations)
  - Kernel ridge regression.
  - SVR.
- Many other families of algorithms: including
  - neural networks.
  - decision trees.
  - boosting trees for regression.

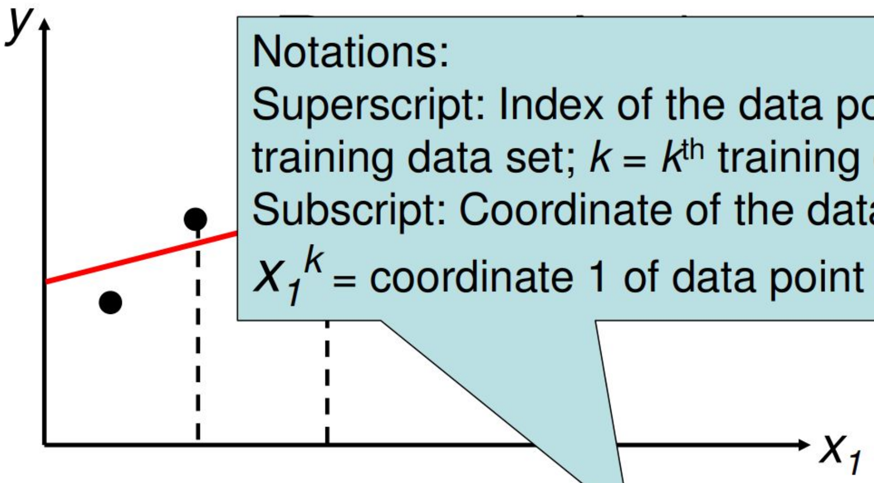
A Simple Problem (Linear Regression)



- We have training data $X = \{x_1^k\}$, i=1,..,N with corresponding output $Y = \{y^k\}$, i=1,..,N
- We want to find the parameters that predict the output Y from the data X in a linear fashion:

$$Y \approx w_o + w_1\, x_1$$

A Simple Problem (Linear

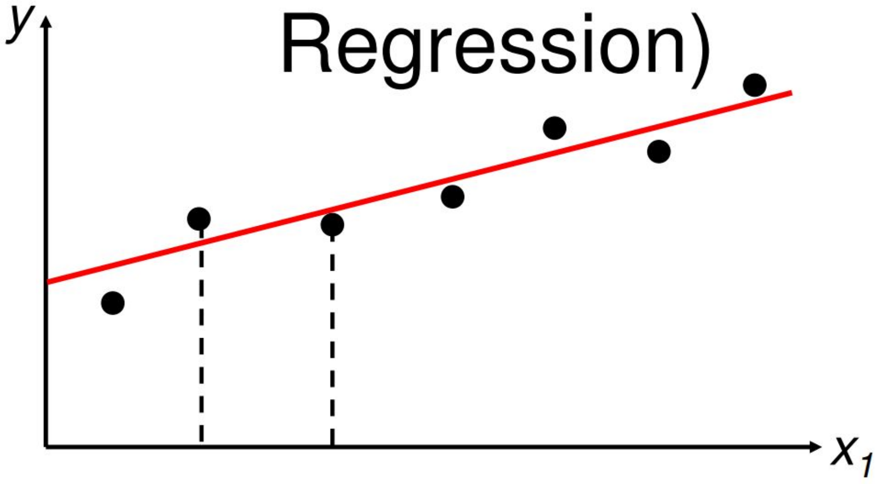


Notations:
Superscript: Index of the data point in the training data set; k = k^{th} training data point
Subscript: Coordinate of the data point;
x_1^k = coordinate 1 of data point k.

- We have training data $X = \{x_1^k\}$, $k=1,..,N$ with corresponding output $Y = \{y^k\}$, $k=1,..,N$
- We want to find the parameters that predict the output Y from the data X in a linear fashion:

$$y^k \approx w_o + w_1\, x_1^k$$

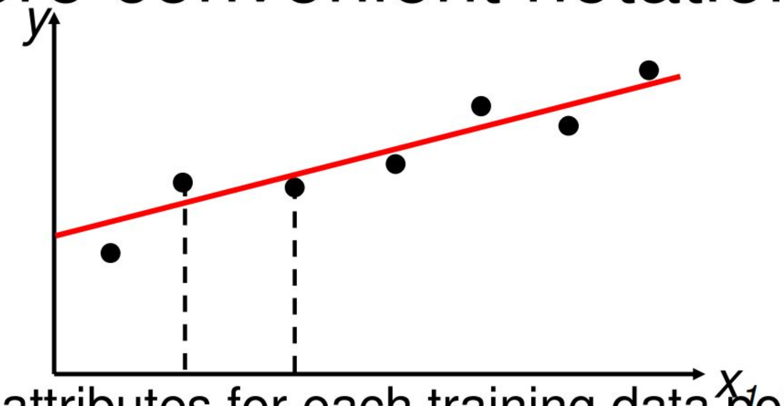
A Simple Problem (Linear Regression)



- It is convenient to define an additional "fake" attribute for the input data: $x_o = 1$
- We want to find the parameters that predict the output *Y* from the data *X* in a linear fashion:

$$y^k \approx w_o x_o^k + w_1 x_1^k$$

More convenient notations



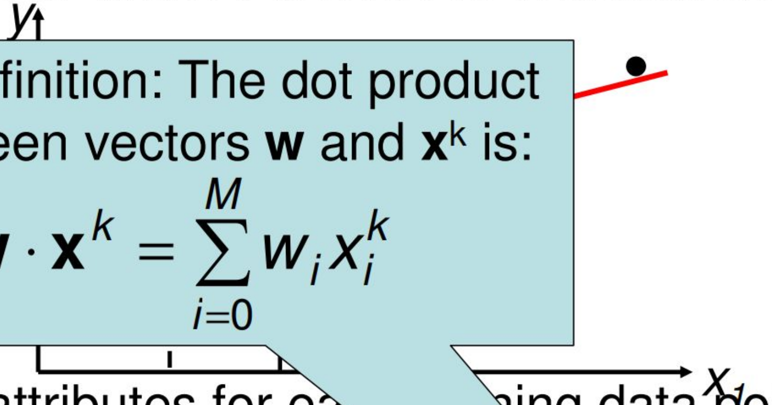
- Vector of attributes for each training data point:

$$\mathbf{x}^k = [x_o{}^k, .., x_M{}^k]$$

- We seek a vector of parameters: $\mathbf{w} = [w_o, .., w_M]$
- Such that we have a linear relation between prediction *Y* and attributes *X*:

$$y^k \approx w_o x_o^k + w_1 x_1^k + \cdots + w_M x_M^k = \sum_{i=0}^{M} w_i x_i^k = \mathbf{w} \cdot \mathbf{x}^k$$

More convenient notations



By definition: The dot product between vectors **w** and $\mathbf{x}^k$ is:

$$\mathbf{w} \cdot \mathbf{x}^k = \sum_{i=0}^{M} w_i x_i^k$$

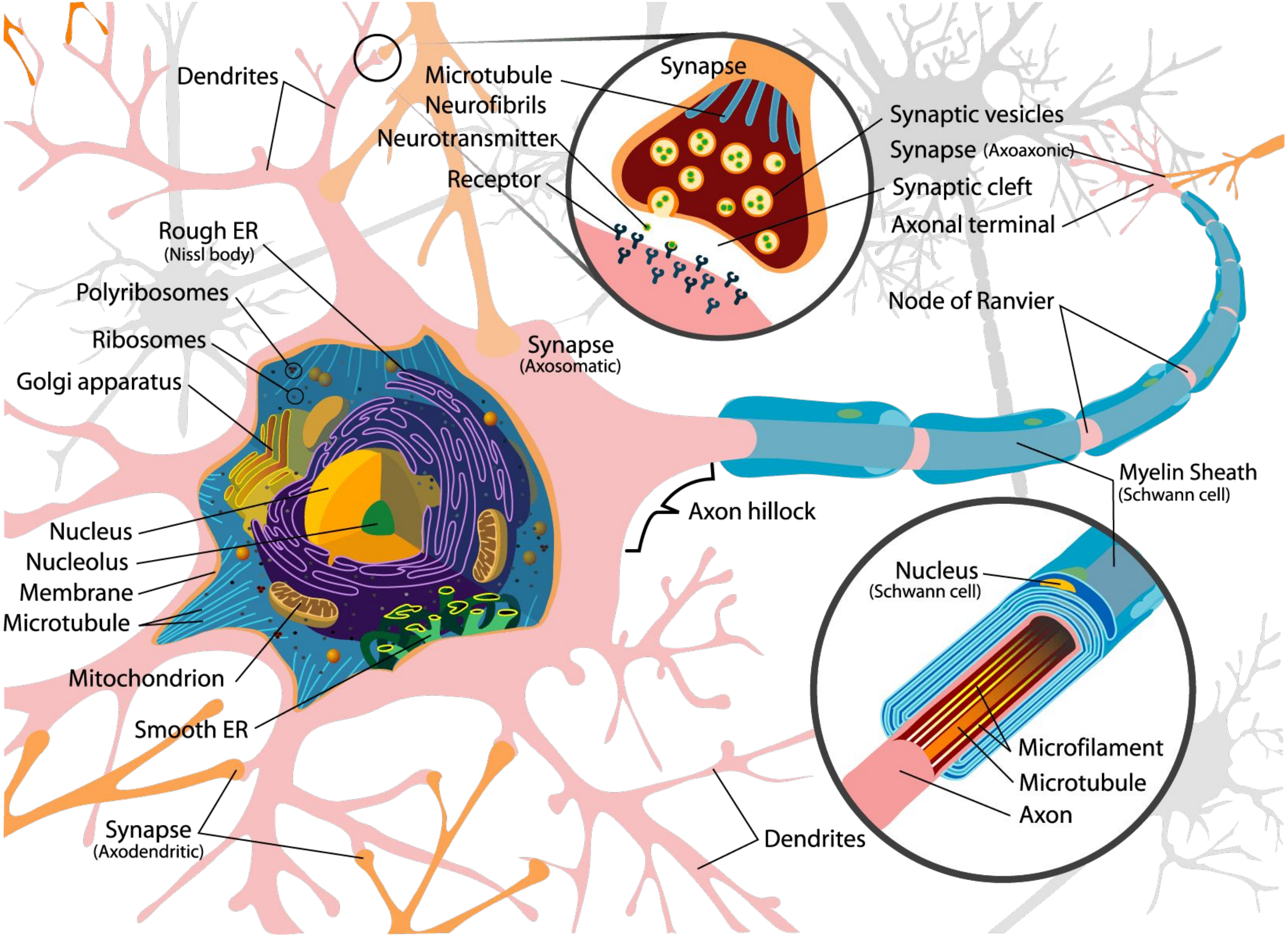
- Vector of attributes for ea… …ning data point:

$\mathbf{x}^i = [x_o^i,..,$ …

- We seek a vector of parameters: … $[w_o,..,w_M]$
- Such that we have a linear relation be… …een prediction Y and attributes X:

$$y^k \approx w_o x_o^k + w_1 x_1^k + \cdots + w_M x_M^k = \sum_{i=0}^{M} w_i x_i^k = \mathbf{w} \cdot \mathbf{x}^k$$

