M L
D M

Machine Learning

Chapter 6

Neural Networks

Contents

Introduction

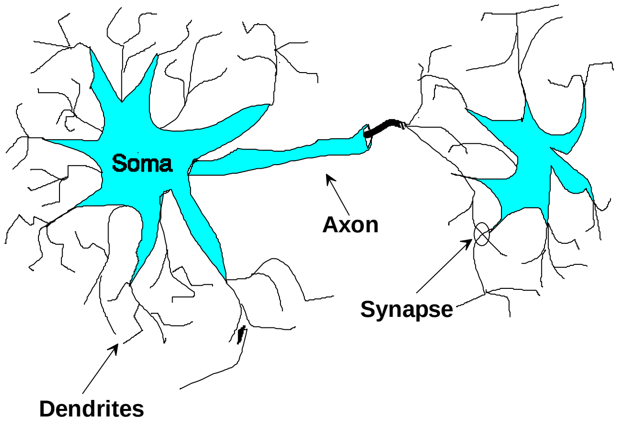
- (Artificial) Neural Networks are
  - Computational models which mimic the brain's learning processes.
  - They have the essential features of neurons and their interconnections as found in the brain.
  - Typically, a computer is programmed to simulate these features.
- Other definitions …
  - A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:
    - Knowledge is acquired by the network from its environment through a learning process
    - Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Introduction

- A neural network is a machine learning approach inspired by the way in which the brain performs a particular learning task:
  - Knowledge about the learning task is given in the form of examples.
  - Inter neuron connection strengths (weights) are used to store the acquired information (the training examples).
  - During the learning process the weights are modified in order to model the particular learning task correctly on the training examples.

Biological Neural Systems

- The brain is composed of approximately 100 billion (10^{11}) neurons



Schematic drawing of two biological neurons connected by synapses

A typical neuron collects signals from other neurons through a host of fine structures called dendrites.

The neuron sends out spikes of electrical activity through a long, thin strand known as an axon, which splits into thousands of branches.

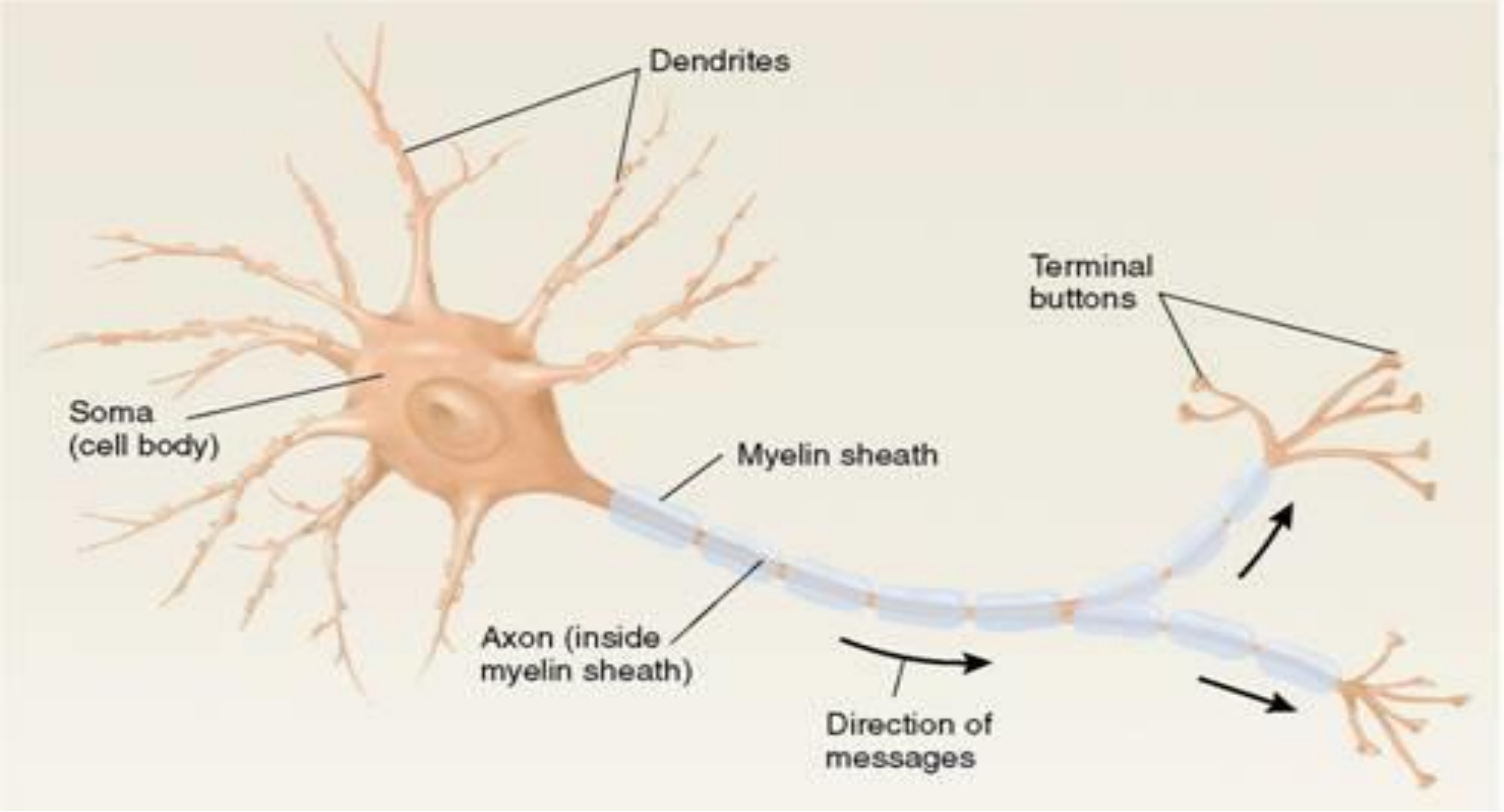
At the end of the branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons.

When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon.

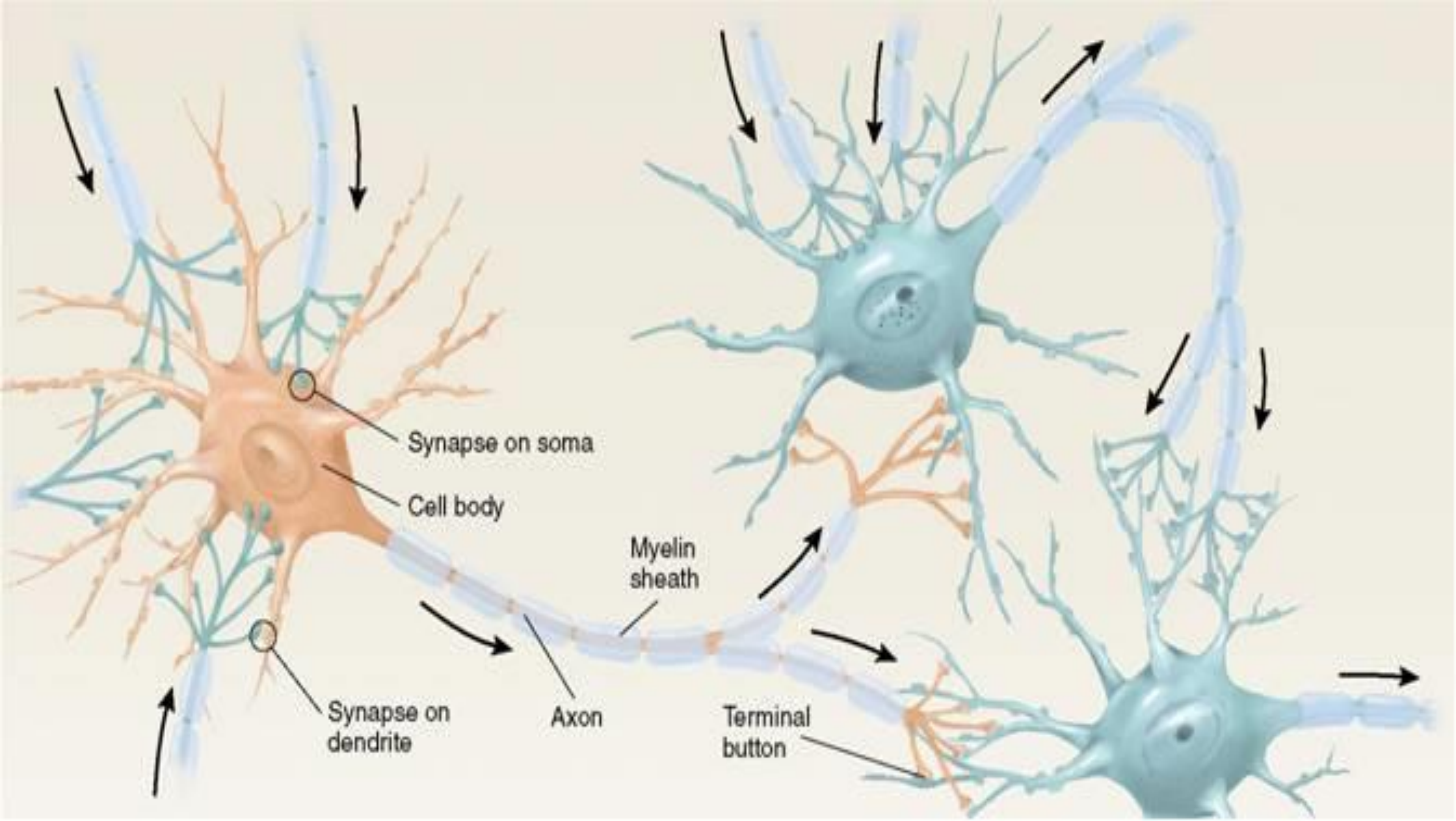
Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on the other changes

Neuron

MIMA


Dendrites
Terminal buttons
Soma (cell body)
Myelin sheath
Axon (inside myelin sheath)
Direction of messages

Interconnections between Neurons



Neural Networks

- A NN is a machine learning approach inspired by the way in which the brain performs a particular learning task
- Various types of neurons
- Various network architectures
- Various learning algorithms
- Various applications

Characteristics of NN’s

- Characteristics of Neural Networks
  - Large scale and parallel processing
  - Robust
  - Self-adaptive and organizing
  - Good enough to simulate non-linear relations
  - Hardware

Historical Background

- 1943 McCulloch and Pitts proposed the first computational models of neuron.
- 1949 Hebb proposed the first learning rule.
- 1958 Rosenblatt’s work in perceptrons.
- 1969 Minsky and Papert’s exposed limitation of the theory.
- 1970s Decade of dormancy for neural networks.
- 1980-90s Neural network return (self-organization, back-propagation algorithms, etc)

Applications

- Combinatorial Optimization
- Pattern Recognition
- Bioinformatics
- Text processing
- Natural language processing
- Data Mining
- …

Types

- Structure
  - Feed-forward
  - Feed-back
- Learning method
  - Supervised
  - Unsupervised
- Signal type
  - Continuous
  - Discrete

The Neuron

- The neuron is the basic information processing unit of a NN. It consists of:
  1. A set of synapses or connecting links, each link characterized by a weight:

     $$W_1, W_2, \ldots, W_m$$

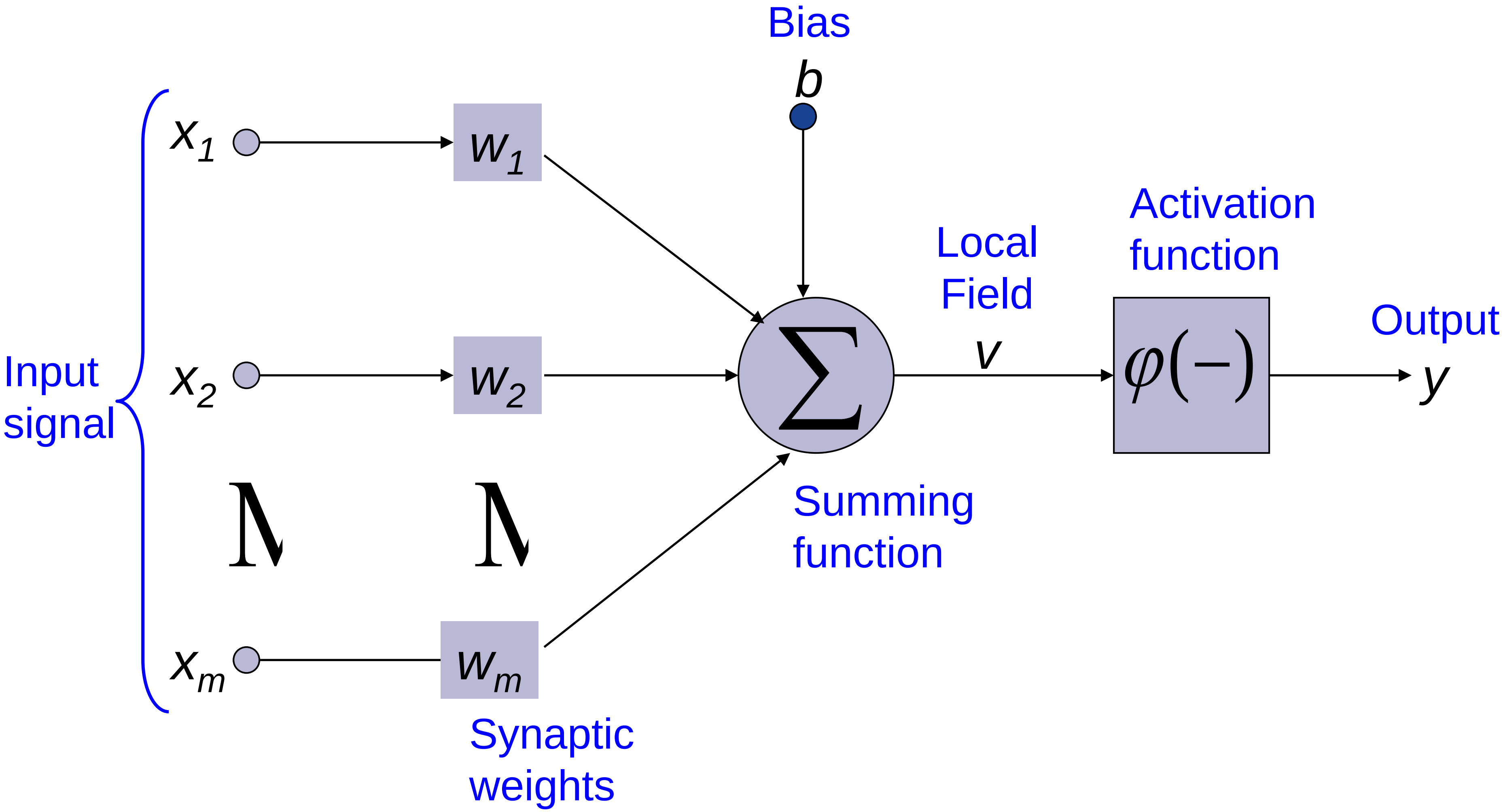
  2. An adder function (linear combiner) which computes the weighted sum of the inputs:

     $$u = \sum_{j=1}^{m} w_j x_j$$

  3. Activation function (squashing function) for limiting the amplitude of the output of the neuron.

     $$y = \varphi(u + b)$$

The Neuron



Bias

b

Input signal

Local Field

v

Activation function

Output

y

Summing function

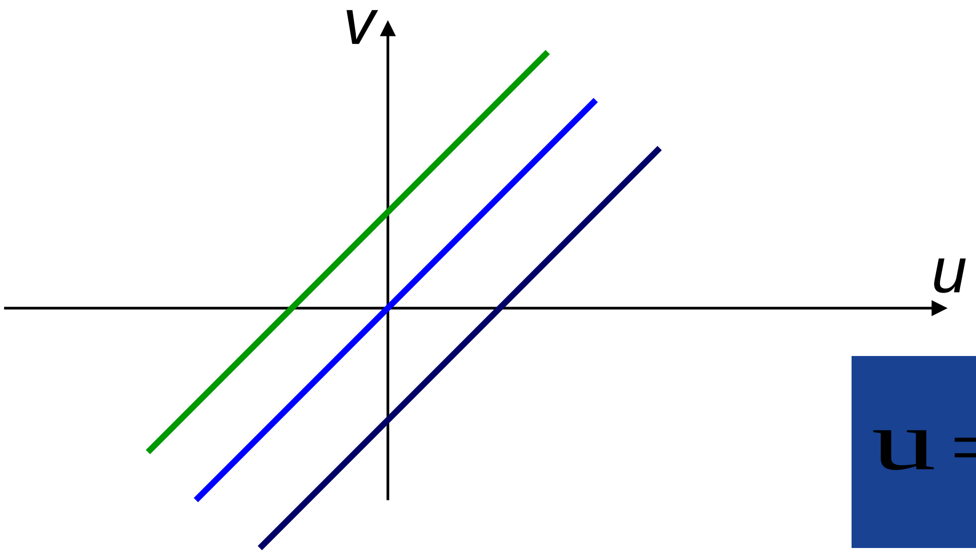
Synaptic weights

Bias of a Neuron

- Bias ***b*** has the effect of applying an affine transformation to ***u***

$$v = u + b$$

- ***v*** is the **induced field** of the neuron



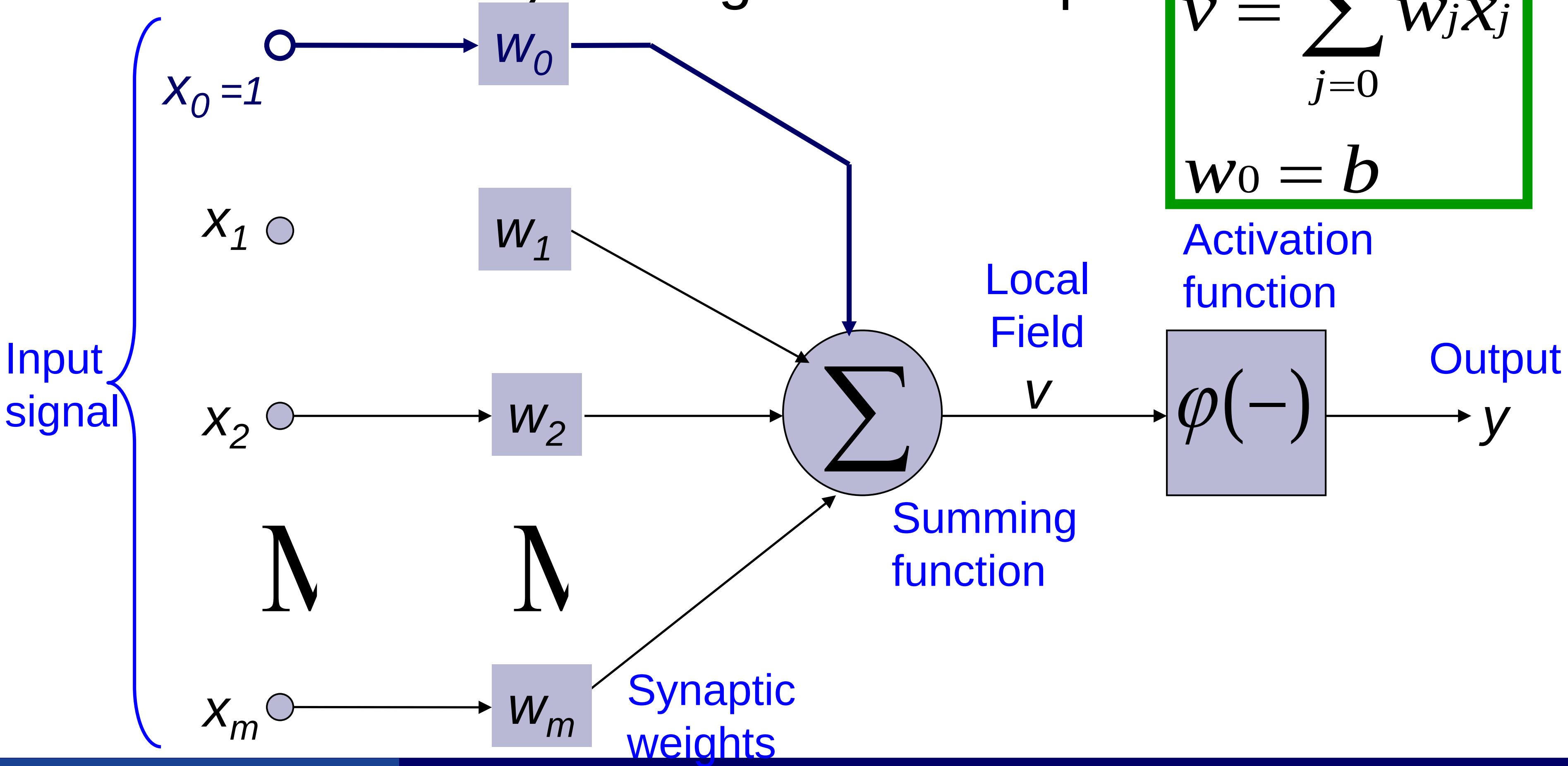
$$u = \sum_{j=1}^{m} w_j x_j$$

Bias as Extra Input

- Bias is an external parameter of the neuron. Can be modeled by adding an extra input.

$$v = \sum_{j=0}^{m} w_j x_j$$

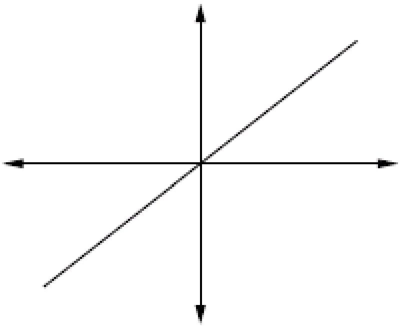
$$w_0 = b$$



Activation Function

- 1.Linear function

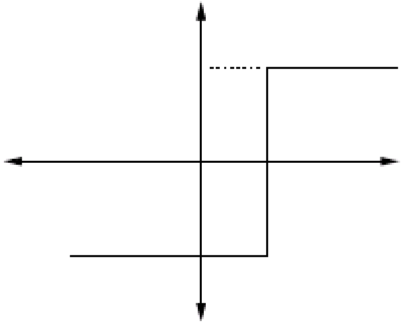
$$f(x) = ax$$



Linear function

- 2. Step function

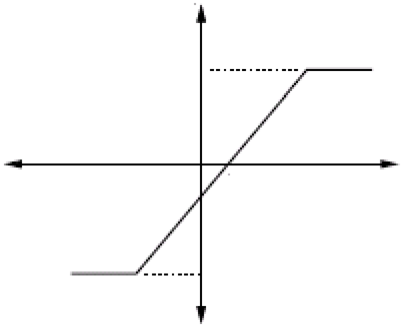
$$f(x) = \begin{cases} a_1 & \text{if } x \geq \theta \\ a_2 & \text{if } x < \theta \end{cases}$$



Step function

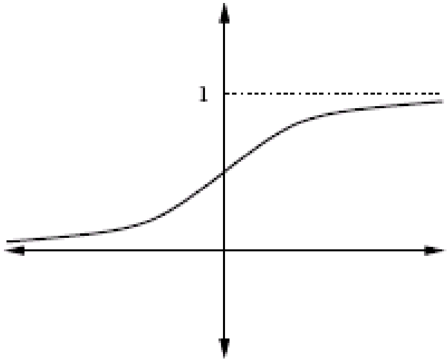
- 3. Ramp function

$$f(x) = \begin{cases} \alpha & \text{if } x \geq \theta \\ x & \text{if } -\theta < x < \theta \\ -\alpha & \text{if } x \leq \theta \end{cases}$$



Ramp function

Activation Function

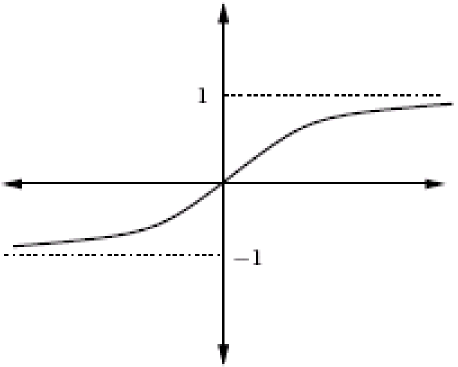


Sigmoid function

- 4. Logistic function

$$f(x)=\frac{1}{1+e^{-\lambda x}}$$

- 5. Hyperbolic tangent

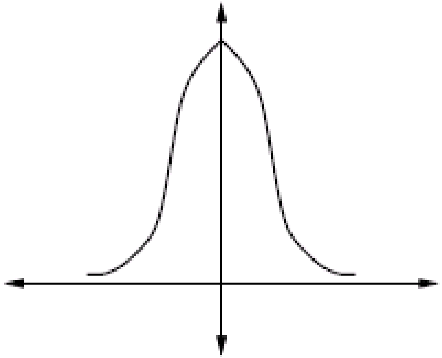


Hyperbolic tangent function

$$f(x)=\frac{e^{\lambda x}-e^{-\lambda x}}{e^{\lambda x}+e^{-\lambda x}}$$

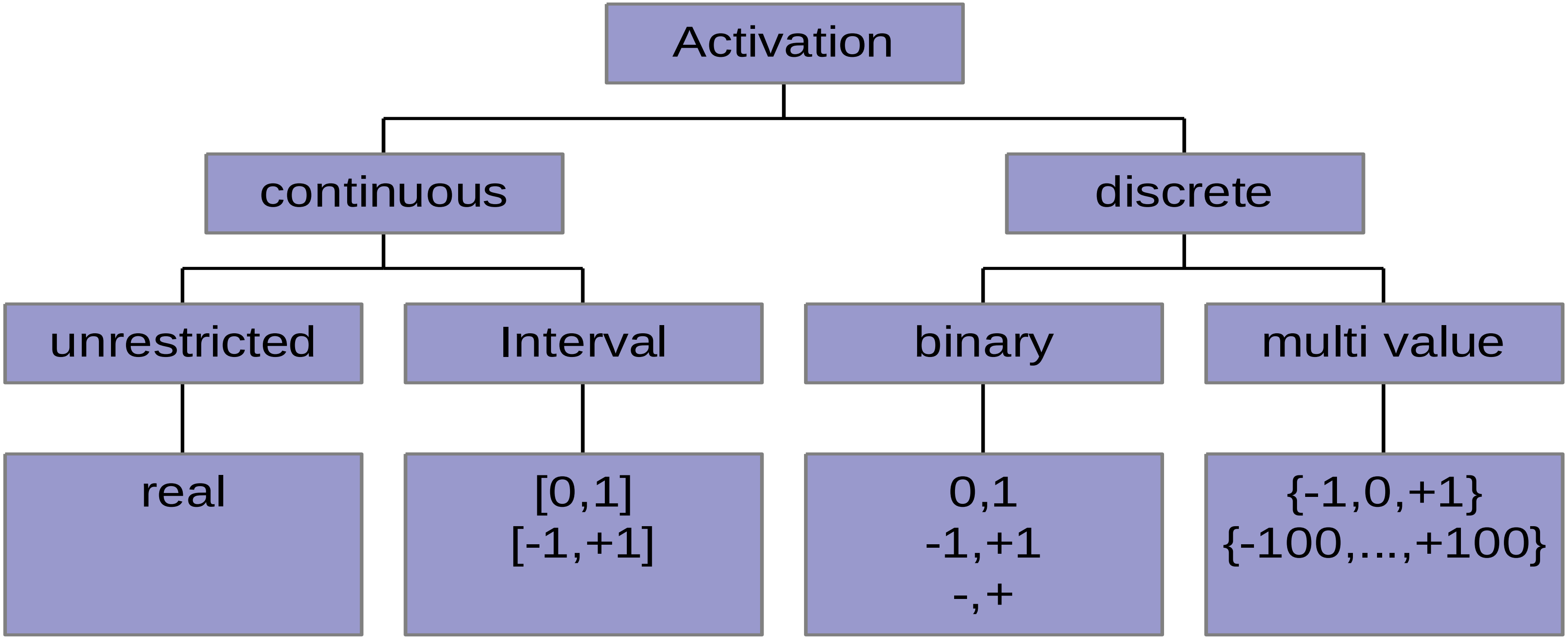
- 6. Gaussian function

$$f(x)=e^{-x^2/\sigma^2}$$



Gaussian function

Activation function


Activation

continuous

discrete

unrestricted

Interval

binary

multi value

real

[0,1]
[-1,+1]

0,1
-1,+1
-,+

{-1,0,+1}
{-100,...,+100}

Perceptron

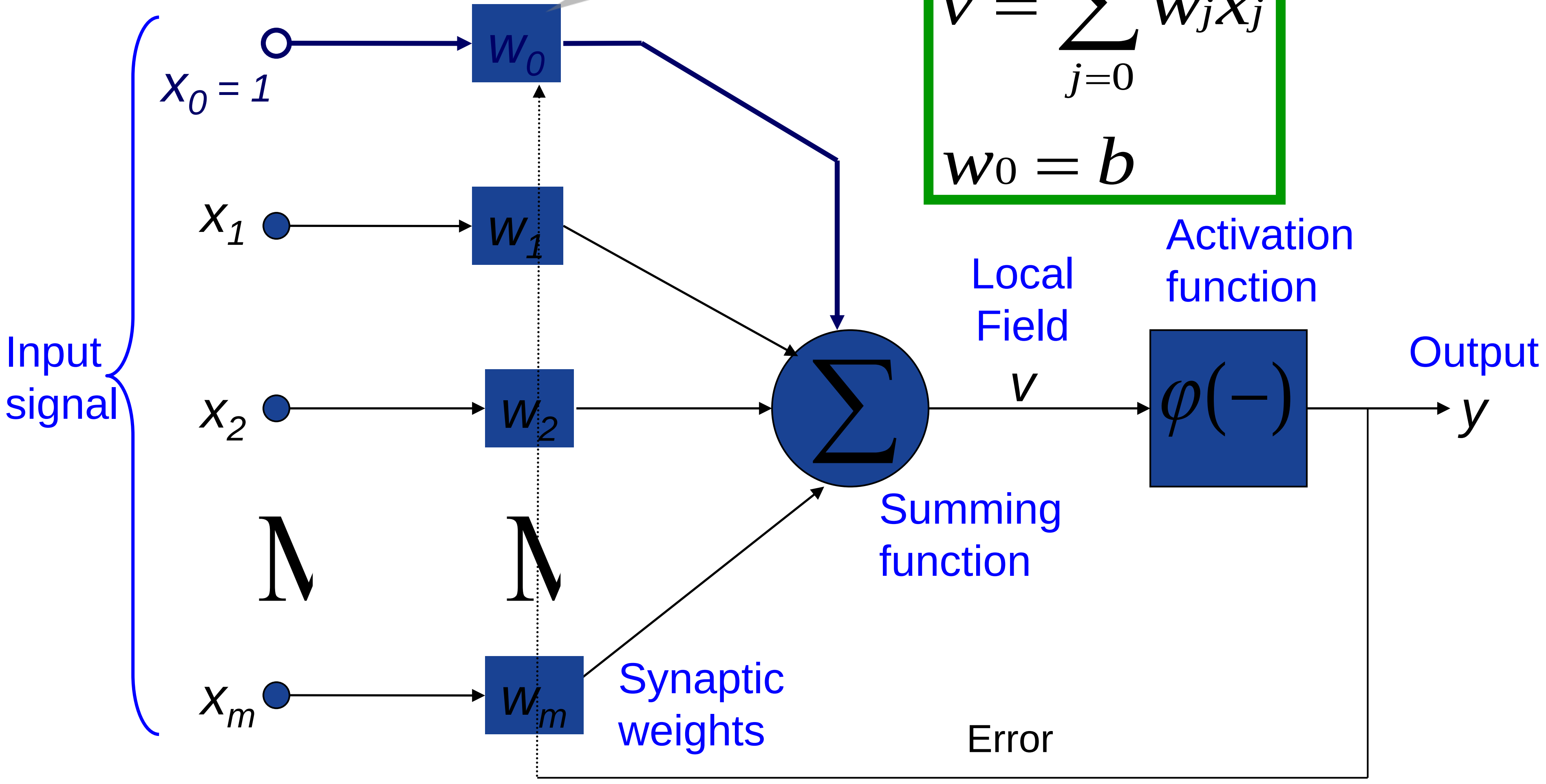
- In 1943, McCulloch and Pitts proposed the first single neuron model.
- Hebb proposed the theory that the learning process is generated from the change of weights between synapses.
- Rosenblatt combined them together, and proposed “Perceptron”.
- Perceptron is just a single neural model, and is composed of synaptic weights and threshold.
- It is the simplest and earliest neural network model, used for classification.

