In [2]:
```
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)

# Trivial implementation
def trainNeuralNet(synapse0, synapse1, epochs, activator, activator_prime):
    for j in range(epochs):
        l1 = activator(np.dot(X,synapse0))
        l2 = activator(np.dot(l1,synapse1))
        l2_delta = (y - l2)*activator_prime(np.dot(l1,synapse1))
        l1_delta = l2_delta.dot(synapse1.T) * activator_prime(np.dot(X,synapse0))
        synapse1 += l1.T.dot(l2_delta) #adjust our synapses up or down as necessary
        synapse0 += X.T.dot(l1_delta)

# Now do it with arrays to generalize it a bit more
def trainNeuralNetArrays(weights, 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] += layers[i-1].T.dot(deltas[i])
            else:
                weights[i] += 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 tuples and Y will be the XOR result of rows in X.

In [4]:
```
# trivial dataset
X, y = getData('data/prepared/trivial.csv')

# X = np.array([ [0,0],[0,1],[1,0],[1,1] ])
# y = np.array([[0,1,1,0]]).T # XOR(X)

np.random.seed(seed=42)
syn0 = 2*np.random.random((X.shape[1],X.shape[0])) - 1
syn1 = 2*np.random.random((y.shape[0],y.shape[1])) - 1
synapses = [syn0, syn1]
epochs = 10000

# trainNeuralNet(syn0, syn1, epochs, lambda x: sigmoid(x), lambda x: sigmoid(x)*(1-sigmoid(x)))
trainNeuralNetArrays(synapses, epochs, lambda x: sigmoid(x), lambda x: sigmoid(x)*(1-sigmoid(x)))

result = transformTestData(X,syn0,syn1,lambda x: sigmoid(x))
# layer1_transform = sigmoid(np.dot(X,syn0))
# result = sigmoid(np.dot(layer1_transform,syn1))

print("MSE: ",0.5*np.sum((y - result)**2))
print("Output of predicted y (2nd and 3rd rows should be close to 1):")
print(result)
```

```
MSE:  0.000477306752279
Output of predicted y (2nd and 3rd rows should be close to 1):
[[ 0.01869055]
 [ 0.98941286]
 [ 0.98293159]
 [ 0.0142077 ]]
```

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 [11]:
```
# 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, 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:  3.99999997882
    MSE:  0.999999981984
    MSE:  2.99999998788
    MSE:  1.00000069577
    MSE:  1.49999985591

MSE against Test data:  1.9999999116
Accuracy:  0.833333333333
```

In [ ]: