Dt:- 09/5/22

Deep Learning :-

Used for

1. Artificial neural network [ANN] [Regresion/classifica-tion]
2. Convolution neural networks [CNN] [computer vision & image classification]
3. Recurrent neural network [RNN] [Time series analysis]

and

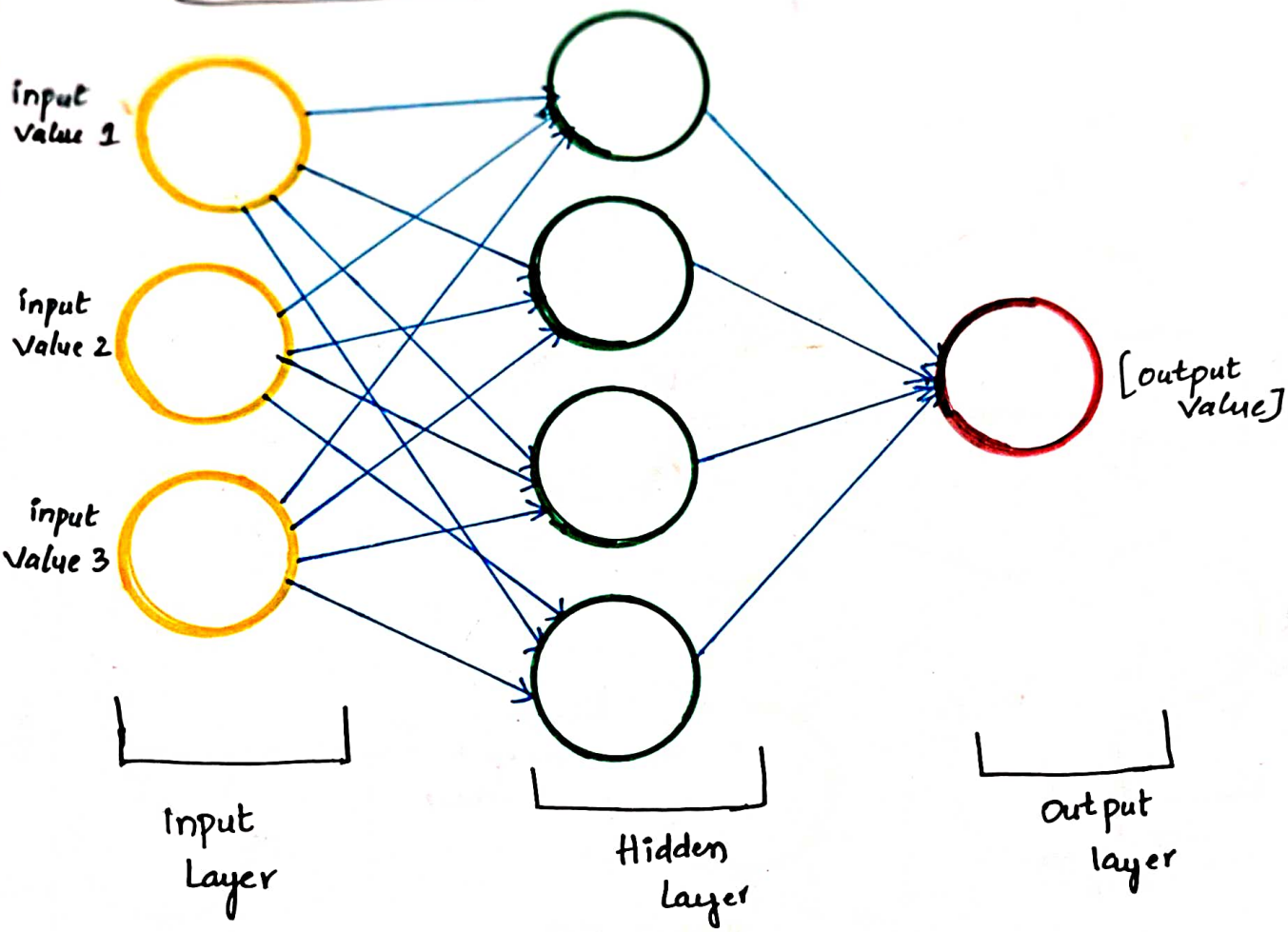
Long short term memory [LSTM]

1. Artificial neural network

* Artificial intelligence :- application which can do its own task with out any human intervention

Ex:- netflix app
* Self driving Cars
* amazon application
* Sofia (robot)

* Neural network :- Connecting the neurons working as one connection is known as "Neural network".



Ex :-

X_1	X_2	X_3	y
-	-	-	-
-	-	-	-
-	-	-	-

Here, we have

* Three input layers :

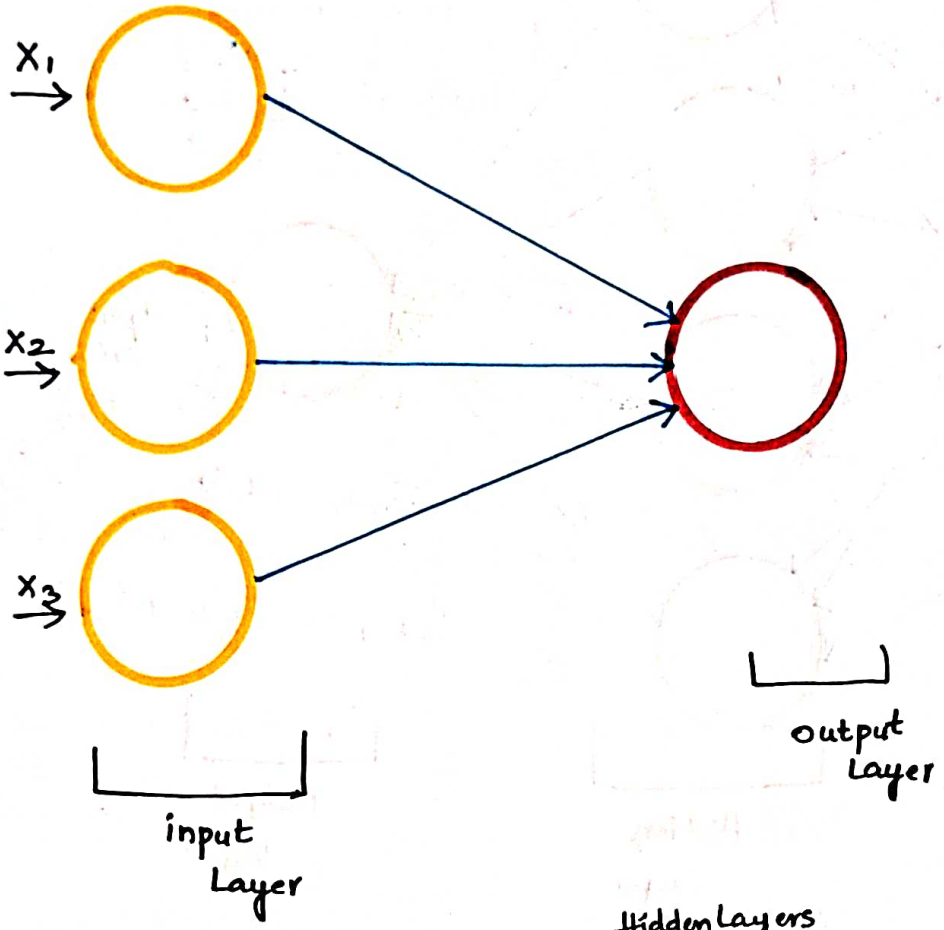
x_1 x_2 x_3

* One output layer :

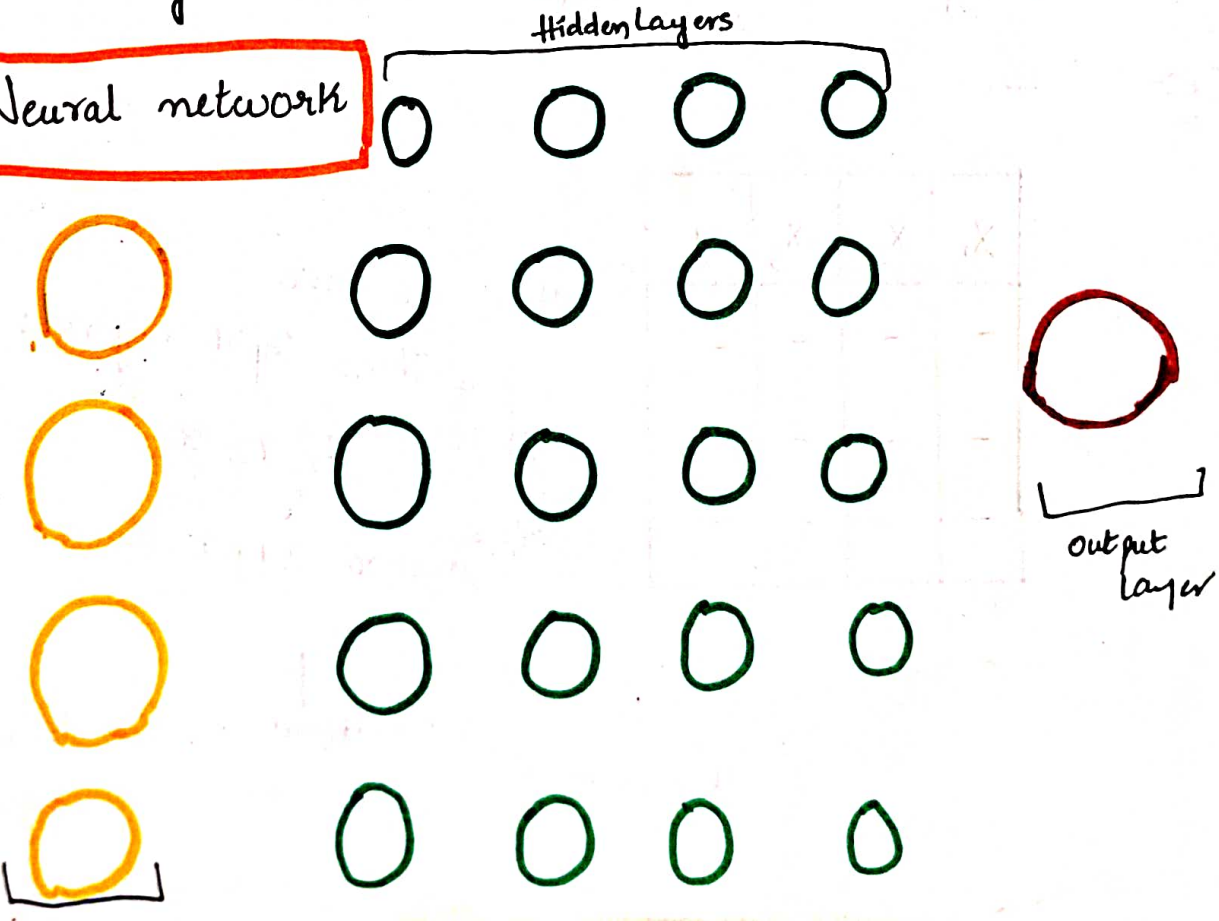
"y"

"y" can be
- continous
- discrete
  - binary
  - multiclass

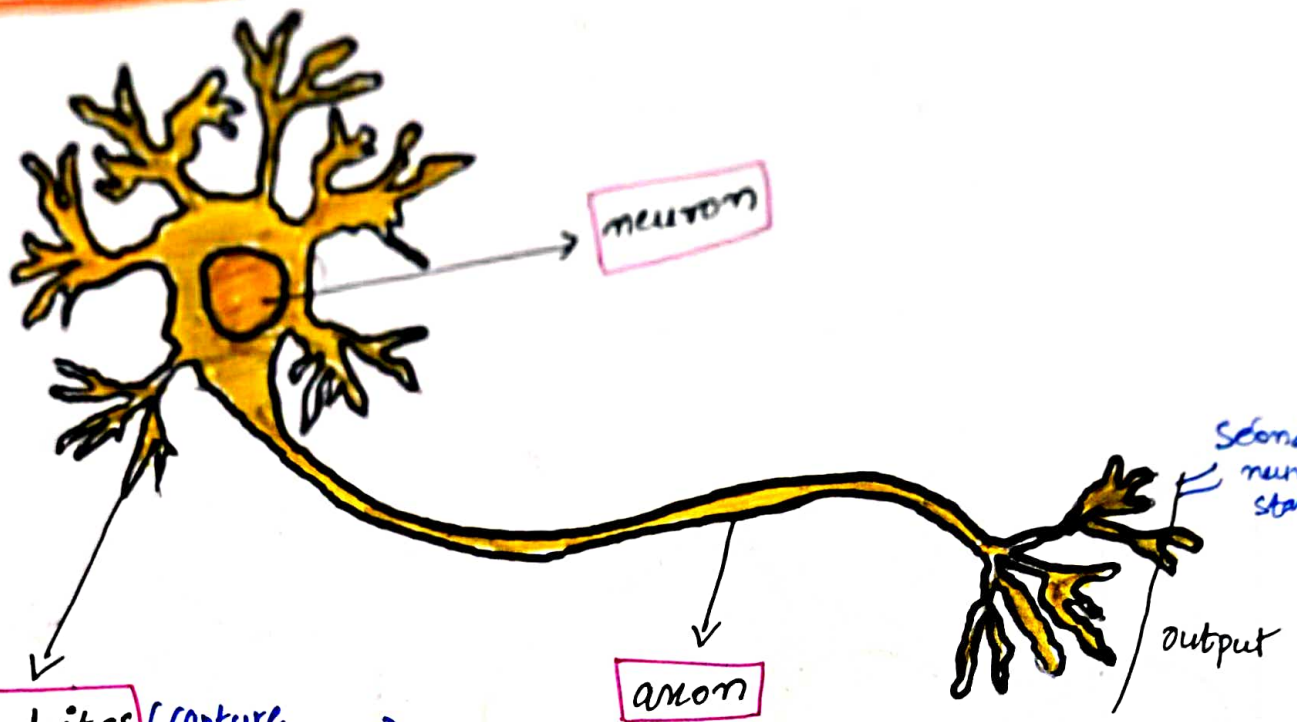
1. **Perceptron** :- connecting Input Layer to output layers directly without any hidden Layers.



2. **Neural network**



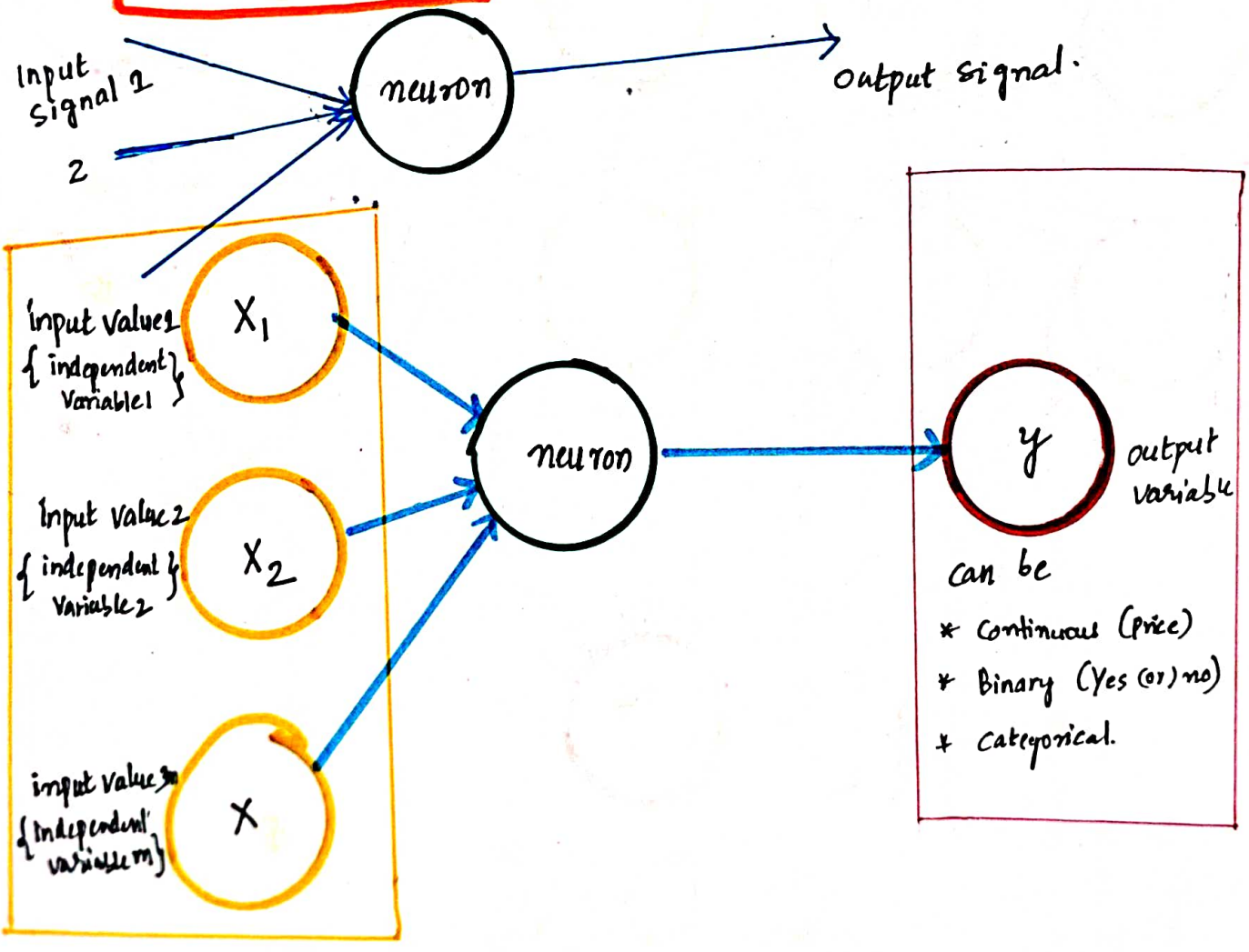
* **The neuron** connected to each other to pass the information



Dendrites (capture information) (input) from other neuron

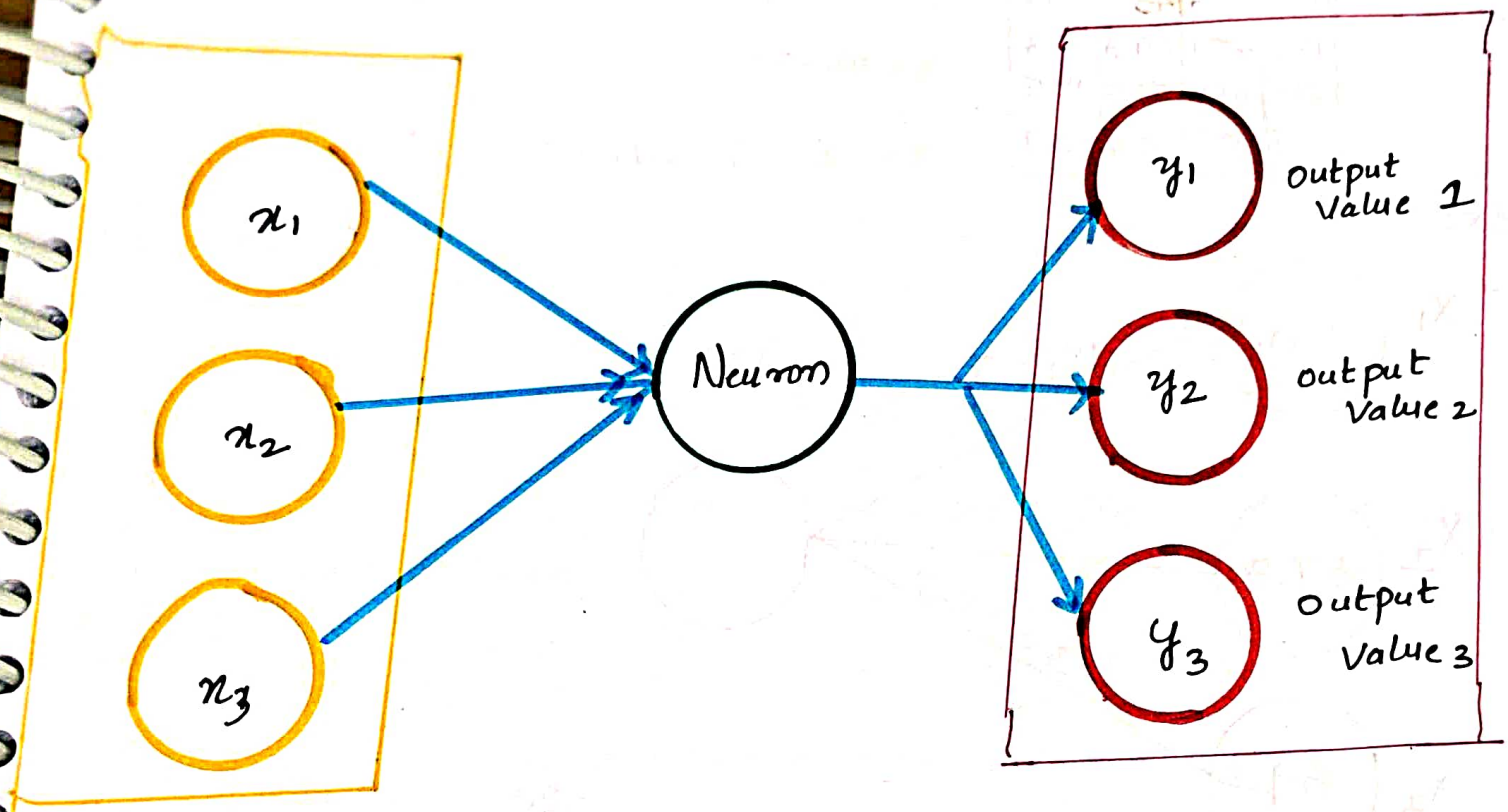
axon (pass the information)

* **Artificial Neuron**



* Binary classification → o/p neuron ⇒ 1 neuron
* Regression → output neuron ⇒ 1 neuron
* Multiclassification → output neuron ⇒ no. of categories (or) classes.

Ex:- Penguin Age, chi, Gentoo



* **Weights**

Machine learning :- $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 = (y - \hat{y})_{min}$

deep learning :- $b + w_1 x_1 + w_2 x_2 + w_3 x_3 = (y - \hat{y})_{min}$

→ it can be written as

$$\sum w_i x_i + b$$

Ex:- linear Regression.

X_1 TV	X_2 Radio	X_3 Newspaper	y Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
151.5	41.3	58.5	18.5
180.8	10.8	58.4	12.9

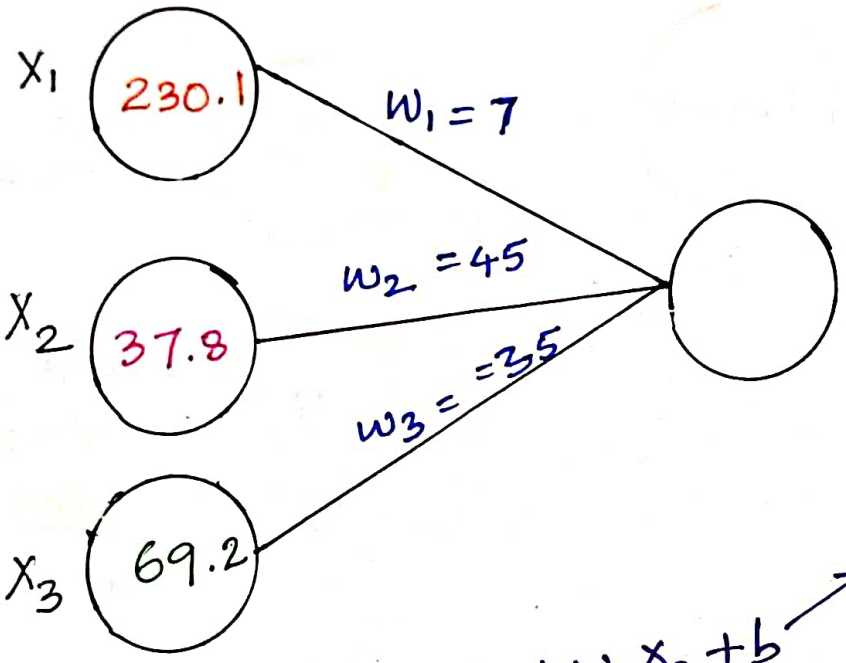
we consider weights

Ex:- Equation $(2x+5=9)$

Initialization of value

	$\hat{y}$	y	error
$x=100 \Rightarrow 2(100)+5$	205	9	-196
$x=99 \Rightarrow 2(99)+5$	203	9	-194
$x=98 \Rightarrow$.	.	.	.
$x=2 \Rightarrow 2(2)+5$	9	9	0

