Perceptron

a Perceptron as a system that learns using labeled examples (i.e., supervised learning) of feature vectors (or raw pixel intensities), mapping these inputs to their corresponding output class labels.The idea is to do weighted sum on the neural nodes and then use step function as activated function.so we implement Perceptron below

1) Script:We first define Perceptron class so that it takes in two parameters,number of features(node) and alpha(learning rate) which is default to 0.1 which make it optional.we have N+1 when initializing the weight because of we want the bias to be learnt along side the weight and not as a separate variable

```
# import the necessary packages
import numpy as np

class Perceptron:
    def __init__(self, N, alpha=0.1):
        # initialize the weight matrix and store the learning rate
        self.W = np.random.randn(N + 1) / np.sqrt(N)
        self.alpha = alpha
```

2) Script:We define the step function as method of the class

```
def step(self, x):
        # apply the step function
        return 1 if x > 0 else 0
```

3) Script:For training we define a fit method which takes in the input X,label y and epoch default to 10.The input is adjusted to make room for bias as the reason given in step 1

```
def fit(self, X, y, epochs=10):
        # insert a column of 1's as the last entry in the feature
        # matrix -- this little trick allows us to treat the bias
        # as a trainable parameter within the weight matrix
        X = np.c_[X, np.ones((X.shape[0]))]
```

4) Script:we loop through each epoch and for each epoch each dataset set.For each row of data we find the step function of the dot product of the row and the weight.the result is our prediction which we check against the target.We update the weight matrix if only prediction and target are not the same.

```
# loop over the desired number of epochs
        for epoch in np.arange(0, epochs):
            # loop over each individual data point
            for (x, target) in zip(X, y):
                # take the dot product between the input features
                # and the weight matrix, then pass this value
                # through the step function to obtain the prediction
                p = self.step(np.dot(x, self.W))

                # only perform a weight update if our prediction
                # does not match the target
                if p != target:
                    # determine the error
                    error = p - target

                    # update the weight matrix
                    self.W += -self.alpha * error * x
```

5) Script: We define the prediction method, taking the input dataset,setting the addBias parameter default to true.We first check the dataset is a matrix,add the bias if not false and do prediction on the dataset as done in the fit method

```
def predict(self, X, addBias=True):
        # ensure our input is a matrix
        X = np.atleast_2d(X)

        # check to see if the bias column should be added
        if addBias:
            # insert a column of 1's as the last entry in the
feature
            # matrix (bias)
            X = np.c_[X, np.ones((X.shape[0]))]

        # take the dot product between the input features and the
        # weight matrix, then pass the value through the step
        # function
        return self.step(np.dot(X, self.W))
```

Now that we have defined our Perceptron we apply on the or,and and xor dataset set

OR Dataset:

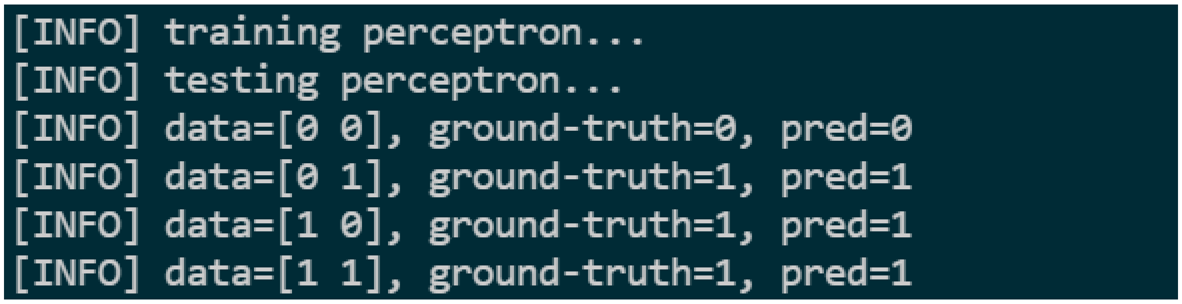
1) Script:we import the Perceptron package and numpy.we set up the or dataset initialize the perceptron and with the number of feature and make the perceptron learn

```python
# import the necessary packages
from pyimagesearch.nn import Perceptron
import numpy as np

# construct the OR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [1]])

# define our perceptron and train it
print("[INFO] training perceptron...")
p = Perceptron(X.shape[1], alpha=0.1)
p.fit(X, y, epochs=20)
```

2) Script:After learning we predict using the sample dataset to see if we would get the same result.

```python
# now that our perceptron is trained we can evaluate it
print("[INFO] testing perceptron...")

# now that our network is trained, loop over the data points
for (x, target) in zip(X, y):
    # make a prediction on the data point and display the result
    # to our console
    pred = p.predict(x)
    print("[INFO] data={}, ground-truth={}, pred={}".format(
        x, target[0], pred))
```

Result:

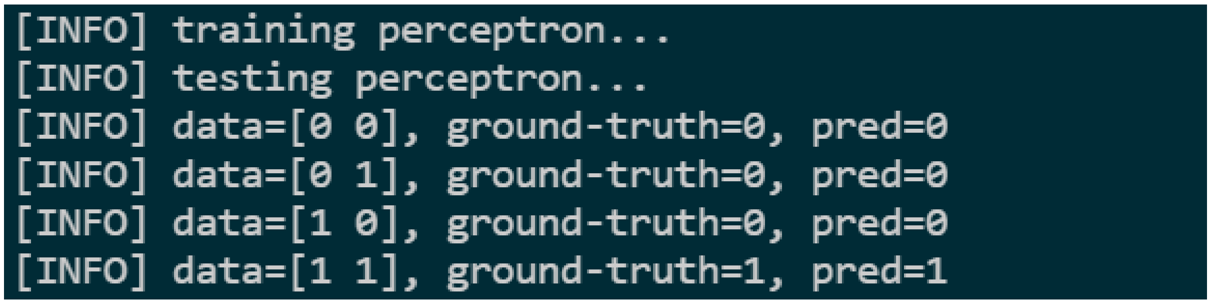


```
[INFO] training perceptron...
[INFO] testing perceptron...
[INFO] data=[0 0], ground-truth=0, pred=0
[INFO] data=[0 1], ground-truth=1, pred=1
[INFO] data=[1 0], ground-truth=1, pred=1
[INFO] data=[1 1], ground-truth=1, pred=1
```

AND Perceptron:We the same process used in OR for AND but using the dataset below for AND

```python

# construct the AND dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [0], [0], [1]])
```

Result:

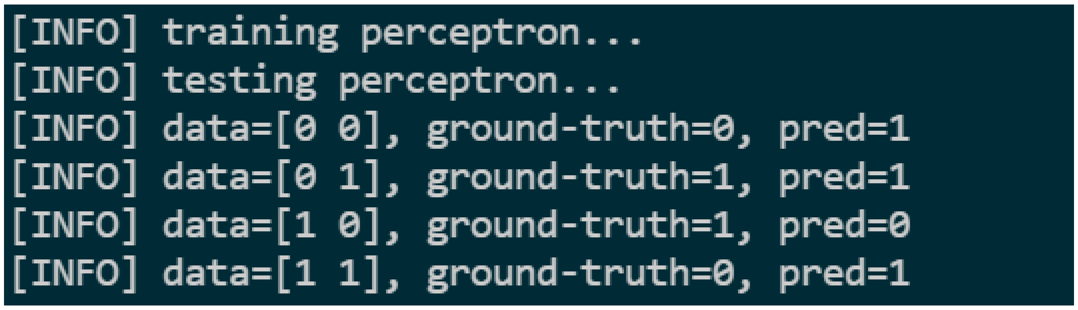


```
[INFO] training perceptron...
[INFO] testing perceptron...
[INFO] data=[0 0], ground-truth=0, pred=0
[INFO] data=[0 1], ground-truth=0, pred=0
[INFO] data=[1 0], ground-truth=0, pred=0
[INFO] data=[1 1], ground-truth=1, pred=1
```

XOR:repeating the same process but with dataset below

```
# construct the XOR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
```

Result:



```
[INFO] training perceptron...
[INFO] testing perceptron...
[INFO] data=[0 0], ground-truth=0, pred=1
[INFO] data=[0 1], ground-truth=1, pred=1
[INFO] data=[1 0], ground-truth=1, pred=0
[INFO] data=[1 1], ground-truth=0, pred=1
```