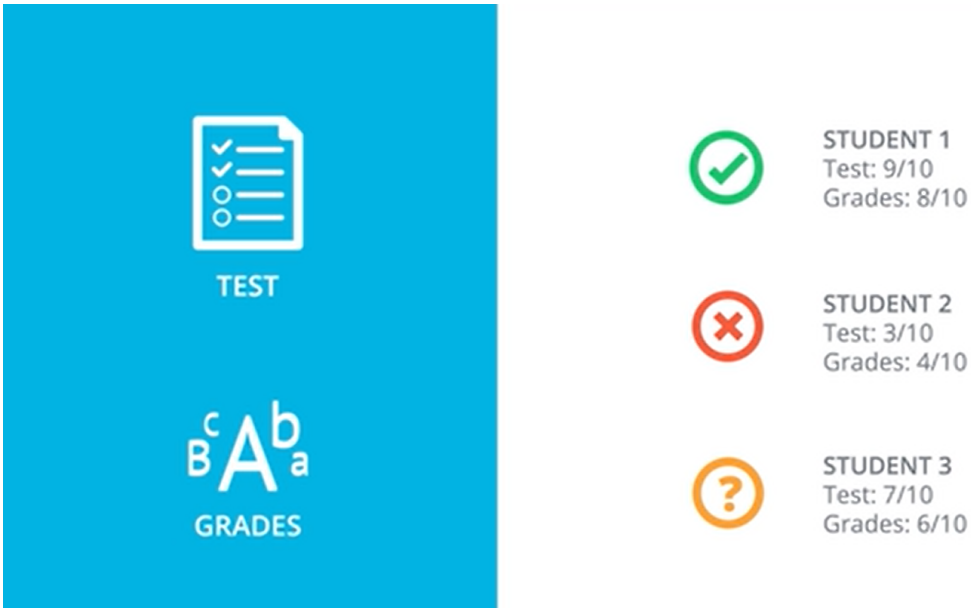
Perceptron_Algorithm

January 28, 2022

1 Perceptron Algorithm

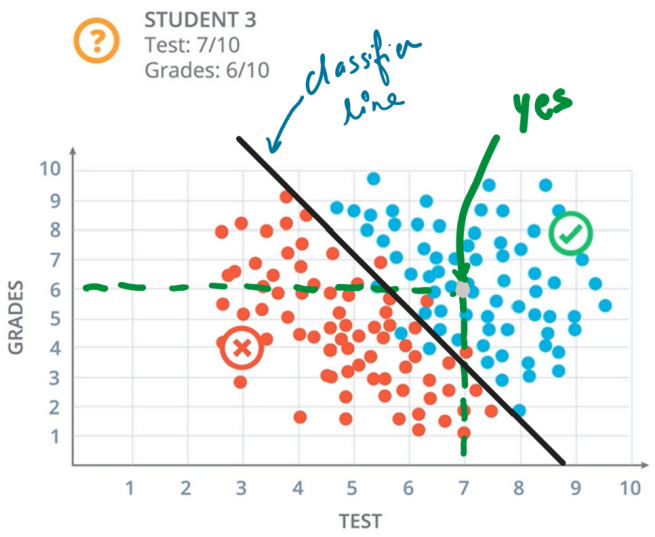
Perceptron algorithm is one of the technique to solve classification type machine learning model. Classification method is one of Supervised machine learning method which provides or rather classifies the answer in one or the other category.

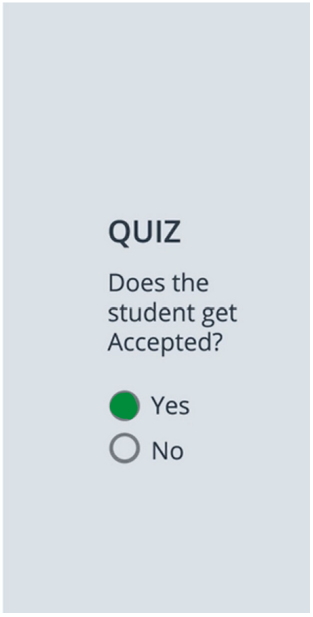
Let's take an example about a student's acceptance in an University. His/her acceptance depends on the two parameters namely, Test score and overall grades. From the image below, it is easy to determine a Student's acceptance for top two Students based on test score and grades. However, it is unclear or not classfied for the third Student of whether or not he/she will accepted. Here's where perceptron algorithm comes as a tool to make classfication easy.



Let's try to put this into deeper perspective by plotting test scores and grades. This plotting comes from the previous data gathered for the students who got accepted and rejected. See, again, data is so Important! And our predictions in machine learning are based on huge collection of data.

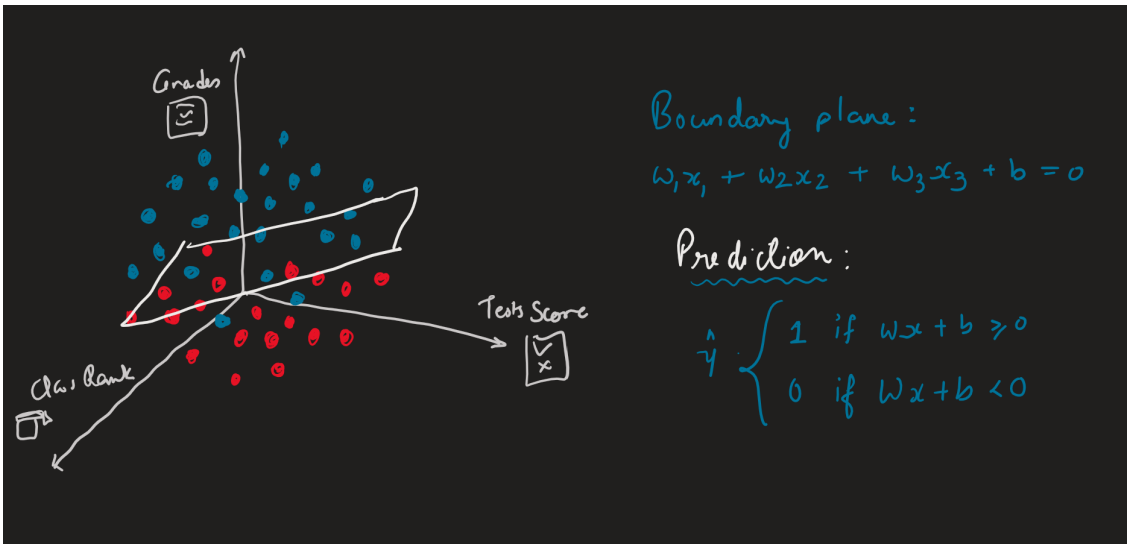
Now, it can be seen from the plot that third Student's test score and grades fall in a classfication zone of accepted students, which means that the third student is also accepted into the university. We can confidently say that the classifier line divides the data safely to categorize acceptance and rejection. But the question is how do we determine this line?

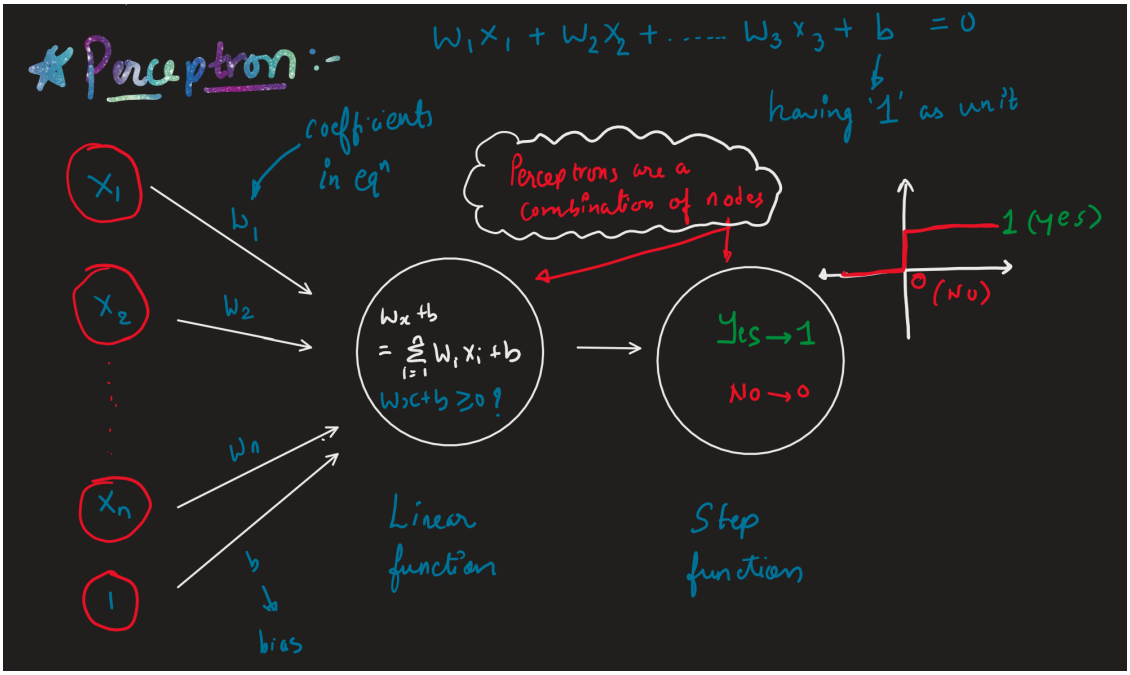




This boundary line can be represented by a linear equation of line, and assuming that horizontal variables are x1 and vertical variables are x2, the eqaution of line for above case will be $2x1 + x2 - 18 = 0$. More general form of this equation can be written as $w1x1 + w2x2 + b = 0$. In vectorized form this can be written as $Wx + b = 0$. If $Wx + b >= 0$, the prediction can be said to be in "Yes" category, in above case "accepcted" category and if $Wx + b < 0$, the prediction can be said to be in "No" category, in above case "rejected" category.

In higher dimensions, the equation can be written as $w1x1 + w2x2 + ... + wnxn + b = 0$. Let's say if we have three different variables, namley Test score, Grades, Class Rank, now in our example of Student Acceptance, our equation will be written in higher dimenions as:

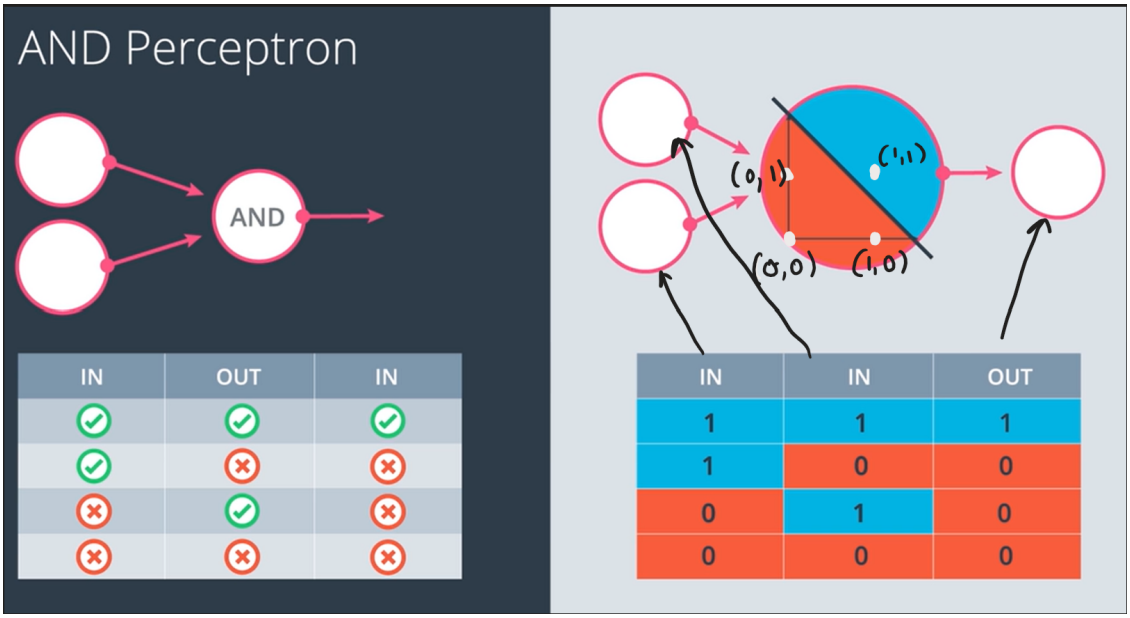




1.0.1 Perceptrons And Logical Operators

One of the great application of perceptron is using them as Logical operators. These logical operators will classify the output/predictive variable based on the input variables. Why not we see that as an example!?

1.0.2 AND Perceptron



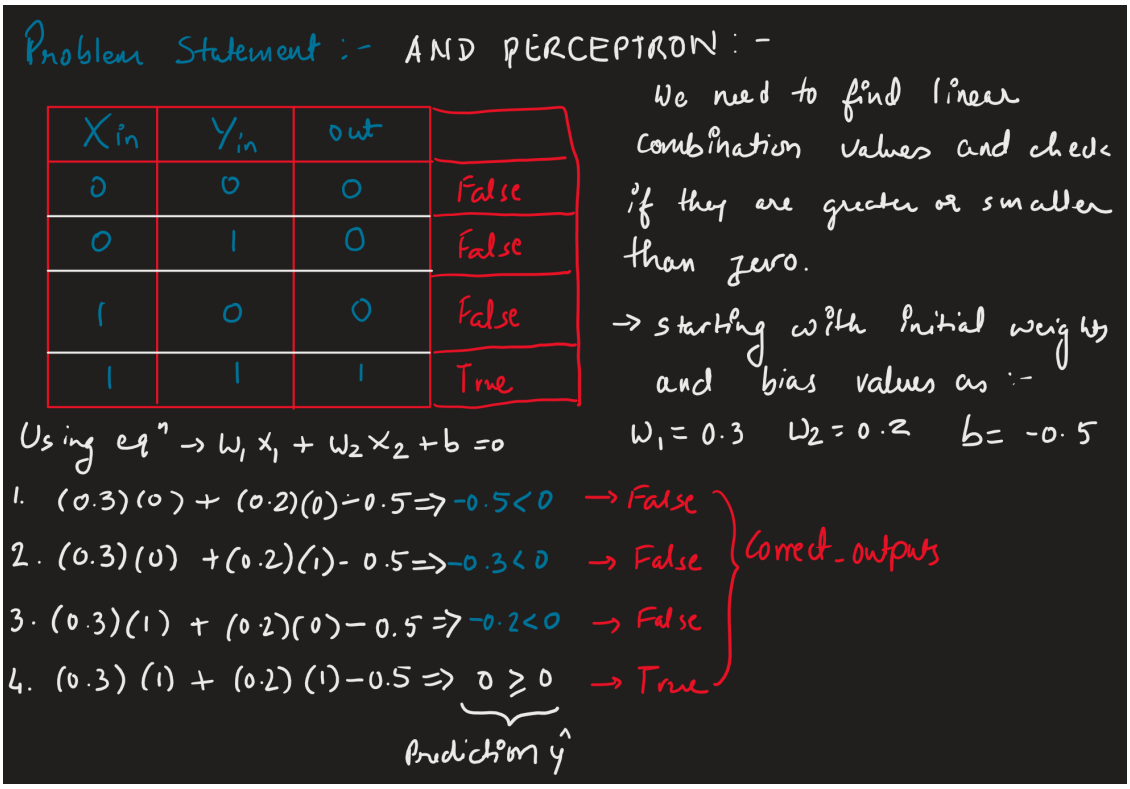
1.0.3 Problem Statement 1:

Set the weights (weight1, weight2) and bias (bias) to values that will correctly determine the AND operation as shown above. More than one set of values will work!

[221]:
```
import pandas as pd

# TODO: Set weight1, weight2, and bias
weight1 = 0.3
weight2 = 0.2
bias = -0.5


# DON'T CHANGE ANYTHING BELOW
# Inputs and outputs
test_inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
correct_outputs = [False, False, False, True]
outputs = []

# Generate and check output
for test_input, correct_output in zip(test_inputs, correct_outputs):
    linear_combination = weight1 * test_input[0] + weight2 * test_input[1] +␣
 ↪bias
    output = int(linear_combination >= 0)
    is_correct_string = 'Yes' if output == correct_output else 'No'
    outputs.append([test_input[0], test_input[1], linear_combination, output,␣
 ↪is_correct_string])

# Print output
num_wrong = len([output[4] for output in outputs if output[4] == 'No'])
output_frame = pd.DataFrame(outputs, columns=['Input 1', '  Input 2', '  Linear␣
 ↪Combination', '  Activation Output', '  Is Correct'])
if not num_wrong:
    print('Nice!  You got it all correct.\n')
else:
    print('You got {} wrong.  Keep trying!\n'.format(num_wrong))
print(output_frame.to_string(index=False))
```

```
Nice!  You got it all correct.

 Input 1    Input 2    Linear Combination    Activation Output    Is Correct
       0          0                  -0.5                    0           Yes
       0          1                  -0.3                    0           Yes
       1          0                  -0.2                    0           Yes
       1          1                   0.0                    1           Yes
```

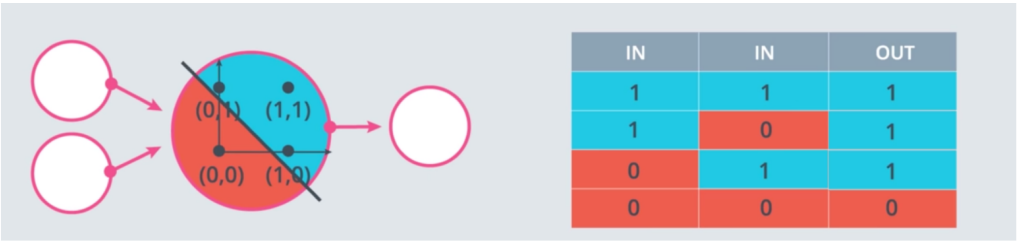
How these input values work with assumed weights and bias have been shown below:



1.0.4 OR Perceptron

The boundary line is moved down the co-ordinates having atleast one input value 1. There are two ways to go from AND perceptron to OR perceptron:

1. By increasing weights
2. By reducing the magnitude of bias



1.0.5 NOT Perceptron

1.0.6 Problem Statement 2:

Unlike the other perceptrons we looked at, the NOT operation only cares about one input. The operation returns a 0 if the input is 1 and a 1 if it's a 0. The other inputs to the perceptron are ignored.

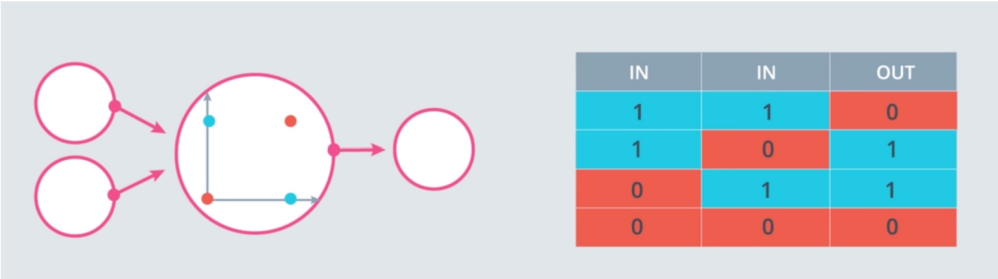
In this quiz, you'll set the weights (weight1, weight2) and bias bias to the values that calculate the NOT operation on the second input and ignores the first input.

[222]:
```
import pandas as pd

# TODO: Set weight1, weight2, and bias
weight1 = 1.75
weight2 = -2
bias = -0


# DON'T CHANGE ANYTHING BELOW
# Inputs and outputs
test_inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
correct_outputs = [True, False, True, False]
outputs = []

# Generate and check output
for test_input, correct_output in zip(test_inputs, correct_outputs):
    linear_combination = weight1 * test_input[0] + weight2 * test_input[1] +␣
 ↪bias
    output = int(linear_combination >= 0)
    is_correct_string = 'Yes' if output == correct_output else 'No'
    outputs.append([test_input[0], test_input[1], linear_combination, output,␣
 ↪is_correct_string])

# Print output
num_wrong = len([output[4] for output in outputs if output[4] == 'No'])
output_frame = pd.DataFrame(outputs, columns=['Input 1', '  Input 2', '  Linear␣
 ↪Combination', '  Activation Output', '  Is Correct'])
if not num_wrong:
    print('Nice!  You got it all correct.\n')
else:
    print('You got {} wrong.  Keep trying!\n'.format(num_wrong))
print(output_frame.to_string(index=False))
```

```
Nice!  You got it all correct.

 Input 1    Input 2    Linear Combination    Activation Output    Is Correct
       0          0                  0.00                    1           Yes
       0          1                 -2.00                    0           Yes
       1          0                  1.75                    1           Yes
       1          1                 -0.25                    0           Yes
```

1.0.7 XOR Perceptron

Last one is XOR perceptron, it gives true only when one of the input is exactly 1.



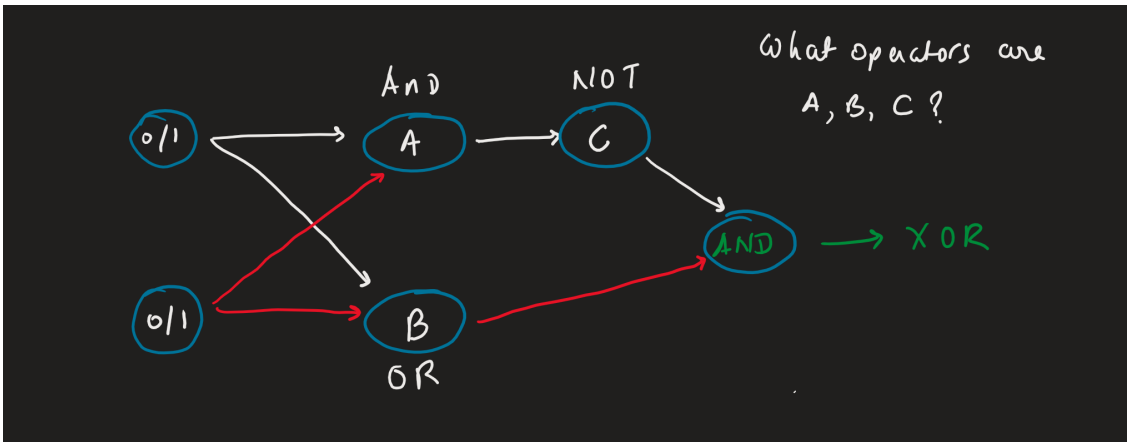
IN	IN	OUT
1	1	0
1	0	1
0	1	1
0	0	0

1.0.8 Problem Statement 3:

Now, let's build a multi-layer perceptron from the AND, NOT, and OR perceptrons to create XOR logic!

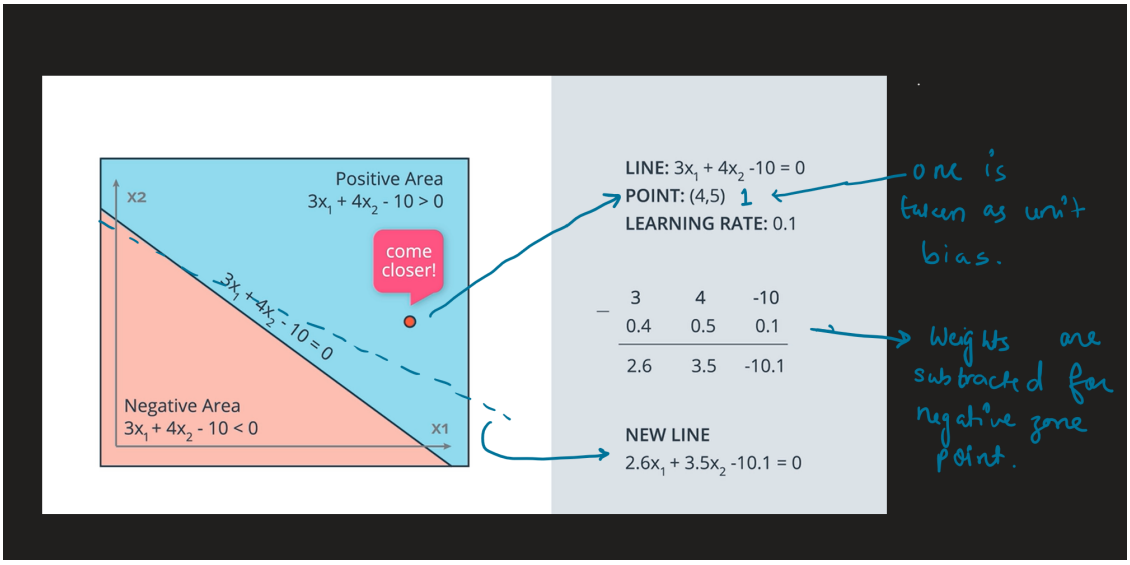
The neural network below contains 3 perceptrons, A, B, and C. The last one (AND) has been given for you. The input to the neural network is from the first node. The output comes out of the last node.

The multi-layer perceptron below calculates XOR. Each perceptron is a logic operation of AND, OR, and NOT. However, the perceptrons A, B, and C don't indicate their operation. In the following quiz, set the correct operations for the perceptrons to calculate XOR.

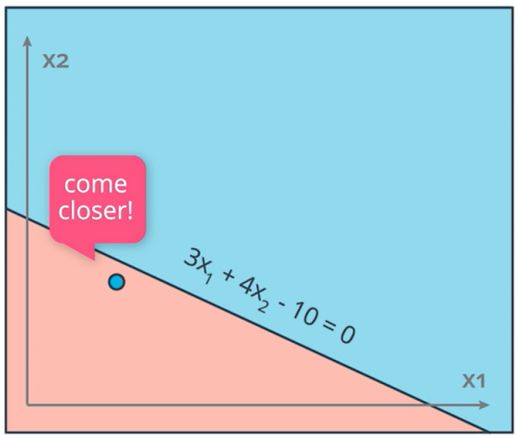


1.1 Perceptron Trick

As we have build up some fundamental and a complex multi layer perceptrons above, in reality they should be able to build themselves. So there is the idea of perceptron trick. At first when the line is created, it will divide the classified points inaccurately. However, our goal is to locate the line in the best position possible to correctly classify the points. Imagine a negative point located incorrectly in the positive area. That would want the line to move closer to it and ultimately surpass it to enter its correct negative zone. An example below demonstrates that,



1.1.1 Problem Statement 3:



LINE: $3x_1 + 4x_2 - 10 = 0$
POINT: (1,1)
LEARNING RATE: 0.1

	3	4	-10
+	0.1	0.1	0.1
	3.1	4.1	-9.9

NEW LINE
$3.1x_1 + 4.1x_2 - 9.9 = 0$

For the second example, where the line is described by 3x1+ 4x2 - 10 = 0, if the learning rate was set to 0.1, how many times would you have to apply the perceptron trick to move the line to a position where the blue point, at (1, 1), is correctly classified?

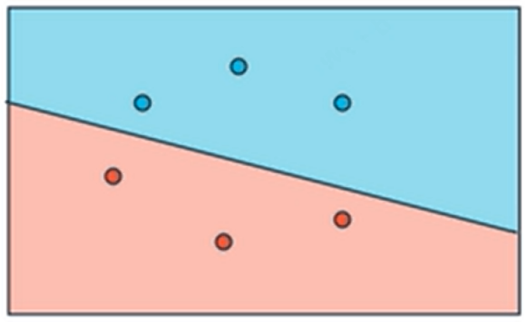
[223]:
```
#weights and bias values from original line
w1, w2, b = 3, 4, -10
  = 0.1  #learning rate

p,q = 1,1 #point of interest (1,1) which is in negative zone so we check keep␣
 ↪on adding the values till we get Wx+b >0
```

```
def new_line(w1,w2,b, ,p,q):
    new_w1 = w1 + p*
    new_w2 = w2 + q*
    new_b = b + 1*
    return new_w1, new_w2, new_b

result = w1*p + w2*q + b
count =0
while(result<0):
    w1,w2,b = new_line(w1,w2,b, ,p,q)
    result = w1*p + w2*q + b
    count += 1

print("Preceptron trick is applied {} times to move the line where the point is␣
 ↪correclty classified".format(count-1))
```

```
Preceptron trick is applied 10 times to move the line where the point is
correclty classified
```

1.1.2 Perceptron Algorithm

Main algorithm can be applied in steps mentioned below. Remember, our main goal is to try to find the best locate the line such that it can correctly classify the points into their respective positive or negative zones.



1. Start with random weights: $w_1, \ldots, w_n, b$

2. For every misclassified point $(x_1,\ldots,x_n)$:
   - 2.1. If prediction = 0:
     - For i = 1 ...n
       - Change $w_i + \alpha x_i$
     - Change b to b + α
   - 2.2. If prediction = 1:
     - For i = 1 ...n
       - Change $w_i - \alpha x_i$
     - Change b to b - α

1.1.3 Problem Statement 4:

In this quiz, you'll have the chance to implement the perceptron algorithm to separate the following data (given in the file data.csv).

Recall that the perceptron step works as follows. For a point with coordinates (p,q)(p,q), label yy,

and prediction given by the equation. $\hat{y} = step(w1x1 + w2x2 + b)$:

1. If the point is correctly classified, do nothing.
2. If the point is classified positive, but it has a negative label, subtract p, q, and from w1, w2 and b respectively.
3. If the point is classified negative, but it has a positive label, add p, q, and to w1, w2 and b respectively.

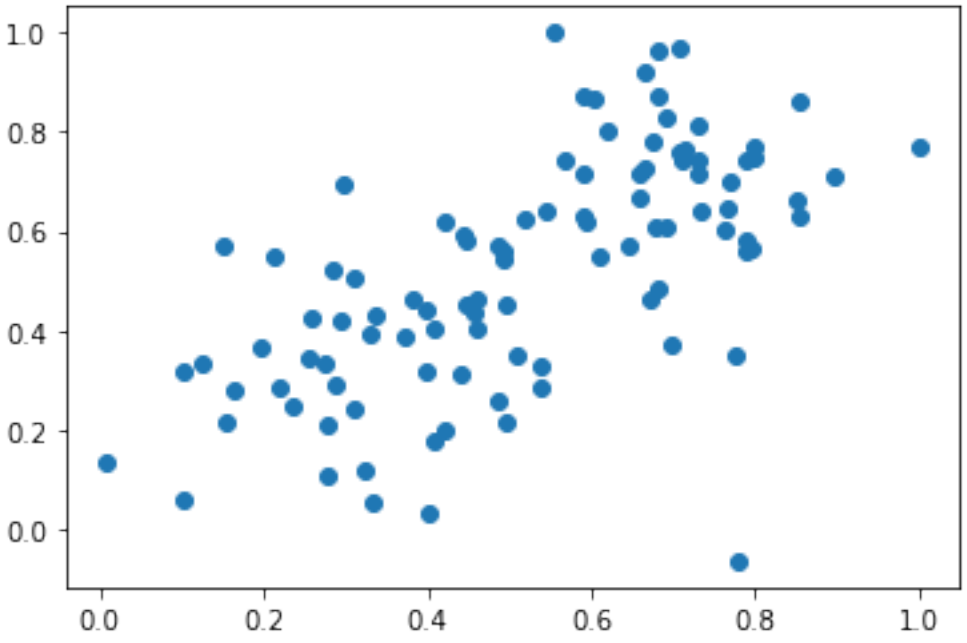
Then click on test run to graph the solution that the perceptron algorithm gives you. It'll actually draw a set of dotted lines, that show how the algorithm approaches to the best solution, given by the black solid line.

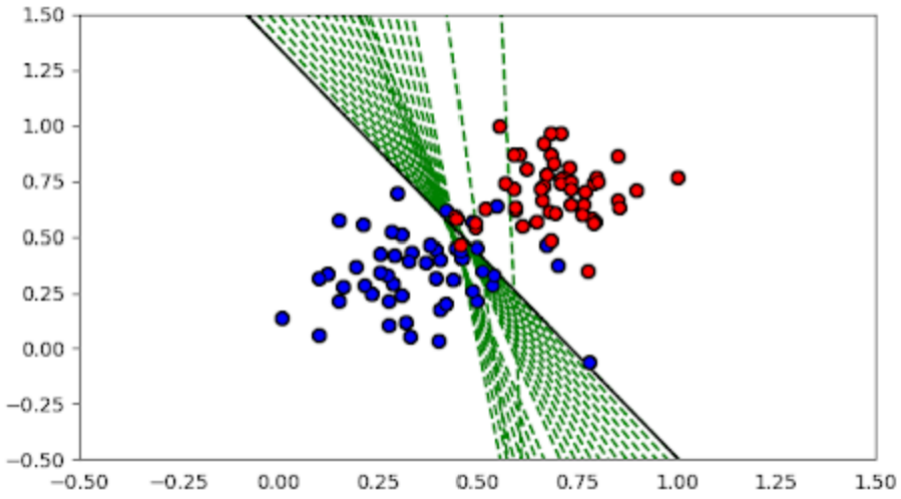
Feel free to play with the parameters of the algorithm (number of epochs, learning rate, and even the randomizing of the initial parameters) to see how your initial conditions can affect the solution!

[224]:
```
import numpy as np
import matplotlib.pyplot as plt
# Setting the random seed, feel free to change it and see different solutions.
np.random.seed(42)

df = pd.read_csv(r'C:\Everything On This PC\Udacity\Intro to ML␣
↪-TensorFlow\Classification\Perceptron_Algorithm\Problem 4\data.csv', header␣
↪=None)
X_pred = df.iloc[:,:2]
X = X_pred.to_numpy()
y = df.iloc[:,-1]


def stepFunction(t):
    if t >= 0:
        return 1
    return 0

def prediction(X, W, b):

    fun = stepFunction((np.matmul(X,W)+b[0]))
    return fun

# TODO: Fill in the code below to implement the perceptron trick.
# The function should receive as inputs the data X, the labels y,
# the weights W (as an array), and the bias b,
# update the weights and bias W, b, according to the perceptron algorithm,
# and return W and b.


def perceptronStep(X, y, W, b, learn_rate = 0.01):
    # Fill in code
    for i in range(len(X)):
```

```
        y_hat = prediction(X[i],W,b)

        if y[i]-y_hat == 1:
            W[0] += X[i][0]*learn_rate
            W[1] += X[i][1]*learn_rate
            b += learn_rate
        elif y[i]-y_hat == -1:
            W[0] -= X[i][0]*learn_rate
            W[1] -= X[i][1]*learn_rate
            b -= learn_rate

    return W, b

# This function runs the perceptron algorithm repeatedly on the dataset,
# and returns a few of the boundary lines obtained in the iterations,
# for plotting purposes.
# Feel free to play with the learning rate and the num_epochs,
# and see your results plotted below.
def trainPerceptronAlgorithm(X, y, learn_rate = 0.01, num_epochs = 25):

    x_min, x_max = min(X.T[0]), max(X.T[0])
    y_min, y_max = min(X.T[1]), max(X.T[1])

    W = np.array(np.random.rand(2,1))
    b = np.random.rand(1)[0] + x_max

    # These are the solution lines that get plotted below.

    boundary_lines = []
    for i in range(num_epochs):
        # In each epoch, we apply the perceptron step.

        W, b = perceptronStep(X, y, W, b, learn_rate)

        boundary_lines.append((-W[0]/W[1], -b/W[1]))

    return boundary_lines


#trainPerceptronAlgorithm(X, y, learn_rate = 0.01, num_epochs = 25)
plt.scatter(X_pred[0],X_pred[1])
```

[224]: <matplotlib.collections.PathCollection at 0x1ded17ef700>





1.2 Summary

1. Perceptron algorithm used to solve Classification problems
2. Perceptron algorithm as logical operators for deep learning
3. Perceptron trick of how to divide the points according to their classification

1.3 Resources:

Images and problem statement are courtesy of Udacity's Nanodegree program on Intro to ML- TensorFlow. https://www.udacity.com/course/intro-to-machine-learning-with-tensorflow-nanodegree–nd230

[ ]: