Feed-Forward Neural Networks

Hankui Zhuo

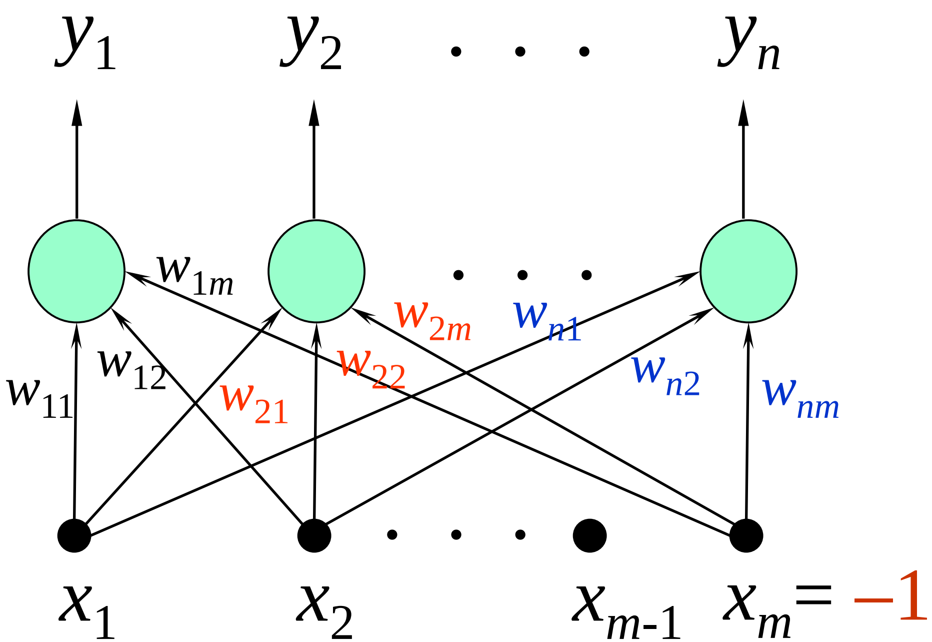
May 15, 2019

http://xplan-lab.org

Content

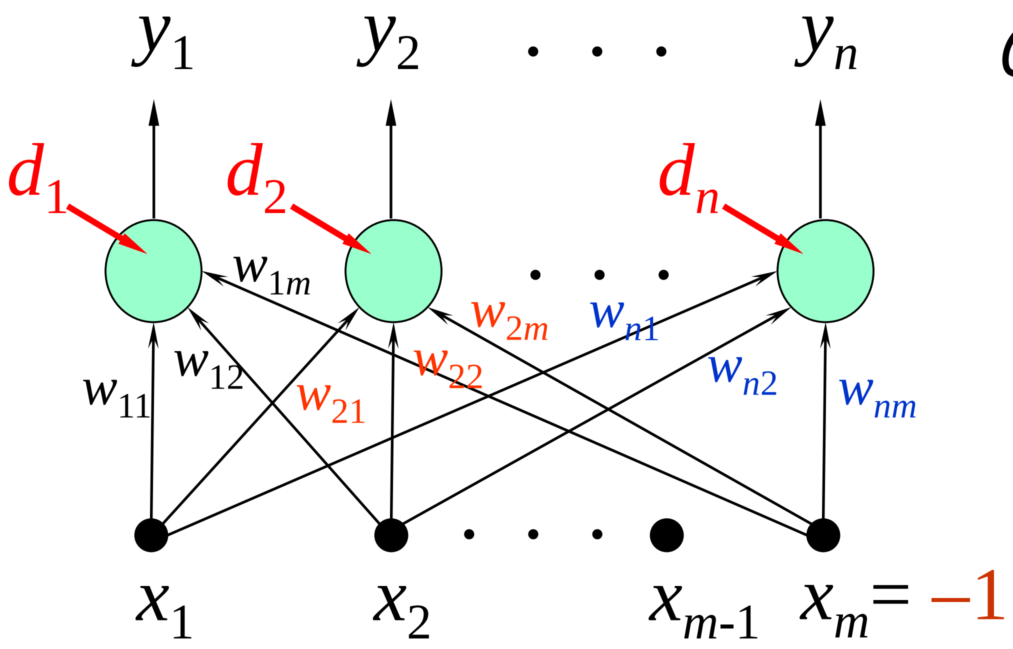
- Single-Layer Perceptron Networks
- Learning Rules for Single-Layer Perceptron Networks
    - Perceptron Learning Rule
    - Adaline Leaning Rule
    - δ-Leaning Rule
- Multilayer Perceptron
- Back Propagation Learning algorithm

The Single-Layered Perceptron



Training a Single-Layered Perceptron

Training Set $\mathbf{T} = \left\{ (\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), \ldots, (\mathbf{x}^{(p)}, \mathbf{d}^{(p)}) \right\}$



Goal:

$$y_i^{(k)} = a(\mathbf{w}_i^T \mathbf{x}^{(k)})$$

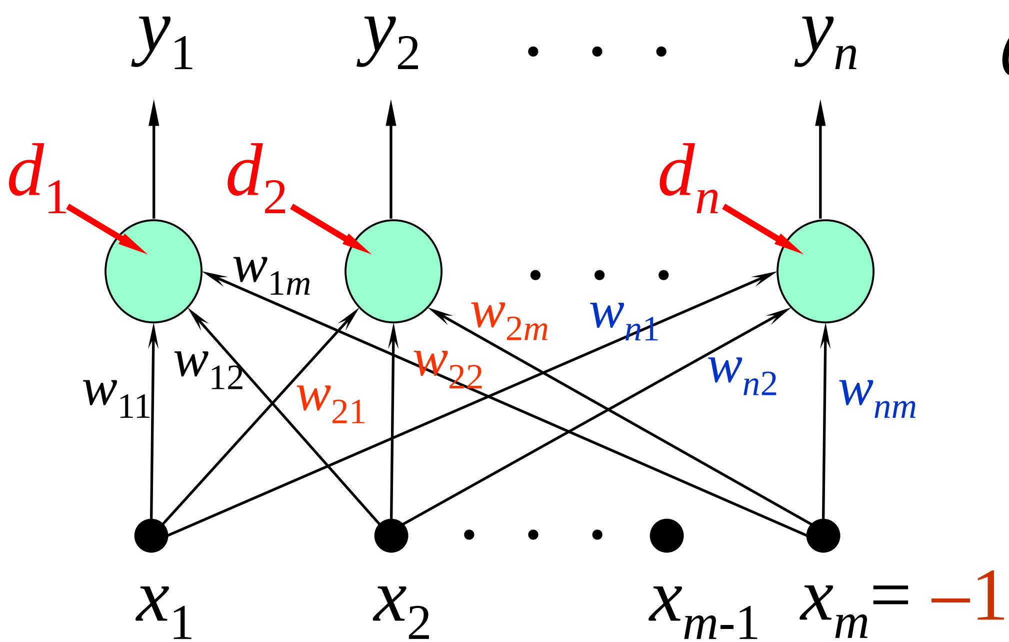
$$= a\left(\sum_{l=1}^{m} w_{il} x_l^{(k)} \right) = d_i^{(k)}$$

$$\forall \begin{array}{l} i = 1, 2, \ldots, n \\ k = 1, 2, \ldots, p \end{array}$$

Learning Rules

Training Set $\mathbf{T} = \left\{(\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), \ldots, (\mathbf{x}^{(p)}, \mathbf{d}^{(p)})\right\}$

- Linear Threshold Units (LTUs) : Perceptron Learning Rule
- Linearly Graded Units (LGUs) : Widrow-Hoff learning Rule



Goal:

$$y_i^{(k)} = a(\mathbf{w}_i^T \mathbf{x}^{(k)})$$

$$= a\left(\sum_{l=1}^{m} w_{il} x_l^{(k)}\right) = d_i^{(k)}$$

$$\forall \begin{array}{l} i = 1, 2, \ldots, n \\ k = 1, 2, \ldots, p \end{array}$$

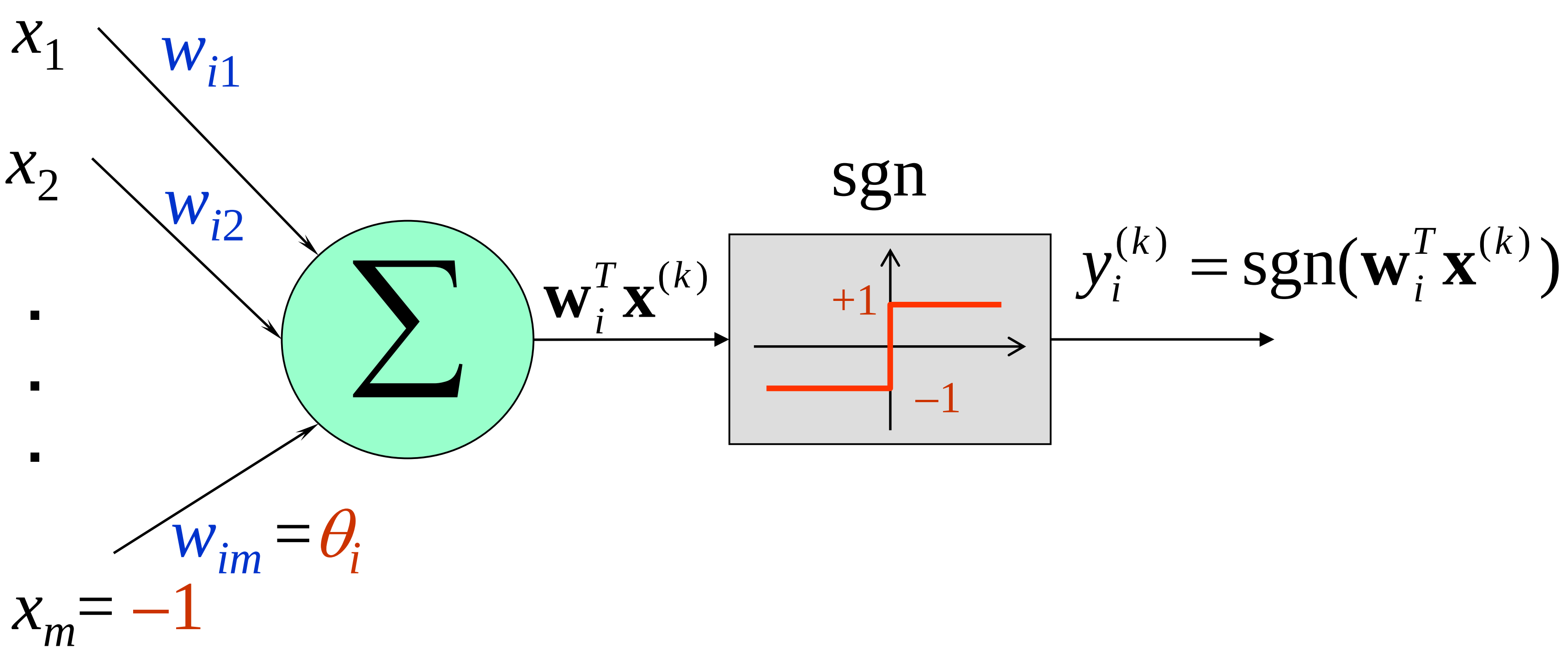
Feed-Forward Neural Networks

Learning Rules for
Single-Layered Perceptron Networks

- Perceptron Learning Rule
- Adline Leaning Rule
- δ-Learning Rule

Perceptron

Linear Threshold Unit



Perceptron

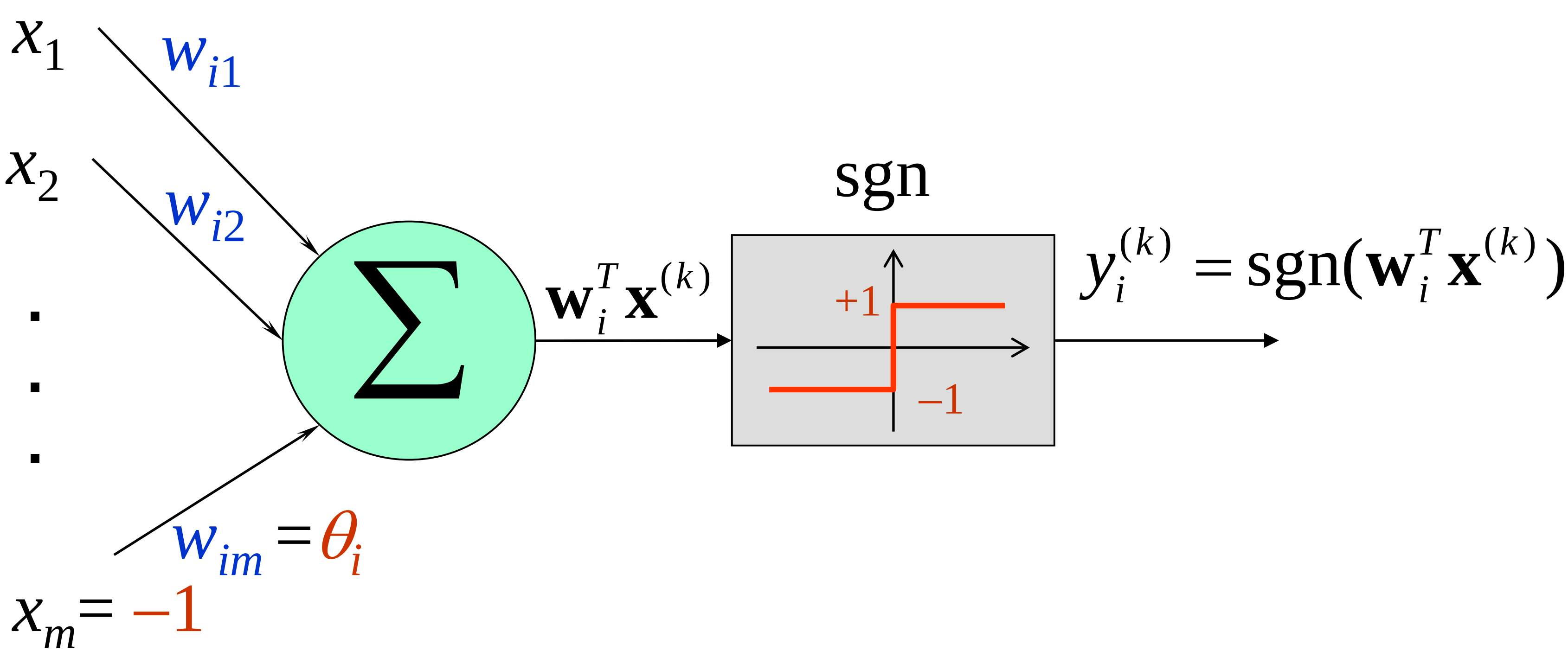
Goal:

$$y_i^{(k)} = \mathrm{sgn}(\mathbf{w}_i^T \mathbf{x}^{(k)}) = d_i^{(k)} \in \{1, -1\}$$

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

$$k = 1, 2, \ldots, p$$

Linear Threshold Unit



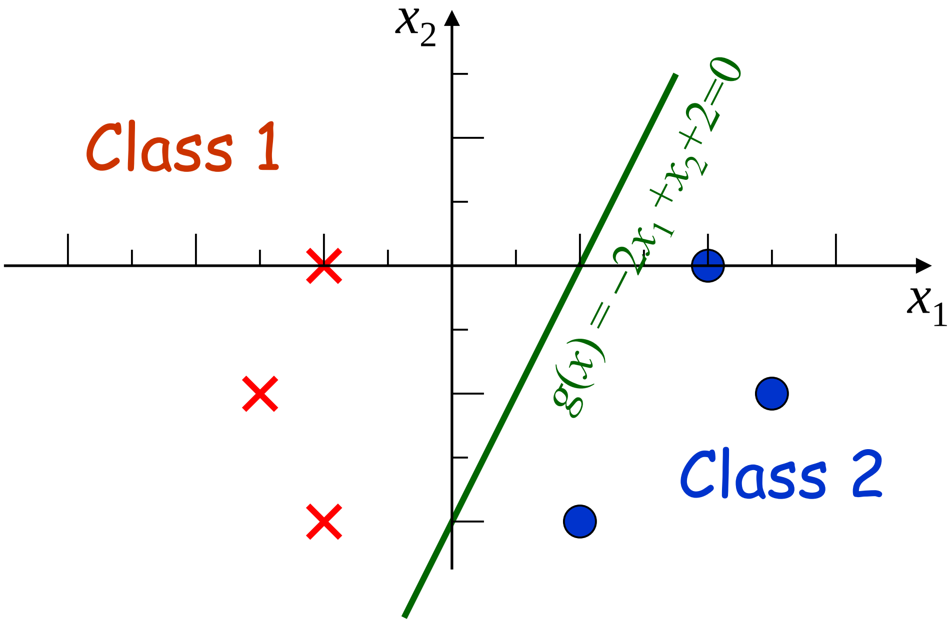
Example

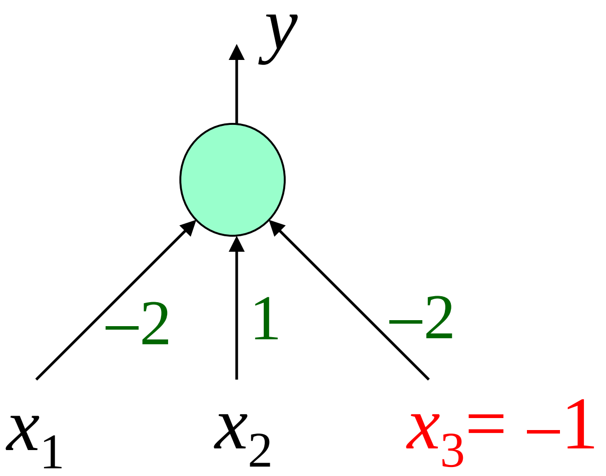
Goal:

$$y_i^{(k)} = \mathrm{sgn}(\mathbf{w}_i^T \mathbf{x}^{(k)}) = d_i^{(k)} \in \{1, -1\}$$
$$i = 1, 2, \ldots, n$$
$$k = 1, 2, \ldots, p$$

Class 1 (+1) — $\{[-1,0]^T, [-1.5,-1]^T, [-1,-2]^T\}$

Class 2 (–1) — $\{[2,0]^T, [2.5,-1]^T, [1,-2]^T\}$





Goal: $y^{(k)} = \mathrm{sgn}(\mathbf{w}^T \mathbf{x}^{(k)}) = d^{(k)}$

$\mathbf{w} = (w_1, w_2, w_3)^T$

Augmented input vector

Class 1 (+1) — $\{[-1,0]^T, [-1.5,-1]^T, [-1,-2]^T\}$

Class 2 (−1) — $\{[2,0]^T, [2.5,-1]^T, [1,-2]^T\}$

Class 1 (+1)

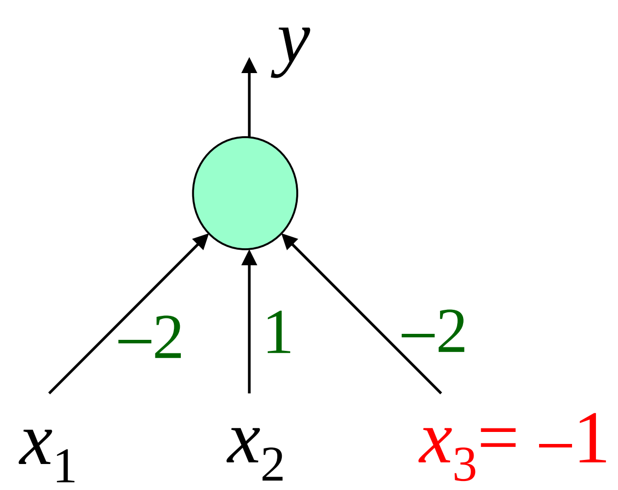
$$x^{(1)} = \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix}, \quad x^{(2)} = \begin{bmatrix} -1.5 \\ -1 \\ -1 \end{bmatrix}, \quad x^{(3)} = \begin{bmatrix} -1 \\ -2 \\ -1 \end{bmatrix}$$

$$d^{(1)} = 1, \quad d^{(2)} = 1, \quad d^{(3)} = 1$$

Class 2 (−1)

$$x^{(4)} = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}, \quad x^{(5)} = \begin{bmatrix} 2.5 \\ -1 \\ -1 \end{bmatrix}, \quad x^{(6)} = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}$$

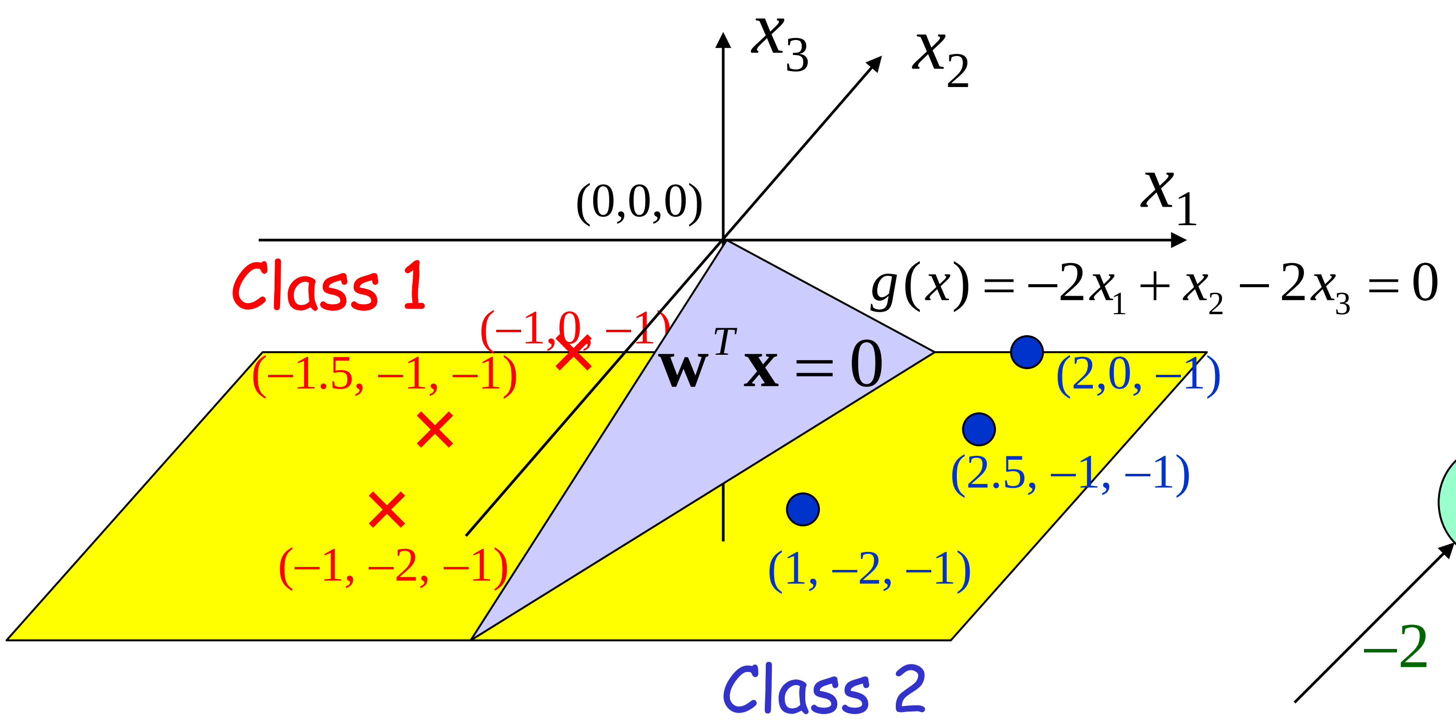
$$d^{(4)} = -1, \quad d^{(5)} = -1, \quad d^{(6)} = -1$$

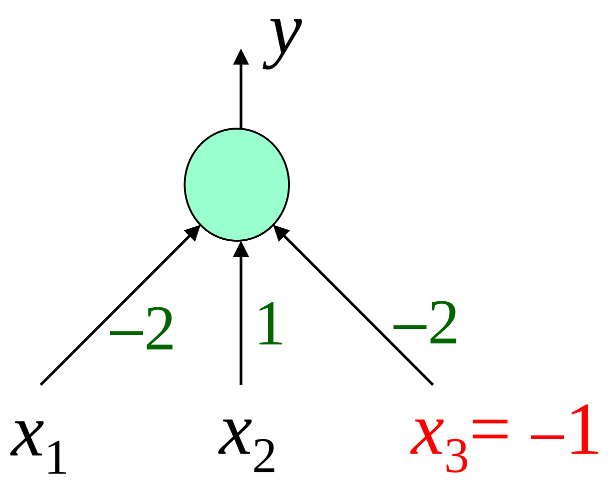


Goal: $y^{(k)} = \mathrm{sgn}(\mathbf{w}^T \mathbf{x}^{(k)}) = d^{(k)}$

$\mathbf{w} = (w_1, w_2, w_3)^T$

Augmented input vector

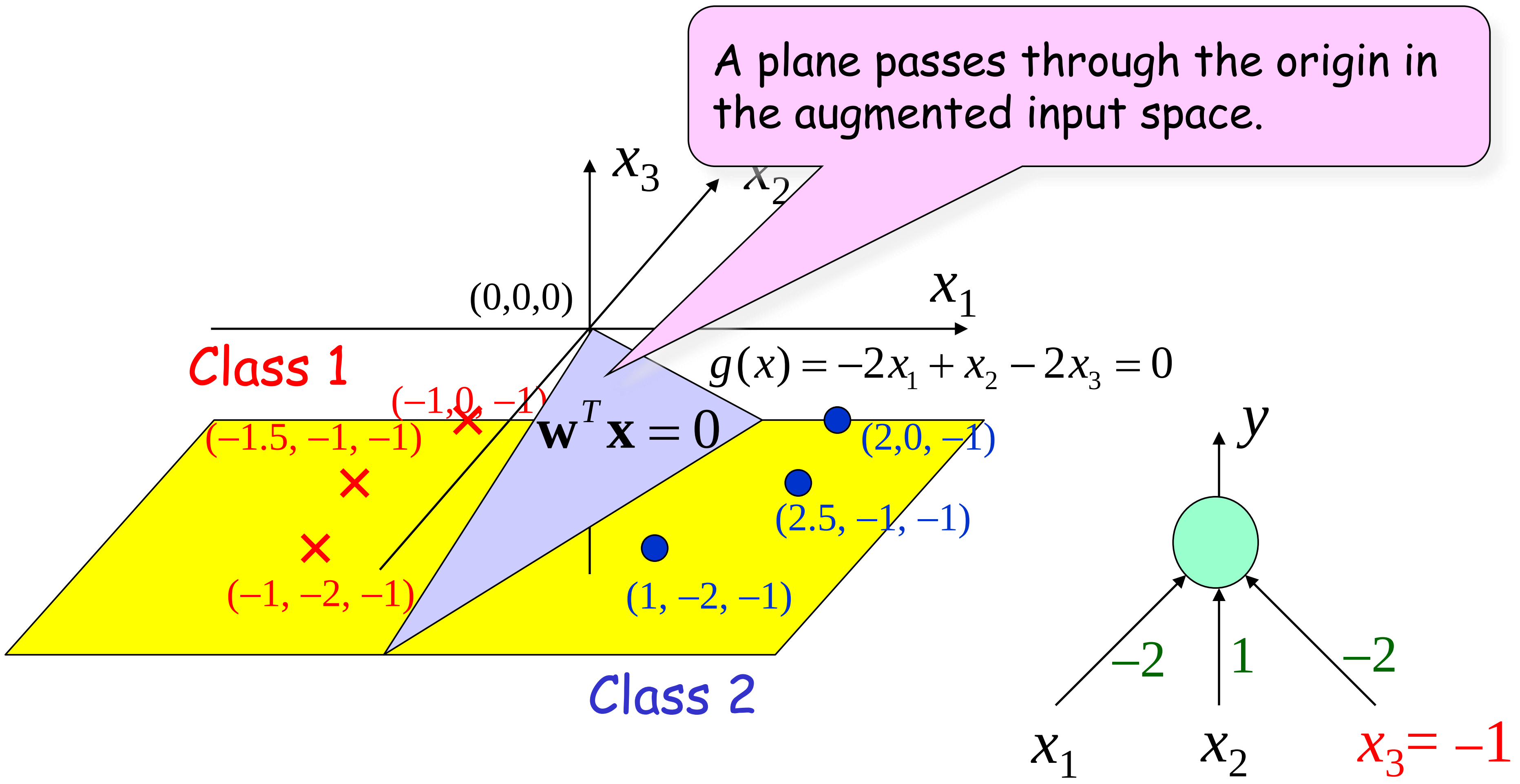




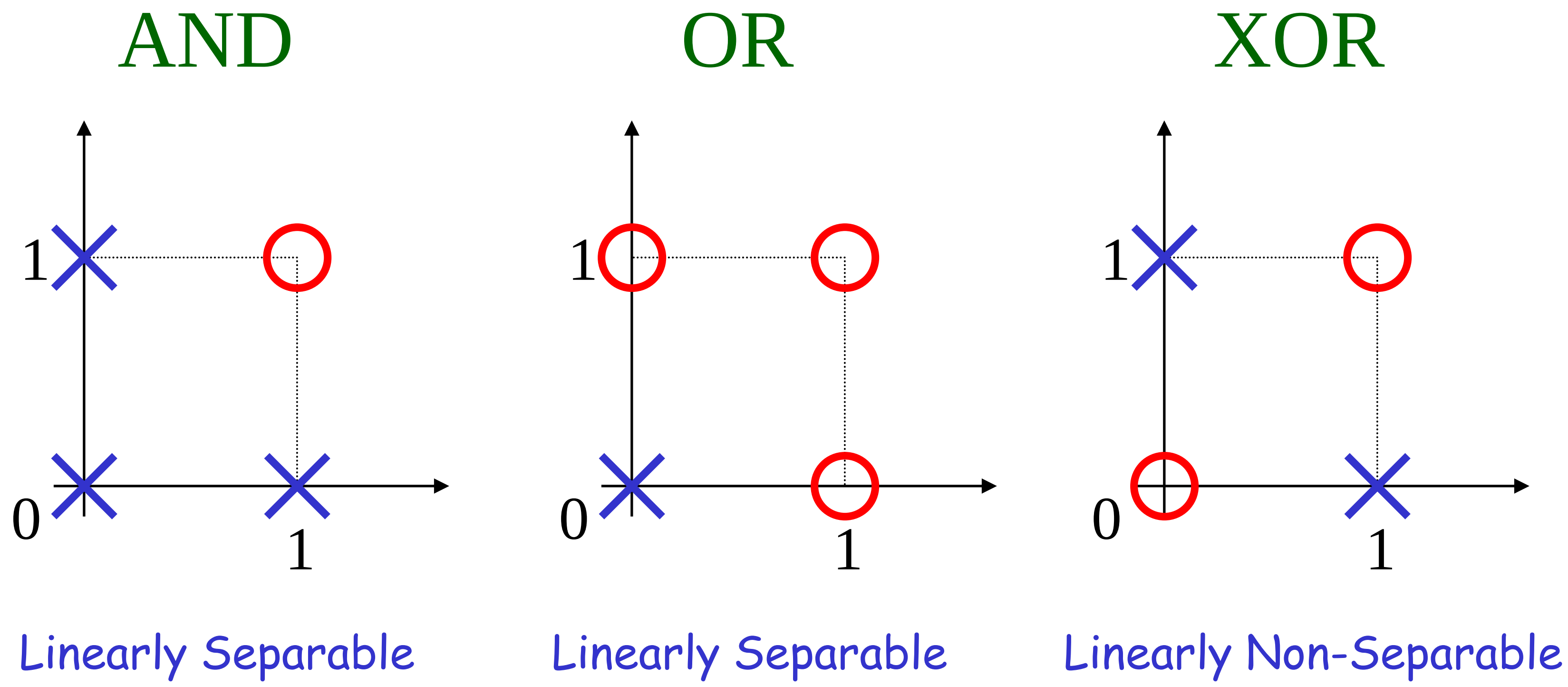
Augmented input vector

Goal: $y^{(k)} = \mathrm{sgn}(\mathbf{w}^T \mathbf{x}^{(k)}) = d^{(k)}$

$\mathbf{w} = (w_1, w_2, w_3)^T$



Linearly Separable vs. Linearly Non-Separable



Goal

- Given training sets $T_1 \in C_1$ and $T_2 \in C_2$ with elements in form of $\mathbf{x}=(x_1, x_2, \ldots, x_{m-1}, x_m)^T$, where $x_1, x_2, \ldots, x_{m-1} \in R$ and $x_m = -1$.
- Assume T_1 and T_2 are linearly separable.
- Find $\mathbf{w}=(w_1, w_2, \ldots, w_m)^T$ such that

$$\operatorname{sgn}(\mathbf{w}^T\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in T_1 \\ -1 & \mathbf{x} \in T_2 \end{cases}$$

Goal

$\mathbf{w}^T\mathbf{x} = 0$ is a hyperplain passes through the origin of *augmented input space.*

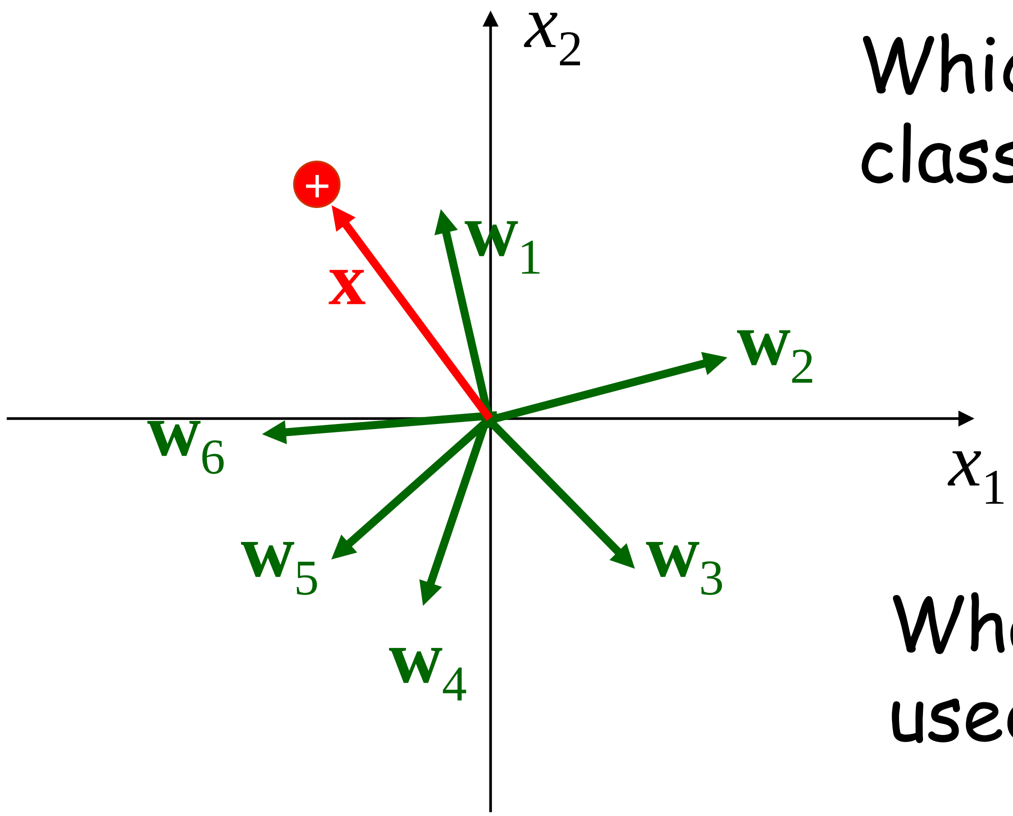
- Given training sets $T_1 \in C_1$ and $T_2 \in C_2$ with elements in form of $\mathbf{x}=(x_1, x_2, \ldots, x_{m-1}, x_m)^T$, where $x_1, x_2, \ldots, x_{m-1} \in R$ and $x_m = -1$.
- Assume T_1 and T_2 are linearly separable.
- Find $\mathbf{w}=(w_1, w_2, \ldots, w_m)^T$ such that

$$\operatorname{sgn}(\mathbf{w}^T\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in T_1 \\ -1 & \mathbf{x} \in T_2 \end{cases}$$

Observation

$d = +1$

$d = -1$



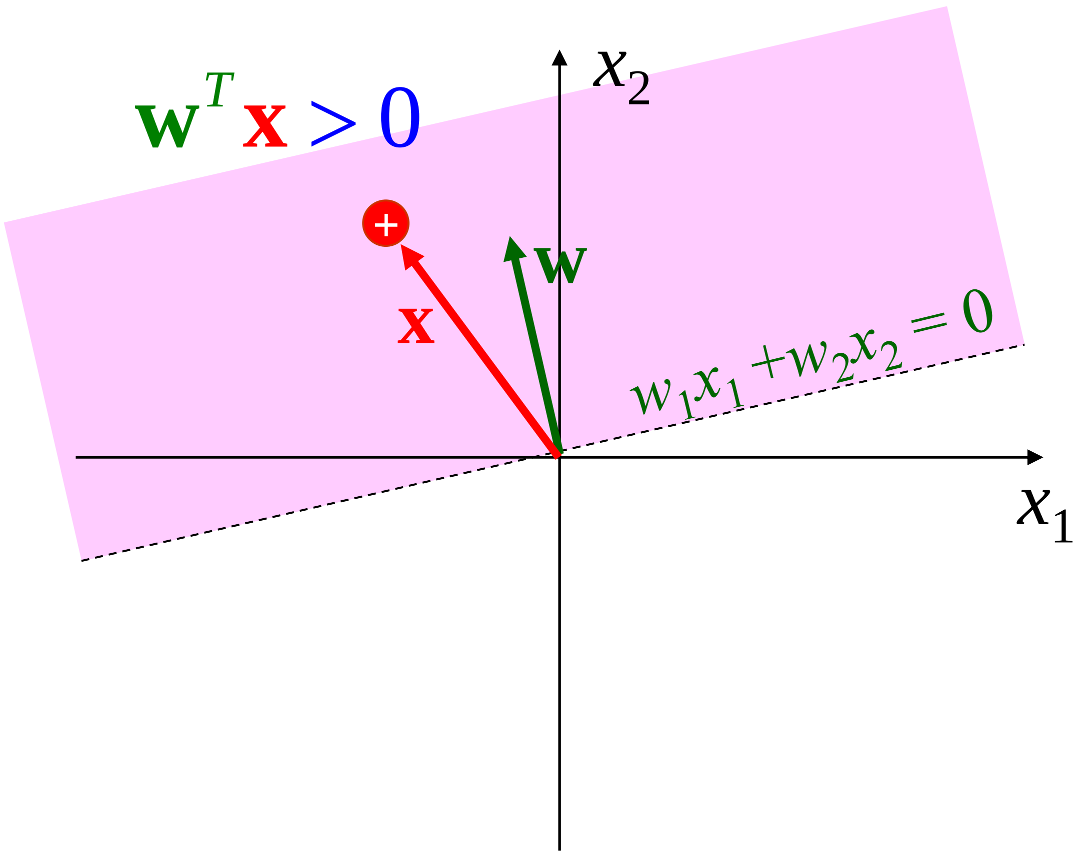
Which $\mathbf{w}$'s correctly classify $\mathbf{x}$?

What trick can be used?

Observation

$d = +1$

$d = -1$

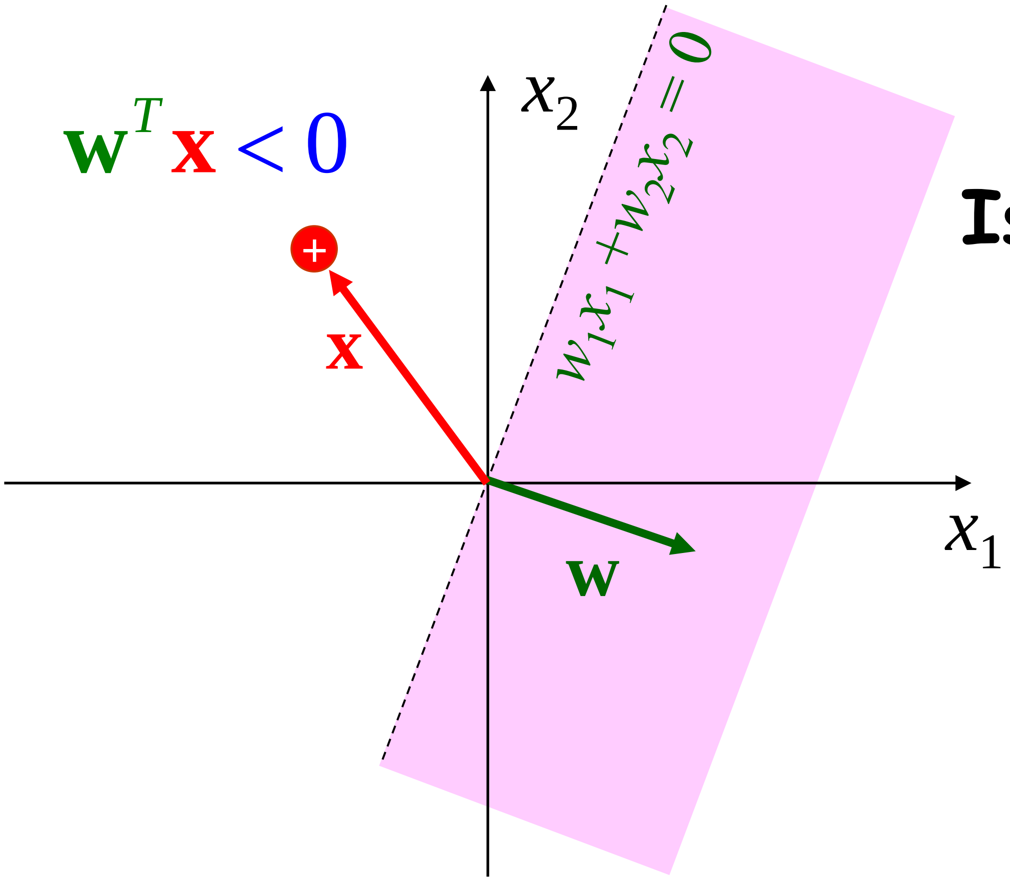


Is this w ok?

Observation

$d = +1$

$d = -1$

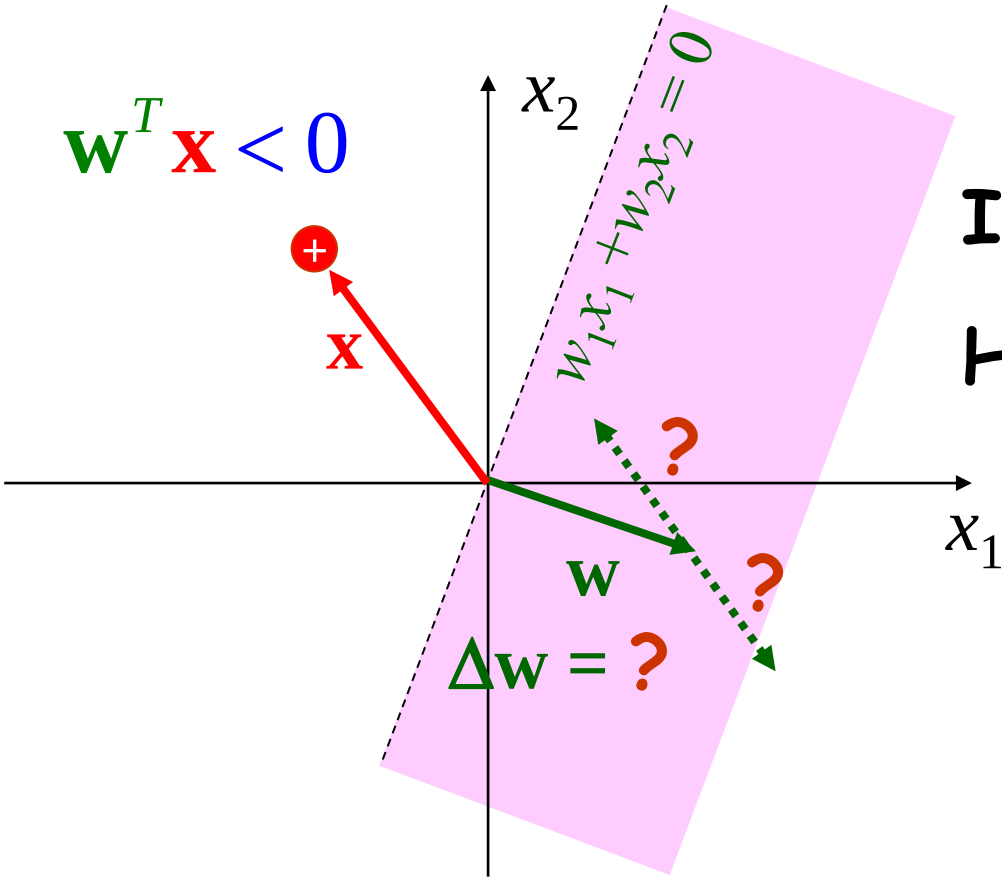


Is this $\mathbf{w}$ ok?

Observation

$d = +1$

$d = -1$



Is this $\mathbf{w}$ ok?

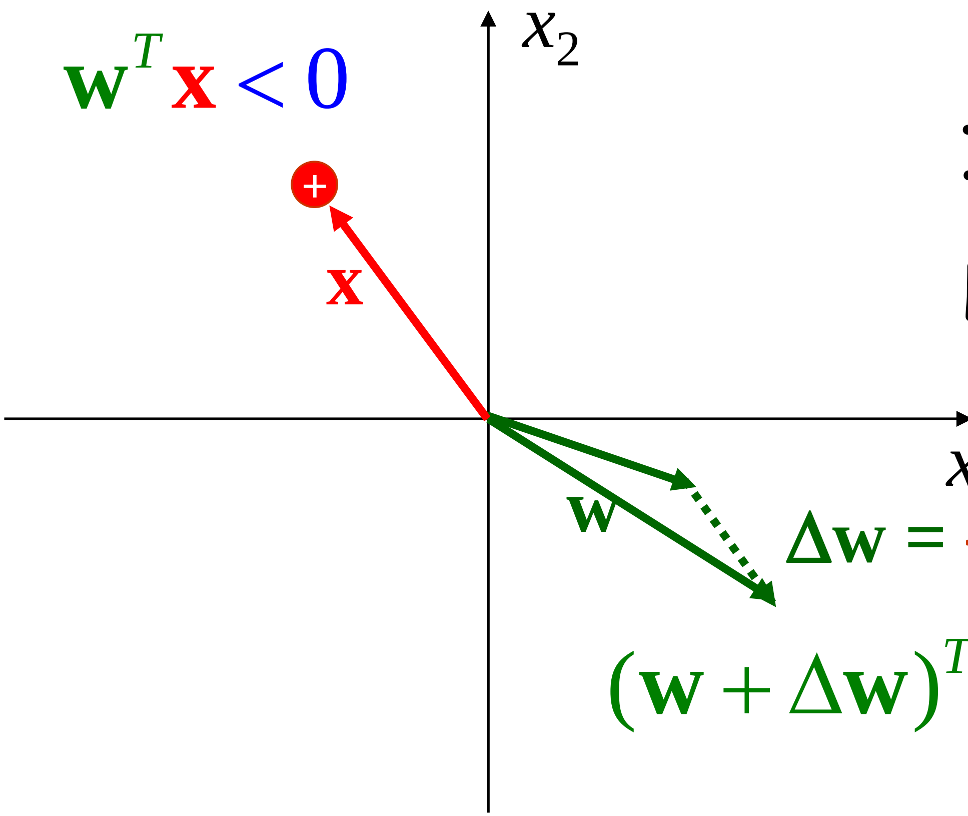
How to adjust $\mathbf{w}$?

Observation

$d = +1$ (+)

$d = -1$ (−)

$\mathbf{w}^T \mathbf{x} < 0$



Is this $\mathbf{w}$ ok?

How to adjust $\mathbf{w}$?

$\Delta \mathbf{w} = -\alpha \mathbf{x}$

reasonable?