Perceptron

A plane passes through the origin in the augmented input space.

$$v = \sum_{j=0}^{m} w_j x_j$$

$$w_0 = b$$



Perceptron

MIMA

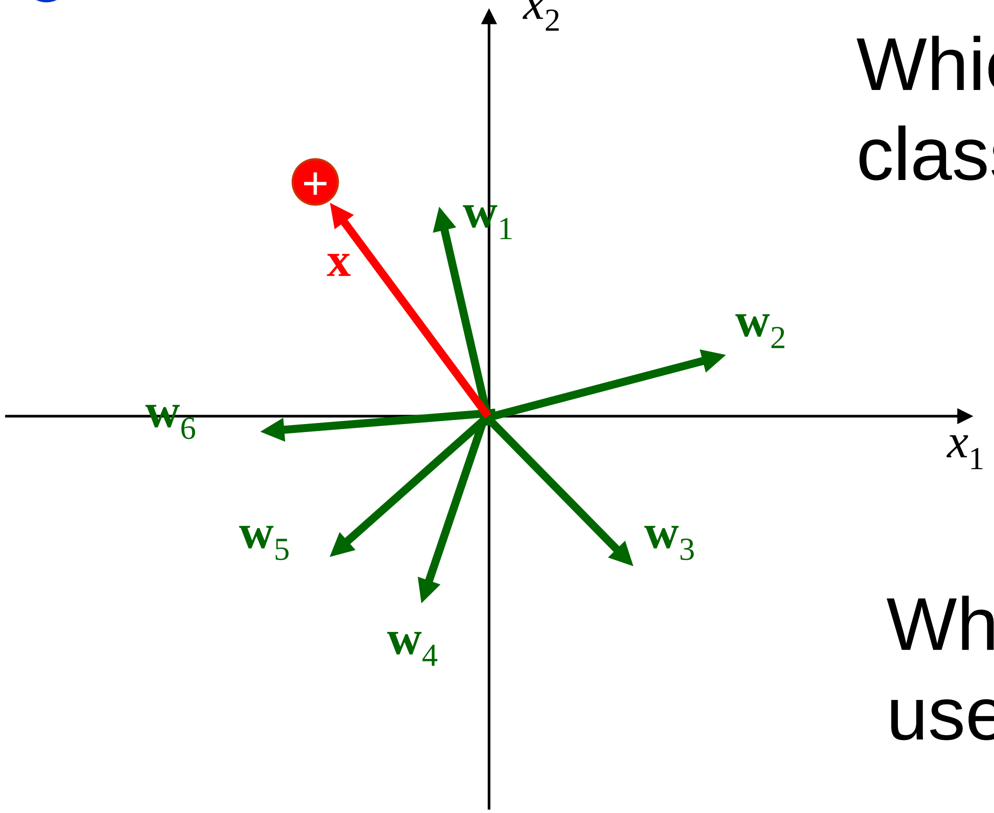
- Given training sets $T_1 \in C_1$ and $T_2 \in C_2$ with elements in form of $\mathbf{x}=(x_0, x_1, x_2, \ldots, x_m)^T$, where $x_1, x_2, \ldots, x_m \in R$ and $x_0 = 1$.
- Assume T_1 and T_2 are linearly separable.
- Find $\mathbf{w}=(w_0, w_1, w_2, \ldots, w_m)^T$ such that

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

Perceptron

⊕ $d = +1$

⊖ $d = -1$



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

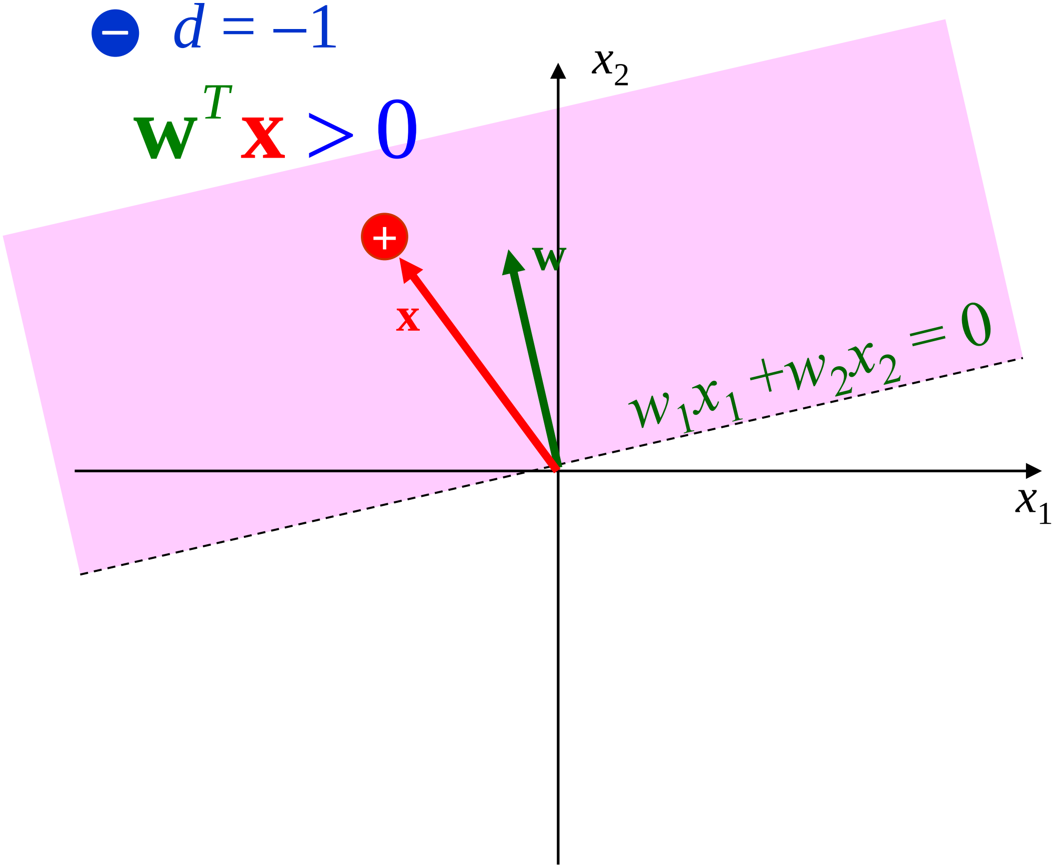
What trick can be used?

Perceptron

⊕ $d = +1$

⊖ $d = -1$

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



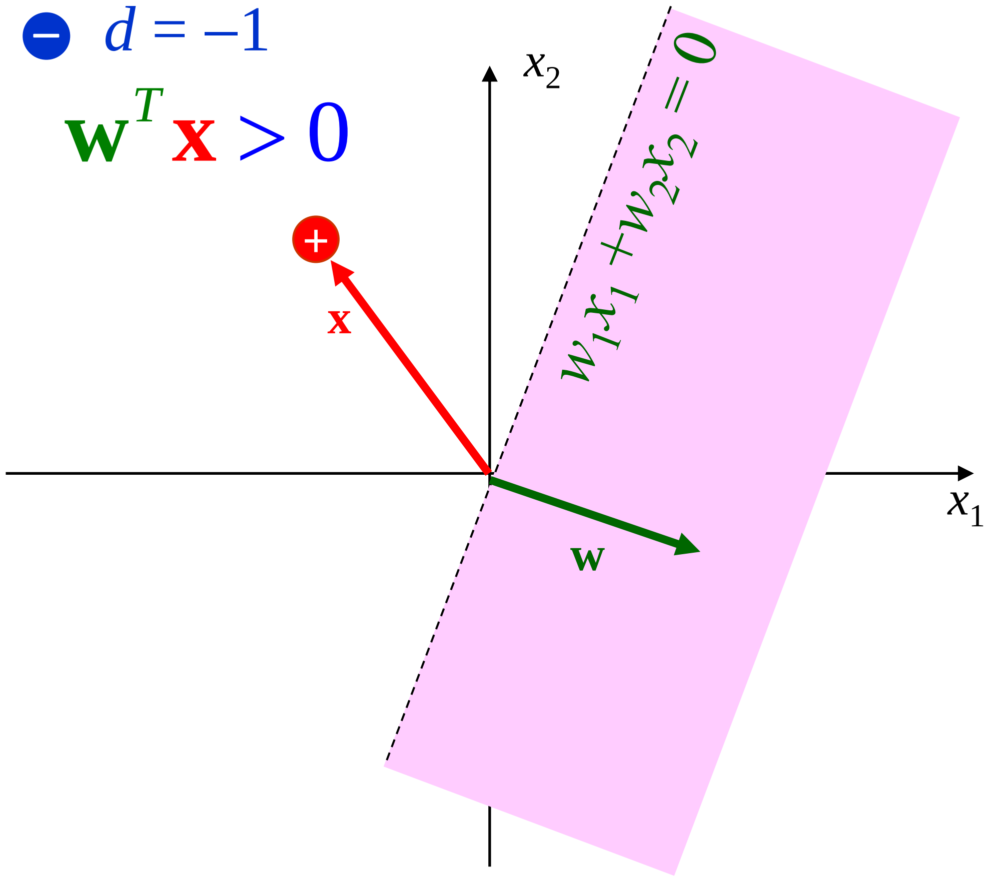
Is this $\mathbf{w}$ ok?

Perceptron

⊕ $d = +1$

⊖ $d = -1$

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



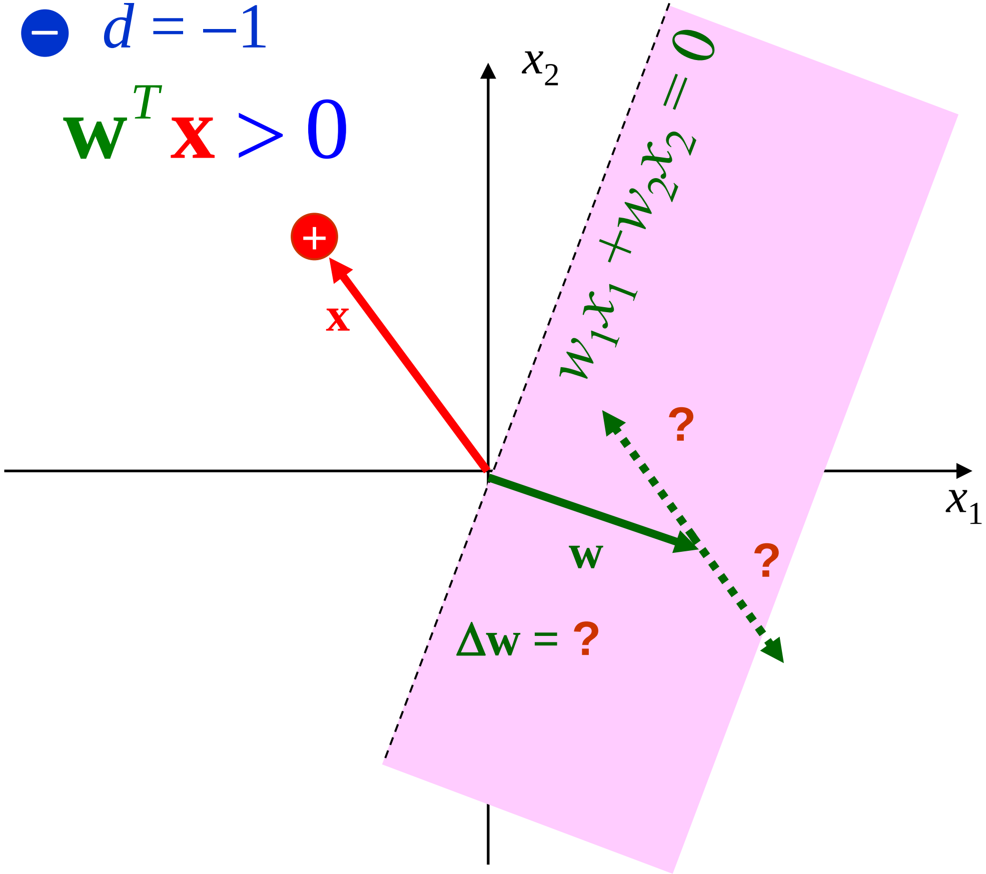
Is this $\mathbf{w}$ ok?

Perceptron

⊕ $d = +1$

⊖ $d = -1$

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



Is this $\mathbf{w}$ ok?

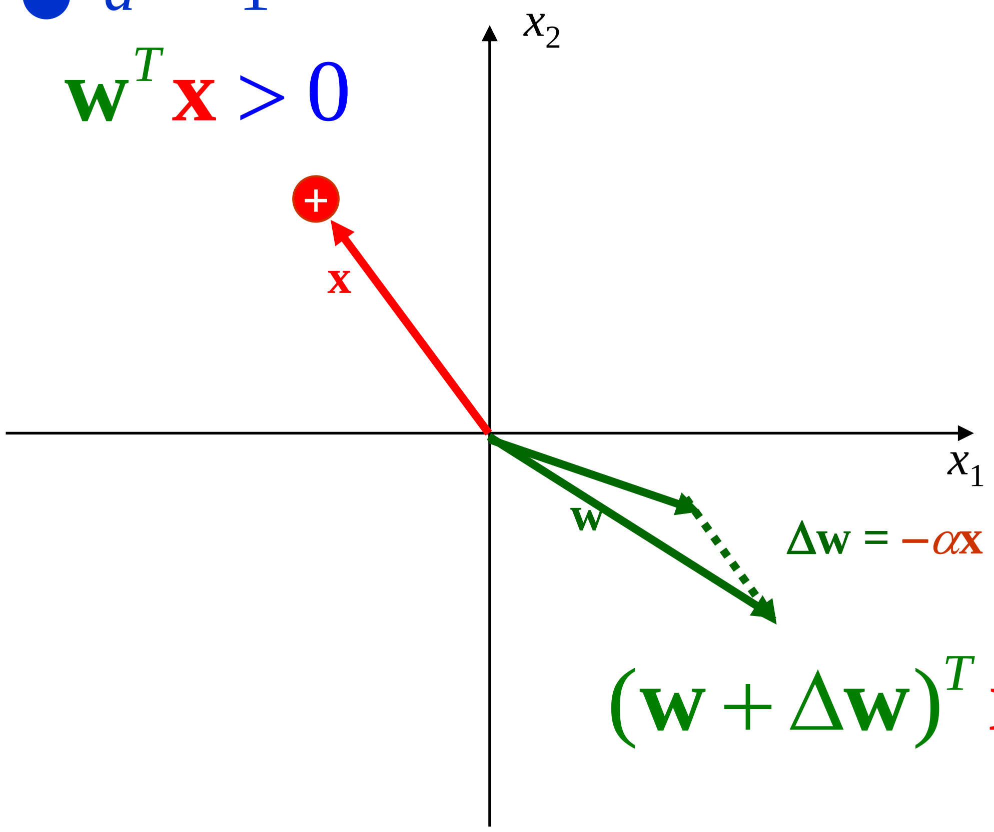
How to adjust $\mathbf{w}$?

Perceptron

⊕ $d = +1$

⊖ $d = -1$

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



Is this $\mathbf{w}$ ok?

How to adjust $\mathbf{w}$?

reasonable?

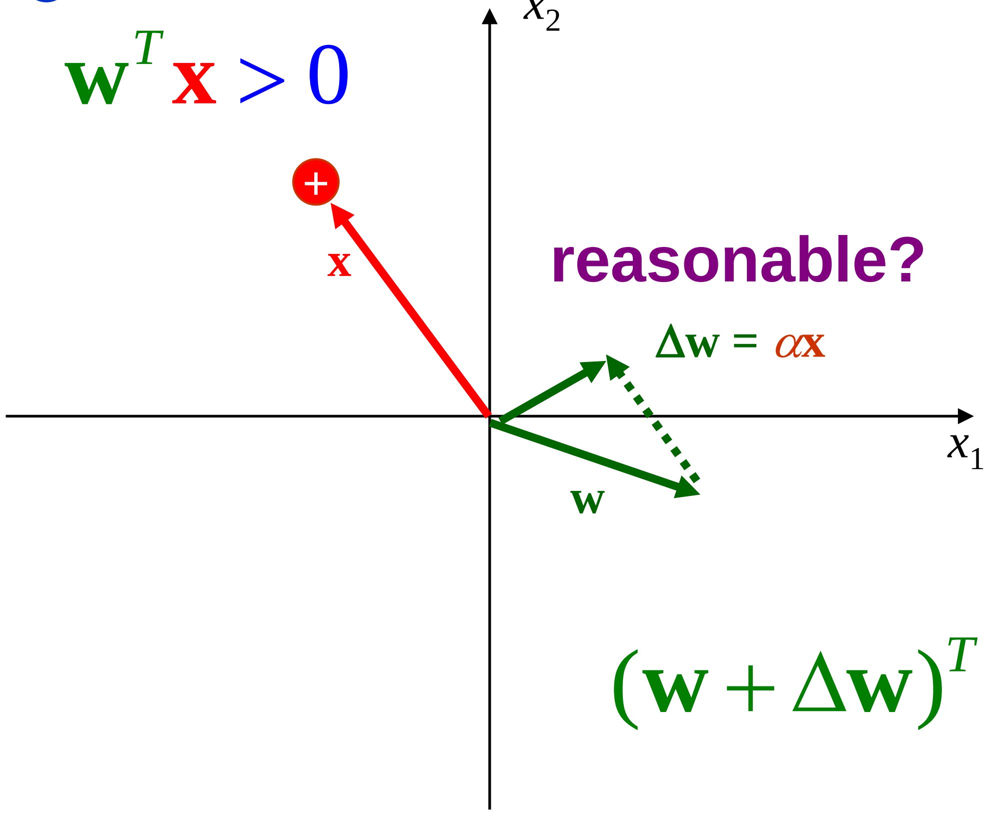
$$(\mathbf{w}+\Delta\mathbf{w})^T\mathbf{x} = \underbrace{\mathbf{w}^T\mathbf{x}}_{<0} - \underbrace{\alpha\mathbf{x}^T\mathbf{x}}_{>0}$$

Perceptron

⊕ $d = +1$

⊖ $d = -1$

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



reasonable?

Is this $\mathbf{w}$ ok?

How to adjust $\mathbf{w}$?

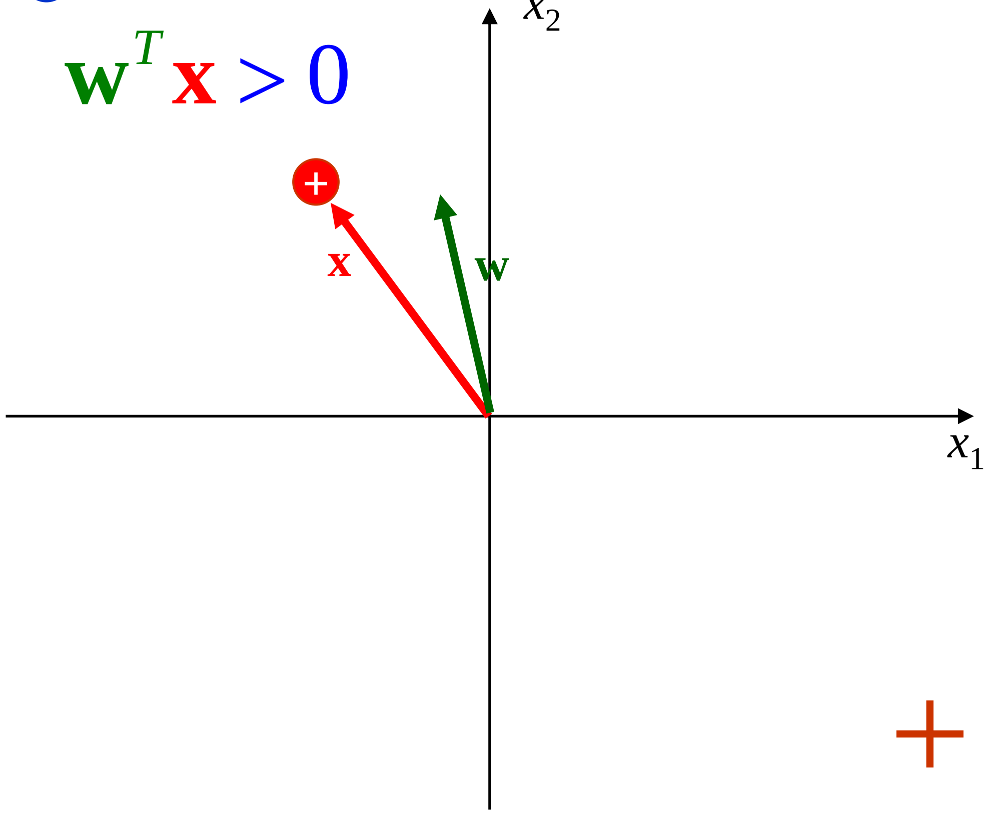
$$(\mathbf{w} + \Delta\mathbf{w})^T \mathbf{x} = \underbrace{\mathbf{w}^T \mathbf{x}}_{<0} + \underbrace{\alpha \mathbf{x}^T \mathbf{x}}_{>0}$$

Perceptron

⊕ $d = +1$

⊖ $d = -1$

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



Is this $\mathbf{w}$ ok?

$$\Delta\mathbf{w} = ?$$

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

Learning Rule

- **Upon misclassification on** $\alpha > 0$

⊕ $d = +1$ $\Delta\mathbf{w} = \alpha\mathbf{x}$

⊖ $d = -1$ $\Delta\mathbf{w} = -\alpha\mathbf{x}$

Define error

$$r = d - y = \begin{cases} +2 & \\ -2 & \\ 0 & \end{cases}$$

⊕ → ⊖

⊖ → ⊕

No error

Learning Rule

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

Define error

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

Learning Rule

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

Learning Rate

Error $(d - y)$

Input

Learning Rule

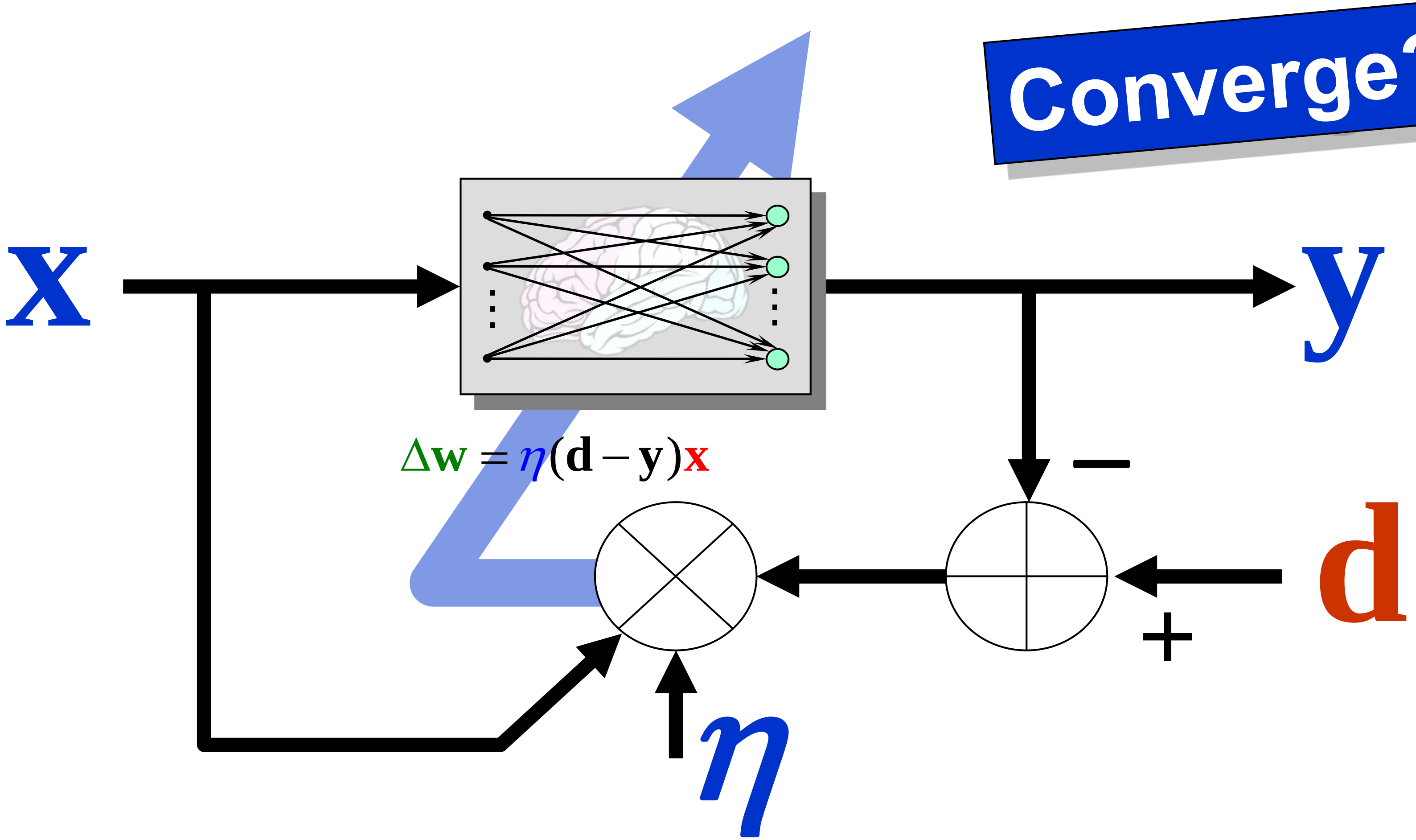
$$\Delta w_i(t) = \eta r_i x_i(t)$$

$$r_i = d_i - y_i = \left\{\begin{array}{l} \begin{array}{ll} 0 & d_i = y_i \qquad \text{correct} \end{array} \\ \left.\begin{array}{ll} +2 & d_i = 1, y_i = -1 \\ -2 & d_i = -1, y_i = 1 \end{array}\right\}\text{incorrect} \end{array}\right.$$

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

Learning Rule

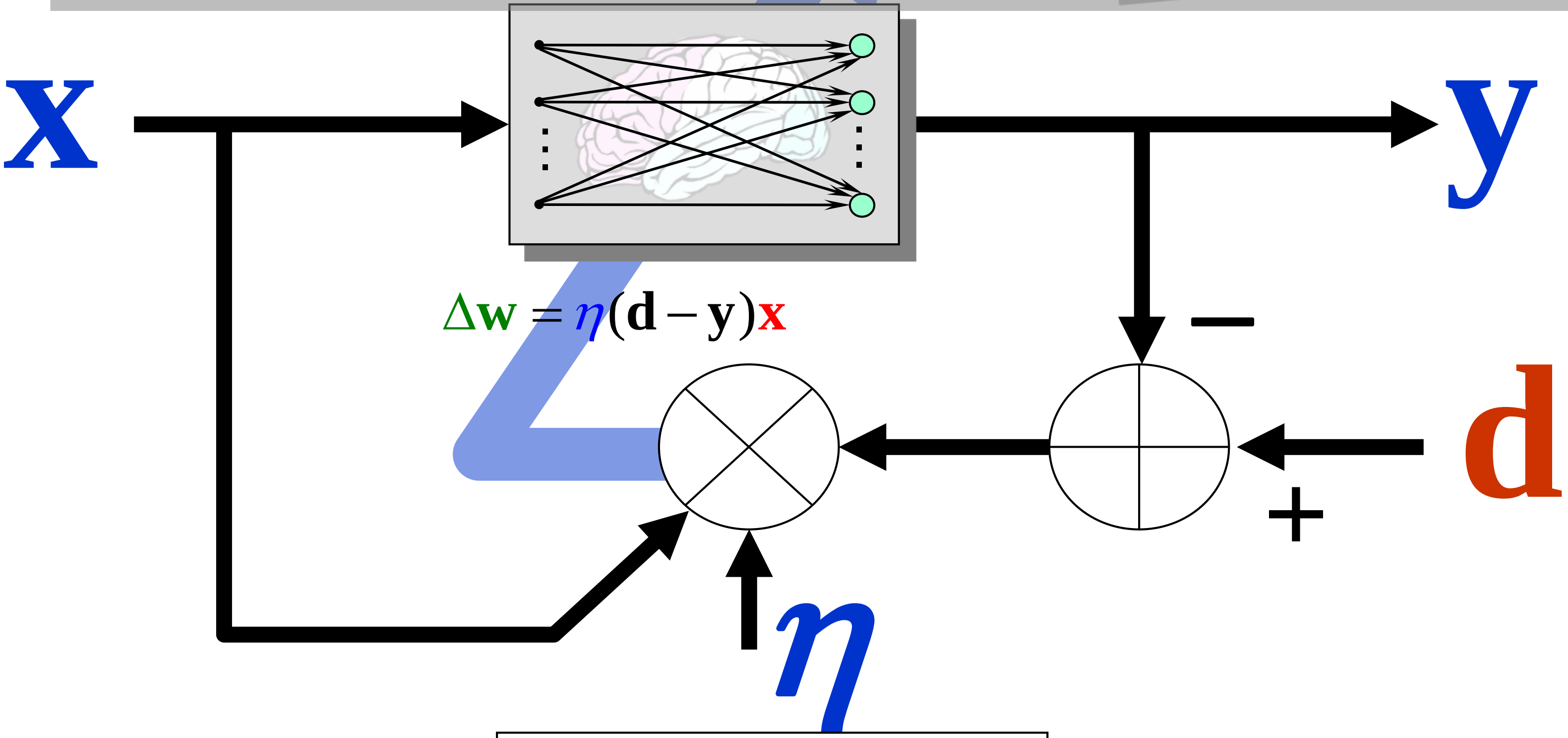
Converge?



$\Delta \mathbf{w} = \eta(\mathbf{d} - \mathbf{y})\mathbf{x}$

Learning Rule

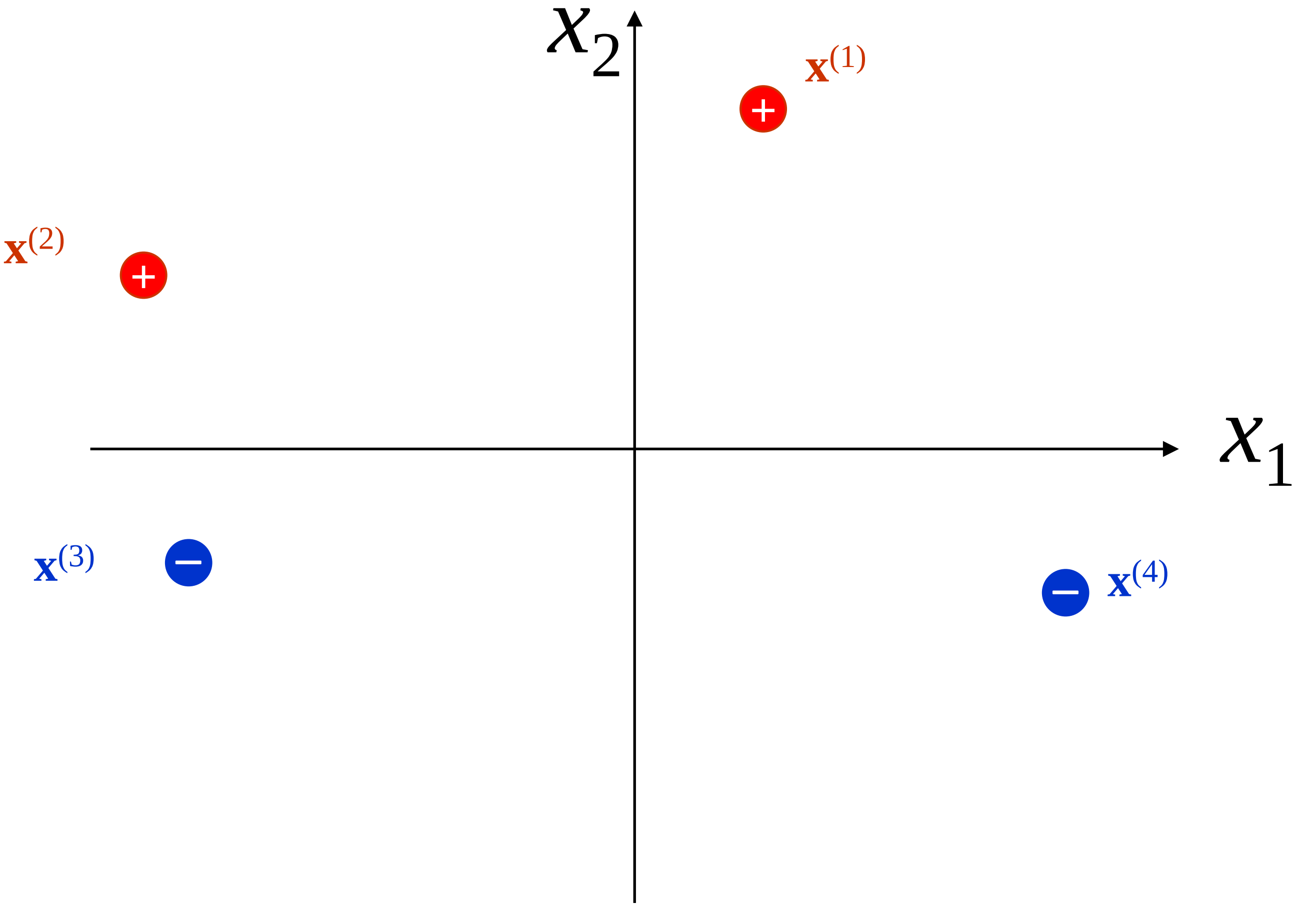
If the given training set is linearly separable, the learning process will converge in a finite number of steps.



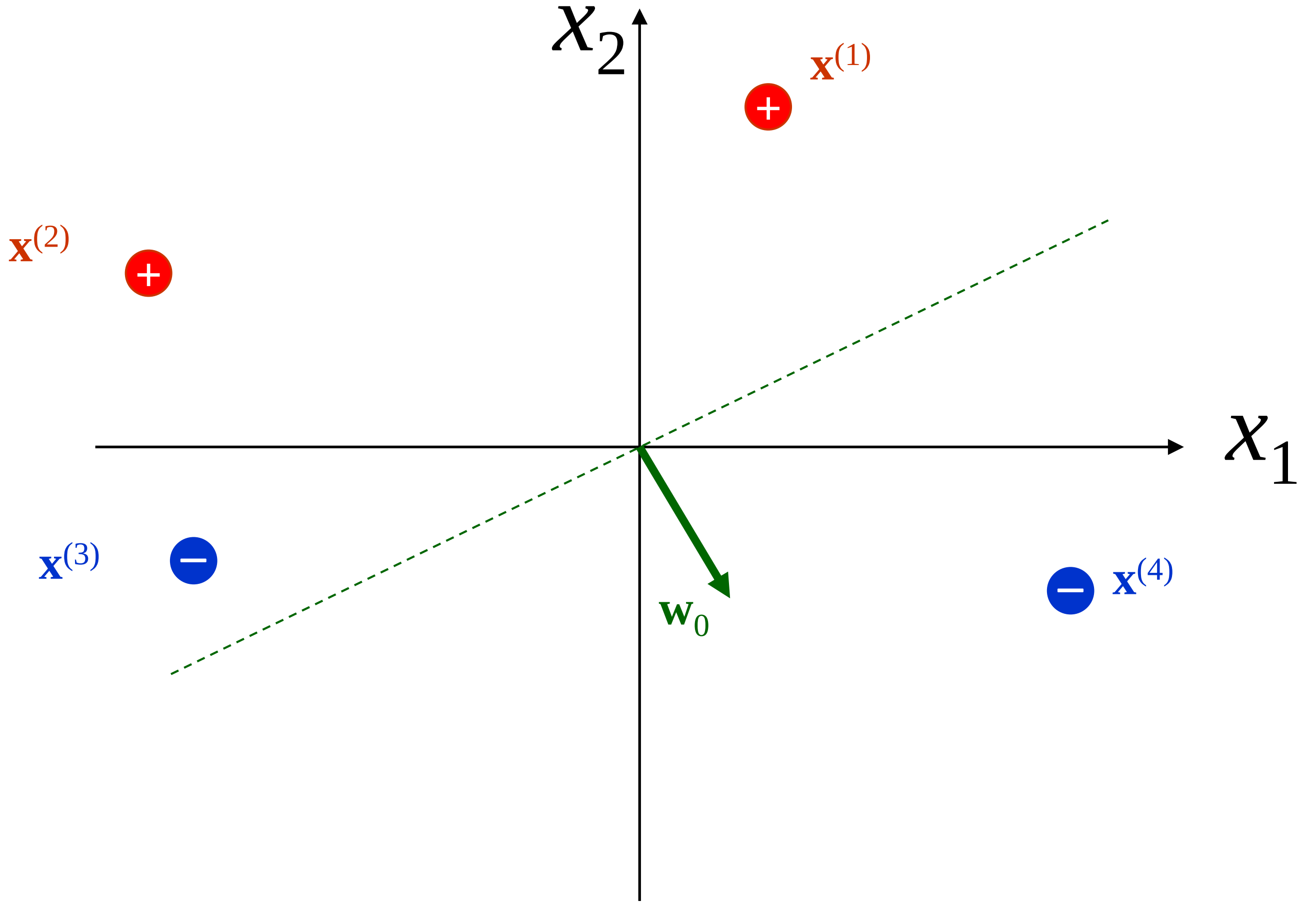
$$\Delta \mathbf{w} = \eta(\mathbf{d} - \mathbf{y})\mathbf{x}$$

How to prove?

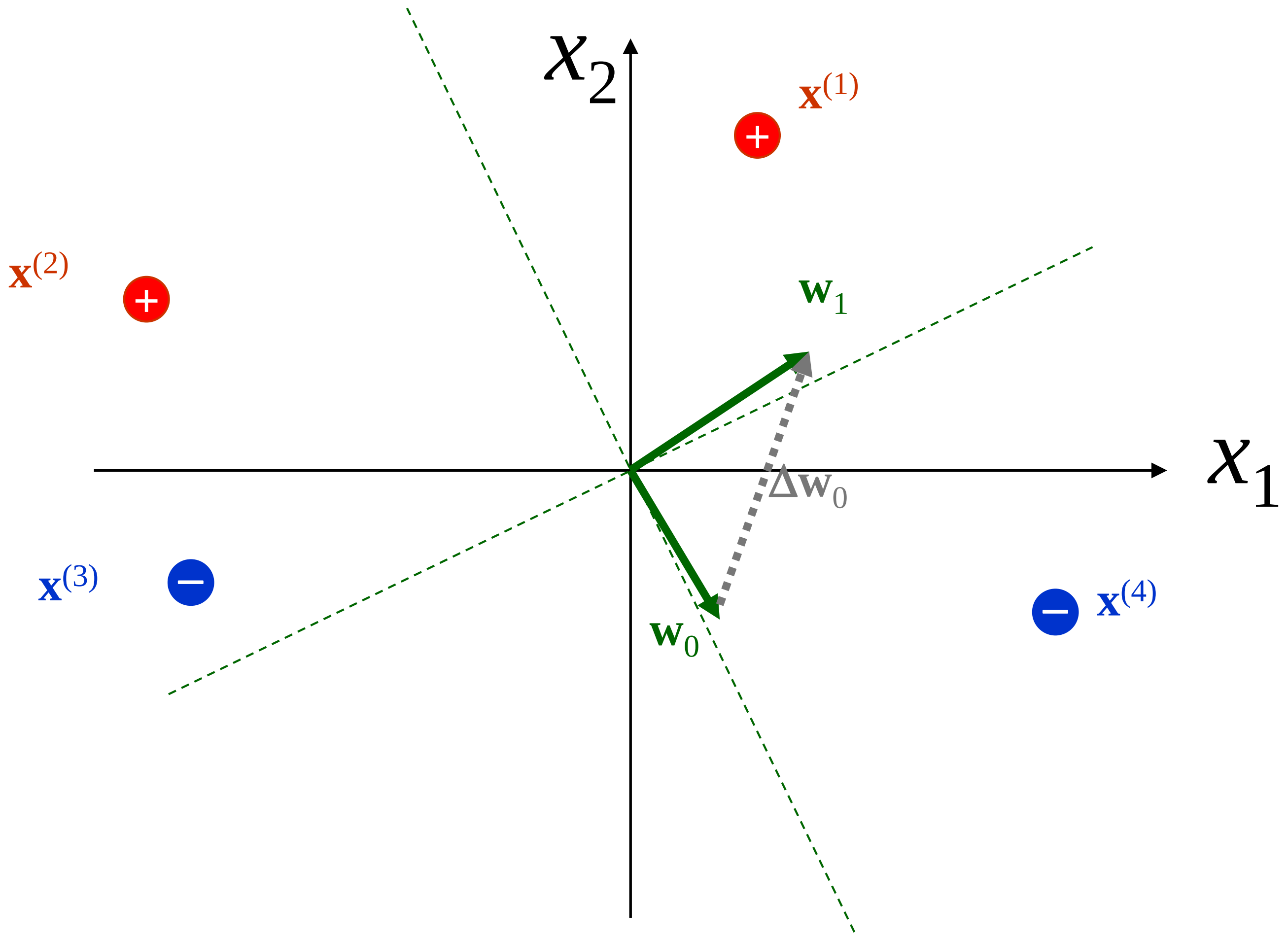
The Learning Scenario



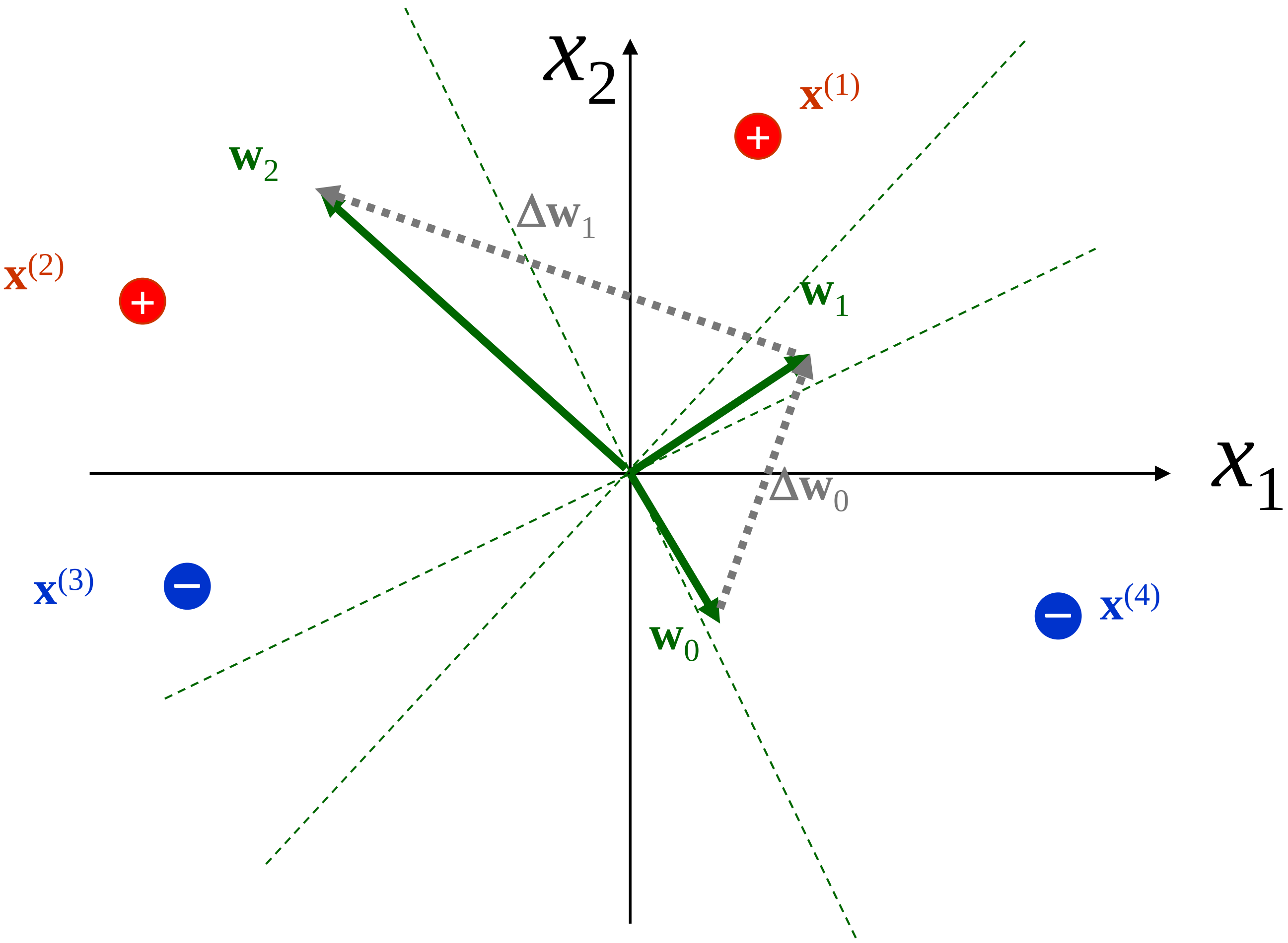
The Learning Scenario



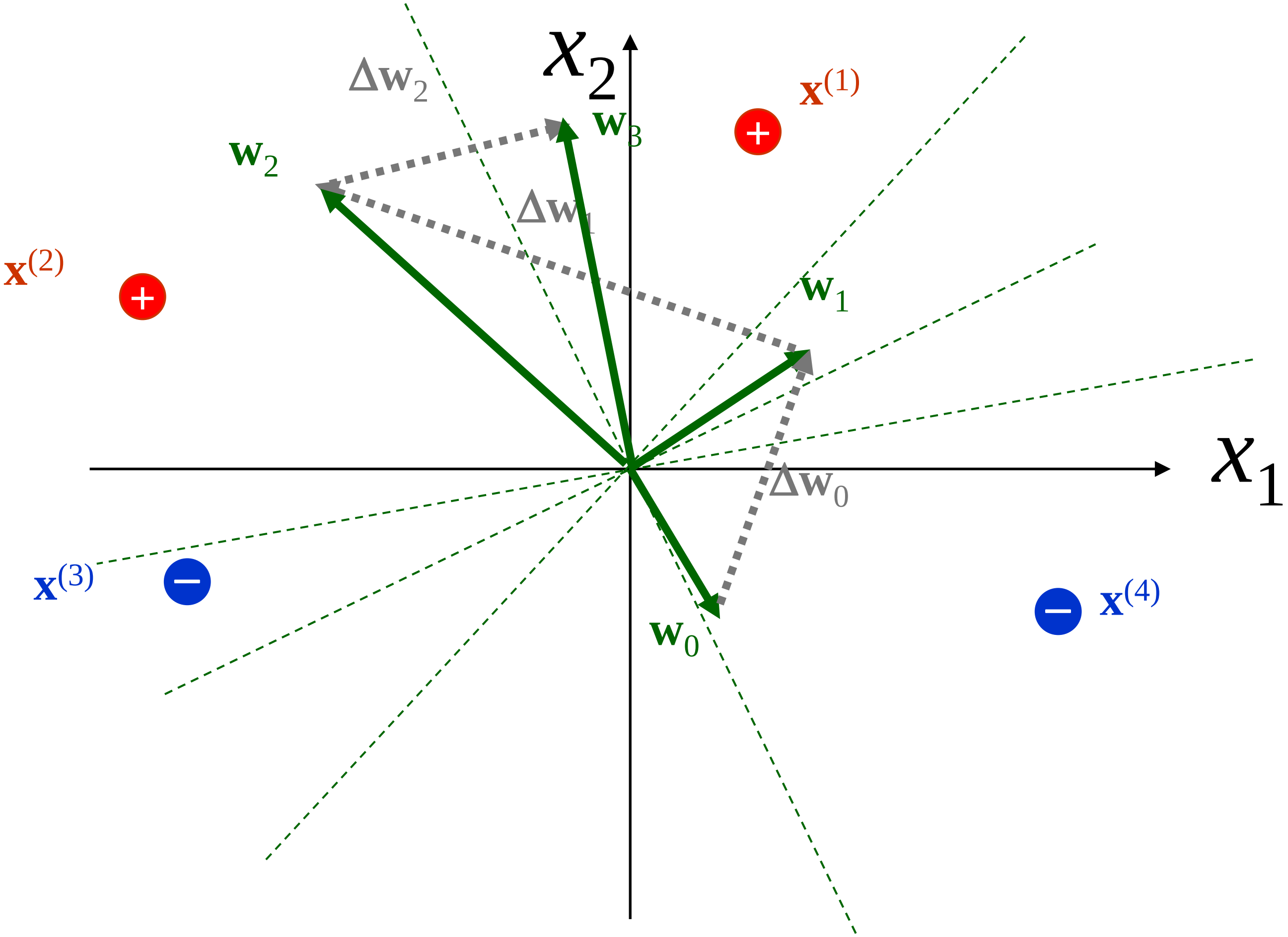
The Learning Scenario



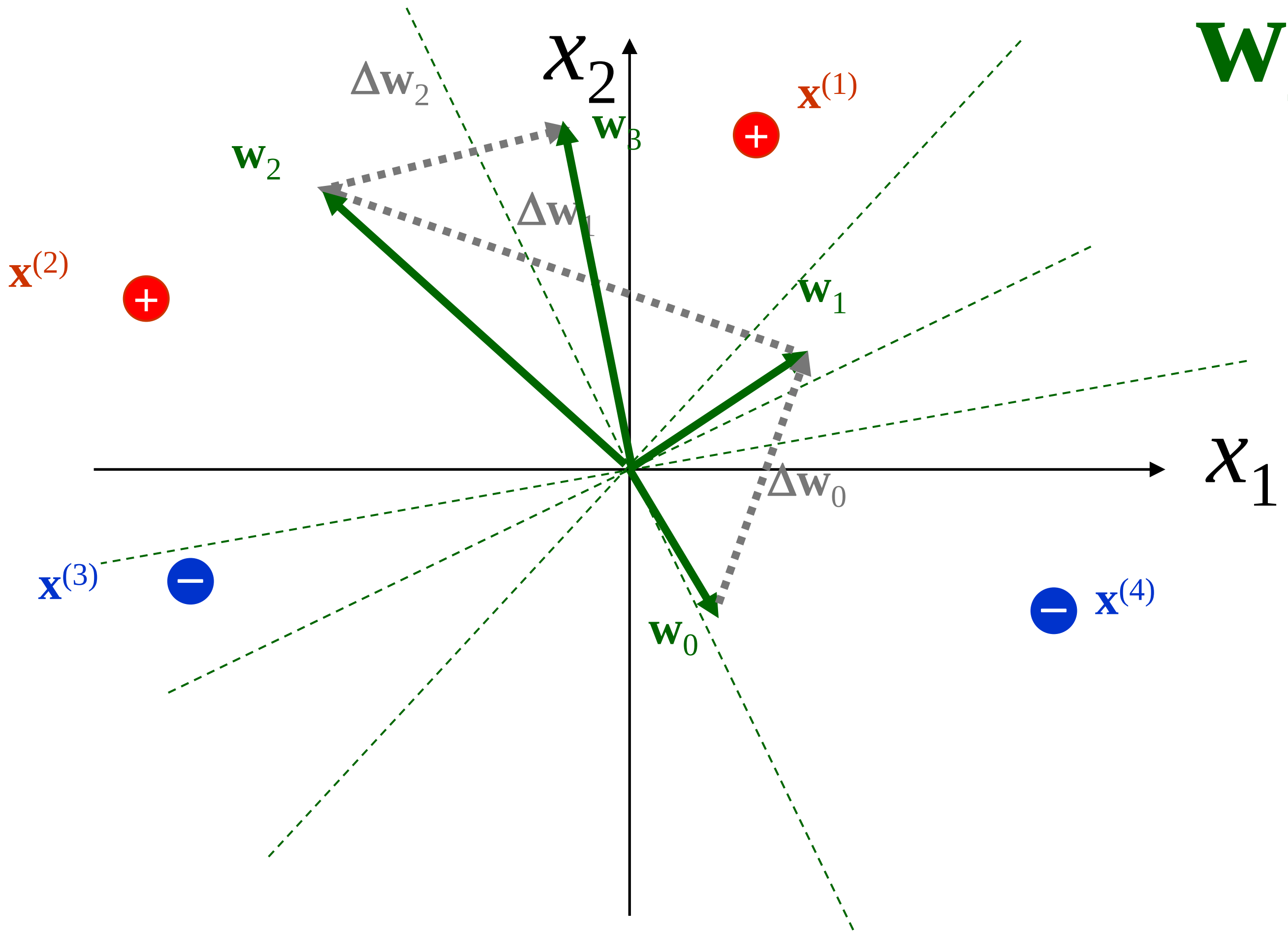
The Learning Scenario



The Learning Scenario



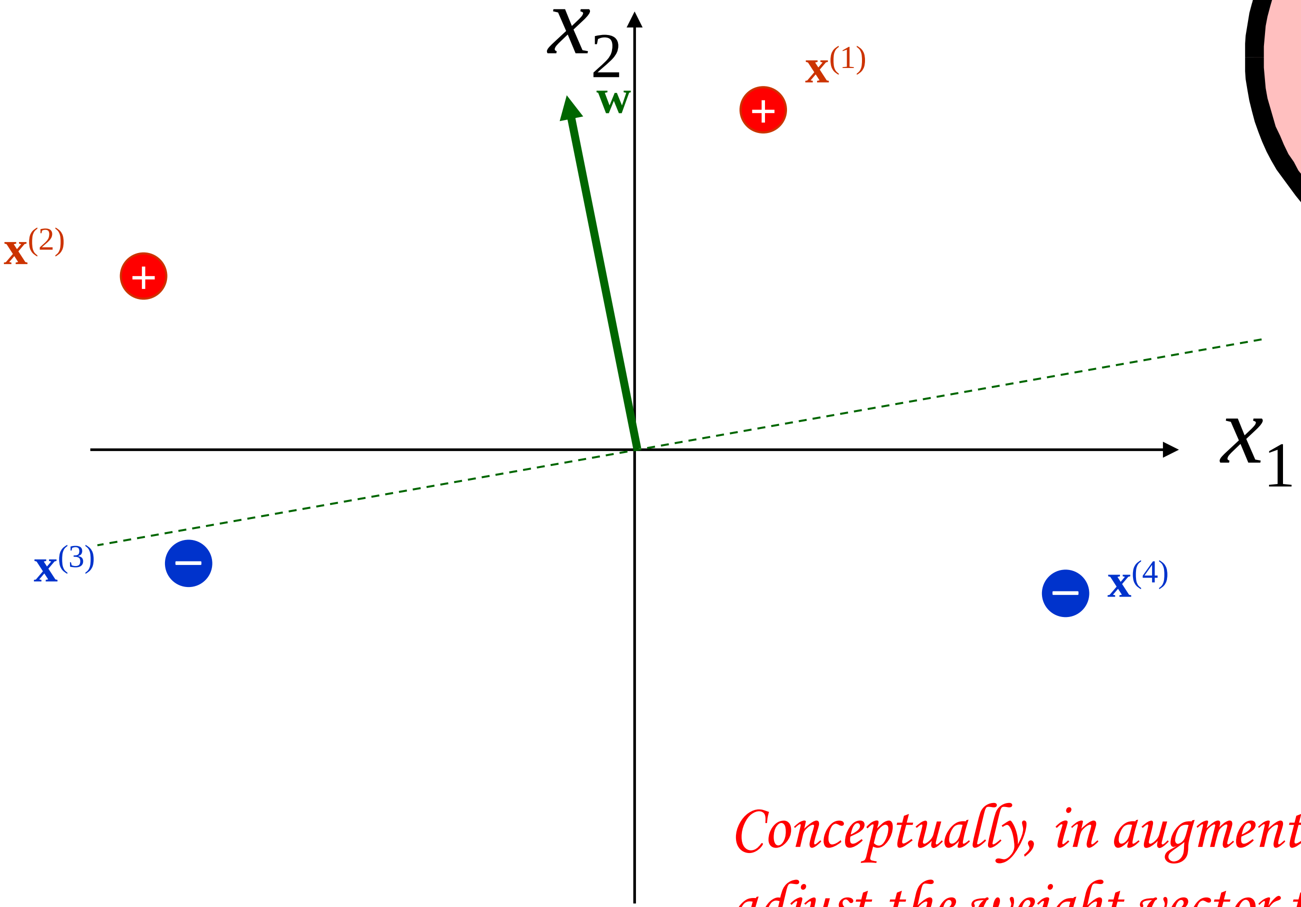
The Learning Scenario

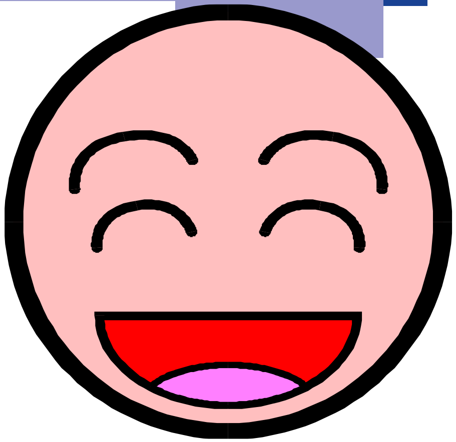


$\mathbf{w}_4 = \mathbf{w}_3$

The Learning Scenario

The demonstration is in augmented space.

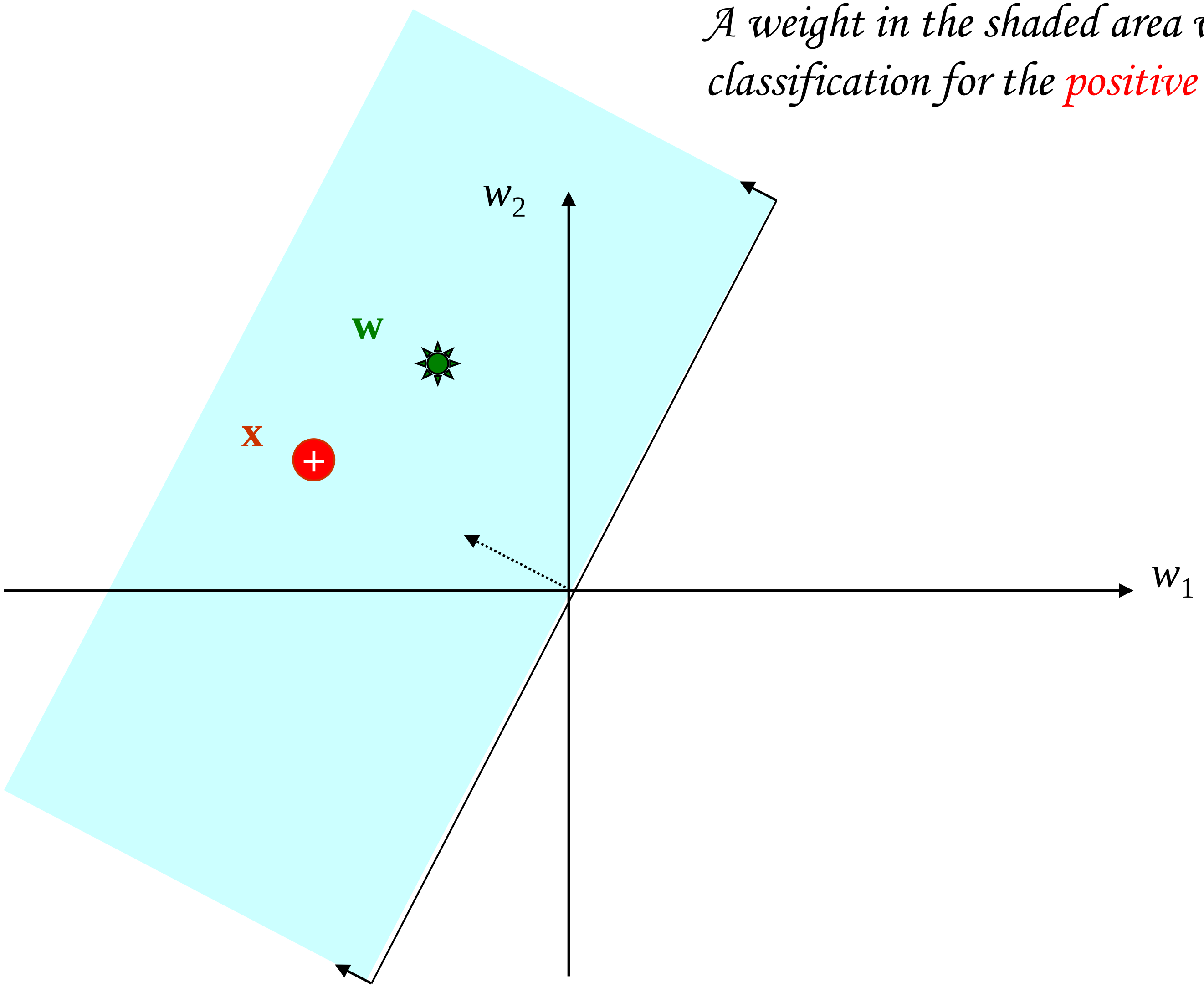




Conceptually, in augmented space, we adjust the weight vector to fit the data.

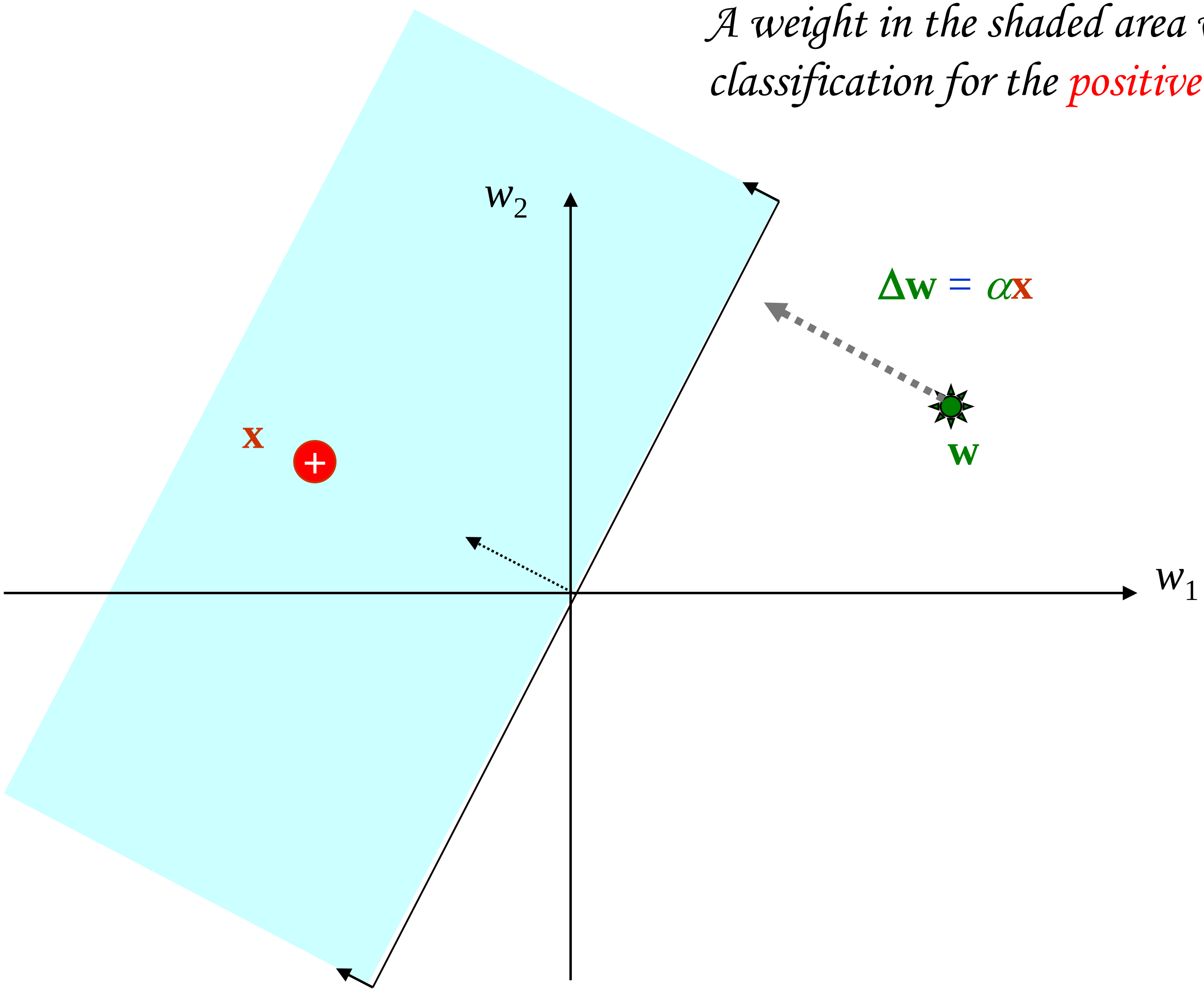
Weight Space

A weight in the shaded area will give correct classification for the positive example.



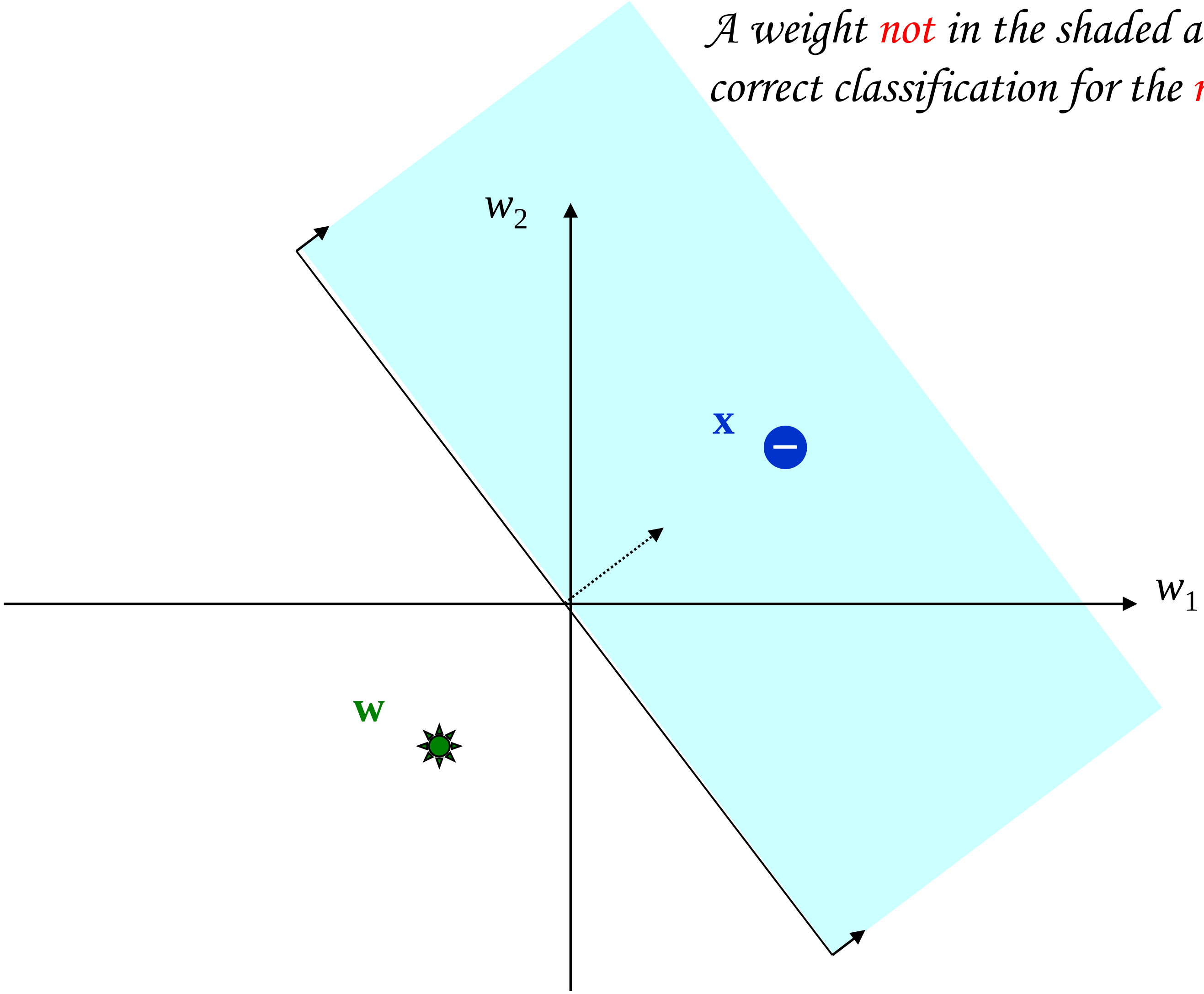
Weight Space

A weight in the shaded area will give correct classification for the positive example.



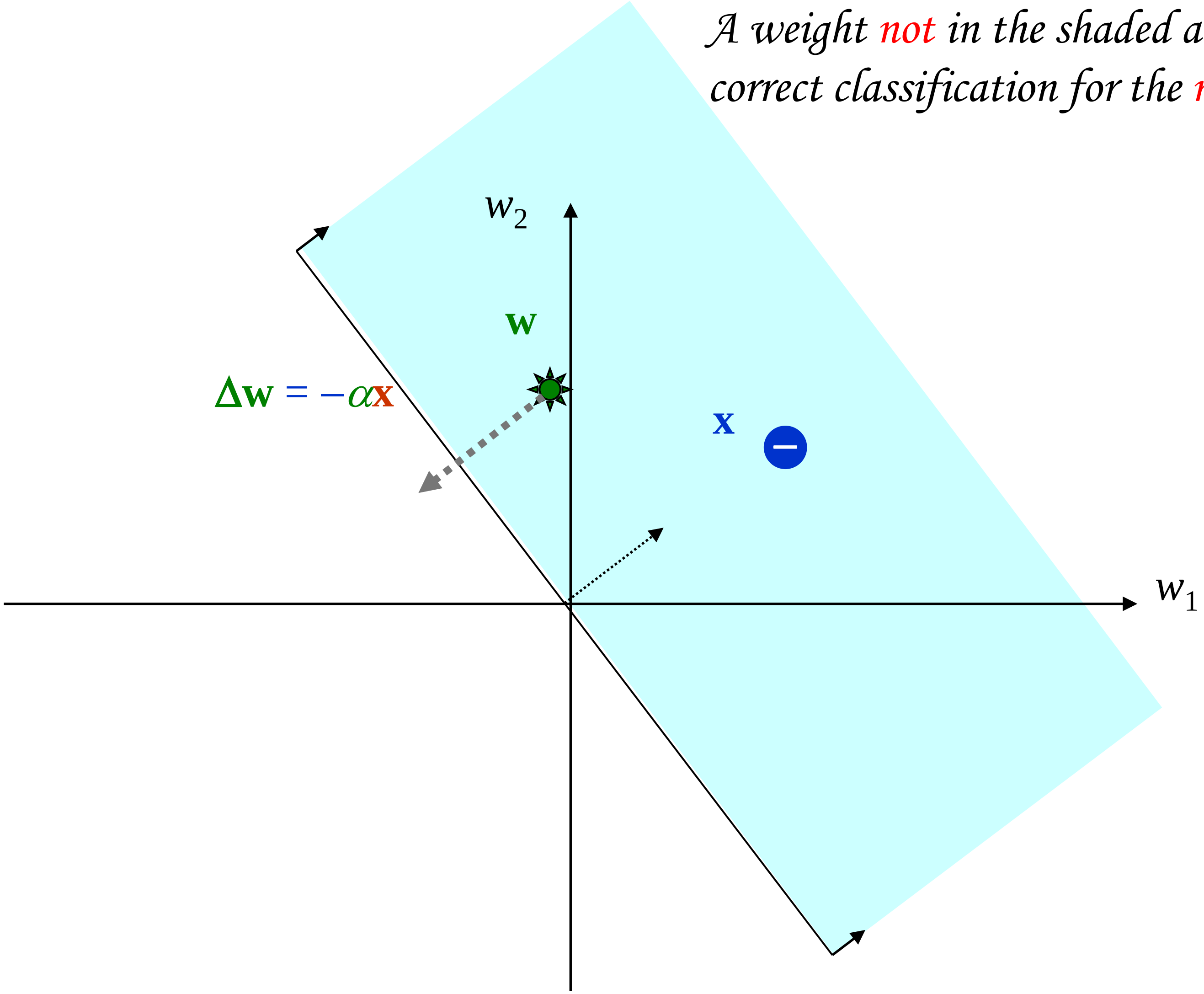
Weight Space

A weight not in the shaded area will give correct classification for the negative example.

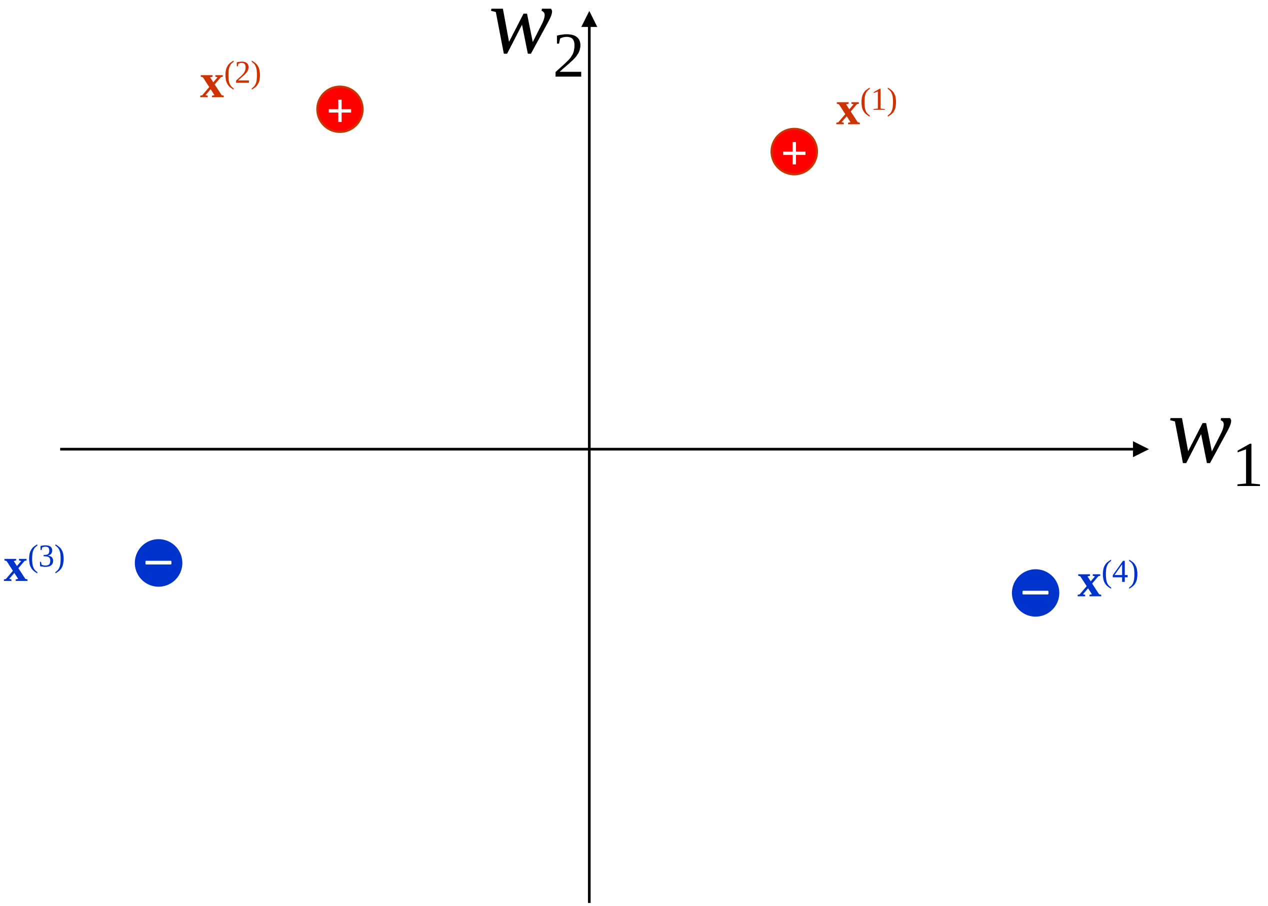


Weight Space

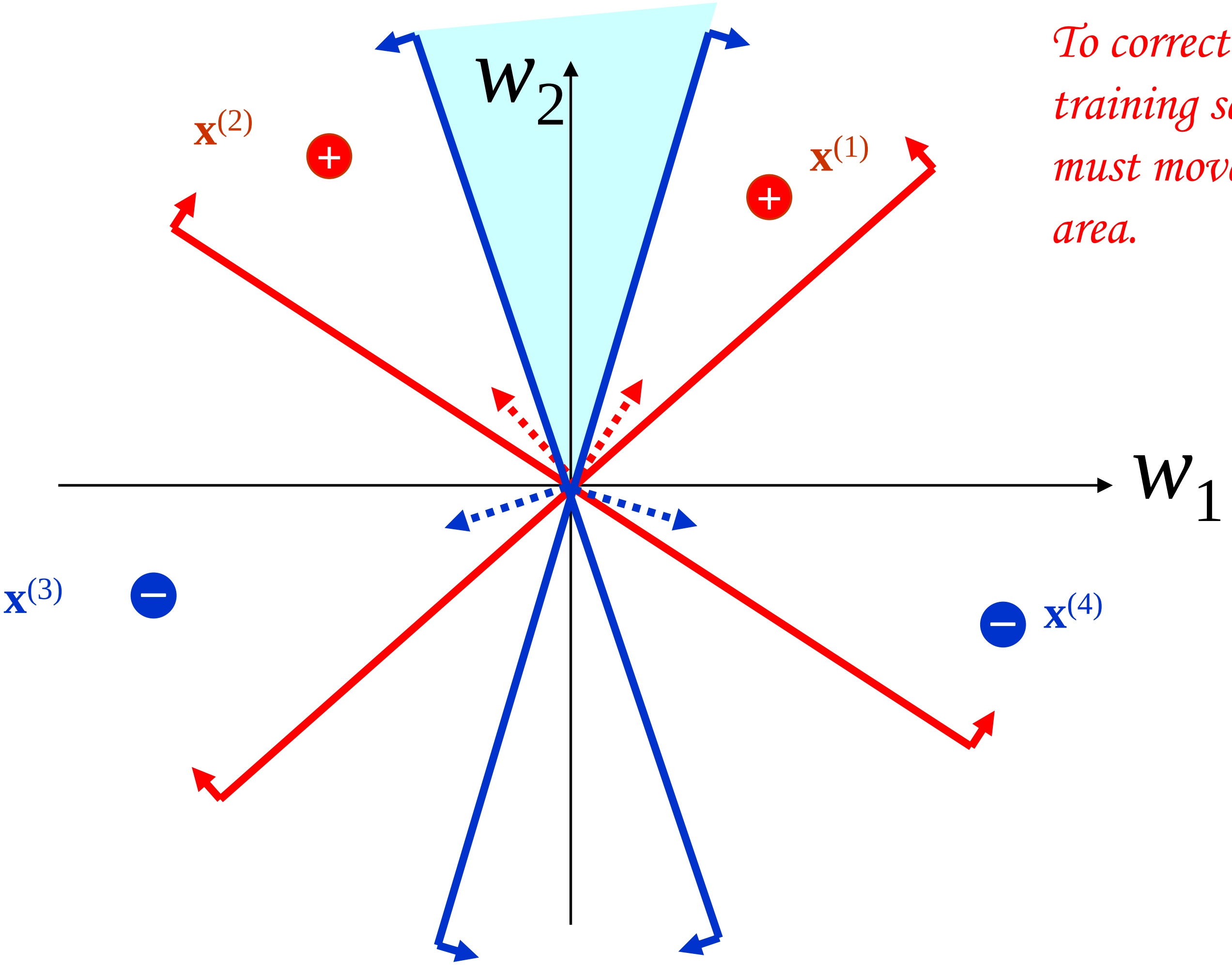
A weight not in the shaded area will give correct classification for the negative example.



Learning Scenario in Weight Space

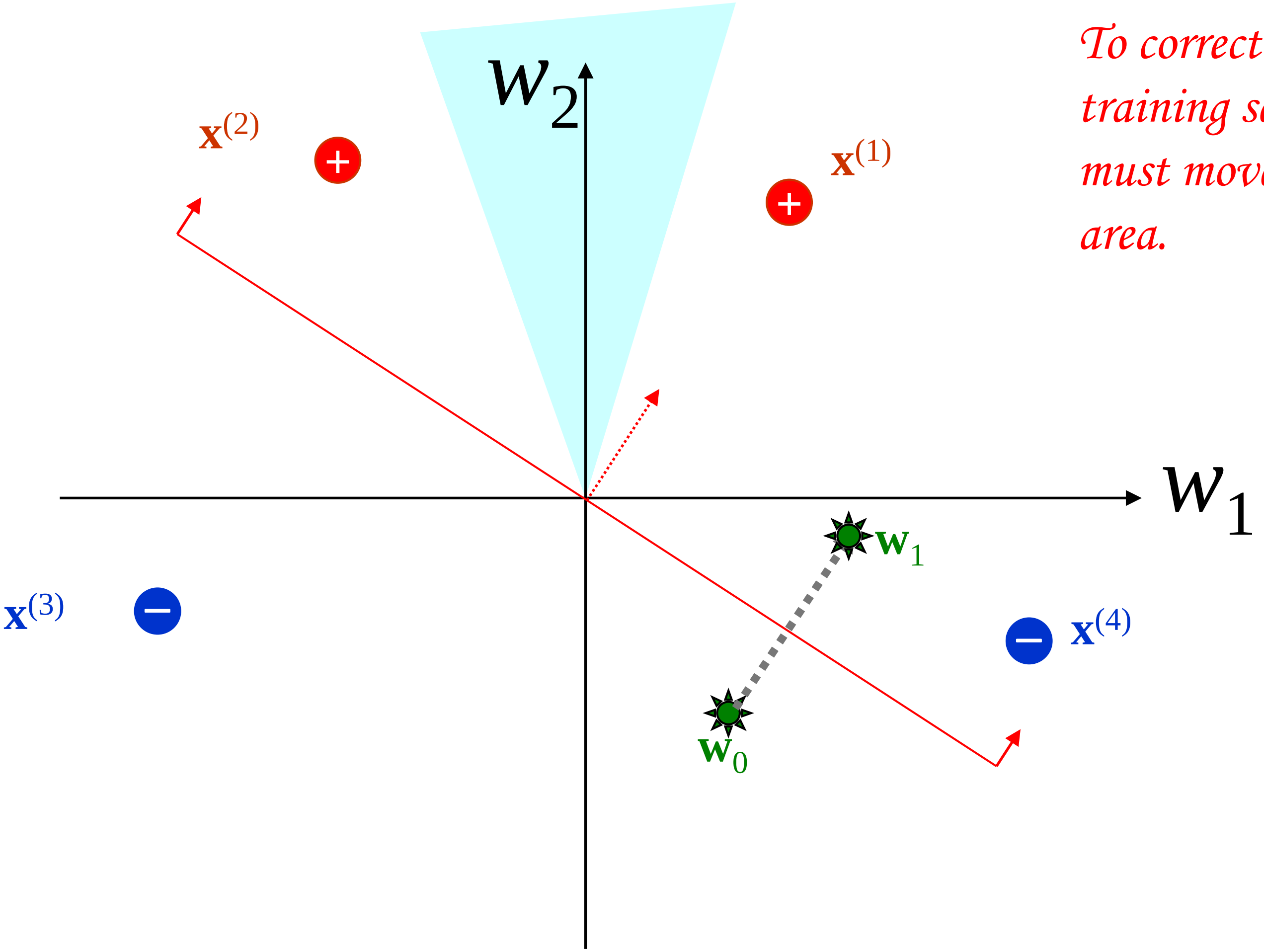


Learning Scenario in Weight Space



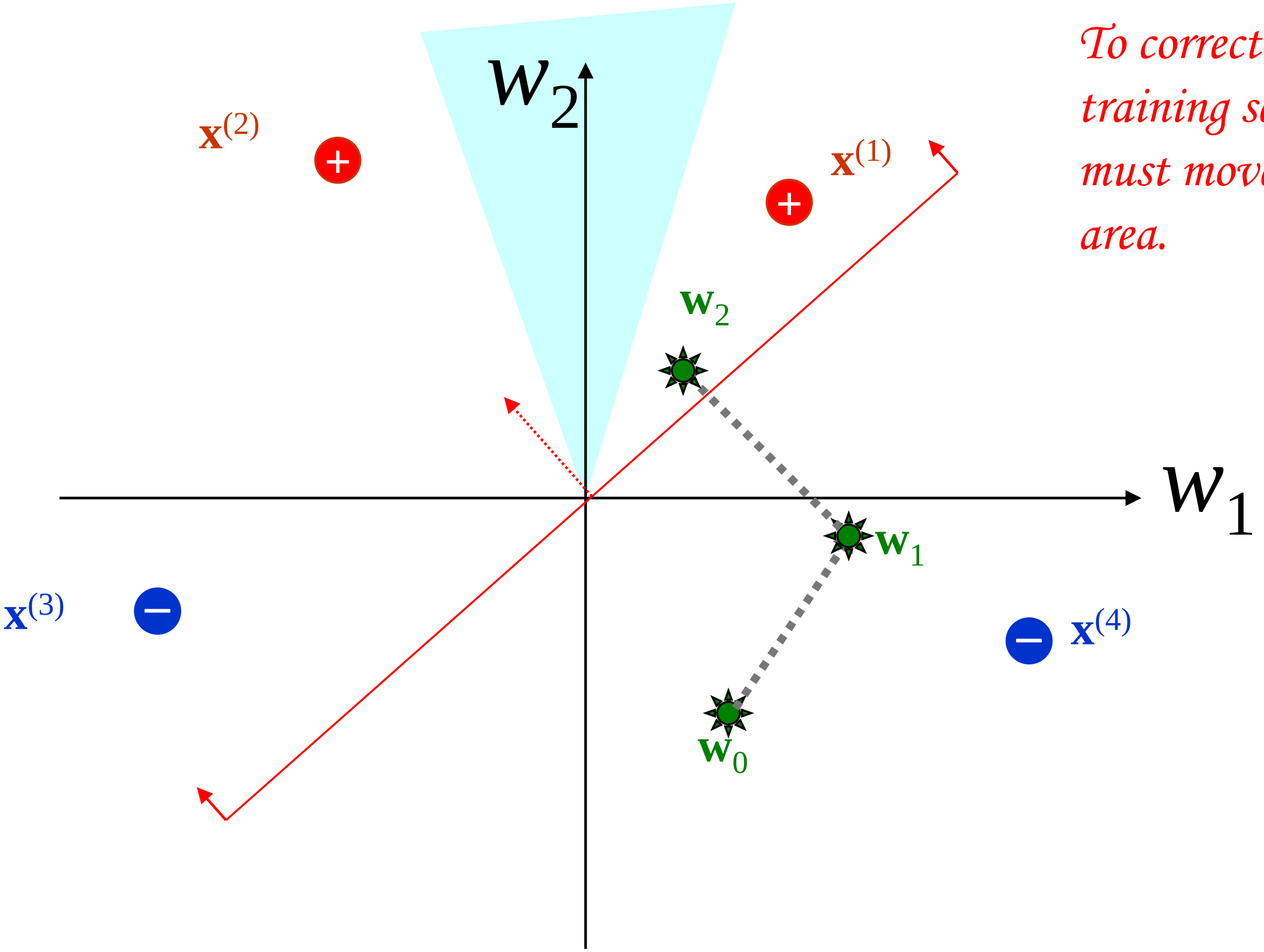
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



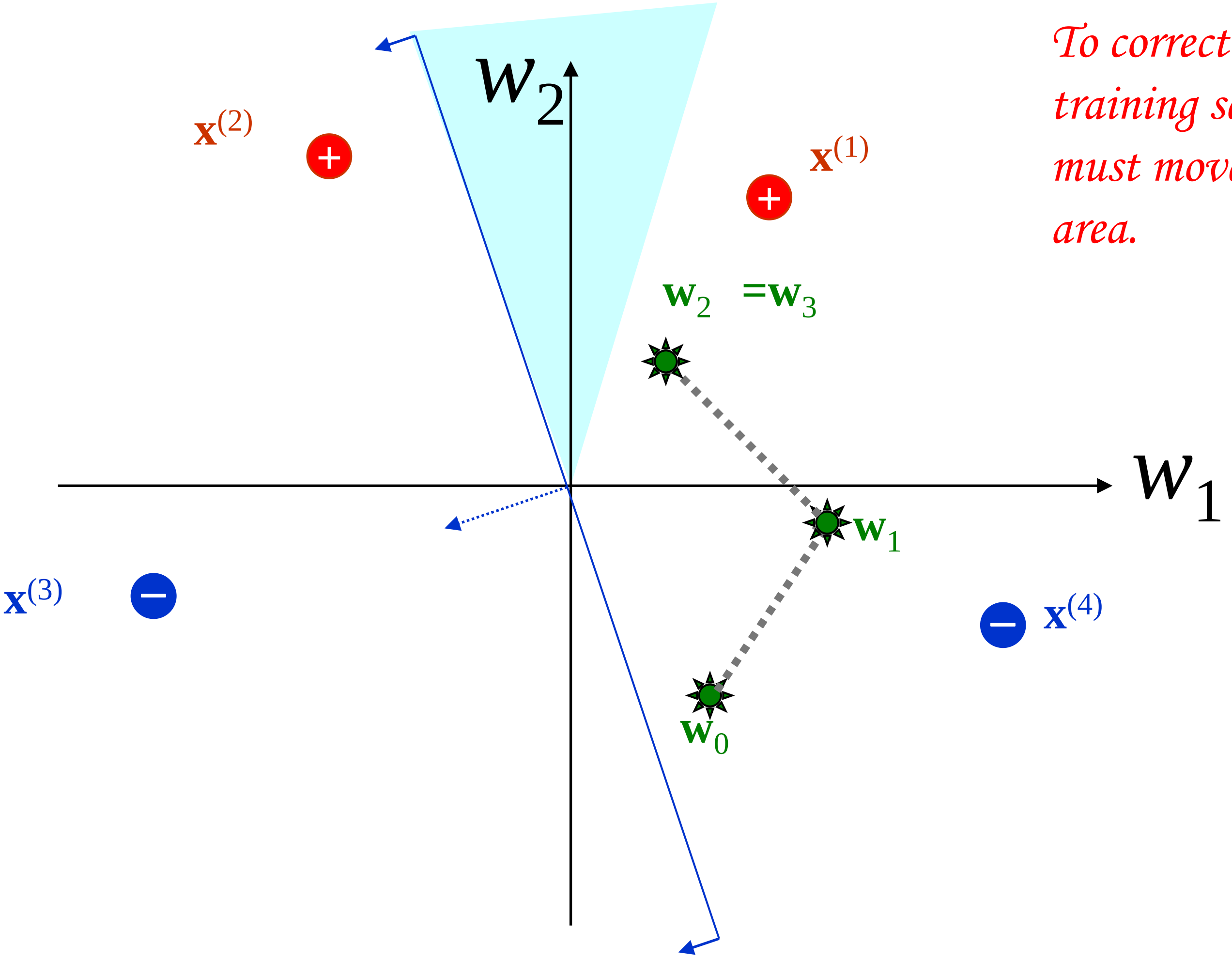
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



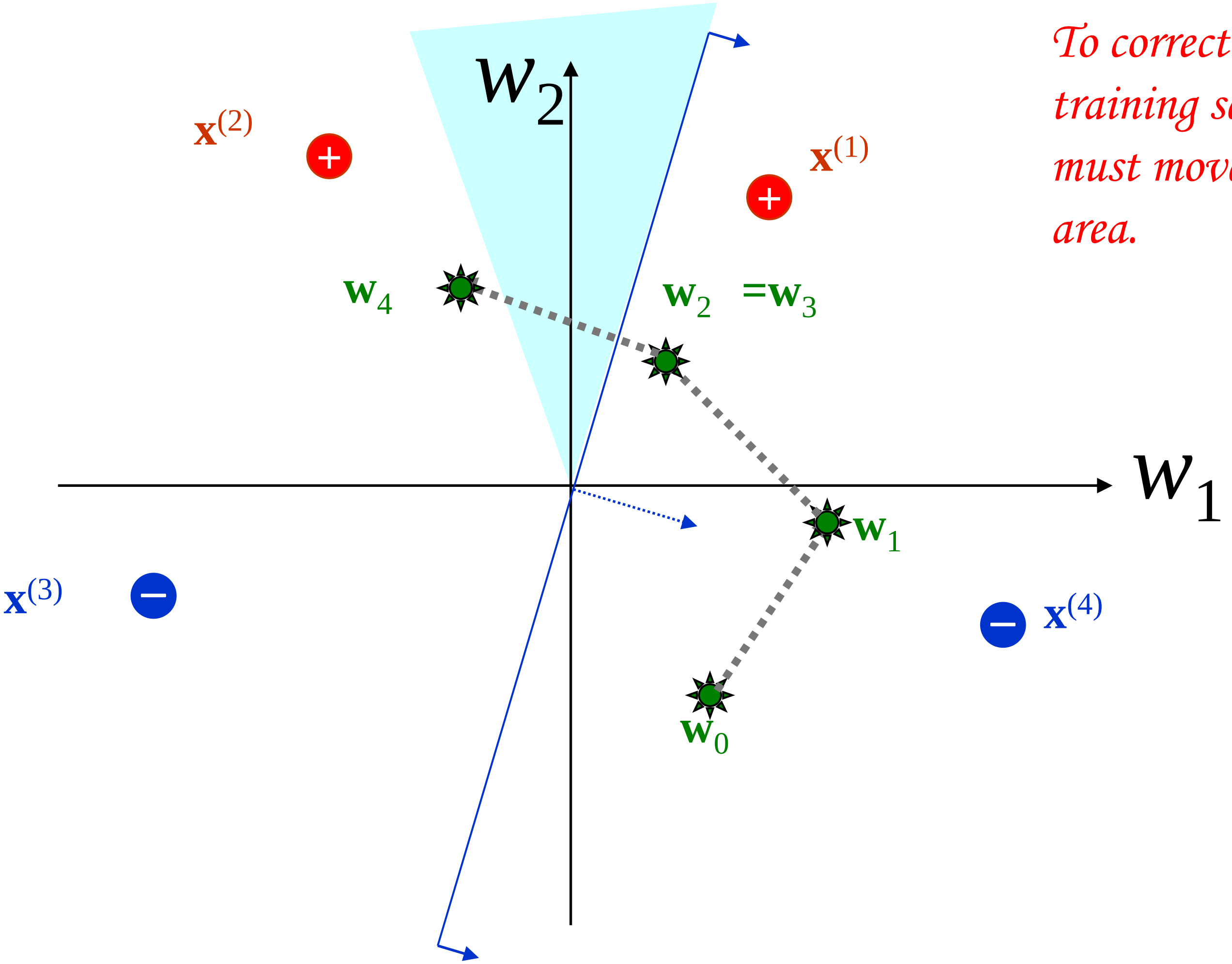
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



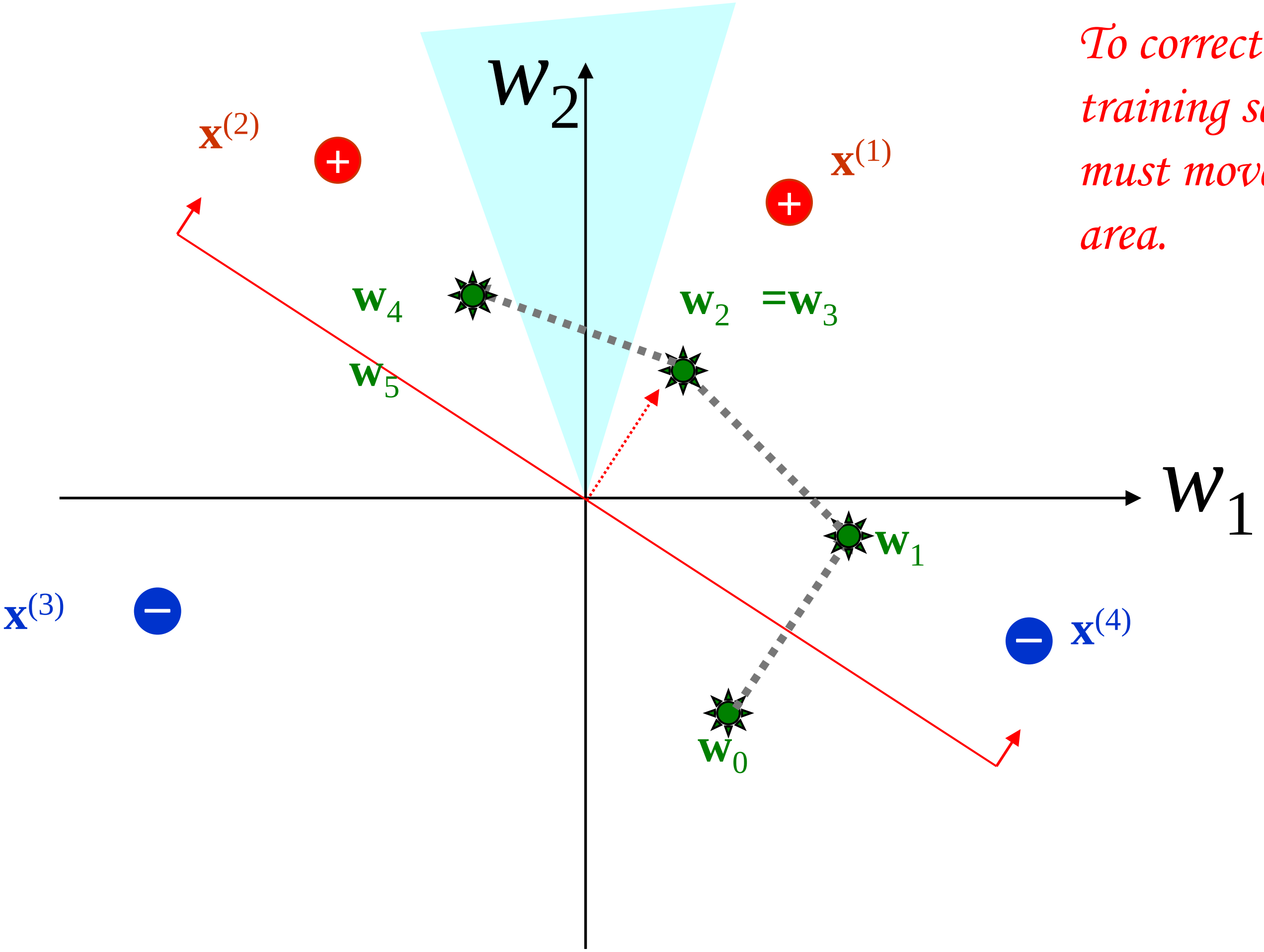
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



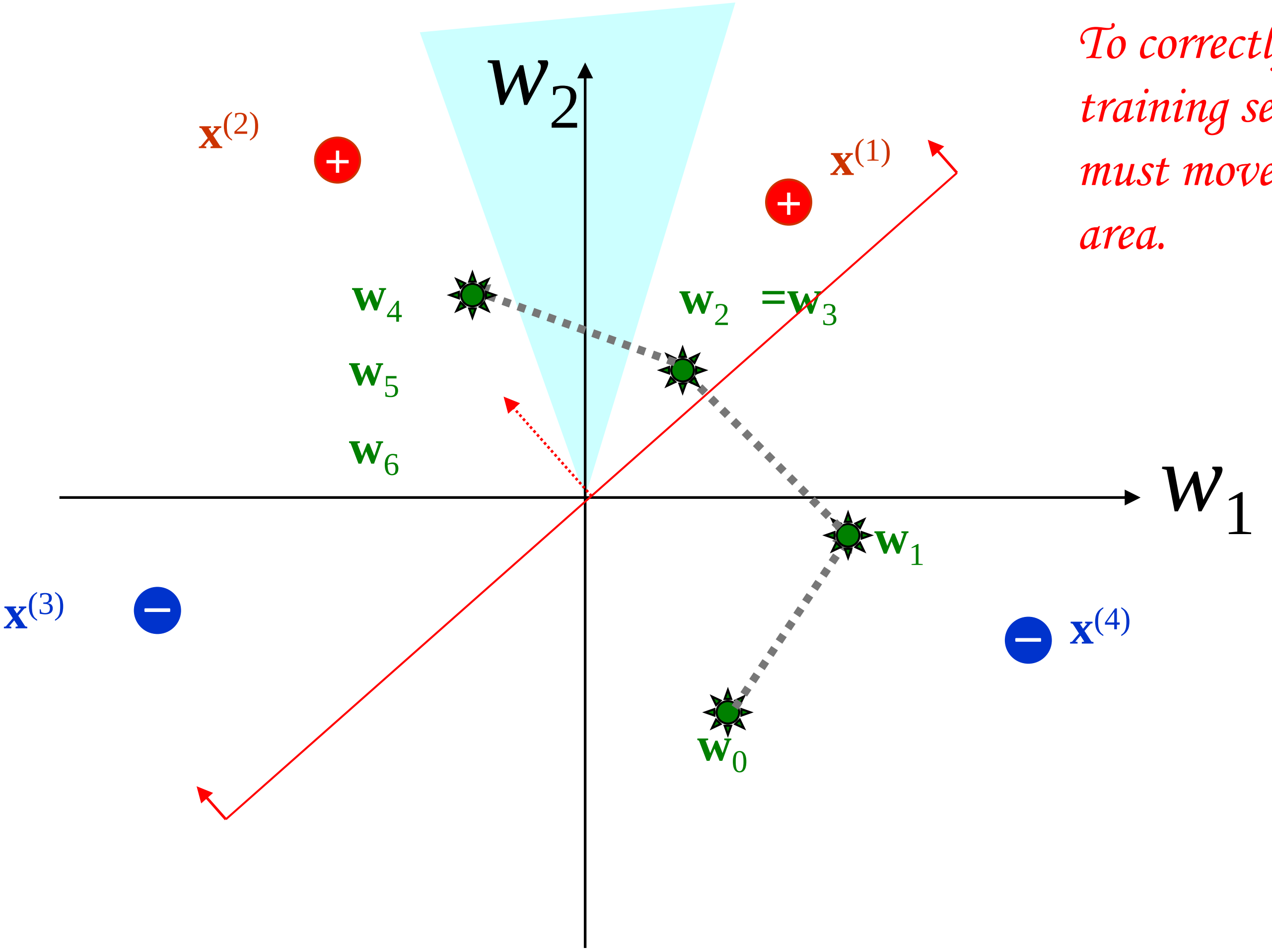
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



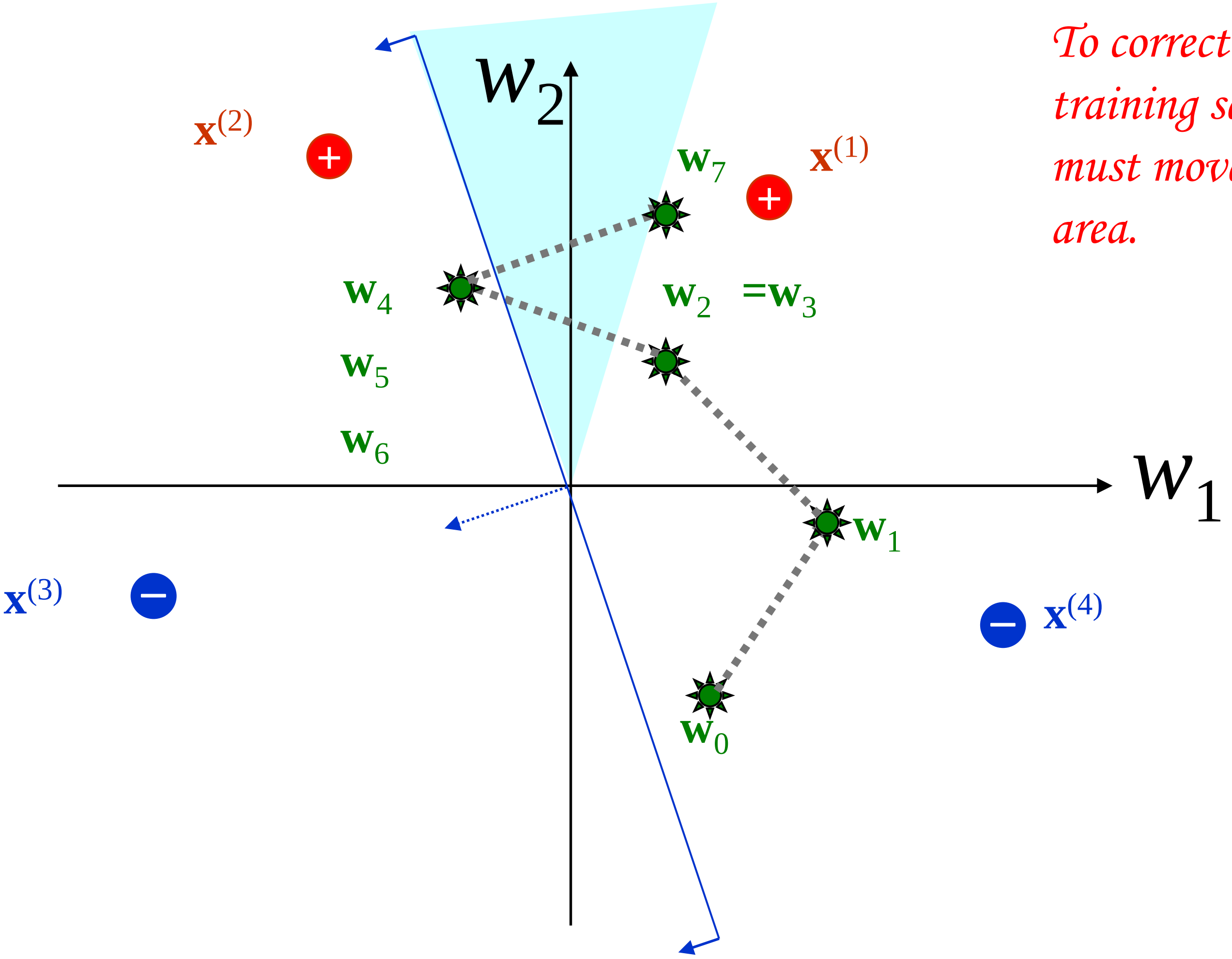
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



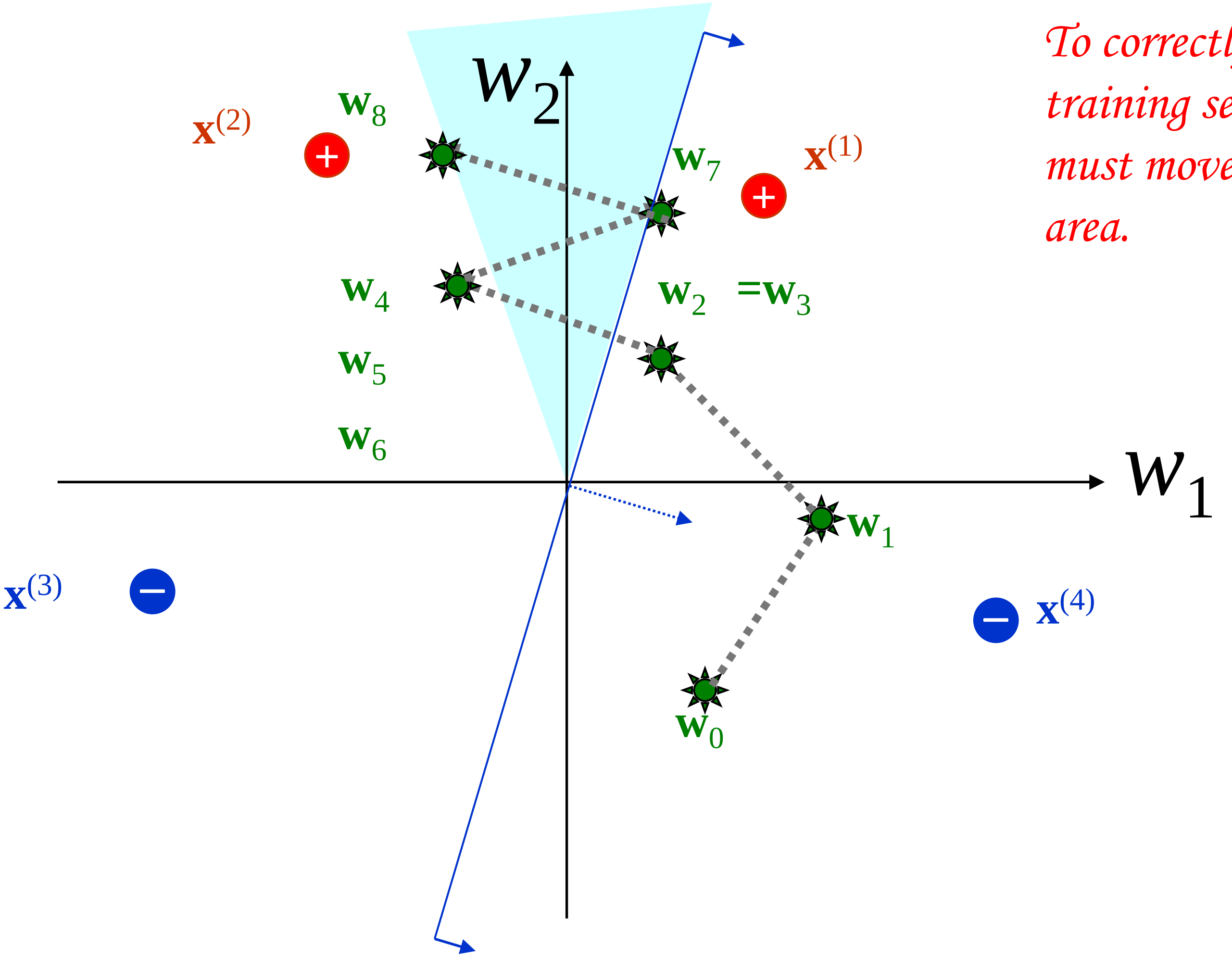
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



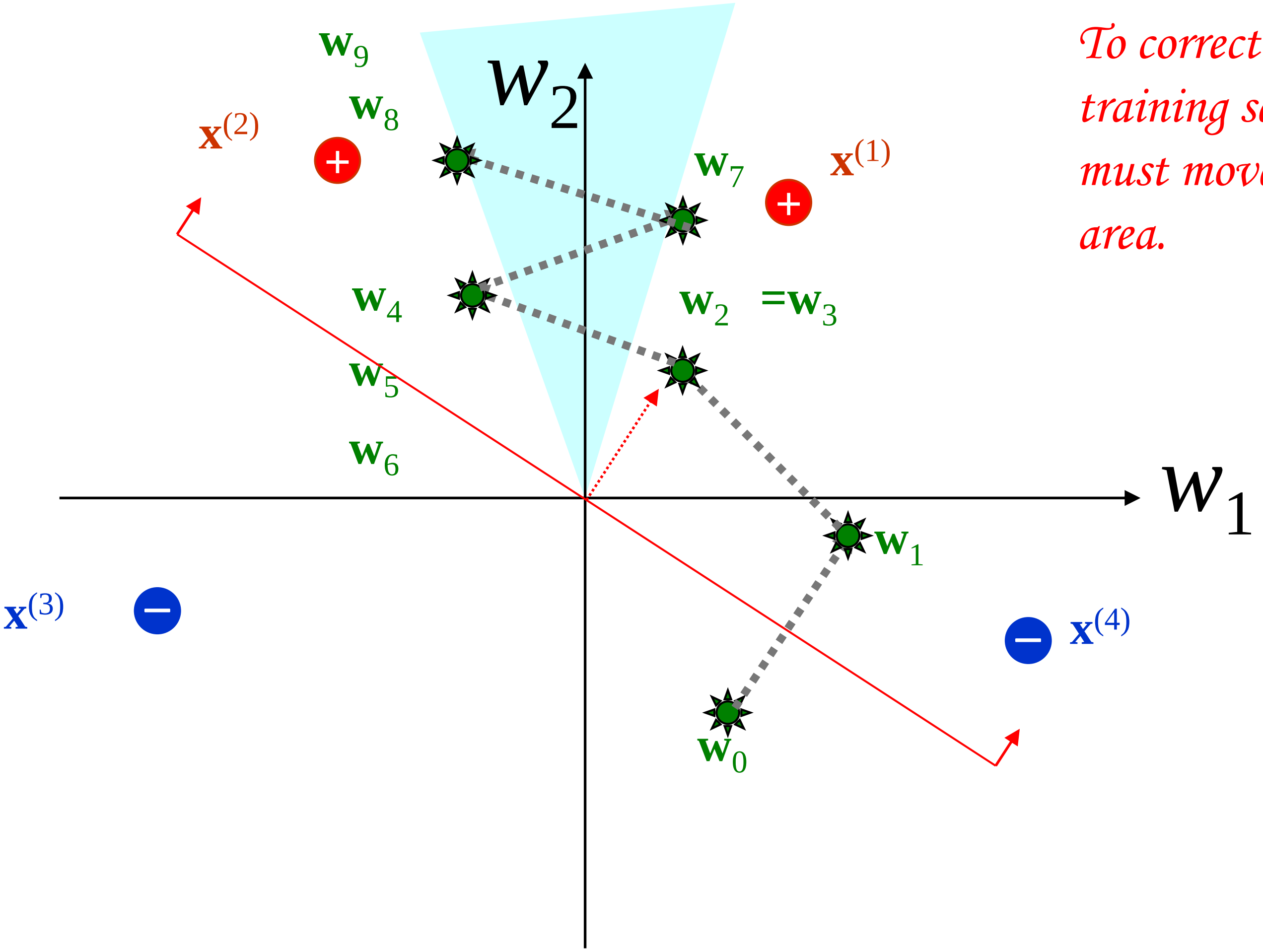
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



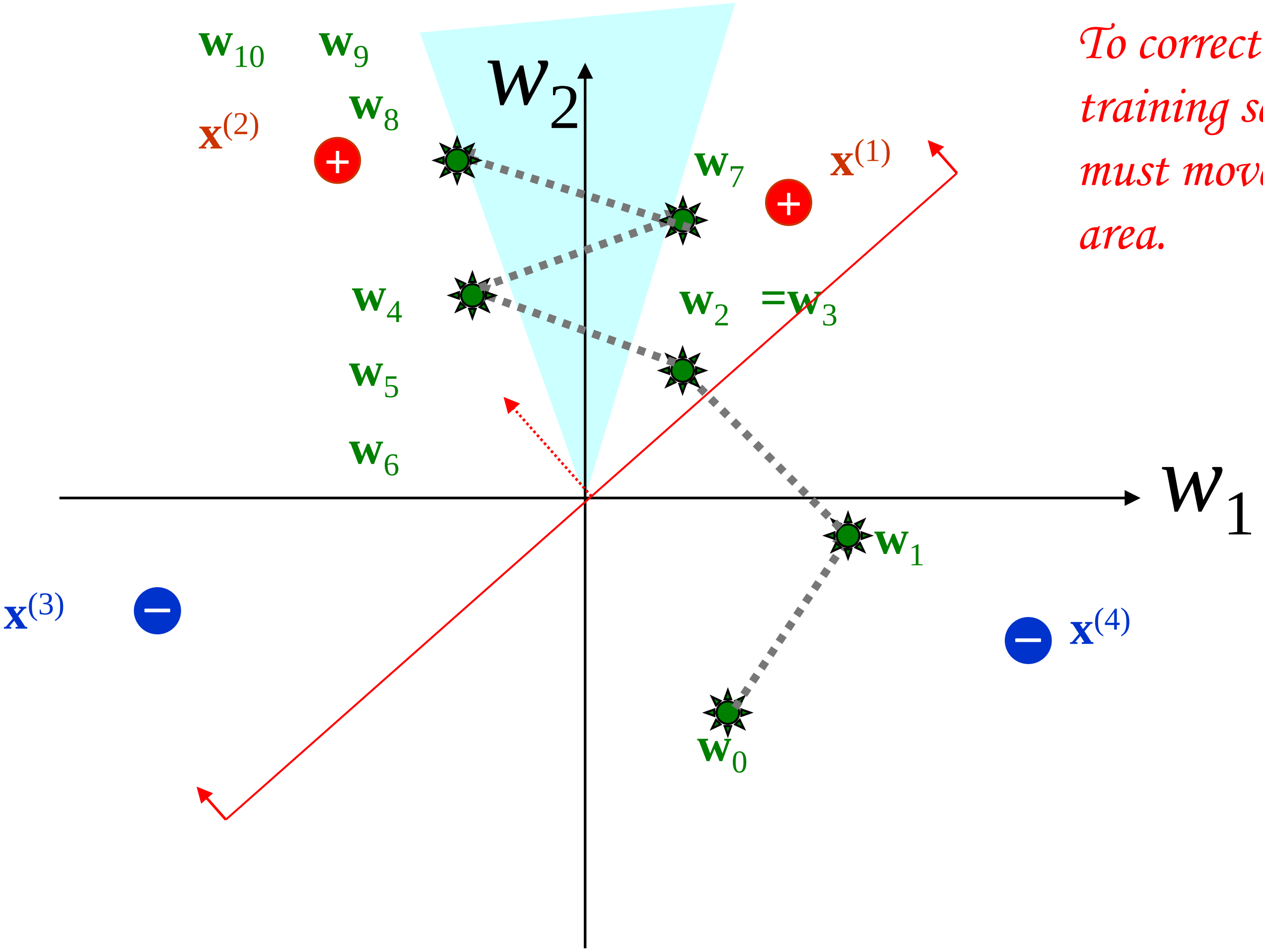
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



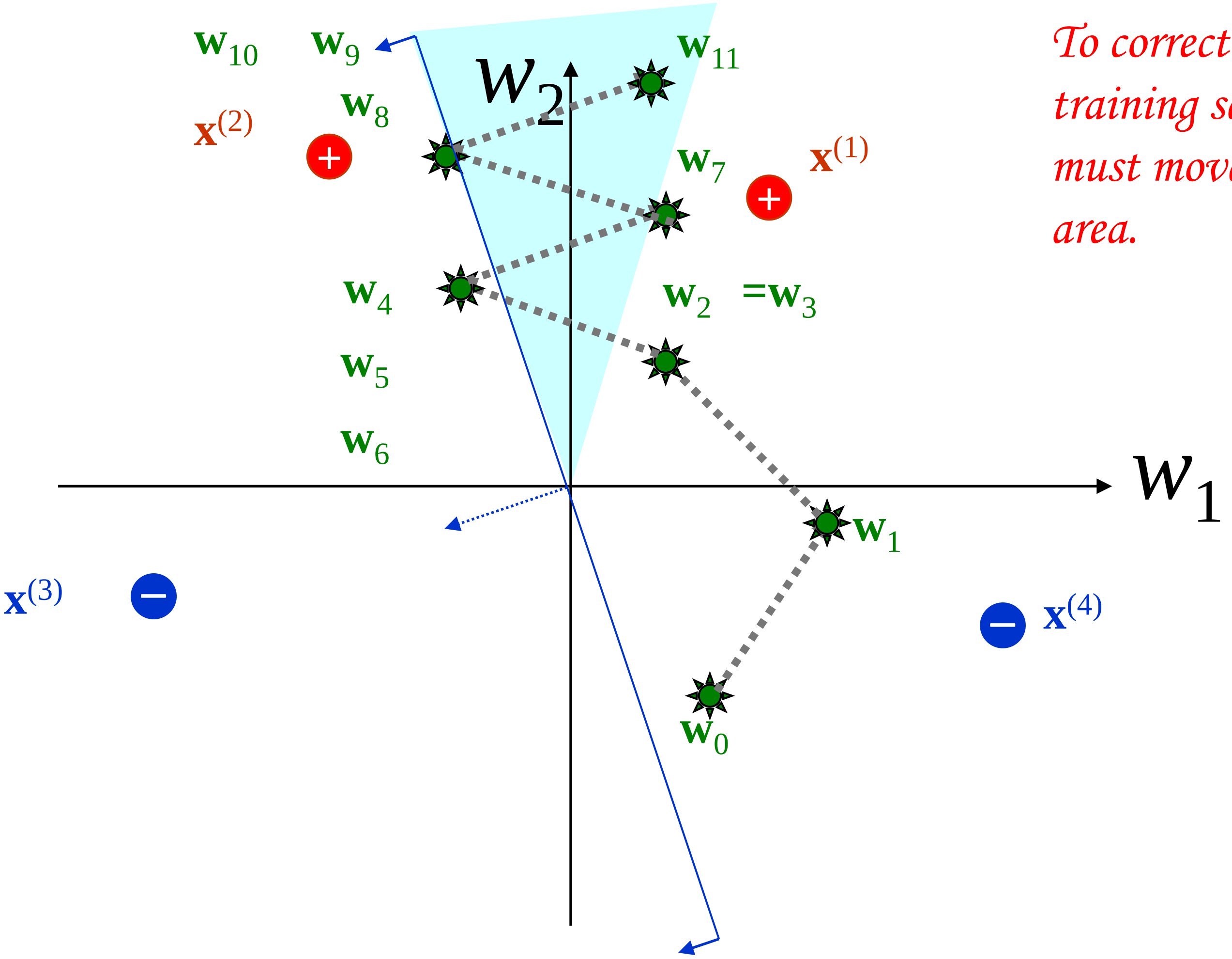
To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space



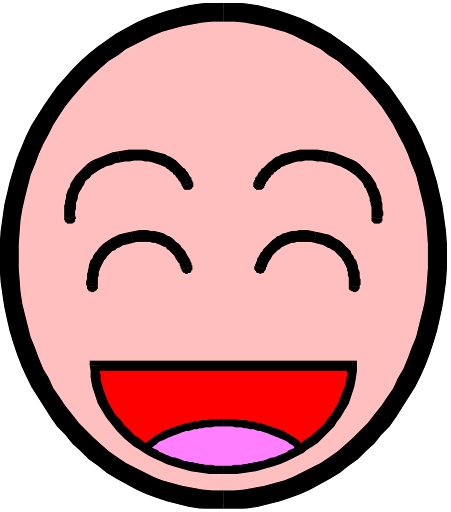
To correctly classify the training set, the weight must move into the shaded area.

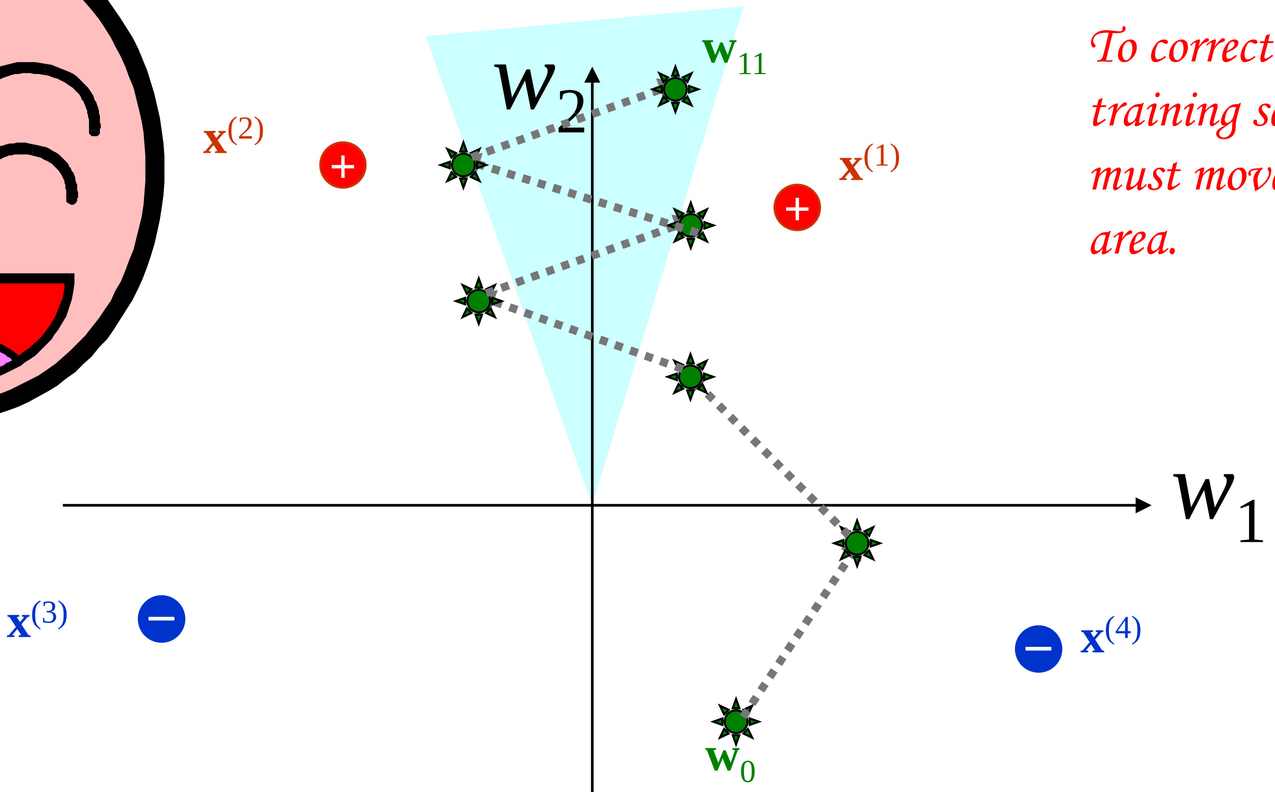
Learning Scenario in Weight Space



To correctly classify the training set, the weight must move into the shaded area.

Learning Scenario in Weight Space





To correctly classify the training set, the weight must move into the shaded area.

Conceptually, in weight space, we move the weight into the feasible region.

Conceptually, in weight space, we move the weight into the feasible region.

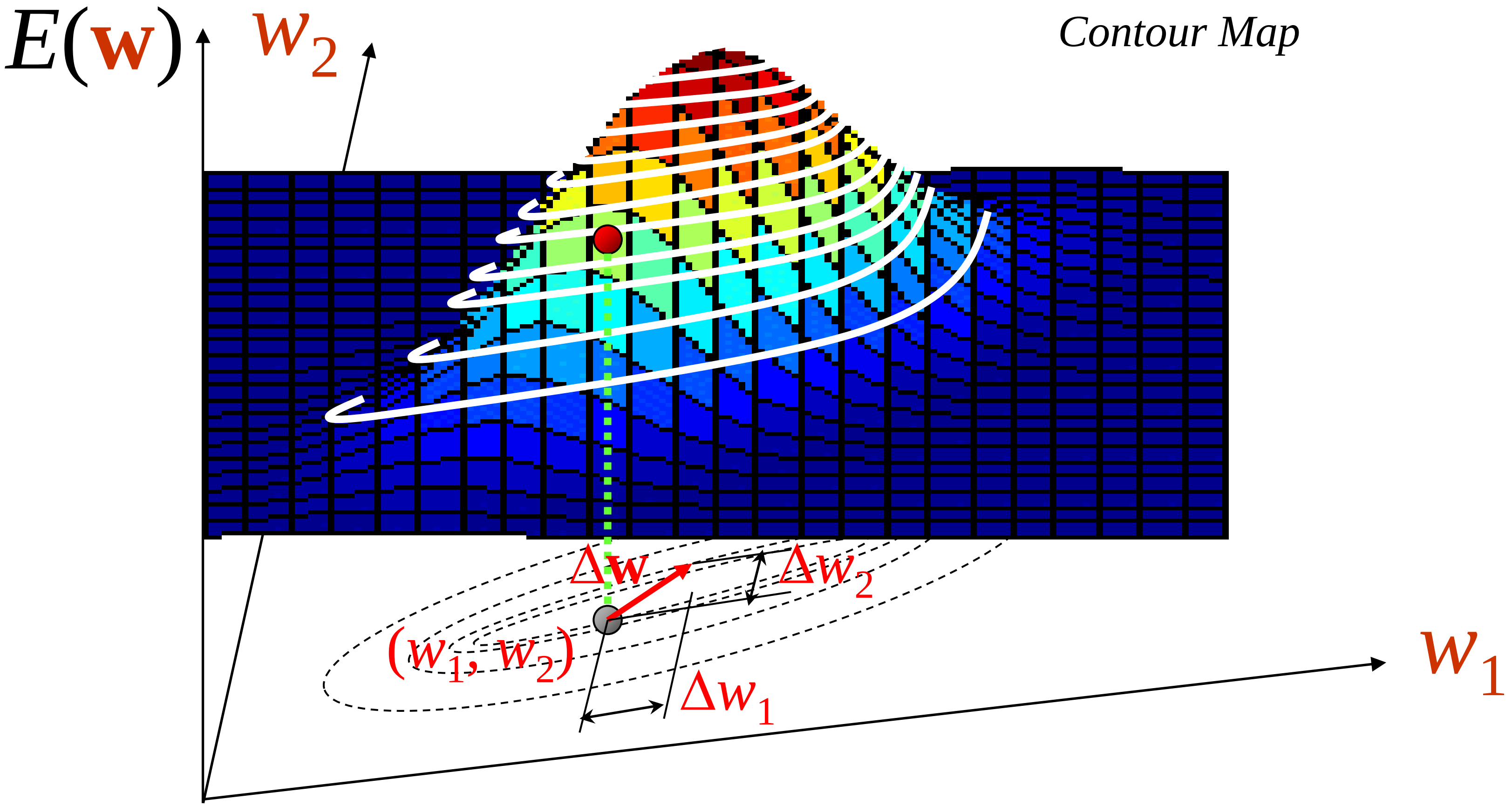
Least Mean Square Learning

- Minimize the cost function (error function):

$$
\begin{aligned}
E(\mathbf{w}) &= \frac{1}{2}\sum_{k=1}^{p}(d^{(k)} - y^{(k)})^2 \\
&= \frac{1}{2}\sum_{k=1}^{p}(d^{(k)} - \mathbf{w}^T\mathbf{x}^{(k)})^2 \\
&= \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - \sum_{l=1}^{m} w_l x_l^{(k)}\right)^2
\end{aligned}
$$

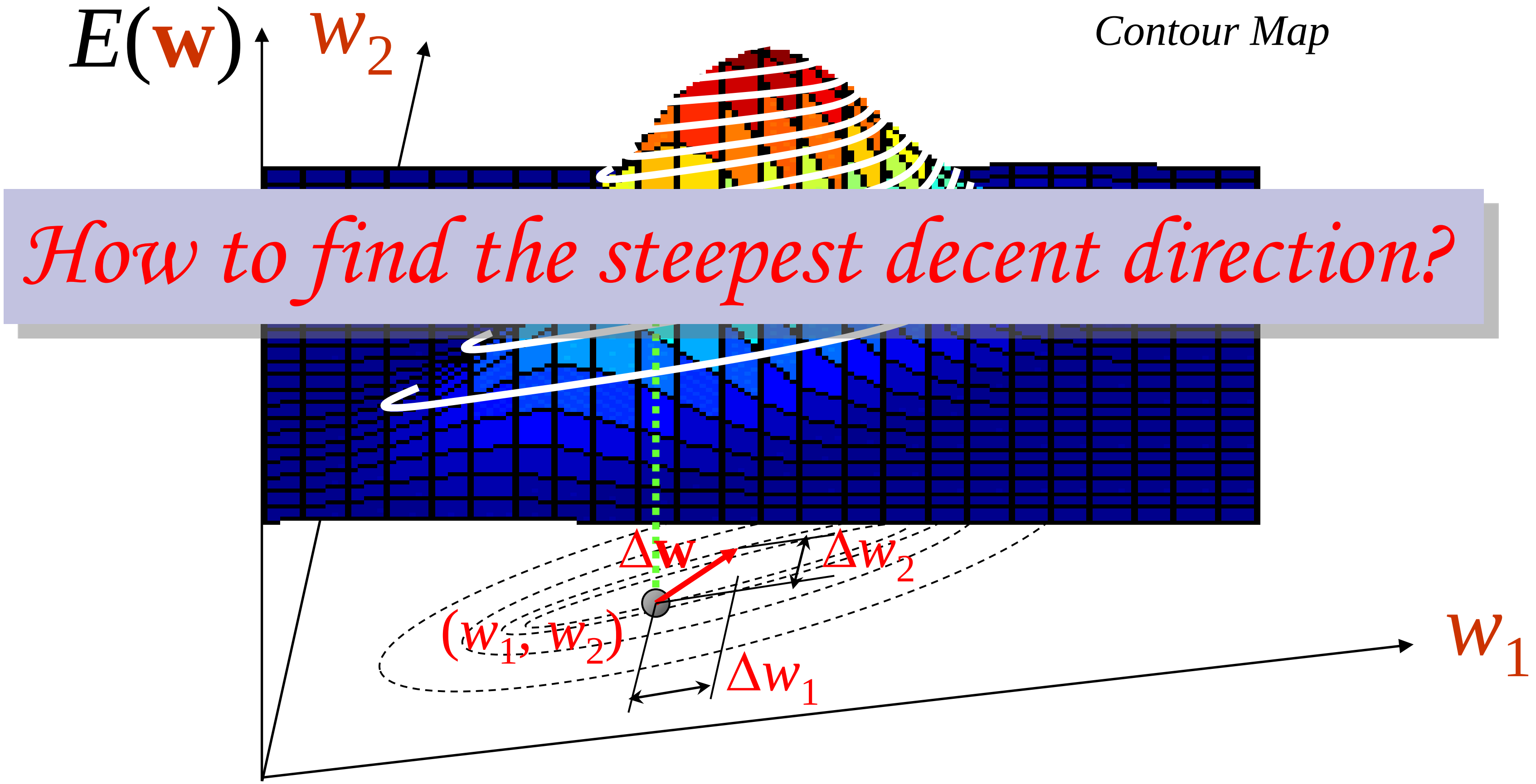
Least Mean Square Learning

- Our goal is to go *downhill*.



Least Mean Square Learning

- Our goal is to go *downhill*.



How to find the steepest decent direction?

Least Mean Square Learning

- Gradient Operator

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

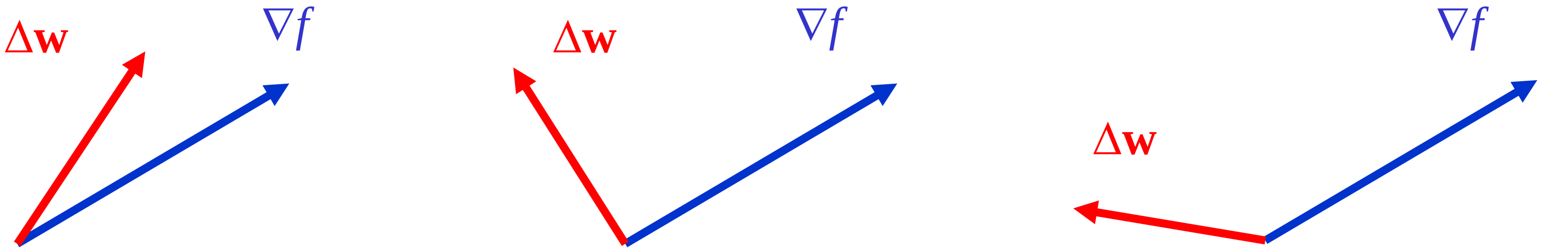
$$df = \frac{\partial f}{\partial w_1} dw_1 + \frac{\partial f}{\partial w_2} dw_2 + \cdots + \frac{\partial f}{\partial w_m} dw_m$$

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

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

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

Least Mean Square Learning



df : positive

Go uphill

df : zero

Plain

df : negative

Go downhill

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

Least Mean Square Learning

To minimize *f* , we choose

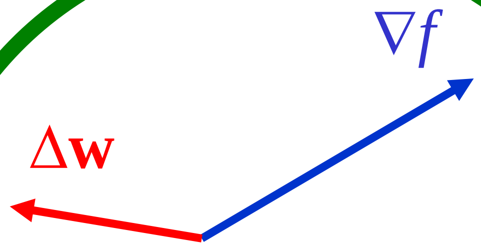
$$\Delta \mathbf{w} = -\eta \nabla f$$

df : positive

Go uphill

df : zero

Plain



df : negative

Go downhill

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

Least Mean Square Learning

MIMA

- Minimize the cost function (error function):

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

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

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

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

Least Mean Square Learning

- 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

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

Least Mean Square Learning

- Learning Modes
  - Batch Learning Mode

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

  - Incremental Learning Mode

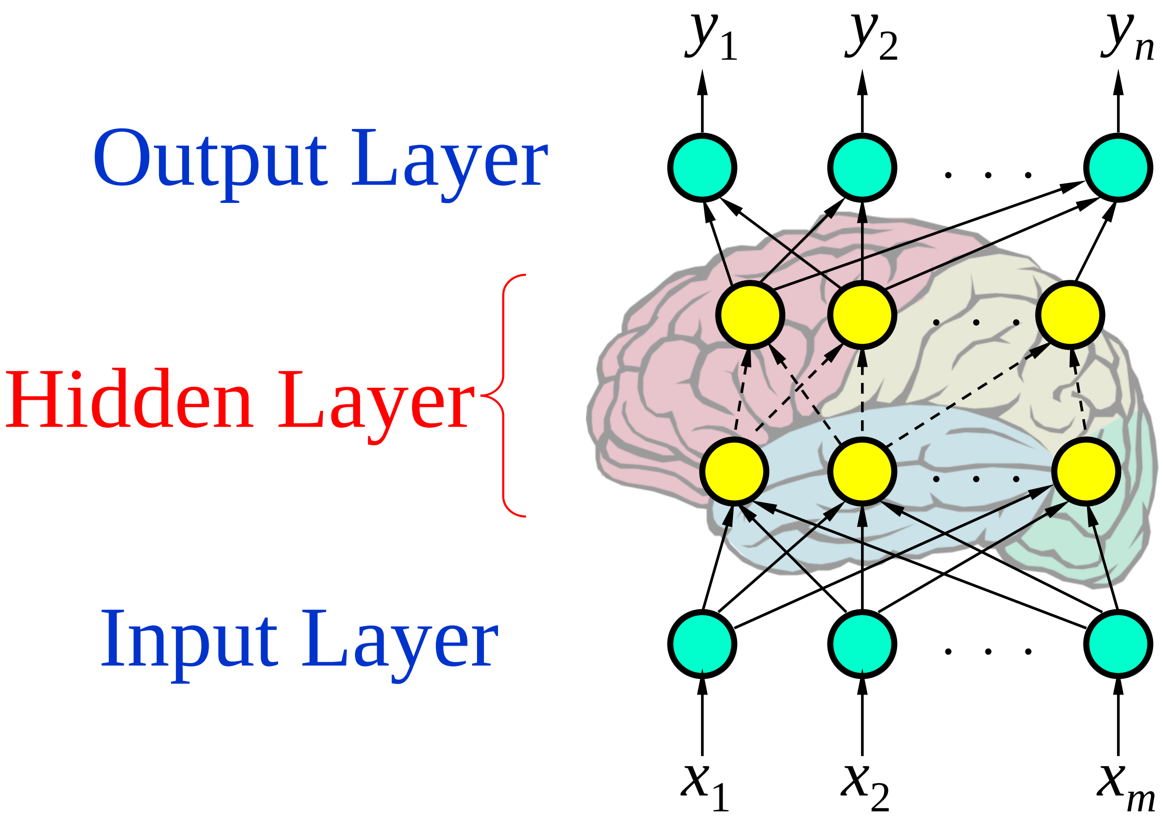
$$\Delta w_j = \eta \delta^{(k)} x_j^{(k)}$$

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

Perceptron

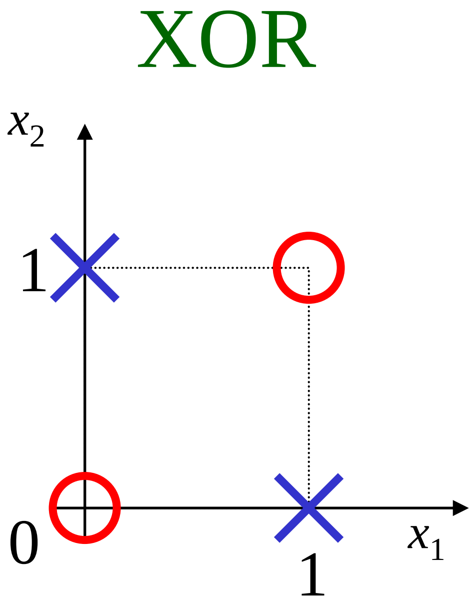
- Summary
  - Separability: some parameters get the training set perfectly correct
  - Convergence: if the training is separable, perceptron will eventually converge (binary case)?
- The Perceptron convergence theorem
- The relation between perceptron and Bayes classifier

Multilayer Perceptron



How an MLP Works?

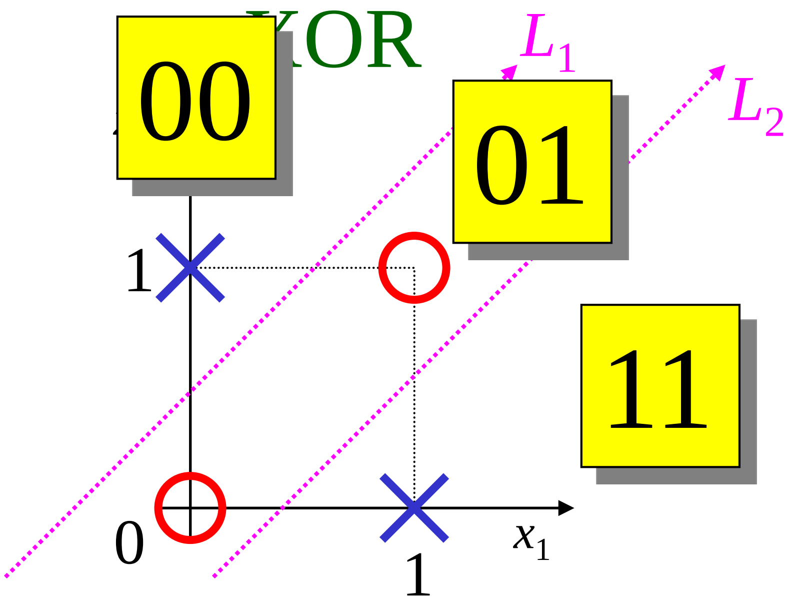
Example:

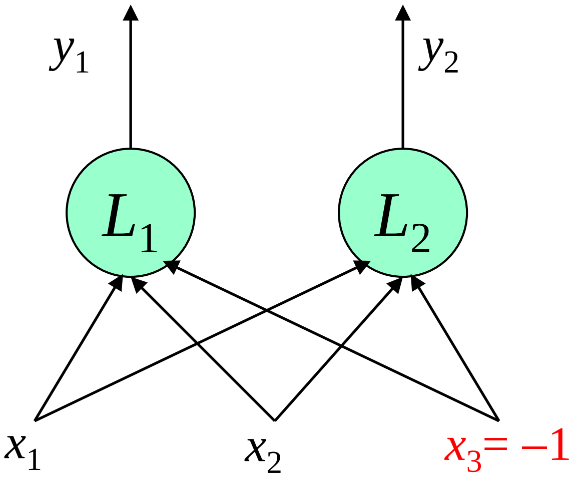


- Not linearly separable.
- Is a single layer perceptron workable?

How an MLP Works?

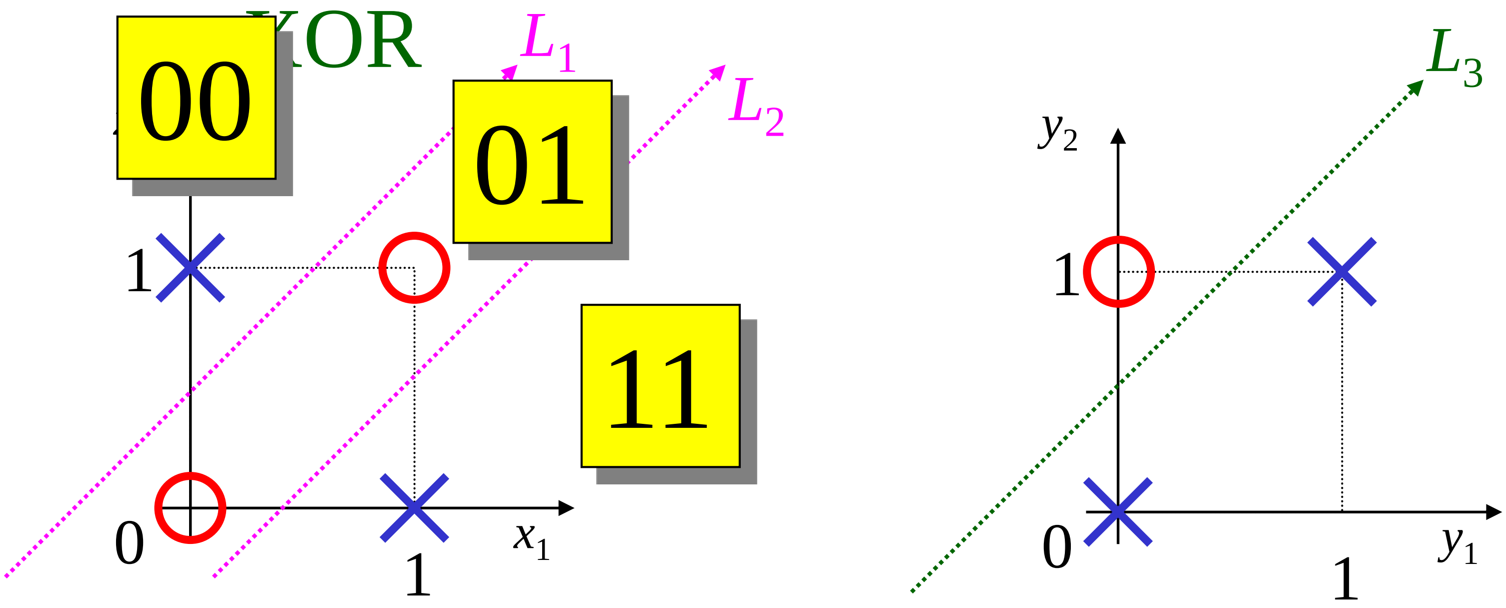
Example:





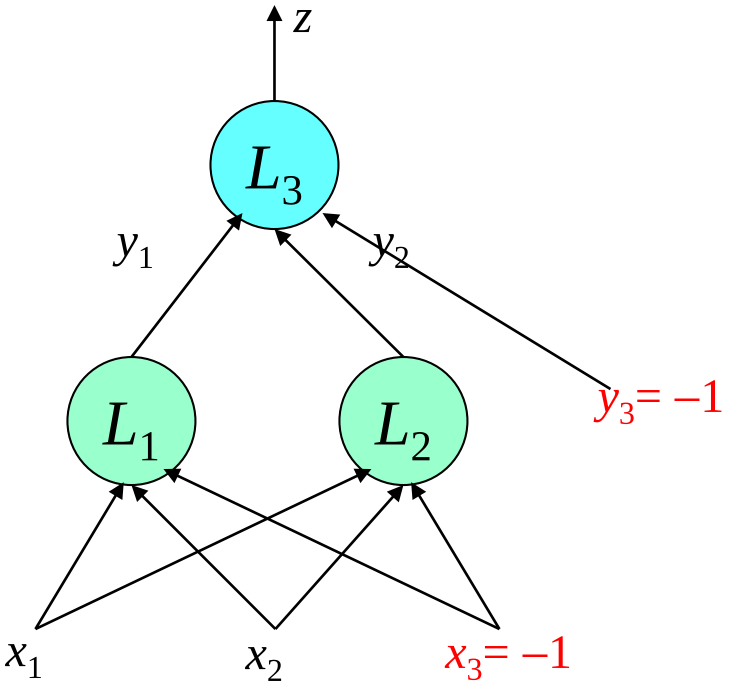
How an MLP Works?

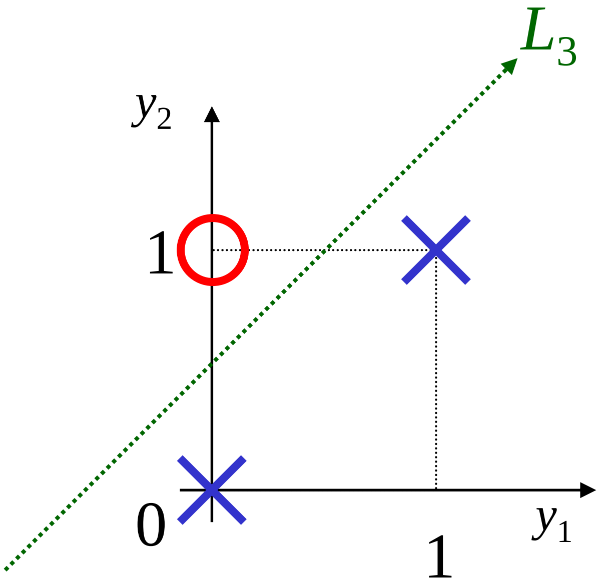
Example:



How an MLP Works?

Example:

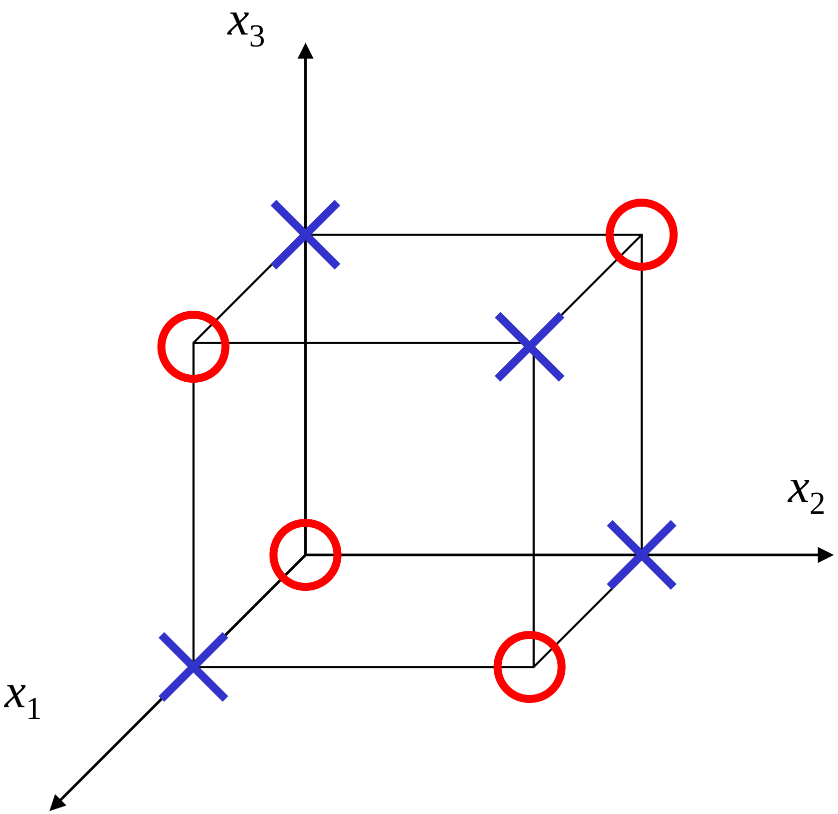




Parity Problem

Is the problem linearly separable?

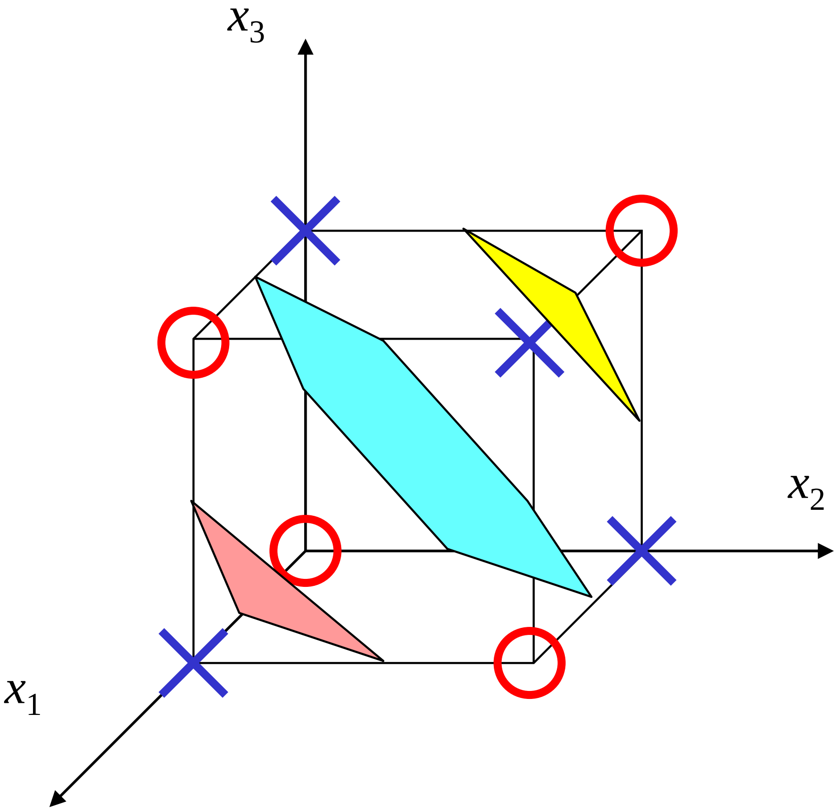
$x_1 x_2 x_3$	
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1



Parity Problem

Is the problem linearly separable?

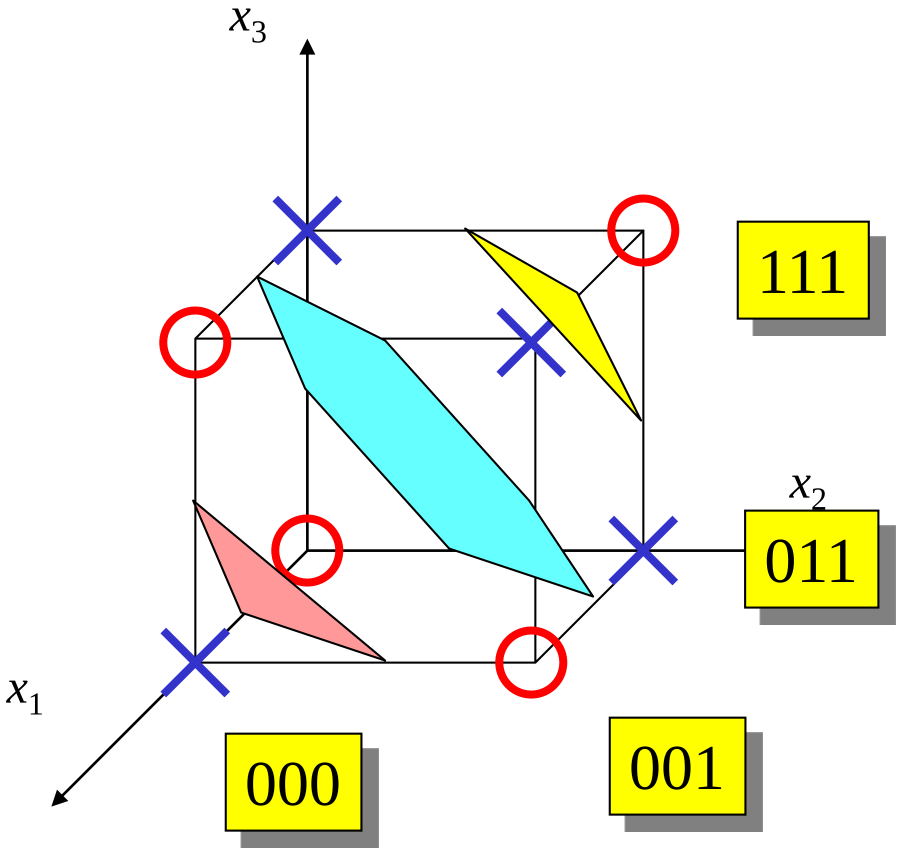
$x_1 x_2 x_3$	
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1



Parity Problem

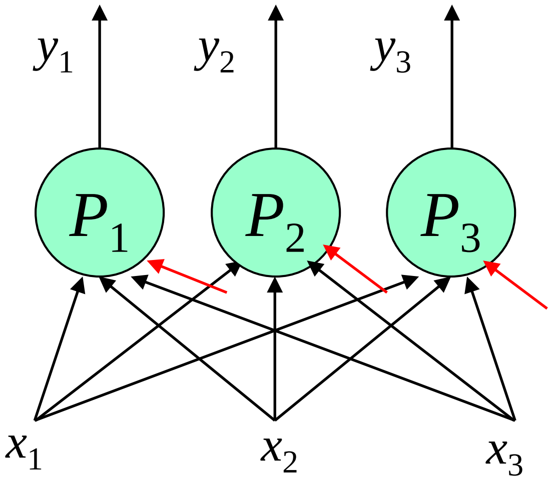
Is the problem linearly separable?

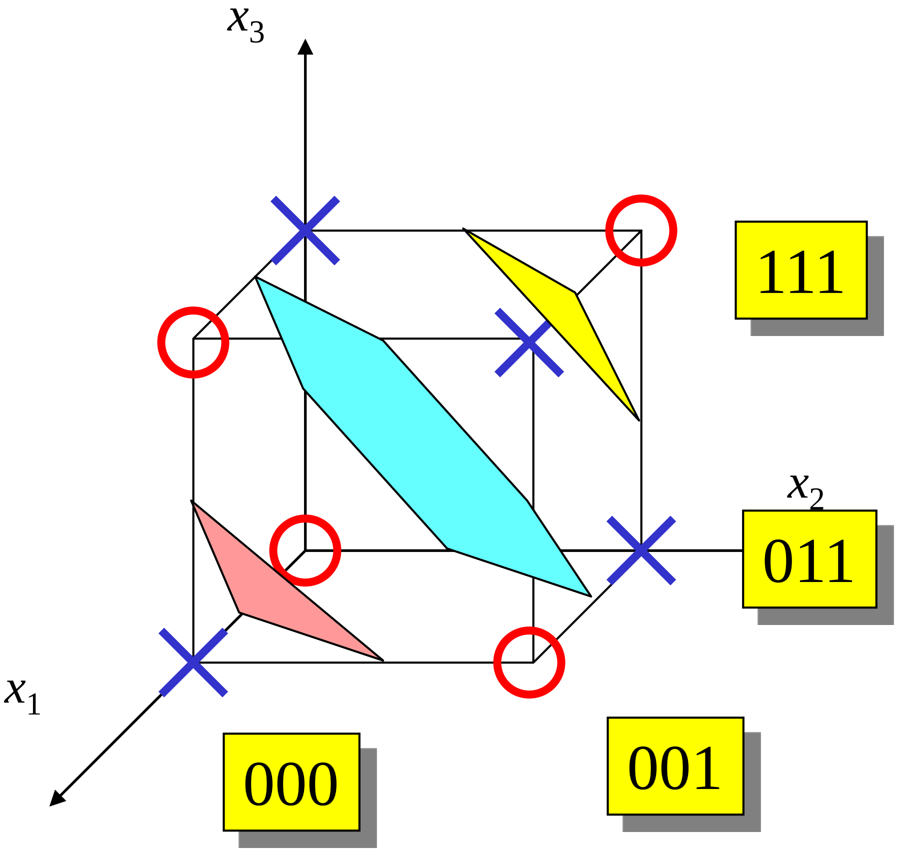
$x_1 x_2 x_3$	
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1



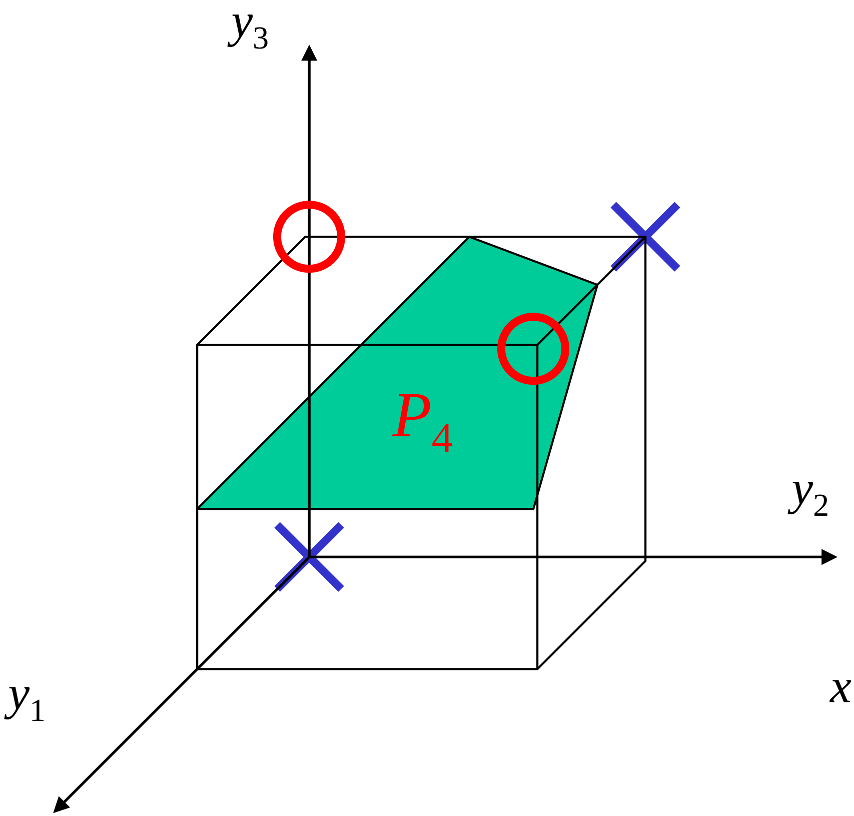
Parity Problem

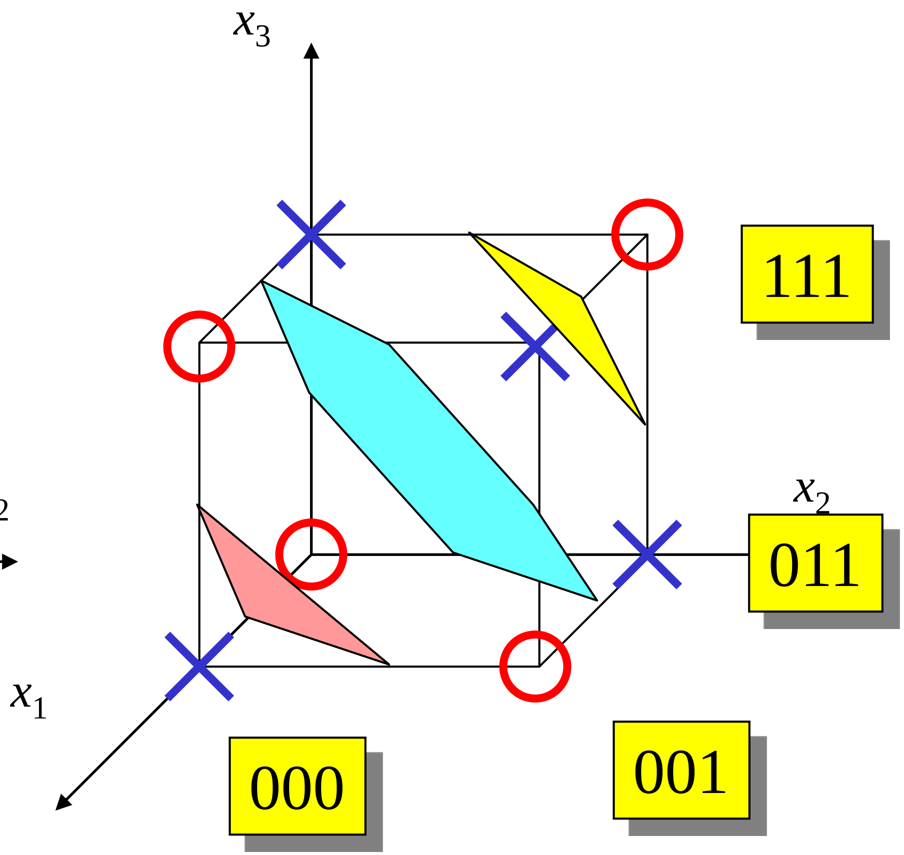
Is the problem linearly separable?



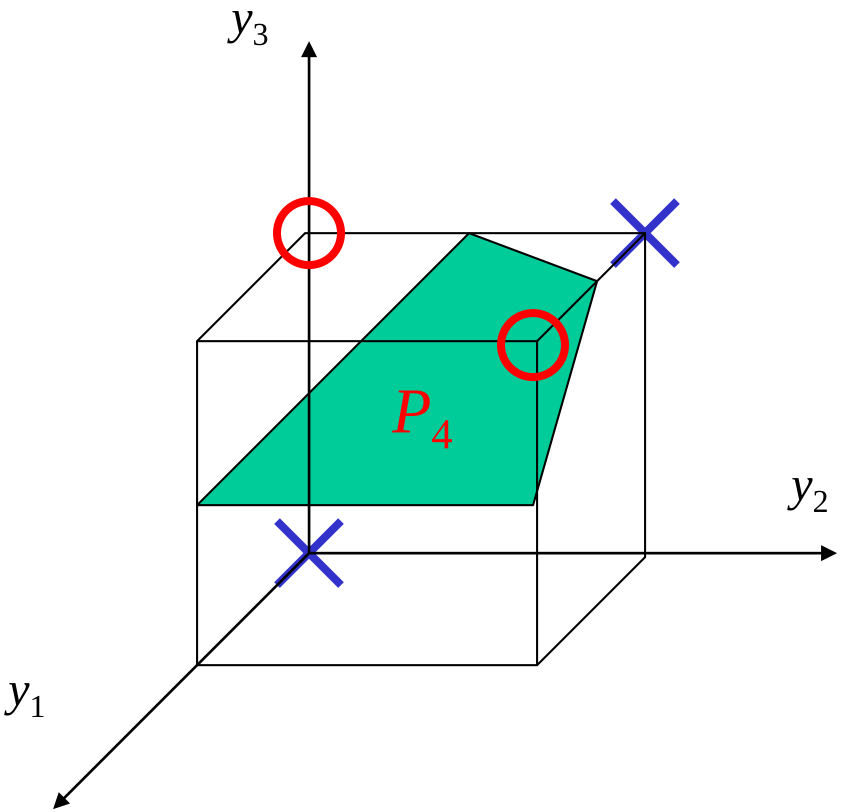


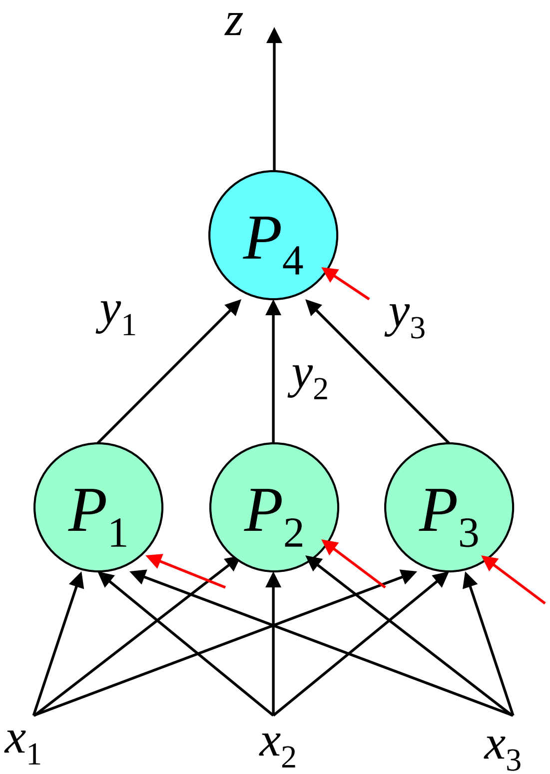
Parity Problem





Parity Problem

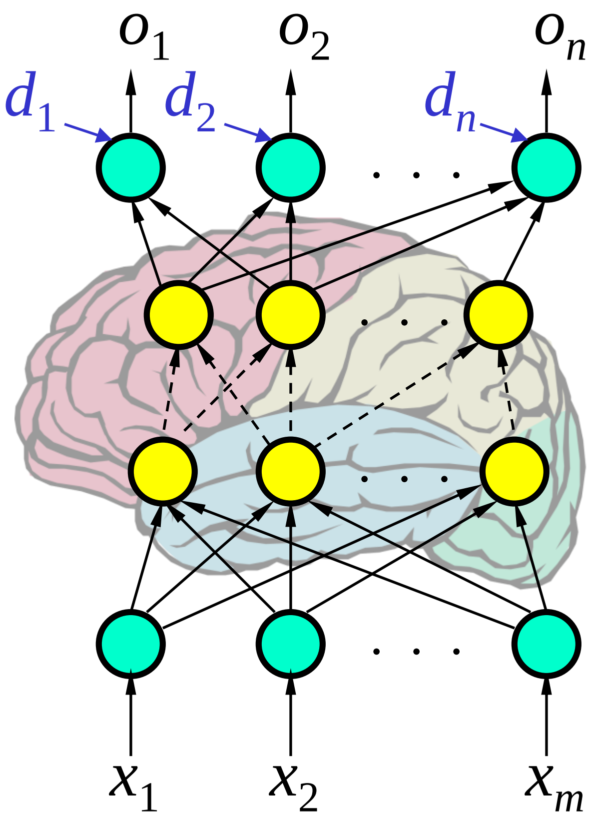




Back Propagation Learning

MIMA

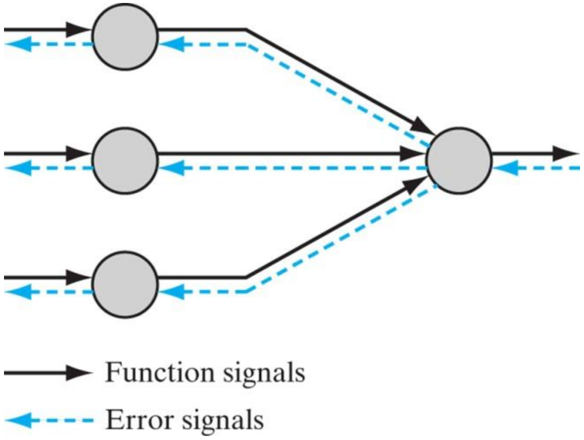
- Learning on Output Neurons
- Learning on Hidden Neurons



- General Learning Rule
  - Measure error
  - Reduce that error
    - By appropriately adjusting each of the weights in the network

Back Propagation Learning

- Forward Pass:
  - Error is calculated from outputs
  - Used to update output weights
- Backward Pass:
  - Error at hidden nodes is calculated by back propagating the error at the outputs through the new weights
  - Hidden weights updated



Symbols

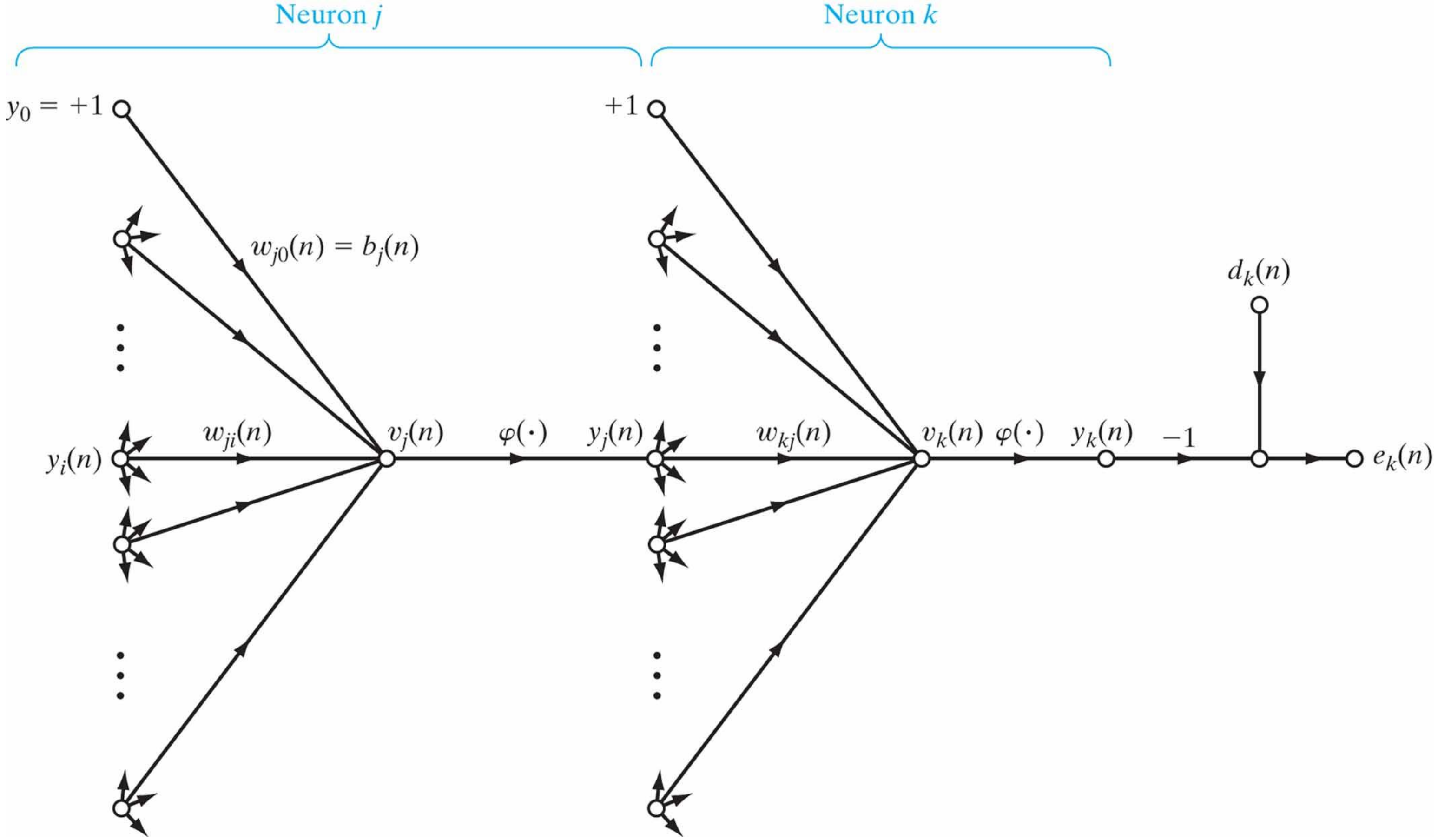
- Subscript i, j, k, represent different neurons, when j is the neuron of hidden layer, then i is on the left side of j, and k is on the right side of j.
- n is the iteration no.
- $E(n)$ is the sum of instantaneous error energy of the n_{th} iteration, its average is E_{av}.
- $e_j(n)$ is the error of the j_{th} neuron on the n_{th} iteration.
- $d_j(n)$ is the expected value of the j_{th} neuron on the n_{th} iteration.

Symbols

- $y_j(n)$ is output of the j_{th} neuron on the n_{th} iteration; if the j_{th} neuron is the output layer, the $O_j(n)$ can be used.
- $w_{ji}(n)$ is the weight from i to j, its change is $\Delta w_{ji}(n)$.
- $v_j(n)$ is the internal state of the j_{th} neuron.
- $\varphi(.)$ is the activation function of the j_{th} neuron.
- θ_j is the threshold of the j_{th} neuron.
- $x_i(n)$ is the i_{th} element of the input sample.
- η is the learning rate.

Symbols

- Signal-flow graph highlighting the details of output neuron *k* connected to hidden neuron *j*.



Back Propagation Learning

MIMA

- The error function of the jth neuron in output layer is:
- $$e_j(n) = d_j(n) - y_j(n)$$ BP-1
- Instantaneous error $E(n)$ is defined as:
- $$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$ BP-2
- E_{av} is defined as (N is the number of training samples):
- $$E_{av} = \frac{1}{N} \sum_{n=1}^{N} E(n)$$ BP-3

Batch vs. On-Line Learning

- In the batch learning, adjustments to synaptic weights of the multilayer perceptron are performed after the presentation of all the N training samples.
- This training process that all the N samples are represented one time is called one epoch of training.
- So the cost function for batch learning is defined by the average error energy E_{av}.
- Advantages
  - Accurate estimation the gradient vector
  - Parallelization
- Disadvantage
  - More storage requirements

BP Learning Details

- At the n_{th} iteration, we can training the network by minimizing E(n), and the output of the j_{th} neuron is given by:
- $$v_j(n) = \sum_{i=0}^{P} w_{ji}(n) y_i(n) \qquad \text{BP-4}$$
- $$y_j(n) = \varphi_j(v_j(n)) \qquad \text{BP-5}$$

BP Learning Details

MIMA

- We define the gradient as:

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$e_j(n) = d_j(n) - y_j(n)$$

- According to BP-4

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = \frac{\partial}{\partial w_{ji}(n)} \left[\sum_{i=0}^{p} w_{ji}(n) y_i(n) \right] = y_i(n)$$

- We further denote

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)}$$

BP Learning Details

- Then the change of $w_{ji}(n)$ is :

$$\Delta w_{ji}(n) = \eta \delta_j(n) \cdot y_i(n)$$

- η is the learning rate, thus the weights can be updated as:

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n) = w_{ji}(n) + \eta \delta_j(n) y_i(n)$$

BP Learning Details

MIMA

- When neuron *j* is a neuron in the output layer, according to BP-1, BP-1, we have:

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)}$$

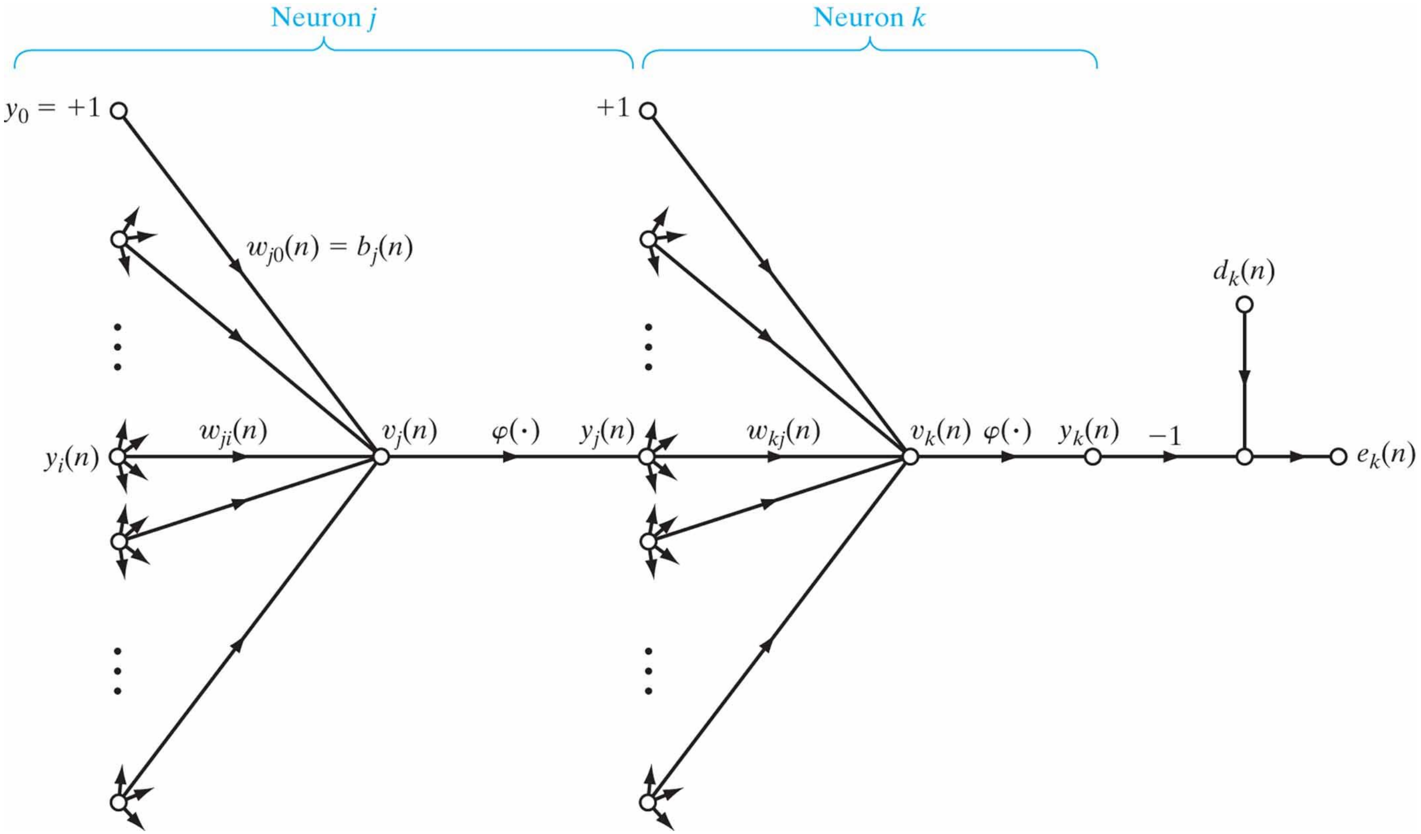
$$= -\frac{\partial}{\partial y_j(n)}\left[\frac{1}{2}\sum_{j\in c}\left(d_j(n) - y_j(n)\right)^2\right]\cdot\frac{\partial\left(\varphi\left(v_j(n)\right)\right)}{\partial v_j(n)}$$

$$= \left(d_j(n) - y_j(n)\right)\varphi_j'\left(v_j(n)\right)$$

$$E(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n) \qquad e_j(n) = d_j(n) - y_j(n)$$

BP Learning Details

- When neuron *j* is in the hidden layer, there is no expected value for us to use. Thus, we use the error propagated from the neuron connected to it:



BP Learning Details

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\varphi'\left(v_j(n)\right)$$

- *j* and *k* connected with w_{kj}

$$\Delta w_{ji}(n) = \eta\delta_j(n)\cdot y_i(n)$$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k \frac{\partial E(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)} = \sum_k \frac{\partial E(n)}{\partial v_k(n)} w_{kj}(n)$$

- If *k* is in the output layer,

$$\frac{\partial E(n)}{\partial v_k(n)} = -\delta_k(n) = -\left(d_k(n) - y_k(n)\right)\varphi'\left(v_k(n)\right)$$

- Then for neuron *j*,

$$\begin{aligned}\delta_j(n) &= \varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n)\\ &= \varphi_j'\left(v_j(n)\right)\sum_k \left(d_k(n) - y_k(n)\right)\varphi'\left(v_k(n)\right) w_{kj}(n)\end{aligned}$$

BP Learning Details

- The previous equations show that we need a function $\varphi(.)$ differentiable, e.g., the sigmoid function:

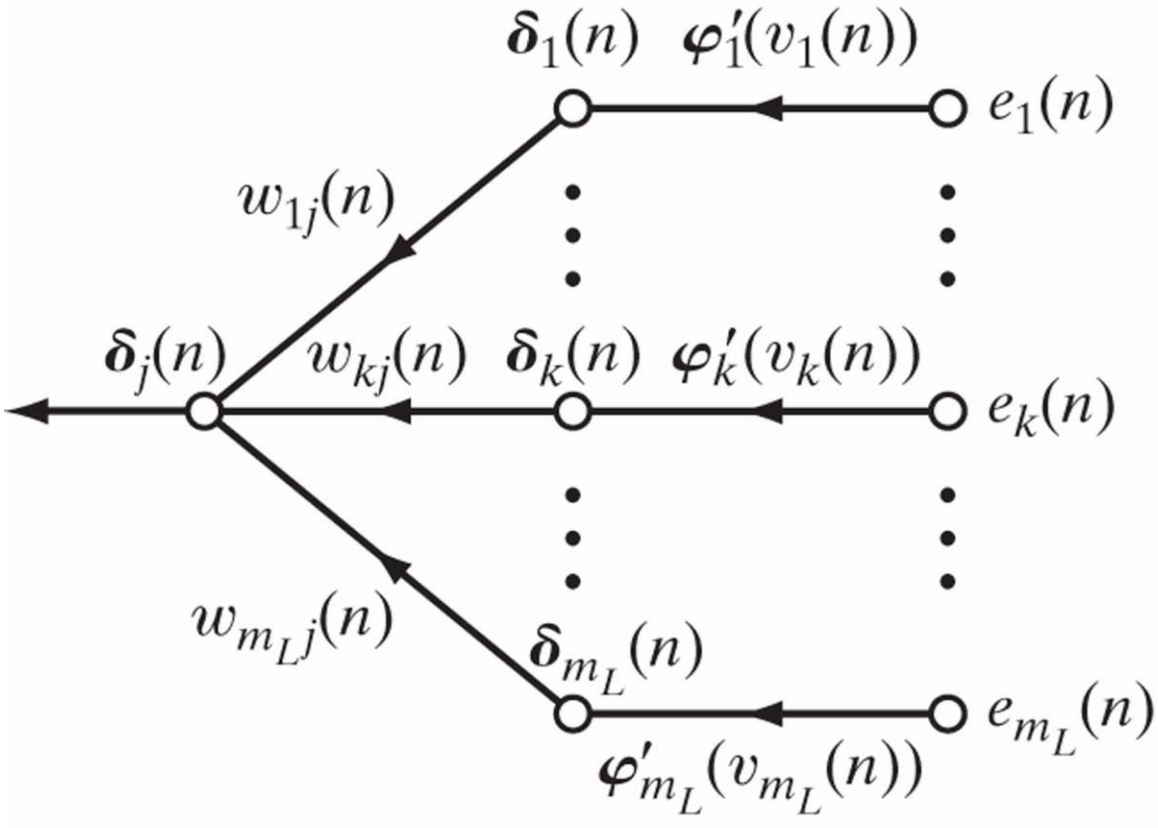
$$y_j(n) = \varphi\left(v_j(n)\right) = \frac{1}{1+\exp\left(-v_j(n)\right)}, -\infty < v_j(n) < \infty$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'\left(v_j(n)\right) = \frac{\exp\left(-v_j(n)\right)}{\left[1+\exp\left(-v_j(n)\right)\right]^2}$$

$$\varphi'\left(v_j(n)\right) = y_j(n)[1-y_j(n)]$$

BP Learning Details

- Signal-flow graph of a part of the adjoint system pertaining to back-propagation of error signals.



Two Passes of Computation

- Forward pass

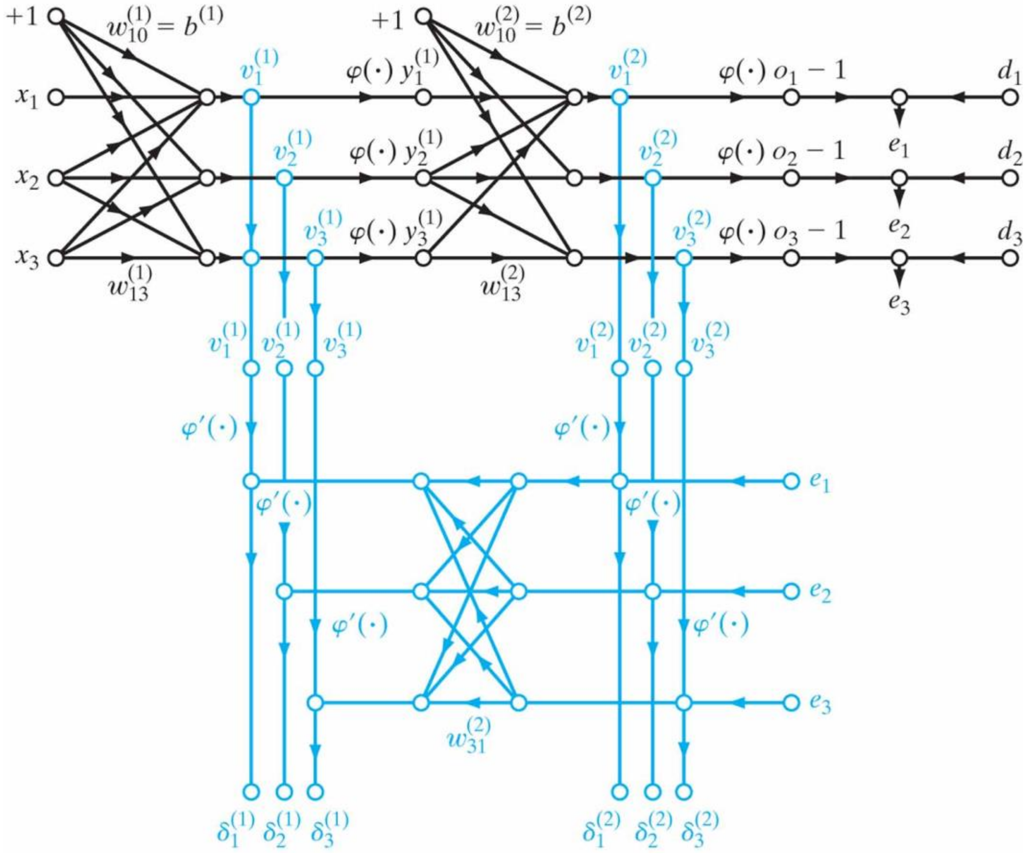
$$v_j(n) = \sum_{i=0}^{P} w_{ji}(n) y_i(n)$$

$$y_j(n) = \varphi_j(v_j(n))$$

- Backward pass
  - This pass stats at the output layer by passing the error signals leftward through the network, layer by layer, and recursively computing delta(local gradient) for each neuron.

Signal-flow graphical summary

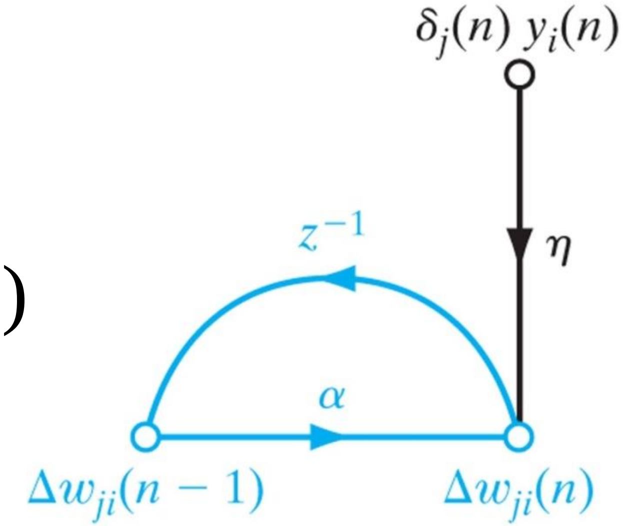
- Top part of the graph: forward pass.
  Bottom part of the graph: backward pass.



Learning Rate

- The learning rate should not be too large or too small.
- In order to avoid the danger of instability, a momentum term can be introduced into the equation.

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$



Learning Rate

- If we rewrite the equation as a time series with index t, the equation becomes:

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-1} \delta_j(t) y_i(t)$$

We can rewrite it as:

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-1} \frac{\partial E(t)}{\partial w_{ji}(t)}$$

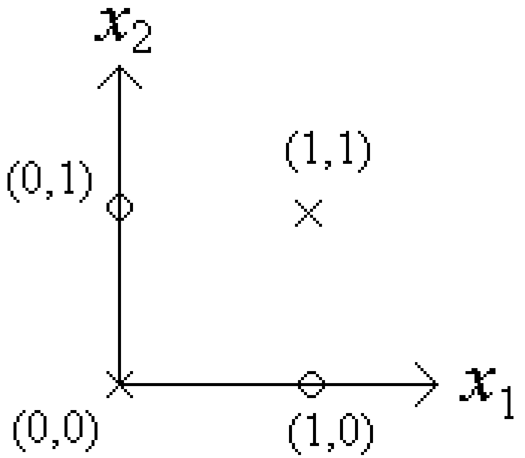
For the time series to be convergent, 0<=|alpha|<1

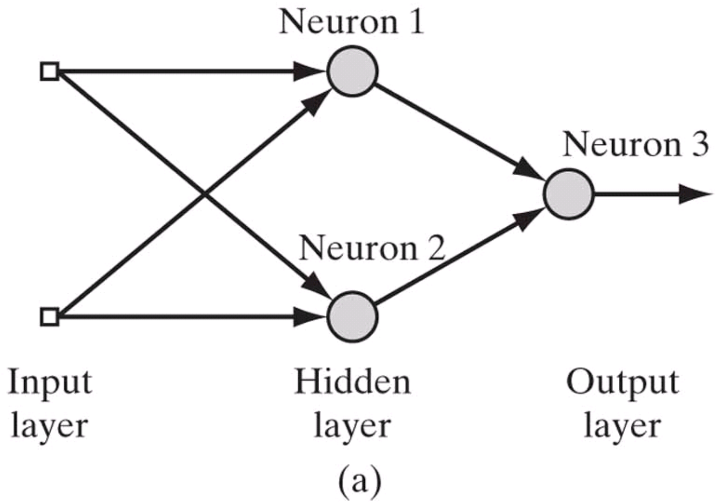
The sign of the partial derivative can affect the speed and stability.

Stopping Criteria

- In general, the BP cannot be shown to converge, and there are no well-defined criteria for stopping its operation.
- However, there are some reasonable criteria that can be used to terminate the weight adjustments, e.g.
  - When the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.
  - When the average squared error per epoch is sufficiently small. Usually, it is in the range of 0.1 to 1 percent per epoch, or as small as 0.01 percent.

XOR Problem Revisiting





XOR Problem Revisiting

- The weights are initialized as:

$$\underline{w}_1(0) = (-1.2,1,1)^T, \underline{w}_2(0) = (0.3,1,1)^T, \underline{w}_3(0) = (0.5,0.4,0.8)^T$$

- η=0. 5
- When the sample (1,1) is given to the network:

$$\begin{cases} y_1 = \dfrac{1}{1+\exp\left[-\left((-1.2)\times(-1)+1\times1+1\times1\right)\right]} = 0.96 \\ y_2 = \dfrac{1}{1+\exp\left[-\left(0.3\times(-1)+1\times1+1\times1\right)\right]} = 0.84 \\ z = \dfrac{1}{1+\exp\left[-\left(0.5\times(-1)+0.4\times0.96+0.8\times0.84\right)\right]} = 0.63 \end{cases}$$

XOR Problem Revisiting

- We have:

$$\delta_j(n) = e_j(n)\varphi'\left(v_j(n)\right)$$
$$= \left(d_j(n) - O_j(n)\right)O_j(n)\left(1 - O_j(n)\right)$$

$$\delta_3 = (0 - 0.63) \times 0.63 \times (1 - 0.63) = -0.147$$

$$\delta_j(n) = \varphi'\left(v_j(n)\right)\sum_k \delta_k(n)w_{kj}(n)$$
$$= y_j(n)\left(1 - y_j(n)\right)\sum_k \delta_k(n)w_{kj}(n)$$

$$\delta_1 = 0.96 \times (1 - 0.96) \times (-0.147) \times 0.4 = -0.002$$

$$\delta_2 = 0.84 \times (1 - 0.84) \times (-0.147) \times 0.8 = -0.0158$$

XOR Problem Revisiting

- Then the weights are updated as:

$$\underline{w}_1(1) = (-1.2,1,1)^T + 0.5 \times (-0.0002)(-1,1,1)^T = (-1.199,0.999,0.999)^T$$

$$\underline{w}_2(1) = (0.3,1,1)^T + 0.5 \times (-0.0158)(-1,1,1)^T = (0.3079,0.992,0.992)^T$$

$$\underline{w}_3(1) = (0.5,0.4,0.8)^T + 0.5 \times (-0.147)(-1,0.96,0.84)^T = (0.5735,0.329,0.738)^T$$

- Finally, we can have:

$$\underline{w}_1(1) = (-1.198,0.912,1.179)^T$$

$$\underline{w}_2(1) = (0.294,0.826,0.98-)^T$$

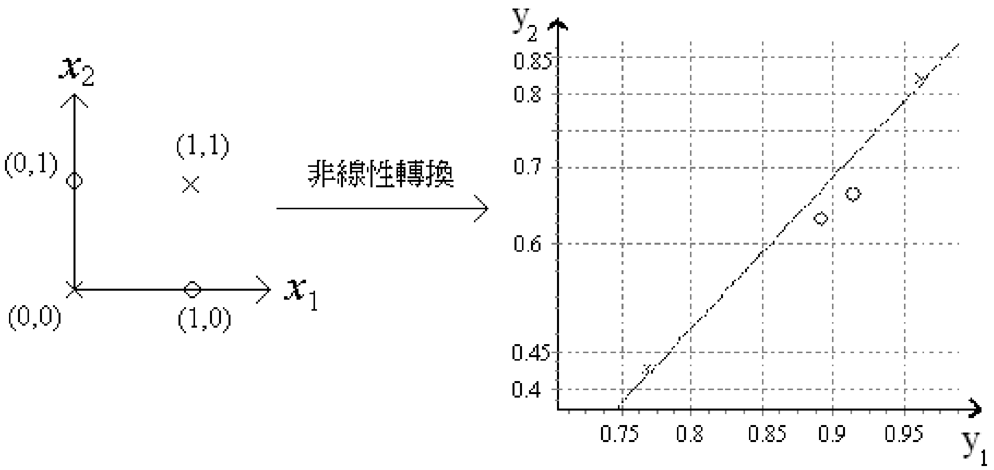
$$\underline{w}_3(1) = (0.216,0.384,-0.189)^T$$

XOR Problem Revisiting

- We can revisit the problem from the view of space transformation, the points in the sample space are transformed into a new space, i.e., (0,0), (1,1), (1,0) and (0,1)are mapped to (0.768,0.427), (0.964,0.819), (0.892,0.629) and (0.915,0.665).

$$\begin{cases} y_1 = \dfrac{1}{1+\exp[-(w_{11}x_1 + w_{12}x_2 - \theta_1)]} \\ y_2 = \dfrac{1}{1+\exp[-(w_{21}x_1 + w_{22}x_2 - \theta_2)]} \end{cases}$$

XOR Problem Revisiting



$$(0,0)\xrightarrow{\text{hiddenlayer}}(0.768,0.427)\xrightarrow{\text{outputlayer}}z=0.4997$$

$$(1,1)\xrightarrow{\text{hiddenlayer}}(0.964,0.819)\xrightarrow{\text{outputlayer}}z=0.4999$$

$$(0,1)\xrightarrow{\text{hiddenlayer}}(0.892,0.629)\xrightarrow{\text{outputlayer}}z=0.5025$$

$$(1,0)\xrightarrow{\text{hiddenlayer}}(0.915,0.665)\xrightarrow{\text{outputlayer}}z=0.5020$$

Other Issues

- Heuristics for BP learning
  - Stochastic versus batch update
  - Maximizing information content
  - Activation function
  - Target values
  - Normalizing the inputs
  - Initialization
  - Learning from hints
  - Learning rates.

Stochastic versus batch update

- The stochastic mode (pattern-by-pattern) is computationally faster than batch mode.
- Especially, when the data is large and redundant, it will much better to use stochastic than to use batch.

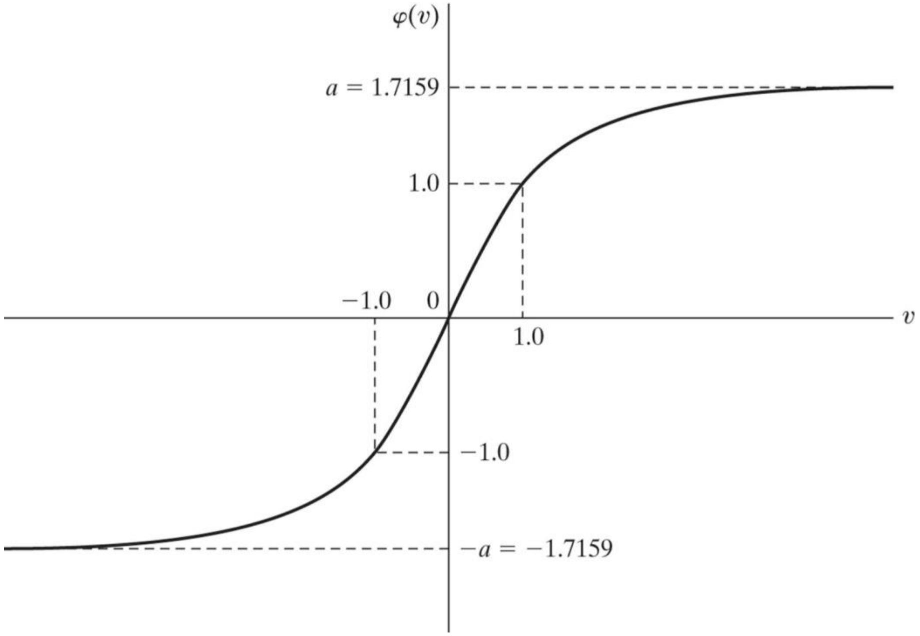
Maximizing information content

- Every training example should be chosen on the basis that its information content is the largest possible for the task at hand.
- How to choose?
  - Use an sample that results in the largest training error.
  - Use an example that radically different from all those previous used.

Activation function

- Graph of the hyperbolic tangent function φ(*v*) =*α* tanh(*bv*) for *α* 1.7159 and *b* = 2/3.The recommended target values are +1 and –1.

$$\varphi(x) = a\frac{e^{bx} - e^{-bx}}{e^{bx} + e^{-bx}}$$



Target values

- The target values should be within the range of the sigmoid activation function.
- Otherwise, the BP algorithm tends to drive the free parameters of the network to infinity, and thereby slow down the learning process by driving the hidden neurons into saturation.

$$d_j = a - \varepsilon$$

$$d_j = -a + \varepsilon$$

Normalizing the inputs

- Each input variable should be preprocessed
  - Mean removal
  - Decorrelation
  - Covariance equalization
- Normalization methods
  - Min-Max

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

  - Z-score standardization

$$z = \frac{x - \mu}{\sigma}$$

Initialization

- Too large
  - The neurons in the network will be driven into saturation
- Too small
  - The BP algorithm will operate on a very flat area around the origin of the error surface.

Learning from hints

- We can make use of some information that we have about the activation function or data.

Learning rates

- The learning rate should be assigned a smaller value in the last layers than in the front layers.
- Neurons with many inputs should have a smaller learning rate than neurons with few inputs.
- Annealing method can be applied.

Stopping Criteria

- In general, the BP cannot be shown to converge, and there are no well-defined criteria for stopping its operation.
- However, there are some reasonable criteria that can be used to terminate the weight adjustments, e.g.
  - When the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.
  - When the average squared error per epoch is sufficiently small. Usually, it is in the range of 0.1 to 1 percent per epoch, or as small as 0.01 percent.

Some problems

- The layers
- The number of hidden layer neurons
  - Kolmogorov theorem: the neurons in hidden layers can be: s=2m+1 (m is the number of neurons in input layer)

- Demos

BP Summary

- **Strengths of BP learning**
    - Great representation power
    - Wide practical applicability
    - Easy to implement
    - Good generalization power
- **Problems of BP learning**
    - Learning often takes a long time to converge
    - The net is essentially a black box?
    - Gradient descent approach only guarantees a local minimum error
    - Not every function that is representable can be learned
    - Generalization is not guaranteed even if the error is reduced to zero

BP Summary

- No well-founded way to assess the quality of BP learning
- Network paralysis may occur (learning is stopped)
- Selection of learning parameters can only be done by trial-and-error
- BP learning is non-incremental (to include new training samples, the network must be re-trained with all old and new samples)

Radial-Basis Functions

- A **radial basis function (RBF)** is a real-valued function whose value depends only on the distance from the origin, so that ; or alternatively on the distance from some other point *c*, called a *center*

$$\varphi(x) = \varphi(\|x\|)$$

$$\varphi(x) = \varphi(\|x - c\|)$$

Interpolation problem

MIMA

- In its strict sense, the problem can be stated as:
  - Given a set of N different points $\{x_i \in R^{m0}\}$ and a corresponding set of N real numbers d_i, find a function F that satisfies the interpolation condition: $F(x_i)=d_i$.
- The Radial-Basis Function (RBF) technique consists of choosing a function F that has the form:

$$F(x) = \sum_{i=1}^{N} w_i \varphi(\| x - x_i \|)$$

How to get the solution?

Radial-Basis Function

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{N1} & \varphi_{N2} & \cdots & \varphi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix}$$

where

$$\varphi_{ij} = \varphi(\|x_j - x_i\|)$$

Micchelli theorem(1986) is proved that if the equation is as the above, then the matrix is nonsingular.

Radial-Basis Functions

- Multiquadrics

$$\varphi(r) = (r^2 + c^2)^{1/2}$$

- Inverse multiquadrics

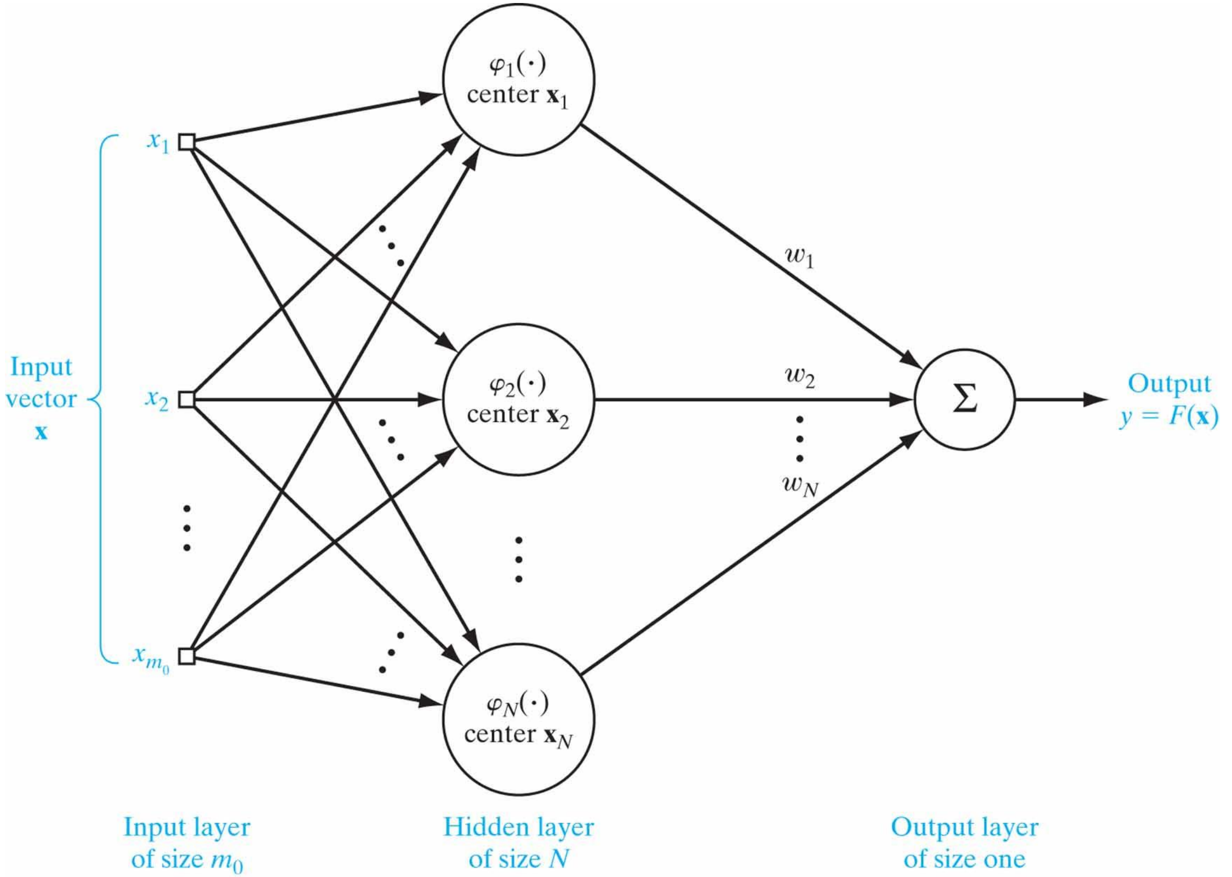
$$\varphi(r) = 1/(r^2 + c^2)^{1/2}$$

- Gaussian functions

$$\varphi(r) = \exp(-\frac{r^2}{2\sigma^2})$$

RBF Networks

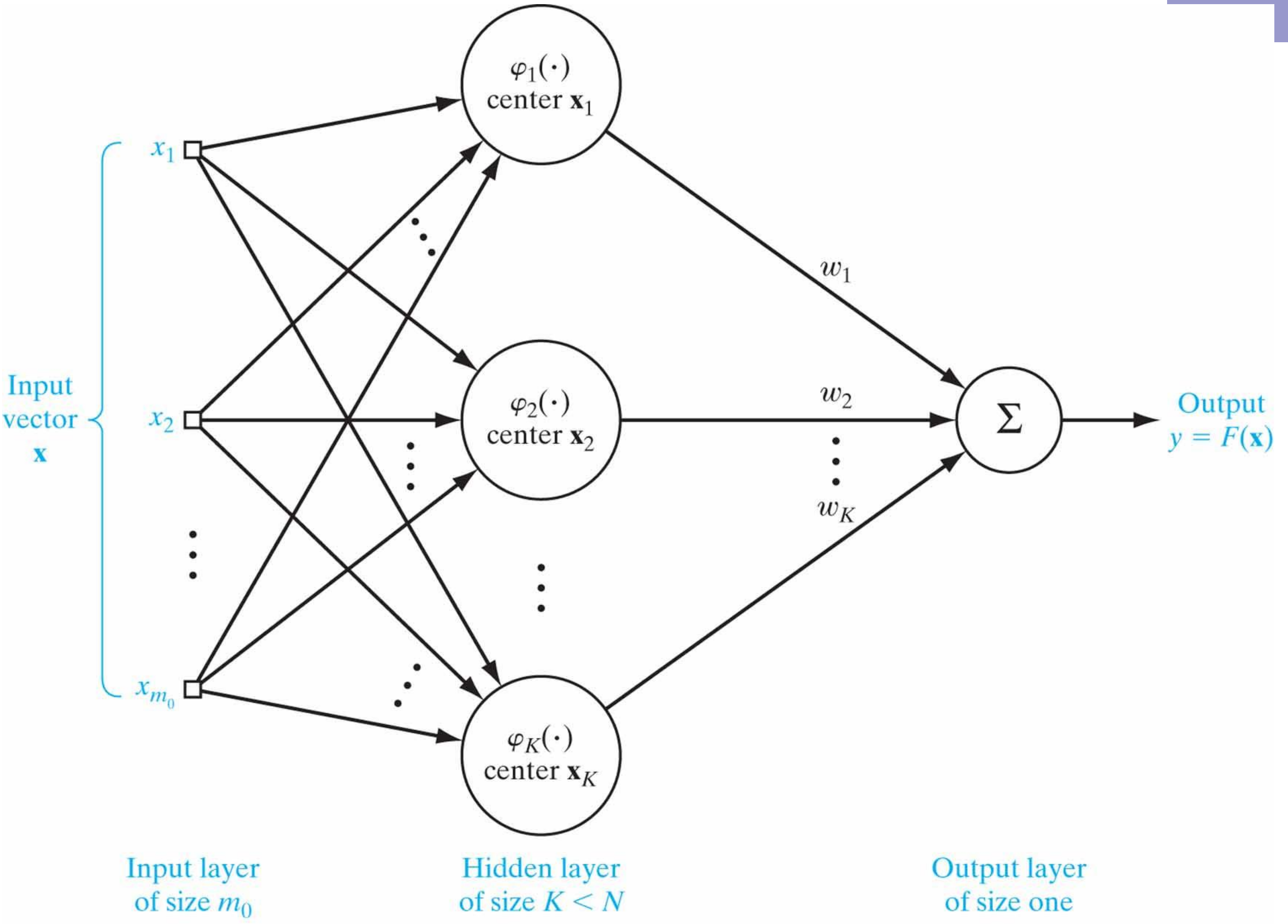
- Structure of an RBF network, based on interpolation theory.



RBF Networks

- The network has three layers:
  - Input layer, which consist of m0 source nodes.
  - Hidden layer, consist of the same number of computation units as the size of the training samples, namely, N.
  - Output layer, there is no restriction on the size of the output layer.

Modifications to RBF Networks



How to get the *k* centers

- This can be computed by un-supervised learning.
  - K-means
  - SOM
- Clustering algorithm can be used here, e.g. we use k-means

$$\min \sum_{j=i}^{K} \sum_{C(i)=j} \| x_i - u_j \|^2$$

Self-Organization Maps

- Teuvo Kohonen (1982, 1984)
- In biological systems
  - Cells tuned to similar orientations tend to be physically located in proximity with one another
  - Microelectrode studies with cats
- So, SOM is motivated by a distinct feature of the human brain:
  - The brain is organized in many places in such a way that different sensory inputs are represented by topologically ordered computation maps.

Self-Organization Maps

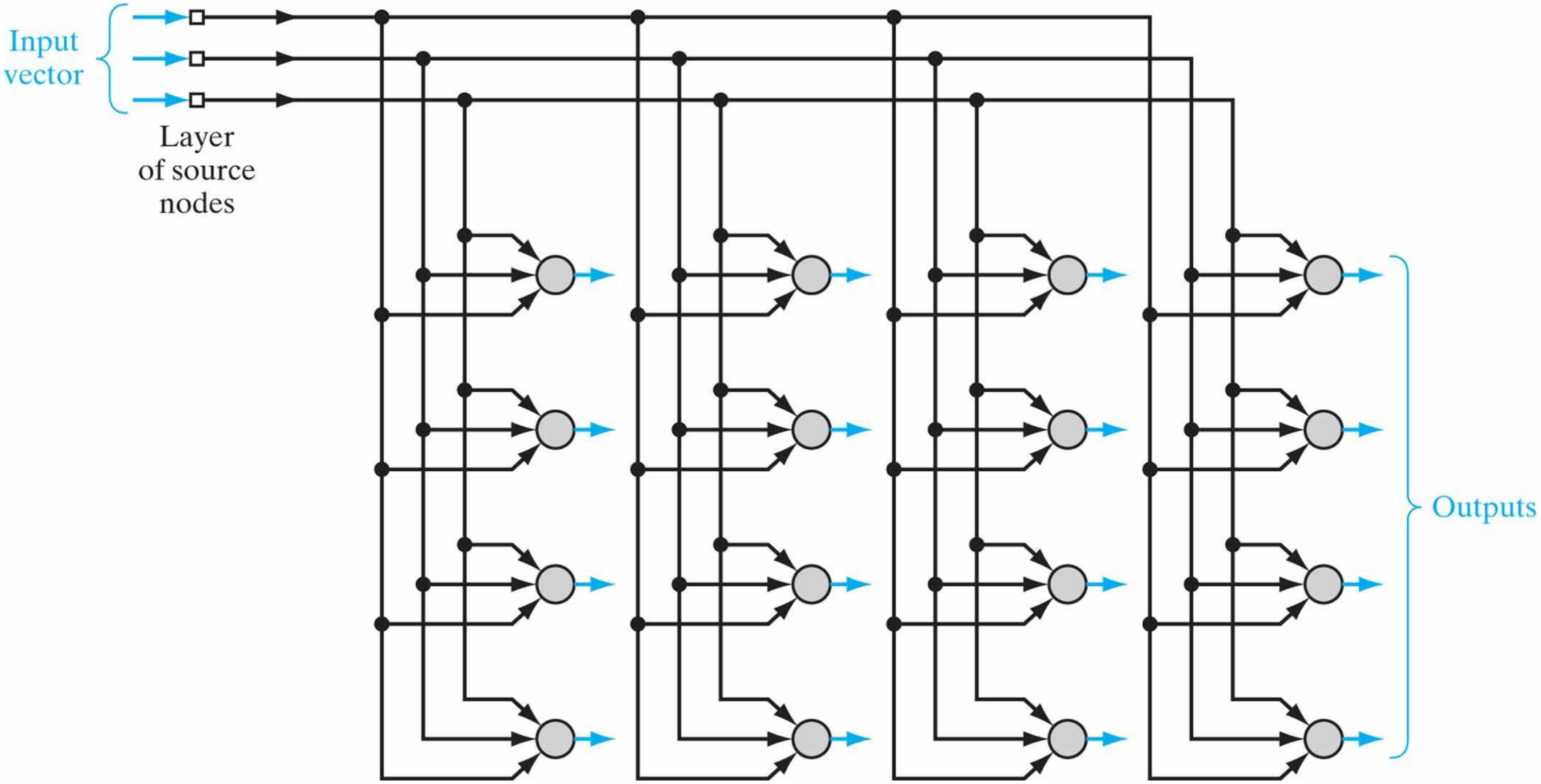
- Orientation tuning over the surface forms a kind of map with similar tunings being found close to each other
  - Topographic feature map
  - Train a network using competitive learning to create feature maps automatically

SOM Clustering

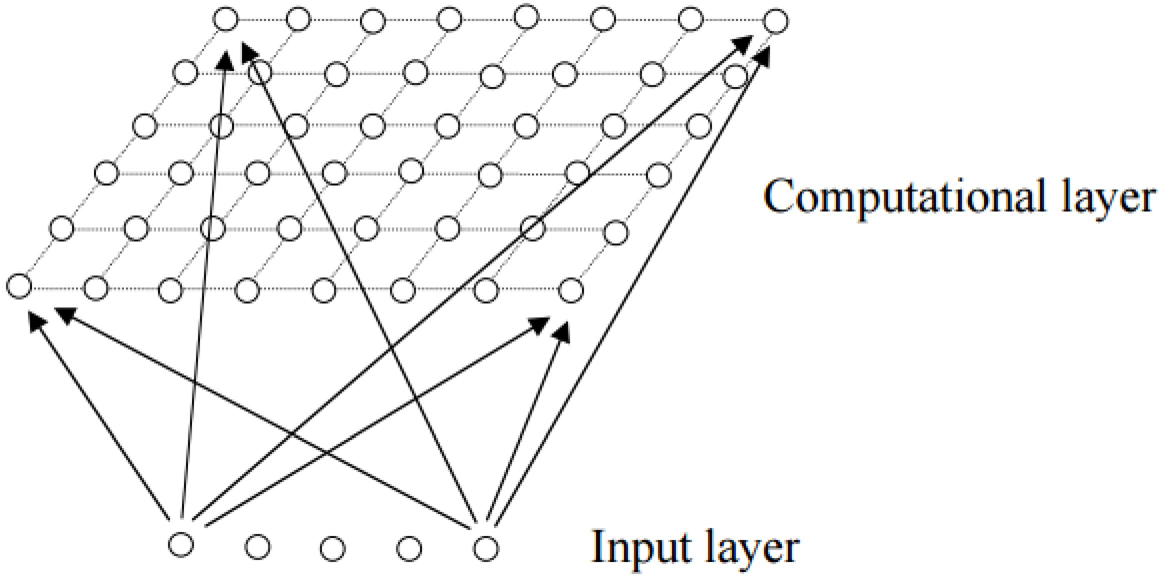
- Self-organizing map (SOM)
  - An unsupervised artificial neural network
  - Mapping high-dimensional data into a one or two-dimensional representation space
  - Similar data may be found in neighboring regions
- Disadvantages
  - Fixed size in terms of the number of units and their particular arrangement
  - Hierarchical relations between the input data are not mirrored in a straight-forward manner

SOM Structure

- Two-dimensional lattice of neurons, illustrated for a three-dimensional input and four-by-four dimensional output (all shown in blue).



SOM Structure



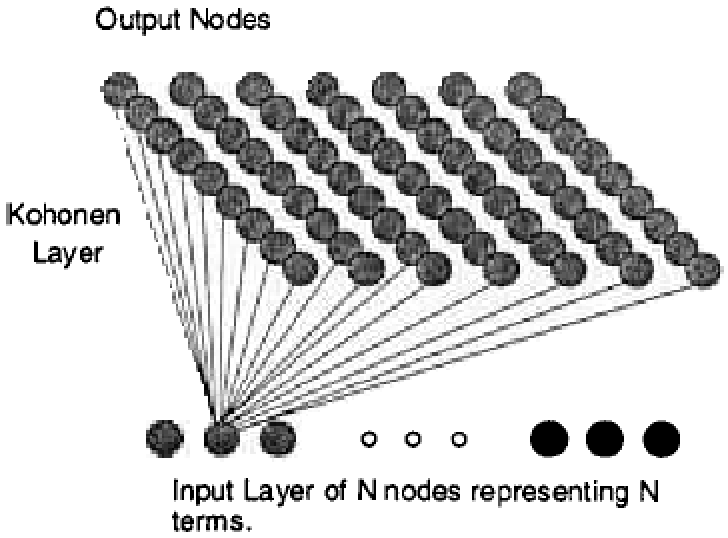
Features

- Kohonen's algorithm creates a vector quantizer by adjusting weight from common input nodes to M output nodes
- Continuous valued input vectors are presented without specifying the desired output
- After the learning, weight will be organized such that topologically close nodes are sensitive to inputs that are physically similar
- Output nodes will be ordered in a natural manner

Initial setup of SOM

MIMA

- Consists a set of units *i* in a two-dimension grid
- Each unit *i* is assigned a weight vector m_i as the same dimension as the input data
- The initial weight vector is assigned random values



Essential processes in SOM

- Competition process
  - Find the best match of the input vector x with the synaptic-weight vectors.
- Cooperation process
  - Decide the topological neighborhood centered on the winning neuron, and make it decay smoothly with lateral distance.
- Synaptic adaptation
  - The weights of corresponding neurons are updated in relation to the input vector.

Competition process

- Winner Selection
  - Initially, pick up a random input vector *x(t)*
  - Compute the unit *c* with the highest activity level (*the winner c(t)*) by Euclidean distance formula

$$x = [x_1, x_2, \ldots, x_m]^T$$

$$w_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T, \; j = 1, 2, \ldots, l$$

$$i(x) = \arg\min \| x - w_j \|, \; j = 1, 2, \ldots, l$$

Neuron i is called the best-matching, or winning neuron for the input vector

Cooperative process

- A neuron that is firing tends to excite the neurons in its immediate neighborhood more that those farther away from it.
- Let h_{ji} denote the topological neighborhood function centered on winning neuron i and encompassing a set of excited neurons.
- Let d_{ij} denote the lateral distance between the winning neuron i and excited neuron j.

Cooperative process

- h_{ji} and d_{ji} should satisfy two distinct requirements:
  - h_{ij} is symmetric about the maximum point defiend by d_{ij} = 0; In other words, it attains its maximum value at the winning neuron i for which the distance d_{ij} is zero.
  - The amplitude of the topological neighborhood h_{ij} decreases monotonically with increasing lateral distance d_{ij}, decaying to zero for d_{ij} ->∞, this is necessary condition for convergence.

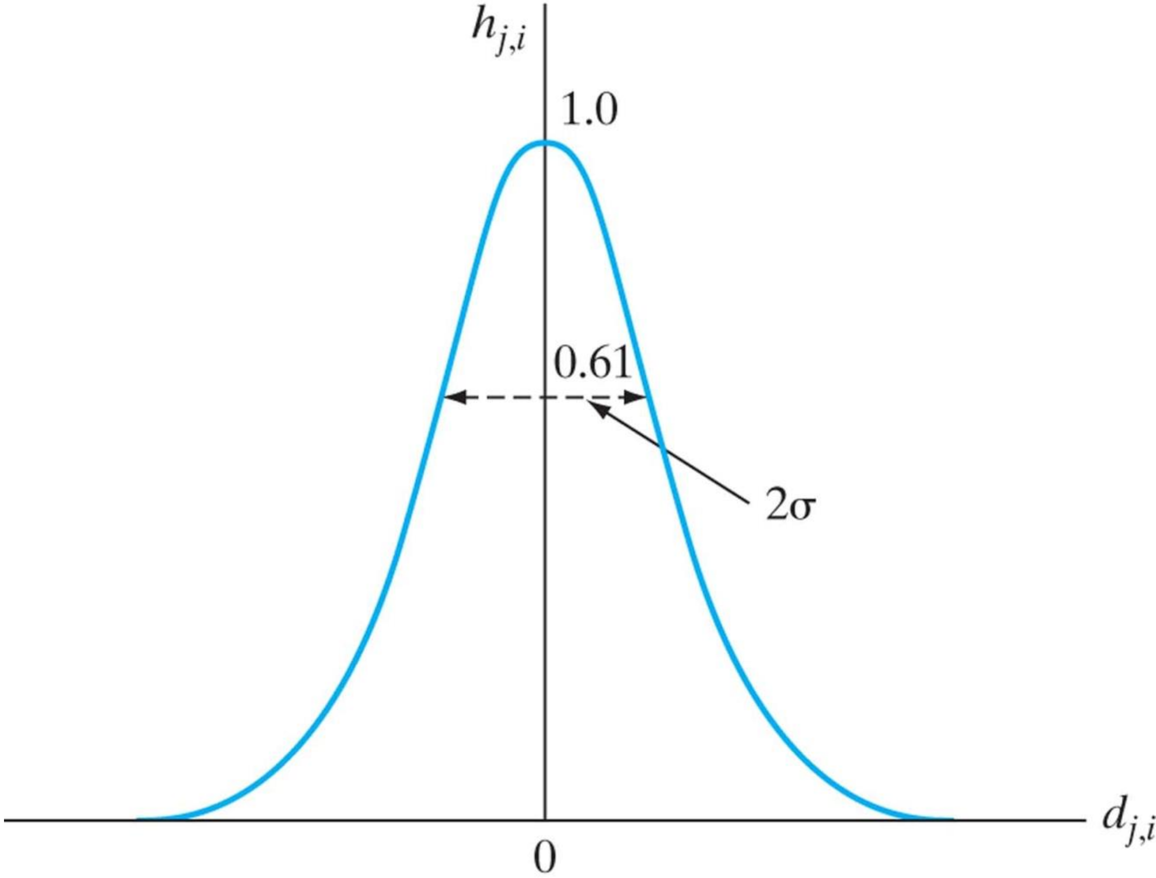
Cooperative process

- A good choice of h_{ij} that satisfies these requirements is the Gaussian function:

$$h_{ji} = \exp(-\frac{d_{ji}^2}{2\sigma^2})$$

$$d_{ji} = | j - i |$$

$$d_{ji} = \| r_j - r_i \|$$



Cooperative process

- Another unique feature of the SOM algorithm is that the size of the topological neighborhood is permitted to shrink with time.
- This requirement can be satisfied by making the width sigma of the topological neighborhood function h_{ji} decrease with time. A popular choice for the dependence of sigma on discrete time n is the exponential decay:

$$\sigma(n) = \sigma_0 \exp(-\frac{n}{\tau_1})$$

Cooperative process

- Correspondingly, the topological neighborhood function assumes a time-varying form of its own, as follows:

$$h_{ji}(n) = \exp\left(-\frac{d_{ji}^{2}}{2\sigma^{2}(n)}\right)$$

Adaptive process

MIMA

- In the stage, the synaptic-weight is required to change in relation to the input vector.

$$w_j(n+1) = w_j(n) + \Delta w_j(n)$$

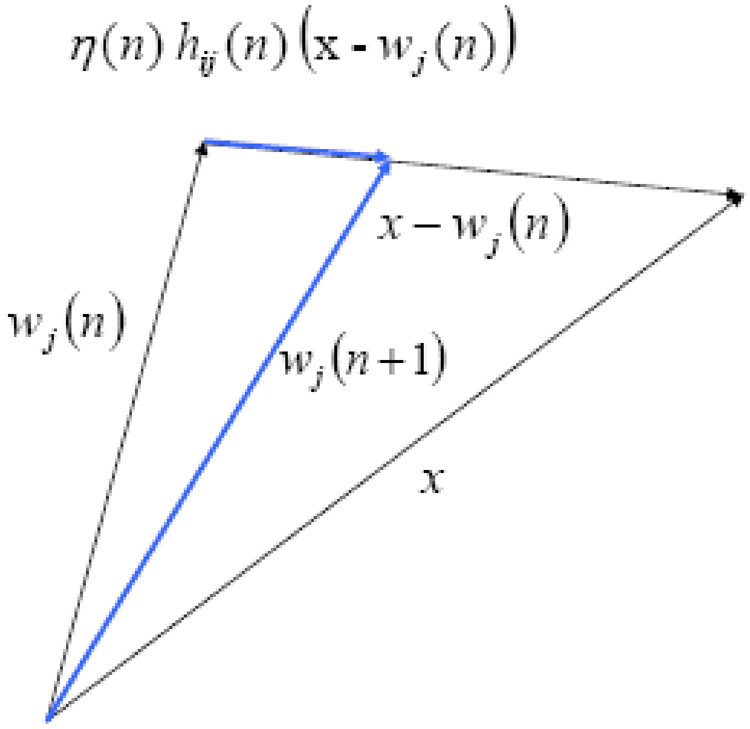
Where the last term can be calculated using the following equation:

$$\Delta w_j(n) = \eta(n) h_{ji}(n)(x(n) - w_j(n))$$

The learning rate η(n) should also be time varying.

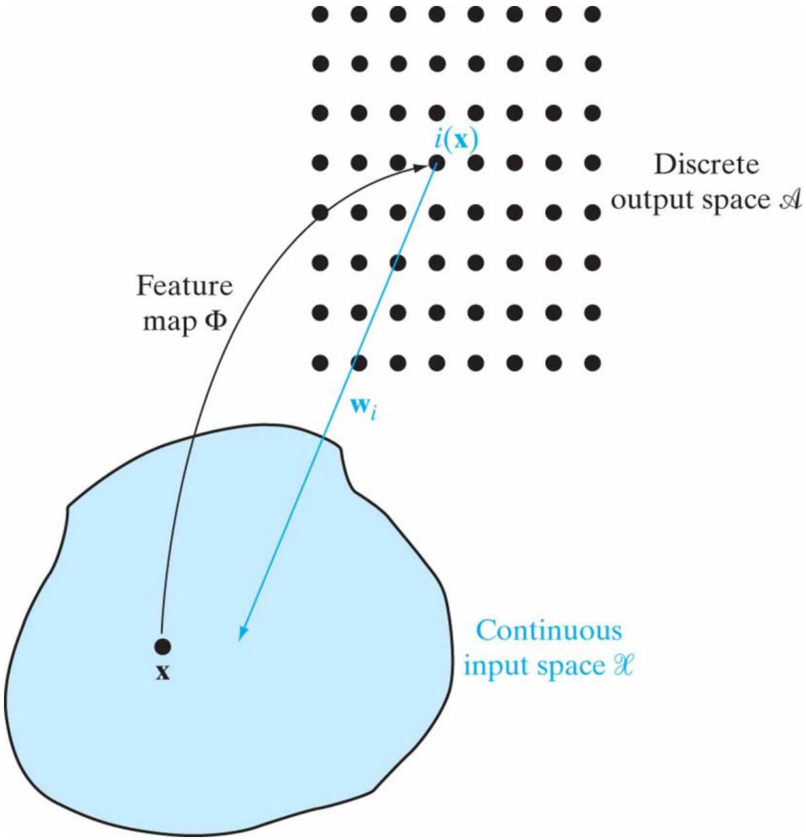
$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right)$$

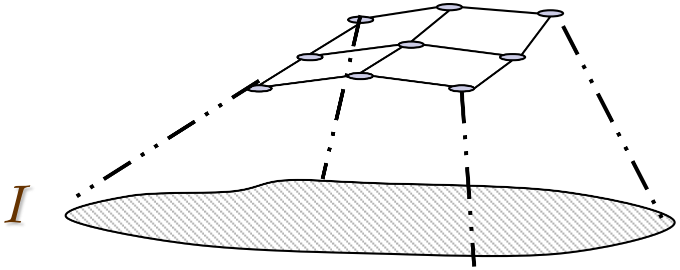
Weight update



Weight update

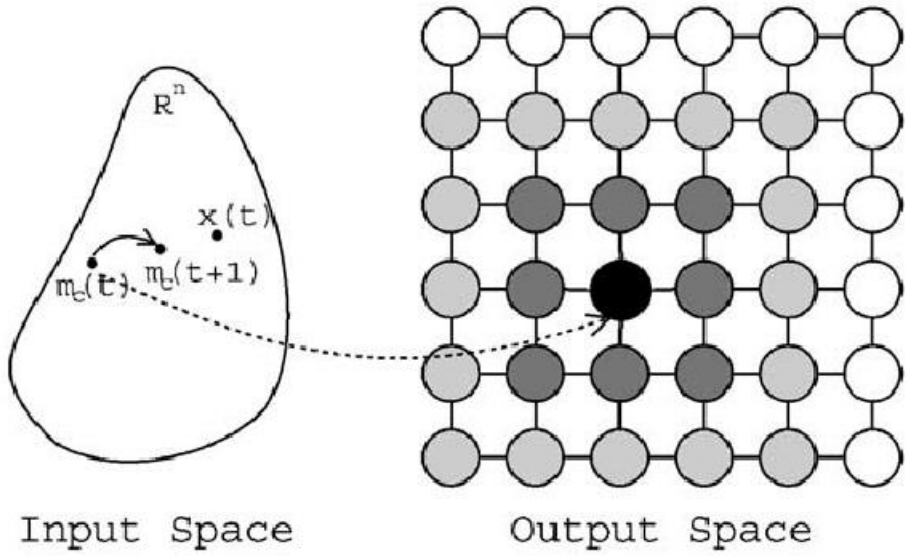
- Illustration of the relationship between feature map Φ and weight vector $\mathbf{w}_i$ of winning neuron *i*





Learning Process (Adaptation)

$$m_i(t+1) = m_i(t) + \alpha(t) \cdot h_{ci}(t) \cdot [x(t) - m_i(t)].$$



SOM Learning Summary

- 1. initialization
  - Choose random values for weights
  - Or choose input vectors randomly to initialize them
- 2.Sampling
  - Draw a sample from input space
- 3. similarity matching
  - Find the best-matching neuron

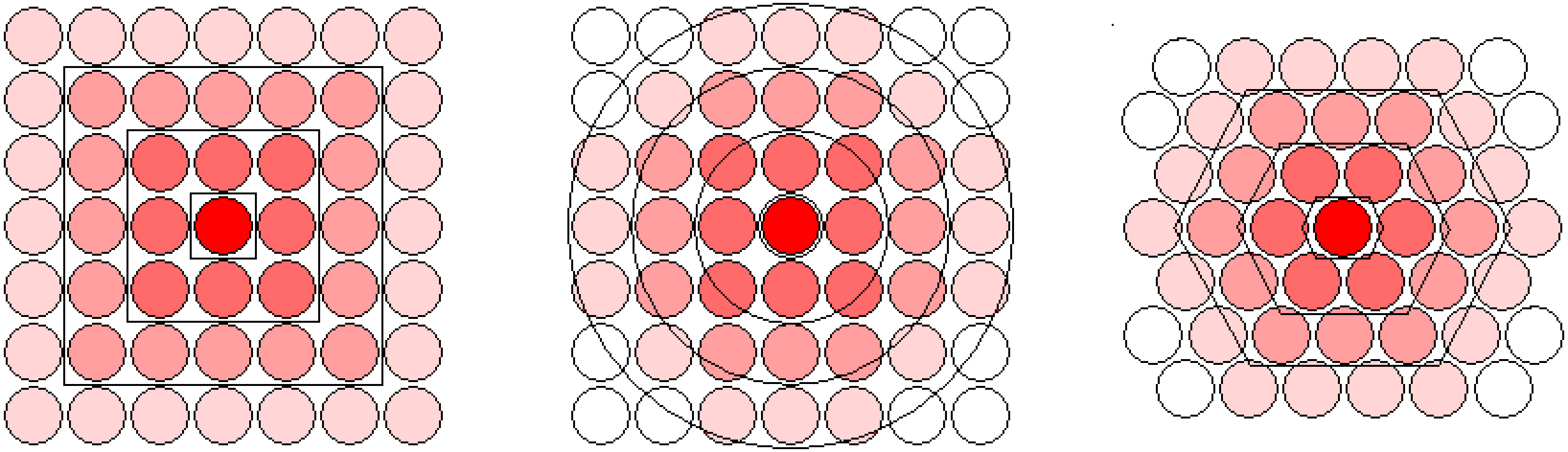
$$i(x) = \arg\min \| x - w_j \|, \; j = 1,2,\ldots,l$$

- 4.Updating

$$\Delta w_j(n) = \eta(n) h_{ji}(n)(x(n) - w_j(n))$$

- 5.Continuation

Neighborhoods

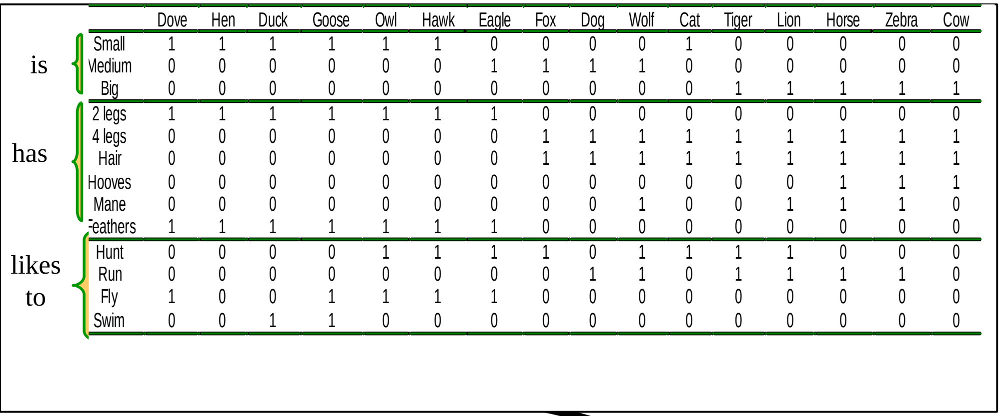


Applications

- Optimization problems
- Clustering problems
- Pattern recognition
- Others

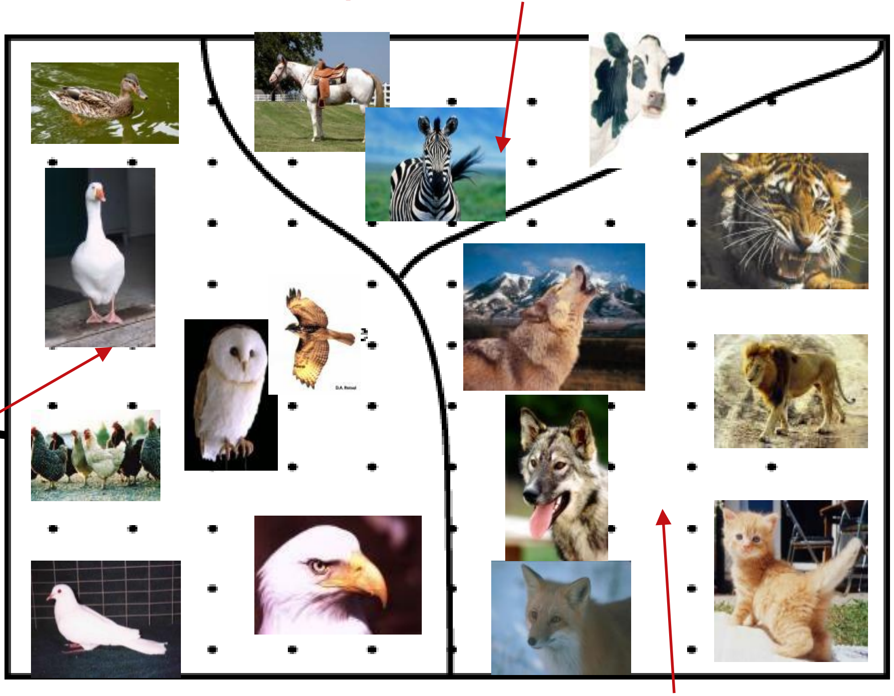
Applications

Animal names and their attributes



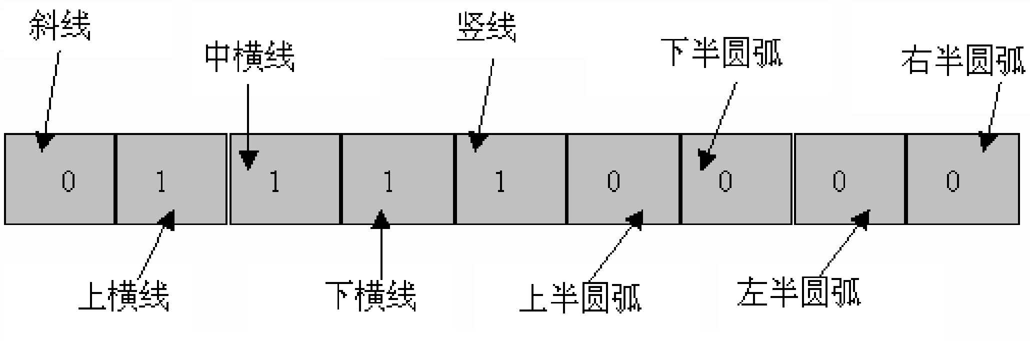
		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
is	Small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	Medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	Big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	Mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	Feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	Hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	Run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	Fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	Swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

A grouping according to similarity has emerged



Application to PR

- Feature



字母	特征向量								
	斜线	上横线	中横线	下横线	竖线	上半圆弧	下半圆弧	左半圆弧	右半圆弧
A	'\002'	‘\0’	‘\001’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’
B	‘\0’	‘\0’	‘\0’	‘\0’	‘\001’	‘\0’	‘\0’	‘\0’	‘\002’
C	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\001’	‘\0’
D	‘\0’	‘\0’	‘\0’	‘\0’	‘\001’	‘\0’	‘\0’	‘\0’	‘\001’
E	‘\0’	‘\001’	‘\001’	‘\001’	‘\001’	‘\0’	‘\0’	‘\0’	‘\0’
F	‘\0’	‘\001’	‘\001’	‘\0’	‘\001’	‘\0’	‘\0’	‘\0’	‘\0’
H	‘\0’	‘\001’	‘\0’	‘\0’	‘\002’	‘\0’	‘\0’	‘\0’	‘\0’
K	‘\002’	‘\0’	‘\0’	‘\0’	‘\001’	‘\0’	‘\0’	‘\0’	‘\0’
P	‘\0’	‘\0’	‘\0’	‘\0’	‘\001’	‘\0	‘\0’	‘\0’	‘\01’
T	‘\0’	‘\001’	‘\0’	‘\0’	‘\001’	‘\0’	‘\0’	‘\0’	‘\0’
U	‘\0’	‘\0’	‘\0’	‘\0’	‘\002’	‘\001’	‘\0’	‘\0’	‘\0’

Applications

- Demos

Thank You !

Any question?