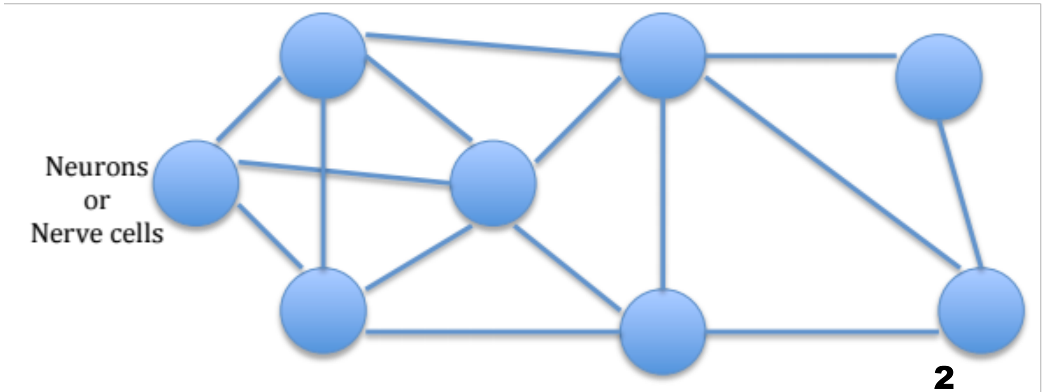
Artificial Neural Networks

Neural networks

- A neural network can be defined as a model of reasoning based on the human brain.
- The brain consists of interconnected set of nerve cells, or basic information-processing units, called **neurons**.
- Signals are propagated from one neuron to another by complex electrochemical reactions.



Neural networks

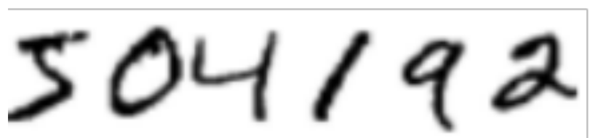
- We know that the human brain has a highly complex, nonlinear, and parallel computer.

- Neural network is complex as well as nonlinear and massively parallel.

- Our brain has counted to have:
  - millions of nerve cells with
  - trillions of interconnections.

Neural networks

- Humans are good at making sense of what our eyes show us.



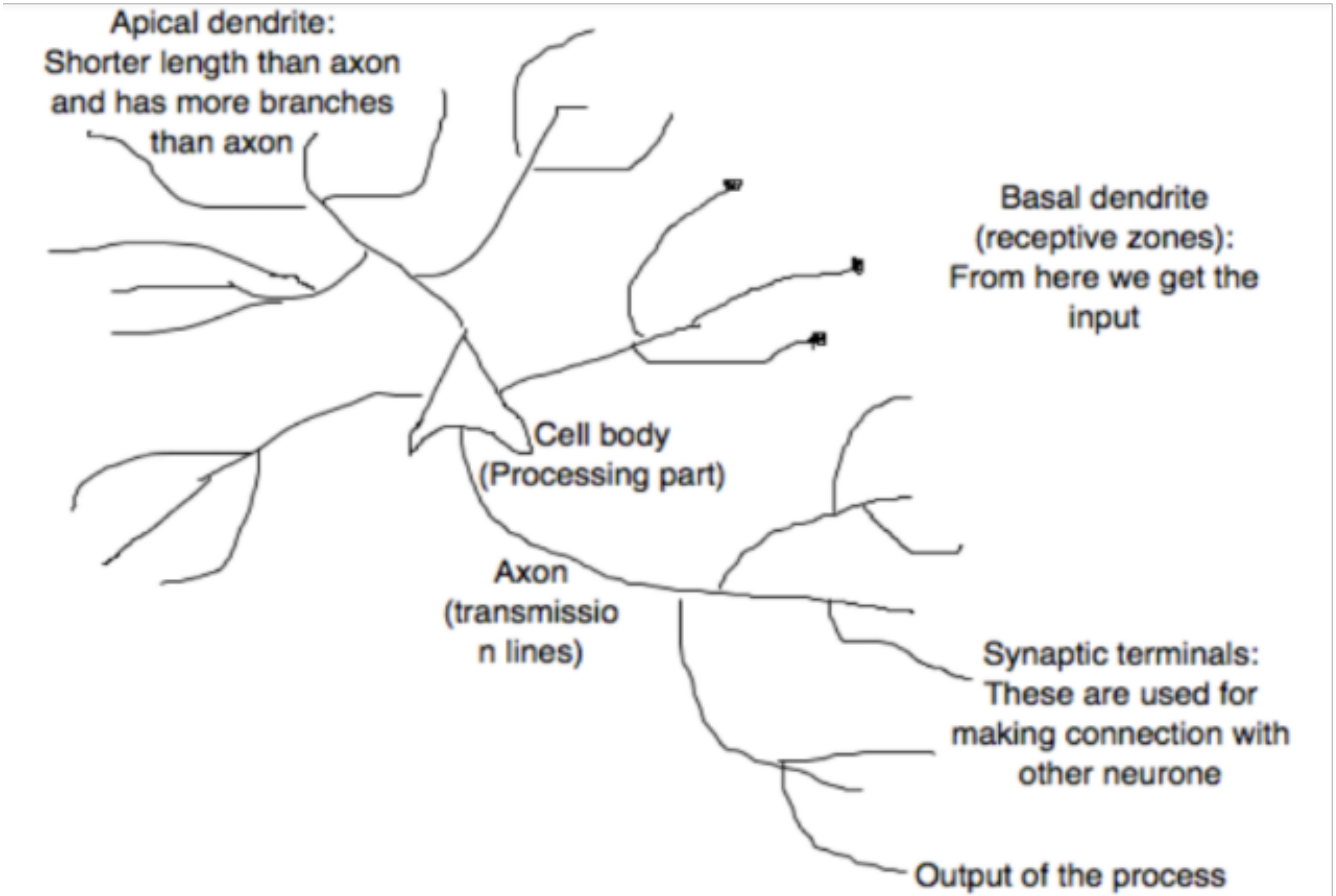
- However, the difficulty of pattern recognition becomes obvious if we try to write a computer program to recognize objects.

Neural networks

- Normally, computer needs time to do the recognition task, however, we human do this task instantly.
- **How are we going to do that very instantly?**
- Let’s compare processing speed of computer with a human:
  - IC processing speed: 1 nanosecond.
  - Human Neuron processing speed: 1 millisecond.
- As you can see human 4-5 orders is slower than computers. But, **how can we faster in real problems?**
- The answer of this question lies on the massively parallel networks on neurons. **10 billions Neurons & 60 trillions of interconnections.**

Neural networks

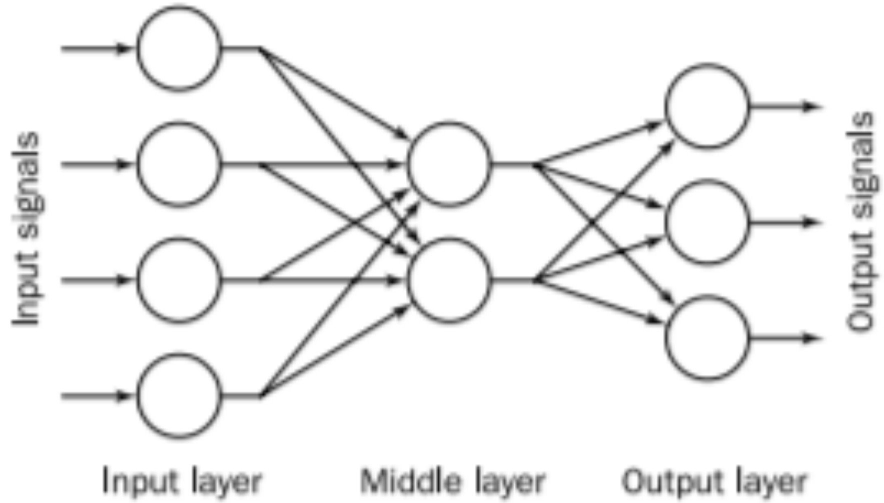
- **Is it possible to mimic the task using computer software?**
- We can mimic neurons, but in different way that biological neurons work.
- In this manner, the term artificial neural networks can be used.
- Now lets compare biological neural networks with electrical model.



Pyramidal Cell

Neural networks

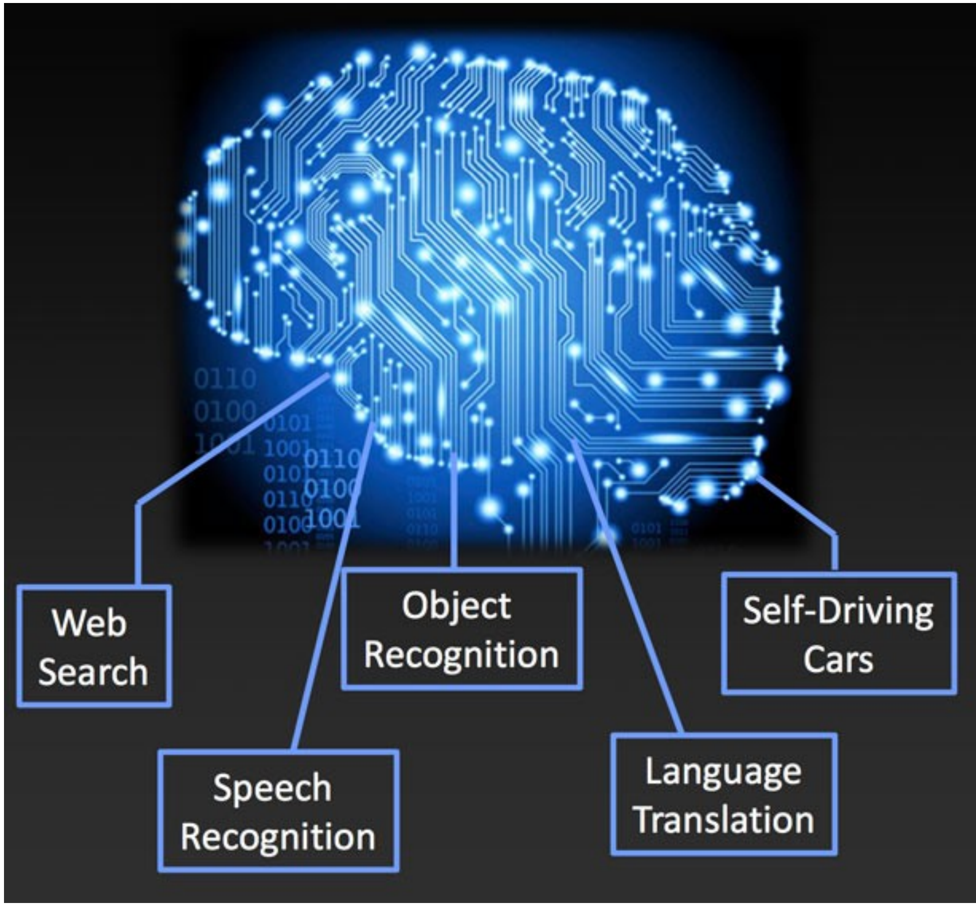
- The neurons are connected by weighted links passing signals from one neuron to another.
- Each neuron receives a number of input signals through its connections; however, it **never** produces more than a single **output signal**.



Architecture of a typical artificial neural network

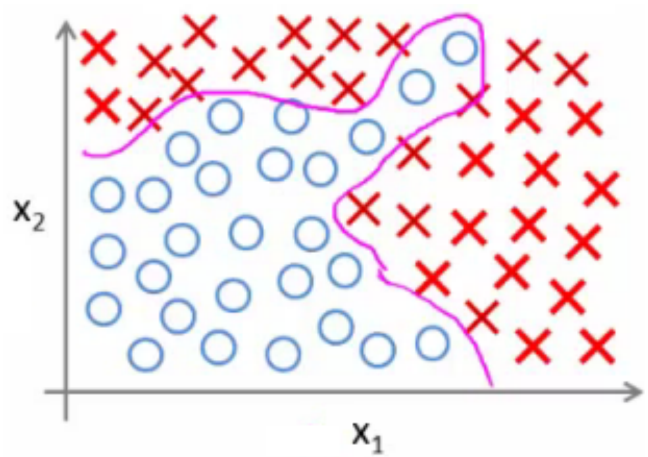
Application Areas of ANNs

- Object Recognition images
- Biometric Recognition
- Web search
- Wind power prediction
- Speech Recognition
- Self-driving cars
- Language Translation



Neural networks

- Remember our house price example (Lab):
  - 100 house features, predict odds of a house being sold in the next 6 months
  - Here, if you included all the quadratic terms (second order)
    - There are lots of them (x_1^2 ,x_1x_2, x_1x_4 ..., x_1x_{100})
    - For the case of n = 100, you have about 5000 features
    - Number of features grows $O(n^2)$
    - This would be computationally expensive to work with as a feature set



Neural networks

- A way around this to only include a subset of features
  - However, if you don't have enough features, often a model won't let you fit a complex dataset

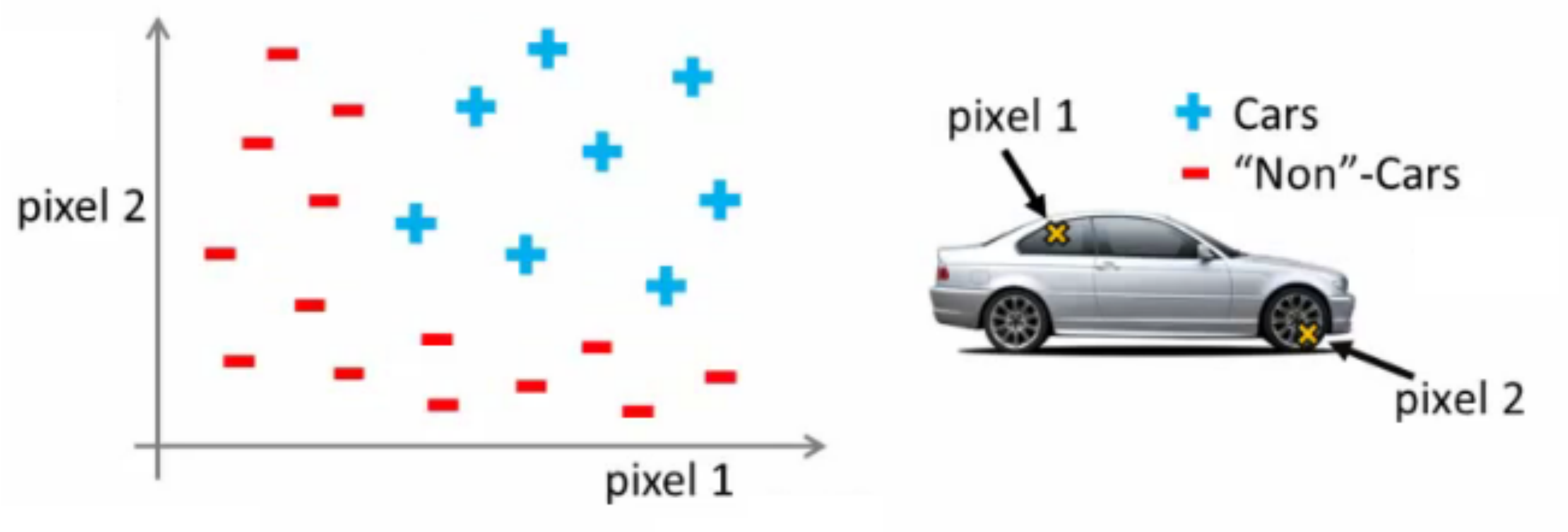
- If you include the cubic terms
  - e.g. ($x_1^2x_2$, $x_1x_2x_3$, $x_1x_4x_{23}$ etc)
  - There are even more features grows $O(n^3)$
  - About 170 000 features for n = 100

- Not a good way to build classifiers when n is large

Example: Problems where n is large - computer vision

- Computer vision sees a matrix of pixel intensity values
  - Look at matrix - explain what those numbers represent
- To build a car detector
  - Build a training set of
    - Not cars
    - Cars
  - Then test against a car
- How can we do this
  - Plot two pixels (two pixel locations)
  - Plot car or not car on the graph

Example: Problems where n is large - computer vision



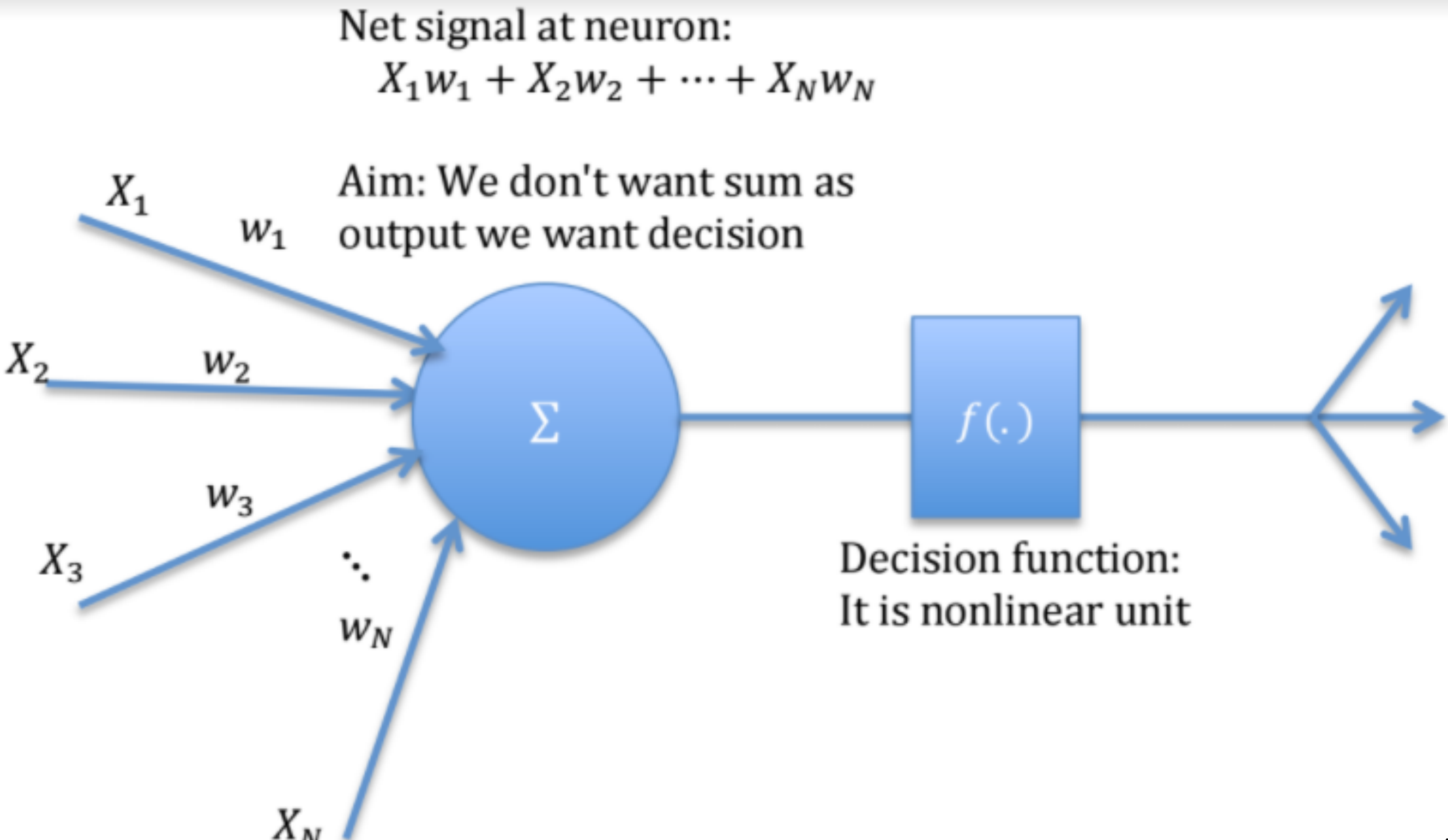
Example: Problems where n is large - computer vision

- Need a non-linear hypothesis to separate the classes
- Feature space
  - If we used 50 x 50 pixels --> 2500 pixels, so n = 2500
  - If RGB then 7500
  - If 100 x 100 RB then --> 50 000 000 features

- So - simple logistic regression here is not appropriate for large complex systems

- Neural networks are much better for a complex nonlinear hypothesis even when feature space is huge

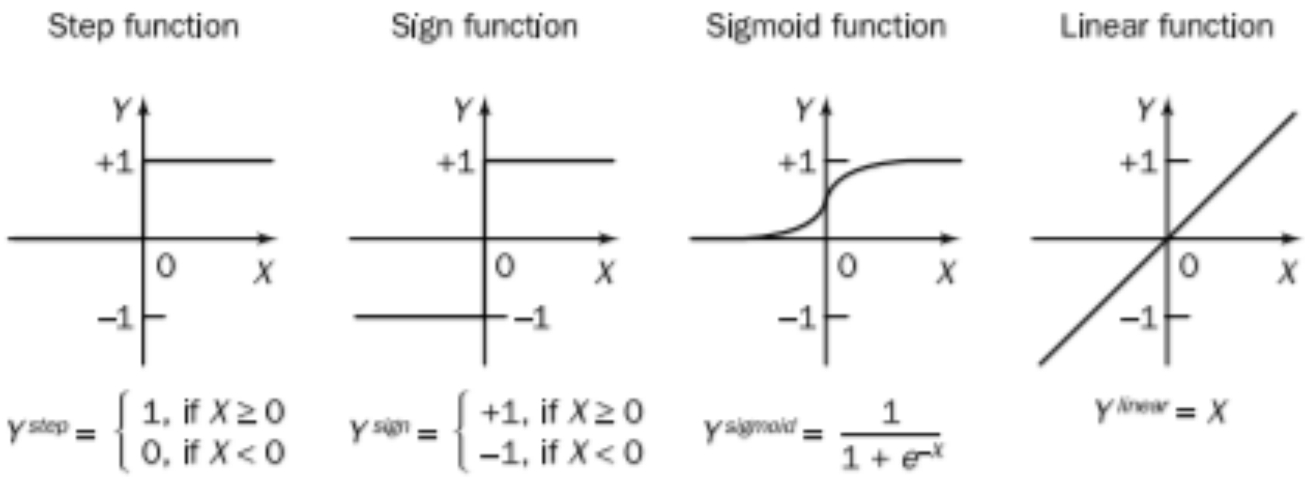
Artificial Neural Network



ANNs: Activation Functions

- Many activation functions have been proposed, but only a few have found practical applications.
- Four common choices – **the step, sign, sigmoid and linear**.
- The step and sign activation functions, also called hard limit functions.
- Linear and sigmoid activation functions are soft limit functions.
- The functions are often used in decision-making neurons for classification and pattern recognition tasks.