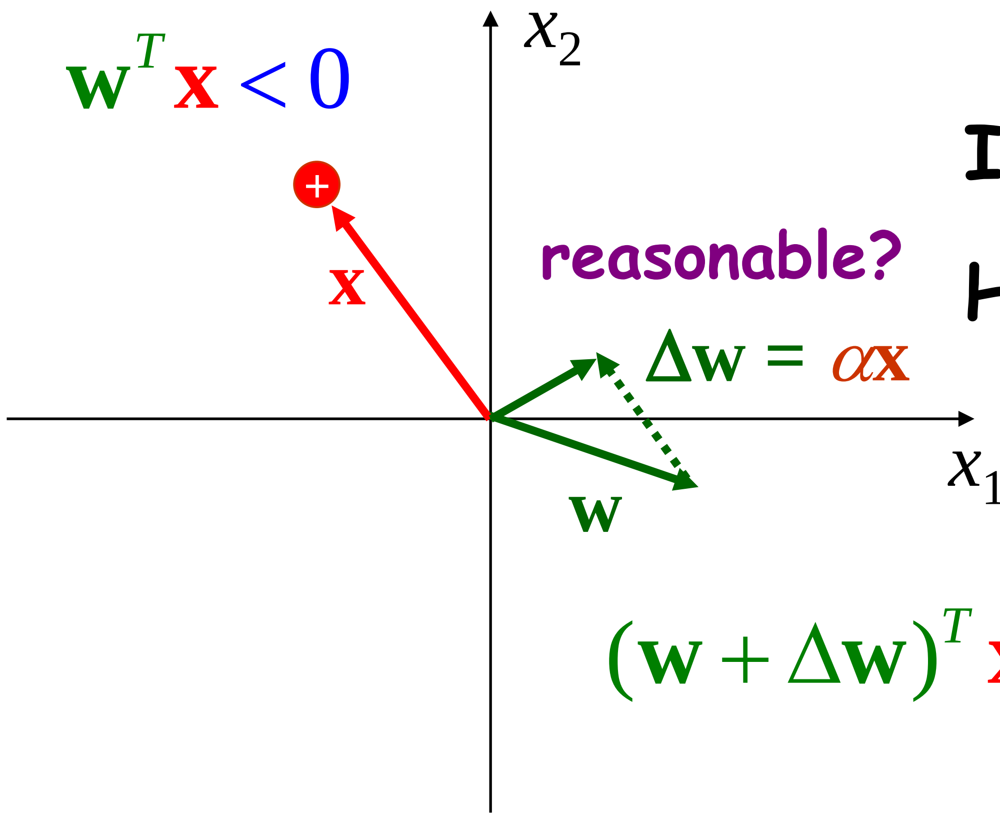
$$(\mathbf{w} + \Delta \mathbf{w})^T \mathbf{x} = \underbrace{\mathbf{w}^T \mathbf{x}}_{<0} - \underbrace{\alpha \mathbf{x}^T \mathbf{x}}_{>0}$$

Observation

$+\ \ d = +1$

$-\ \ d = -1$

$\mathbf{w}^T\mathbf{x} < 0$



Is this w ok?

How to adjust w?

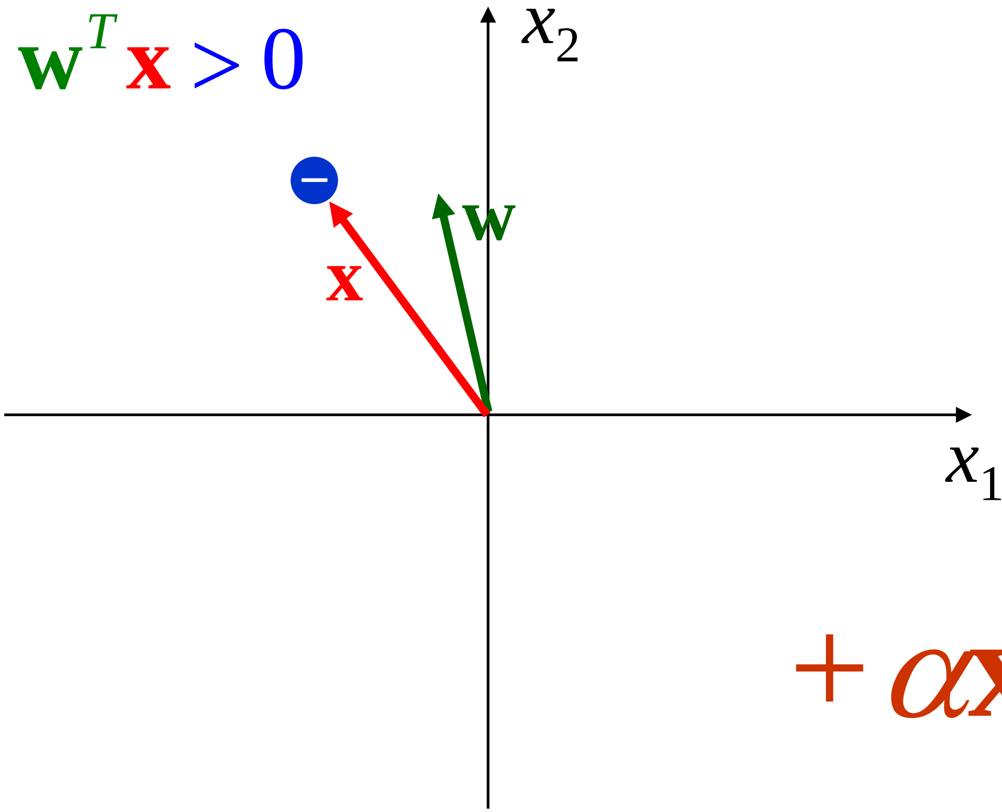
$$(\mathbf{w}+\Delta\mathbf{w})^T\mathbf{x} = \underbrace{\mathbf{w}^T\mathbf{x}}_{<0} + \underbrace{\alpha\mathbf{x}^T\mathbf{x}}_{>0}$$

Observation

$d = +1$

$d = -1$

$\mathbf{w}^T \mathbf{x} > 0$



Is this $\mathbf{w}$ ok?

$\Delta \mathbf{w} = ?$

$+\alpha\mathbf{x}$ or $-\alpha\mathbf{x}$

Perceptron Learning Rule

$$\alpha > 0$$

Upon misclassification on

$\oplus$ $\quad d = +1 \qquad \Delta\mathbf{w} = \alpha\mathbf{x}$

$\ominus$ $\quad d = -1 \qquad \Delta\mathbf{w} = -\alpha\mathbf{x}$

Define error

$$r = d - y = \begin{cases} +2 & \oplus \to \ominus \\ -2 & \ominus \to \oplus \\ 0 & \text{No error} \end{cases}$$

Perceptron Learning Rule

$$\Delta \mathbf{w} = \eta r \mathbf{x}$$

Define error

$$r = d - y = \begin{cases} +2 & \oplus \rightarrow \ominus \\ -2 & \ominus \rightarrow \oplus \\ 0 & \text{No error} \end{cases}$$

Perceptron Learning Rule

$$\Delta \mathbf{w} = \eta r \mathbf{x}$$

Learning Rate

Error $(d - y)$

Input

Based on the general weight learning rule.

Summary – Perceptron Learning Rule

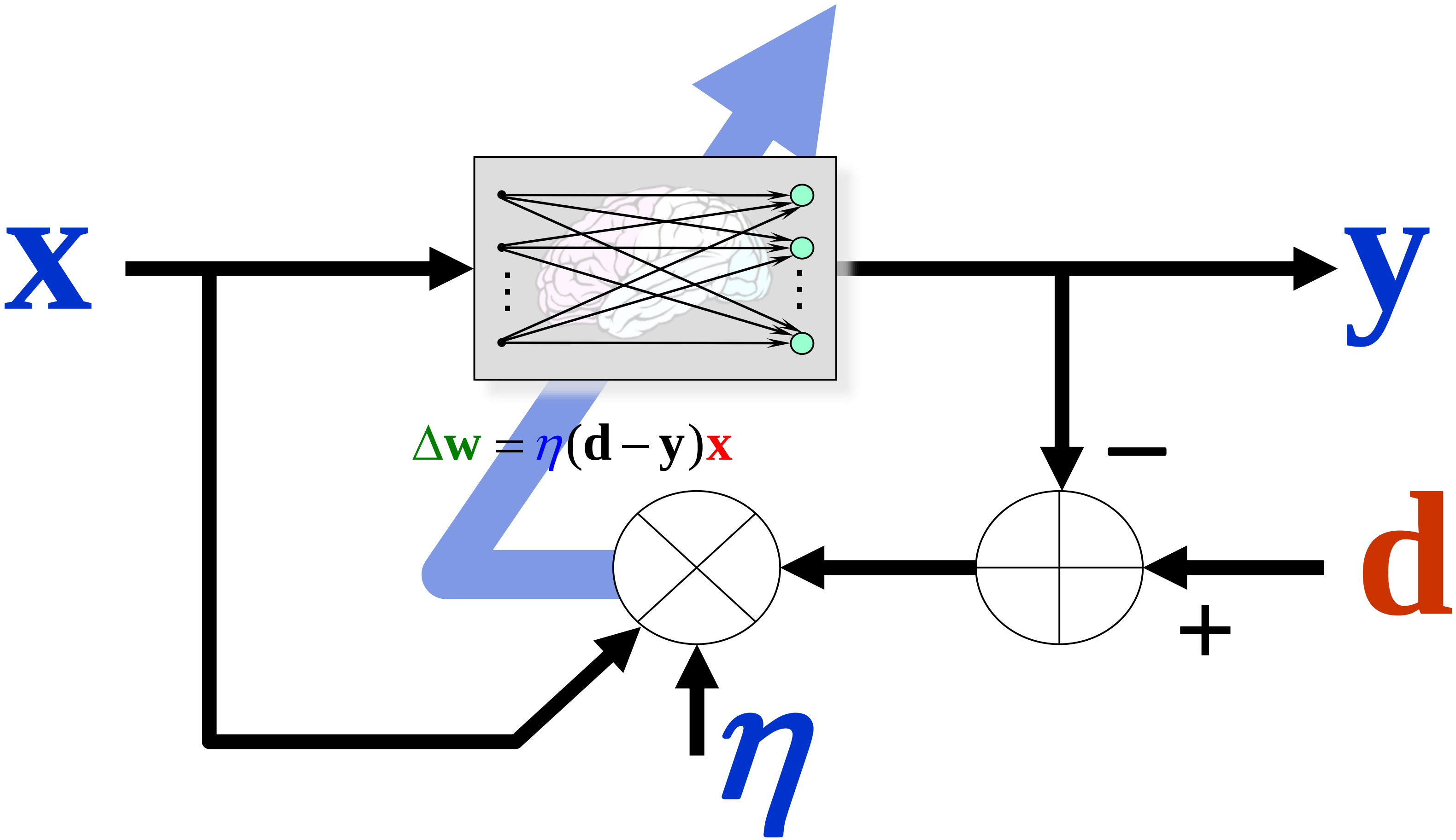
$$\Delta w_i(t) = \eta r_i x_i(t)$$

$$r_i = d_i - y_i = \begin{cases} 0 & d_i = y_i & \text{correct} \\ +2 & d_i = 1, y_i = -1 \\ -2 & d_i = -1, y_i = 1 \end{cases} \left.\begin{matrix} \\ \\ \end{matrix}\right\} \text{incorrect}$$

$$\Delta w_i(t) = \eta (d_i - y_i) x_i(t)$$

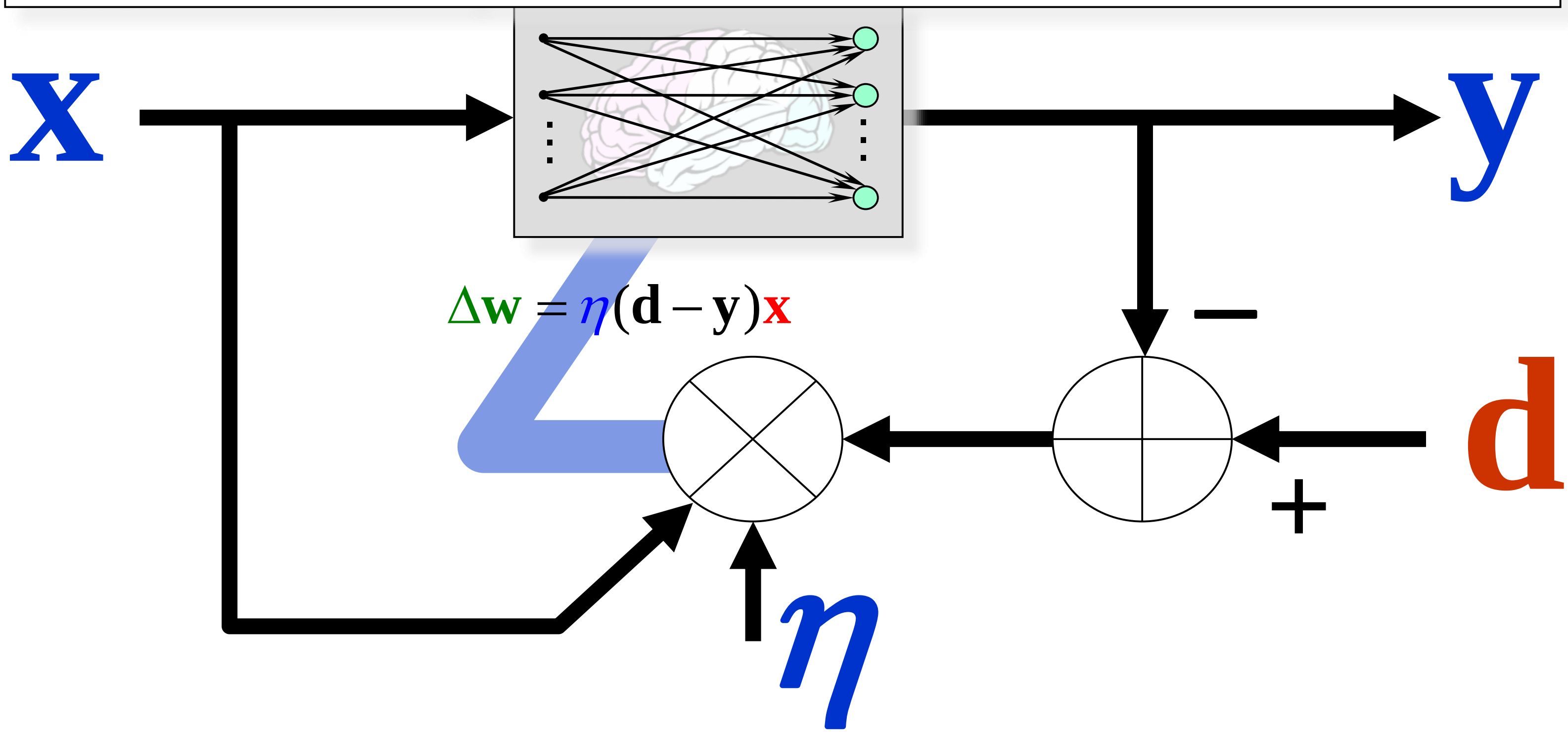
Summary – Perceptron Learning Rule

Converge?



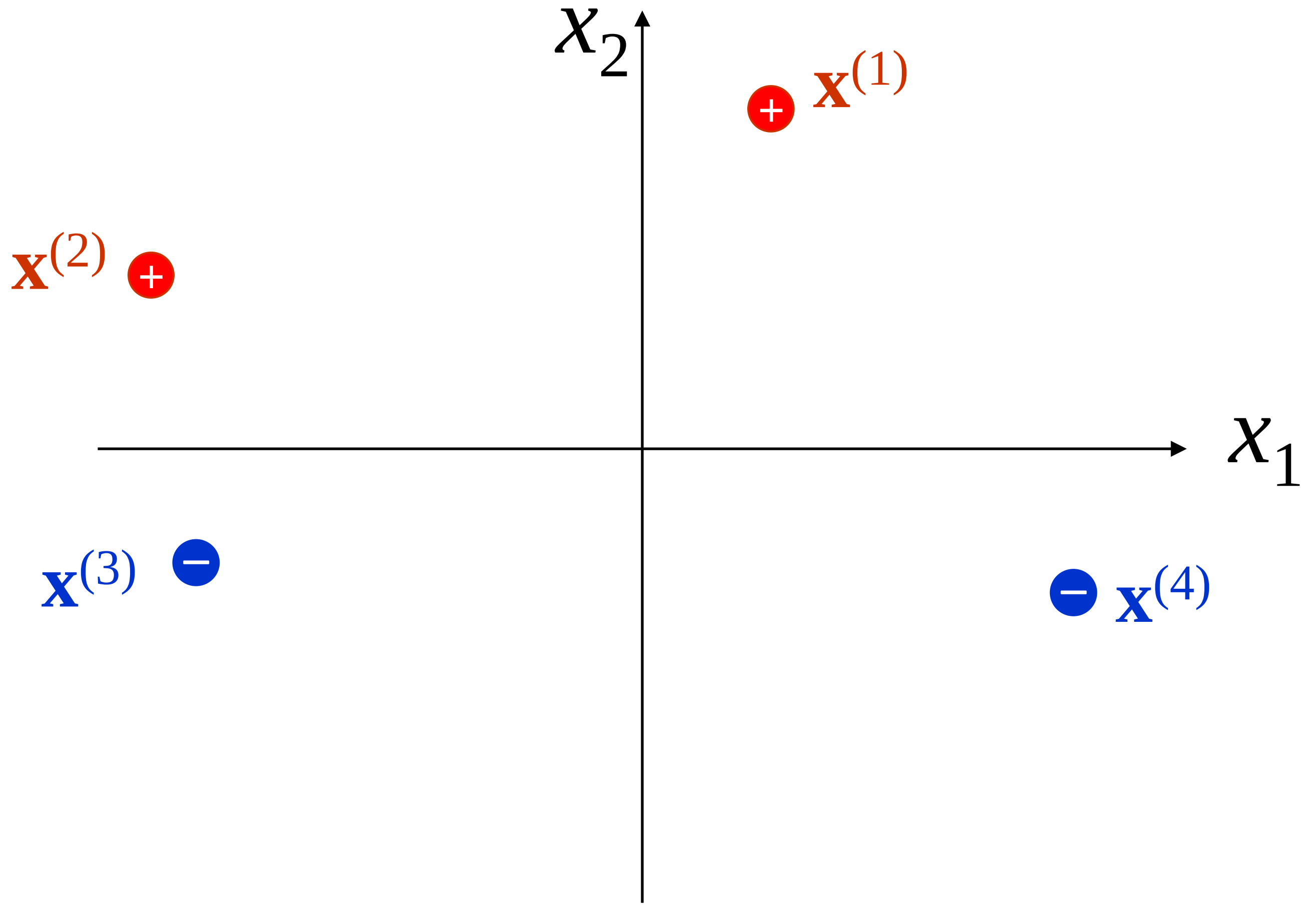
Perceptron Convergence Theorem

If the given training set is linearly separable, the learning process will converge in a finite number of steps.

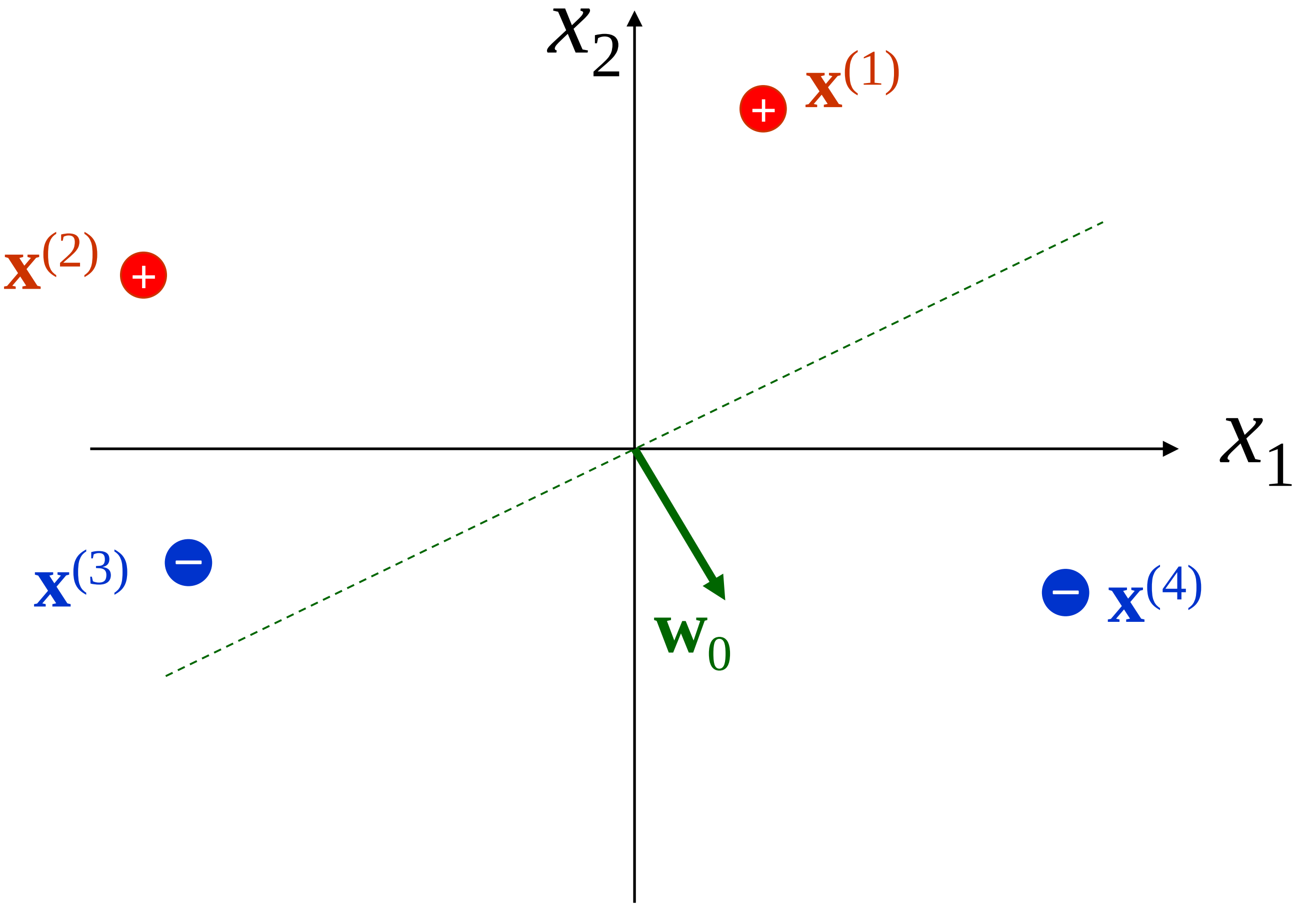


Linearly Separable.

