In [1]:

```
import numpy as np

# Sigmoid activation function
def activate(x, deriv = False):
    if(deriv == True):
        return x * (1 - x)

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

x_input = np.array([[0,1,0,1], # 0
                    [1,1,0,0],  # 1
                    [0,0,1,1],  # 0
                    [1,1,1,1],  # 1
                    [0,1,0,1]]) # 0

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

np.random.seed(5)
# Making 2 layers
# Synapse
syn0 = 2 * np.random.random((4,5)) - 1
syn1 = 2 * np.random.random((5,3)) - .78
syn2 = 2 * np.random.random((3,1)) - .5

# Looping and Layering
for j in range(70000):

    # Layers

    # Forward Propogate:
    layer0 = x_input
    layer1 = activate(np.dot(layer0, syn0))
    layer2 = activate(np.dot(layer1, syn1))
    layer3 = activate(np.dot(layer2, syn2))

    # Error 3
    lay3_e = y_output - layer3

    if(j % 10000) == 0:
        print"Error:" + str(np.mean(np.abs(lay3_e)))

    # 3 Gradient
    l3_grad = lay3_e * activate(layer3, True)

    # Error 2
    lay2_error = l3_grad.dot(syn2.T)

    # Gradient Layer 2
    lay2_grad = lay2_error * activate(layer2, True)
```

```
    # Error 1
    lay1_error = lay2_grad.dot(syn1.T)

    # Gradient Layer 1
    lay1_grad = lay1_error * activate(layer1, True)

    # Update Weights:
    syn2 += layer2.T.dot(l3_grad)
    syn1 += layer1.T.dot(lay2_grad)
    syn0 += layer0.T.dot(lay1_grad)

print "After Training"
print layer3
```

```
Error:0.513526208163
Error:0.00432529297149
Error:0.00303050777889
Error:0.00246626311188
Error:0.00213230647902
Error:0.00190533731883
Error:0.0017382432143
After Training
[[ 0.00132699]
 [ 0.99838605]
 [ 0.00147415]
 [ 0.99769891]
 [ 0.00132699]]
```

In [ ]: