

Deep Learning

Lesson Objectives



On completion of this lesson, you will be able to understand the key-concepts of Artificial Neural Networks(ANN) :

- Nodes
- Activation Function
- How do Neural Networks work
- How do Neural Networks learn
- Gradient Descent
- Backpropagation

Followed by a demonstration on creating and testing an ANN

What is Deep-Learning?



- Jeffrey Hinton is known as the godfather of deep learning
- Deep-Learning mimics the behavior of human-brain
- There are billions of NEURONS in the human-brain



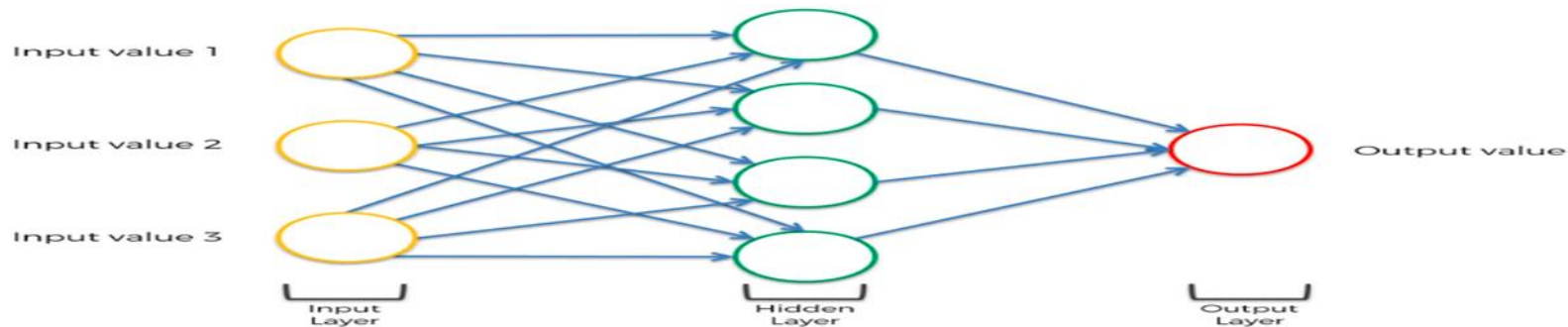
Cerebellum of the human-brain



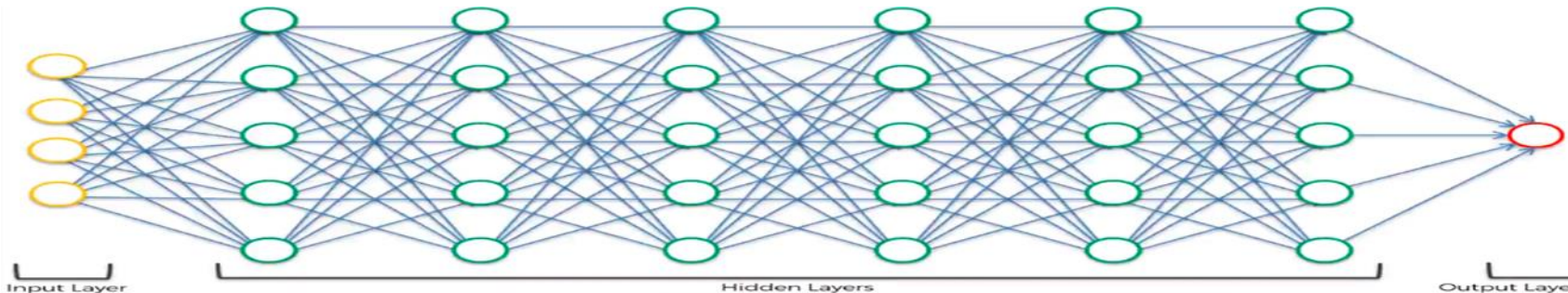
Inter-connected Neurons in the human-brain



- Mimicking neurons-like behavior in computers using Artificial Neural Network(ANN)
- An ANN can have several layers of nodes or neurons
- The layers of an ANN could be Input-Layer, Hidden-Layers and Output-Layer



- This is called Deep-Learning because we can take this to the next level by having many hidden layers(like neurons) and then we will inter-connect them just like the neurons in the human brain





Following are the key-concepts of ANN :

- Nodes
- Activation Function
- How do Neural Networks work
- How do Neural Networks learn
- Gradient Descent
- Backpropagation

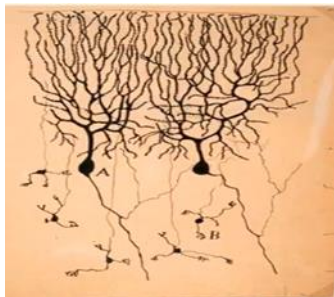
Neurons : The basic building block of an ANN



- A neuron is the basic building block of your Artificial Neural Networks



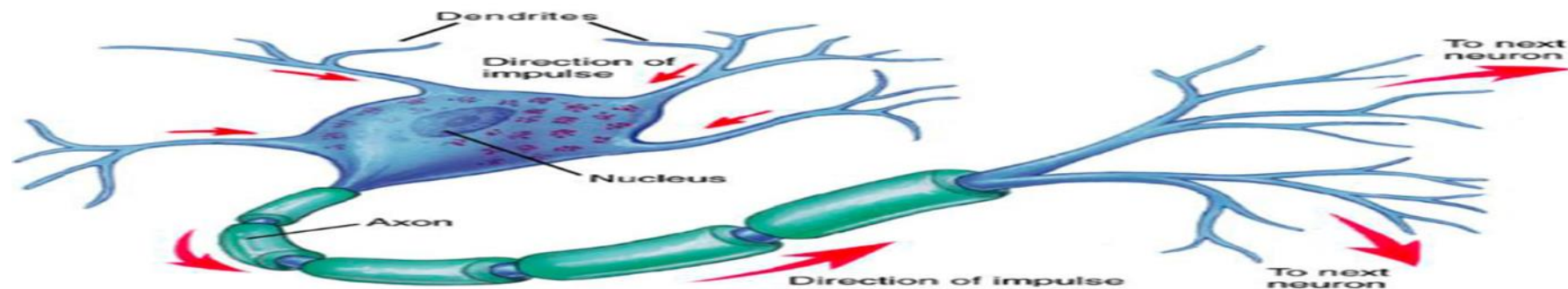
- It is body-structure with lots of different tails/branches
- Image of a neuron created by a Spanish neuroscientist Santiago Ramon y Cajal 1899



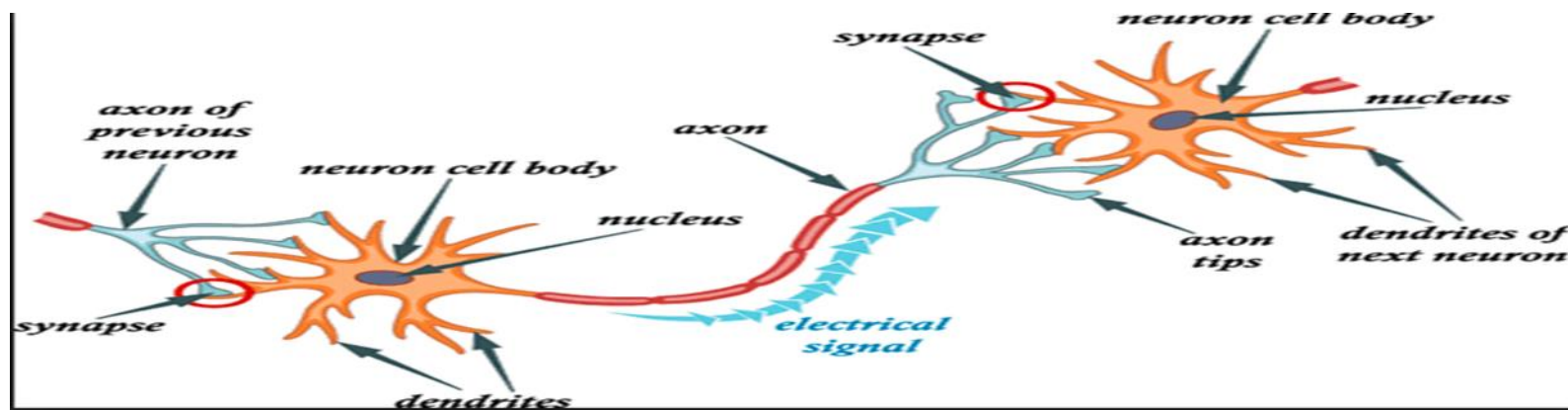
- An important task in creating an **Artificial Neural Network(ANN)** would be to create a layer of Neurons.



- Typical neuron-structure in the human-brain
- Lots of such neurons are connected to build or form a strong powerful unit

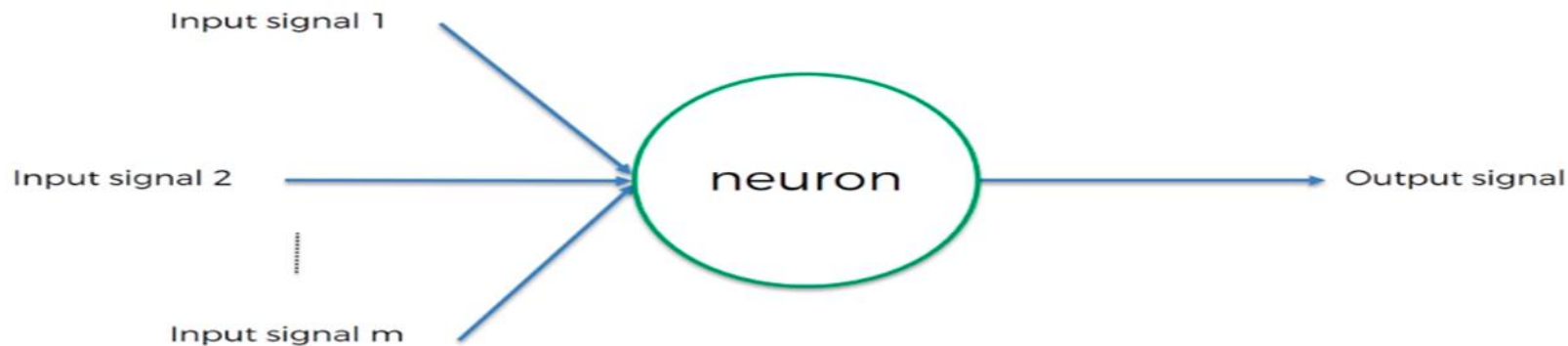


- DENDRITES, AXONS and the NUCLEUS of a neuron

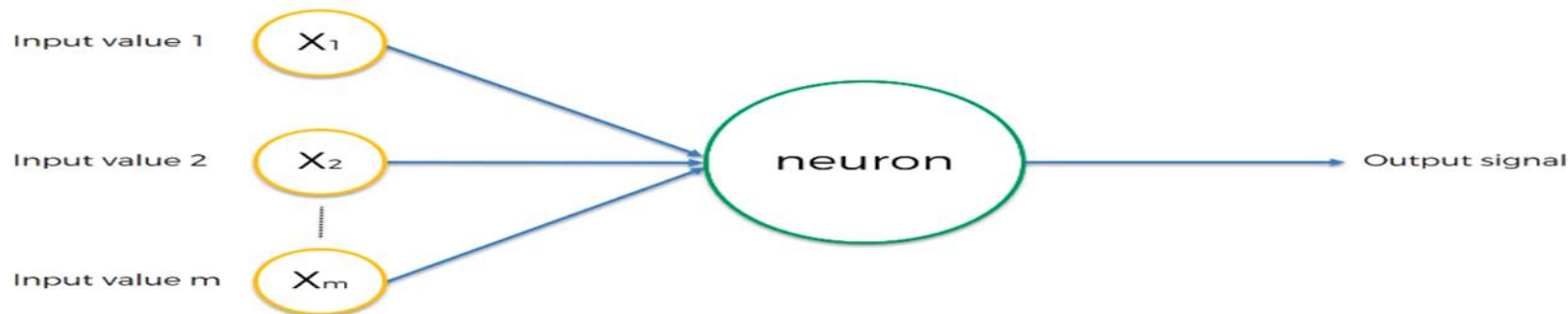




- Neurons are known as Nodes in an ANN



- Signals or Inputs will come from some other neurons or nodes.

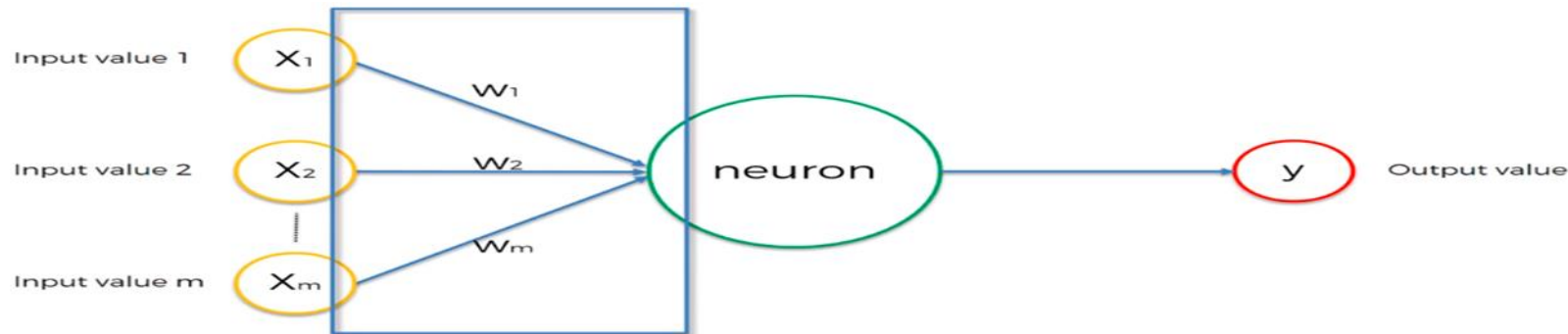




- The impulses are your input-values or independent variables



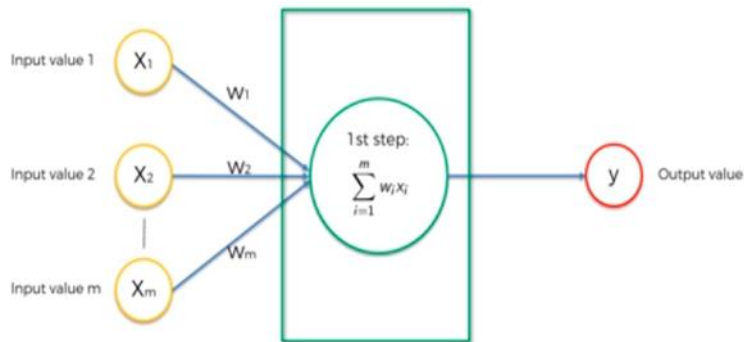
- Input-layer are the IVs from a single row or observation from your dataset
- Output value/values can be Continuous-values, Binary-values or Categorical-value
- Weights are assigned to the inputs and the weighted-inputs are added-up



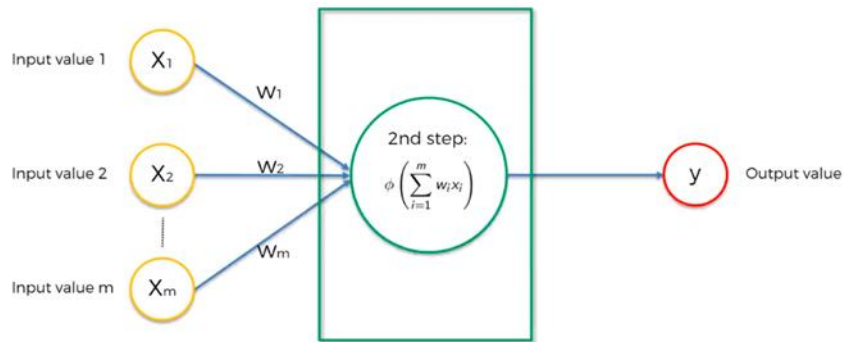


- The activation-function is applied to the sum of the weighted input-values

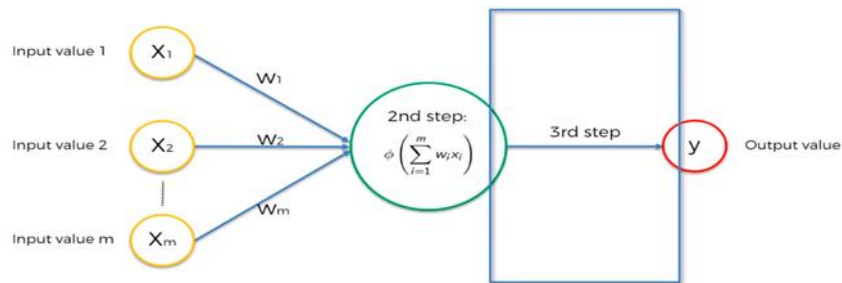
Step 1 : Sum-up the weighted input-values



Step 2 : Apply the activation function to the summed-up weighted input values



Step 3 : Pass on the signal to the next layer of neurons

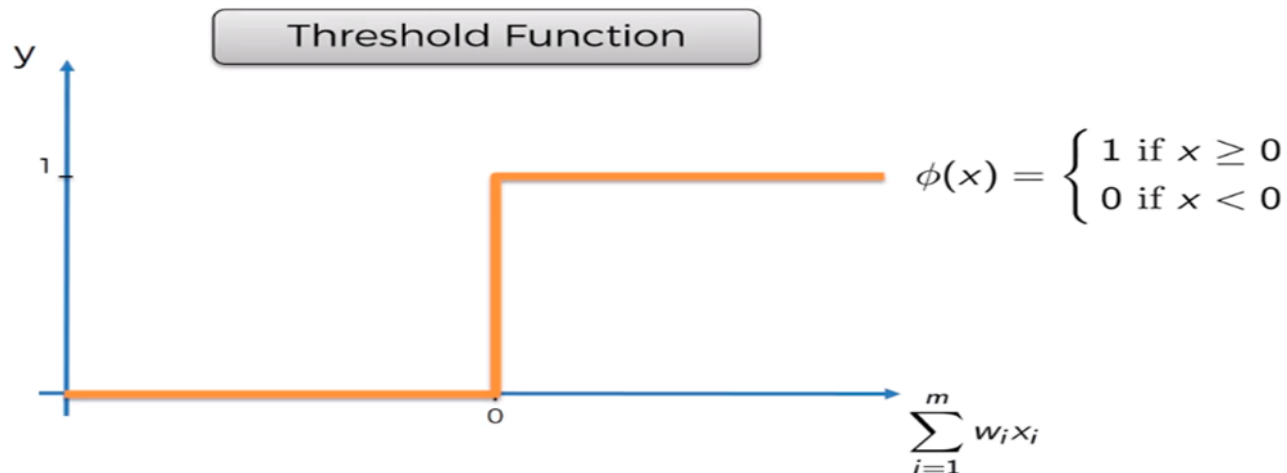




- The activation-function is the logic applied by a neuron to the weighted sum of input-values
- Their main purpose is to convert the input-signals of a node in a ANN to an output-signal.
- An ANN without an activation-function would be a Linear Regression Model
- Activation functions are needed to understand complex, high-dimensional, non-linear data like images, videos , audio , speech
- Some activation functions :
 - Threshold
 - Sigmoid
 - Rectifier
 - Hyperbolic Tangent



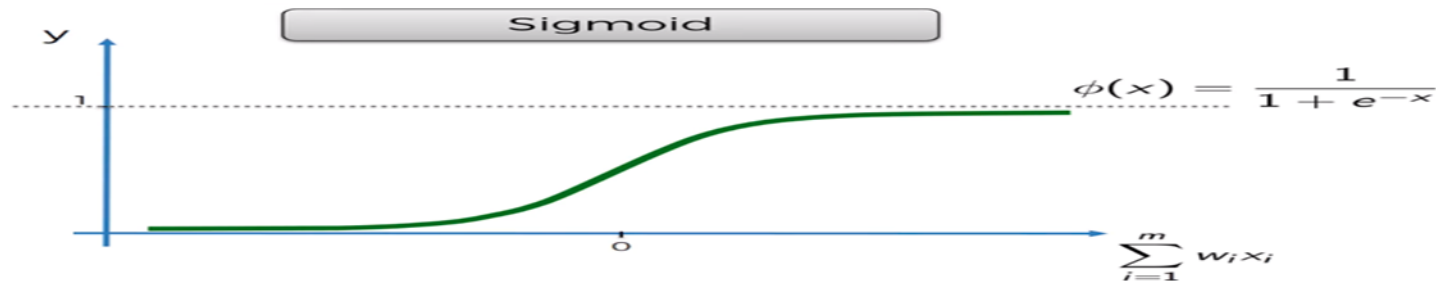
- Threshold Function is good for Yes/No type of values



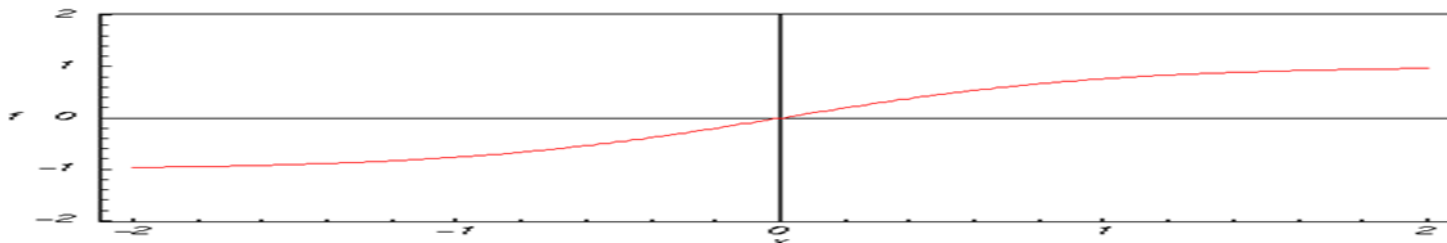
- After applying the Threshold function on the weighted sum of inputs, if the value returned is less than 0, then it returns 0, and if the value returned is 0 or more, then it returns 1



- **Sigmoid Function** is mostly used in the output-layer of an ANN for classification type of problems

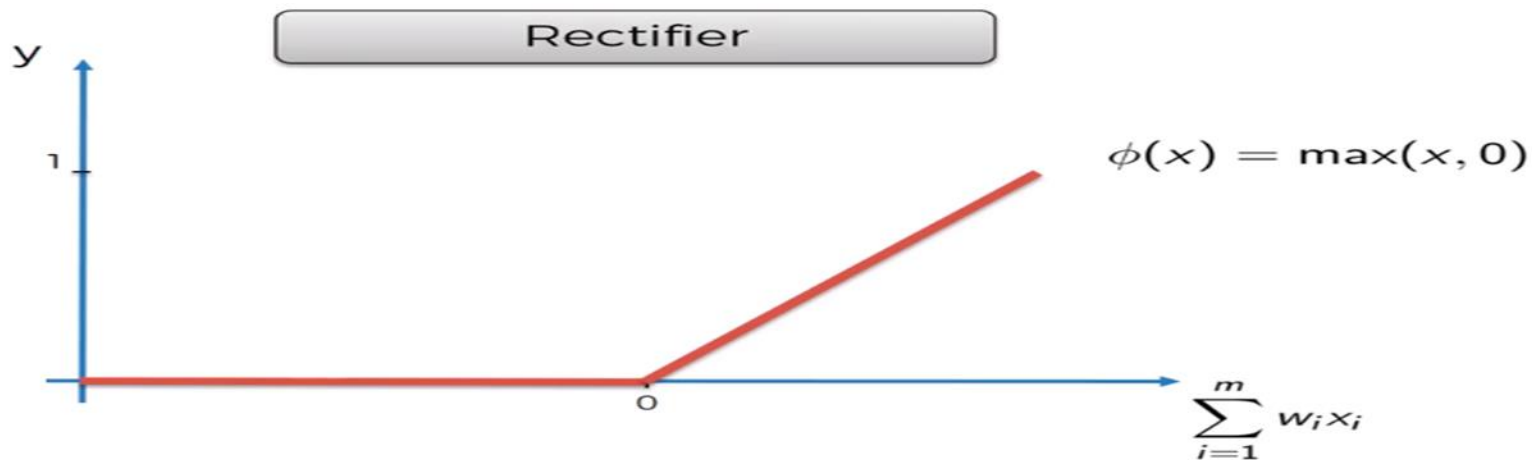


- Smoother than the Threshold function.
- Output is in the range 0 to 1
- **Hyperbolic Tangent Function** ($f(x) = \tanh(x)$) has values in the range from **0 to -1 on the negative-side** and **0 to +1 on the positive-side**.
- Stronger Gradient





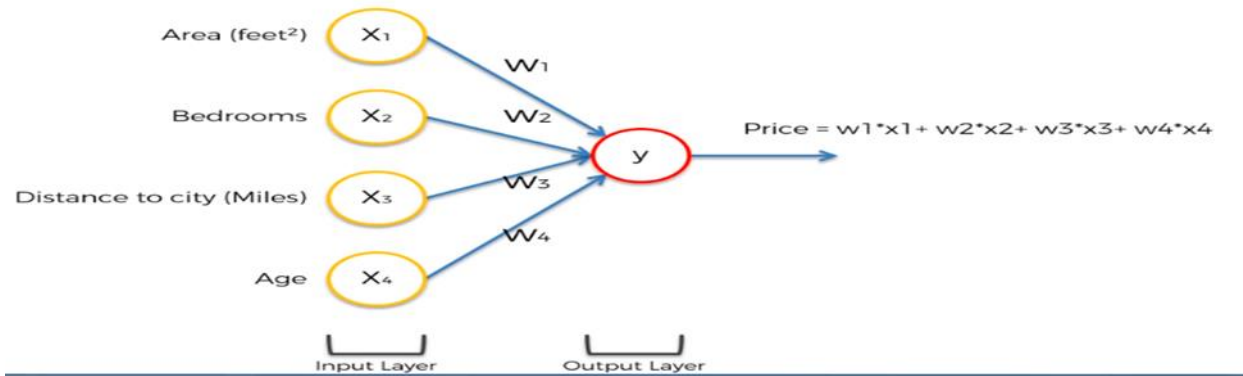
- Rectifier Function is mostly used function in ANNs



- It gradually and linearly progresses from 0 to a positive-value as the weighted sum input-values increases.
- Example : It will return a value 0 upto positive-value depending upon how much more is the experience of a person above say 10 years



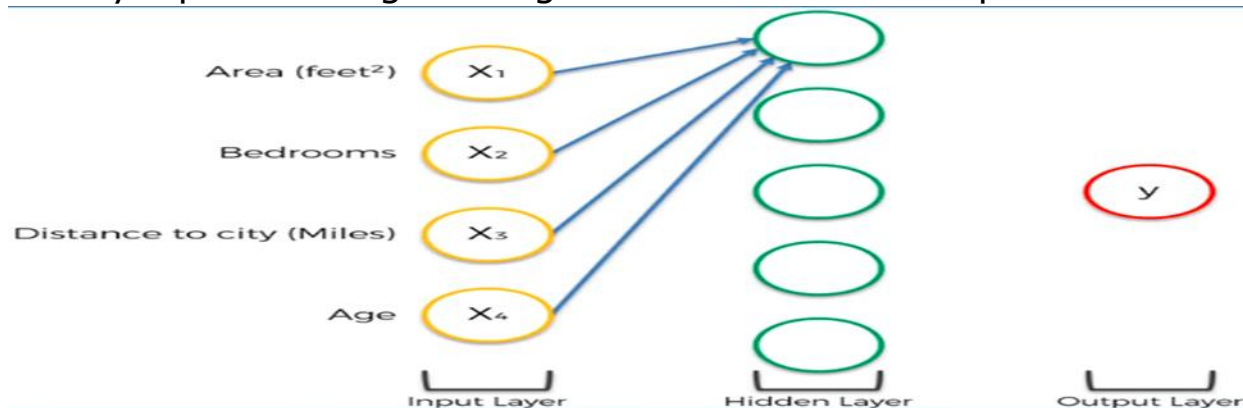
- Consider a trained-system or model which should predict the cost of a property based on 4 input parameters like Area, Number of Bedrooms, Distance from the city and age of the property.



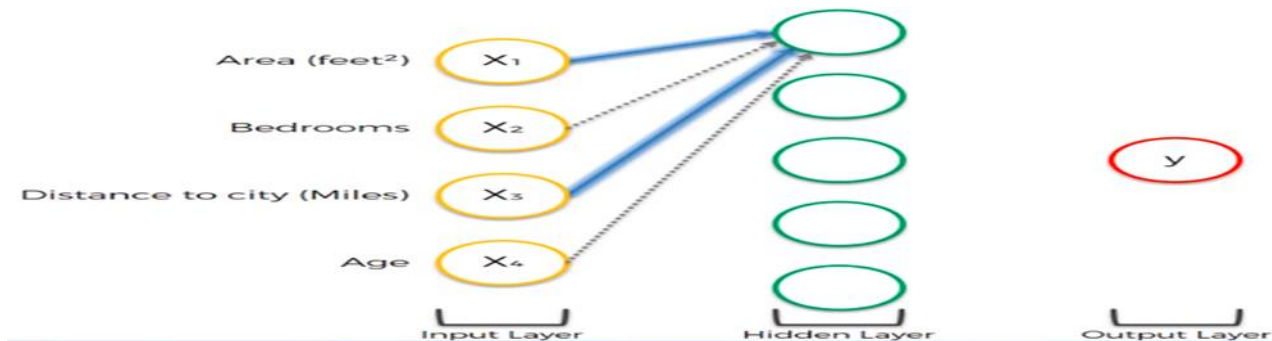
- In a very basic form, a neural network can have an input-layer and directly an output-layer without any hidden layers
- $\text{Price} = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4$
- In comparison to Machine-Learning, neural networks we will have the flexibility and power to increase the accuracy of the final-output through its hidden-layers



- The synapse would give weights to the individual input values of a row

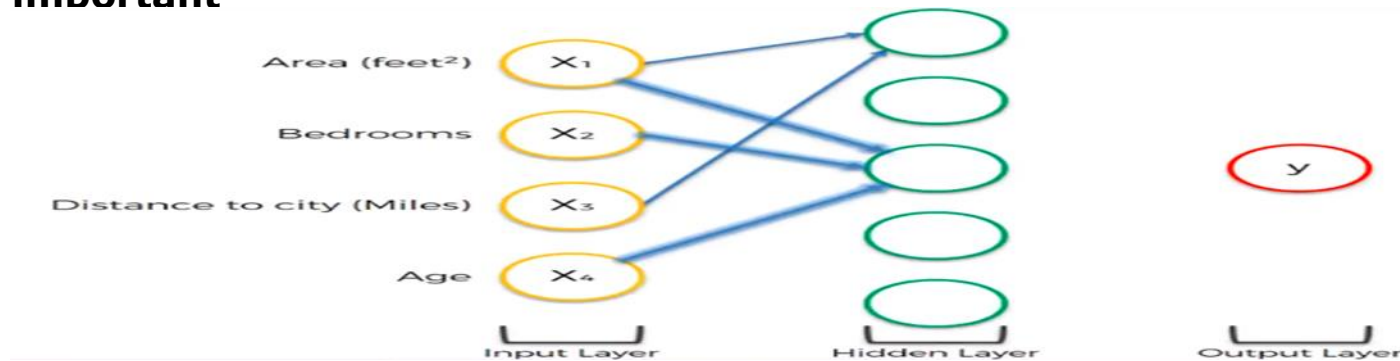


- Based on the data studied in the learning-phase, assume the **1st neuron** judged that the **Area-value and the value of Distance-from-the-city are important**

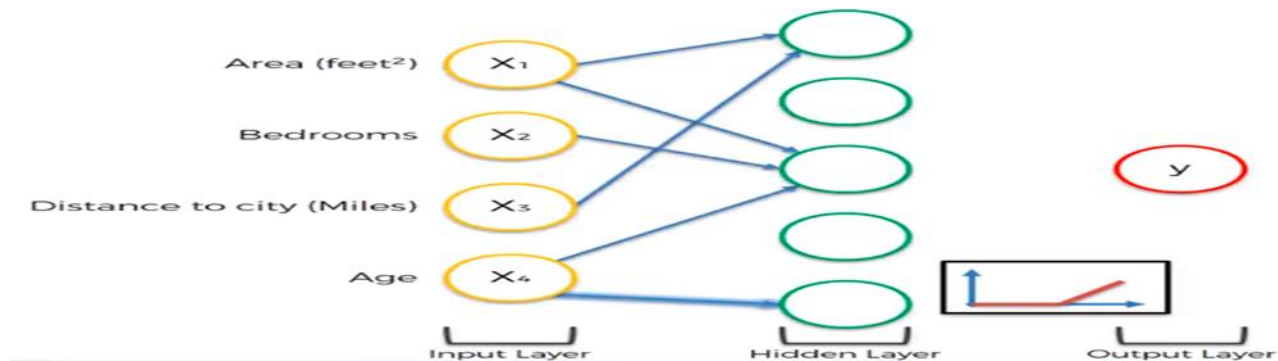




- The **3rd neuron** took **Area, number-of-bedrooms** and the **Age of the property** as **important**

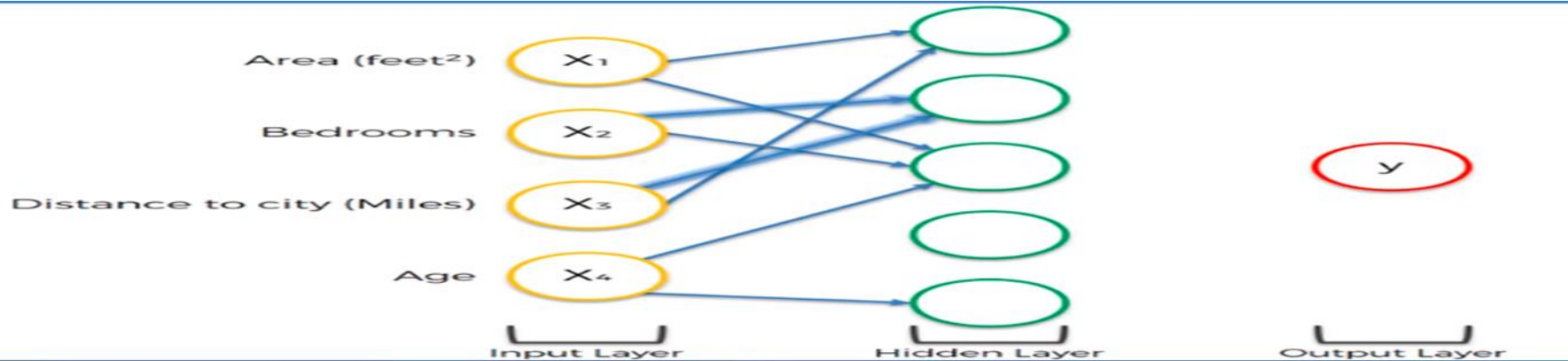


- The **5th neuron** only focused on the **Age of the property**

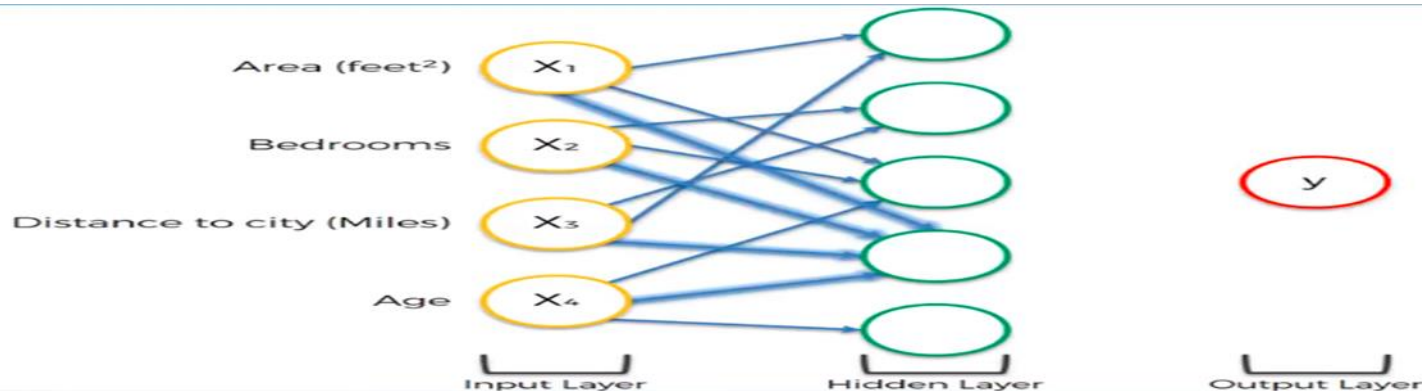




- The 2nd neuron picked up some logic was based on only the number-of-bedrooms and Distance-from-the-city

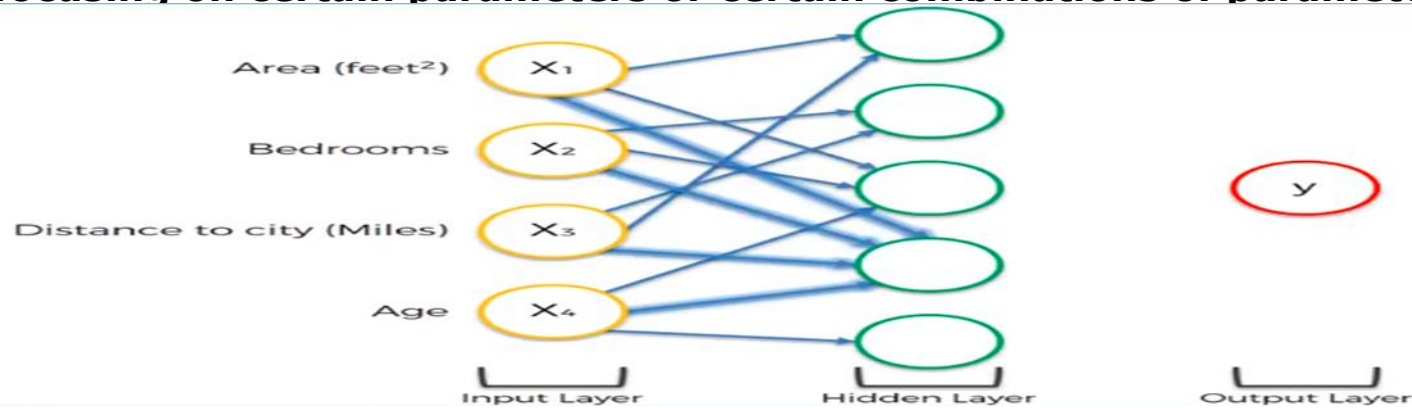


- The 4th neuron picked up some logic which was based on all the 4 parameters.

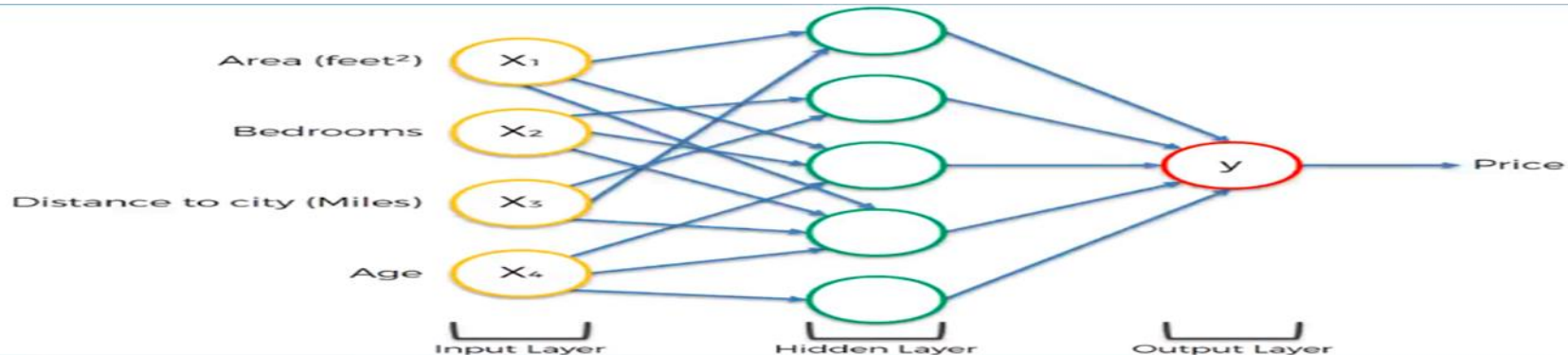




- Thus, the ANN improves the power and flexibility of this system by having nodes focusing on certain parameters or certain combinations of parameters

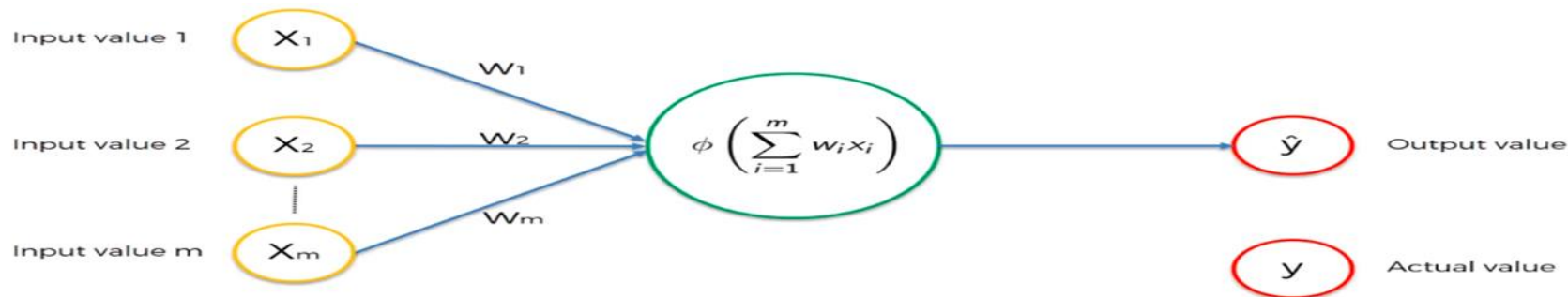


- Individual neurons may not give us the best predicted price



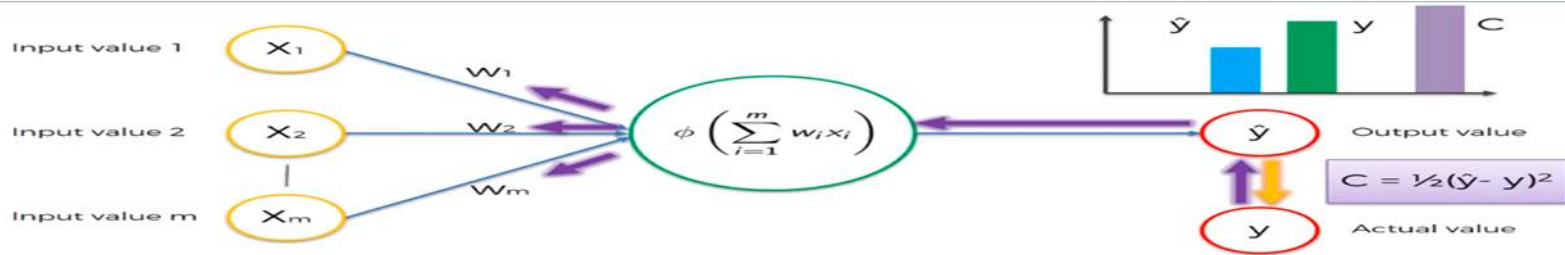


- Through an ANN, you create a facility for the program to be able to understand what it needs to do on its own
- You provide the ANN with the functionalities/algorithms which it should use
- The functionalities of that ANN-architecture will deduce a near-to-perfect business-logic from the trends in that labelled-data
- A **Perceptron is a single layer neural network** and a **multi-layer perceptron is called Neural Network**.
- A perceptron consists of : Input Values, Weights, Sum of the weighted inputs and an Activation-function

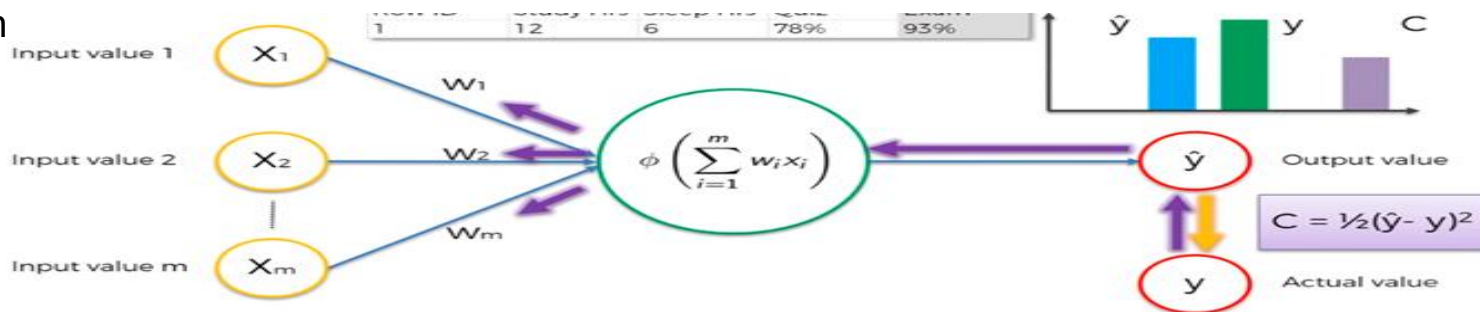




- We can compare the \hat{y} with y and use a cost-function to calculate the difference



- Based upon the value calculated by the cost-function, information is fed back into the neural network and then the weights get adjusted
- After adjusting the weights, the values X_1 , X_2 and X_n of that single-row are again fed back into the system



- This happens iteratively until we get a value from cost-function which is 0 or near to 0



- An **epoch** is a complete pass through a given dataset
- Assume you have a set of 8-rows. For each row, we get \hat{y}



- The \hat{y} s are compared with their respective y s





- Calculate the cost-function

$$C = \sum \frac{1}{2}(\hat{y} - y)^2$$

- Pass the information back-into the neural network to update the weights

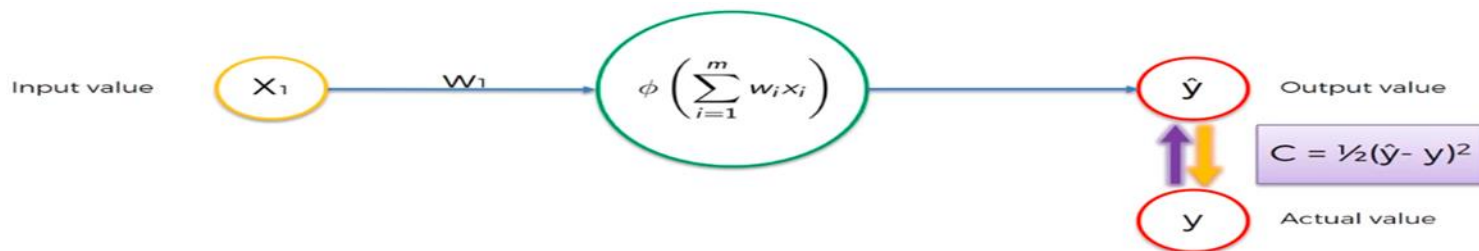


- Here one epoch is over
- Repeat this until you get the least-value from the cost-function
- This whole process is called **back-propagation**

Brute-force method to calculate optimal weight



- Single-layered single-column-input ANN



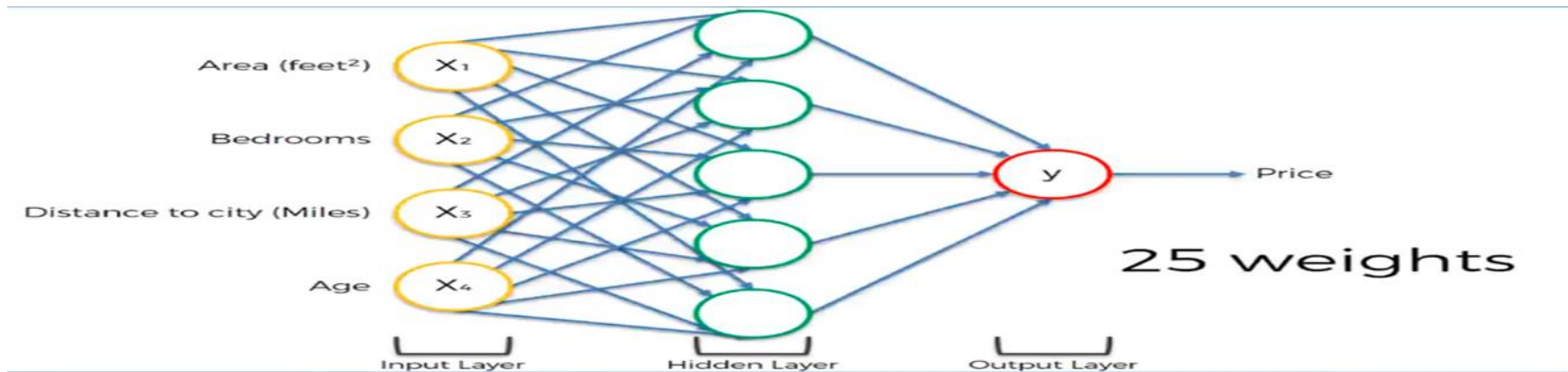
- In Brute-Force approach, we look different values for the weights to find the most optimal weight
- We can have a chart to find the best value for the weight



- Brute-force method is good for single weight and faces problems of dimensionality with multiple weights



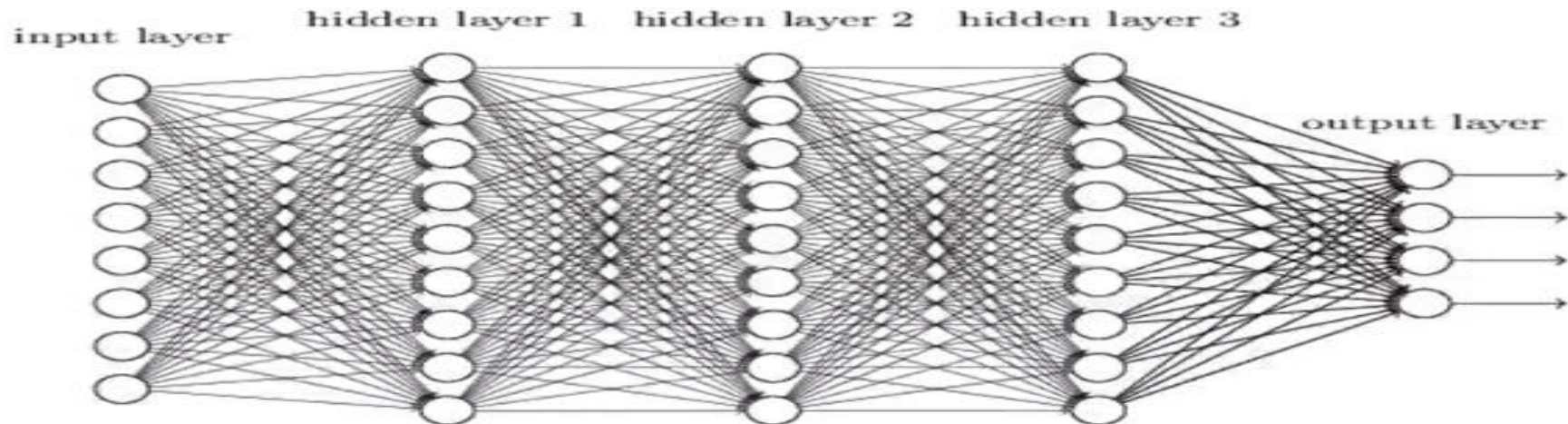
- Assuming 4 columnar-data to predict the value of a property



- We will have $4 \times 5 = 20$ weights from the input-layer to the hidden-layer. Plus 5 weights from the hidden-layer to the output-layer. So, 25 weights
- Assuming we are planning to try 1000 values across each of the 25 weights through Brute-force method. It will be 1000 raised to 25 combinations of weights

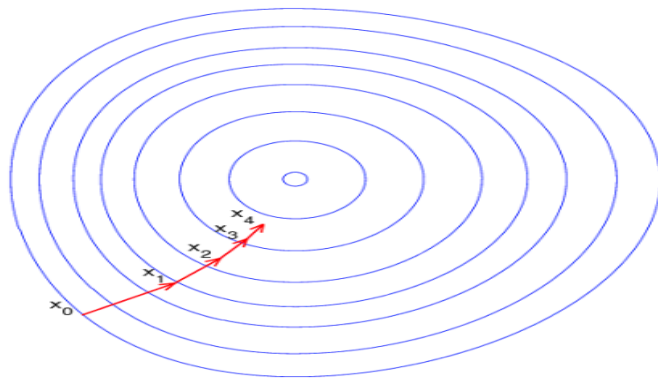


- Sunway TaihuLight, one of the fastest super-computers, operates at the speed of 93 PFLOPS(FLoating-point Operations Per Second). PFLOPS = 93×10^{15} operations per second
- With Brute-force method, for 1000^{25} combinations or 10^{75} combinations, this super-computer will need 10^{75} combinations / 93×10^{15} seconds, which is 1.08×10^{58} seconds, which is 3.42×10^{50} years
- Brute-force method of trying combinations of weights for a multi-layered multi-parametered(IVs) is not suitable





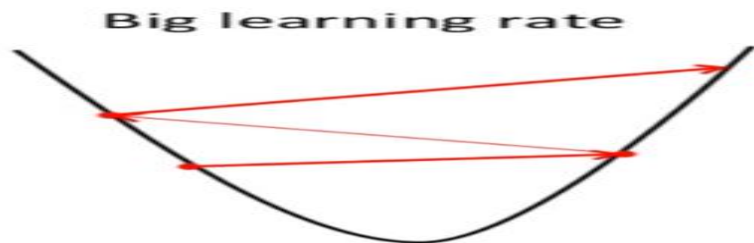
- Gradient Descent method is an optimization method to decide the optimal values of the weights to minimize the value given by cost-function until you reach the minimum
- Initial values for weights are set and then Gradient Descent iteratively adjusts the values
- It measures how much the output of the cost-function changes if you change the weights
- Imagine a man wants to climb a hill in the fewest possible steps. Initially, he just starts climbing the hill by taking really big steps in the steepest direction, as long as he is not close to the top