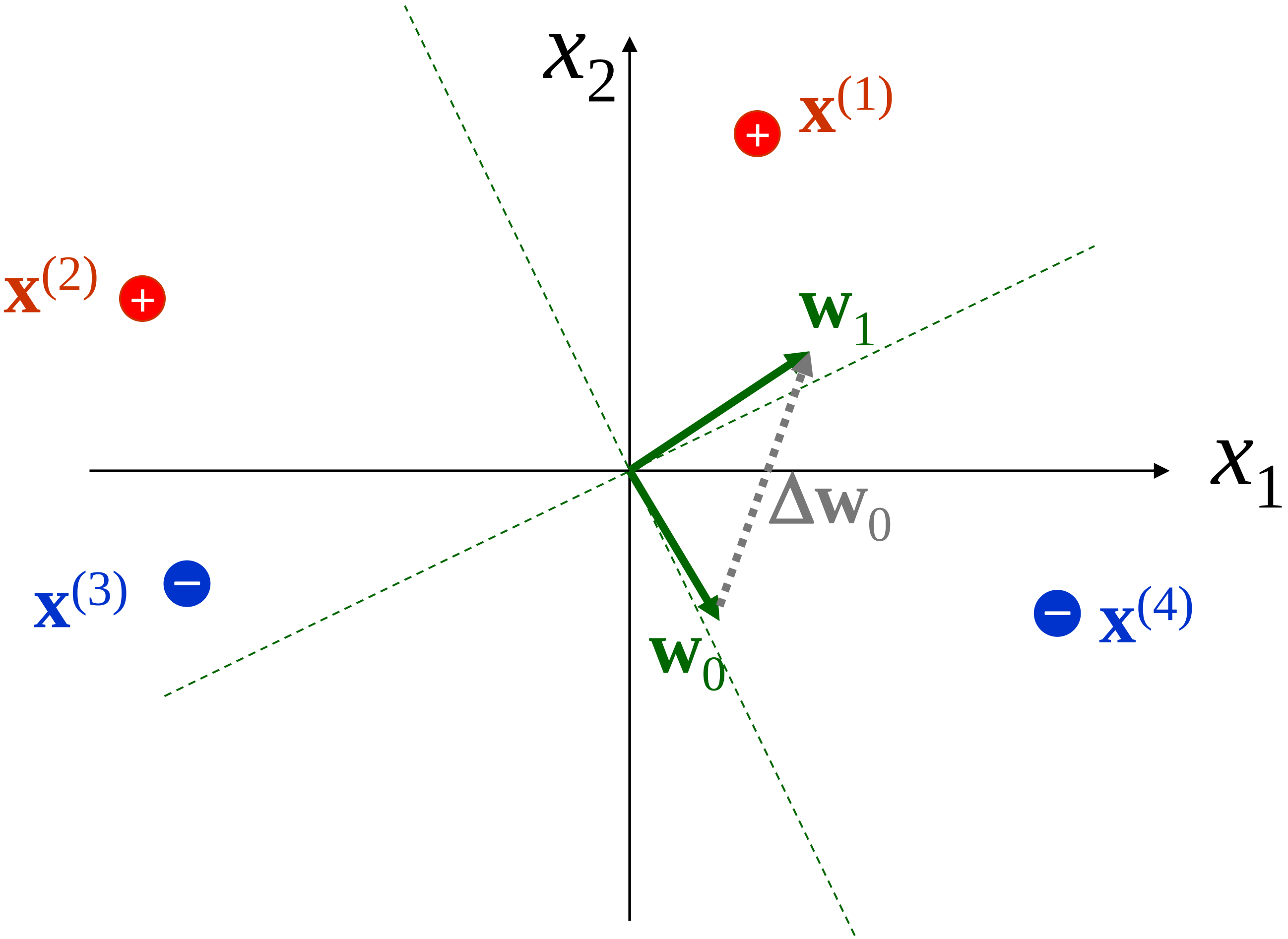
The Learning Scenario



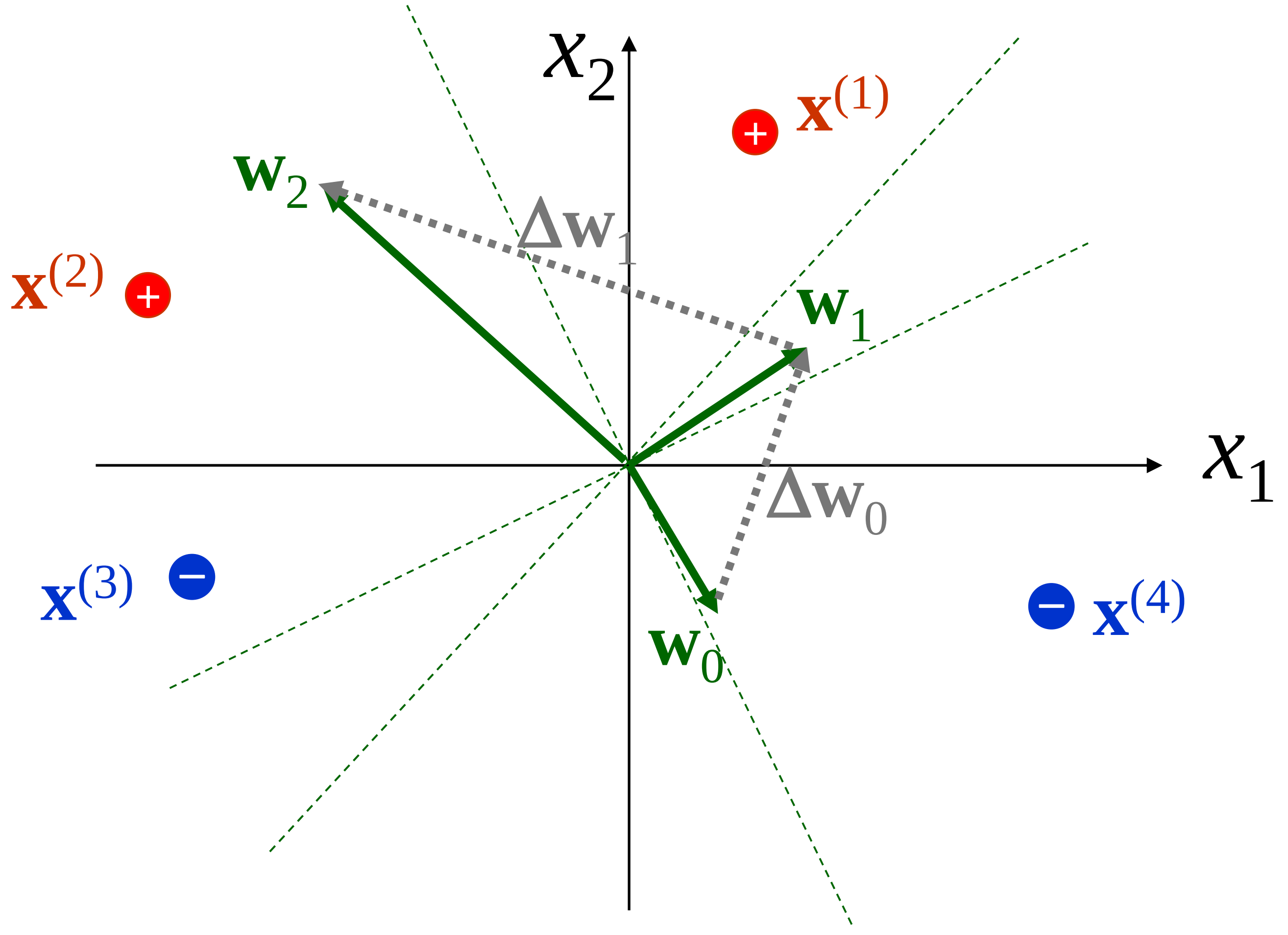
The Learning Scenario



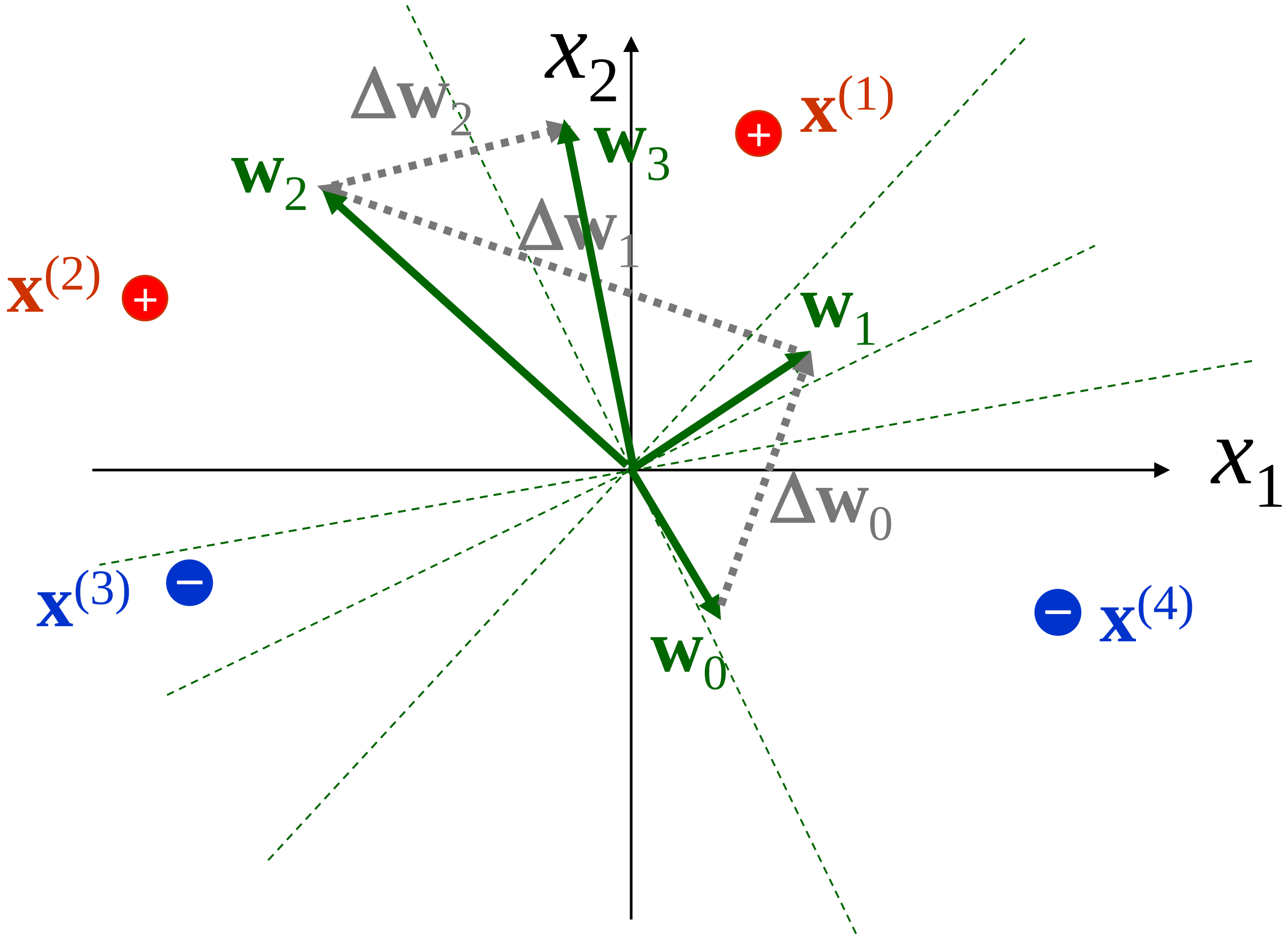
The Learning Scenario



The Learning Scenario

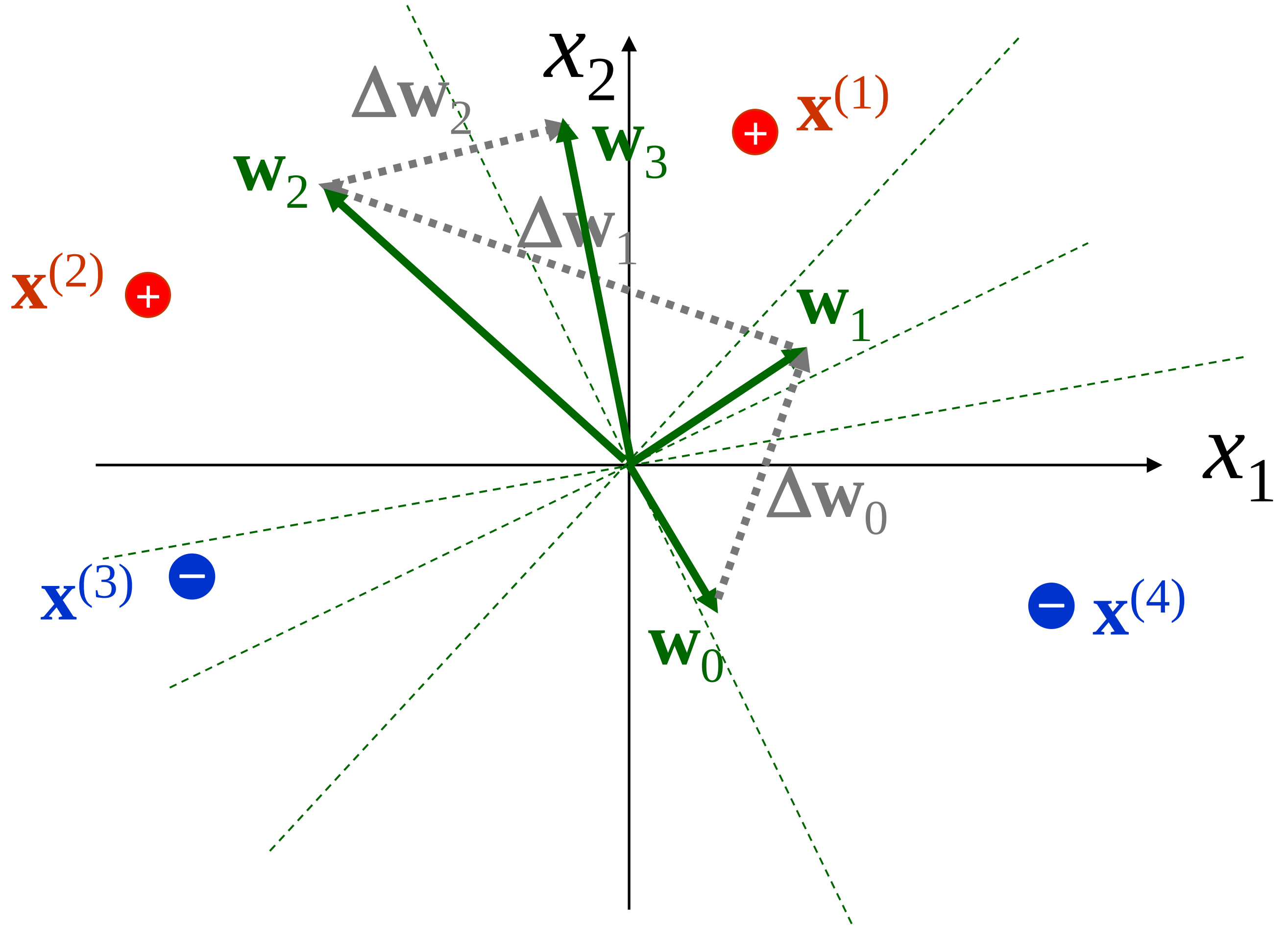


The Learning Scenario

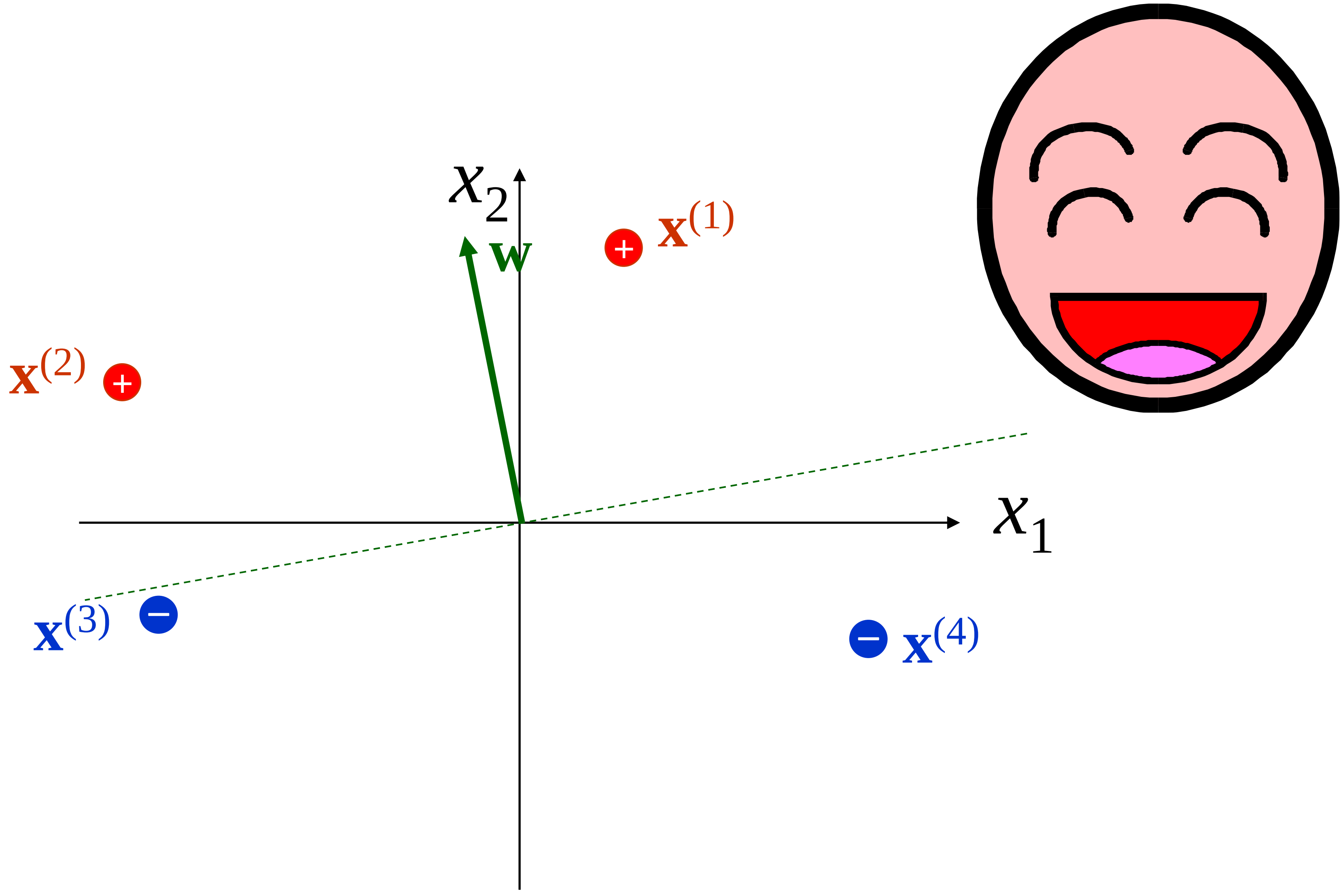


The Learning Scenario

$$\mathbf{w}_4 = \mathbf{w}_3$$

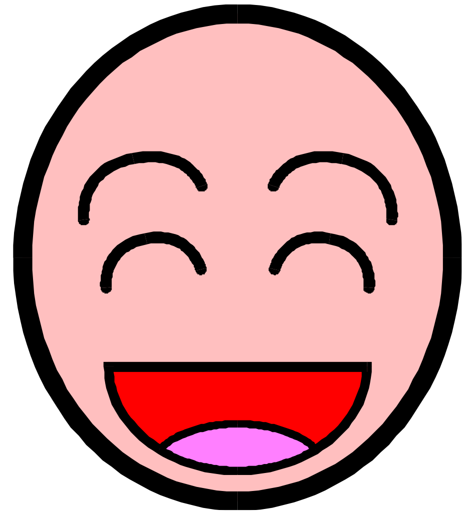


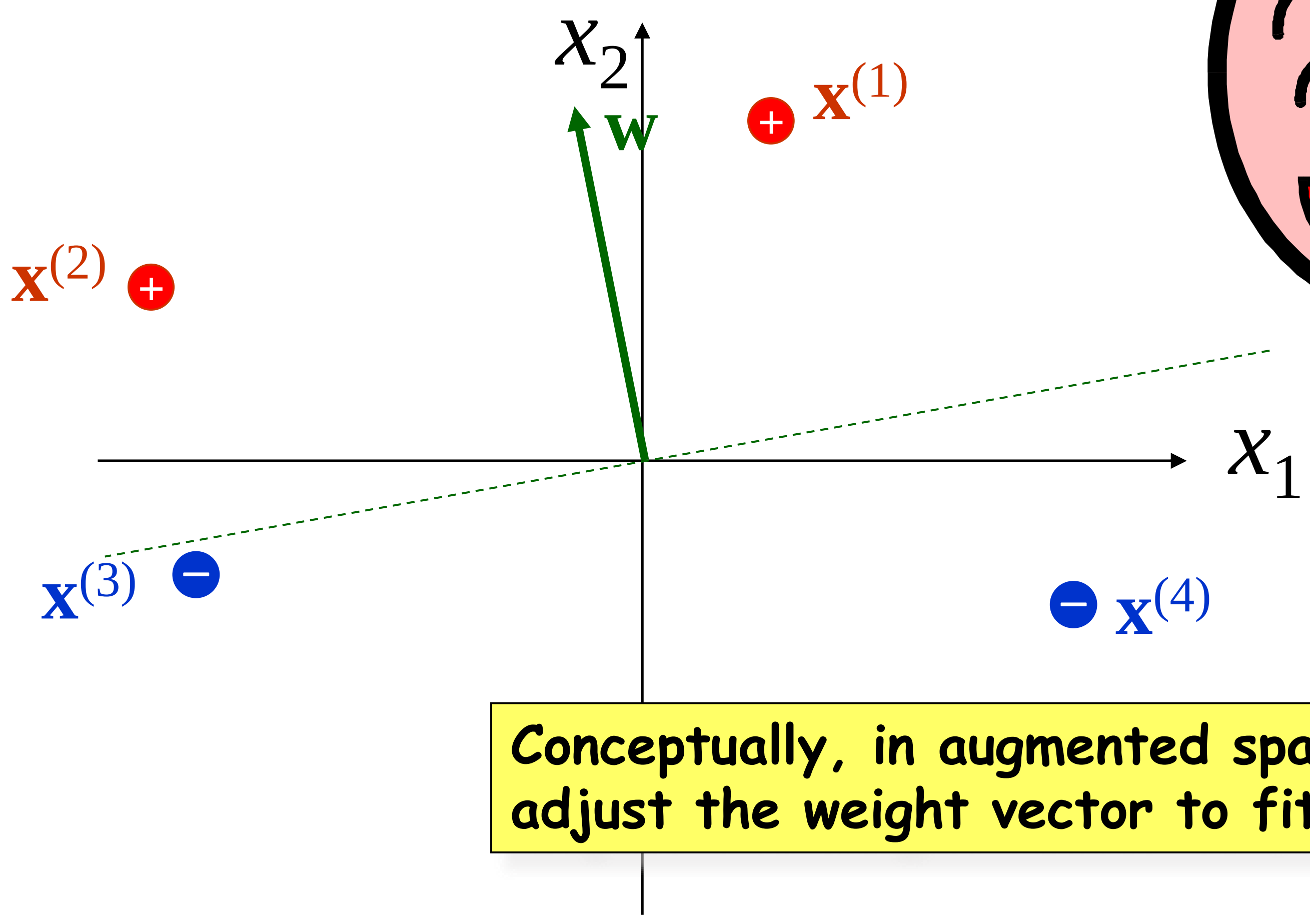
The Learning Scenario



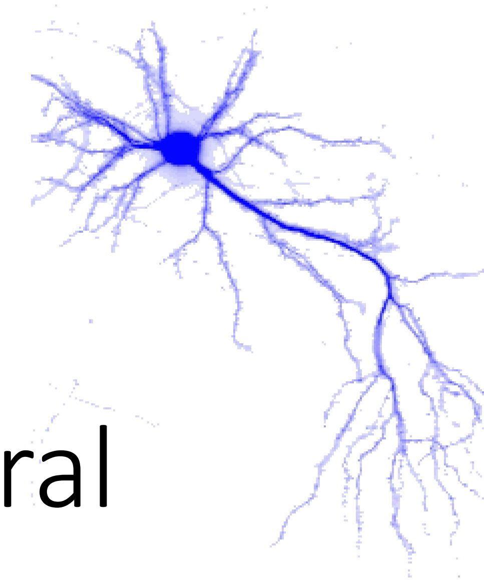
The demonstration is in **augmented space**.

The Learning Scenario





Conceptually, in augmented space, we adjust the weight vector to fit the data.



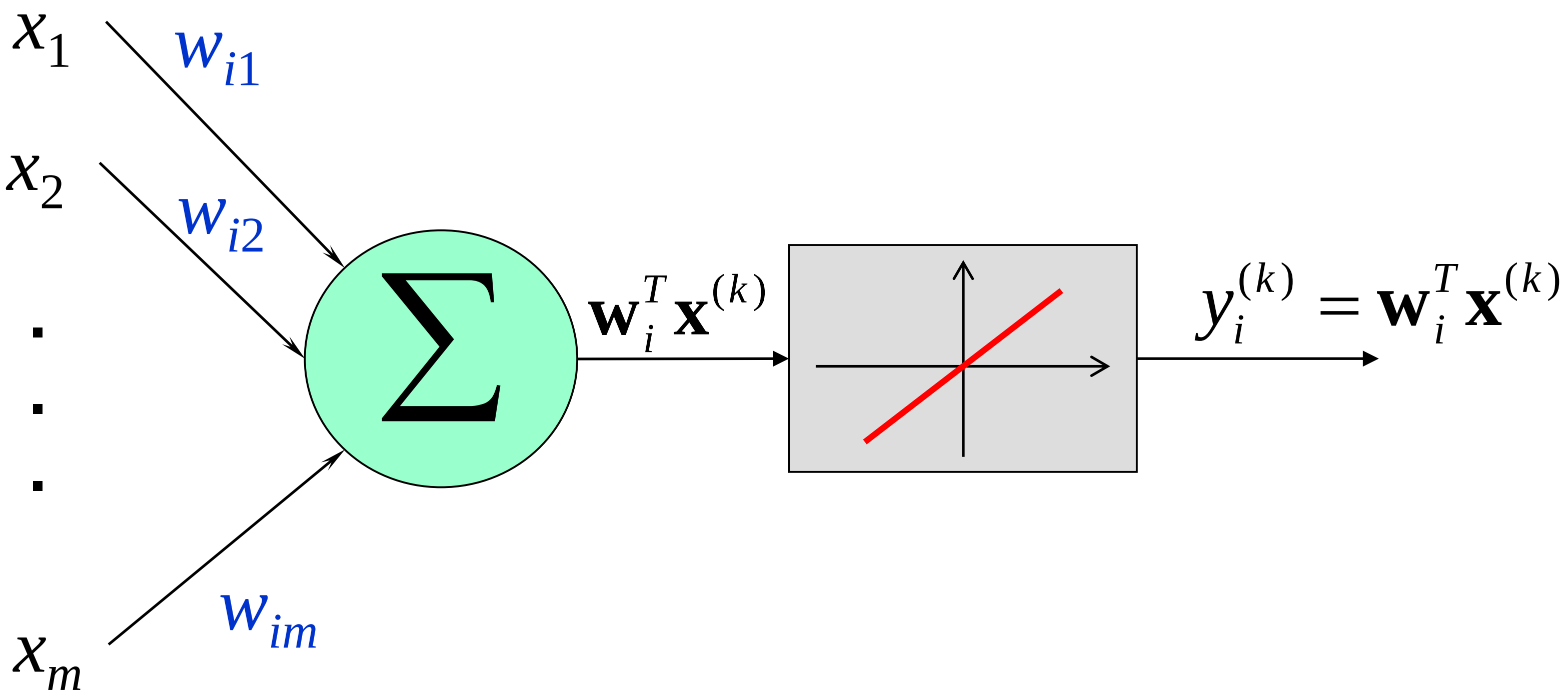
Feed-Forward Neural Networks

Learning Rules for
Single-Layered Perceptron Networks

- Perceptron Learning Rule
- Adaline Leaning Rule
- δ-Learning Rule

Adaline (Adaptive Linear Element)

Widrow [1962]



In what condition, the goal is reachable?

