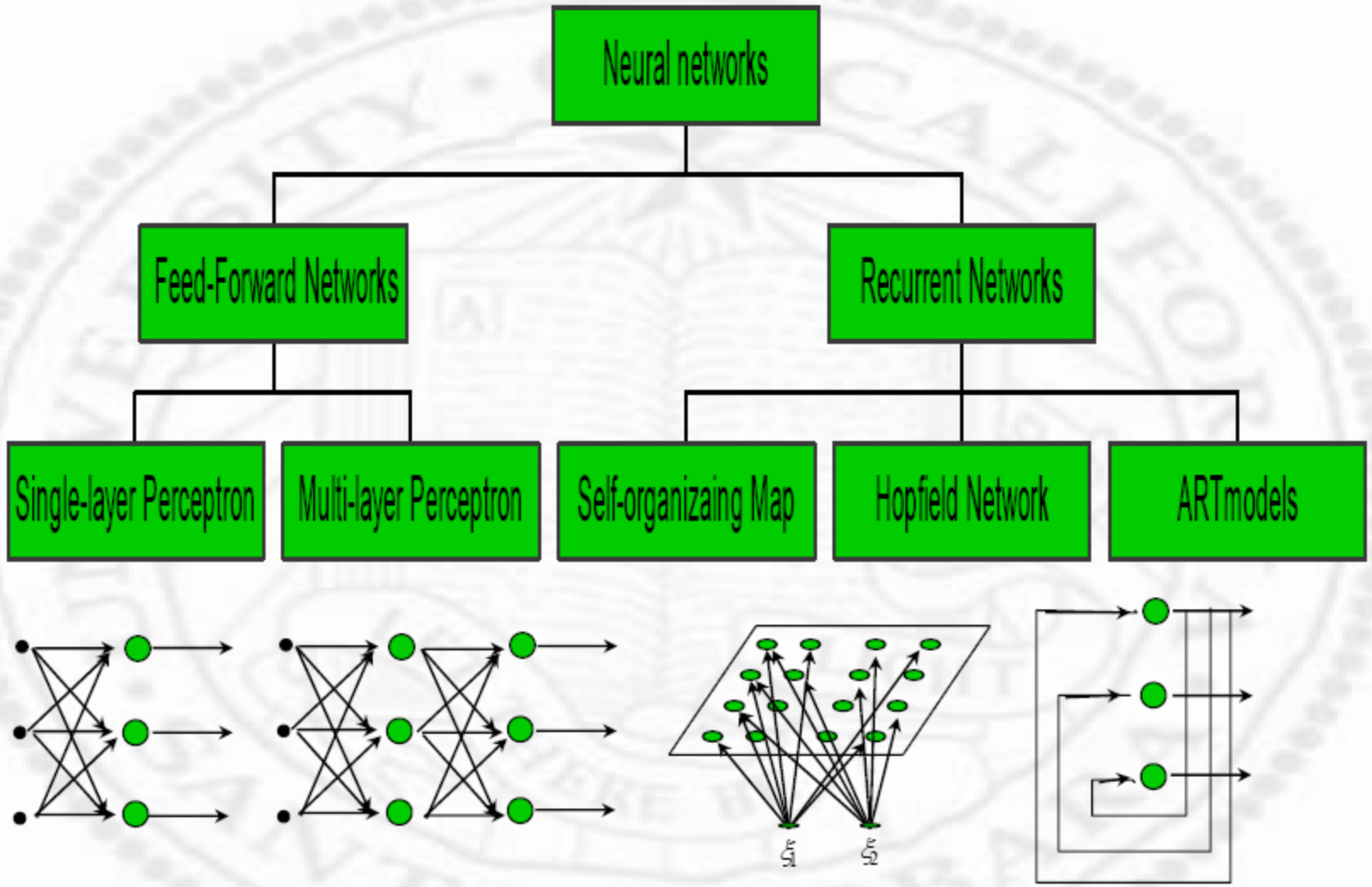
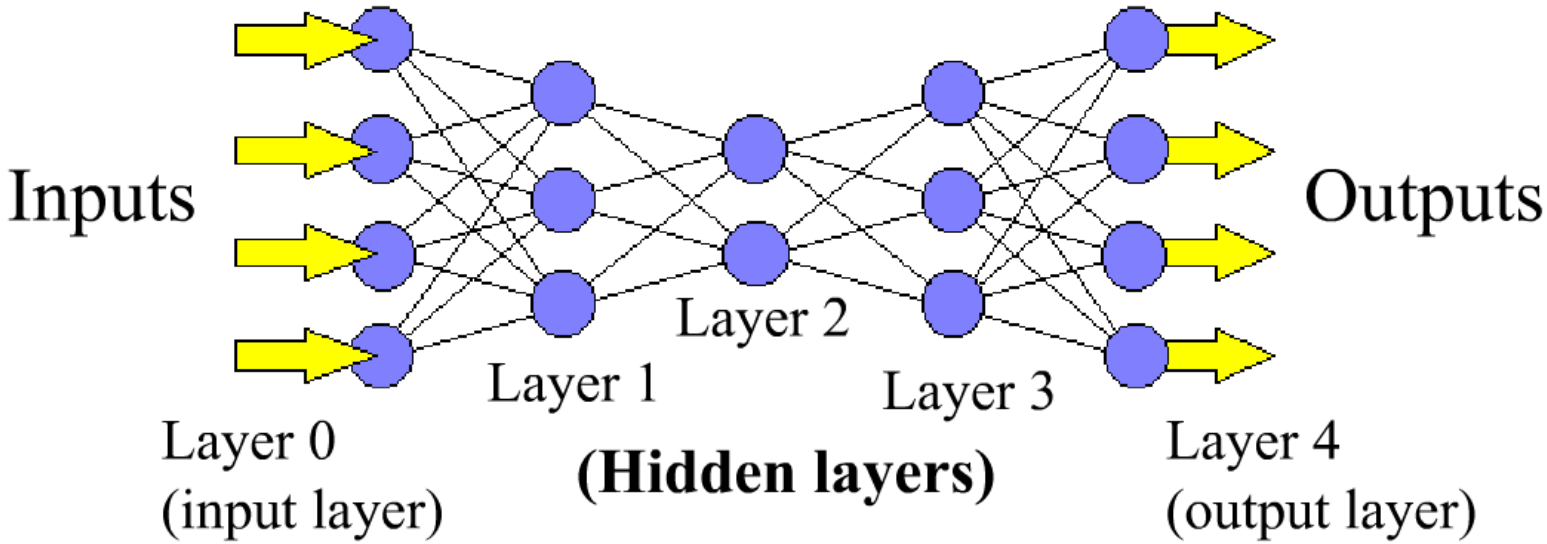
# Perceptron

# Computational Network Architecture



# Feedforward Networks

An example of a feedforward network:




Perceptrons are arranged in layers

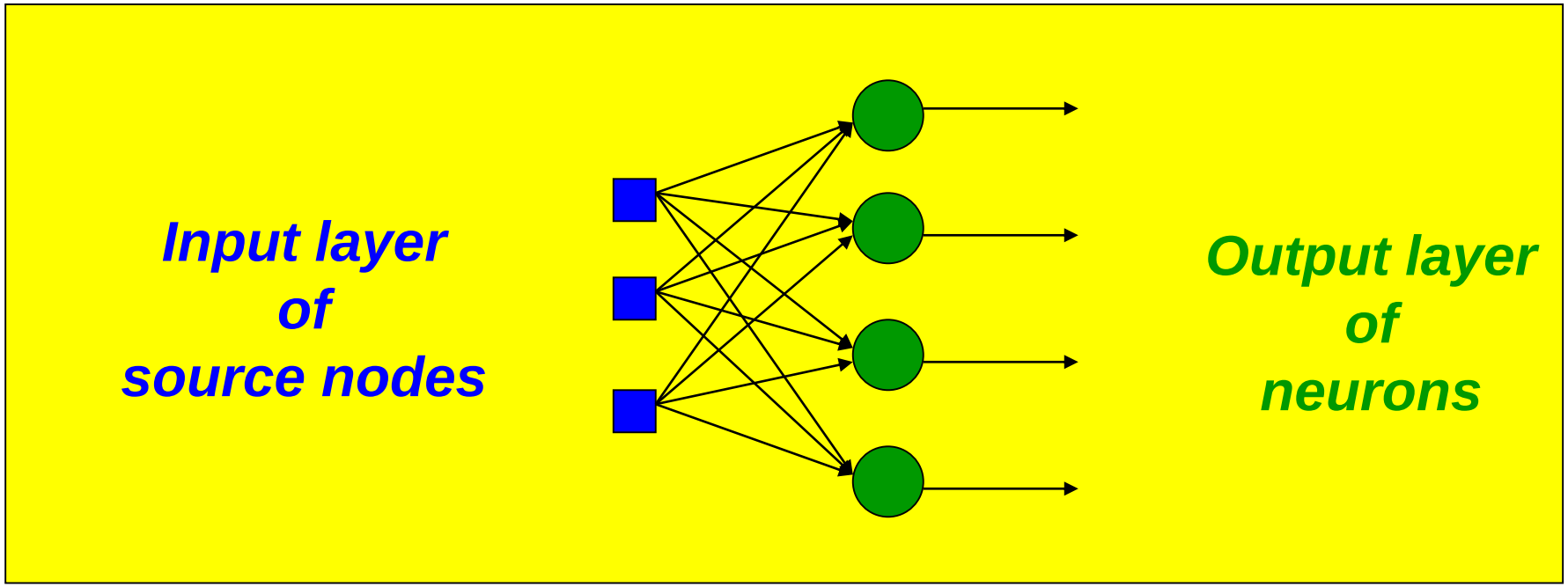
Interlayer connection: nodes in layer $i$ are connected to nodes in layer $i + 1$

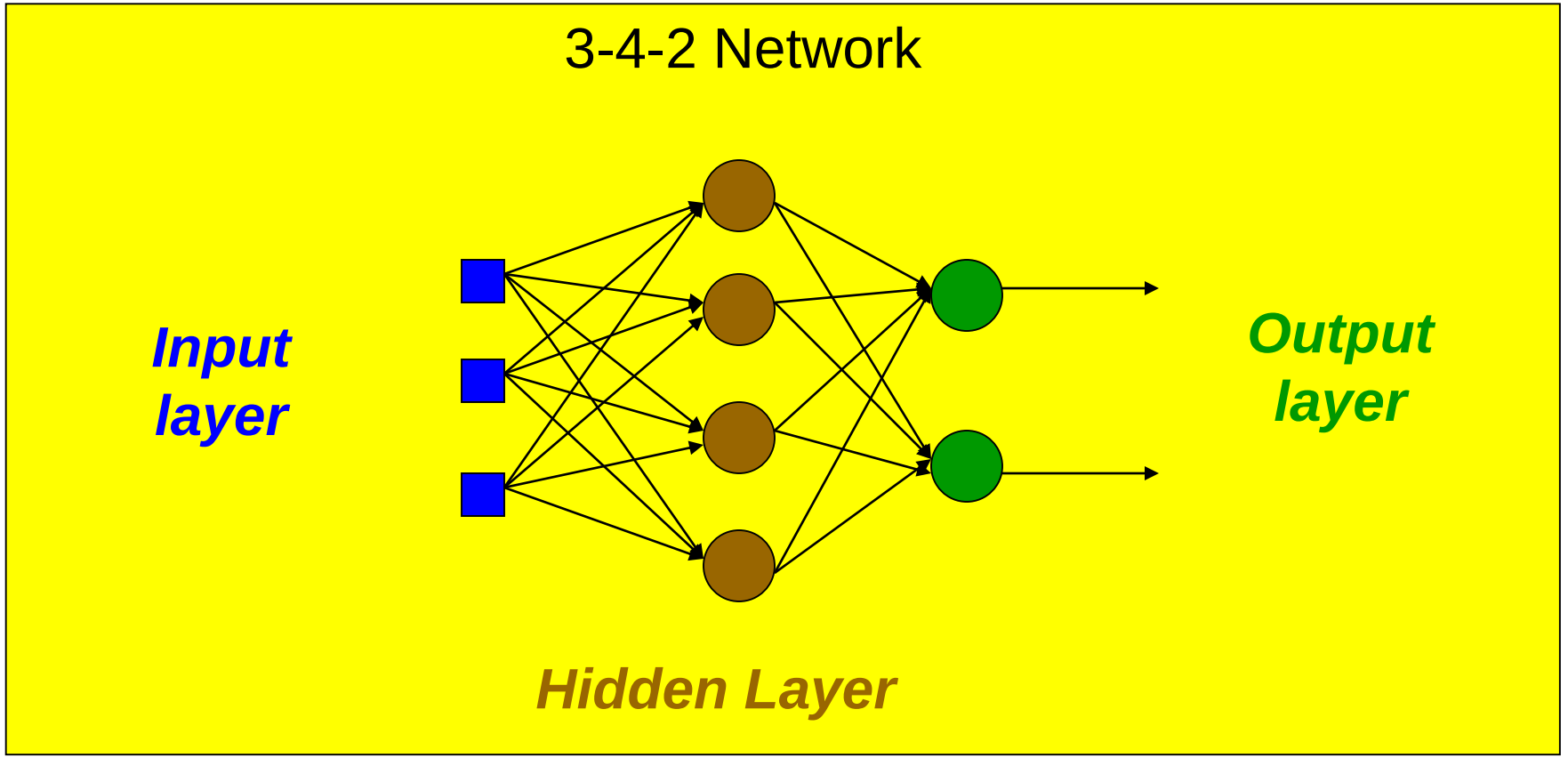
There are no intralayer connections

✱ *A feed-forward multi-layered network computes a function of the inputs and the weights.*

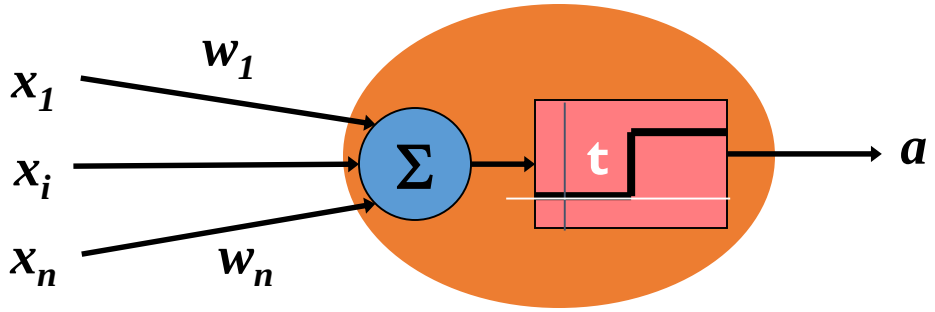
- Input units (on left or bottom):
  - activation is determined by the environment
- Output units (on right or top):
  - activation is the result
- Hidden units (between input and output units):
  - cannot observe directly

- Perceptrons have input units followed by one layer of output units, i.e. no hidden units


Input layer
of
source nodes
Output layer
of
neurons


3-4-2 Network
Input layer
Output layer
Hidden Layer

- LTUs where studied in the 1950s!
  - mainly as single-layered nets
  - since an effective learning algorithm was known
- Perceptrons:
  - simple 1-layer network, units act independently
  - composed of linear threshold units (LTU)
    - a unit's inputs, $x_i$, are weighted, $w_i$, and combined
    - step function computes activation level, $a$

