MACHINE LEARNING

1211635

MAHASARAKHAM
UNIVERSITY

ARTIFICIAL NEURAL NETWORKS

Introduction and application

Olarik Surinta, PhD.
Lecturer

Neural network

- Nerve cells in the brain are called neurons. There is an estimated 1010 to the power(1013) neurons in the human brain.
- Each neuron can make contact with several thousand other neurons.
- Neurons are the unit which the brain uses to process information.

What does a neuron look like

- A neuron consists of a cell body, with various extensions from it. Most of these are branches called dendrites.
- There is one much longer process (possibly also branching) called the axon.

What does a neuron look like

- The dashed line shows the axon hillock, where transmission of signals starts.

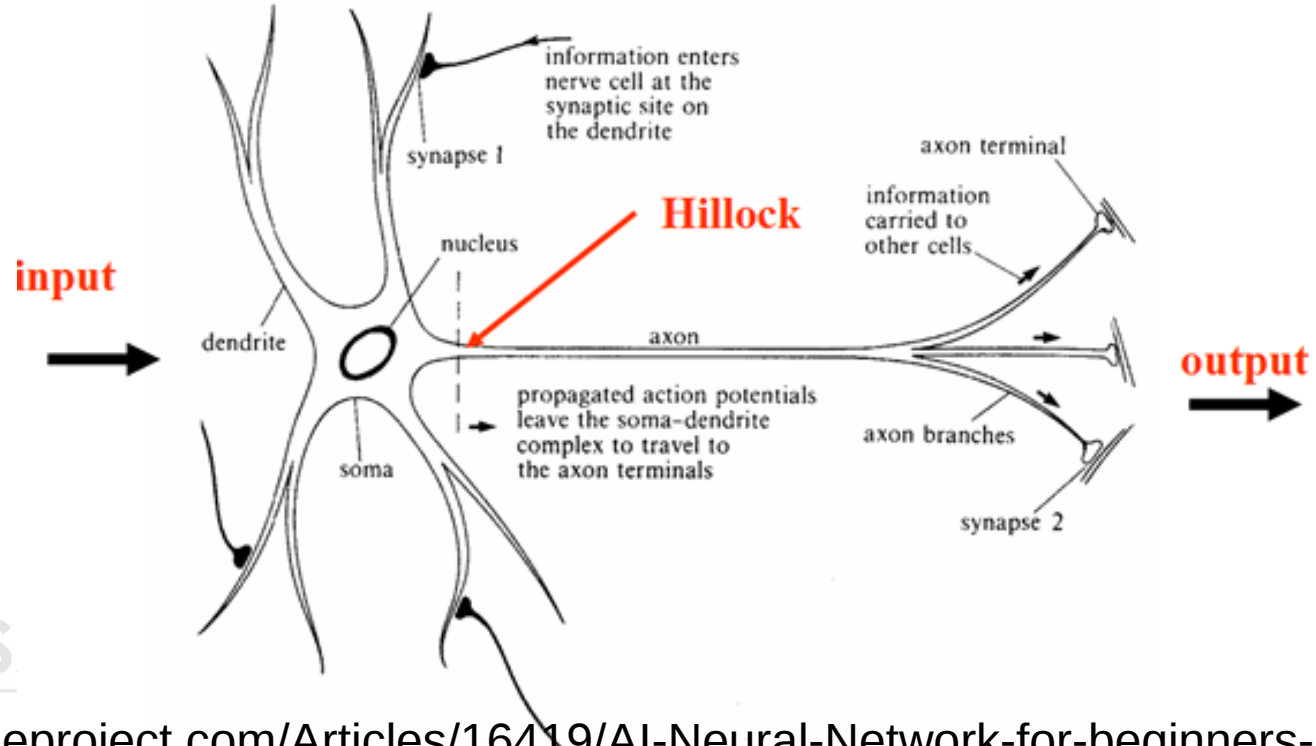


Figure: Neuron

Cr. https://www.codeproject.com/Articles/16419/AI-Neural-Network-for-beginners-Part-of

What does a neuron look like

- The boundary of the neuron is known as the cell membrane.
- There is a voltage difference (the membrane potential) between the inside and outside of the membrane.
- If the input is large enough, an action potential is then generated. The action potential (neuronal spike) then travels down the axon, away from the cell body.

What does a neuron look like

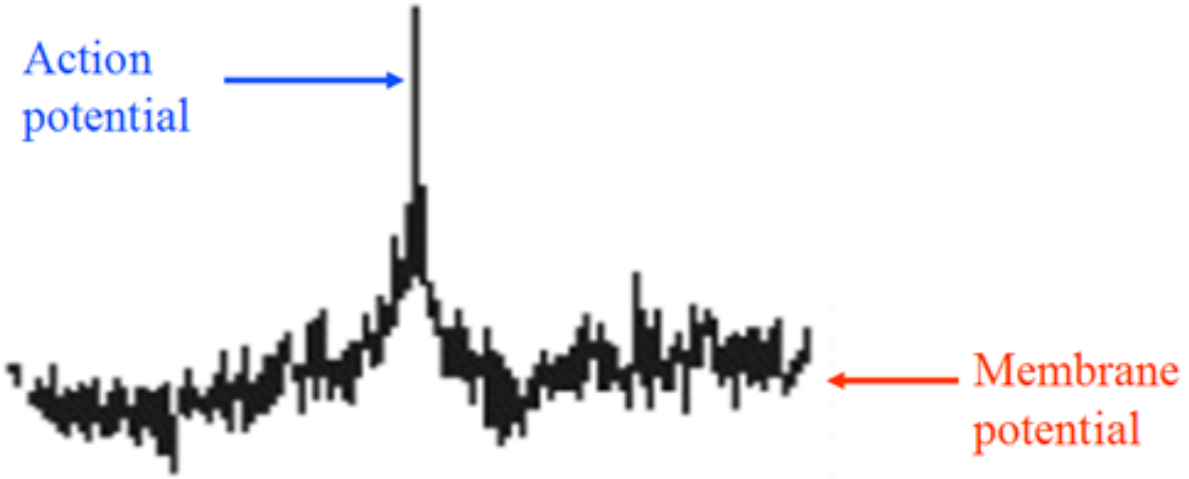


Figure: Neuron spiking

What does a neuron look like

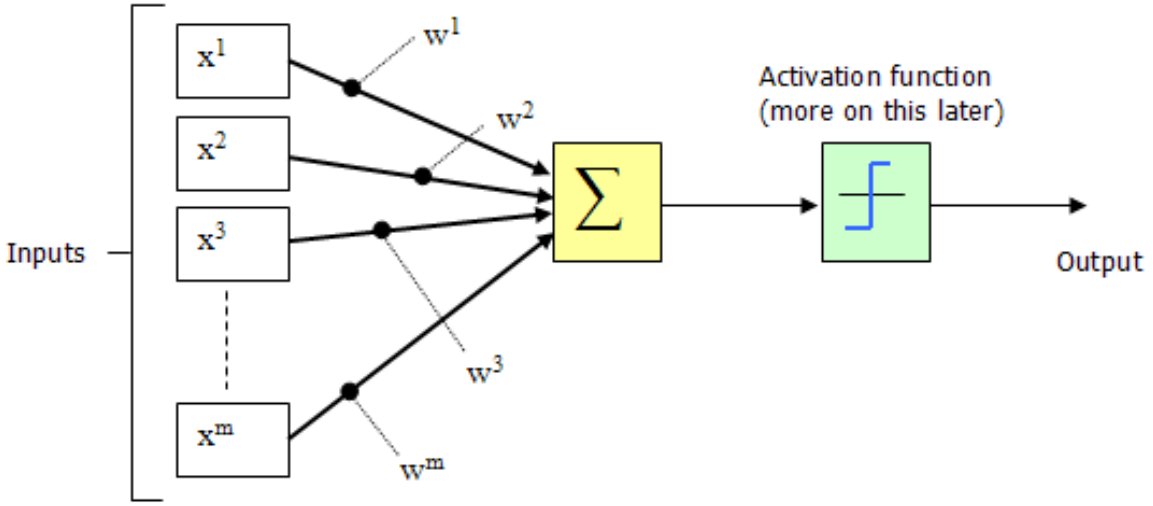
- Synapses
  - The connections between one neuron and another are called synapses.
  - Information always leaves a neuron via its axon, and is then transmitted across a synapse to the receiving neuron.

What does a neuron look like

- Spikes (signals) are important, since other neurons receive them.
- Neurons communicate with spikes.
- The information sent is coded by spikes.

How about artificial neural networks

- Suppose that we have a firing rate at each neuron. Also suppose that a neuron connects with **m** other neurons and so receives m-many inputs “x1, x2, …, xm”



How about artificial neural networks

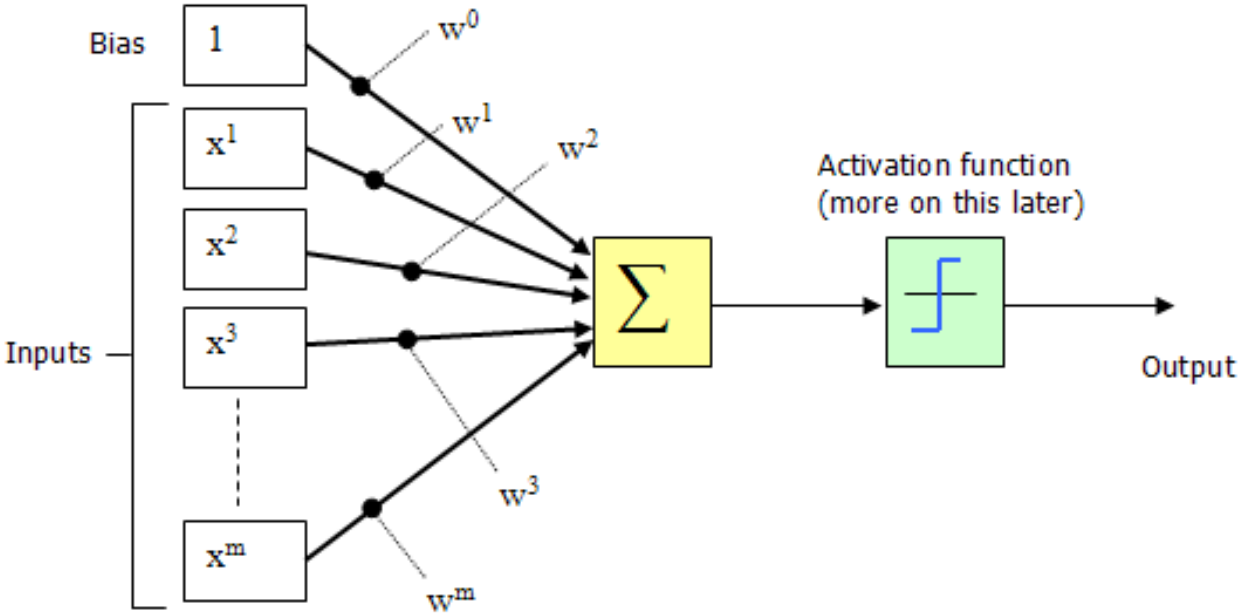
- This configuration is actually called a **Perceptron**. The perceptron was one of the earliest neural network models.
- A perceptron models a neuron by taking a weighted sum of inputs and sending the output 1, if the sum is greater than some adjustable threshold value also called an activation function.

How about artificial neural networks

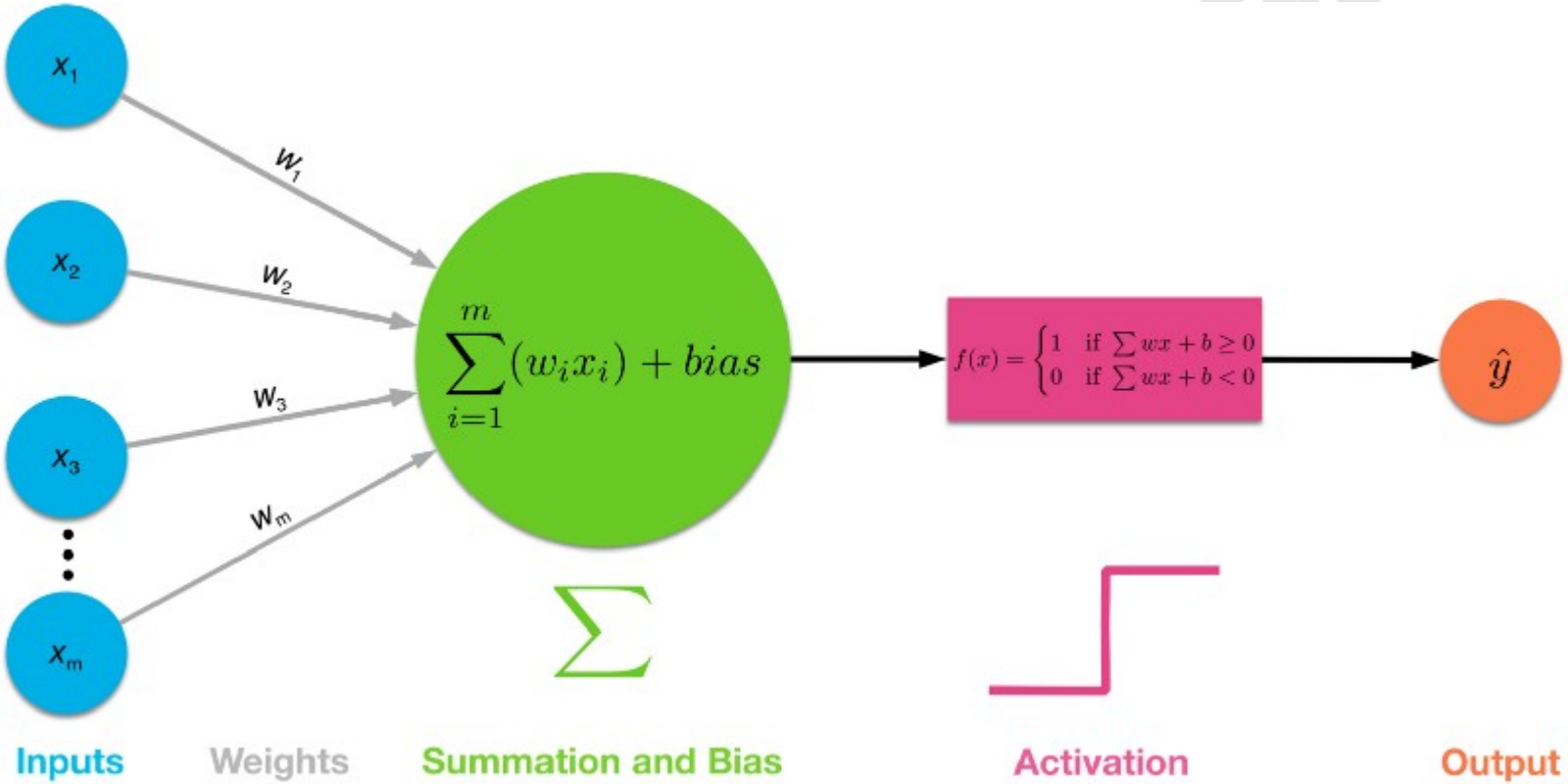
- The input (**x1, x2, …, xm**) and connection weights (**w1, w2, …, wm**) are typically real values, both positive (+) and negative (-).
- If the feature of some **xi** tends to cause the perceptron to fire, the weight **wi** will be positive; if the feature xi inhibits the perceptron, the wight **wi** will be negative.

How about artificial neural networks

- The perceptron itself, consists of weights, the summation processor, and activation function, and an adjustable threshold processor (called **bias**).

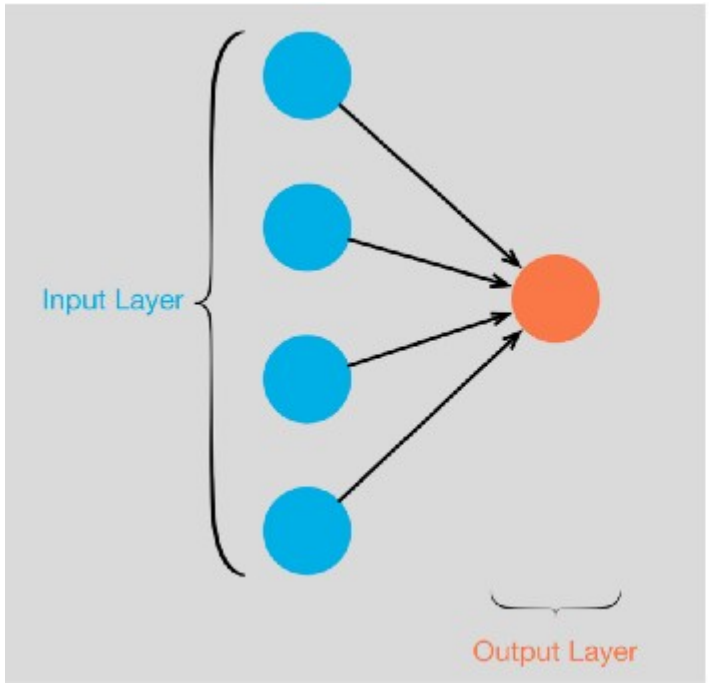


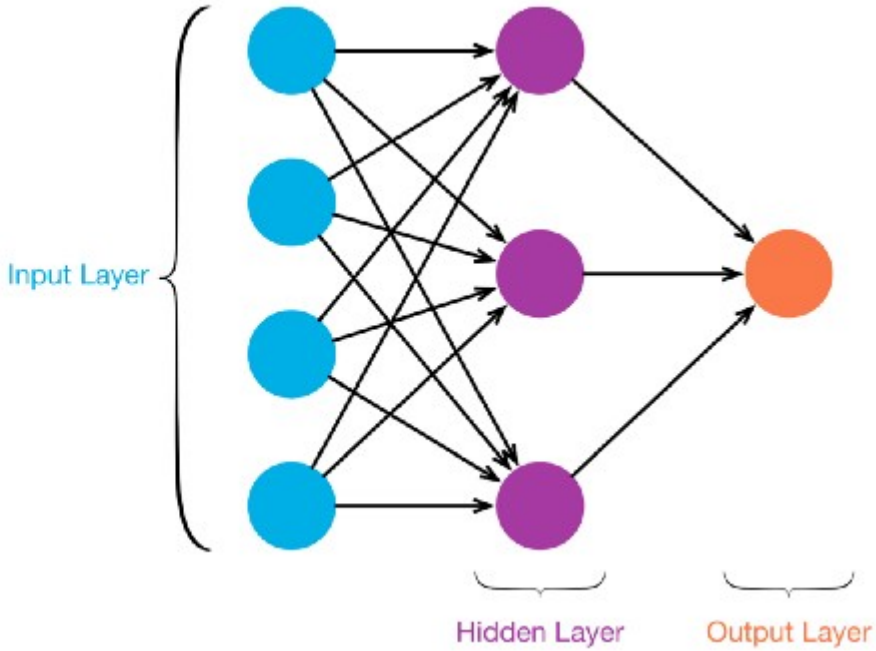
Procedures of a single-layer perceptron network



Cr. https://medium.com/towards-data-science/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Single-layer vs multiple-layer perceptron

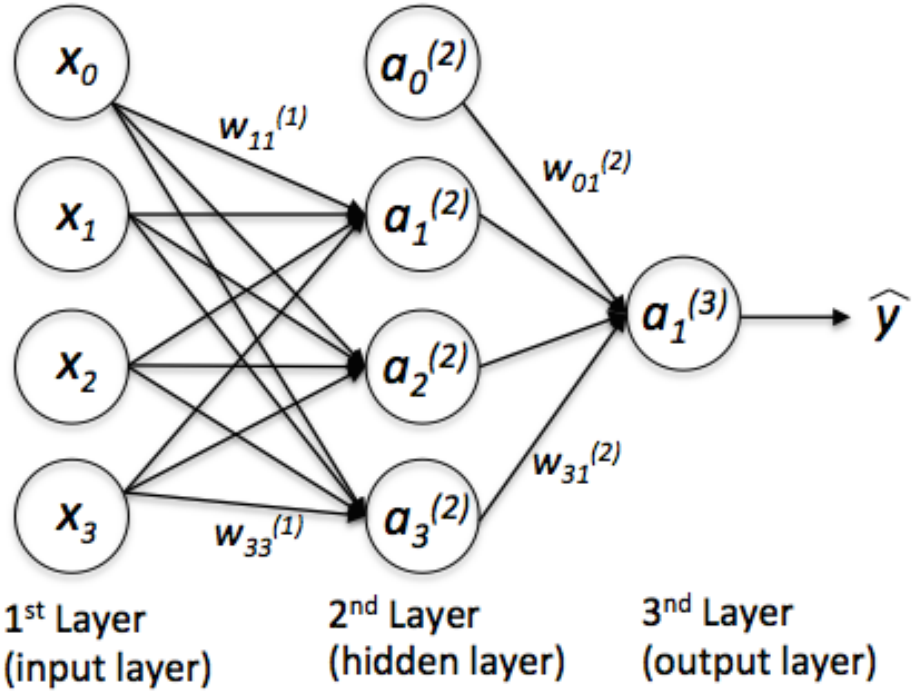




Cr. https://medium.com/towards-data-science/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

How about artificial neural networks

- The architecture of fully connected, feed-forward neural network, looks like this



Cr. https://www.kdnuggets.com/2016/08/role-activation-function-neural-network.html

How about artificial neural networks

- The bias can be thought of as the propensity of the perceptron to fire irrespective of its inputs.

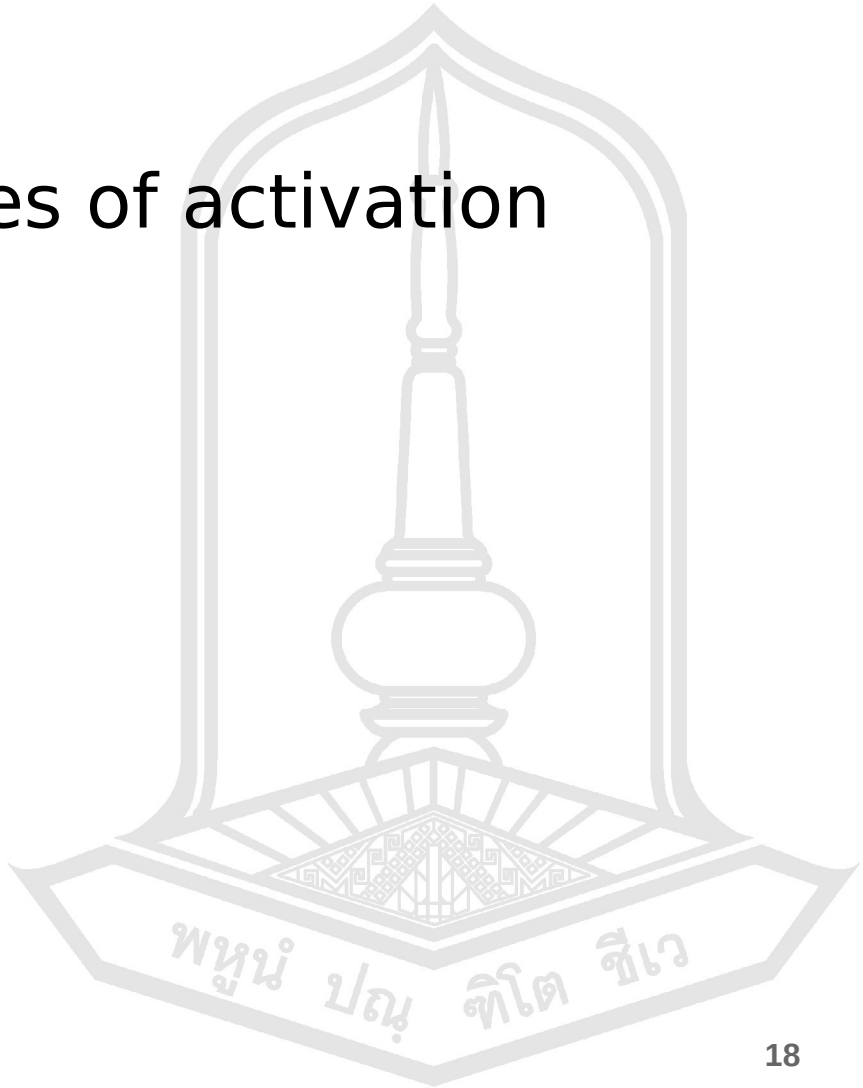
$$\sum_{i=1}^{m} bias + (w^i x^i)$$

$$W \cdot X = w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = \sum_{i=1}^{m} w_i x_i$$

Definition of a Dot Product

How about artificial neural networks

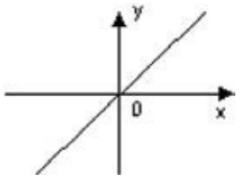
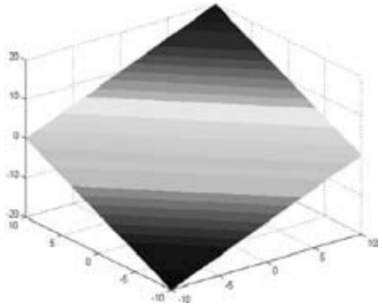
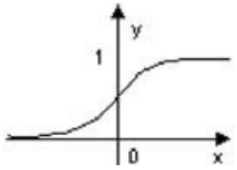
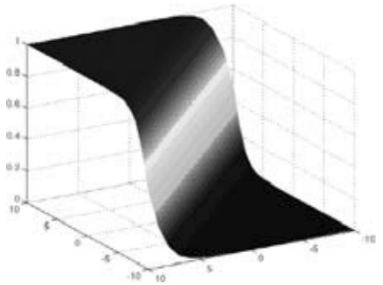
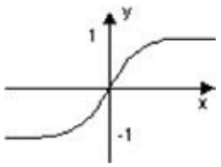
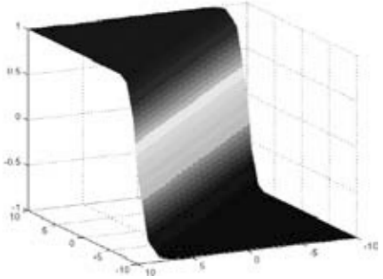
- Activation Function
  - There are many types of activation functions
    - Linear
    - Sigmoid (logistic)
    - Hyperbolic tangent
    - Etc.



Forward propagation

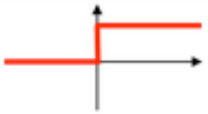
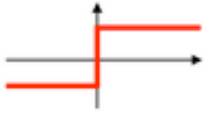
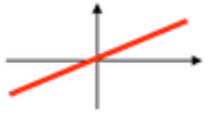
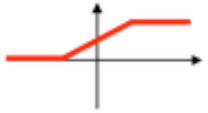
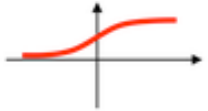
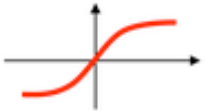
- Because we used a random set of initial weights, the value of the output neuron is off the mark; in this case by +0.77 (since the target is 0).
- If we stopped here, this set of weights would be a great neural network for inaccurately representing the XOR operation.
- We can fix that by using back propagation to adjust the weights to improve the network.

Activation function

Activation Function	Mathematical Equation	2D Graphical Representation	3D Graphical Representation
Linear	$y = x$		
Sigmoid (logistic)	$y = \frac{1}{1 + e^{-x}}$		
Hyperbolic tangent	$y = \frac{1 - e^{-2x}}{1 + e^{2x}}$		

Cr. https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

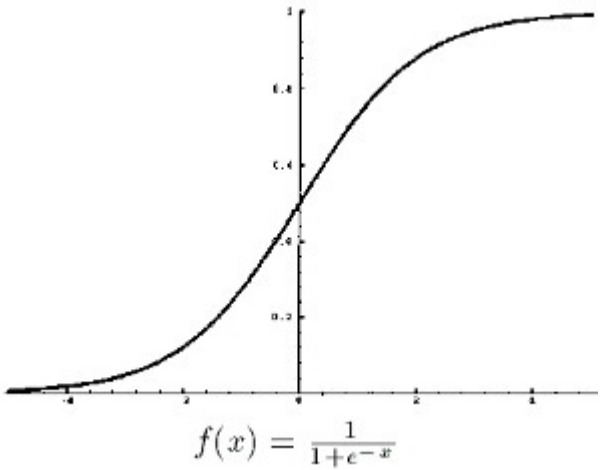
Activation function

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

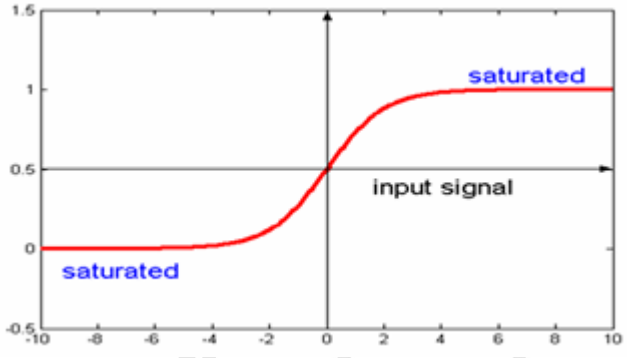
Cr. https://www.kdnuggets.com/2016/08/role-activation-function-neural-network.html

Activation function

- The sigmoid function looks like this, graphically:



$$f(x) = \frac{1}{1+e^{-x}}$$

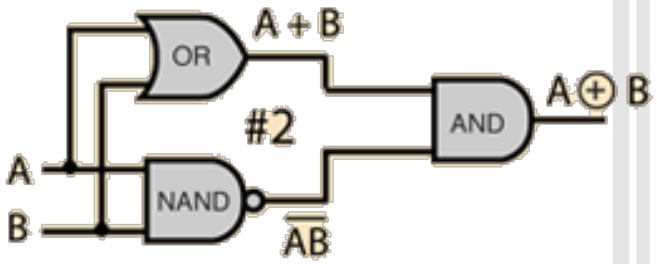


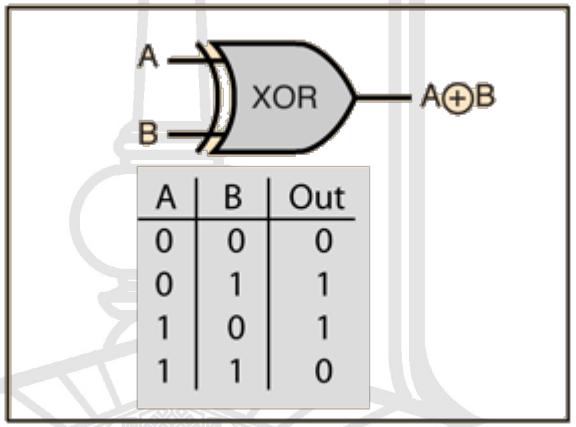
Cr. https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Forward propagation

- The **XOR function** can be represented by the mapping of the below inputs and outputs

input	output
0, 0	0
0, 1	1
1, 0	1
1, 1	0

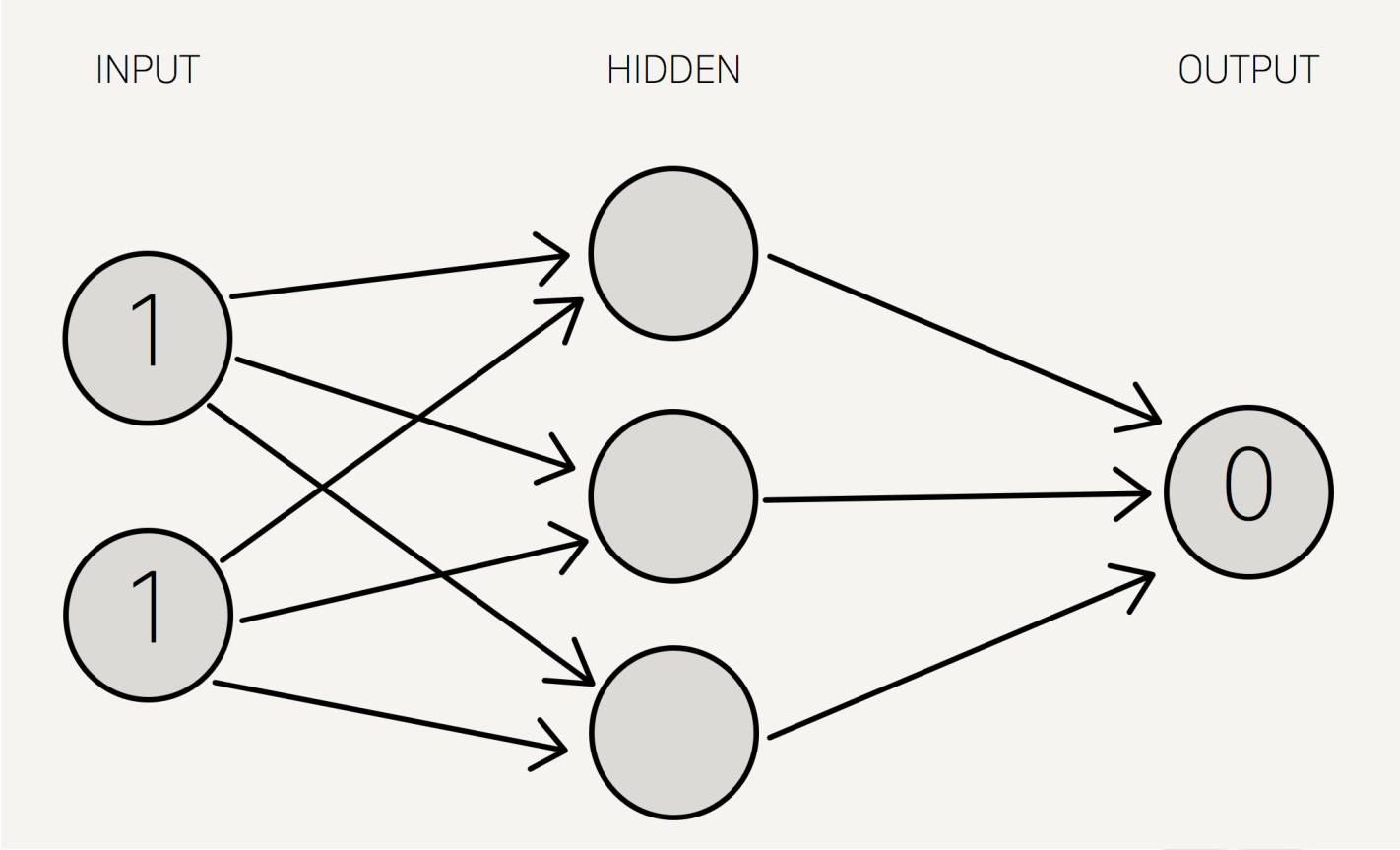




Cr. http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/xor.html

Forward propagation

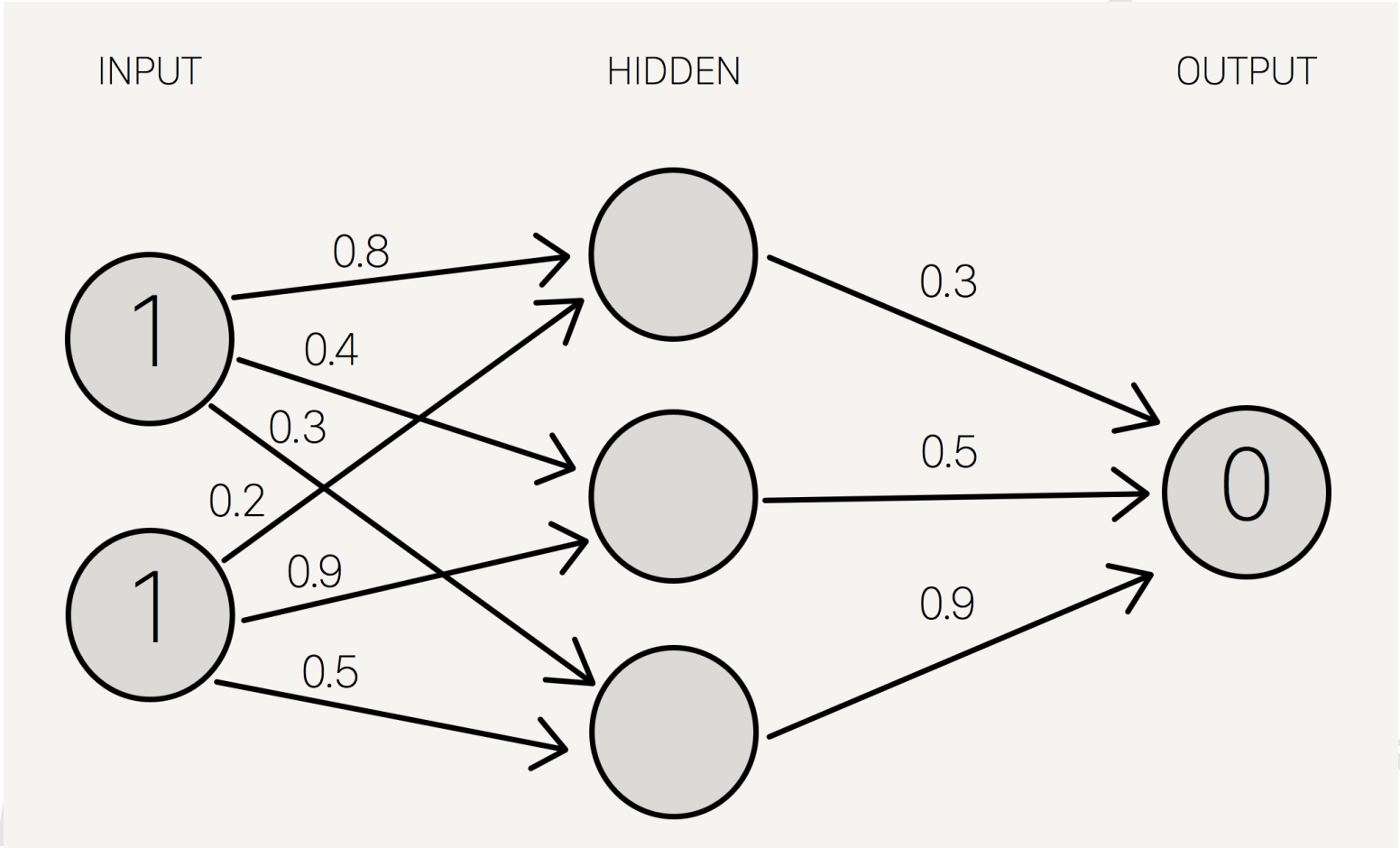
- We use a single hidden layer with only three neurons for this example.



Forward propagation

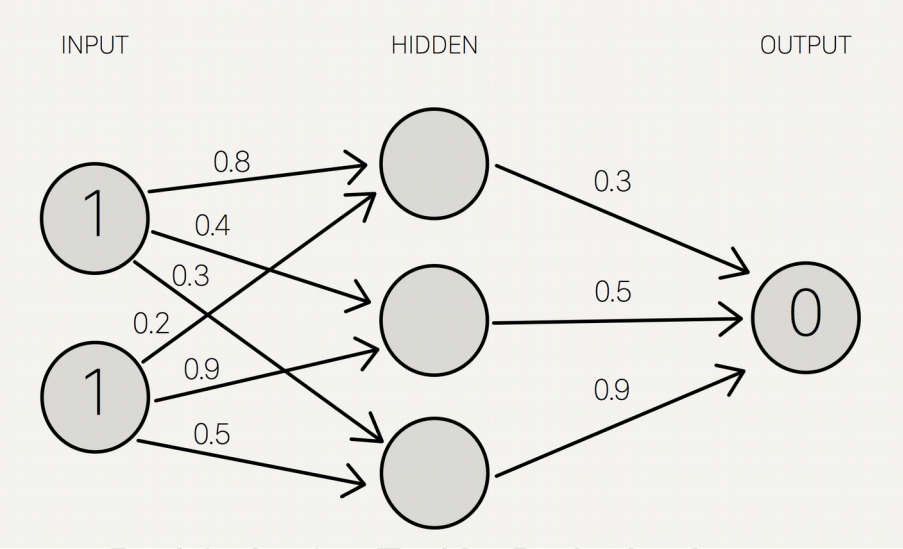
- We assign weights to all of the synapses. These weights are selected randomly (based on Gaussian distribution).
- The *initial weights* will be between 0 and 1.

Forward propagation



Forward propagation

- We sum the product of the inputs with their corresponding set of weights to arrive at the first values of the hidden layer.



Hidden layer

1 * 0.8 + 1 * 0.2 = 1

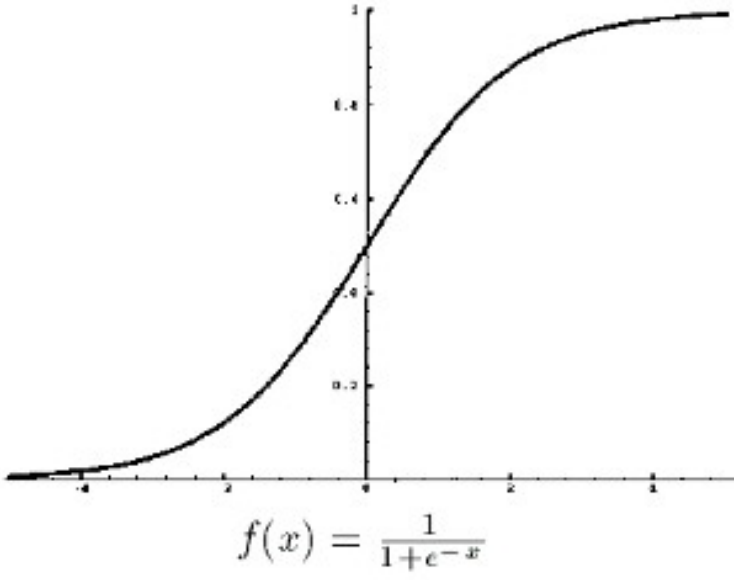
1 * 0.4 + 1 * 0.9 = 1.3

1 * 0.3 + 1 * 0.5 = 0.8

Forward propagation

- To get the final value, we apply the **activation function** to the hidden layer sums.
- The purpose of the activation function is to transform the input signal into an output signal and are necessary for neural networks to model complex non-linear patterns that simpler models might miss.

Forward propagation



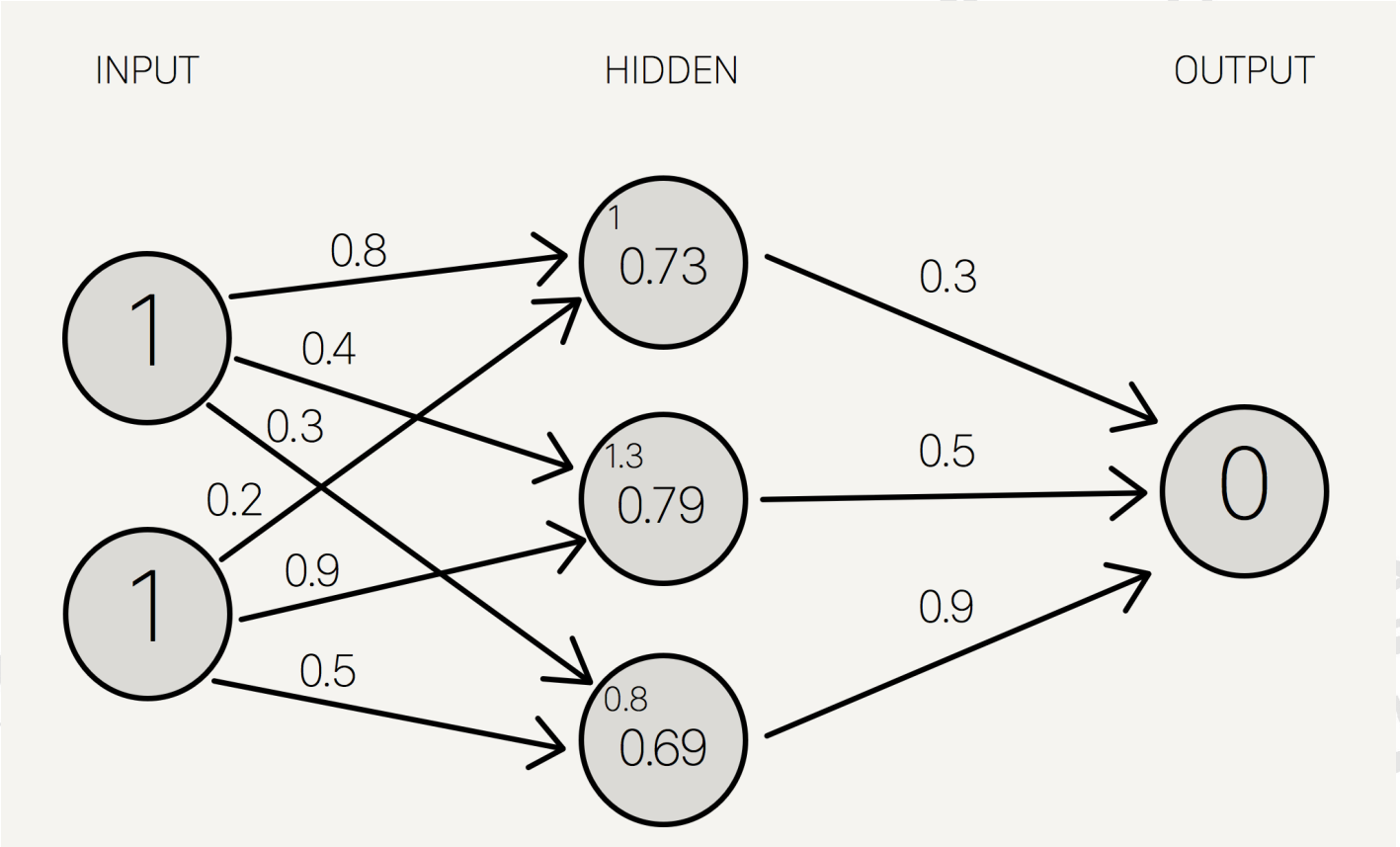
S(1.0) = 0.73105857863
S(1.3) = 0.78583498304
S(0.8) = 0.68997448112

S(1.0) = 1 / (1+(EXP(1)^-1.0))
= 0.73105

e = exponential value, approximately 2.718

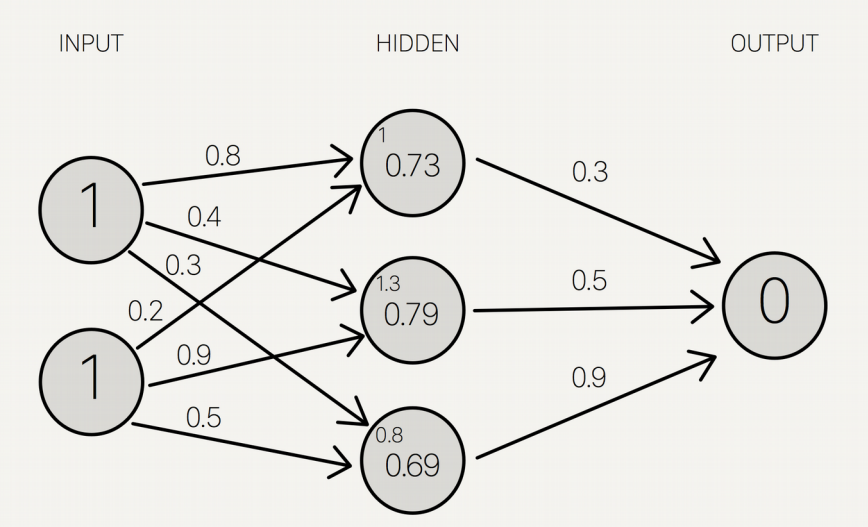
Forward propagation

- We add that to our neural networks as hidden layer results:



Forward propagation

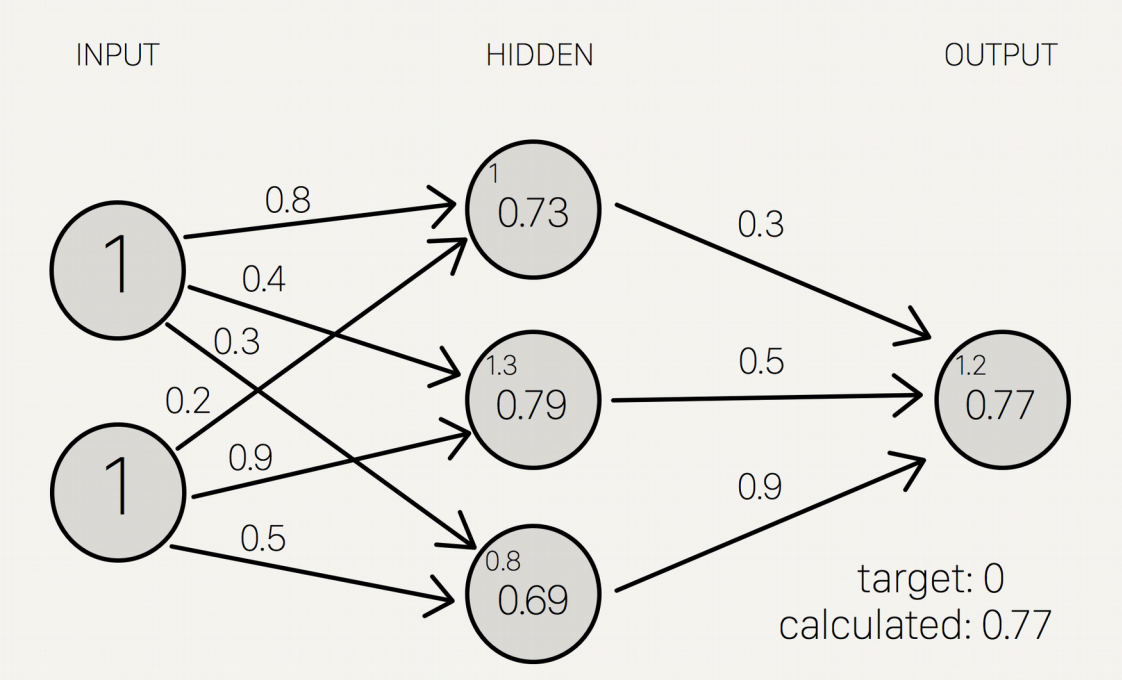
- Then, we sum the *product* of the hidden layer results with the second set of weights to determine the **output sum**.



= (0.73 * 0.3) + (0.79 * 0.5)
+ (0.69 * 0.9)
= **1.235**

Forward propagation

- Finally we apply the **activation function** to get the final output result.



$$f(x) = \frac{1}{1+e^{-x}}$$

S(1.235) = 1 / (1+(EXP(1)^-1.235))
= 0.77469

S(1.235) = 0.77469

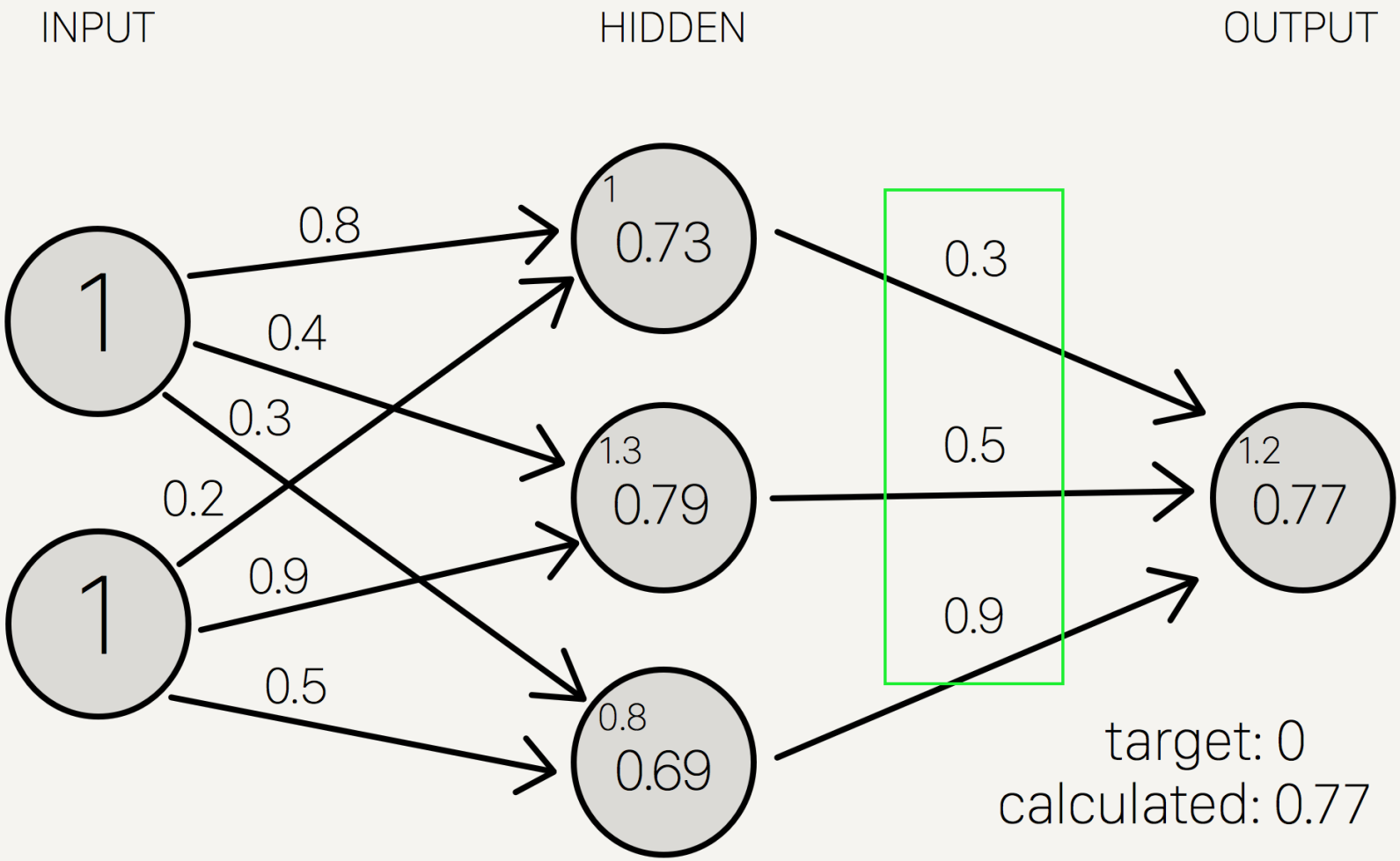
Forward propagation

- Because we used a random set of initial weights, the value of the output neuron is off the mark; in this case by +0.77 (since the target is 0).
- If we stopped here, this set of weights would be a great neural network for inaccurately representing the XOR operation.
- We can fix that by using back propagation to adjust the weights to improve the network.

Back propagation

- To improve the model, we first have to quantify just how wrong our predictions are. Then, we ***adjust the weights*** accordingly so that the margin of errors are decreased.
- Similar to forward propagation, back propagation calculations occur at each "*layer*".
- We begin by changing the weights between the hidden layer and the output layer.

Back propagation



Back propagation

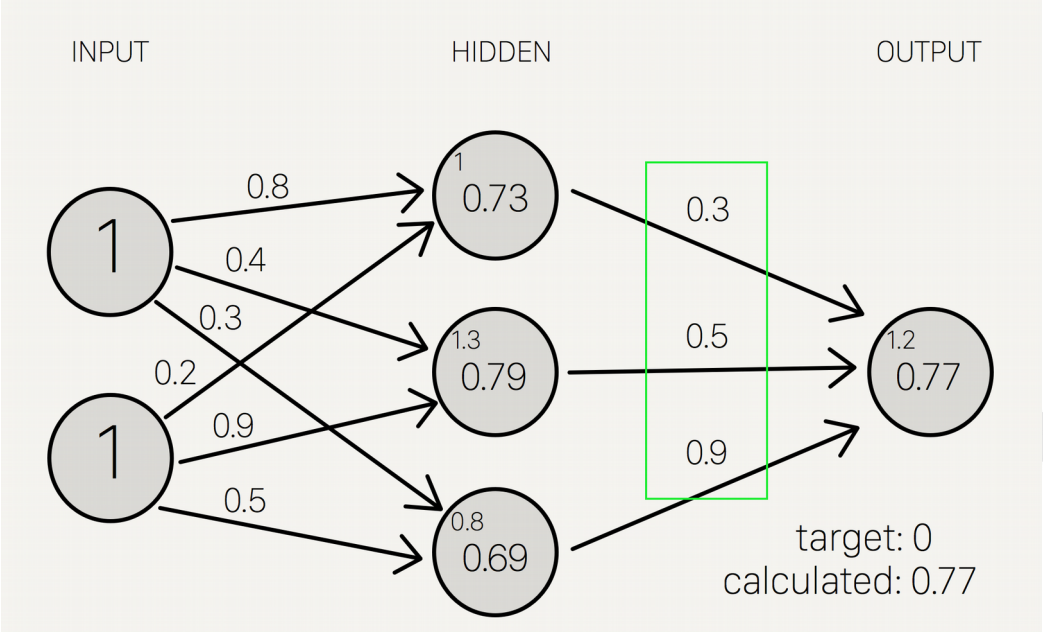
- Calculating the incremental change to these weights happens in two steps:
  - 1. find the margin of error of the output result (what we get after applying the activation function) to back out the necessary change in the output sum (we call this ***delta output sum***)
  - 2. extract the change in weights by multiplying ***delta output sum*** by the hidden layer results.

delta = Δ

Back propagation

- The **output sum margin of error** is the target output result minus the calculated output result:

$$\text{Output sum margin of error} = target - calculated$$



Target = 0
Calculated = 0.77
Target - calculated = -0.77

Back propagation

- To calculate the necessary change in the output sum, or **delta output sum (Δsum)**, we take the derivate of the activation function and apply it to the output sum. In this example, the activation function is the sigmoid function.

Back propagation

- The sigmoid function takes the sum and returns the result:

$$S(sum) = result$$

- So, the ***derivative of sigmoid***, also known as *sigmoid prime*, will give the rate of change (or "slope") of the activation function at the output sum:

*derivative of sigmoid = x * (1-x)*

where *x* is *S(sum)*

Back propagation

- Since the output **sum margin of error** is the difference in the result, we simply multiply that with the rate of change to give the **delta output sum (Δsum)**

*Δsum = derivative of sigmoid * (target result – calculated result)*

Back propagation

*Δsum = derivative of sigmoid * (target result – calculated result)*

= 0 – 0.77
= -0.77

*derivative of sigmoid = x * (1-x)*

where *x* is *S(sum)*

*= s(1.235) * (1 – s(1.235))*
*= 0.7746 * (1-0.7746)*
= 0.174544

$$f(x) = \frac{1}{1+e^{-x}}$$

S(1.235) = 1 / (1+(exp(1)^-1.235))
= 0.77469

*Δsum = 0.174544 * -0.77*
*= **-0.13439891***

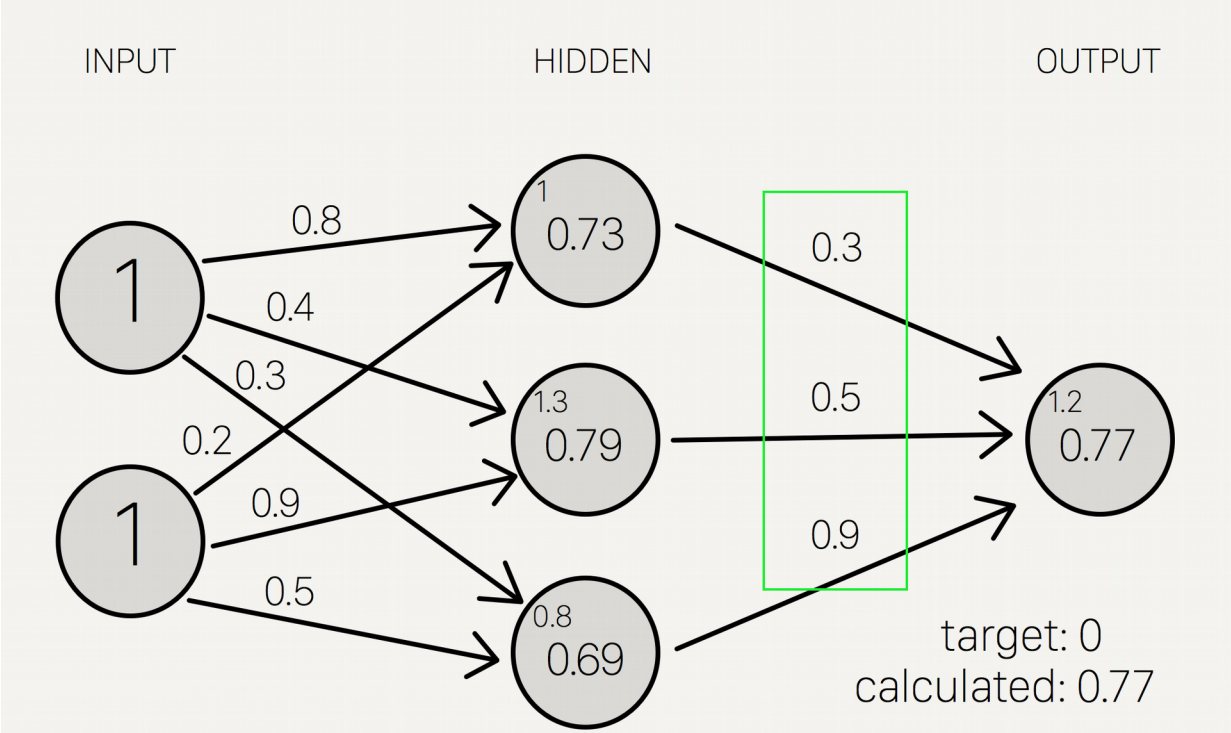
derivative of sigmoid = **S'**

Back propagation

- Now we have the proposed change in the output layer sum (-0.13), let's use this in the derivative of the output sum function to determine the **new change in weights**.
- The formulas used to modify the weight ($w_{j,k}$) between the output node (k) and the node (j)

$$W_{j,k} = W_{j,k} + \Delta W_{j,k}$$

Back propagation



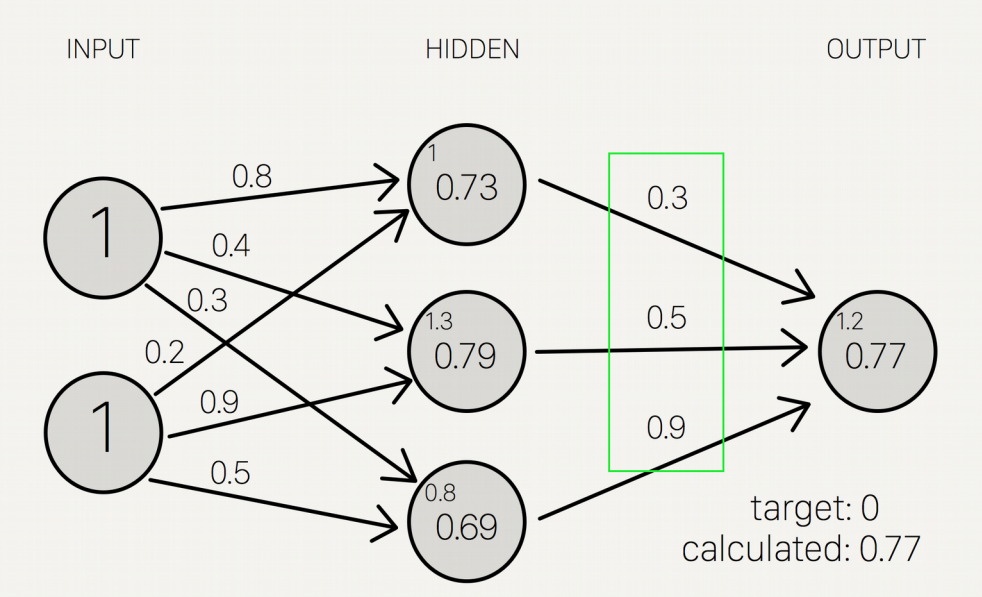
Δweights (Δ*w*) = delta output sum (Δ*sum*) / hidden layer results

Δ*w* = -0.1344 / [0.73, 0.79, 0.69]

Δ*w* = [-0.1838, -0.1710, -0.1920]

Back propagation

$$w_{j,k} = w_{j,k} + \Delta w_{j,k}$$



old w7 = 0.3
old w8 = 0.5
old w9 = 0.9

Δw = [-0.1838, -0.1710, -0.1920]

new w7 = 0.3+(-0.18) = 0.116
new w8 = 0.5+(-0.17) = 0.329
new w9 = 0.9+(-0.19) = 0.708

Back propagation

- Then we can calculate **the delta hidden sum**:

*Δhidden sum = Δsum / hidden-to-outer weights * S' (hidden sum)*
*= -0.134 / [0.3, 0.5, 0.9] * S'([1, 1.3, 0.8])*
*= [-0.448, -0.2688, -0.1493] * [0.1966, 0.1683, 0.2139]*
= [-0.088, -0.0452, -0.0319]

*derivative of sigmoid = x * (1-x)*

*= s(1) * (1 – s(1))*
*= 0.7310 * (1-0.7310)*
*= **0.1966***

$$f(x) = \frac{1}{1+e^{-x}}$$

S(1) = 1 / (1+(exp(1)^-1))
= 0.7310

Back propagation

- Once we get the ***Δhidden sum***, we calculate the change in weights between the input and hidden layer by dividing it with the input data (1,1)

Back propagation

input 1 = 1
input 2 = 1

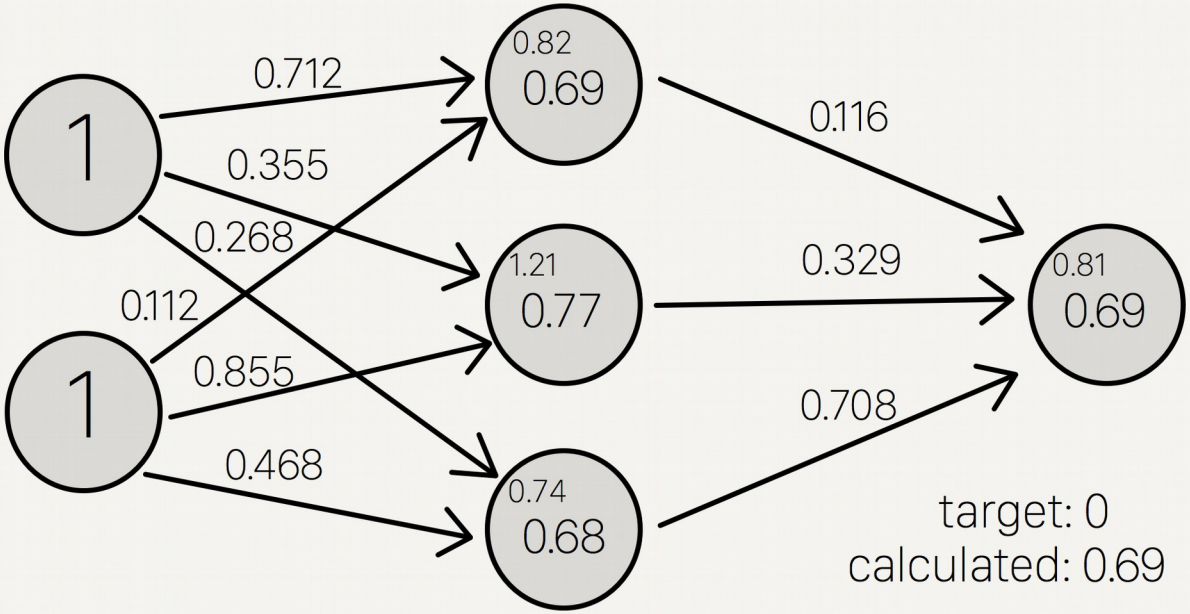
Δweights = Δhidden sum / input data

Δweights = [-0.088, -0.0452, -0.0319] / [1, 1]

Δweights = [-0.088, -0.0452, -0.0319], [-0.088, -0.0452, -0.0319]

old w1 = 0.8	new w1 = 0.712
old w2 = 0.4	new w2 = 0.3548
old w3 = 0.3	new w3 = 0.2681
old w4 = 0.2	new w4 = 0.112
old w5 = 0.9	new w5 = 0.8548
old w6 = 0.5	new w6 = 0.4681

Back propagation

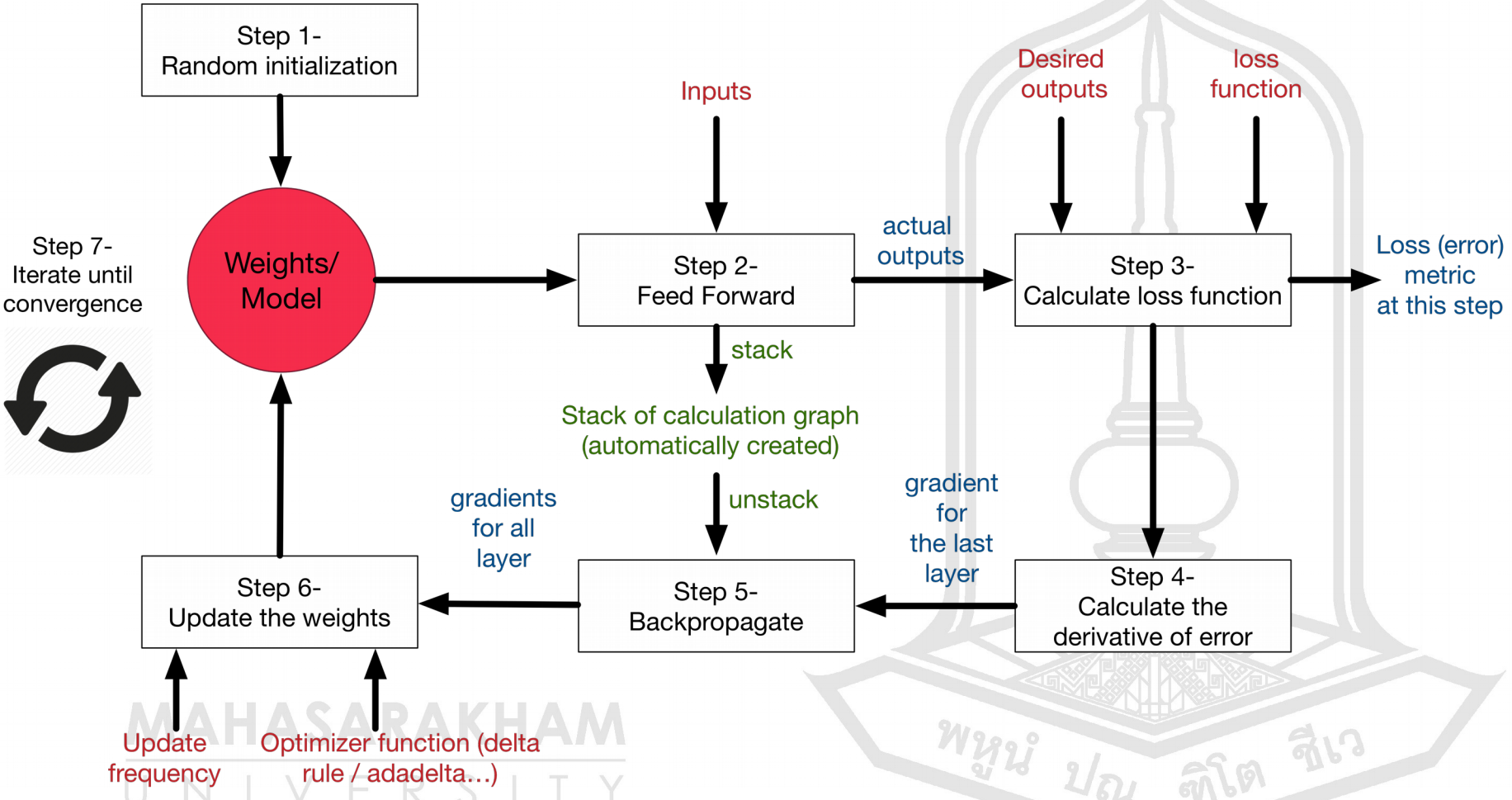


old	new
w1: 0.8	w1: 0.712
w2: 0.4	w2: 0.3548
w3: 0.3	w3: 0.2681
w4: 0.2	w4: 0.112
w5: 0.9	w5: 0.8548
w6: 0.5	w6: 0.4681
w7: 0.3	w7: 0.1162
w8: 0.5	w8: 0.329
w9: 0.9	w9: 0.708

Back propagation

- Once we arrive at the adjusted weights, we start again with forward propagation.
- When training a neural network, it is common to repeat both these processes thousands of times.

The learning process on neural networks



Cr. http://datathings.com/blog/post/neuralnet/

References

- https://www.codeproject.com/Articles/16419/AI-Neural-Network-for-beginners-Part-of
- http://natureofcode.com/book/chapter-10-neural-networks/
- https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/