Adaline (Adaptive Linear Element)

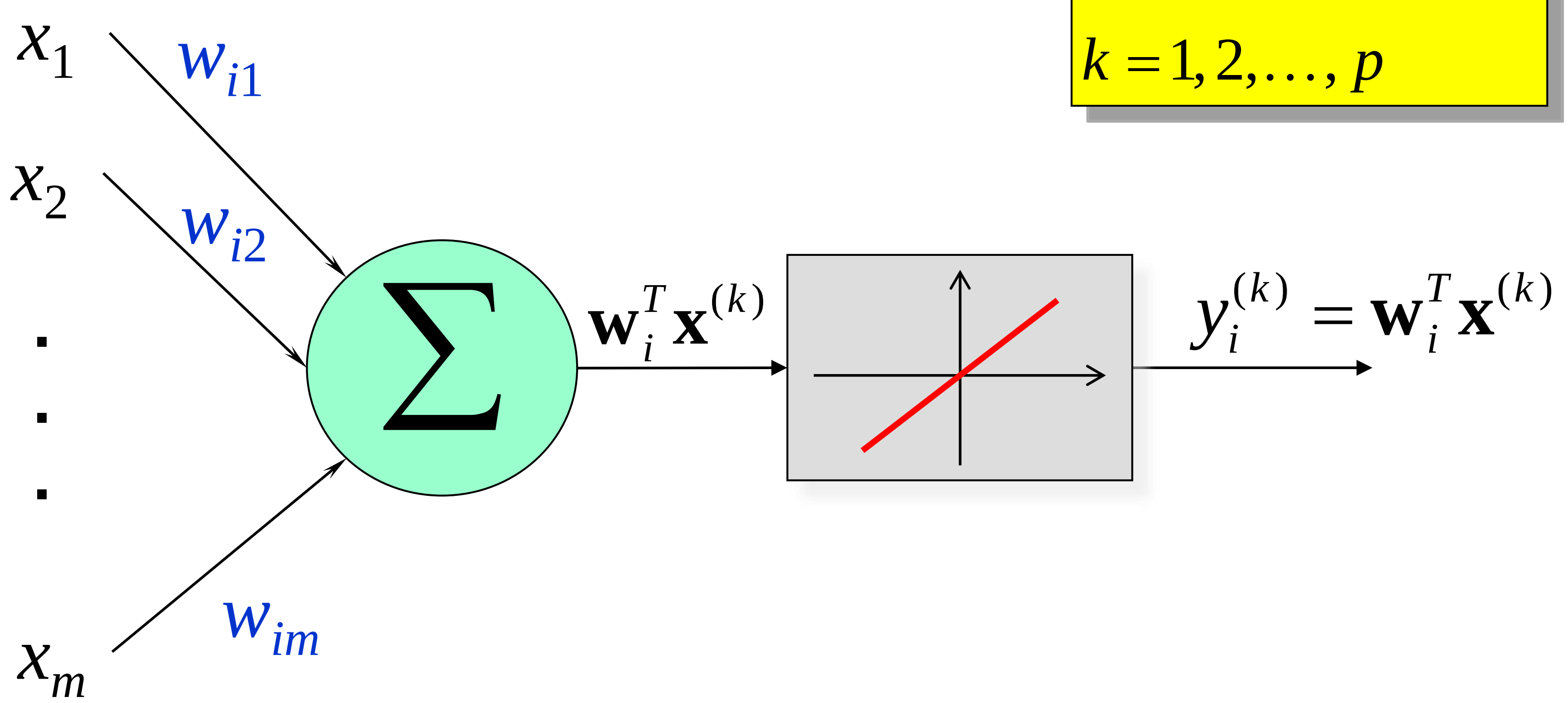
Widrow [1962]

Goal:

$$y_i^{(k)} = \mathbf{w}_i^T \mathbf{x}^{(k)} = d_i^{(k)}$$

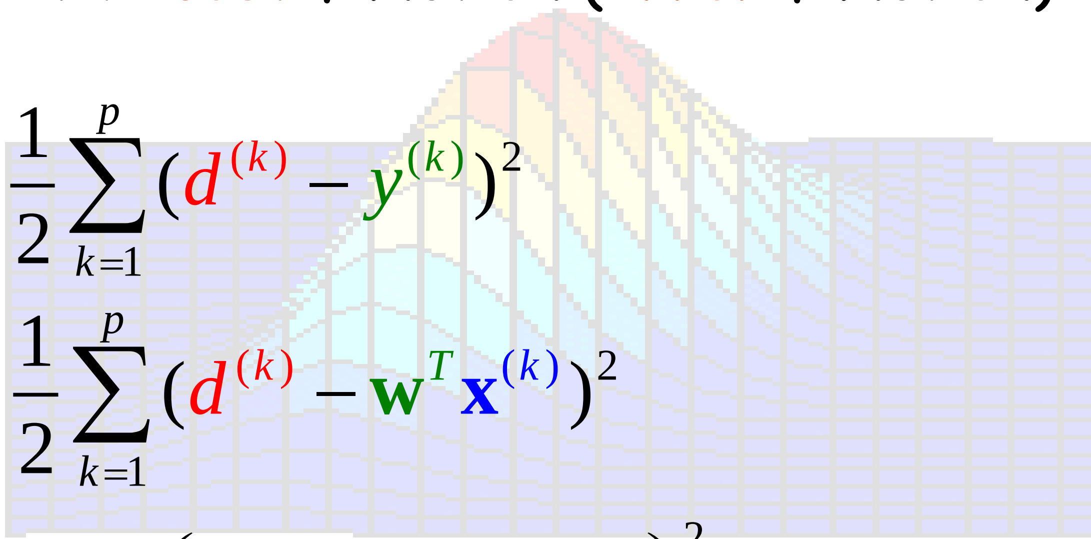
$$i = 1, 2, \ldots, n$$

$$k = 1, 2, \ldots, p$$



LMS (Least Mean Square)

Minimize the cost function (error function):



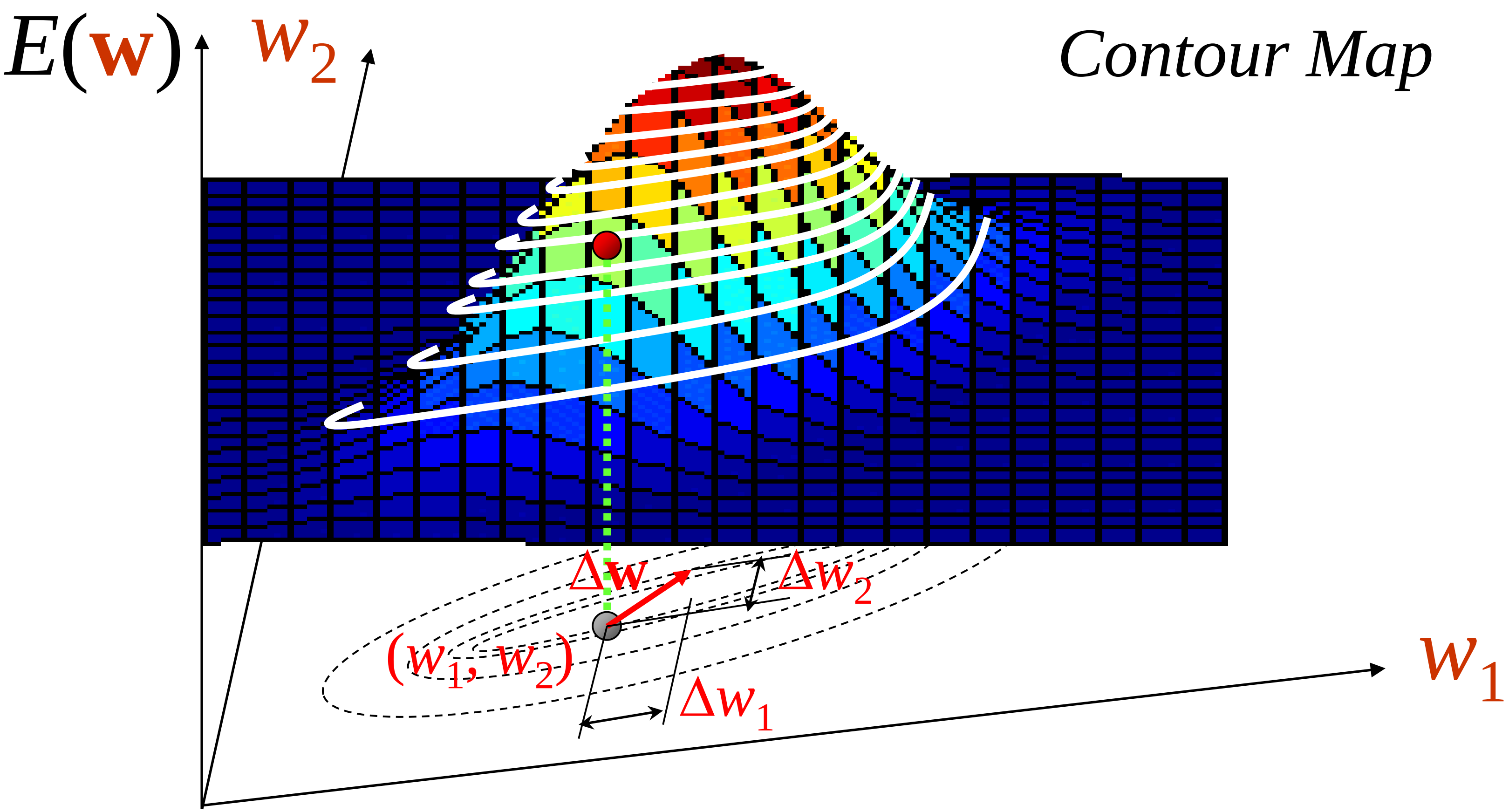
$$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - y^{(k)}\right)^2$$

$$= \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - \mathbf{w}^T\mathbf{x}^{(k)}\right)^2$$

$$= \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - \sum_{l=1}^{m} w_l x_l^{(k)}\right)^2$$

Our goal is to go *downhill*.

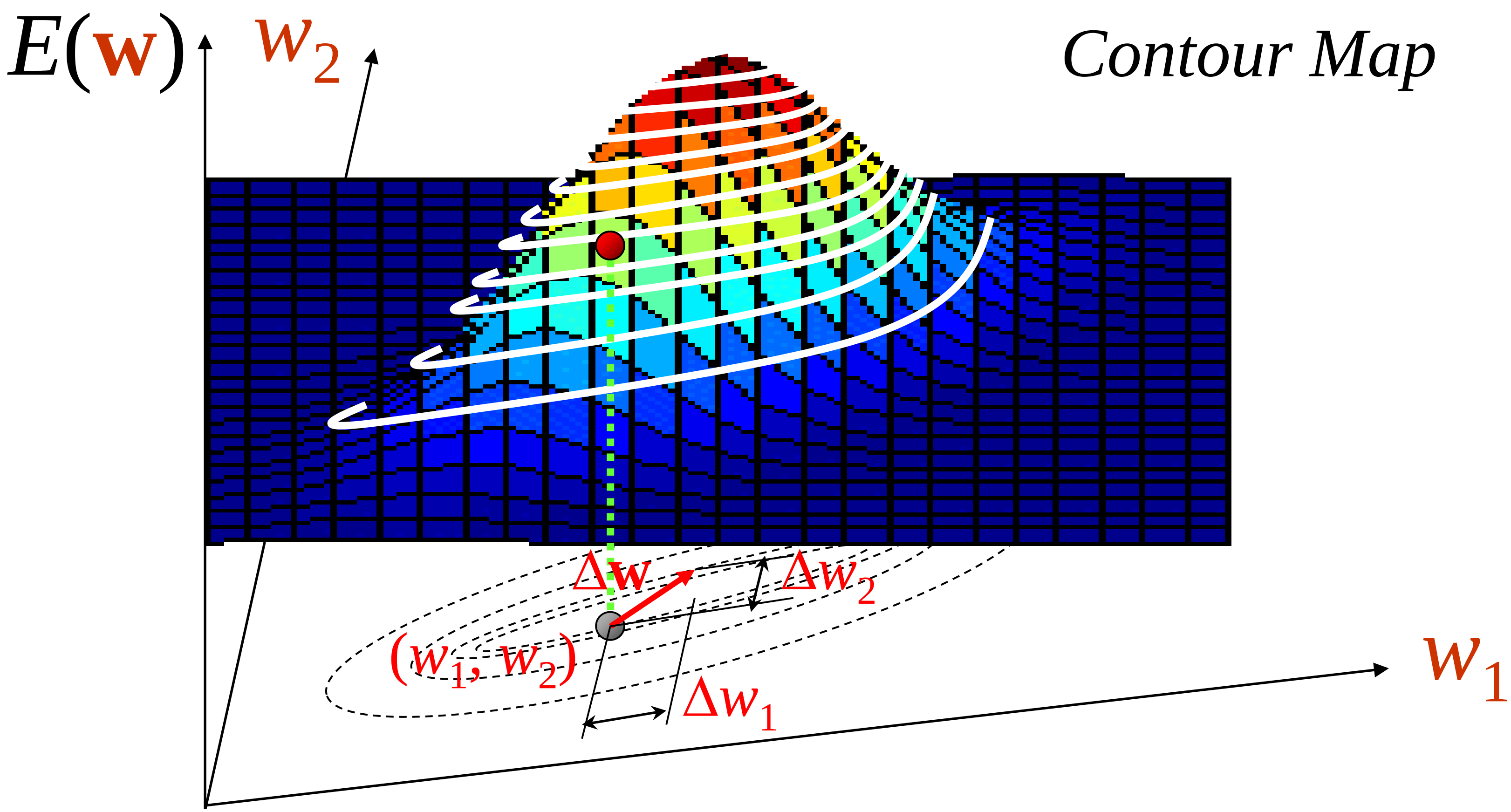
Gradient Decent Algorithm



Our goal is to go *downhill*.

Gradient Decent Algorithm

How to find the steepest decent direction?



Gradient Operator

Let $f(\mathbf{w}) = f\,(w_1, w_2, \ldots, w_m)$ be a function over R^m.

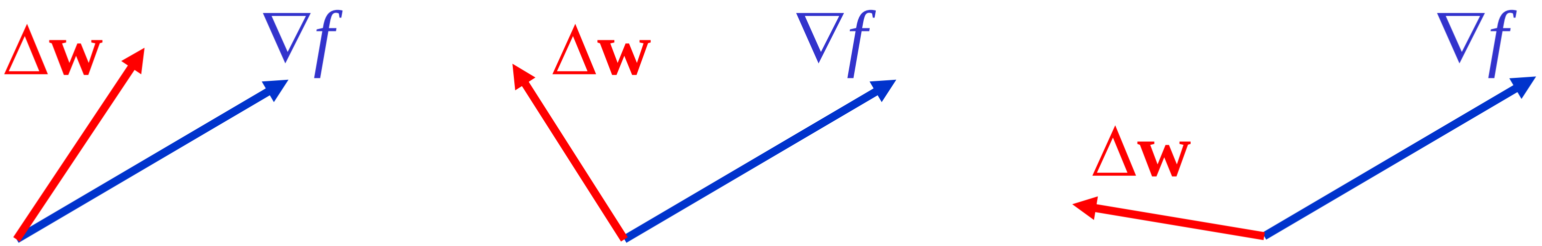
$$df = \frac{\partial f}{\partial w_1} dw_1 + \frac{\partial f}{\partial w_2} dw_2 + \ldots + \frac{\partial f}{\partial w_m} dw_m$$

Define $$\nabla f = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_m} \right)^T$$

$$\Delta \mathbf{w} = \left(dw_1, dw_2, \ldots, dw_m \right)^T$$

$$df = \langle \nabla f, \Delta \mathbf{w} \rangle = \nabla f \bullet \Delta \mathbf{w}$$

Gradient Operator



df : positive

Go uphill

df : zero

Plain

df : negative

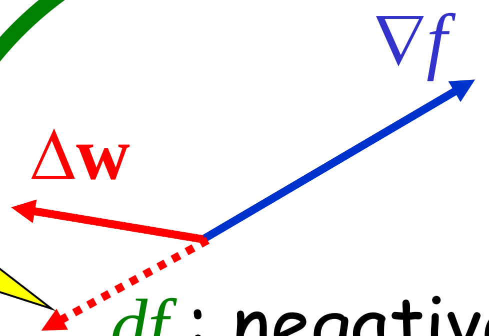
Go downhill

$$df = \langle \nabla f, \Delta \mathbf{w} \rangle = \nabla f \bullet \Delta \mathbf{w}$$

The Steepest Decent Direction

To minimize f, we choose

$$\Delta \mathbf{w} = -\eta \nabla f$$



df : positive	df : zero	df : negative
Go uphill	Plain	Go downhill

$$df = \langle \nabla f, \Delta \mathbf{w} \rangle = \nabla f \bullet \Delta \mathbf{w}$$

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -\sum_{k=1}^{p} \delta^{(k)} x_j^{(k)} \qquad \delta^{(k)} = d^{(k)} - y^{(k)}$$

LMS (Least Mean Square)

Minimize the cost function (error function):

$$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - \sum_{l=1}^{m} w_l x_l^{(k)}\right)^2$$

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -\sum_{k=1}^{p}\left(d^{(k)} - \sum_{l=1}^{m} w_l x_l^{(k)}\right) x_j^{(k)}$$

$$= -\sum_{k=1}^{p}\left(d^{(k)} - \mathbf{w}^T \mathbf{x}^{(k)}\right) x_j^{(k)} = -\sum_{k=1}^{p}\left(\overbrace{d^{(k)} - y^{(k)}}^{\delta^{(k)}}\right) x_j^{(k)}$$

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -\sum_{k=1}^{p} \delta^{(k)} x_j^{(k)} \qquad \delta^{(k)} = d^{(k)} - y^{(k)}$$

Adaline Learning Rule

Minimize the cost function (error function):

$$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - \sum_{l=1}^{m} w_l x_l^{(k)} \right)^2$$

$$\nabla_w E(\mathbf{w}) = \left(\frac{\partial E(\mathbf{w})}{\partial w_1}, \frac{\partial E(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial E(\mathbf{w})}{\partial w_m} \right)^T$$

$$\Delta \mathbf{w} = -\eta \nabla_{\mathbf{w}} E(\mathbf{w})$$ — Weight Modification Rule

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -\sum_{k=1}^{p} \delta^{(k)} x_j^{(k)} \qquad \delta^{(k)} = d^{(k)} - y^{(k)}$$

Learning Modes

- Batch Learning Mode:

$$\Delta w_j = \eta \sum_{k=1}^{p} \delta^{(k)} x_j^{(k)}$$

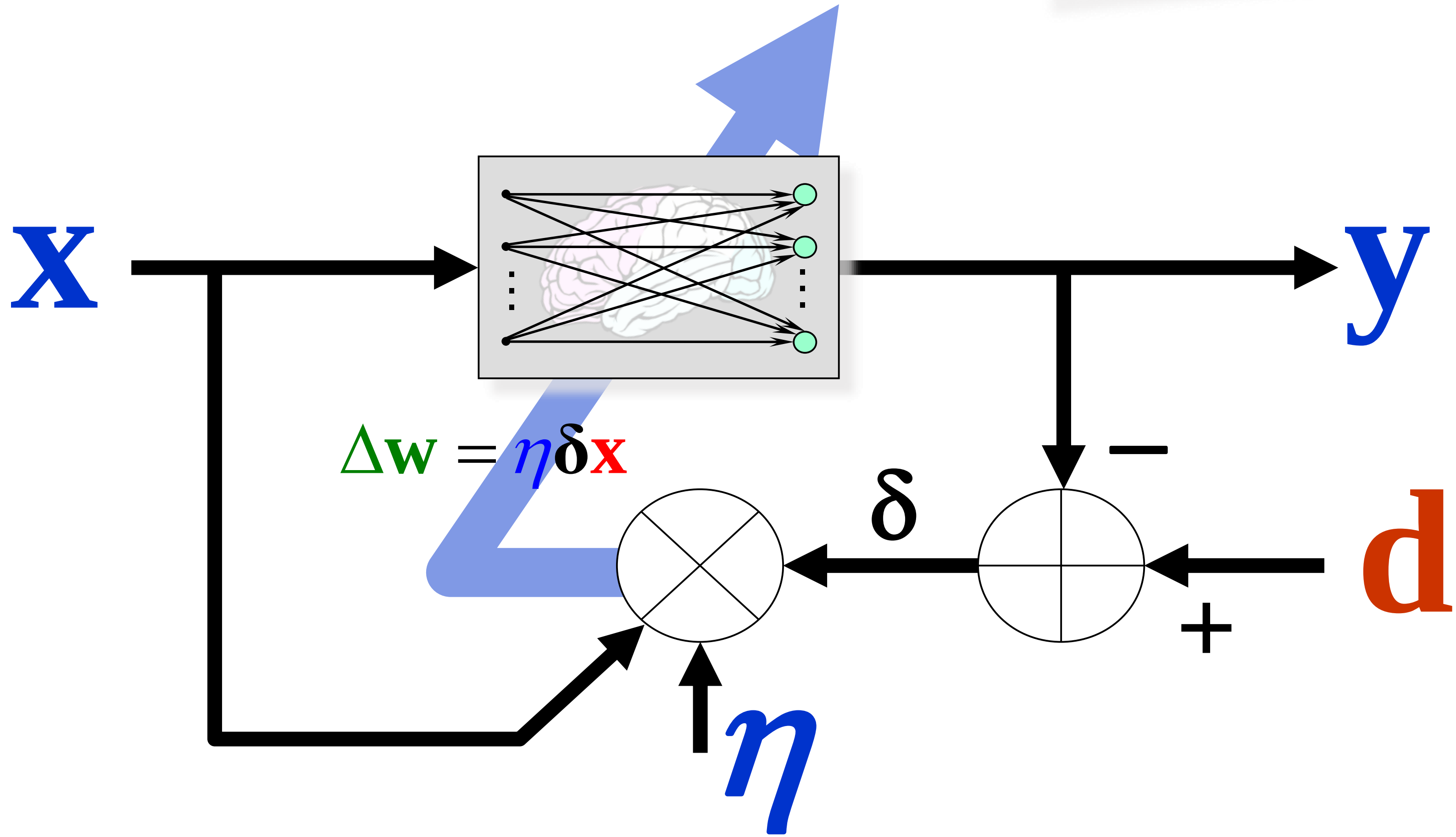
- Incremental Learning Mode:

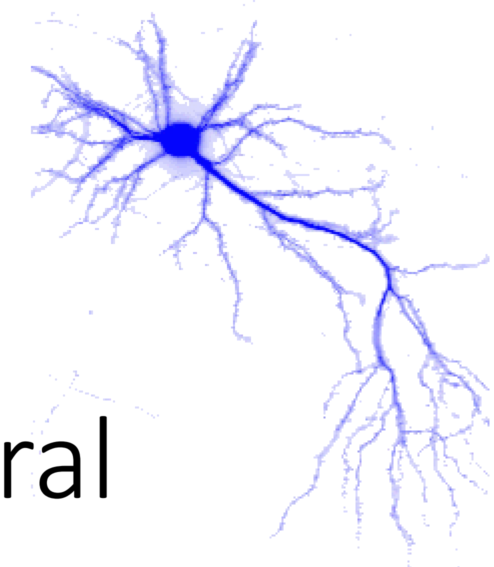
$$\Delta w_j = \eta \delta^{(k)} x_j^{(k)}$$

Summary – Adaline Learning Rule

δ-Learning Rule
LMS Algorithm
Widrow-Hoff Learning Rule

Converge?

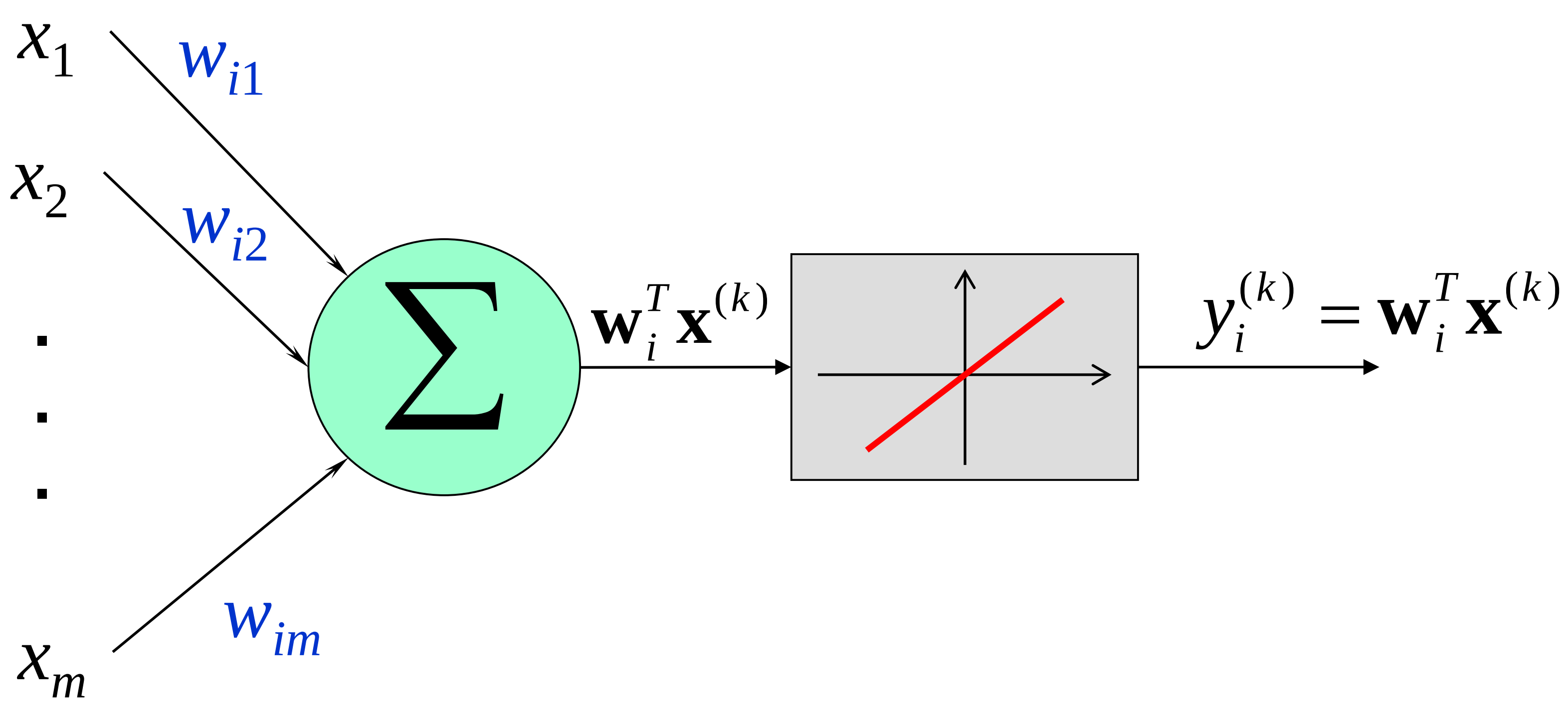




Feed-Forward Neural Networks

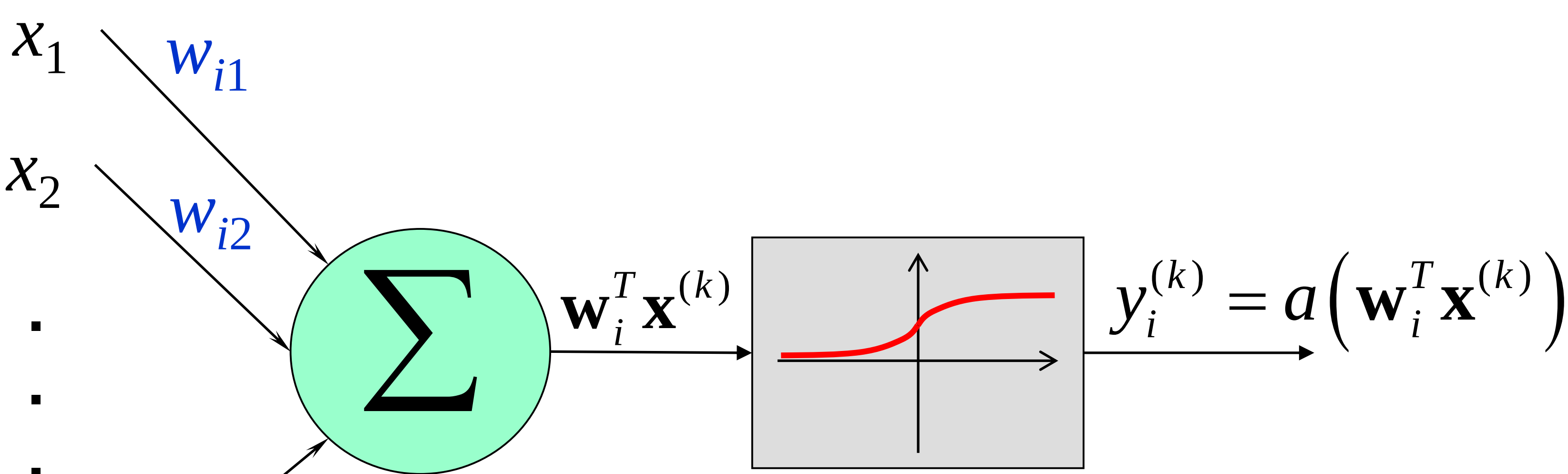
Learning Rules for
Single-Layered Perceptron Networks
● Perceptron Learning Rule
● Adaline Leaning Rule
● δ-Learning Rule

Adaline



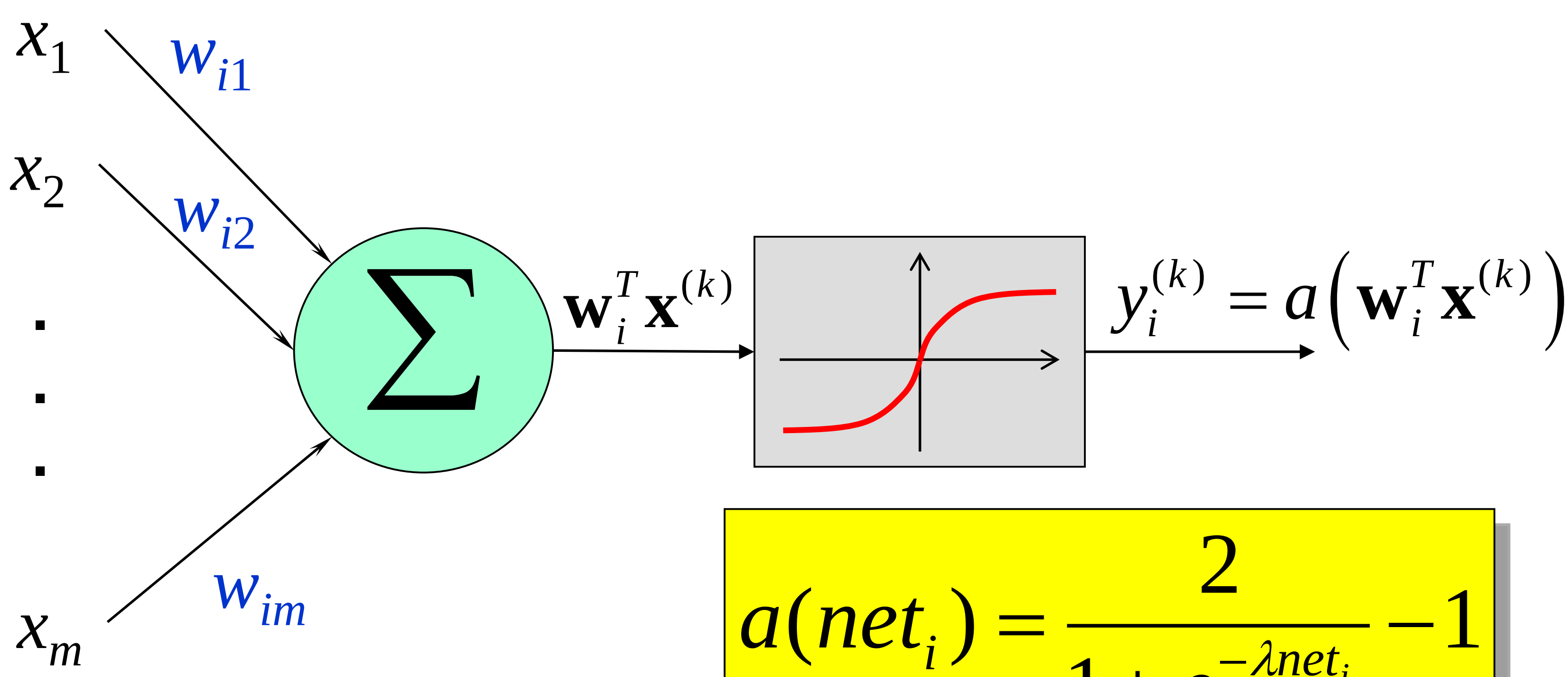
x_1
w_{i1}
x_2
w_{i2}
$\vdots$
x_m
w_{im}
Σ
$\mathbf{w}_i^T \mathbf{x}^{(k)}$
$y_i^{(k)} = \mathbf{w}_i^T \mathbf{x}^{(k)}$

Unipolar Sigmoid



$$a(net_i) = \frac{1}{1 + e^{-\lambda net_i}}$$

Bipolar Sigmoid

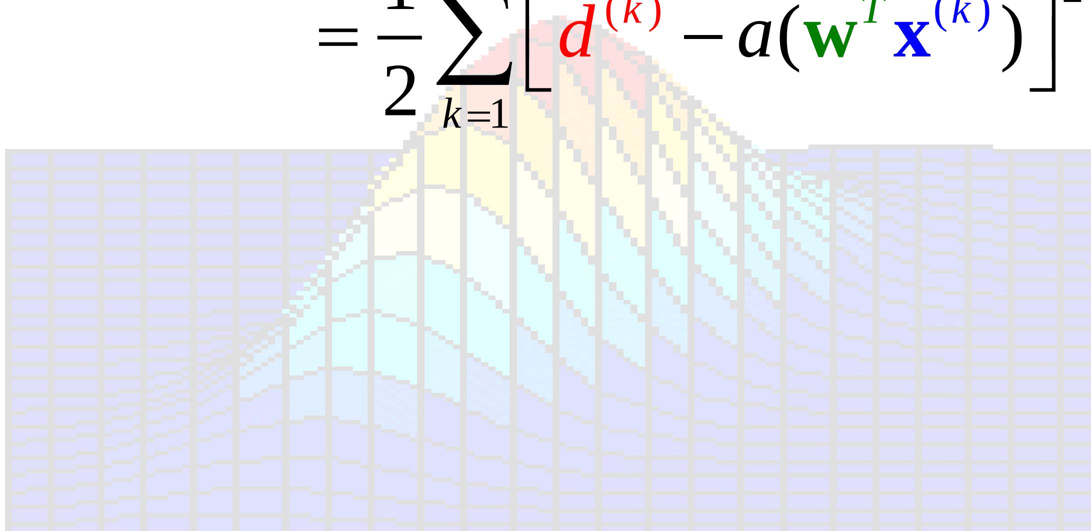


$$a(net_i) = \frac{2}{1 + e^{-\lambda net_i}} - 1$$

Goal

Minimize $$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - y^{(k)}\right)^2$$

$$= \frac{1}{2}\sum_{k=1}^{p}\left[d^{(k)} - a\left(\mathbf{w}^T\mathbf{x}^{(k)}\right)\right]^2$$

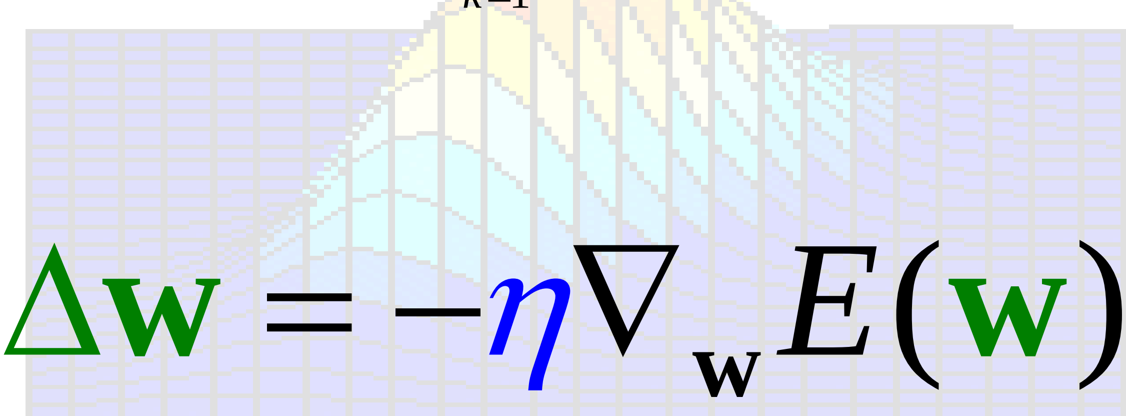


$$\nabla_w E(\mathbf{w}) = \left(\frac{\partial E(\mathbf{w})}{\partial w_1}, \frac{\partial E(\mathbf{w})}{\partial w_2}, \cdots, \frac{\partial E(\mathbf{w})}{\partial w_m} \right)^T$$

Gradient Decent Algorithm

Minimize $E(\mathbf{w}) = \dfrac{1}{2}\sum_{k=1}^{p}(d^{(k)} - y^{(k)})^2$

$$= \frac{1}{2}\sum_{k=1}^{p}\left[d^{(k)} - a(\mathbf{w}^T\mathbf{x}^{(k)})\right]^2$$



$$\Delta\mathbf{w} = -\eta\nabla_{\mathbf{w}} E(\mathbf{w})$$

$$\nabla_w E(\mathbf{w}) = \left(\frac{\partial E(\mathbf{w})}{\partial w_1}, \frac{\partial E(\mathbf{w})}{\partial w_2}, \cdots, \frac{\partial E(\mathbf{w})}{\partial w_m} \right)^T$$

The Gradient

$$\boxed{y^{(k)} = a\left(\mathbf{w}^T \mathbf{x}^{(k)}\right)}$$

Minimize $E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{p} (d^{(k)} - y^{(k)})^2$

Depends on the activation function used.

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -\sum_{k=1}^{p} (d^{(k)} - y^{(k)}) \frac{\partial y^{(k)}}{\partial w_j}$$

$$= -\sum_{k=1}^{p} (d^{(k)} - y^{(k)}) \underbrace{\frac{\partial a\left(net^{(k)}\right)}{\partial net^{(k)}}}_{?} \underbrace{\frac{\partial net^{(k)}}{\partial w_j}}_{?}$$

$$net^{(k)} = \mathbf{w}^T \mathbf{x}^{(k)} = \sum_{i=1}^{m} w_i x_i^{(k)} \Rightarrow \frac{\partial net^{(k)}}{\partial w_j} = x_j^{(k)}$$

$$\nabla_w E(\mathbf{w}) = \left(\frac{\partial E(\mathbf{w})}{\partial w_1}, \frac{\partial E(\mathbf{w})}{\partial w_2}, \cdots, \frac{\partial E(\mathbf{w})}{\partial w_m} \right)^T$$

Weight Modification Rule

$$\boxed{y^{(k)} = a\left(net^{(k)}\right)}$$

$$\boxed{\delta^{(k)} = d^{(k)} - y^{(k)}}$$

Minimize $$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{p}(d^{(k)} - y^{(k)})^2$$

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -\sum_{k=1}^{p}(d^{(k)} - y^{(k)})x_j^{(k)} \frac{\partial a\left(net^{(k)}\right)}{\partial net^{(k)}}$$

Learning Rule

Batch $$\Delta w_j = \eta \sum_{k=1}^{p} \delta^{(k)} x_j^{(k)} \frac{\partial a\left(net^{(k)}\right)}{\partial net^{(k)}}$$

Incremental $$\Delta w_j = \eta \delta^{(k)} x_j^{(k)} \frac{\partial a\left(net^{(k)}\right)}{\partial net^{(k)}}$$

$$\nabla_w E(\mathbf{w}) = \left(\frac{\partial E(\mathbf{w})}{\partial w_1}, \frac{\partial E(\mathbf{w})}{\partial w_2}, \cdots, \frac{\partial E(\mathbf{w})}{\partial w_m} \right)^T$$

The Learning Efficacy

$$\boxed{y^{(k)} = a\left(net^{(k)}\right)}$$

Minimize $$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - y^{(k)}\right)^2$$

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -\sum_{k=1}^{p}\left(d^{(k)} - y^{(k)}\right)x_j^{(k)} \boxed{\frac{\partial a\left(net^{(k)}\right)}{\partial net^{(k)}}}$$

Sigmoid

Adaline	Unipolar	Bipolar
$a(net) = net$	$a(net) = \frac{1}{1+e^{-\lambda net}}$	$a(net) = \frac{2}{1+e^{-\lambda net}} - 1$
$\frac{\partial a(net)}{\partial net} = 1$	$\frac{\partial a(net)}{\partial net} = \lambda y^{(k)}\left(1 - y^{(k)}\right)$	**Exercise**

$$\nabla_w E(\mathbf{w}) = \left(\frac{\P E(\mathbf{w})}{\P w_1}, \frac{\P E(\mathbf{w})}{\P w_2}, \square \;, \frac{\P E(\mathbf{w})}{\partial w_m} \right)^T$$

Learning Rule – Unipolar Sigmoid

$$\boxed{\delta^{(k)} = d^{(k)} - y^{(k)}}$$

Minimize $$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{p}(d^{(k)} - y^{(k)})^2$$

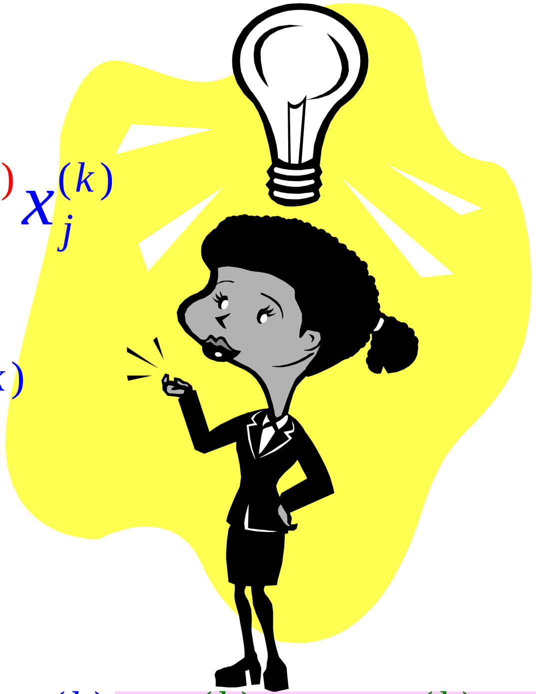
$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -\sum_{k=1}^{p}(d^{(k)} - y^{(k)})x_j^{(k)}\lambda y^{(k)}(1 - y^{(k)})$$

$$= -\sum_{k=1}^{p}\delta^{(k)}x_j^{(k)}\lambda y^{(k)}(1 - y^{(k)})$$

$$\Delta w_j = \eta\sum_{k=1}^{p}\delta^{(k)}x_j^{(k)}\lambda y^{(k)}(1 - y^{(k)})$$ — Weight Modification Rule

Comparisons

$$\lambda y^{(k)}(1 - y^{(k)})$$



Adaline

Batch $\Delta w_j = \eta \sum_{k=1}^{p} \delta^{(k)} x_j^{(k)}$

Incremental $\Delta w_j = \eta \delta^{(k)} x_j^{(k)}$

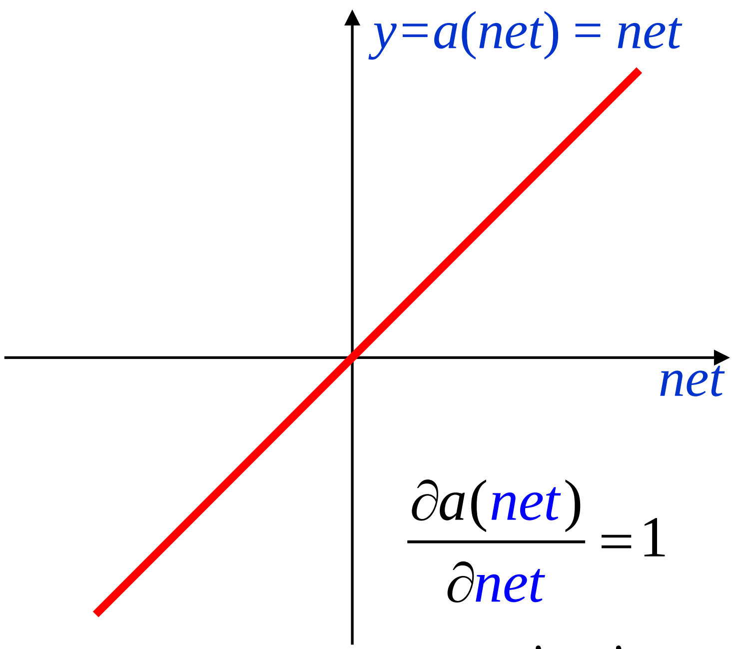
Sigmoid

Batch $\Delta w_j = \eta \sum_{k=1}^{p} \delta^{(k)} x_j^{(k)} \lambda y^{(k)}(1 - y^{(k)})$

Incremental $\Delta w_j = \eta \delta^{(k)} x_j^{(k)} \lambda y^{(k)}(1 - y^{(k)})$

The Learning Efficacy

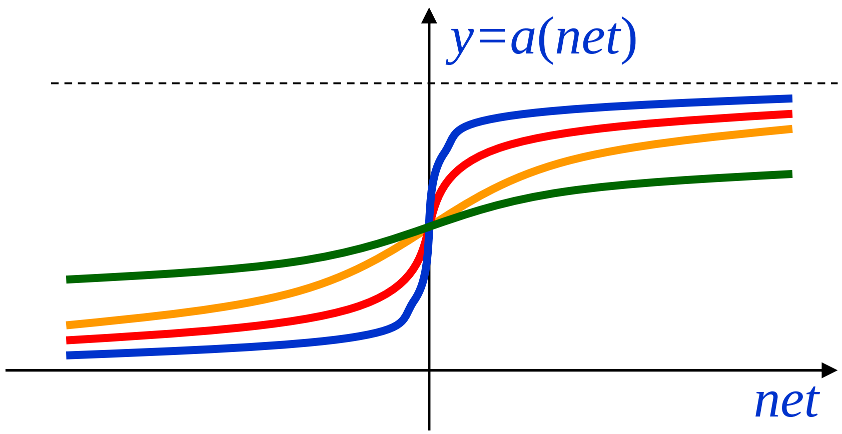
Adaline



$$\frac{\partial a(net)}{\partial net} = 1$$

constant

Sigmoid

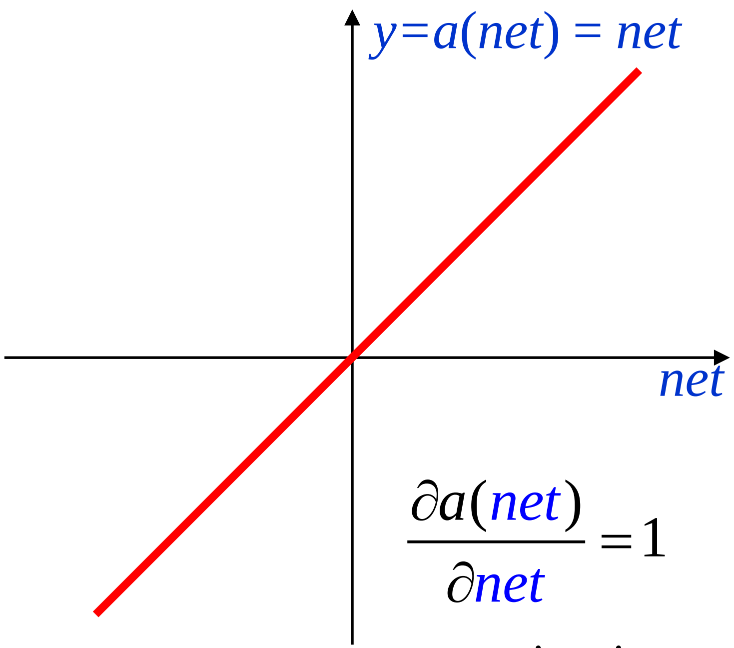


$$\frac{\partial a(net)}{\partial net} = \lambda y(1-y)$$

depends on output

The Learning Efficacy

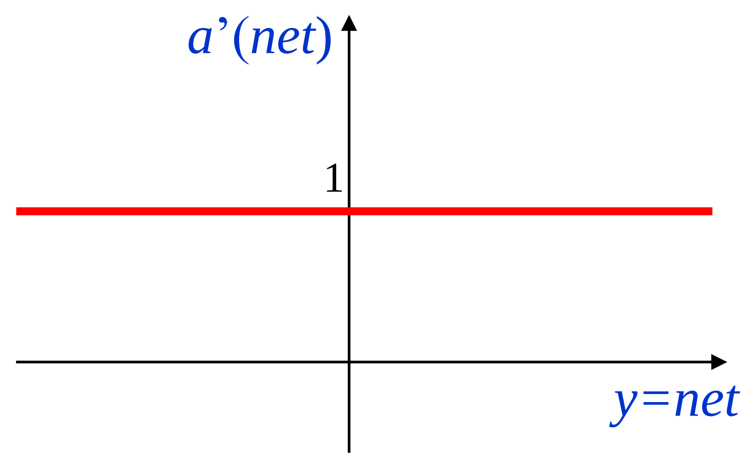
Adaline



$$\frac{\partial a(net)}{\partial net} = 1$$

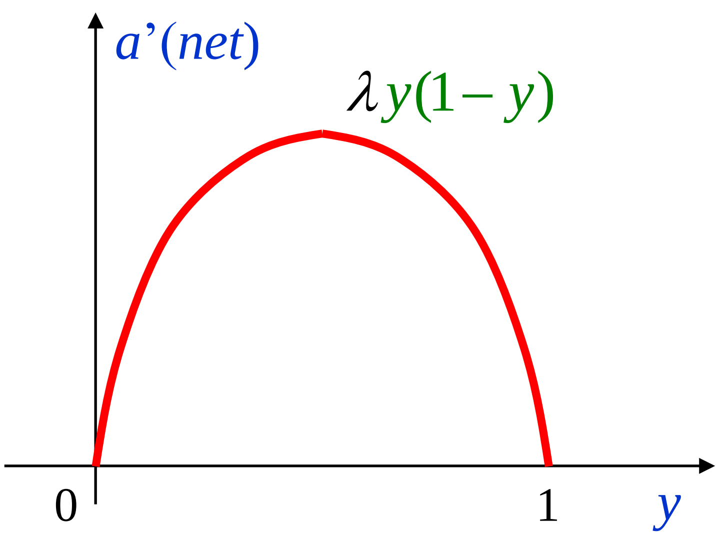
constant

The learning efficacy of Adaline is constant meaning that the Adline will never get saturated.

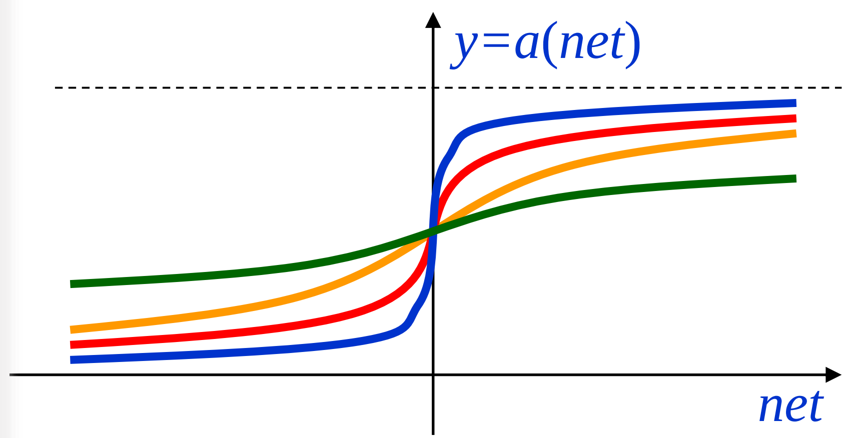


The Learning Efficacy

The sigmoid will get saturated if its output value nears the two extremes.



Sigmoid

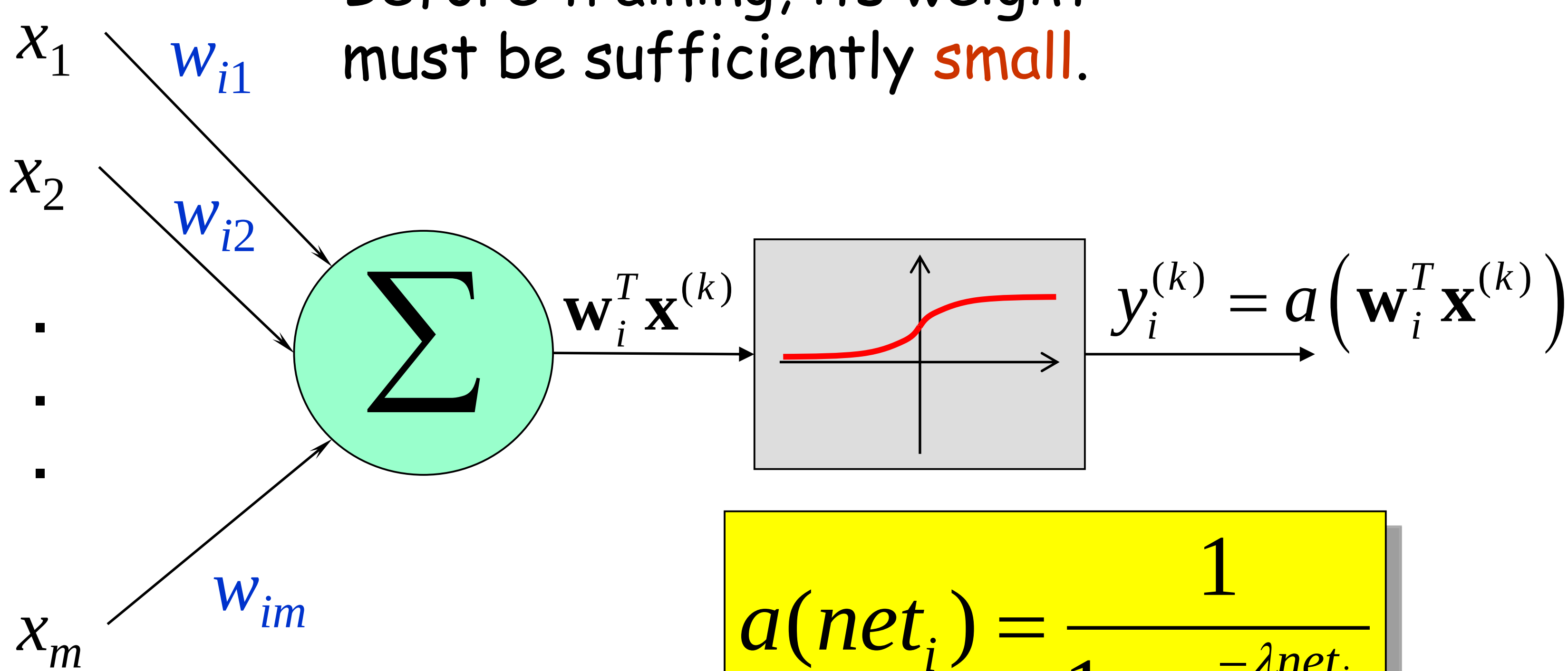


$$\frac{\partial a(net)}{\partial net} = \lambda y(1-y)$$

depends on output

Initialization for Sigmoid Neurons

Before training, its weight must be sufficiently small.



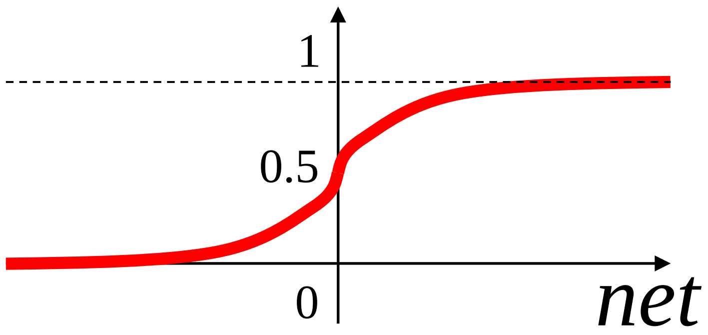
$$a(net_i) = \frac{1}{1+e^{-\lambda net_i}}$$

Feed-Forward Neural Networks

Back Propagation Learning algorithm

Activation Function — Sigmoid

$$y = a(net) = \frac{1}{1+e^{-\lambda net}}$$



$$a'(net) = -\left(\frac{1}{1+e^{-\lambda net}}\right)^2 \cdot (-\lambda)e^{-\lambda net}$$

$$e^{-\lambda net} = \frac{1-y}{y}$$

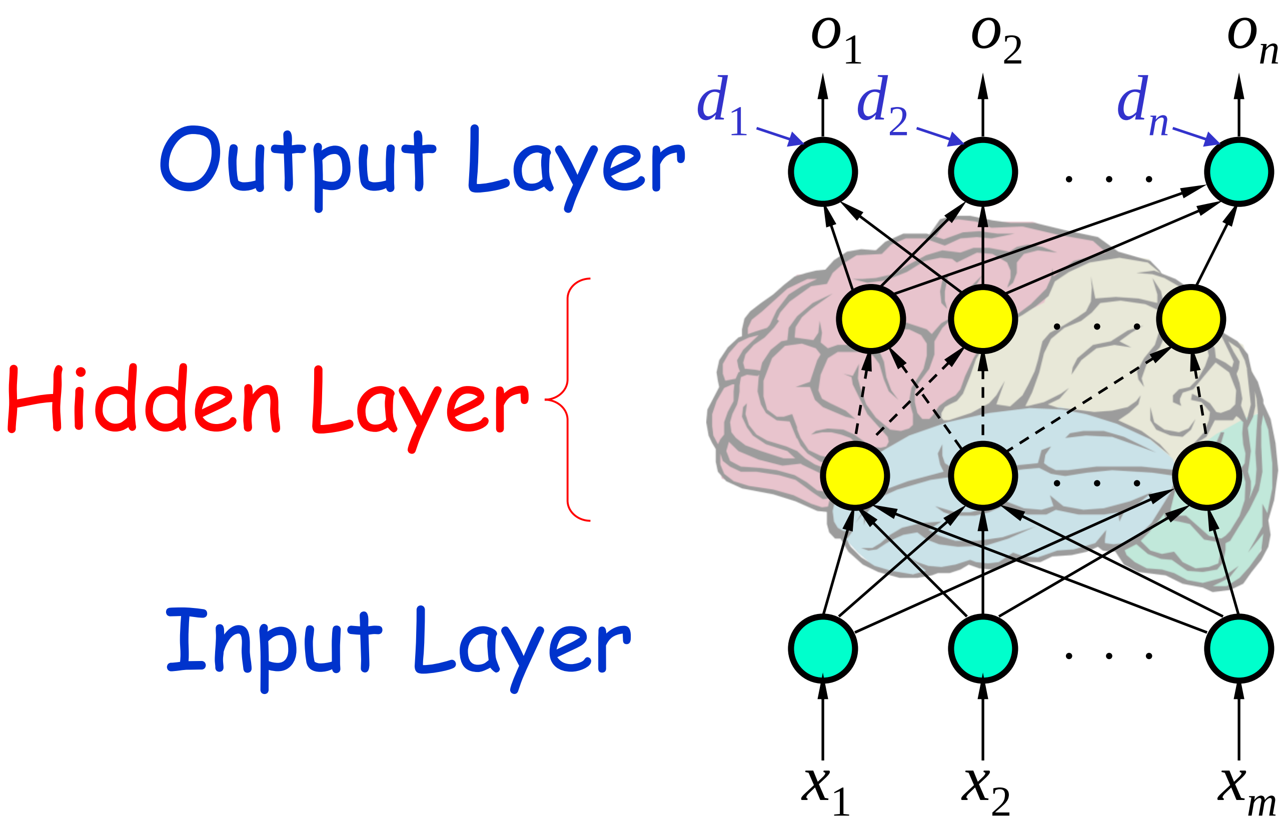
$$a'(net) = \lambda y(1-y)$$

Remember this

Training Set

$$\mathbf{T} = \left\{ (\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), \ldots, (\mathbf{x}^{(p)}, \mathbf{d}^{(p)}) \right\}$$

Supervised Learning



Training Set

$$\mathbf{T} = \left\{ (\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), \ldots, (\mathbf{x}^{(p)}, \mathbf{d}^{(p)}) \right\}$$

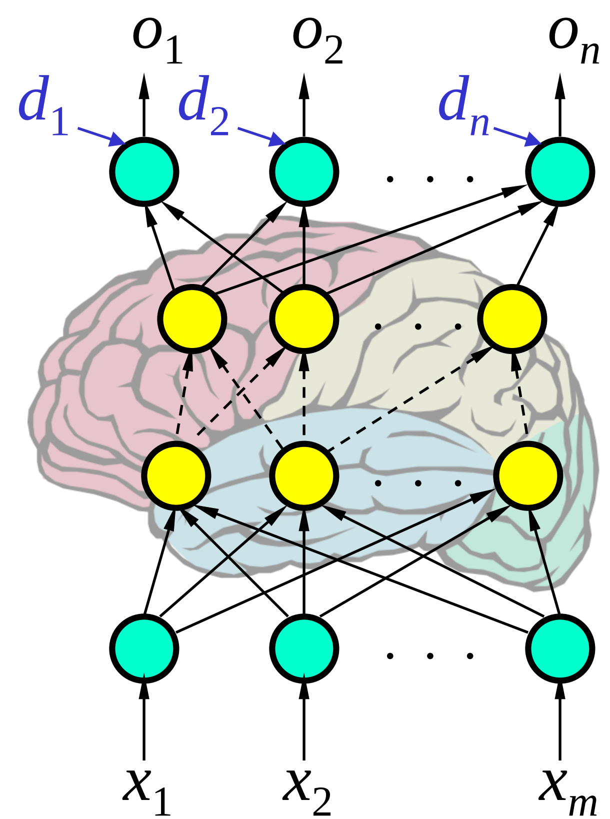
Supervised Learning

Sum of Squared Errors

$$E^{(l)} = \frac{1}{2} \sum_{j=1}^{n} \left[d_j^{(l)} - o_j^{(l)} \right]^2$$

Goal:

Minimize $E = \sum_{l=1}^{p} E^{(l)}$

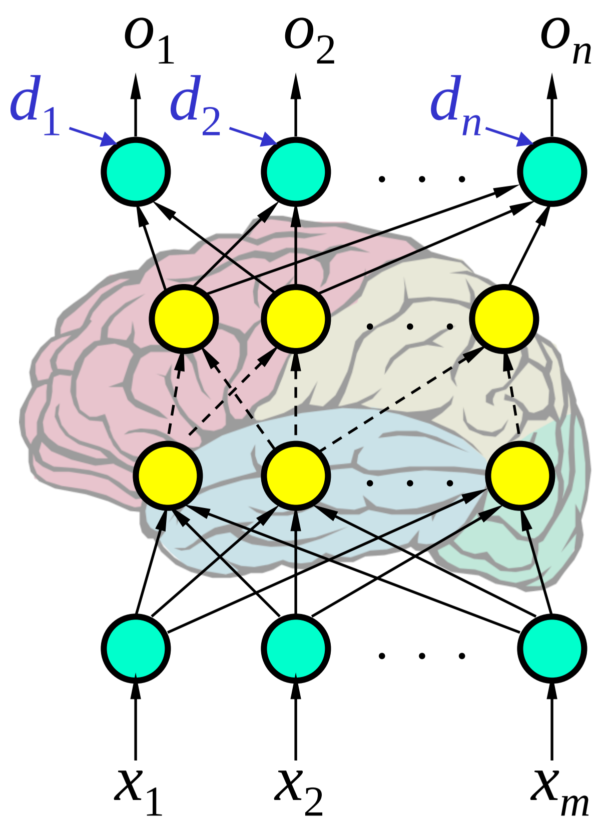


$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Back Propagation Learning Algorithm

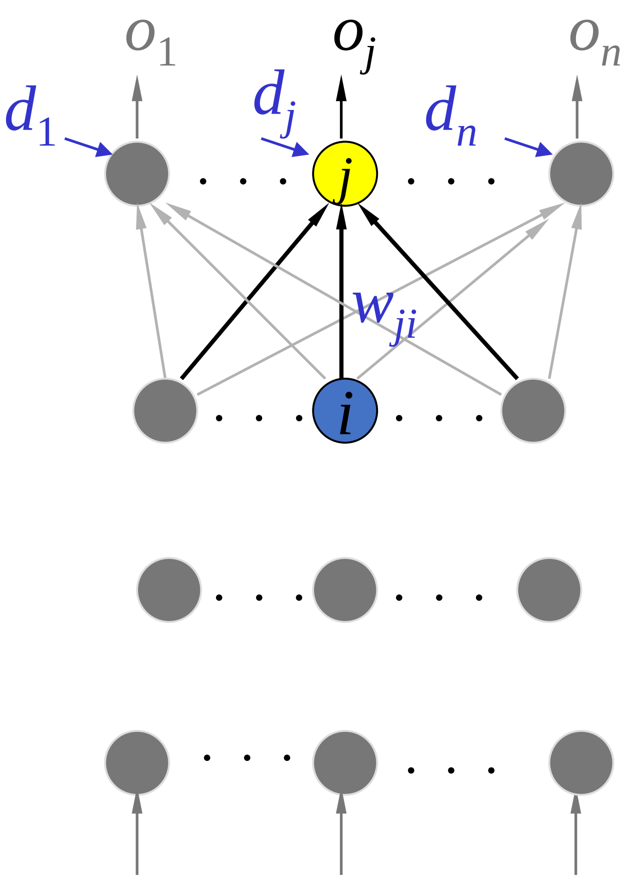
- Learning on Output Neurons
- Learning on Hidden Neurons



$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Output Neurons



$$o_j^{(l)} = a(net_j^{(l)})$$

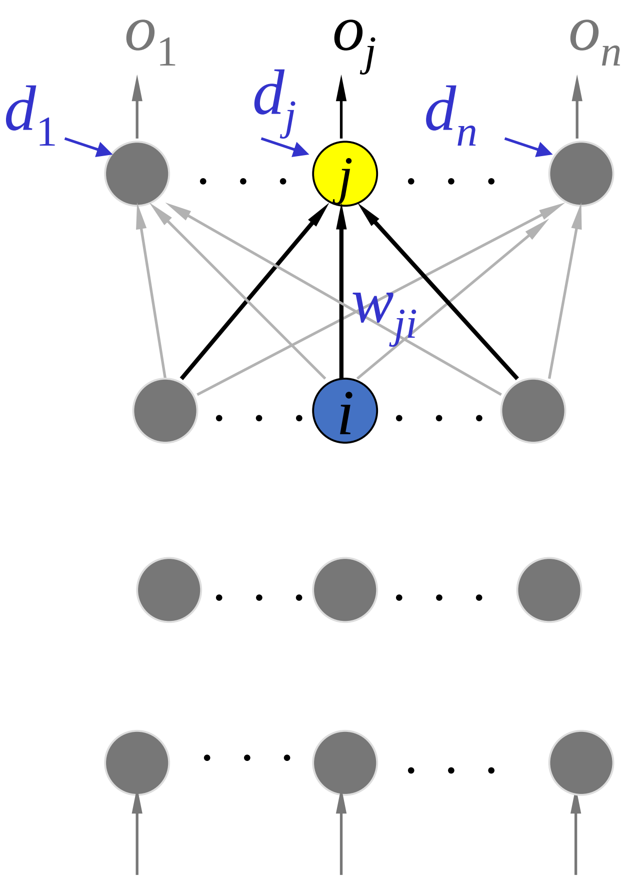
$$net_j^{(l)} = \sum w_{ji} o_i^{(l)}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p}\frac{\partial E^{(l)}}{\partial w_{ji}}$$

$$\frac{\partial E^{(l)}}{\partial w_{ji}} = \underbrace{\frac{\partial E^{(l)}}{\partial net_j^{(l)}}}_{?}\;\underbrace{\frac{\partial net_j^{(l)}}{\partial w_{ji}}}_{?}$$

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2 \qquad E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Output Neurons



$$o_j^{(l)} = a(net_j^{(l)}) \qquad net_j^{(l)} = \sum w_{ji} o_i^{(l)}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p} \frac{\partial E^{(l)}}{\partial w_{ji}}$$

$$\frac{\partial E^{(l)}}{\partial w_{ji}} = \boxed{\frac{\partial E^{(l)}}{\partial net_j^{(l)}}} \frac{\partial net_j^{(l)}}{\partial w_{ji}}$$

$$\frac{\partial E^{(l)}}{\partial net_j^{(l)}} = \frac{\partial E^{(l)}}{\partial o_j^{(l)}} \frac{\partial o_j^{(l)}}{\partial net_j^{(l)}}$$

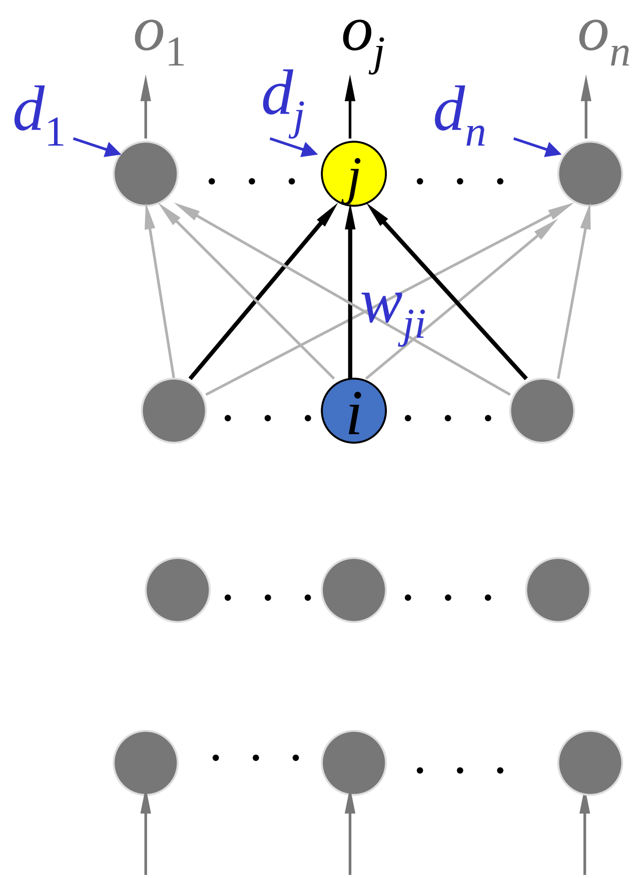
$$-(d_j^{(l)} - o_j^{(l)})$$

depends on the activation function

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Output Neurons



$$o_j^{(l)} = a(net_j^{(l)})$$

$$net_j^{(l)} = \sum w_{ji} o_i^{(l)}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p} \frac{\partial E^{(l)}}{\partial w_{ji}}$$

$$\frac{\partial E^{(l)}}{\partial w_{ji}} = \boxed{\frac{\partial E^{(l)}}{\partial net_j^{(l)}}} \frac{\partial net_j^{(l)}}{\partial w_{ji}}$$

$$\frac{\partial E^{(l)}}{\partial net_j^{(l)}} = \frac{\partial E^{(l)}}{\partial o_j^{(l)}} \frac{\partial o_j^{(l)}}{\partial net_j^{(l)}}$$

$-(d_j^{(l)} - o_j^{(l)})$

Using sigmoid,

$\lambda o_j^{(l)}(1 - o_j^{(l)})$

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

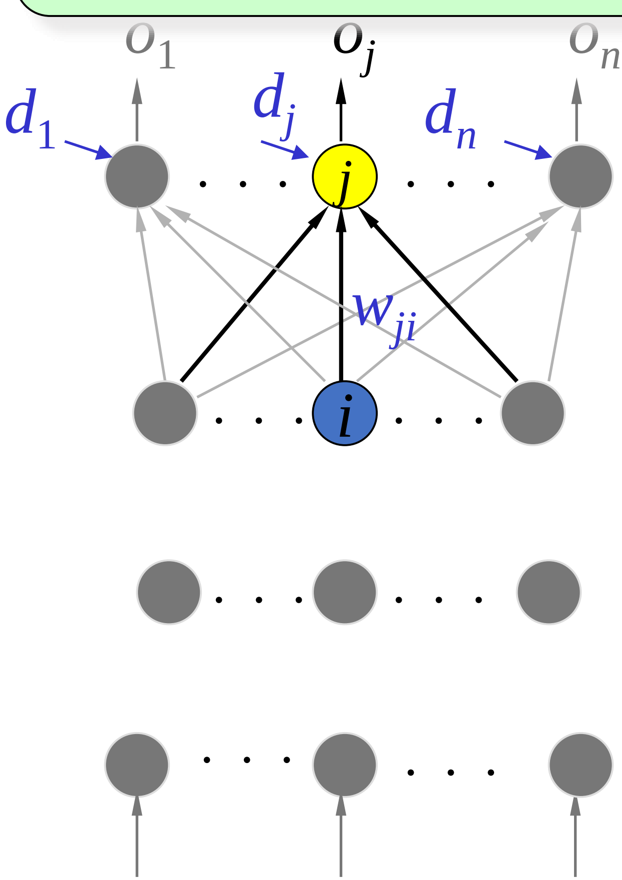
Learning on Output Neurons

$$\delta_j^{(l)} = \frac{\partial E^{(l)}}{\partial net_j^{(l)}} = -(d_j^{(l)} - o_j^{(l)})\lambda o_j^{(l)}(1-o_j^{(l)})$$