Neural Networks


Dendrites
Microtubule
Neurofibrils
Neurotransmitter
Receptor
Synapse
Synaptic vesicles
Synapse (Axoaxonic)
Synaptic cleft
Axonal terminal
Rough ER
(Nissl body)
Polyribosomes
Ribosomes
Golgi apparatus
Synapse
(Axosomatic)
Node of Ranvier
Myelin Sheath
(Schwann cell)
Axon hillock
Nucleus
Nucleolus
Membrane
Microtubule
Mitochondrion
Smooth ER
Nucleus
(Schwann cell)
Microfilament
Microtubule
Axon
Synapse
(Axodendritic)
Dendrites

The McCulloch-Pitts Neuron

- The first mathematical model of a neuron [Warren McCulloch and Walter Pitts, 1943]

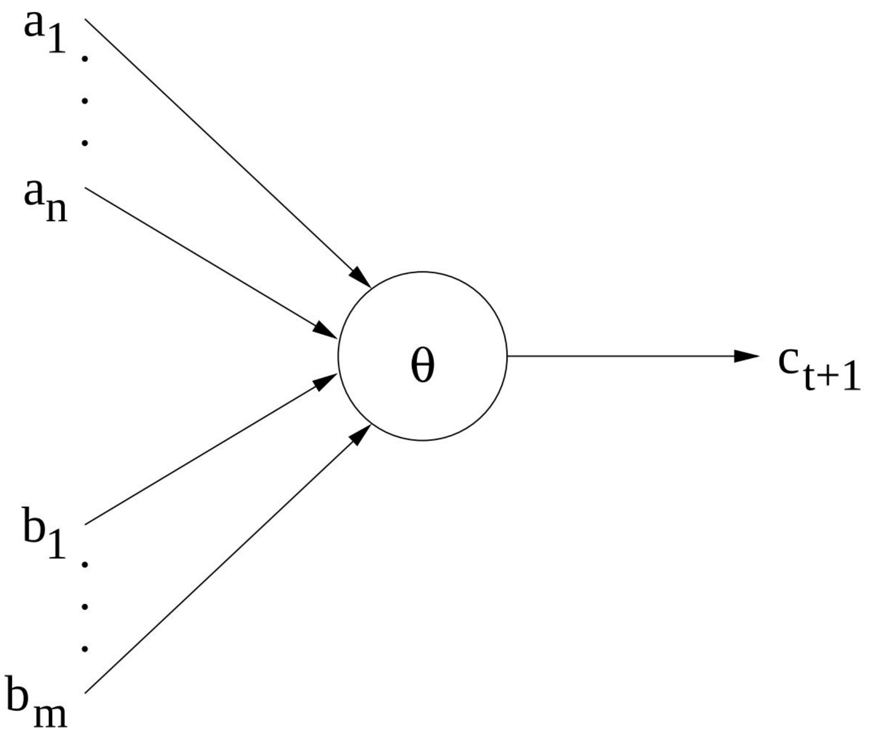
- Binary activation: *fires* (1) or *not fires* (0)

- Excitatory inputs: the a's, and
  Inhibitory inputs: the b's

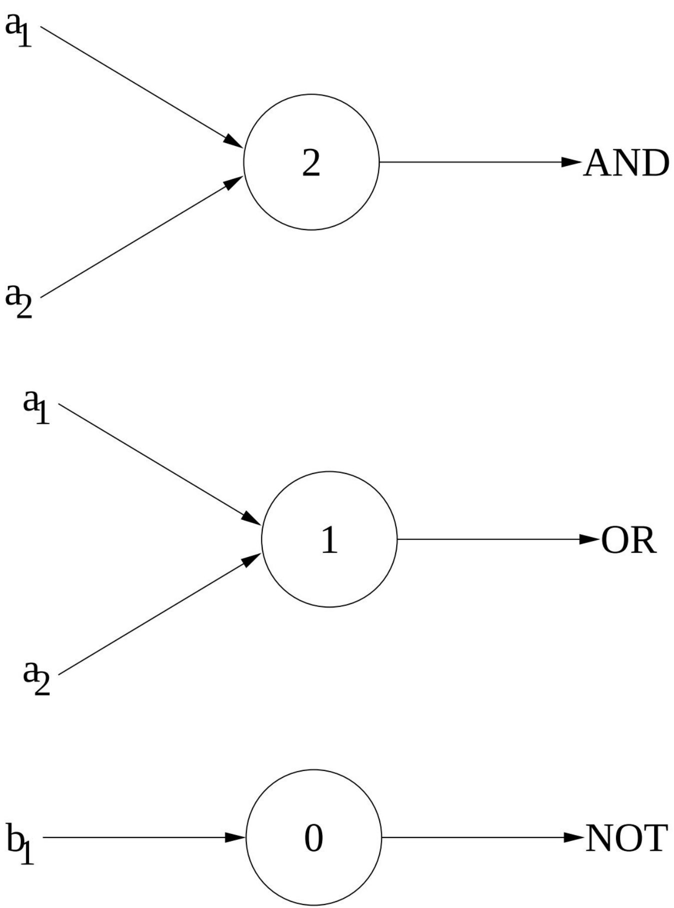
- Unit weights and fixed threshold θ

- Absolute inhibition

$$c_{t+1} = \begin{cases} 1 & \text{If } \sum_{i=0}^{n} a_{i,t} \geq \theta \text{ and } b_{1,t} = \cdots = b_{m,t} = 0 \\ 0 & \text{Otherwise} \end{cases}$$



Computing with McCulloch-Pitts Neurons

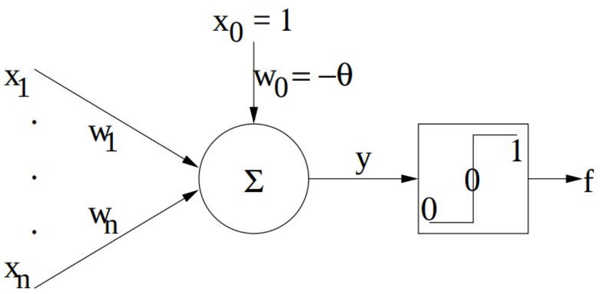


Any task or phenomenon that can be represented as a logic function can be modelled by a network of MP-neurons

- $\{$OR, AND, NOT$\}$ is functionally complete
- Any Boolean function can be implemented using OR, AND and NOT
- Canonical forms: CSOP or CPOS forms
- MP-neurons $\Leftrightarrow$ Finite State Automata

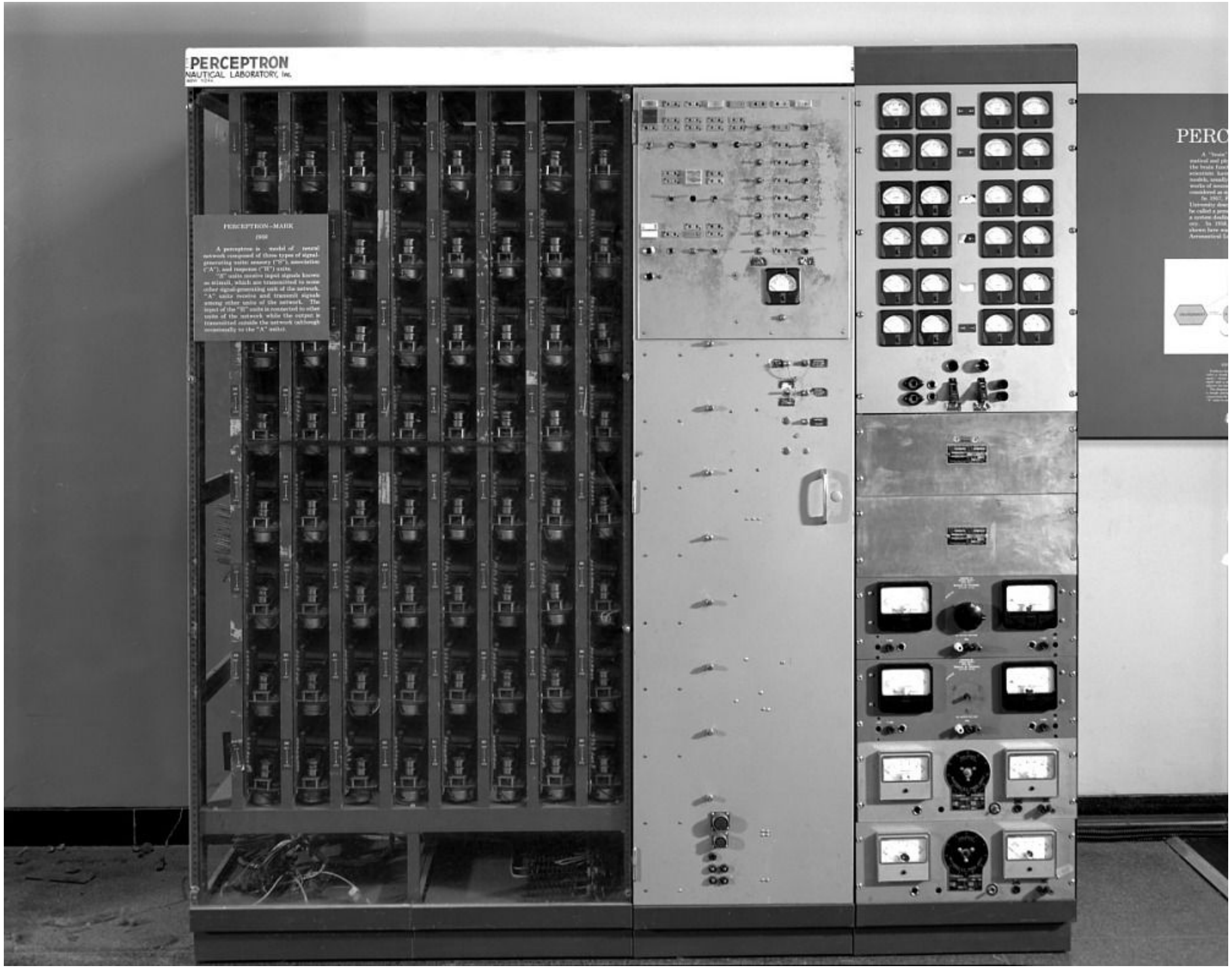
- Problems with MP-neurons
  - Weights and thresholds are analytically determined. Cannot learn
  - Very difficult to minimize size of a network
  - What about non-discrete and/or non-binary tasks?
- Perceptron solution [Rosenblatt, 1958]
  - Weights and thresholds can be determined analytically or by a learning algorithm
  - Continuous, bipolar and multiple-valued versions
  - Efficient minimization heuristics exist

Perceptron



- Architecture
  - Input: $\vec{x} = (x_0 = 1, x_1, \ldots, x_n)$
  - Weight: $\vec{w} = (w_0 = -\theta, w_1, \ldots, w_n)$, θ = bias
  - Net input: $y = \vec{w}\vec{x} = \sum_{i=0}^{n} w_i x_i$
  - Output $f(\vec{x}) = g(\vec{w}\vec{x}) = \begin{cases} 0 & \text{If } \vec{w}\vec{x} < 0 \\ 1 & \text{If } \vec{w}\vec{x} \geq 0 \end{cases}$

g: activation function

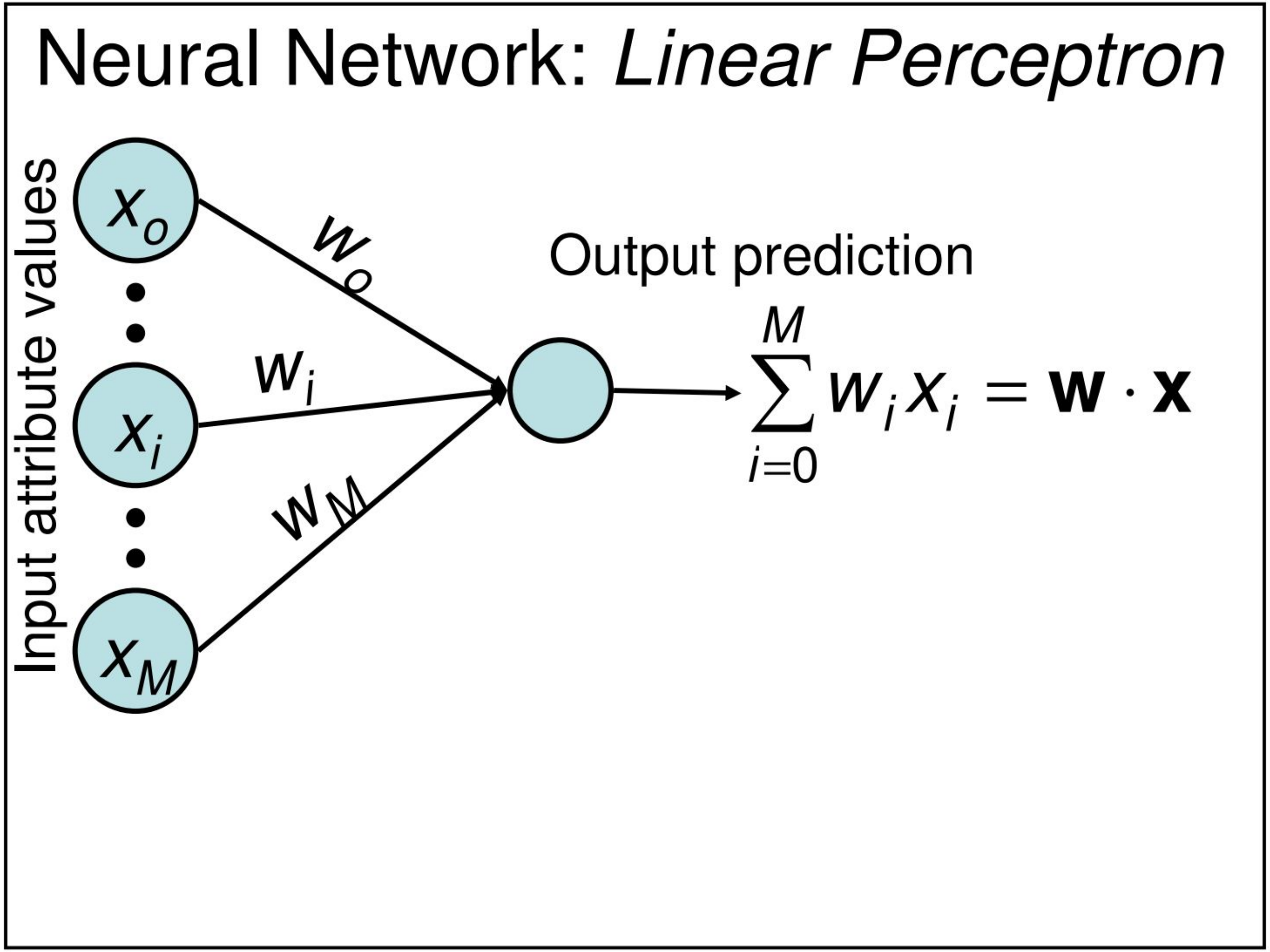


In July 1958, the U.S. Office of Naval Research unveiled a remarkable invention.

An IBM 704 – a 5-ton computer the size of a room – was fed a series of punch cards. After 50 trials, the computer taught itself to distinguish cards marked on the left from cards marked on the right.

It was a demonstration of the “perceptron” – “the first machine which is capable of having an original idea,” according to its creator, Frank Rosenblatt.

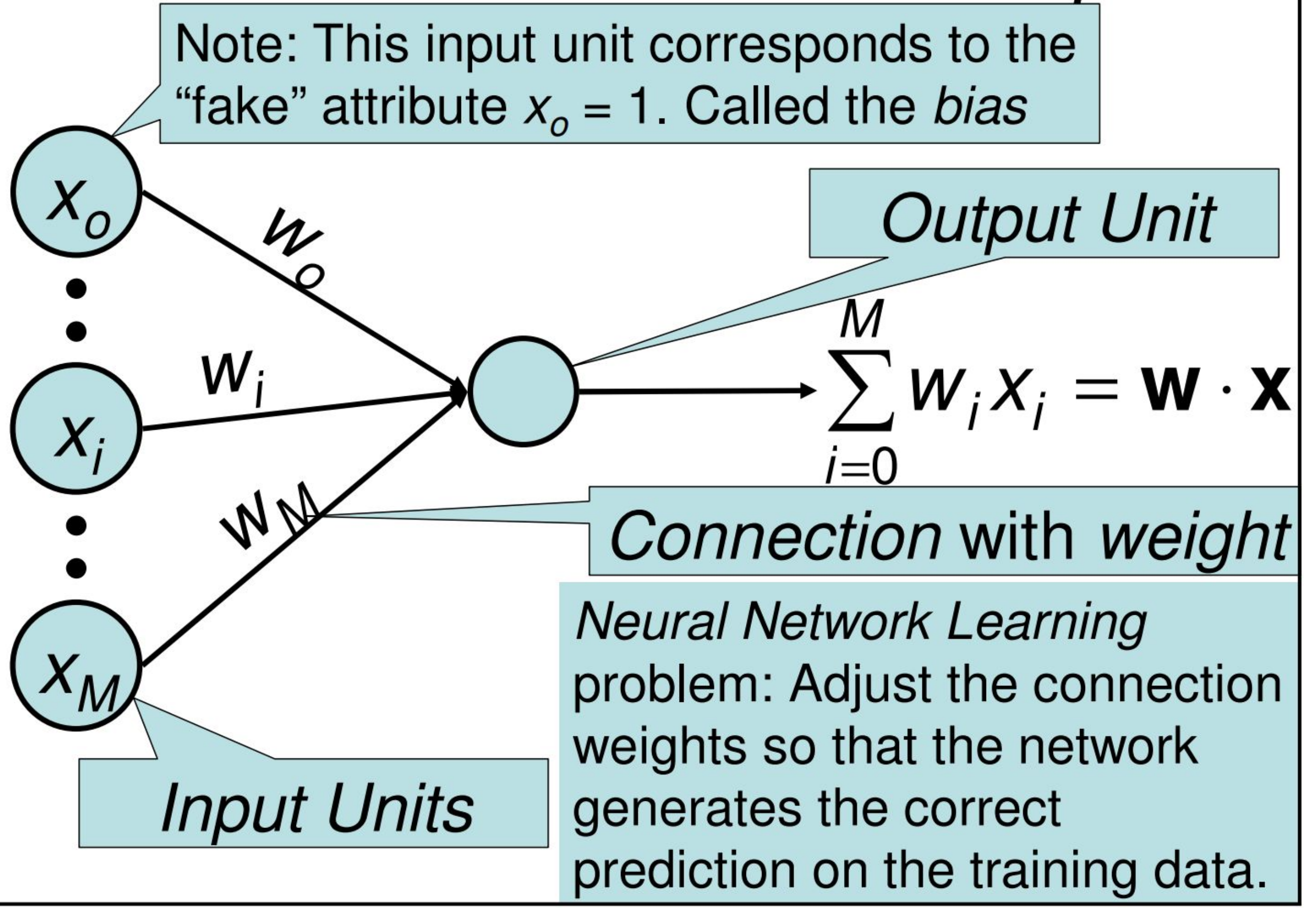
[Cornell Chronicle]



Linear: no activation function

Ισοδύναμο με τη συνάρτηση δυναμικού του ADALINE (Widrow-Hoff, 1960). Βλέπε σύγκριση Perceptron - Adaline

Neural Network: *Linear Perceptron*



Linear Regression: Gradient Descent

- We seek a vector of parameters: $\mathbf{w} = [w_o,..,w_M]$ that minimizes the error between the prediction Y and and the data X:

$$E = \sum_{k=1}^{N} \left(y^k - (w_o x_o^k + w_1 x_1^k + \cdots + w_M x_M^k) \right)^2$$

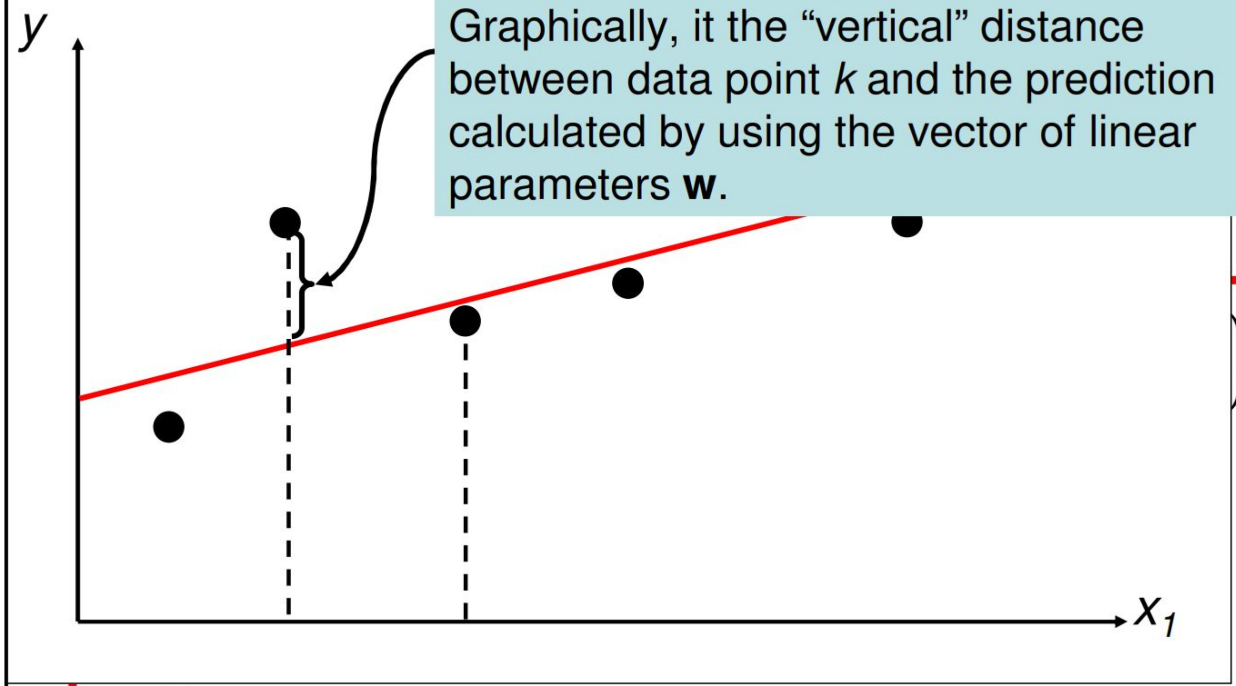
$$= \sum_{k=1}^{N} \left(y^k - \mathbf{w} \cdot \mathbf{x}^k \right)^2$$

$$= \sum_{k=1}^{N} \delta_k^2 \qquad \delta_k = y^k - \mathbf{w} \cdot \mathbf{x}^k$$

E: θα την δούμε και με 1/2 μπροστά για κανονικοποίηση

Linear R

δ_k is the error between the input **x** and the prediction *y* at data point *k*.
Graphically, it the “vertical” distance between data point *k* and the prediction calculated by using the vector of linear parameters **w**.



$$= \sum_{k=1}^{N} \delta_k^2 \qquad \delta_k = y^k - \mathbf{w} \cdot \mathbf{x}^k$$

Gradient Descent

- The minimum of E is reached when the derivatives with respect to each of the parameters w_i is zero:

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}_i} = -2\sum_{\boldsymbol{k}=1}^{\boldsymbol{N}} \left(\boldsymbol{y}^{\boldsymbol{k}} - (\boldsymbol{w}_{\boldsymbol{o}} \boldsymbol{x}_{\boldsymbol{o}}^{\boldsymbol{k}} + \boldsymbol{w}_1 \boldsymbol{x}_1^{\boldsymbol{k}} + \cdots + \boldsymbol{w}_{\boldsymbol{M}} \boldsymbol{x}_{\boldsymbol{M}}^{\boldsymbol{k}}) \right) \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}}$$

$$= -2\sum_{\boldsymbol{k}=1}^{\boldsymbol{N}} \left(\boldsymbol{y}^{\boldsymbol{k}} - \mathbf{w} \cdot \mathbf{x}^{\boldsymbol{k}} \right) \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}}$$

$$= -2\sum_{\boldsymbol{k}=1}^{\boldsymbol{N}} \delta_{\boldsymbol{k}} \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}}$$

Gradient Descent

- The minimum of E is reached when the derivatives with respect to each of the parameters w_i is zero:

$$\frac{\partial E}{\partial w_i} = -2\sum_{k=1}^{N}\left(y^k - (w_1 x_1 + \cdots + w_M x_M)\right)x_i^k$$

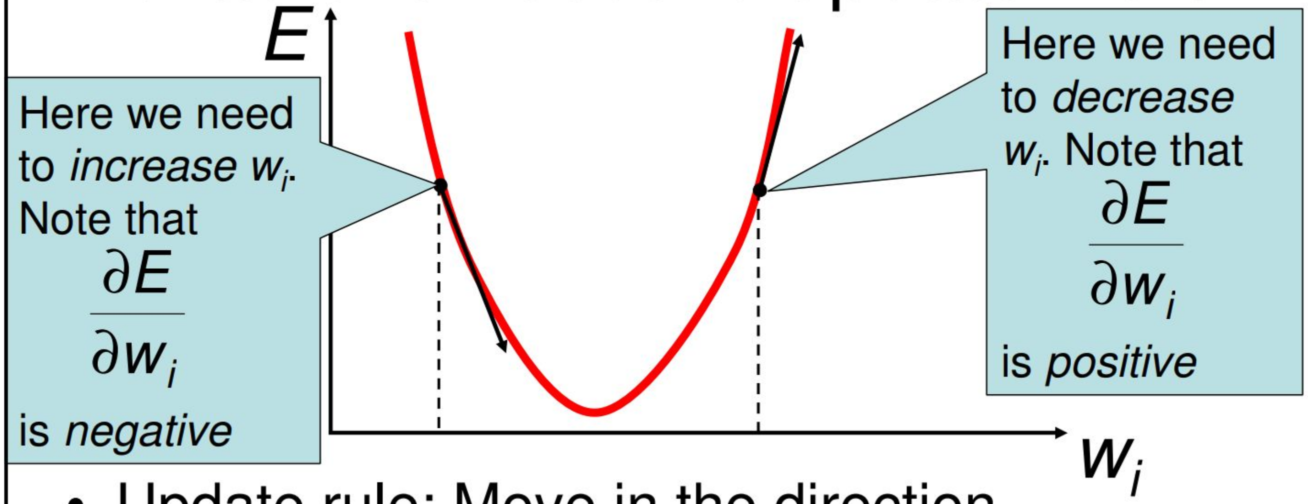
$$= -2\sum_{k=1}^{N}\left(y^k - w^T x^k\right)x_i^k$$

$$= -2\sum_{k=1}^{N}\delta_k x_i^k$$

Note that the contribution of training data element number k to the overall gradient is $-\delta_k x_i^k$

Δείτε και το “Single-Layer Neural Networks and Gradient Descent” του Raschka με παραδείγματα σε Python

Gradient Descent Update Rule



- Update rule: Move in the direction opposite to the gradient direction

$$w_i \leftarrow w_i - \alpha \frac{\partial E}{\partial w_i}$$

Perceptron Training

- Given input training data x^k with corresponding value y^k

1. Compute error:

$$\delta_k \leftarrow y^k - \mathbf{w} \cdot \mathbf{x}^k$$

2. Update NN weights:

$$w_i \leftarrow w_i + \alpha \delta_k x_i^k$$

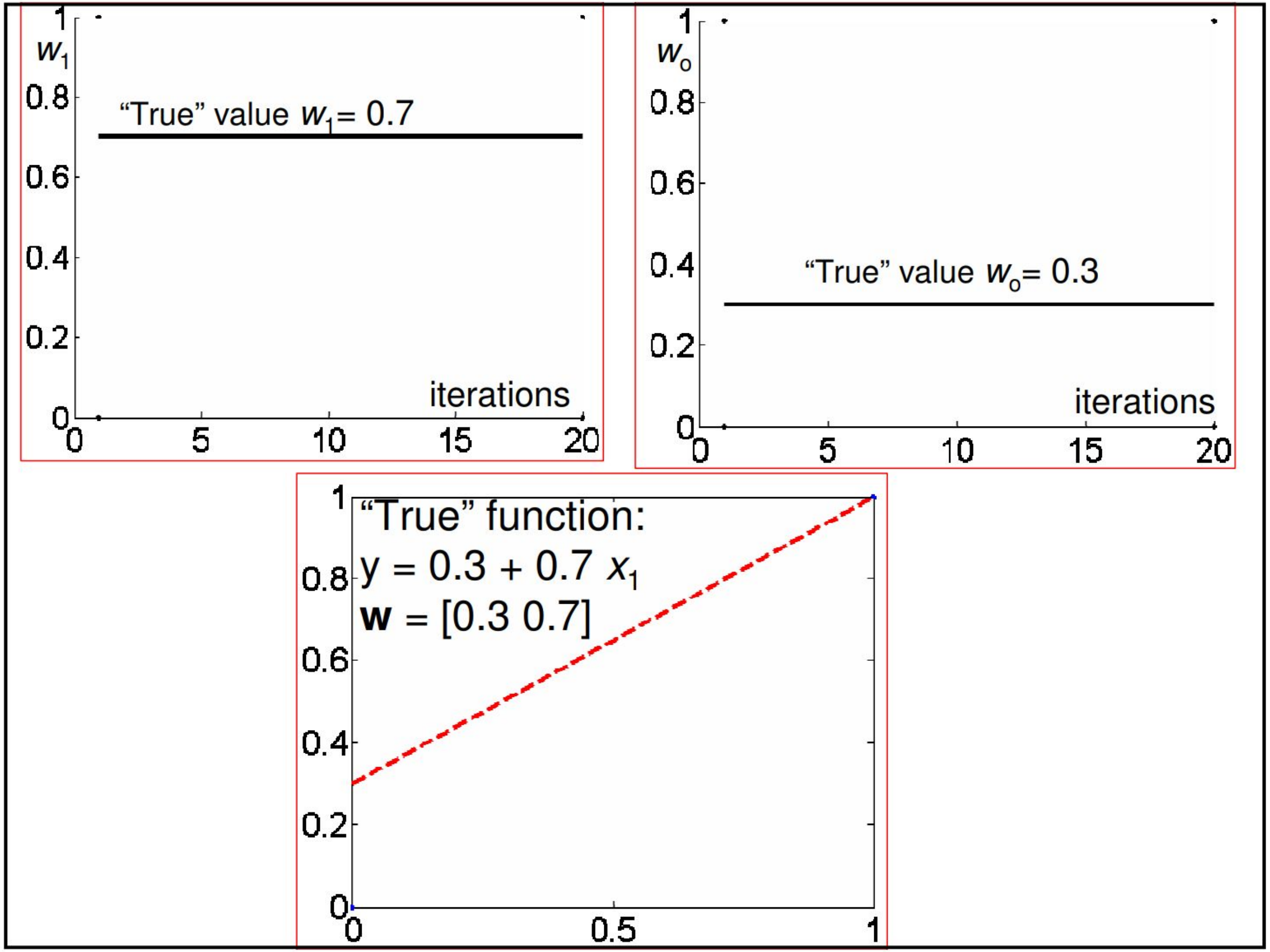
α is the learning rate.
α too small: May converge slowly and may need a lot of training examples
α too large: May change **w** too quickly and spend a long time oscillating around the minimum.

1. Compute error:

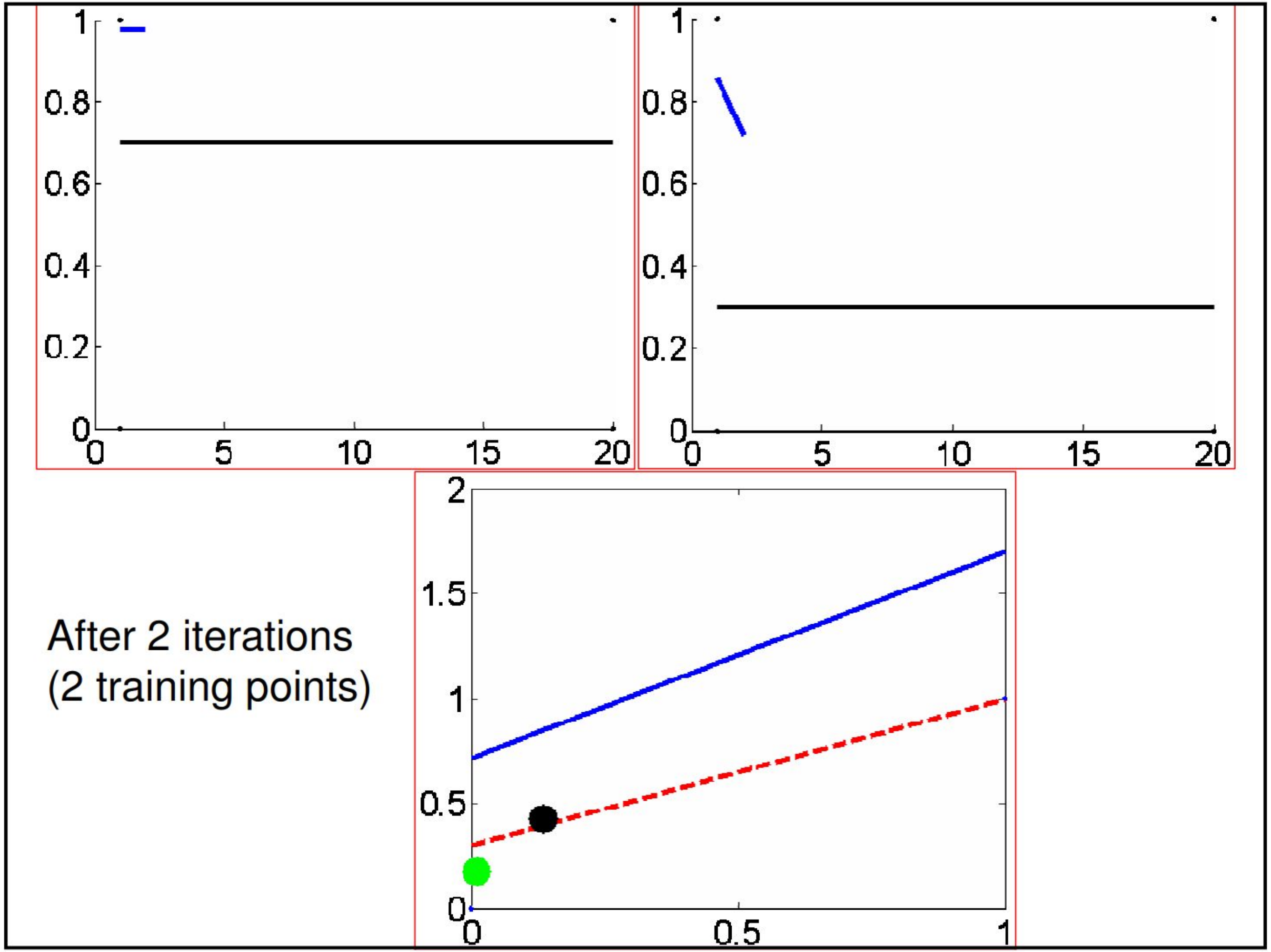
$$\delta_k \leftarrow y^k - \ldots \cdot \mathbf{x}^k$$

2. Update NN weights:

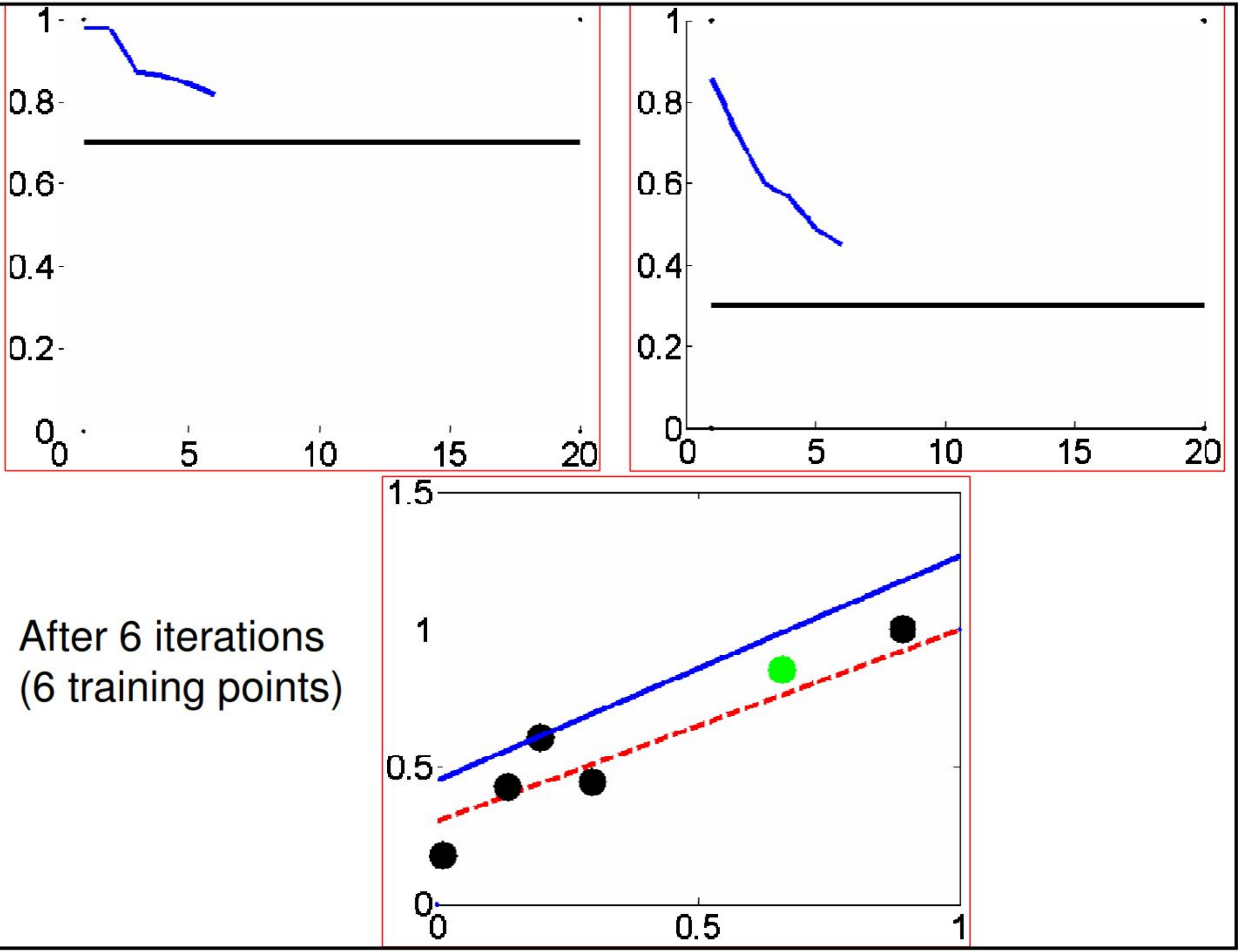
$$w_i \leftarrow w_i + \alpha \delta_k x_i^k$$



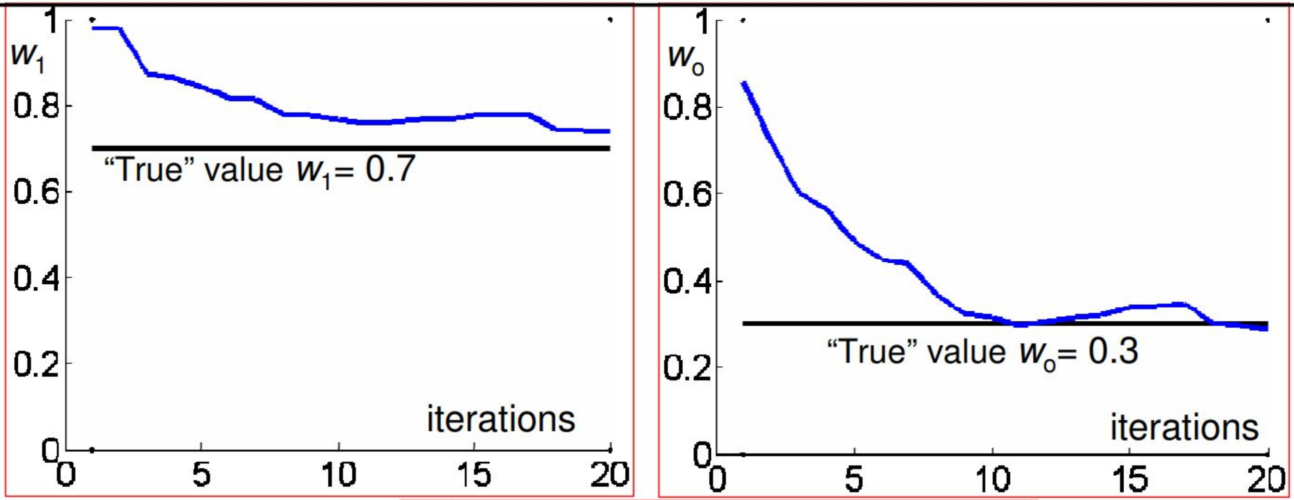
1
w_1
0.8
"True" value w_1= 0.7
0.6
0.4
0.2
iterations
0
0
5
10
15
20
1
w_o
0.8
0.6
0.4
"True" value w_o= 0.3
0.2
iterations
0
0
5
10
15
20
1
"True" function:
$y = 0.3 + 0.7\ x_1$
0.8
$\mathbf{w}$ = [0.3 0.7]
0.6
0.4
0.2
0
0
0.5
1



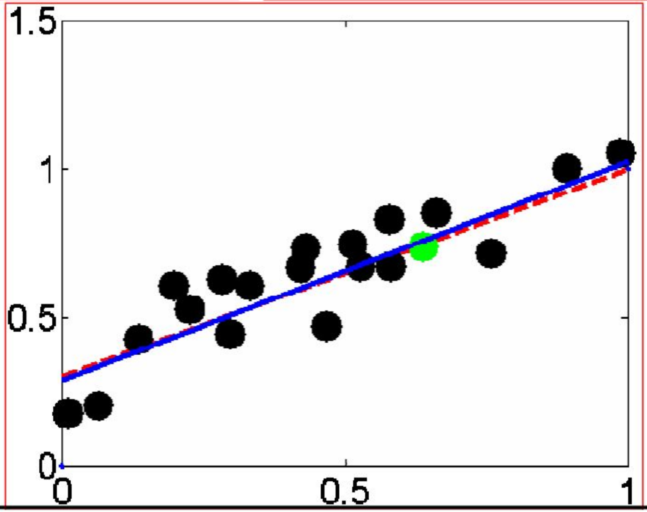
After 2 iterations
(2 training points)



After 6 iterations
(6 training points)



After 20 iterations
(20 training points)

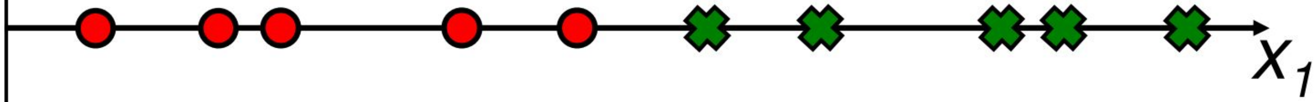


Perceptrons: Remarks

- Update has many names: delta rule, gradient rule, LMS rule…..
- Update is ***guaranteed*** to converge to the best linear fit (global minimum of *E*)
- Of course, there are more direct ways of solving the linear regression problem by using linear algebra techniques. It boils down to a simple matrix inversion (not shown here).
- In fact, the perceptron training algorithm can be much, much slower than the direct solution

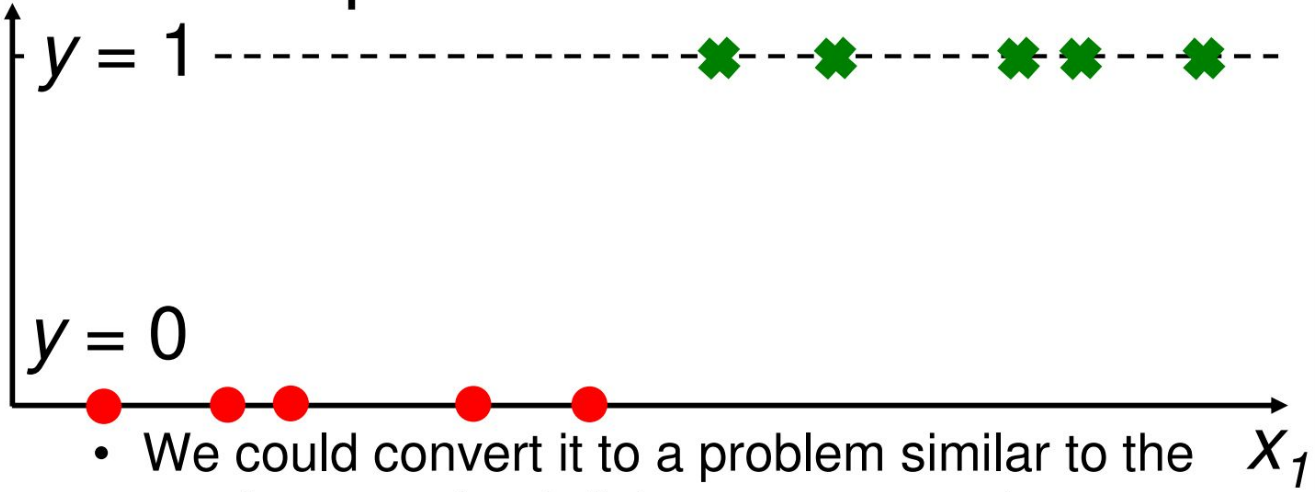
A Simple Classification Problem

Training data:



- Suppose that we have one attribute x_1
- Suppose that the data is in two classes (red dots and green dots)
- Given an input value x_1, we wish to predict the most likely class.

A Simple Classification Problem

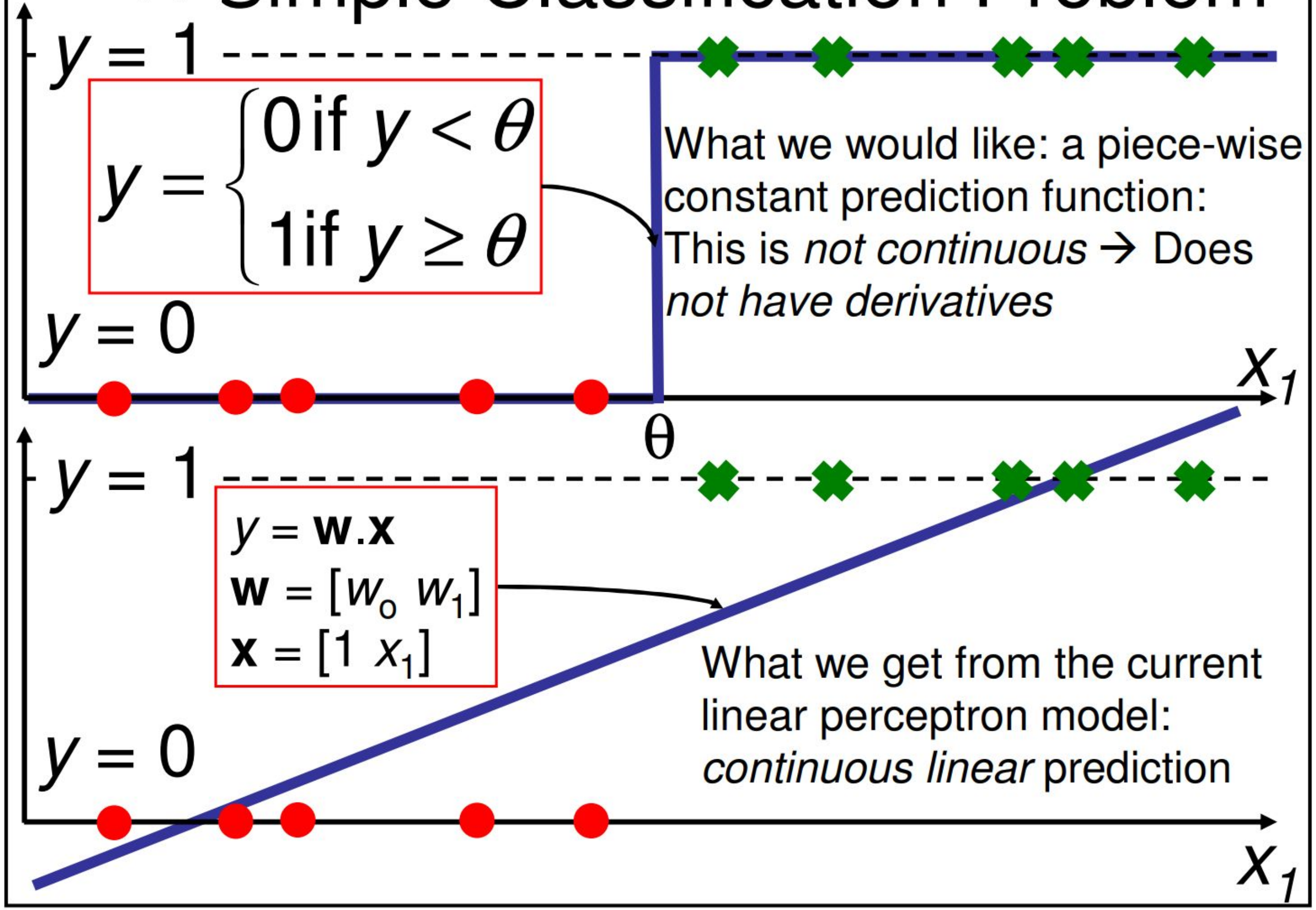


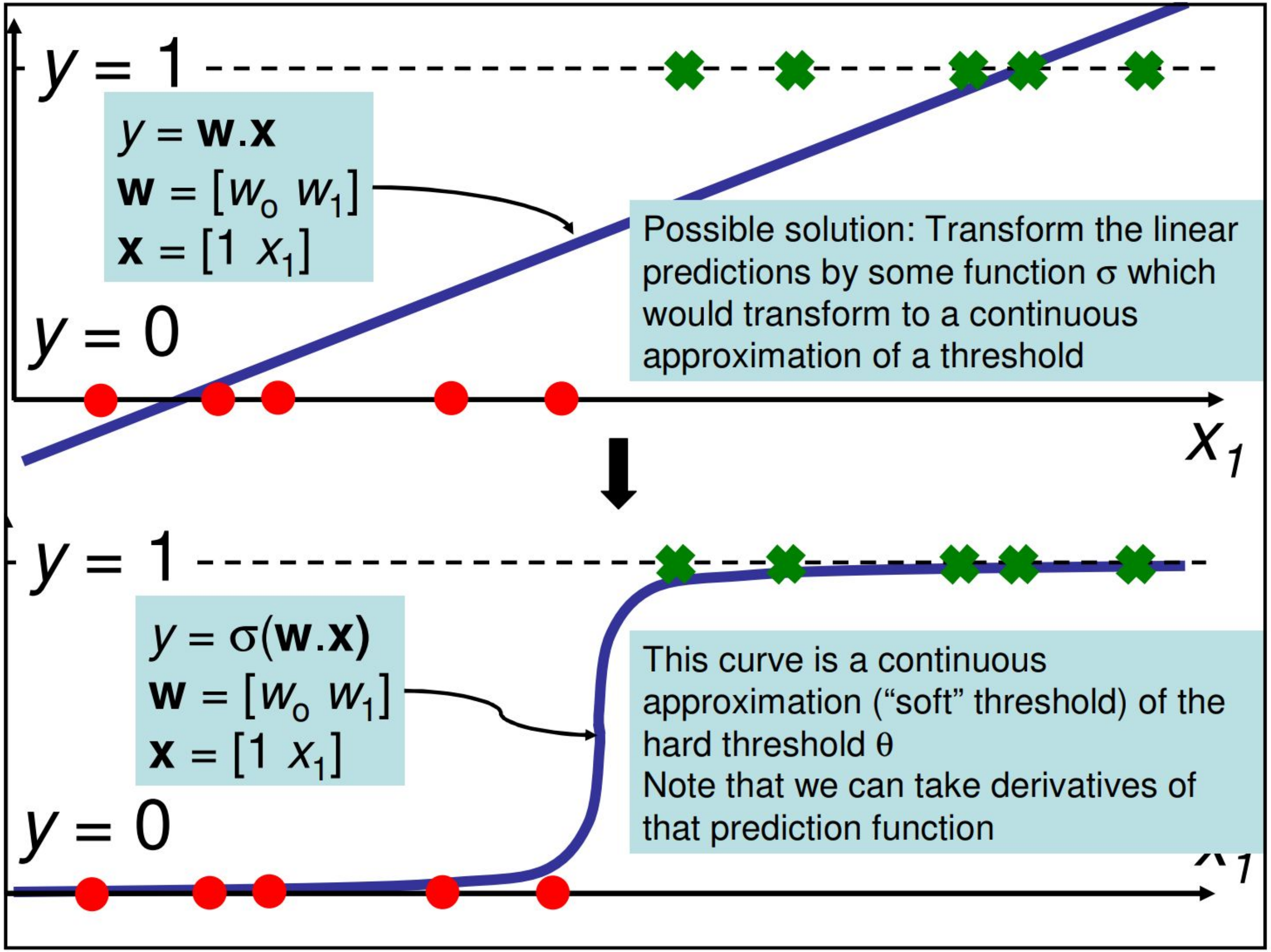
- We could convert it to a problem similar to the previous one by defining an output value y

$$y = \begin{cases} 0 \text{ if in red class} \\ 1 \text{ if in green class} \end{cases}$$

- The problem now is to learn a mapping between the attribute x_1 of the training examples and their corresponding class output y

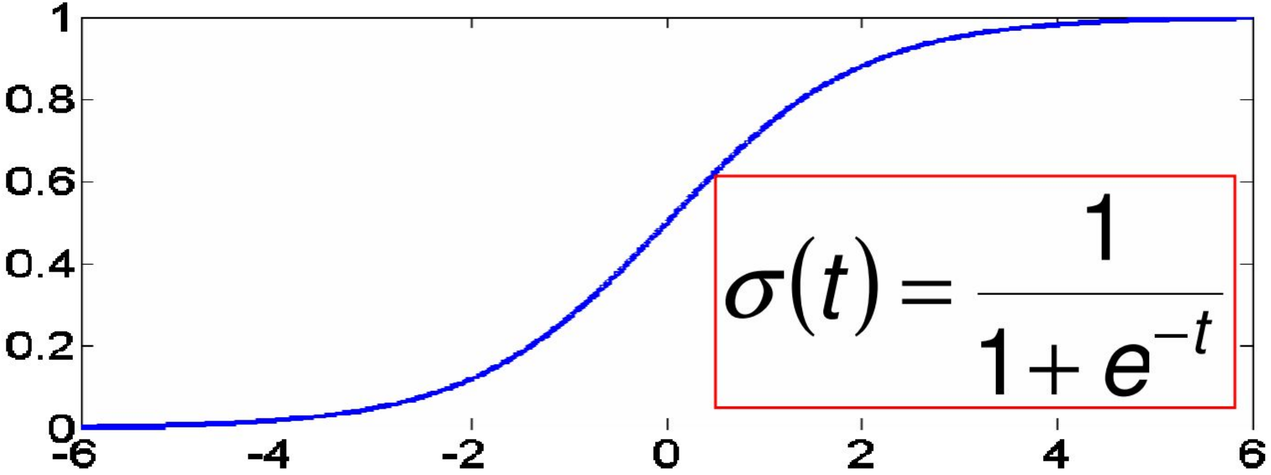
A Simple Classification Problem





y = 1
y = w.x
w = [w0 w1]
x = [1 x1]
Possible solution: Transform the linear predictions by some function σ which would transform to a continuous approximation of a threshold
y = 0
x1
y = 1
y = σ(w.x)
w = [w0 w1]
x = [1 x1]
This curve is a continuous approximation ("soft" threshold) of the hard threshold θ
Note that we can take derivatives of that prediction function
y = 0

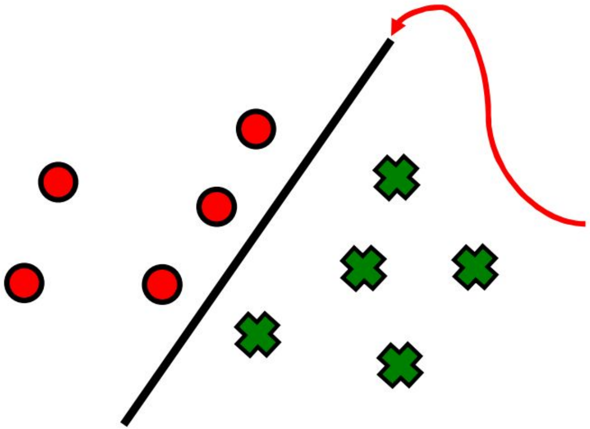
The Sigmoid Function



$$\sigma(t) = \frac{1}{1+e^{-t}}$$

- Note: It is ***not*** important to remember the exact expression of σ (in fact, alternate definitions are used for σ). What is important to remember is that:
  - It is smooth and has a derivative σ’ (exact expression is unimportant)
  - It approximates a hard threshold function at $x = 0$

Generalization to *M* Attributes



A linear separation is parameterized like a line:

$$\sum_{i=0}^{M} w_i x_i = \mathbf{w} \cdot \mathbf{x} = 0$$

- Two classes are linearly separable if they can be separated by a linear combination of the attributes:
  - Threshold in 1-d
  - Line in 2-d
  - Plane in 3-d
  - Hyperplane in *M*-d

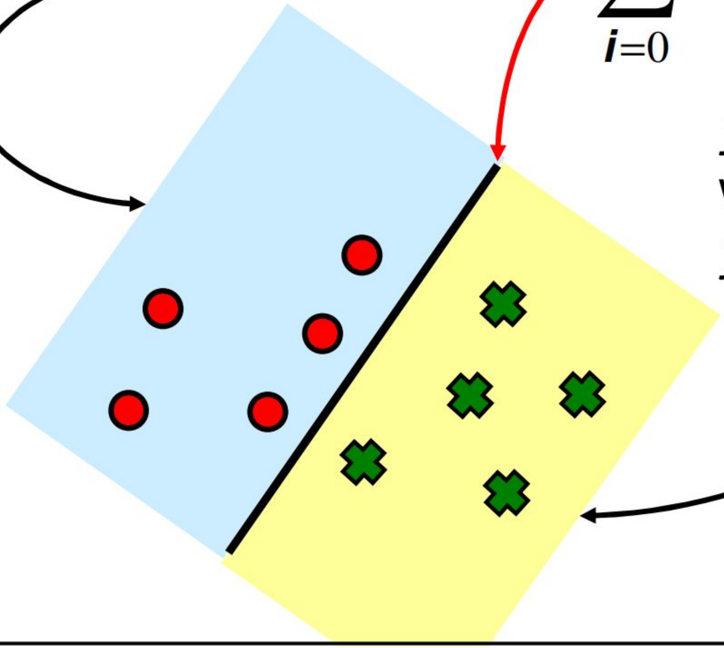
Generalization to *M* Attributes

y = 0 in this region, we can approximate *y* by σ(**w**.**x**) ≈ 0

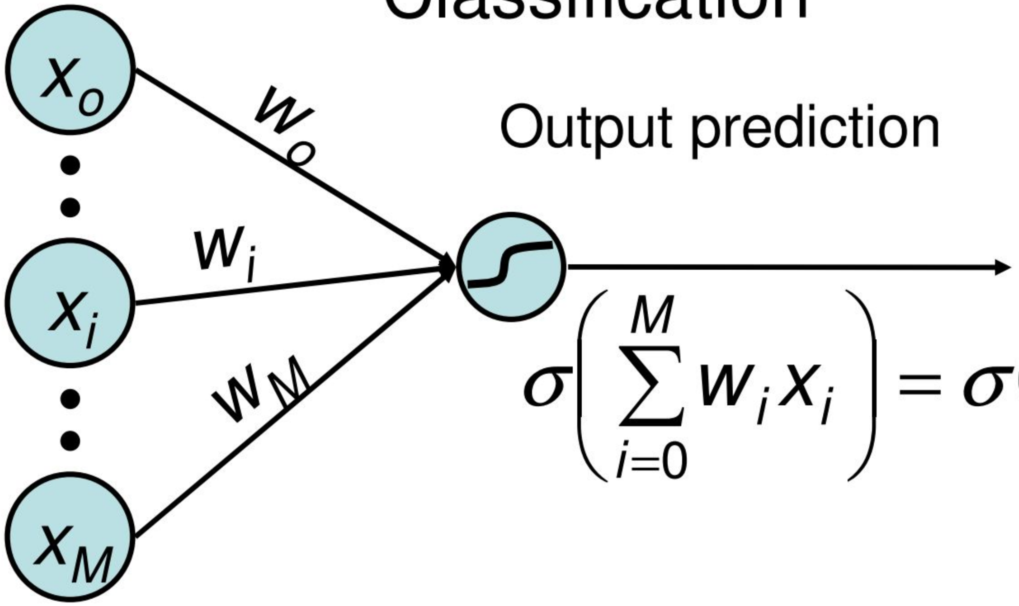
A linear separation is parameterized like a line:

$$\sum_{i=0}^{M} w_i x_i = \mathbf{w} \cdot \mathbf{x} = 0$$

y = 1 in this region, we can approximate *y* by σ(**w**.**x**) ≈ 1



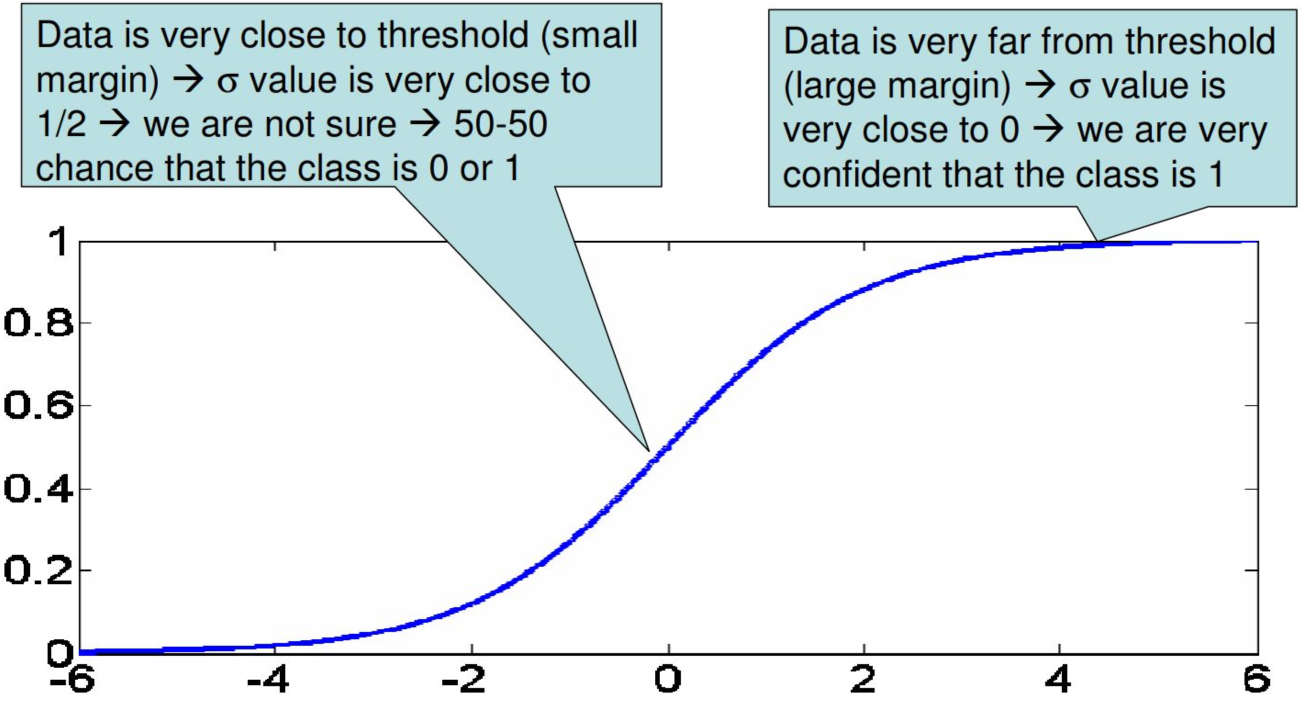
Single Layer Network for Classification



$$\sigma\left(\sum_{i=0}^{M} w_i x_i\right) = \sigma(\mathbf{w} \cdot \mathbf{x})$$

- Term: Single-layer Perceptron

Interpreting the Squashing Function



- Roughly speaking, we can interpret the output as how confident we are in the classification: Prob(y=1|x)

Training

- Given input training data x^k with corresponding value y^k

1. Compute error:

$$\delta_k \leftarrow y^k - \sigma(\mathbf{w} \cdot \mathbf{x}^k)$$

2. Update NN weights:

$$w_i \leftarrow w_i + \alpha \delta_k x_i^k \sigma'(\mathbf{w} \cdot \mathbf{x}^k)$$

Note: It is exactly the same as before, except for the additional complication of passing the linear output through σ

- Given input tra ata x^k with corresponding val

1. Compute error:

$$\delta_k \leftarrow y^k - \sigma\left(\mathbf{w} \cdot \mathbf{x}^k\right)$$

This formula derived by direct application of the chain rule from calculus

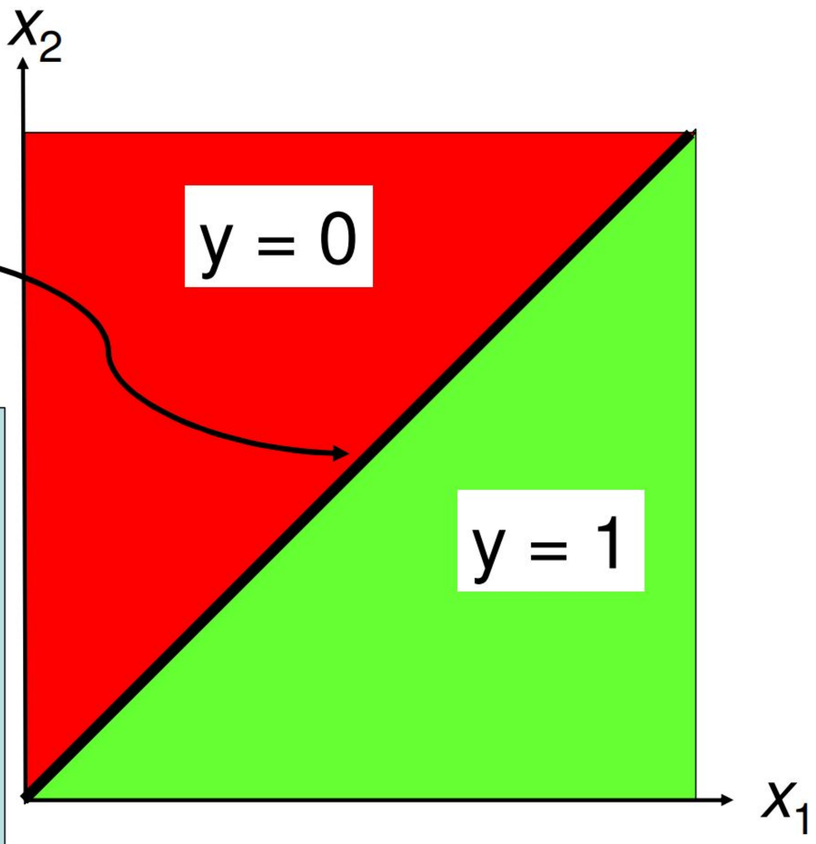
$$w_i \leftarrow w_i + \alpha \delta_k x_i^k \sigma'\left(\mathbf{w} \cdot \mathbf{x}^k\right)$$

