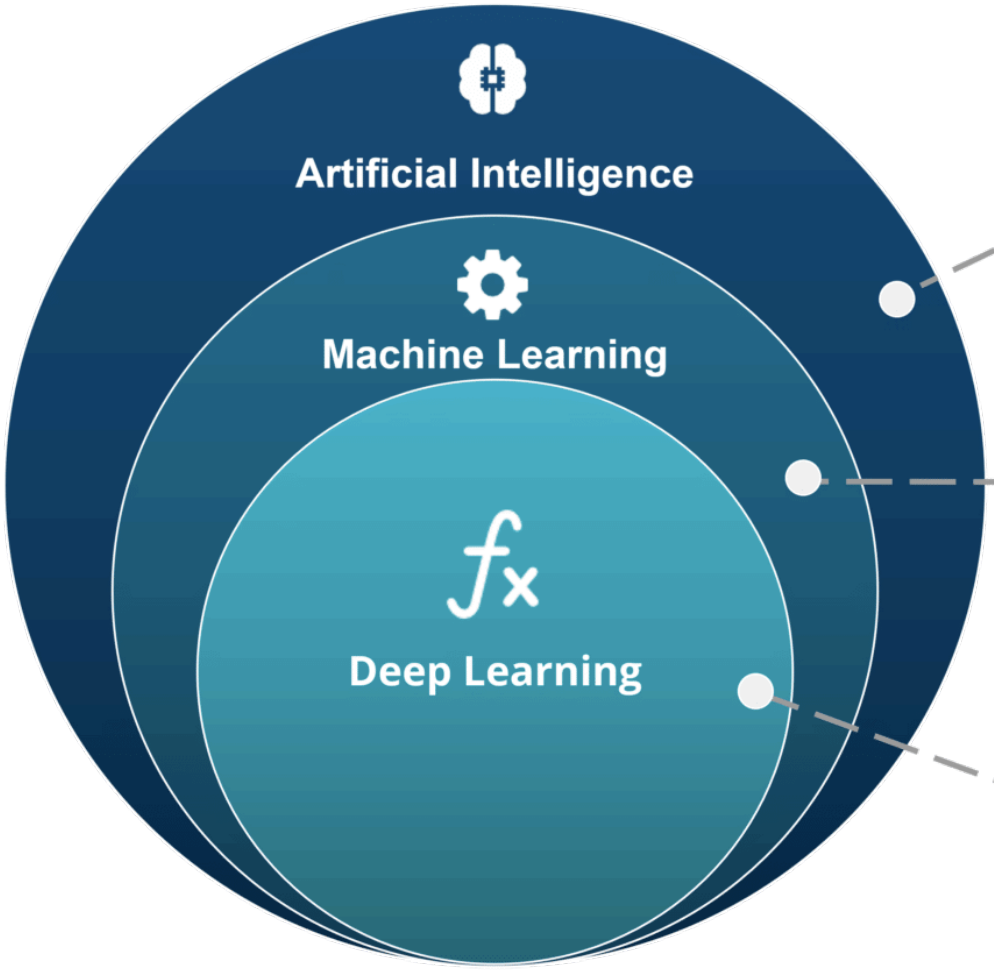
UNIVERSITÀ DEGLI STUDI DI SALERNO

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Carmen Bisogni, PhD Student

# C.A.S.A. Course

CONTEXT AWARE SECURITY ANALYTICS IN COMPUTER VISION
Lesson 4



## ARTIFICIAL INTELLIGENCE

A technique which enables machines to mimic human behaviour

## MACHINE LEARNING

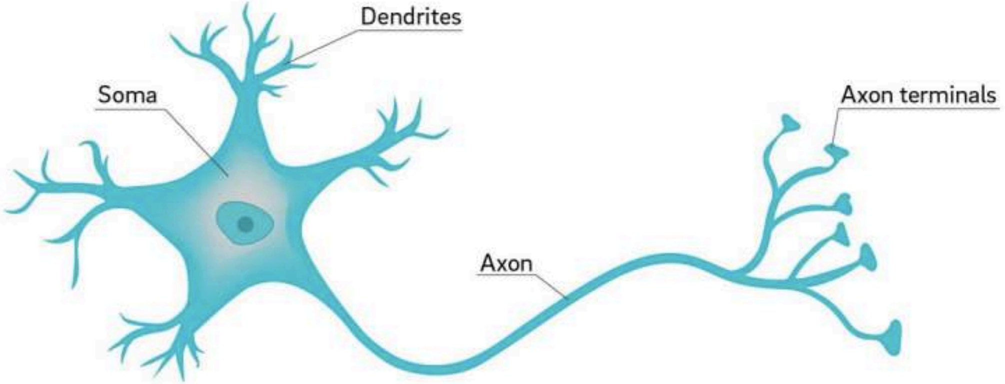
Subset of AI technique which use statistical methods to enable machines to improve with experience

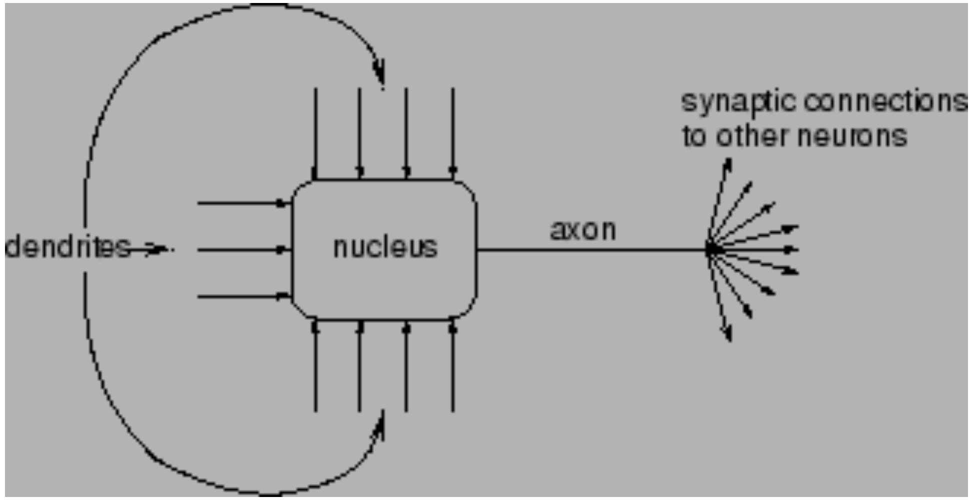
## DEEP LEARNING

Subset of ML which make the computation of multi-layer neural network feasible

# NN Theory

The NN is a network consisting of connected neurons. The center of the neuron is called the nucleus. The nucleus is connected to other nucleuses by means of the dendrites and the axon. This connection is called a synaptic connection.





The human brain consists of around $10^{11}$ neurons which are highly interconnected with around $10^{15}$ connections.  These neurons activates in parallel as an effect to internal and external sources. The brain is connected to the rest of the nervous system, which allows it to receive information by means of the five senses and also allows it to control the muscles.

# NN Theory

It is not possible (at the moment) to make an artificial brain, but it is possible to make simplified artificial neurons and artificial neural networks.

A single artificial neuron can be implemented in many different ways. The general mathematic definition is

number of input

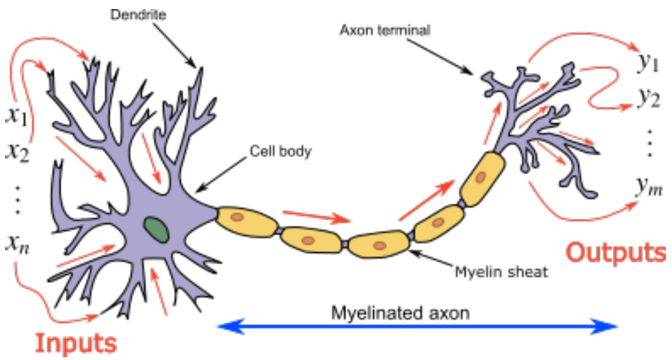
input

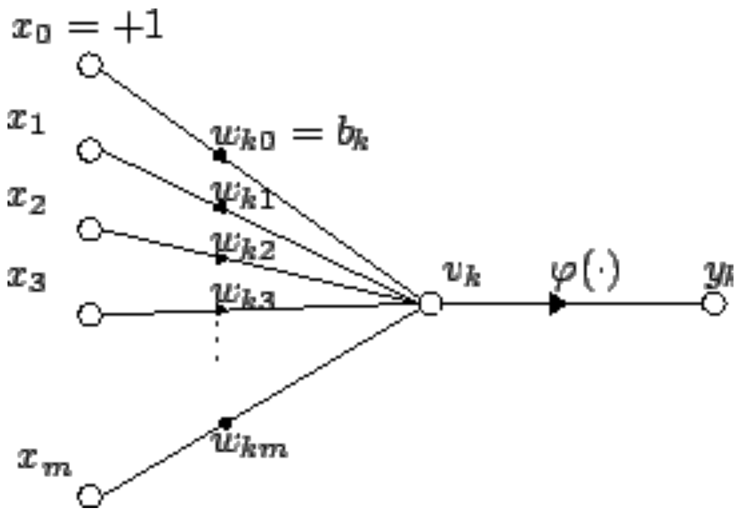
$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

weights

activation function

Biological model

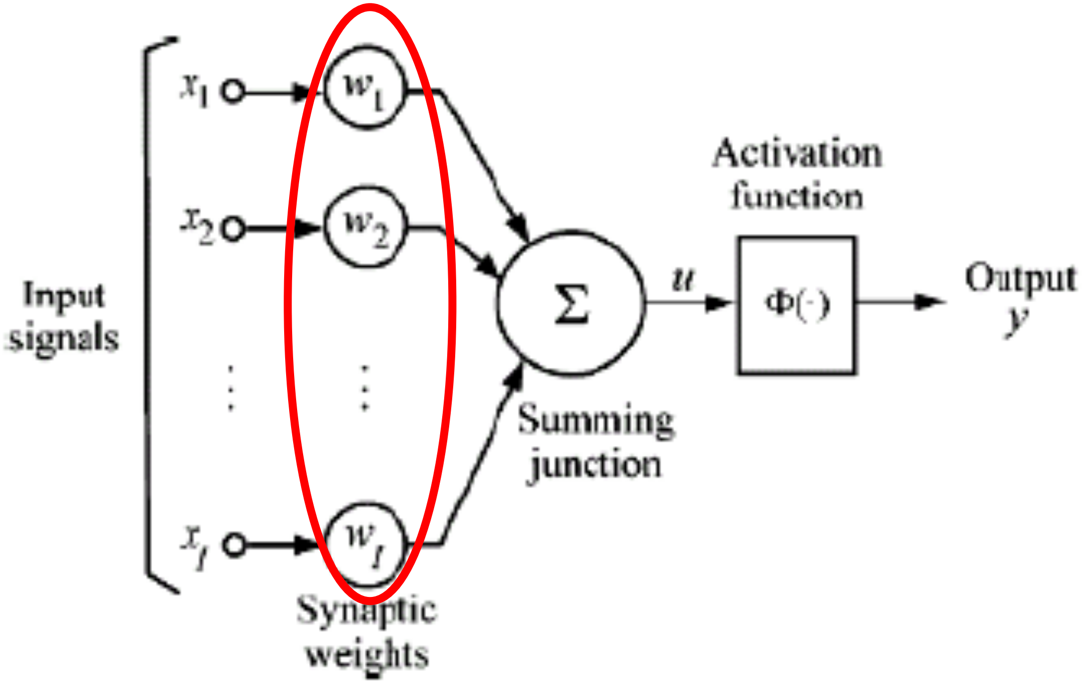




# NN Theory: Components

But not only neurons are in a neural network…

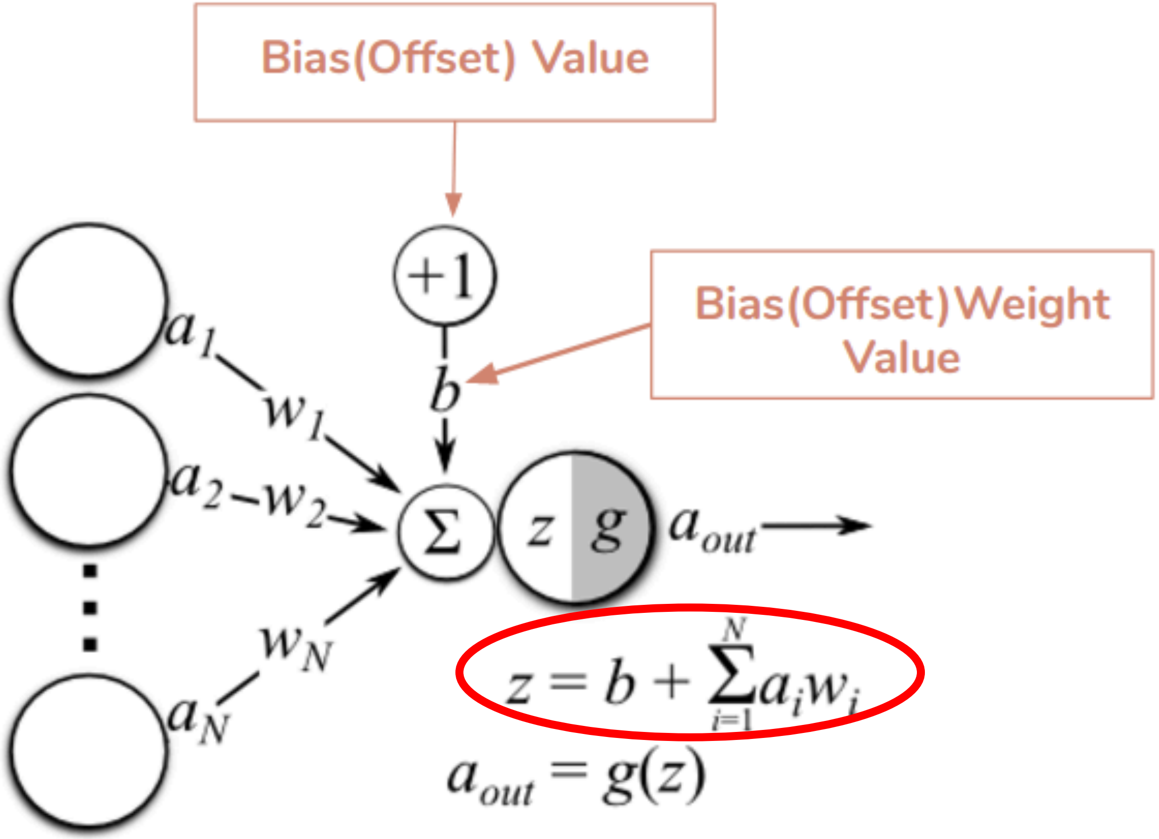
- Connections and weights: The network consists of connections, each connection providing the output of one neuron as an input to another neuron. Each connection is assigned a weight that represents its relative importance. A given neuron can have multiple input and output connections.



# NN Theory: Components

- Propagation function: The propagation function computes the input to a neuron from the outputs of its predecessor neurons and their connections as a weighted sum. A bias term can be added to the result of the propagation.

The bias neuron is a special neuron added to each layer in the neural network, which simply stores the value of 1



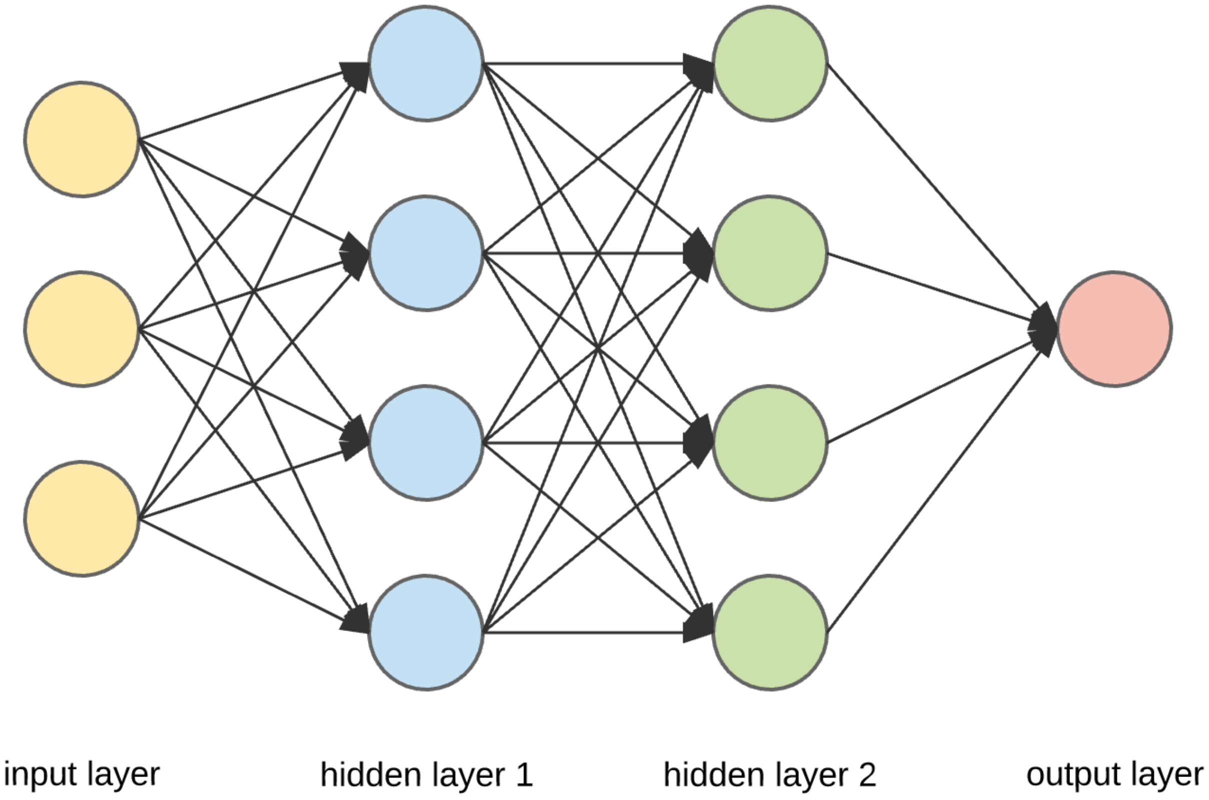
$$z = b + \sum_{i=1}^{N} a_i w_i$$

$$a_{out} = g(z)$$

# NN Theory: Structure

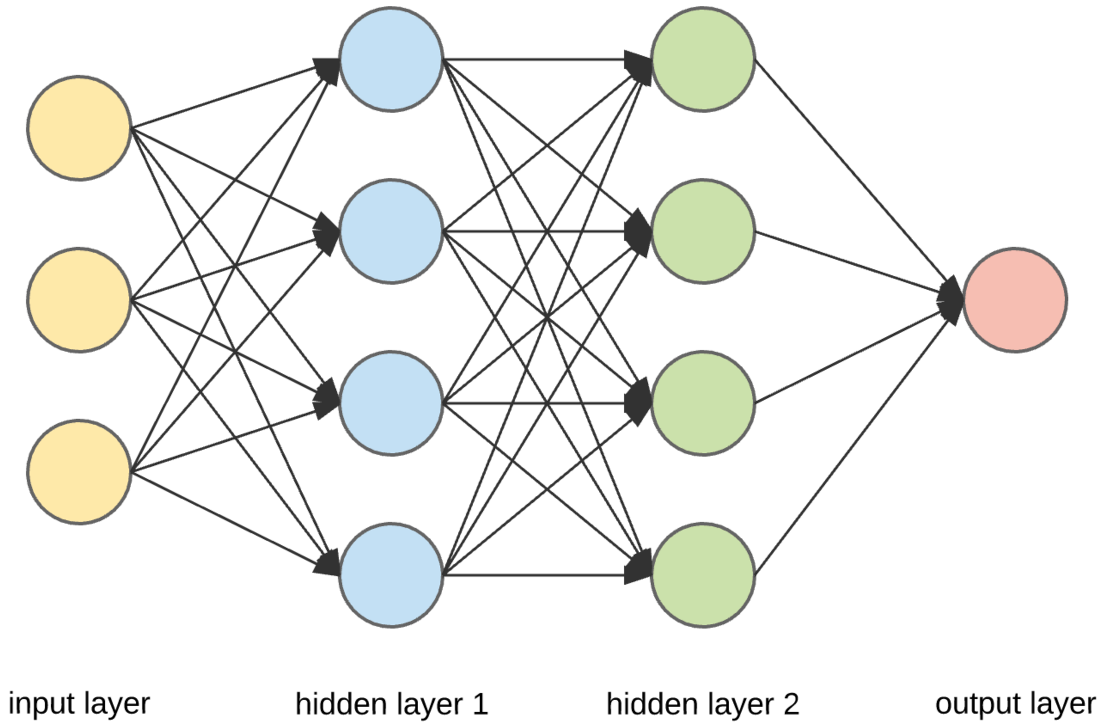
The Neural Network is constructed from 3 type of layers:

1. Input layer: initial data for the neural network.
2. Hidden layers: intermediate layer between input and output layer and place where all the computation is done.
3. Output layer: produce the result for given inputs.



# NN Theory: Structure

In this image:
Let’s look at the top blue node (“Image 1”). All the nodes from the previous layer (yellow) are connected with it. All these connections represent the weights (impact). When all the node values from the yellow layer are multiplied with their weight and all this is summarized it gives some value for the top blue node. The blue node has predefined “activation” function (unit step function on “Image 2”) which defines if this node will be “activated” or how “active” it will be, based on the summarized value. The additional node with value 1 is called “bias” node.

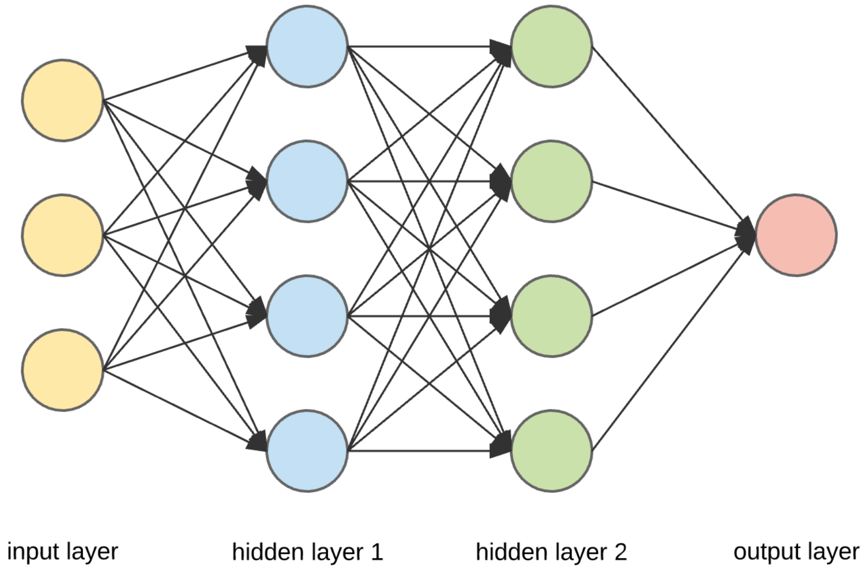


# NN Theory: Structure

What's the connection between this

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

…and this?

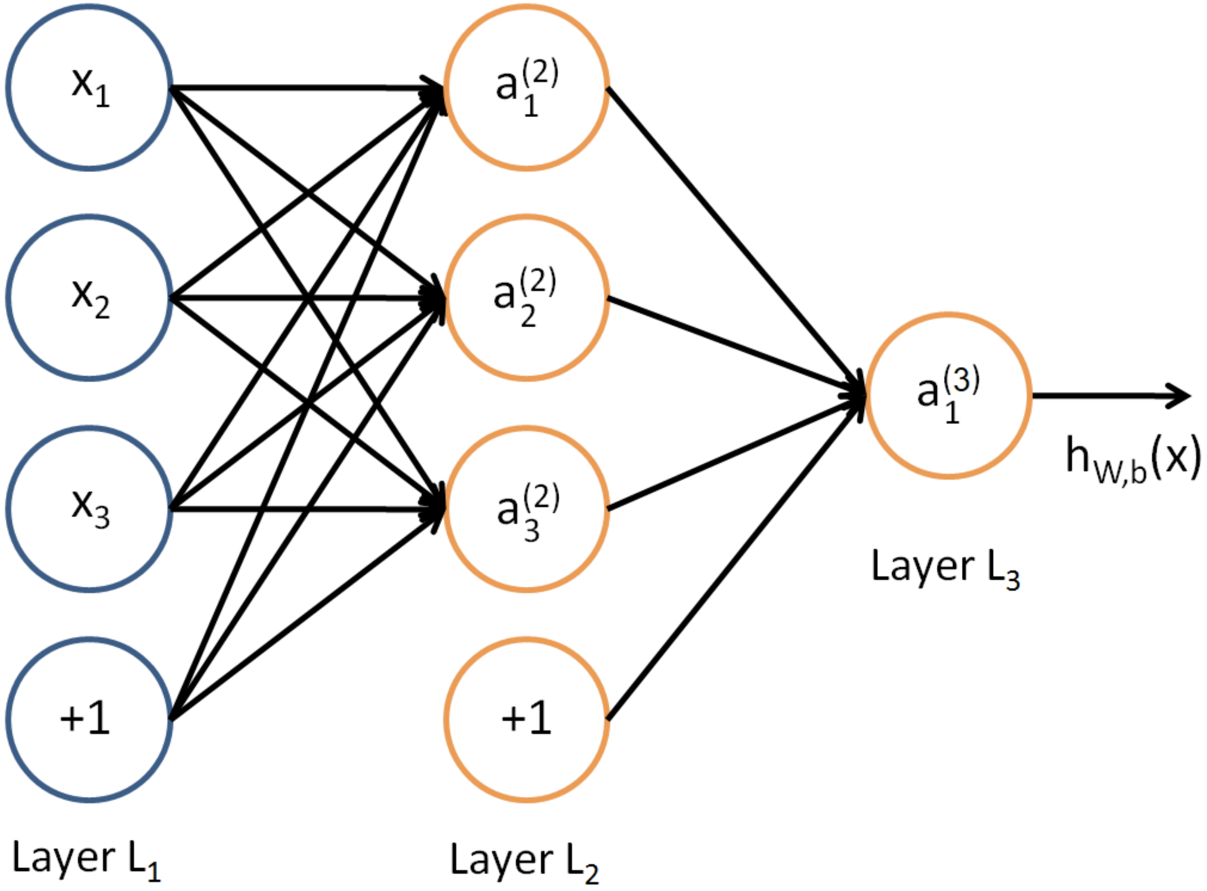


# NN Theory: Structure

This model have 4 input nodes (3 + 1 “bias”). One hidden layer with 4 nodes (3 + 1 “bias”) and one output node.

We are going to mark the “bias” nodes as x₀ and a₀ respectively. So, the input nodes can be placed in one vector X and the nodes from the hidden layer in vector A.

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad A = \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$



# NN Theory: Structure

The weights (arrows) are usually noted as θ or W. In this case I will note them as θ. The weights between the input and hidden layer will represent 3x4 matrix. And the weights between the hidden layer and the output layer will represent 1x4 matrix.

$$\theta^{(1)} = \begin{bmatrix} \theta_{10} & \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{20} & \theta_{21} & \theta_{22} & \theta_{23} \\ \theta_{30} & \theta_{31} & \theta_{32} & \theta_{33} \end{bmatrix}$$

Next, what we want is to compute the “activation” nodes for the hidden layer. In order to do that we need to multiply the input vector X and weights matrix $\theta^1$ for the first layer ($X*\theta^1$)and then apply the activation function g. What we get is :

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\ a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\ a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \end{aligned}$$

# NN Theory:
# Structure

And by multiplying hidden layer vector with weights matrix θ for the second layer(A*θ) we get output for the hypothesis function:

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

This example is with only one hidden layer and 4 nodes there. If we try to generalize for Neural Network with multiple hidden layers and multiple nodes in each of the layers we would get a very complex formula.

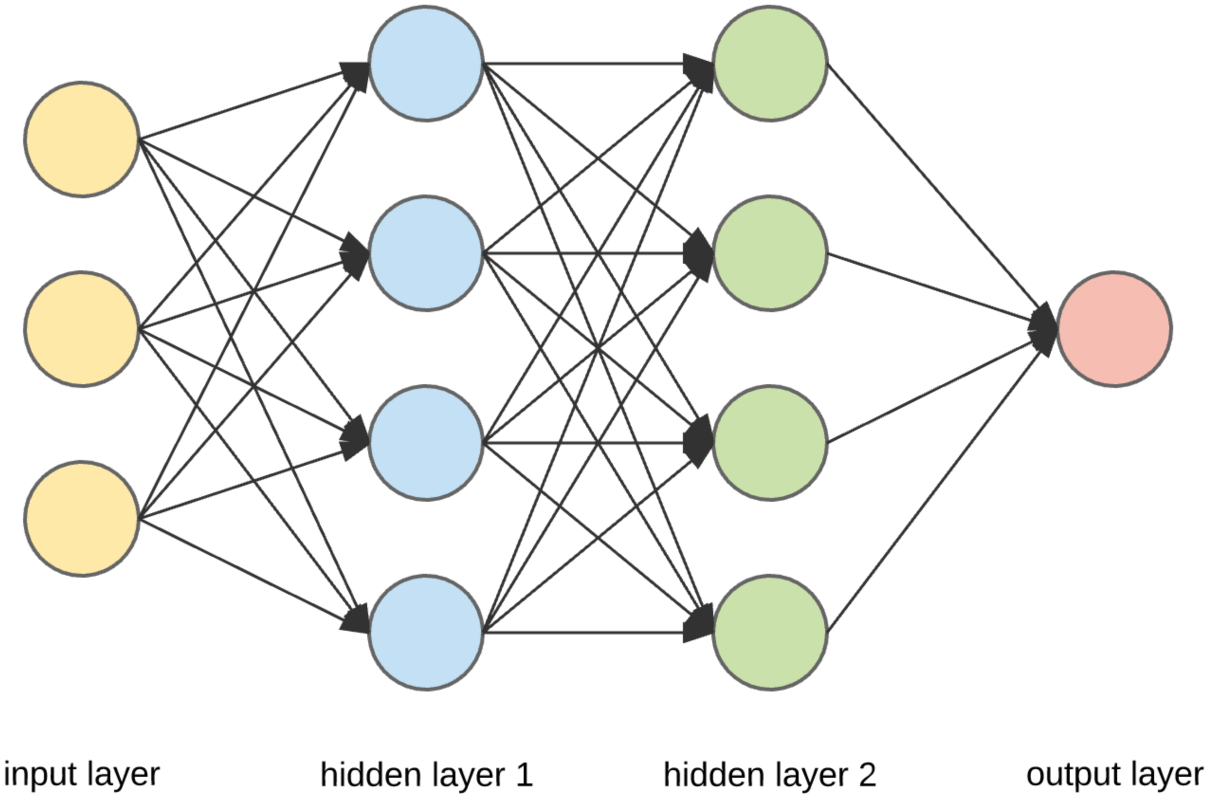
But the power of the Neural Networks is that each element of the formula is very simple…a sum!

# NN Theory: Structure

# of input layer?

# of output layer?

# of hidden layer?



# NN Theory: Structure

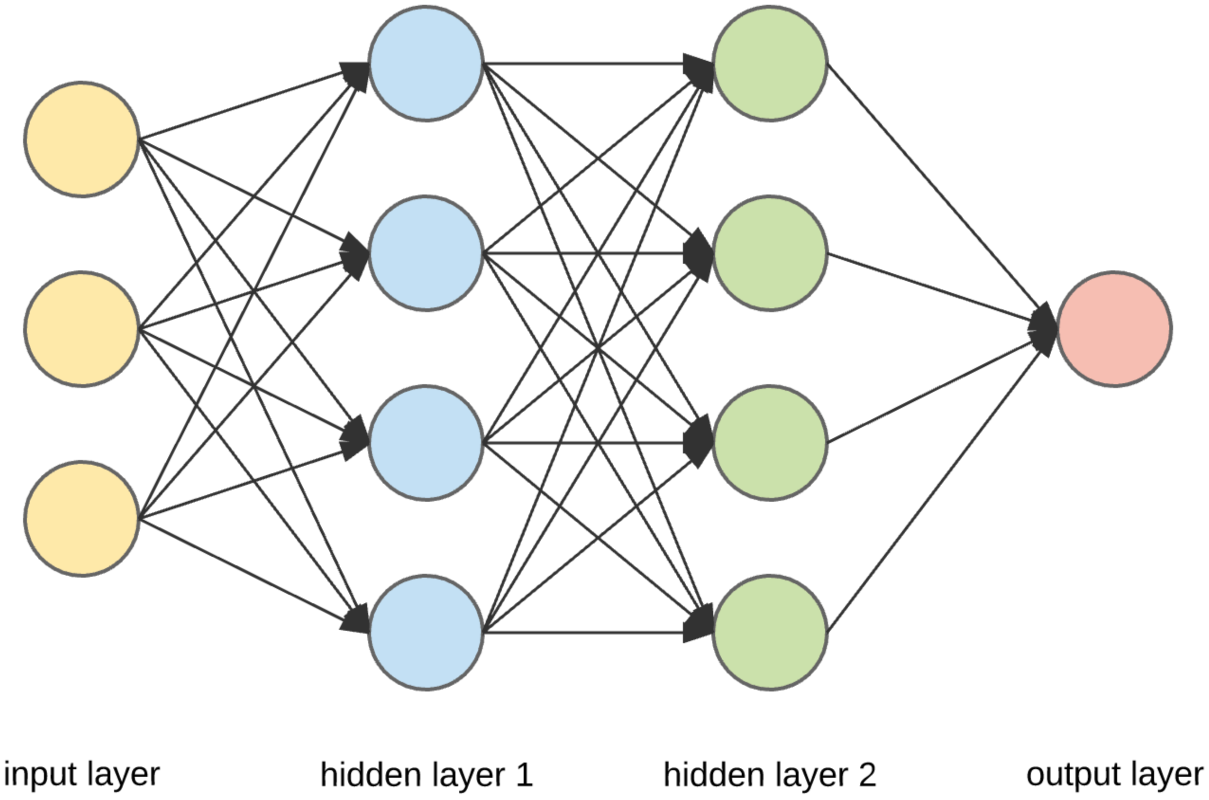
# of input layer?

# of features

# of output layer?

# of classes

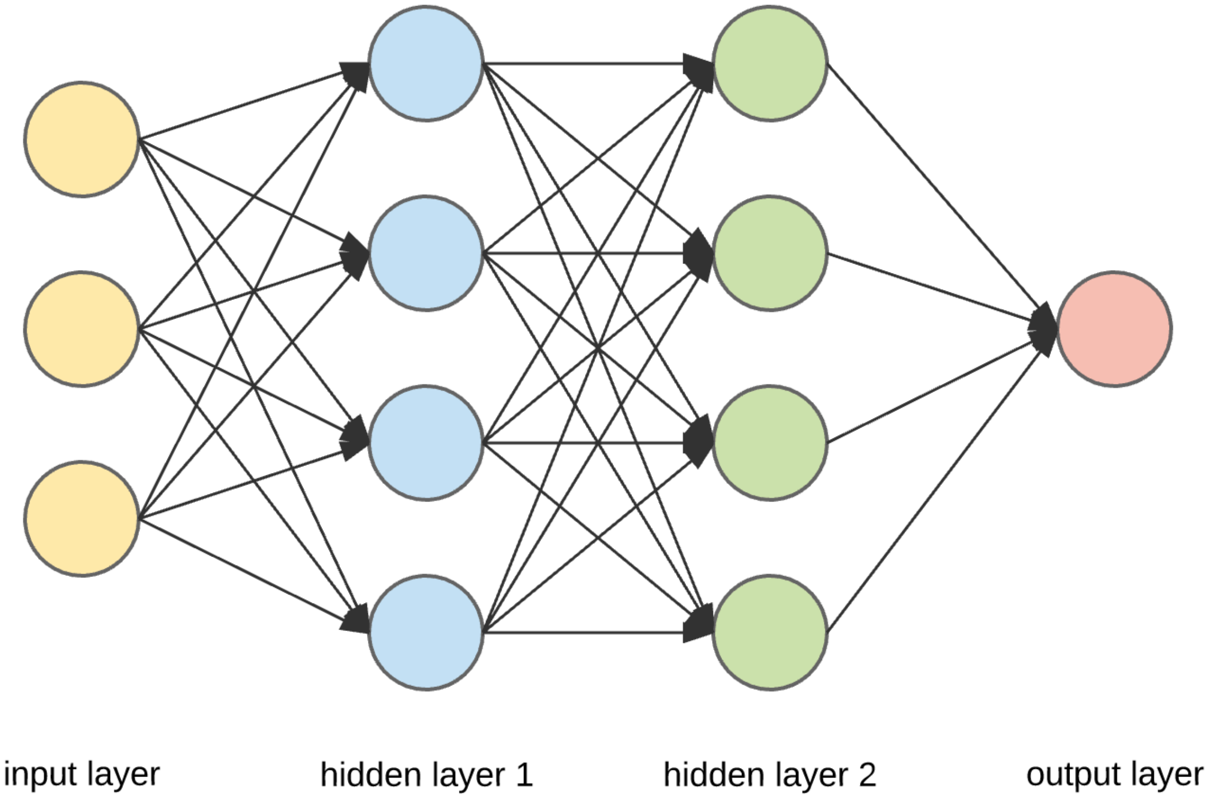
# of hidden layer?



# NN Theory: Structure

# of hidden layer?

HYPERPARAMETER!



# NN Theory:
# Hyperparameters

Hyperparameters are the variables which determines the network structure(Eg: Number of Hidden Units) and the variables which determine how the network is trained(Eg: Learning Rate).

We will start with hyperparameters related to the network structure.

- Number of hidden layers:

“Very simple. Just keep adding layers until the test error does not improve anymore.”

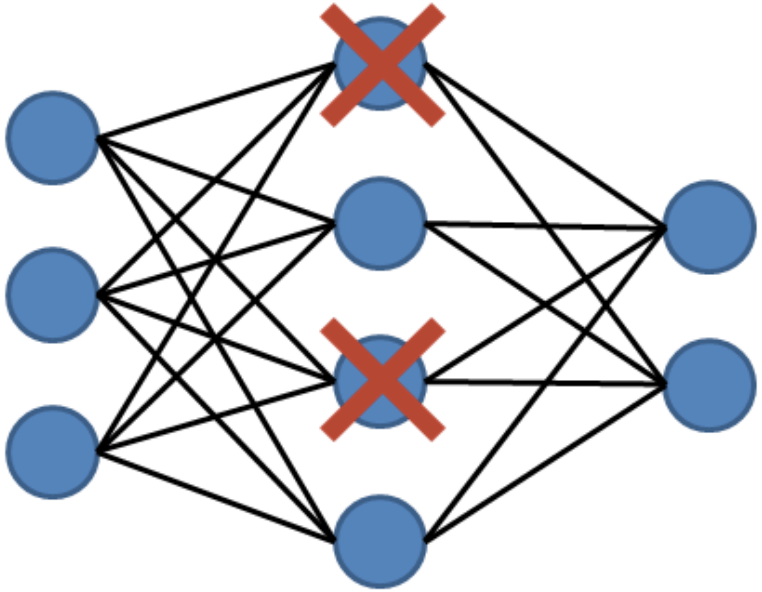
Many hidden units within a layer with regularization techniques can increase accuracy. Smaller number of units may cause underfitting.

# NN Theory: Hyperparameters

- Dropout: is regularization technique to avoid overfitting (increase the validation accuracy) thus increasing the generalizing power.

  -Generally, use a small dropout value of 20%-50% of neurons with 20% providing a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network.

  -Use a larger network. You are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations.



# NN Theory:
# Hyperparameters

- Network Weight Initialization:

Ideally, it may be better to use different weight initialization schemes according to the activation function used on each layer. Mostly uniform distribution is used.

Generally, weight are initializated assigning random values to weights but there is one thing to keep in my mind is that what happens if weights are initialized high values or very low values and what is a reasonable initialization of weight values.

Neural networks are trained by fine-tuning weights, to discover the optimal set of weights that generates the most accurate prediction.

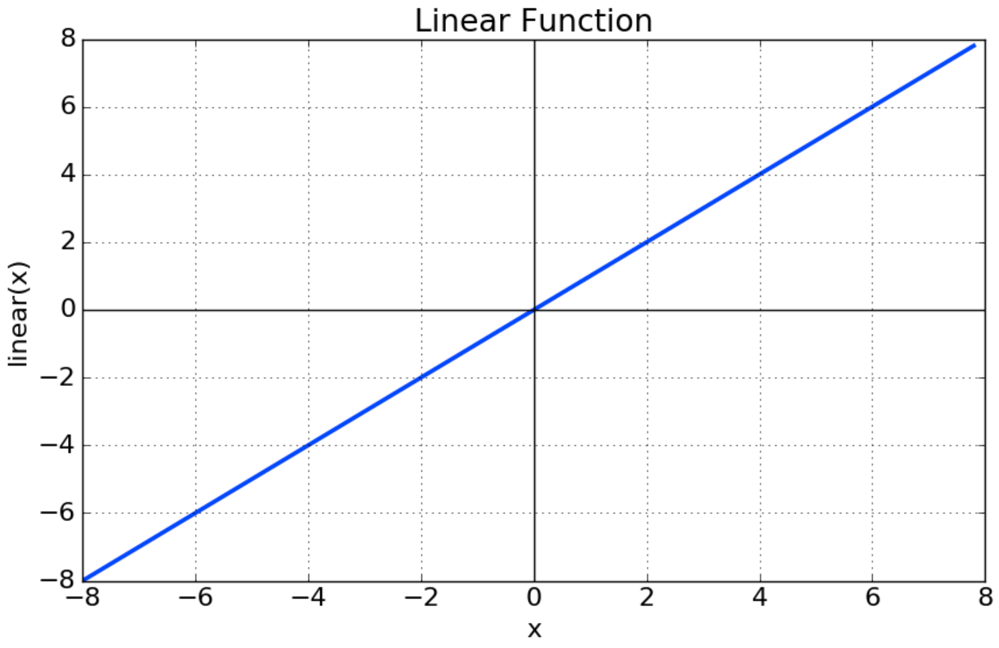
# NN Theory:
# Activation Functions

Activation function:
It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

The Activation Functions can be basically divided into 2 types:

- Linear Activation Function: As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range. It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.

Example: f(x)=x

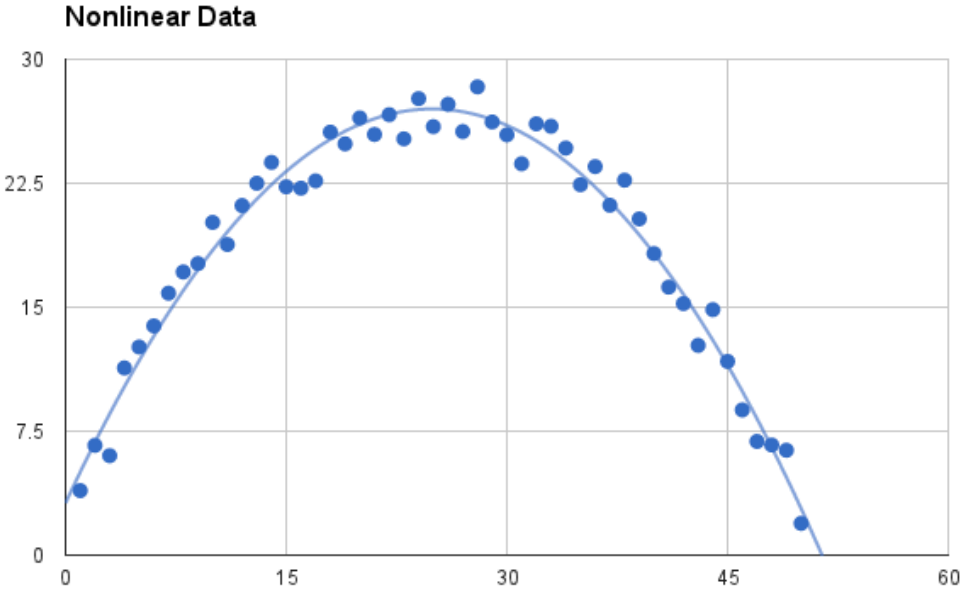


# NN Theory:
# Activation Functions

- Non-linear Activation Functions: The Nonlinear Activation Functions are the most used activation functions.

It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.

The main terminologies needed to understand for nonlinear functions are:



**Derivative or Differential**: Change in y-axis w.r.t. change in x-axis.It is also known as slope.

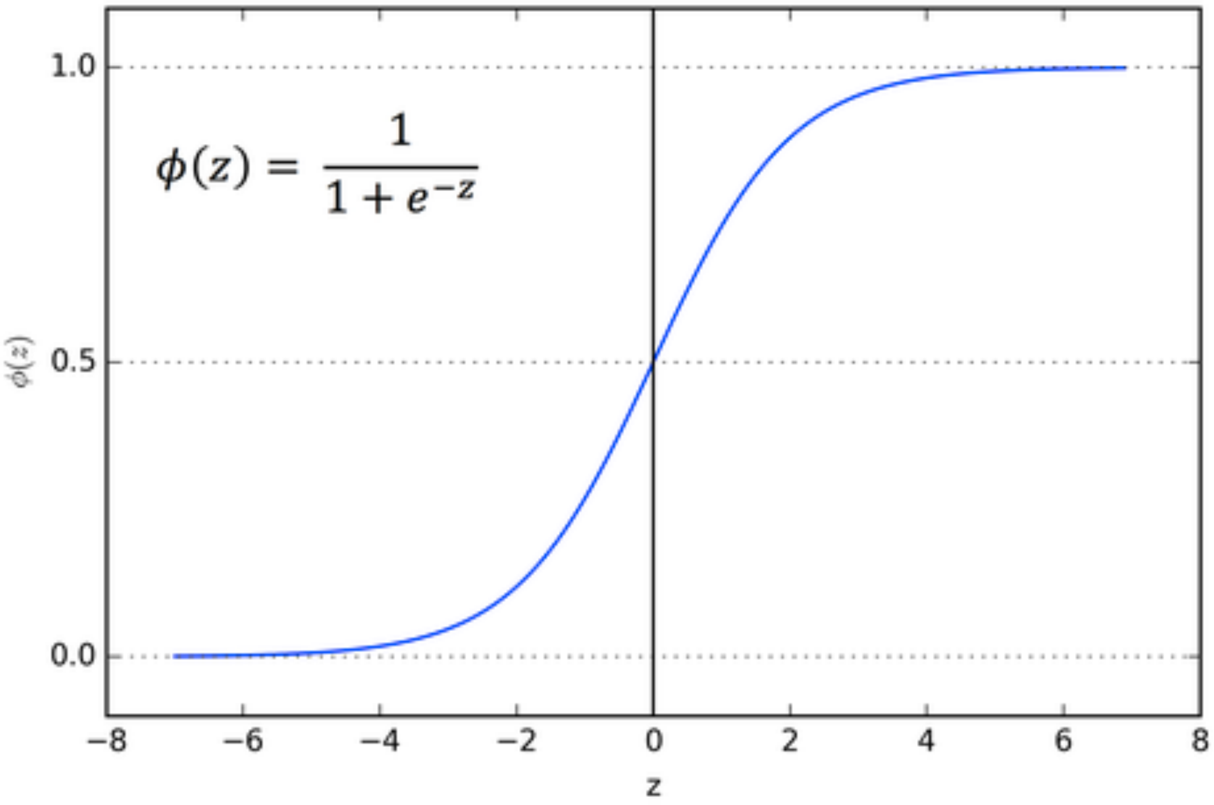
**Monotonic function**: A function which is either entirely non-increasing or non-decreasing.

The Nonlinear Activation Functions are mainly divided on the basis of their range or curves-

# NN Theory: Activation Functions

Example of Non-Linear activation function:

**Sigmoid or Logistic Activation Function**
The Sigmoid Function curve looks like a S-shape.
The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.



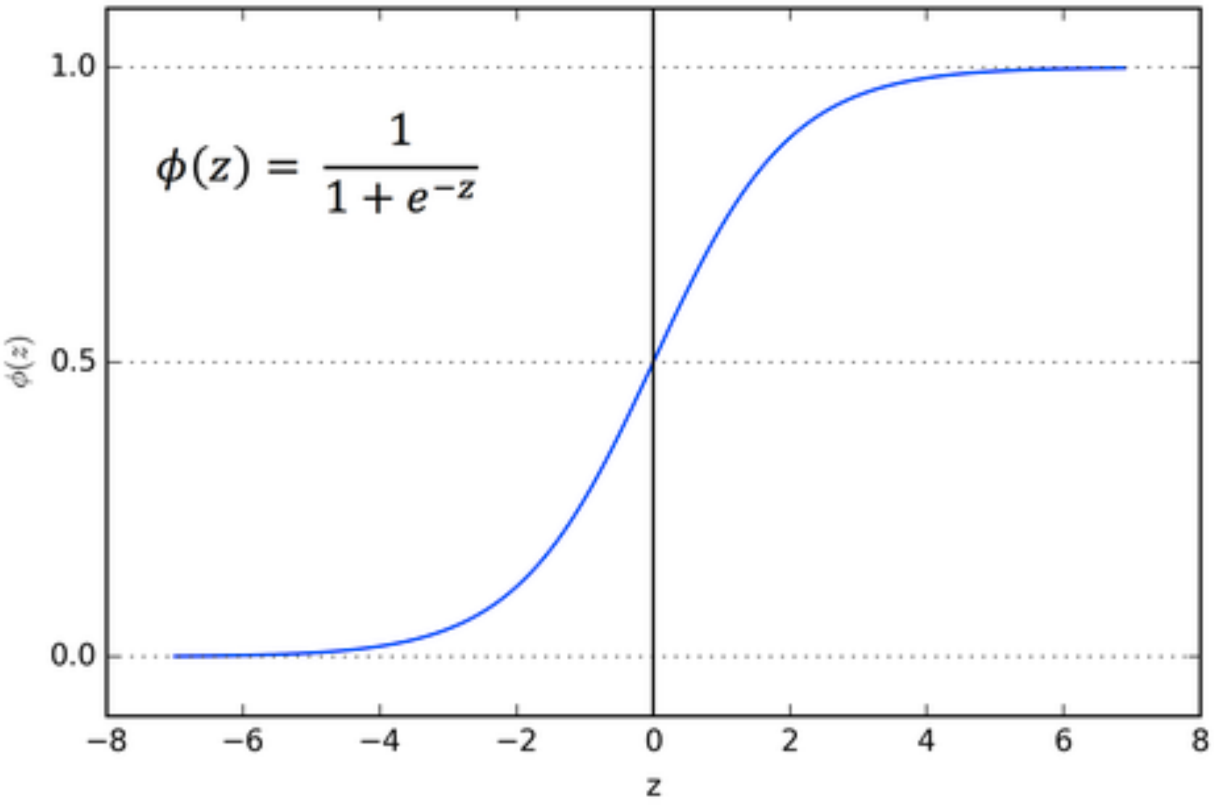
# NN Theory:
# Activation Functions

The sigmoud function is differentiable.That means, we can find the slope of the sigmoid curve at any two points.

The function is monotonic but function’s derivative is not.

The logistic sigmoid function can cause a neural network to get stuck at the training time.

The softmax function is a more generalized logistic activation function which is used for multiclass classification.



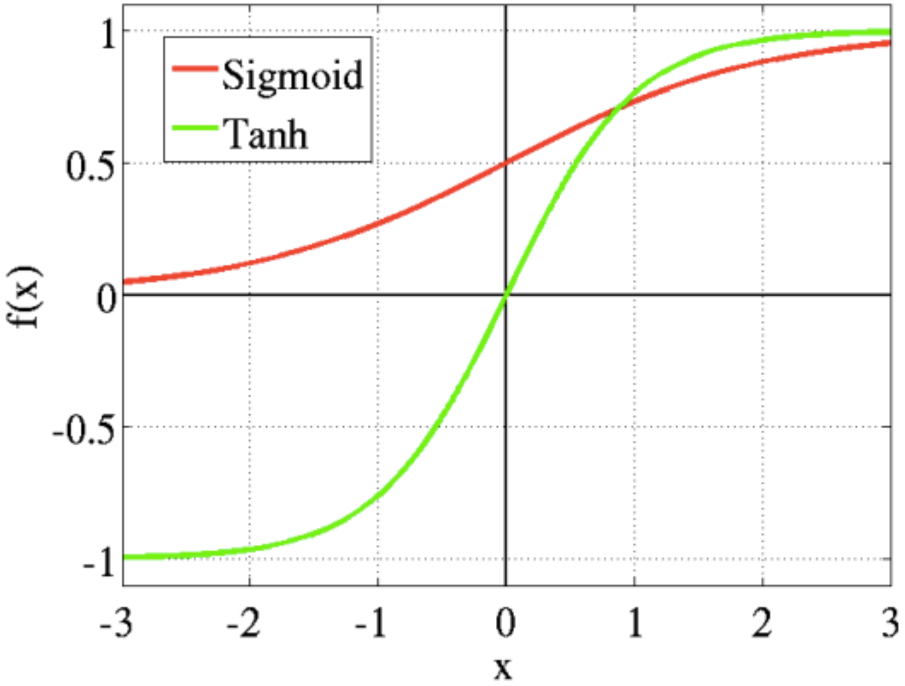
# NN Theory: Activation Functions

**Tanh or hyperbolic tangent Activation Function**

tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).

The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

The function is differentiable. The function is monotonic while its derivative is not monotonic. The tanh function is mainly used classification between two classes. Both tanh and logistic sigmoid activation functions are used in feed-forward nets. (keep in mind)



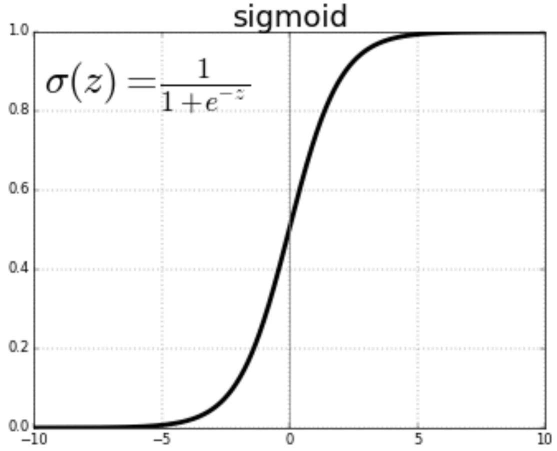
# NN Theory:
# Activation Functions

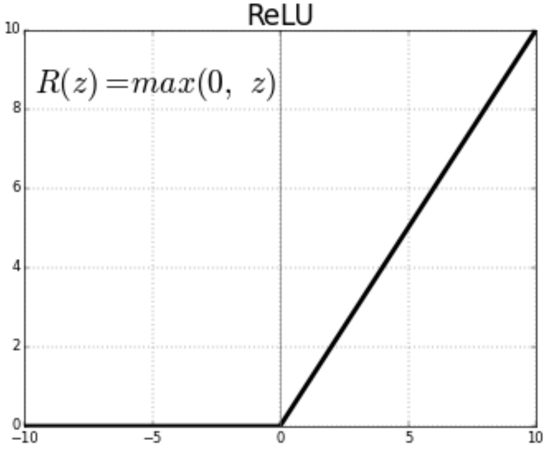
**ReLU (Rectified Linear Unit) Activation Function**

The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning. (We will see CNN in next lesson).

f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero. Range from 0 to infinity. The function and its derivative both are monotonic.

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.





# NN Theory:
# Activation Functions

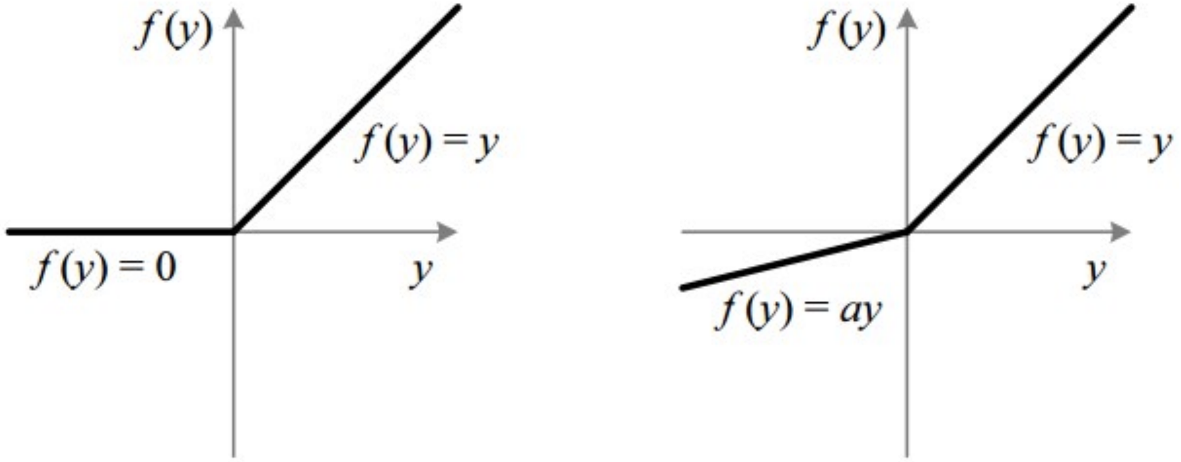
**Leaky ReLU**

It is an attempt to solve the dying ReLU problem. The leak helps to increase the range of the ReLU function. Usually, the value of a is 0.01 or so.

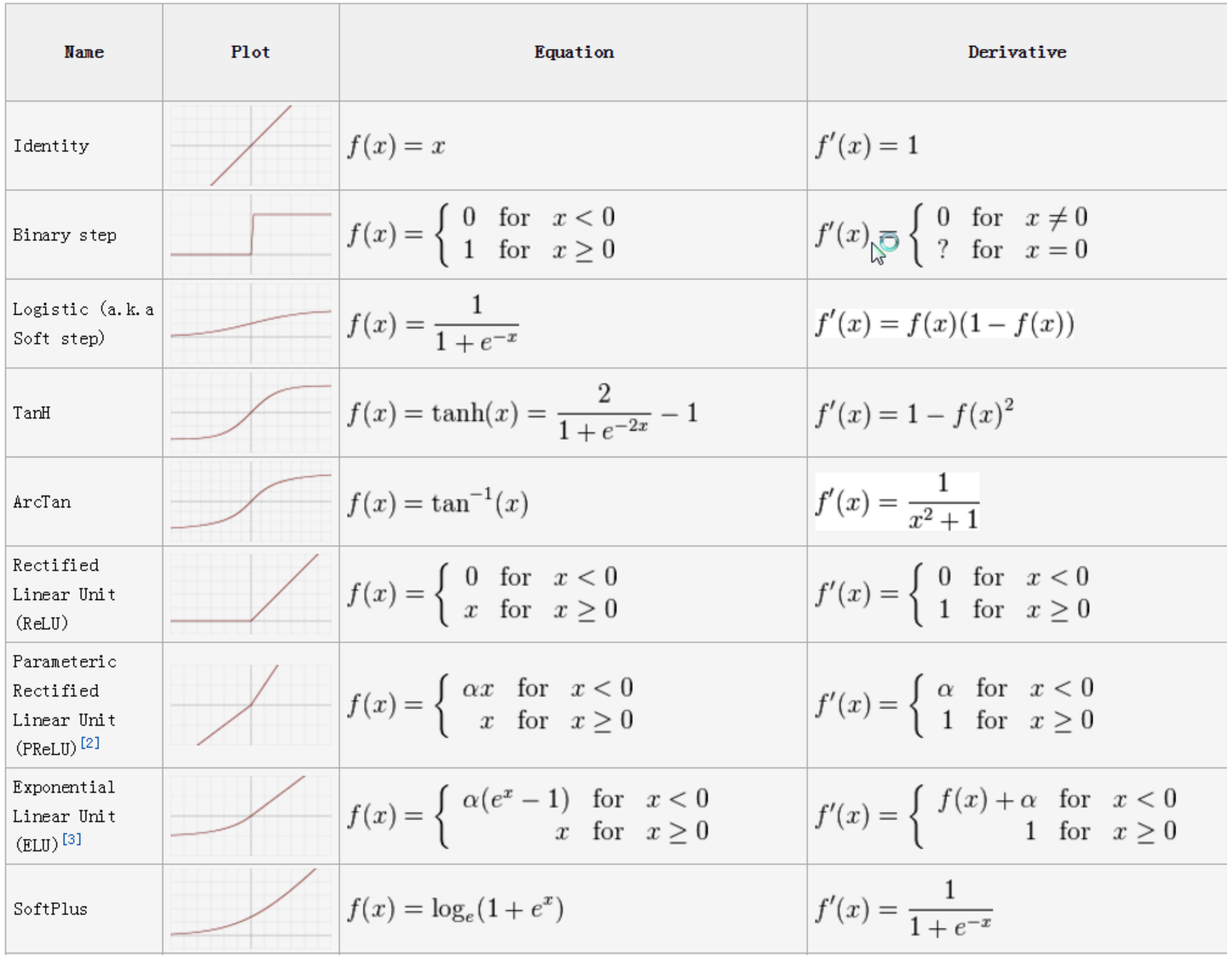
When a is not 0.01 then it is called Randomized ReLU.

Therefore the range of the Leaky ReLU is (-infinity to infinity).

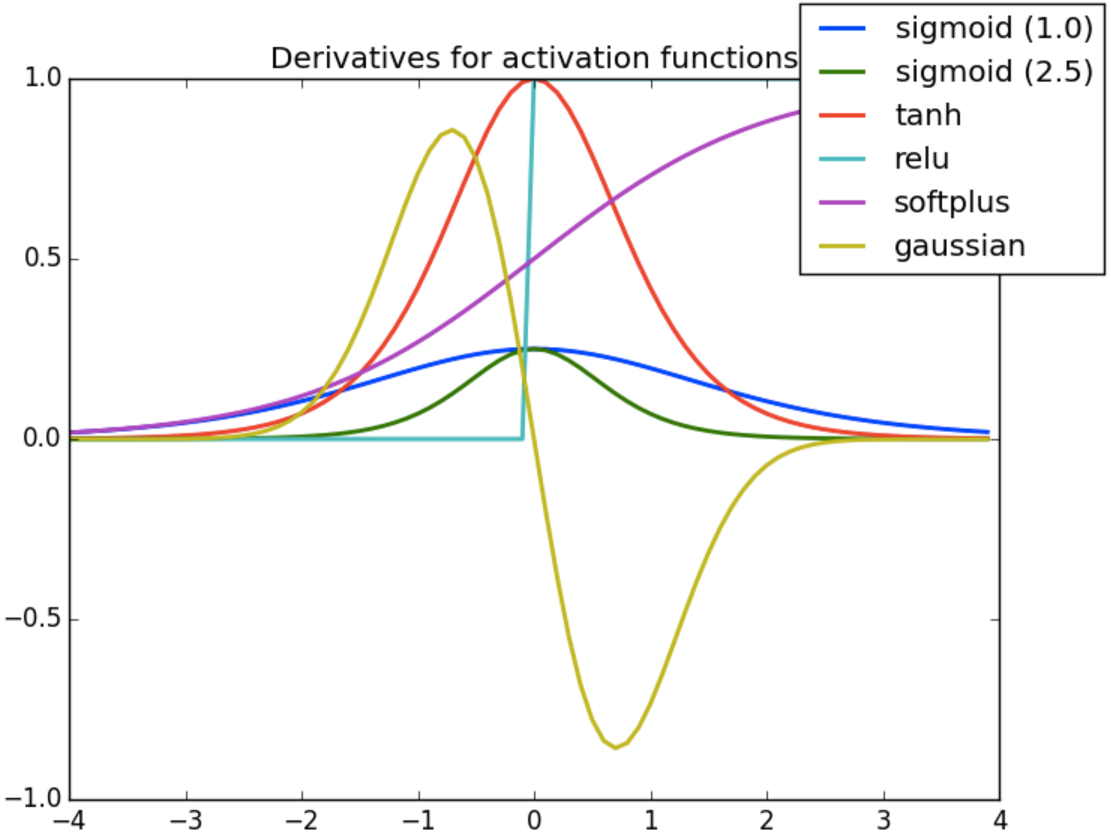
Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives also monotonic in nature.



# NN Theory:
# Activation Functions



| Name | Plot | Equation | Derivative |
|---|---|---|---|
| Identity | | $f(x) = x$ | $f'(x) = 1$ |
| Binary step | | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$ |
| Logistic (a.k.a Soft step) | | $f(x) = \frac{1}{1+e^{-x}}$ | $f'(x) = f(x)(1 - f(x))$ |
| TanH | | $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$ | $f'(x) = 1 - f(x)^2$ |
| ArcTan | | $f(x) = \tan^{-1}(x)$ | $f'(x) = \frac{1}{x^2+1}$ |
| Rectified Linear Unit (ReLU) | | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Parameteric Rectified Linear Unit (PReLU)[2] | | $f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Exponential Linear Unit (ELU)[3] | | $f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| SoftPlus | | $f(x) = \log_e(1 + e^x)$ | $f'(x) = \frac{1}{1+e^{-x}}$ |



# NN Theory:
# Cost function

This function represent the sum of the error, difference between the predicted value and the real (labeled) value.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

Cost function returns a scalar value called 'cost' , that tells how good or bad your model is. There are several cost functions that can be used. Less cost represent a good model. The reason cost functions are used in neural networks is that 'cost is used by models to improve’.

*The cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared.*

There are several cost functions.

# NN Theory:
# Cost function

## Maximum Likelihood

Maximum likelihood estimation, or MLE, is a framework for inference for finding the best statistical estimates of parameters from historical training data: exactly what we are trying to do with the neural network.

We have a training dataset with one or more input variables and we require a model to estimate model weight parameters that best map examples of the inputs to the output or target variable.

Given input, the model is trying to make predictions that match the data distribution of the target variable.

Under maximum likelihood, a loss function estimates how closely the distribution of predictions made by a model matches the distribution of target variables in the training data. A benefit of using maximum likelihood as a framework for estimating the model parameters (weights) for neural networks and in machine learning in general is that as the number of examples in the training dataset is increased, the estimate of the model parameters improves. This is called the property of “consistency.”

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

# NN Theory:
# Cost function

**Maximum Likelihood and Cross-Entropy**

Under the framework maximum likelihood, the error between two probability distributions is measured using cross-entropy.

When modeling a classification problem where we are interested in mapping input variables to a class label, we can model the problem as predicting the probability of an example belonging to each class. In a binary classification problem, there would be two classes, so we may predict the probability of the example belonging to the first class. In the case of multiple-class classification, we can predict a probability for the example belonging to each of the classes.

Technically, cross-entropy comes from the field of information theory and has the unit of "bits." It is used to estimate the difference between an estimated and predicted probability distributions.

The maximum likelihood approach was adopted almost universally not just because of the theoretical framework, but primarily because of the results it produces. Specifically, neural networks for classification that use a sigmoid or softmax activation function in the output layer learn faster and more robustly using a cross-entropy loss function.

# NN Theory:
# Cost function

The choice of loss/cost function is directly related to the activation function used in the output layer of your neural network. These two design elements are connected.

In general:

**Regression Problem**

A problem where you predict a real-value quantity.

**Output Layer Configuration**: One node with a linear activation unit.

**Loss Function**: Mean Squared Error (MSE).

**Binary Classification Problem**

A problem where you classify an example as belonging to one of two classes.

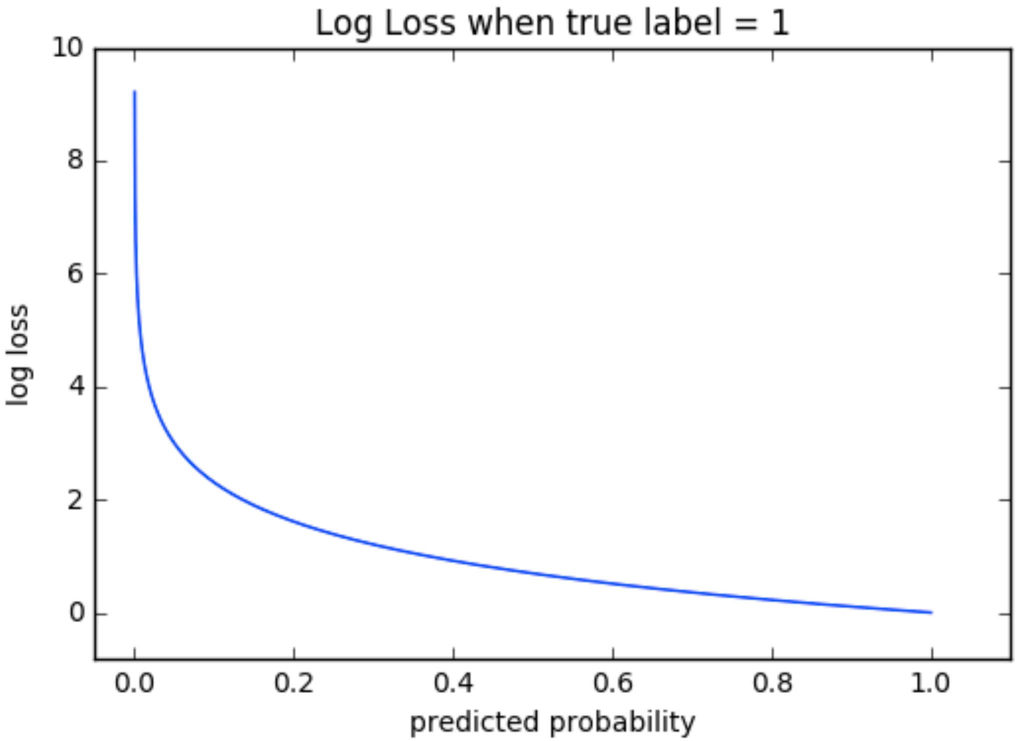
The problem is framed as predicting the likelihood of an example belonging to class one, e.g. the class that you assign the integer value 1, whereas the other class is assigned the value 0.

**Output Layer Configuration**: One node with a sigmoid activation unit.

**Loss Function**: Cross-Entropy, also referred to as Logarithmic loss.

# NN Theory:
# Cost function

**Multi-Class Classification Problem**
A problem where you classify an example as belonging to one of more than two classes.
The problem is framed as predicting the likelihood of an example belonging to each class.
•**Output Layer Configuration**: One node for each class using the softmax activation function.
•**Loss Function**: Cross-Entropy, also referred to as Logarithmic loss.



# NN Theory:
# Cost function

What happend when we use a sigmoid as cost function?

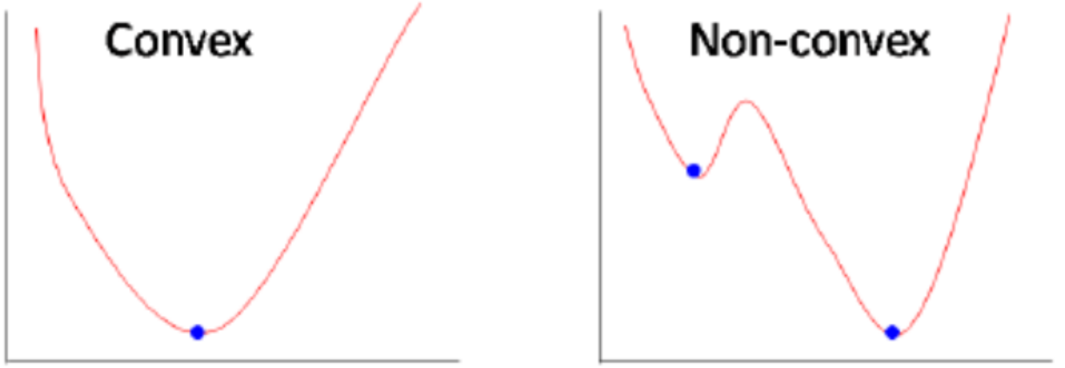
Sigmoid function is a “non-convex” function which means that there are multiple local minimums. So it’s not guaranteed to converge (find) to the global minimum. What we need is “convex” function.

Why?

Because the optimization algorithm used to find the minimum (error), is the **gradient descendent**.

This algorithm is built to move towards the direction of the solution in order to find the minimum.

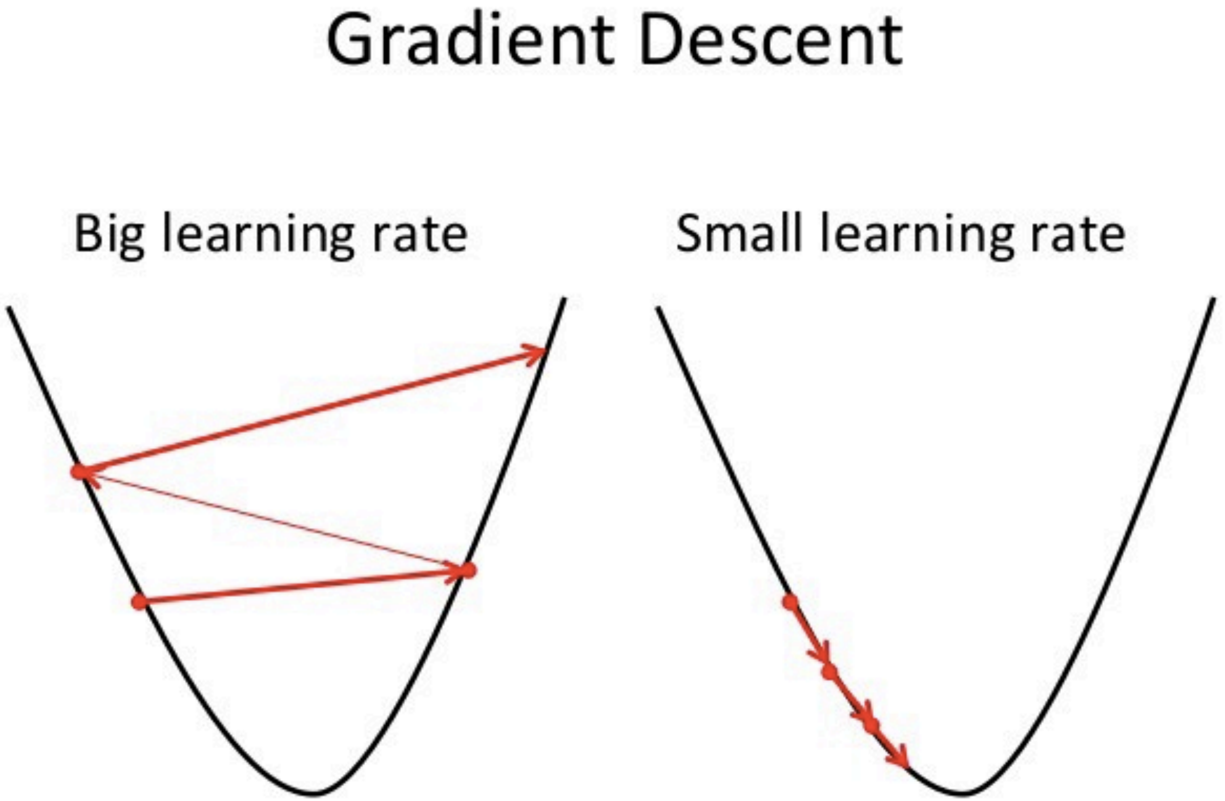
It is for this reason that we will use log function (as in cross-entropy).



# NN Theory:
# Learning Hyperparameters

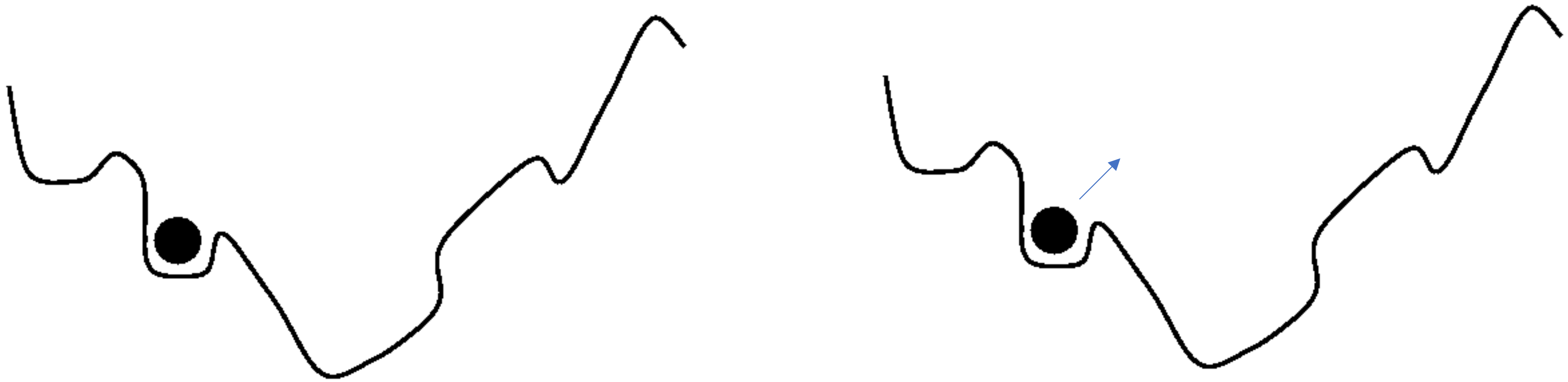
Now we can discuss hyperparameters connected to the Learning.

- **Learning rate**: The learning rate defines how quickly a network updates its parameters. Low learning rate slows down the learning process but converges smoothly. Larger learning rate speeds up the learning but may not converge. Usually a decaying Learning rate is preferred.



# NN Theory:
# Learning Hyperparameters

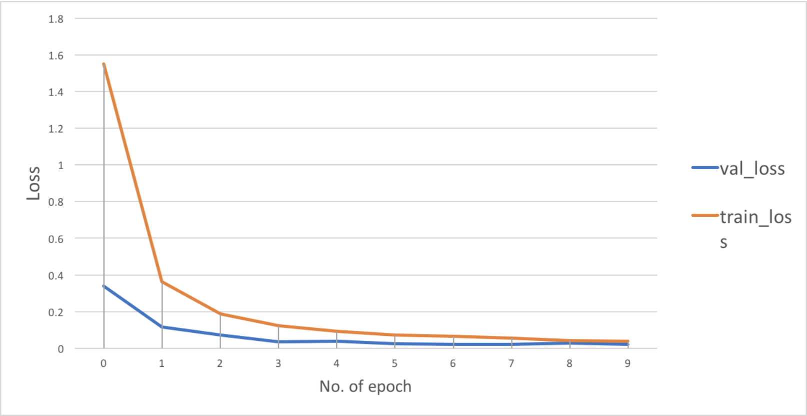
- **Momentum**: Momentum helps to know the direction of the next step with the knowledge of the previous steps. It helps to prevent oscillations. A typical choice of momentum is between 0.5 to 0.9. The momentum help the gradient descendent method to «run away» from local minimum. This means: move along a worst solution in order to reach the global minimum.

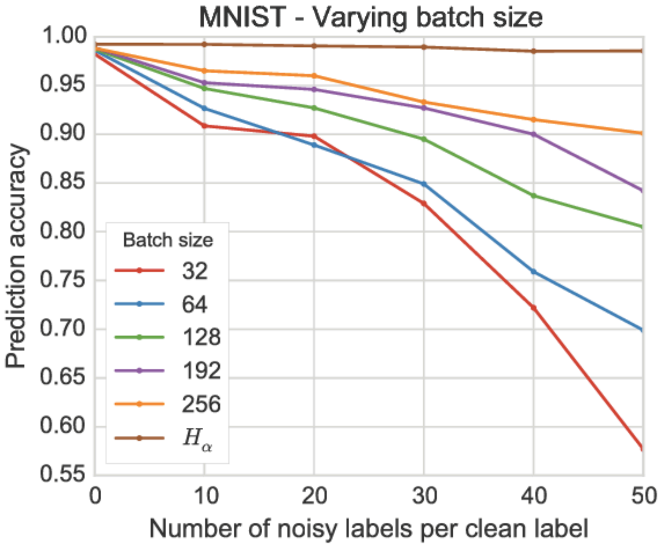


# NN Theory:
# Learning Hyperparameters

- **Number of epochs:** Number of epochs is the number of times the whole training data is shown to the network while training. Increase the number of epochs until the validation accuracy starts decreasing even when training accuracy is increasing(overfitting).

- **Batch size**: Mini batch size is the number of sub samples given to the network after which parameter update happens. A good default for batch size might be 32. Also try 32, 64, 128, 256, and so on.

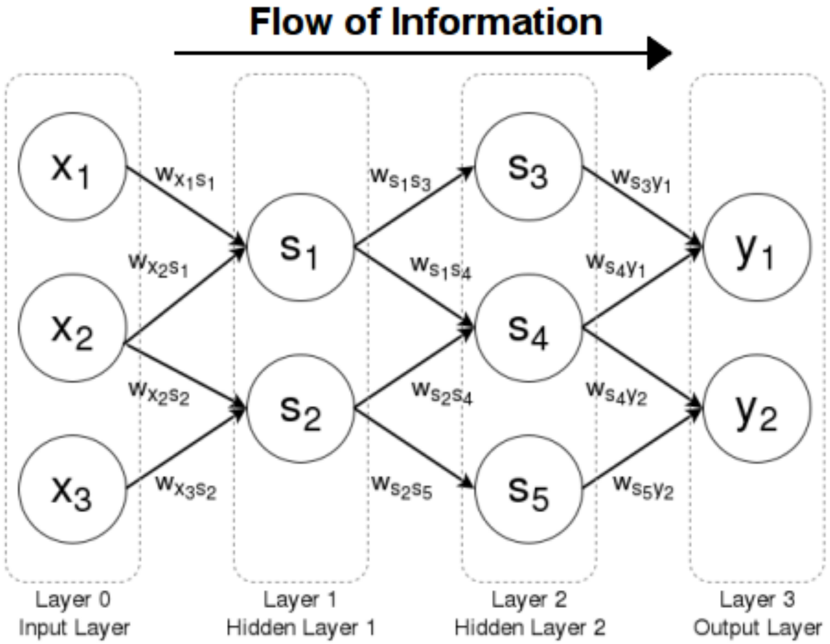




# How the Neural Network predicts?

# NN Theory: Feedforward

In a multilayer feedforward ANN, the neurons are ordered in layers, starting with an input layer and ending with an output layer. Between these two layers are a number of hidden layers. Connections in these kinds of network only go forward from one layer to the next.

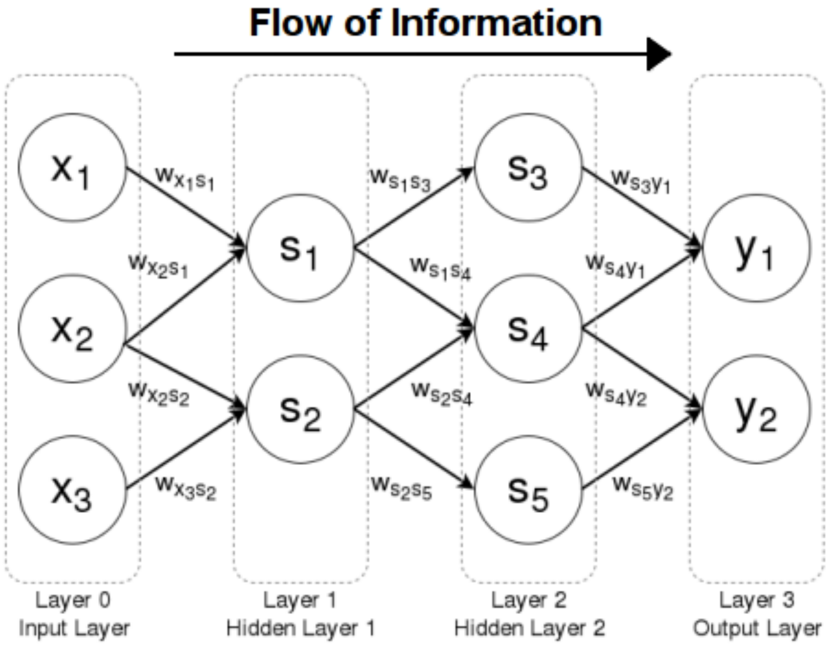


# NN Theory: Feedforward

Step in a feedforward network:

- An input is presented to the input layer
- The input is propagated through all the layers
- The input reaches the output layer

In feedforward network there are no loop!



# NN Theory:
# Feedforward details

This process of Forward propagation is actually getting the Neural Network output value based on a given input. This algorithm is used to calculate the cost value.

As we already discuss, we have this connection between neurons:

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

After we got the h(x) value (hypothesis) we use the Cost function equation to calculate the cost for the given set of inputs.

$\theta$ are the weights.

Here we can notice how forward propagation works and how a Neural Network generates the predictions.

$$x = a^{(1)}$$

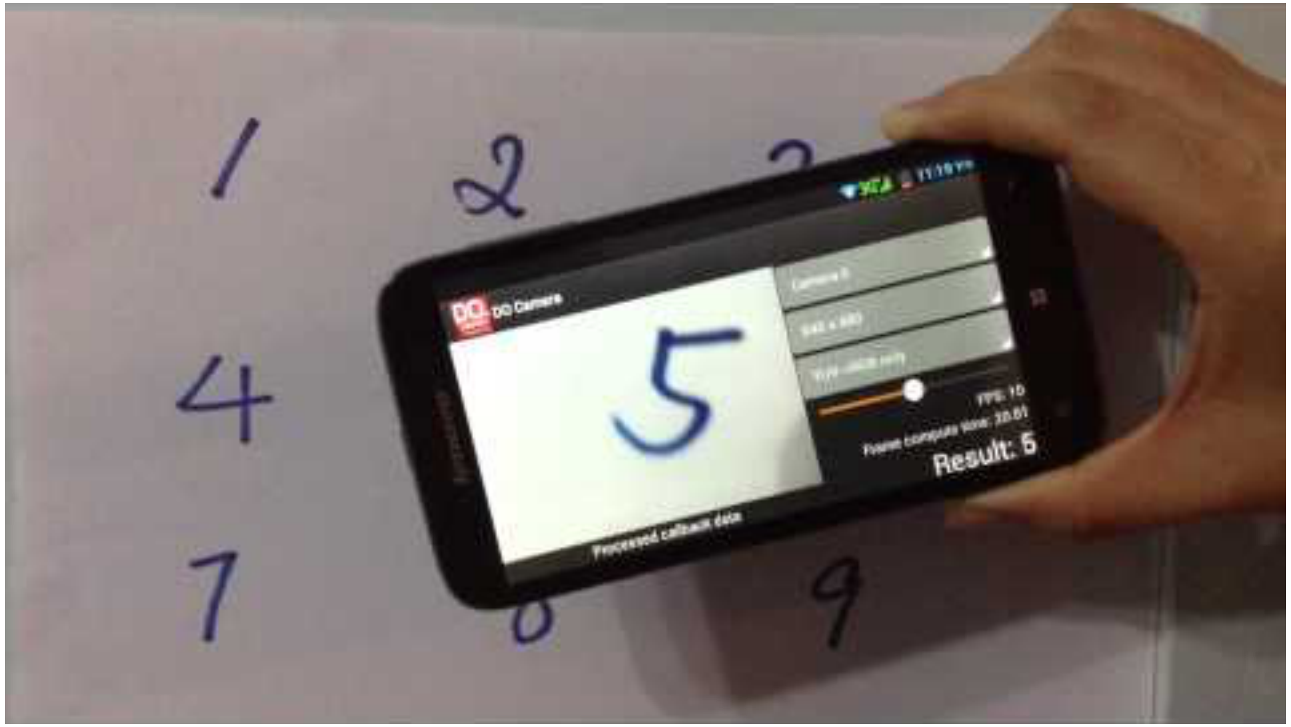
$$z^{(j+1)} = \theta^{(j)} a^{(j)}$$

$$a^{(j+1)} = \sigma(z^{(j+1)})$$

$$h_\theta(x) = a^{(L)} = \sigma(z^{(L)})$$

# NN Theory:
# Feedforward application

Hand-written Character Recognition



https://www.youtube.com/watch?v=lkuucfUW36k

# In the next lesson:
# Backpropagation and Convolutional Neural Networks