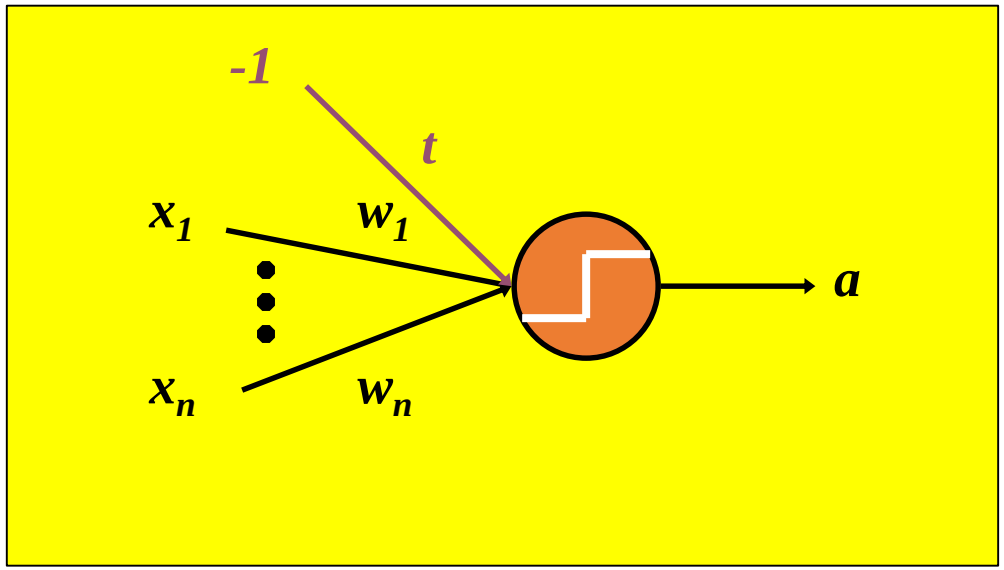


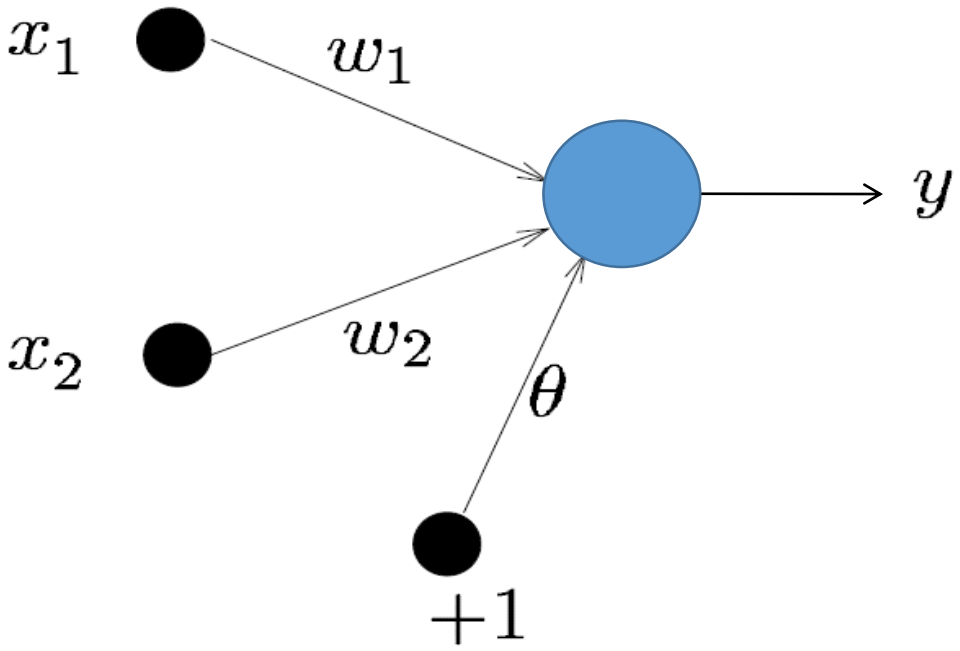
- Threshold is just another weight (called the bias):

  $(w_1 * x_1) + (w_2 * x_2) + \ldots + (w_n * x_n) >= t$

  is equivalent to

  $(w_1 * x_1) + (w_2 * x_2) + \ldots + (w_n * x_n) + (t * -1) >= 0$



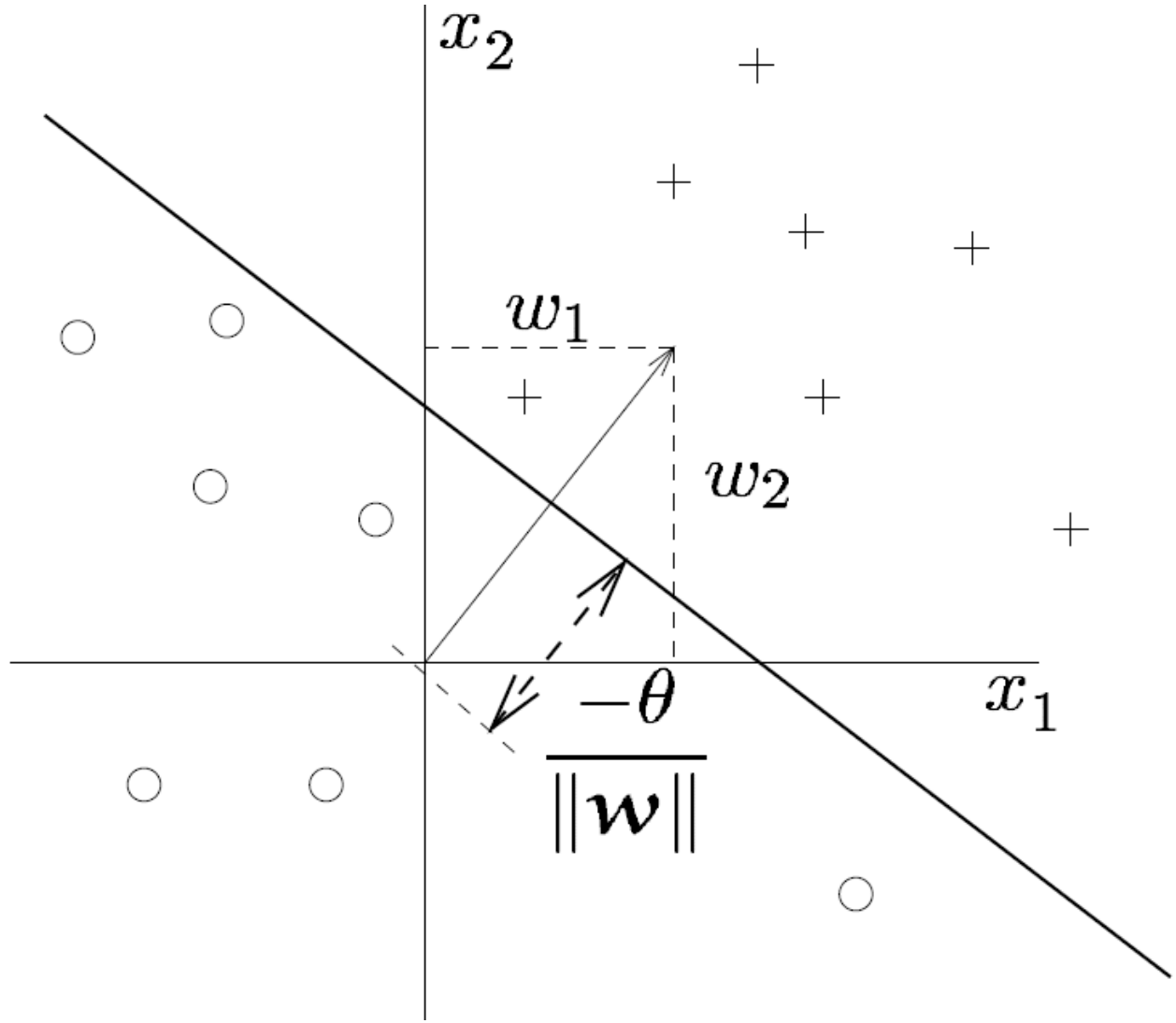



$$y = \mathcal{F}\left(\sum_{i=1}^{2} w_i x_i + \theta\right)$$

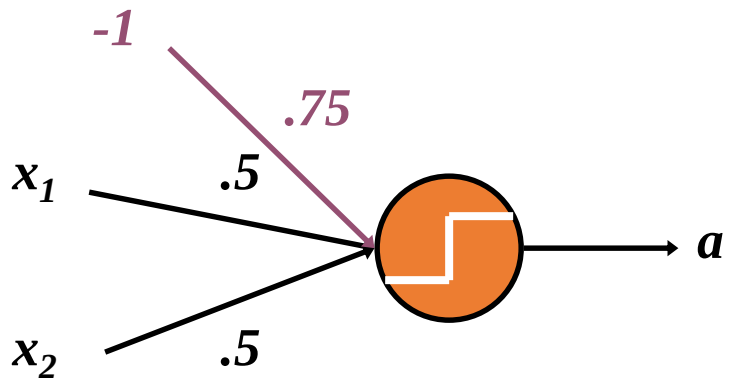
$$\mathcal{F}(s) = \begin{cases} 1 & \text{if } s > 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$w_1 x_1 + w_2 x_2 + \theta = 0$$

