MACHINE LEARNING

NAME: ADITYA RANJAN
REG. NO.: 16BCE2310

Consider the forward propagation first and do the following needful (i.e print the output) for forward propagation with epoch=1000, 3000, 6000 and hidden layer 2 and then 3 and then 4 respectively.

A.

x1 or x2 or x3 or x4 (Explain each steps with your own understanding)

CODE

```
import numpy as np

X=np.array([[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],[0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],[1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],[1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]])

y=np.array([[0],[1],[1],[1],[1],[1],[1],[1],[1],[1],[1],[1],[1],[1],[1],[1]])



def sigmoid (x):

    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):

    return x * (1 - x)

epoch=5000

lr=0.1

inputlayer_neurons = X.shape[1]

hiddenlayer_neurons = 3

output_neurons = 1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))

bh=np.random.uniform(size=(1,hiddenlayer_neurons))

wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))

bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):

#Forward Propogation
```

```
    hidden_layer_input1=np.dot(X,wh)

    hidden_layer_input=hidden_layer_input1 + bh

    hiddenlayer_activations = sigmoid(hidden_layer_input)

    output_layer_input1=np.dot(hiddenlayer_activations,wout)

    output_layer_input= output_layer_input1+ bout

    output = sigmoid(output_layer_input)

print("Forward Propagation")

print(output)

#Backpropagation

E = y-output

slope_output_layer = derivatives_sigmoid(output)

slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)

d_output = E * slope_output_layer

Error_at_hidden_layer = d_output.dot(wout.T)

d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer

wout += hiddenlayer_activations.T.dot(d_output) *lr

bout += np.sum(d_output, axis=0,keepdims=True) *lr

wh += X.T.dot(d_hiddenlayer) *lr

bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

print("BACK PROPAGATION")

print("Delta error for each hidden neuron")

print(bh)

print("Updated weights")

print(wh)
```

STEPS:

- Initialises the input and output
- Initialize the activation function (sigmoid)
- Inialize the derivative sigmoid
- Iniatilise randomly the weights and the biases for the nodes and paths
- For forward propagation calculate the value at each neuron by multiplying the weigths with neurons.
- Calculate the delta for each neurons .
- Update the new weights for the path following back propagation

OUTPUT

Epoch=1000

```
In [7]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Desktop/cs2
project-master/tests')
Forward Propagation
[[0.8288842 ]
 [0.86501624]
 [0.85988934]
 [0.88029641]
 [0.85000238]
 [0.87563156]
 [0.87172649]
 [0.88564401]
 [0.84931445]
 [0.87528319]
 [0.87125143]
 [0.88537249]
 [0.86429571]
 [0.88226163]
 [0.87913771]
 [0.88884895]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.44773264 0.83657115 0.50830846]]
Updated weights
[[0.56042864 0.49159612 0.16676898]
 [0.40621911 0.63607055 0.19490002]
 [0.95370221 0.73554435 0.41766057]
 [0.98945688 0.8336889  0.86928581]]
```

Epoch=3000

```
In [8]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/User
project-master/tests')
Forward Propagation
[[0.83478082]
 [0.8502181 ]
 [0.85108156]
 [0.86294514]
 [0.84793324]
 [0.86087396]
 [0.86133705]
 [0.87081718]
 [0.87136881]
 [0.87945468]
 [0.87964182]
 [0.88523557]
 [0.87829971]
 [0.88449998]
 [0.8845107 ]
 [0.8887668 ]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.93203264 0.12262275 0.65793837]]
Updated weights
[[0.97028153 0.91080951 0.96252408]
 [0.14375418 0.45263915 0.22829995]
 [0.8391155  0.8500401  0.0357292 ]
 [0.17317343 0.57033434 0.25649231]]
```

Epoch=6000

```
In [10]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Desktop/cs23
final-project-master/tests')
Forward Propagation
[[0.78389532]
 [0.79793799]
 [0.8064268 ]
 [0.81707097]
 [0.80898727]
 [0.81850556]
 [0.82466449]
 [0.8311536 ]
 [0.80520765]
 [0.81553618]
 [0.82193953]
 [0.82907057]
 [0.8235086 ]
 [0.82973832]
 [0.83414453]
 [0.8380875 ]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.10753715 0.95854883 0.32402715]]
Updated weights
[[0.93937772 0.383905   0.5170586 ]
 [0.80313278 0.88647954 0.57376828]
 [0.20040247 0.24121452 0.78112535]
 [0.65865753 0.37734575 0.27257976]]
```