ANNs: Activation Functions



Activation functions of a neuron

- Example: Use find y (output) for x=5 (input) and use four activation functions

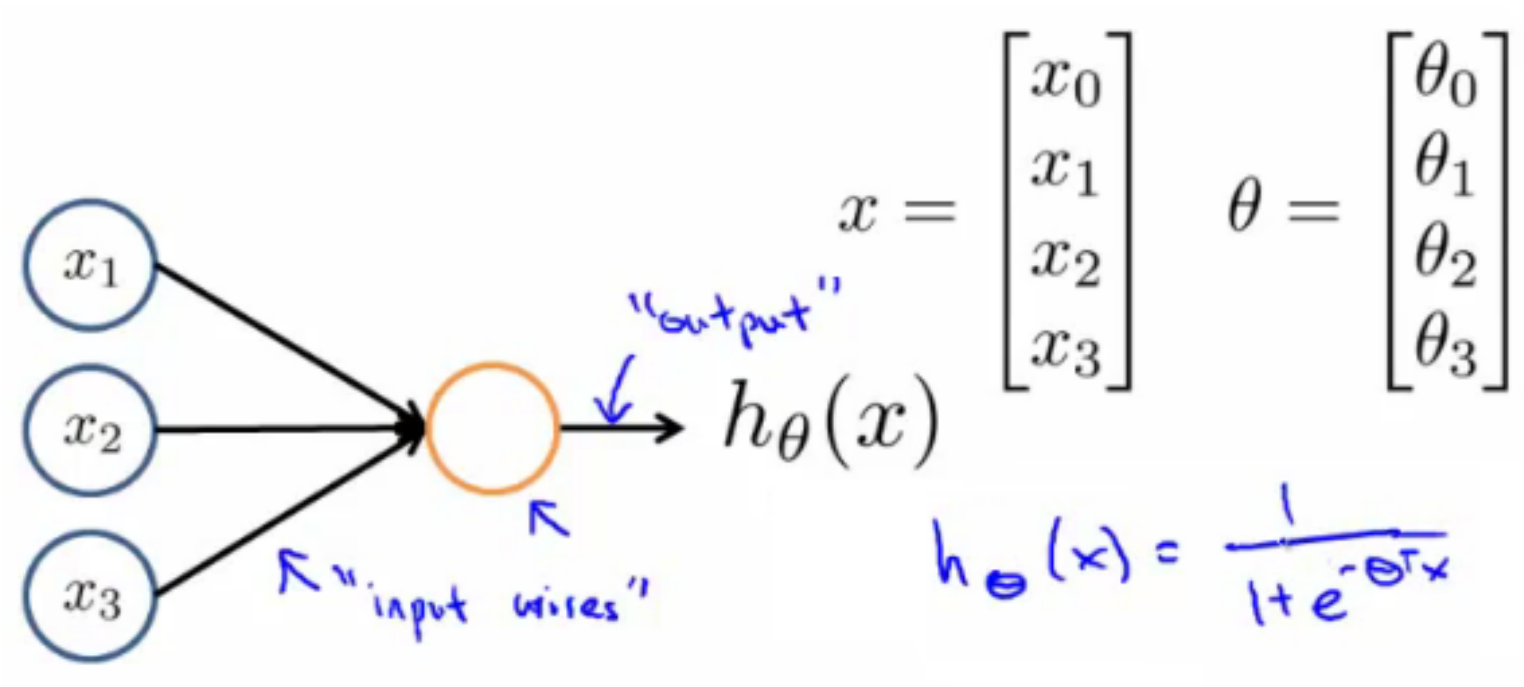
  y = +1 (Step function)

  y = +1 (Sign function)

  y = $1/1+e^{-5}$ = 0.9933 (Sigmoid function)

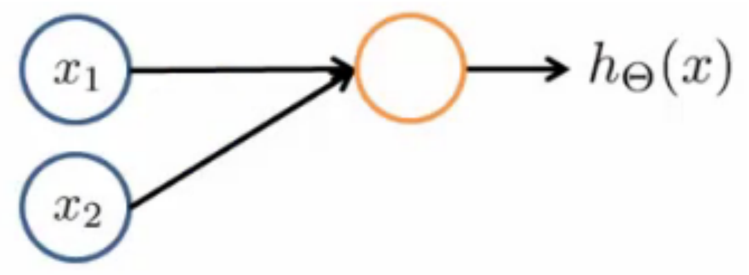
  y = 5 (Linear function)

ANNs- representation of a neuron with the Sigmoid Activation Function



Neural network example

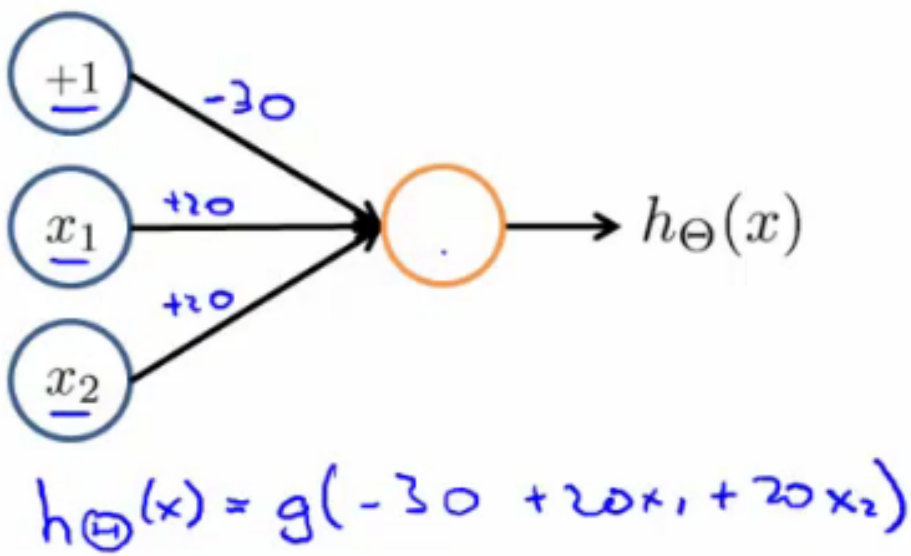
- And Function



- Can we get a one-unit neural network to compute this logical AND function? (probably...)
  - Add a bias unit
  - Add some weights for the networks
    - What are weights?
      - Weights are the parameter values which multiply into the input nodes (i.e. θ)

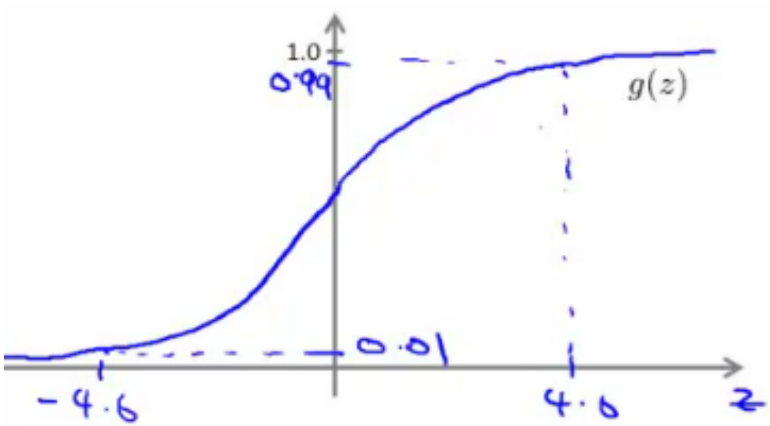
Neural network example

- Sometimes it's convenient to add the weights into the diagram
  - These values are in fact just the θ parameters so
    - $\theta_{10}^{1} = -30$
    - $\theta_{11}^{1} = 20$
    - $\theta_{12}^{1} = 20$



