Lecture 4:
“Artificial Neural Network”

Dr. Ehab Essa

Computer Science Department, Mansoura University

Today

- Artificial Neural Network
- Biological Neural Network
- McCulloch-Pitts
- Hebb Net

- Readings
  - **Chapter 1 and 2**, Fundamentals of Neural Networks: Architectures, Algorithms, and Applications by Laurene Fausett

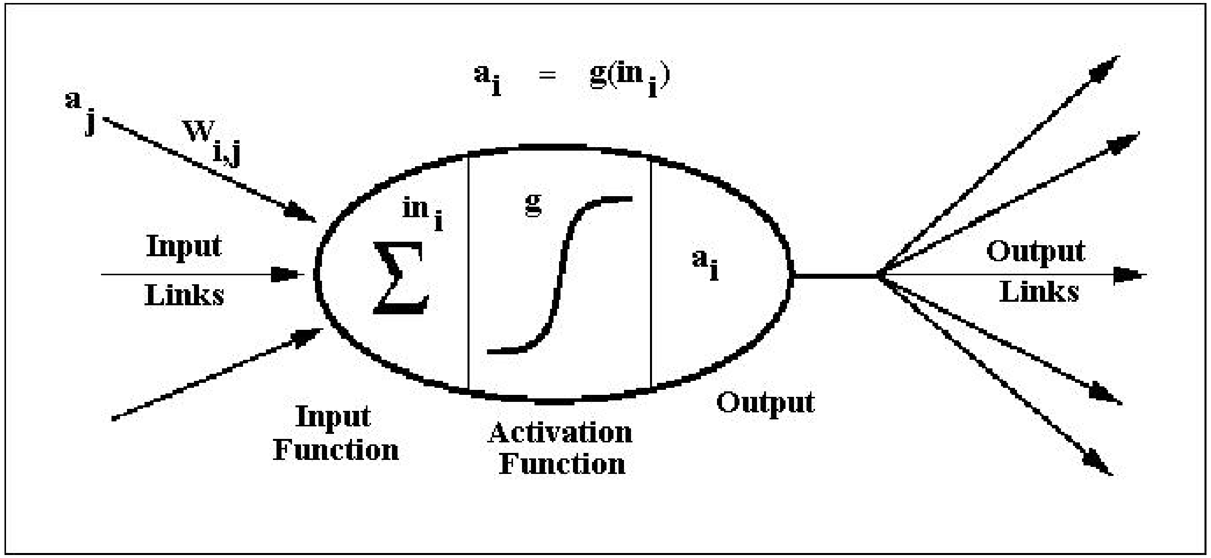
Neural Network Definition

- An *artificial neural network* is an information-processing system that has certain performance characteristics in common with biological neural networks.

- Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:
  - Information processing occurs at many simple elements called neurons.
  - Signals are passed between neurons over connection links.
  - Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
  - Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

Neural Network Definition

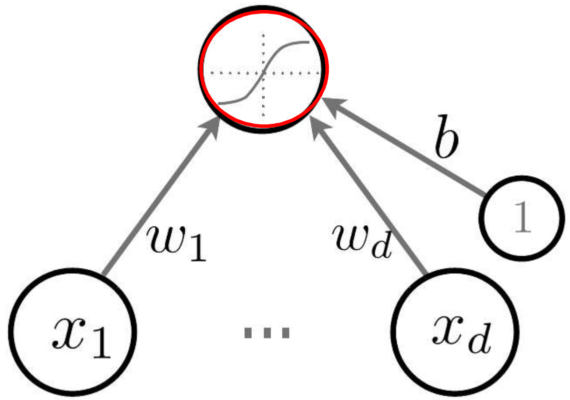
- A neural network is characterized by
  1. its pattern of connections between the neurons (called its *architecture),*
  2. its method of determining the weights on the connections (called its *training,* or *learning, algorithm),* and
  3. its *activation function*



Artificial Neuron

- Artificial neuron is the basic building block that construct complicated neural networks.
  - an artificial neuron is a computational unit which will make a particular computation based on other units it's connected to.

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$



- Neuron (output) activation

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

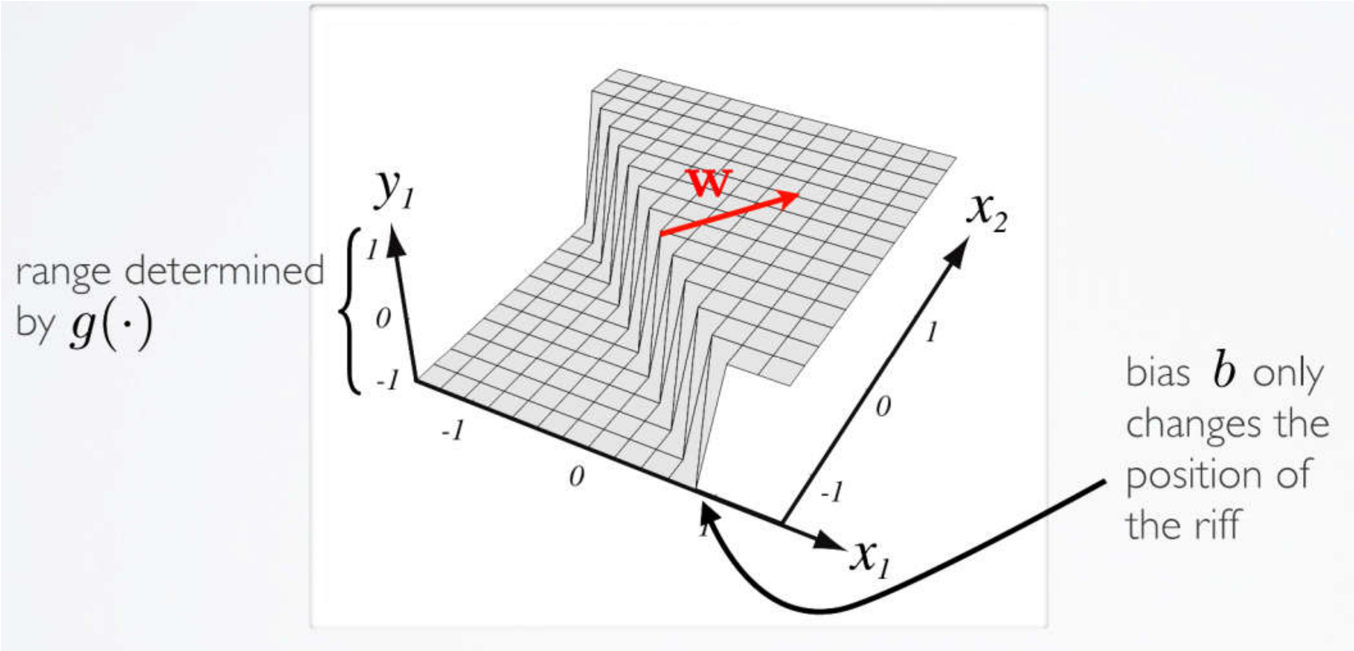
$\mathbf{w}$ are the connection weights

b is the neuron bias

$g(\cdot)$ is called the activation function

Artificial Neuron

- Topics: connection weights, bias, activation function

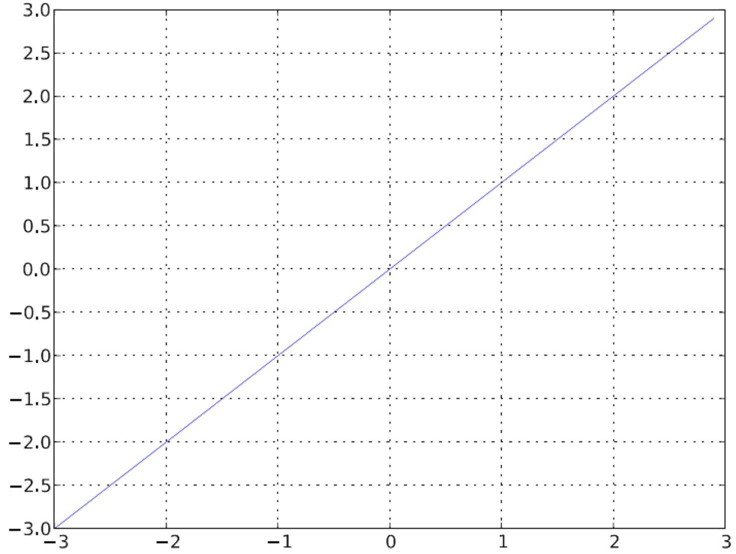


(from Pascal Vincent's slides)

a bias value allows you to shift the activation function to the left or right.

ACTIVATION FUNCTION

- Topics: **linear activation function**
- Performs no input squashing
- Not very interesting...

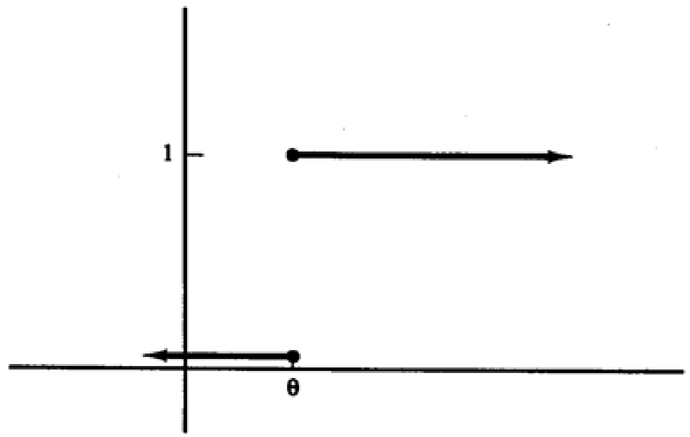


$$g(a) = a$$

Activation Functions

(ii) Binary step function (with threshold):

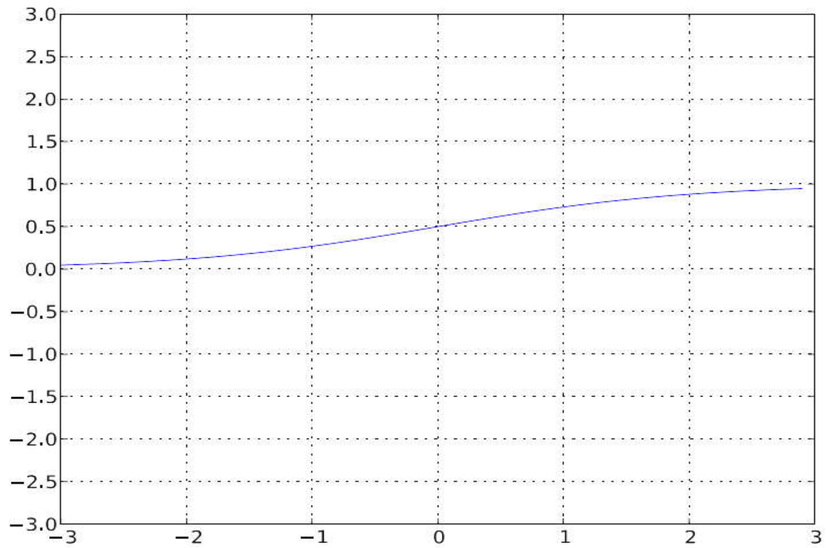
- In early model, single-layer nets use a step function to convert the net input, which is a continuously valued variable, to an output unit that is a binary (l or 0) or bipolar (l or -1)
- The binary step function is also known as the threshold function or Heaviside function.



$$g(a) = \begin{cases} 1 & \text{if } a \geq \theta \\ 0 & \text{if } a < \theta \end{cases}$$

ACTIVATION FUNCTION

- Topics: **sigmoid activation function**
- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

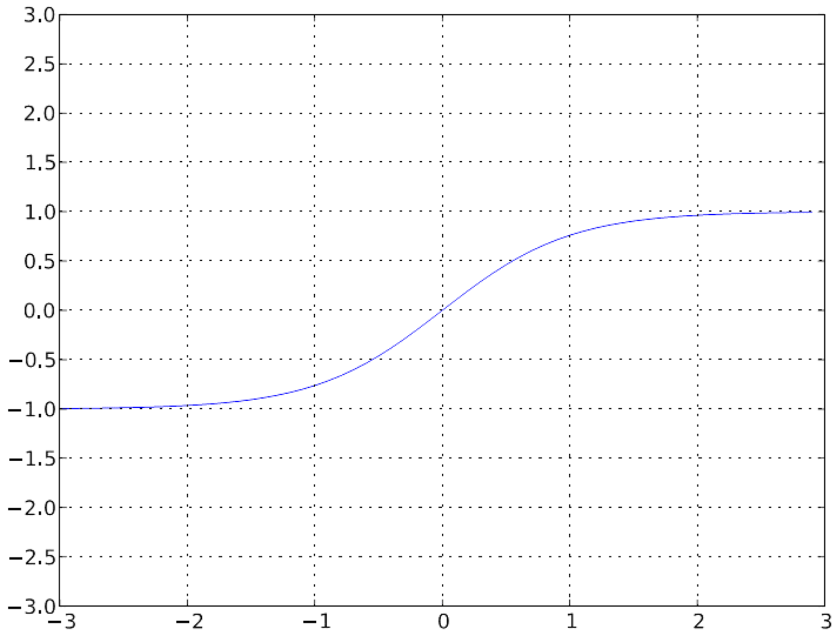


$$g(a) = \mathrm{sigm}(a) = \frac{1}{1+\exp(-a)}$$

$$g'(a) = g(a)(1 - g(a))$$

ACTIVATION FUNCTION

- Topics: **hyperbolic tangent** (''tanh'') activation function
- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

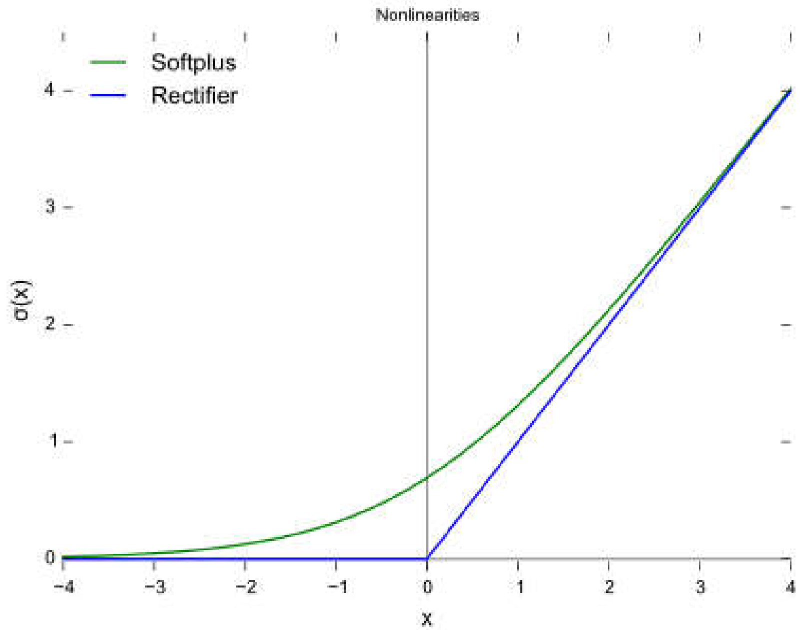


$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

$$g'(a) = (1 - g(a)^2)$$

ACTIVATION FUNCTION

- Topics: **rectified linear activation function**
- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities



- A smooth approximation to the rectifier is a softplus function

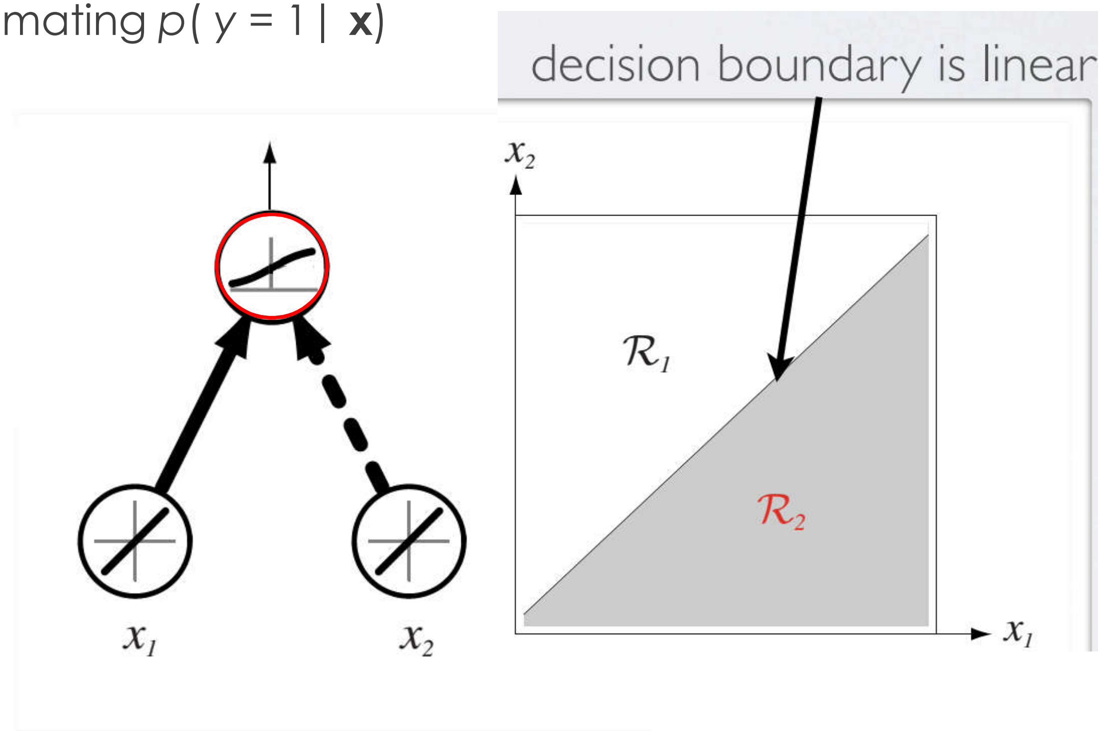
$$g(a) = ln(1 + e^{a})$$

$$g(a) = \mathrm{reclin}(a) = \max(0, a)$$

$$g'(a) = 1, \text{if a} > 0$$

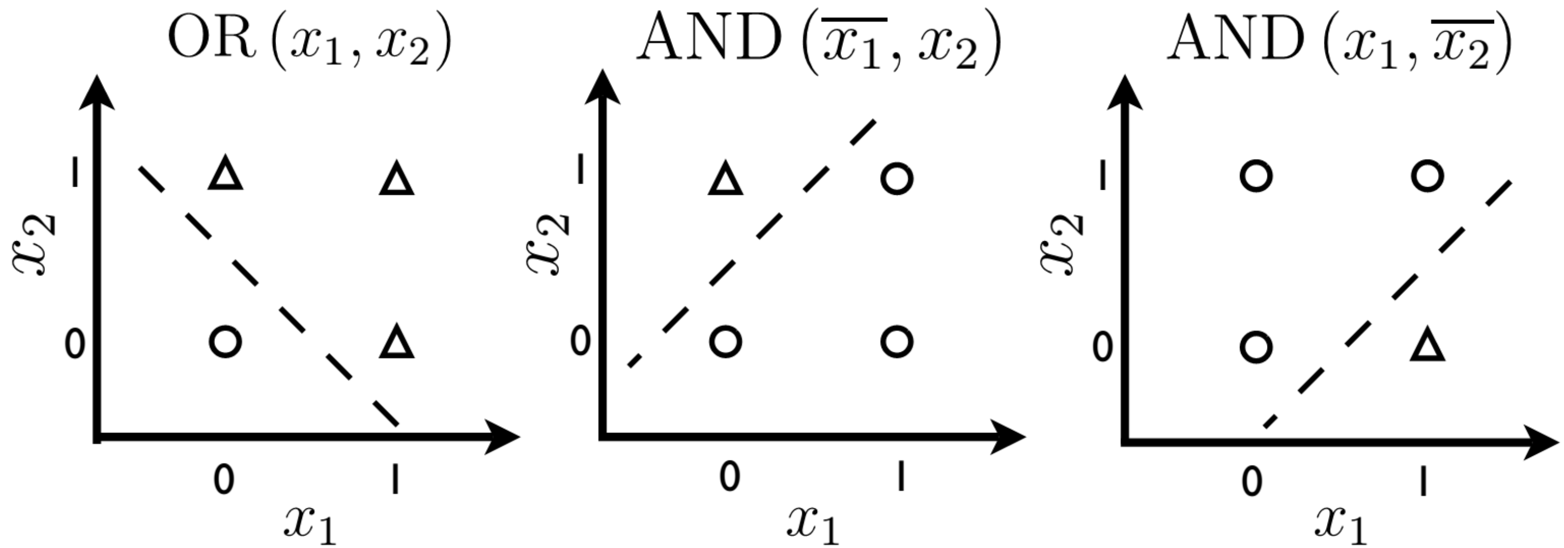
Capacity of single neuron

- Could do binary classification:
- with sigmoid, can interpret neuron as estimating $p(\, y = 1 \mid \mathbf{x})$
- also known as logistic regression classifier
  - if greater than 0.5, predict class 1
  - otherwise, predict class 0



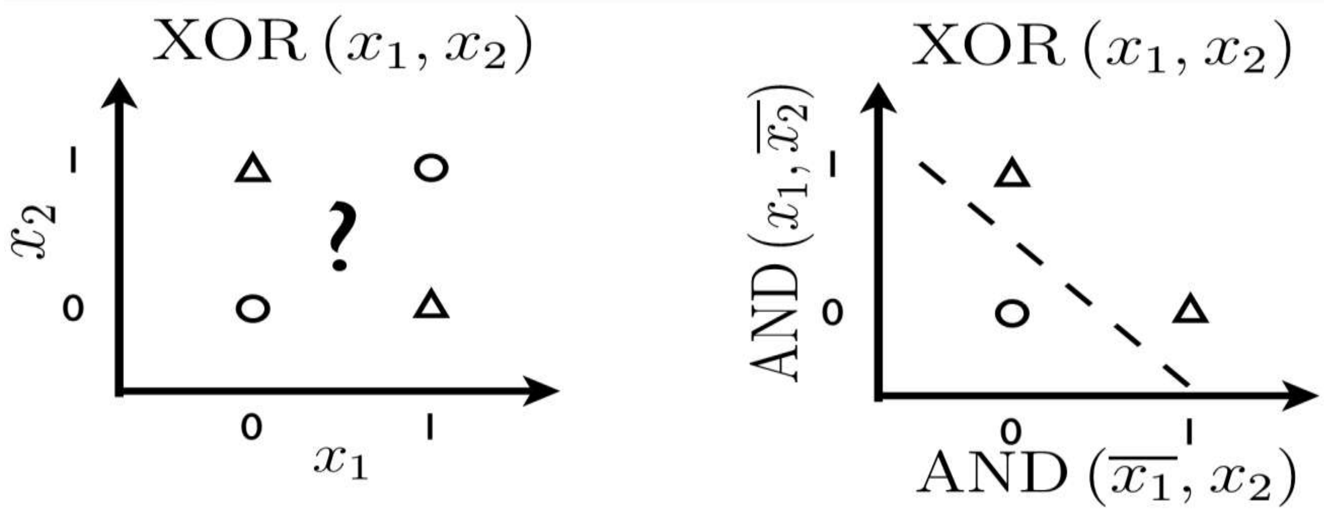
Capacity of single neuron

- Can solve linearly separable problems



Capacity of single neuron

- Can't solve non linearly separable problems...

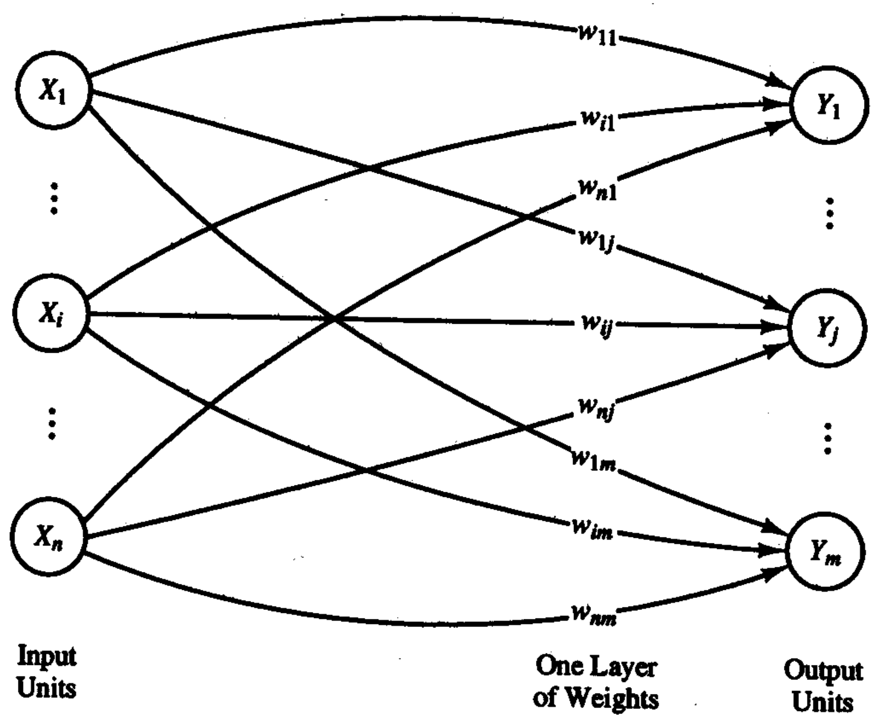


- ... unless the input is transformed in a better representation

NETWORK ARCHITECTURES

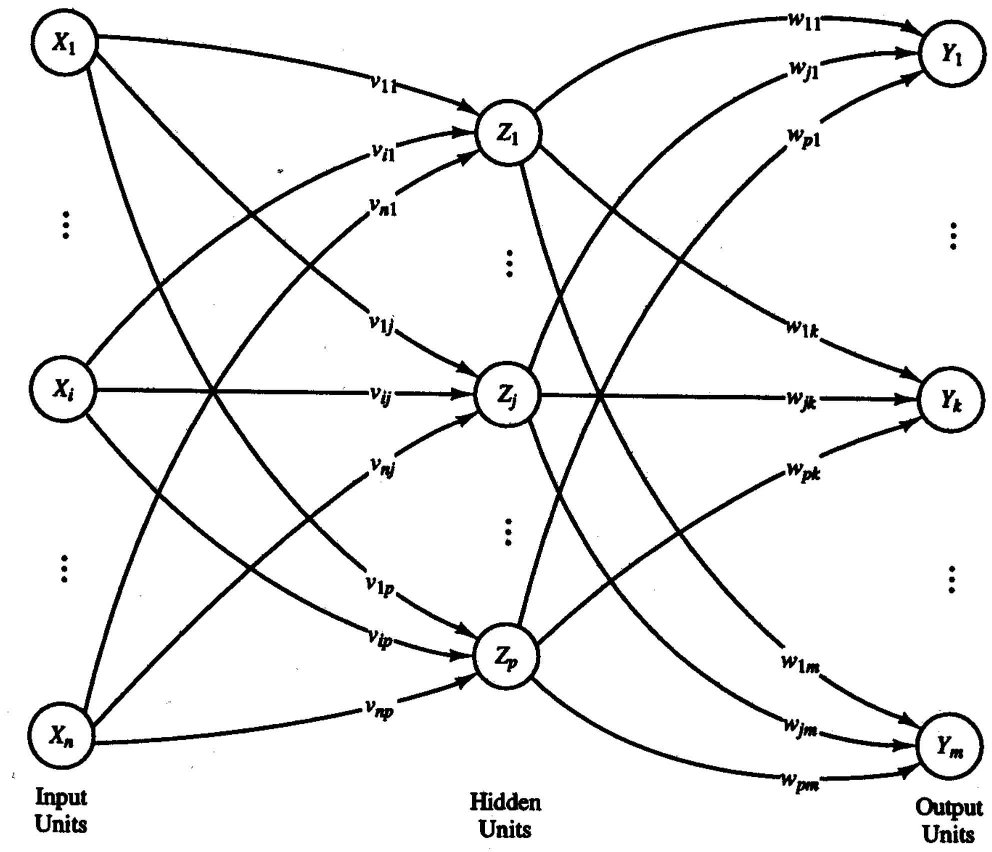
- The arrangement of neurons into layers and the connection patterns within and between layers is called the *net architecture*.

- **Single-Layer (Feedforward) Networks**
  - an input layer of source nodes that projects onto an output layer of neurons.
  - the input units are fully connected to output units but are not connected to other input units, and the output units are not connected to other output units.



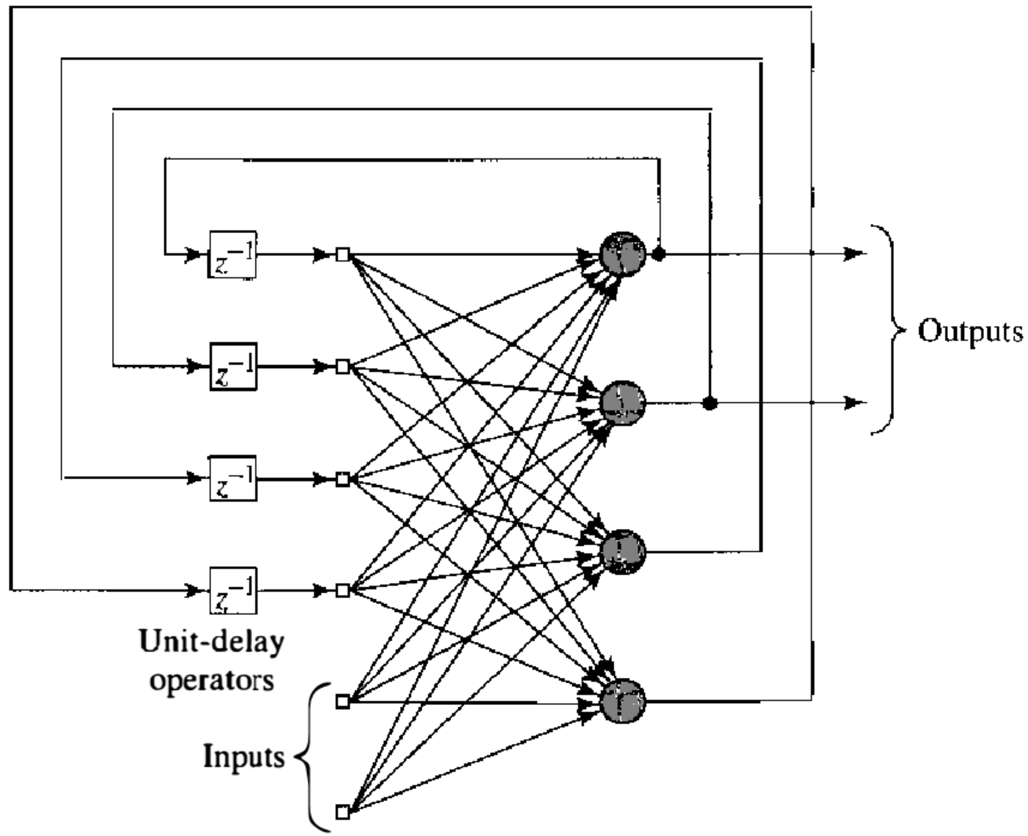
NETWORK ARCHITECTURES

- **Multilayer (Feedforward) Networks**
  - a net with one or more layers (or levels) of nodes (the so-called hidden units) between the input units and the output units.
  - Typically, there is a layer of weights between two adjacent levels of units (input, hidden, or output).
  - Multilayer nets can solve more complicated problems than can single-layer nets, but training may be more difficult.



NETWORK ARCHITECTURES

- **Recurrent or feedback net**
  - it has at least one feedback loop
  - This creates an internal state of the network which allows it to exhibit dynamic temporal behavior.
  - Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs.



Single-hidden Layer Networks

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

y_in $\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$

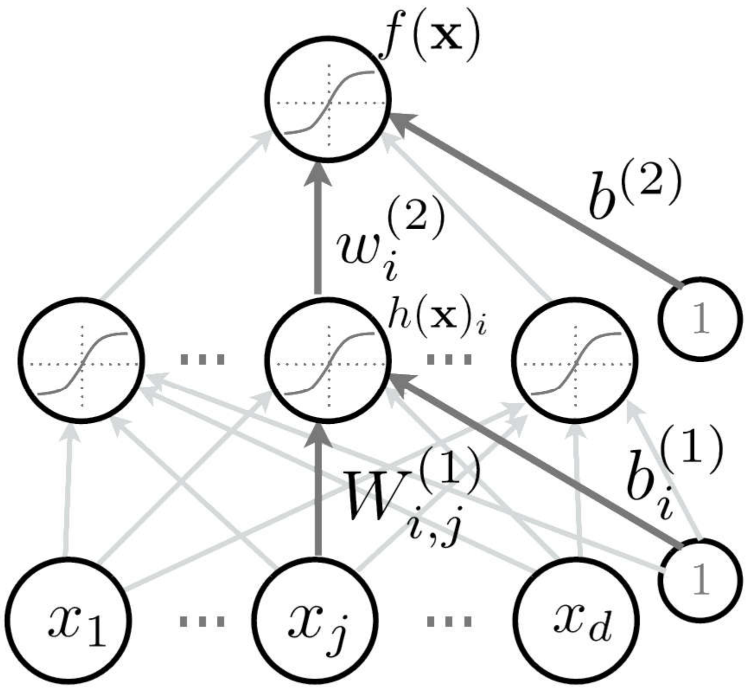
- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)}{}^{\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

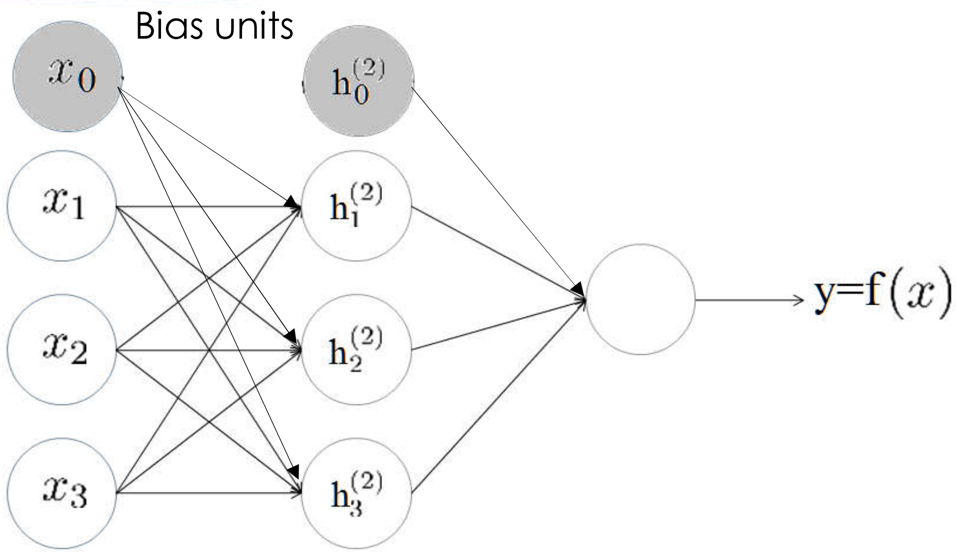
output activation function



Single-hidden Layer Networks

$h_i^{(j)}$ = "activation" of unit i in layer j

$w^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$



$$h_1^2 = g(w_{10}^{(1)}x_0 + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3)$$
$$h_2^2 = g(w_{20}^{(1)}x_0 + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3)$$
$$h_3^2 = g(w_{30}^{(1)}x_0 + w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3)$$

$$y = h_1^3 = g(w_{10}^{(2)}h_0^2 + w_{11}^{(2)}h_1^2 + w_{12}^{(2)}h_2^2 + w_{13}^{(2)}h_3^2)$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $w^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

multilayer neural network

- Could have L hidden layers:
  - layer pre-activation for $k>0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

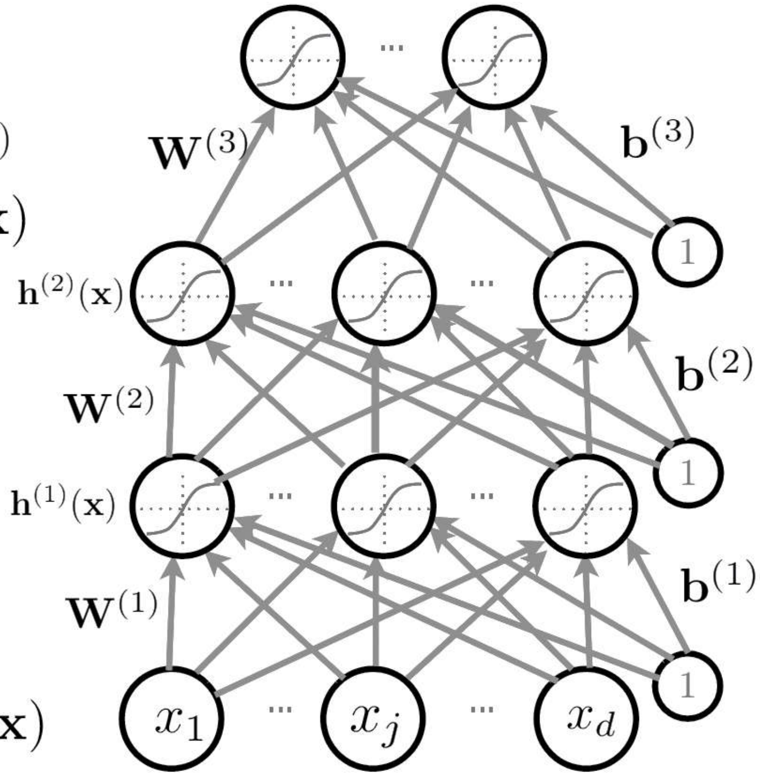
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation ($k=L+1$):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



Softmax activation function

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p(y = c|\mathbf{x})$

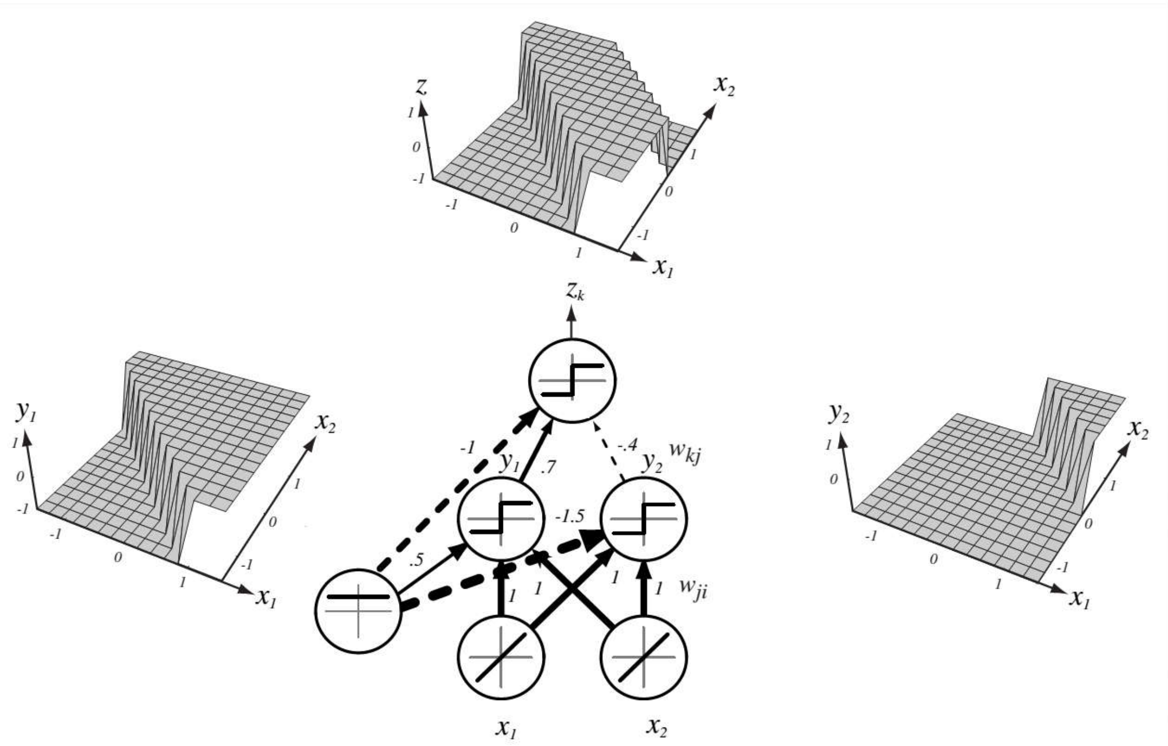
$$\mathbf{o}(\mathbf{a}) = \mathrm{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)}\right]^\top$$

Vector of output neurons, where a_c is pre-activation of neuron c.

- We use the softmax activation function at the output:
  - strictly positive
  - sums to one
- Predicted class is the one with highest estimated probability

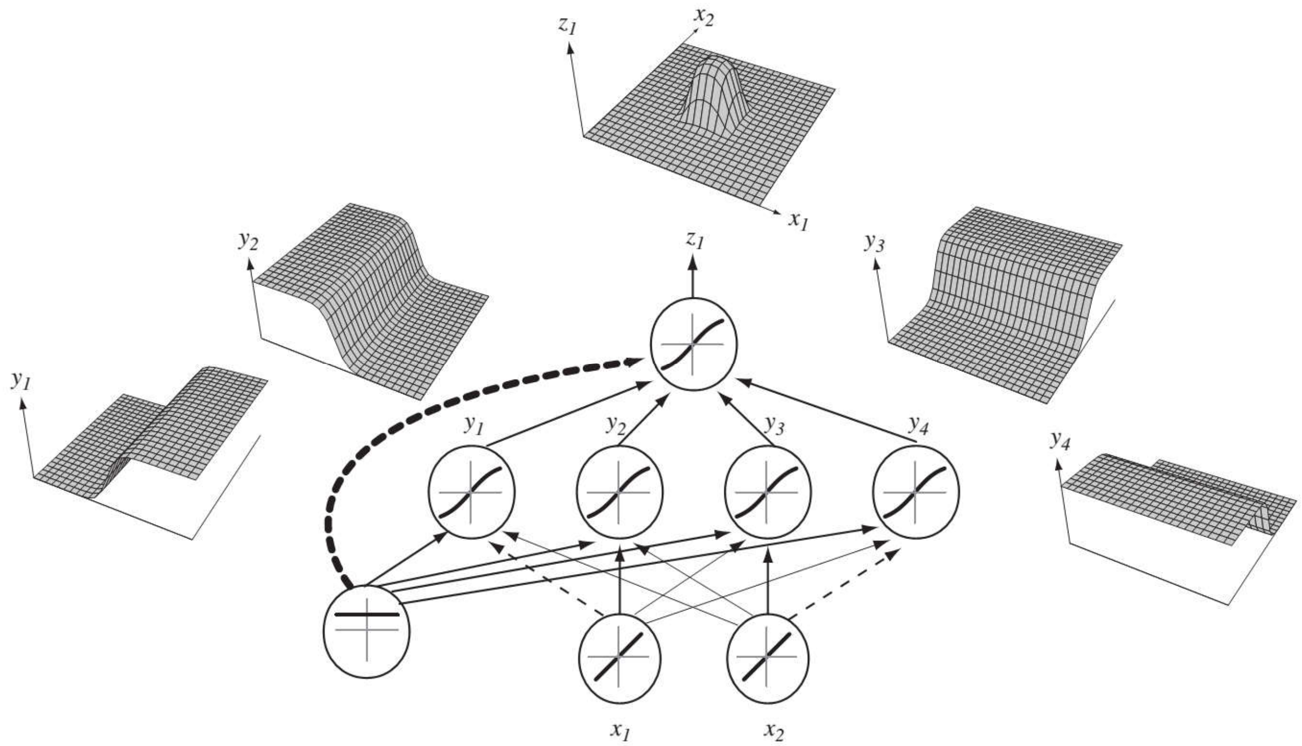
CAPACITY OF NEURAL NETWORK

- single hidden layer neural network



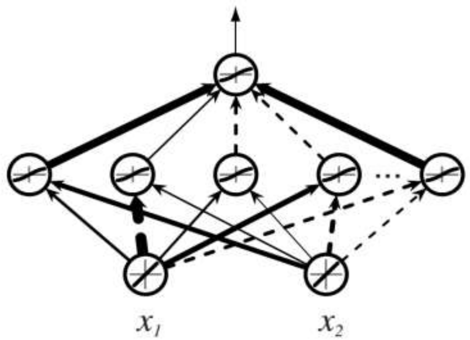
CAPACITY OF NEURAL NETWORK

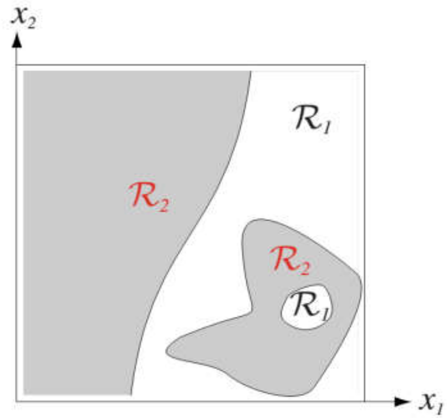
- Single hidden layer neural network



Universal approximation

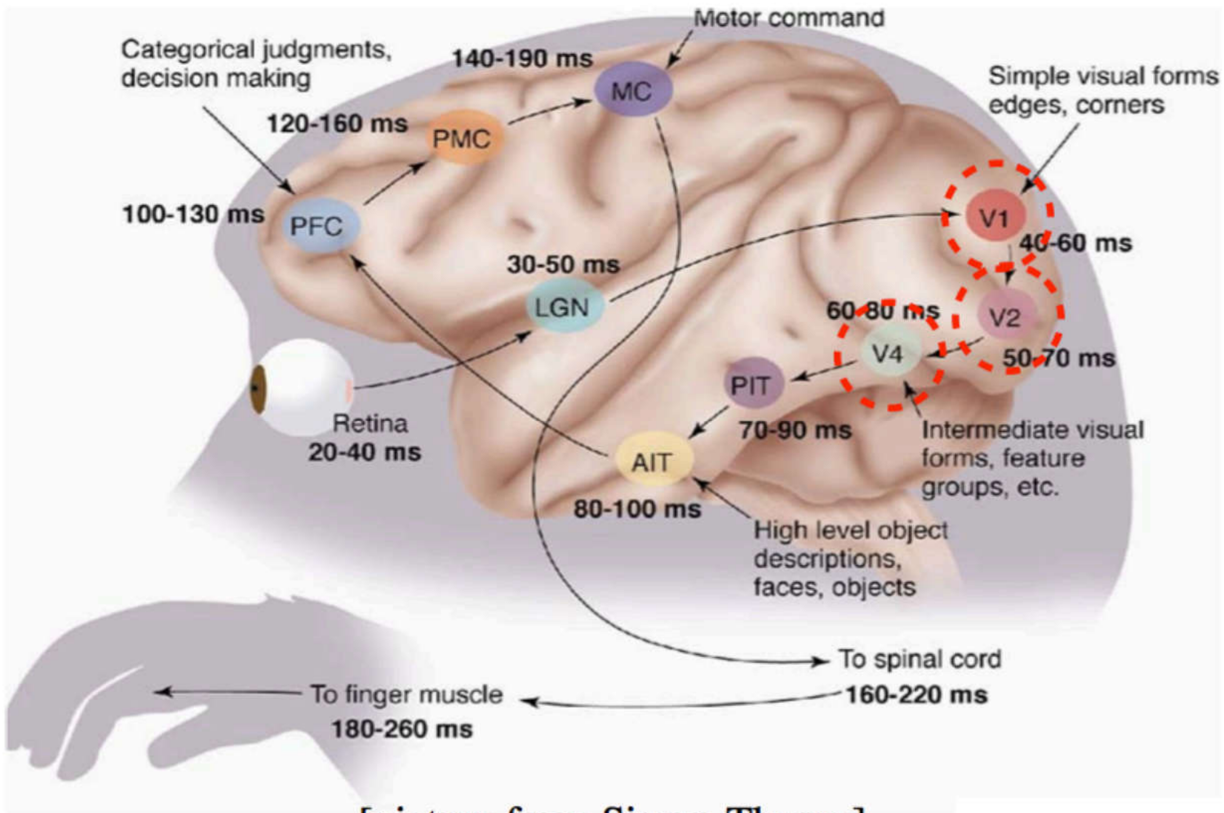
- Universal approximation theorem (Hornik, 1991):
  - ''a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units''
- The result applies for sigmoid, tanh and many other hidden layer activation functions.
- This is a good result, but it doesn't mean there is a learning algorithm that can find the necessary parameter values!





Visual Cortex

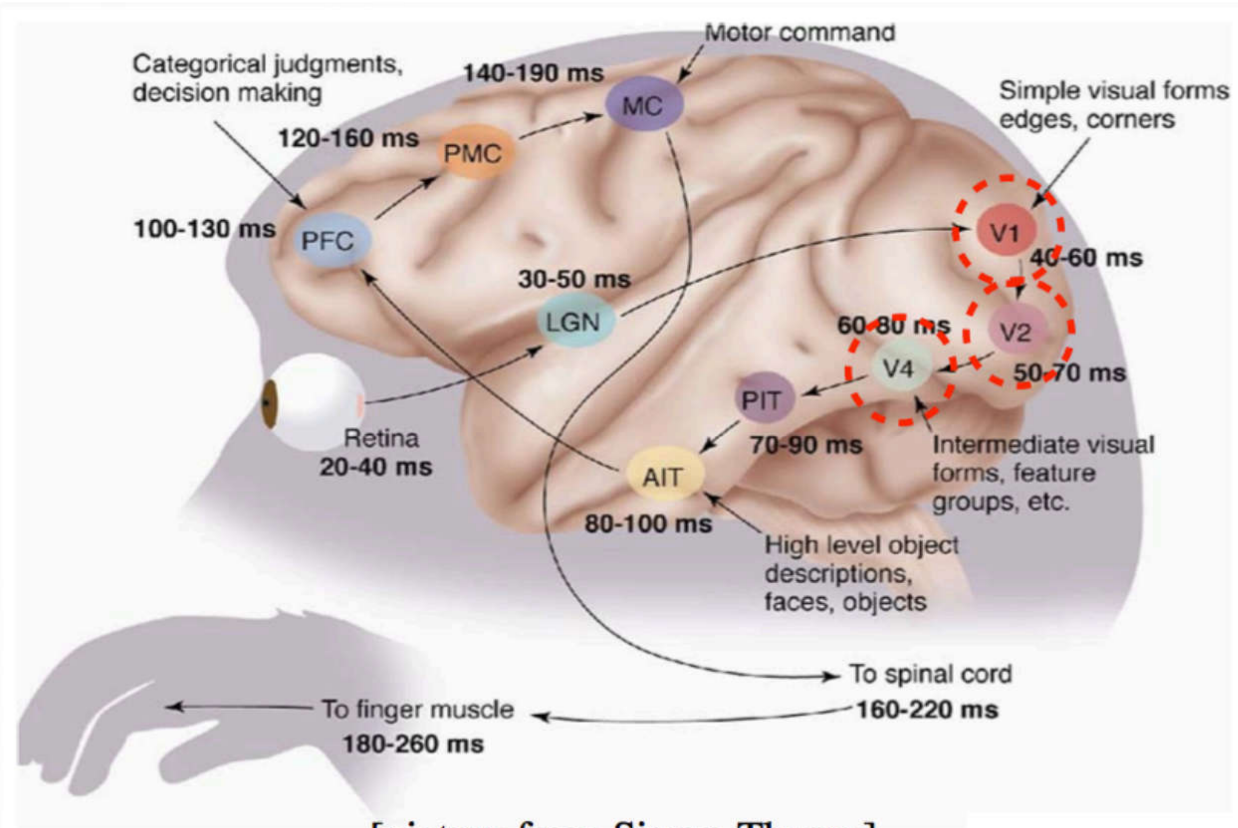
- The **visual cortex** is the part of the brain responsible for processing visual information that we get from our retina.



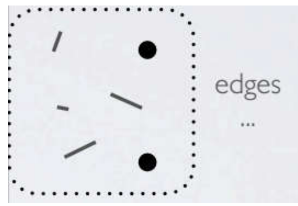
[picture from Simon Thorpe]

Visual Cortex

- **V1** has a set of neurons (simple cells) that are sensitive to particular simple visual forms like edges and corners

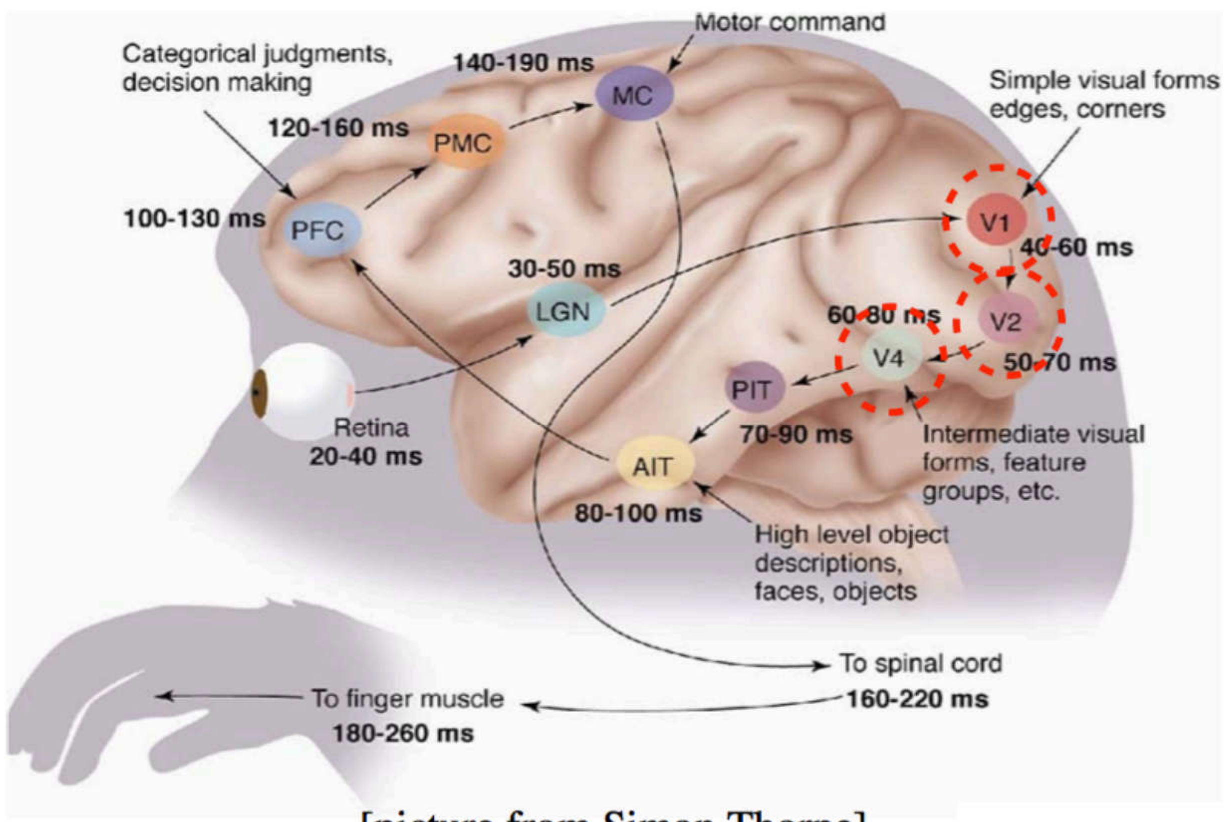


[picture from Simon Thorpe]

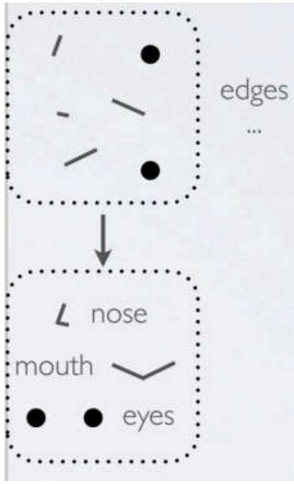


Visual Cortex

- **V4** is tuned for object features of intermediate complexity, like simple geometric shapes.

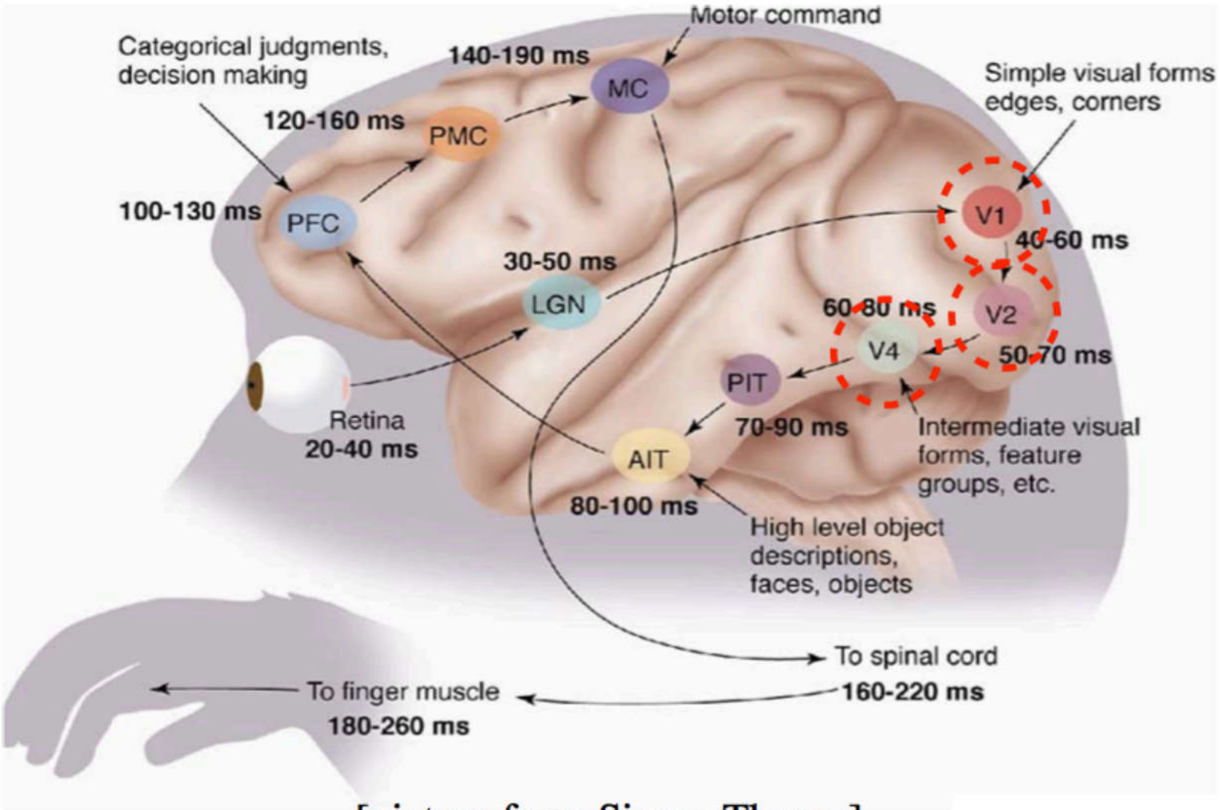


[picture from Simon Thorpe]

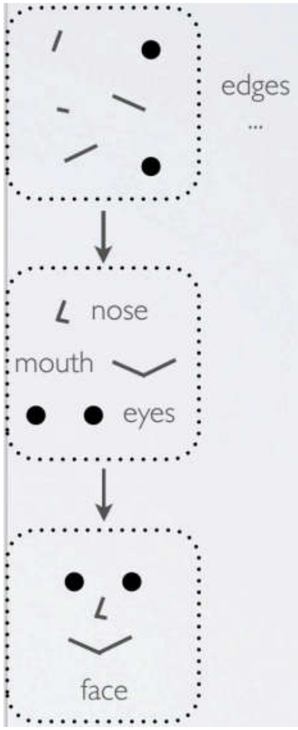


Visual Cortex

- AIT

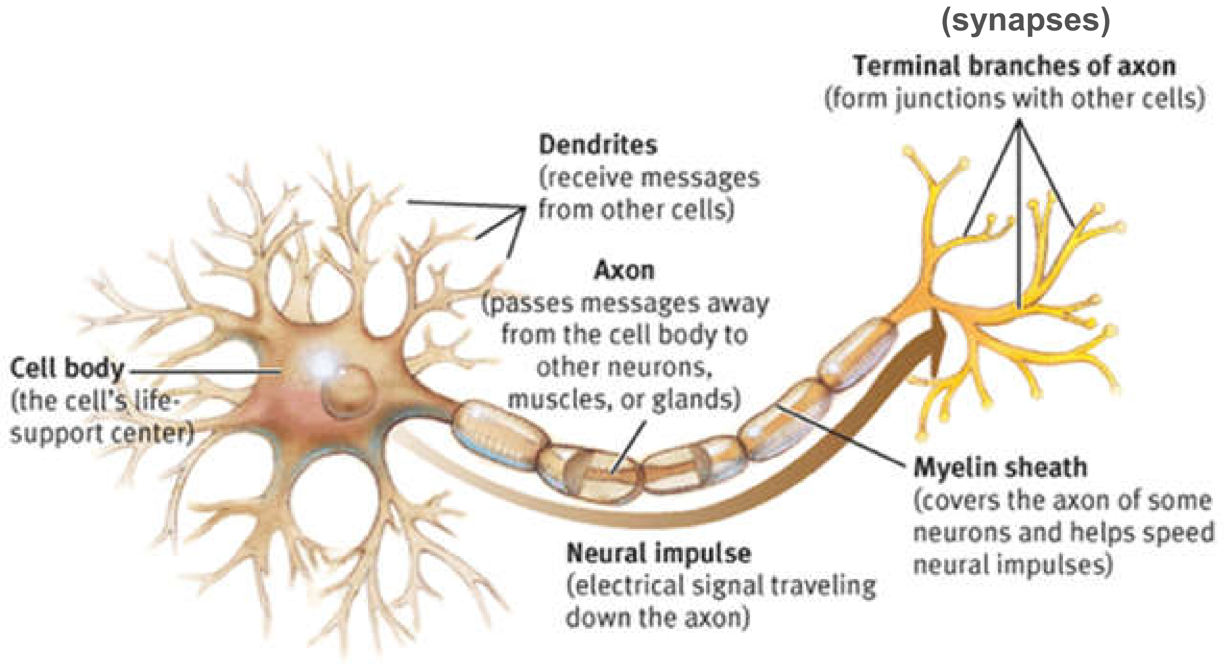


[picture from Simon Thorpe]



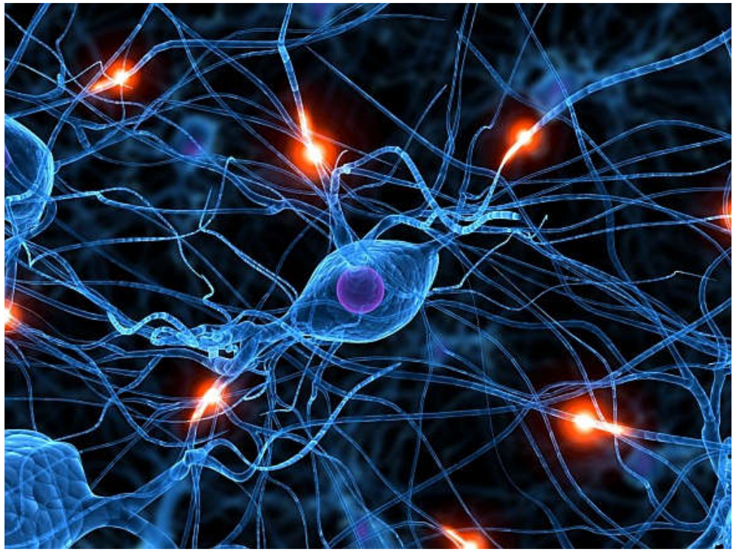
BIOLOGICAL NEURONS

- We estimate around 10^{10} and 10^{11} the number of neurons in the human brain:
  - they receive information from other neurons through their **dendrites**
  - the ''process'' the information in their **cell body** (soma)
  - they send information through a ''cable'' called an **axon**
  - the point of connection between the axon branches and other neurons' dendrites are called **synapses**, which regulate a chemical connection whose strength affects the input to the cell.



BIOLOGICAL NEURONS

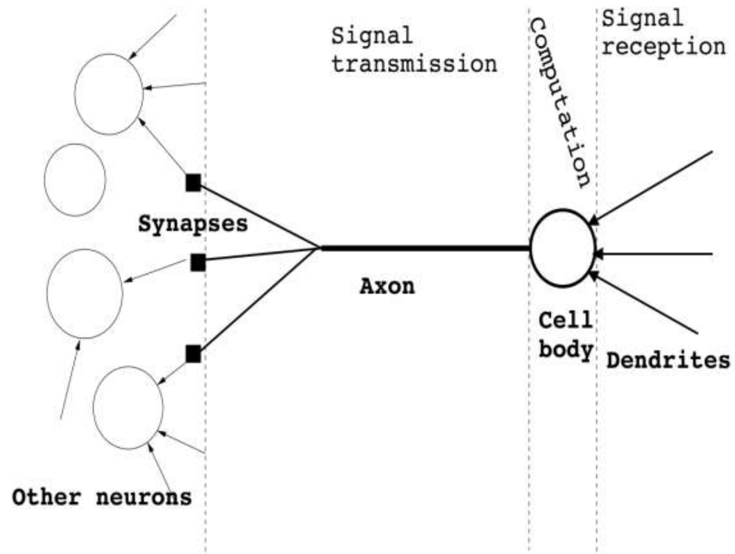
- An **action potential** is an electrical impulse that travels through the axon:
  - this is how neurons communicate
  - it generates a ''spike'' in the electric potential (voltage) of the axon
  - an action potential is generated at neuron only if it receives enough (over some threshold) of the ''right'' pattern of spikes from other neurons



- Neurons can generate several such spikes every seconds:
  - the frequency of the spikes, called **firing rate**, is what characterizes the activity of a neuron
  - neurons are always firing a little bit, (spontaneous firing rate), but they will fire more, given the right stimulus

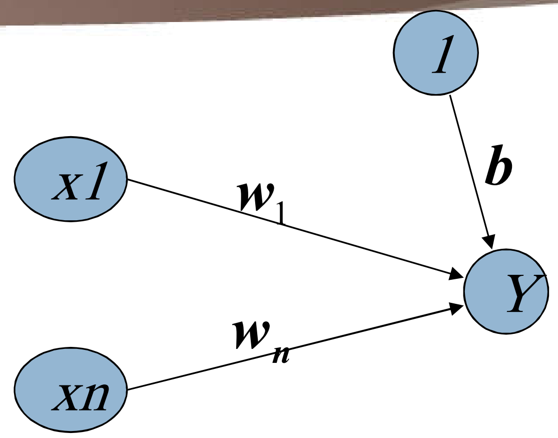
BIOLOGICAL NEURONS

- Firing rates of different input neurons combine to influence the firing rate of other neurons:
  - depending on the dendrite and axon, a neuron can either work to increase (excite) or decrease (inhibit) the firing rate of another neuron
- This is what artificial neurons approximate:
  - the activation corresponds to a ‘‘sort of ’’ firing rate
  - the weights (synapse) between neurons model whether neurons excite or inhibit each other
  - the activation function and bias model the thresholded behaviour of action potentials (cell body)



General Artificial Neural Network architecture

Single layer



net input to *Y*: $$y\_in = b + \sum_{i=1}^{n} x_i w_i$$

bias ***b*** is treated as the weight from a special unit with constant output 1. threshold $\boldsymbol{\theta}$ related to *Y*

output $$y = f(y\_in) = \begin{cases} 1 & \text{if } y\_\text{in} \geq \theta \\ -1 & \text{if } y\_\text{in} < \theta \end{cases}$$

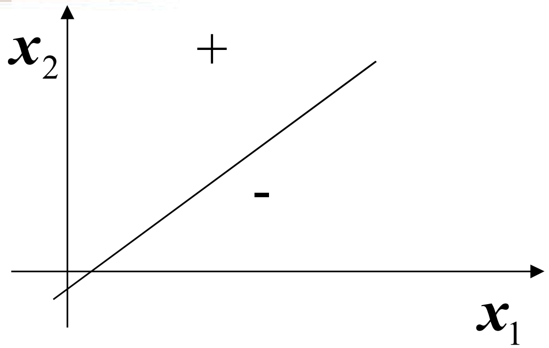
classify $(\boldsymbol{x}_1, \ldots\ldots \boldsymbol{x}_n)$ into one of the two classes

Decision region/boundary

n = 2, b != 0, θ = 0

$$b + x_1 w_1 + x_2 w_2 = 0 \text{ or}$$

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



is a line, called *decision boundary*, which partitions the plane into two decision regions

If a point/pattern (x_1, x_2) is in the positive region, then

$b + x_1 w_1 + x_2 w_2 \geq 0$, and the output is one (belongs to class one)

Otherwise, $b + x_1 w_1 + x_2 w_2 < 0$, output –1 (belongs to class two)

n = 2, b = 0, θ != 0 would result a similar partition

Linear Separability Problem

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

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

  then they are said to be linearly separable. The simple network can correctly classify any patterns.
- Decision boundary (i.e., ***W*, *b*** or θ) 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 network , more sophisticated architecture is needed.

The McCulloch-Pitts Model

- The McCulloch-Pitts neuron is perhaps the earliest artificial neuron [McCulloch & Pitts, 1943].

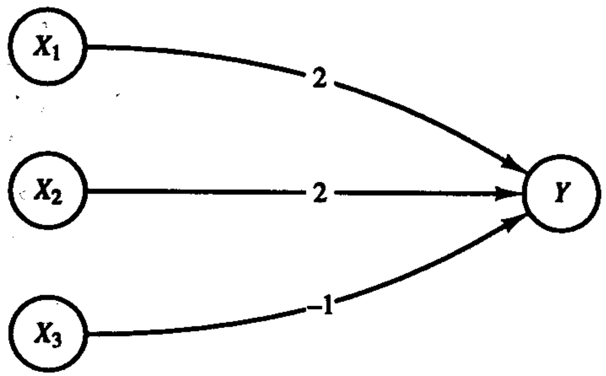
- The activation of a McCulloch-Pitts neuron is binary. That is, at any time step, the neuron either fires (has an activation of 1) or does not fire (has an activation of 0).

- Each neuron has a fixed threshold such that if the net input to the neuron is greater than the threshold, the neuron fires.

$$f(y\_in) = \begin{cases} 1 & \text{if } y\_in \geq \theta \\ 0 & \text{if } y\_in < \theta \end{cases}$$

The McCulloch-Pitts Model

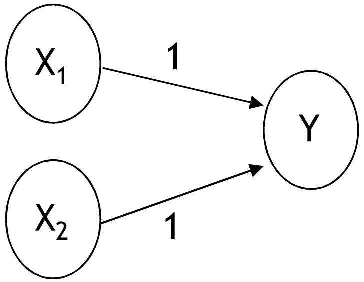
- **Architecture**
  - In general, a McCulloch-Pitts neuron *Y* may receive signals from any number of other neurons.
  - Each connection path is either excitatory, with weight $w > 0$, or inhibitory, with weight $-p$ $(p > 0)$.
  - All excitatory connections into a particular neuron have the same weights.



The McCulloch-Pitts Model

- **Algorithm**
  - The weights for a McCulloch-Pitts neuron are set, together with the threshold for the neuron's activation function,
  - The analysis, rather than a training algorithm, is used to determine the values of the weights and threshold.
  - Logic functions will be used as simple examples for a number of neural nets.

McCulloch-Pitts for AND



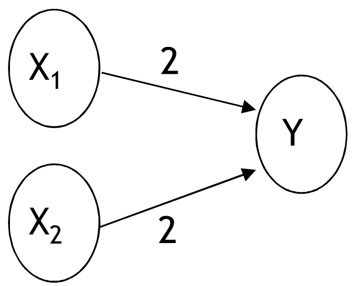
AND Function

AND		
X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

Threshold(Y) = 2

	X1 W1	+ X2 W2	= Yin		output
First pattern	1 * 1	+ 1 * 1	= 2	>= θ	1
second pattern	1 * 1	+ 0 * 1	= 1	< θ	0
third pattern	0 * 1	+ 1 * 1	= 1	< θ	0
fourth pattern	0 * 1	+ 0 * 1	= 0	< θ	0

McCulloch-Pitts for OR



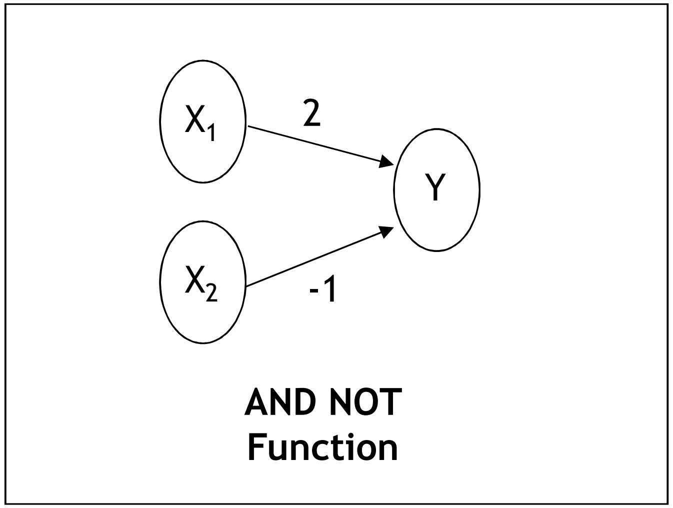
OR Function

Threshold(Y) = 2

OR		
X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

	X1 W1	+ X2 W2	= Yin		output
First pattern	1 * 2	+ 1 * 2	= 4	>= θ	1
second pattern	1 * 2	+ 0 * 2	= 2	>= θ	1
third pattern	0 * 2	+ 1 * 2	= 2	>= θ	1
fourth pattern	0 * 2	+ 0 * 2	= 0	< θ	0

McCulloch-Pitts for AND NOT

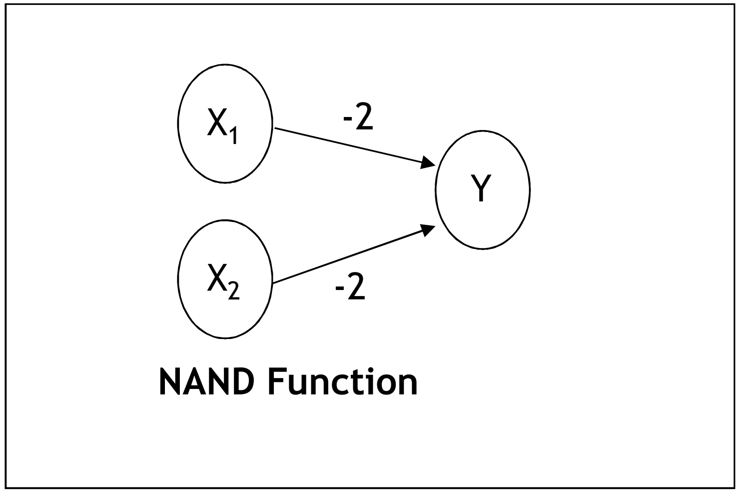


Threshold(Y) = 2

AND NOT			
X1	X2	X'2	Y
1	1	0	0
1	0	1	1
0	1	0	0
0	0	1	0

	X1 W1	+ X2 W2	= Yin			output
First pattern	1 * 2	+ 1 * -1	= 1	<	θ	0
second pattern	1 * 2	+ 0 * -1	= 2	>=	θ	1
third pattern	0 * 2	+ 1 * -1	= -1	<	θ	0
fourth pattern	0 * 2	+ 0 * -1	= 0	<	θ	0

McCulloch-Pitts for NAND

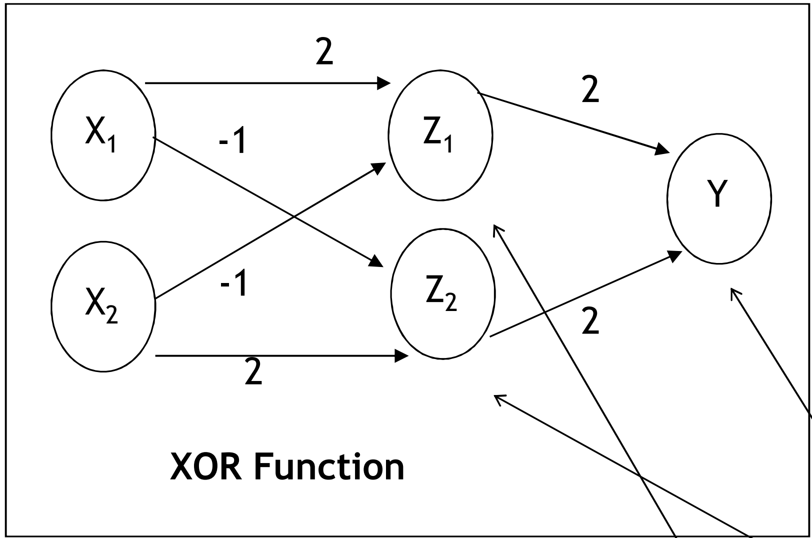


Threshold(Y) = -3

x_1	x_2	NAND	
0	0	1	) $0 \geq \theta$
0	1	1	) $w_2 \geq \theta$
1	0	1	) $w_1 \geq \theta$
1	1	0	) $w_1 + w_2 < \theta$

(e.g.: $w_1 = w_2 = -2$, $\theta = -3$)

McCulloch-Pitts for XOR



Threshold(Z_1, Z_2 ,Y) = 2

XOR		
X1	X2	Y
1	1	0
1	0	1
0	1	1
0	0	0

X_1 XOR X_2 = (X_1 AND NOT X_2) OR (X_2 AND NOT X_1)

Hebb Nets

- Hebb, in his influential book *The organization of Behavior* (1949), claimed
  - Behavior changes are primarily due to the changes of synaptic strengths (w_{ij}) between neurons I and j
  - w_{ij} increases only when both neurons I and j are "on": the **Hebbian learning law**
  - In ANN, Hebbian law can be stated: w_{ij} increases only if the outputs of both units x_i and y_j have the same sign.
  - In our simple network (one output and n input units)

$$w_{ij}(new) = w_{ij}(old) + x_i y$$

Hebb Nets

- Hebb net (supervised) learning algorithm (p.49)

```
Step 0.  Initialization: b = 0, w_i = 0, i = 1 to n
Step 1.  For each of the training sample s:t do steps 2 -4
         /* s is the input pattern, t the target output of the sample */
Step 2.      x_i := s_i , (i = 1 to n)              /* set s to input units */
Step 3.      y := t                                 /* set y to the target */
Step 4.      w_i := w_i + x_i * y, (i = 1 to n)     /* update weight */
             b := b + y                             /* update bias */
```

Notes:

1) each training sample is used only once.

2) Weight change can be written $\Delta w_{ij} = x_i y$

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$

- Activation function If binary data : binary step function If bipolar data : bipolar sign function

Examples: AND function

- Hebb net for **and function** : binary input and output

Input	target	Weight changes			weights		
(x1,x2,1)	Y=t	Δw1	Δw2	Δb	w1	w2	b
					0	0	0
(1,1,1)	1						
(1,0,1)	0						
(0,1,1)	0						
(0,0,1)	0						

bias unit

Examples: AND function

$$\Delta w_{ij} = x_i y$$

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$

- Hebb net for **<u>and function</u>** : binary input and output

Input	target	Weight changes			weights		
(x1,x2,1)	Y=t	Δw1	Δw2	Δb	w1	w2	b
					0	0	0
(1,1,1)	1	1	1	1	1	1	1
(1,0,1)	0	0	0	0	1	1	1
(0,1,1)	0	0	0	0	1	1	1
(0,0,1)	0	0	0	0	1	1	1

bias unit

Examples: AND function

- Is it correctly learned after using each sample once?

- **Testing second pattern**
- (1,0) : x1=1 ,,, x 2 = 0 ,,, w1 =1 ,, w2 = 1 ,, b =1 ,, θ = 0
- X1*w1 + x2*w2 + b = 1 + 0 + 1 = 2 > θ → output = 1 which is wrong

Example 2: AND function

- Bipolar units (1, -1)

Input	target	Weight changes			weights		
(x1,x2,1)	Y=t	Δw1	Δw2	Δb	w1	w2	b
					0	0	0
(1,1,1)	1	1	1	1	1	1	1
(1,-1,1)	-1	-1	1	-1	0	2	0
(-1,1,1)	-1	1	-1	-1	1	1	-1
(-1,-1,1)	-1	1	1	-1	2	2	-2

<u>Testing</u> (1,1) : x1=1 ,,, x 2 = 1 ,,, w1 =2 ,, w2 = 2 ,, b =-2 ,, $\theta = 0$
X1w1 + x2w2 + b = 2 + 2 - 2 = 2 > θ → output = 1

<u>Testing</u> (1,-1) : x1=1 ,,, x 2 = -1 ,,, w1 =2 ,, w2 = 2 ,, b =-2 ,, $\theta = 0$

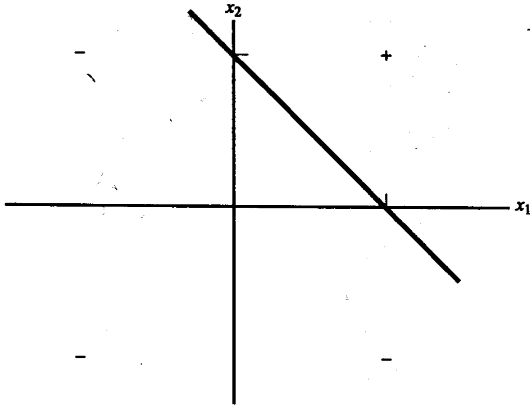
X1w1 + x2w2 + b = 2 - 2 - 2 = -2 < θ → output = -1

Example 2: AND function

- Bipolar units (1, -1)

Input	target	Weight changes			weights		
(x1,x2,1)	Y=t	Δw1	Δw2	Δb	w1	w2	b
					0	0	0
(1,1,1)	1	1	1	1	1	1	1
(1,-1,1)	-1	-1	1	-1	0	2	0
(-1,1,1)	-1	1	-1	-1	1	1	-1
(-1,-1,1)	-1	1	1	-1	2	2	-2

A correct boundary
$-1 + x1 + x2 = 0$
is successfully learned



Testing (-1,1) : x1=-1 ,,, x 2 = 1 ,,, w1 =2 ,, w2 = 2 ,, b =-2 ,, $\theta = 0$
X1w1 + x2w2 + b = -2 + 2 - 2 = -2 < θ → output = -1

Testing (-1,-1) : x1=-1 ,,, x 2 = -1 ,,, w1 =2 ,, w2 = 2 ,, b =-2 ,, $\theta = 0$

X1w1 + x2w2 + b = -2 -2 - 2 = -6 < θ → output = -1

Examples: Character Recognition

- Convert the patterns to input vectors by replace each # with 1 and . with -1

Pattern 1		Pattern 2
# . . . #		. # # # .
. # . # .		# . . . #
. . # . .	and	# . . . #
. # . # .		# . . . #
# . . . #		. # # # .

ect. Pattern 1 then becomes

1 −1 −1 −1 1, −1 1 −1 1 −1, −1 −1 1 −1 −1, −1 1 −1 1 −1,
1 −1 −1 −1 1,

and pattern 2 becomes

−1 1 1 1 −1, 1 −1 −1 −1 1, 1 −1 −1 −1 1, 1 −1 −1 −1 1, −1 1 1 1 −1,

- The correct response for the first pattern is +1 and the second pattern is -1

Hebb Nets

- It will fail to learn x1 ^ x2 ^ x3, even though the function is linearly separable.

- Stronger learning methods are needed.
  - **Error driven**: for each sample s:t, compute y from s based on current W and b, then compare y and t
  - **Use training samples repeatedly**, and each time only change weights slightly (a << 1)
  - Learning methods of Perceptron and Adaline are good examples