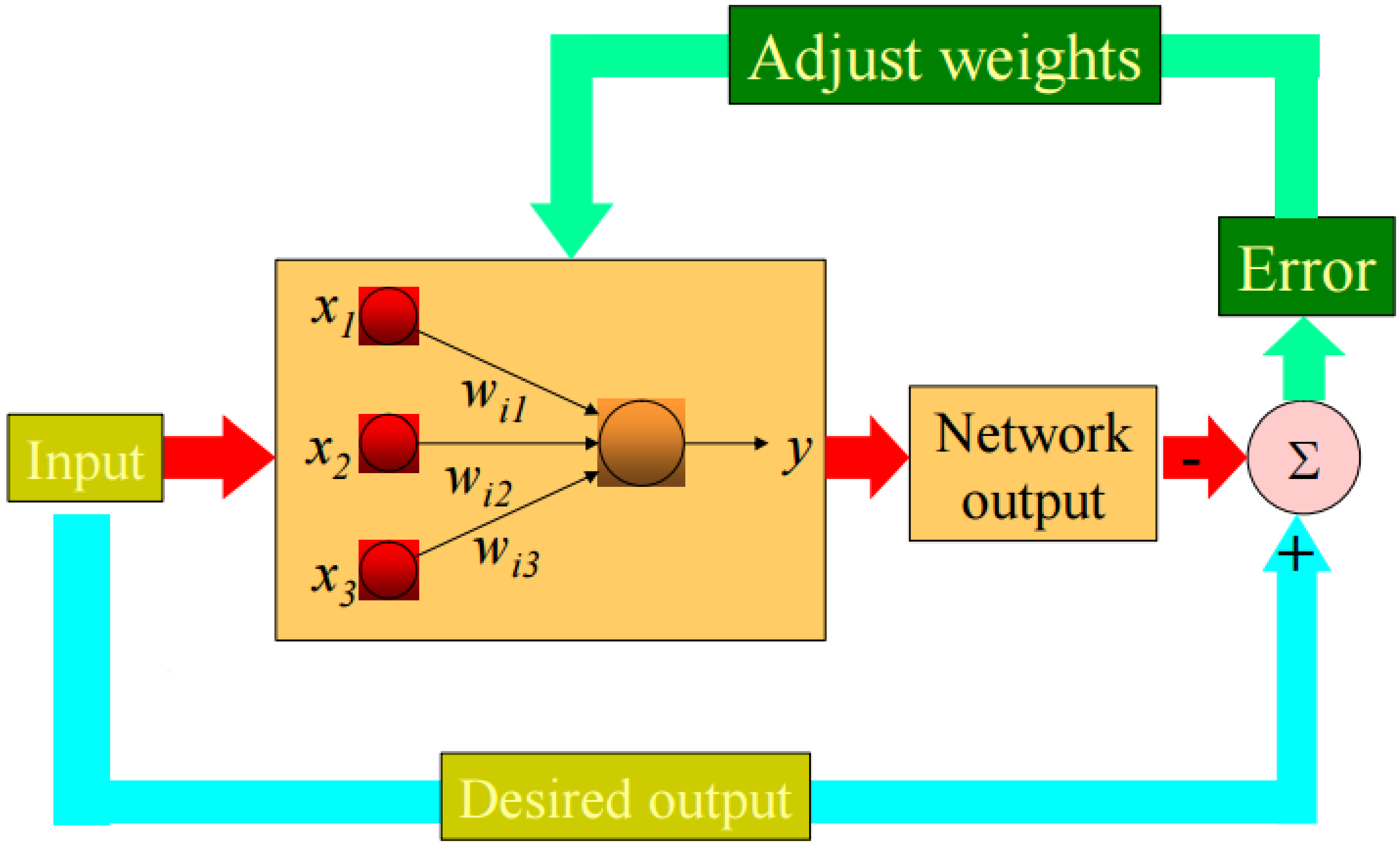
MORE ON PERCEPTRON LEARNING

INT301 Bio-computation, Week 2, 2021

Outline

- Convergence of Perceptron
- Perceptron as network for classification
- Decision boundary
- Limitations of one-layer network

Recall：Perceptron



Review of Perceptron rule

- ***Perceptron rule***: a sequential learning procedure for updating the weights.

- Perceptron Learning Algorithm

$$\boxed{\Delta w = \text{learning rate} \times \underbrace{(\text{teacher} - \text{output})}_{\text{error}} \times \text{input}}$$

Perceptron Rule:
Further Discussion

- A weight of connection changes ***only if*** both the input value and the error of the output unit are not equal to 0.
  - If the output is correct ($y_e = o_e$) the weights w_i are not changed
  - If the output is incorrect ($y_e \neq o_e$) the weights w_i are changed such that the output of the perceptron for the new weights is *closer* to y_e.

- The algorithm **converges** to the correct classification, if
  - the training data is **linearly separable**;
  - and the learning rate is sufficiently small, usually set below 1, which **determines the amount of correction made in a single iteration.**

Perceptron convergence Theorem

For any data set *that's* ***linearly separable***, the learning rule is guaranteed to find a solution in a finite number of steps.

Assumptions:

- At least one such set of weights, w*, exists
- There are a finite number of training patterns.
- The threshold function is uni-polar (output is 0 or 1).

Network Performance for Perceptron

The network performance during training session can be measured by a *root-mean-square (RMS) error value*.

$$RMS = \sqrt{\frac{\sum_{p=0}^{n_p}\sum_{j=0}^{n_o} e_{jp}^2}{n_p n_o}} = \sqrt{\frac{\sum_{p=0}^{n_p}\sum_{j=0}^{n_o} (t_{jp} - X_{jp})^2}{n_p n_o}}$$

where

n_p is the number of patterns in the training set and
n_o is the number of units in the output layer

- As the target output values t_{jp} and n_p and n_o numbers are constants, **the RMS error is a function of the instant output values X_{jp} only**

The Network Performance

$$RMS = \sqrt{\frac{\sum_{p=0}^{n_p}\sum_{j=0}^{n_o}(t_{jp} - X_{jp})^2}{n_p n_o}}$$

- In turn, the instant outputs X_{jp} are functions of the input values a_{ip} , which are also constants, and of the weights of connections w_{ji}

$$X_{jp} = f(S_{jp} = \sum_{i=0}^{n_i} w_{ji} a_{ip}) = \tilde{f}(w_{ji}, a_{ip}),$$

So the performance of the network measured by the RMS error also is function of the weights of connections **only**

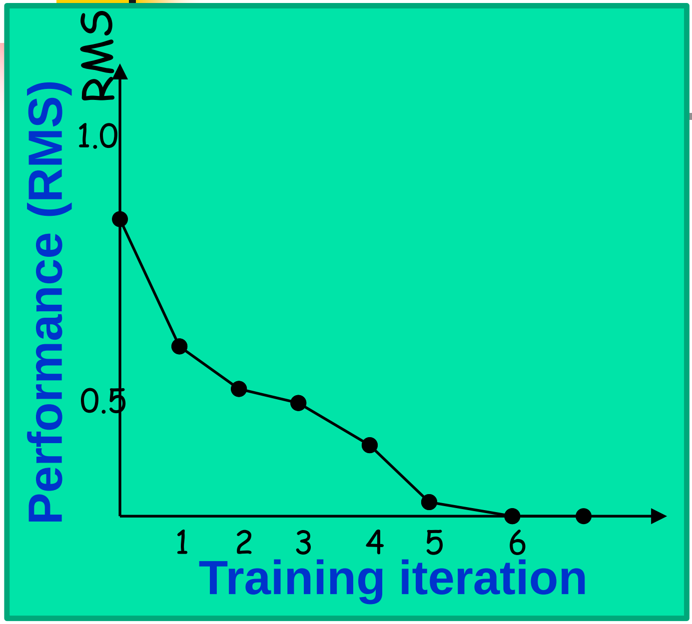
More on The Network Performance

$$RMS = F(w_{ji}, a_{ip})$$

- Performance of the network measured by the RMS error *is function of the weights of connections only.*

- The *best performance of the network* corresponds to the minimum of the RMS error, and we adjust the weights of connections in order to get that minimum.

RMS on Training Set



$$RMS = F(w_{ji}, a_{ip})$$

Shown is a ***learning curve***, i.e., dependence of the RMS error on the number of iterations for the training set.

- Initially, the adaptable weights are all set to small random values, and the network does not perform very well.
- As weights are adjusted during training, performance improves; when the error rate is low enough, training stops and the network is said to have *converged*.

RMS on the Training/Testing Data

RMS on the Training Data

$$RMS^{training} = \sqrt{\frac{\sum_{p=0}^{n_p}\sum_{j=0}^{n_o}(t_{jp} - X_{jp}^{trained})^2}{n_p n_o}}$$