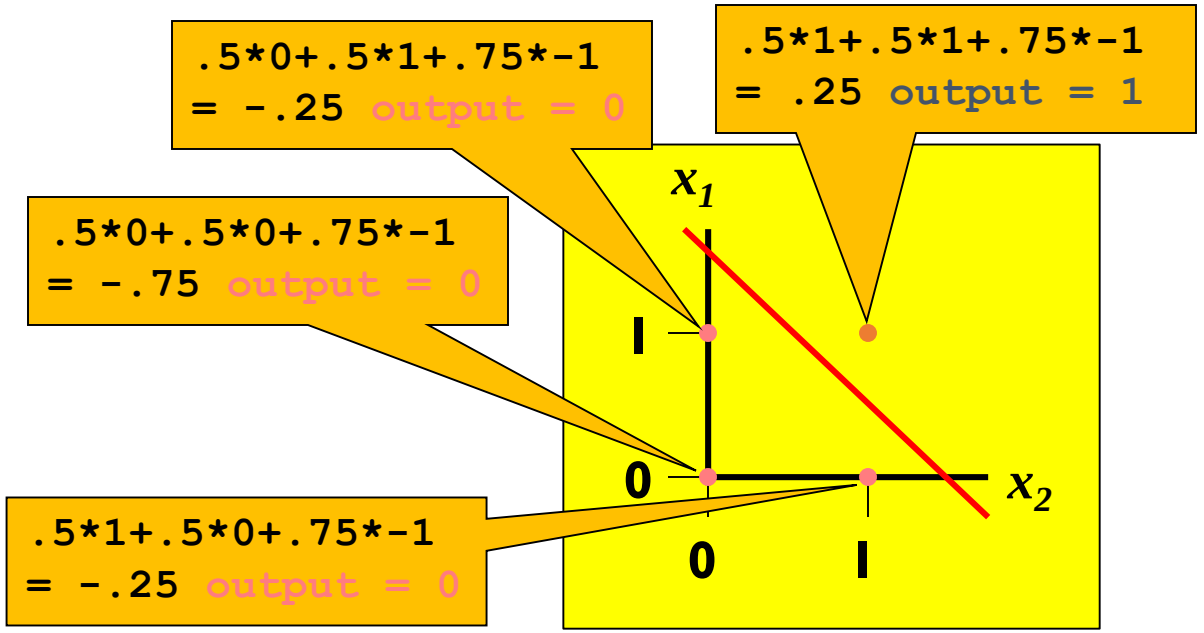
$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{\theta}{w_2}$$



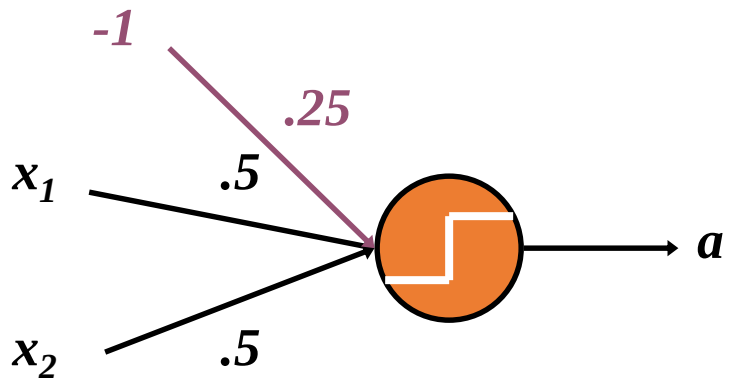
- AND Perceptron:
  - inputs are 0 or 1
  - output is 1 when both $x_1$ and $x_2$ are 1



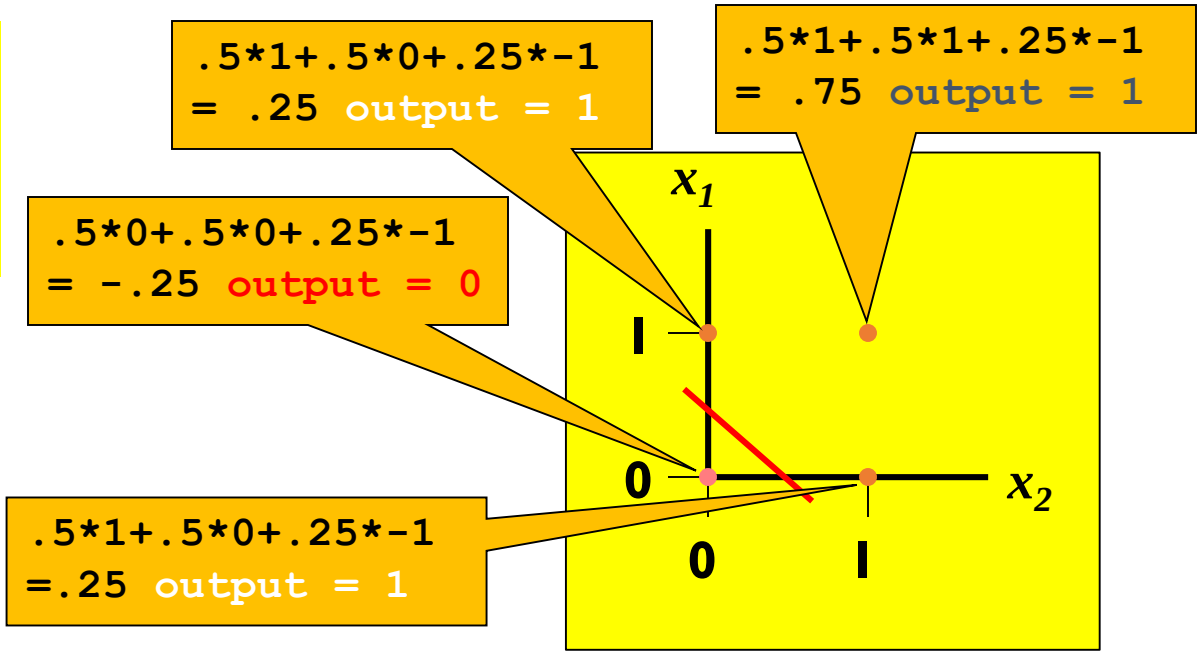

- **2-D input space**
  - 4 possible data points



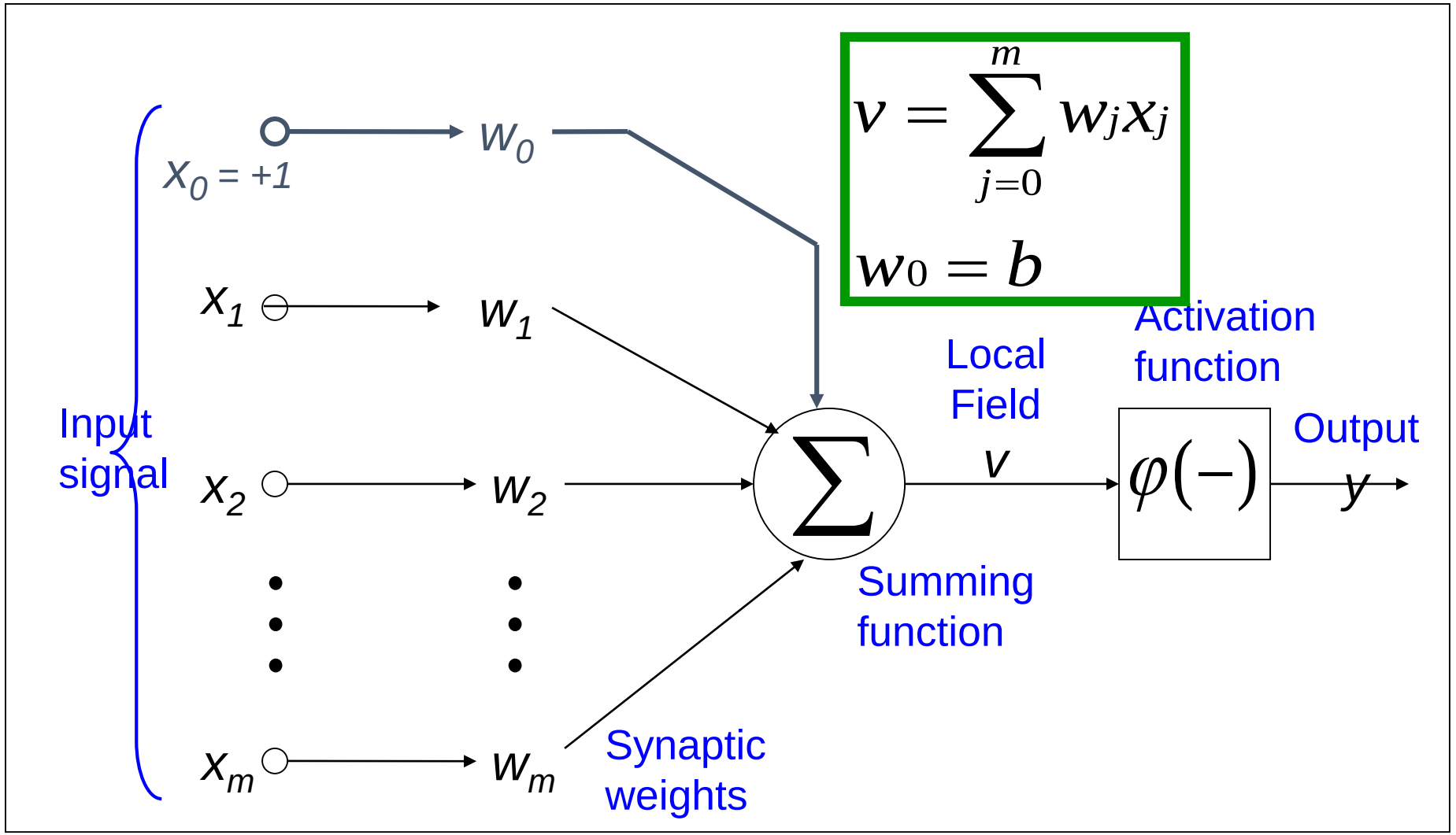
- OR Perceptron:
  - inputs are 0 or 1
  - output is 1 when either $x_1$ and/or $x_2$ are 1




- **2-D input**
  - 4 possible data points



• The bias is an external parameter of the neuron. *It* can be modeled by adding an extra input.



# ☞ How might perceptrons learn?

- Programmer specifies:
  - numbers of units in each layer
  - connectivity between units

  *So the only unknown is the weights*

- Perceptrons learn by changing their weights
  - supervised learning is used
  - the correct output is given for each training example
    - an example is a list of values for the input units
    - correct output is a list desired values for the output units

1. Initialize the weights in the network
   (usually with random values)
2. Repeat until all examples correctly classified
   or some other stopping criterion is met

   for each example *e* in training set do

   *a.* *O = neural_net_output(network, e)*

   *b.* *T = desired output*, i.e Target or Teacher's output

   *c.* *update_weights(e, O, T)*

- Unlike other learning techniques, Perceptrons need to "see" all of the training examples multiple times.

- Each pass through all of the training examples is called an epoch.

# ☞ How should the weights be updated?

- Determining how to update the weights
  is a case of the credit assignment problem.

- Perceptron Learning Rule:
  - $w_i = w_i + \Delta w_i$
  - where $\Delta w_i = \alpha * x_i * (T - O)$
    - where $x_i$ is the value associated with $i$th input unit
    - $\alpha$ is a constant between 0.0 and 1.0
      called the **learning rate**

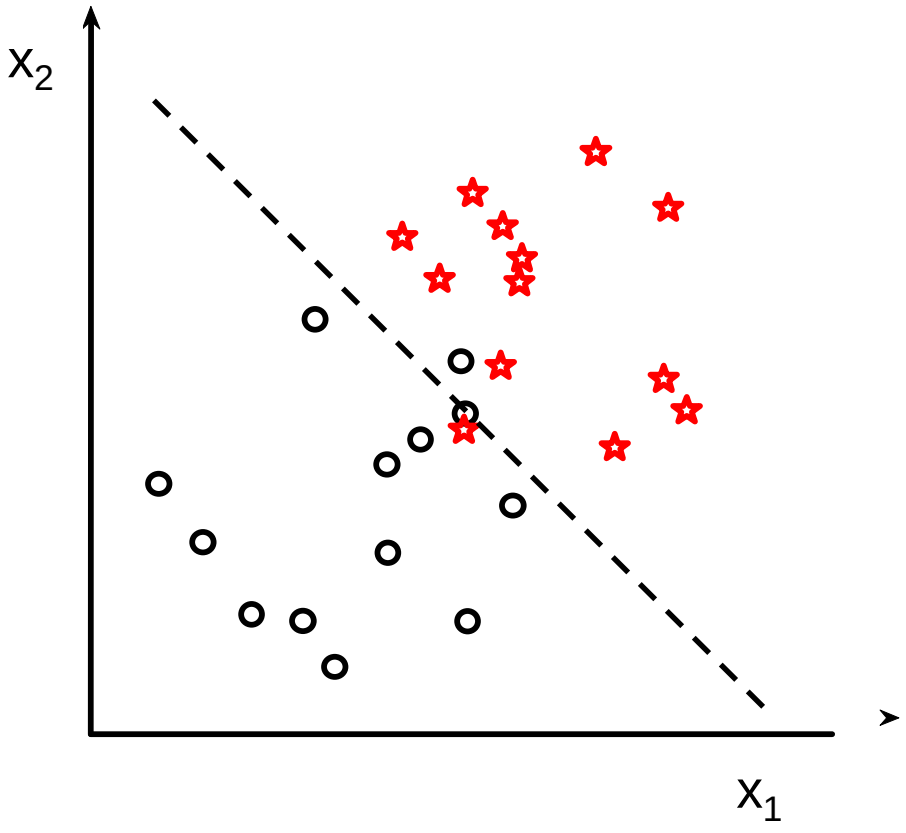
- $Dw_i = a * x_i * (T - O)$
  - note it doesn't depend on $w_i$

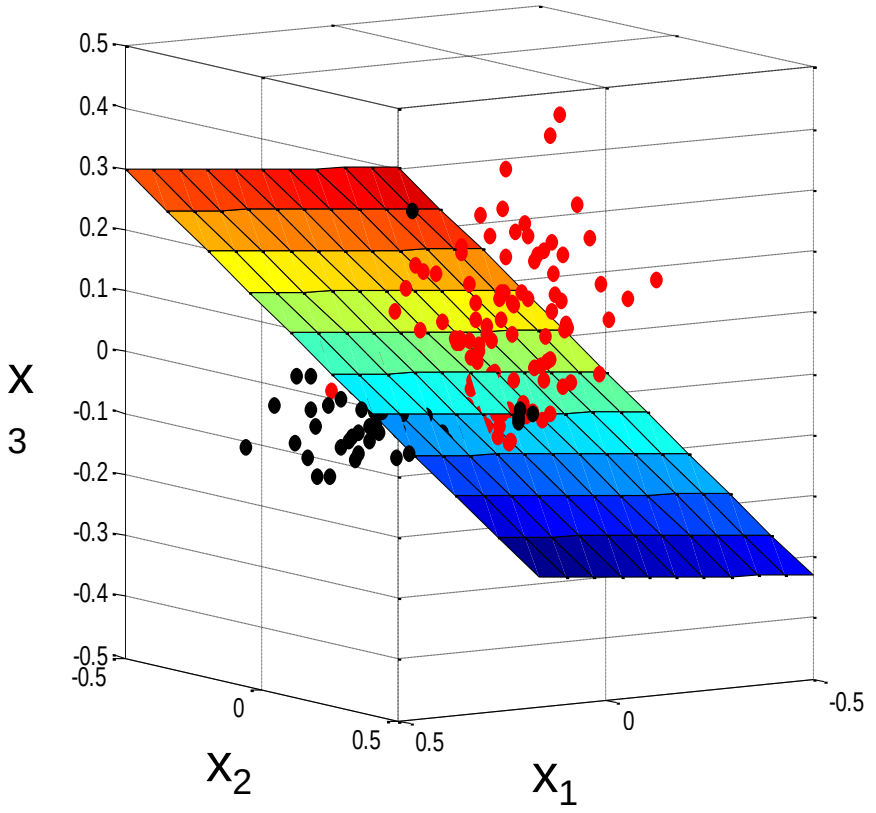
☞ When won't the weight be change (i.e. $Dw_i = 0$)?
  - **correct output**, i.e. $T = O$ gives $a * x_i * 0 = 0$
  - **zero input**, i.e. $x_i = 0$ gives $a * 0 * (T - O) = 0$

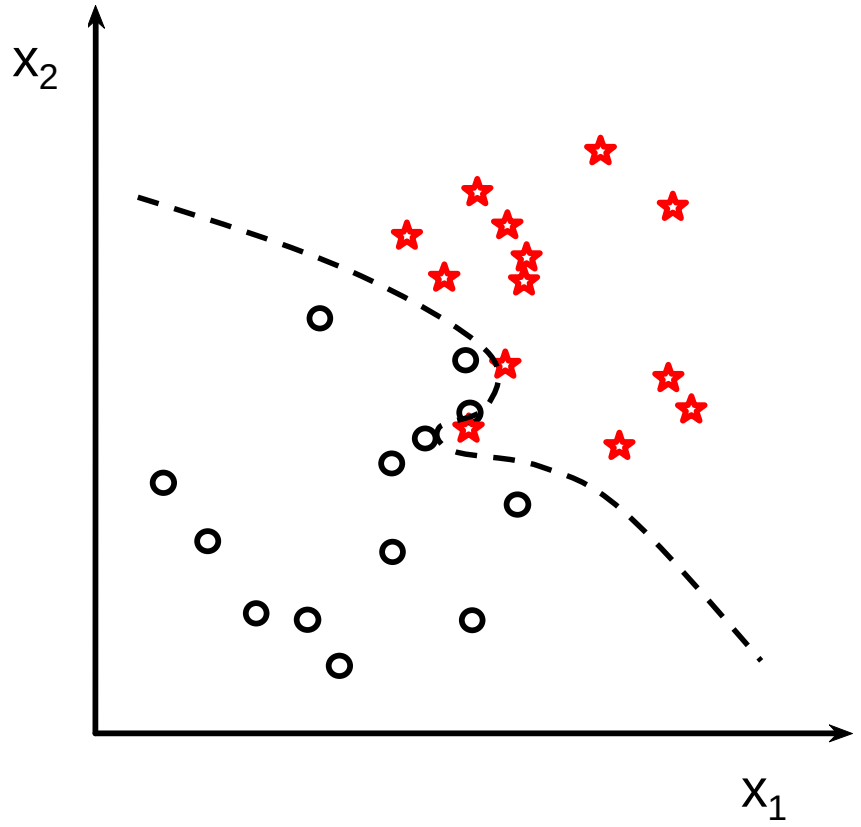
☞ What should happen to the weight if $T=1$ & $O=0$?
  - **Increase it** so that maybe next time the weighted sum will exceed the threshold causing output to be 1

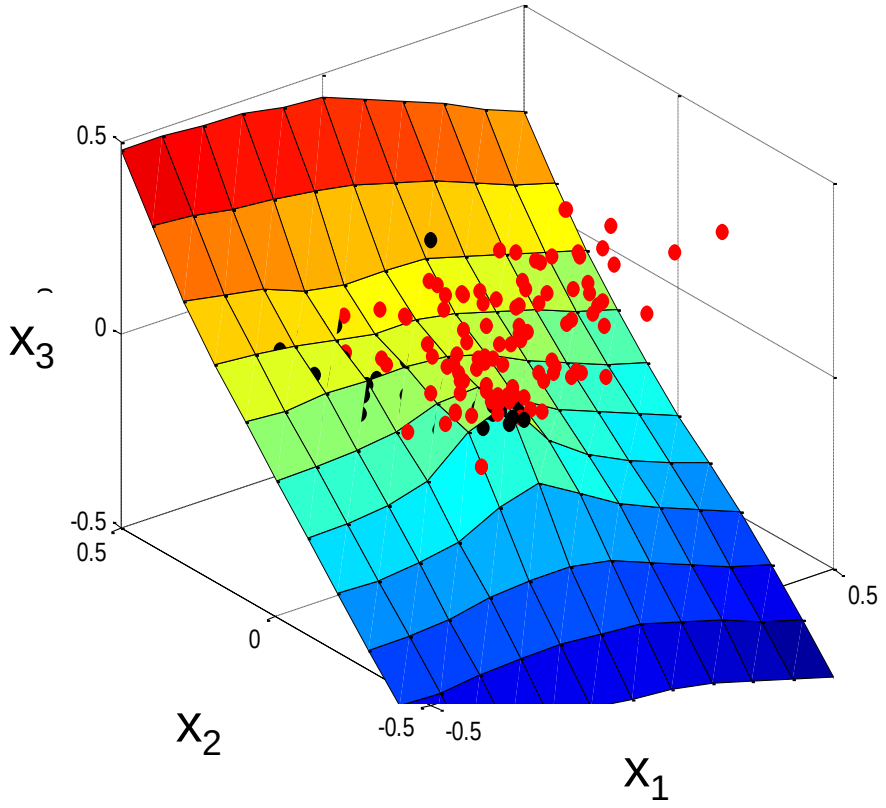
☞ What should happen to the weight if $T=0$ & $O=1$?
  - **Decrease it** so that maybe next time the weighted sum won't exceed the threshold causing output to be 0

# hyperplane




$x_2$
$x_1$


0.5
0
-0.5
0.5
0
-0.5
-0.5
0.5
$x_3$
$x_2$
$x_1$

- PLR is a "local" learning rule
  only local information in the network
  is needed to update a weight

- PLR performs gradient descent in "weight space“ this rule iteratively adjusts all of the weights so that at for each training example the error is monotonically non-increasing, i.e. ~decreases

✱ Perceptron Convergence Theorem says
*if a set of examples are learnable,*
*then PLR will find the necessary weights*

– in a finite number of steps

– independent of the initial weights

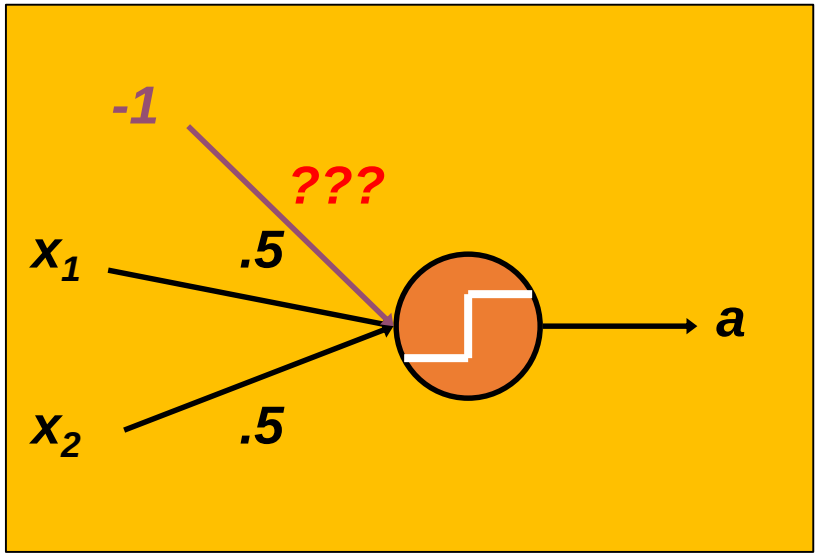
- This theorem says that if a solution exists, PLR's gradient descent is guaranteed to find an optimal solution (i.e., 100% correct classification) for any 1-layer neural network

## ☞ What are the limitations of perceptron learning?

- A single perceptron's output is determined by the separating hyperplane defined by $(w_1 * x_1) + (w_2 * x_2) + \ldots + (w_n * x_n) = t$

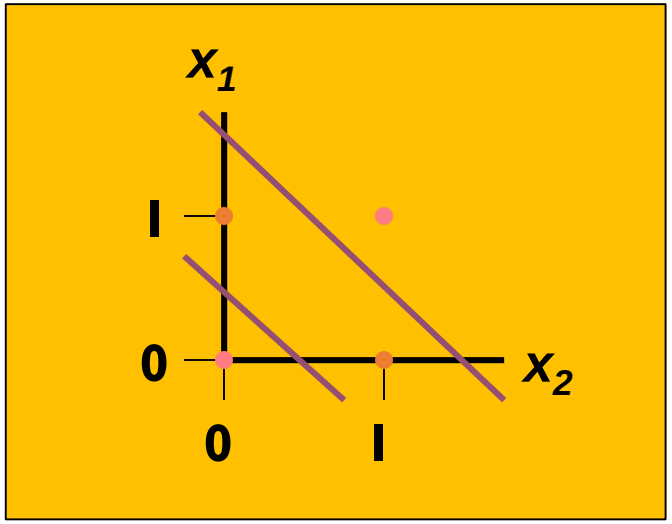
- So, Perceptrons can only learn functions that are linearly separable (in input space).

- XOR Perceptron:
  - inputs are 0 or 1
  - output is 1 when
    $x_1$ is 1 and $x_2$ is 0 or
    $x_1$ is 0 and $x_2$ is 1




- **2-D input space with 4 possible data points**

☞ **How do you separate positives from negatives using a straight line?**



- In general, the goal of learning in a perceptron is to adjust the separating hyperplane

- which divides an n-dimensional input space
  where n is the number of input units

- by modifying the weights (and biases) until all of the examples with target value 1 are on one side of the hyperplane,
- and all of the examples with target value 0
  are on the other side of the hyperplane.

- *Perceptrons as a computing model are too weak because they can only learn linearly-separable functions.*

- *To enhance the computational ability, general neural networks have multiple layers of units.*

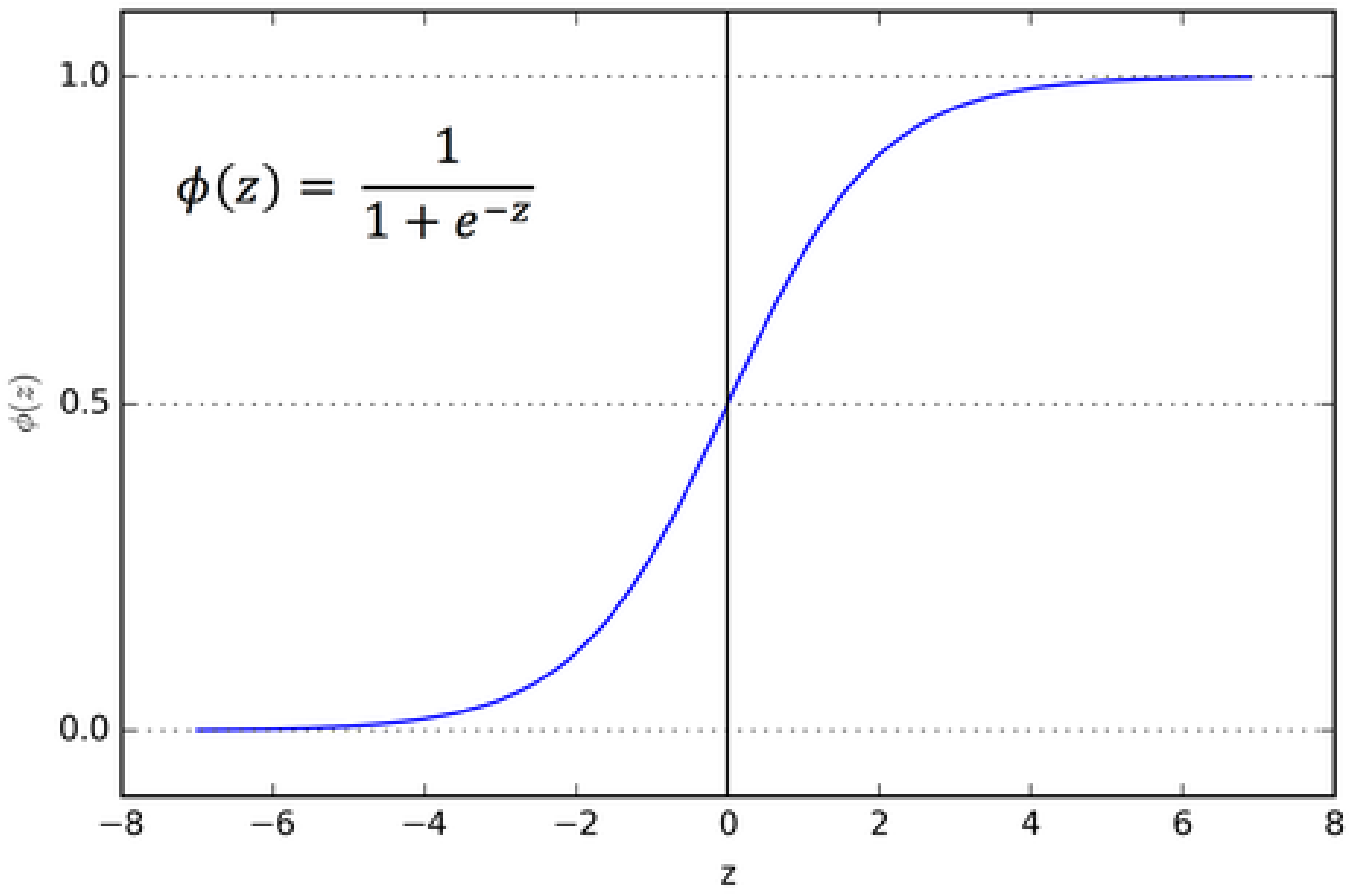
- *The challenge is to find a learning rule that works for multi-layered networks.*

- *NN's with one hidden layer of a sufficient number of units, can compute functions associated with convex classification regions in input space.*

- *NN's with two hidden layers are universal computing devices, although the complexity of the function is limited by the number of units.*
  - **If too few**, the network will be unable to represent the function.
  - **If too many**, the network will memorize examples and is subject to overfitting.

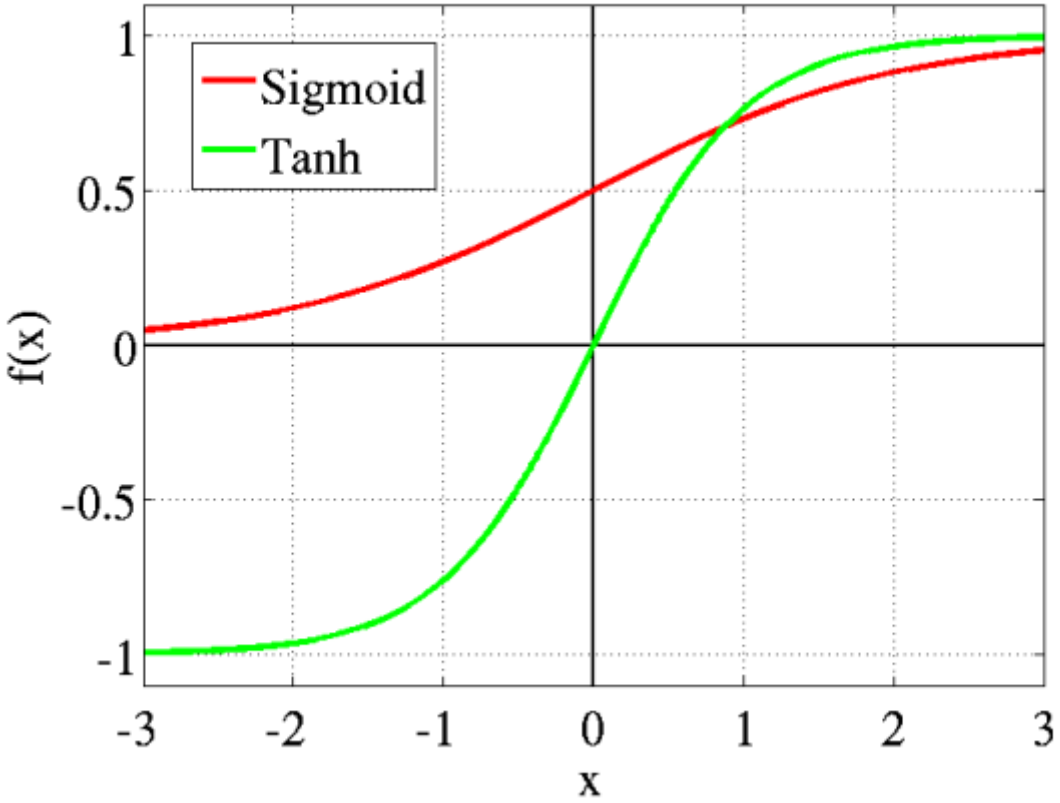
- Basic Idea – go over all existing data patterns, whose labeling is known, and check their classification with a current weight vector
- If correct, continue
- If not, add to the weights a quantity that is proportional to the product of the input pattern with the desired output Z (1 or –1)

# Activation functions -Sigmoid




- sigmoid function is because it exists between **(0 to 1).**
- The Sigmoid Function curve looks like a S-shape
- used for models where we have to **predict the probability** as an output.Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice.

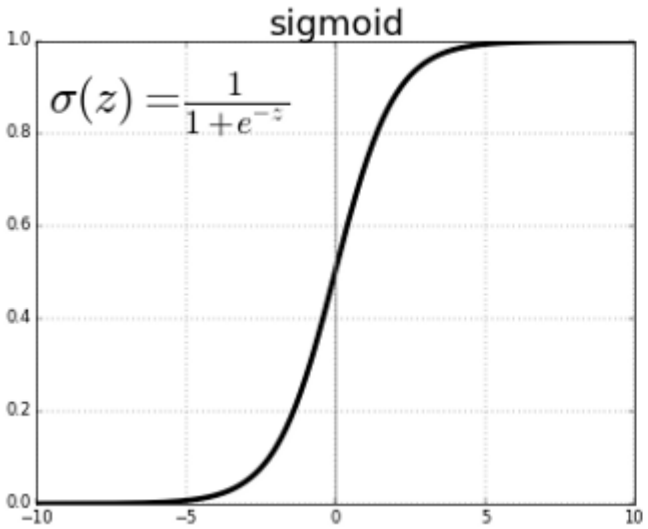
# Activation functions -tanh

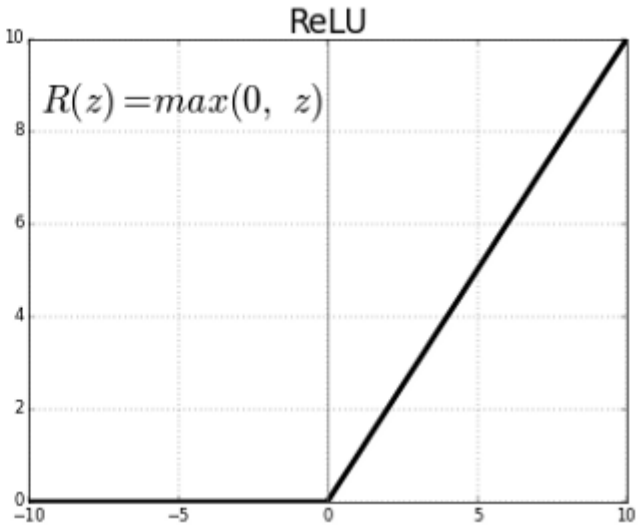



- tanh is also like logistic sigmoid but better.
- range of the tanh function is from (-1 to 1).
- tanh is also sigmoidal (s - shaped).
- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph

❖ *Both tanh and logistic sigmoid activation functions are used in feed-forward nets.*

# Activation functions –RELU(Rectified Linear Unit)

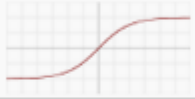
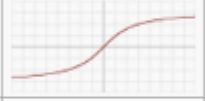



- ReLU is the most used activation function in the world right now
- Used in CNN
- As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

**Range:** [ 0 to infinity)

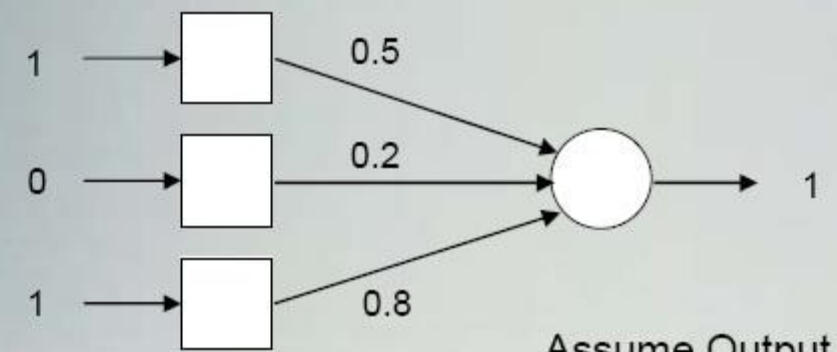
# Activation functions

| Name | Plot | Equation | Derivative |
|---|---|---|---|
| Identity |  | $f(x) = x$ | $f'(x) = 1$ |
| Binary step |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$ |
| Logistic (a.k.a Soft step) |  | $f(x) = \frac{1}{1+e^{-x}}$ | $f'(x) = f(x)(1-f(x))$ |
| TanH |  | $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$ | $f'(x) = 1 - f(x)^2$ |
| ArcTan |  | $f(x) = \tan^{-1}(x)$ | $f'(x) = \frac{1}{x^2+1}$ |
| Rectified Linear Unit (ReLU) |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Parameteric Rectified Linear Unit (PReLU)[2] |  | $f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Exponential Linear Unit (ELU)[3] |  | $f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| SoftPlus |  | $f(x) = \log_e(1+e^x)$ | $f'(x) = \frac{1}{}$ |

# How Do Perceptrons Learn?

- Uses supervised training
- If the output is not correct, the weights are adjusted according to the formula:
  - $W_{new} = W_{old} + \alpha(\text{desired} - \text{output})*\text{input}$

$\alpha$ is the learning rate




1 * 0.5 + 0 * 0.2 + 1 * 0.8 = 1.3

Assuming Output Threshold = 1.2

1.3 > 1.2

Assume Output was supposed to be 0
→ update the weights

Assume $\alpha = 1$

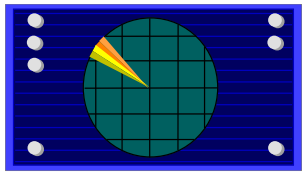
$W_{1new} = 0.5 + 1*(0-1)*1 = -0.5$
$W_{2new} = 0.2 + 1*(0-1)*0 = 0.2$
$W_{3new} = 0.8 + 1*(0-1)*1 = -0.2$

# APPLICATIONS

# Signature recognition

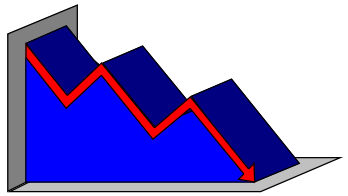
- Each person's signature is different.
- There are structural similarities which are difficult to quantify.
- One company has manufactured a machine which recognizes signatures to within a high level of accuracy.
  - Considers speed in addition to gross shape.
  - Makes forgery even more difficult.

# Sonar target recognition



- Distinguish mines from rocks on sea-bed
- The neural network is provided with a large number of parameters which are extracted from the sonar signal.
- The training set consists of sets of signals from rocks and mines.

# Stock market prediction



- "Technical trading" refers to trading based solely on known statistical parameters; e.g. previous price
- Neural networks have been used to attempt to predict changes in prices.
- Difficult to assess success since companies using these techniques are reluctant to disclose information.

# Mortgage assessment

- Assess risk of lending to an individual.
- Difficult to decide on marginal cases.
- Neural networks have been trained to make decisions, based upon the opinions of expert underwriters.
- Neural network produced a 12% reduction in delinquencies compared with human experts.

# Neural Network Problems

- Many Parameters to be set
- Overfitting
- long training times
- ...

# Parameter setting

- Number of layers
- Number of neurons
    - too many neurons, require more training time
- Learning rate
    - from experience, value should be small ~0.1
- Momentum term
- ..

# Over-fitting

- With sufficient nodes can classify any training set exactly
- May have poor generalisation ability.
- Cross-validation with some patterns
  - Typically 30% of training patterns
  - Validation set error is checked each epoch
  - Stop training if validation error goes up

# Training time

- How many epochs of training?
  - Stop if the error fails to improve (has reached a minimum)
  - Stop if the rate of improvement drops below a certain level
  - Stop if the error reaches an acceptable level
  - Stop when a certain number of epochs have passed