Ex:-



* $w_1x_1 + w_2x_2 + w_3x_3 + b$ → bias

* $[7 \times 230] + [45 \times 37] + [35 \times 69] + 100$

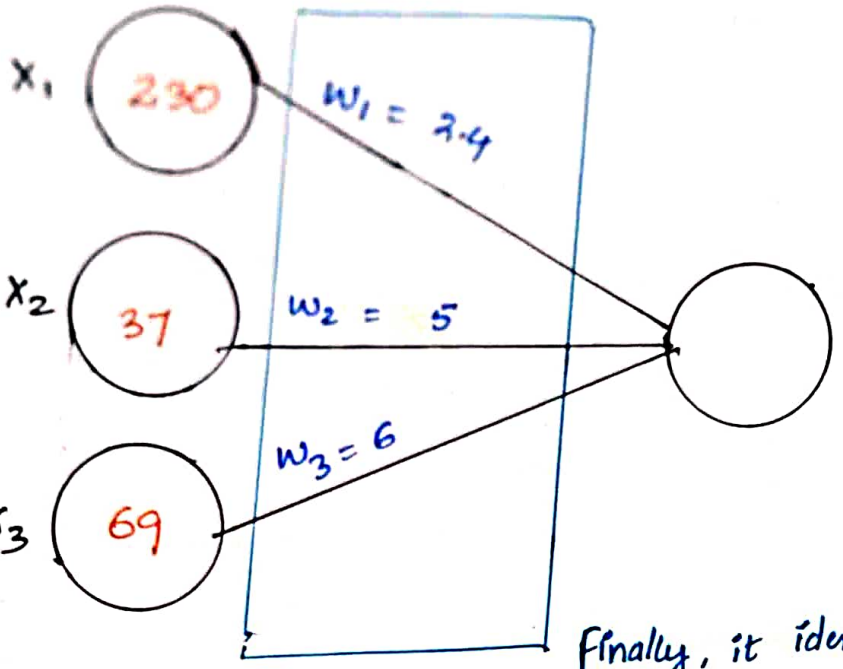
$\hat{y} = 5790.$

y	$\hat{y}$	$y-\hat{y}$
22.1	5790	5767.9

So, $(y-\hat{y})$ Largest amount of difference is there,

- weights will be readjusted

* weights will readjusted

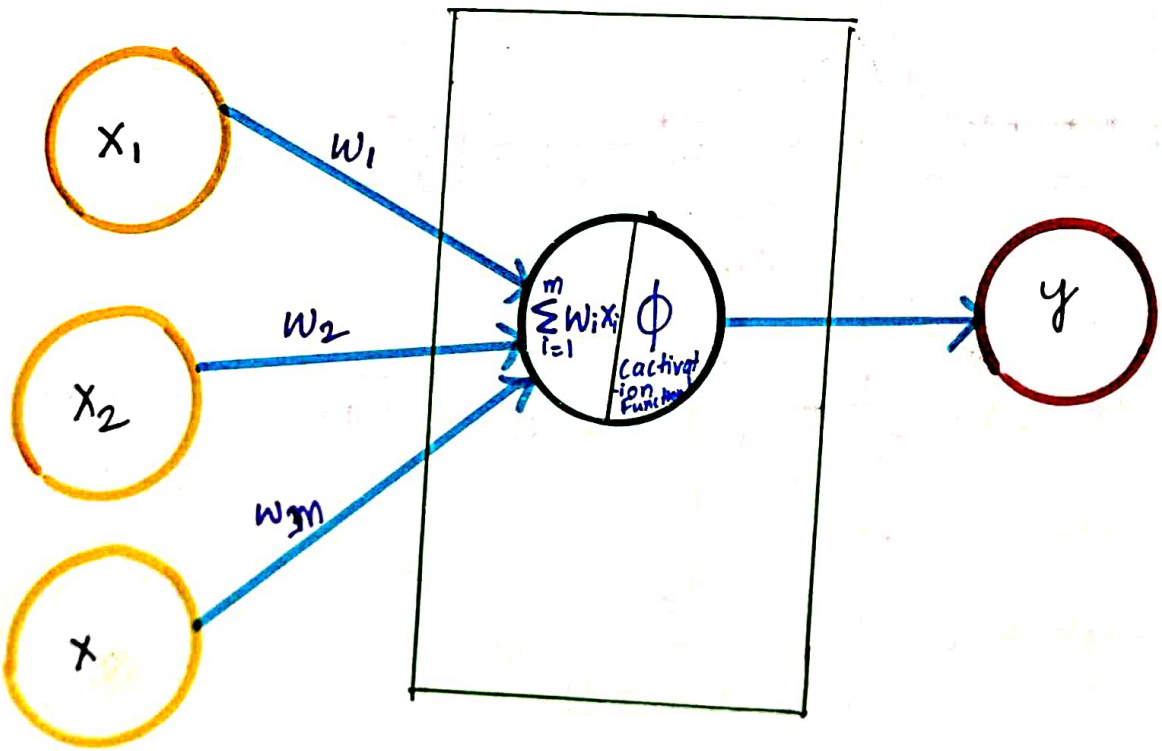


Finally, it identify the best weights which gives sum of square Errors to minimum.

Here, it's identify only weights.

$(SSE)_{min}$.

* **Steps in neuron**

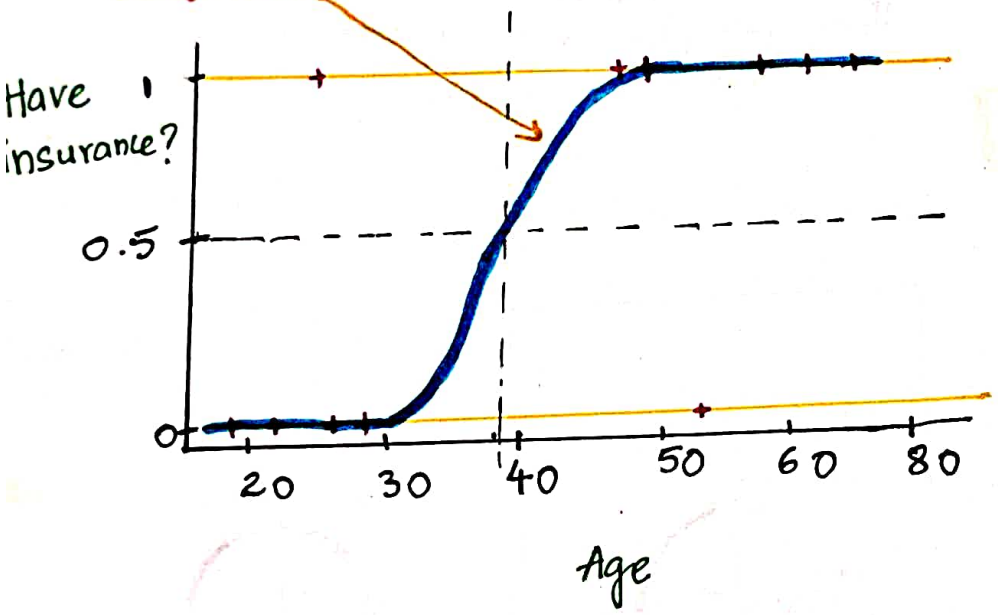


Ex :- **Binary classification**

age	have_insurance
22	0
25	0
47	1
52	0
46	1
56	1
⋮	⋮

Given, an age of a person, come up with a function that can predict if person will buy insurance (or) not.

* **Sigmoid** or **Logit function**



$$\text{Sigmoid}(z) = \frac{1}{1+e^{-y}}$$

$\therefore e$ = Euler's number ~ 2.71828

- $\text{Sigmoid}(200) = \frac{1}{1+2.71^{-200}} =$ almost close to 1

$\because -200$

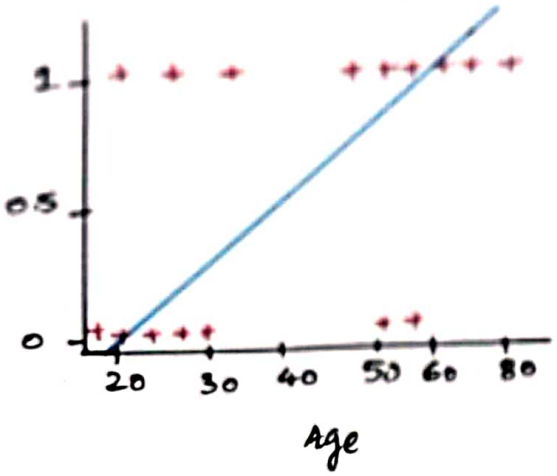
- $\text{Sigmoid}(-200) = \frac{1}{1+2.71^{+200}} =$ almost close to 0

* Sigmoid function converts input into range 0 to 1.

STEP : 1

$y = m * x + b$

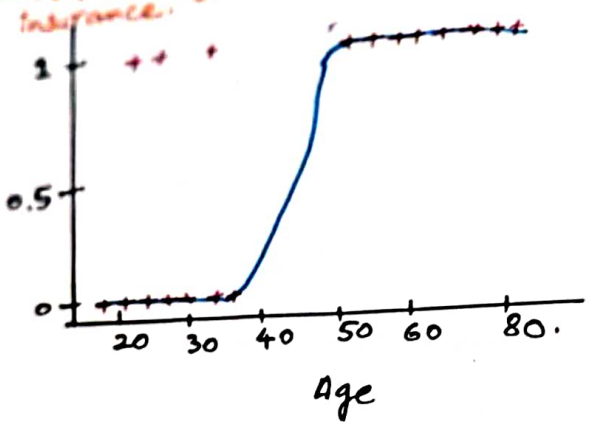
x → age



step : 2

$Z = \frac{1}{1 + e^{-y}}$

Z → If person will by insurance.



$y = 0.042 * x - 1.53$

x → Age.

- **STEPS :-**

$y = 0.042 * x - 1.53$	$Z = \frac{1}{1 + e^{-y}}$

- Value < 0.5 = Person will n Buy
- Value > 0.5 = Person will buy insur

Age = 35

$y = 0.042 \times 35 - 1.53$ $= -0.06$	$Z = \frac{1}{1 + 2.71^{0.06}}$ $= 0.48$

→ 0.48

Age = 43

$y = 0.042 \times 43 - 1.53$ $= 0.276$	$Z = \frac{1}{1 + 2.71^{-0.276}}$ $= 0.568$

→ 0.57

$$y = 0.042 * x - 1.53$$ (b)

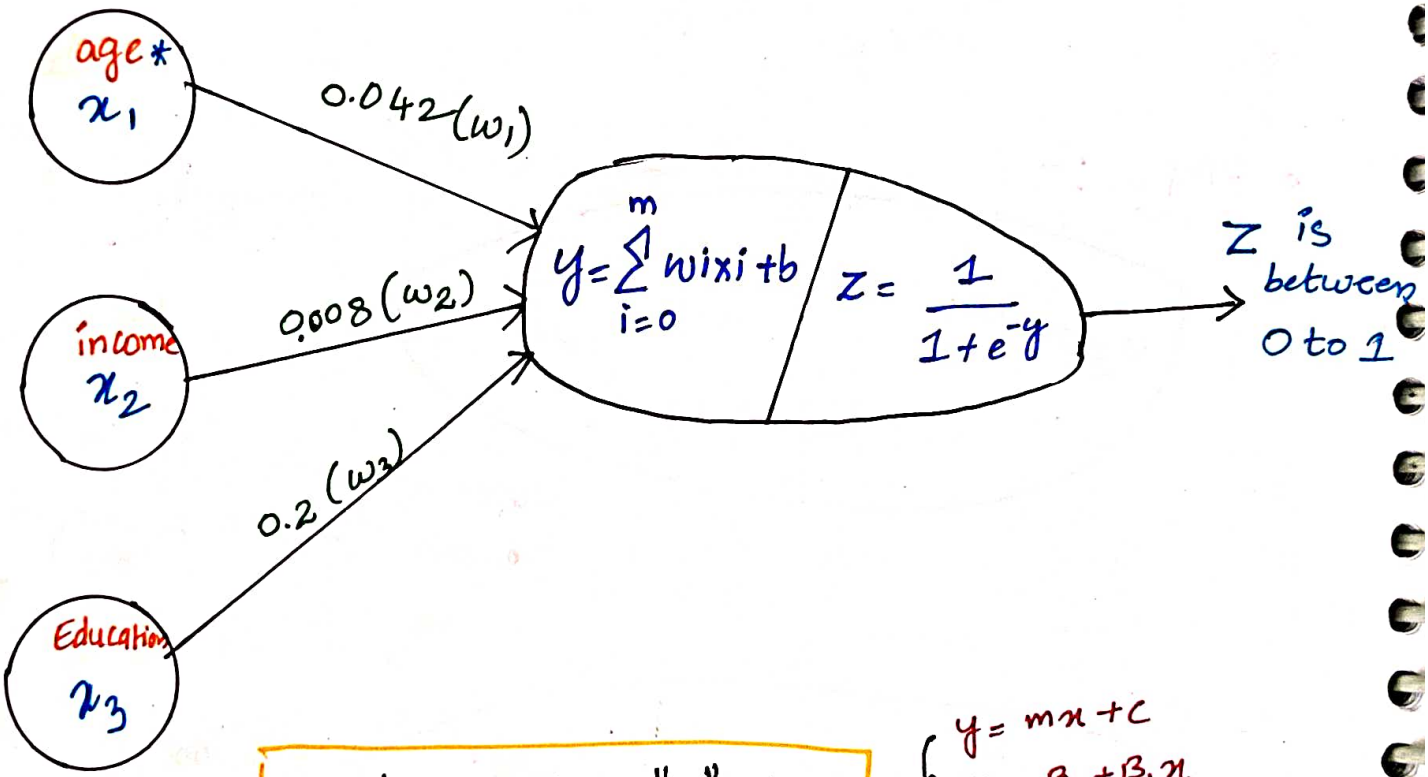
↳ age

$$y = [0.042 * x_1] + [0.008 * x_2] + [0.2 * x_3] - 1.53$$ (b)

↳ Age ↳ income ↳ Education.

$$y = [w_1 * x_1] + [w_2 * x_2] + [w_3 * x_3] + b$$

$$y = \sum_{i=0}^{m} w_i x_i + b$$



* "weights" and "b" values can be +ve, -ve

$y = mx + c$

$y = \beta_0 + \beta_1 x$

$y = w_1 x_1 + b$

all are same (but change is notation)

Ex:-

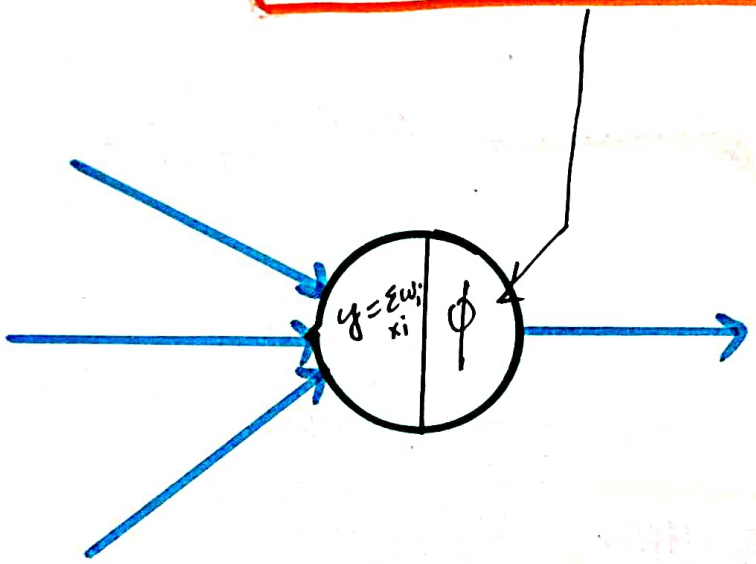
x	y
1	7
2	17
3	27
4	37
5	47

$\therefore y = 10x - 3$

$= w_1 x_1 + b$ → w, b can be (+ve) (or) (-ve)

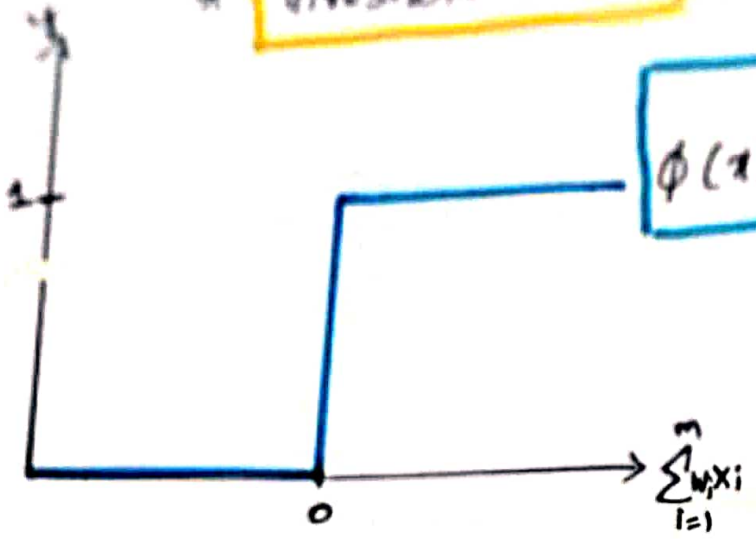
$= 10[x] + [-3]$

Activation function



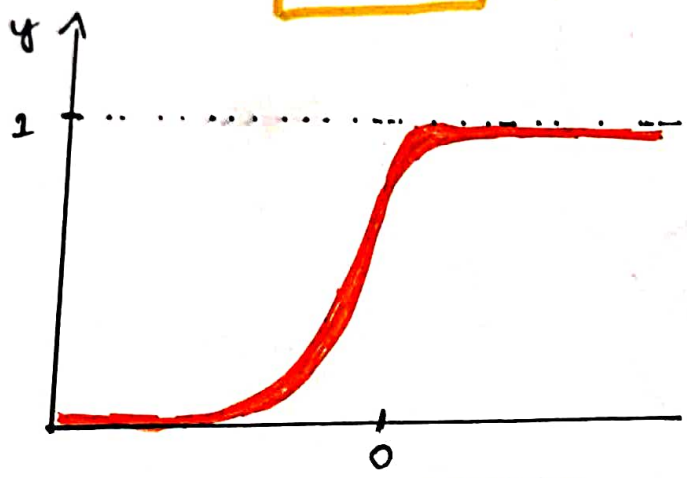
1. Threshold Function (or) step function
2. Sigmoid Function
3. Rectifier (or) relu.
4. Hyperbolic Tangent (tanh)

* Threshold function (or) Step function



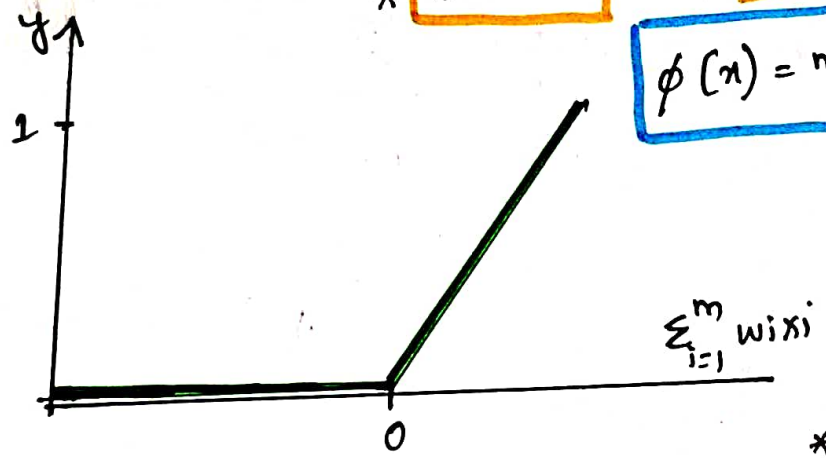
$$\phi(x) = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & \text{if } y < 0 \end{cases}$$

* Sigmoid



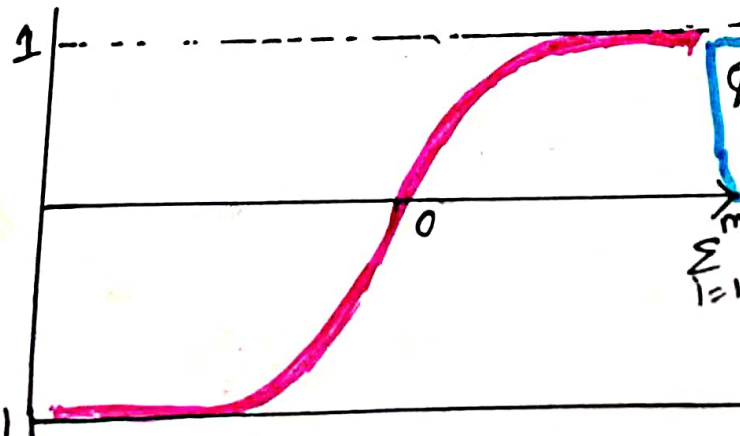
$$\phi(x) = \frac{1}{1+e^{-y}}$$

* Rectifier (or) Relu



$$\phi(x) = \max(x, 0)$$

* Hyperbolic Tangent (Tanh)



$$\phi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

* Threshold function (or) Step function

$\phi($ • if $y \geq 0 = 1$
• if $y \leq 0 = 0)$

* Sigmoid function

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

$$\Rightarrow \frac{ey}{1+ey}$$

* Rectifier (or) relu

• Relu = Rectifier Linear Unit.

$$\phi(x) = \max(x, 0)$$

* Tanh (or) hyperbolic Tangent

$$\phi(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$$

Ex:- Regression → Relu

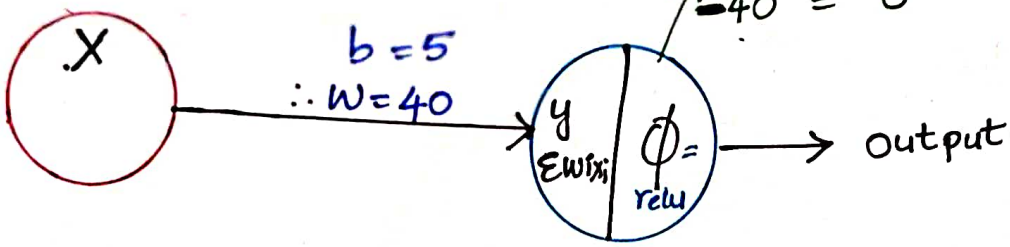
x	y	$\hat{y}$	$y-\hat{y}$
1	11	45	-
2	12	85	-
3	13	125	-
4	14	165	-
5	15	205	-

$\varepsilon(y-\hat{y})^2 = -$

$\hat{y}$

45 = 45

-40 = 0

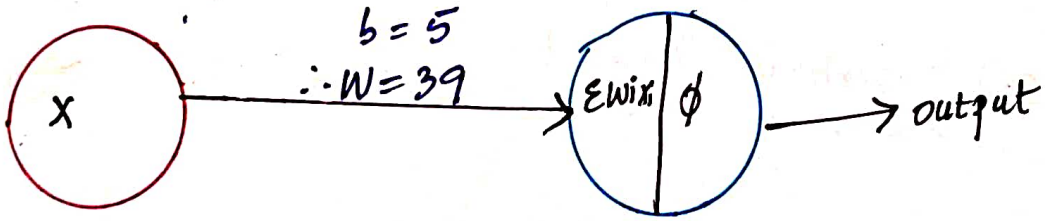


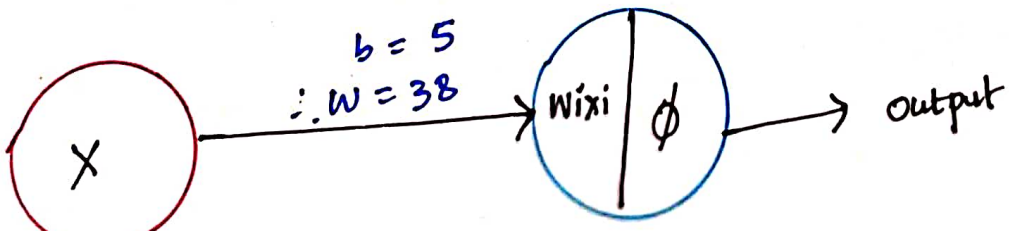
$\therefore \hat{y} = wx + b$

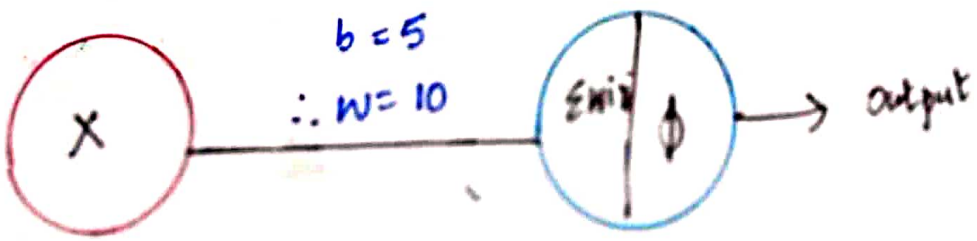
$x = 1, \quad \hat{y} = 40(1) + 5 = 45$

$x = 2, \quad \hat{y} = 40(2) + 5 = 85$

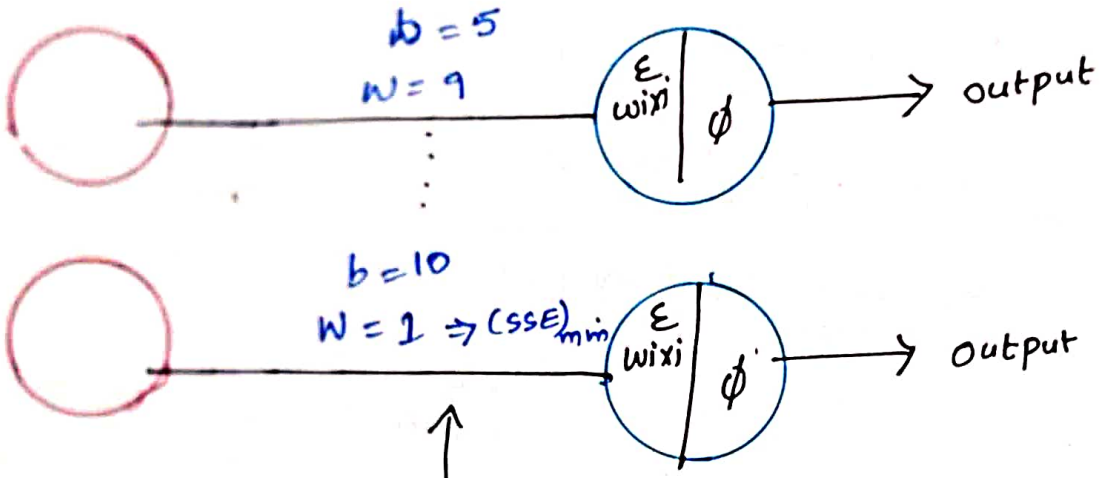
* adjusting weights







* after that it changes 'bias" ⇒ 'b'



$\therefore y = wx + b$

$\Rightarrow y = 1(x) + 10$

* iterations :- try & Errors [Apply loop]

For better answer.

---

important points

- no. of input Neurons = no. of input features.
- no. of output Neurons =
  - Regression :- 1
  - binary class :- 1
  - Multiclass :- no. of categories

* Activation function of output Neuron :

1. regression ⇒ relu
2. binary classification ⇒ Sigmoid
3. Multiclass ⇒ Soft max

↓

Calculate the probability of Each class. and which of them having highest probablity we consider into that class.

Ex :- drug(x), drug(y), drug(z)

P(drug x)
Prob(drug y)
Prob (drug z)
} → highest

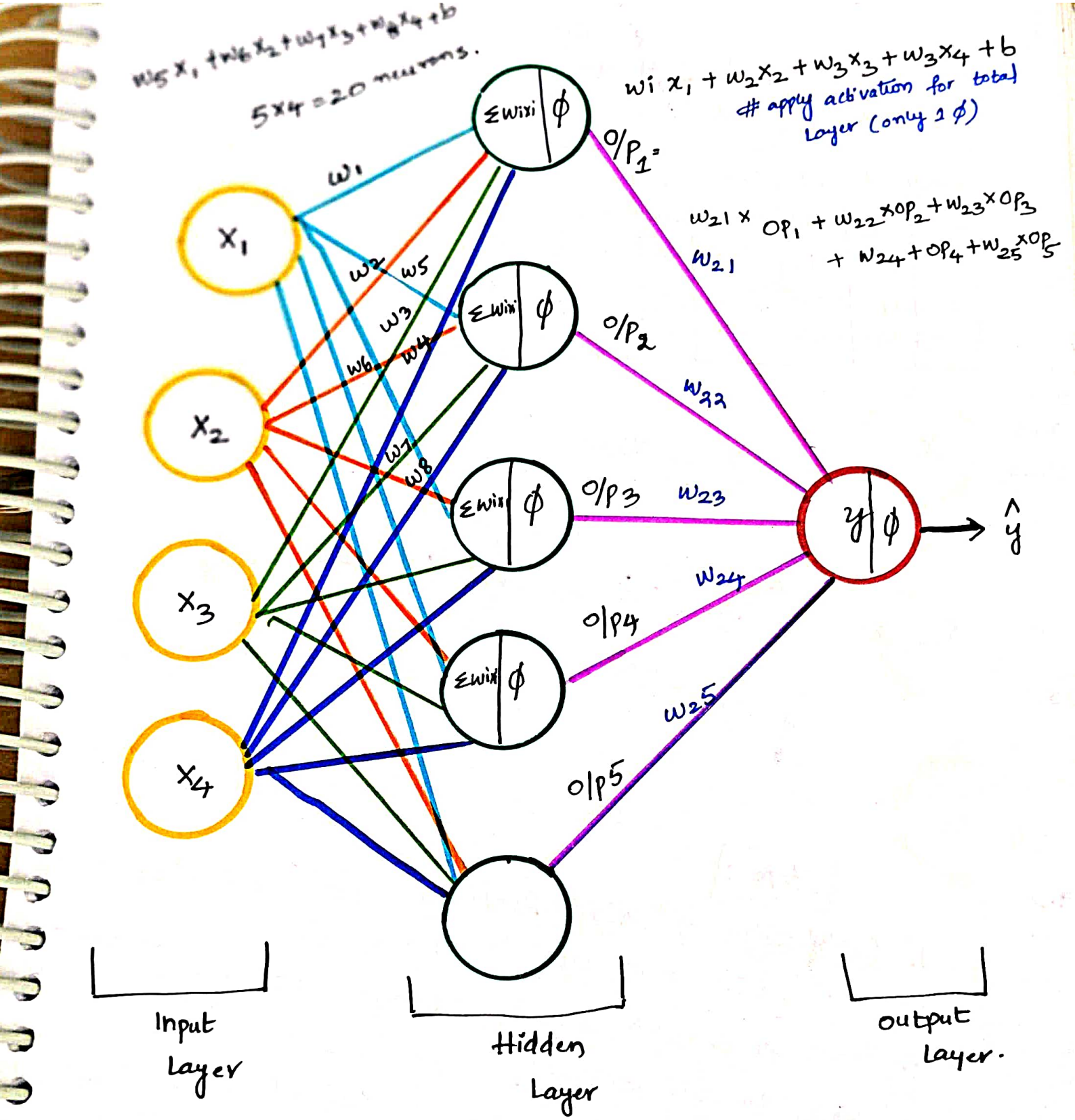
09/05/22
5:30pm.

Dt:-9/05/22
7:40pm.

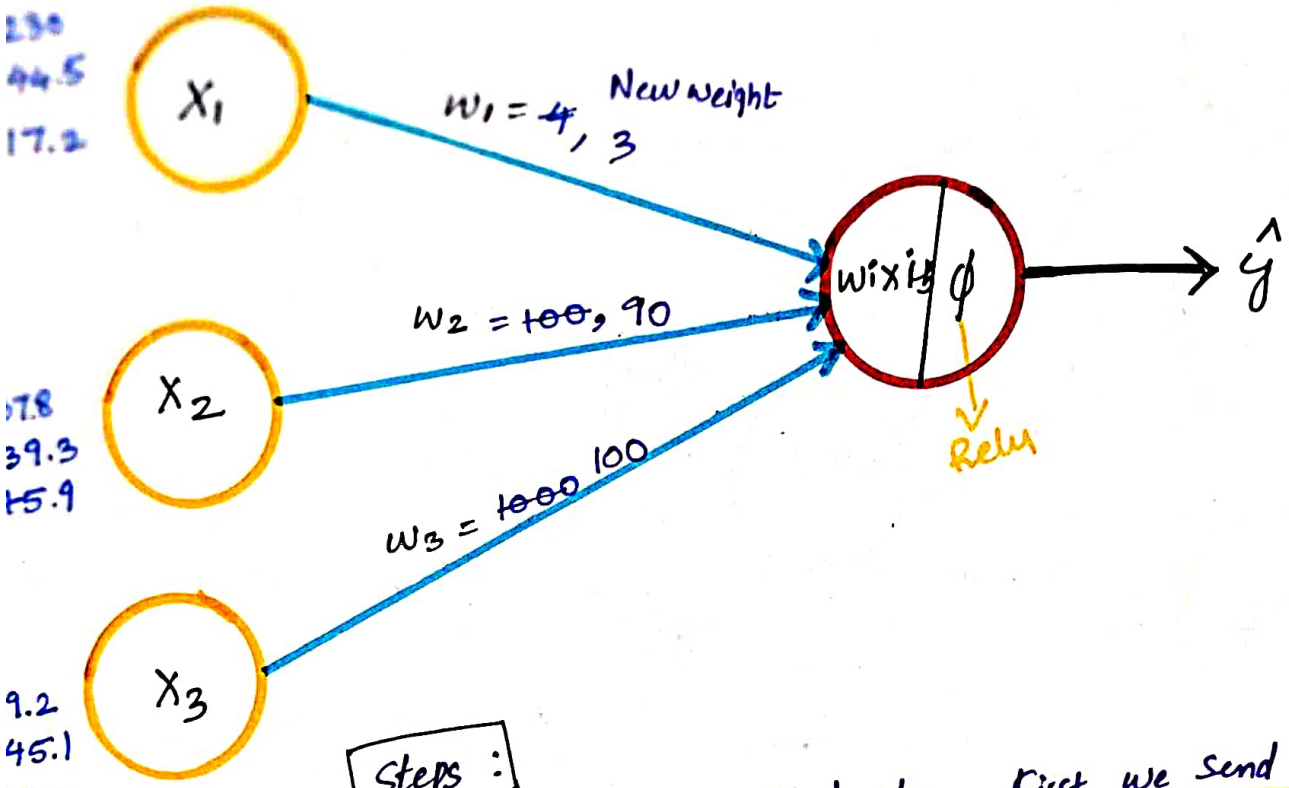
How do neural network Work ?

We can Take as per choice, Hidden Layers in Neural network

and Same hidden Layers, we can Take as many as no. of neurons in the neural network.


w5x1 + w6x2 + w7x3 + w8x4 + b
5 x 4 = 20 neurons.
wi x1 + w2x2 + w3x3 + w3x4 + b
apply activation for total Layer (only 1 ϕ)
Σwixi ϕ
o/p1
o/p2
o/p3
o/p4
o/p5
w21 x op1 + w22 x op2 + w23 x op3 + w24 + op4 + w25 x op5
w21
w22
w23
w24
w25
w1
w2
w3
w4
w5
w6
w7
w8
x1
x2
x3
x4
y ϕ
ŷ
Input Layer
Hidden Layer
output Layer.

TV	Radio	Newspaper	Sales	$\hat{y}$	$y-\hat{y}$	$(y-\hat{y})^2$
230.1	37.8	69.2	22.1	—		
44.5	39.3	45.1	10.4	—		
17.2	45.9	69.3	9.3	—		
151.5	41.3	58.5	18.5			
180.8	10.9	58.4	12.9			



Steps :

1. Intialization of weights Randomly, First we send 230.1, 37.8, 69.2 into input layer (x_1, x_2, x_3)
2. Than, we calculate 230×4, 37.8×100, 69.2×1000 with weights and going to calculate $(\hat{y})$.
3. In the neuron, we calculate $(wixi+b)$ and (ϕ)(Relu) and output is predicted $(\hat{y})$
4. Similarly same as first, we continue with all data. --
5. Than, it Calculate Error $(y-\hat{y})$
6. After that, it Calculate $(y-\hat{y})^2$ (predicted - Actual)^2.

7. at last, we want (SSE)min.
8. In order to reduce Error, the weights will recapdate To 3,90,100
9. again, it will do same pattern, in loop. (In forward & Backward).
10. Finally it make weights such that, it gives (SSE)min. till that, it process.

Ex:- **Clear Example of Updating weights**

$2x+5=8$

	$\hat{y}$	y	$y-\hat{y}$
* $x=100$	$2(100)+5=205$	8	-197
* $x=105$	$2(105)+5=215$	8	-207
* $x=95$	$2(95)+5=195$	8	-187

As "x" value increasing, Error also increasing.
so, we decrease "x".

as "x" value decreasing, Error also decreasing.

$\frac{dx}{\Delta x}=5$ (change in $x=5$)

* $x=90$
* $x=85$
* $x=80$
* $x=5$
* $x=0$

$2(5)+5=15$	8	-7
$2(0)+5=5$	8	3

so, The Error is in between -5 & 8

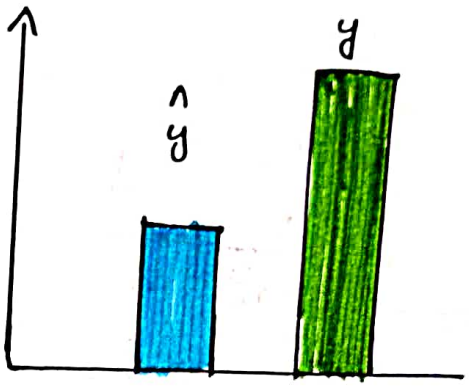
* $x=1$
* $x=2$
* $x=1.5$

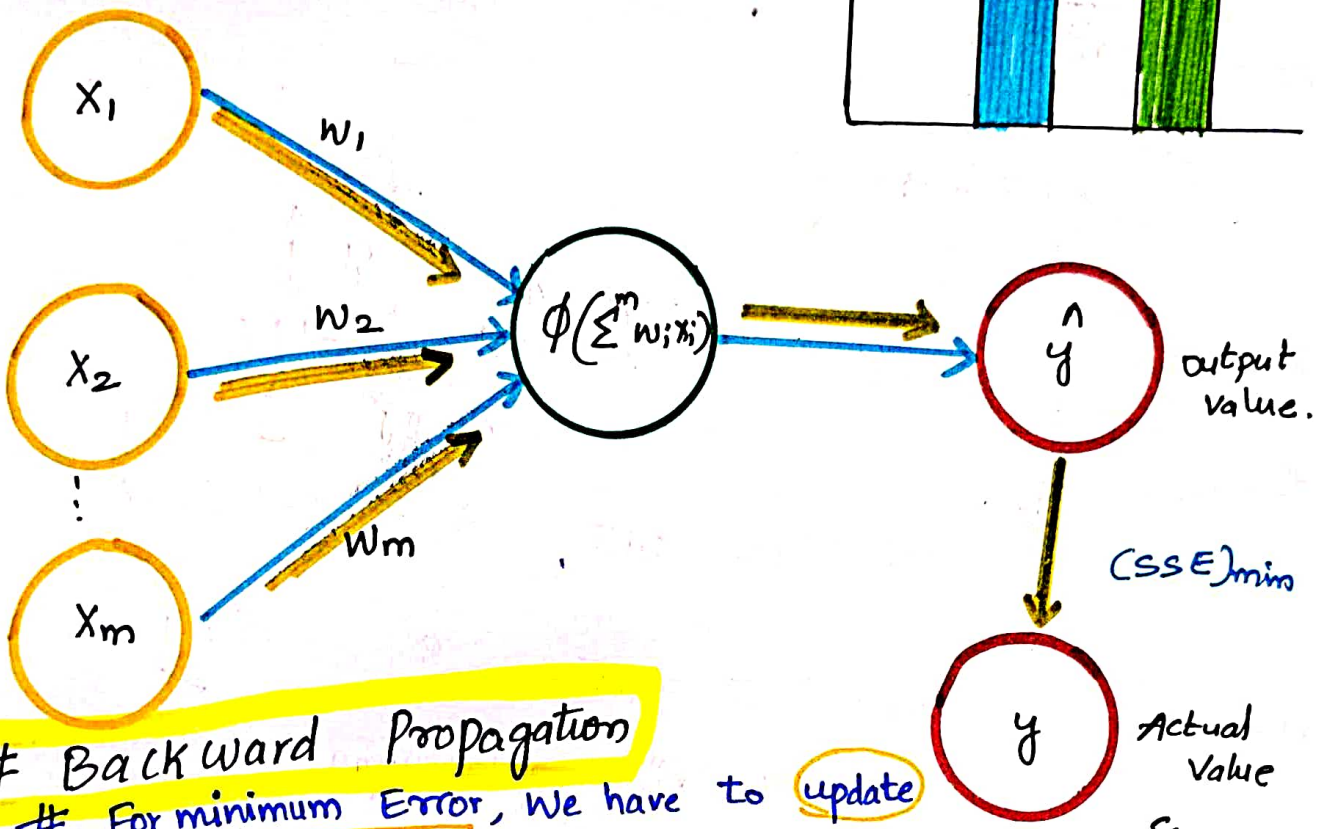
We updated:
till we get minimum value

How do neural networks Learn ?

Neural networks on based on Connections.

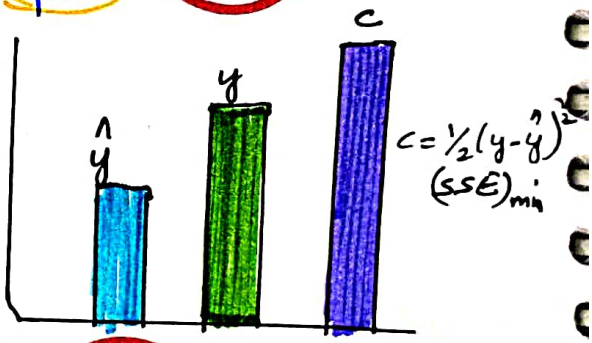
Forward propagation

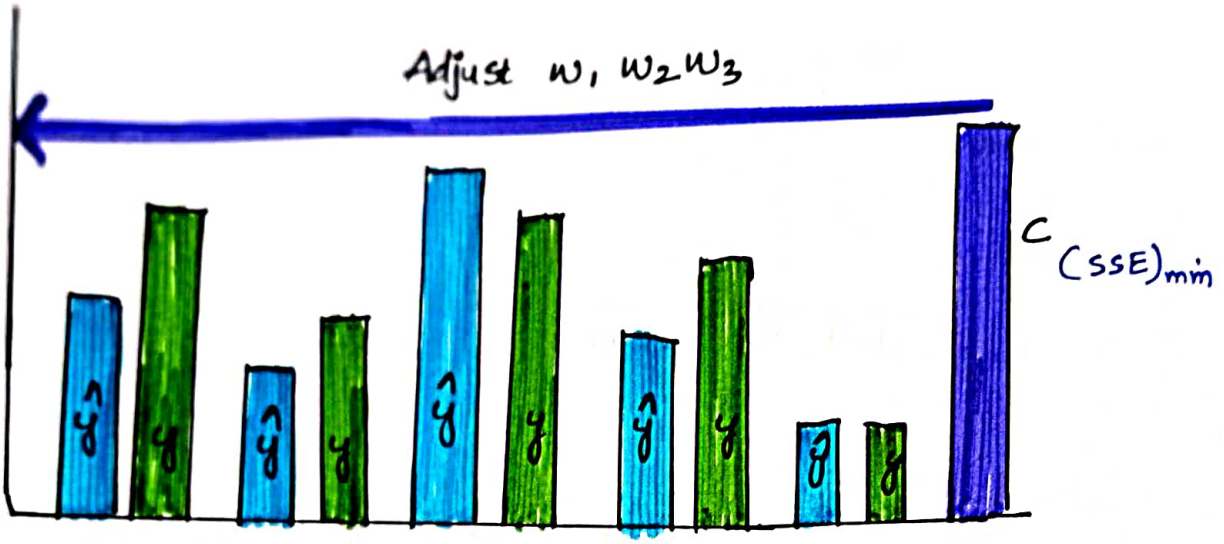




Backward Propagation

For minimum Error, We have to update weights.





forward propagation :-

First, Calculating $\Sigma w_i x_i + b$ and Calculating the output Variable in The forward direction is known as forward propagation.

backward propagation :

Adjusting (or) updating the weights, coming backward direction is known as backward propagation.

intially, weights are assigned randomly, and from that we update weights.

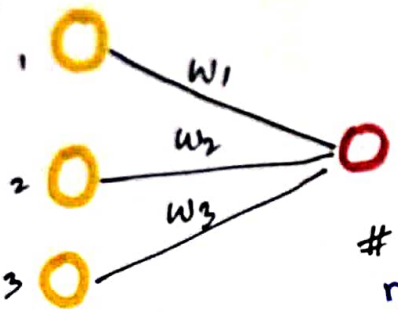
The weights will be updated in a way that $(SSE)_{min}$

Weight initialization

1. **Uniform** = $\left[\frac{-1}{\sqrt{inputs}}, \frac{+1}{\sqrt{inputs}}\right]$

we have select the weights, that forms, uniform distribution.

Ex:-



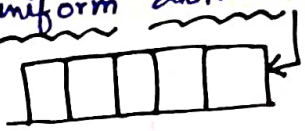
$\left[\frac{-1}{\sqrt{3}}, \frac{+1}{\sqrt{3}}\right]$

we have select values in these only.

These weights should form uniform distribution.

Backend code.

np.random.uniform()



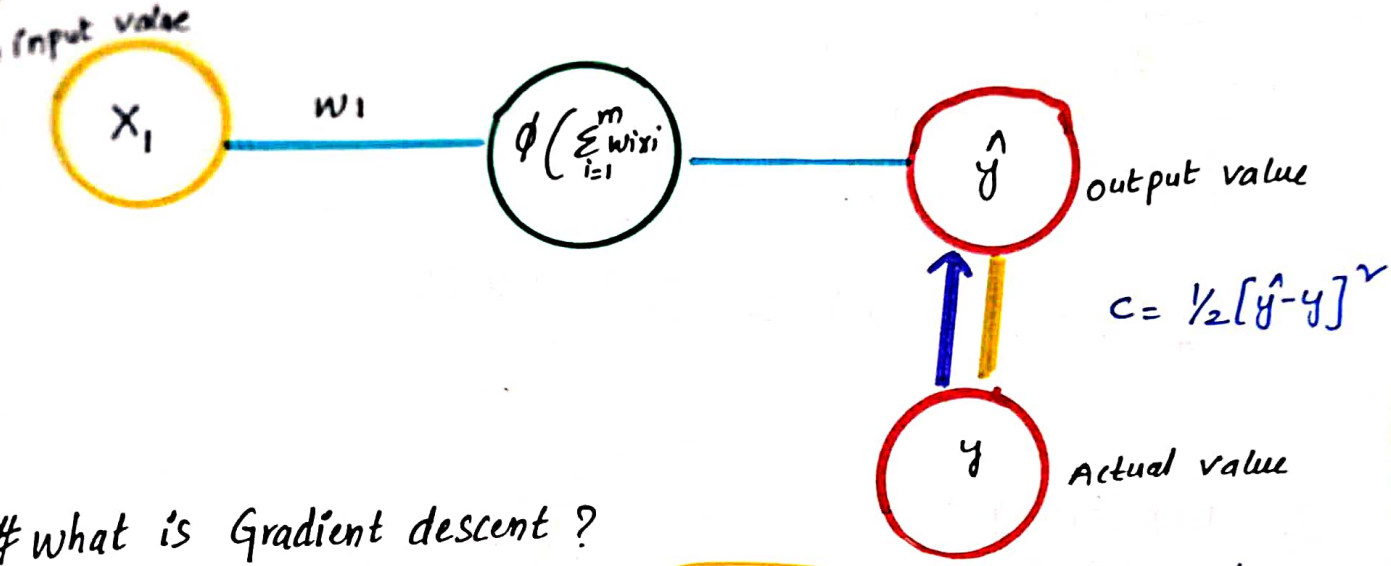
2. **Xavier Uniform** = $U\left[-\sqrt{\frac{6}{intout}}, +\sqrt{\frac{6}{intout}}\right]$

3. **Xavier normal** = $N\left[0, \sqrt{\frac{2}{intout}}\right]$

4. **He_int Uniform** = $U\left[-\sqrt{\frac{6}{in}}, \sqrt{\frac{6}{in}}\right]$

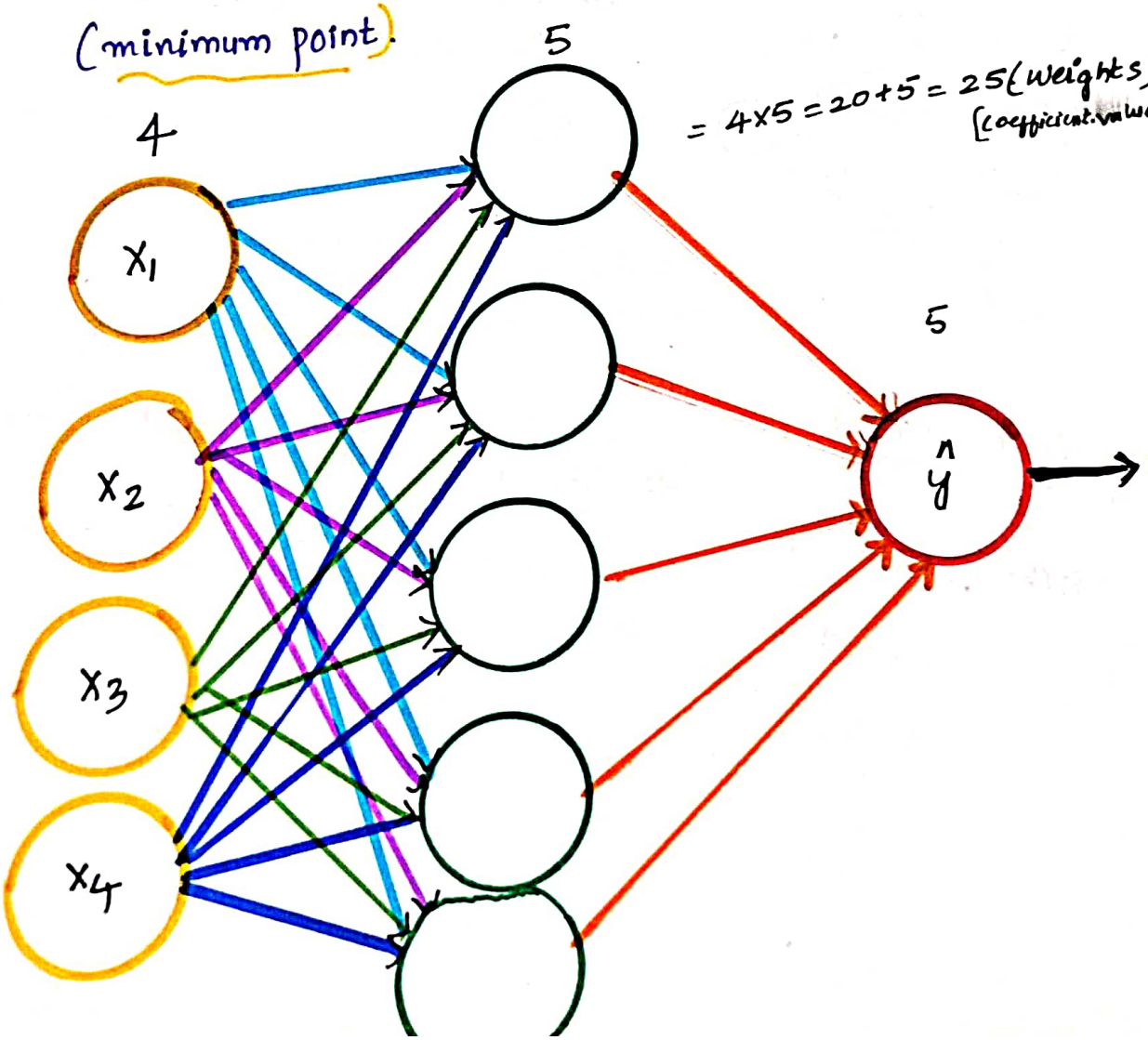
∴ **He_int normal** = $N\left[0, \sqrt{\frac{2}{int}}\right]$

Gradient Descent



What is Gradient descent ?

Gradually decreasing and identifying the Local minima (minimum point).



In Machine learning :

Ex:- In multiple linear regression

model.intercept_ , model.coef_

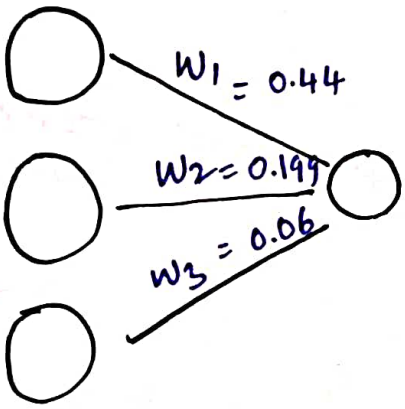
out : (2.708), array (0.44, 0.199, 0.006)

β_0 β_1 β_2 β_3

$\hat{y} = \underset{(b)}{\beta_0} + \underset{(w_1)}{\beta_1 x_1} + \underset{(w_2)}{\beta_2 x_2} + \underset{(w_3)}{\beta_3 x_3}$

$= 2.7 + 0.44[x_1] + 0.199[x_2] + 0.06[x_3]$

In deep learning :



By Using this Technique, it gives "accurate results".

Gradient descent

Ex:- $2x + 5 = 8$

$x = 100$

$x = 99$

⋮

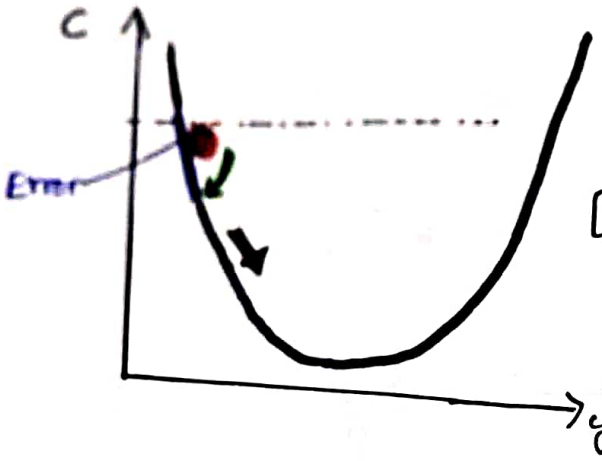
$x = 0$

$x = 1$

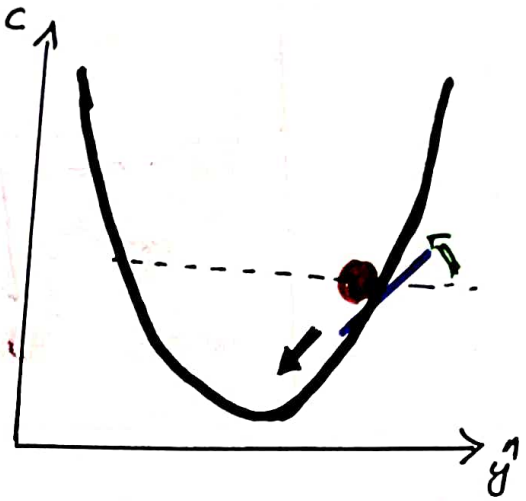
$x = 1.5$

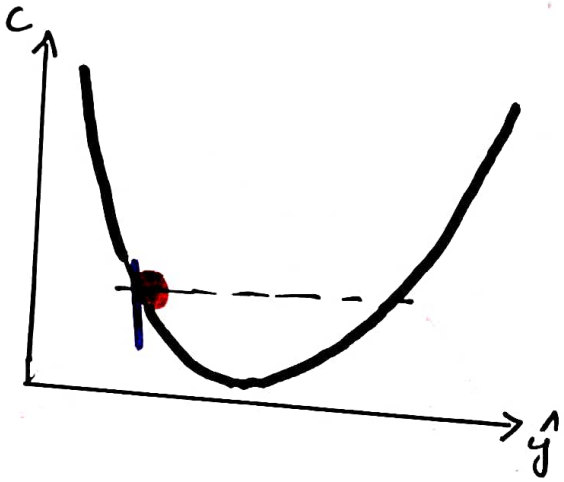
Gradually reducing weights. and making Error as minimum value.

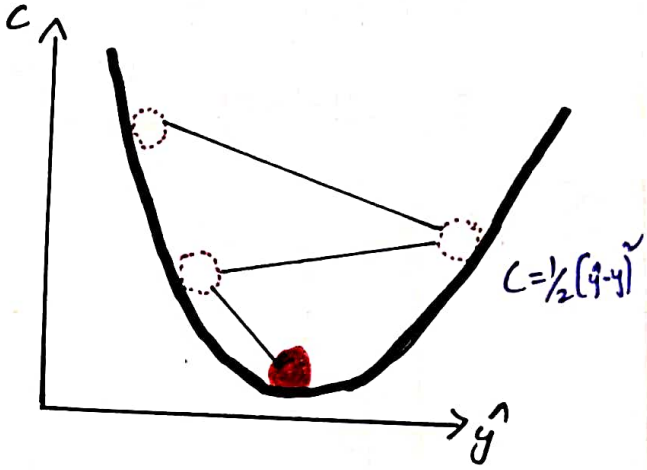
* By applying different "x" values and identfing the "perfect "x" value till, we get error as "0" ⇒ geradient decent



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





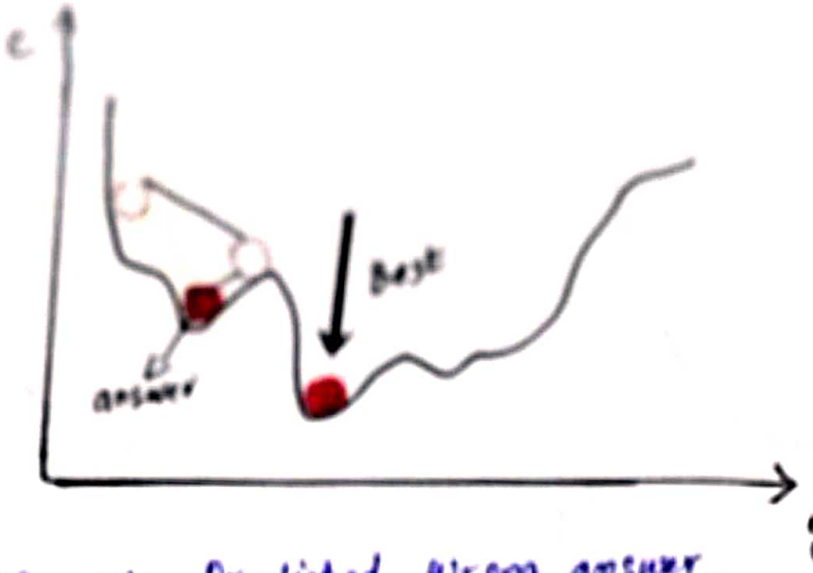


Stochastic Gradient Descent

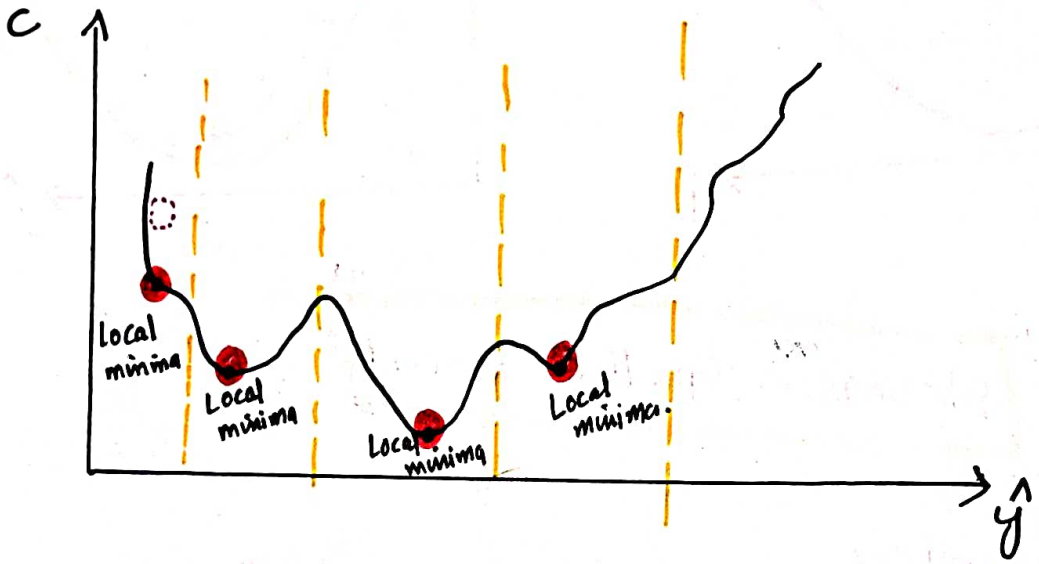
* Local minima
* Global minima

EX:- • Every state 1st ranker (local minima)

• All India ranker 1st ranker (global minima)



Here, we predicted wrong answer,
But, there is another curve. So, How we find that?
by dividing into parts.



identifing the Global minima is called
"Stochastic Gradient Descent"
(or)
Entire data we divide into parts (or) batches. Each part
We are identifying the "Local minima" and from
that overall identify "Global minima".
↓
"Error".

2nd 09/05/22 10:20pm

DL: 10/5/20 10:00 am

ANN - Classification

Code :- on Tensorflow, Keras.

Tensorflow, CNTK, theano

Google | Micro soft | FaceBook

```
# PIP install tensorflow
# PIP install keras
```

```
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
# from tensorflow import keras.
# df = pd.read_csv("churn_modelling.csv")
# df.head()
```

out:

Row no.	customer id	Surname	Credit Score	Geo graphy	Gender	Age	Tenure	Bal ance	num of Products	Has card	Is active numb	Estimated Salary
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348
2	15647311	Hill	608	Spain	Female	41	1	83807	1	0	1	112542
3	15619304	Onio	502	France	Female	42	8	159660.	3	1	0	113931
4	15701354	Boni	699	Feance	Female	39	1	0.00	2	0	0	93826.
5	15737888	Mitchell	850	Spain	Femal	43	2	125510	1	1	1	79084.

```
# df.info()
```

out:

```
Range Index : 10000 entries , 0 to 9999
Data columns.                Nonnull columns      Dtype.
0.  Row number               10000 Non null       int64
1.  Customer Id              1000 Non null        int64
```

3	Credit score	10000 non null	int 64
2	Surname	10000 non null	object
categorical data 4	Geography	10000 non-null	object] convert to numeric
5	Gender	"	
6	Age	"	int64
7	Tenure	"	int 64
8	Balance	"	float 64
9	No. of Products	"	int 64
10	has Cr Card	"	int 64
11	Is Active member	"	int 64
12	Estimated Salary	"	float 64
13	Exited	"	int 64

we can use drop function also [x & y]

```
# x = df.iloc[:, 3:13].values
# y = df.iloc[:, 13].values
```

: → all; 3:13 → 3 to 12; [:,13] → only 13 column

original data: 13 input — 3 drop (row no., customer id, surname) → 10 inputs; 1 output → 1 column

```
# x.shape
```

Out: (10000, 10)

```
# y.shape
```

Out: (10000,)

ordinal data (sequence) — natural

↑ Label Encoder

```
# encoding [categorical data]
from sklearn.preprocessing import
# x[:, 2]
```

Out: array ["Female", "Female", ... "male", "Female"].

```
# labelencoder = LabelEncoder()
# x[:,2] = labelencoder.fit_transform(x[:,2])
# changed to numerical.
# x[:,2]
out: array [0,0,0 .... 0,1,0]
```

```
# one hot encoding (nominal data = Geography (column)) # We can use get_dummies.
from sklearn.preprocessing import OneHotEncoder
***
from sklearn.compose import ColumnTransformer
                              ↳ Transforms data.

# x[:,1]
out: array([["France", "spain", "France" ... "France", "Germany"].
           dtype= object.

# ct = ColumnTransformer([["encoder", OneHotEncoder(), [1]]
                          , remainder = "passthrough")
# x = np.array(ct.fit_transform(x))
# changed to numerical
# x[:,1]
out: array([[0.0, 0.0, 0.0, .... 0.0, 1.0, 0.0].
```

train_test

```
from sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test = train_test_split(x, y,
                                     test_size = 0.2, random_state = 0)
```

```
# Shape.
# x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out: (8000, 12), (2000, 12), (8000,), (2000,)

↳ after applying dummies it converts into 12 columns.

Scaling

```
from sklearn.preprocessing import StandardScaler

# sc = StandardScaler()
# x_train = sc.fit_transform(x_train)
# x_test = sc.fit_transform(x_test)
```

MODELLING

Initializing The ANN

```
from keras.models import Sequential
# ann = Sequential()
```

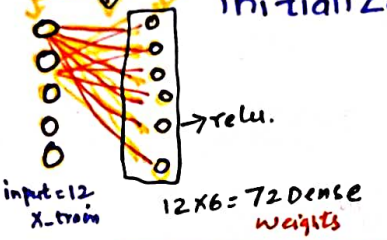
Adding The input layer and first hidden layer.

```
from keras.layers import Dense.

# ann.add(Dense(input_dim = 12, units = 6, kernel_
    initializer = "uniform", activation = "relu"
    ))
```

connections

Hidden Layer

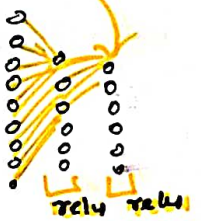


$\cup \left[\frac{-1}{\sqrt{in}}, \frac{+1}{\sqrt{in}}\right]$

$\cup \left[\frac{-1}{\sqrt{12}}, \frac{1}{\sqrt{12}}\right]$

Adding the second hidden Layer.

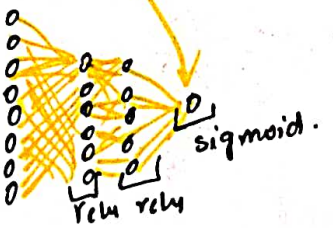
```
# ann.add(Dense(units = 6, kernel_initializer = "uniform",
    activation = "relu"))
```



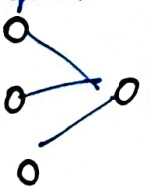
Adding the output layer

output

```
# ann.add(Dense(units = 1, kernel_initializer = "uniform",
    activation = "sigmoid"))
```



Ex:- Perceptron model.



```
(ann.add(Dense(input_dim = 3, units = 1, kernel_initialization
    = "uniform", activation = "relu")).
```

Part-3 Training the ANN

Compiling the ANN

Ex: current
past: Connection is made (wiring), (data)

$\sum(y-\hat{y})^2$
regression = "mean_squared_error"

```
# ann.compile(optimizer = "adam", loss = "binary_crossentropy",
                  Gradient descent          function
                  metrics = ["accuracy"])
```

model is created, fit the data

training the ANN on training Set

```
# ann.fit(x_train, y_train, epochs = 100)
```

epochs → no. of iteration. (backward, forward directions)

Out

```
Epoch 1/100
250/250 [=========] - 2s 1ms/step - loss: 0.6457, - accuracy - 0.7830)
Epoch 2/100
. . . . . .
. . . . . .
. . . . . .
Epoch 100/100
```

accu: 0.838

Part-4 Predictions & Evaluating model

making the predictions

Predict
Test accuracy.

```
# y_pred = ann.Predict(x_test)
# y_pred = (y_pred > 0.5)
```

Evaluating the model

```
from sklearn.metrics import confusion_matrix, accuracy_score

# print ("Test Accuracy:", accuracy_score (y_test, y_pred))

# confusion_matrix (y_test, y_pred)
```

Out: Test accuracy : 08455

```
array([[1536, 59
        [150, 155]].
```

cross validate the model

```
# user defined function.

def build_cross_classifier():
    classifier = Sequential()
    classifier.add(Dense(input_dim=12, units=6, kernel_
                         initializer="uniform", activation="relu
    classifier.add(Dense(units=6, kernel_initializer=
                         "uniform", activation="relu"))
    classifier.add(Dense(units=1, kernel_initializer
                         "uniform", activation="sigmo
    classifier.compile(optimizer="adam", loss="binary_
                       cross_entropy, metrics=["Acc

    return classifier
```

ann (or) anything we can use

input layer

hidden layer

output layer

connecting

Classifier

```
from keras.wrappers.scikit_learn import KerasClassifier

# classifier = KerasClassifier(build_fn = build_cross_classifier
                               , batch_size = 10 , epochs = 100)
```

build_fn = User defined function

batch_size = if stochastic descent

epochs = iterations

Cross_validation

```
from sklearn.model_selection import cross_val_score

# accuracies = cross_val_score(estimator = classifier ,
                               X = x_train , y = y_train, cv=5)
```

out :

```
Epoch 1/100
640/640 [======] - 1s  605us/step - loss: 0.4922, -accr: 0.7997
Epoch 2/100
640/640 [======] - 0s  617us/step - loss: 0.4296, -acc, 0.8003
. . . . . .
. . . . . .
. . . . . .
. . . . . .
Epoch 101/100
640/640 [======] 1s 805/step - loss: 0.4003. acc: 0.8342]
160/160 [======] 0s 666us/step - loss: 0.4110 . acc: 0.8375 ]
```

```
# print(accuracies)
# accuracies.mean()
[0.8237 . 0.8343 , 0.8362 , 0.8362, 0.8374)
```

out

0.83362

⇒ Hyperparameter Tunning

Part-5 - Improving and Tuning the ANN

This can done by 3 options

1. Hyperparamter tuning
2. Regularization (L1 & L2) to reduce overfitting if needed (for regression problems)
3. Dropout

Hyperparameter tuning Can be done for identifying no. of hidden Layers & no. of neurons in each hidden Layer.

- Layers = [[20],[10],[30],[40]] → 1 hidden Layer with different options of no. of neurons in hidden Layer
- Layers = [[20],[40,20],[45,30,15]] → Multiple hidden Layers with different options of no. of neurons.

Hyperparameter tuning Can be done for identifying best activation function for hidden Layers.

- activations = ["relu", "sigmoid"]

Hyperpameter tuning Can be done for identify best optimizers

- Optimizer = ["adam", "rmsprop"]
  - "adam" → gradient desent
  - "rmsprop" → Stochatic gradient desent.

Hyper parameter tunning for batchsize for building/training model.

- batch size = [10, 16, 32, 64, 128, 256]

Code: tunning :-

```
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(input_dim=12, units=6, kernel_
                         initializer="uniform",
    classifier.add(Dense(units=6, kernel_initializer="uniform
                         ,activation="relu"))
    classifier.add(Dense(units=1, kernel_initializer=
                         "uniform", activation="sigmoid"))
    classifier.compile(optimizer=optimizer, loss="binary_
                       crossentropy", metrics=["accuracy"])
    return classifier

# Classifier = KerasClassifier(build_fn=build_classifier)
# parameters = {"batch_size": [10,32], "epochs": [50,100],
               "optimizer": ["adam", "rmsprop"]}
```

```
from sklearn.model_selection import GridSearchCV

# grid = GridSearchCV(estimator = classifier, param_grid =
                      parameters, scoring = "accuracy", cv= 5)

# grid_result = grid.fit(x_train, y_train)
```

Out: Epoch 1/100
200/200 [==========] - 0s 711us/step - loss: 0.5877, accu. 0.7117]
Epoch 2/100
200/200 [==========] - 0s 630us/step - loss: 0.4392 - acc -0.8003)

.....

Epoch 100/100
200/200 [==========] - 0s 83 0us/step - loss: 0.6392. acc-84]

```
# grid_result.best_params_
```

Out: {"batch_size": 32, "epochs": 50, "optimizer": "adam"}

best accuracy

```
# grid_result.best_score_
```

Out: 0.844125

```
# y_pred = grid_result.predict(x_test)
# y_pred = (y_pred > 0.5)
```

accuracy, confusion_matrix

```
from sklearn.metrics import confusion_matrix, accuracy_
                            score.

# print ("test accuracy:", accuracy_score (y_test, y_pred))
# confusion_matrix (y_test, y_pred)
```

Out: Test accuracy : 0.8405

```
array([[1530,  65]
       [ 254,  151]],
```

10/05/22

5:30 pm.