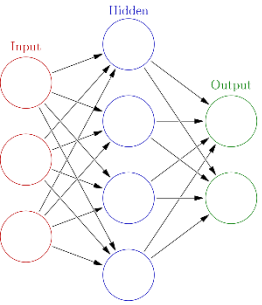
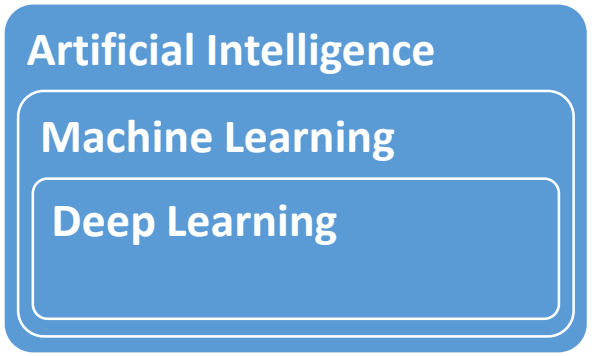
Deep Learning

Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data

Dr. Sarwan Singh

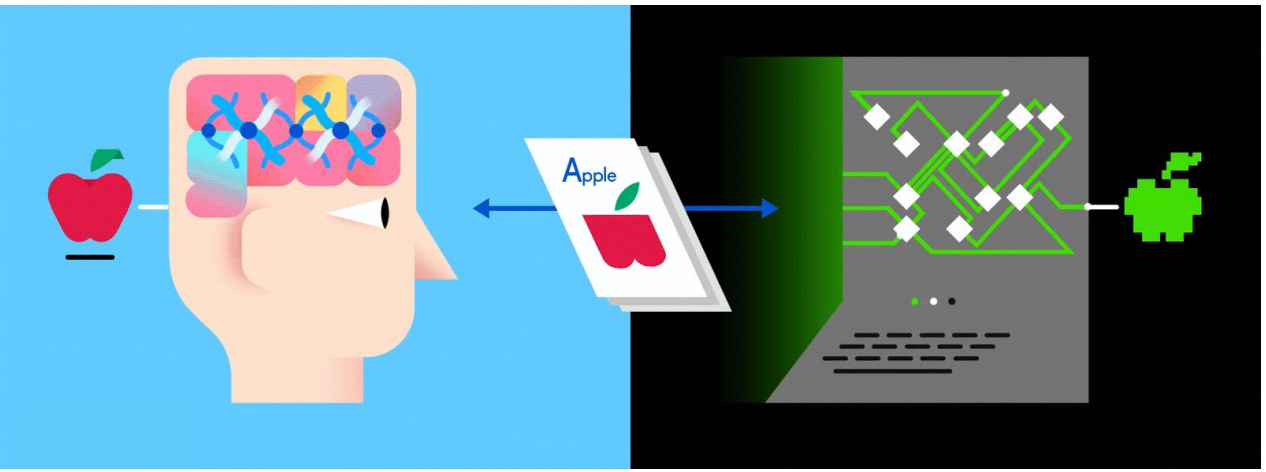
NIELIT Chandigarh

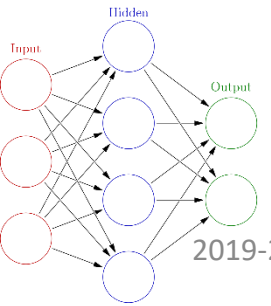




agenda

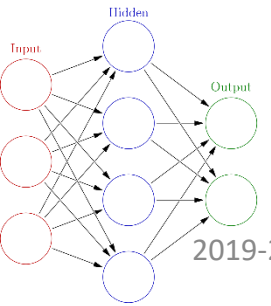
- Perceptron- an artificial neuron
- Working of perceptron
- Biological overview – neuron
- Layers
- Activation function
- Loss function





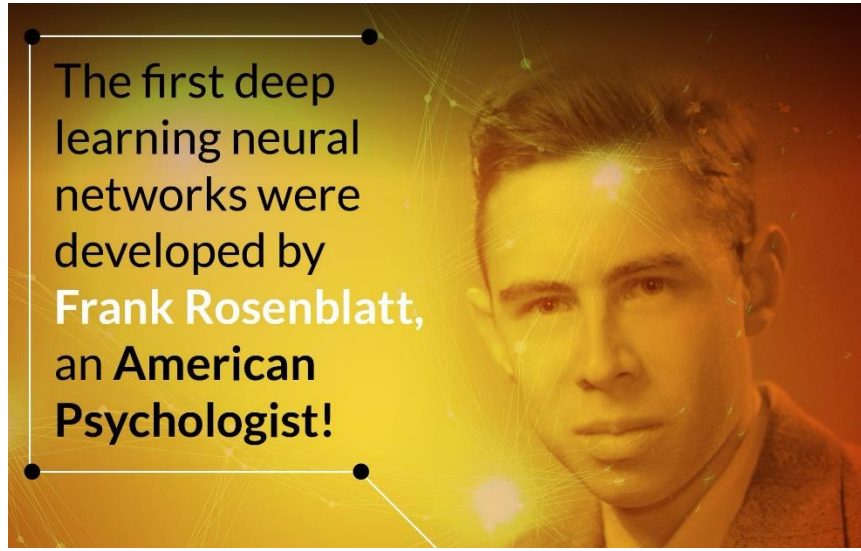
References

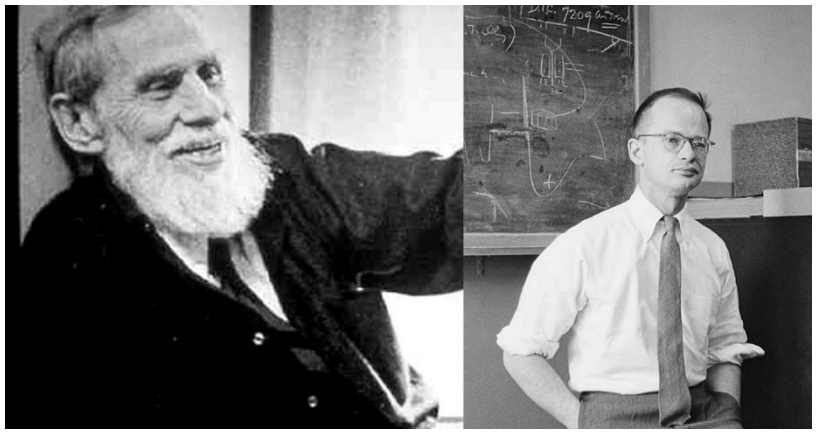
- Medium.com - Blockchain
- neuralnetworksanddeeplearning.com
- Fundamentals of Deep Learning - *Designing Next-Generation Machine Intelligence Algorithms* … Nikhil B
- Wikipedia

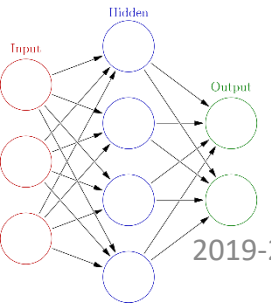


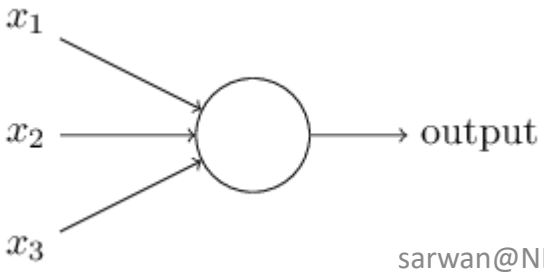
Perceptron … an artificial neuron

- Perceptrons were developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts.
- The main neuron model used is one called the *sigmoid neuron.*
- A perceptron takes several binary inputs, $x_1, x_2, \ldots$ x1,x2,…, and produces a single binary output





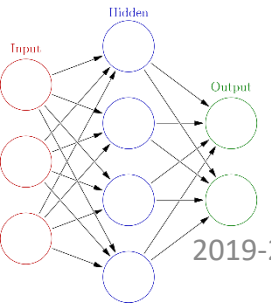




Working of perceptron

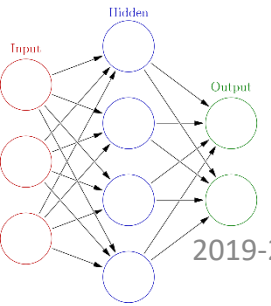
- Rosenblatt proposed a simple rule to compute the output.
- He introduced *weights*, $w_1, w_2, \ldots w1, w2, \ldots$, real numbers expressing the importance of the respective inputs to the output.
- The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum_j w_j x_j$ is less than or greater than some threshold value.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$



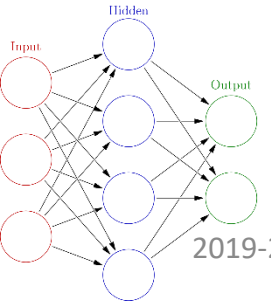
Neural Network

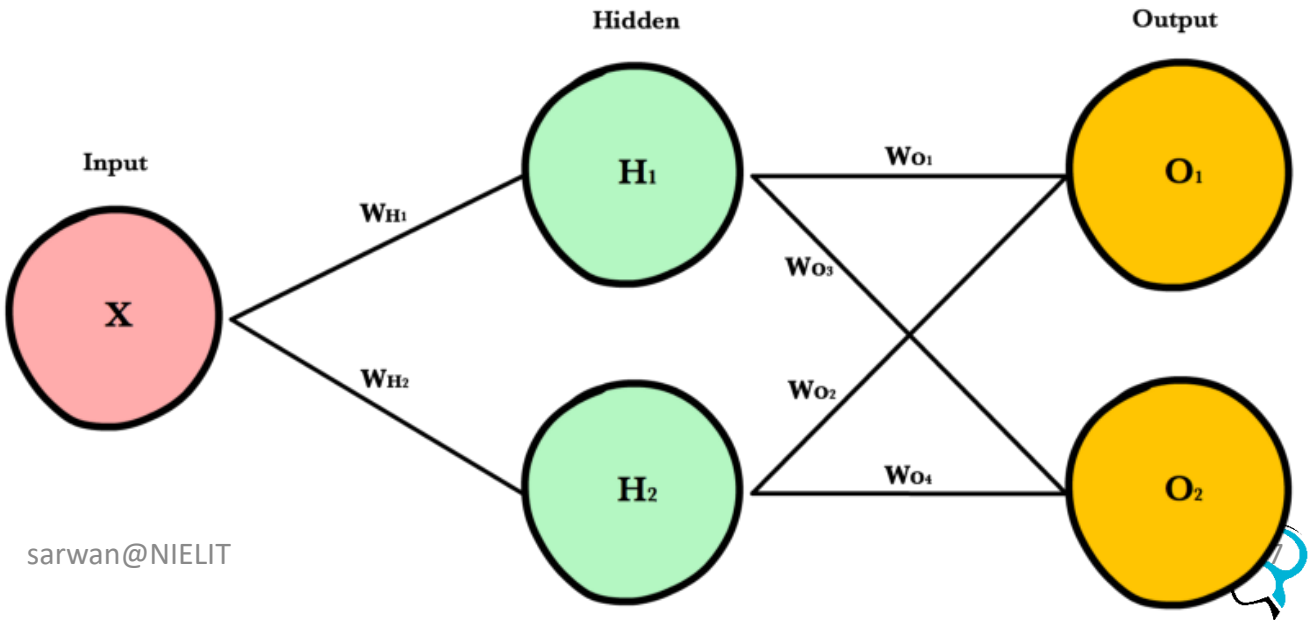
- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data
- Deep learning, a powerful set of techniques for learning in neural networks
- Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing



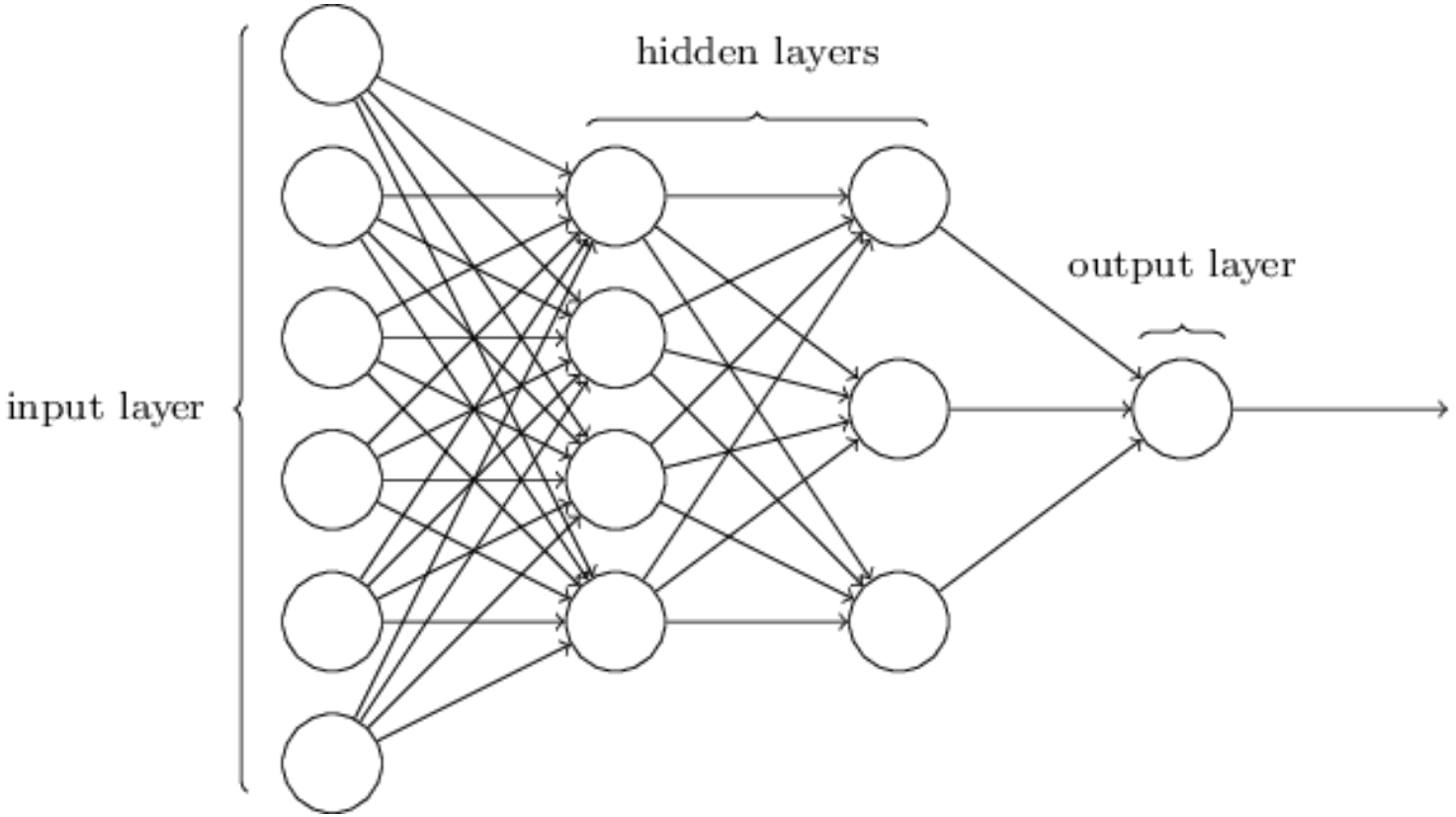
Neural Network

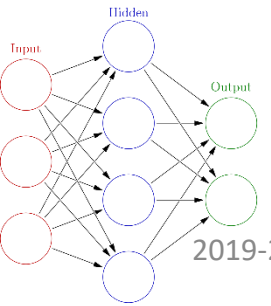
- Neural networks are a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions.
- A neural network takes an input, passes it through multiple layers of hidden neurons (mini-functions with unique coefficients that must be learned), and outputs a prediction representing the combined input of all the neurons.





