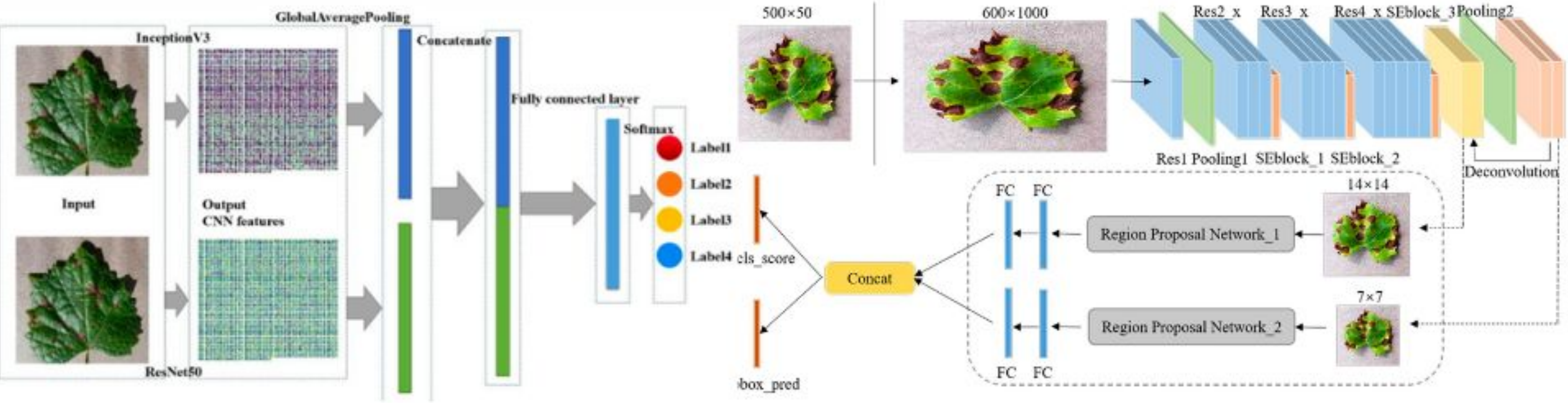


Identifying Diseases in Plants with Image Categorization in Edge Devices

Delivered By : Amal Mathew

Feel free to connect with me here : LinkedIN

Extensive Syllabus

Session 1: Introduction to Deep Learning

- Basics of Machine Learning & Deep Learning
- Why Deep Learning
- Linear and Nonlinear functions
- What are learnable parameters?
- Forward propagation and Backward pass in DL
- Loss function and its significance
- Implementation of a simple Neural Network

Session 2: Enhancing the Learning process

- Bias-Variance Tradeoff
- What is Underfitting and Overfitting
- Various optimizers for deep learning
- Implementation of an Image Classifier using FNN(keras)

Session 3: Convolution Neural Networks

- What are CNNs and Why they are introduced
- Filters, Channels, Pooling layers
- Learning process in CNNs
- Data Augmentation
- Pretrained Models
- Transfer Learning
- Implementation of an Image Classifier using CNN(keras)

Session 4: Real-world case study (Edge Deployment)

- What are edge devices
- Model Quantization and the significance
- Different Quantization Techniques
- Building an Image Classifier to Identify disease in the plants
- Model Conversion to TFlite format
- Edge Compiler and quantized model

Course Structure

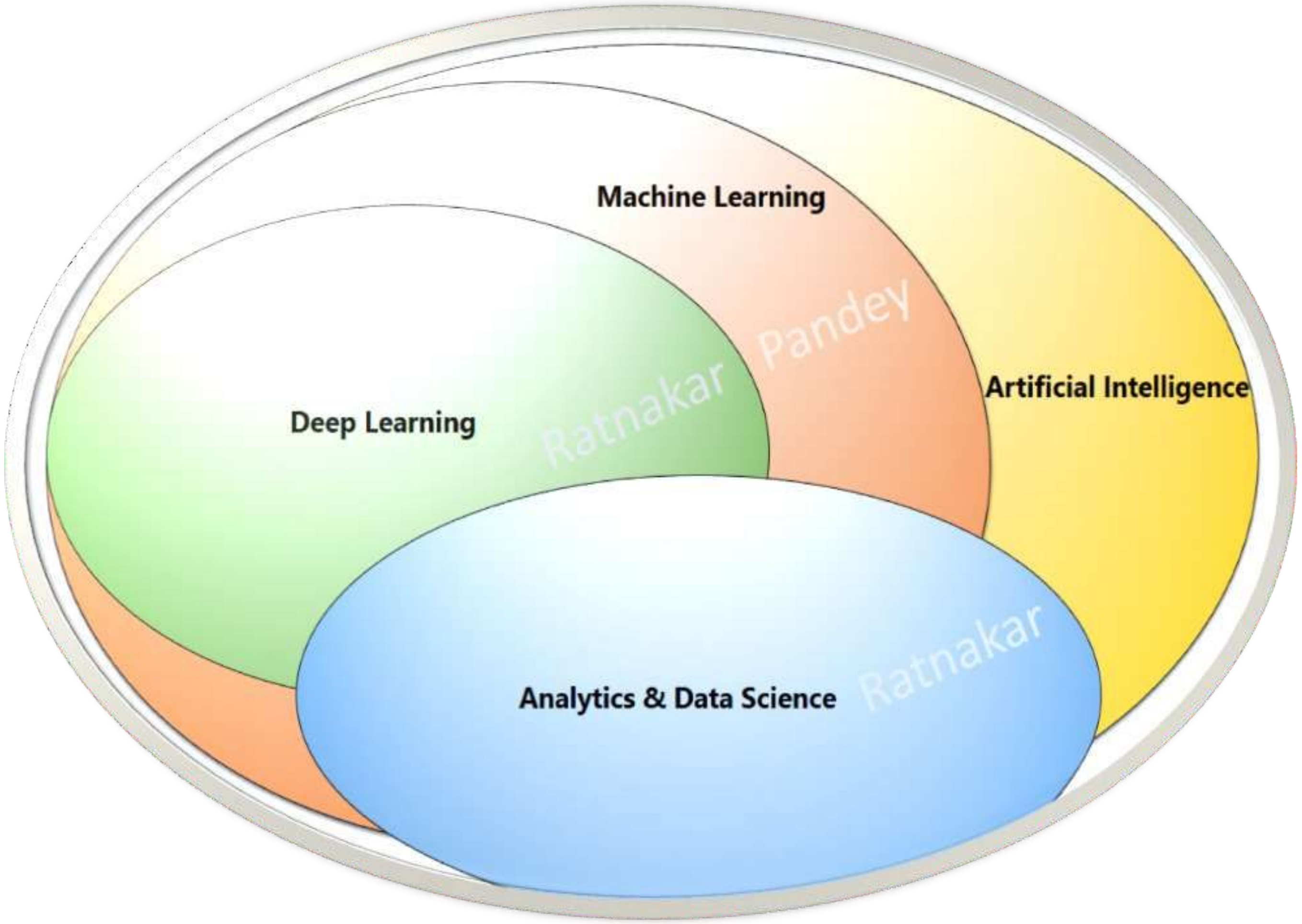
1. Domain Knowledge + Problem Solving
2. Lecture (Understanding the concepts)
3. Hands on Coding
4. Reporting
5. Lectures per week
6. Guided Lab
7. Unguided Project

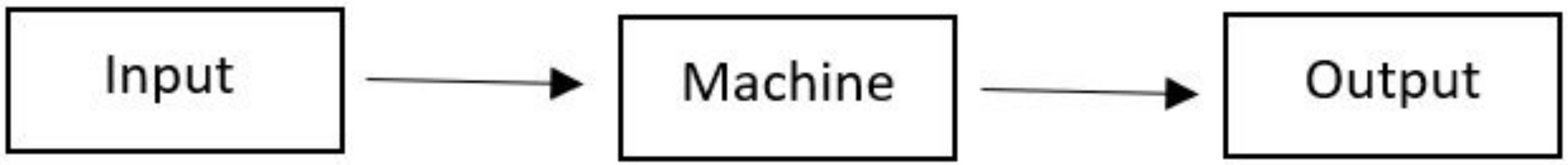
Photo by Jason Leung on Unsplash

Introduction to Deep Learning – Week 1

- Artificial Intelligence, Machine Learning & Deep Learning
- Types of Machine Learning
- Types of Data (Structured & Unstructured)
- Regression and Classification
- Why Deep Learning?
- What is a Neuron? What are Neural Networks?
- linear functions and Nonlinear functions
- Weights and Biases
- Forward & Backward Propagations
- Loss function
- Parameter Initialization & Prediction
- Implementation of a simple Neural Networks using Numpy and Keras(Tensorflow)
- Implementation of a simple Neural Networks using Pytorch

Artificial Intelligence, Machine Learning & Deep Learning





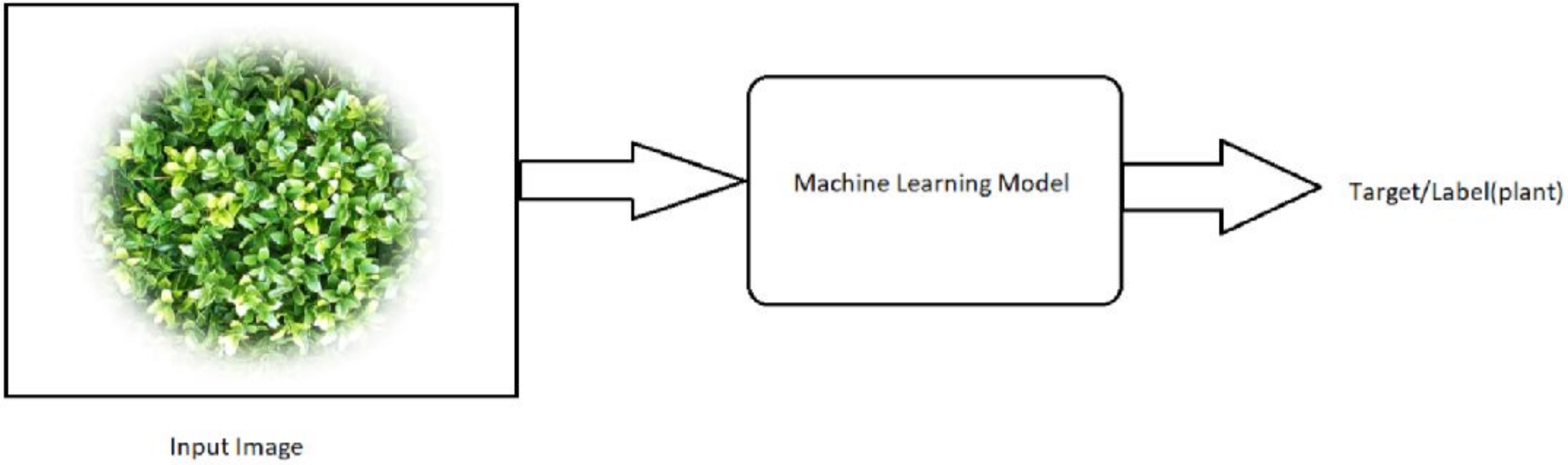
Machine Learning Types

The major classification of ML is based on the learning process itself along with its outcome. The predominant grouping includes,

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

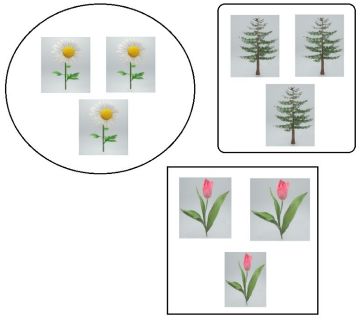
The input to any ML algorithm is called predictors/features/variables and the output from the algorithm is referred to as a target/label.

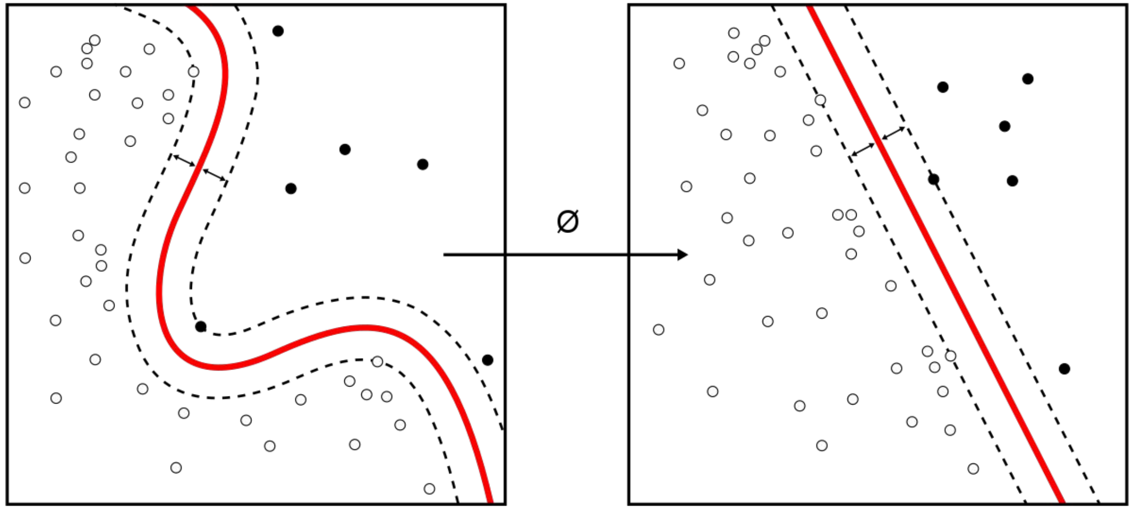
Supervised Learning:



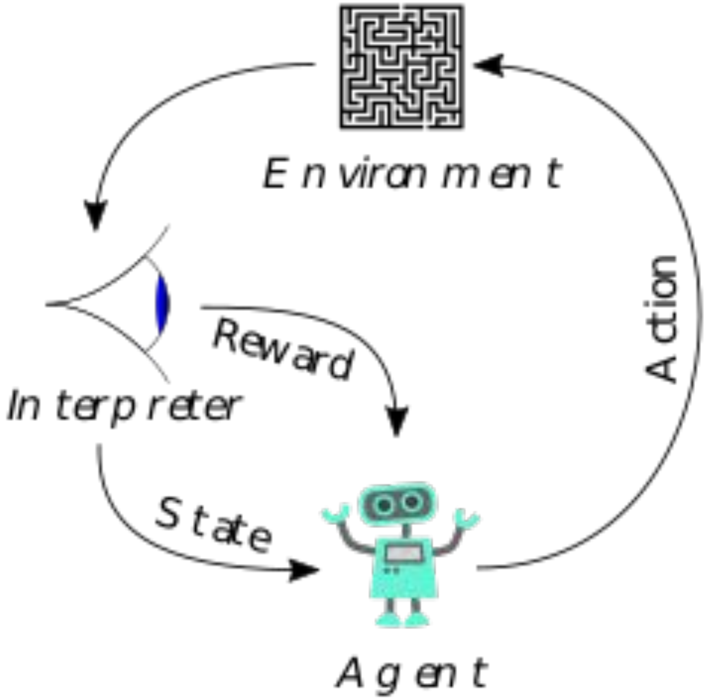
Machine Learning Types

Unsupervised Learning —





Reinforcement Learning —



Data types

Data is broadly classified into two categories:

- Structured data
- Unstructured data

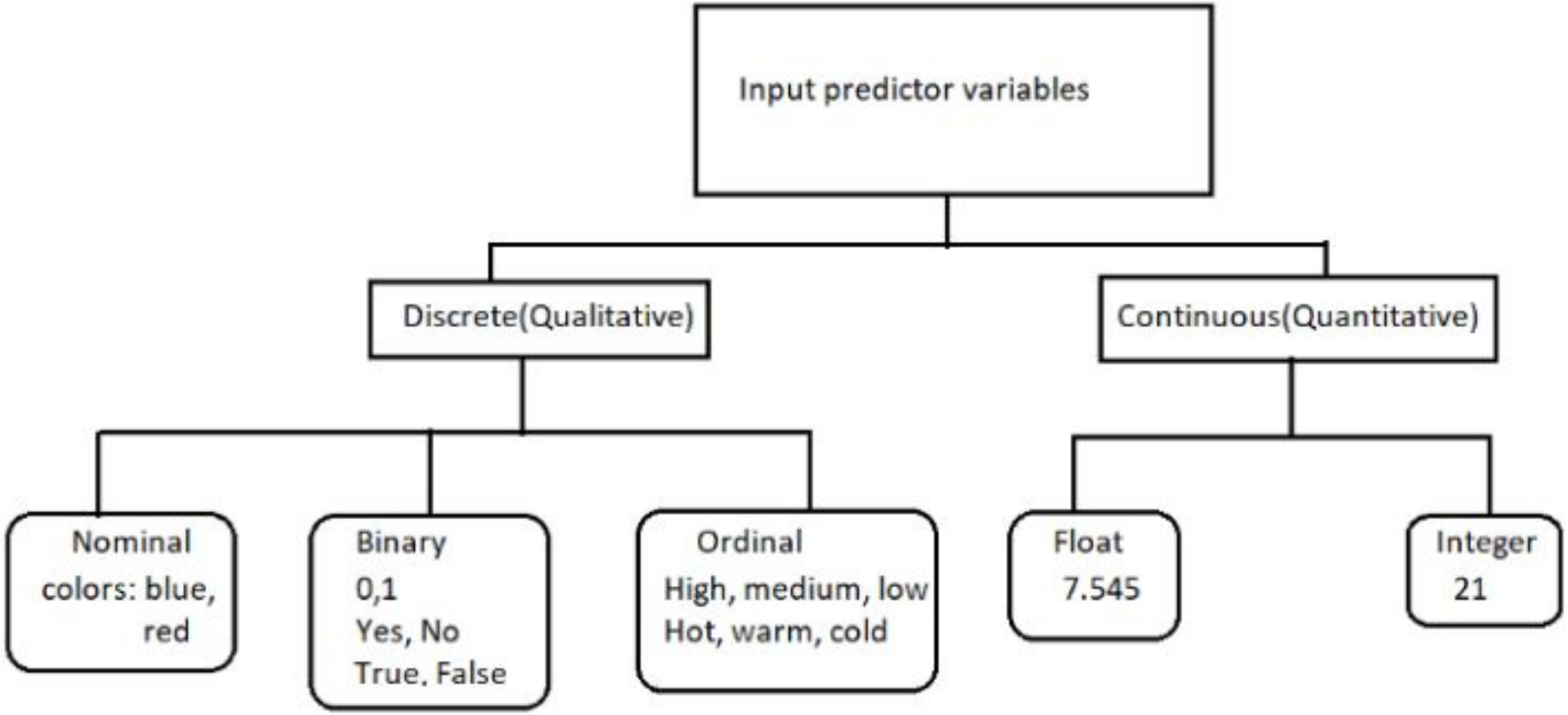
Structure data:

Fence	MiscFeatu	MiscVal	MoSold	YrSold	SaleType	SaleCondition
MnPrv	NA	0	6	2010	WD	Normal
NA	Gar2	12500	6	2010	WD	Normal
MnPrv	NA	0	3	2010	WD	Normal

Unstructured data: l


```
productCode;orderNumber;quantityOrdered;priceEach;orderLineNumber;customerNumber;quantityInStock;buyPrice;MSRP;month;year;profit
S10_1678;287441.0;1057.0;2384.8799999999997;152.0;6498.0;222124;1366.6799999999992;2679.5999999999995;203.0;56108.0;1312.9200000000005
S10_1949;287289.0;961.0;5524.66;162.0;7416.0;204540;2760.239999999999;6000.400000000003;200.0;56107.0;3240.159999999998
S10_2016;287432.0;999.0;3080.5299999999993;140.0;6885.0;185500;1931.72;3330.320000000001;202.0;56108.0;1398.6000000000006
S10_4698;287436.0;985.0;4824.07;142.0;6337.0;156296;2548.56;5422.479999999998;191.0;56109.0;2873.92
S10_4757;287364.0;1030.0;3478.8799999999997;195.0;8362.0;91056;2399.0400000000004;3808.0;196.0;56108.0;1408.959999999999
S10_4962;287304.0;932.0;3690.5700000000006;166.0;7072.0;190148;2895.760000000001;4136.7199999999975;202.0;56107.0;1240.9600000000003
S12_1099;277075.0;933.0;4656.049999999999;179.0;7313.0;1836;2574.1800000000003;5253.389999999999;195.0;54103.0;2679.21
S12_1108;276928.0;1019.0;5051.61;182.0;6109.0;97713;2580.9300000000003;5610.600000000003;189.0;54103.0;3029.6700000000005
S12_1666;287296.0;972.0;3453.6599999999994;171.0;7039.0;44212;2181.200000000001;3826.760000000001;201.0;56107.0;1645.5599999999995
S12_2823;287447.0;1028.0;3700.7300000000005;164.0;6971.0;279916;1855.5599999999997;4217.359999999998;194.0;56109.0;2361.799999999999
S12_3148;276923.0;963.0;3719.6199999999994;178.0;6084.0;186462;2406.7800000000007;4079.159999999999;188.0;54103.0;1672.380000000001
S12_3380;277079.0;925.0;2879.6300000000006;169.0;7350.0;246321;2029.3200000000008;3170.880000000001;195.0;54103.0;1141.5599999999995
S12_3891;276931.0;965.0;4271.910000000001;144.0;5950.0;28323;2242.35;4671.540000000002;190.0;54103.0;2429.1899999999996
S12_3990;277077.0;900.0;1911.22;223.0;7380.0;152901;861.8399999999998;2154.5999999999995;195.0;54103.0;1292.7600000000002
S12_4473;287330.0;1056.0;2909.259999999999;174.0;6450.0;171500;1559.6000000000008;3318.0;205.0;56107.0;1758.3999999999992
S12_4675;277085.0;992.0;2802.9999999999995;204.0;7735.0;197721;1585.7100000000003;3109.3199999999993;195.0;54103.0;1523.6100000000001
S18_1097;287292.0;999.0;2954.0900000000006;165.0;7321.0;73164;1633.2399999999996;3266.760000000001;200.0;56107.0;1633.5199999999995
S18_1129;277103.0;947.0;3275.25;165.0;7135.0;107325;2254.7700000000004;3821.5799999999995;195.0;54103.0;1566.8099999999995
S18_1342;287264.0;1111.0;2578.7799999999997;172.0;6865.0;243404;1697.3599999999988;2876.719999999998;198.0;56107.0;1179.3599999999997
S18_1367;287267.0;960.0;1340.1899999999998;157.0;7391.0;241780;679.2799999999999;1509.4800000000002;198.0;56107.0;830.1999999999996
S18_1589;256370.0;914.0;2792.4400000000005;160.0;6375.0;226050;1649.0000000000005;3111.000000000001;181.0;50094.0;1462.0000000000002
S18_1662;287387.0;1040.0;3894.9500000000007;172.0;9255.0;149240;2163.56;4415.32;197.0;56108.0;2251.7600000000007
S18_1749;256169.0;918.0;3842.0;176.0;7192.0;68100;2167.5000000000005;4250.0;172.0;50093.0;2082.4999999999995
S18_1889;277087.0;972.0;1850.3100000000002;175.0;7431.0;238302;1455.3000000000004;2079.0;195.0;54103.0;623.7000000000003
S18_1984;277105.0;917.0;3493.7099999999987;155.0;7137.0;263844;2535.03;3840.75;195.0;54103.0;1305.7199999999996
S18_2238;287333.0;986.0;4036.0;184.0;6483.0;132272;2842.2800000000016;4584.439999999999;205.0;56107.0;1742.1600000000003
S18_2248;256171.0;832.0;1357.3199999999997;160.0;7176.0;13500;832.4999999999997;1513.4999999999995;172.0;50094.0;681.0000000000001
S18_2319;287325.0;1053.0;3112.4600000000005;215.0;6300.0;231224;2096.0799999999986;3436.44;205.0;56107.0;1340.3599999999997
S18_2325;287246.0;957.0;3198.6099999999997;180.0;7452.0;261912;1637.4400000000003;3559.640000000002;198.0;56107.0;1922.200000000001
S18_2432;287310.0;998.0;1571.5199999999995;206.0;6512.0;56504;697.7599999999999;1701.5599999999997;202.0;56107.0;1003.8000000000004
S18_2581;287412.0;917.0;2112.86;175.0;7576.0;27776;1372.0;2365.44;200.0;56108.0;993.4400000000004
```

Types of inputs/predictor variables



Machine Learning & Deep Learning

Machine learning uses various algorithms(linear regression, logistic regression, decision trees, SVM) to predict the target. The input features or predictors are fed into these algorithms and trained to produce machine learning models(basically a trained algorithm). The target or predicted outcome decides the accuracy of the model. Higher accuracy implies the actual value and predicted values are closer.

Example of Linear regression:

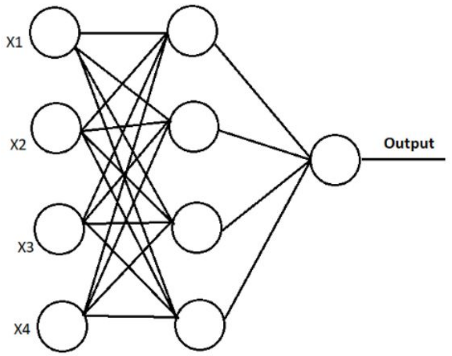
y = w1x1 + w2x2 + b

where y – predicted output,

x1&x2 – input features

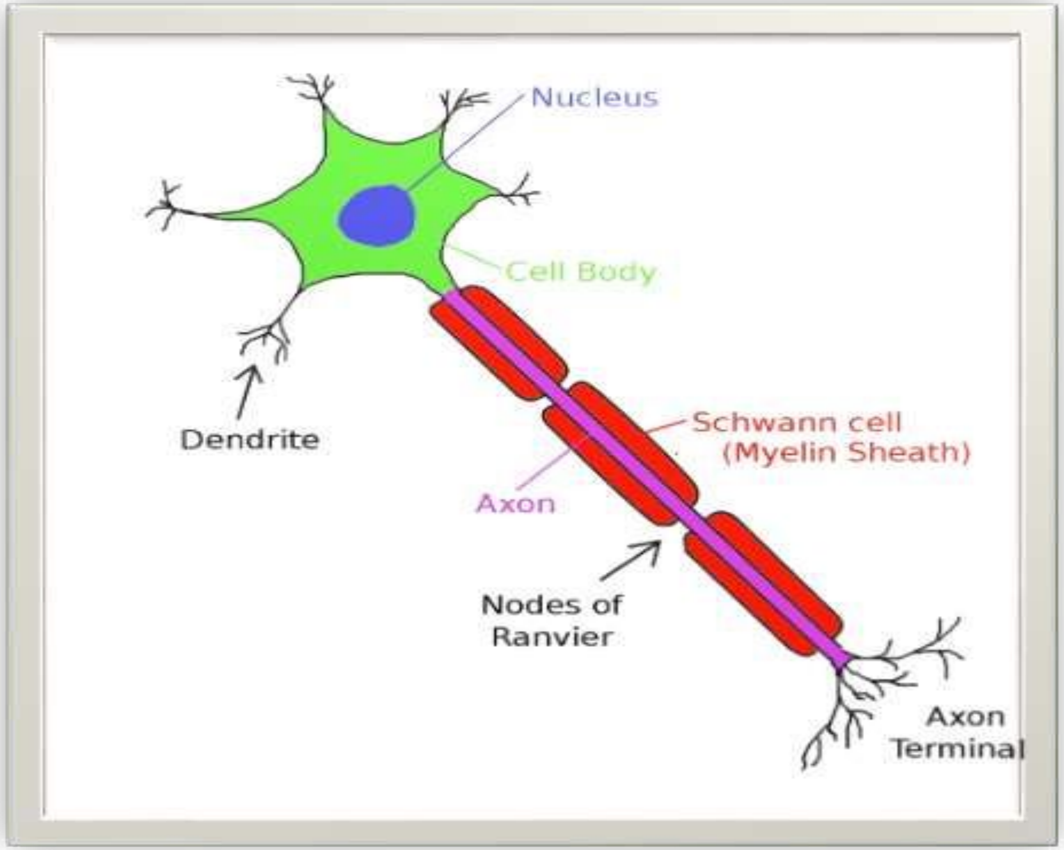
w1,w2&b – parameters learnt by the model

Deep Learning is a subfield of machine learning where the main focus is on Unstructured data(Images and texts). On the contrary to machine learning(which uses prebuilt algorithms), deep learning refers to training a large neural network. The large neural network is formed by the combination of many hidden layers. If we take the case of image classification using neural networks, the first hidden layer tries to find the edges and horizontal lines, and then the subsequent layer finds parts of the face like eyes and nose. Consecutively, the final layer detects the face. The initial phases of the neural network take the responsibility of figuring out high-level details in the images such as edges, whereas the deeper and latter layers take the responsibility of identifying complex patterns in the picture.



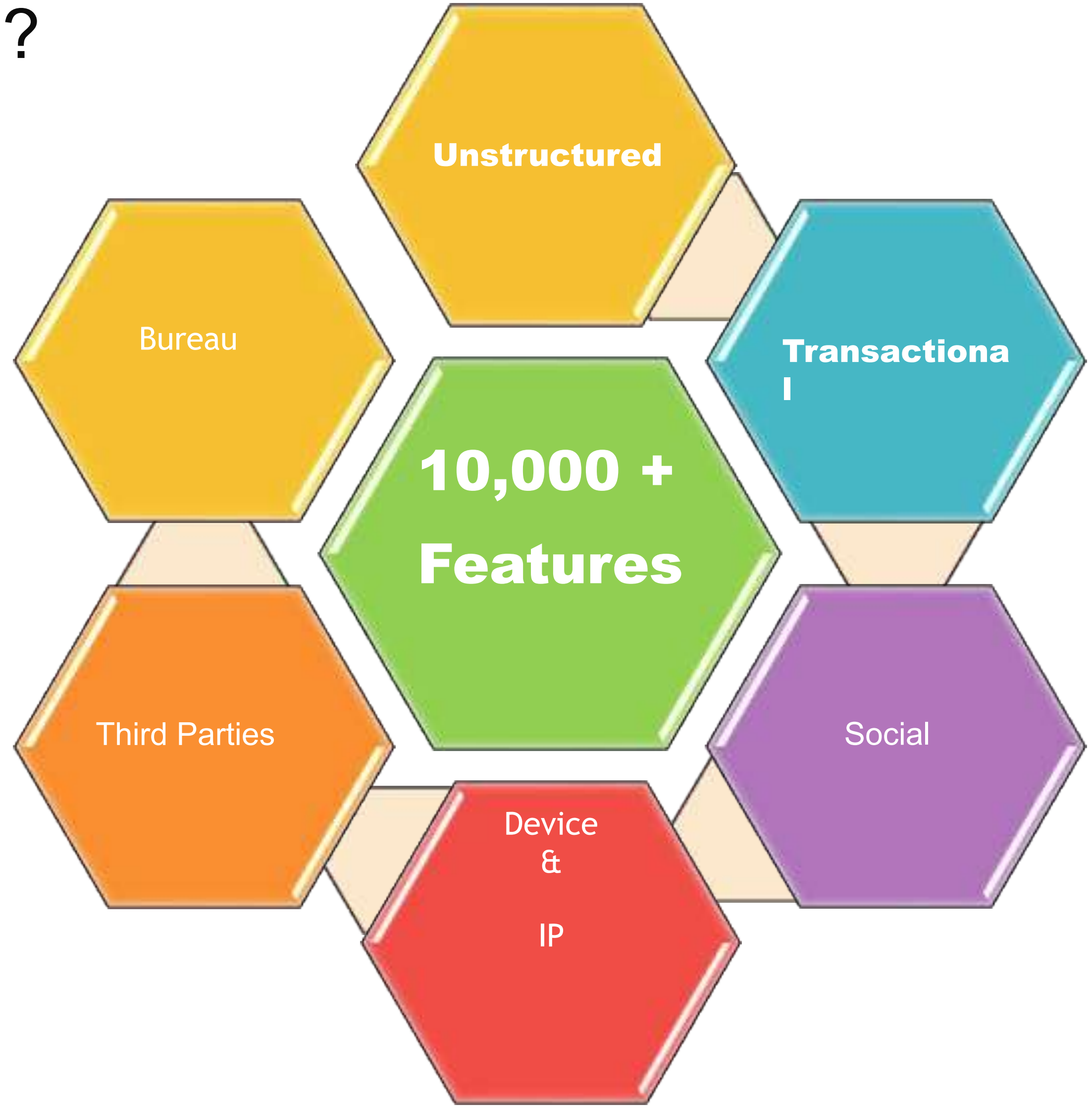
What is Deep learning ?

- Deep learning is a sub field of Machine Learning that very closely tries to mimic human brain's working using neurons.
- These techniques focus on building Artificial Neural Networks (ANN) using several hidden layers.
- There are variety of deep learning networks such as Multilayer Perceptron (MLP), Autoencoders (AE), Convolution Neural Network (CNN), Recurrent Neural Network (RNN)



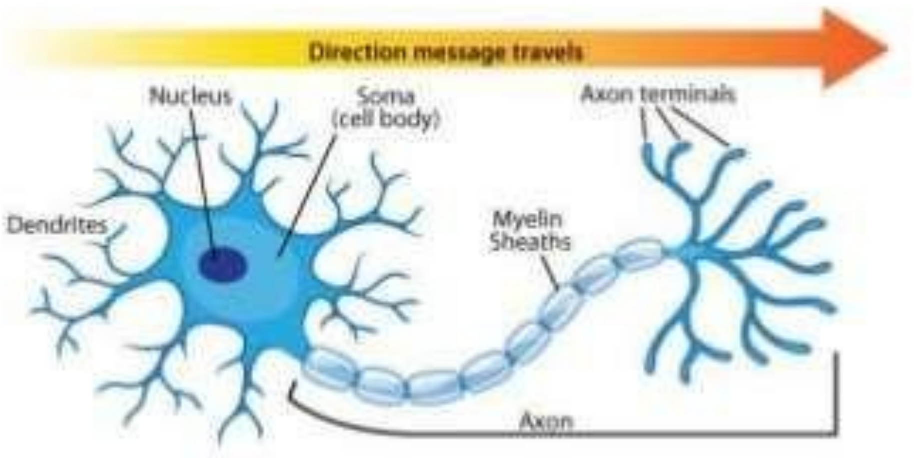
Why Deep Learning is Growing ?

- Uncover hard to detect patterns (using traditional techniques) when the incidence rate is low
- Find latent features (super variables) without significant manual feature engineering
- Real time fraud detection and self learning models using streaming data (KAFKA, MapR)
- Ensure consistent customer experience and regulatory compliance
- Higher operational efficiency

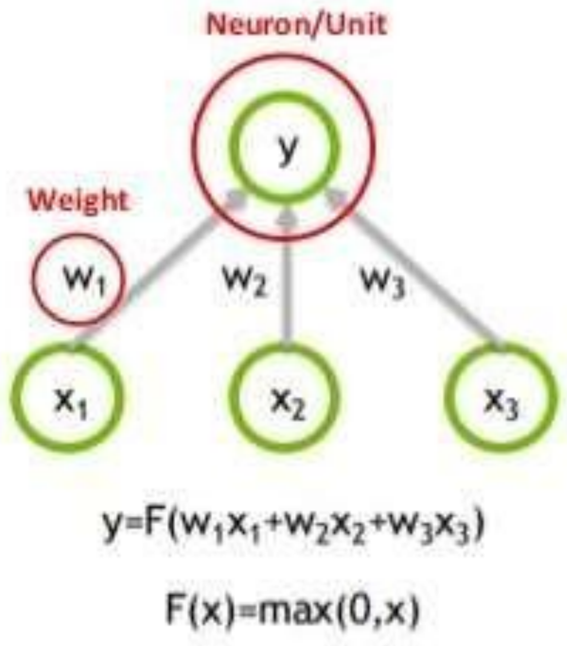


Artificial Neural Network & Natural Neural Network

Human Brain

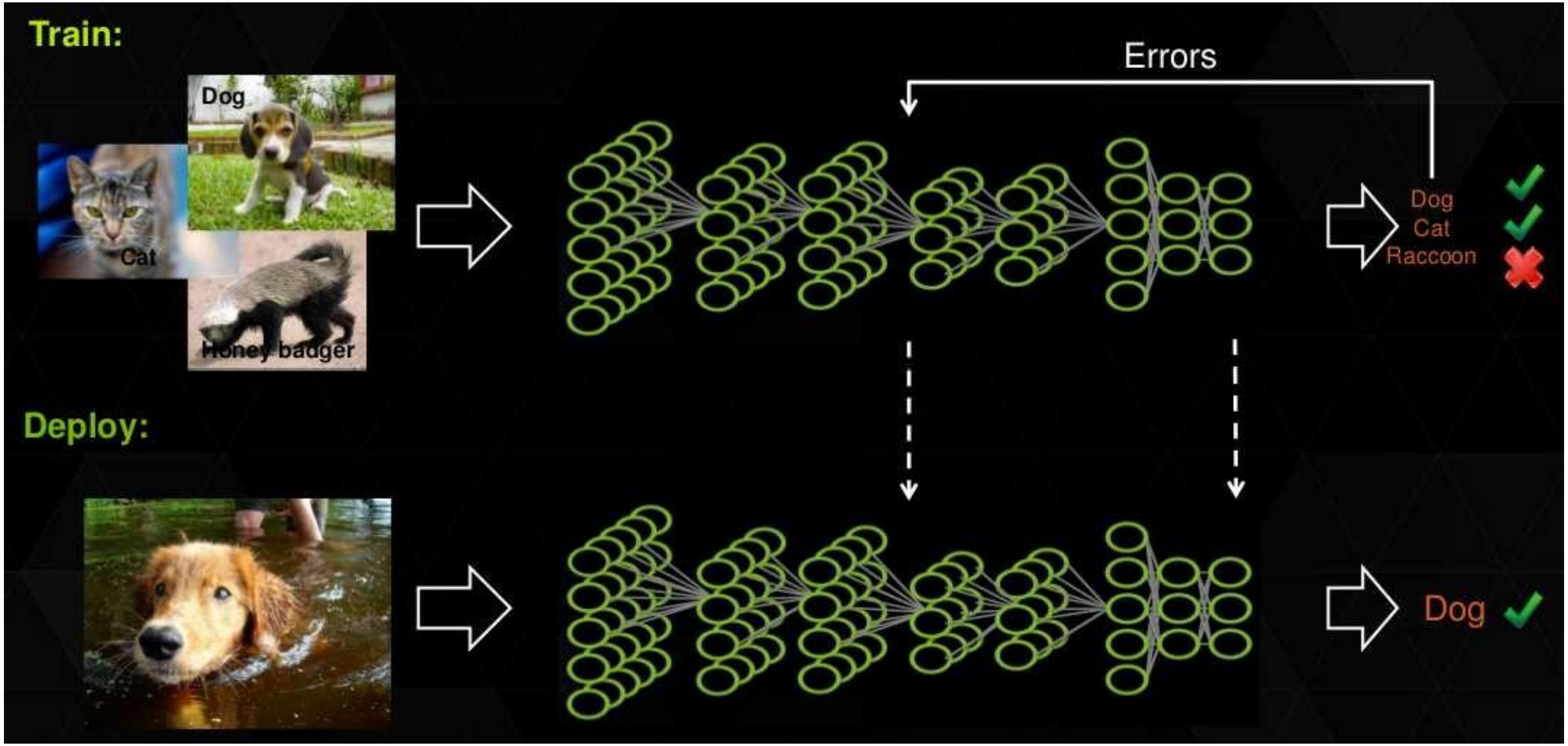


Biological Neuron



Artificial Neuron

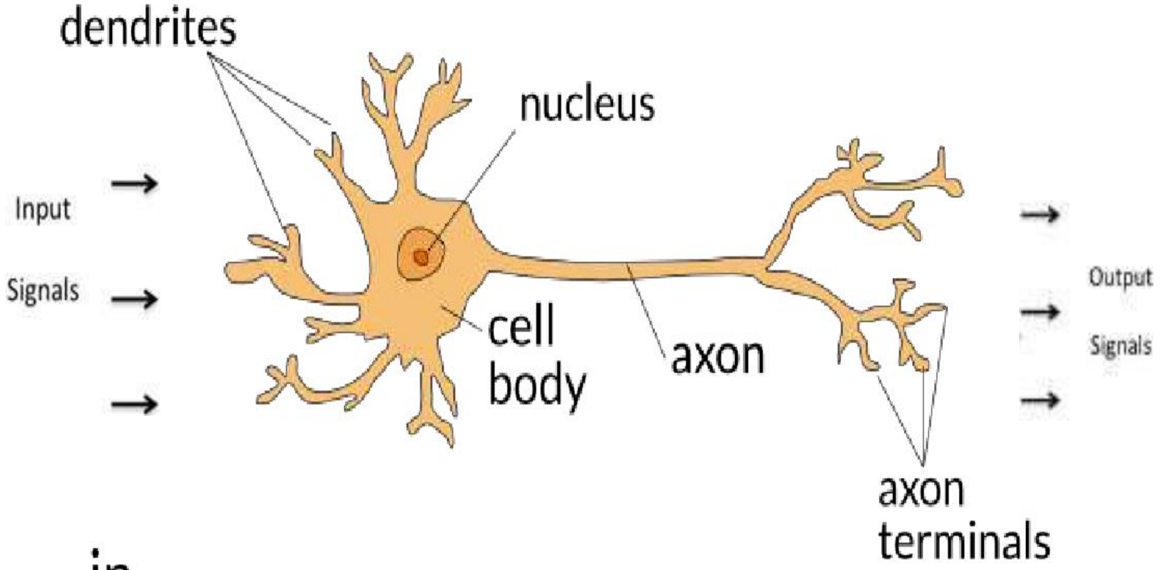
Artificial Neural Network & Natural Neural Network

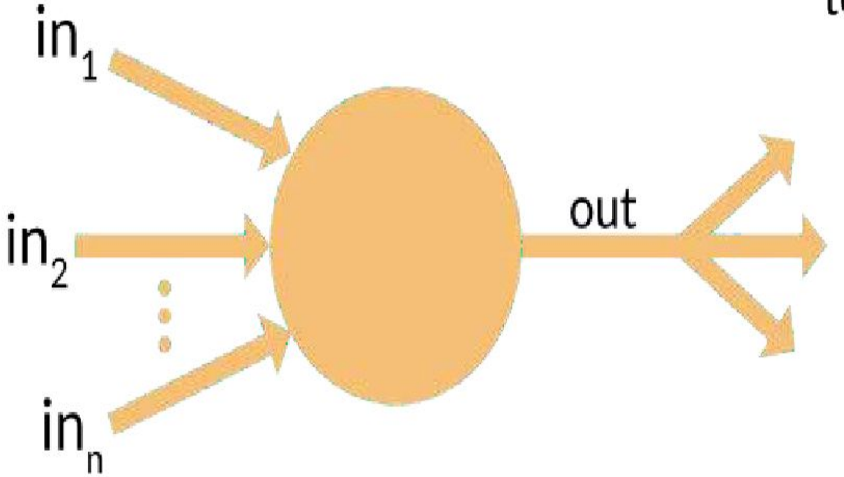


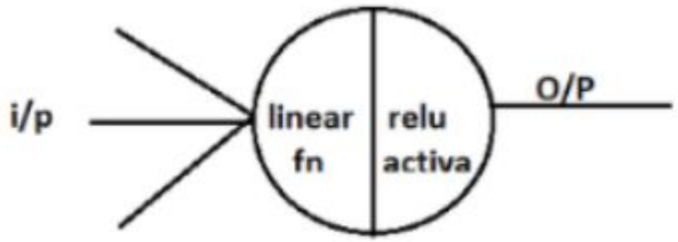
Building Blocks of Neural Network

What is a Neural Net?

- Motivation: create an Artificial Neural Network to solve problems the same way a human brain would







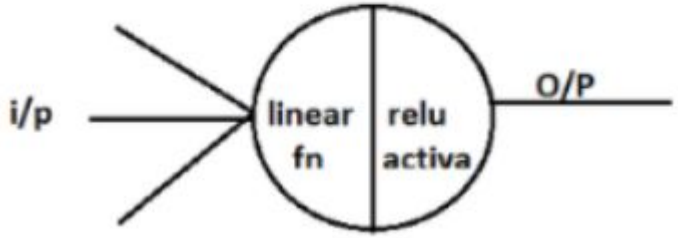
Artificial Neural Network & Natural Neural Network

***Why DL over ML?* :-** The Machine learning algorithms are systematic (i.e) they follow a set of rules to forecast the target. These algorithms demand the input features be properly engineered to produce accurate predictions. Another shortcoming with these algorithms is its inability to handle high dimensional data(requires complex function).

Think of an image which is represented by an enormous number of pixels, as the resolution of the picture increases so as the count of the pixels. So, one single training instance would be a combination of all these pixels(**2,073,600 features).** Due to the curse of dimensionality, the algorithms will not be able to make the relation out of these complex dataset.

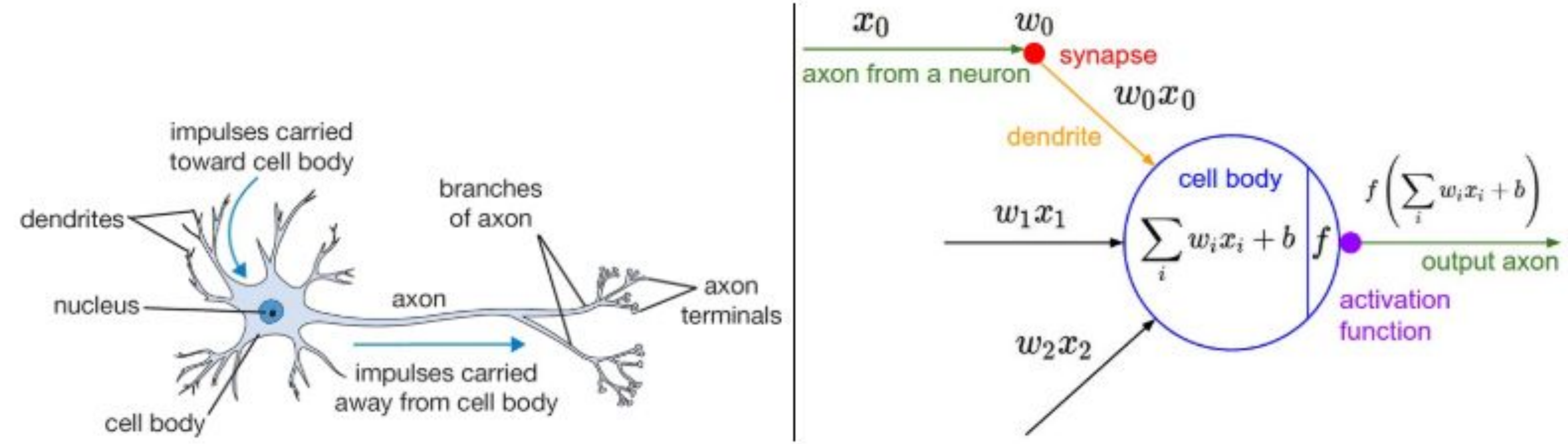
Deep learning deals with unstructured data such as images and texts in a superior way (i.e) it extracts the significant features through the learning process rather than hand-engineering. The deep neural network is comprised of multiple layers of neurons with each layer containing a non-linear function(ReLU). When we connect all these non-linear layers it will end up in forming a highly complex function as a whole.

What is a single neuron?:- A neuron derived its name and meaning from the neuron in the brain. It can be thought of as an on/off switch which either passes the input data to the next layer(on) or blocks the information(off). In terms of DL, it is referred to as Artificial neuron or perceptron.



The input passed to the neuron will be a set of input features which gets multiplied by the weights through linear function. This is followed by ReLU activation function(non-linear) to create the weighted output.

Linear Functions & Activations



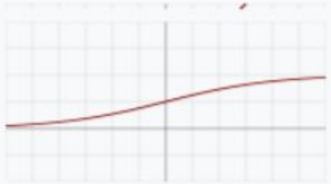
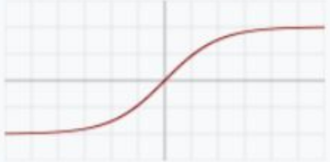
A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Linear Functions & Activations

Recollecting the equation for the linear regression y = w1*x1 + w2*x2 + w3*x3 …. wn*xn + b where ‘w’ represents weights of the model and ‘b’ indicates the bias. The ReLU activation function outputs the values as max(y,0). So if ‘y’ is a positive number, then the result will be y or else it will be zero.

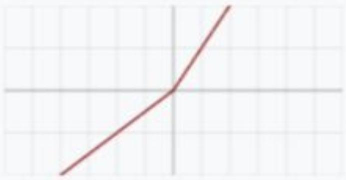
Non-linear functions (Sigmoid, Tanh and ReLU):-

ref : wiki

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f(x)(1-f(x))$	$(0,1)$
tanh		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1,1)$
Rectified linear unit (ReLU)[11]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max\{0,x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0,\infty)$

Non - Linear Functions & Activations

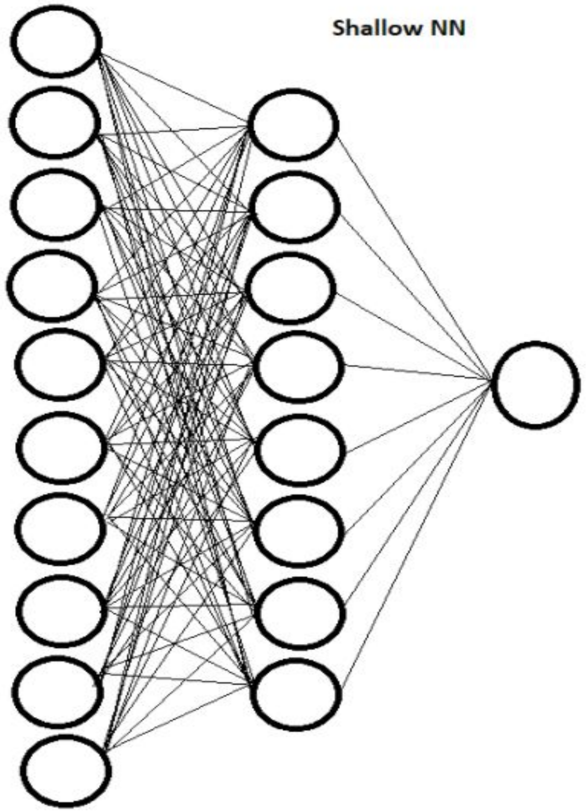
- ***Why ReLU is preferred?***
- ***Leaky Relu - a relu variant !***

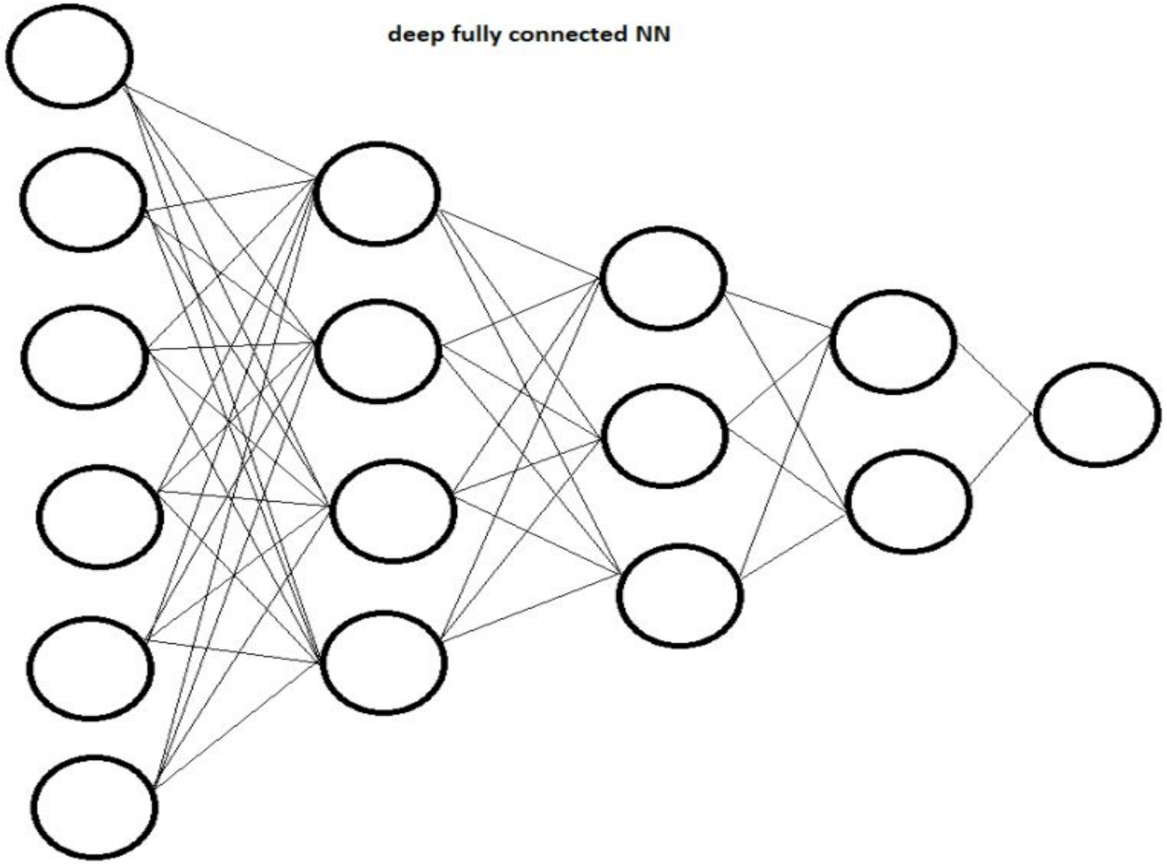
Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$
Leaky rectified linear unit (Leaky ReLU)[15]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$

A simple & complex neural network

Note1: All the layers between input and output is termed as hidden layers

Note2: We use non-linear functions as activations instead linear, because the combination of all the linear functions would result in a scaled version of one linear function.





When to use Deep Learning/Machine Learning?

- DL requires a huge amount of data to train the deep neural networks from scratch. Since it is computationally heavy, it is necessary to incorporate GPU’s and TPU’s in contrary to the traditional CPU’s.
- ML Algorithms can be used for structured data with well-defined features. When there is only a limited quantity of information available, then this approach can be chosen.
- One of the notable observations between ML & DL is, the ML algorithms perform effectively with an increase in the incoming data but the accuracy does not get improved after a certain limit. Whereas DL’s performance keeps on improving with the magnitude of the training dataset.
- Conversely, even if we build a complex Artificial Neural Network(ANN), the outcomes are not productive if the input instances are less in counts. Thus, DL model prediction power greatly depends on the amount of the relevant training samples.

Understanding Learning process with simple linear regression

Let’s pick the simplest form of Supervised Learning Algorithm (i.e) Linear Regression. The intent of the algorithm is to predict a numerical target(Regression-based) given the list of input predictors. Before we analyze the algorithm, let’s recollect the high school math for the equation of a straight line which is y=mx+c where ‘m’ is the slope and ‘c’ is the y-intercept.

The slope(m) of a straight line is calculated by using the formula y2 — y1 / x2 — x1(rise/run), how much change in y happened for the change in x. If the rate of change in both coordinates is same, then the value of slope(m) = 1. The y-intercept(c) signifies it is not always necessary for the ‘y’ becoming zero when x is zero. Even when x is zero, y can take either positive or negative values. y-intercept also indicates the point at which the line crosses y-axis when x=0.

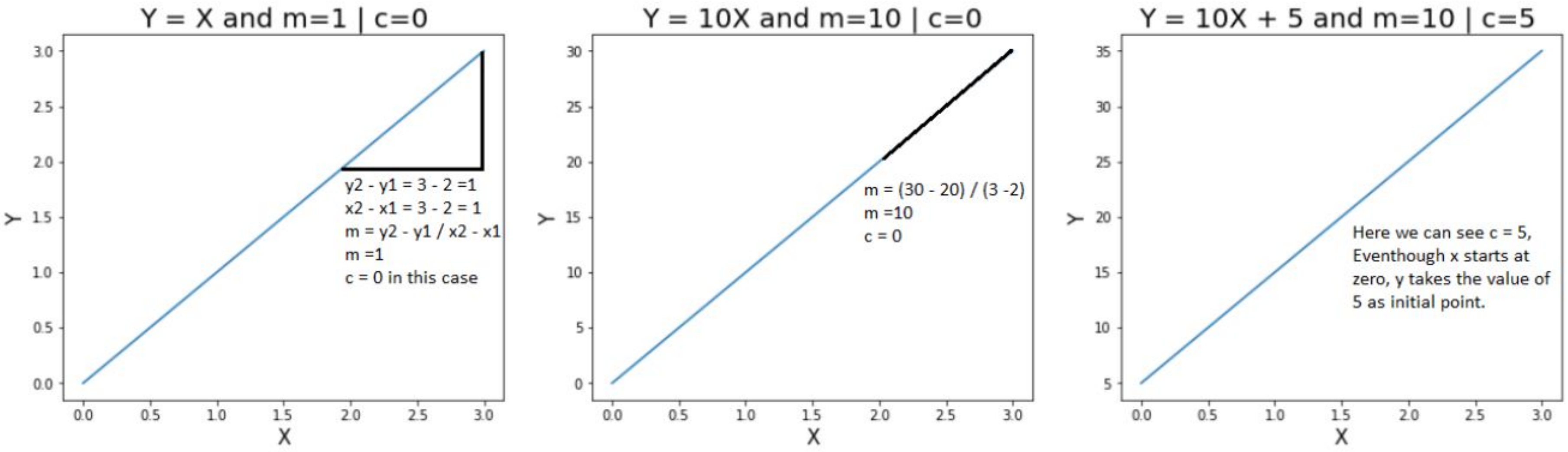


Fig1 — shows the equation of a straight line with varying values for slope and y-intercept

Understanding Learning process with simple linear regression

$y = m*x + c$

y = Target outcome, x = Input predictor, m = weight (internal parameter learnt by the model), c=bias (internal parameter learnt by the model).

No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
1	2012.917	32	84.87882	10	24.98298	121.54024	37.9
2	2012.917	19.5	306.5947	9	24.98034	121.53951	42.2
3	2013.583	13.3	561.9845	5	24.98746	121.54391	47.3
4	2013.5	13.3	561.9845	5	24.98746	121.54391	54.8
5	2012.833	5	390.5684	5	24.97937	121.54245	43.1
6	2012.667	7.1	2175.03	3	24.96305	121.51254	32.1
7	2012.667	34.5	623.4731	7	24.97933	121.53642	40.3
8	2013.417	20.3	287.6025	6	24.98042	121.54228	46.7
9	2013.5	31.7	5512.038	1	24.95095	121.48458	18.8
10	2013.417	17.9	1783.18	3	24.96731	121.51486	22.1

Y = (w1 * X1) + (w2 * X2) + (w3 * X3) + (w4 * X4) + (w5 * X5) + (w6 * X6) + b

w1, w2, w3, w4, w5, w6 — weights / slopes to be learnt , b — bias / y-intercept to be learnt.

These are referred to as **'Learnable Parameters'**.

Understanding Learning process with simple linear regression

So, how does the algorithm learn the parameters to build a Machine Learning Model?

- **Step1 — Random Initialization of learnable parameters(weights & bias)**
- **Step 2 — Forward pass which calculates the output**
- **Step 3 — Loss Estimation**
- **Step 4 — Backpropagation to revise the parameters**

Step1: Random initialization of weights and bias since we have absolutely no clue on where to start.

```
from numpy.random import rand

rand(6)

array([0.75357646, 0.75622356, 0.81593269, 0.67752147, 0.18660467,
       0.30955071])
```

w1 = 0.75357646, w2 = 0.75622356, w3 = 0.81593269, w4 = 0.67752147, w5 = 0.18660467, w6 = 0.30955071, bias = 0

Step2: Forward pass — using the random weights and bias, the output result is computed.

Substituting the values in the formula for the first training sample.

X1 = 2012.917, X2 = 32, X3=84.87882, X4=10, X5=24.98298, X6=121.54024.

y(predicted result) = (0.75357646 * 2012.917) + (0.75622356 * 32) + (0.81593269 * 84.87882) + (0.67752147 * 10) + (0.18660467 * 24.98298) + (0.30955071 * 121.54024) + **0 (bias)**

y = 1516.886867 + 24.19915392 + 69.25540393 + 6.7752147 + 4.661940739 + 37.62286759 + 0

y(predicted value) = 1659.401448

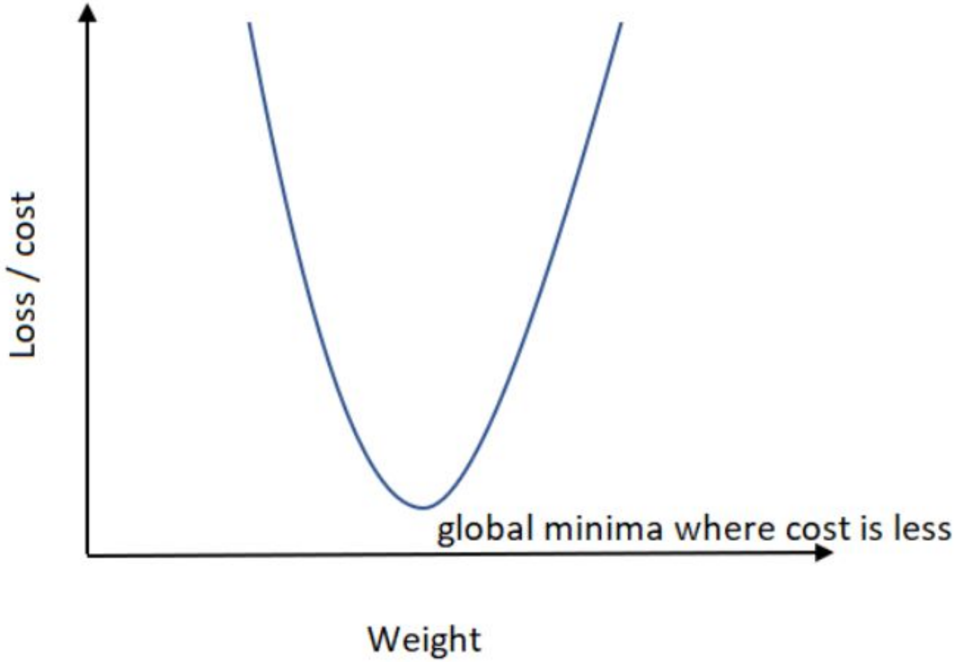
Understanding Learning process with simple linear regression

Step3: Loss function and Error Calculation —

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2$$

L — loss function , y = actual value, yhat = predicted value, N = Total number of training samples.

Why mean squared error?

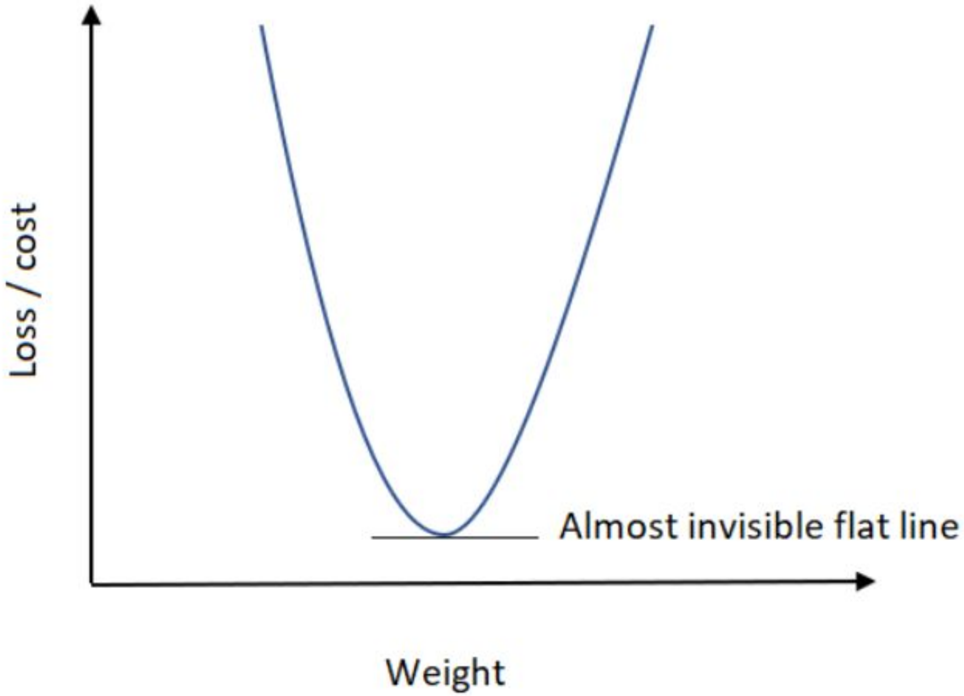


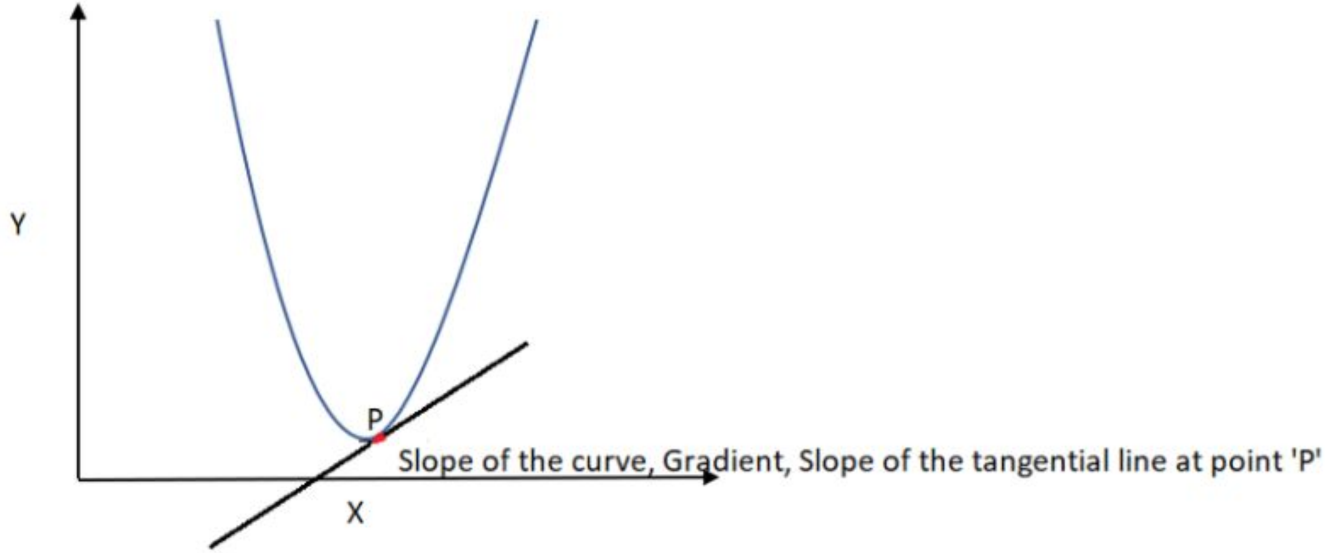
Understanding Learning process with simple linear regression

Finding global Minima:

As we know slope = (change in y / change in x) = 0 and this makes us focus more on making the slope to zero.

The other names for the slope of the curve include **the rate of change** and **Gradient**.





Understanding Learning process with simple linear regression

Step4: Back Propagation — The actual parameter update happens in this step, the error value calculated through the loss function is back propagated to improve the weights and bias. Consider the loss function of linear regression(i.e) Mean squared error,

Sigmoid Function, Forward & Backward Propagation

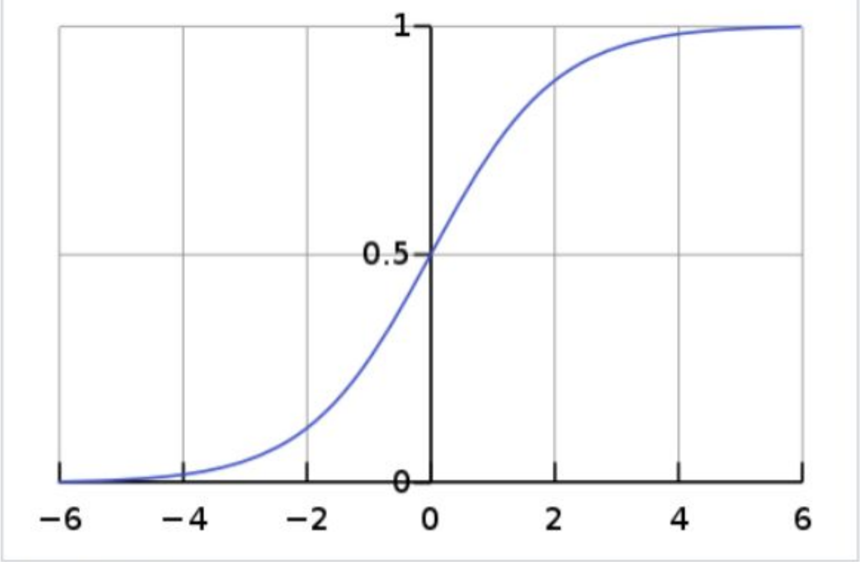
What is a sigmoid function: In the linear regression, we used a straight-line equation y=mx+c to find the relation between input(x) and numerical output(y). Having said that, the logistic regression aims to create categorical results ‘0’ or ‘1’ with the help of sigmoid function S(x) = 1 / (1+e^-x). when

x=0 , S(x) = 0.5

x > 0, S(x) = slowly increasing from 0.5 then flattens out at 1

x <0, S(x) = slowly decreasing from 0.5 then levels out at 0

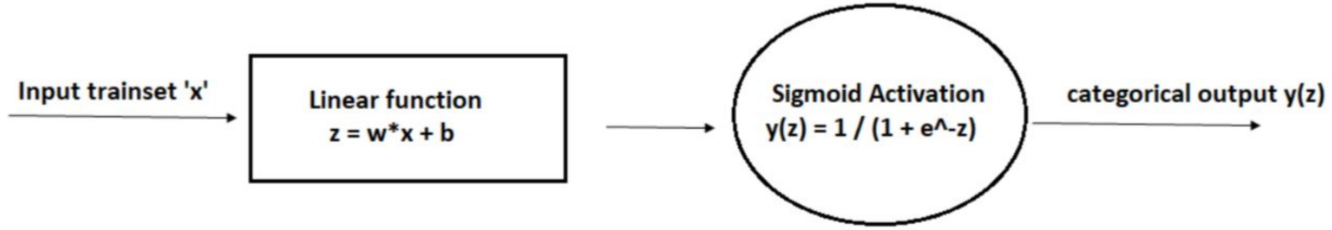
$$S(x) = \frac{1}{1+e^{-x}}$$



The logistic curve

In machine learning, the sigmoid function is termed as an activation function. The characteristics of the sigmoid include *non-linearity* between input and target, the output value is compressed between 0 &1 and monotonic in nature(follows one single direction either increasing or decreasing).

***Forward Propagation in Logistic Regression* —** During the forward pass, the dependent target is determined by first calculating the linear function followed by the sigmoid activation.



Sigmoid Function, Forward & Backward Propagation

Binary Cross-Entropy (log-likelihood) — A loss function:

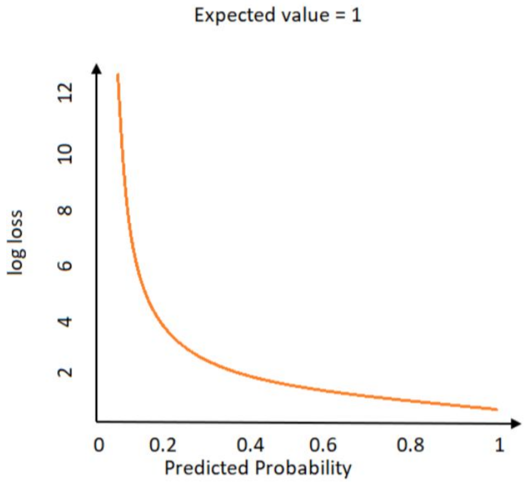
$$\text{Binary Cross Entropy} = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$

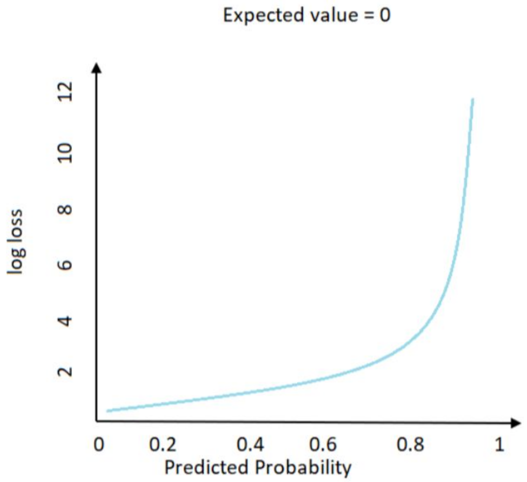
y_n => Expected value

$\hat{y}_n$ => Predicted value

when $y_n = 0$, $(1-y_n)\log(1-\hat{y}_n)$ used for evaluation

$y_n = 1$, $y_n \log \hat{y}_n$ used for evaluation



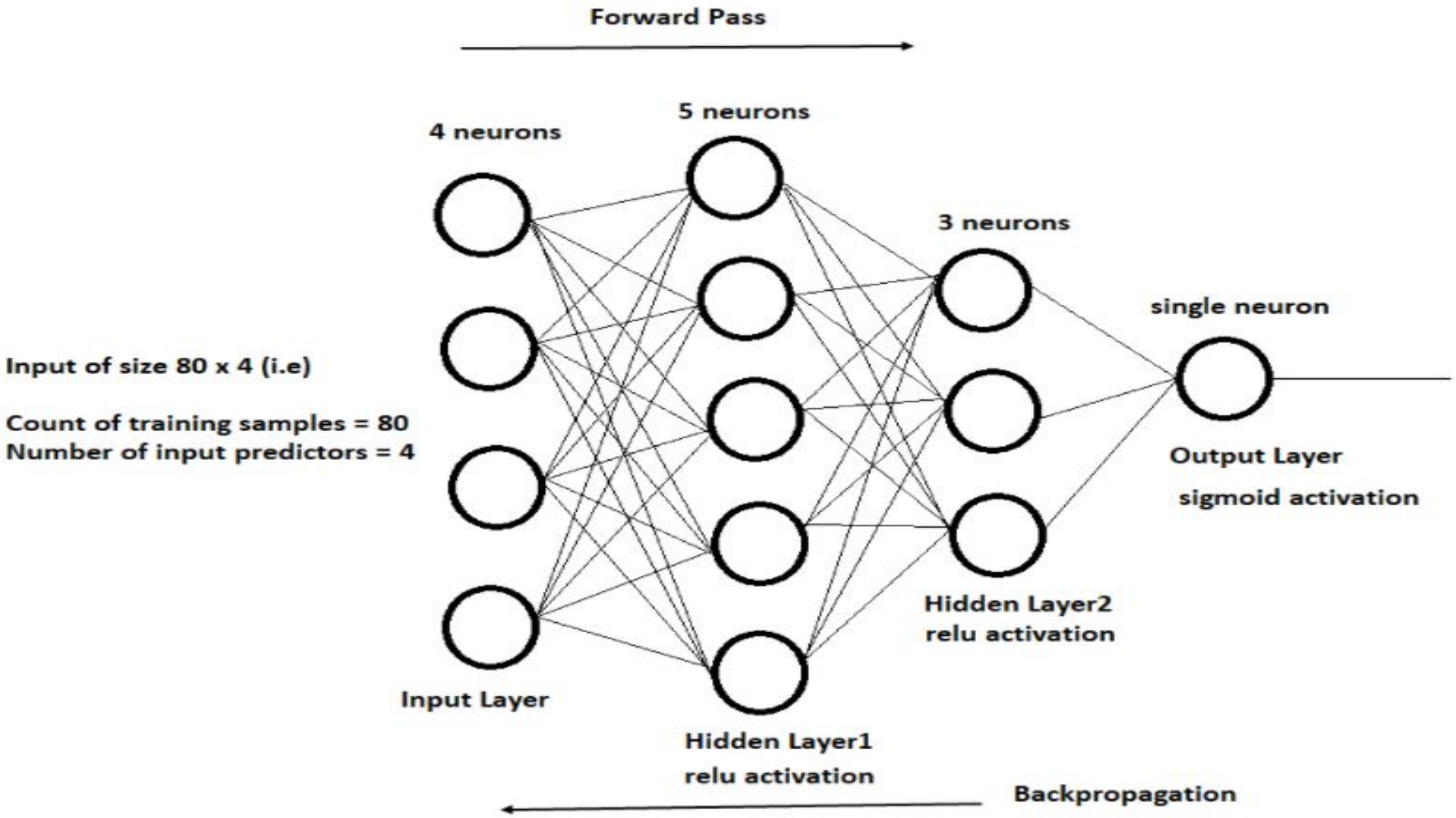


In the figures, when the predicted probability and the expected value are the same, then the loss value is minimal else there will be a high penalty resulting in huge loss value.

Using the gradient descent algorithm the parameters are updated recursively, till an acceptable cost is attained.

Implementation of a Binary Classifier Using Simple NN

Implementation of ANN from scratch using Python & Numpy:



Implementation of a Binary Classifier Using simple ANN in Keras

Import Basic keras and sklearn libraries

```
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

Implementation of a Binary Classifier Using simple ANN in Keras

```
...
# load dataset
dataframe = pandas.read_csv("sonar.csv", header=None)
dataset = dataframe.values

# split into input (X) and output (Y) variables
X = dataset[:,0:60].astype(float)
Y = dataset[:,60]
```

0.0324	0.0232	0.0027	0.0065	0.0159	0.0072	0.0167	0.018	0.0084	0.009	0.0032	R
0.0061	0.0125	0.0084	0.0089	0.0048	0.0094	0.0191	0.014	0.0049	0.0052	0.0044	R
0.0106	0.0033	0.0232	0.0166	0.0095	0.018	0.0244	0.0316	0.0164	0.0095	0.0078	R
0.0294	0.0241	0.0121	0.0036	0.015	0.0085	0.0073	0.005	0.0044	0.004	0.0117	R
0.0046	0.0156	0.0031	0.0054	0.0105	0.011	0.0015	0.0072	0.0048	0.0107	0.0094	R
0.0081	0.0104	0.0045	0.0014	0.0038	0.0013	0.0089	0.0057	0.0027	0.0051	0.0062	R
0.0159	0.0195	0.0201	0.0248	0.0131	0.007	0.0138	0.0092	0.0143	0.0036	0.0103	R
0.0178	0.0052	0.0081	0.012	0.0045	0.0121	0.0097	0.0085	0.0047	0.0048	0.0053	R
0.0439	0.0061	0.0145	0.0128	0.0145	0.0058	0.0049	0.0065	0.0093	0.0059	0.0022	R
0.0198	0.0118	0.009	0.0223	0.0179	0.0084	0.0068	0.0032	0.0035	0.0056	0.004	R
0.0073	0.0062	0.0062	0.012	0.0052	0.0056	0.0093	0.0042	0.0003	0.0053	0.0036	R
0.0217	0.0188	0.0133	0.0265	0.0224	0.0074	0.0118	0.0026	0.0092	0.0009	0.0044	R
0.0266	0.0174	0.0176	0.0127	0.0088	0.0098	0.0019	0.0059	0.0058	0.0059	0.0032	R
0.0068	0.0187	0.0059	0.0095	0.0194	0.008	0.0152	0.0158	0.0053	0.0189	0.0102	R
0.0167	0.0078	0.0083	0.0057	0.0174	0.0188	0.0054	0.0114	0.0196	0.0147	0.0062	R

Implementation of a Binary Classifier Using simple ANN in Keras

```
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
```

```
def create_baseline():
	# create model

	model = Sequential()
	model.add(Dense(60, input_dim=60, activation='relu'))
	model.add(Dense(1, activation='sigmoid'))

	# Compile model
	model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
	return model
```

Implementation of a Binary Classifier Using simple ANN in Keras

```
# evaluate model with standardized dataset

estimator = KerasClassifier(build_fn=create_baseline, epochs=100, batch_size=5, verbose=0)

kfold = StratifiedKFold(n_splits=10, shuffle=True)

results = cross_val_score(estimator, X, encoded_Y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
df = pd.read_csv("spine.csv")
df.head()
```

2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Class_att
52586	39.609117	40.475232	98.672917	-0.254400	0.744503	12.5661	14.5386	15.30468	-28.658501	43.5123	Abnormal
60991	25.015378	28.995960	114.405425	4.564259	0.415186	12.8874	17.5323	16.78486	-25.530607	16.1102	Abnormal
18482	50.092194	46.613539	105.985135	-3.530317	0.474889	26.8343	17.4861	16.65897	-29.031888	19.2221	Abnormal
52878	44.311238	44.644130	101.868495	11.211523	0.369345	23.5603	12.7074	11.42447	-30.470246	18.8329	Abnormal
2075	28.317406	40.060784	108.168725	7.918501	0.543360	35.4940	15.9546	8.87237	-16.378376	24.9171	Abnormal

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
df['Class_att'] = df['Class_att'].astype('category')
encode_map = {
   'Abnormal': 1,
   'Normal': 0
}

df['Class_att'].replace(encode_map, inplace=True)

X = df.iloc[:, 0:-1]
y = df.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=69)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

EPOCHS = 50
BATCH_SIZE = 64
LEARNING_RATE = 0.001
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
## train data
class TrainData(Dataset):

  def __init__(self, X_data, y_data):
      self.X_data = X_data
      self.y_data = y_data

  def __getitem__(self, index):
      return self.X_data[index], self.y_data[index]

  def __len__ (self):
      return len(self.X_data)

train_data = TrainData(torch.FloatTensor(X_train),
torch.FloatTensor(y_train))
```

```
## test data
class TestData(Dataset):

  def __init__(self, X_data):
      self.X_data = X_data

  def __getitem__(self, index):
      return self.X_data[index]

  def __len__ (self):
      return len(self.X_data)


test_data = TestData(torch.FloatTensor(X_test))
```

```
class CustomDataset(Dataset):
  def __init__(self):
      pass
  def __getitem__(self, index):
      pass
  def __len__(self):
      pass
```

x: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

y: 0, 0, 0, 1, 0, 1, 1, 0, 0, 1

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
class CustomDataset(Dataset):

    def __init__(self, X_data, y_data):
        self.X_data = X_data
        self.y_data = y_data

    def __getitem__(self, index):
        return self.X_data[index], self.y_data[index]

    def __len__ (self):
        return len(self.X_data)
```

```
data =
CustomDataset(torch.FloatTensor(X),
torch.FloatTensor(y))
```

```
data.__len__()

################### OUTPUT #####################

10
```

Printing out the 4th element (3rd index) from out data.

```
data.__getitem__(3)

################### OUTPUT #####################

(tensor(4.), tensor(1.))
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
class BinaryClassification(nn.Module):
    def __init__(self):
        super(BinaryClassification, self).__init__()
        # Number of input features is 12.
        self.layer_1 = nn.Linear(12, 64)
        self.layer_2 = nn.Linear(64, 64)
        self.layer_out = nn.Linear(64, 1)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.1)
        self.batchnorm1 = nn.BatchNorm1d(64)
        self.batchnorm2 = nn.BatchNorm1d(64)

    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.batchnorm1(x)
        x = self.relu(self.layer_2(x))
        x = self.batchnorm2(x)
        x = self.dropout(x)
        x = self.layer_out(x)

        return x
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
model = BinaryClassification()
model.to(device)


print(model)


criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)


###################### OUTPUT ######################


BinaryClassification(
  (layer_1): Linear(in_features=12, out_features=64, bias=True)
  (layer_2): Linear(in_features=64, out_features=64, bias=True)
  (layer_out): Linear(in_features=64, out_features=1, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.1, inplace=False)
  (batchnorm1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (batchnorm2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
def binary_acc(y_pred, y_test):
    y_pred_tag = torch.round(torch.sigmoid(y_pred))

    correct_results_sum = (y_pred_tag == y_test).sum().float()
    acc = correct_results_sum/y_test.shape[0]
    acc = torch.round(acc * 100)

    return acc
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
model.train()
for e in range(1, EPOCHS+1):
    epoch_loss = 0
    epoch_acc = 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()

        y_pred = model(X_batch)

        loss = criterion(y_pred, y_batch.unsqueeze(1))
        acc = binary_acc(y_pred, y_batch.unsqueeze(1))

        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()


    print(f'Epoch {e+0:03}: | Loss: {epoch_loss/len(train_loader):.5f}
| Acc: {epoch_acc/len(train_loader):.3f}')

###################### OUTPUT ######################

Epoch 001: | Loss: 0.04027 | Acc: 98.250
Epoch 002: | Loss: 0.12023 | Acc: 96.750
Epoch 003: | Loss: 0.02067 | Acc: 99.500
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
y_pred_list = []


model.eval()
with torch.no_grad():
    for X_batch in test_loader:
        X_batch = X_batch.to(device)
        y_test_pred = model(X_batch)
        y_test_pred = torch.sigmoid(y_test_pred)
        y_pred_tag = torch.round(y_test_pred)
        y_pred_list.append(y_pred_tag.cpu().numpy())



y_pred_list = [a.squeeze().tolist() for a in y_pred_list]
```

Implementation of a Binary Classifier Using simple ANN in Pytorch

```
print(classification_report(y_test, y_pred_list))


######################## OUTPUT ########################

precision    recall  f1-score   support

           0       0.66      0.74      0.70        31
           1       0.88      0.83      0.86        72

    accuracy                           0.81       103
   macro avg       0.77      0.79      0.78       103
weighted avg       0.81      0.81      0.81       103
```

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F_\beta = (1 + \beta^2) * \frac{(Precision * Recall)}{(\beta^2 * Precision) + Recall}$$

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$