NN-3Layers

June 26, 2018

1 Programming Assignment - 1

Subdivision 3,4,5,7 Goals: 1) Implement a 3 layer NN (1 hidden layer) 2) Train the model with linear data 3) Visualize the decision boundary learned by this model 4) Train this model on non-linear data 5) Visualize the decision boundary learned by this model 6) Visualize the effect of learning rate on NN 7) Tabularize the effect of number of nodes in the hidden layer 8) L2 Regularization

In [ ]:
```
# Import Python libraries
import numpy as np
import matplotlib.pyplot as plt
import copy as cpy
import pandas as pd
import time as tm
```

#1) Implement a 3 layer NN (1 hidden layer)

In [72]:
```
class NeuralNetwork_3layer:
    """
    This module implements a Neural Netowrk with hidden layers.
    """
    def __init__(self, input_dim, hidden_dim, output_dim):
        self.W1 = (np.random.randn(input_dim+1, hidden_dim))/np.sqrt(input_dim)
        self.W2 = (np.random.randn(hidden_dim+1, output_dim))/np.sqrt(hidden_dim)
        #self.exp_v = np.vectorize(lambda x : math1.e ** x)

    def compute_cost(self,X,y):
        num_examples = np.shape(X)[0]
        a1 = np.append(np.ones([len(X),1]),X,1)
        z1 = np.dot(a1,self.W1)
        #exp_z1 = math1.e**(z1)
        exp_z1 = np.exp((-1)*z1)
        x2 = (1.)/(1 + exp_z1)
        a2 = np.append(np.ones([len(x2),1]),x2,1)
        z2 = np.dot(a2,self.W2)
        #exp_z2 = math1.e**(z2)
        exp_z2 = np.exp(z2)
        softmax_scores = exp_z2 / np.sum(exp_z2, axis=1, keepdims=True)
```

```
        #exp_z2 = np.exp((-1)*z2)
        #softmax_scores = 1./ (exp_z2 + 1)
        one_hot_y = np.zeros((num_examples,np.max(y)+1))
        logloss = np.zeros((num_examples,))
        for i in range(np.shape(X)[0]):
            one_hot_y[i,y[i]] = 1
            logloss[i] = -np.sum(np.log(softmax_scores[i,:]) * one_hot_y[i,:])
        data_loss = np.sum(logloss)
        return 1./num_examples * data_loss

    def predict(self,X):
        a1 = np.append(np.ones([len(X),1]),X,1)
        z1 = np.dot(a1,self.W1)
        #exp_z1 = self.exp_v((-1)*z1)
        exp_z1 = np.exp((-1)*z1)
        x2 = (1.)/(1 + exp_z1)
        a2 = np.append(np.ones([len(x2),1]),x2,1)
        z2 = np.dot(a2,self.W2)
        #exp_z2 = self.exp_v((-1)*z2)
        exp_z2 = np.exp(z2)
        softmax_scores = exp_z2 / np.sum(exp_z2, axis=1, keepdims=True)

        #exp_z2 = np.exp((-1)*z2)
        #softmax_scores = 1./ (exp_z2 + 1)
        predictions = np.argmax(softmax_scores, axis = 1)
        return predictions

    def plot_learningRate(self,X,y,num_epochs, alpha, lambda1):
        #Error in each epoch:
        Error = np.zeros(num_epochs,)
        for epoch in range(0, num_epochs):
            self.one_epoch(X, y, alpha, lambda1)
            p = self.predict(X)
            Error[epoch] = 0.5*(np.sum((y-p)**2) + lambda1*(np.sum(self.W2**2)) + laml
            #print(Error)
        return Error

    def one_epoch(self, X, y, alpha, lambda1):
        # Forward propagation
        a1 = np.append(np.ones([len(X),1]),X,1)
        z1 = np.dot(a1,self.W1)
        #exp_z1 = self.exp_v((-1)*z1)
        exp_z1 = np.exp((-1)*z1)
        x2 = (1.)/(1 + exp_z1)
        a2 = np.append(np.ones([len(x2),1]),x2,1)
        z2 = np.dot(a2,self.W2)
        #exp_z2 = self.exp_v((-1)*z2)
```

```
            exp_z2 = np.exp(z2)
            softmax_scores = exp_z2 / np.sum(exp_z2, axis=1, keepdims=True)

            #exp_z2 = np.exp((-1)*z2)
            #softmax_scores = 1./ (exp_z2 + 1)

            # Backpropagation
            #compute gradients
            Count_updates = 0
            beta3 = np.zeros_like(softmax_scores)
            one_hot_y = np.zeros_like(softmax_scores)
            for i in range(X.shape[0]):
                one_hot_y[i,y[i]] = 1
            beta3 =  softmax_scores - one_hot_y
            gdash2 = np.multiply(x2,(1-x2))
            beta2 = np.multiply(np.dot(beta3, np.transpose(self.W2[1:,:])) , gdash2)
            #Update weights
            updW2 = alpha * (np.dot(np.transpose(a2),beta3) + lambda1*self.W2)
            updW1 = alpha * (np.dot(np.transpose(a1),beta2) + lambda1*self.W1)
            self.W2 = self.W2 - updW2
            self.W1 = self.W1 - updW1
            Count_updates = Count_updates + (np.sum(updW1!=0) + np.sum(updW2!=0))
            #print(epoch, self.W1, beta2)
            return Count_updates

        def fit(self,X,y,num_epochs,alpha=0.01,lambda1=0):
            #print(self.W1, self.W2)
            for epoch in range(0, num_epochs):
                count_upd = self.one_epoch(X, y, alpha, lambda1)
            return count_upd
```

```
In [62]: X = np.genfromtxt('/Users/pavithraraghavan/Downloads/DATA/NonlinearX.csv', delimiter=
         y = np.genfromtxt('/Users/pavithraraghavan/Downloads/DATA/NonlinearY.csv', delimiter=
         from sklearn.model_selection import train_test_split
         Xtrain, Xtest, ytrain, ytest = train_test_split(X,y)
```