Neural network example

- So, as we can see, when we evaluate each of the four possible input, only (1,1) gives a positive output



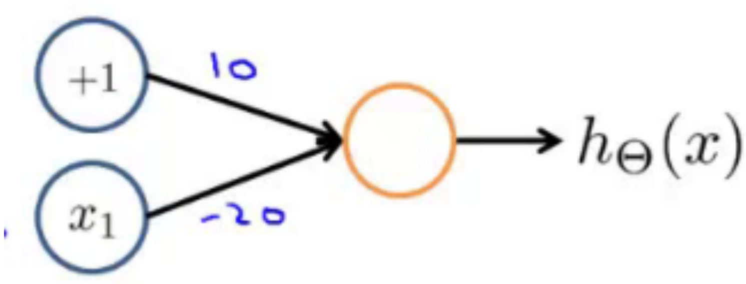
Sigmoid function (reminder)

x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$h_\Theta(x) \approx x_1 \text{ AND } x_2$

Neural network example

- NOT Function



x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

- This is achieved by putting a large negative weight in front of the variable you want to be negative

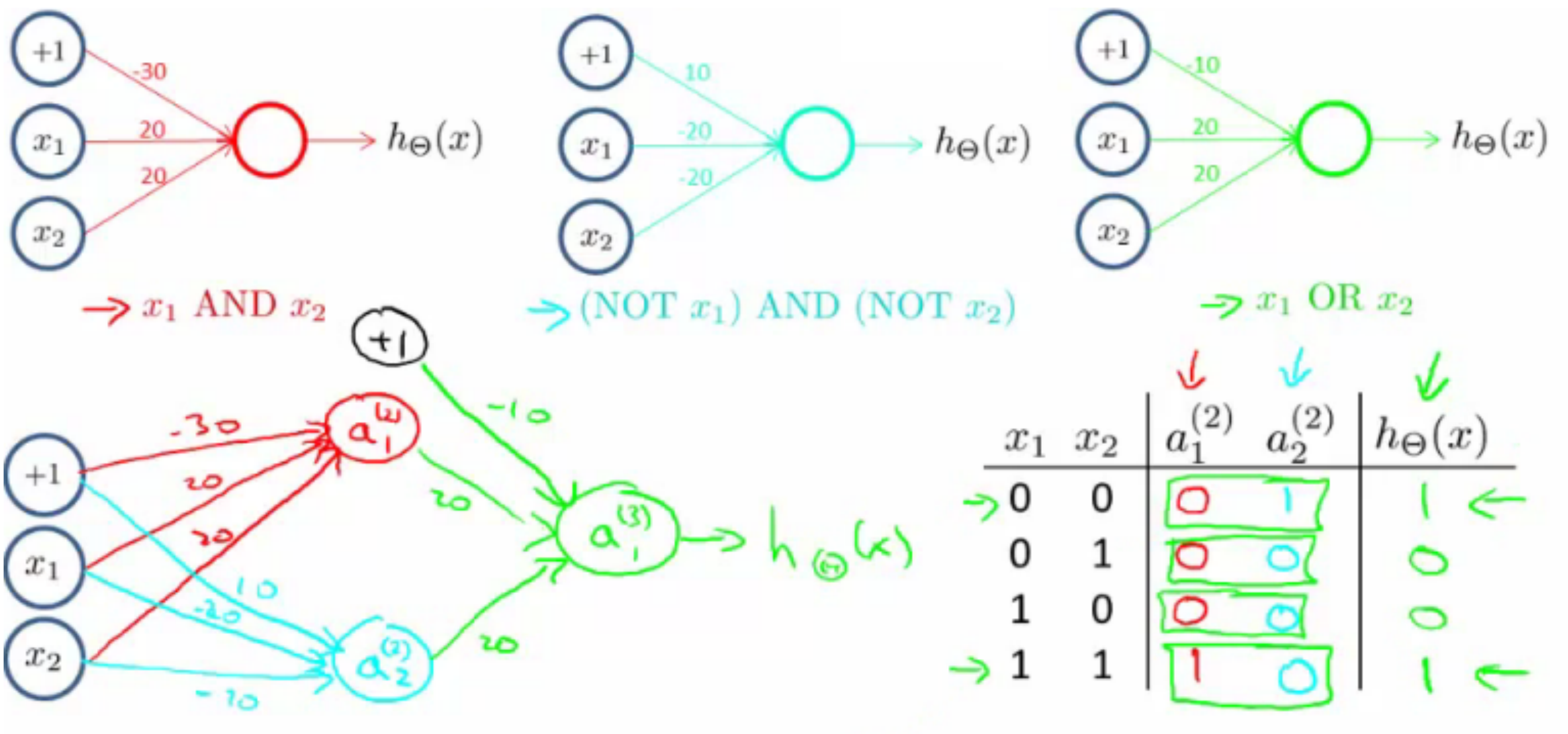
Neural network example

- XNOR Function:

- So how do we make the XNOR function work?
  - XNOR is short for NOT XOR
  - So we want to structure this so the input which produce a positive output are:
    - AND (i.e. both true) or Neither

- So we combine these into a neural network as shown below:

Neural network example



ANNs: Perceptron

- It is the simplest form of a neural network.
- It consists of a single neuron with adjustable synaptic weights and a hard limiter.
- A neuron receives several signals from its input links, computes a new activation level and sends it as an output signal through the output links.
- The input signal can be raw data or outputs of other neurons.
- The output signal can be either a final solution to the problem or an input to other neurons

ANNs: Perceptron

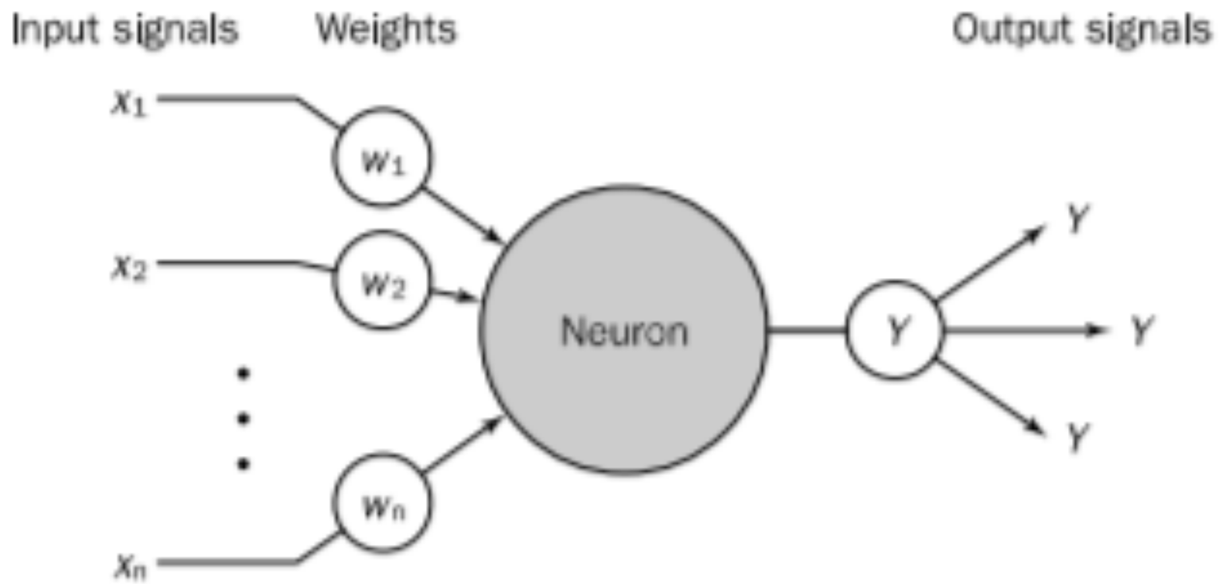
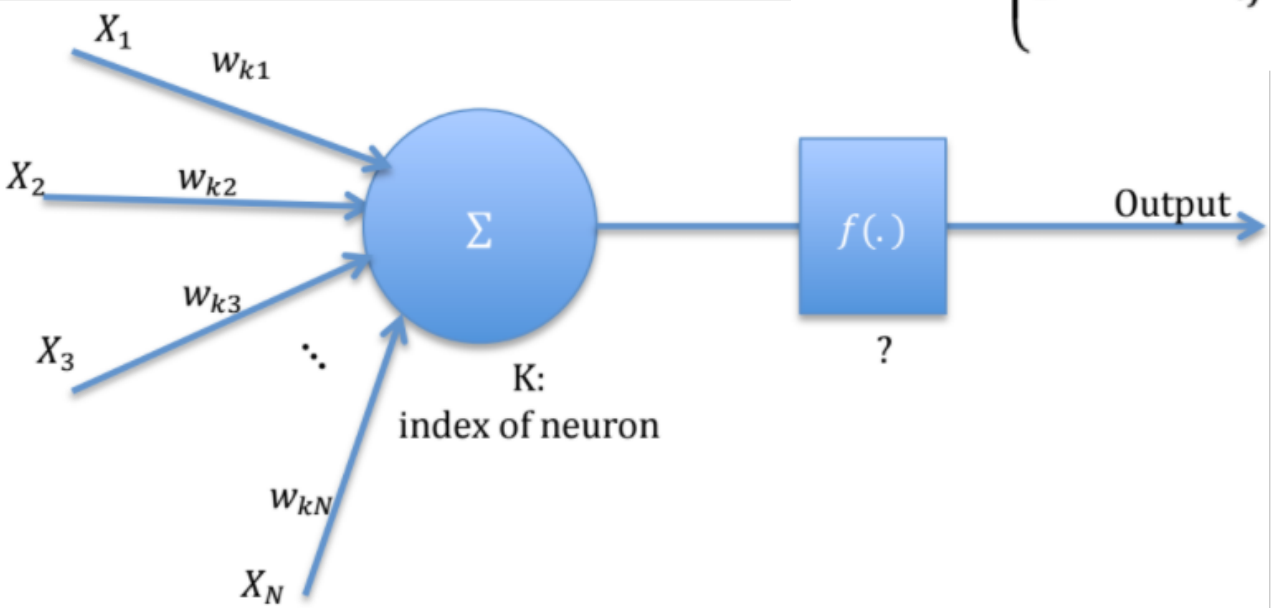


Diagram of a neuron

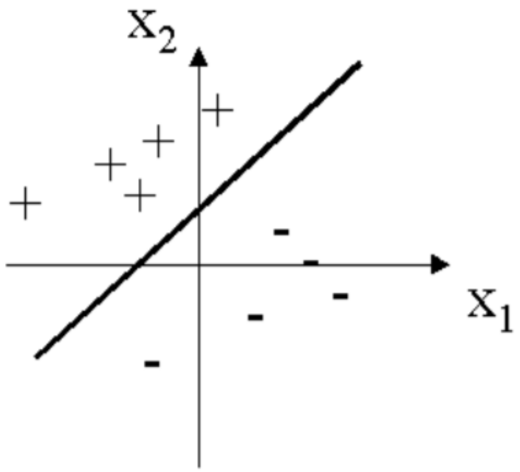
ANNs: Perceptron

$$Output = \begin{cases} 0 & if \sum_j X_j w_{kj} < thershold \\ 1 & if \sum_j X_j w_{kj} > thershold \end{cases}$$

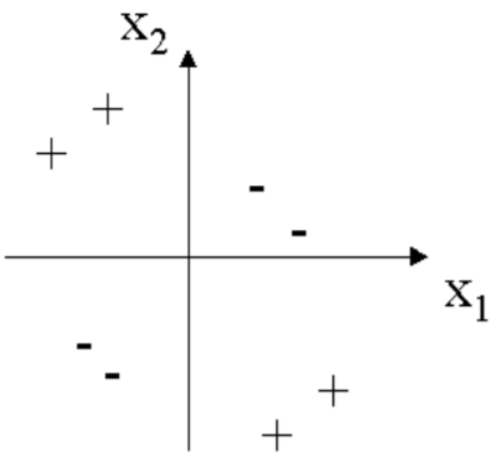


w_{kj}: Connection weight from neuron j to neuron k

ANNs: Perceptron

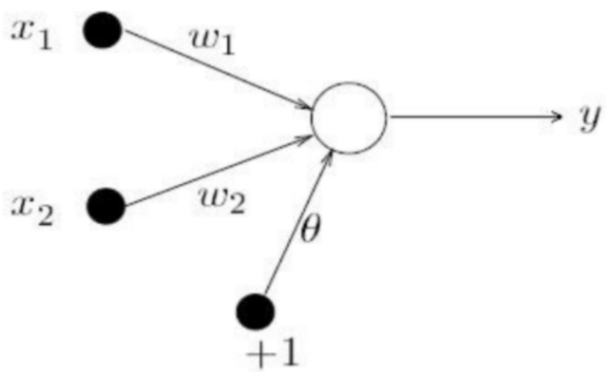


Linearly Separable



Not Linearly Separable

$$w_1x_1 + w_2x_2 + \theta = 0$$



ANNs: Perceptron

- **How does a perceptron learn its classification tasks?**
- By updating weights
- By reducing the difference between the actual and desired outputs of the perceptron.
- Iteration based

ANNs: Perceptron

- If at iteration p,
  - the actual output is $Y(p)$ and the desired output is $Y_d(p)$, then the error is given by

$$e(p) = Y_d(p) - Y(p) \text{ for } p = 1, 2, 3 \ldots$$

- If the error, $e(p)$, is positive,
  - we need to increase perceptron output $Y(p)$,
- but if it is negative,
  - we need to decrease $Y(p)$.

ANNs: Perceptron

- Each perceptron input contributes $x_i(p)$ x $w_i(p)$ to the total input $X(p)$,
- if input value $x_i(p)$ is positive, an increase in its weight $w_i(p)$ tends to increase perceptron output $Y(p)$,
- whereas if $x_i(p)$ is negative, an increase in $w_i(p)$ tends to decrease $Y(p)$.
- Thus, the following perceptron learning rule can be established:

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p),$$

where α is the learning rate.

ANNs: Perceptron

Step 1: ***Initialisation***
Set initial weights $w_1, w_2, \ldots, w_n$ and threshold θ to random numbers in the range $[-0.5, 0.5]$.

Step 2: ***Activation***
Activate the perceptron by applying inputs $x_1(p), x_2(p), \ldots, x_n(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)w_i(p) - \theta\right],$$

where n is the number of the perceptron inputs, and $step$ is a step activation function.

Step 3: ***Weight training***
Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p),$$

where $\Delta w_i(p)$ is the weight correction at iteration p.
The weight correction is computed by the **delta rule**:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p)$$

Step 4: ***Iteration***
Increase iteration p by one, go back to Step 2 and repeat the process until convergence.

Training basic logical operations

- The table presents all possible combinations of values for two variables, x_1 and x_2, and the results of the operations.
- The perceptron must be trained to classify the input patterns

Truth tables for the basic logical operations

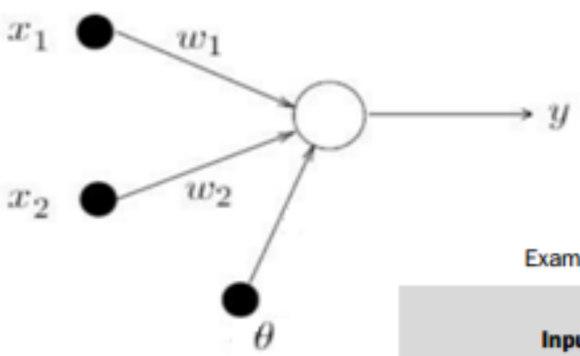
Input variables x_1	x_2	AND $x_1 \cap x_2$	OR $x_1 \cup x_2$	Exclusive-OR $x_1 \oplus x_2$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Training basic logical operations

- Let us first consider the operation AND.
- After completing the initialisation step, the perceptron is activated by the sequence of four input patterns representing an epoch.
- The perceptron weights are updated after each activation. This process is repeated until all the weights converge to a uniform set of values.

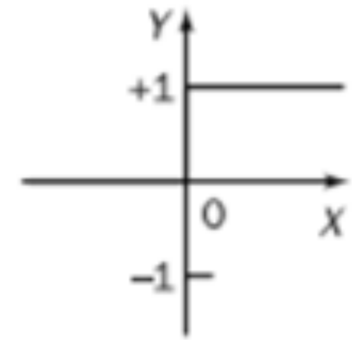
Input variables x_1	x_2	AND $x_1 \cap x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Example with one perceptron:



$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)w_i(p) - \theta\right],$$

Step function



$$Y^{step} = \begin{cases} 1, & \text{if } X \geq 0 \\ 0, & \text{if } X < 0 \end{cases}$$

Example of perceptron learning: the logical operation AND

	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
Epoch	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0		
	0	1	0	0.3	−0.1	0	0		
	1	0	0	0.3	−0.1	1	−1		
	1	1	1	0.3	−0.1	1	0		

- y = step(0x0.3+0x (-0.1) - 0.2) = step(-0.2) = 0
- y = step(0x0.3+1x (-0.1) - 0.2) = step(-0.3) = 0
- y = step(1x0.3+0x (-0.1) - 0.2) = step(0.1) = 1
- y = step(1x0.3+1x (-0.1) - 0.2) = step(0) = 1

These results are not same with the desired results so it is necessary to update the weights using step 3 in the algorithm.

Example with one perceptron:

Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p)$$

x_1=0, x_2=0

w1 = 0.3 + 0.1 x 0 x 0 = 0.3

w2 = -0.1 + 0.1 x 0 x 0 = -0.1

x_1=0, x_2=1

w1 = 0.3 + 0.1 x 0 x 0 = 0.3

w2 = -0.1 + 0.1 x 1 x 0 = -0.1

x_1=1, x_2=0

w1 = 0.3 + 0.1 x 1 x -1 = 0.2

w2 = -0.1 + 0.1 x 0 x -1 = -0.1

x_1=1, x_2=1

w1 = 0.3 + 0.1 x 1 x 0 = 0.3

w2 = -0.1 + 0.1 x 1 x 0 = -0.1

Example of perceptron learning: the logical operation AND

	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
Epoch	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0		
	0	1	0	0.3	−0.1	0	0		
	1	0	0	0.3	−0.1	1	−1		
	1	1	1	0.3	−0.1	1	0		

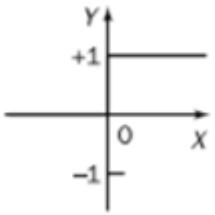
Example of perceptron learning: the logical operation AND

	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
Epoch	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.3	−0.1	1	0	0.3	−0.1

Example with one perceptron:

y = step(0x0.2+0x (-0.1) - 0.2) = step(-0.2) = 0
y = step(0x0.2+1x (-0.1) - 0.2) = step(-0.3) = 0
y = step(1x0.2+0x (-0.1) - 0.2) = step(0) = 1
y = step(1x0.2+1x (-0.1) - 0.2) = step(-0.1)=0

Step function



$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$$

Example of perceptron learning: the logical operation AND

Epoch	Inputs x_1	Inputs x_2	Desired output Y_d	Initial weights w_1	Initial weights w_2	Actual output Y	Error e	Final weights w_1	Final weights w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.3	−0.1	1	0	0.3	−0.1
2	0	0	0	0.2	−0.1	0	0		
	0	1	0	0.2	−0.1	0	0		
	1	0	0	0.2	−0.1	1	−1		
	1	1	1	0.2	−0.1	0	1		

Example with one perceptron:

x_1=0, x_2=0

w1 = 0.2 + 0.1 x 0 x 0 = 0.2

w2 = -0.1 + 0.1 x 0 x 0 = -0.1

x_1=0, x_2=1

w1 = 0.2 + 0.1 x 0 x 0 = 0.2

w2 = -0.1 + 0.1 x 1 x 0 = -0.1

x_1=1, x_2=0

w1 = 0.2 + 0.1 x 1 x -1 = 0.1

w2 = -0.1 + 0.1 x 0 x -1 = -0.1

x_1=1, x_2=1

w1 = 0.2 + 0.1 x 1 x 1 = 0.3

w2 = -0.1 + 0.1 x 1 x 1 = 0

Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p)$$

Example of perceptron learning: the logical operation AND

Epoch	Inputs x_1	Inputs x_2	Desired output Y_d	Initial weights w_1	Initial weights w_2	Actual output Y	Error e	Final weights w_1	Final weights w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.3	−0.1	1	0	0.3	−0.1
2	0	0	0	0.2	−0.1	0	0		
	0	1	0	0.2	−0.1	0	0		
	1	0	0	0.2	−0.1	1	−1		
	1	1	1	0.2	−0.1	0	1		

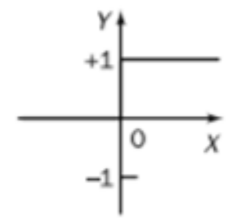
Example of perceptron learning: the logical operation AND

Epoch	Inputs x_1	Inputs x_2	Desired output Y_d	Initial weights w_1	Initial weights w_2	Actual output Y	Error e	Final weights w_1	Final weights w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.3	−0.1	1	0	0.3	−0.1
2	0	0	0	0.2	−0.1	0	0	0.2	−0.1
	0	1	0	0.2	−0.1	0	0	0.2	−0.1
	1	0	0	0.2	−0.1	1	−1	0.1	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.

Example with one perceptron:

y = step(0x0.3+0x (0) - 0.2) = step(-0.2) = 0
y = step(0x0.3+1x (0) - 0.2) = step(-0.2) = 0
y = step(1x0.3+0x (0) - 0.2) = step(0.1) = 1
y = step(1x0.3+1x (0) - 0.2) = step(0.1)=1

Step function



$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$$

Example of perceptron learning: the logical operation AND

	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
Epoch	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.3	−0.1	1	0	0.3	−0.1
2	0	0	0	0.2	−0.1	0	0	0.2	−0.1
	0	1	0	0.2	−0.1	0	0	0.2	−0.1
	1	0	0	0.2	−0.1	1	−1	0.1	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.
3	0	0	0	0.3	0.	0	0		
	0	1	0	0.3	0.	0	0		
	1	0	0	0.3	0.	1	−1		
	1	1	1	0.3	0.	1	0		

Example with one perceptron:

Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p)$$

x_1=0, x_2=0

w1 = 0.3 + 0.1 x 0 x 0 = 0.3

w2 = 0 + 0.1 x 0 x 0 = 0

x_1=0, x_2=1

w1 = 0.3 + 0.1 x 0 x 0 = 0.3

w2 = 0 + 0.1 x 1 x 0 = 0

x_1=1, x_2=0

w1 = 0.3 + 0.1 x 1 x -1 = 0.2

w2 = 0 + 0.1 x 0 x -1 = 0

x_1=1, x_2=1

w1 = 0.3 + 0.1 x 1 x 0 = 0.3

w2 = 0 + 0.1 x 1 x 0 = 0

Example of perceptron learning: the logical operation AND

	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
Epoch	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.3	−0.1	1	0	0.3	−0.1
2	0	0	0	0.2	−0.1	0	0	0.2	−0.1
	0	1	0	0.2	−0.1	0	0	0.2	−0.1
	1	0	0	0.2	−0.1	1	−1	0.1	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.
3	0	0	0	0.3	0.	0	0		
	0	1	0	0.3	0.	0	0		
	1	0	0	0.3	0.	1	−1		
	1	1	1	0.3	0.	1	0		

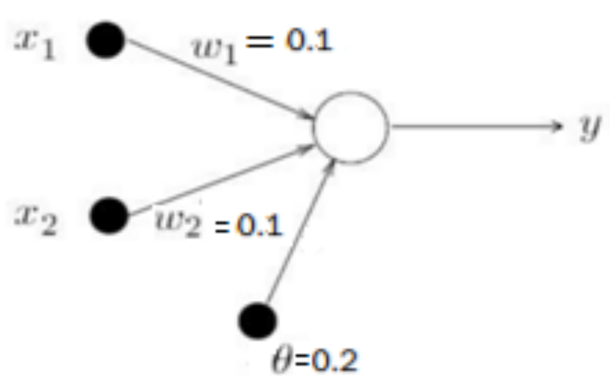
Example of perceptron learning: the logical operation AND

	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
Epoch	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.3	−0.1	1	0	0.3	−0.1
2	0	0	0	0.2	−0.1	0	0	0.2	−0.1
	0	1	0	0.2	−0.1	0	0	0.2	−0.1
	1	0	0	0.2	−0.1	1	−1	0.1	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.
3	0	0	0	0.3	0.	0	0	0.3	0.
	0	1	0	0.3	0.	0	0	0.3	0.
	1	0	0	0.3	0.	1	−1	0.2	0.
	1	1	1	0.3	0.	1	0	0.3	0.

Example with one perceptron:

Example of perceptron learning: the logical operation AND

Epoch	Inputs x_1	Inputs x_2	Desired output Y_d	Initial weights w_1	Initial weights w_2	Actual output Y	Error e	Final weights w_1	Final weights w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.3	−0.1	1	0	0.3	−0.1
2	0	0	0	0.2	−0.1	0	0	0.2	−0.1
	0	1	0	0.2	−0.1	0	0	0.2	−0.1
	1	0	0	0.2	−0.1	1	−1	0.1	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.
3	0	0	0	0.3	0.	0	0	0.3	0.
	0	1	0	0.3	0.	0	0	0.3	0.
	1	0	0	0.3	0.	1	−1	0.2	0.
	1	1	1	0.3	0.	1	0	0.3	0.
4	0	0	0	0.2	0.	0	0	0.2	0.
	0	1	0	0.2	0.	0	0	0.2	0.
	1	0	0	0.2	0.	1	−1	0.1	0.
	1	1	1	0.2	0.	1	0	0.2	0.
5	0	0	0	0.1	0.	0	0	0.1	0.
	0	1	0	0.1	0.	0	0	0.1	0.
	1	0	0	0.1	0.	0	0	0.1	0.
	1	1	1	0.1	0.	0.	1	0.2	0.1
6	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.2	0.1	1	0	0.2	0.1
7	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0.	0.	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1



Example

- In a similar manner, the perceptron can learn the operation OR.

- However, a single-layer perceptron cannot be trained to perform the operation Exclusive-OR.

Multilayer Neural Networks

- A multilayer perceptron is a feedforward neural network with one or more hidden layers.

- Typically, the network consists of an input layer of source neurons, at least one middle or hidden layer of computational neurons, and an output layer of computational neurons.

- The input signals are propagated in a forward direction on a layer-by-layer basis.

- A multilayer perceptron with two hidden layers is shown in next slide.

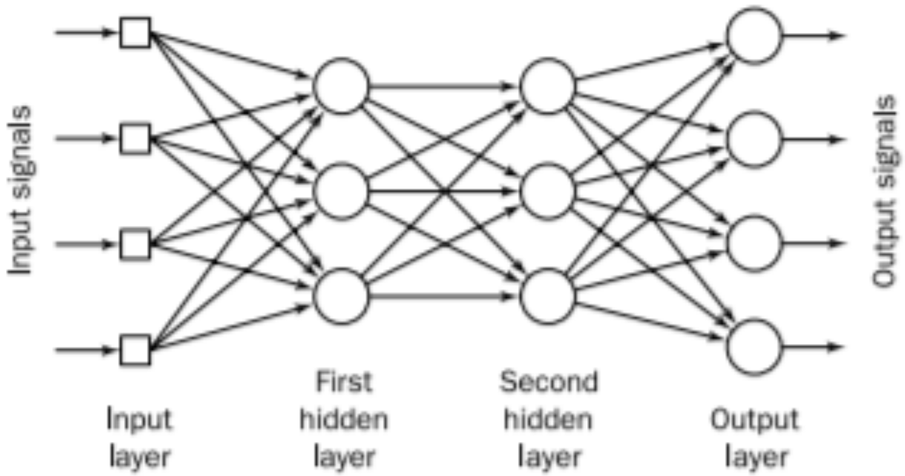
Multilayer Neural Networks

- Each layer in a multilayer neural network has its own specific function.
- The input layer accepts input signals from the outside world and redistributes these signals to all neurons in the hidden layer.
- Actually, the input layer rarely includes computing neurons, and thus does not process input patterns.
- The output layer accepts output signals, or in other words a stimulus pattern, from the hidden layer and establishes the output pattern of the entire network.

Multilayer Neural Networks

- Neurons in the hidden layer detect the features; the weights of the neurons represent the features hidden in the input patterns.

- These features are then used by the output layer in determining the output pattern.



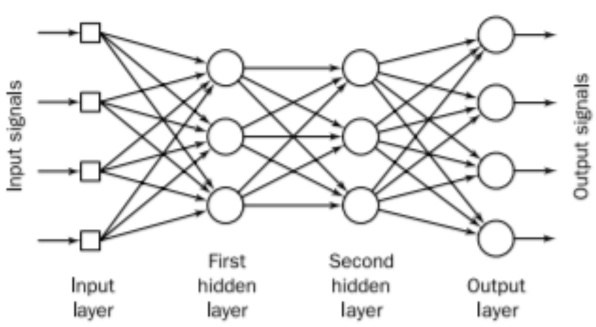
Multilayer perceptron with two hidden layers

Hidden Layer

- **Why is a middle layer in a multilayer network called a ‘hidden’ layer? What does this layer hide?**
  - A hidden layer ‘hides’ its desired output.
  - Neurons in the hidden layer cannot be observed through the input/output behaviour of the network.
  - There is no obvious way to know what the desired output of the hidden layer should be.

Feedforward Neural Networks

- A **feedforward neural network** is an artificial neural network wherein connections between the units do *not* form a cycle.
- Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs.
- During normal operation, that is when it acts as a classifier, there is no feedback between layers.
- This is why they are called *feedforward* neural networks.



Multilayer perceptron with two hidden layers