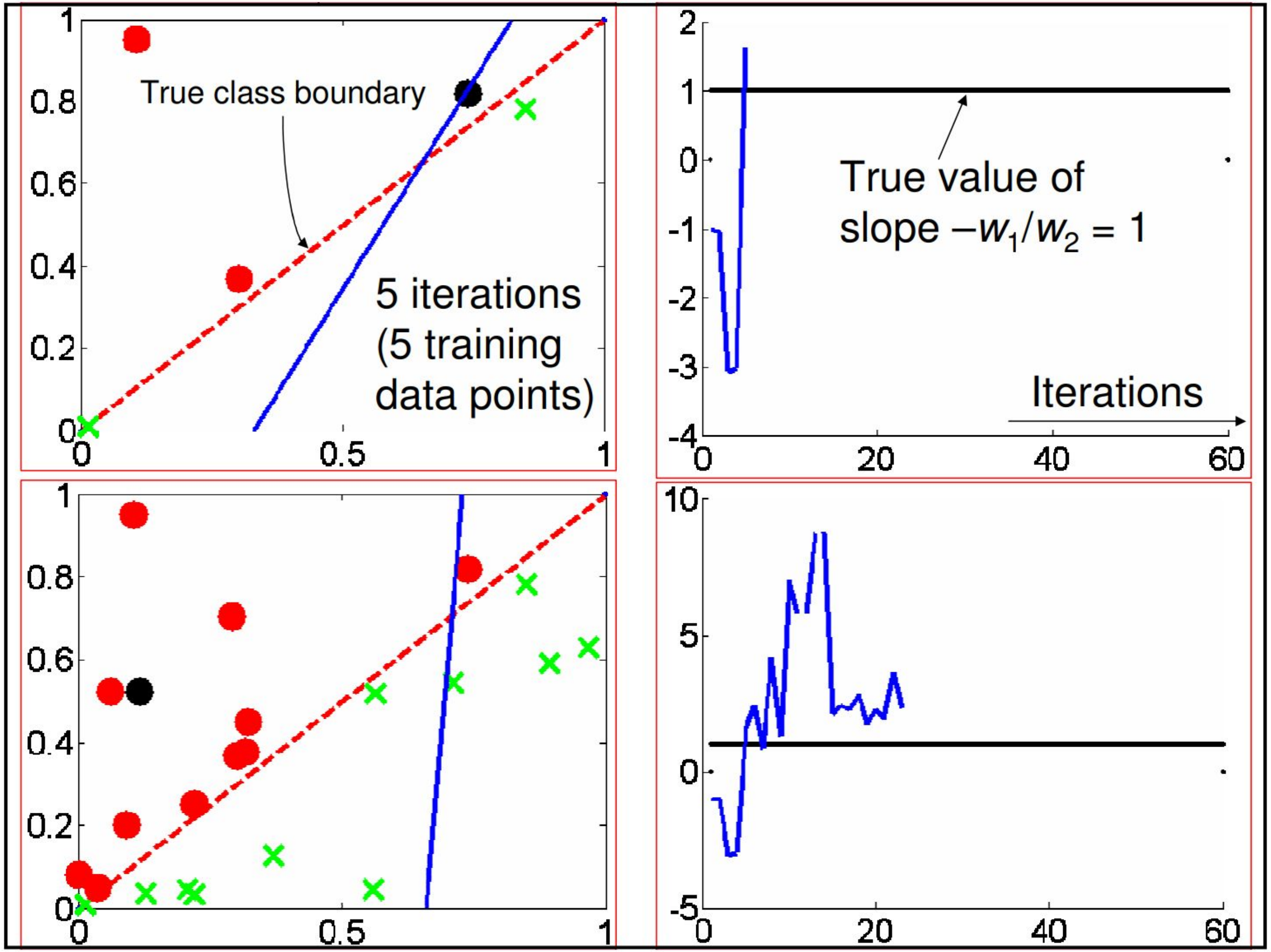
Example

$x_2 = x_1$

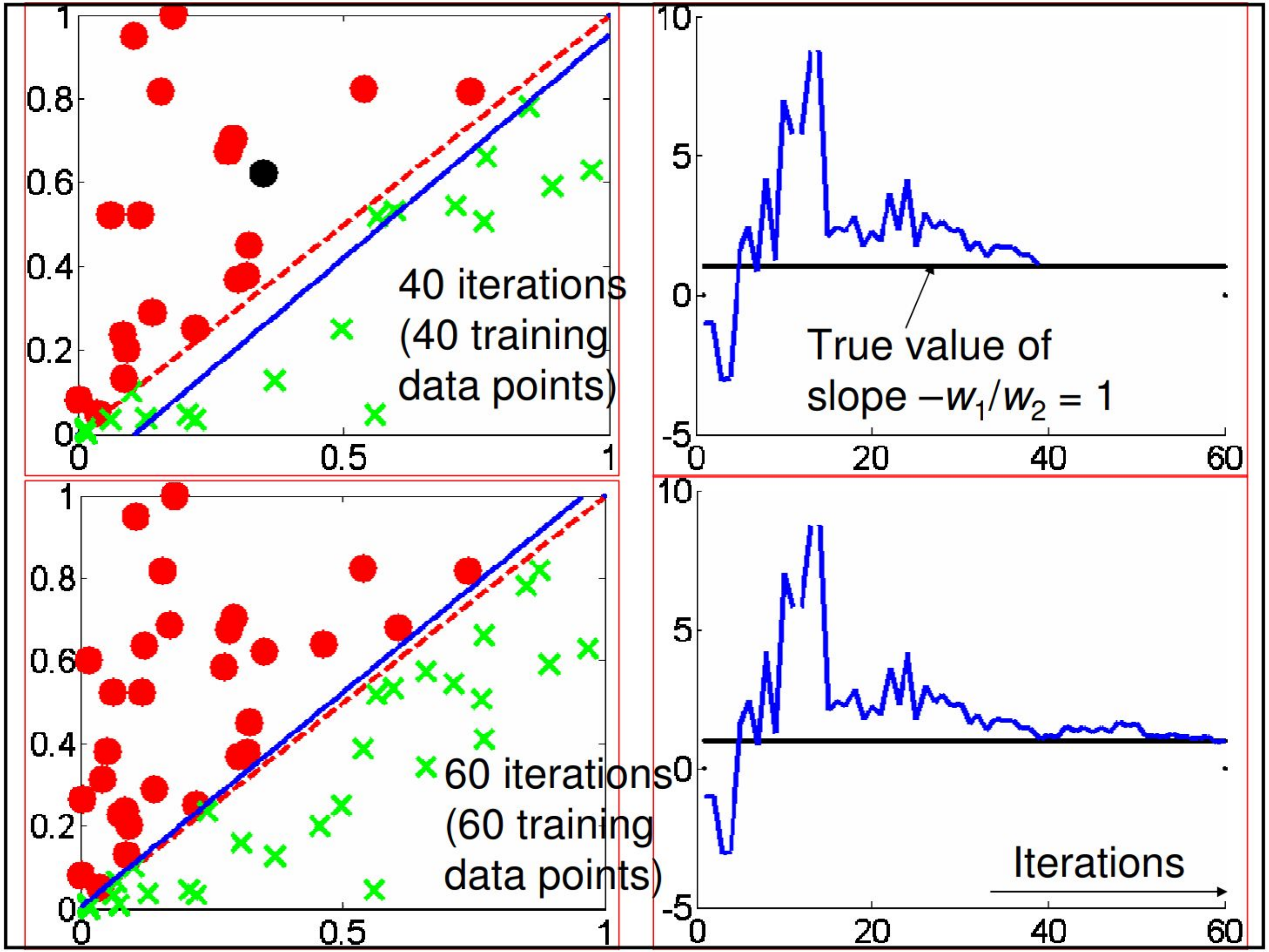
$\mathbf{w}$ = [0 1 -1]

Annoying detail: We get the same separating line if we multiplying all of **w** by some constant, so we are really interested in the relative values, such as the slope of the line, $-w_2/w_1$.





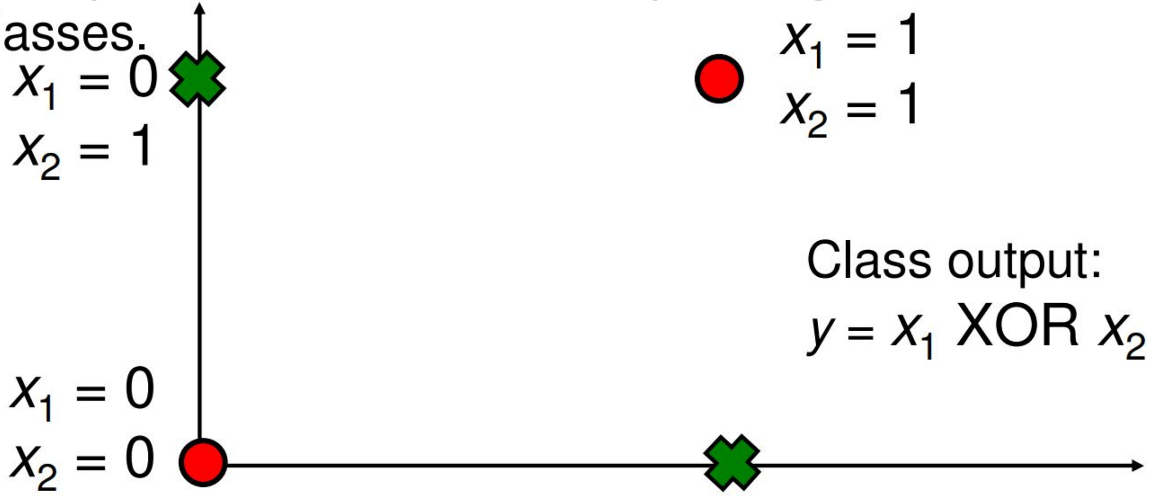
True class boundary
5 iterations
(5 training
data points)
True value of
slope $-w_1/w_2 = 1$
Iterations



40 iterations
(40 training
data points)
True value of
slope $-w_1/w_2 = 1$
60 iterations
(60 training
data points)
Iterations

Single Layer: Remarks

- Good news: Can represent any problem in which the decision boundary is *linear*.
- Bad news: *NO* guarantee if the problem is not linearly separable
- Canonical example: Learning the XOR function from example → There is no line separating the data in 2 classes.



Hyperplanes over R^d have VC-dim = d+1

The Minsky-Papert XOR affair (1969)