राष्ट्रीय न्यायालयिक विज्ञान विश्वविद्यालय
(राष्ट्रीय महत्त्व का संस्थान, गृह मंत्रालय, भारत सरकार)
National Forensic Sciences University
(An Institution of National Importance under Ministry of Home Affairs, Government of India)

Neural Network

Introduction

- The definition of a neural network, as provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielson, in Neural Network Primer—Part I, is as follows:

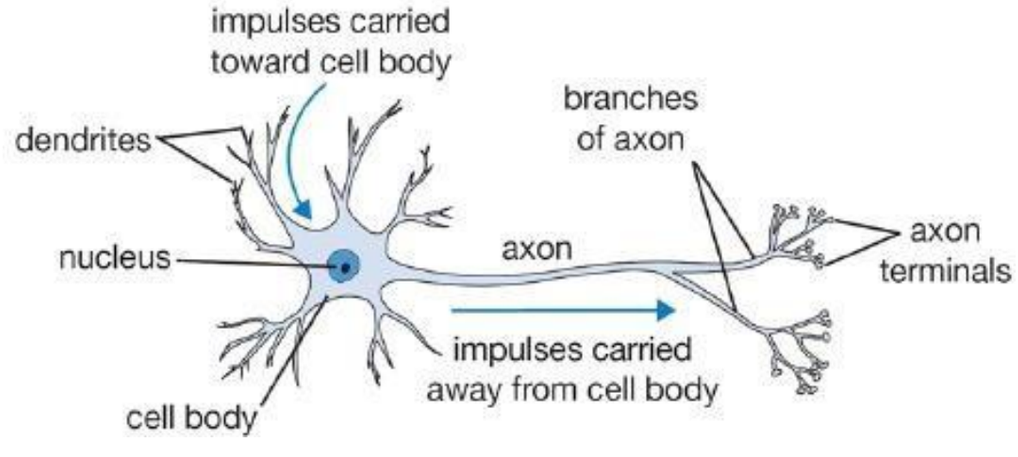
"A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Introduction

- In practice, we can think of artificial neural networks as a computational model that is based on **how the brain is believed to work**.
- Hence, the mathematical model is inspired by **biological neurons**.

Biological Neuron

- The main computational units of the brain are known as neurons; in the human nervous system, approximately 86 billion neurons can be found, all of which are connected by synapses. The following diagram shows a biological neuron:



Biological Neuron

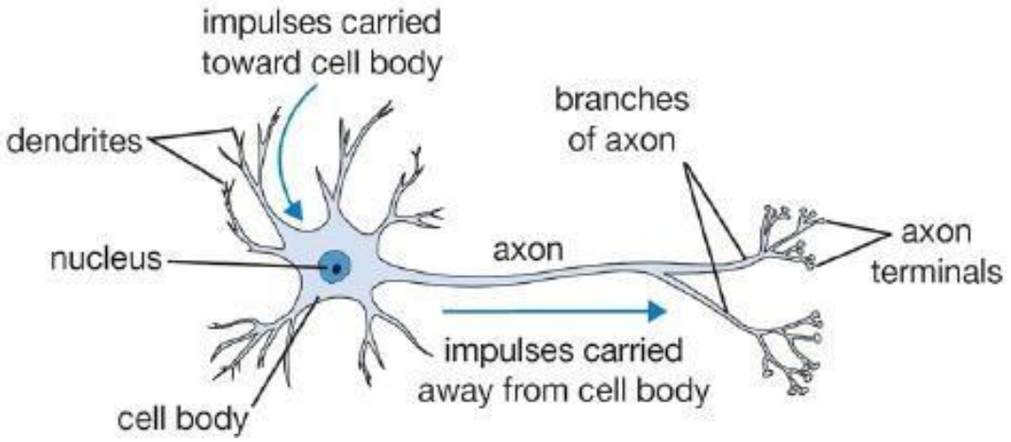
Biological neurons are made up of the following:

- **Dendrites**: Minor fibers that carry information, in the form of an electric signal, from the outside to the nucleus.
- **Synapses**: These are the connection points among neurons. Neurons receive input signals on the synapses that are connected to the dendrites.
- **Nucleus**: This receives the signals from the dendrites, elaborates on them, and produces a response (output signal) that it sends to the axon.
- **Axon**: The output channel of the neuron. It can be connected to other neuron synapses.

Biological Neuron

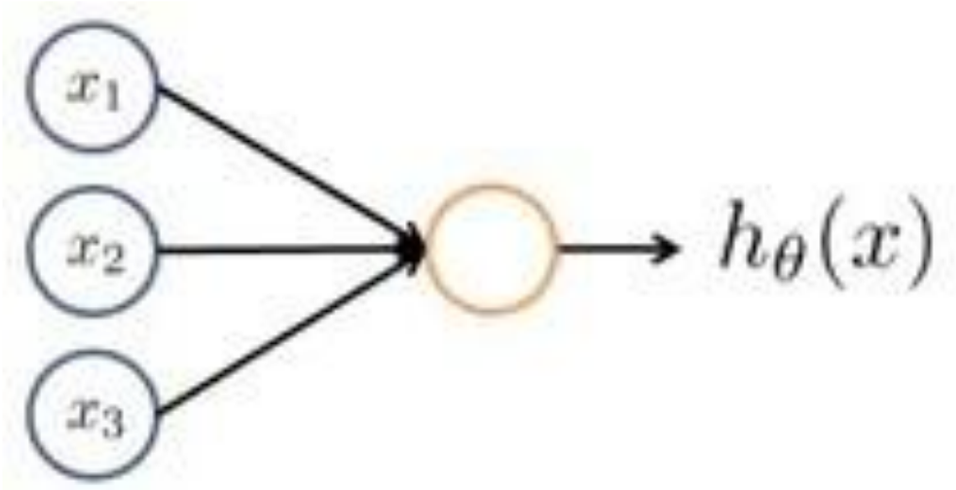
- Each neuron receives input signals from its dendrites and transports them to the nucleus where they are processed; dendrites process the signals, thereby integrating (adding up or combining) excitation and inhibition from every input synapse.
- The nucleus receives the integrated signals and adds them. If the final sum is above a certain threshold, the neuron fires and the resulting information is carried down through the axon and thus to any other connected neuron.
- The amount of signal that's transmitted among neurons depends on the strength of the connections.
- It is the arrangement of the neurons and the strength of these synapses that establish the function of the neural network.

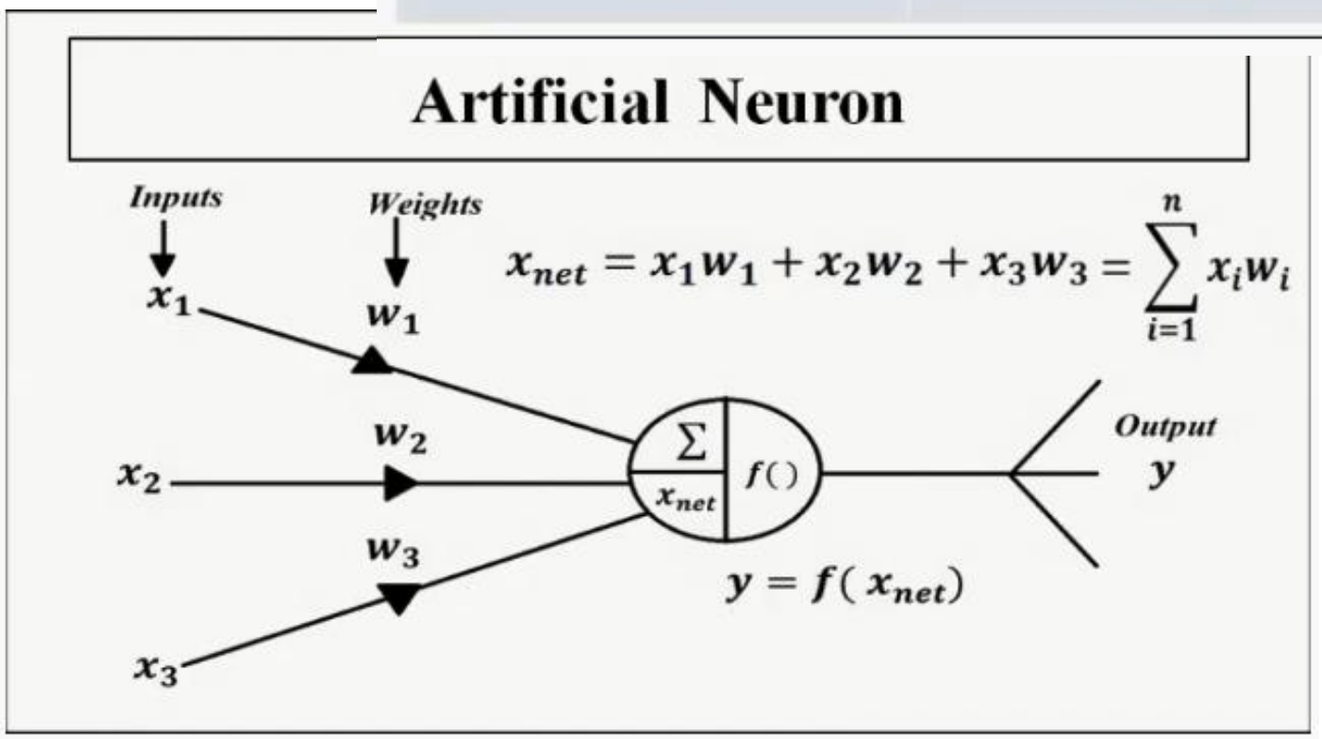
Artificial Neuron



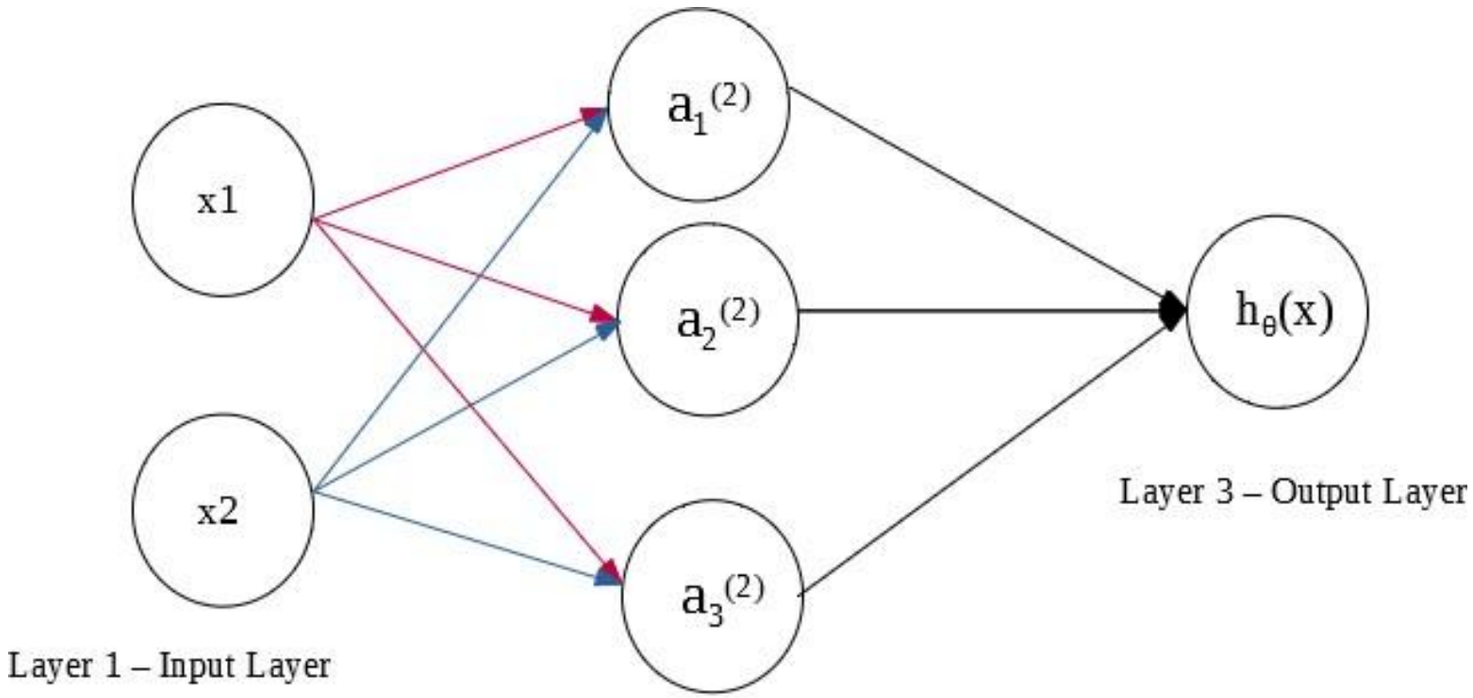
Correlating Biological and Artificial Neuron

Biological Neuron	Artificial Neuron
Cell	Neuron / Node
Dendrites / Synapse	Weights
Soma	Net Input
Axon	Output





Artificial Neural Network [ANN]



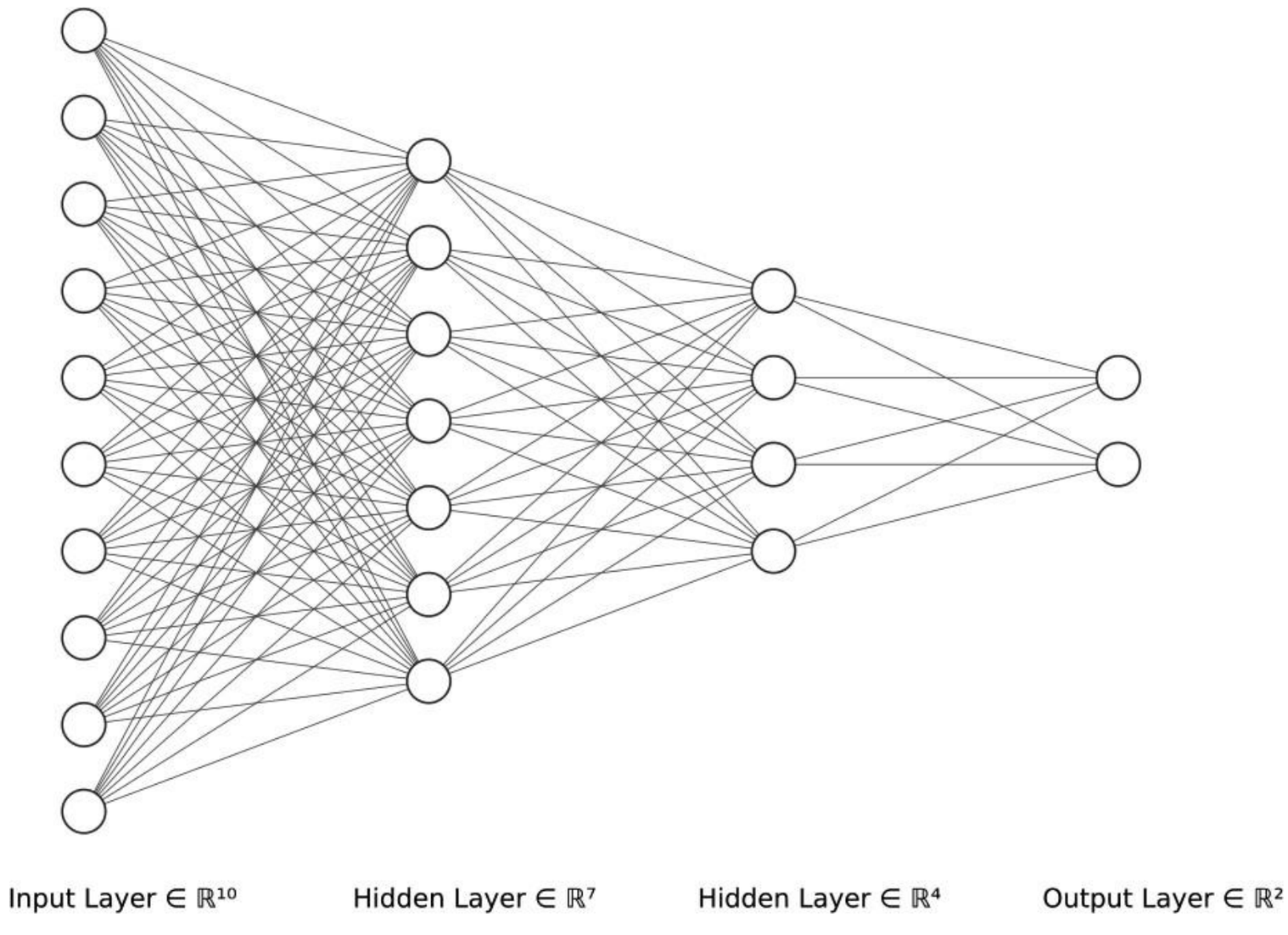
Artificial Neural Network [ANN]

The above diagram depicts a simple ANN with 3 layers:

- Layer 1 is also known as input layer (X)
- Layer 3 is also known as output layer(y)
- Layer 2 is in general referred as hidden layer.

राष्ट्रीय न्यायालयिक विज्ञान विश्वविद्यालय
(राष्ट्रीय महत्त्व का संस्थान, गृह मंत्रालय, भारत सरकार)
National Forensic Sciences University
(An Institution of National Importance under Ministry of Home Affairs, Government of India)

Artificial Neural Network [ANN]



ANN - Architecture

- Organizing networks into layers allows us to create stacks of fully connected layers, with a different number of neurons per layer.
- We can think about a multi-layer neural network as a model with visible and hidden layers.
- The visible layers are just the input and output layers; the hidden layers are the ones that aren't connected to the outside.
- The number of neurons in the hidden layers is entirely arbitrary, and it changes the learning capacity of the network.
- The input and output layers, instead, have a fixed dimension due to the task we are going to solve (for example, if we want to solve an n-classes classification on D-dimensional inputs, then we need an input layer with D inputs and an output layer with n outputs).