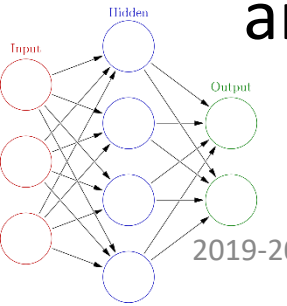
multilayer perceptron or *MLPs*





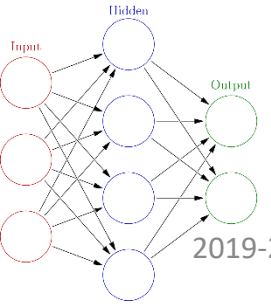
Neural Network

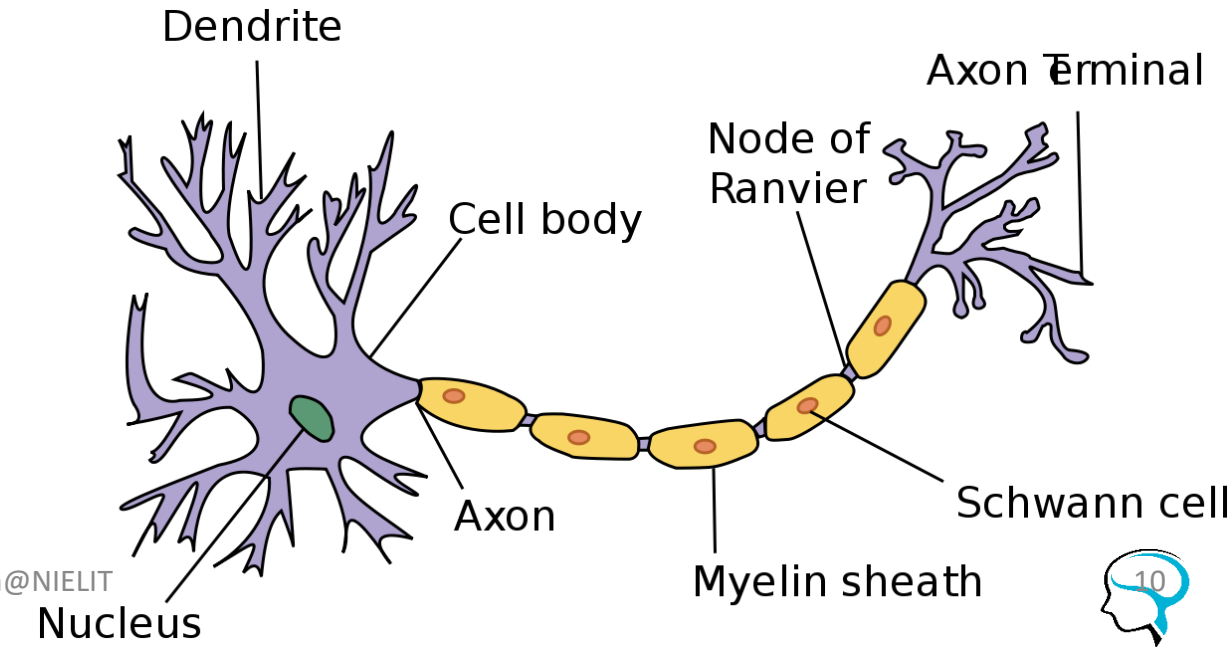
- Neural networks are trained iteratively using optimization techniques like gradient descent.
- After each cycle of training, an error metric is calculated based on the difference between prediction and target.
- The derivatives of this error metric are calculated and propagated back through the network using a technique called backpropagation.
- Each neuron’s coefficients (weights) are then adjusted relative to how much they contributed to the total error.
- This process is repeated iteratively until the network error drops below an acceptable threshold.



Neuron….*biological definition*

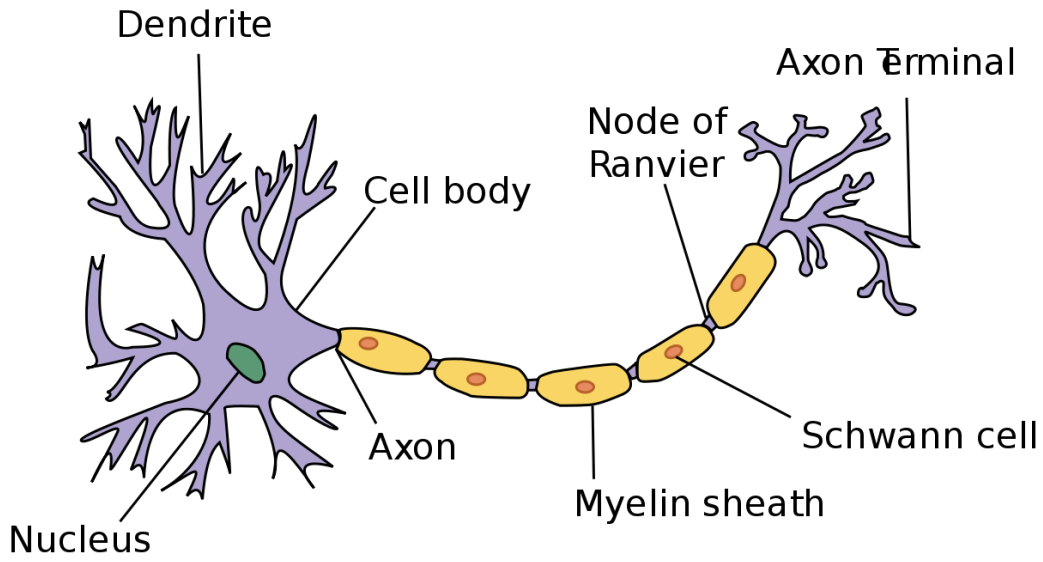
- A neuron is a nerve cell that carries electrical impulses.
- Neurons are the basic units of our nervous system.
- There are about 86 billion neurons in the human brain, which is about 10% of all brain cells.
- The human brain has about 16 billion neurons in the cerebral cortex.

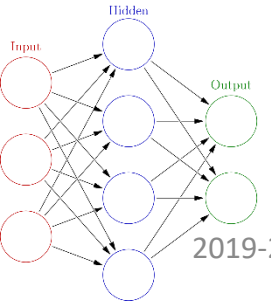




Neuron….*biological definition*

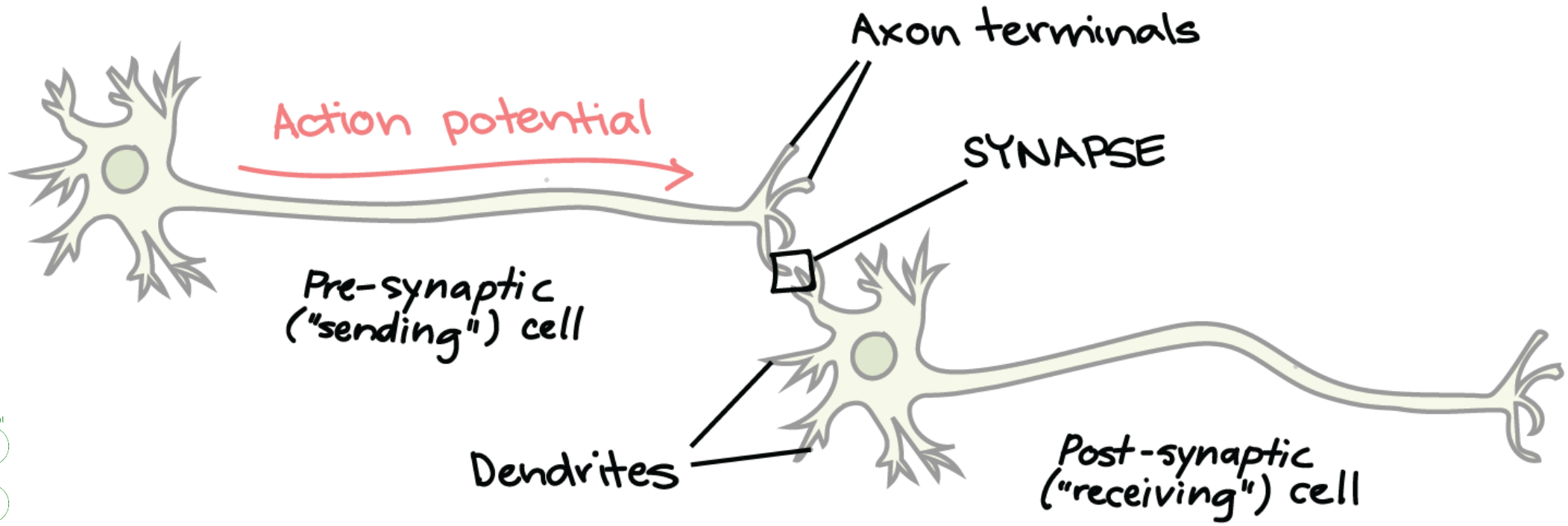
- Neurons have a cell body (soma or cyton), dendrites and an axon
- The neurons are supported by glial cells and astrocytes.
- Neurons are connected to one another, but they do not actually touch each other. Instead they have tiny gaps called synapses.
- These gaps are chemical synapses or electrical synapses which pass the signal from one neuron to the next.

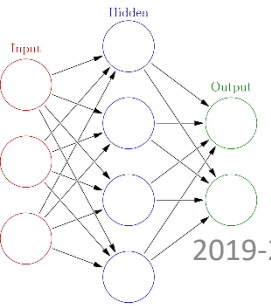




Neuron Synapse

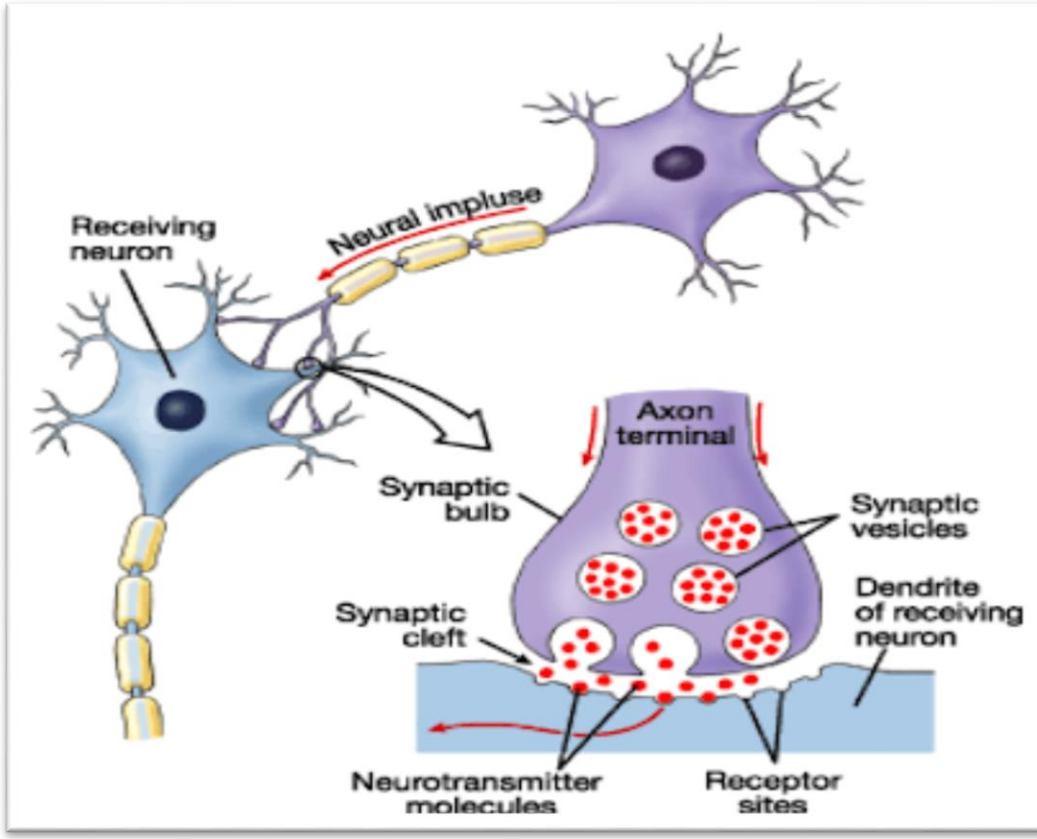
- Synapses are like roads in a neural network. They connect inputs to neurons, neurons to neurons, and neurons to outputs.

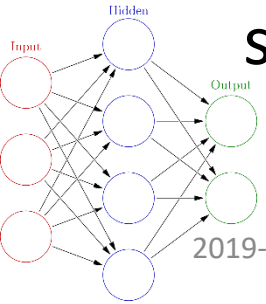


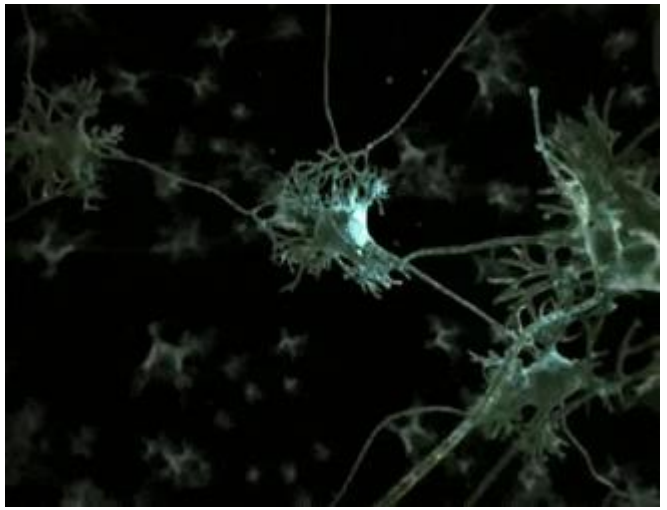


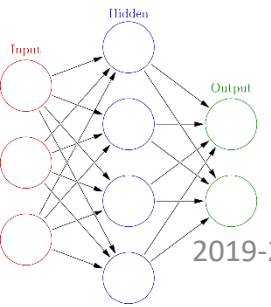
Neuron Synapse

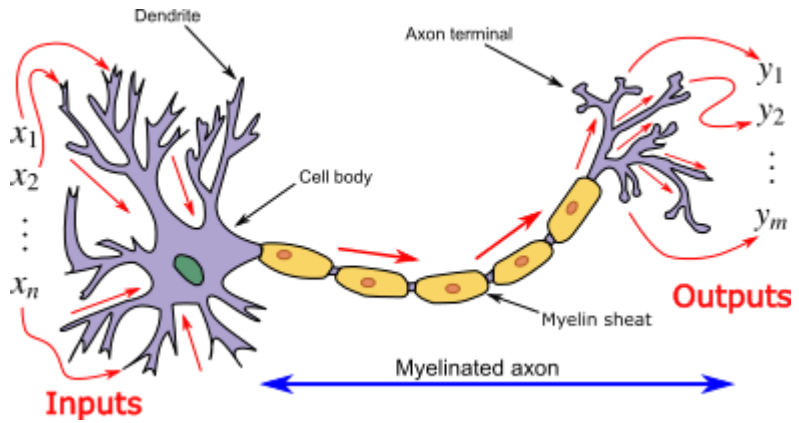
- In order to get from one neuron to another, you have to travel along the synapse paying the “toll” (weight) along the way.
- Each connection between two neurons has a unique synapse with a unique weight attached to it.
- When we talk about updating weights in a network, we’re really talking about adjusting the weights on these synapses.





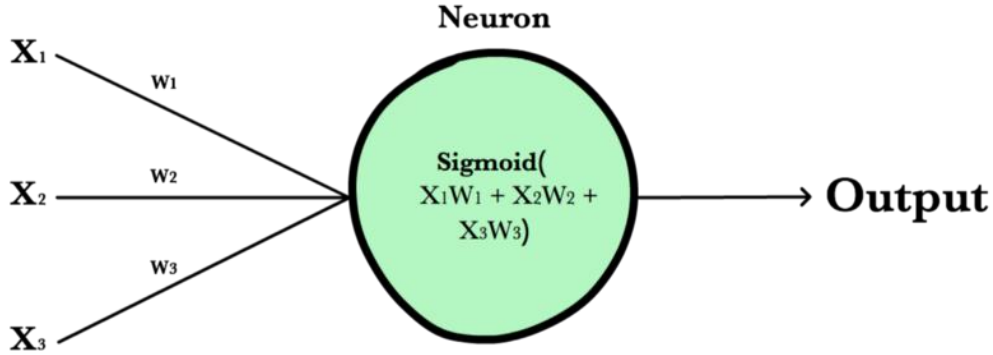


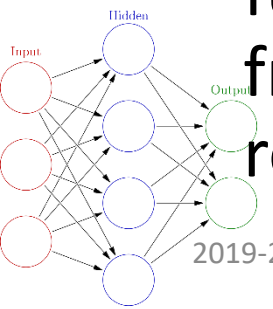

Hidden
Input
Output


Dendrite
Axon terminal
x_1
x_2
x_n
Cell body
y_1
y_2
y_m
Outputs
Myelin sheat
Myelinated axon
Inputs

Neuron

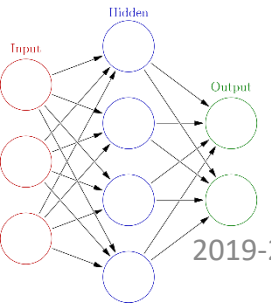
- A neuron takes a group of weighted inputs, applies an activation function, and returns an output.
- Inputs to a neuron can either be features from a training set or outputs from a previous layer's neurons.
- Weights are applied to the inputs as they travel along synapses to reach the neuron.
- The neuron then applies an activation function to the "sum of weighted inputs" from each incoming synapse and passes the result on to all the neurons in the next layer.





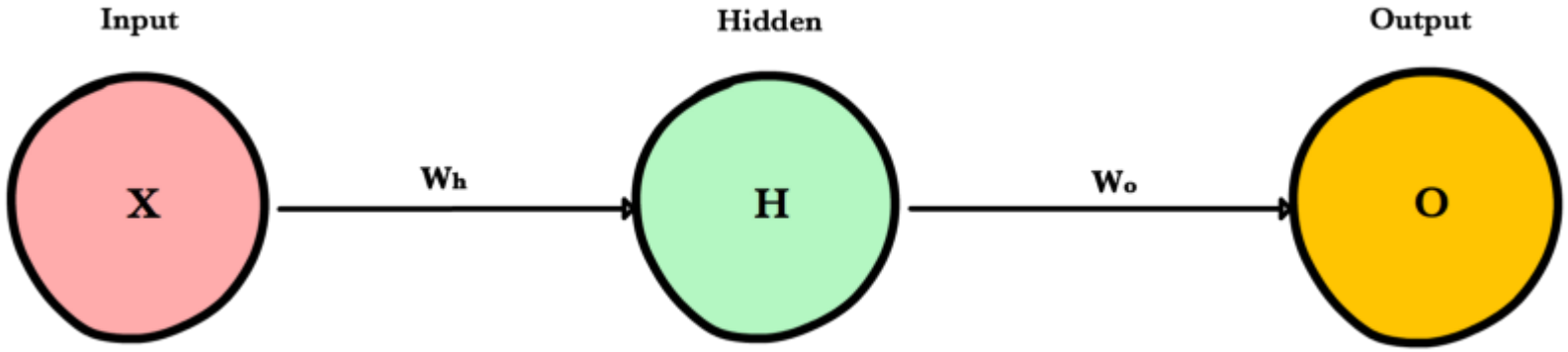
Weights…Bias

- Bias terms are additional constants attached to neurons and added to the weighted input before the activation function is applied.
- Bias terms help models represent patterns that do not necessarily pass through the origin.
- For example, if all your features were 0, would your output also be zero? Is it possible there is some base value upon which your features have an effect?
- Bias terms typically accompany weights and must also be learned by your model.



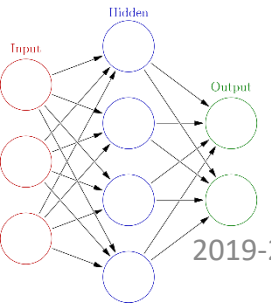
Layers

- Input Layer
- Hidden Layer
- Output Layer

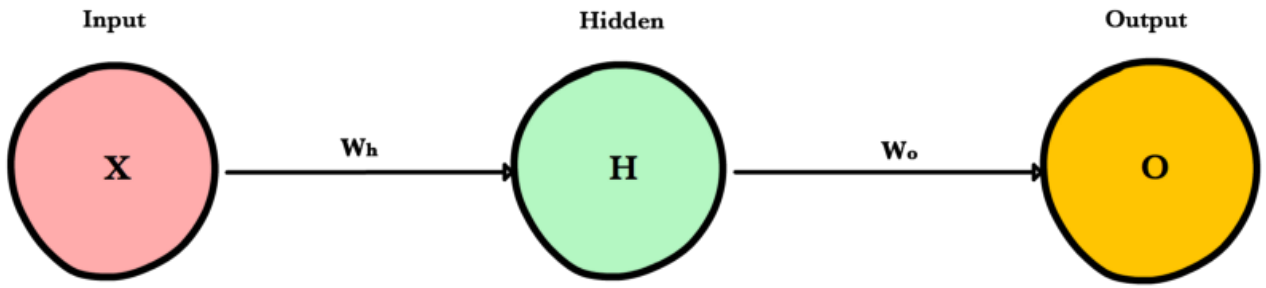


- **Input Layer**

Holds the data your model will train on. Each neuron in the input layer represents a unique attribute in your dataset (e.g. height, hair color, etc.).



Layers

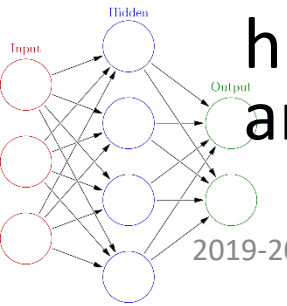


Hidden Layer

- Sits between the input and output layers and applies an activation function before passing on the results. There are often multiple hidden layers in a network. In traditional networks, hidden layers are typically fully-connected layers — each neuron receives input from all the previous layer's neurons and sends its output to every neuron in the next layer. This contrasts with how convolutional layers work where the neurons send their output to only some of the neurons in the next layer.

Output Layer

- The final layer in a network. It receives input from the previous hidden layer, optionally applies an activation function, and returns an output representing your model's prediction.



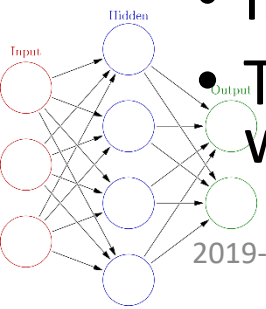
Weighted Input

- A neuron's input equals the sum of weighted outputs from all neurons in the previous layer. Each input is multiplied by the weight associated with the synapse connecting the input to the current neuron. If there are 3 inputs or neurons in the previous layer, each neuron in the current layer will have 3 distinct weights — one for each each synapse.
- Single Input

  Z = Input ·Weight = XW
- Multiple Input

$$Z = \sum_{i=1}^{n} x_i w_i$$
$$= x_1 w_1 + x_2 w_2 + x_3 w_3$$

- it's exactly the same equation we use with linear regression!
- In fact, a neural network with a single neuron is the same as linear regression!
- The only difference is the neural network post-processes the weighted input with an activation function.



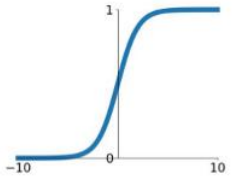
Activation Function

- Activation functions live inside neural network layers and modify the data they receive before passing it to the next layer.
- Activation functions give neural networks their power — allowing them to model complex non-linear relationships.
- By modifying inputs with non-linear functions neural networks can model highly complex relationships between features.

Popular activation functions

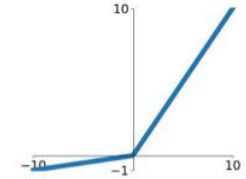
Sigmoid

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



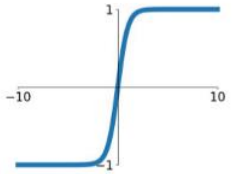
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

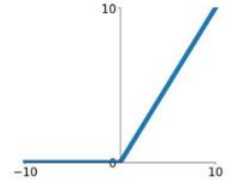


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

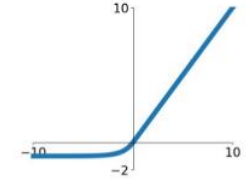
ReLU

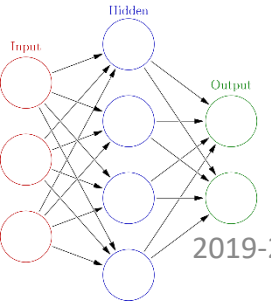
$\max(0, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$





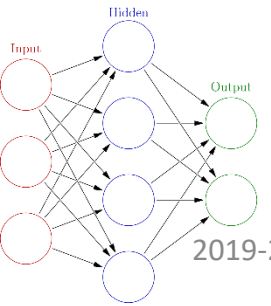
Why Activation function is needed

- The complex configuration of weights possessed by neural networks makes it difficult to solve the problem with activation function.
- The main reason lies in the concept of Non-Linearity

Linearity vs non-linearity

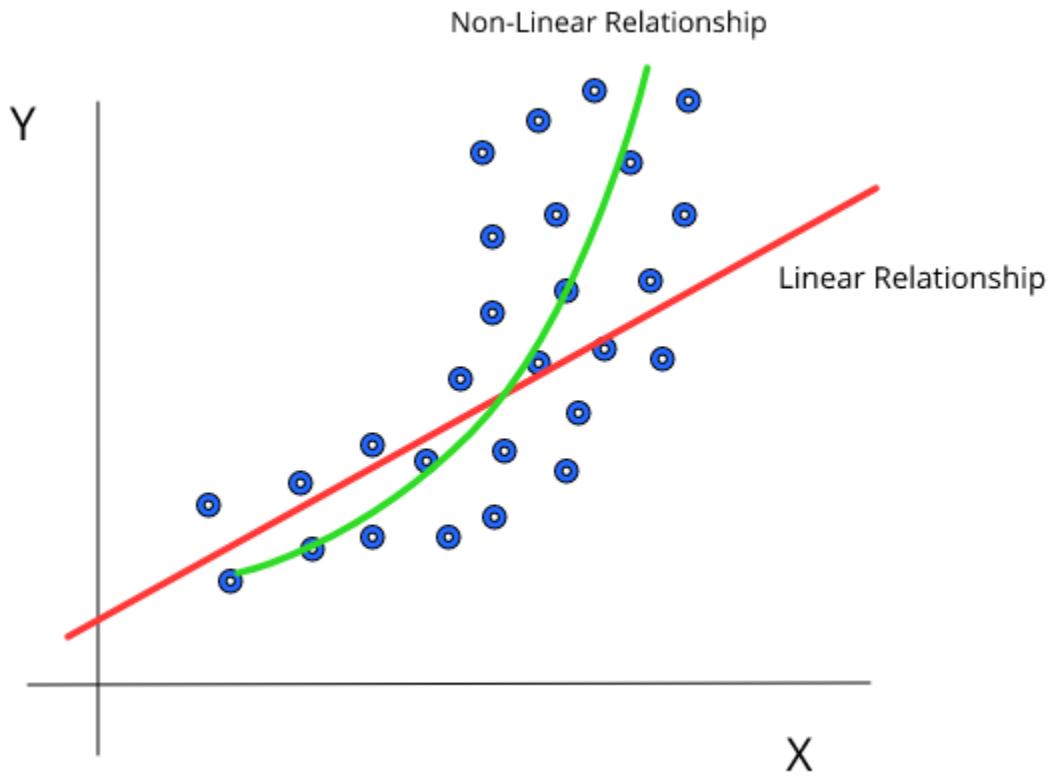
$Y = W_1 * X_1 + W_2 * X_2$ is a Linear function

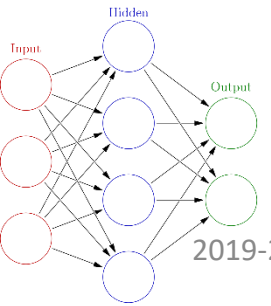
- The above equation represents a **linear relationship** between Y and X1,X2. Regardless of what values W1 and W2 have, at the end of the day the change of value of X1 and X2 will result in a **linear** change in Y.



Why Activation function is needed

- By analyzing the real world data, its realized its actually not desirable because data often has **non linear** relationships between the input and output variables.

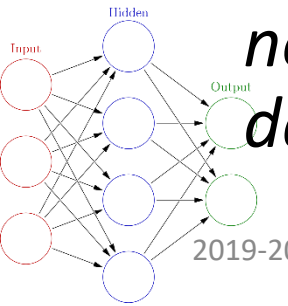


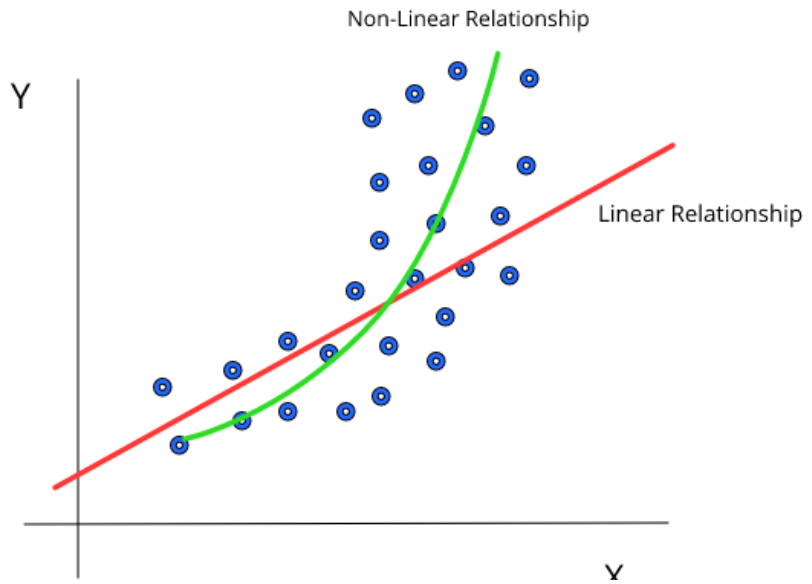


Why Activation function is needed

- If the data scientist tries to fit in the linear relationship red is generated as an output, which is not accurate.
- But if **non linear** relationships is tried then green line is generated which is much better and near to the desired results.

✓*So keep some non linear function as the activation function for each neuron and our neural network is now **capable** of fitting on non linear data.*





Without Activation Function

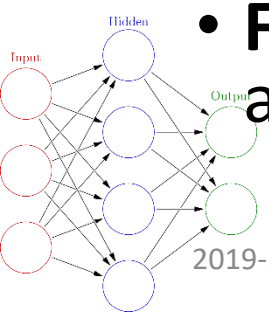
$$y = \sum_{i=0}^{n}(W_i * X_i) + B$$

With Activation Function

$$y = f(\sum_{i=0}^{n}(W_i * X_i) + B)$$

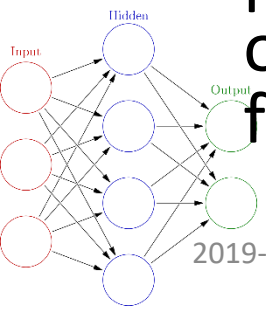
Activation function - Properties

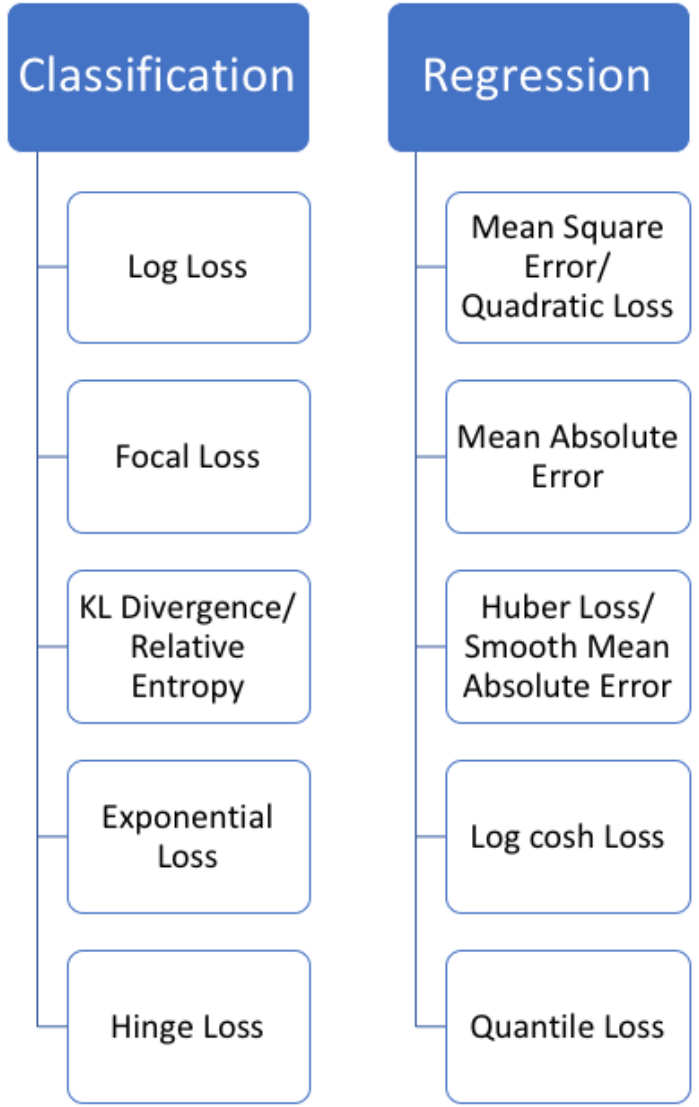
- **Non-linear** - In linear regression we're limited to a prediction equation that looks like a straight line. This is nice for simple datasets with a one-to-one relationship between inputs and outputs, but what if the patterns in our dataset were non-linear? (e.g. x2, sin, log). To model these relationships we need a non-linear prediction equation.[1] Activation functions provide this non-linearity.
- **Continuously differentiable** — To improve our model with gradient descent, we need our output to have a nice slope so we can compute error derivatives with respect to weights. If our neuron instead outputted 0 or 1 (perceptron), we wouldn't know in which direction to update our weights to reduce our error.
- **Fixed Range** — Activation functions typically squash the input data into a narrow range that makes training the model more stable and efficient.



