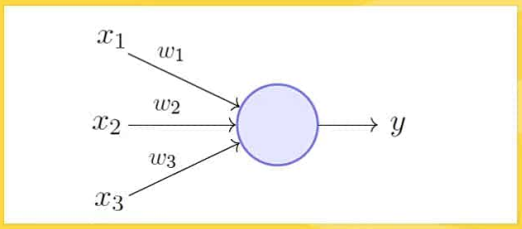
Perceptron

1. The perceptron takes in a vector x as the input
2. Multiplies it by the corresponding weight vector w
3. Then adds it to the bias, b

Input signals weighted and combined as **Net Input**

4. The output is then passed through an activation function to map the input between the required values – Binary Classification



1_DL_Perceptron.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk

Editing

```
[19] from numpy import array #For Array Initialization
     from numpy import random #For Randomly choosing Numbers
     from numpy import dot #For Doing DOT Product
     from random import choice
```

Activation Function

```
activationFn = lambda x: 0 if x < 0 else 1 #step activation function (if i/p is negative o/p is 0
```

Initialized Dataset

0s completed at 20:00

1_DL_Perceptron.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk

Editing

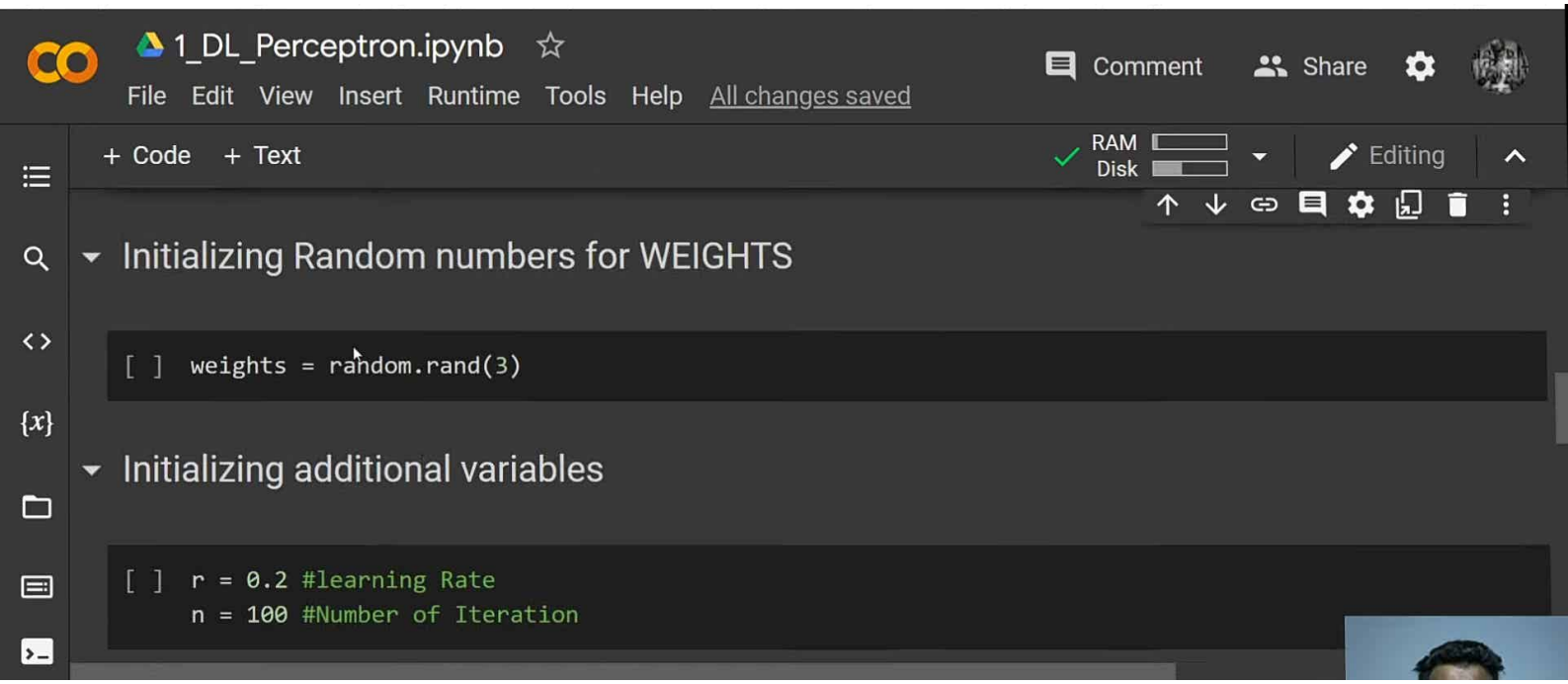
Initialized Dataset

```
dataset = [
    (array([0,0,1]), 0), #array([x,y,b],e) x,y=Input , b=bias, e=Expected O/P to validate
    (array([0,1,1]), 1),
    (array([1,0,1]), 1),
    (array([1,1,1]), 1),
]
print(dataset)
```

```
[(array([0, 0, 1]), 0), (array([0, 1, 1]), 1), (array([1, 0, 1]), 1), (array([1, 1,
```

0s completed at 20:00

basic.py


1_DL_Perceptron.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing
Initializing Random numbers for WEIGHTS
[ ] weights = random.rand(3)
Initializing additional variables
[ ] r = 0.2 #learning Rate
n = 100 #Number of Iteration

1_DL_Perceptron.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk

Editing

Training our Model

```
for j in range(n):
    x, expected = choice(dataset) #random input set, Includes repeated numbers
    #print(x,expected)
    result = dot(weights, x) #dot product of the input and the weight vectors
    err = expected-activationFn(result)
    weights += r * err * x #calculate the correction factor, added to weights to improve o/p in n
    #If the expected value is bigger, the weights should be increased, and if expected value is s
```

0s

Evaluating Model

0s completed at 20:00

basic.py

1_DL_Perceptron.ipynb

File Edit View Insert Runtime Tools Help

Comment Share

+ Code + Text

RAM Disk

Editing

```
for x, _ in dataset:
    result = dot(x, weights)
    print("Input: {}  ResultBAFn: {} ResultAFn: {}".format(x[:2], round(result,3), activationFn(r
```

```
Input: [0 0]  ResultBAFn: -0.102 ResultAFn: 0
Input: [0 1]  ResultBAFn: 0.784 ResultAFn: 1
Input: [1 0]  ResultBAFn: 0.57 ResultAFn: 1
Input: [1 1]  ResultBAFn: 1.456 ResultAFn: 1
```

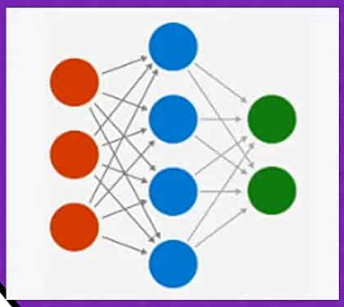
0s completed at 20:00

basic.py

Deep Learning Terminology - 1

Backpropagation

- Backward propagation of error
- After each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases)
- The level of adjustment is determined by the gradients of the function with respect to those parameters
- Proper tuning of the weights ensures lower error rates



Backpropagation is for Calculating Gradients Effectively

Deep Learning Terminology - 2

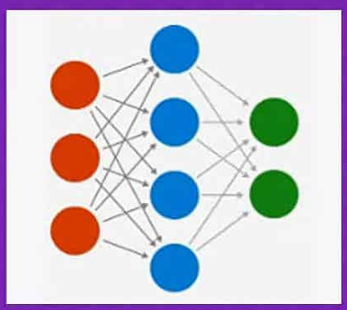
Loss Function

- Repeatedly estimating error of the Model, So that weights can be updated to reduce loss

1. Regression Loss Function
   - Mean squared error loss
   - Mean squared logarithmic error loss
   - Mean absolute error loss
2. Binary Classification loss function
   - Binary Cross Entropy
   - Hinge Loss
   - Squared Hinge Loss
3. Multiclass Classification loss function
   - Multiclass cross entropy
   - Sparse multiclass cross entropy
   - Kullback–Leibler divergence

Deep Learning Terminology - 3

Gradient Descent - Optimizer



- **Optimizer-** To minimize the loss function or error term
- Gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates
- Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error

1. Batch gradient descent
2. Stochastic gradient descent
3. Mini-batch gradient descent

Optimizer train NN using Gradient computed with Back Propagation