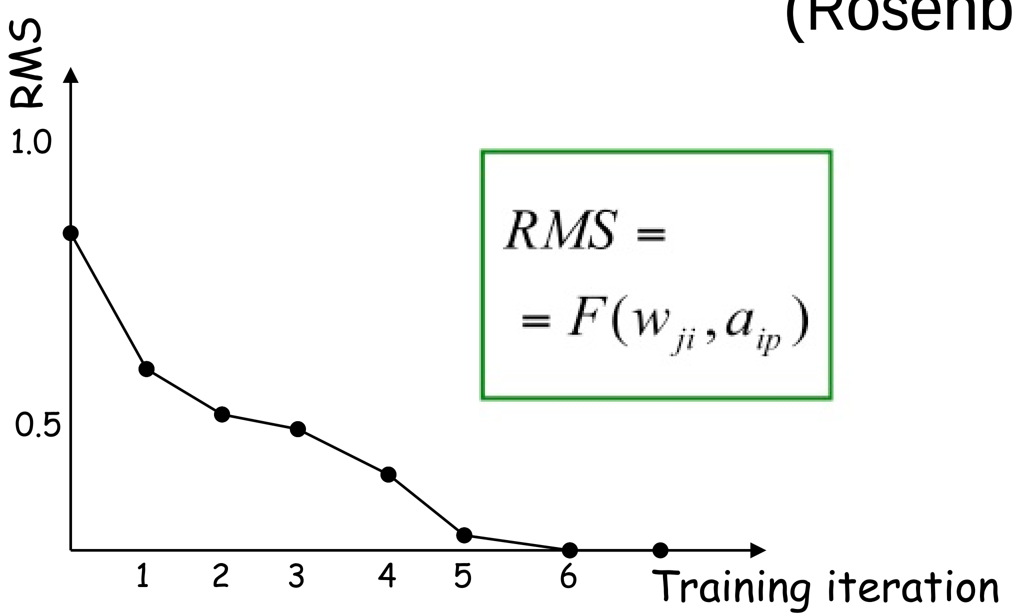
RMS on the Testing Data

$$RMS^{testing} = \sqrt{\frac{\sum_{p=0}^{n_p}\sum_{j=0}^{n_o}(t_{jp} - X_{jp}^{predicted})^2}{n_p n_o}}$$

Recall: Perceptron Convergence Theorem

***If** a set of weights that allow the perceptron to respond correctly to all of the training patterns exists, **then** the perceptron's learning method will find the set of weights, and it will do it in a finite number of iterations.*

(Rosenblatt, 1962)



$$RMS = F(w_{ji}, a_{ip})$$

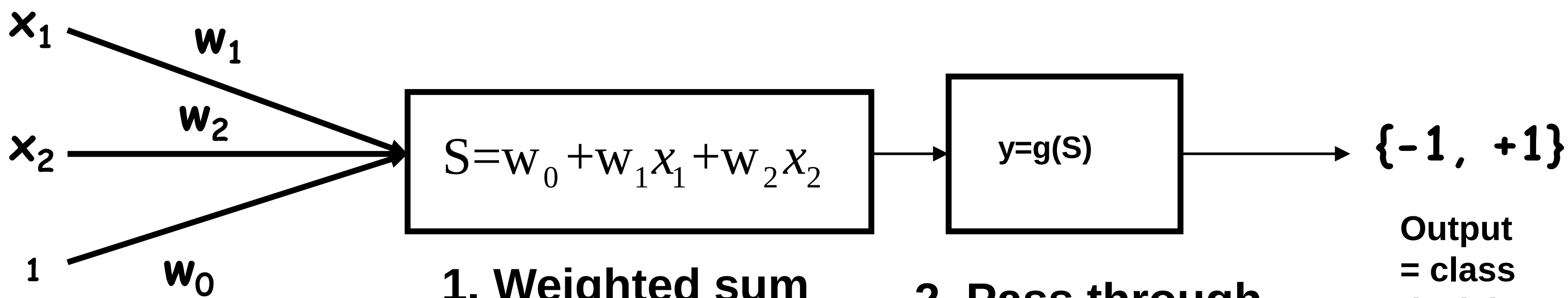
More on Perceptron Convergence

- There might be another possibility during a training session:
  - eventually performance stops improving, and the RMS error does not get smaller regardless of number of iterations.
- That means the network has failed to learn all of the answers correctly.

- *If the training is successful*, the perceptron is said
  - to have *gone through* the supervised learning, and
  - is able to classify patterns *similar to those of the training set.*

Perceptron As a Classifier

For *d*-dimensional data, perceptron consists of d-weights, a bias, and a thresholding activation function. For 2D data example， we have:



View the bias as another weight from an input which is constantly on

1. Weighted sum of the inputs

2. Pass through Heaviside function:
T(S)= -1 if S < 0
T(S)= 1 if S>= 0

Output = class decision

If we group the weights as a vector w , the net output y can be expressed as:

$$y = g(\mathrm{w} \cdot \mathrm{x} + \mathrm{w}_0)$$

Further Discussion

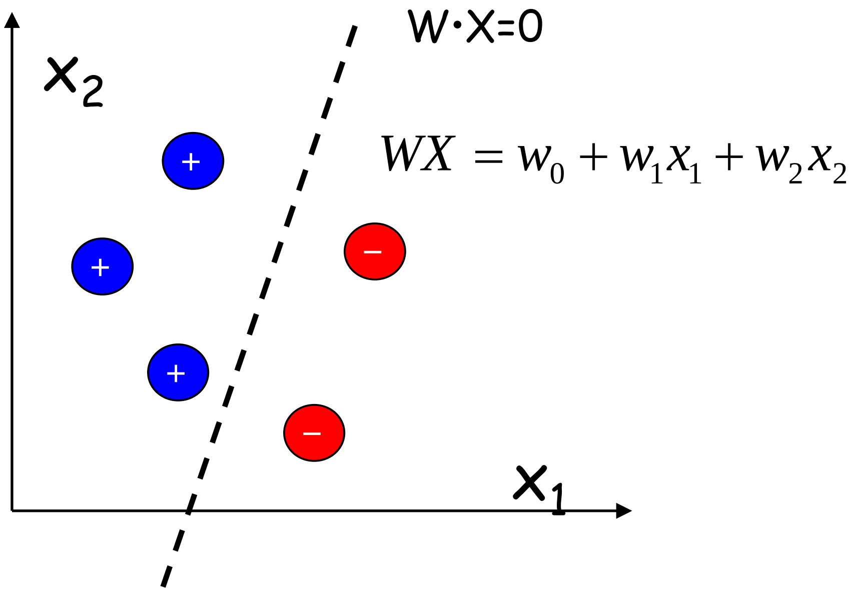
Perceptron As a Classifier

- A perceptron training is to compute weight vector:

$$W = [w_0, w_1, w_2, \ldots \ldots, w_p]$$

to correctly classify all the training examples.

E.g.,
consider when *p*=2



W·X is a **hyperplane**, which in 2d is a straight line.

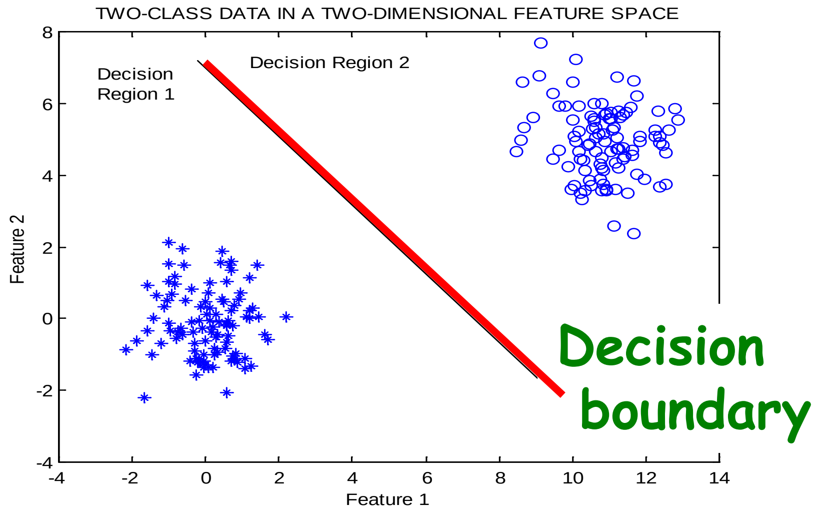
Neural Network as Classifier

- For 2 classes, view net output as a **discriminant function y(x, w),** where:

  y(x, w) = 1 , if x in class 1 (C1)

  y(x, w) = - 1, if x in class 2 (C2)

Example



- For m classes, a classifier should partition the feature space into m ***decision regions***
  - The **line or curve** separating the classes is the ***decision boundary.***
  - In more than 2 dimensions, this is a hyperplane.

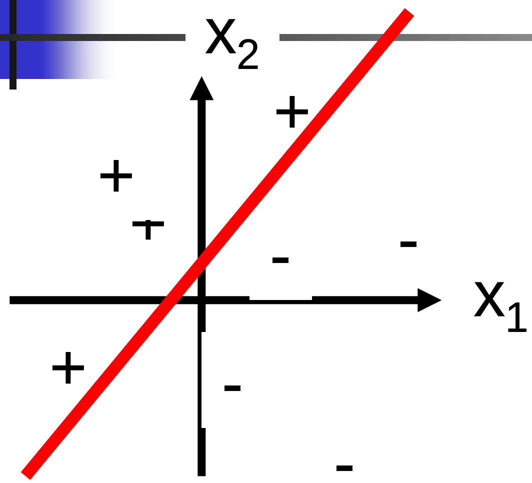
Further on Perceptron Decision Boundary

A perceptron represents a ***hyperplane decision surface*** in d-dimensional space, for example, a line in 2D, a plane in 3D, etc.

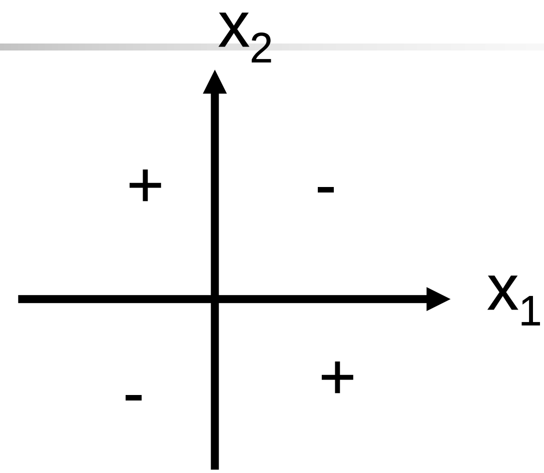
The equation of the hyperplane is $\mathbf{w} \cdot \mathbf{x}^T = 0$

This is the equation for points in x-space that are **on** the boundary

Decision boundary of Perceptron



Linearly separable



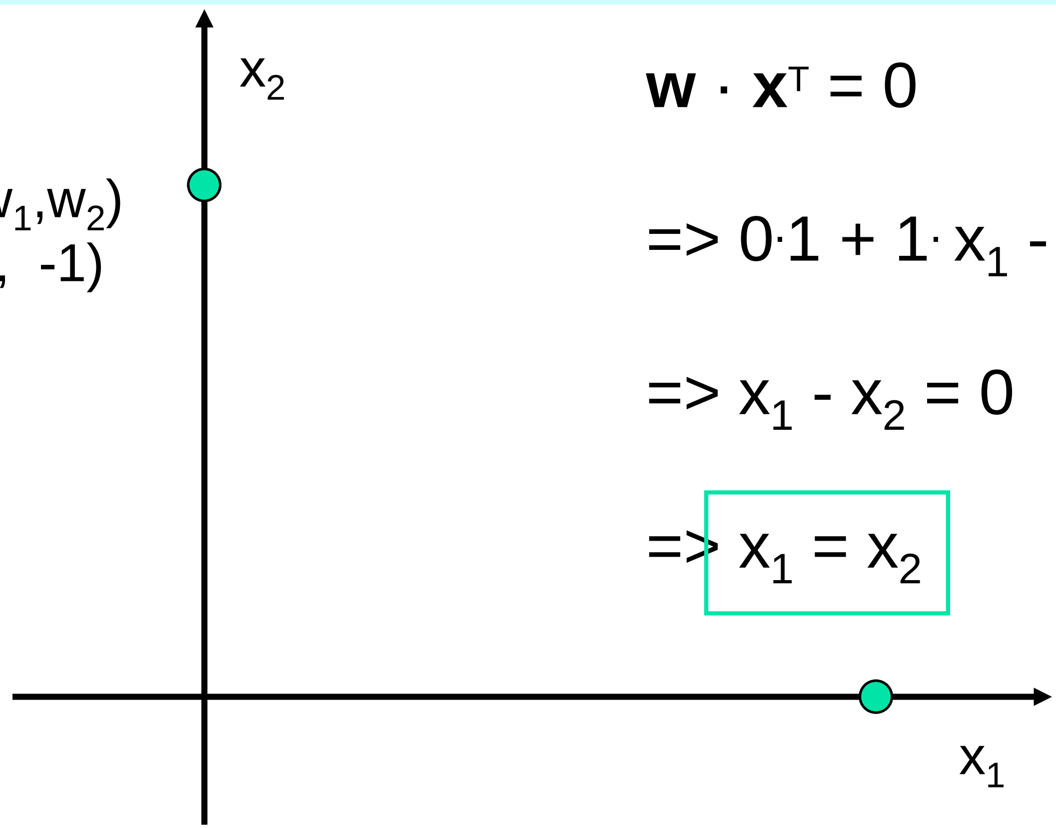
Non-Linearly separable

- Perceptron is able to represent some useful functions
- But functions that are not linearly separable (e.g. XOR) are not representable

Example of Perceptron Decision Boundary

- Decision surface is the surface at which the output of the unit is precisely equal to the threshold, i.e. $\sum w_i x_i = \theta$

$w = (w_0, w_1, w_2)$
$= (0, 1, -1)$



$\mathbf{w} \cdot \mathbf{x}^T = 0$

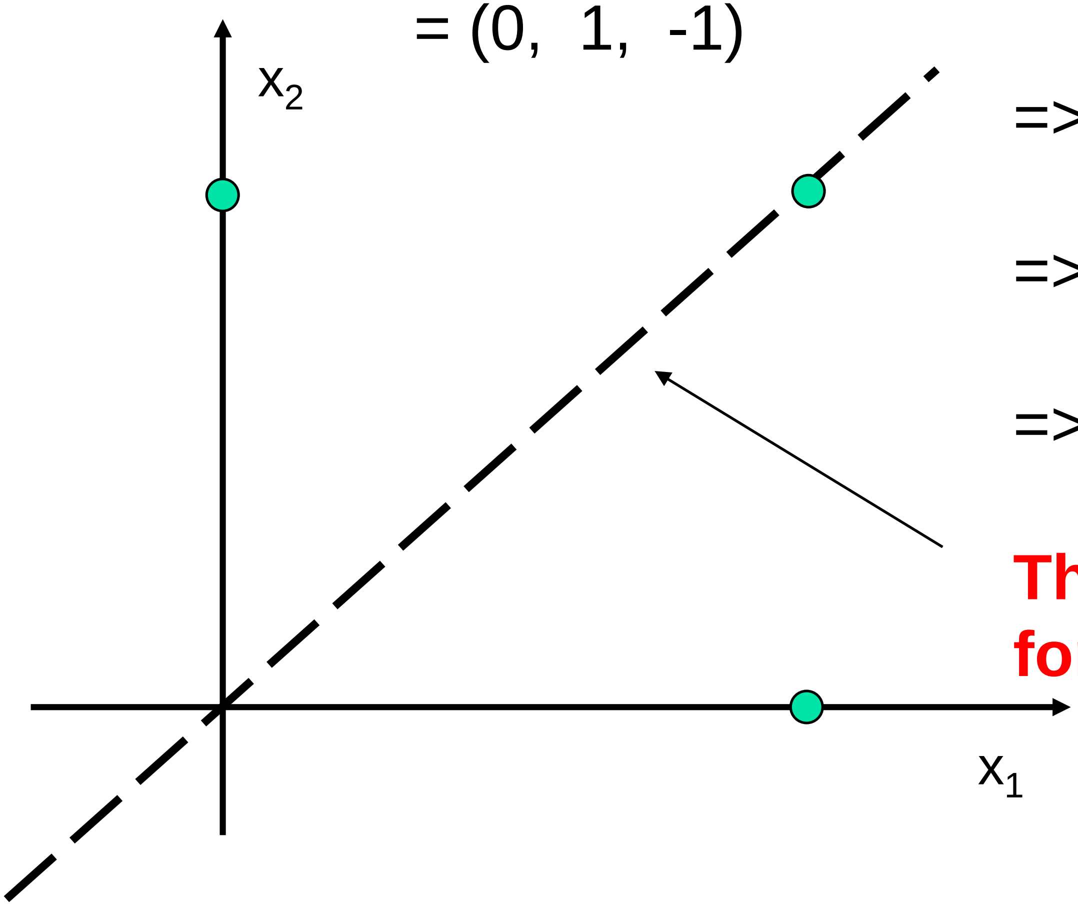
$\Rightarrow 0 \cdot 1 + 1 \cdot x_1 - 1 \cdot x_2 = 0$

$\Rightarrow x_1 - x_2 = 0$

$\Rightarrow \boxed{x_1 = x_2}$

Example of Perceptron Decision Boundary

$w = (w_0, w_1, w_2)$
$= (0, 1, -1)$



$\mathbf{w} \cdot \mathbf{x}^T = 0$

$\Rightarrow 0 \cdot 1 + 1 \cdot x_1 - 1 \cdot x_2 = 0$

$\Rightarrow x_1 - x_2 = 0$

$\Rightarrow \boxed{x_1 = x_2}$

This is the equation for the decision boundary

Example of Perceptron Decision Boundary

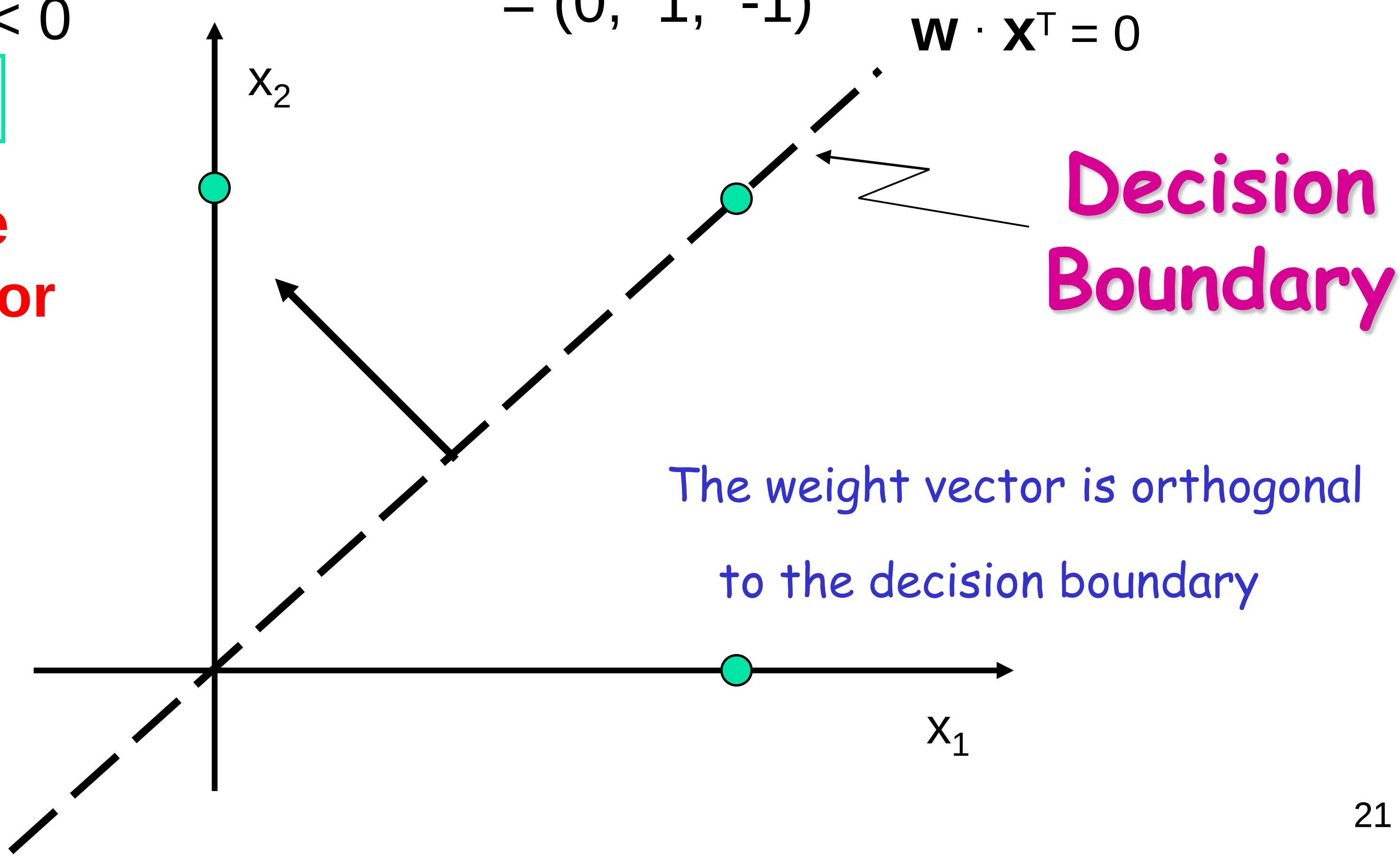
$\mathbf{w} \cdot \mathbf{x}^T < 0$

$\Rightarrow x_1 - x_2 < 0$

$\Rightarrow \boxed{x_1 < x_2}$

(this is the equation for decision region -1)

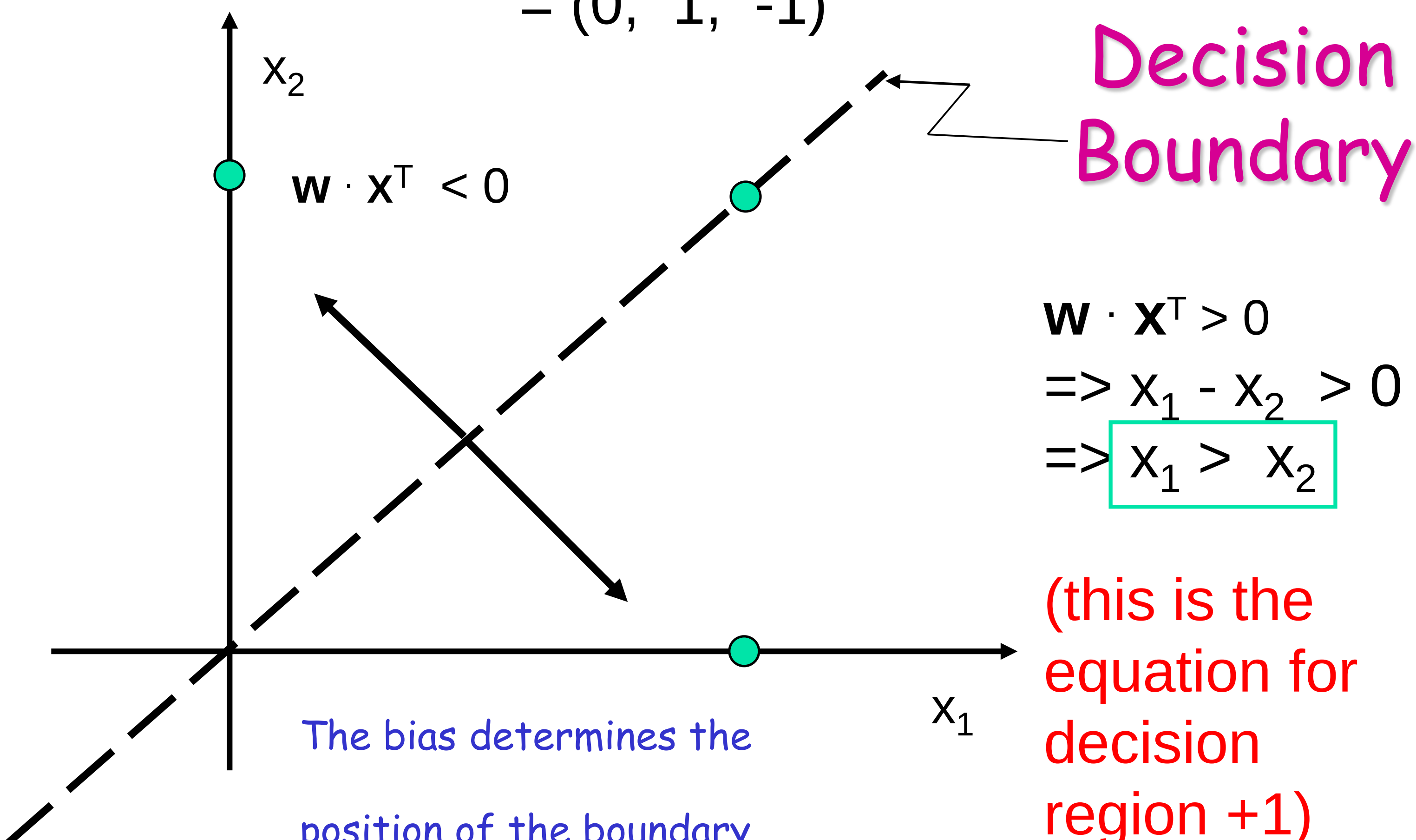
$\mathbf{w} = (w_0, w_1, w_2)$
$= (0, 1, -1)$



Example of Perceptron Decision Boundary

$\mathbf{w} = (w_0, w_1, w_2)$
$= (0,\ 1,\ -1)$

$\mathbf{w} \cdot \mathbf{x}^T = 0$



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

$\Rightarrow x_1 - x_2 > 0$

$\Rightarrow \boxed{x_1 > x_2}$

(this is the equation for decision region +1)

Linear Separability Problem

- If two classes of patterns can be separated by a decision boundary, represented by the linear equation

$$\boldsymbol{b} + \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{x}_{\boldsymbol{i}} \boldsymbol{w}_{\boldsymbol{i}} = 0$$

then they are said to be *linearly separable* and the perceptron can correctly classify any patterns

NOTE: without the bias term, the hyperplane will be forced to intersect origin.

Linear Separability Problem

- Decision boundary (i.e., ***W, b***) of linearly separable classes can be determined either by some learning procedures, or by solving linear equation systems based on representative patterns of each classes

- If such a decision boundary does not exist, then the two classes are said to be linearly inseparable.

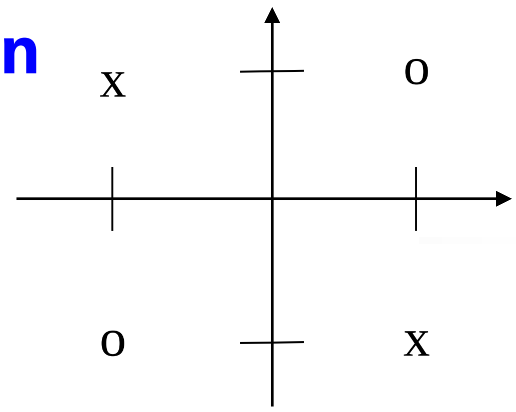
- Linearly inseparable problems cannot be solved by the simple perceptron network, more sophisticated architecture is needed.

- Examples of linearly inseparable classes

  - **Logical XOR (exclusive OR) function**

patterns (bipolar) decision boundary

x_1	x_2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



x: class I (y = 1)
o: class II (y = -1)

No line can separate these two classes, as can be seen from the fact that the following linear inequality system has no solution

$$\begin{cases} \boldsymbol{b} - \boldsymbol{w}_1 - \boldsymbol{w}_2 < 0 & (1) \\ \boldsymbol{b} - \boldsymbol{w}_1 + \boldsymbol{w}_2 \geq 0 & (2) \\ \boldsymbol{b} + \boldsymbol{w}_1 - \boldsymbol{w}_2 \geq 0 & (3) \\ \boldsymbol{b} + \boldsymbol{w}_1 + \boldsymbol{w}_2 < 0 & (4) \end{cases}$$

because we have b < 0 from (1) + (4),

and b >= 0 from (2) + (3), which is a contradiction

Examples of linearly separable classes

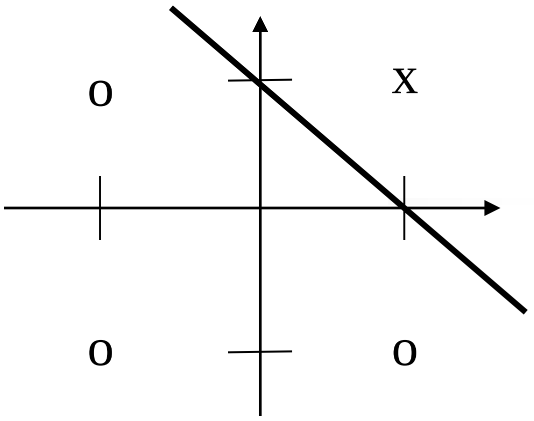
- Logical **AND** function

patterns (bipolar) decision boundary

x_1	x_2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

$w_1 = 1$
$w_2 = 1$
$b = -1$
$\theta = 0$

$$\boxed{-1 + x_1 + x_2 = 0}$$



x: class I (y = 1)
o: class II (y = -1)

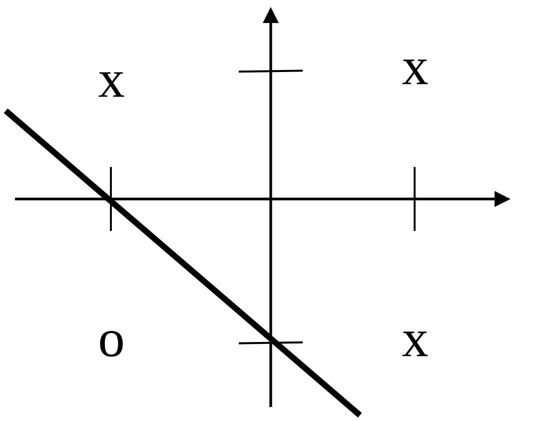
-Logical **OR** function

patterns (bipolar) decision boundary

x_1	x_2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

$w_1 = 1$
$w_2 = 1$
$b = 1$
$\theta = 0$

$$\boxed{1 + x_1 + x_2 = 0}$$



x: class I (y = 1)
o: class II (y = -1)

Tips for Building ANN

Formulating neural network solutions for particular problems is a multi-stage process:

1. Understand and specify the problem in terms of inputs and required outputs
2. Take the simplest form of network you think might be able to solve your problem
3. Try to find the appropriate connection weights (including neuron thresholds) so that the network produces the right outputs for each input in its training data
4. Make sure that the network works on its training data and test its generalization by checking its performance on new testing data
5. If the network doesn't perform well enough, go back to stage 3 and try harder
6. If the network still doesn't perform well enough, go back to stage 2 and try harder
7. If the network still doesn't perform well enough, go back to stage 1 and try harder
8. Problem solved – or not

THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學