In my blog post about “Introduction to neural network”, we saw how Artificial Neural Network (ANN) functions. In this blog post we will see how ANN works actually with coding in python step by step.

Lets first we discuss example on which we will apply ANN. I take a simple complex example of XOR gate

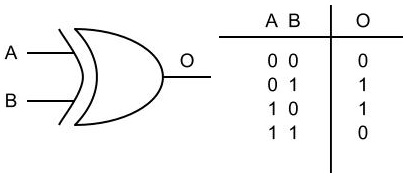
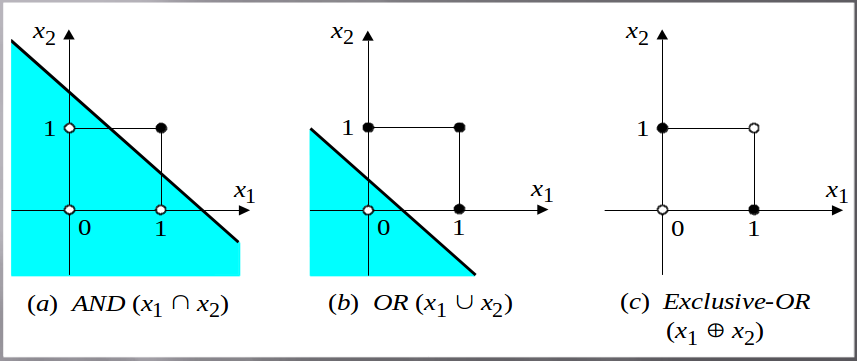
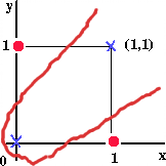


Fig1

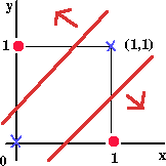
**Why Simple**?? It takes two inputs and produce one output, so when we will code the same, we can very easily understand

**Why complex**?? XOR gate is non linear compare to other logical gates like OR,AND,and NOR gate. Looking at complexity given in figure 3 we can eaily infer that sigle straight line can seperate outputs in AND & OR gate but it cannot differentiate between two outputs in XOR gate.

There is no single function that produces a hyper-plane capable of separating the points of the XOR function. The curve in the image separates the points, but it is not a function.



To separate the points of XOR, you'll have to use at least two lines (or any other shaped functions). This will require two separate perceptrons. Then, you could use a third perceptron to separate the intermediate results on the basis of sign.



Its very clear that one perceptron that cannot solve XOR function, we will design the smallest network that solves this function.

We will write python code to solve XOR gate and we will use following steps throught this tutorial

1)

2)

3)

4)

**[network diagram here]**

Give expanation about network

**Step 1. Train Set**

**XOR = [[0, 1, 1], [1, 1, 0], [1, 0, 1], [0, 0, 0]]**

**pythonically XOR[i][0], XOR[i][1] will give input 1 and 2 and XOR[i][2] will give output for any value of i:[0,1,2,3]**

**Where in each ssub array first two element represents input 1 and 2 and third element represent desired output.**

**Step 2: Initialisation**

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range (0-1):

**In our python solution we did it manually**

w13 = 0.5

w14 = 0.9

w23 = 0.4

w24 = 1.0

w35 = -1.2

w45 = 1.1

t3 = 0.8

t4 = -0.1

t5 = 0.3

**Step 3: Activation**

lets say we have two inputs, 1 and 2 and we have to calculate forward pass for the same at point 3, we do it in following way.

****

here j = 3 and i = 1,2 so our eualtion for forward pass result at 3 would be

**sigmoid (X1 \* W13 + X2 \* W23)**. Sigmoid function can be represented by

over all process by equation it can be represented as :



and python code can be written as:

y3 = 1 / (1 + math.exp(-((XOR[i][0] \* w13) + (XOR[i][1] \* w23))))

y4 = 1 / (1 + math.exp(-(XOR[i][0] \* w14 + XOR[i][1] \* w24)))

y5 = 1 / (1 + math.exp(-(y3 \* w35 + y4 \* w45)))

**Step 3: Calculating error**

After calculating output at 3,4 and 5. y5 is our final output. Errror from the output can be calculate as

error = ActualOutput – y5

For our problem it can be pythonically written as **error = XOR[i][2] – y5**

**Step 4 making updates**

**4.1 For hidden to outer layer**

**the error we got at y5 is total error of the network. To make update we require partial erro at each of output and hidden layer perceptron. calculating partial changes to each is done by**

**for output percepton it is given by δOutput = output \* (1 – output) \* error**

**del5 = y5 \* (1 - y5) \* error**

****

**For the hidden layers it is calculated as : here δbi is error at any perceptron i in layer b and δcj is any peerceptron j in next layer c.**

**For our case δbi = δ3 & δ4 And δcj = δ5.**

**In pythonic way it can be calculated as**

del3 = y3 \* (1 - y3) \* del5 \* w35

del4 = y4 \* (1 - y4) \* del5 \* w45

Bias are represented by **θ,** heare actaully we are not using bias in network still will see how to update it: Δ**θbi = α \* (-1) \* δbi , where θbi and δbi belongs to perceptron of the same node.**

dt5 = alpha \* (-1) \* del5

**4.1 For input to hidden layer**

**Weight and bias chages will be calculated in simillar way :**

dw13 = alpha \* XOR[i][0] \* del3

dw23 = alpha \* XOR[i][1] \* del3

dt3 = alpha \* (-1) \* del3

dw14 = alpha \* XOR[i][0] \* del4

dw24 = alpha \* XOR[i][1] \* del4

dt4 = alpha \* (-1) \* del4

5 Updating all parmaters

Wij(p+1) = Wij(p) +ΔWij(p+1)  
θi(p+1)= Δθi(p) + Δθi(p+1)

Pythonically it is written as

w14 = w14 + dw14

w23 = w23 + dw23

w24 = w24 + dw24

w35 = w35 + dw35

w45 = w45 + dw45

t3 = t3 + dt3

t4 = t4 + dt4

t5 = t5 + dt5