(x1 AND x2) or x3 or x4 (Explain each steps with your own understanding)

```
import numpy as np

X=np.array([[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],[0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],[1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],[1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]])

y=np.array([[0],[1],[1],[1],[0],[1],[1],[0],[1],[1],[1],[1],[1],[1],[1],[1]])
```

```
def sigmoid (x):

    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):

    return x * (1 - x)

epoch=6000

lr=0.1

inputlayer_neurons = X.shape[1]

hiddenlayer_neurons = 3

output_neurons = 1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))

bh=np.random.uniform(size=(1,hiddenlayer_neurons))

wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))

bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):

#Forward Propogation

    hidden_layer_input1=np.dot(X,wh)

    hidden_layer_input=hidden_layer_input1 + bh

    hiddenlayer_activations = sigmoid(hidden_layer_input)

    output_layer_input1=np.dot(hiddenlayer_activations,wout)

    output_layer_input= output_layer_input1+ bout

    output = sigmoid(output_layer_input)

print("Forward Propagation")

print(output)

#Backpropagation

E = y-output

slope_output_layer = derivatives_sigmoid(output)

slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)

d_output = E * slope_output_layer
```

```
Error_at_hidden_layer = d_output.dot(wout.T)

d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer

wout += hiddenlayer_activations.T.dot(d_output) *lr

bout += np.sum(d_output, axis=0,keepdims=True) *lr

wh += X.T.dot(d_hiddenlayer) *lr

bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

print("BACK PROPAGATION")

print("Delta error for each hidden neuron")

print(bh)

print("Updated weights")

print(wh)
```

Epoch 1000

```
In [12]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Desktop/cs231n-final-project-master/tests')
Forward Propagation
[[0.77985494]
 [0.79603072]
 [0.79184218]
 [0.80413067]
 [0.78819807]
 [0.80163351]
 [0.79801304]
 [0.80800643]
 [0.7914347 ]
 [0.80353616]
 [0.8005199 ]
 [0.80940184]
 [0.79783899]
 [0.80761548]
 [0.80505778]
 [0.81213362]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.80097101 0.99804948 0.78868131]]
Updated weights
[[0.3279133  0.41160907 0.45069314]
 [0.53220036 0.27000642 0.24324751]
 [0.64195781 0.5135162  0.1161328 ]
 [0.30437356 0.55877805 0.94265411]]
```

Epoch 3000

```
In [13]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Desktop/cs2
final-project-master/tests')
Forward Propagation
[[0.71280017]
 [0.73223713]
 [0.73069633]
 [0.74391337]
 [0.72958373]
 [0.74330278]
 [0.74222922]
 [0.75100499]
 [0.71701363]
 [0.73514962]
 [0.73381393]
 [0.74593588]
 [0.73267851]
 [0.74528836]
 [0.74438444]
 [0.75233559]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.94463817 0.82203355 0.62604619]]
Updated weights
[[0.08380409 0.48875398 0.66889602]
 [0.51324277 0.52318646 0.35060339]
 [0.57763796 0.24722702 0.44684626]
 [0.62250993 0.36395868 0.82039235]]
```

Epoch 6000

```
In [11]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='
final-project-master/tests')
Forward Propagation
[[0.80326606]
 [0.80831112]
 [0.82738732]
 [0.83097114]
 [0.83108531]
 [0.83491947]
 [0.84696952]
 [0.84963168]
 [0.8319833 ]
 [0.83557553]
 [0.84780297]
 [0.85021271]
 [0.84883066]
 [0.85151895]
 [0.85876105]
 [0.86056378]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.34908947 0.4645458  0.88208877]]
Updated weights
[[0.46686386 0.90673591 0.44974152]
 [0.13611122 0.89826875 0.47377612]
 [0.36780652 0.34113456 0.69237   ]
 [0.50227506 0.01737209 0.08688913]]
```

(x1 AND x2) AND (x3 or x4) (Explain each steps with your own understanding)

```
import numpy as np

X=np.array([[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],[0,1,0,0],[0,1,0,1],[0,1,1,0],[
0,1,1,1],[1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],[1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1
,1,1]])

y=np.array([[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[0],[1],[1],[1]])



def sigmoid (x):

    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):

    return x * (1 - x)

epoch=3000

lr=0.1

inputlayer_neurons = X.shape[1]

hiddenlayer_neurons = 3
```

```
output_neurons = 1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))

bh=np.random.uniform(size=(1,hiddenlayer_neurons))

wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))

bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):

#Forward Propogation

    hidden_layer_input1=np.dot(X,wh)

    hidden_layer_input=hidden_layer_input1 + bh

    hiddenlayer_activations = sigmoid(hidden_layer_input)

    output_layer_input1=np.dot(hiddenlayer_activations,wout)

    output_layer_input= output_layer_input1+ bout

    output = sigmoid(output_layer_input)

print("Forward Propagation")

print(output)

#Backpropagation

E = y-output

slope_output_layer = derivatives_sigmoid(output)

slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)

d_output = E * slope_output_layer

Error_at_hidden_layer = d_output.dot(wout.T)

d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer

wout += hiddenlayer_activations.T.dot(d_output) *lr

bout += np.sum(d_output, axis=0,keepdims=True) *lr

wh += X.T.dot(d_hiddenlayer) *lr

bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

print("BACK PROPAGATION")

print("Delta error for each hidden neuron")

print(bh)
```

```
print("Updated weights")

print(wh)
```

Epoch=1000

```
In [15]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Desktop/cs231
final-project-master/tests')
Forward Propagation
[[0.80908573]
 [0.83434553]
 [0.824872  ]
 [0.8451687 ]
 [0.83774107]
 [0.85553248]
 [0.84763925]
 [0.86173101]
 [0.82589967]
 [0.84522555]
 [0.8392599 ]
 [0.85419929]
 [0.84982202]
 [0.86276251]
 [0.85781208]
 [0.86767926]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.54718707 0.5589175  0.46321815]]
Updated weights
[[0.52536473 0.17511079 0.07863941]
 [0.30496527 0.92235972 0.8238178 ]
 [0.17715265 0.21420335 0.84056844]
 [0.80231175 0.08653784 0.68317857]]
```

Epoch=3000

```
In [14]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Desktop/cs231n-
final-project-master/tests')
Forward Propagation
[[0.82633173]
 [0.84977091]
 [0.85062043]
 [0.8664259 ]
 [0.84579614]
 [0.86295622]
 [0.86445111]
 [0.87537516]
 [0.85003538]
 [0.8672257 ]
 [0.86642037]
 [0.87759554]
 [0.86464756]
 [0.87627643]
 [0.87651287]
 [0.88365187]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.59920135 0.11352715 0.4089791 ]]
Updated weights
[[0.08093053 0.67958338 0.64531887]
 [0.92342661 0.46448666 0.28016544]
 [0.42402252 0.04054723 0.95310555]
 [0.94348164 0.43043223 0.48023263]]
```

Epoch=6000

```
In [16]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/De
final-project-master/tests')
Forward Propagation
[[0.75237039]
 [0.7732997 ]
 [0.77029976]
 [0.78740555]
 [0.79698673]
 [0.81017133]
 [0.80887569]
 [0.81897822]
 [0.78127431]
 [0.7955705 ]
 [0.79357584]
 [0.80485025]
 [0.81347092]
 [0.82208872]
 [0.82127418]
 [0.82778347]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.13649143 0.49941711 0.67928293]]
Updated weights
[[-8.65940257e-04  5.65930047e-01  8.59545380e-01]
 [ 7.72406300e-01  3.65994796e-01  8.02881604e-01]
 [ 1.04383245e-01  9.92058866e-01  1.86003093e-01]
 [ 1.72643094e-01  4.92124155e-01  3.53255063e-01]]
```

(x1 or x2) AND (x3 or x4) (Explain each steps with your own understanding)

```
import numpy as np

X=np.array([[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],[0,1,0,0],[0,1,0,1],[0,1,1,0],[
0,1,1,1],[1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],[1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1
,1,1]])

y=np.array([[0],[0],[0],[0],[0],[0],[1],[1],[0],[1],[1],[1],[0],[1],[1],[1]])



def sigmoid (x):

    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):

    return x * (1 - x)

epoch=6000

lr=0.1

inputlayer_neurons = X.shape[1]

hiddenlayer_neurons = 3

output_neurons = 1
```

```
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))

bh=np.random.uniform(size=(1,hiddenlayer_neurons))

wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))

bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):

#Forward Propogation

    hidden_layer_input1=np.dot(X,wh)

    hidden_layer_input=hidden_layer_input1 + bh

    hiddenlayer_activations = sigmoid(hidden_layer_input)

    output_layer_input1=np.dot(hiddenlayer_activations,wout)

    output_layer_input= output_layer_input1+ bout

    output = sigmoid(output_layer_input)

print("Forward Propagation")

print(output)

#Backpropagation

E = y-output

slope_output_layer = derivatives_sigmoid(output)

slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)

d_output = E * slope_output_layer

Error_at_hidden_layer = d_output.dot(wout.T)

d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer

wout += hiddenlayer_activations.T.dot(d_output) *lr

bout += np.sum(d_output, axis=0,keepdims=True) *lr

wh += X.T.dot(d_hiddenlayer) *lr

bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

print("BACK PROPAGATION")

print("Delta error for each hidden neuron")

print(bh)

print("Updated weights")
```

```
print(wh)
```

EPOCH=1000

```
In [18]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Des
final-project-master/tests')
Forward Propagation
[[0.80551757]
 [0.82174202]
 [0.81889958]
 [0.83136929]
 [0.82251257]
 [0.83602169]
 [0.8341687 ]
 [0.8442369 ]
 [0.83432602]
 [0.84362962]
 [0.84185163]
 [0.84858865]
 [0.84510698]
 [0.85242317]
 [0.85149128]
 [0.85661206]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.19625865 0.79576815 0.62249468]]
Updated weights
[[0.69469016 0.77044114 0.83467273]
 [0.91368632 0.04451154 0.22538083]
 [0.02204734 0.61827136 0.41894346]
 [0.13330116 0.31968643 0.7114491 ]]
```

EPOCH=3000

```
In [19]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Desktop/c
final-project-master/tests')
Forward Propagation
[[0.81967723]
 [0.83967112]
 [0.83595424]
 [0.85341661]
 [0.85531786]
 [0.86700659]
 [0.86696913]
 [0.87663632]
 [0.85921325]
 [0.87045176]
 [0.86934246]
 [0.87865302]
 [0.87841181]
 [0.88430907]
 [0.88527276]
 [0.88988177]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.83235687 0.14297972 0.22857608]]
Updated weights
[[0.86266897 0.90068968 0.66134379]
 [0.90327986 0.92543723 0.37886357]
 [0.0056325  0.12374178 0.70084298]
 [0.38530723 0.5055613  0.21072795]]
```

EPOCH=6000

```
In [17]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/H
final-project-master/tests')
Forward Propagation
[[0.8641064 ]
 [0.88389012]
 [0.89540869]
 [0.90549861]
 [0.8751117 ]
 [0.8906654 ]
 [0.9009837 ]
 [0.90852849]
 [0.89473803]
 [0.90520062]
 [0.91203376]
 [0.91691596]
 [0.90022105]
 [0.90816398]
 [0.91444896]
 [0.91817482]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.52878898 0.87209872 0.66487841]]
Updated weights
[[0.92182566 0.40527301 0.52763095]
 [0.50946562 0.00506227 0.01482086]
 [0.87933719 0.25069539 0.96473519]
 [0.72842997 0.07625498 0.30214261]]
```

2. For both the questions apply back propagation consider learning rate as 0.2, 0.3,0,3 and print the output.(Explain each steps with your own understanding—Write 5 to 6 sentences)

Learing rate =0.1

```
In [20]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Deskto
final-project-master/tests')
Forward Propagation
[[0.84280342]
 [0.86106847]
 [0.86326899]
 [0.87787276]
 [0.87654798]
 [0.88800736]
 [0.88879495]
 [0.89697229]
 [0.86492674]
 [0.8784512 ]
 [0.88120694]
 [0.89125297]
 [0.8893268 ]
 [0.89692074]
 [0.89832225]
 [0.90345972]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.45247513 0.01659508 0.00282329]]
Updated weights
[[0.05975032 0.99615028 0.12632861]
 [0.27837267 0.89419723 0.92884096]
 [0.14166734 0.28748751 0.7757118 ]
 [0.43527873 0.45788064 0.35806226]]
```

Learning rate =0.2

```
In [21]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/
final-project-master/tests')
Forward Propagation
[[0.88464902]
 [0.88992969]
 [0.90864578]
 [0.91192057]
 [0.8932171 ]
 [0.89759192]
 [0.91318315]
 [0.91590342]
 [0.90700732]
 [0.91060897]
 [0.92122937]
 [0.92335228]
 [0.91235193]
 [0.91531681]
 [0.92402104]
 [0.92579115]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.9161282  0.04950314 0.95002794]]
Updated weights
[[0.00850622 0.98593939 0.64214771]
 [0.17869364 0.05527162 0.64821351]
 [0.33261385 0.79219315 0.92612199]
 [0.17311964 0.08806602 0.1405061 ]]
```

Learning rate=0.3

```
In [22]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP/Desktop/c
final-project-master/tests')
Forward Propagation
[[0.80090156]
 [0.8266905 ]
 [0.83421917]
 [0.85229457]
 [0.82495217]
 [0.8459201 ]
 [0.84625969]
 [0.86140895]
 [0.82720307]
 [0.84393075]
 [0.85298191]
 [0.86370053]
 [0.8466936 ]
 [0.85964807]
 [0.86227479]
 [0.87087486]]
BACK PROPAGATION
Delta error for each hidden neuron
[[0.35412791 0.52773849 0.57358331]]
Updated weights
[[0.81100045 0.28055465 0.16441697]
 [0.05677173 0.00525468 0.92080972]
 [0.33253799 0.26212252 0.98375281]
 [0.8436079  0.00950047 0.19200197]]
```

B. Implement two input OR, two input AND functions (Explain each steps with your own understanding-write 3 to 4 sentences of your understanding)

OR function

Code

```
def perceptron(x, w, b):

	v = np.dot(w, x) + b
	y = unit_step(v)
	return y

def unit_step(v):
	if v >= 0:
		return 1
	else:
		return 0
# OR function
def OR_percep(x):
   w = np.array([1, 1])
   b = -0.5
   return perceptron(x, w, b)

or1 = np.array([1, 1])
or2 = np.array([1, 0])
or3 = np.array([0, 1])
or4 = np.array([0, 0])

print("OR({}, {}) = {}".format(1, 1, OR_percep(or1)))
print("OR({}, {}) = {}".format(1, 0, OR_percep(or2)))
print("OR({}, {}) = {}".format(0, 1, OR_percep(or3)))
```

print("OR({}, {}) = {}".format(0, 0, OR_percep(or4)))

```
In [28]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir=
final-project-master/tests')
OR(1, 1) = 1
OR(1, 0) = 1
OR(0, 1) = 1
OR(0, 0) = 0
```

AND function

def AND_percep(x):

w = np.array([1, 1])

b = -1.5

return perceptron(x, w, b)

and1 = np.array([1, 1])

and2 = np.array([1, 0])

and3 = np.array([0, 1])

and4 = np.array([0, 0])

print("AND({}, {}) = {}".format(1, 1, AND_percep(and1)))

print("AND({}, {}) = {}".format(1, 0, AND_percep(and2)))

print("AND({}, {}) = {}".format(0, 1, AND_percep(and3)))

print("AND({}, {}) = {}".format(0, 0, AND_percep(and4)))

```
In [29]: runfile('C:/Users/HP/Desktop/cs231n-final-project-master/tests/untitled0.py', wdir='C:/Users/HP
final-project-master/tests')
AND(1, 1) = 1
AND(1, 0) = 0
AND(0, 1) = 0
AND(0, 0) = 0
```

3. Classify the following data(as train) with the predictor as pass (1=pass and 0= fail) using R package of **nntool (or neural net)** and also try with python programming to find the **probable class with your own choice of test data(you decide) and find the predicted accuracy. Using R/python you can also find the confusion matrix. Can you also find FP,TP,FN and TN.** .(Explain each steps with your own understanding—Write 10 to 12 sentences)

Student Name	Physics	Chemistry	Maths	Pass
Akash	90	80	20	0
Akram	70	60	10	1
Rahul	60	50	50	0
Sakshi	90	90	90	1
Dinesh	65	67	89	1
Rakesh	20	20	30	0
Yogesh	50	30	20	0
Raja	76	67	58	1
Shivani	80	70	96	1

CODE

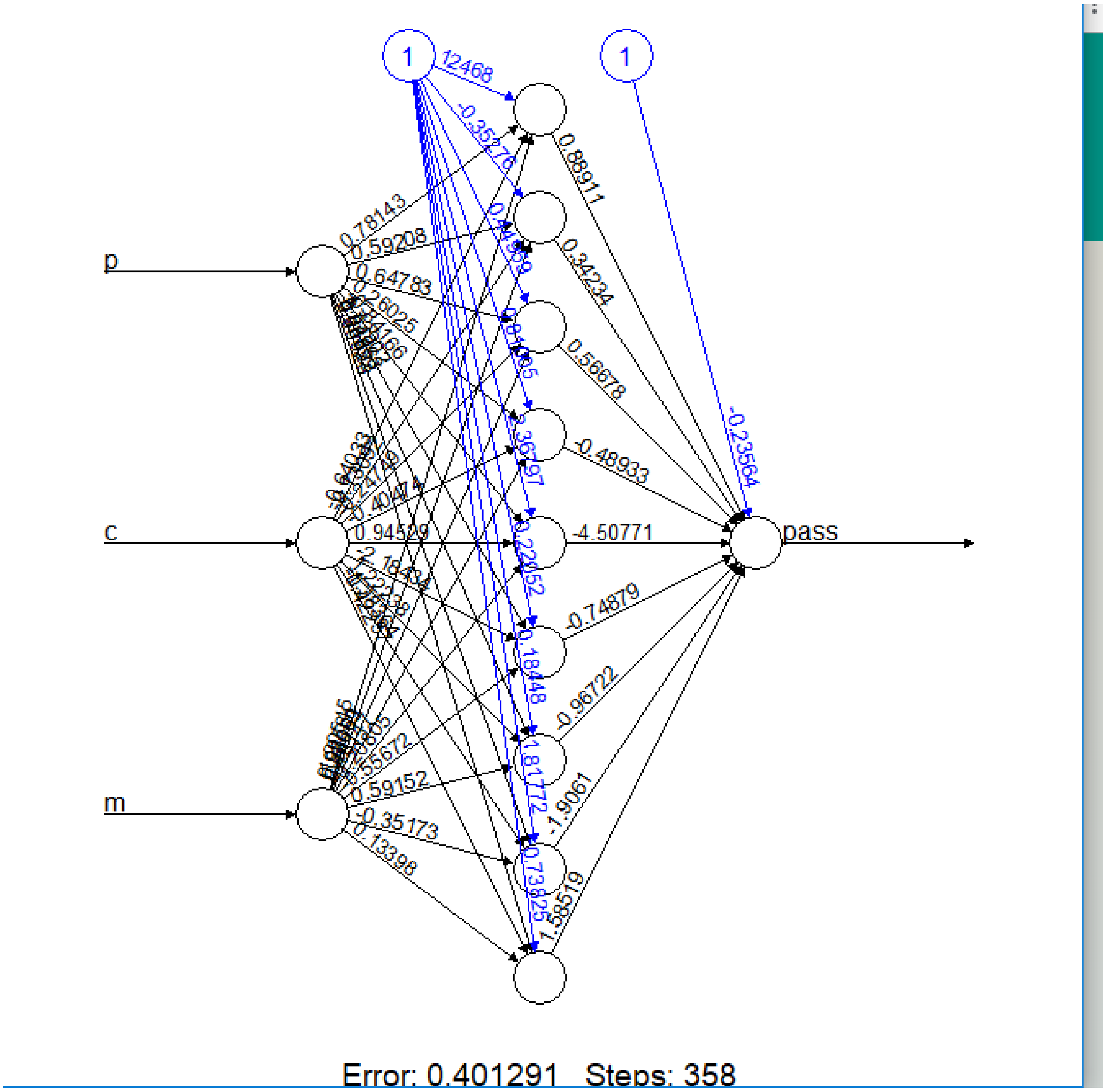
```
> library("neuralnet")
> library("nnet")
> name<-c("Akash","Akram","Rahul","Sakshi","Dinesh","Rakesh","Yogesh","Raj
a","Shivani")
> p<-c(90,70,60,90,65,20,50,76,80)
> c<-c(80,60,50,90,67,20,30,67,70)
> m<-c(20,10,50,90,89,30,20,58,96)
> pass<-c(0,1,0,1,1,0,0,1,1)
> data=data.frame(name,p,c,m,pass)
> model=neuralnet(formula=pass~p+c+m,data=data,hidden=9,threshold = 0.01)
> print(model)
$`call`
neuralnet(formula = pass ~ p + c + m, data = data, hidden = 9,
    threshold = 0.01)

$response
  pass
1    0
2    1
3    0
4    1
5    1
6    0
7    0
8    1
9    1

$covariate
```

```
       p  c  m
 [1,] 90 80 20
 [2,] 70 60 10
 [3,] 60 50 50
 [4,] 90 90 90
 [5,] 65 67 89
 [6,] 20 20 30
 [7,] 50 30 20
 [8,] 76 67 58
 [9,] 80 70 96

$model.list
$model.list$`response`
[1] "pass"

$model.list$variables
[1] "p" "c" "m"


$err.fct
function (x, y)
{
    1/2 * (y - x)^2
}
<bytecode: 0x000001718976deb8>
<environment: 0x000001718ee2a768>
attr(,"type")
[1] "sse"

$act.fct
function (x)
{
    1/(1 + exp(-x))
}
<bytecode: 0x0000017189769128>
<environment: 0x000001718ee2a260>
attr(,"type")
[1] "logistic"

$linear.output
[1] TRUE

$data
     name  p  c  m pass
1   Akash 90 80 20    0
2   Akram 70 60 10    1
3   Rahul 60 50 50    0
4  Sakshi 90 90 90    1
5  Dinesh 65 67 89    1
6  Rakesh 20 20 30    0
7  Yogesh 50 30 20    0
8    Raja 76 67 58    1
9 Shivani 80 70 96    1

$exclude
NULL

$net.result
```

```
$net.result[[1]]
              [,1]
 [1,] 8.795087e-04
 [2,] 9.995289e-01
 [3,] 8.006009e-01
 [4,] 7.958069e-01
 [5,] 7.980489e-01
 [6,] 7.269991e-03
 [7,] 1.908845e-05
 [8,] 8.001805e-01
 [9,] 8.021131e-01


$weights
$weights[[1]]
$weights[[1]][[1]]
           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
[,7]      [,8]       [,9]
[1,]  0.1246791 -0.3527575  0.4495908  0.8100515  2.3679702  0.2205216 0.1
8447678  1.817721 -0.7382474
[2,]  0.7814306  0.5920756  0.6478345  0.2602511 -0.8416627  1.7390168 0.0
1963424  0.868221  0.3692339
[3,] -0.6403264 -0.1309166 -0.2474873 -0.4047432  0.9452880 -2.1843371 1.2
2237794 -1.383644 -0.4225695
[4,] -1.2054452  0.9956058  0.9866878  0.4814666 -0.2080541 -0.5567214 0.5
9152305 -0.351729  0.1339850

$weights[[1]][[2]]
            [,1]
 [1,] -0.2356381
 [2,]  0.8891115
 [3,]  0.3423375
 [4,]  0.5667776
 [5,] -0.4893252
 [6,] -4.5077127
 [7,] -0.7487865
 [8,] -0.9672183
 [9,] -1.9061027
[10,]  1.5851886



$generalized.weights
$generalized.weights[[1]]
               [,1]          [,2]         [,3]
 [1,]  5.734289e+02 -6.350919e+02 1.086179e+02
 [2,]  1.091862e+03 -1.197811e+03 1.604903e+02
 [3,]  3.685300e-03 -4.211176e-03 1.302246e-03
 [4,]  2.590339e-02 -2.920547e-02 7.015266e-03
 [5,]  1.646188e-02 -1.856858e-02 4.502712e-03
 [6,]  7.113732e+01 -8.004923e+01 1.841727e+01
 [7,] -2.350035e+03  4.192794e+03 3.124150e+02
 [8,]  4.913171e-03 -5.606377e-03 1.693382e-03
 [9,]  2.101742e-05 -2.404977e-05 7.607115e-06


$startweights
$startweights[[1]]
```

```
$startweights[[1]][[1]]
           [,1]       [,2]       [,3]       [,4]        [,5]       [,6]
[,7]       [,8]        [,9]
[1,]  0.1529922 -0.7527575  0.1495908  0.9781608  0.08415396  0.3526737 0.
18447678  0.6504415 -0.17483867
[2,]  0.8421614  0.1920756  0.3478345  0.4292834 -0.82720115  1.8711568 0.
01963424 -0.2990580  0.27603387
[3,] -0.5788912 -0.5309166 -0.5474873 -0.2357345  0.95376208 -2.0337267 1.
22237794 -2.5509228 -0.51755928
[4,] -1.1411186  0.5956058  0.6866878  0.6440345 -0.14630616 -0.3469351 0.
59152305 -1.5190081  0.02782938

$startweights[[1]][[2]]
             [,1]
 [1,] -0.49224519
 [2,] -0.19055676
 [3,]  0.08573037
 [4,]  0.31017047
 [5,] -0.96102271
 [6,] -0.36867874
 [7,] -0.80575581
 [8,] -1.22382539
 [9,] -0.73882361
[10,]  1.24931911



$result.matrix
                                [,1]
error                     0.40129091
reached.threshold         0.00974118
steps                   358.00000000
Intercept.to.1layhid1     0.12467909
p.to.1layhid1             0.78143056
c.to.1layhid1            -0.64032640
m.to.1layhid1            -1.20544521
Intercept.to.1layhid2    -0.35275750
p.to.1layhid2             0.59207559
c.to.1layhid2            -0.13091656
m.to.1layhid2             0.99560578
Intercept.to.1layhid3     0.44959080
p.to.1layhid3             0.64783454
c.to.1layhid3            -0.24748731
m.to.1layhid3             0.98668782
Intercept.to.1layhid4     0.81005148
p.to.1layhid4             0.26025110
c.to.1layhid4            -0.40474316
m.to.1layhid4             0.48146657
Intercept.to.1layhid5     2.36797018
p.to.1layhid5            -0.84166266
c.to.1layhid5             0.94528803
m.to.1layhid5            -0.20805410
Intercept.to.1layhid6     0.22052159
p.to.1layhid6             1.73901678
c.to.1layhid6            -2.18433706
m.to.1layhid6            -0.55672140
Intercept.to.1layhid7     0.18447678
p.to.1layhid7             0.01963424
```

```
c.to.1layhid7          1.22237794
m.to.1layhid7          0.59152305
Intercept.to.1layhid8  1.81772053
p.to.1layhid8          0.86822104
c.to.1layhid8         -1.38364369
m.to.1layhid8         -0.35172903
Intercept.to.1layhid9 -0.73824738
p.to.1layhid9          0.36923390
c.to.1layhid9         -0.42256946
m.to.1layhid9          0.13398496
Intercept.to.pass     -0.23563808
1layhid1.to.pass       0.88911151
1layhid2.to.pass       0.34233748
1layhid3.to.pass       0.56677758
1layhid4.to.pass      -0.48932516
1layhid5.to.pass      -4.50771272
1layhid6.to.pass      -0.74878653
1layhid7.to.pass      -0.96721828
1layhid8.to.pass      -1.90610268
1layhid9.to.pass       1.58518857

attr(,"class")
[1] "nn"
> plot(model)
> name<-c("Aman","Preeti","Ratan","Soumya")
> p<-c(90,20,45,69)
> c<-c(90,50,88,56)
> m<-c(40,30,70,40)
> pass<-c(0,1,1,1)
> testset<-data.frame(name,p,c,m,pass)
> temp_test <- subset(testset, select = c("p","c", "m"))
> head(temp_test)
   p  c  m
1 90 90 40
2 20 50 30
3 45 88 70
4 69 56 40
> model.results <- compute(model, temp_test)
> results <- data.frame(actual = testset$pass, prediction = model.results$
net.result)
> roundedresults<-sapply(results,round,digits=0)
> roundedresultsdf=data.frame(roundedresults)
> attach(roundedresultsdf)
> table(actual,prediction)
      prediction
actual -5 -4 1
     0  0  1 0
     1  2  0 1
```



Python implementation

```
StudentName=["Akash","Akram","Rahul","Sakshi","Dinesh","Rakesh","Yogesh","Raja","Shivani"]

Physics=[90,70,60,90,65,20,50,76,80]

Chemistry=[80,60,50,90,67,20,30,67,70]

Maths=[20,10,50,90,89,30,20,58,96]

Pass=[0,1,0,1,1,0,0,1,1]

table=np.array([["Akash",90,80,20,0],["Akram",70,60,10,1],["Rahul",60,50,50,0],["Sakshi",90,90,90,1],["Dinesh",65,67,89,1],["Rakesh",20,
```

```
20,30,0],["Yogesh",50,30,20,0],["Raja",76,67,58,1],["Shivani",80,70,
96,1]])
```

```
Out[41]: array([['Akash', '90', '80', '20', '0'],
                ['Akram', '70', '60', '10', '1'],
                ['Rahul', '60', '50', '50', '0'],
                ['Sakshi', '90', '90', '90', '1'],
                ['Dinesh', '65', '67', '89', '1'],
                ['Rakesh', '20', '20', '30', '0'],
                ['Yogesh', '50', '30', '20', '0'],
                ['Raja', '76', '67', '58', '1'],
                ['Shivani', '80', '70', '96', '1']], dtype='<U7')
```

```
from sklearn.model_selection import train_test_split

X=table[:, [0,1,2,3]]

y_=table[:,[4]]

X_train, X_test, y_train, y_test = train_test_split(X, y_,
test_size=0.33, random_state=0)

print('Training set length: {}.\nTest set length:
{}'.format(X_train.shape[0], X_test.shape[0]))

from sklearn.preprocessing import StandardScaler


#from each value subtract its average and divide by the standard
deviation

sc = StandardScaler()

sc.fit(X_train)


X_train_std = sc.transform(X_train)

X_test_std = sc.transform(X_test)


from sklearn.svm import SVC

from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns

svm = SVC(kernel='linear', probability=True, random_state=0)

svm.fit(X_train_std, y_train)


from sklearn.metrics import confusion_matrix

import itertools
```

```
def plot_confusion_matrix(y, y_predict, classes,
                          title='Conf Matrix for Student Marks',
                          cmap=plt.cm.YlOrRd):
    sns.set(font_scale=2.5, rc={'figure.figsize':(12, 10)})
    cm = confusion_matrix(y, y_predict)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], 'd'),
horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')

print('The accuracy on training data is
{:.1f}%'.format(svm.score(X_train_std, y_train) * 100))
print('The accuracy on test data is
{:.1f}%'.format(svm.score(X_test_std, y_test) * 100))

y_svm_train_predict = svm.predict(X_train_std)
print(y_svm_train_predict)

y_svm_train_predict_proba = svm.predict_proba(X_train_std)
plot_confusion_matrix(y_train, y_svm_train_predict, y_, 'SVM MODEL')
```

```
C:\Users\Lenovo\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
The accuracy on training data is 100.0%
The accuracy on test data is 66.7%
['1' '1' '0' '1' '0' '0']
```


```
The accuracy on training data is 100.0%
The accuracy on test data is 66.7%
['1' '1' '0' '1' '0' '0']
```