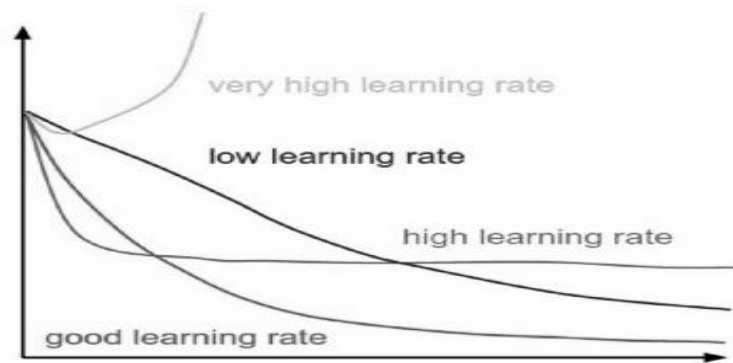
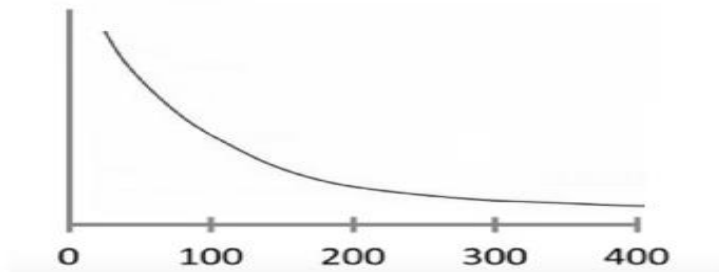
- It is called as Gradient **DESCENT** because it is aimed at reducing the output of the cost-function



- The Learning-rate determines how fast or slow we will move towards the optimal weights
- The Learning-rate needs to be an appropriate value

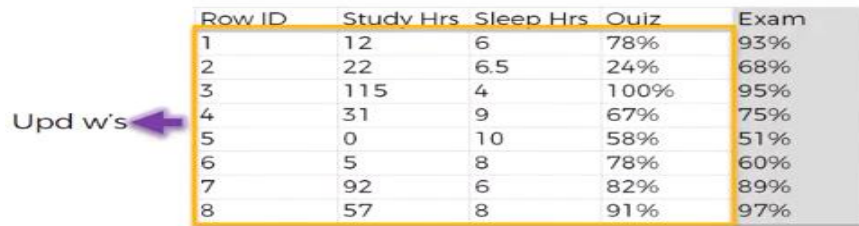


- To ensure a good Learning-rate you may plot a graph with the number of epochs on the x-axis and the cost-function on the y-axis.

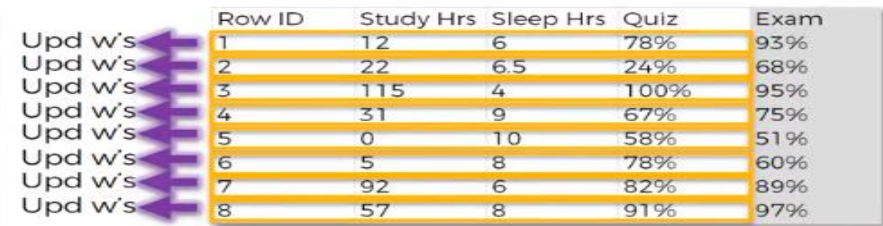




- Gradient Descent method is a useful method when you have multiple IVs or parameters and/or multi-layered neurons
- Gradient Descent can be of 3 major types : Batch Gradient Descent, Stochastic Gradient Descent and Mini Batch Gradient Descent
- In **Batch Gradient Descent** also known as Vanilla Gradient Descent the weights are adjusted only after all the rows are completed in an epoch from the training
- In **Stochastic Gradient Descent** the error is calculated per row and the weights get updated for every row



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

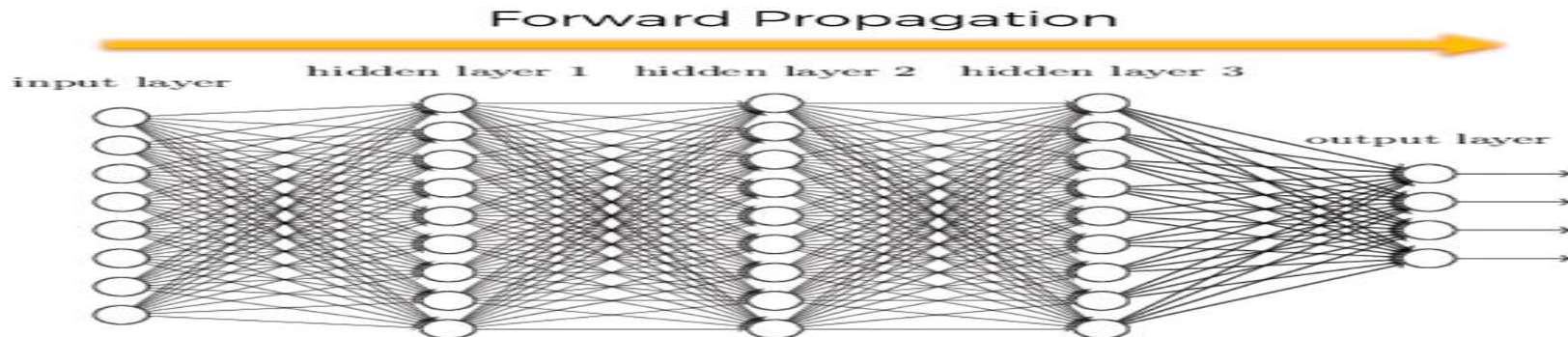
- In **Mini Batch Gradient Descent**, the training dataset is split into small batches and an update of weights is performed for each of these batches



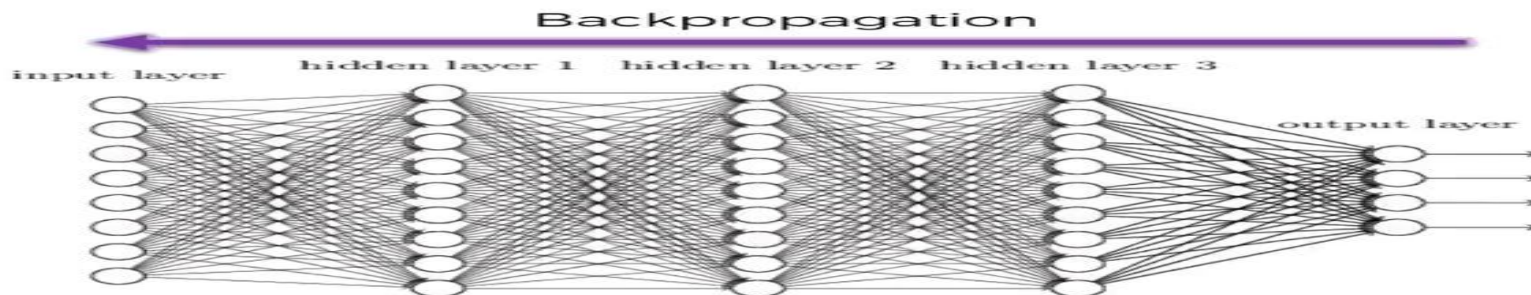
- Mini Batch Gradient Descent is a combination of Batch Gradient Descent and Stochastic Gradient Descent
- It creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.



- Data moving from the input-layer and being propagated through the various hidden-layers to give us the predicted value is called as Forward Propagation



- Feeding the errors calculated by the cost-function back into the system to adjust the weights is called Back Propagation



- Backward propagation is an mathematical algorithm to re-adjust the weights



- Step 1 : Randomly initialize the weights to small numbers close to zero but not zero
- Step 2 :Take each row of your dataset, and treat each column as an input-node to pass the training-data to the layers of your neural network
- Step 3(Forward Propagation) : The neurons are activated based upon how the activation-function is working on the sum of the weighted-inputs
- Step 4: Compare the actual value with the predicted value
- Step 5(Back Propagation) : Update the weights according to how much they are responsible for the error.
- Step 6 : Repeat Steps 2 to 5 by re-adjusting the weights for every mini-batch
- Step 7 : Once an epoch is over, redo more epochs



Thank You