Loss Function

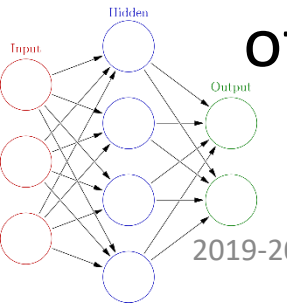
- A loss function, or cost function, is a wrapper around our model's predict function that tells us "how good" the model is at making predictions for a given set of parameters.
- The loss function has its own curve and its own derivatives.
- The slope of this curve tells us how to change our parameters to make the model more accurate! We use the model to make predictions.
- We use the cost function to update our parameters. Our cost function can take a variety of forms as there are many different cost functions available.





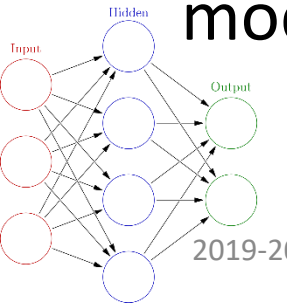
Gradient accumulation

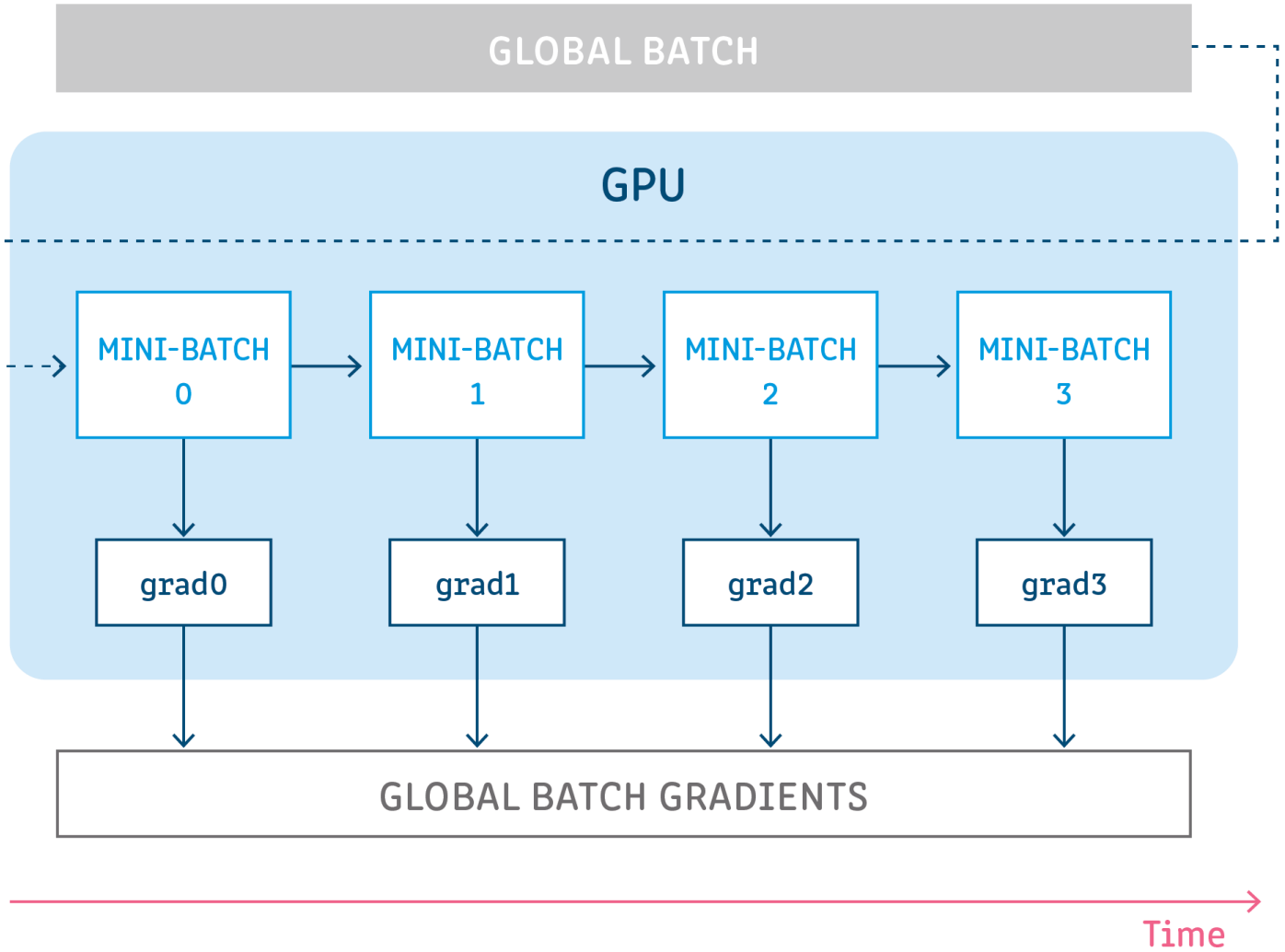
- Gradient accumulation is a mechanism to split the batch of samples—used for training a neural network—into several mini-batches of samples that will be run sequentially.
- This is used to enable using large batch sizes that require more GPU memory than available. Gradient accumulation helps in doing so by using mini-batches that require an amount of GPU memory that can be satisfied.
- Gradient accumulation means running all mini-batches sequentially (generally on the same GPU) while accumulating their calculated gradients and not updating the model variables - the weights and biases of the model.



Gradient accumulation

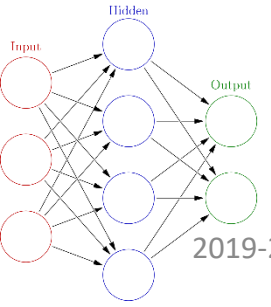
- The model variables must not be updated during the accumulation in order to ensure all mini-batches use the same model variable values to calculate their gradients.
- Only after accumulating the gradients of all those mini-batches will we generate and apply the updates for the model variables.

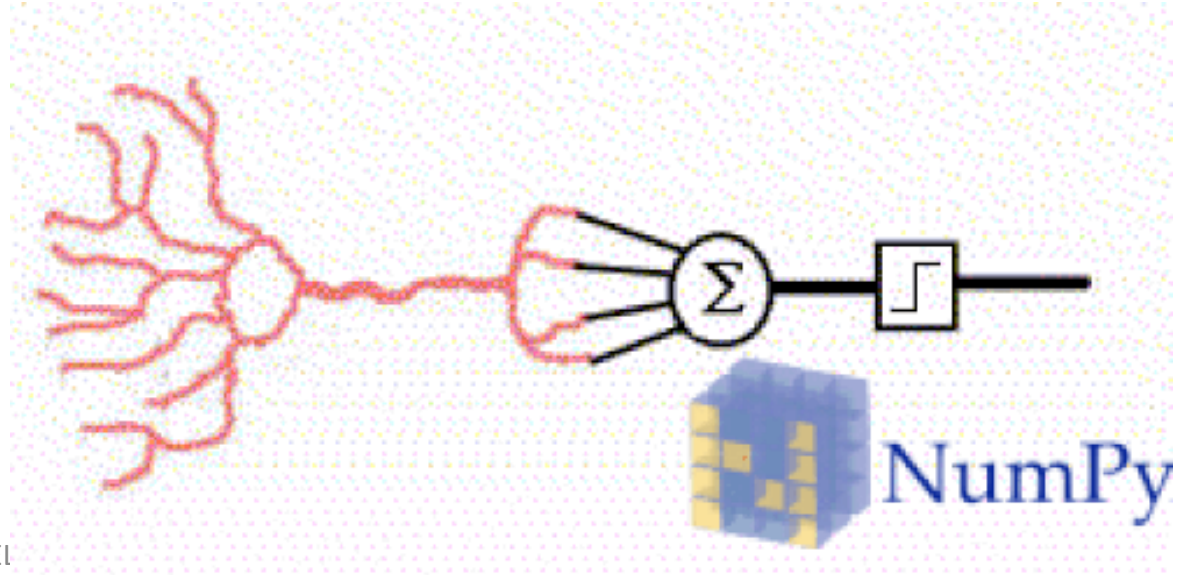




Learning in an Artificial Neural Network –

* *forward propagation,*
* *backward propagation*





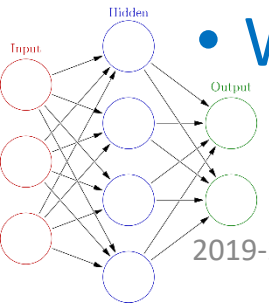
Forward Propagation (Simple Network)

- Forward propagation is how neural networks make predictions. Input data is “forward propagated” through the network layer by layer to the final layer which outputs a prediction. For the toy neural network above, a single pass of forward propagation translates mathematically to:

$$Prediction = A(A(XW_h)W_o)$$

Where

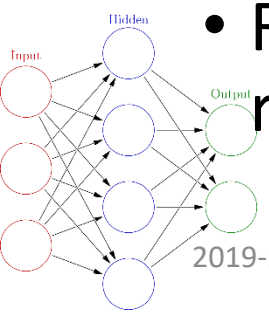
- A is activation function,
- X is input
- W_h and W_o are weights



Method feed_forward() to propagate input data through simple network of 1 hidden layer. The output of this method represents our model's prediction.

Steps :

- Calculate the weighted input to the hidden layer by multiplying X by the hidden weight W_h
- Apply the activation function and pass the result to the final layer
- Repeat step 2 except this time X is replaced by the hidden layer's output, H

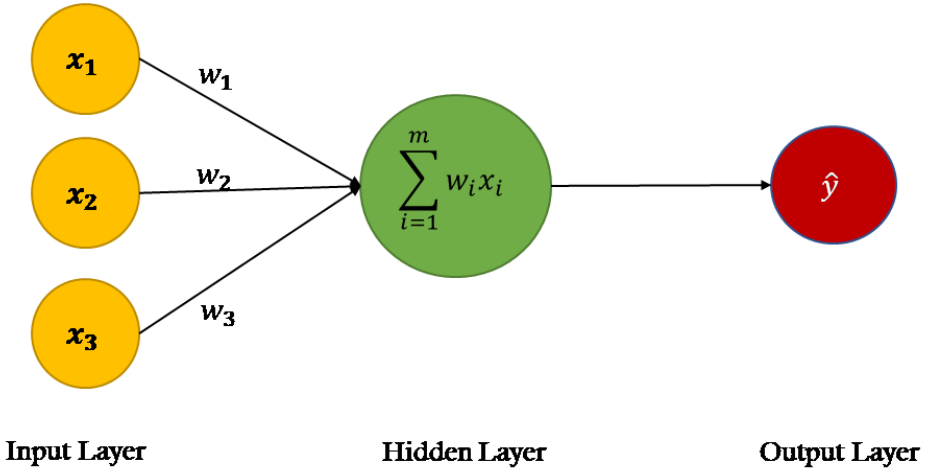


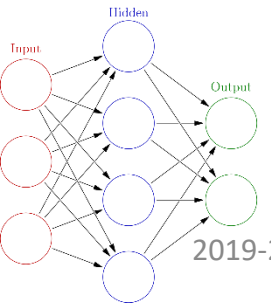

```
def relu(z):
    return max(0,z)

        x is input to the network
        Z_o , Z_h  are weighted inputs
        W_h, W_o are weights
def feed_forward(x, W_h, W_o):
    # Hidden layer
    Z_h = x * W_h
    H = relu(Zh)

    # Output layer
    Z_o = H * W_o
    output = relu(Zo)
    return output
```

Single layer Artificial Neural Network

Steps to follow:

1. Define independent variables and dependent variable
2. Define Hyperparameters
3. Define Activation Function and its derivative
4. Train the model
5. Make predictions

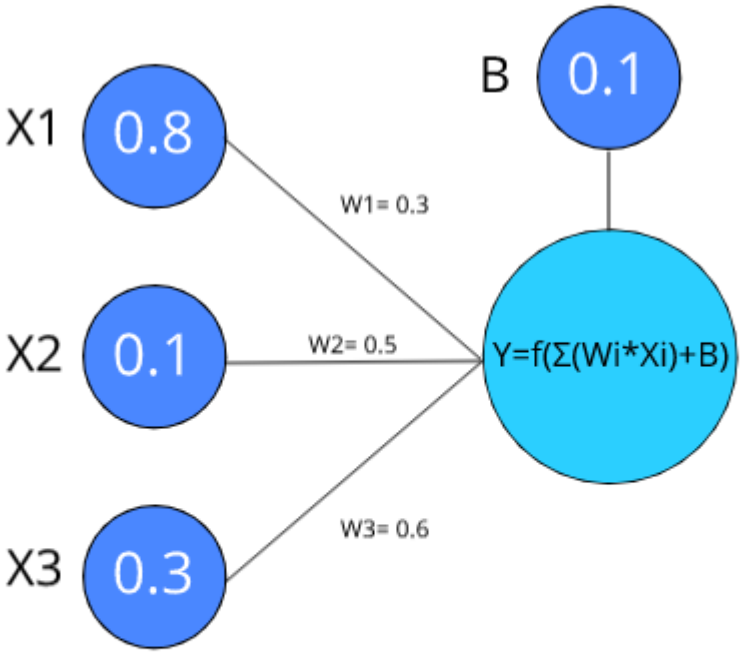




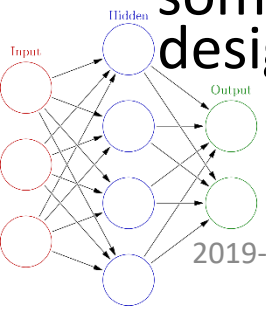
Step : 1

$$y = f(\sum_{i=1}^{n}(W_i * X_i) + B)$$

- Y is the final value of the output neuron.
- W represents the weights between the nodes in the previous layer and the output neuron. Here there are three of them. (i = 1, 2, 3)
- X represents the values of the nodes of the previous layer.
- B represents bias, which is an additional value present for each neuron. Bias is essentially a weight without an input term. It's useful for having an extra bit of adjustability which is not dependant on previous layer.
- f() is called an Activation function and it is something we as the neural network designer will choose.



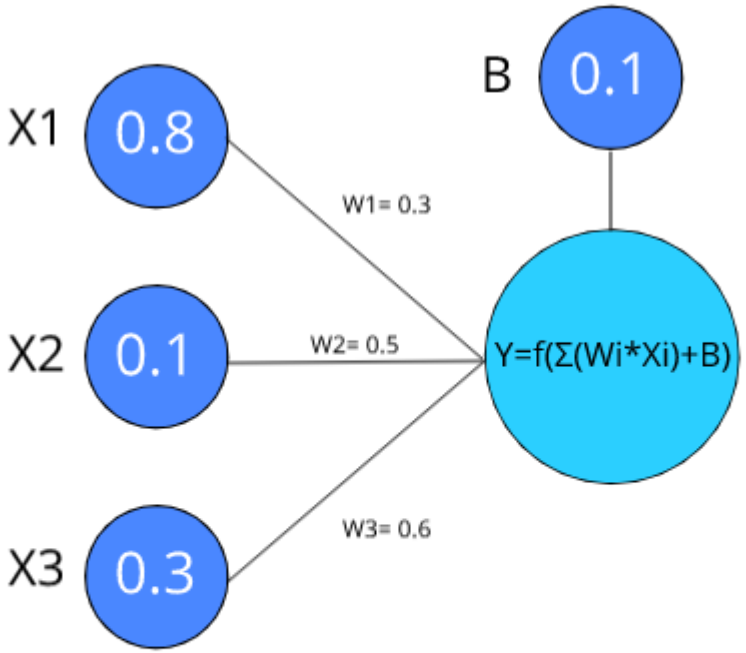
Y=f((0.3)*(0.3) + (0.5)*(0.1) + (0.6)*(0.3) + 0.1) = f(0.57)



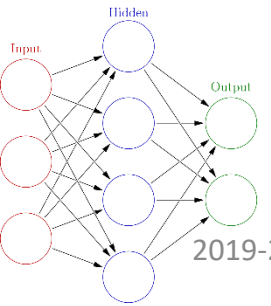
Step : 1

Step : 1

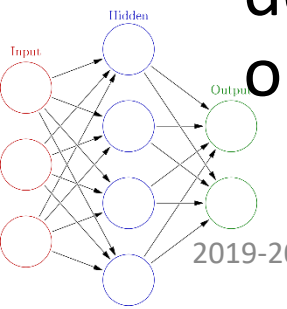
$$y = f(\sum_{i=1}^{n}(W_i * X_i) + B)$$



Y=f((0.3)*(0.3) + (0.5)*(0.1) + (0.6)*(0.3) + 0.1) = f(0.57)



- **Forward Propagation:** Send some inputs, multiplied by their weights, into our activation function (I'll finally reveal it when we start coding). This will give us our neuron's output, y.
- **In between:** Compute the error of our neuron by computing the cost function, y-ŷ. This tells us how wrong our neuron is, and how much to adjust by.
- **Backpropagation:** Calculate the adjustment for each weight using the derivative of the activation function, andthen adjust the weights.
- Each cycle of Forward and Backpropagation is called one **epoch**. We'll do many, many epochs to determine our final answer (1,000 for us in our problem).



- https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html
- https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html