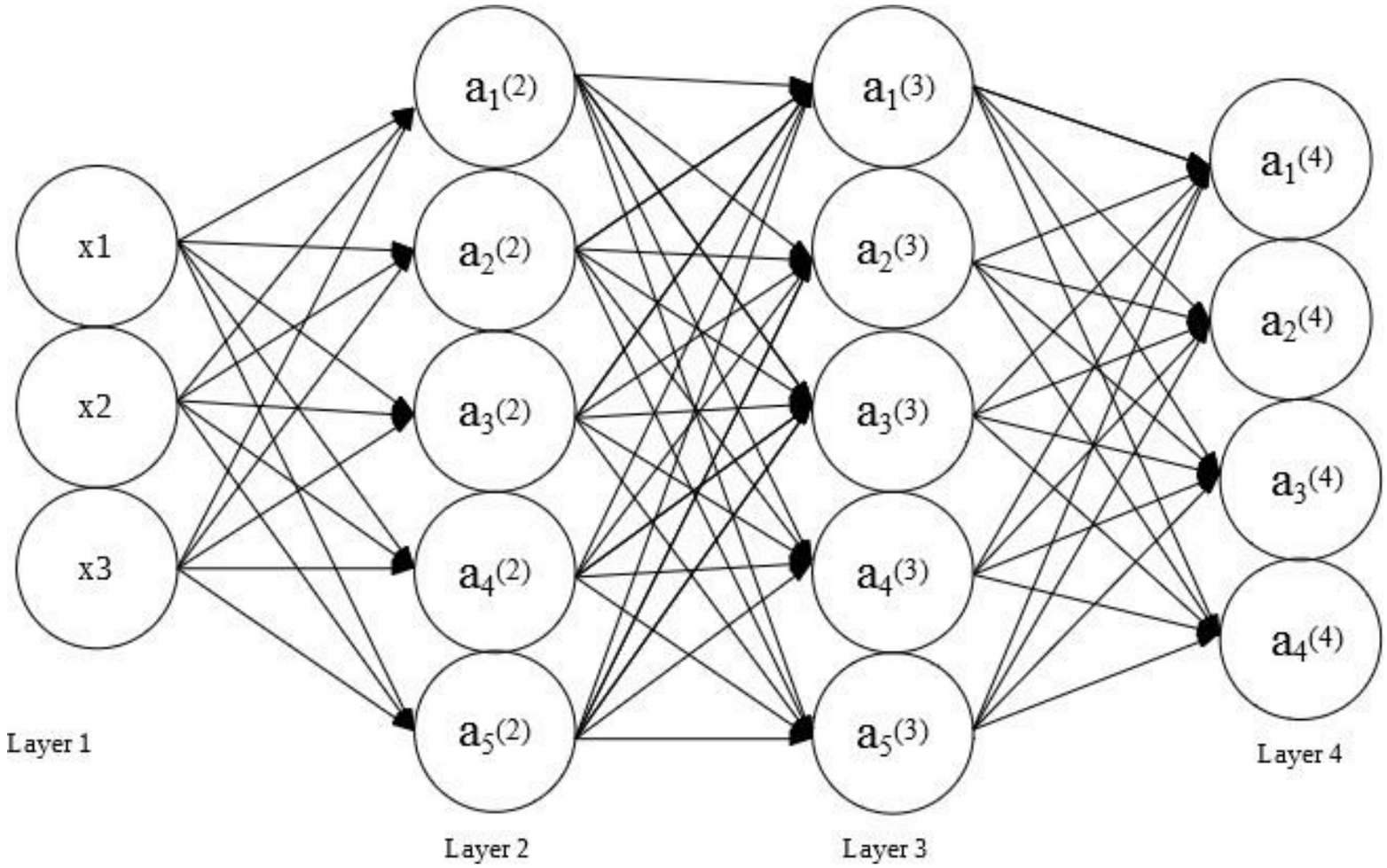
ANN - Architecture

- While the design of the input and output layers of a neural network is straightforward, the design of the hidden layers is not so simple.
- There are no rules; neural networks researchers have developed many design heuristics for hidden layers which help to get the correct behavior.
- In general, increasing the number of neurons per layer and/or the number of layers in a neural network means having to increase the network capacity.
- This means that the neural network can express more complicated functions and that the space of representable functions grows; however, this is good and bad at the same time.

Why ANN?

- Neural networks can be arranged in different topologies, and the geometry changes what the neural networks see (the input stimuli). Moreover, it's straightforward to create layers upon layers of neural networks with different topologies, creating deep models.
- One of the greatest strengths of neural networks is their ability to become feature extractors: other machine learning models need the input data to be processed, have their meaningful features extracted, and only on those features (manually defined!) can the model be applied.
- Neural networks, on the other hand, can extract meaningful features from any input data by themselves (depending on the topology of the layers that are used).

Artificial Neural Network [ANN]

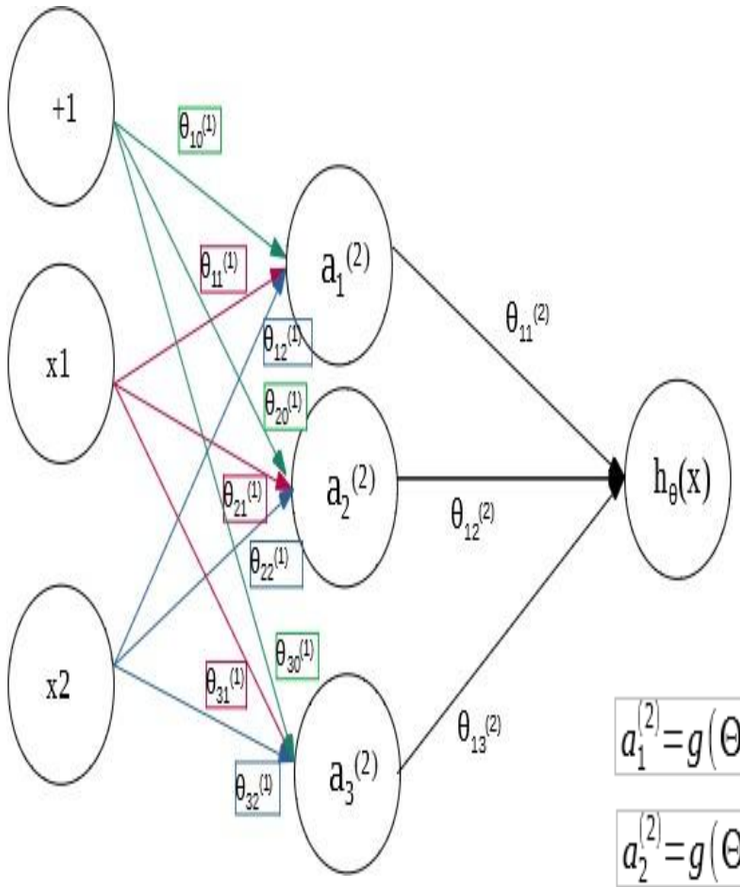


ANN Feed Forward

A feedforward neural network is a biologically inspired classification algorithm. It consist of a (possibly large) number of simple neuron-like processing *units*, organized in *layers*.

Every unit in a layer is connected with all the units in the previous layer. These connections are not all equal: each connection may have a different strength or *weight*. The weights on these connections encode the knowledge of a network. Often the units in a neural network are also called *nodes*.

Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. During normal operation, that is when it acts as a classifier, there is no feedback between layers. This is why they are called *feedforward* neural networks.



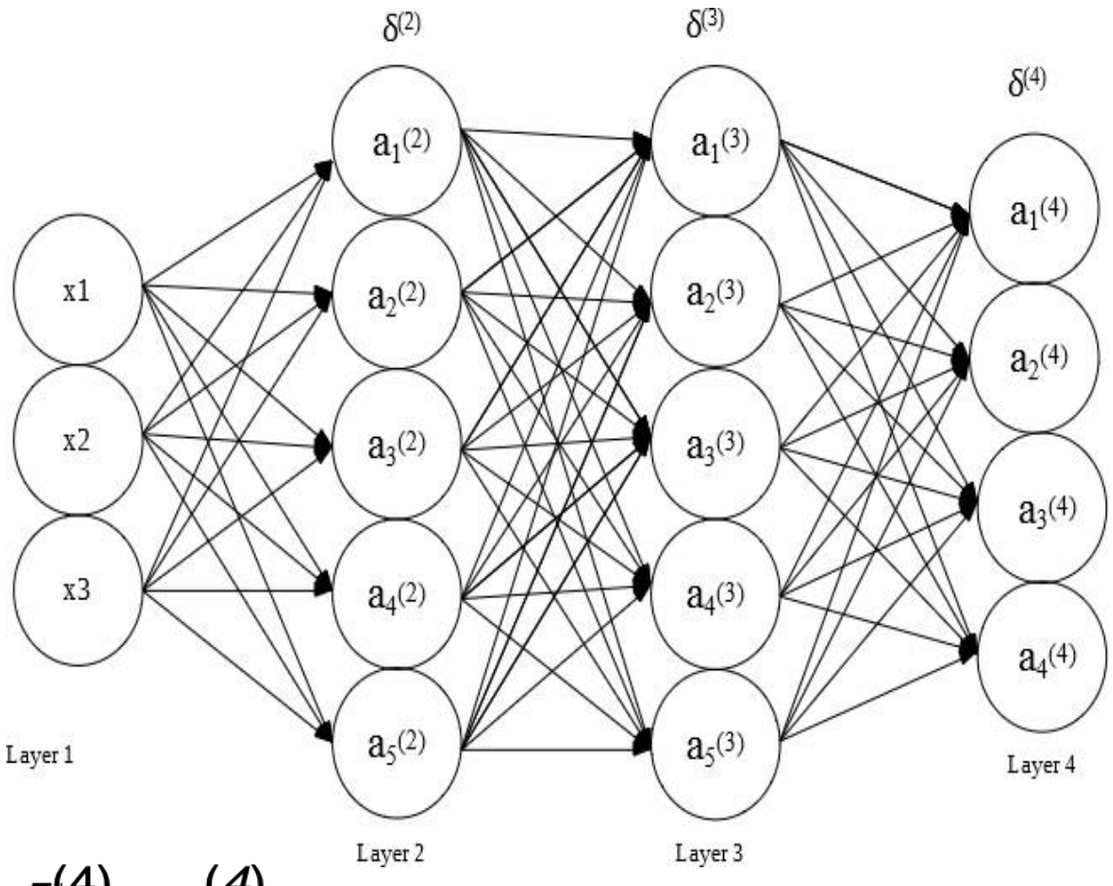
$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2\right)$$

$$h_\theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

ANN – Back Propagation

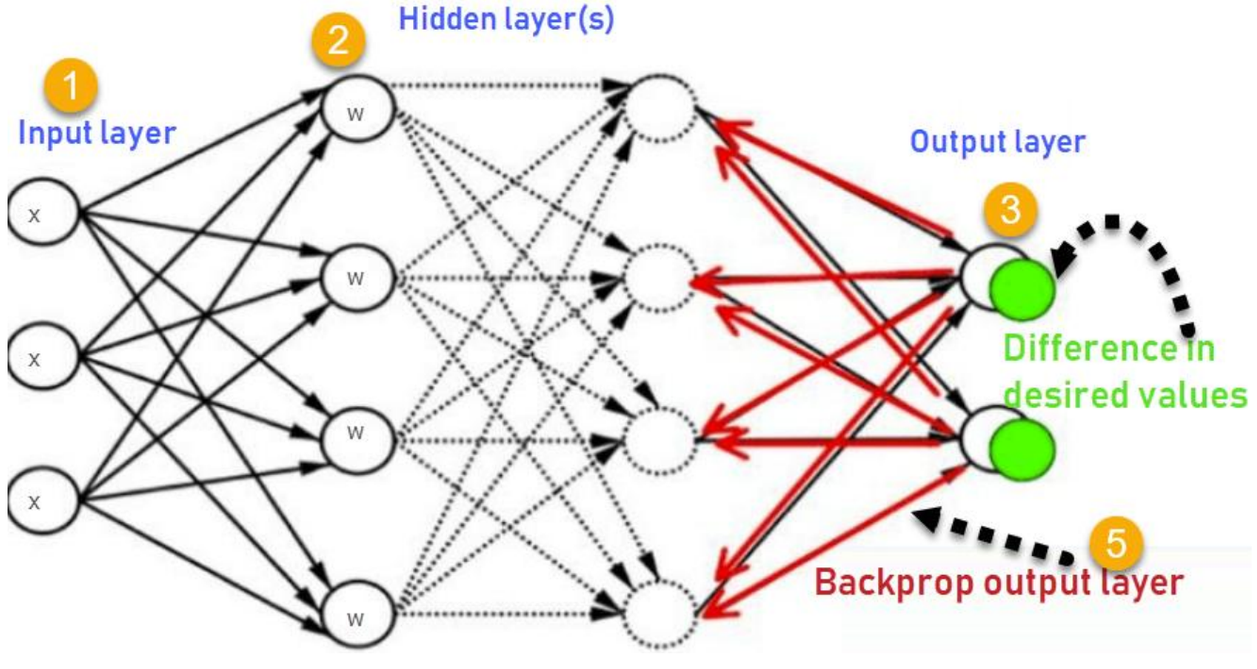


$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g(z^{(2)})$$

Back-propagation is just **a way of propagating the total loss back into the neural network** to know how much of the loss every node is responsible for, and subsequently updating the weights in such a way that minimizes the loss by giving the nodes with higher error rates lower weights and vice versa.



How Backpropagation Algorithm Works

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

- Inputs X, arrive through the preconnected path
- Input is modeled using real weights W.
- The weights are usually randomly selected.
- Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
- Calculate the error in the outputs
- $Error_B$= Actual Output – Desired Output
- Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Why We Need Backpropagation?

Most prominent advantages of Backpropagation are:

Backpropagation is fast, simple and easy to program

It has no parameters to tune apart from the numbers of input

It is a flexible method as it does not require prior knowledge about the network

It is a standard method that generally works well

It does not need any special mention of the features of the function to be learned.

Putting everything together

- Randomly initialize weights
- Implement Forward Propagation to get…… $h_\Theta(x^{(i)})$ for any x(i)
- Calculate Cost Function …… $J(\Theta)$
- Implement backpropagation to compute partial derivatives $\frac{\partial}{\partial \Theta_{ij}^{(}}$
- Use gradient descent with backpropagation to minimize $J(\Theta)$

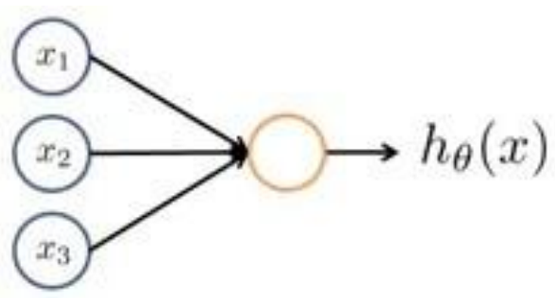
Perceptrons

- A perceptron (as defined by Frank Rosenblatt) can be considered as a single neuron in an artificial neural network (ANN).
- A perceptron can be defined using six specific mathematical representations that demonstrate its learning mechanism.
- These representations are the inputs, weights, bias term, summation, and the activation function.

Perceptrons

Inputs:

- Essentially, it takes in feature inputs.
- The following perceptron only has three input channels, these being x1, x2, and x3.
- These feature inputs (x1, x2, and x3) can be any independent variable.



Perceptrons

Weights:

- You can think of weight as the relative importance a specific input feature has with respect to the output.
- E.g. the area and locality may have higher weights compared to parking facility in case of the housing price prediction model.
- These weights are randomly initialized at first, and are learned as our models see more and more data.

Perceptrons

Summations:

- Now we have our input features flowing into our perceptron, with each input feature paired up with a randomly initialized weight.
- We want to use these two matrices to represent the combined effect of our input features and their weights.

Perceptrons

Non-Linearity:

- Unfortunately, linearity is often not guaranteed with real-world data, as we model observations using multiple features, each of which could have a varied and disproportional contribution towards determining our output classes.
- In fact, our world is extremely non-linear, and hence, to capture this non-linearity in our perceptron model, we need it to incorporate non-linear functions that are capable of representing such phenomena.

Perceptrons

Non-Linearity:

- By doing so, we increase the capacity of our neuron to model more complex patterns that actually exist in the real world, and draw decision boundaries that would not be possible using linear functions.
- These types of functions, used to model non-linear relationships in our data, are known as activation functions.

Perceptrons

Activation Function:

- Basically we want our perceptron to figure out the ideal combinations of weights and a threshold, allowing it to reliably match our inputs to the correct output class.
- Hence, we compare our reduced feature representation with a threshold value, and then activate our perceptron unit.
- This very function that compares our reduced feature value against a threshold, is known as an activation function.

Perceptrons

Bias Term:

- We can think of this bias as the weight of a fictional input.
- This fictional input is said to be always present, allowing our activation unit to fire at will, without requiring any input features to be explicitly present.
- The motivation behind this term is to be able to manipulate the shape of our activation function, which in turn impacts the learning of our model. We want our shape to flexibly fit different patterns in our data.
- The weight of the bias term is updated in the same manner as all the other weights .

Perceptrons

Bias Term:

- This bias term is randomly initialized at the beginning of our training session, and is iteratively updated as the model sees more examples, and learns from these examples.
- It is important to understand that although we randomly initialize model parameters, such as the weights and biases, our hope is to actually show the model enough input examples and their corresponding output classes.
- In doing so, we want our model to learn from its errors, searching for the ideal parametric combinations of weights and bias corresponding to the correct output classes.

References

1. Hands-On Neural Networks with TensorFlow 2.0 by Paolo Galeone
2. Machine Learning by Prof. Andrew NG, Stanford University
3. Hands-On Neural Networks with Keras By Niloy Purkait