Week 3 – Introduction to Artificial Neural Networks

ELEC0144 Machine Learning for Robotics

Dr. Chow Yin Lai

Email: uceecyl@ucl.ac.uk

Schedule

Week	Lecture	Workshop	Assignment Deadlines
1	Introduction; Image Processing	Image Processing	
2	Camera and Robot Calibration	Camera and Robot Calibration	
3	Introduction to Neural Networks	Camera and Robot Calibration	Friday: Camera and Robot Calibration
4	MLP and Backpropagation	MLP and Backpropagation	
5	CNN and Image Classification	MLP and Backpropagation	
6	Object Detection	MLP and Backpropagation	Friday: MLP and Backpropagation
7	Path Planning	Path Planning	
8	Kalman Filter SLAM	Path Planning	
9	Extended Kalman Filter SLAM	Path Planning	
10	Particle Filter SLAM	Path Planning	Friday: Path Planning

Content

- Introduction to Neural Networks
- The Biological Neuron
- The Artificial Neuron
- Activation Functions
- Network Architectures
- Perceptrons
- Multilayer Perceptrons

Content

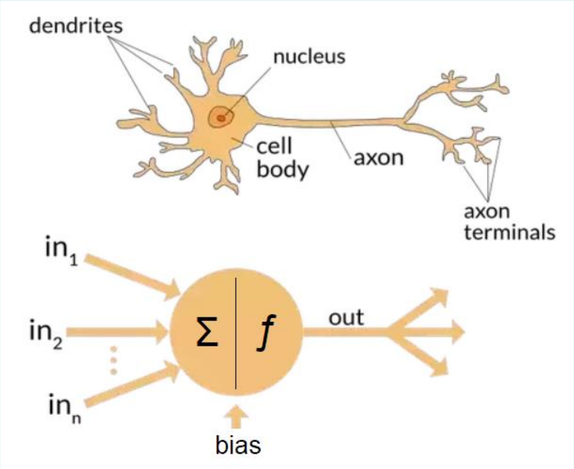
- Introduction to Neural Networks
- The Biological Neuron
- The Artificial Neuron
- Activation Functions
- Network Architectures
- Perceptrons
- Multilayer Perceptrons

What is a Neural Network (NN)? (1)

- A neural network is a massively parallel distributed processor that has a natural propensity for:
  - Storing experimental knowledge, and
  - Making it available for use.

What is a Neural Network (NN)? (2)

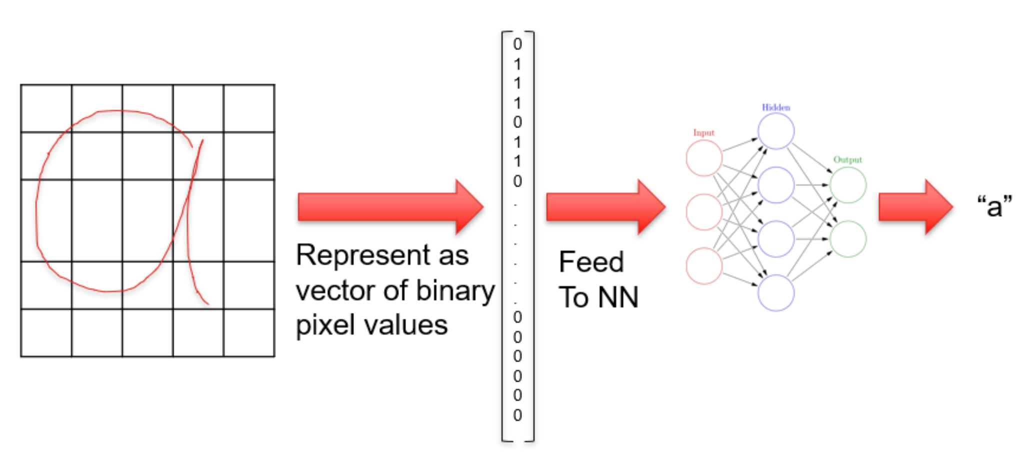
- It is similar to the brain in two ways:
  - Knowledge is acquired by the network through a learning process.
  - Knowledge is stored using interneuron connection strengths known as synaptic weights.



Biological vs Artificial NN
https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network)

Applications of NNs (1)

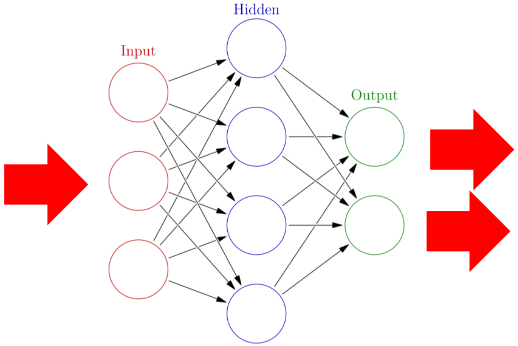
- NNs are mainly used for two types of applications:
- 1. Pattern Recognition or Classification
  - Example: Text recognition – Classify a handwritten alphabet as one of the 26 lower case letters.



Applications of NNs (2)

- Another example: Differentiate between a cat and a dog.

https://commons.wikimedia.org/wiki/File:Dog_Breeds.jpg

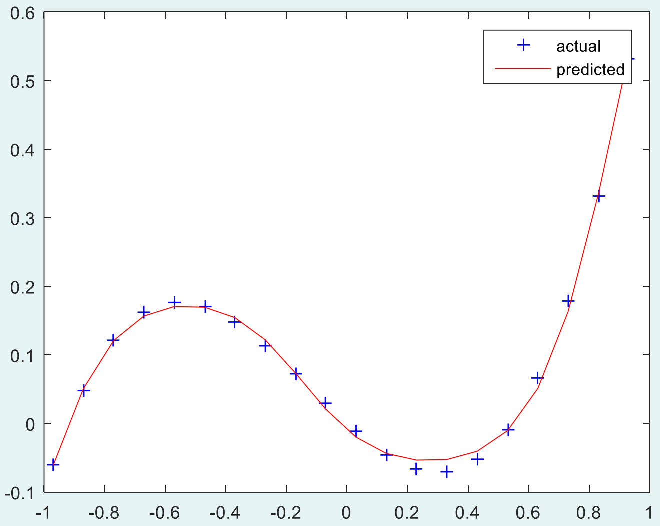


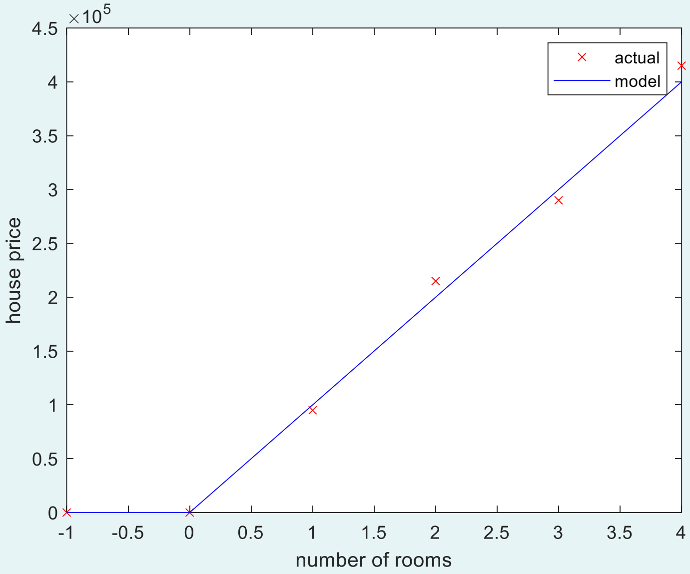
90% dog

10% cat

Applications of NNs (3)

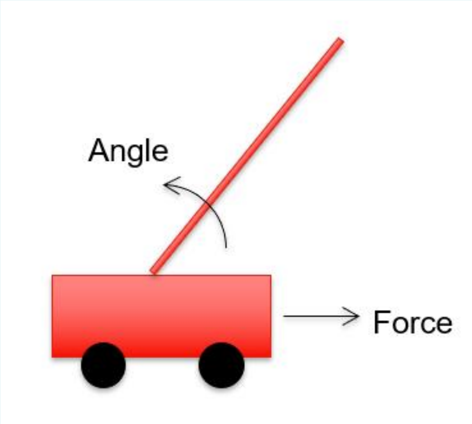
- 2. Regression or Function Approximation
  - Example Left: Data fitting.
  - Example Right: House price prediction.





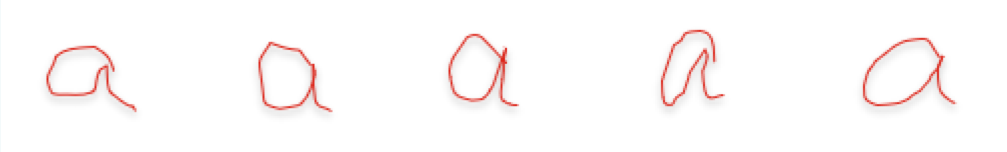
Applications of NNs (4)

- Another example: To understand the input-output relationship of an inverted pendulum (Input = Force, Output = Angle).
- $\boxed{\text{Angle = function(Force)}}$ → Learnt by NN automatically!



Learning of NNs (1)

- Recall that the knowledge is acquired through a learning process.
- For text recognition (e.g. “a”), we need to first show the NN many different handwritings of “a” – This is called “Training”.



Learning of NNs (2)

- After training, we can test if NN recognizes a new (unseen before) “a” correctly – This is called “Testing / Generalisation”.
- This somewhat resembles human learning:
  - When you went to the kindergarten, you would see many different “a” from different teachers.
  - After some time, you learn to recognize the letter even if it was written by someone new.

Learning of NNs (3)

- Similarly, for inverted pendulum (broom balancing), one would slowly learn the best hand motion in order to keep the broom upright.
- At the end, the brain would have learnt (unknowingly) the relationship between force and angular position!

https://imgur.com/gallery/bZ919

Historical Perspective (1)

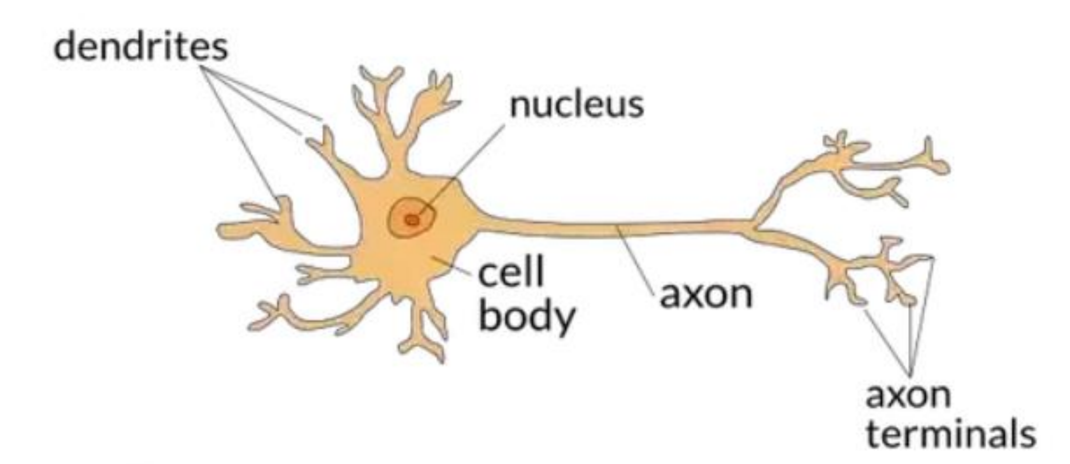
- 1943 McCulloch and Pitts published the first description of an artificial NN.
- 1950's – 60's First ANN developed by Marvin Minsky etc.: Single layer networks called perceptron, used for weather prediction, vision etc.
- 1970's Research virtually stopped, as many limitations were found e.g. incapable of solving many simple problems.

Historical Perspective (2)

- 1982 Hopfield proposed associative memory model: NN evolves to minimize an energy function; renewed interest in NNs.
- 1986 Back propagation learning rule for multi-layered networks proposed, overcoming limitations of simple perceptrons. Explosion of research.
- 2010's Deep neural network achieved amazing results in vision and research in deep network explodes – It is now the state of the art for object recognition and many other applications.

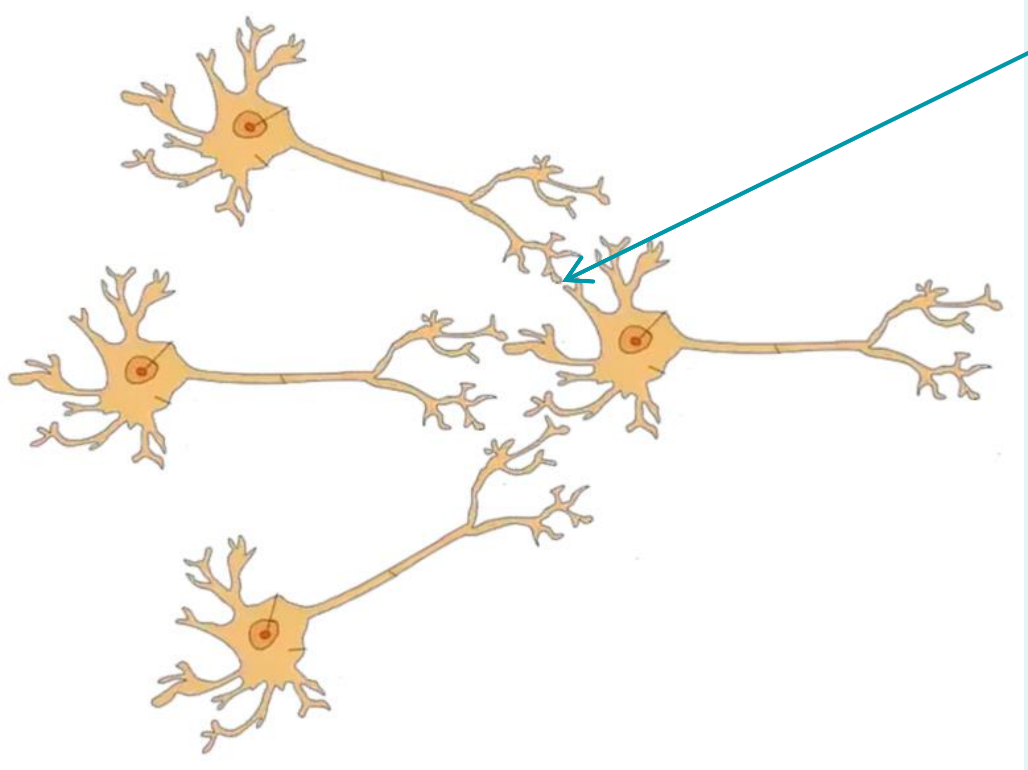
Content

One Biological Neuron



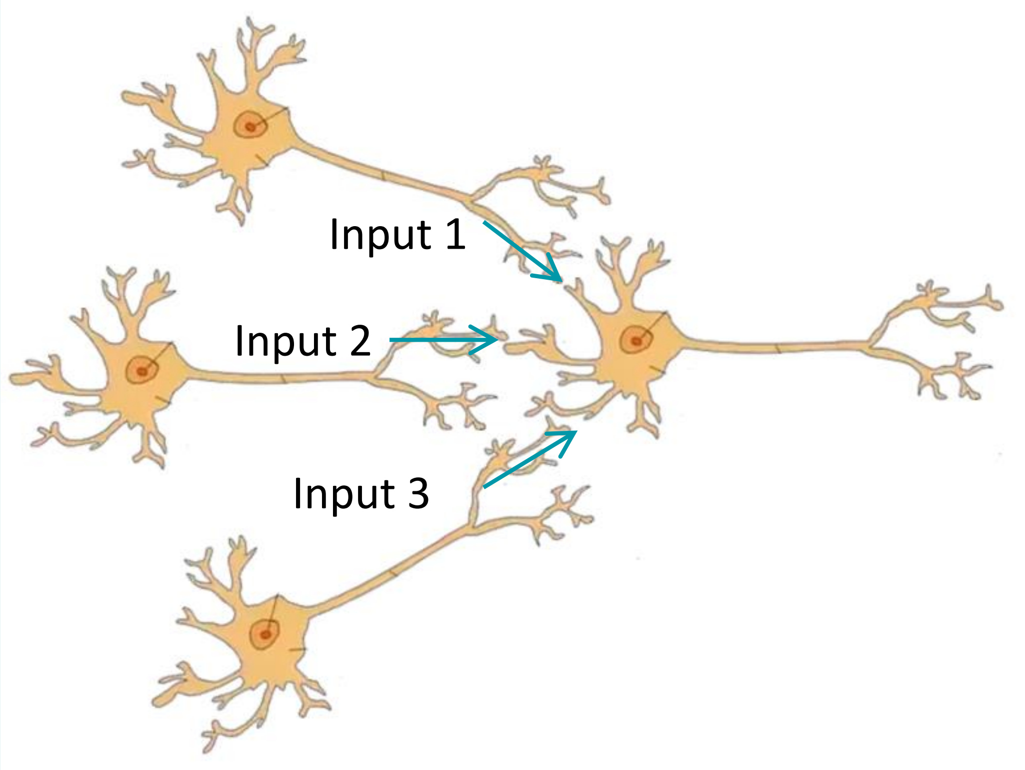
https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network)

A Few Neurons Together



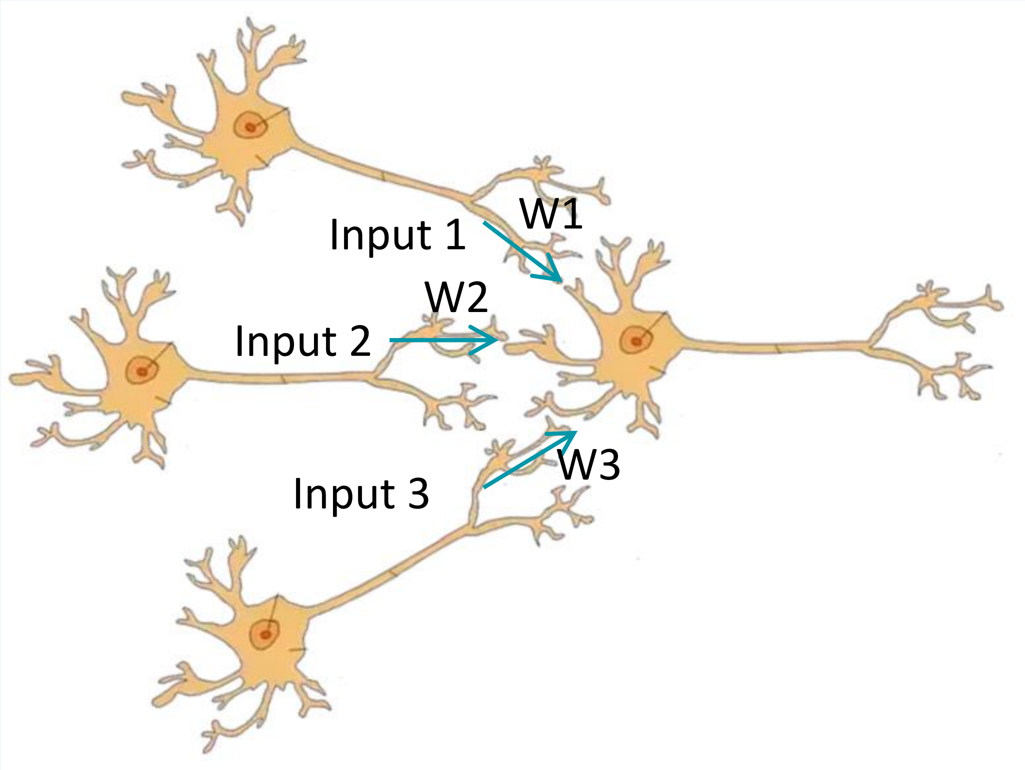
- Axon of one neuron almost touches dendrites of another neuron
- The small gap in between is the Synapses.
- Synapses can impose excitation (active) or inhibition (inactive) on the receptive neuron.

Looking at the Neuron on the Right (1)



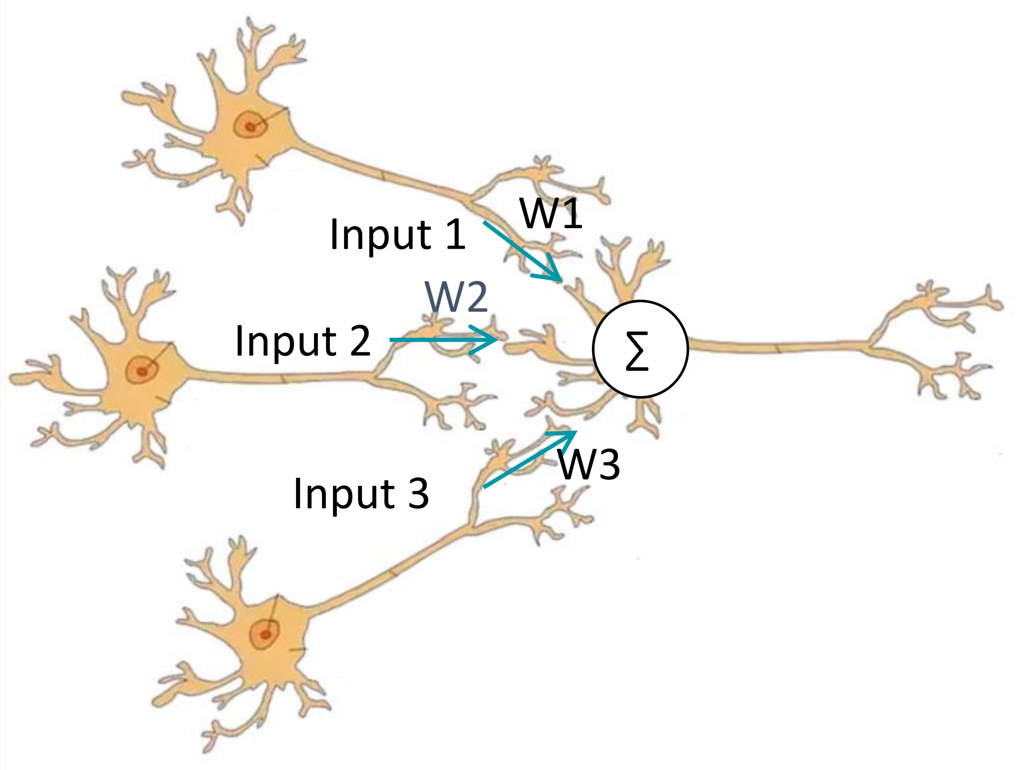
1. Input signals come from other neighbouring neurons.

Looking at the Neuron on the Right (2)



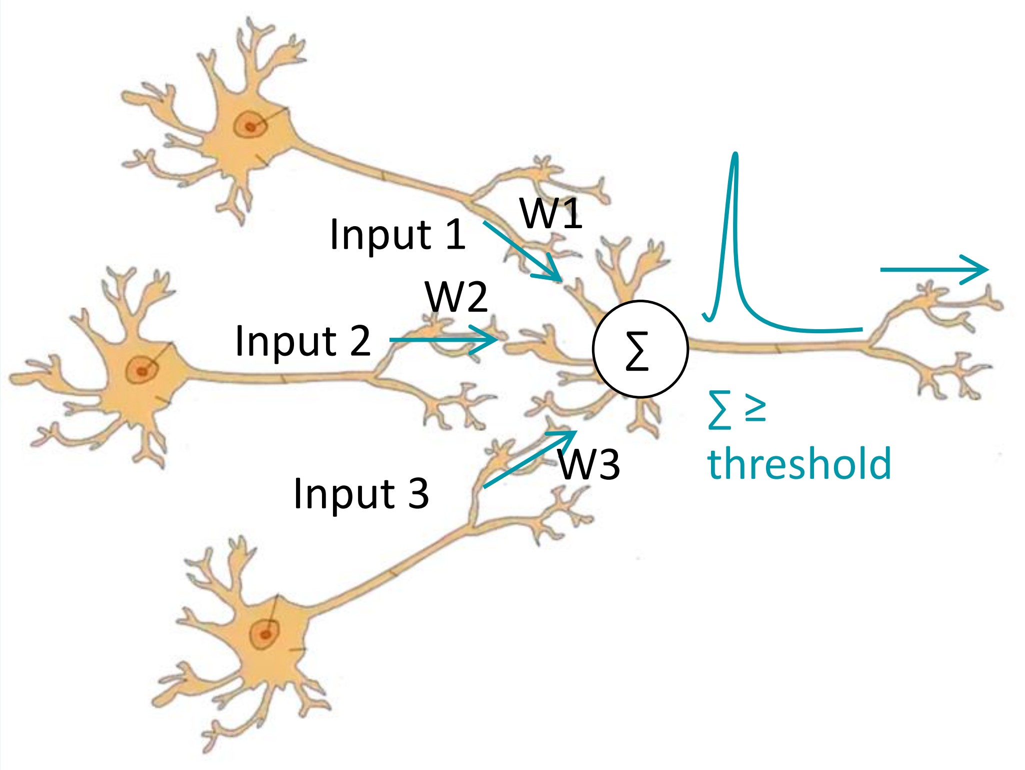
2. The synapses impose excitation or inhibition of the signal. This can be thought of as multiplying with a weight of 0 or 1.

Looking at the Neuron on the Right (3)



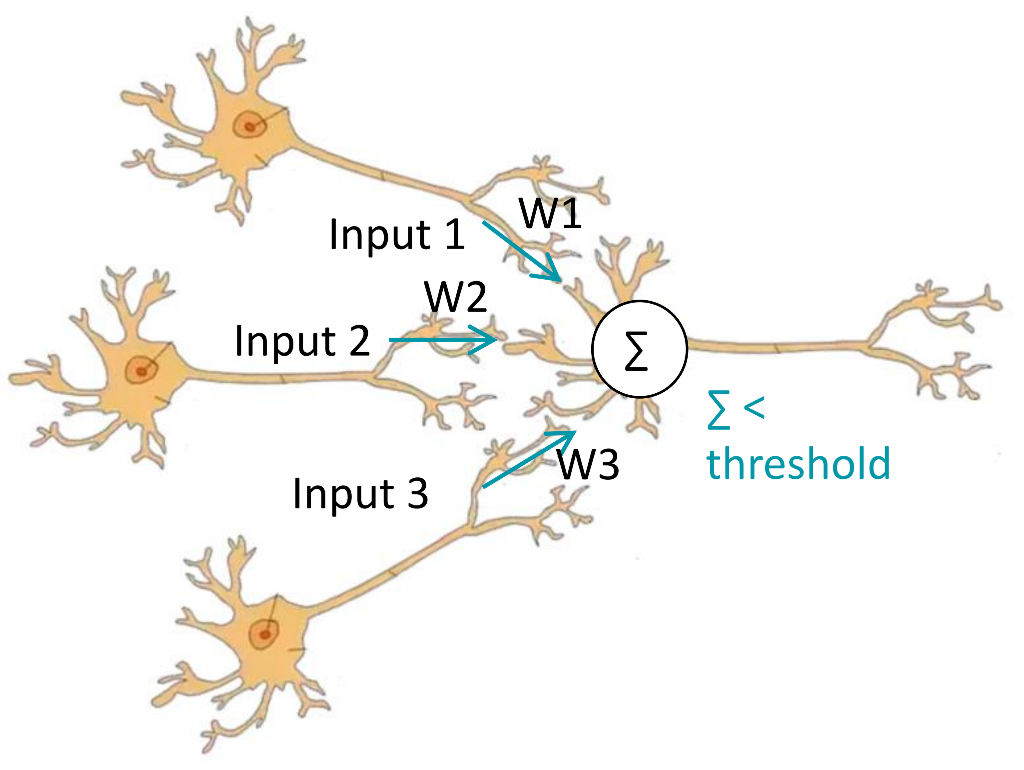
3. The cell body sums the incoming weighted signal.

Looking at the Neuron on the Right (4)



4. When sufficient input is received (more than threshold), the neuron fires, i.e. generate a spike, which is transmitted to the axon.

Looking at the Neuron on the Right (5)



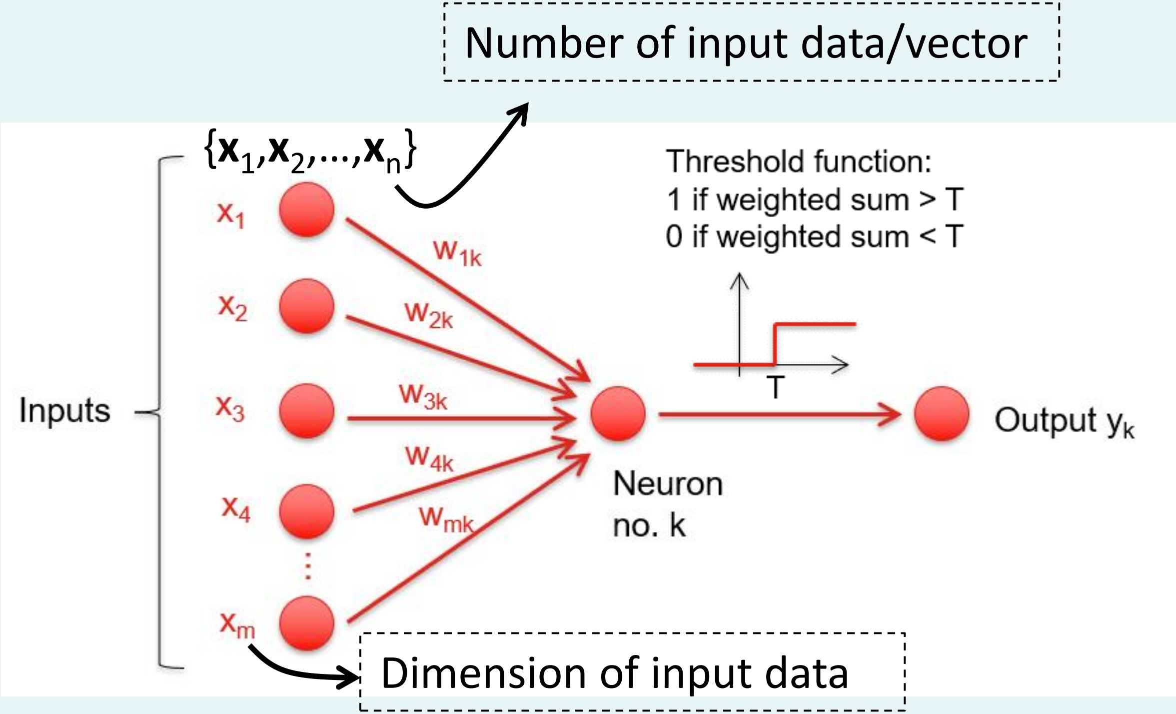
5. If input is less than threshold, then no firing occurs.

Content

- Introduction to Neural Networks
- The Biological Neuron
- The Artificial Neuron
- Activation Functions
- Network Architectures
- Perceptrons
- Multilayer Perceptrons

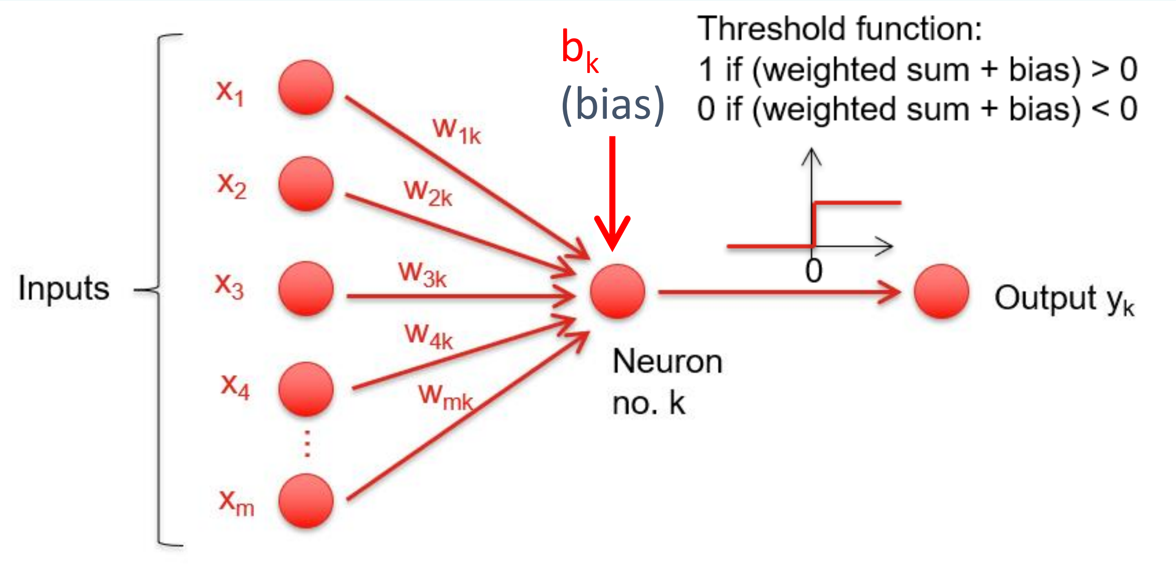
Early Model of Neuron

- Very similar to biological neuron.



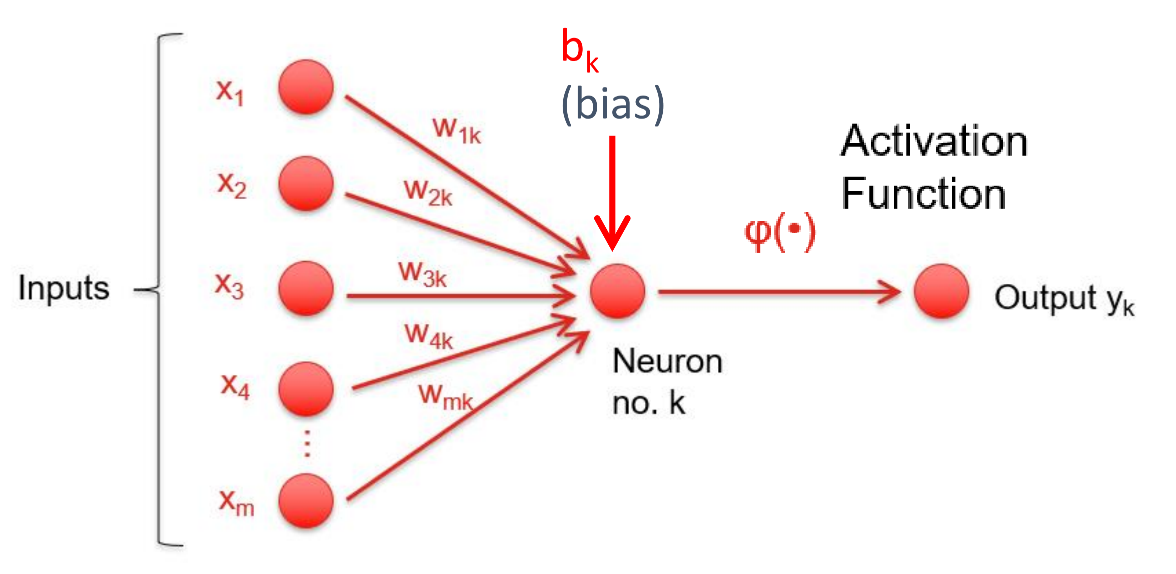
Improved Model of Neuron

- Having threshold function not symmetrical about 0 makes some calculations difficult. Therefore, a bias is added.



Further Improvement

- Various activation functions were also proposed in place of the threshold function.



Mathematical Model of Neuron (1)

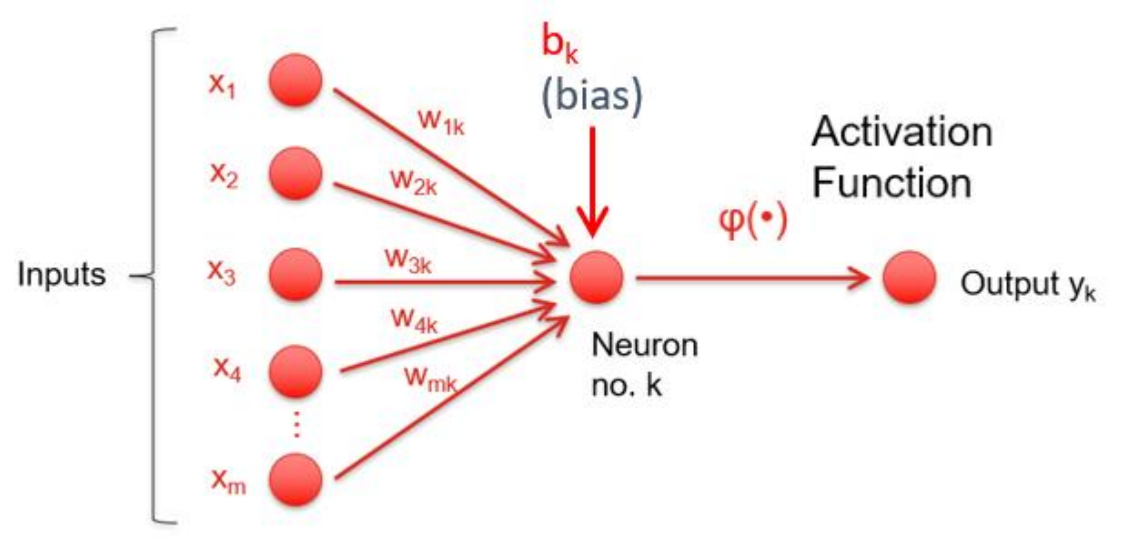
- As you can see, the model of a neuron has three basic components:
  - A set of synapses or connecting links, characterized by weights.
  - An adder for summing the weighted input signals.
  - An activation function to introduce nonlinearity into the output of a neuron, for e.g. limiting the amplitude of the output of a neuron within the range of [0, 1], [-1, 1], [0, ∞] etc.
    - The above ranges are due to “log sigmoid”, “tanh” and “rectified linear unit (ReLU)” respectively, which will be discussed later.

Mathematical Model of Neuron (2)

- Mathematically, for a neuron k:

$$\boxed{y_k = \varphi\left(\underbrace{\sum_{j=1}^{m} w_{jk} x_j + b_k}_{v_k}\right)}$$

$$\boxed{y_k = \varphi(v_k)}$$

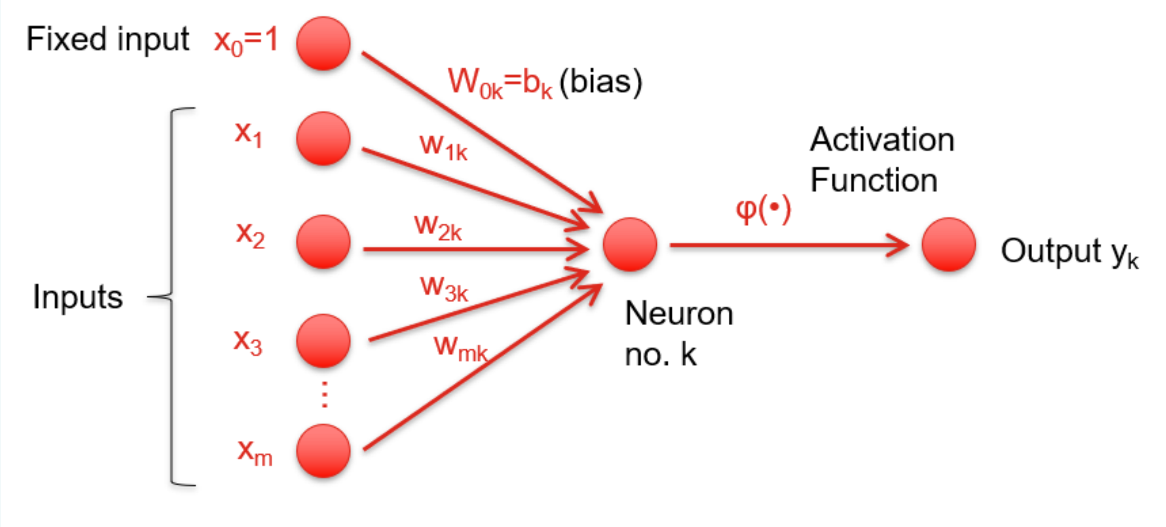


Mathematical Model of Neuron (3)

- Alternatively, we can think of the bias as input $x_0 = 1$, multiplied by weight $w_{0k} = b_k$.

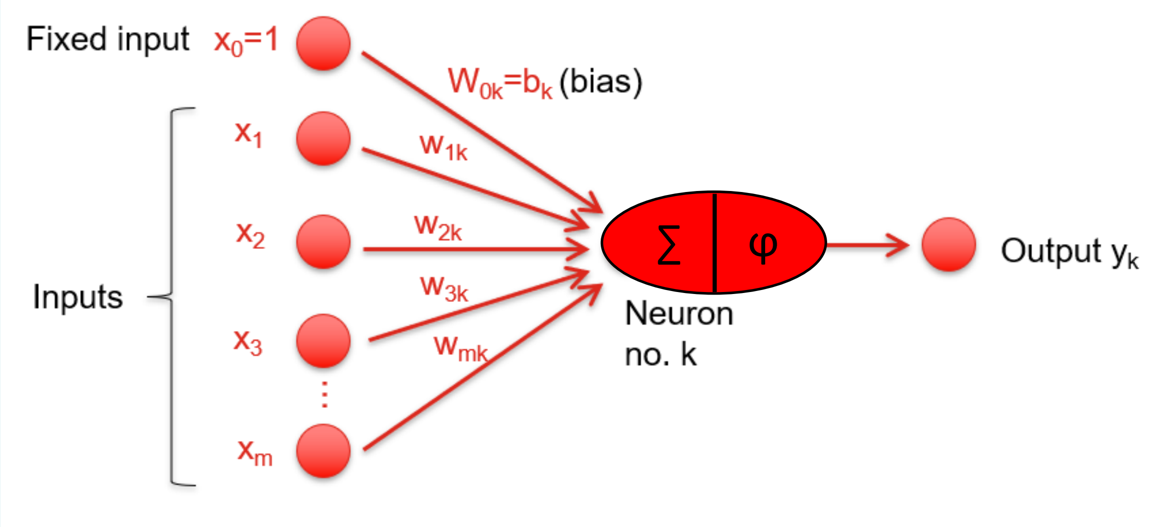
$$\boxed{y_k = \varphi\left(\underbrace{\sum_{j=0}^{m} w_{jk} x_j}_{v_k}\right)}$$

$$\boxed{y_k = \varphi(v_k)}$$



Mathematical Model of Neuron (4)

- It is useful to think of the neuron having two halves:
  - An adder ∑,
  - Followed by activation function φ.

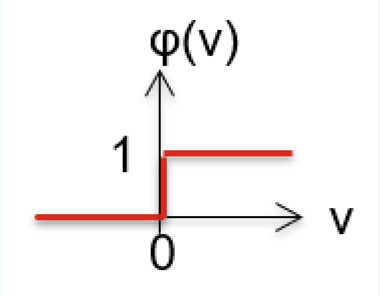


Content

- Introduction to Neural Networks
- The Biological Neuron
- The Artificial Neuron
- Activation Functions
- Network Architectures
- Perceptrons
- Multilayer Perceptrons

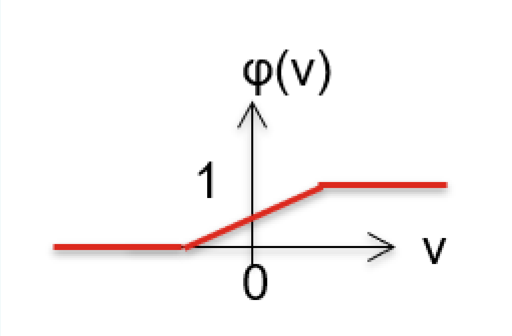
Activation Functions (1)

- Various activation functions have been proposed for neural networks.
- Threshold function (hard-limiter):
  - Note: McCulloch-Pits model (1943) of neuron used this form of function.



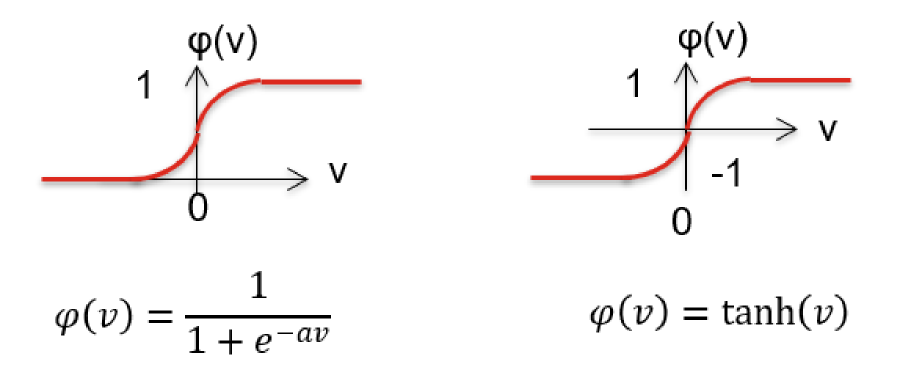
Activation Functions (2)

- Piecewise linear function:
  - Linear combiner within certain a certain range, then saturated to 0 or 1.
  - If gradient of linear region is very high, it would reduce to threshold function.



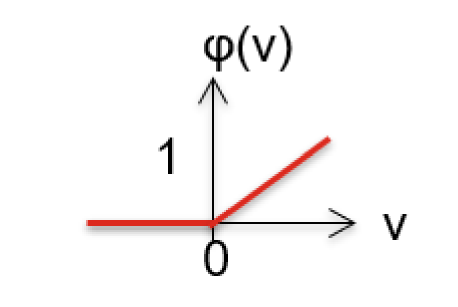
Activation Functions (3)

- Sigmoid function (s-shaped):
  - Commonly used in the past.
  - Strictly increasing function.
  - Asymptotically approach the saturation values.
- Continuous & Differentiable everywhere (very useful for optimisation later).
- Example: Logistic function (left) and hyperbolic tangent function (right).



Activation Functions (4)

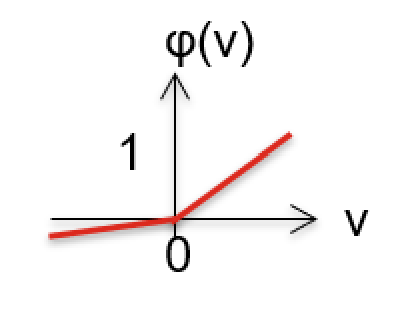
- Rectified Linear Unit (ReLU):
  - Output = 0, if input is negative.
  - Output = input, if input is positive.
  - Mathematicallly: $\boxed{\varphi(v) = \max\{0, v\} = \begin{cases} 0, v < 0 \\ v, v \geq 0 \end{cases}}$



  - Widely used nowadays, in the era of deep learning.
    - Multiplication of the gradient of sigmoid / tanh could become very small if there are many layers, leading to tiny change in weights and slow convergence (the "Vanishing Gradient" Problem)
    - On the other hand, the gradient of ReLU is 0 or 1, so after many layers the gradient will include the product of 1's which is not too small.

Activation Functions (5)

- Leaky ReLU:
  - ReLU has a problem that if too many v's are negative, then most of the ReLU output will simply be zero.
    - This would prohibit learning.
  - Leaky ReLU solves this by giving a small gradient when v is negative, for e.g.



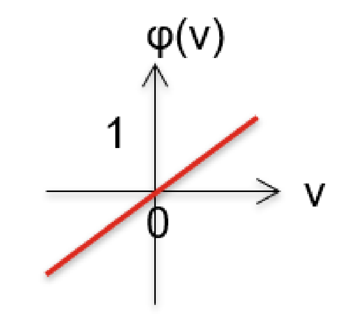
$$\boxed{\varphi(v) = max\{0.01v, v\} = \begin{cases} 0.01v, v < 0 \\ v, v \geq 0 \end{cases}}$$

Activation Functions (6)

- Linear:
  - The output is the same as the input.

$$\boxed{\varphi(v) = v}$$

  - Although no nonlinearity is created by this unit, it is useful as the activation function for the output layer of a multilayer NN, particularly for regression problem.

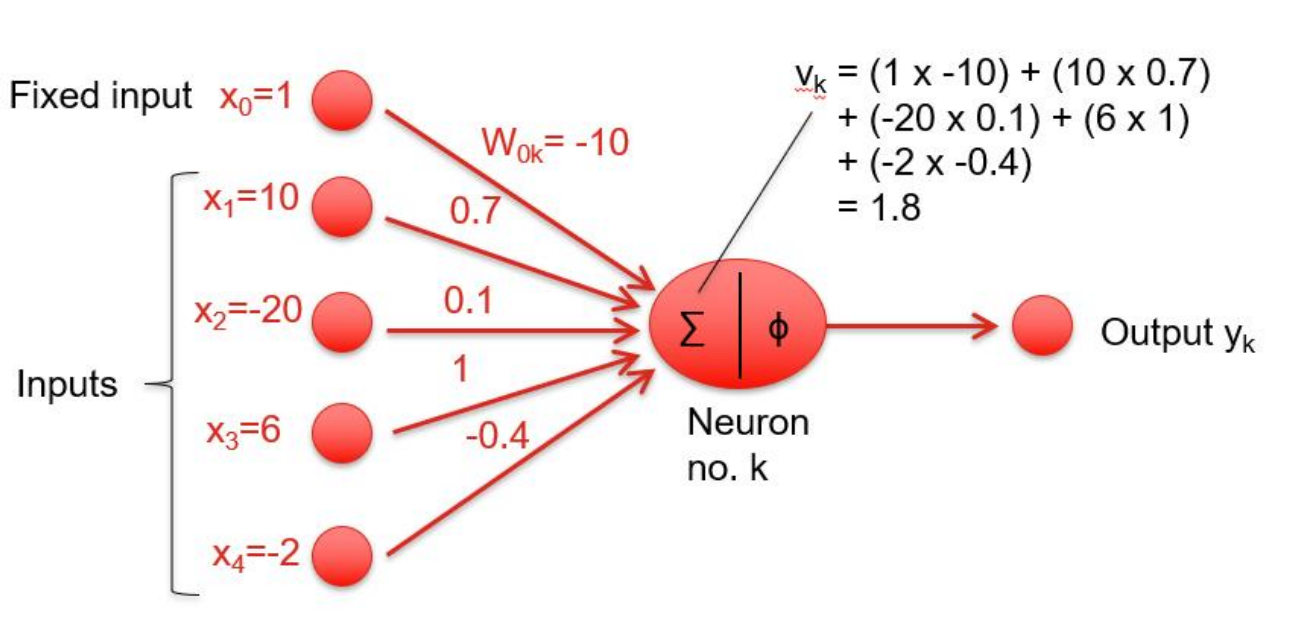


Example (1)

- A neuron k receives inputs from four other neurons whose activity levels are 10, -20, 6 and -2. The respective synaptic weights of neuron k are 0.7, 0.1, -1 and -0.4.
- Assume that the bias applied to the neuron is -10
- Calculate the output of neuron k for the following two situations:
  a) The neuron is linear / ReLU / leaky ReLU.
  b) The neuron is represented by a hard limiter.
  c) The neuron is represented by Sigmoid / Tanh.

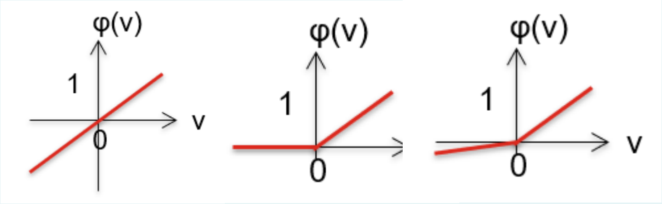
Example (2)

- Solution: Let's calculate the weighted sum first.

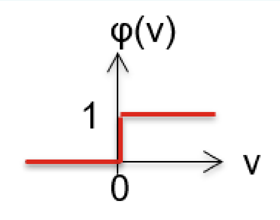


Example (3)

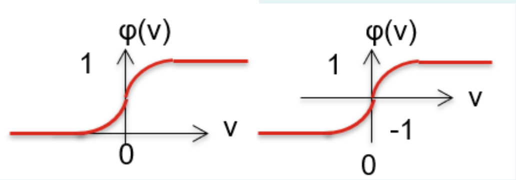
- Then calculate the output of activation function.
  - Linear / ReLU / Leaky Relu: $\varphi(1.8) = 1.8$



  - Hard limiter: $\varphi(1.8) = 1$

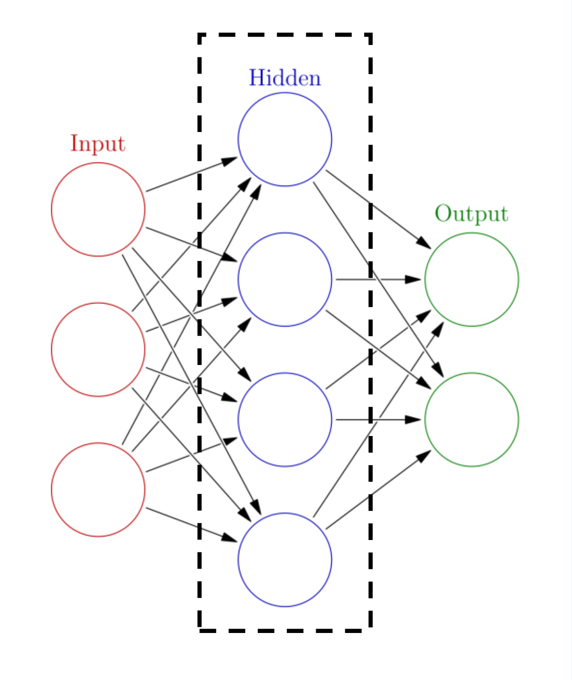


  - Sigmoid (with $a = 1$): $\varphi(1.8) = \frac{1}{1+e^{-1.8}} = 0.8581$
  - Tanh: $\varphi(1.8) = \tanh(1.8) = 0.9468$



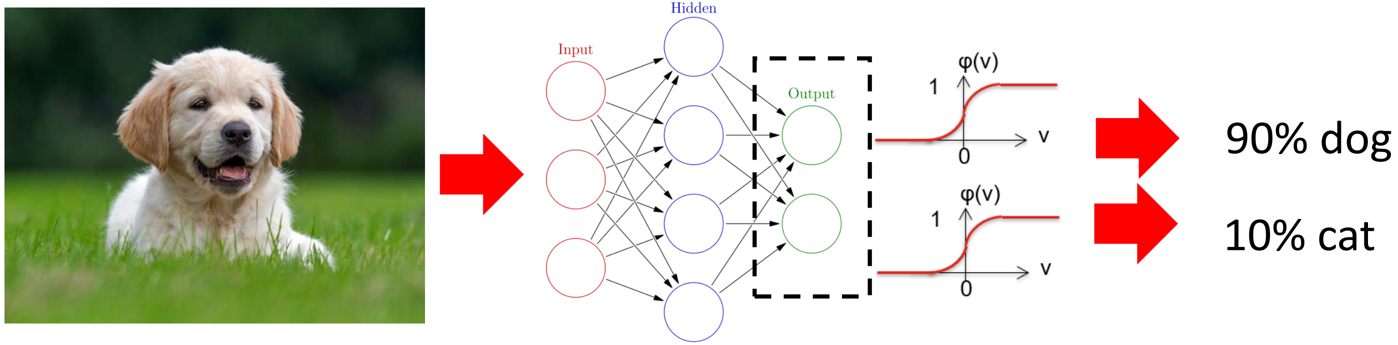
Choice of Activation Functions (1)

- Later, you will learn about the terms “hidden layer” and “output layer”.
- For hidden layer, it is common to use ReLU, leaky ReLU, tanh and log sigmoid as activation functions.
  - Usually selected based on trial-and-error, on a case-by-case basis.



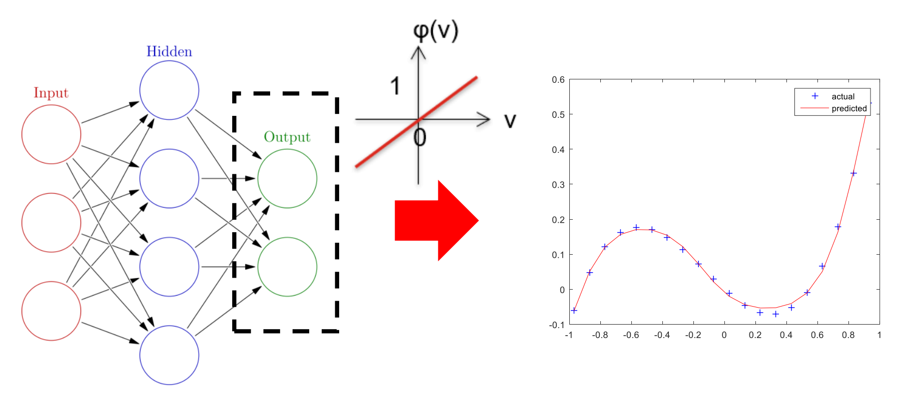
Choice of Activation Functions (2)

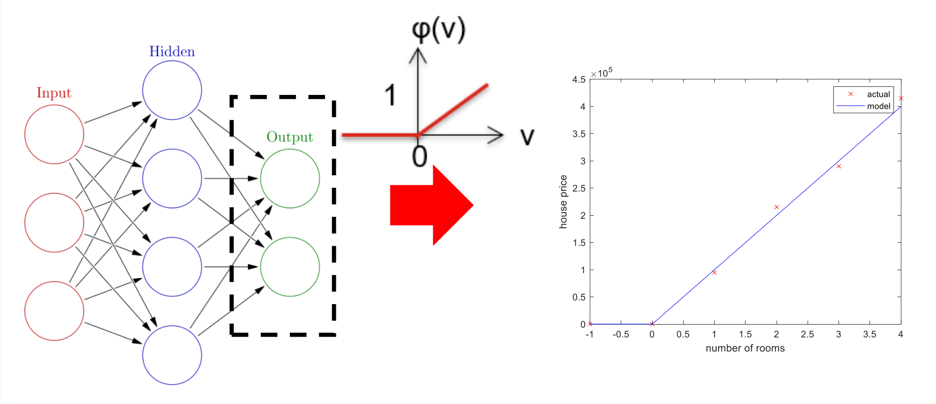
- For output layer, the choice will depend on the application and data.
  - For classification, in which the output is either 0 or 1, log sigmoid is an obvious choice.
    - E.g. output = 0.9 → 90% confident that the image is a dog.



Choice of Activation Functions (3)

- For regression, if the output is unconstrained (left), then linear function is an good option. It simply sums the nonlinearity created by the hidden layer.
- If the output is constrained (for e.g. right – house price cannot be less than 0!), then ReLU is a good choice.

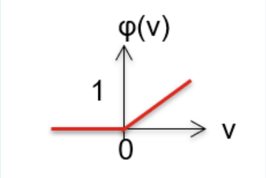




Differentiation of Activation Fcns (1)

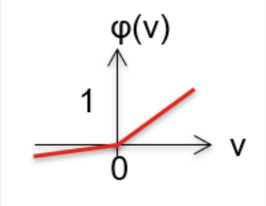
- For training of the neural networks, the weights will be updated through optimization / minimization of cost function.
- This involves differentiation of the cost function.
- Let's look at the derivatives of several activation functions:
  - ReLU:

$$\boxed{\varphi = max\{0, v\} = \begin{cases} 0, v < 0 \\ v, v \geq 0 \end{cases}} \rightarrow \boxed{\frac{d\varphi}{dv} = \begin{cases} 0, v < 0 \\ 1, v \geq 0 \end{cases}}$$



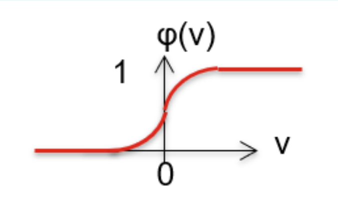
  - Leaky ReLU:

$$\boxed{\varphi(v) = max\{0.01v, v\} = \begin{cases} 0.01v, v < 0 \\ v, v \geq 0 \end{cases}} \rightarrow \boxed{\frac{d\varphi}{dv} = \begin{cases} 0.01, v < 0 \\ 1, v \geq 0 \end{cases}}$$



Differentiation of Activation Fcns (2)

- Log Sigmoid: $\boxed{\varphi(v) = \frac{1}{1+e^{-av}} = (1+e^{-av})^{-1}}$



$$\frac{d\varphi}{dv} = -(1+e^{-av})^{-2} \cdot e^{-av} \cdot (-a) = a \cdot \frac{e^{-av}}{(1+e^{-av})^2}$$

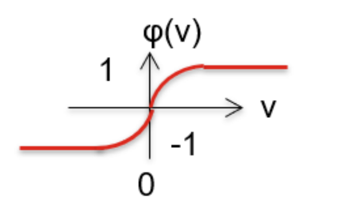
$$= a \cdot \frac{e^{-av}}{1+e^{-av}} \cdot \frac{1}{1+e^{-av}} = a \cdot \frac{1+e^{-av}-1}{1+e^{-av}} \cdot \frac{1}{1+e^{-av}}$$

$$= a \cdot \left(1 - \frac{1}{1+e^{-av}}\right) \cdot \frac{1}{1+e^{-av}} = a \cdot \big(1-\varphi(v)\big) \cdot \varphi(v)$$

$$\rightarrow \boxed{\frac{d\varphi}{dv} = a \cdot \big(1-\varphi(v)\big) \cdot \varphi(v)}$$

- That means, if you have the output value $\varphi(v)$, then you can get the derivative easily!

Differentiation of Activation Fcns (3)



- Tanh: $\boxed{\varphi(v) = \tanh(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}}}$

$$\frac{d\varphi}{dv} \text{ using quotient rule} = \frac{(e^v + e^{-v})(e^v + e^{-v}) - (e^v - e^{-v})(e^v - e^{-v})}{(e^v + e^{-v})^2}$$

$$= \frac{(e^v + e^{-v})^2 - (e^v - e^{-v})^2}{(e^v + e^{-v})^2} = 1 - \frac{(e^v - e^{-v})^2}{(e^v + e^{-v})^2} = 1 - \varphi^2(v)$$

$$\rightarrow \boxed{\frac{d\varphi}{dv} = 1 - \varphi^2(v)}$$

- Again, if you have the output value $\varphi(v)$, then you can get the derivative easily!

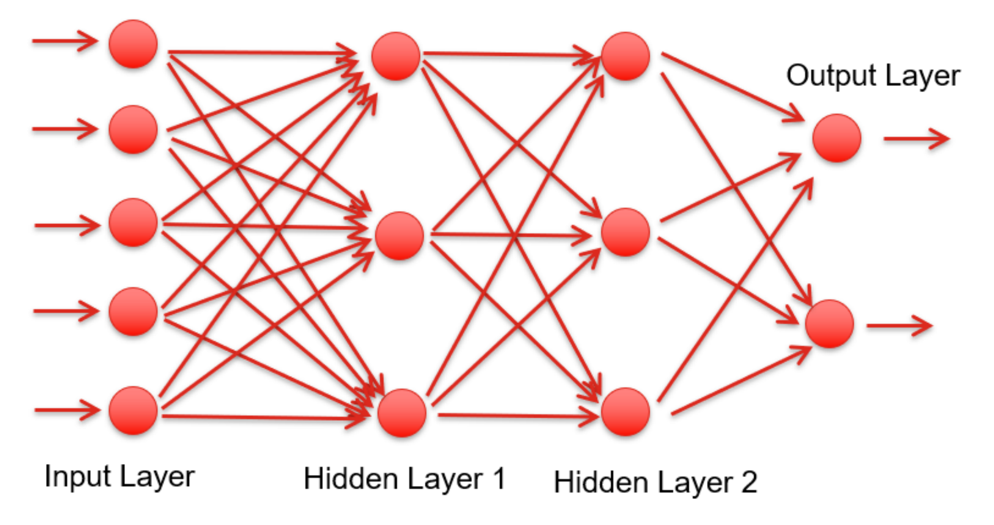
Content

Network Architectures (1)

- One single neuron is only able to solve some very simple problems.
- To solve for more complex problems, networks with large number of neurons are required.

Network Architectures (2)

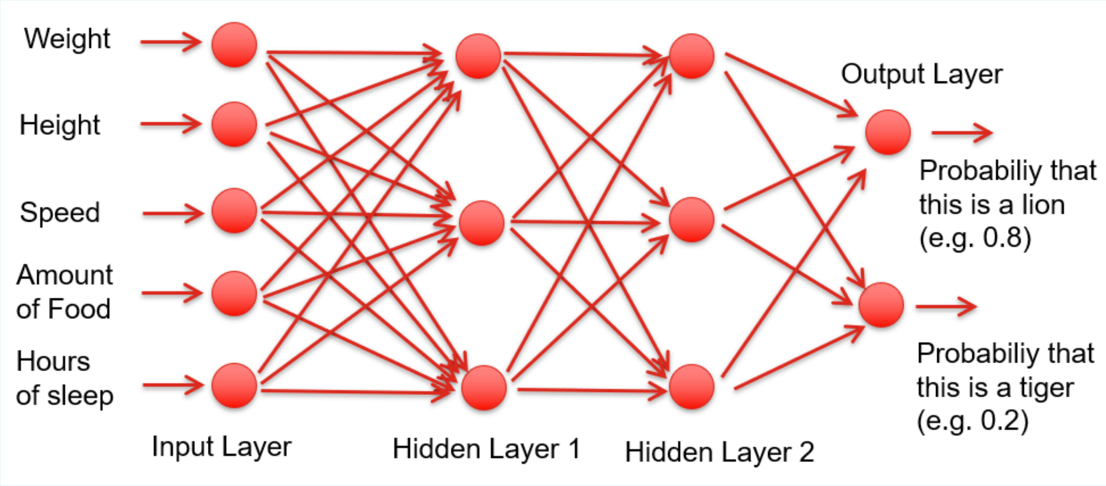
- Multilayer feedforward neural networks: Connections only from left to right.



- The above is a “5-3-3-2” network.

Network Architectures (3)

- Input Layer:
  - NO process carried out here. Only pass signal to the next layer.
  - NOT a design parameter.
    - Number of nodes will be the same as dimension of inputs.



Network Architectures (4)

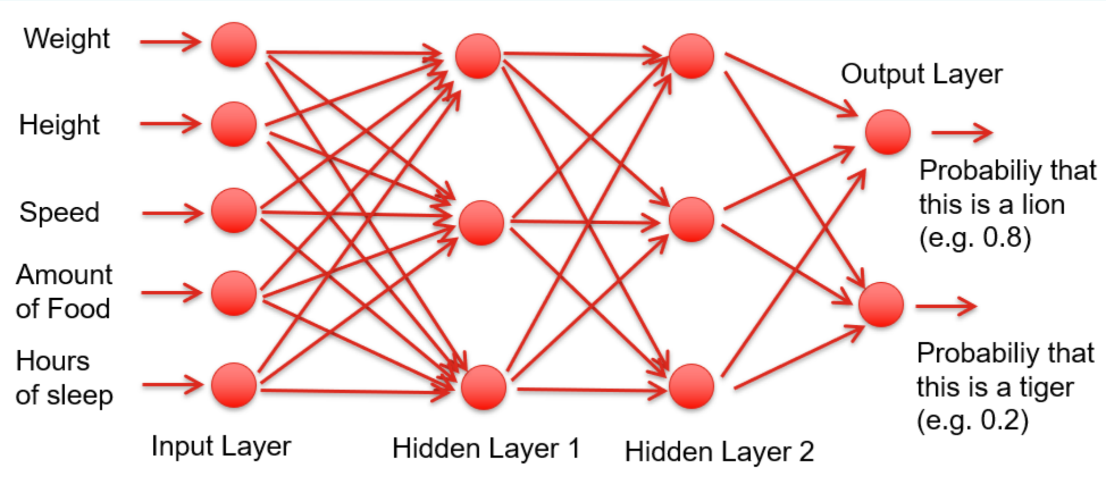
- Hidden Layer:
  - Design parameters:
    - Number of hidden layers.
    - Number of nodes in each hidden layer.
    - Activation functions – can be different for different layers.
  - It can be mathematically proven that one hidden layer is enough to approximate any bounded continuous function.
    - So why add more hidden layers?

Network Architectures (5)

- Advantage of having more hidden layers:
  - The total number of synapses weight in multilayer network is less → less parameters to tune
    - E.g. 1-9-1 network: 28 weights (including biases)
    - 1-3-3-1 network: 22 weights (including biases)
- Disadvantage of having more hidden layers:
  - More prone to local minima due to its more complicated structure.

Network Architectures (6)

- Output Layer:
  - Number of nodes is NOT a design parameters. Will be the same as dimension of outputs (# functions for regression, # classes for classification).
  - Activation function is a design parameter.
    - Depends on the expected output, as already discussed in the section “activation function”.



Content

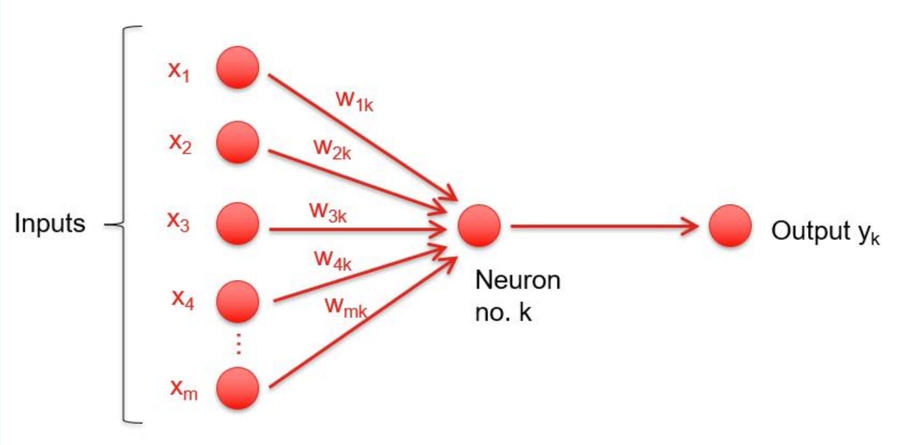
- Introduction to Neural Networks
- The Biological Neuron
- The Artificial Neuron
- Activation Functions
- Network Architectures
- Perceptrons
- Multilayer Perceptrons

Perceptron (1)

- Earlier on, we mentioned that knowledge is acquired by a neural network through learning.
- Let’s use a simple network – a “perceptron” – to demonstrate what this means.

Perceptron (2)

- Perceptron is the simplest form of a NN for classification of patterns.
  - It learns via examples, how to assign input vectors (samples) to different classes (Rosenblatt, 1958).



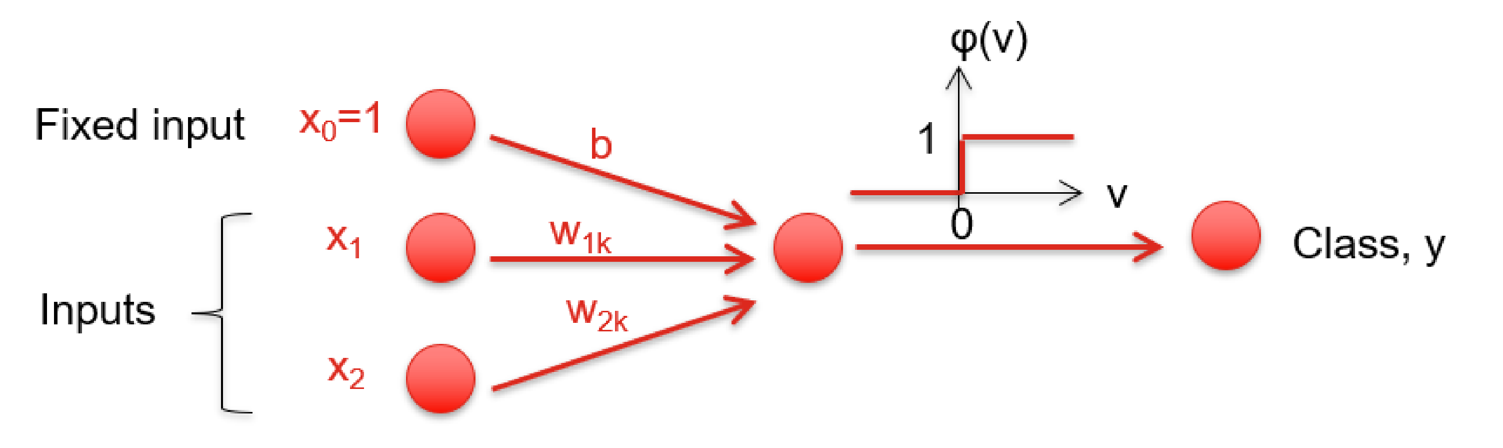
2D example (1)

- A 2-dimensional example: To correctly classify the external inputs $\underbrace{\{x_1, x_2\}}_{\text{2D}}$ into one of two classes $\{C_1$ or $C_0\}$.
- E.g. AND problem:

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
Y (output)	0	0	0	1

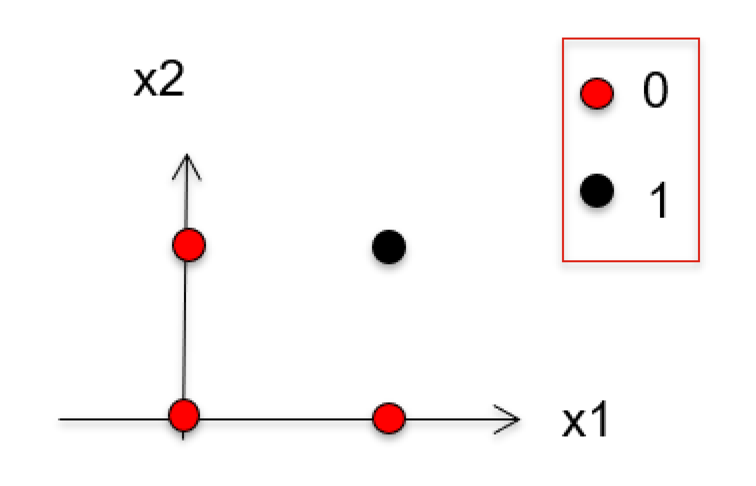
2D example (2)

- The perceptron is shown below. We want to find b, w_{k1} and w_{k2}, so that when given x1 and x2, the correct y will be calculated.



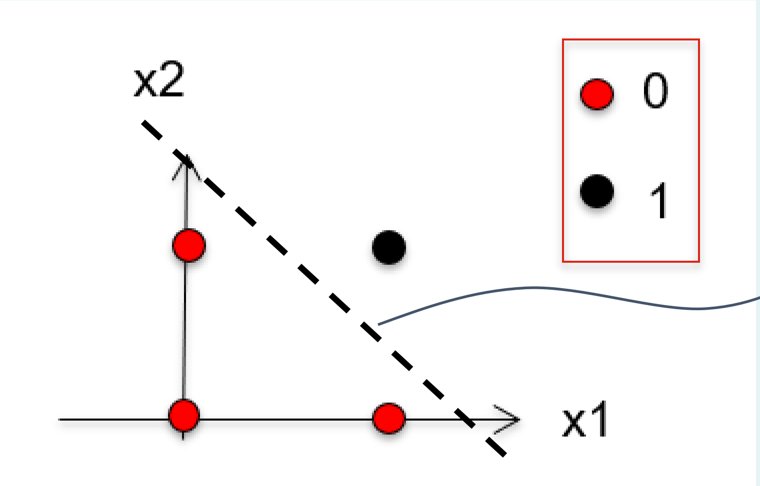
Analytical Solution (1)

- Let's solve this analytically first.
- This is a simple 2-dimensional problem, thus we can sketch the input-output space:



Analytical Solution (2)

- The classes are linearly separable and we can easily get a straight line to separate the two classes.



Decision boundary:

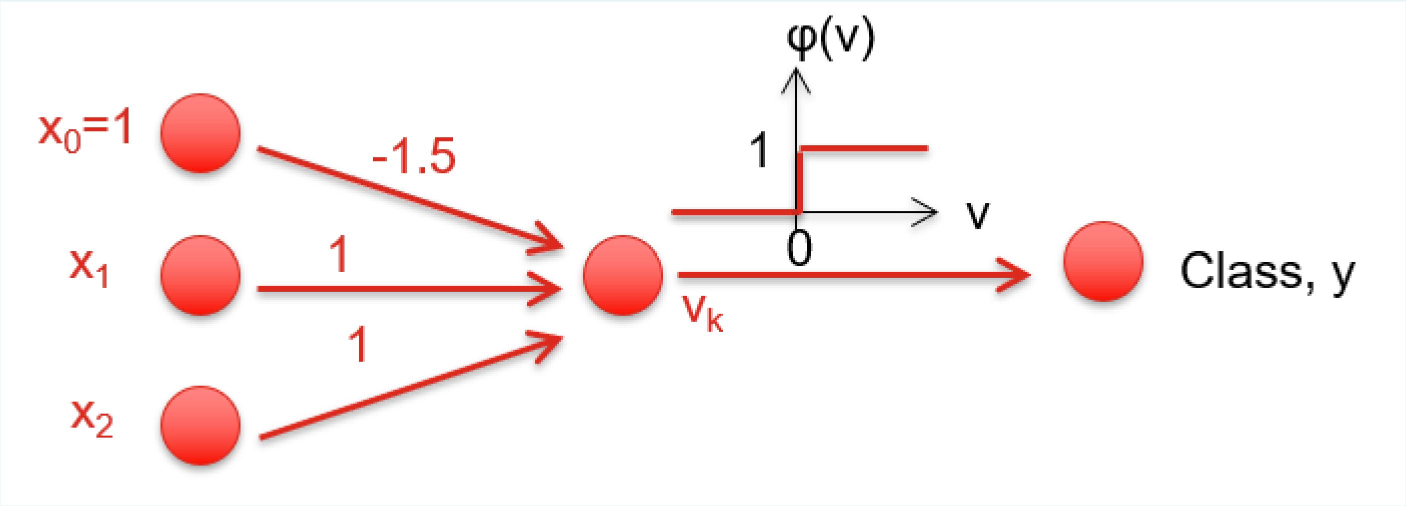
$\boxed{x2 = -x1 + 1.5}$

Analytical Solution (3)

- From the line equation, we would either have:
  - $v_k = -x1 - x2 + 1.5$, or
  - $v_k = x1 + x2 - 1.5$.
- Which one is correct?
- Try with x1 = 0, x2 = 0:
  - First equation gives $v_k = 1.5$, then $\varphi(1.5) = 1 \rightarrow$ Wrong
  - Second equation gives $v_k = -1.5$, then $\varphi(-1.5) = 0 \rightarrow$ Correct!

Analytical Solution (4)

- Thus, the complete perceptron is as follows:
  - (from $v_k = x1 + x2 - 1.5$)



Learning Method (1)

- We will now use a learning procedure to train the perceptron.
  - A training set of input-output vectors, i.e. exemplars, is given.
  - The weight vector will be tuned in such a way that the best classification of the training vectors is achieved.
- The learning procedure works even for more general cases (more than 2D), so we will define some terms on the next page.

Learning Method (2)

- The input vector is:

$$\boxed{x'(t) = [1, x_1(t), x_2(t), \dots, x_m(t)\ ]^T}$$

- The weight vector is:

$$\boxed{w'(t) = [b(t), w_1(t), w_2(t), \dots, w_m(t)\ ]^T}$$

- Where t denotes the iteration step.
- The intermediate output (before the hard limiter) is:

$$\boxed{v(t) = w'^T(t) \cdot x'(t)}$$

Learning Method (3)

- The equation $w'^T \cdot x' = 0$, plotted in an m-dimensional space with coordinates $x_1, x_2, \ldots, x_m$, defines a hyperplane which separates the two classes.
- With the help of hard limiter, we finally get:

$$\boxed{\begin{array}{l} w'^T \cdot x' \geq 0 \text{ Class 1} \\ w'^T \cdot x' < 0 \text{ Class 0} \end{array}}$$

Perceptron Learning Algorithm

- Start with randomly chosen weight vector $w'(0)$.
- Let t $= 1$.
- **While** there exist input vectors that are wrongly classified by $w'(t-1)$**, do**
  - If x' is a misclassified input vector,
  - Update the weight vector to

$$\boxed{w'(t) = \mathrm{w}'(t-1) + \eta(d-y)x'}$$

  - Where $\eta > 0$ and $d = \begin{cases} 1 \text{ if } x \text{ belongs to Class } 1 \\ 0 \text{ if } x \text{ belongs to Class } 0 \end{cases}$, and $\begin{matrix} d = \text{desired output} \\ y = \text{network output} \end{matrix}$
  - Increment n
- **End While**

AND-Example Revisited (1-1)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- Randomly initiated weight: $w'(0) = [0.5, 0.5, 0.5]^T$

- First epoch, first column of data

$$w'(0)^T \cdot x = [0.5, 0.5, 0.5] \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0.5 \to y = \varphi(0.5) = 1, d = 0 \to \text{misclassified}$$

- Because of misclassification, update w to:

$$w'(1) = \mathrm{w}'(0) + \underbrace{\eta}_{0.1} (d - y)x' = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} + 0.1(0-1) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.4 \\ 0.5 \\ 0.5 \end{bmatrix}$$

(Goes to next step)

AND-Example Revisited (1-2)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(1) = [0.4, 0.5, 0.5]^T$

- First epoch, second column of data

$$w'(1)^T \cdot x = [0.4, 0.5, 0.5] \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 0.9 \rightarrow y = \varphi(0.9) = 1, d = 0 \rightarrow \text{misclassified}$$

- Because of misclassification, update w to:

$$w'(2) = \mathrm{w}'(1) + \eta(d - y)x' = \begin{bmatrix} 0.4 \\ 0.5 \\ 0.5 \end{bmatrix} + 0.1(0 - 1)\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.5 \\ 0.4 \end{bmatrix}$$

(Goes to next step)

AND-Example Revisited (1-3)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(2) = [0.3, 0.5, 0.4]^T$

- First epoch, third column of data

$$w'(2)^T \cdot x = [0.3, 0.5, 0.4] \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 0.8 \rightarrow y = \varphi(0.8) = 1, d = 0 \rightarrow \text{misclassified}$$

- Because of misclassification, update w to:

$$w'(3) = \mathrm{w}'(2) + \eta(d - y)x' = \begin{bmatrix} 0.3 \\ 0.5 \\ 0.4 \end{bmatrix} + 0.1(0 - 1) \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.4 \end{bmatrix}$$

(Goes to next step)

AND-Example Revisited (1-4)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(3) = [0.2, 0.4, 0.4]^T$

- First epoch, fourth column of data

$$w'(3)^T \cdot x = [0.2, 0.4, 0.4] \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1 \rightarrow y = \varphi(1) = 1, d = 1 \rightarrow \text{correct}$$

- Because the classification is correct, w remains the same.

$$w'(4) = \mathrm{w}'(3) = [0.2, 0.4, 0.4]^T$$

- We have completed the first epoch, i.e. all the data has been presented once.

AND-Example Revisited (2-1)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(4) = [0.2, 0.4, 0.4]^T$

- Continue to the second epoch, first column of data

$$w'(4)^T \cdot x = [0.2, 0.4, 0.4] \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0.2 \rightarrow y = \varphi(0.2) = 1, d = 0 \rightarrow \text{misclassified}$$

- Because of misclassification, update w to:

(Goes to next step)

$$w'(5) = \mathrm{w}'(4) + \eta(d - y)x' = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.4 \end{bmatrix} + 0.1(0 - 1) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.4 \\ 0.4 \end{bmatrix}$$

AND-Example Revisited (2-2)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(5) = [0.1, 0.4, 0.4]^T$

- Second epoch, second column of data

$$w'(5)^T \cdot x = [0.1, 0.4, 0.4] \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 0.5 \rightarrow y = \varphi(0.5) = 1, d = 0 \rightarrow \text{misclassified}$$

- Because of misclassification, update w to:

$$w'(6) = \mathrm{w}'(5) + \eta(d - y)x' = \begin{bmatrix} 0.1 \\ 0.4 \\ 0.4 \end{bmatrix} + 0.1(0 - 1)\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.4 \\ 0.3 \end{bmatrix}$$

(Goes to next step)

AND-Example Revisited (2-3)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(6) = [0, 0.4, 0.3]^T$

- Second epoch, third column of data

$$w'(6)^T \cdot x = [0, 0.4, 0.3] \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 0.4 \rightarrow y = \varphi(0.4) = 1, d = 0 \rightarrow \text{misclassified}$$

- Because of misclassification, update w to:

$$w'(7) = \mathrm{w}'(6) + \eta(d - y)x' = \begin{bmatrix} 0 \\ 0.4 \\ 0.3 \end{bmatrix} + 0.1(0 - 1) \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0.3 \\ 0.3 \end{bmatrix}$$

(Goes to next step)

AND-Example Revisited (2-4)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(7) = [-0.1, 0.3, 0.3]^T$

- Second epoch, fourth column of data

$$w'(7)^T \cdot x = [-0.1, 0.3, 0.3] \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0.5 \rightarrow y = \varphi(0.5) = 1, d = 1 \rightarrow \text{correct}$$

- Because the classification is correct, w remains the same.

$$w'(8) = \mathrm{w}'(7) = [-0.1, 0.3, 0.3]^T$$

- Second epoch completed.

AND-Example Revisited (3-1)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(8) = [-0.1, 0.3, 0.3]^T$

- Continue to third epoch, first column of data

$$w'(8)^T \cdot x = [-0.1, 0.3, 0.3] \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = -0.1 \rightarrow y = \varphi(-0.1) = 0, d = 0 \rightarrow \text{correct}$$

- Because the classification is correct, w remains the same.

$$w'(9) = \mathrm{w}'(8) = \begin{bmatrix} -0.1 \\ 0.3 \\ 0.3 \end{bmatrix}$$

(Goes to next step)

AND-Example Revisited (3-2)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(9) = [-0.1, 0.3, 0.3]^T$

- Third epoch, second column of data

$$w'(9)^T \cdot x = [-0.1, 0.3, 0.3] \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 0.2 \rightarrow y = \varphi(0.2) = 1, d = 0 \rightarrow \text{misclassified}$$

- Because of misclassification, update w to:

$$w'(10) = \mathrm{w}'(9) + \eta(d-y)x' = \begin{bmatrix} -0.1 \\ 0.3 \\ 0.3 \end{bmatrix} + 0.1(0-1)\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.3 \\ 0.2 \end{bmatrix}$$

(Goes to next step)

AND-Example Revisited (3-3)

X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(10) = [-0.2, 0.3, 0.2]^T$
- Third epoch, third column of data

$$w'(10)^T \cdot x = [-0.2, 0.3, 0.2] \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 0.1 \rightarrow y = \varphi(0.1) = 1, d = 0 \rightarrow \text{misclassified}$$

- Because of misclassification, update w to:

(Goes to next step)

$$w'(11) = \mathrm{w}'(10) + \eta(d - y)x' = \begin{bmatrix} -0.2 \\ 0.3 \\ 0.2 \end{bmatrix} + 0.1(0 - 1) \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.3 \\ 0.2 \\ 0.2 \end{bmatrix}$$

AND-Example Revisited (3-4)

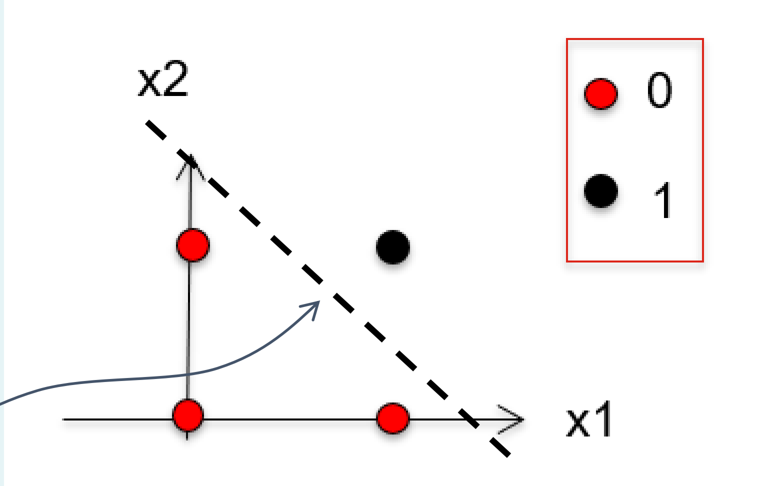
X1 (input)	0	0	1	1
X2 (input)	0	1	0	1
d (desired output)	0	0	0	1

- After previous step: $w'(11) = [-0.3, 0.2, 0.2]^T$

- We can continue doing the same, and it will be observed that with $w' = [-0.3,0.2,0.2]^T$ always gives the correct classification!
- The perceptron is successfully trained after 3 epochs!

AND-Example Revisited (4)

- Summary: After three epochs, we arrive at the weight $w' = [-0.3, 0.2, 0.2]^T$ which correctly classifies all the data point.
- It *looks* different from what we manually calculated ($w'_{\text{manual}} = [-1.5, 1, 1]^T$) but is actually equivalent!
  - Manual: x2 = -x1 + 1.5
  - Trained: 0.2 x2 = -0.2 x1 + 0.3



MATLAB Code (1)

```
clear all
close all
clc

%%%%%%%%
% Data %
%%%%%%%%

% In the order of [x0 x1 x2 d]
Data = [1 0 0 0;
    1 0 1 0;
    1 1 0 0;
    1 1 1 1];

%%%%%%%%%%%%%%
% Parameters %
%%%%%%%%%%%%%%

w = [0.5, 0.5, 0.5]'; % [bias w1 w2]
eta = 0.1; % Try changing this and observe results
epochs = 4;
wrecord = w;
```

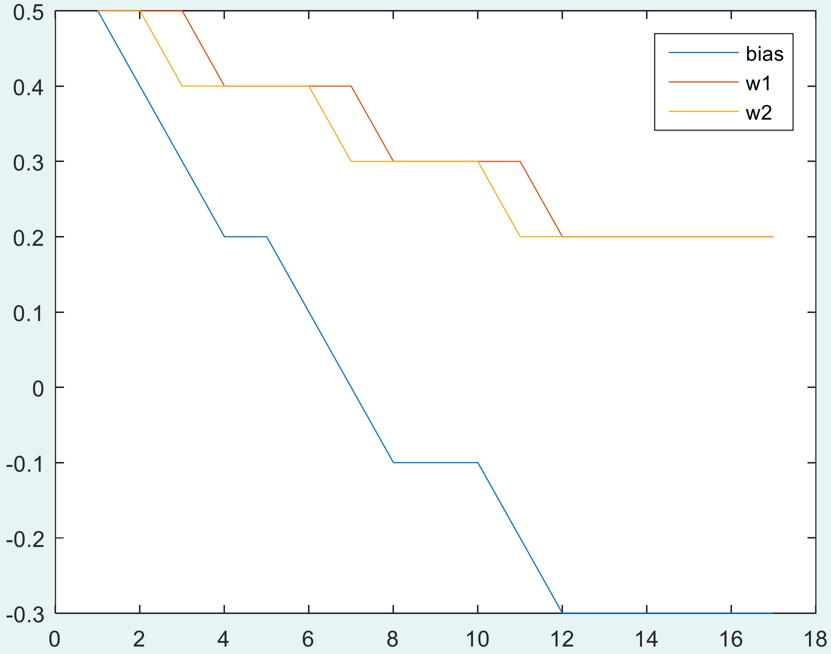
```
%%%%%%%%%%%%%
% Algorithm %
%%%%%%%%%%%%%

[ndata,mdata] = size(Data);

for i = 1:epochs
    for j = 1:ndata
        x = Data(j,1:3)';
        v = w'*x;
        if v>=0
            y=1;
        else
            y=0;
        end
        d = Data(j,4);
        w = w + eta*(d-y)*x;
        wrecord = [wrecord w];
    end
end
figure, plot(wrecord(1,:))
hold on, plot(wrecord(2,:))
hold on, plot(wrecord(3,:))
legend('bias','w1','w2')
```

MATLAB Code (2)

- Progress of the weights during training.



Learning Rate (1)

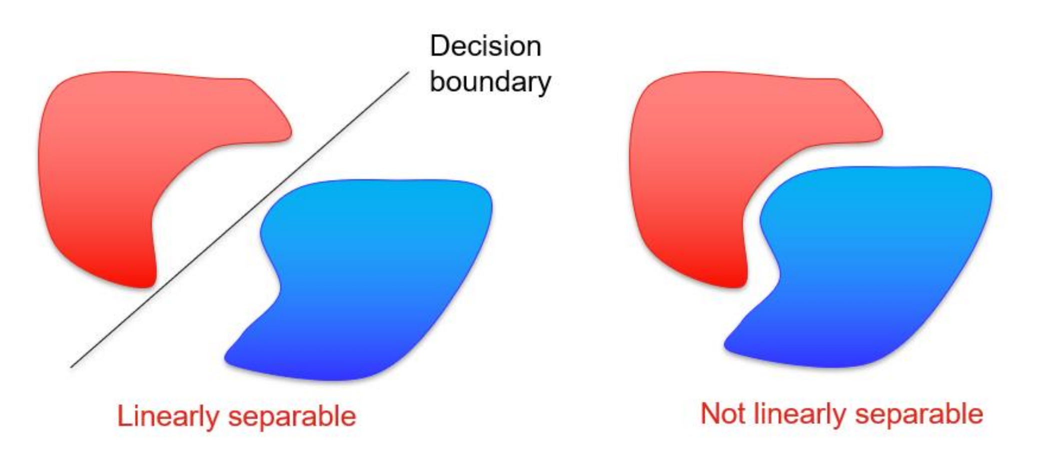
- The parameter $\eta > 0$ influences the learning rate.
- Small value leads to slow learning.
- Large value can “spoil” the learning that has taken place earlier with respect to other data points.
- Therefore, some medium value is the best.
  - What “medium” means depends on the problem being solved.

Learning Rate (2)

- Exercise: Try changing the η value in the MATLAB code earlier, and observe the results.
  - Note: You might need to increase the epochs to allow the weights to converge.

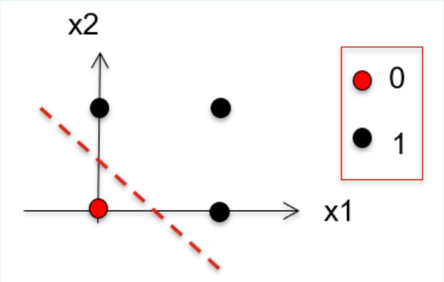
Limitations of Perceptrons (1)

- Perceptrons can only classify two classes which are linearly separable:
  - E.g. 2-D case, if the classes cannot be separated by a straight line, then they cannot be classified by simple perceptrons.

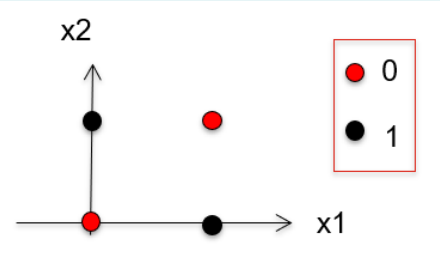


Limitations of Perceptrons (2)

- AND – Linearly separable, as already seen earlier.
- OR – also linearly separable:



- XOR - ???

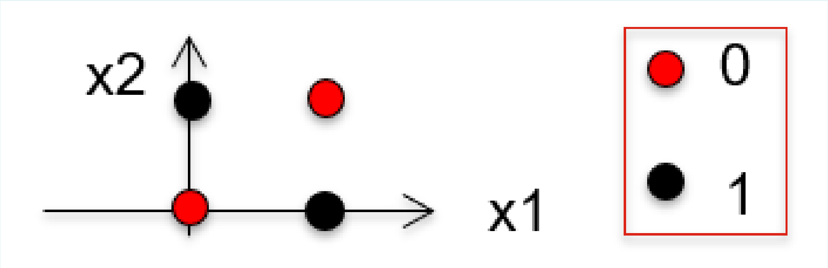


Content

- Introduction to Neural Networks
- The Biological Neuron
- The Artificial Neuron
- Activation Functions
- Network Architectures
- Perceptrons
- Multilayer Perceptrons

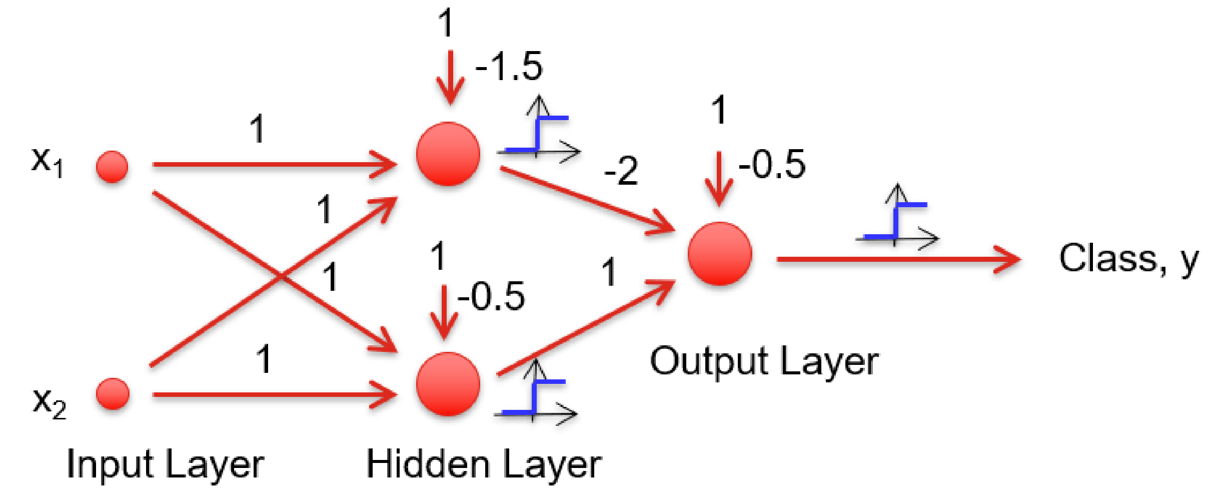
XOR Problem (1)

- We saw that a single perceptron is not able to solve the XOR problem.



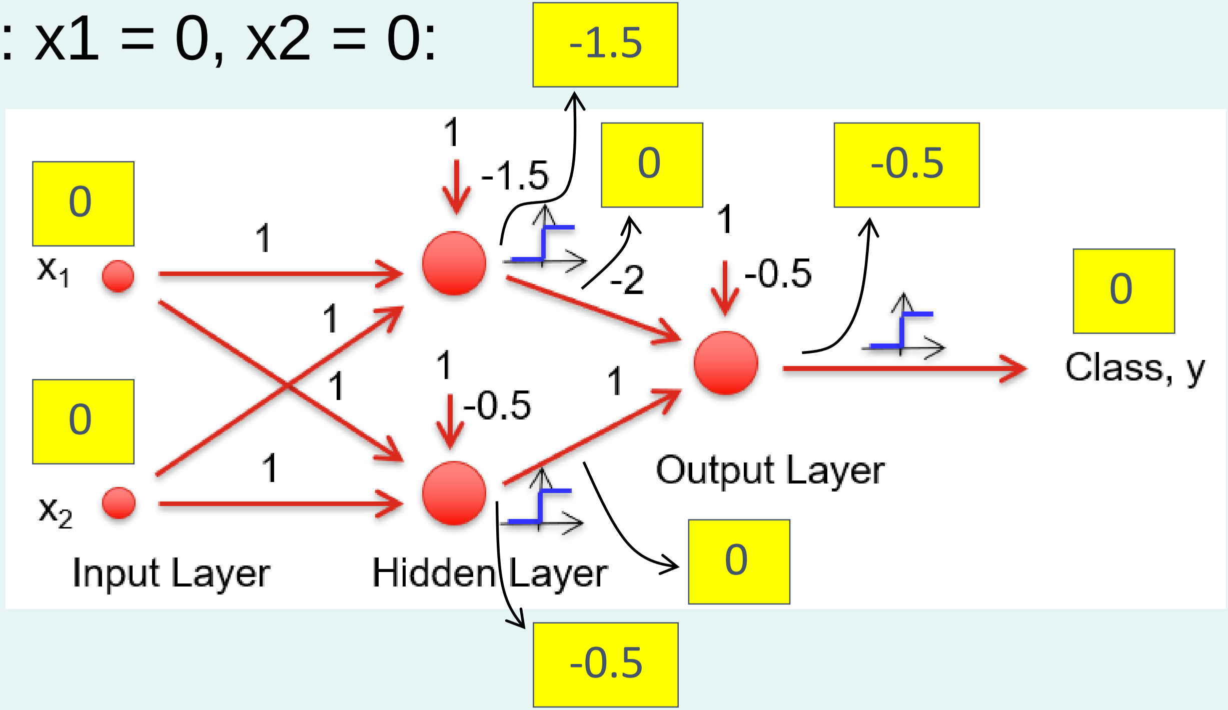
XOR Problem (2)

- To do that, we need a few perceptrons working together.
- For instance, consider the following network by Touretzy and Pomerlau (1989):



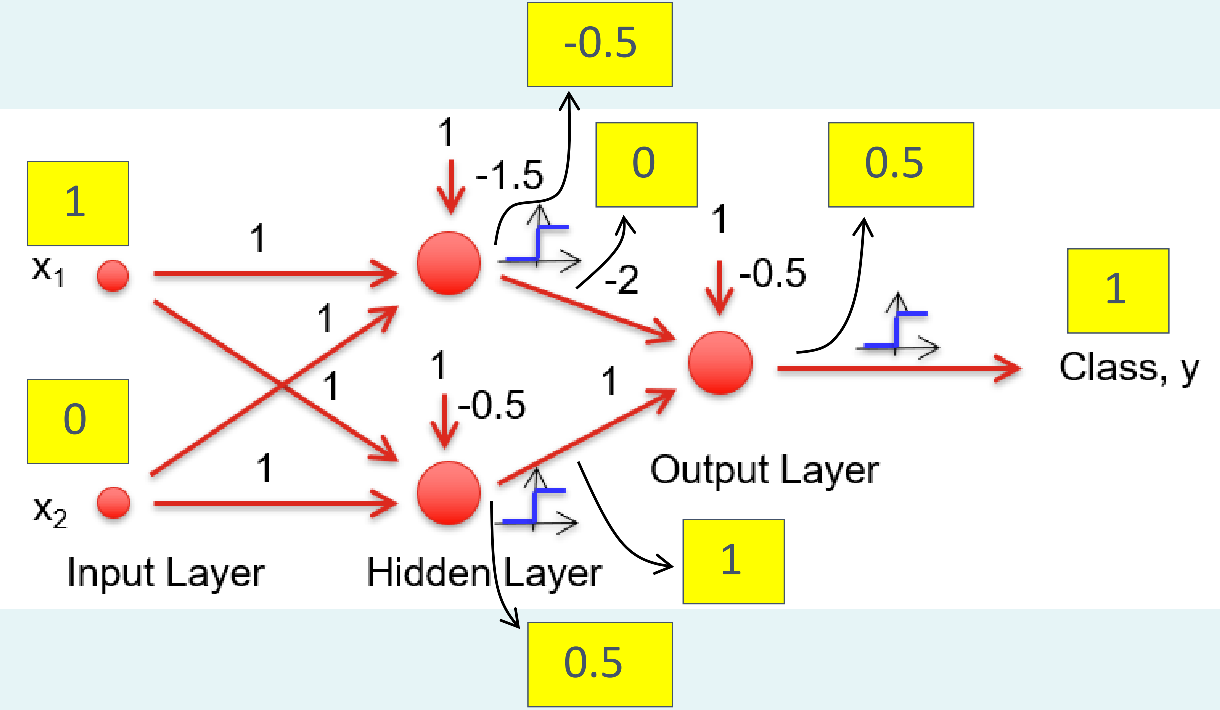
XOR Problem (3)

- Let's check the results:
- Case 1: x1 = 0, x2 = 0:



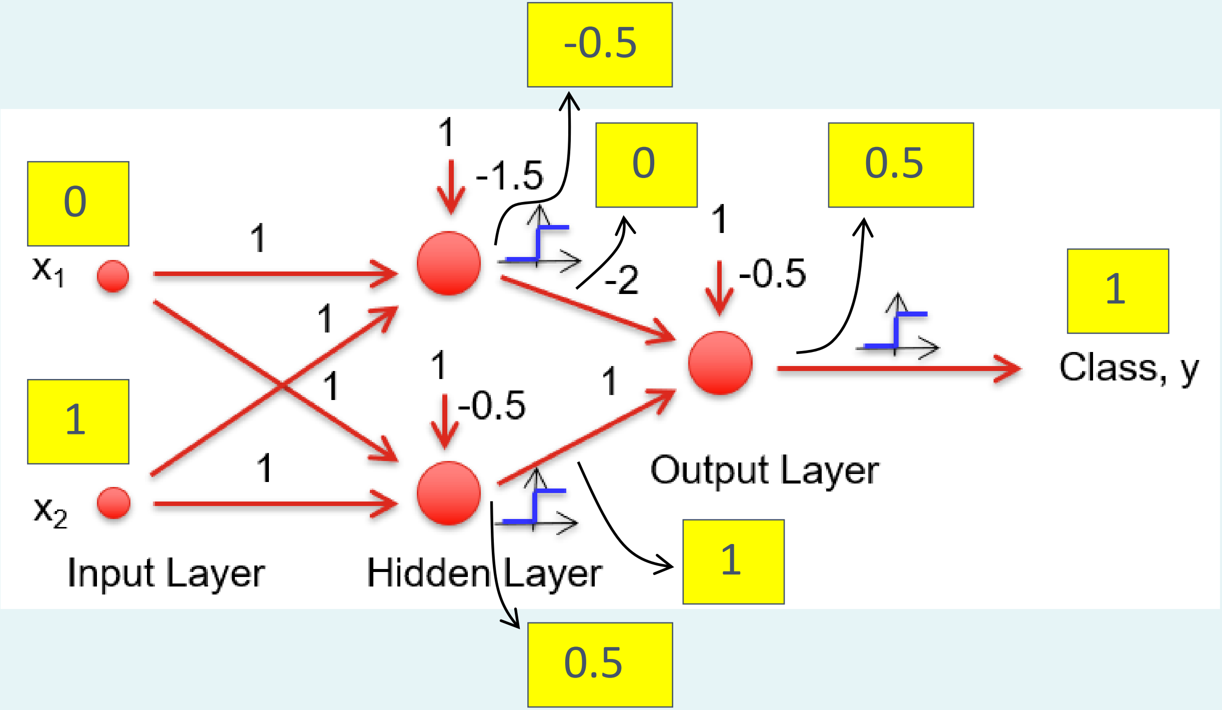
XOR Problem (4)

- Case 2: x1 = 1, x2 = 0:



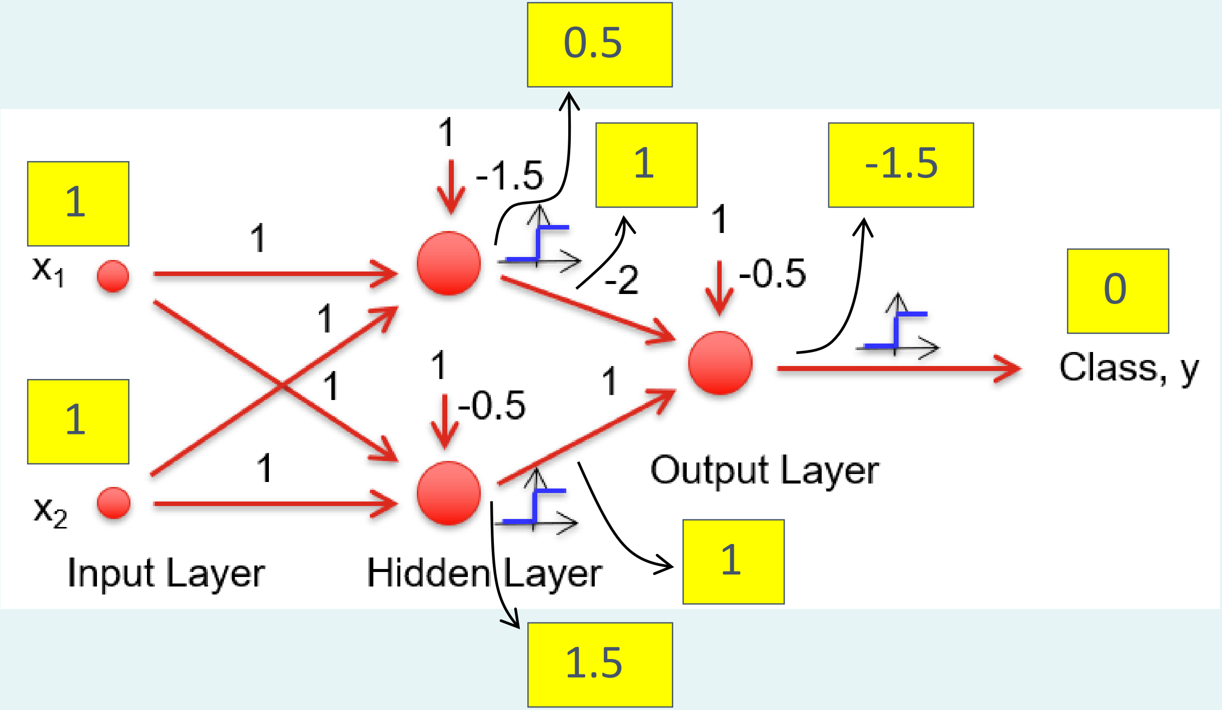
XOR Problem (5)

- Case 3: x1 = 0, x2 = 1:



XOR Problem (6)

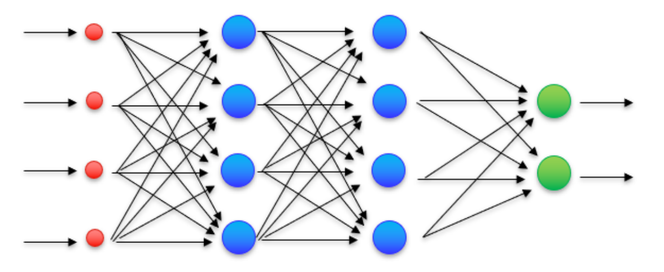
- Case 4: x1 = 1, x2 = 1:



Correctly solves the XOR problem!

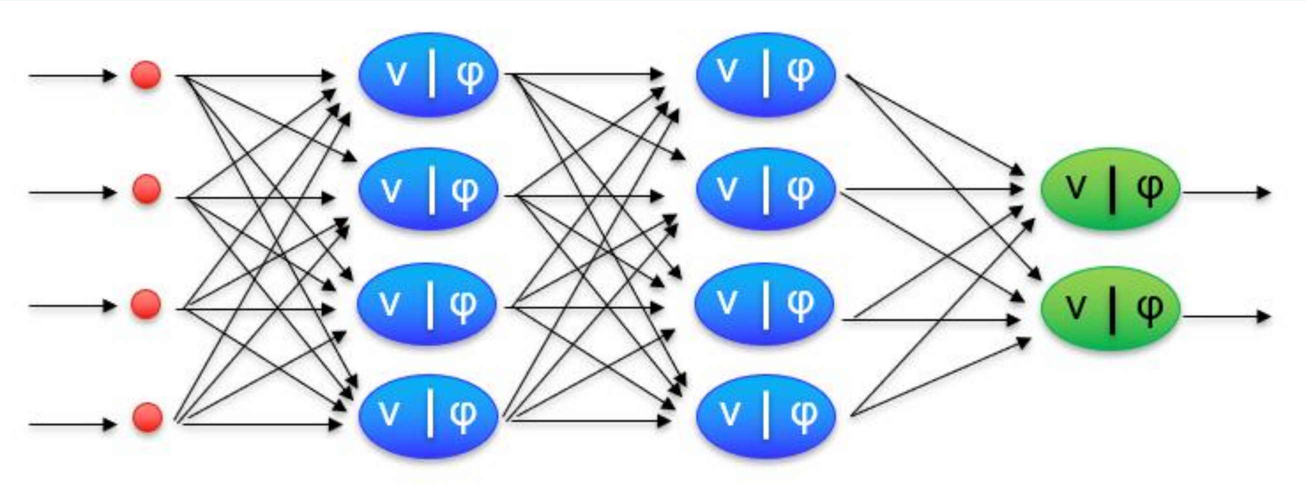
Multilayer Perceptrons MLP (1)

- As can be seen, by connecting multiple perceptrons together, more complicated problems can be solved.
- A Multilayer Perceptron (MLP) consists of:
  - An input layer (note: no computation here).
  - One or more hidden layers of computation nodes.
  - An output layer of computation nodes.
- E.g.



Multilayer Perceptrons MLP (2)

- Again, it is useful to think of the hidden layer nodes and output layer nodes as two halves (adder + activation):



  - For convenience, we will also not explicitly draw out the biases.

Multilayer Perceptrons MLP (3)

- As mentioned previously, MLP generally uses differentiable activation functions instead of threshold function.
  - This is because for training of the weights, we perform optimization which require the functions to be differentiable.

Thank you for your attention!

Any questions?