$(net_j^{(l)})$

net ... $o_i^{(l)}$

$\delta_j^{(l)}$



$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p}\frac{\partial E^{(l)}}{\partial w_{ji}}$$

$$\frac{\partial E^{(l)}}{\partial w_{ji}} = \boxed{\frac{\partial E^{(l)}}{\partial net_j^{(l)}}}\frac{\partial net_j^{(l)}}{\partial w_{ji}}$$

$$\frac{\partial E^{(l)}}{\partial net_j^{(l)}} = \frac{\partial E^{(l)}}{\partial o_j^{(l)}}\frac{\partial o_j^{(l)}}{\partial net_j^{(l)}}$$

$-(d_j^{(l)} - o_j^{(l)})$

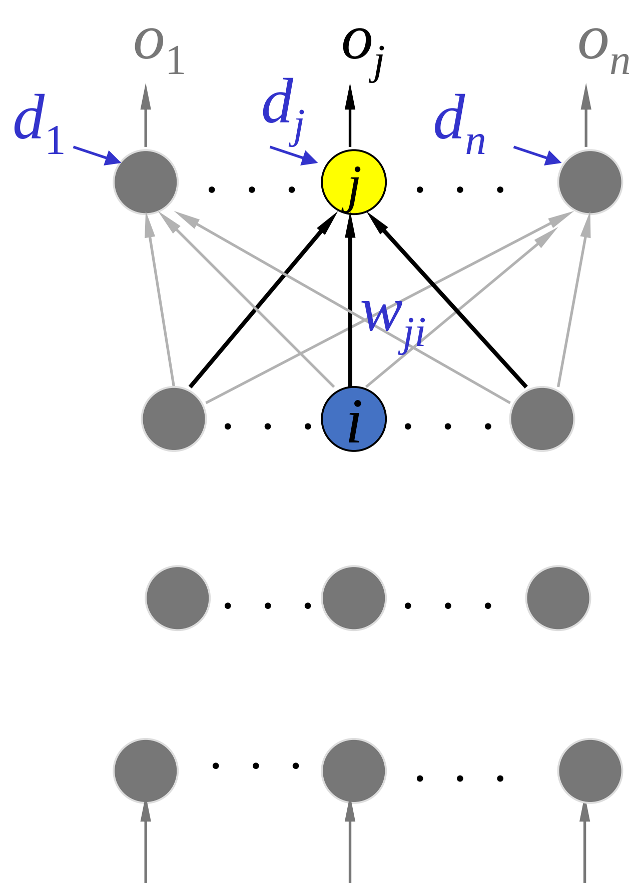
Using sigmoid,

$\lambda o_j^{(l)}(1-o_j^{(l)})$

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Output Neurons



$$o_j^{(l)} = a(net_j^{(l)})$$

$$net_j^{(l)} = \sum w_{ji} o_i^{(l)}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p} \frac{\partial E^{(l)}}{\partial w_{ji}}$$

$$\frac{\partial E^{(l)}}{\partial w_{ji}} = \frac{\partial E^{(l)}}{\partial net_j^{(l)}} \boxed{\frac{\partial net_j^{(l)}}{\partial w_{ji}}} \rightarrow o_i^{(l)}$$

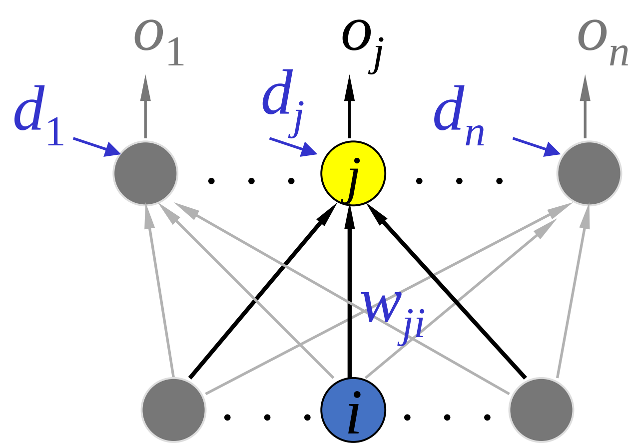
$$\frac{\partial E^{(l)}}{\partial w_{ji}} = \delta_j^{(l)} o_i^{(l)}$$

$$= -(d_j^{(l)} - o_j^{(l)}) \lambda o_j^{(l)} (1 - o_j^{(l)}) o_i^{(l)}$$

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Output Neurons



$$o_j^{(l)} = a(net_j^{(l)})$$

$$net_j^{(l)} = \sum w_{ji} o_i^{(l)}$$

$$\frac{\partial E}{\quad} \quad \frac{\partial}{\quad} \sum^{p} E^{(l)} \quad \sum^{p} \frac{\partial E^{(l)}}{\quad}$$

How to train the weights connecting to output neurons?

$$\partial w_{ji} \qquad \partial net_j^{(l)} \qquad \boxed{\partial w_{ji}} \rightarrow o_i^{(l)}$$

$$\frac{\partial E}{\partial w_{ji}} = \sum_{l=1}^{p} \delta_j^{(l)} o_i^{(l)}$$

$$\Delta w_{ji} = -\eta \sum_{l=1}^{p} \delta_j^{(l)} o_i^{(l)}$$

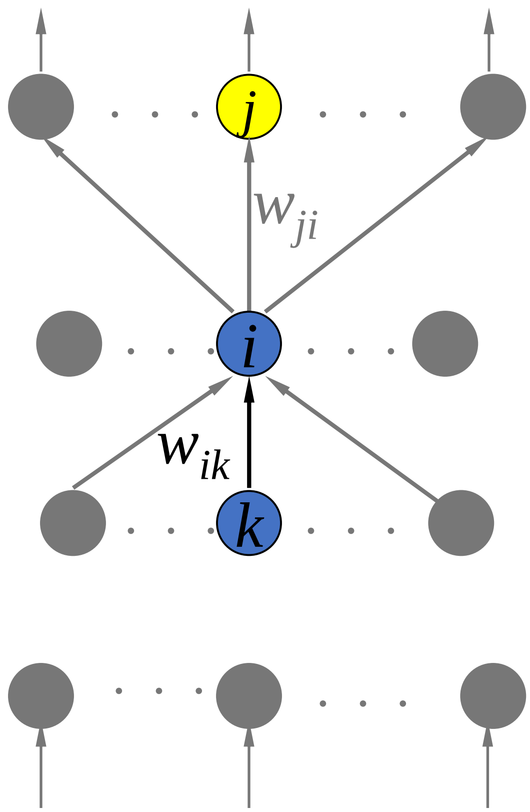
$$\frac{\partial E^{(l)}}{\partial w_{ji}} = \delta_j^{(l)} o_i^{(l)}$$

$$= -(d_j^{(l)} - o_j^{(l)})\lambda o_j^{(l)}(1 - o_j^{(l)}) o_i^{(l)}$$

$$\boxed{E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2}$$

$$\boxed{E = \sum_{l=1}^{p} E^{(l)}}$$

Learning on Hidden Neurons



$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}}\sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p}\frac{\partial E^{(l)}}{\partial w_{ik}}$$

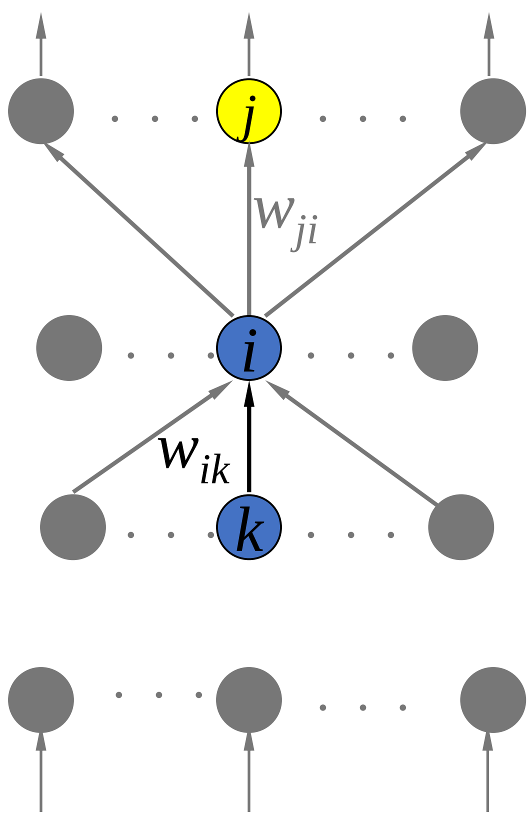
$$\frac{\partial E^{(l)}}{\partial w_{ik}} = \underbrace{\frac{\partial E^{(l)}}{\partial net_i^{(l)}}}_{?}\;\underbrace{\frac{\partial net_i^{(l)}}{\partial w_{ik}}}_{?}$$

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Hidden Neurons

$$\delta_i^{(l)}$$



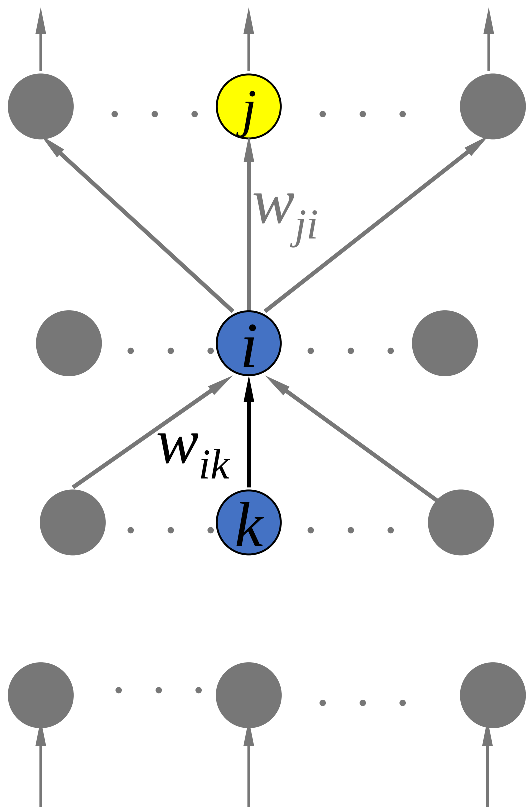
$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}}\sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p}\frac{\partial E^{(l)}}{\partial w_{ik}}$$

$$\frac{\partial E^{(l)}}{\partial w_{ik}} = \frac{\partial E^{(l)}}{\partial net_i^{(l)}} \boxed{\frac{\partial net_i^{(l)}}{\partial w_{ik}}} \rightarrow o_k^{(l)}$$

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Hidden Neurons



$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}}\sum_{l=1}^{p}E^{(l)} = \sum_{l=1}^{p}\frac{\partial E^{(l)}}{\partial w_{ik}}$$

$\delta_i^{(l)}$

$$\frac{\partial E^{(l)}}{\partial w_{ik}} = \boxed{\frac{\partial E^{(l)}}{\partial net_i^{(l)}}}\,\frac{\partial net_i^{(l)}}{\partial w_{ik}} \rightarrow o_k^{(l)}$$

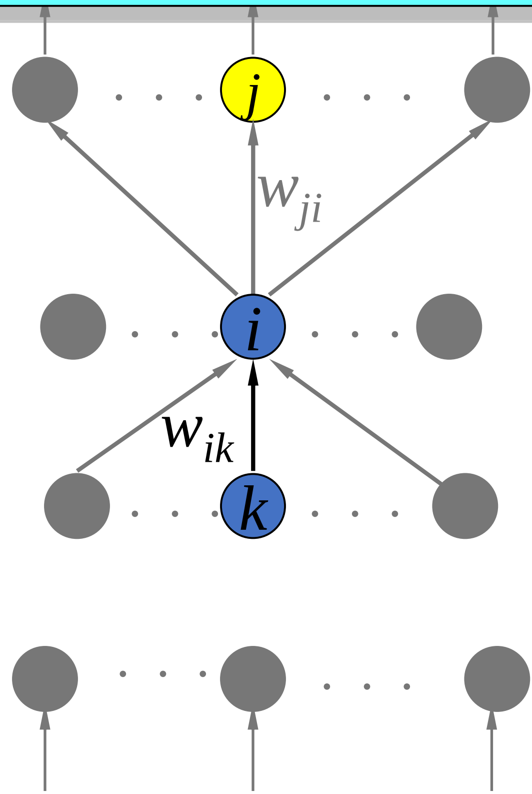
$$\frac{\partial E^{(l)}}{\partial net_i^{(l)}} = \underbrace{\frac{\partial E^{(l)}}{\partial o_i^{(l)}}}_{?}\,\frac{\partial o_i^{(l)}}{\partial net_i^{(l)}} \rightarrow \lambda o_i^{(l)}(1-o_i^{(l)})$$

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Hidden Neurons

$$\delta_i^{(l)} = \frac{\partial E^{(l)}}{\partial net_i^{(l)}} = \lambda o_i^{(l)}(1-o_i^{(l)})\sum_j w_{ji}\delta_j^{(l)}$$



$\delta_i^{(l)}$

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}}\sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p}\frac{\partial E^{(l)}}{\partial w_{ik}}$$

$$\frac{\partial E^{(l)}}{\partial w_{ik}} = \boxed{\frac{\partial E^{(l)}}{\partial net_i^{(l)}}}\frac{\partial net_i^{(l)}}{\partial w_{ik}} \rightarrow o_k^{(l)}$$

$$\frac{\partial E^{(l)}}{\partial net_i^{(l)}} = \boxed{\frac{\partial E^{(l)}}{\partial o_i^{(l)}}}\frac{\partial o_i^{(l)}}{\partial net_i^{(l)}} \rightarrow \lambda o_i^{(l)}(1-o_i^{(l)})$$

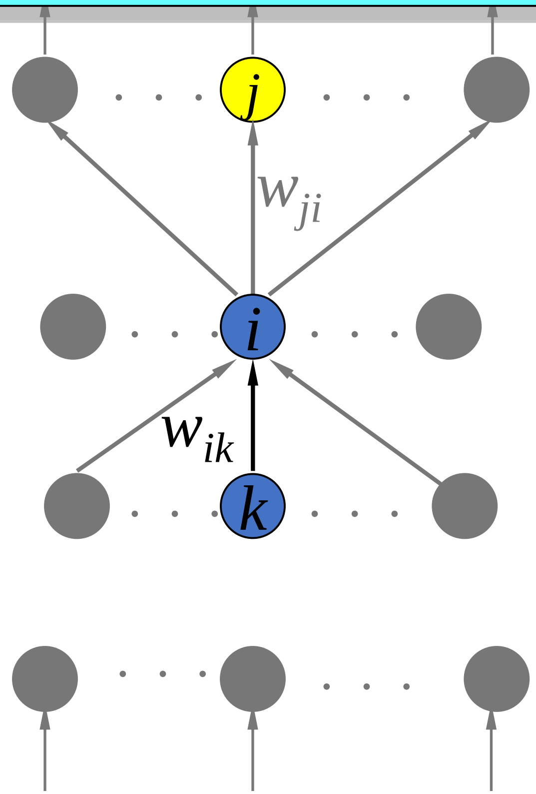
$$\frac{\partial E^{(l)}}{\partial o_i^{(l)}} = \sum_j \underbrace{\frac{\partial E^{(l)}}{\partial net_j^{(l)}}}_{\delta_j^{(l)}}\underbrace{\frac{\partial net_j^{(l)}}{\partial o_i^{(l)}}}_{w_{ji}}$$

$$E^{(l)} = \frac{1}{2}\sum_{j=1}^{n}\left[d_j^{(l)} - o_j^{(l)}\right]^2$$

$$E = \sum_{l=1}^{p} E^{(l)}$$

Learning on Hidden Neurons

$$\delta_i^{(l)} = \frac{\partial E^{(l)}}{\partial net_i^{(l)}} = \lambda o_i^{(l)}(1 - o_i^{(l)})\sum_j w_{ji}\delta_j^{(l)}$$



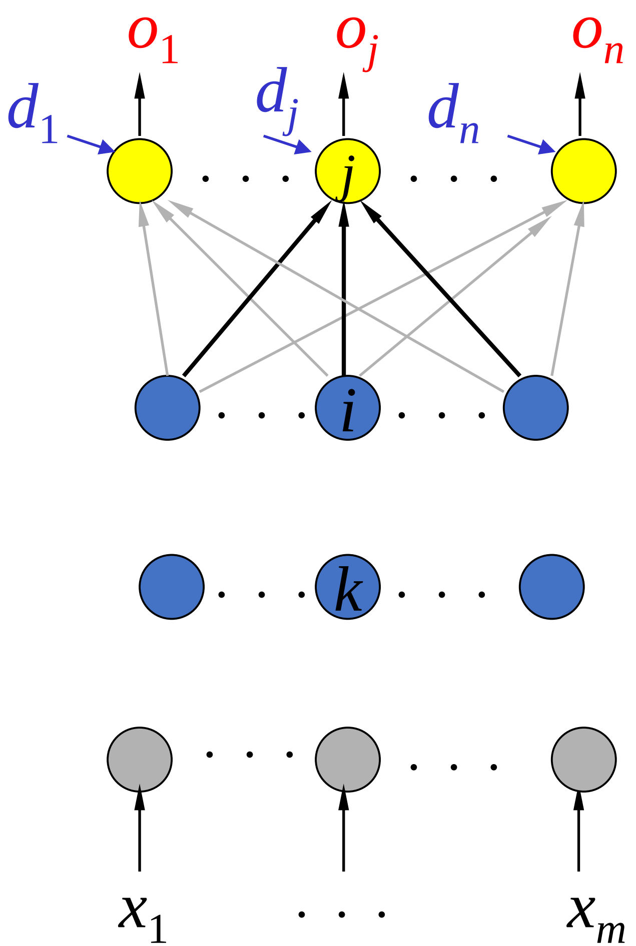
$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}}\sum_{l=1}^{p} E^{(l)} = \sum_{l=1}^{p}\frac{\partial E^{(l)}}{\partial w_{ik}}$$

$$\frac{\partial E^{(l)}}{\partial w_{ik}} = \boxed{\frac{\partial E^{(l)}}{\partial net_i^{(l)}}}\frac{\partial net_i^{(l)}}{\partial w_{ik}} \longrightarrow o_k^{(l)}$$

$$\frac{\partial E}{\partial w_{ik}} = \sum_{l=1}^{p}\delta_i^{(l)} o_k^{(l)}$$

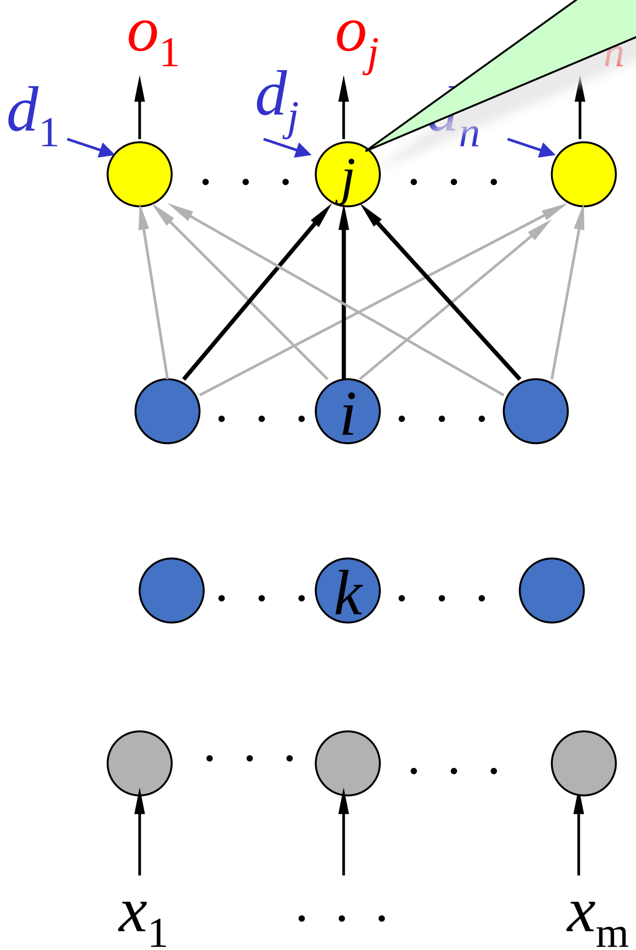
$$\Delta w_{ik} = -\eta\sum_{l=1}^{p}\delta_i^{(l)} o_k^{(l)}$$

Back Propagation



Back Propagation

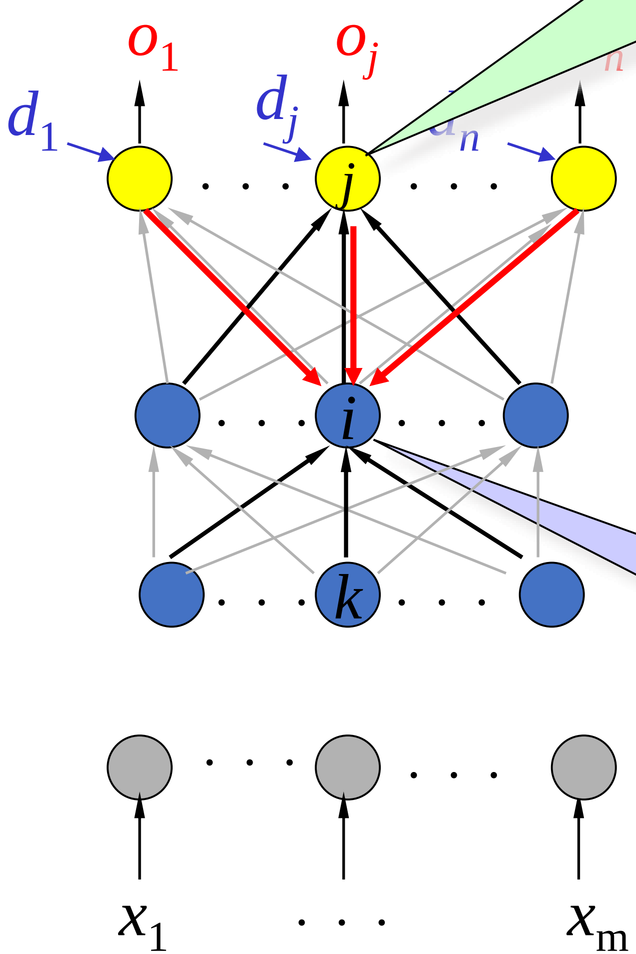
$$\delta_j^{(l)} = \frac{\partial E^{(l)}}{\partial net_j^{(l)}} = -\lambda(d_j^{(l)} - o_j^{(l)})o_j^{(l)}(1 - o_j^{(l)})$$



$$\Delta w_{ji} = -\eta \sum_{l=1}^{p} \delta_j^{(l)} o_i^{(l)}$$

Back Propagation

$$\delta_j^{(l)} = \frac{\partial E^{(l)}}{\partial net_j^{(l)}} = -\lambda(d_j^{(l)} - o_j^{(l)})o_j^{(l)}(1-o_j^{(l)})$$



$$\Delta w_{ji} = -\eta \sum_{l=1}^{p} \delta_j^{(l)} o_i^{(l)}$$

$$\Delta w_{ik} = -\eta \sum_{l=1}^{p} \delta_i^{(l)} o_k^{(l)}$$

$$\delta_i^{(l)} = \frac{\partial E^{(l)}}{\partial net_i^{(l)}} = \lambda o_i^{(l)}(1-o_i^{(l)}) \sum_j w_{ji} \delta_j^{(l)}$$

Backpropagation Training Algorithm

Create the K-layer network with *H* hidden units. Set weights to small random real values.

Until all training examples produce correct value (within ε), or mean squared error ceases to decrease, or other termination criteria:

 Begin epoch

 For each training example, *d*, do:

  Calculate network output for *d*'s input values

  Compute error between output and label for *d*

  Update weights using learning rule

 End epoch

Reading

- Shi Zhong and Vladimir Cherkassky, "Factors Controlling Generalization Ability of MLP Networks." In Proc. IEEE Int. Joint Conf. on Neural Networks, vol. 1, pp. 625-630, Washington DC. July 1999. (http://www.cse.fau.edu/~zhong/pubs.htm)

- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). "Learning Internal Representations by Error Propagation," in Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. I, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. MIT Press, Cambridge (1986).

  (http://www.cnbc.cmu.edu/~plaut/85-419/papers/RumelhartETAL86.backprop.pdf).