Perceptron

Perceptron Model (1957)

- Perceptron was conceptualized by Frank Rosenblatt in the year 1957 and it is the most primitive form of artificial neural networks.
- McCulloch-Pitts neuron had very limiting capabilities and was not a true machine learning model either.
- **Limitation of McCulloch-Pitts Neuron**:
  - McCulloch-Pitts neuron could only take boolean values as inputs. But the real world problems are not limited to only boolean values. For e.g. you cannot give real number like age, price, area etc. as inputs.
  - It does not considers weights to the inputs. Weights are very important to indicate which of the input features plays more important role in output and which features plays very little role. So without weights it cannot determine the importance of input data.
  - The only parameter which is there in this model is Threshold Θ and that too we are computing it on our own. There is no learning taking place to determine the optimum value of Threshold based on past data. So this is not a true machine learning model.

Perceptron Model

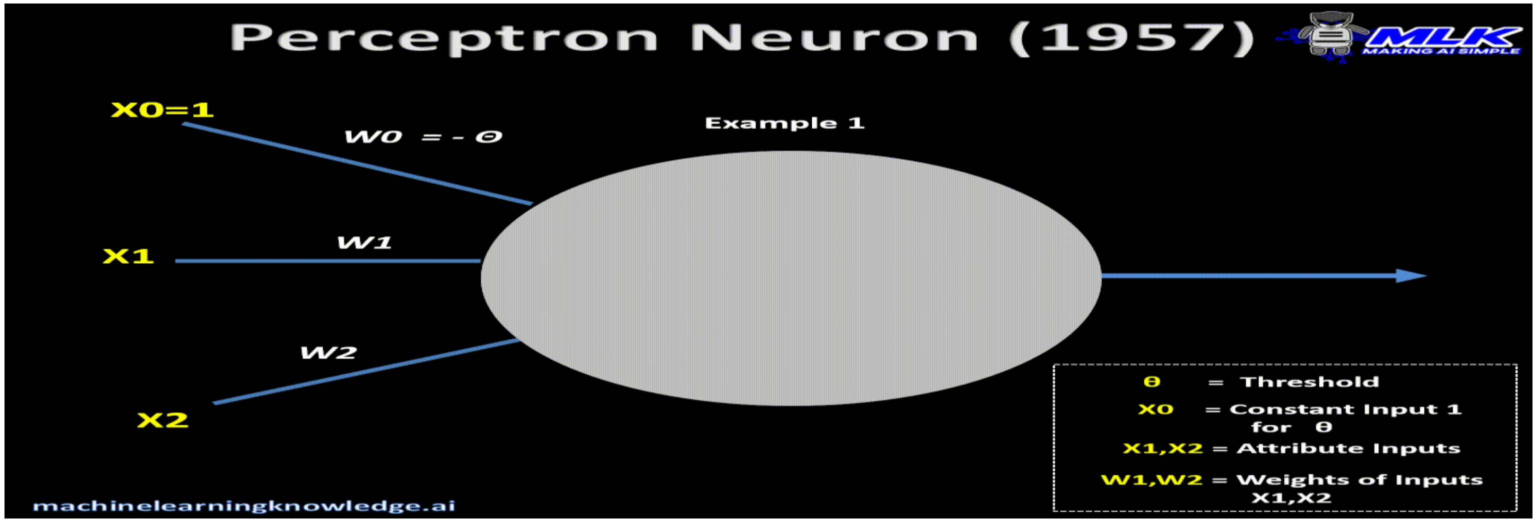
- It’s various parts:
  - Inputs: The inputs to perceptron is no longer limited to boolean. The input attributes can represent any real number.
  - Weights: Each input has a weight associated with it in perceptron model. These weights are initially unknown but are learned by Perceptron during training phase.
  - Neuron: Neuron is a computational unit which has incoming input signals. The input signals are computed and an output is fired.
  - Output: This is simply the output of the neuron which produces only binary values of 0 or 1. The value of 0 indicates that the neuron does not fire, the value of 1 indicates the neuron does fire.

Activation Function

- Here the activation function used is the *step function.*
- It sees if the summation is more than equal to calculated Threshold value , if yes then neuron should fire (i.e. output =1) if not the neuron should not fire (i.e. output =0).
- *Neuron fires*: Output =1 ,
  if *Summation(Inputs*Weights) >= Threshold*
- *Neuron does not fire*: Output =0 , if
  Summation(Inputs*Weights) < Threshold

How Perceptron Model works

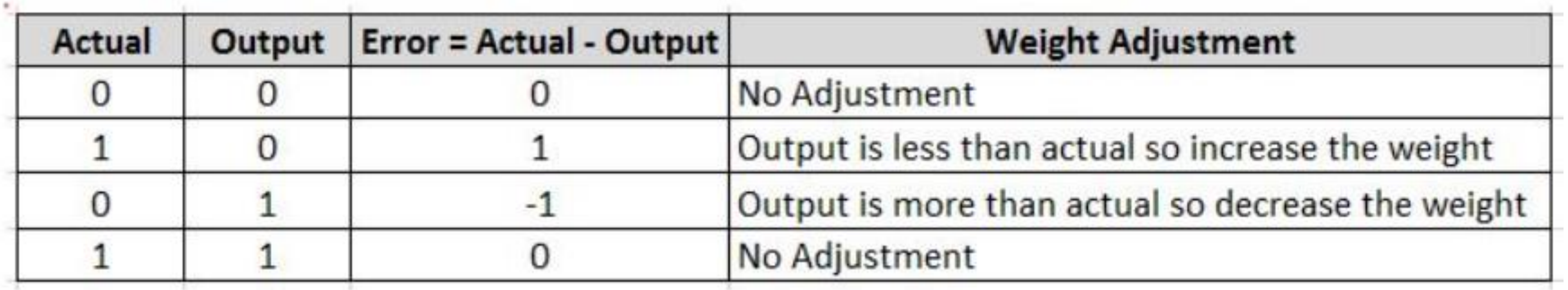
- The input data is enabled into the Perceptron.
- Apply dot product between input & it's respective weights and sum them up.
- Apply step function on above summation –
  - If output of step function >= 0 then perceptron is fired, output = 1
  - If output of step function < 0 then perceptron does not fire, output = 0



Training a Perceptron

- **Learning Parameters**:
  - Machine learning models have parameters whose optimal value it learns during the training phase.
  - In a Perceptron the parameters that are learned during training phase are –
    - Weights of input parameters
    - bias
    - Threshold value

- **Errors and Weight adjustment:**
  - During training phase, the weights are initialized randomly and with each data, the error is calculated and weights are adjusted accordingly so as to reduce the error.
  - As you have seen above, the possible output of perceptron is only binary i.e 0 or 1. So there are only four possible combination of actual value and output value.
  - Below chart shows this combination and will help you to get intuition how weights should be adjusted in case of error.



Actual	Output	Error = Actual - Output	Weight Adjustment
0	0	0	No Adjustment
1	0	1	Output is less than actual so increase the weight
0	1	-1	Output is more than actual so decrease the weight
1	1	0	No Adjustment

Perceptron Learning Algorithm

- Let us first see the pseudo code of this algorithm with respect to our example in above animation.

```
■ Initialize all the weights randomly

  Start Loop

     For each data in dataset –

    1. Give input to Perceptron and get the output
    2. Calculate the Error = Actual – Output
    3. Adjust the weights of each input as –
       ■ Weight(X0) = Weight(X0) +  Error * Input(X0)
       ■ Weight(X1) = Weight(X1) +  Error * Input(X1)
       ■ Weight(X2) = Weight(X2) +  Error * Input(X2)

     End For

  Continue Loop while Error still exists
```

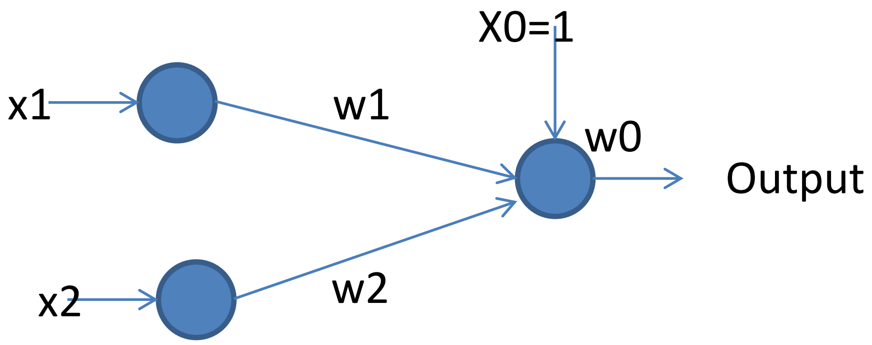
Intuition behind Perceptron Learning Algorithm

- For most part of the pseduo code, things should be self explanatory.
- We are doing multiple loops over dataset and in each loop we are passing data one by one through perceptron, calculating error and adjusting weights.
- The loop is continued till there are no errors.
- Example 1:
  - **Actual =1 , Output =0, Error =1**
  - In this case, since Output is less than actual, we would like to increase the weight to push the output towards actual. It requires a positive weight adjustment
  - Weight = Weight + Error*Input = Weight + 1*Input (weight adjustment is positive)
- Example 2:
  - **Actual =0 , Output =1, Error = -1**
  - In this case, since Output is more than actual, we would like to decrease the weight to pull back the output towards actual. It requires a negative weight adjustment
  - Weight = Weight + Error*Input = Weight – 1*Input (weight adjustment is negative)
- Example 3:
  - **Actual =0 , Output =0, Error =0**
  - In this case since Output and Actual is same, there is no error so zero weight adjustment is required.
  - Weight = Weight + Error*Input = Weight – 0*Input = Weight (no weight adjustment)

Numerical: AND function using Perceptron

- Training Data:

X1	X2	Actual
0	0	0
0	1	0
1	0	0
1	1	1



- Number of weights: 3
- Suppose initial weights are equal to 1.
  - i.e. w0=w1=w2=1
- Consider threshold =0
- Epochs = 3

Epoch 1: Error Calculation

- Weights: w0=1, w1=1, w2= 1
- Ynet=x0*w0+x1*w1+x2*w2
- Input (0,0):
  - Ynet=1*1+0*1+0*1=1
  - Ynet>=0: neuron fires
  - Output=1
  - Error=Actual-Output=0-1=-1 (Error non zero, change weights)
  - W0=1+(-1)*1=0
  - W1=1+(-1)*0=1
  - W2=1+(-1)*0=1
- Updated weights: w0=0, w1=1, w2=1

Epoch 1: Error Calculation

- weights: w0=0, w1=1, w2=1
- Ynet=x0*w0+x1*w1+x2*w2
- Input (0,1):
  - Ynet=1*0+0*1+1*1=1
  - Ynet>=0: neuron fires
  - Output=1
  - Error=Actual-Output=0-1=-1 (Error non zero, change weights)
  - W0=0+(-1)*1=-1
  - W1=1+(-1)*0=1
  - W2=1+(-1)*1=0
- Updated weights: w0=-1, w1=1, w2=0

Epoch 1: Error Calculation

- weights: w0=-1, w1=1, w2=0
- Ynet=x0*w0+x1*w1+x2*w2
- Input (1,0):
  - Ynet=1*-1+1*1+0*0=0
  - Ynet>=0: neuron fires
  - Output=1
  - Error=Actual-Output=0-1=-1 (Error non zero, change weights)
  - W0=-1+(-1)*1=-2
  - W1=1+(-1)*1=0
  - W2=0+(-1)*0=0
- Updated weights: w0=-2, w1=0, w2=0

Epoch 1: Error Calculation

- weights: w0=-2, w1=0, w2=0
- Ynet=x0*w0+x1*w1+x2*w2
- Input (1,1):
  - Ynet=1*-2+1*0+1*0=-2
  - Ynet>=0: neuron does not fire
  - Output=0
  - Error=Actual-Output=1-0=1 (Error non zero, change weights)
  - W0=-2+(1)*1=-1
  - W1=0+(1)*1=1
  - W2=0+(1)*1=1
- Updated weights: w0=-1, w1=1, w2=1

- Final Weights after 1st iteration:
  - w0=-1, w1=1, w2=1

Epoch 2: Error Calculation

- Weights: w0=-1, w1=1, w2= 1
- Ynet=x0*w0+x1*w1+x2*w2
- Input (0,0):
  - Ynet=1*(-1)+0*1+0*1=-1
  - Ynet>=0: neuron does not fire
  - Output=0
  - Error=Actual-Output=0-0=0 (Zero Error, No change in weights)

Epoch 2: Error Calculation

- weights: w0=-1, w1=1, w2=1
- Ynet=x0*w0+x1*w1+x2*w2
- Input (0,1):
  - Ynet=1*(-1)+0*1+1*1=0
  - Ynet>=0: neuron fires
  - Output=1
  - Error=Actual-Output=0-1=-1 (Error non zero, change weights)
  - W0=(-1)+(-1)*1=-2
  - W1=1+(-1)*0=1
  - W2=1+(-1)*1=0
- Updated weights: w0=-2, w1=1, w2=0

Epoch 2: Error Calculation

- weights: w0=-2, w1=1, w2=0
- Ynet=x0*w0+x1*w1+x2*w2
- Input (1,0):
  – Ynet=1*-2+1*1+0*0=-1
  – Ynet>=0: neuron does not fire
  – Output=0
  – Error=Actual-Output=0-0=0 (Error zero, No change in weights)

Epoch 2: Error Calculation

- weights: w0=-2, w1=1, w2=0
- Ynet=x0*w0+x1*w1+x2*w2
- Input (1,1):
  - Ynet=1*-2+1*1+1*0=-1
  - Ynet>=0: neuron does not fire
  - Output=0
  - Error=Actual-Output=1-0=1 (Error non zero, change weights)
  - W0=-2+(1)*1=-1
  - W1=1+(1)*1=2
  - W2=0+(1)*1=1
- Updated weights: w0=-1, w1=2, w2=1

- Final Weights after 2nd iteration:
  - w0=-1, w1=2, w2=1

Epoch 3: Error Calculation

- Weights: w0=-1, w1=2, w2= 1
- Ynet=x0*w0+x1*w1+x2*w2
- Input (0,0):
  - Ynet=1*(-1)+0*2+0*1=-1
  - Ynet>=0: neuron does not fire
  - Output=0
  - Error=Actual-Output=0-0=0 (Zero Error, No change in weights)

Epoch 3: Error Calculation

- weights: w0=-1, w1=2, w2= 1
- Ynet=x0*w0+x1*w1+x2*w2
- Input (0,1):
  - Ynet=1*(-1)+0*2+1*1=0
  - Ynet>=0: neuron fires
  - Output=1
  - Error=Actual-Output=0-1=-1 (Error non zero, change weights)
  - W0=(-1)+(-1)*1=-2
  - W1=2+(-1)*0=2
  - W2=1+(-1)*1=0
- Updated weights: w0=-2, w1=2, w2=0

Epoch 3: Error Calculation

- weights: w0=-2, w=2, w2=0
- Ynet=x0*w0+x1*w1+x2*w2
- Input (1,0):
  - Ynet=1*-2+1*2+0*0=0
  - Ynet>=0: neuron fires
  - Output=1
  - Error=Actual-Output=0-1=-1 (Error non zero, change weights)
  - W0=(-2)+(-1)*1=-3
  - W1=2+(-1)*1=1
  - W2=0+(-1)*0=0
- Updated weights: w0=-3, w1=1, w2=0

Epoch 3: Error Calculation

- weights: w0=-3, w1=1, w2=0
- Ynet=x0*w0+x1*w1+x2*w2
- Input (1,1):
  - Ynet=1*-3+1*1+1*0=-2
  - Ynet>=0: neuron does not fire
  - Output=0
  - Error=Actual-Output=1-0=1 (Error non zero, change weights)
  - W0=-3+(1)*1=-2
  - W1=1+(1)*1=2
  - W2=0+(1)*1=1
- Updated weights: w0=-2, w1=2, w2=1

- Final Weights after 3rd iteration:
  - w0=-2, w1=2, w2=1

- So, Even after 3rd iteration, error is non-zero.
- We have to train the model for more iterations.
- **Implement the Perceptron Algorithm in Python for AND function, and find the final weights.**

Some points to consider

- Perceptron addresses the three drawbacks that we discussed about McCulloch-Pitts neuron above. It can take real number as inputs and weights are also associated with inputs. This is also a true machine learning model which can be trained using past data.
- It still, however can produce only binary output. So perceptron can be used in case of binary classification problems only.
- Perceptron can classify only those data which are linearly separable. So it cannot solve a XOR problem.
- If the data is linearly separable, perceptron learning algorithm will always converge in finite number of loops.

Thanks