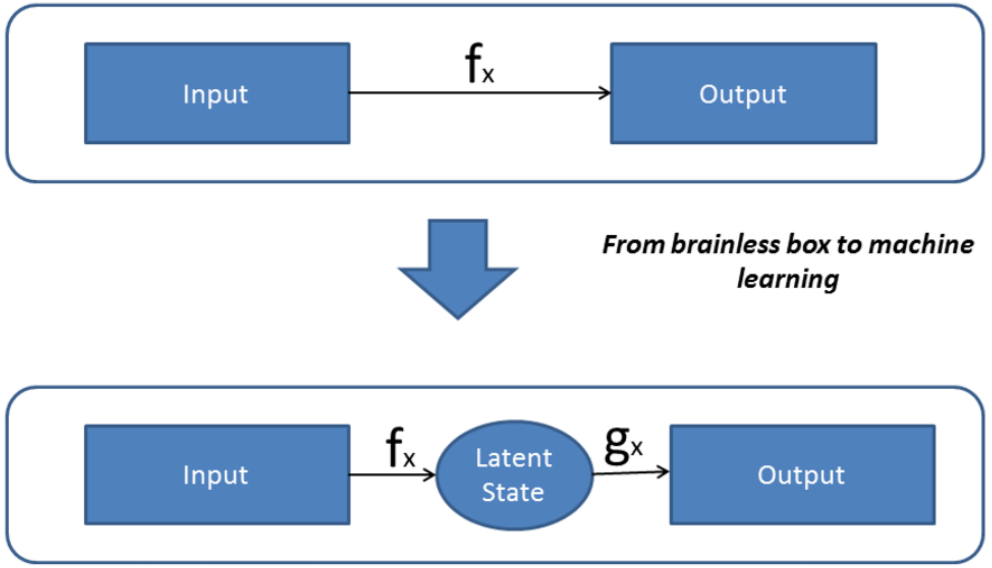
Deep Learning with Python

Copyright © www.ITbodhi.com

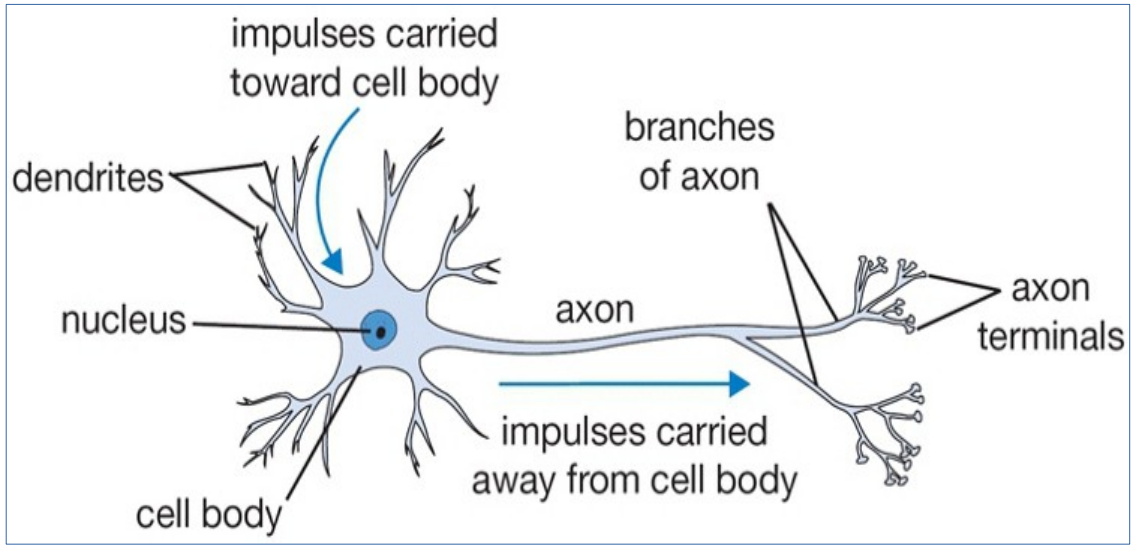
Brain vs Machine

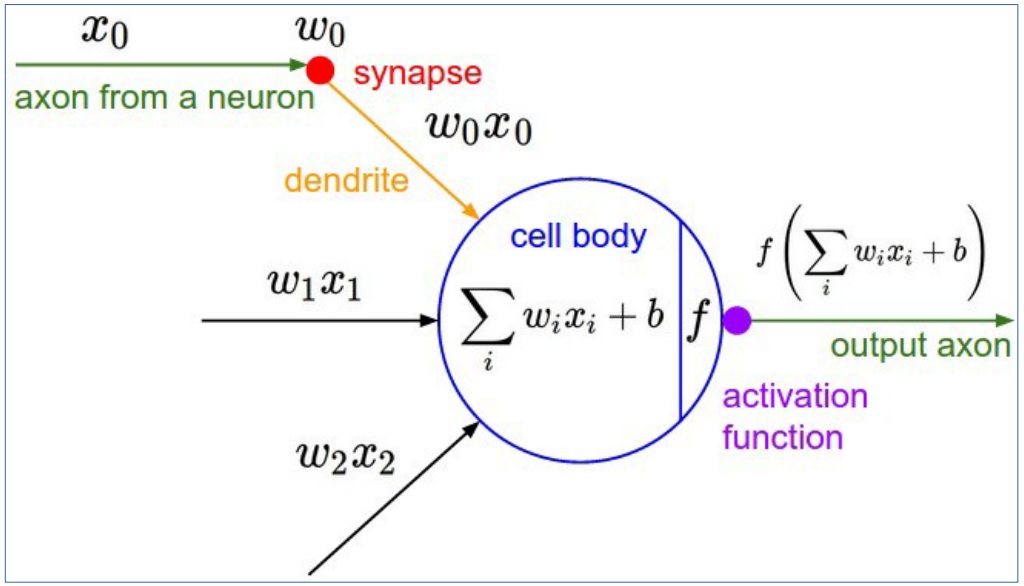
- A simple machine is a set of algorithm, which converts input(s) to output(s). In this scenario, the same input will always lead to the same output.
- Human brain, on the other hand, has a unique characteristic of creating transient states through neurons in between the sensory organs and the brain (decision taking unit). Hence, the probabilistic interim state brings out a factor of randomness, which brings out what we call “Creativity”.
- There are problems that are incredibly simple for a computer to solve, but difficult for humans. Taking the square-root of 987543256. There are, on the other hand, problems that are incredibly simple for you or me to solve, but not so easy for a computer. Show any toddler a picture of a kitten or puppy and they’ll be able to tell you very quickly which one is which. But for a machine, its very very difficult.



Copyright © www.ITbodhi.com

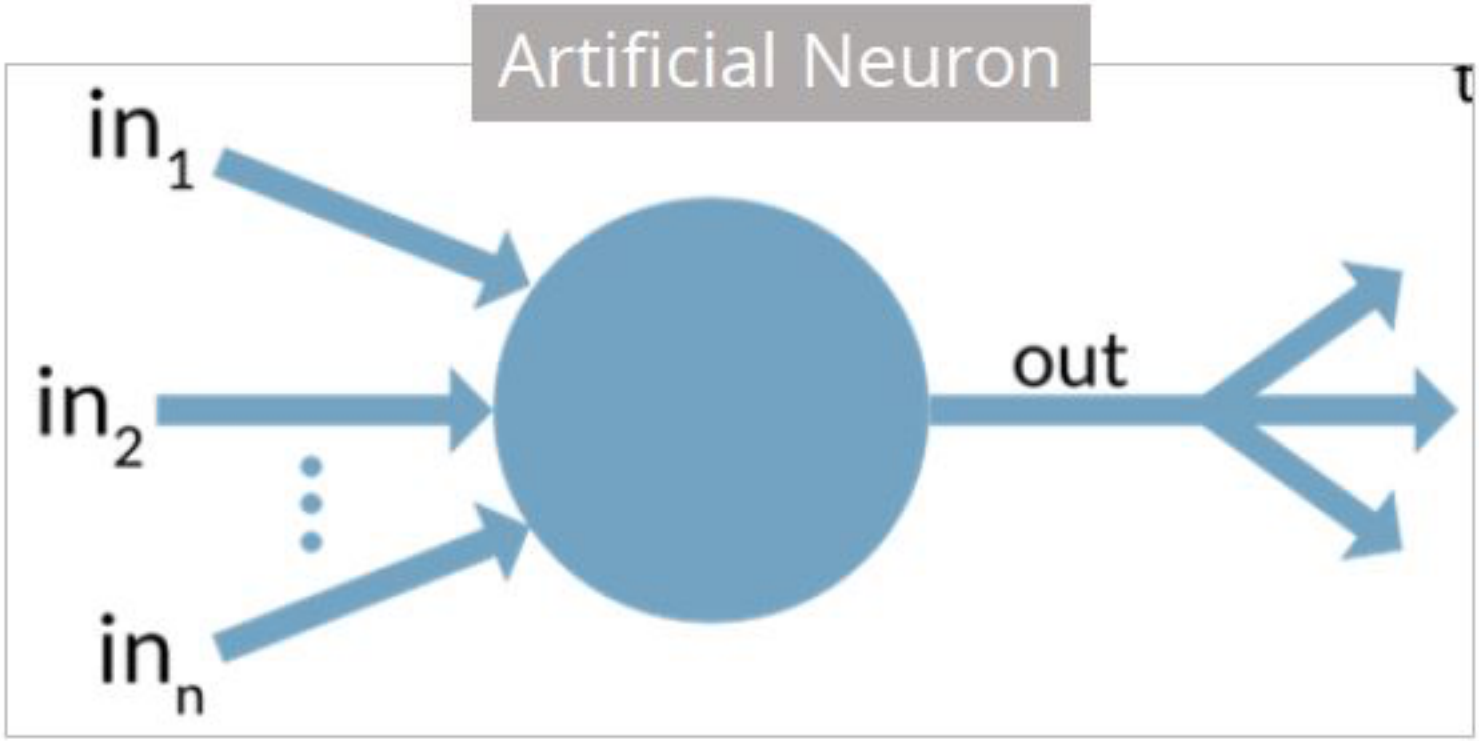
From Natural to Artificial





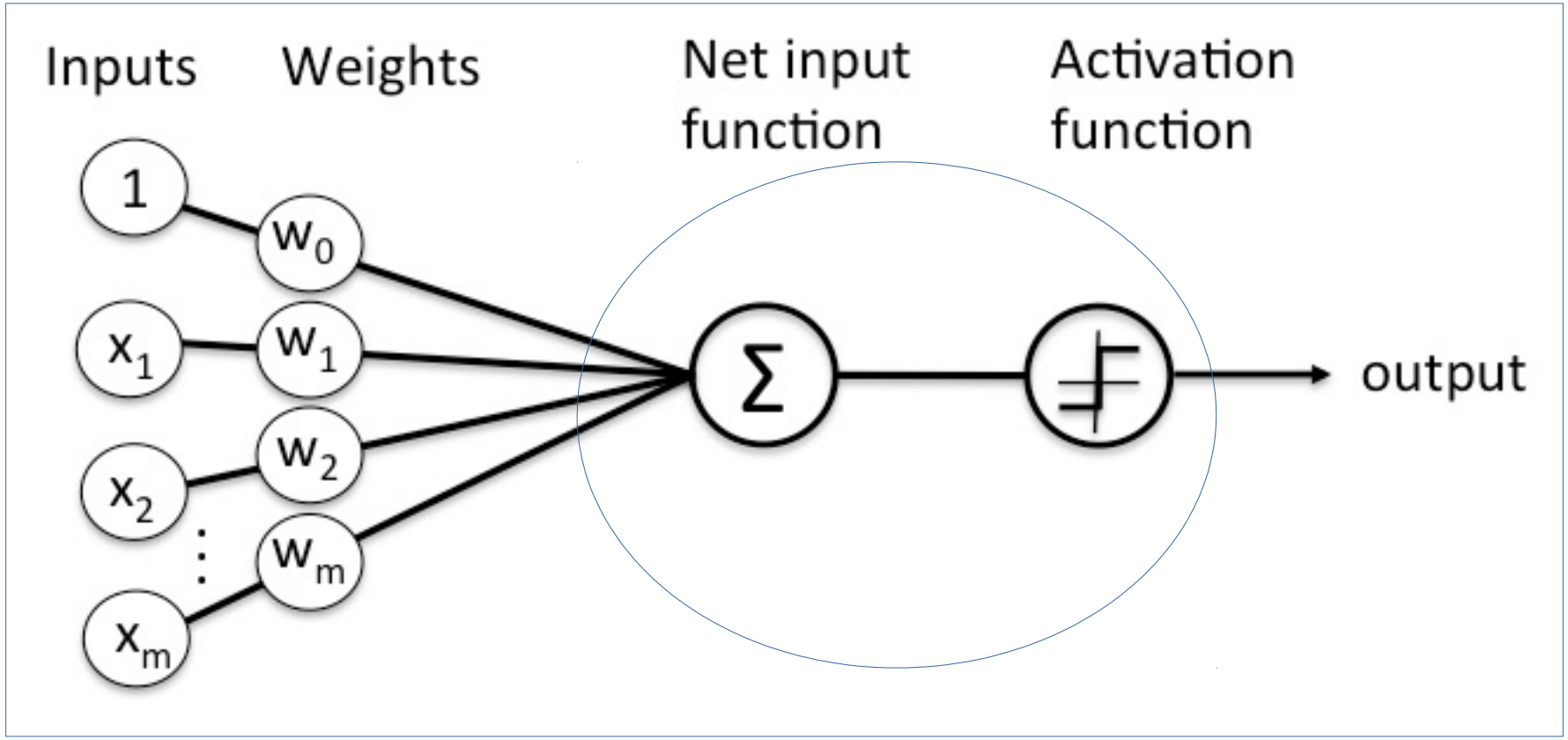
Copyright © www.ITbodhi.com

Perceptron : An Artificial Neuron



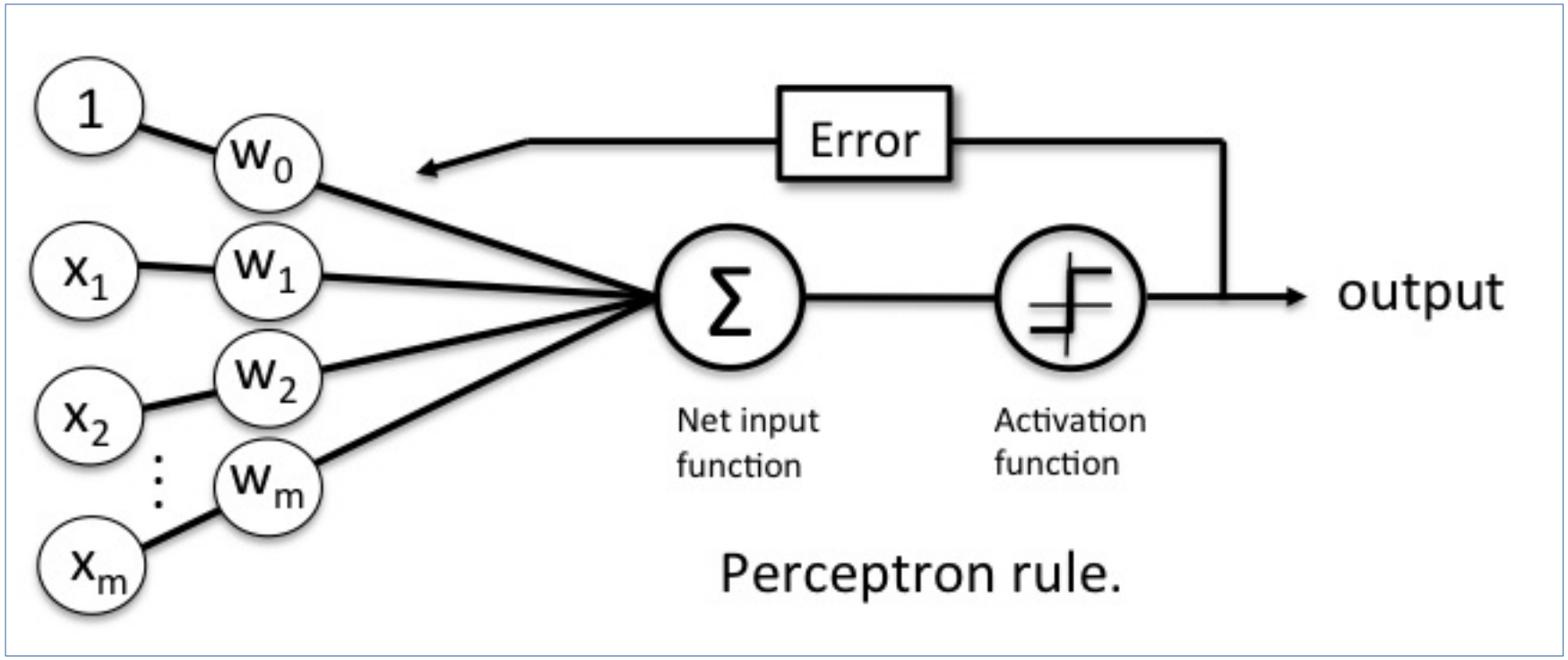
Copyright © www.ITbodhi.com

Perceptron : An Artificial Neuron



Copyright © www.ITbodhi.com

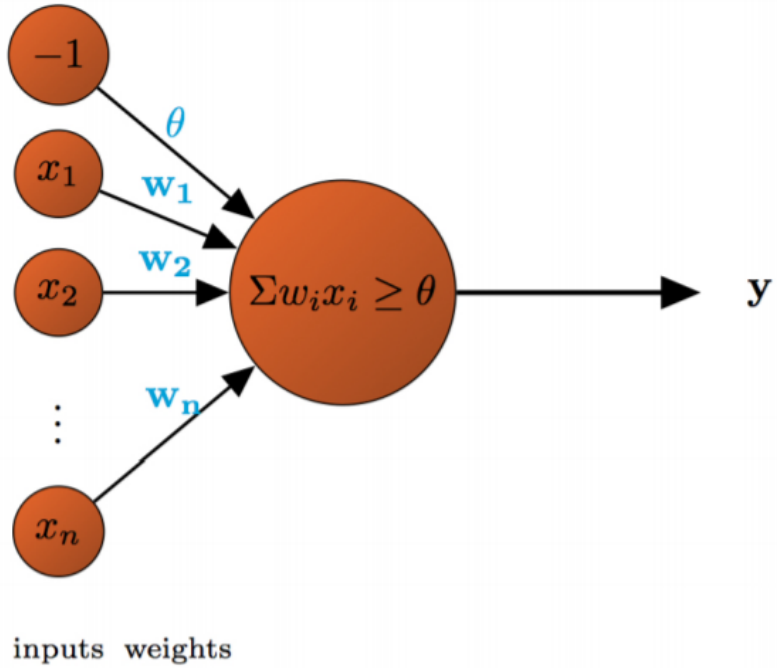
Perceptron : Learning Rule



Copyright © www.ITbodhi.com

Perceptron

- Perceptron is a prototypical example of an Artificial Neural Network.
- Its the simplest neural network possible: a computational model of a single neuron.
- A perceptron follows the “feed-forward” model, meaning inputs are sent into the neuron, are processed, and result in an output
- Think of perceptron as a hyperplane in n dimensions perpendicular to the vector (w1,w2…..wn). It classifies things on one side of the plane as positive and things on the other side as negative that separates data of 2 categories. (not necessarily the best hyperplane)
- The perceptron is said to use “Online Learning” which is a term used to denote techniques where the training data is consumed one point at a time
- Mathematically, the perceptron computes output according to the below rule where theta is a firing threshold.



$$\hat{y} = \chi\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

Copyright © www.ITbodhi.com

Perceptron

- The weights and inputs can be written as a dot product
- The firing threshold can be written as bias (b). It can be thought of as how much flexible the perceptron is. *It allows us to move the line up and down to fit the prediction with the data better.*
- A simple perceptron rule can be written as:

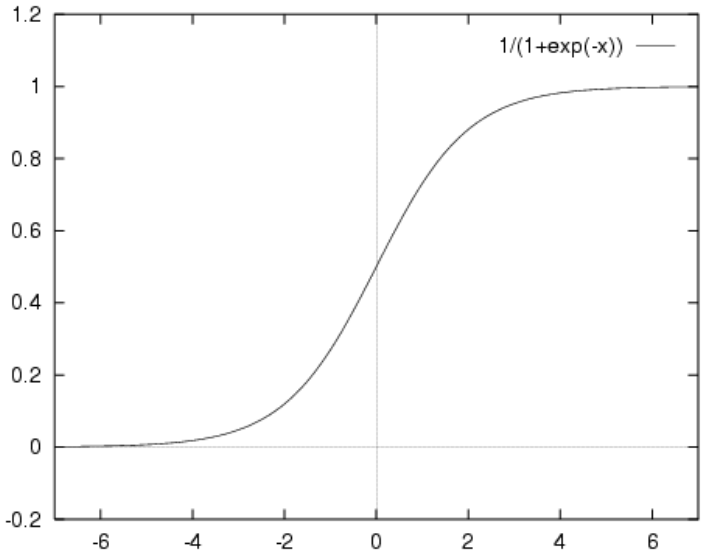
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

- L*earning algorithms* can automatically tune the weights and biases of a network of artificial neurons.
- A small in weight(or bias) will produce a small change in output. This property will make learning possible
- The main thing that ch $\Delta \text{output} \approx \sum_j \frac{\partial \,\text{output}}{\partial w_j} \Delta w_j + \frac{\partial \,\text{output}}{\partial b} \Delta b,$ vation function is that the particular values for the partial derivatives chan

Copyright © www.ITbodhi.com

Activation Functions

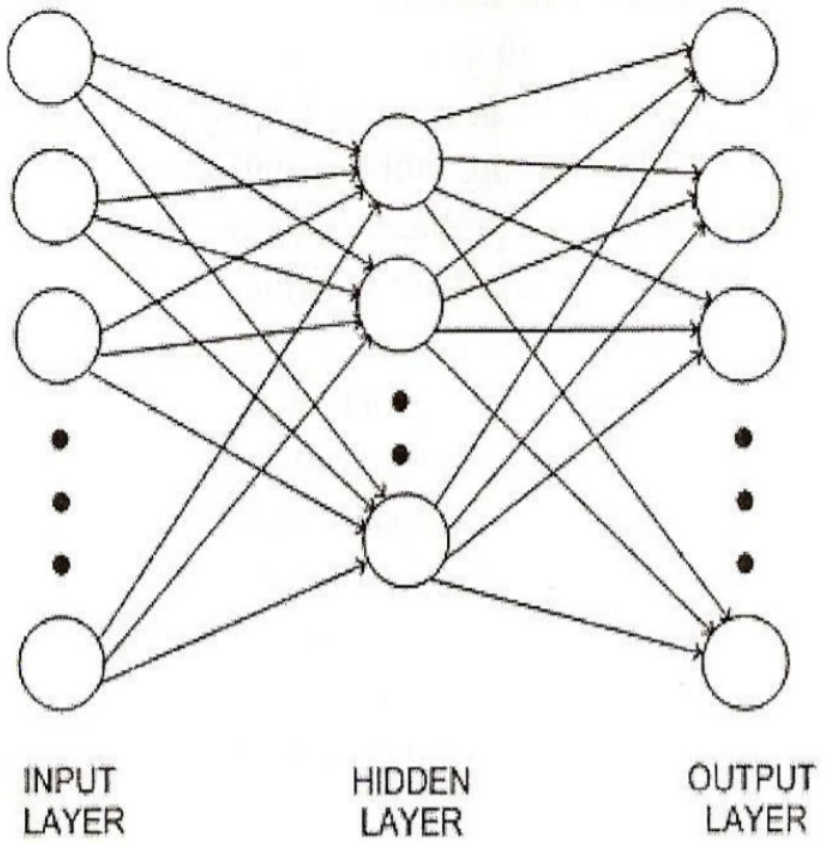
- The main purpose of an Activation Function is to convert a input signal of a node in a A-NN model to an output signal. That output signal now is used as a input in the next layer in the stack.
- *They introduce non-linear properties to our Network*
- If we do not apply a Activation function then the output signal would simply be a simple *linear function.* Now, a linear equation is easy to solve but they are limited in their complexity and have less power to learn complex functional mappings from data. A Neural Network without Activation function would simply be a **Linear regression Model,** which has limited power and does not performs good most of the times.
- Sigmoid is the most famous Activation Function
- Other activation functions are: Tanh, ReLU



Copyright © www.ITbodhi.com

Artificial Neural Networks

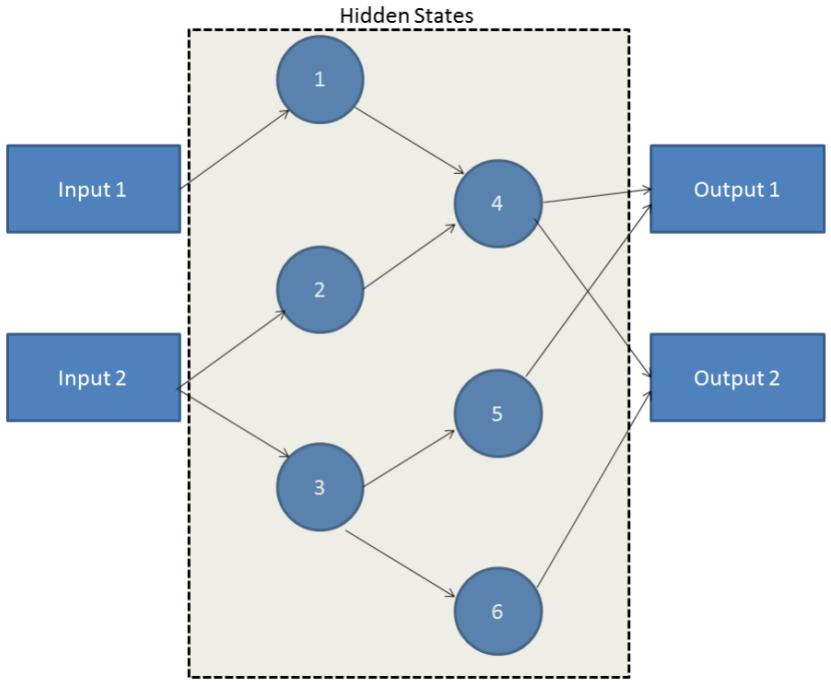
- Neural Network is a learning structure designed to mimic the function of a web of biological neurons.
- A biological neuron takes an electric input and pops out another electrical signal. However, there is a threshold that must be reached before an output is produced.
- The basic idea of the neuron model in ANNs is that the inputs to a neuron are combined (as a weighted sum) into a single value, Then, an activation function, is applied to determine whether or not the neuron fires.
- Deep Learning can be understood as an algorithm which is composed of hidden layers of multiple neural networks
- A function that takes an input signal and generates an output signal but takes into account some kind of threshold is called an **Activation function.**
- The computational systems we write are procedural; a program starts at the first line of code, executes it, and goes on to the next, following instructions in a linear fashion. A true neural network does not follow a linear path. Rather, information is processed collectively, in parallel throughout a network of nodes (the nodes, in this case, being neurons)



Copyright © www.ITbodhi.com

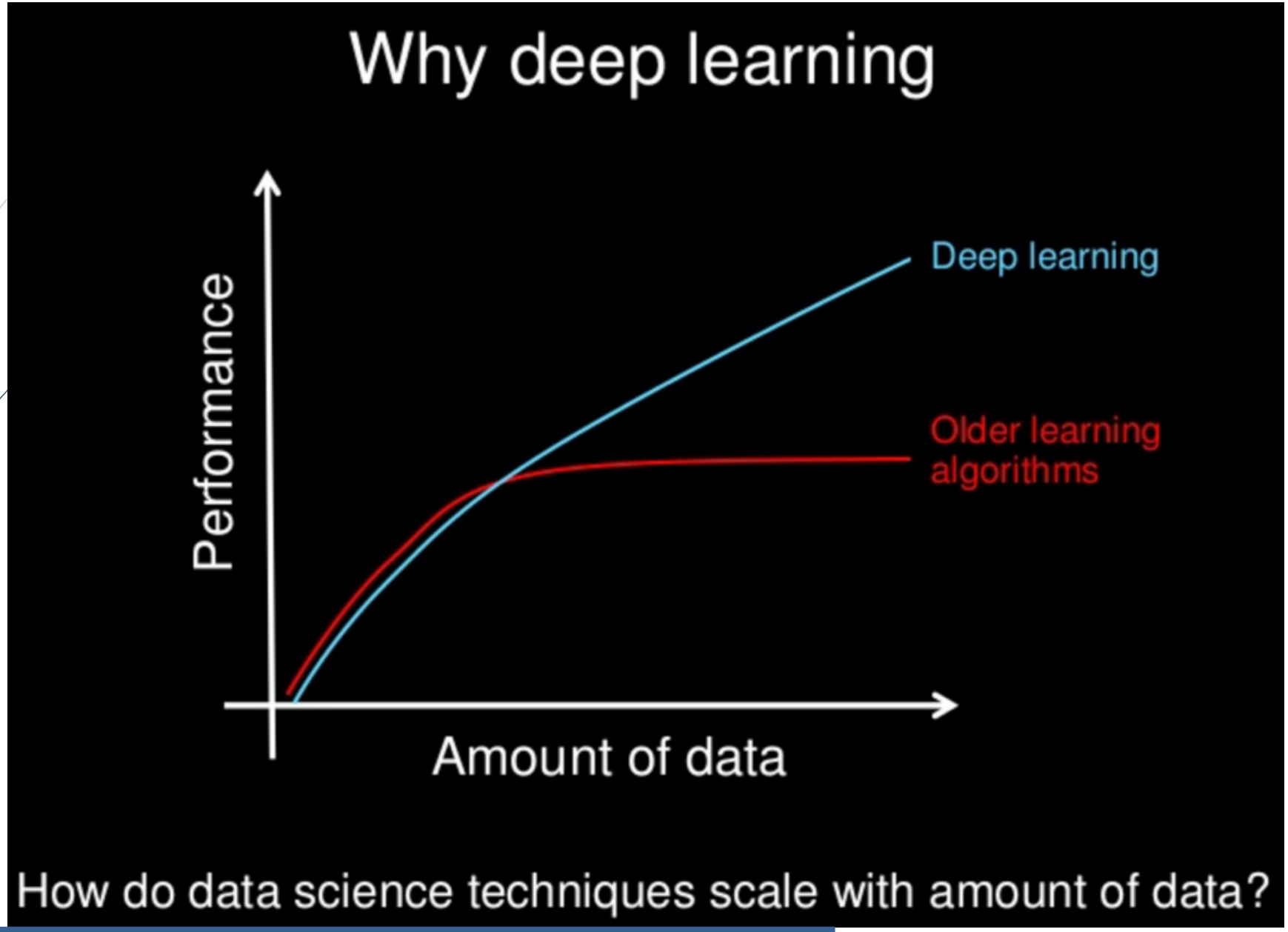
Artificial Neural Networks

- Each of these hidden state is a transient form which has a probabilistic behavior. A grid of such hidden state act as a bridge between the input and the output.
- We have a vector of three inputs and we intend to find the probability that the output event will fall into class 1 or class 2. For this prediction we need to predict a series of hidden classes in between (the bridge). The vector of the three inputs in some combination predicts the probability of activation of hidden nodes from 1 – 3.
- The probabilistic combination of hidden state 1-3 are then used to predict the activation rate of hidden nodes 4-6, which finally predicts the outcome. The intermediate latent states allows the algorithm to learn from every prediction.



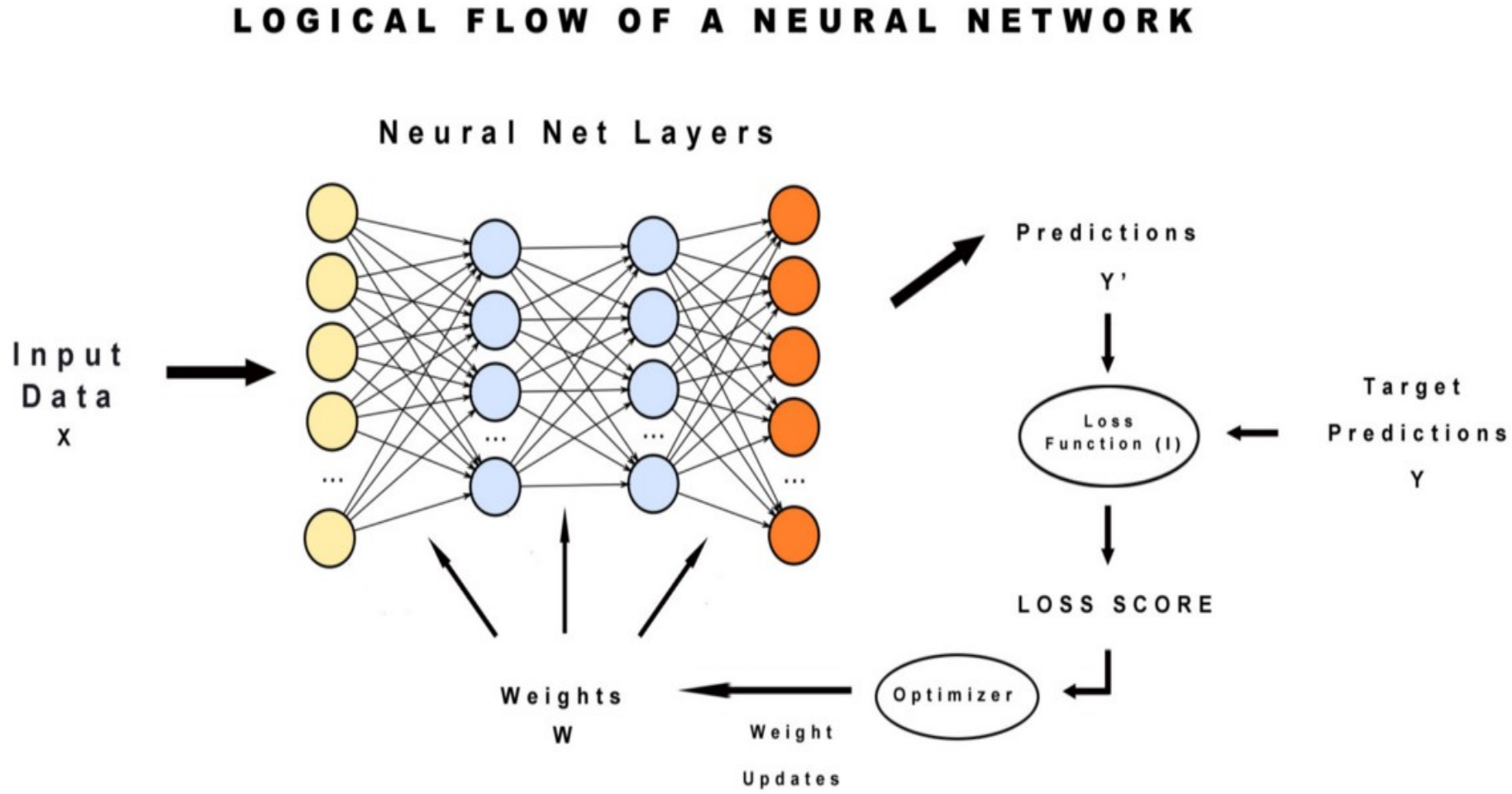
Copyright © www.ITbodhi.com

Why Deep Learning?



Copyright © www.ITbodhi.com

Logical Flow of NN



Copyright © www.ITbodhi.com

NN Characteristics

The main characteristics of a Neural network are:

- Training
- Input data
- Layers
- Weights
- Targets
- Loss function
- Optimizer function
- Predictions

Copyright © www.ITbodhi.com

NN Characteristics - Training

Training is how we teach a neural network what we want it to learn. It follows a simple five step process:

- Create a **training data set**, which we will call **x** and load its **labels as targets y**
- **Feed the x data forward** through the network with the **result being predictions y'**
- Figure out the **"loss" of the network**, which is the **difference between the predictions y' and the correct targets y**
- Compute the **"gradient" of the loss (l)** and which tells us how fast we're moving towards or away from the correct targets
- **Adjust the weights** of the network in the **opposite direction of the gradient** and go back to step two to try again

Copyright © www.ITbodhi.com

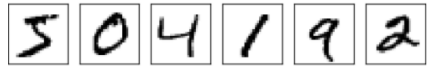
NN Characteristics - Weights

- **Weights are the strength of the connection between the various neurons**.
- In your brain, you have a series of **biological neurons**. They're connected to other neurons with electrical/chemical signals passing between them.
- But the connections are not static. Over time **some of those connections get stronger and some weaker**.
- The more electro-chemical signals flowing between two biological neurons, the stronger those connections get. In essence, your brain rewires itself constantly as you have new experiences. It encodes your memories and feelings and ideas about those experiences by strengthening the connections between some neurons.
- Just like the neurons in your head form stronger or weaker connections, the weights in artificial neural network define the strength of the connections between perceptrons. Wiring them together allows them to have a more comprehensive view of the business problem when taken together. The ones that have stronger connections are considered more important for the problem we're trying to solve.

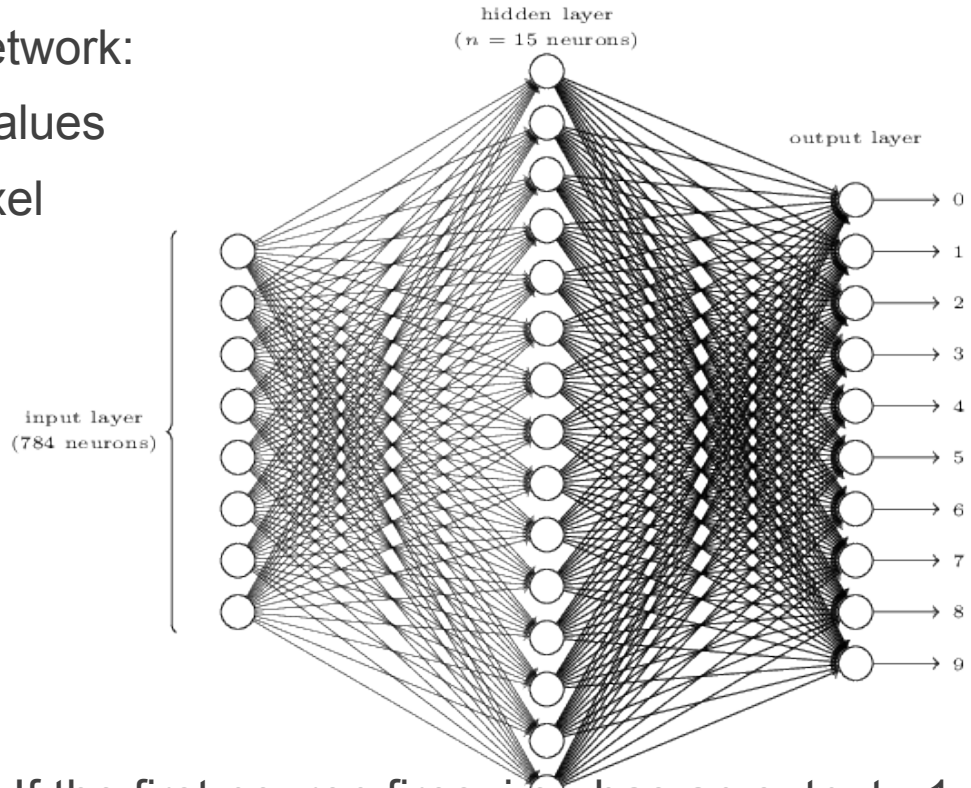
Copyright © www.ITbodhi.com

Neural Networks - Example

Suppose we want to recognize hand-written numbers:



To recognize individual digits we will use a three-layer neural network: The input layer of the network contains neurons encoding the values of the input pixels. Lets say, the training data contains 28*28 pixel Images. So the input layer contains 784 = 28*28 input neurons. The input pixels are greyscale, with a value of 0.0 representing White, a value of 1.0 representing Black and in between values representing gradually darkening shades of grey.



The second layer of the network is a hidden layer. The example shown illustrates a small hidden layer, containing just 15 nodes. The output layer of the network contains 10 neurons. If the first neuron fires, i.e., has an output =1, then that will indicate that the network thinks the digit is a 0. If the second neuron fires then that will indicate that the network thinks the digit is a 1. And so on.

Copyright © www.ITbodhi.com

Neural Networks - Intuition

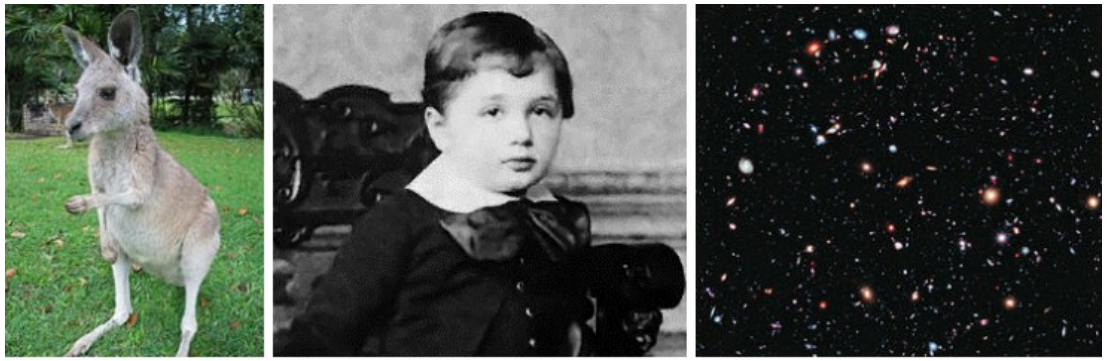
The basic functionality of a Neural network is as follows:

- It takes several input, processes it through multiple neurons from multiple hidden layers and returns the result using an output layer. This result estimation process is technically known as "**Forward Propagation**".
- Next, the result is compared with actual output. The task is to make the output to neural network as close to actual (desired) output. Each of these neurons are contributing some error to final output.
- Then, iteratively the value/ weight of neurons those are contributing more to the error, is minimized and this happens while traveling back to the neurons of the neural network and finding where the error lies. This process is known as "**Backward Propagation**".
- The neural networks use a common algorithm known as "Gradient Descent", which helps to optimize the task quickly and efficiently.
- One round of forward and back propagation iteration is known as one training iteration aka "**Epoch**".

Copyright © www.ITbodhi.com

Interpreting Neural Networks

Suppose we want to determine whether an image shows a human face or not:

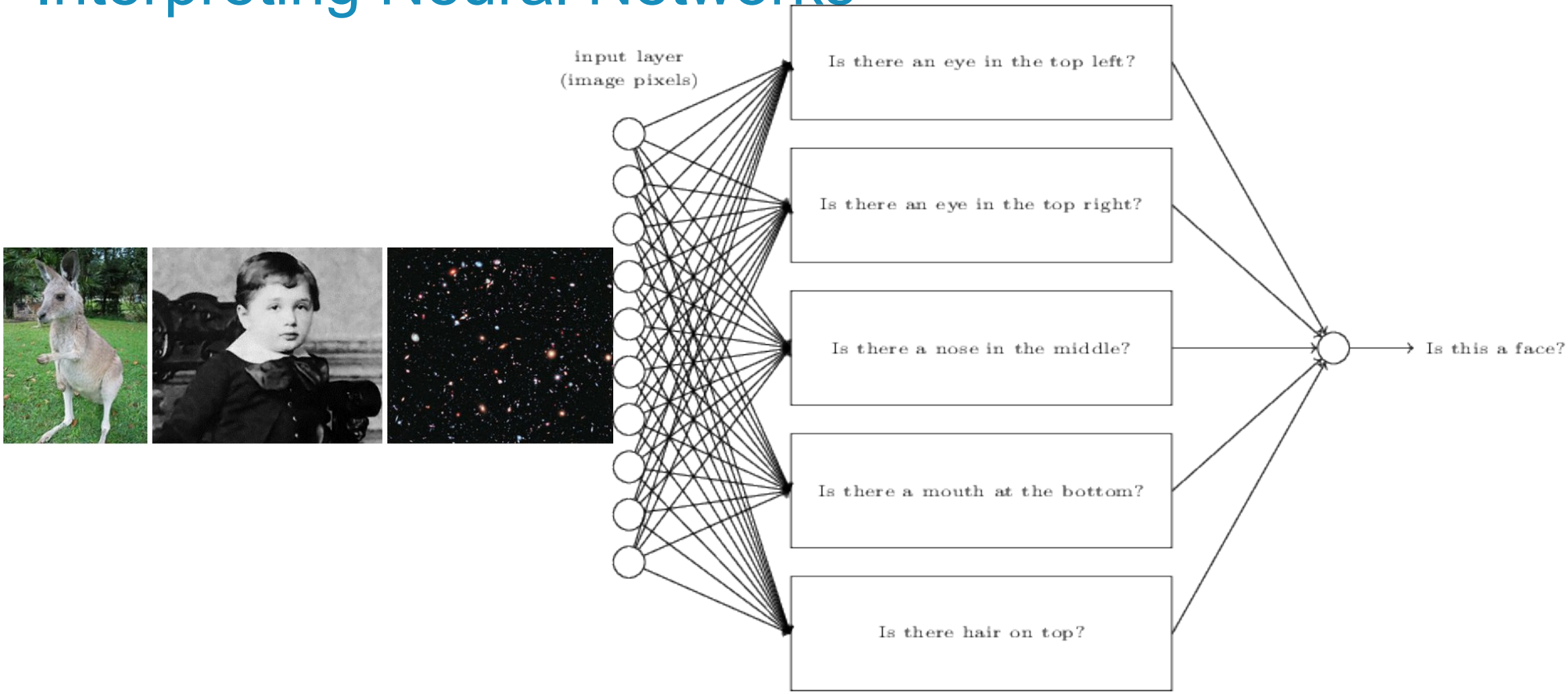


The pixels in the image will be used as input to a neural network, with the output from the network a single neuron indicating either "Yes, it's a face" or "No, it's not a face".

The problem can be decomposed into sub-problems: does the image have an eye in the top left? Does it have an eye in the top right? Does it have a nose in the middle? Does it have a mouth in the bottom middle? Is there hair on top? And so on. If the answers to several of these questions are "yes", or even just "probably yes", then we'd conclude that the image is likely to be a face. Conversely, if the answers to most of the questions are "no", then the image probably isn't a face.

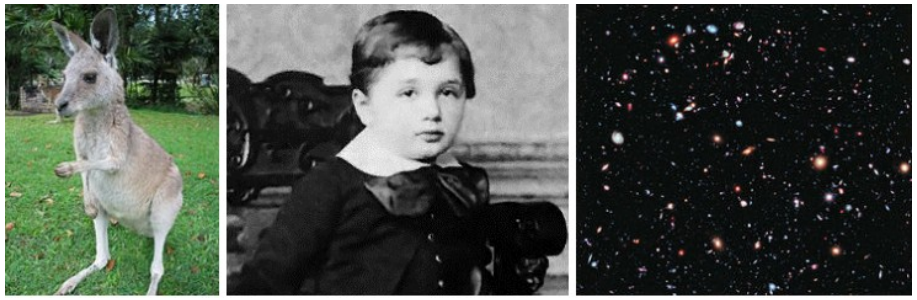
Copyright © www.ITbodhi.com

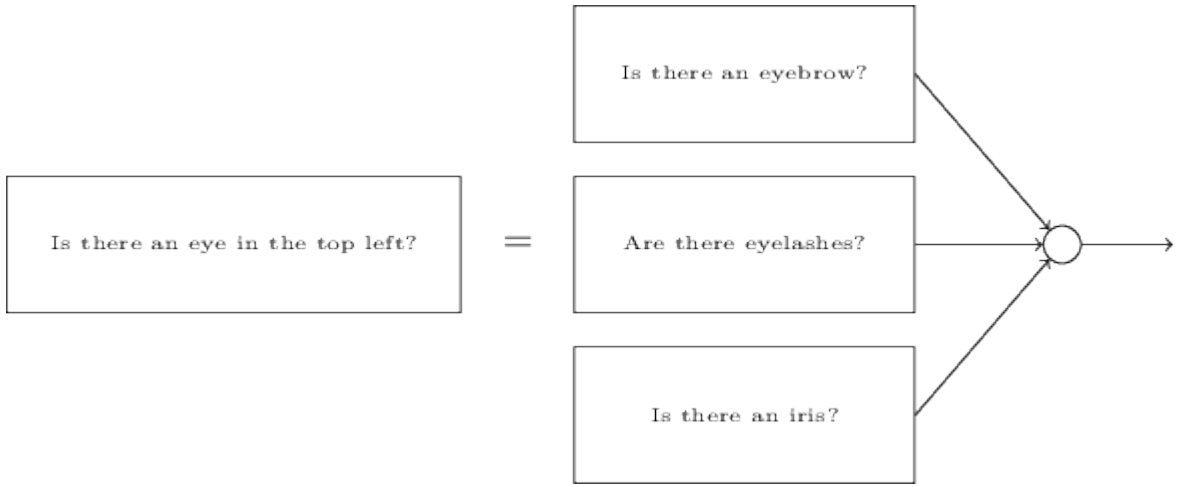
Interpreting Neural Networks



Copyright © www.ITbodhi.com

Interpreting Neural Networks





Those questions too can be broken down, further and further through multiple layers. Ultimately, we'll be working with sub-networks that answer questions so simple they can easily be answered at the level of single pixels. Those questions might, for example, be about the presence or absence of very simple shapes at particular points in the image. Such questions can be answered by single neurons connected to the raw pixels in the image.

The end result is a network which breaks down a very complicated question - does this image show a face or not - into very simple questions answerable at the level of single pixels. It does this through a series of many layers, with early layers answering very simple and specific questions about the input image, and later layers building up a hierarchy of ever more complex and abstract concepts.

Networks with this kind of many-layer structure - two or more hidden layers - are called *deep neural networks*.

Copyright © www.ITbodhi.com

Feedforward / Recurrent

- Neural networks where the output from one layer is used as input to the next layer are called *feedforward* neural networks. This means there are no loops in the network - information is always fed forward, never fed back.

- There are other models of artificial neural networks in which feedback loops are possible. These models are called recurrent neural networks. The idea in these models is to have neurons which fire for some limited duration of time, before becoming quiescent. That firing can stimulate other neurons, which may fire a little while later, also for a limited duration. That causes still more neurons to fire, and so over time we get a cascade of neurons firing.

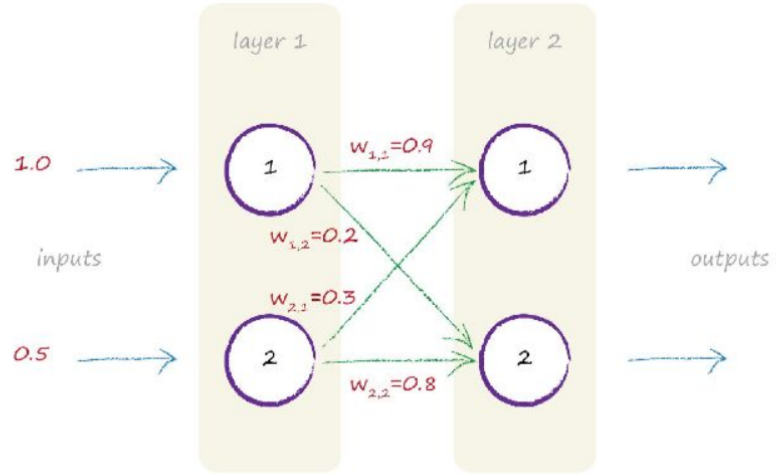
Copyright © www.ITbodhi.com

ANN – Things to remember

- For each observation ANN does multiple re-calibrations for each linkage weights. Hence, the time taken by the algorithm rises much faster than other traditional algorithm for the same increase in data volume.
- ANN is rarely used for predictive modelling. The reason being that Artificial Neural Networks (ANN) usually tries to over-fit the relationship. ANN is generally used in cases where what has happened in past is repeated almost exactly in same way. That is the reason it works very well in cases of image recognition and voice recognition.
- Artificial Neural Networks (ANN) have many different coefficients, which it can optimize. Hence, it can handle much more variability as compared to traditional models.

Copyright © www.ITbodhi.com

ANN – An Example



So let's first focus on node 1 in the layer 2. Both nodes in the first input layer are connected to it. Those input nodes have raw values of 1.0 and 0.5. The link from the first node has a weight of 0.9 associated with it. The link from the second has a weight of 0.3. So the combined moderated input is:

```
x = (output from first node * link weight) + (output from second node
                                              * link weight)

x = (1.0 * 0.9) + (0.5 * 0.3)

x = 0.9 + 0.15

x = 1.05
```

If we didn't moderate the signal, we'd have a very simple addition of the signals 1.0 + 0.5, but we don't want that. It is the weights that do the learning in a neural networks as they are iteratively refined to give better and better results.

So, we've now got **x** = 1.05 for the combined moderated input into the first node of the second layer. We can now, finally, calculate that node's output using the activation function $y = \frac{1}{(1+e^{-x})}$. Feel free to use a calculator to do this. The answer is **y** = 1 / (1 + 0.3499) = 1 / 1.3499. So **y** = 0.7408.

Copyright © www.ITbodhi.com

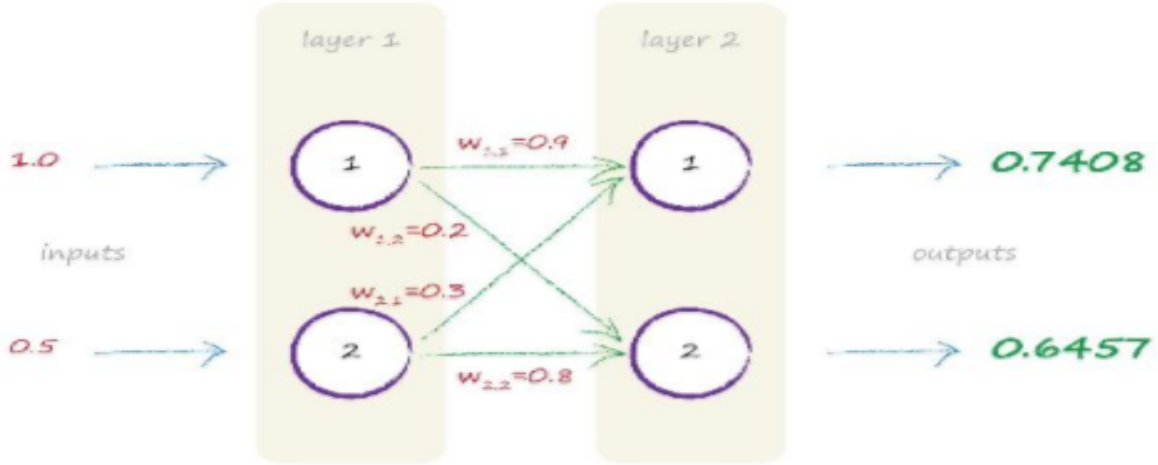
ANN – An Example

Let's do the calculation again with the remaining node which is node 2 in the second layer. The combined moderated input x is:

```
x = (output from first node * link weight) + (output from second node
                                              * link weight)

x = (1.0 * 0.2) + (0.5 * 0.8)

x = 0.2 + 0.4

x = 0.6
```

So now we have **x**, we can calculate the node's output using the sigmoid activation function **y** = 1/(1 + 0.5488) = 1/(1.5488). So **y** = 0.6457.

The following diagram shows the network's outputs we've just calculated:



Copyright © www.ITbodhi.com

Matrix Multiplication

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} (1*5)+(2*7) & (1*6)+(2*8) \\ (3*5)+(4*7) & (3*6)+(4*8) \end{pmatrix}$$

$$= \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

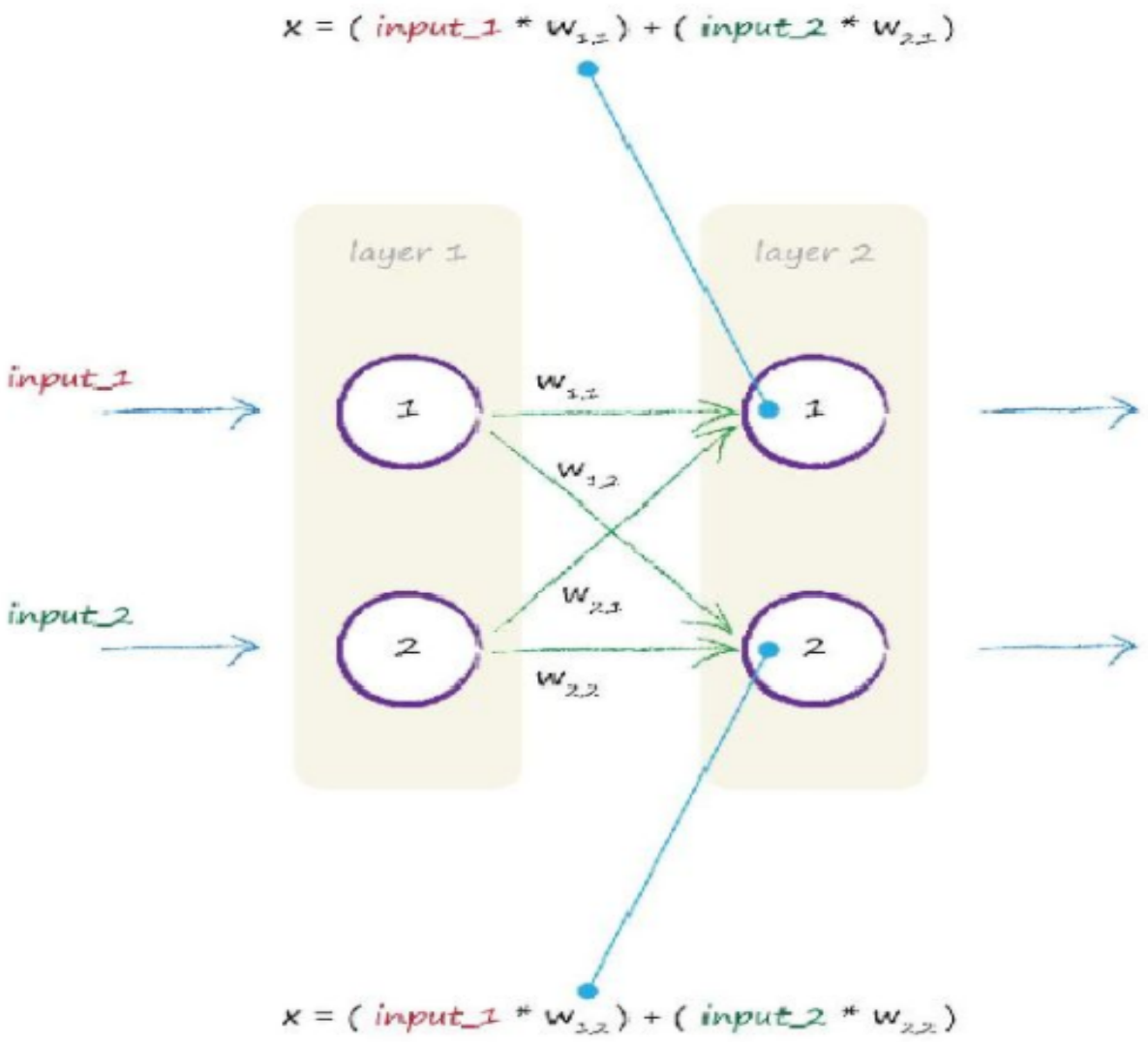
Copyright © www.ITbodhi.com

ANN – An Example

$$\begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} input\_1 \\ input\_2 \end{pmatrix} = \begin{pmatrix} (input\_1 * w_{1,1}) + (input\_2 * w_{2,1}) \\ (input\_1 * w_{1,2}) + (input\_2 * w_{2,2}) \end{pmatrix}$$

