In [18]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold

def getData(filePath):
    data = np.genfromtxt(filePath, delimiter=',')
    x, y = np.array(data[:,0:-1], dtype=float), np.array(data[:,-1],dtype=int)
    y = y.reshape(1,len(y)).T
    return x,y

def splitInputOutput(data):
    x, y = np.array(data[:,0:-1], dtype=float), np.array(data[:,-1],dtype=int)
    y = y.reshape(1,len(y)).T
    return x,y

def sigmoid(x):
    return 1/(1+np.exp(-x))

def make_sigmoid_prime(x):
    return x*(1-x)

def trainNeuralNetArrays(weights, eta, epochs, activator, activator_prime):
    n = len(weights)
    layers = [None] * n
    deltas = [None] * n
    for j in range(epochs):
        for i in range(n):
            if i == 0: # Push our input into the first weight
                layers[i] = activator(np.dot(X,weights[i]))
            else: # Push previous layer into current later using weights_i
                layers[i] = activator(np.dot(layers[i-1],weights[i]))
        for i in reversed(range(n)):
            if i == n-1: # The delta closest to our output is (y-t) -- or "y - layers[i]" in this case
                deltas[i] = (y - layers[i])*activator_prime(np.dot(layers[i-1],weights[i]))
            elif i > 0: # While were not the first or last delta use the d_i+1, weights_i+1, layers_i-1, weights_i
                deltas[i] = deltas[i+1].dot(weights[i+1].T)*activator_prime(np.dot(layers[i-1],weights[i]))
            else: # update d_0 using X instead of one of the layers
                deltas[i] = deltas[i+1].dot(weights[i+1].T)*activator_prime(np.dot(X,weights[i]))
        for i in reversed(range(n)): # Back propagation time, start at the last weight and move forward
            if i != 0:  # testing !=0 so show we backpropogate back to front {else statement}
                weights[i] += eta*layers[i-1].T.dot(deltas[i])
            else:
                weights[i] += eta*X.T.dot(deltas[i])

def transformTestData(x, syn0, syn1, func):
    layer1_transform = func(np.dot(x,syn0))
    return func(np.dot(layer1_transform,syn1))
```

```
Test our neural network trainer against a simple dataset: X will contain binary pairs and Y will be the XOR result of rows
in X.  We will initialize a random dataset of 0's and 1's with length 20 for A and B (X) and XOR the result for our y.
```

In [19]:
```
np.random.seed(42)
a = np.random.randint(0,2,20) # gen a bunch of random numbers either 0 or 1
b = np.random.randint(0,2,20) # do it again
c = a ^ b #set C to be the XOR result of A XOR B
X, y = splitInputOutput(np.asmatrix([a,b,c]).T)

kf = KFold(n_splits=5,random_state=None, shuffle=False) #our dataset was already random, no need to shuffle

lowest_mse = 1e8 #arbitrary high value
lowest_syn0 = []
lowest_syn1 = []

lowest_synapses = None

eta = 0.1
epochs = 10000

print("Performing K-fold cross validation, splits = 5")
# do our cross validation with training data
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    np.random.seed(seed=42)
    syn0 = 2*np.random.random((X_train.shape[1],X_train.shape[0])) - 1
    syn1 = 2*np.random.random((y_train.shape[0],y_train.shape[1])) - 1
    synapses = [syn0, syn1]

    trainNeuralNetArrays(synapses, eta, epochs, lambda x: sigmoid(x), lambda x: sigmoid(x)*(1-sigmoid(x)))

    for i in range(len(synapses)):
        if i==0:
            result = sigmoid(np.dot(X_test,synapses[i]))
        else:
            result = sigmoid(np.dot(result,synapses[i]))

    mse = 0.5*np.sum((y_test - result)**2)
    print("    MSE: ",mse)
    if (mse < lowest_mse):
        lowest_mse = mse
        lowest_synapses = synapses
        lowest_syn0 = syn0
        lowest_syn1 = syn1


#gen more random data for test set
np.random.seed(7)
```

```
a = np.random.randint(0,2,5)
b = np.random.randint(0,2,5)
c = a ^ b
X, y = splitInputOutput(np.asmatrix([a,b,c]).T)

# apply the weights
for i in range(len(lowest_synapses)):
    if i==0:
        result = sigmoid(np.dot(X,lowest_synapses[i]))
    else:
        result = sigmoid(np.dot(result,lowest_synapses[i]))

mse = 0.5*np.sum((y - result)**2)
print ("X_test: ",X)
print ("Y_test: ",y)
print ("Y_pred: ",result)
print()
print("MSE against Test data: ",mse)
print("Accuracy: ",1-np.sum(np.abs(y-result))/y.shape[0])
```

```
Performing K-fold cross validation, splits = 5
    MSE:  0.000473691424228
    MSE:  0.000402454142081
    MSE:  0.00066181651387
    MSE:  0.000532135327975
    MSE:  0.000895891246303
X_test:  [[ 1.  1.]
 [ 0.  1.]
 [ 1.  1.]
 [ 0.  0.]
 [ 1.  1.]]
Y_test:  [[0]
 [1]
 [0]
 [0]
 [0]]
Y_pred:  [[ 0.01916215]
 [ 0.98792081]
 [ 0.01916215]
 [ 0.0201313 ]
 [ 0.01916215]]

MSE against Test data:  0.000826370048276
Accuracy:  0.982060612017
```

Looks good. Now lets load our accute inflamation dataset. Our dataset was randomized. It contained 120 rows and we split 80/20 for train&validation(96) / test(24). We'll use scikit learn's KFold utility class to get our indices for a 5 k-fold cross validation and pick the best model to run our test data against.

In [20]:
```
# X,y = getData('data/prepared/dataWithTemp.csv')

df = pd.read_csv('data/prepared/dataWithTempRandomized.train.csv',sep=',',names=["Temp", "Nausea", "Lumbar", "Pushing","Micturi
df["Temp"] = df.transform(lambda x: x - 37)

X,y = splitInputOutput(df.as_matrix())

kf = KFold(n_splits=5,random_state=None, shuffle=True)

lowest_mse = 1e8 #arbitrary high value
lowest_syn0 = []
lowest_syn1 = []

lowest_synapses = None

epochs = 10000

print("Performing K-fold cross validation, splits = 5")
# do our cross validation with training data
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    np.random.seed(seed=42)
    syn0 = 2*np.random.random((X_train.shape[1],X_train.shape[0])) - 1
    syn1 = 2*np.random.random((y_train.shape[0],y_train.shape[1])) - 1
    synapses = [syn0, syn1]

    trainNeuralNetArrays(synapses, 0.1, epochs, lambda x: sigmoid(x), lambda x: sigmoid(x)*(1-sigmoid(x)))

    for i in range(len(synapses)):
        if i==0:
            result = sigmoid(np.dot(X_test,synapses[i]))
        else:
            result = sigmoid(np.dot(result,synapses[i]))
#     layer1_transform = sigmoid(np.dot(X_test,syn0))
#     result = sigmoid(np.dot(layer1_transform,syn1))
    mse = 0.5*np.sum((y_test - result)**2)
    print("    MSE: ",mse)
    if (mse < lowest_mse):
        lowest_mse = mse
        lowest_synapses = synapses
        lowest_syn0 = syn0
        lowest_syn1 = syn1

df = pd.read_csv('data/prepared/dataWithTempRandomized.test.csv',sep=',',names=["Temp", "Nausea", "Lumbar", "Pushing","Micturit
df["Temp"] = df.transform(lambda x: x - 37)
```

```
X,y = splitInputOutput(df.as_matrix())

for i in range(len(lowest_synapses)):
    if i==0:
        result = sigmoid(np.dot(X,lowest_synapses[i]))
    else:
        result = sigmoid(np.dot(result,lowest_synapses[i]))

# layer1_transform = sigmoid(np.dot(X,lowest_syn0))
# result = sigmoid(np.dot(layer1_transform,lowest_syn1))
mse = 0.5*np.sum((y - result)**2)
result = np.double(result > 0.5)
print()
print("MSE against Test data: ",mse)
print("Accuracy: ",1-np.sum(y-result)/y.shape[0])

# NOTES: 1 hidden node outperforms accuracy of 2 hidden node of len(X) x len(X) by almost 1.5:1
```

```
Performing K-fold cross validation, splits = 5
    MSE:  8.24976336748e-05
    MSE:  9.70778642222e-05
    MSE:  5.6819720276e-05
    MSE:  8.78963118942e-05
    MSE:  3.06015703821e-05

MSE against Test data:  9.83419583742e-05
Accuracy:  1.0
```