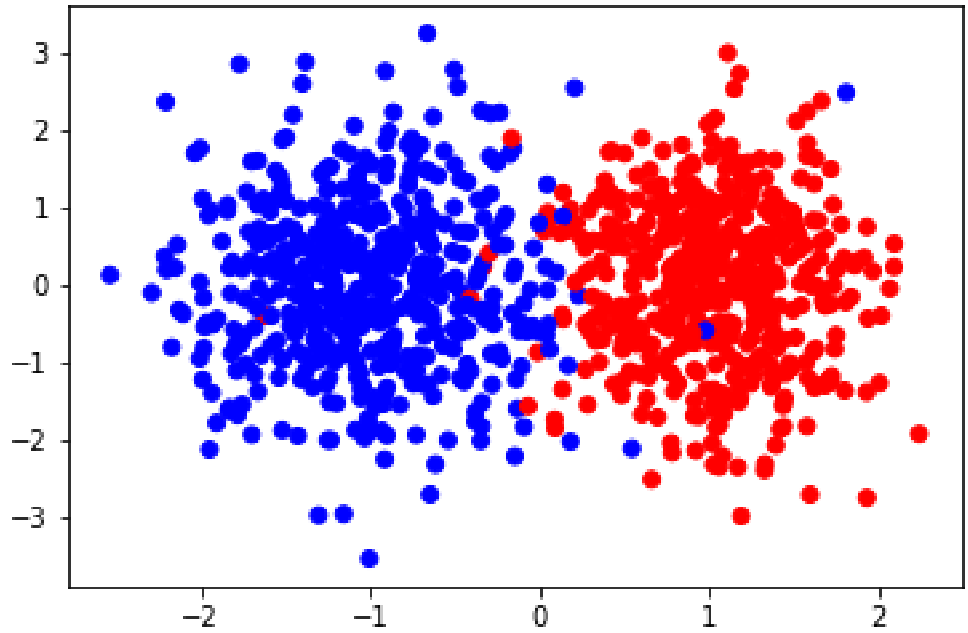
```
In [107]: def plot_decision_boundary(model, X, y, title, filename):

              x1_array, x2_array = np.meshgrid(np.arange(-4, 4, 0.01), np.arange(-4, 4, 0.01))
              grid_coordinates = np.c_[x1_array.ravel(), x2_array.ravel()]
              Z = model.predict(grid_coordinates)
              Z = Z.reshape(x1_array.shape)
              plt.contourf(x1_array, x2_array, Z, cmap=plt.cm.bwr)
              plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.bwr)
              plt.title(title)
              plt.savefig(filename)
              plt.show()
```

Load Data (Linear)

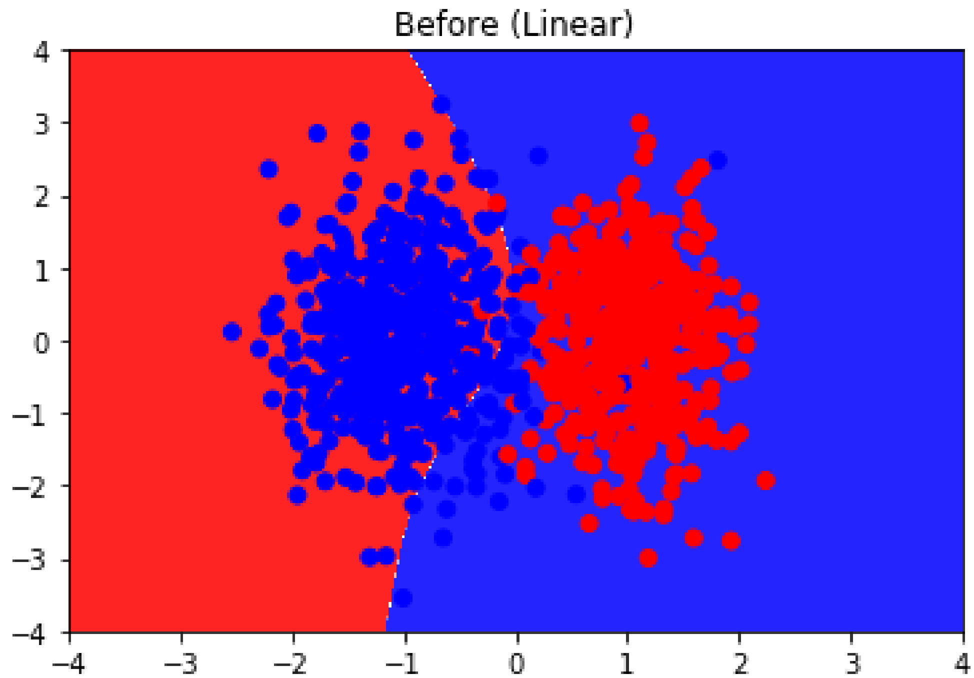
```
In [138]: X = np.genfromtxt('/Users/pavithraraghavan/Downloads/Lab3/DATA/LinearX.csv', delimite
          y = np.genfromtxt('/Users/pavithraraghavan/Downloads/Lab3/DATA/Lineary.csv', delimite

In [111]: #Plot data
          plt.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.bwr)
          plt.show()
```




```
In [112]: #Initialize model
          input_dim = np.shape(X)[1]
          output_dim = np.max(y) + 1
          hidden_dim = 10
          NN = NeuralNetwork_3layer(input_dim, hidden_dim, output_dim)

In [113]: #Plot decision boundary
          plot_decision_boundary(NN, X, y, 'Before (Linear)','/Users/pavithraraghavan/Document
```



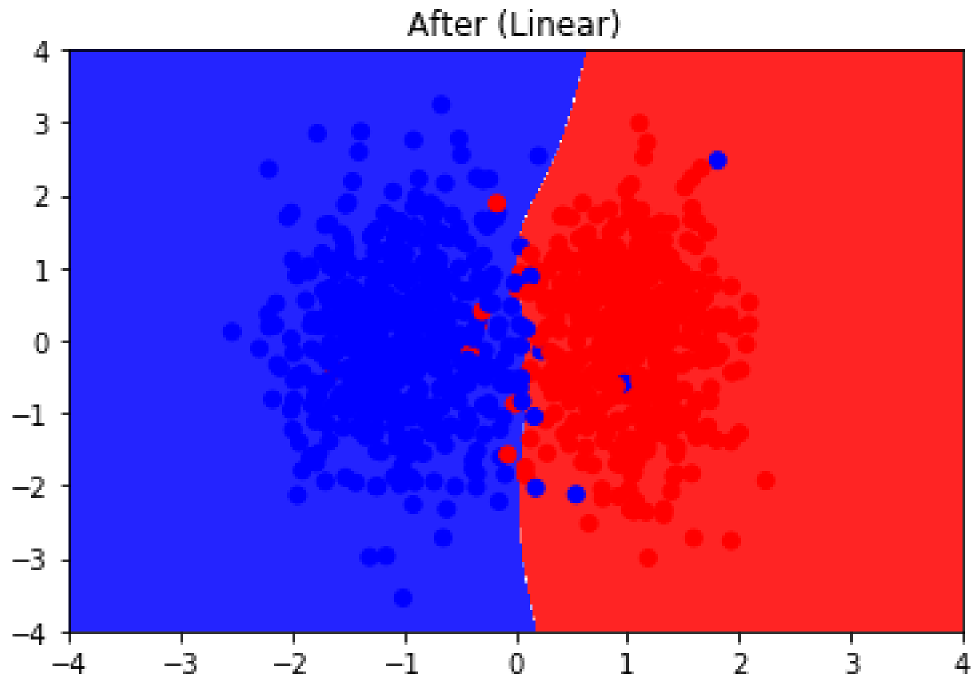
2 2)Training the Model with Linear data

In [114]:
```
NN.fit(X,y,750,alpha=0.01, lambda1=0)
```

Out[114]: 52

#3)Visualize & evaluate the decision boundary learned by this model

In [115]:
```
#Plot decision boundary after training
plot_decision_boundary(NN, X, y,'After (Linear)' ,'/Users/pavithraraghavan/Documents/
```

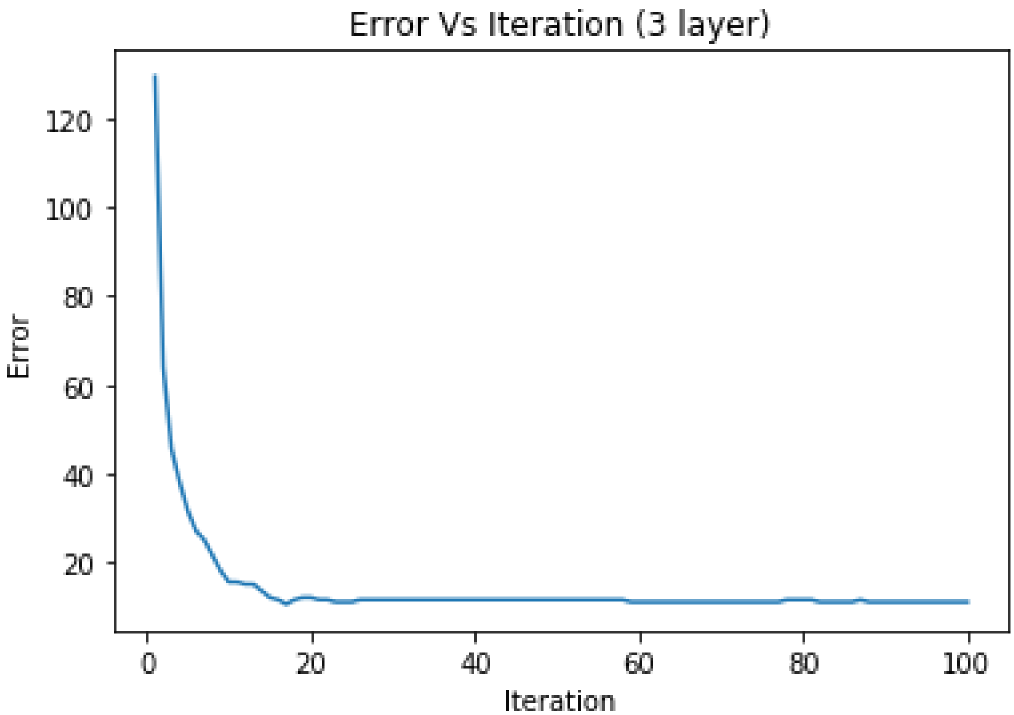



```
In [116]: #Compute accuracy and confusion matrix
          acc = 0
          y_pred = NN.predict(X)
          con_mat = np.zeros((output_dim, output_dim))
          for i in range(len(y_pred)):
              con_mat[y_pred[i], y[i]] += 1
              if y[i] == y_pred[i]:
                  acc += 1.0
          acc = acc/len(y_pred)
          print ('ACCURACY: ', acc)
          print ('CONFUSION MATRIX: \n', con_mat)
```

```
('ACCURACY: ', 0.983)
('CONFUSION MATRIX: \n', array([[492.,   8.],
       [  9., 491.]]))
```

```
In [142]: #Plot error w.r.t number of iterations, for different learning rates:
          x = np.linspace(1, 100, 100)
          input_dim = np.shape(X)[1]
          output_dim = np.max(y) + 1
          hidden_dim = 10
          NN1 = NeuralNetwork_3layer(input_dim, hidden_dim, output_dim)
          Error = NN1.plot_learningRate(X,y,100, 0.001, lambda1=0)
          plt.plot(x, Error)
```

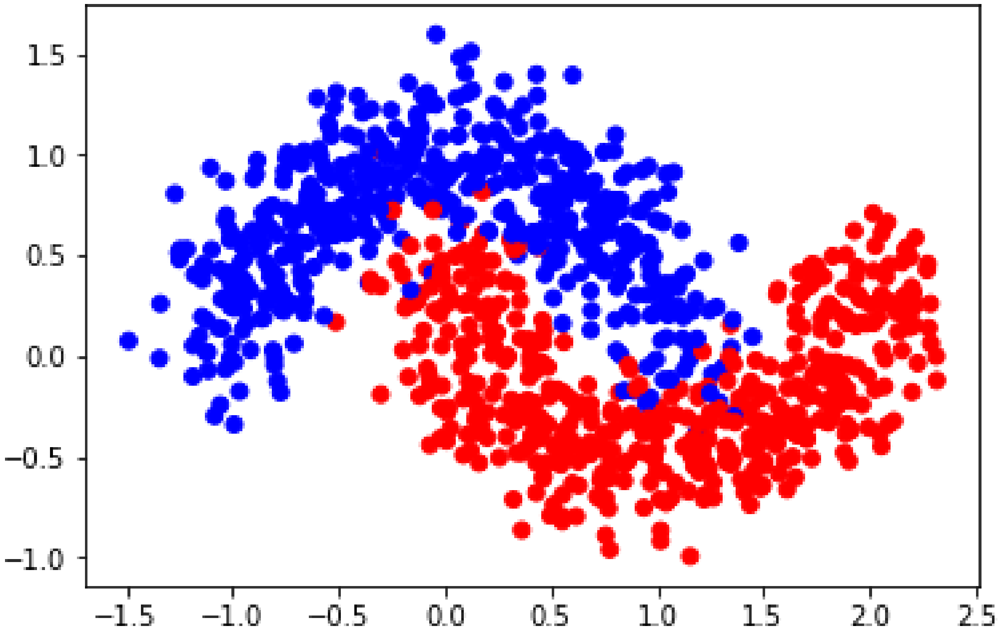
```
#print(Error)
#plt.legend(loc='best')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error Vs Iteration (3 layer)')
plt.savefig('/Users/pavithraraghavan/Documents/BU/2nd_sem/AI/P1/Plots/LR_Hidden.png')
plt.show()
```



Load Data (Non-Linear)

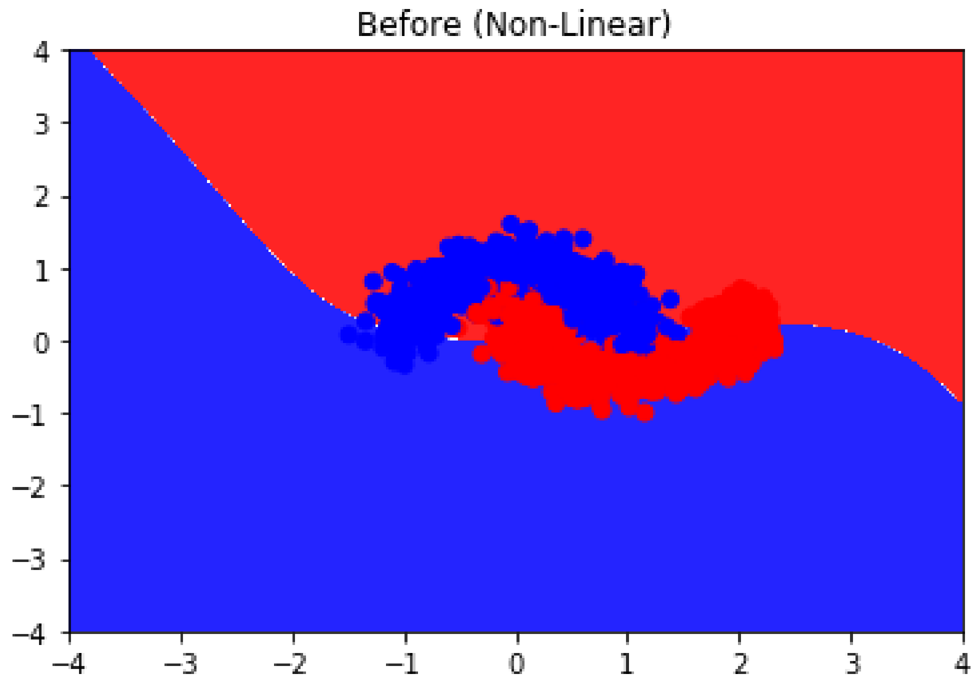
```
In [143]: X = np.genfromtxt('/Users/pavithraraghavan/Downloads/DATA/NonlinearX.csv', delimiter=
          y = np.genfromtxt('/Users/pavithraraghavan/Downloads/DATA/NonlinearY.csv', delimiter=
```

```
In [118]: #plot data
          plt.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.bwr)
          plt.show()
```



In [125]:
```
#Initialize model
input_dim = np.shape(X)[1]
output_dim = np.max(y) + 1
hidden_dim = 10
NN = NeuralNetwork_3layer(input_dim, hidden_dim, output_dim)
```

In [126]:
```
#Plot decision boundary
plot_decision_boundary(NN, X, y,'Before (Non-Linear)','/Users/pavithraraghavan/Docume
```



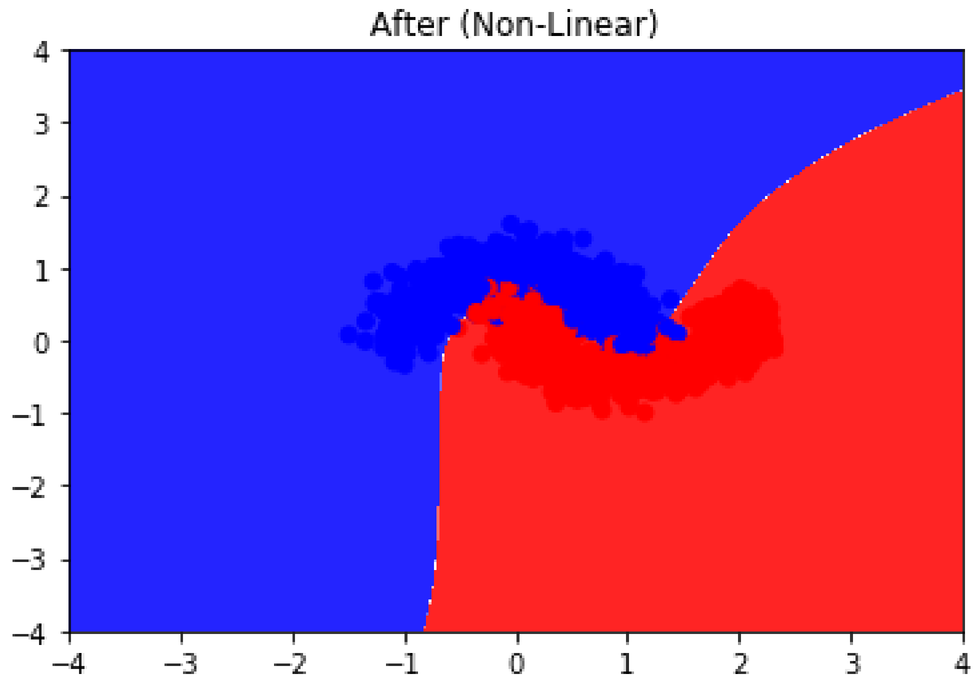
3 4)Training the Model with Non-Linear data

```
In [127]: NN.fit(X,y,1000,alpha=0.01, lambda1=0)

Out[127]: 52
```

#5) Visualize and Evaluate decision boundary learned by this Model

```
In [128]: #Plot decision boundary after training
          plot_decision_boundary(NN, X, y,'After (Non-Linear)', '/Users/pavithraraghavan/Docume
```

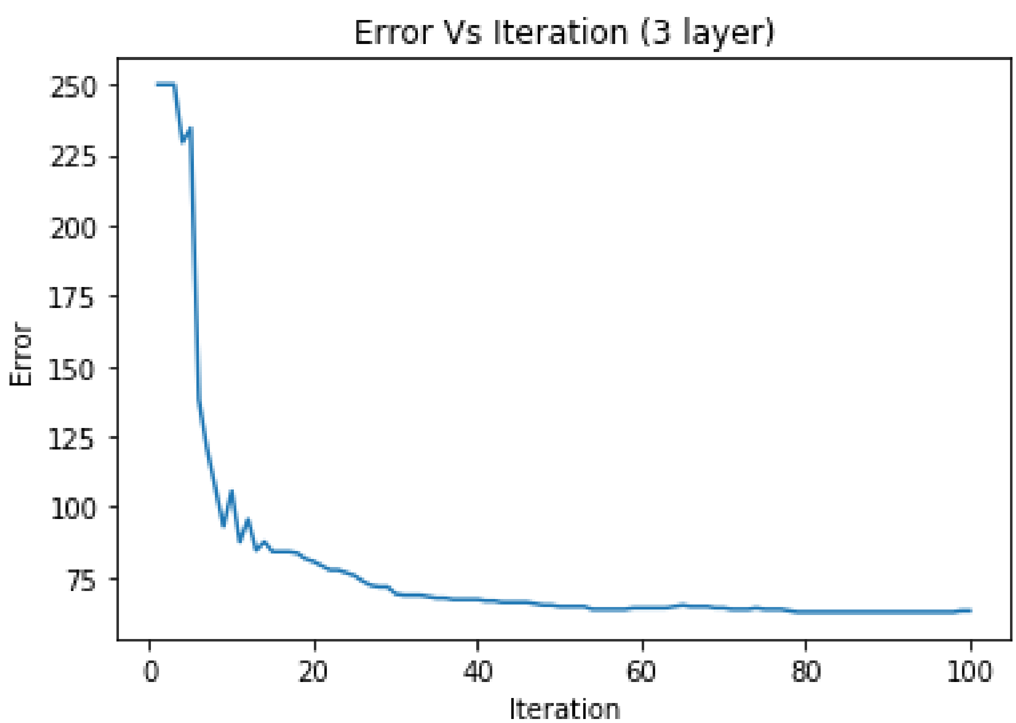


In [17]:
```
#Compute accuracy and confusion matrix
acc = 0
y_pred = NN.predict(X)
con_mat = np.zeros((output_dim, output_dim))
for i in range(len(y_pred)):
    con_mat[y_pred[i], y[i]] += 1
    if y[i] == y_pred[i]:
        acc += 1.0
print(acc)
#print(np.sum(y_pred==y))
acc = acc/len(y_pred)
print ('ACCURACY: ', acc)
print ('CONFUSION MATRIX: ')
print(con_mat)
```

```
969.0
('ACCURACY: ', 0.969)
CONFUSION MATRIX:
[[483.  14.]
 [ 17. 486.]]
```

In [144]:
```
#Plot error w.r.t number of iterations, for different learning rates:
x = np.linspace(1, 100, 100)
input_dim = np.shape(X)[1]
```

```
output_dim = np.max(y) + 1
hidden_dim = 10
NN1 = NeuralNetwork_3layer(input_dim, hidden_dim, output_dim)
Error = NN1.plot_learningRate(X,y,100, 0.001, lambda1=0)
plt.plot(x, Error)
#print(Error)
#plt.legend(loc='best')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error Vs Iteration (3 layer)')
plt.savefig('/Users/pavithraraghavan/Documents/BU/2nd_sem/AI/P1/Plots/LR_Hidden.png')
plt.show()
```



4 6) Effect of learning rate on NN

NN has 10 hidden layers

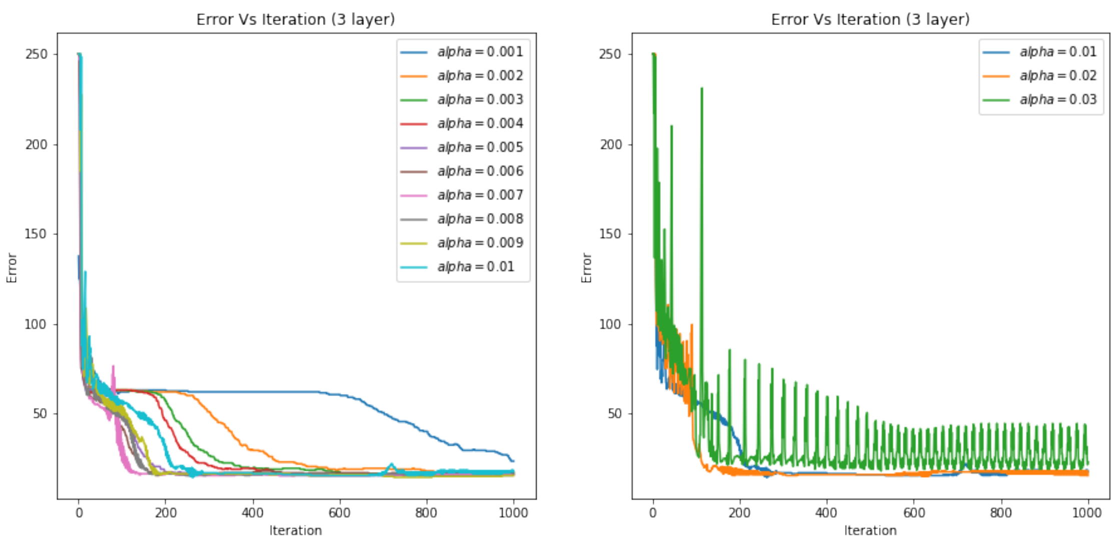
In [148]:
```
#Plot error w.r.t number of iterations, for different learning rates:
x = np.linspace(1, 1000, 1000)
input_dim = np.shape(X)[1]
output_dim = np.max(y) + 1
hidden_dim = 10
NN = NeuralNetwork_3layer(input_dim, hidden_dim, output_dim)
```

```
f,ax = plt.subplots(1,2,figsize=(15,15))
for i in range(1, 11): #10 plots
    NN1 = cpy.deepcopy(NN)
    Error = NN1.plot_learningRate(X,y,1000, 0.001*i, lambda1=0)
    plt.subplot(221)
    plt.plot(x, Error, label='$alpha = {j}$'.format(j=0.001*i))
    #print(Error)

for i in range(1, 4): #10 plots
    NN1 = cpy.deepcopy(NN)
    Error = NN1.plot_learningRate(X,y,1000, 0.01*i, lambda1=0)
    plt.subplot(222)
    plt.plot(x, Error, label='$alpha = {j}$'.format(j=0.01*i))
plt.subplot(221)
plt.legend(loc='best')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error Vs Iteration (3 layer)')

plt.subplot(222)
plt.legend(loc='best')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.title('Error Vs Iteration (3 layer)')

plt.savefig('/Users/pavithraraghavan/Documents/BU/2nd_sem/AI/P1/Plots/LR_Hidden_new.
plt.show()
```



5 7) Effect of number of Hidden Layer nodes

Evaluation through Accuracy, Cost function, time elapsed and number of updates

In [149]:
```
#for different number of nodes in the hidden layer:
x = np.linspace(1, 1000, 1000)
X = np.genfromtxt('/Users/pavithraraghavan/Downloads/DATA/NonlinearX.csv', delimiter=
y = np.genfromtxt('/Users/pavithraraghavan/Downloads/DATA/NonlinearY.csv', delimiter=
input_dim = np.shape(X)[1]
output_dim = np.max(y) + 1
display_mat = np.zeros([20,5]) #num of nodes, accuracy, cost, time, num of updates
#print(display_mat)
for i in range(0, 20): #20 plots
    hidden_dim = i+1
    NN = NeuralNetwork_3layer(input_dim, hidden_dim, output_dim)
    start_time = tm.time()
    count_upd = NN.fit(X,y,1000, 0.01, 0)
    elapsed_time = tm.time() - start_time
    y_pred = NN.predict(X)
    display_mat[i,0] = i+1
    display_mat[i,1] = (np.sum(y_pred==y))*1. / len(y_pred)
    display_mat[i,2] = NN.compute_cost(X,y)
    display_mat[i,3] = elapsed_time
    display_mat[i,4] = count_upd
```

In [150]:
```
#export matrix to csv
pd.set_option('display.precision',5)
df =pd.DataFrame(display_mat)
df.columns = ['#Nodes','Accuracy','CostFunction','TimeElapsed','#Updates']
df.style
print(df)
df.to_csv('/Users/pavithraraghavan/Documents/BU/2nd_sem/AI/P1/Plots/NodesNum.csv', en
```

```
    #Nodes  Accuracy  CostFunction  TimeElapsed  #Updates
0      1.0     0.871       0.32527      0.61274       7.0
1      2.0     0.887       0.32277      1.04259      12.0
2      3.0     0.881       0.31302      0.89794      17.0
3      4.0     0.965       0.08812      0.85636      22.0
4      5.0     0.949       0.13034      0.59349      27.0
5      6.0     0.967       0.08321      0.61423      32.0
6      7.0     0.970       0.08226      0.89329      37.0
7      8.0     0.965       0.08770      0.79272      42.0
8      9.0     0.966       0.08741      0.65206      47.0
9     10.0     0.969       0.07930      0.70333      52.0
10    11.0     0.969       0.07901      0.69074      57.0
11    12.0     0.966       0.08566      0.85673      62.0
12    13.0     0.967       0.07928      0.99102      67.0
13    14.0     0.967       0.07828      0.77222      72.0
14    15.0     0.968       0.07905      0.75163      77.0
```

15	16.0	0.965	0.08733	0.76503	82.0
16	17.0	0.968	0.07993	0.89266	87.0
17	18.0	0.969	0.07796	0.90861	92.0
18	19.0	0.966	0.08168	0.93453	97.0
19	20.0	0.967	0.07899	0.96632	102.0

6 8) L2 Regularization

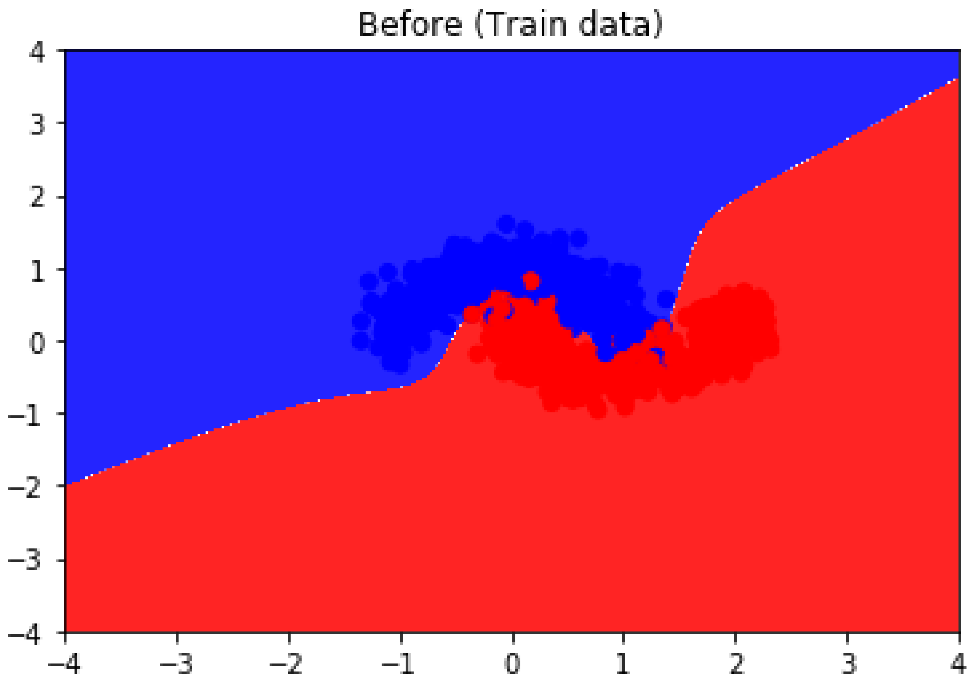
In [129]:
```
X = np.genfromtxt('/Users/pavithraraghavan/Downloads/DATA/NonlinearX.csv', delimiter=
y = np.genfromtxt('/Users/pavithraraghavan/Downloads/DATA/NonlinearY.csv', delimiter=
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X,y)
```

In [130]:
```
#Initialize model
input_dim = np.shape(X)[1]
output_dim = np.max(y) + 1
hidden_dim = 20
NN = NeuralNetwork_3layer(input_dim, hidden_dim, output_dim)
NN_L2 = cpy.deepcopy(NN)
```

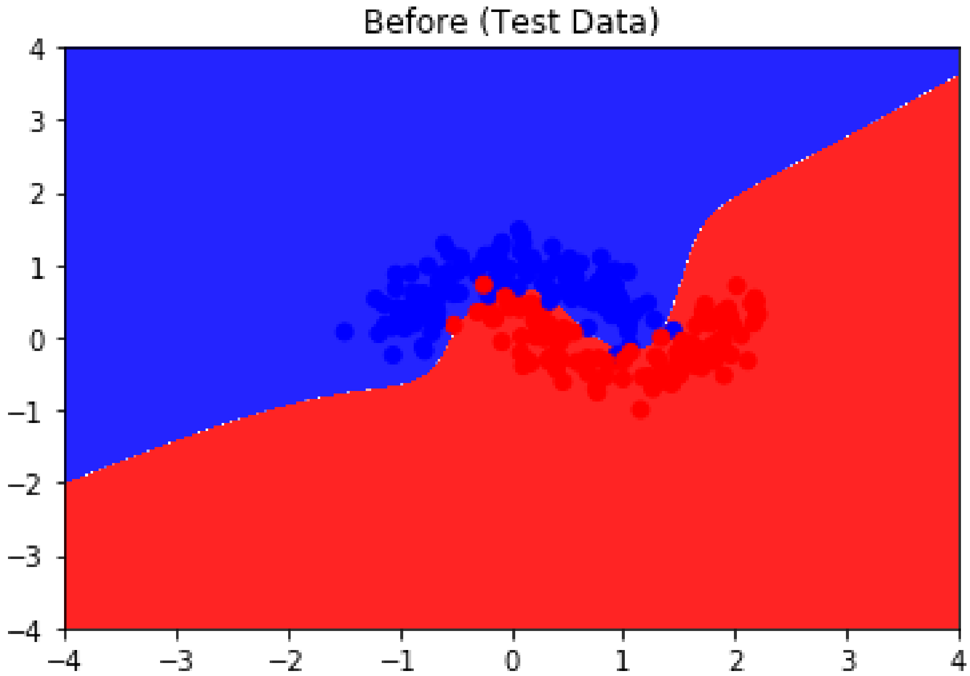
In [131]:
```
NN.fit(Xtrain,ytrain,1000,alpha=0.01, lambda1=0)
```

Out[131]: 102

In [132]:
```
#Plot decision boundary after training
plot_decision_boundary(NN, Xtrain, ytrain,'Before (Train data)' ,'/Users/pavithraragh
```



In [133]:
```
y_pred_test = NN.predict(Xtest)
y_pred_train = NN.predict(Xtrain)
plot_decision_boundary(NN, Xtest, ytest, 'Before (Test Data)', '/Users/pavithraragha
```



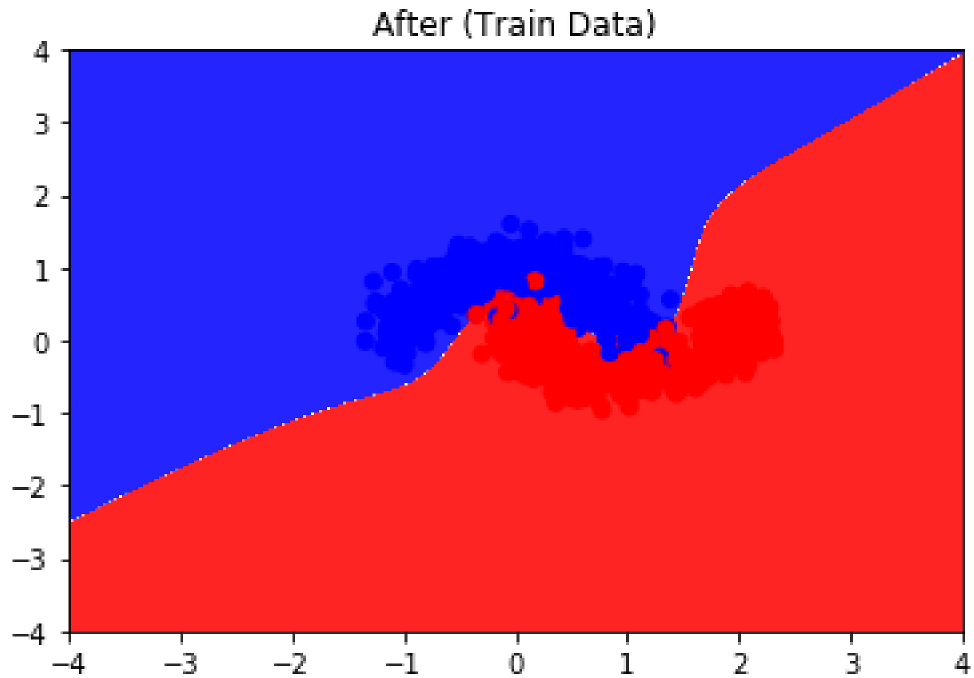
In [134]:
```
train_acc = (np.sum(y_pred_train==ytrain))*1. / len(y_pred_train)
test_acc = (np.sum(y_pred_test==ytest))*1. / len(y_pred_test)
print("Training accuracy: %0.4f" % (train_acc))
print("Testing accuracy: %0.4f" % (test_acc))
```

```
Training accuracy: 0.9627
Testing accuracy: 0.9800
```

In [135]:
```
NN_L2.fit(Xtrain,ytrain,1000,alpha=0.01, lambda1=0.03)
```

Out[135]: 102

In [136]:
```
#Plot decision boundary after training
plot_decision_boundary(NN_L2, Xtrain, ytrain,'After (Train Data)', '/Users/pavithrara
```



In [137]:
```
y_pred_test = NN_L2.predict(Xtest)
y_pred_train = NN_L2.predict(Xtrain)
plot_decision_boundary(NN_L2, Xtest, ytest, 'After (Test Data)','/Users/pavithraragha
```

In [106]:
```
train_acc = (np.sum(y_pred_train==ytrain))*1. / len(y_pred_train)
test_acc = (np.sum(y_pred_test==ytest))*1. / len(y_pred_test)
print("Training accuracy: %0.4f" % (train_acc))
print("Testing accuracy: %0.4f" % (test_acc))
```

```
Training accuracy: 0.9733
Testing accuracy: 0.9520
```