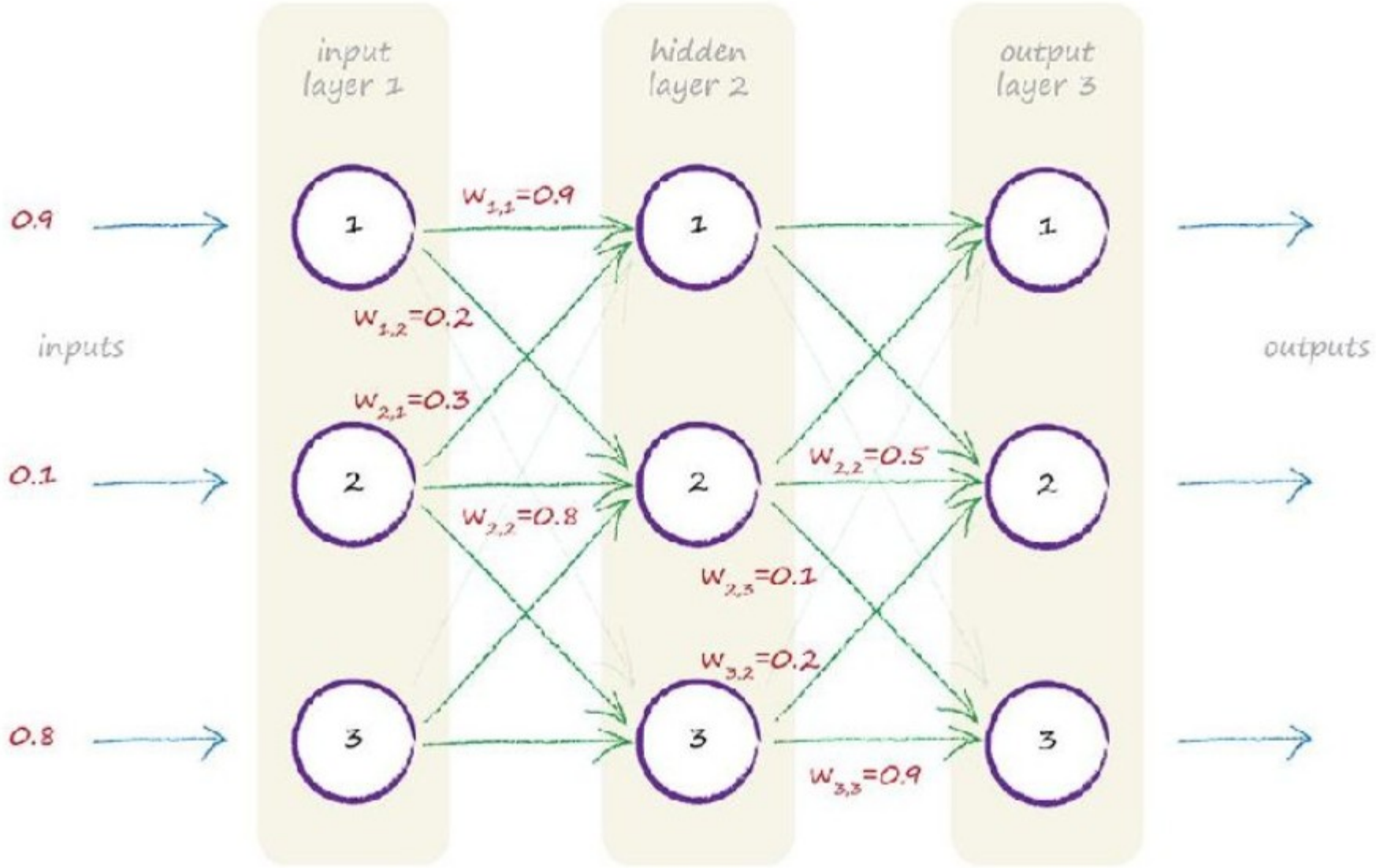
Copyright © www.ITbodhi.com

ANN – An Example



Copyright © www.ITbodhi.com

ANN – An Example



Copyright © www.ITbodhi.com

ANN – An Example

$$I = \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$$

$$W_{input\_hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix}$$

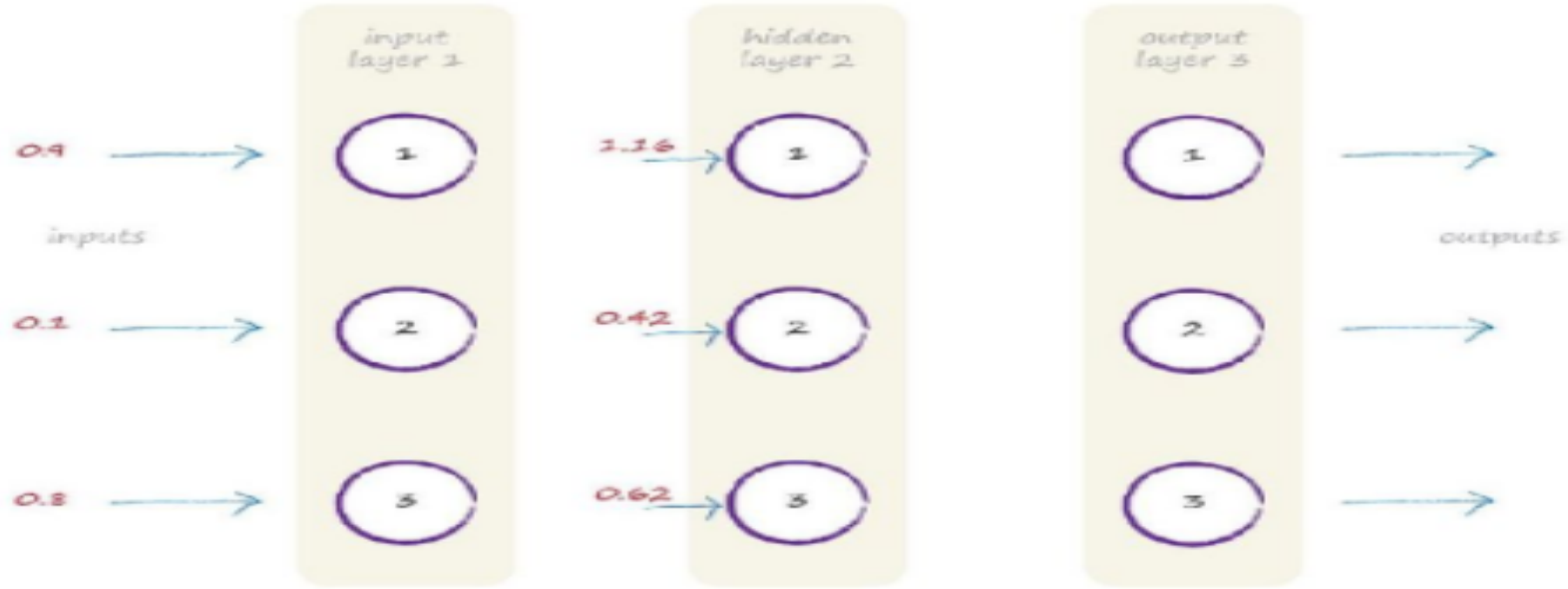
$$W_{hidden\_output} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix}$$

$$X_{hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix} \cdot \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$$

$$X_{hidden} = \begin{pmatrix} 1.16 \\ 0.42 \\ 0.62 \end{pmatrix}$$

Copyright © www.ITbodhi.com

ANN – An Example



So far so good, but there's more to do. You'll remember those nodes apply a sigmoid activation function to make the response to the signal more like those found in nature. So let's do that:

$$O_{hidden} = \text{sigmoid}(X_{hidden})$$

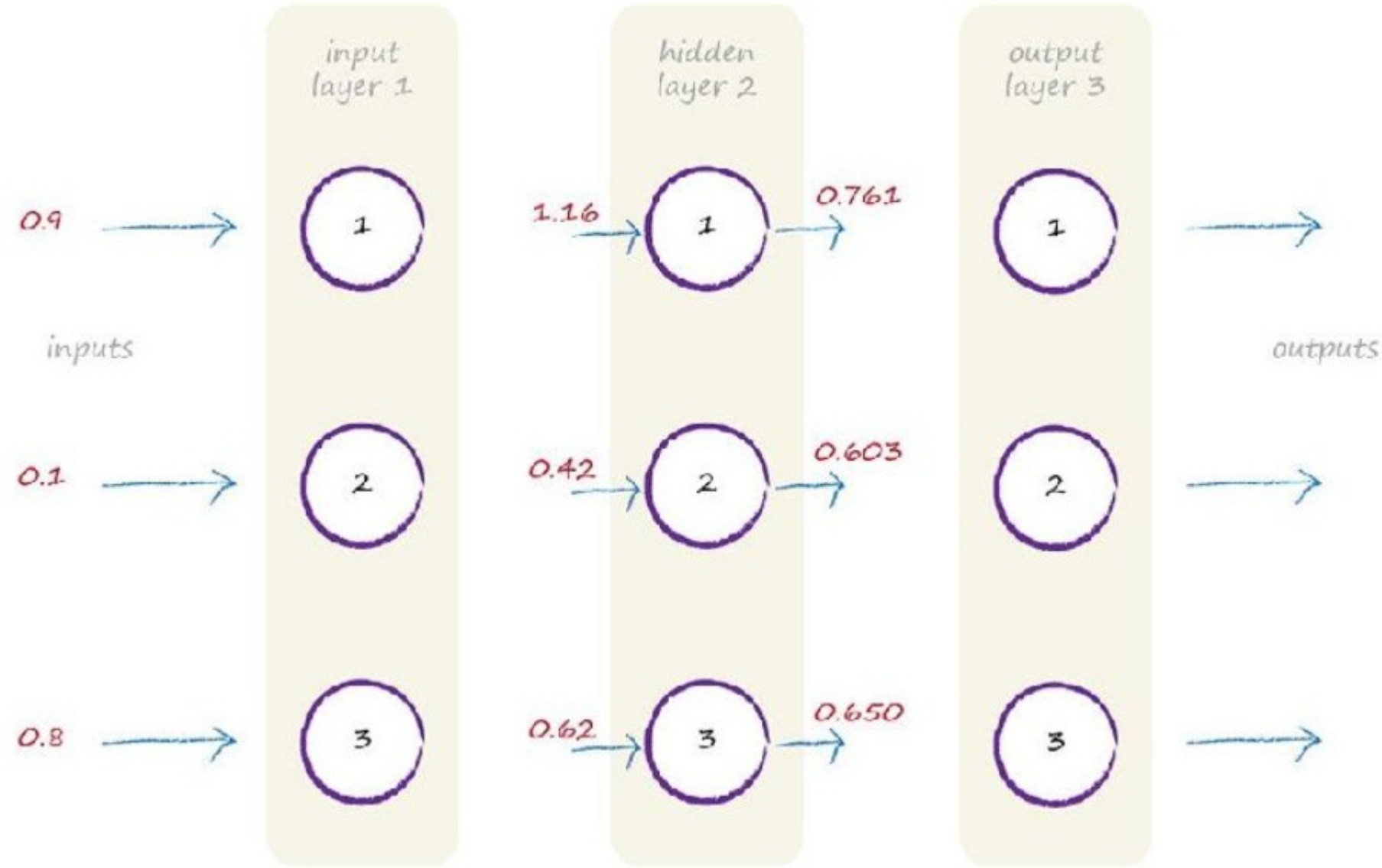
The sigmoid function is applied to each element in X_{hidden} to produce the matrix which has the output of the middle hidden layer.

$$O_{hidden} = \text{sigmoid} \begin{pmatrix} 1.16 \\ 0.42 \\ 0.62 \end{pmatrix}$$

$$O_{hidden} = \begin{pmatrix} 0.761 \\ 0.603 \\ 0.650 \end{pmatrix}$$

Copyright © www.ITbodhi.com

ANN – An Example



Copyright © www.ITbodhi.com

ANN – An Example

$$X_{output} = W_{hidden\_output} \cdot O_{hidden}$$

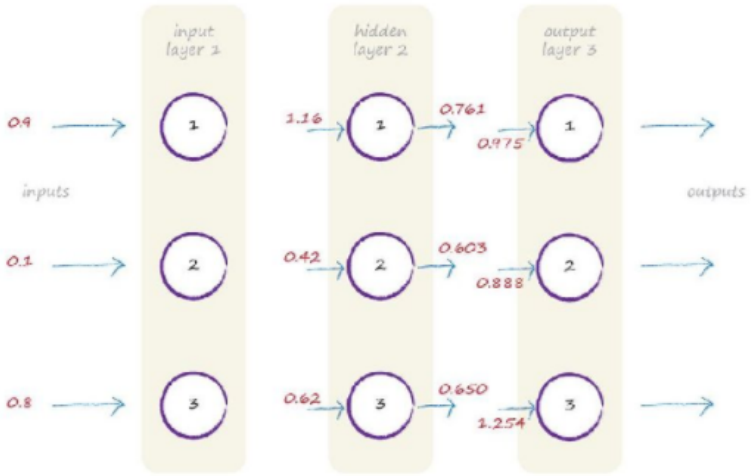
So working this out just in the same way gives the following result for the combined moderated inputs into the final output layer.

$$X_{output} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix} \cdot \begin{pmatrix} 0.761 \\ 0.603 \\ 0.650 \end{pmatrix}$$

$$X_{output} = \begin{pmatrix} 0.975 \\ 0.888 \\ 1.254 \end{pmatrix}$$

Copyright © www.ITbodhi.com

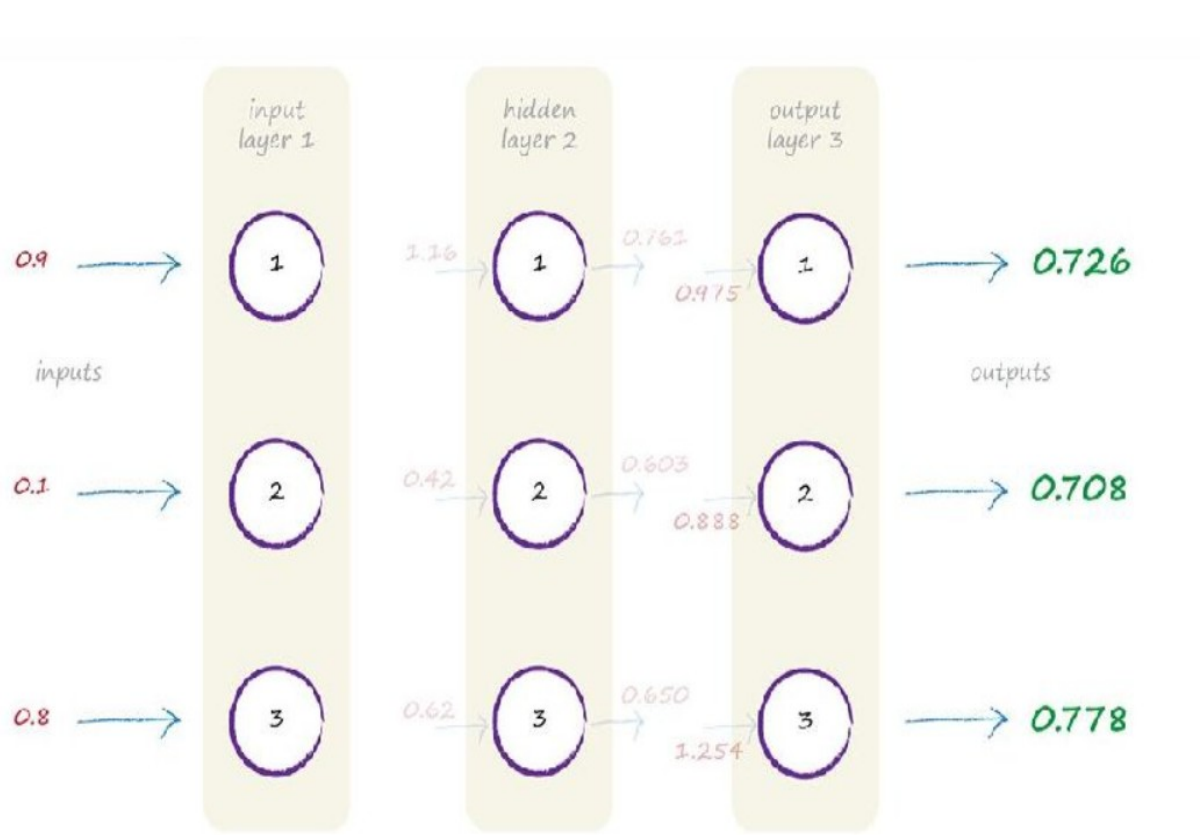
ANN – An Example



All that remains is to apply the sigmoid activation function, which is easy.

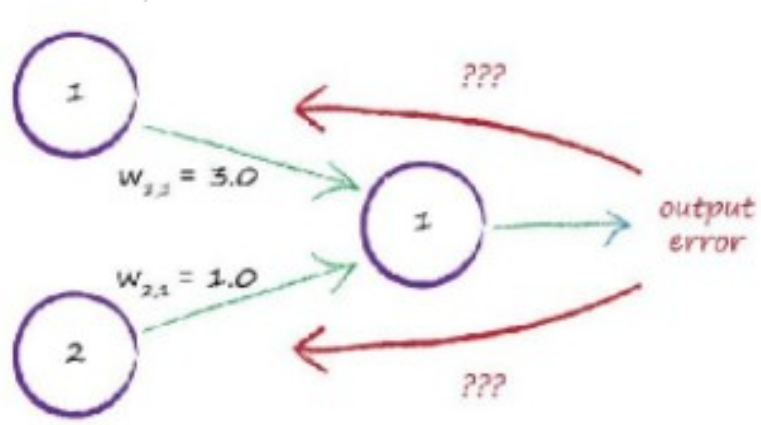
$$O_{output} = sigmoid\begin{pmatrix} 0.975 \\ 0.888 \\ 1.254 \end{pmatrix}$$

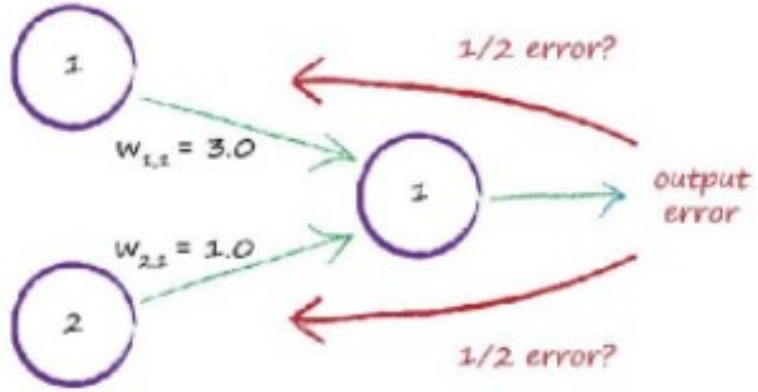
$$O_{output} = \begin{pmatrix} 0.726 \\ 0.708 \\ 0.778 \end{pmatrix}$$



Copyright © www.ITbodhi.com

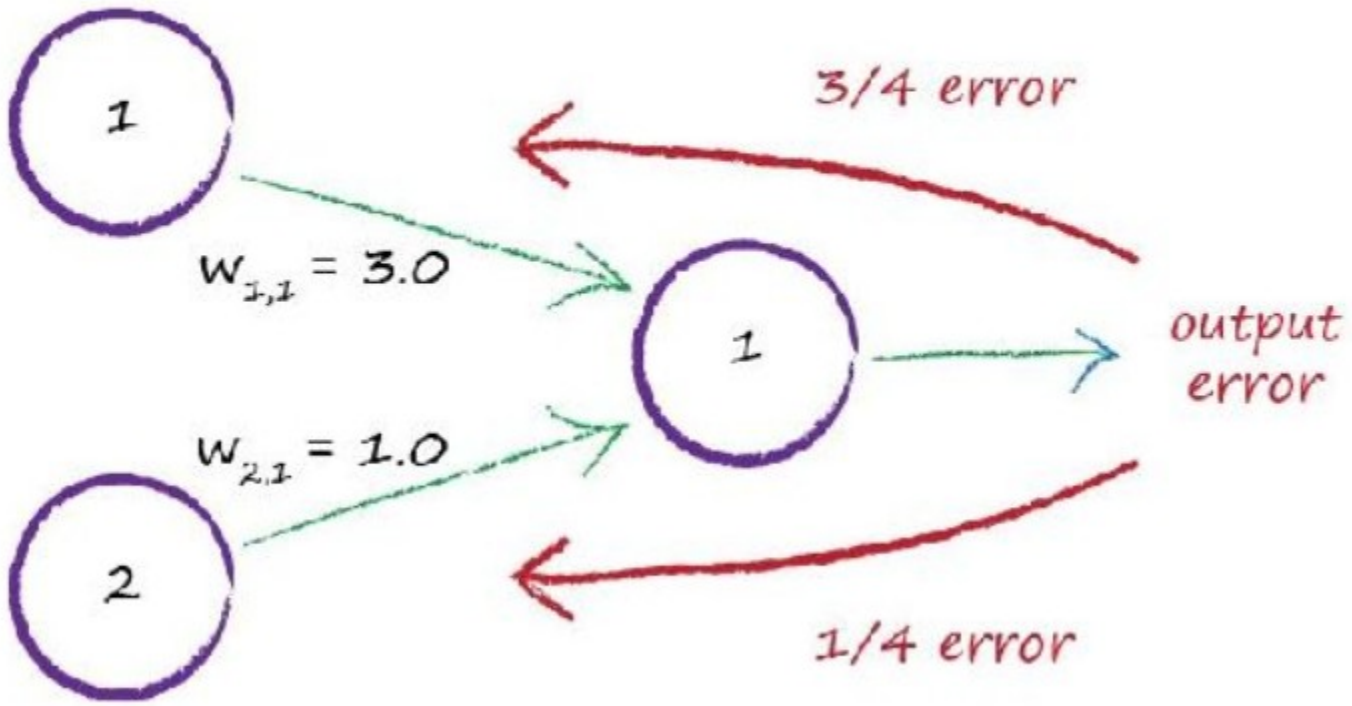
ANN – An Example





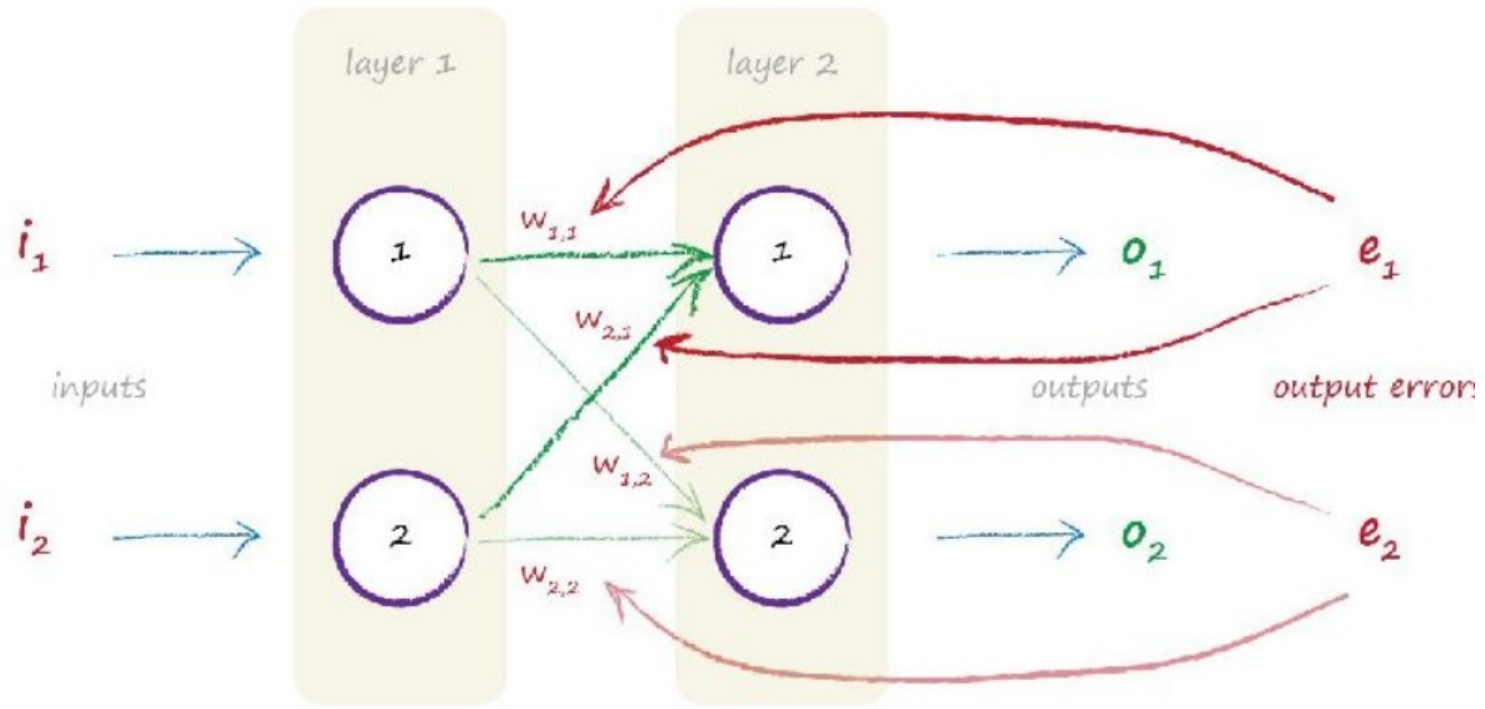
Copyright © www.ITbodhi.com

ANN – An Example



Copyright © www.ITbodhi.com

ANN – An Example



$$\frac{w_{11}}{w_{11} + w_{21}}$$

$$\frac{w_{21}}{w_{11} + w_{21}}$$

Copyright © www.ITbodhi.com

Training Process of a Neural Network

During training, the objective is to find the weights for each layer, which minimize the error on the training set.

- Initialize some random weights
- Using the inputs and the linkages, find the activation rate of hidden nodes
- Using the activation rate of Hidden nodes and the linkages to output, find the activation rate of output nodes
- Evaluate some error function(e.g Desired Output – Current Output). Back-propagate the errors.
- Adjust the weights by a small increment
- Repeat the process until the error is minimized.(or a certain no of times)

Copyright © www.ITbodhi.com