Implement Backpropogation Neural Network

In [1]:
```
import numpy as np
import time
```

In [2]:
```
def initializeWeights(layers):
  weights = [np.random.randn(o, i+1) for i, o in zip(layers[:-1], layers[1:])]
  return weights

def addBiasTerms(X):
    X = np.array(X)
    if X.ndim==1:
      X = np.reshape(X, (1, len(X)))
    X = np.insert(X, 0, 1, axis=1)

    return X
```

In [3]:
```
def sigmoid(a):
  return 1/(1 + np.exp(-a))
def forwardProp(X, weights):
  outputs = []
  inputs = X

  for w in weights:
    inputs = addBiasTerms(inputs)
    outputs.append(sigmoid(np.dot(inputs, w.T)))
    inputs = outputs[-1]

  return outputs
```

In [4]:
```
def nnCost(weights, X, Y):
  yPred = forwardProp(X, weights)[-1]
  J = 0.5*np.sum((yPred-Y)**2)/len(Y)
  return J
```

Initialize network

In [5]:
```
layers = [2, 2, 1]
weights = initializeWeights(layers)
```

In [6]:
```
X = np.array([[0,0], [0,1], [1,0], [1,1]])
Y = np.array([[0], [0], [0], [1]])
```

In [7]:
```
J = nnCost(weights, X, Y)
print(J)
```

```
0.25523604544501327
```

In [8]:
```
layers = [2, 2, 1]
weights = initializeWeights(layers)

print("weights:")
for i in range(len(weights)):
    print(i+1); print(weights[i].shape); print(weights[i])
```

```
weights:
1
(2, 3)
[[ 2.36751796  0.34798916 -1.17156387]
 [-0.06244016  0.71833303  0.2173351 ]]
2
(1, 3)
[[0.64818689 1.2021787  0.89234995]]
```

In [9]:
```
X = np.array([[0,0], [0,1], [1,0], [1,1]])
Y = np.array([[0], [0], [0], [1]])
```

Forward Propogation

In [10]:
```
outputs = forwardProp(X, weights)
print("outputs"); print(outputs)
```

```
outputs
[array([[0.91431661, 0.48439503],
       [0.76780426, 0.5386465 ],
       [0.93793551, 0.65833718],
       [0.82403723, 0.70541693]]), array([[0.89840336],
       [0.88613805],
       [0.91397931],
       [0.90621789]])]
```

In [11]:
```
yPred = outputs[-1]
print(yPred.shape); print(yPred)
```

```
(4, 1)
[[0.89840336]
 [0.88613805]
 [0.91397931]
 [0.90621789]]
```

In [12]:
```
error = yPred - Y
print(error.shape); print(error)
```

```
(4, 1)
[[ 0.89840336]
 [ 0.88613805]
 [ 0.91397931]
 [-0.09378211]]
```

In [13]:
```
delta = np.multiply(np.multiply(error, yPred), 1-yPred)
print(delta.shape); print(delta)
```

```
(4, 1)
[[ 0.08200155]
 [ 0.08940903]
 [ 0.07185808]
 [-0.00797026]]
```

In [14]:
```
xL = addBiasTerms(outputs[-2])
print(xL.shape); print(xL)
```

```
(4, 3)
[[1.         0.91431661 0.48439503]
 [1.         0.76780426 0.5386465 ]
 [1.         0.93793551 0.65833718]
 [1.         0.82403723 0.70541693]]
```

In [15]:
```
deltaW = -np.dot(delta.T, xL)/len(Y)
print(deltaW.shape); print(deltaW)
```

```
(1, 3)
[[-0.0588246  -0.05111362 -0.03239137]]
```

In [16]:
```
newWeights = [np.array(w) for w in weights]
newWeights[-1] += deltaW

print("old weights:")
for i in range(len(weights)):
    print(i+1); print(weights[i].shape); print(weights[i])

print("new weights:")
for i in range(len(newWeights)):
    print(i+1); print(newWeights[i].shape); print(newWeights[i])

print("old cost:"); print(nnCost(weights, X, Y))
print("new cost:"); print(nnCost(newWeights, X, Y))
```

```
old weights:
1
(2, 3)
[[ 2.36751796  0.34798916 -1.17156387]
 [-0.06244016  0.71833303  0.2173351 ]]
2
(1, 3)
[[0.64818689 1.2021787  0.89234995]]
new weights:
1
(2, 3)
[[ 2.36751796  0.34798916 -1.17156387]
 [-0.06244016  0.71833303  0.2173351 ]]
2
(1, 3)
[[0.58936229 1.15106508 0.85995858]]
old cost:
0.30456531460044817
new cost:
0.29715513606516364
```

Backward Propagate

In [17]:
```
def backProp(weights, X, Y):
    outputs = forwardProp(X, weights)
    bpError = outputs[-1] - Y

    for l, w in enumerate(reversed(weights)):
        yPred = outputs[-l-1]
        delta = np.multiply(np.multiply(bpError, yPred), 1-yPred)
        if l==len(weights)-1:
            xL = addBiasTerms(X)
        else:
            xL = addBiasTerms(outputs[-l-2])
        deltaW = -np.dot(delta.T, xL)/len(Y)
        bpError = np.dot(delta, w)


        bpError = bpError[:,1:]
        w += deltaW
```

In [18]:

```
oldWeights = [np.array(w) for w in weights]
print("old weights:")
for i in range(len(oldWeights)):
    print(i+1); print(oldWeights[i].shape); print(oldWeights[i])

print("old cost:"); print(nnCost(oldWeights, X, Y))
```

```
old weights:
1
(2, 3)
[[ 2.36751796  0.34798916 -1.17156387]
 [-0.06244016  0.71833303  0.2173351 ]]
2
(1, 3)
[[0.64818689 1.2021787  0.89234995]]
old cost:
0.30456531460044817
```

In [19]:

```
def evaluate(weights, X, Y):
    yPreds = forwardProp(X, weights)[-1]
    yes = sum( int( ( np.argmax(yPreds[i]) == np.argmax(Y[i]) ) and
                ( (yPreds[i][np.argmax(yPreds[i])]>0.5) == (Y[i][np.argmax(Y[i])]>0.5) ) )
          for i in range(len(Y)) )
    print(str(yes)+" out of "+str(len(Y))+" : "+str(float(yes/len(Y))))
```

In [20]:
```
weights = [np.array(w) for w in oldWeights]

print("old cost: "); print(nnCost(weights, X, Y))
print("old accuracy: "); print(evaluate(weights, X, Y))
for i in range(1000):
  backProp(weights, X, Y)
  if i%50==0:
    time.sleep(1)
    print(i)
    print("new cost:"); print(nnCost(weights, X, Y))
    print("new accuracy: "); evaluate(weights, X, Y)
    print(forwardProp(X, weights)[-1])
```

```
old cost:
0.30456531460044817
old accuracy:
1 out of 4 : 0.25
None
0
new cost:
0.29687120530436817
new accuracy:
1 out of 4 : 0.25
[[0.88644473]
 [0.87326854]
 [0.90303866]
 [0.8946026 ]]
50
new cost:
0.0918103502238895
new accuracy:
3 out of 4 : 0.75
```

In [ ]: