Assignment 1
IEORE 4742 Deep Learning for OR and FE

Ahmad Shayaan[1]
as5948

$\{$ahmad.shayaan@columbia.edu$\}$@columbia.edu
Columbia University
September 29, 2019

[1]All authors contributed equally to this work.

Contents

Chapter 1

Solution for Question 1

1.1 Question 1 Part 1

We had to design an algorithm that linearly separates two curves defined on the space Ω. Where is $\Omega = [-1, 1] \times [-1, 1]$ and the equation of the curves is given as follows.

$$y_1 = -0.6 \sin\left(\frac{\pi}{2} + 3x\right) - 0.35$$

$$y - 2 = -0.6 \sin\left(\frac{\pi}{2} + 3x\right) + 0.25$$

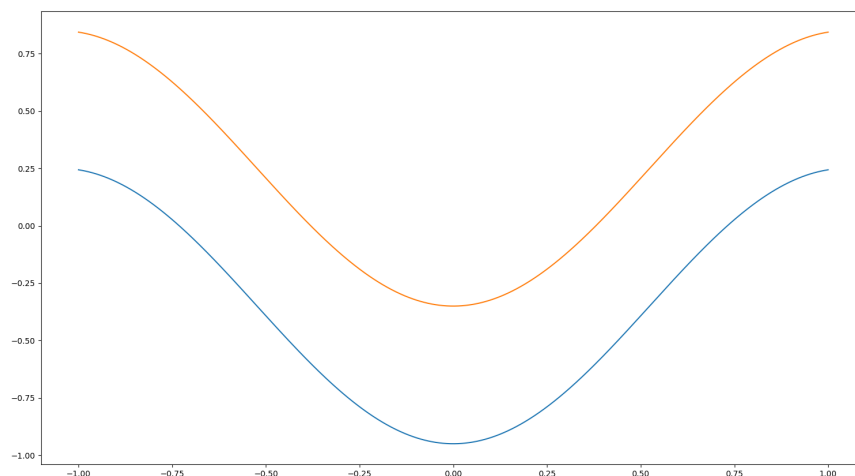The curve plotted in the space is shown in Figure1.1

Figure 1.1: Curves Plotted in the space Ω

I decided to label one of the figure with class label one and the other with class label zero. I then proceeded to train a logistic regression separator with loss criteria as Binary

Cross entropy loss. The implementation of the linear separator class is given below. I used python and PyTorch library to implement the linear separator.

```python
class LinearSperator():
    """docstring for LinearSperator"""
    def __init__(self,learning_rate):
        self.learning_rate = learning_rate
        self.W = torch.ones(1,dtype=torch.float64,requires_grad=True)
        self.B = torch.ones(1,dtype=torch.float64,requires_grad=True)
        self.optimizer = torch.optim.SGD([self.W,self.B],lr = self.learning_rate)
        self.loss = nn.BCELoss(reduction='mean')


    def forward(self,data):
        return torch.sigmoid(self.W*data + self.B)


    def train(self,data,label):
        output = self.forward(data)
        loss = self.loss(output,label)
        print (loss.item())
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
```

At each iteration the model is fed in a curve and the output is then used to adjust the weights to correctly classify the curves. I used the Stochastic Gradient Descent optimizer of PyTorch library to adjust the weights. Figure1.2 shows the linear separator.
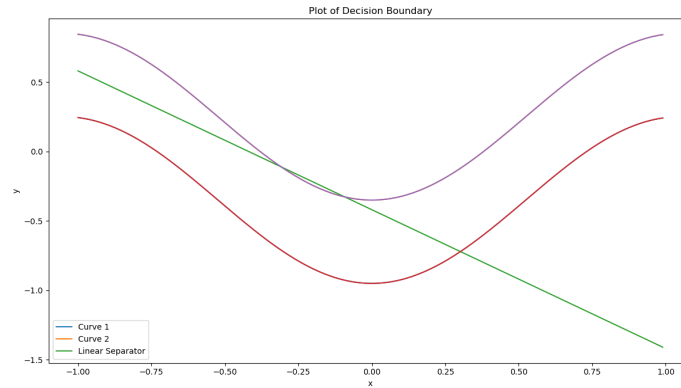
Figure 1.2: Linear Separator for the two curves

The separator almost perfectly able to separate the two curves.

1.2   Question 1 Part 2

In this question we had to do a gird search to find the optimal parameters for the transformed space and also the parameters for the separator in the transformed space. I used a grid search approach to find the global optimal for the set $\{W_{11}, W_{12}, W_{21}, W_{22}, b_1, b_2, a, b, c\}$. The source code for my solution is shown below.

```
1   def generatingSpace():
      for w11 in range(-3,3):
3       for w12 in range(-3,3):
          for w21 in range(-3,3):
5           for w22 in range(-3,3):
              for b1 in range(-1,2):
7               for b2 in range(-1,2):
                  for a in range(-6,6):
9                   for b in range(-6,6):
                      for c in range(-6,6):
11                      w = np.array([a,b,c])
                        Flag = True

13
                        x_hat = np.tanh(w11*x + w21*y + b1)
15                      y_hat = np.tanh(w12*x + w22*y + b2)

17                      x_hat_curve_1 = np.tanh(w11*x + w21*y1 + b1
      )
                        y_hat_curve_1 = np.tanh(w12*x + w22*y1 + b2
      )
```

```
19
                               x_hat_curve_2 = np.tanh(w11*x + w21*y2 + b1
     )
21                             y_hat_curve_2 = np.tanh(w12*x + w22*y2 + b2
     )

23                             for (i,j) in zip(x_hat_curve_1,
     y_hat_curve_1):
                               product = np.dot(w, np.array([i,j,1]))
25                             if product >= 0:
                               Flag = False
27                             break

29                             if Flag==True:
                                 for (i,j) in zip(x_hat_curve_2,
     y_hat_curve_2):
31
                                 product = np.dot(w, np.array([i,j,1]))
33                                 if product < 0:
                                 Flag = False
35                                 break

37                             if Flag==True:
                                 return w11, w12, w21, w22, b1, b2, a, b,
     c
```

Using grid search we are able to find the best linear separator as it enumerates over all the possible combinations in the parameter space. The Linear Separator is shown in Figure1.3.
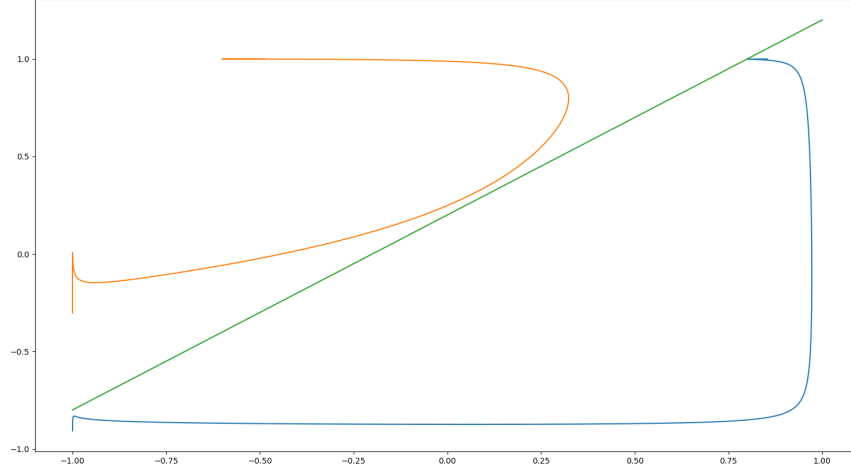
Figure 1.3: Grid Search Linear Separator

Table 1.1 shows the values of the optimal parameter set obtained by gird search.

Table 1.1: Optimal Parameter Set

Parameters	Value
W11	-3
W12	-3
W21	-3
W22	2
b1	-1
b2	1
a	-5
b	5
c	-1

1.3   Question 1 Part 2

In this question I used other strategies to find the optimal parameter set for the transformation and the linear separator. I used logistic regression and SVM to find the optimal parameter set. I stopped searching for the optimal parameter set when the mean accuracy of classification of the two curve is above some threshold. The threshold for stop searching was set empirically.

The source for finding the optimal parameters sets using logistic regression and support vector machines is given below.

```python
def generatingSpace():
  for w11 in range(-3,4,1):
    for w12 in range(-3,4,1):
      for w21 in range(-3,4,1):
        for w22 in range(2,4,1):
          for b1 in range(0,2,1):
            for b2 in range(0,2,1):
              clf = svm.SVC(kernel='linear',gamma='auto')
              lr = LogisticRegression()
              x_hat_curve_1 = np.tanh(w11*x + w21*y1 + b1)
              y_hat_curve_1 = np.tanh(w12*x + w22*y1 + b2)

              x_hat_curve_2 = np.tanh(w11*x + w21*y2 + b1)
              y_hat_curve_2 = np.tanh(w12*x + w22*y2 + b2)


              x_hat = np.concatenate((x_hat_curve_1,
x_hat_curve_2),axis=0)
              y_hat = np.concatenate((y_hat_curve_1,
y_hat_curve_2),axis=0)


              label1 = np.repeat(np.array([1]),x_hat_curve_1.
shape[0])
              label2 = np.repeat(np.array([-1]),x_hat_curve_2.
shape[0])
              labels = np.concatenate((label1,label2))

              df = pd.DataFrame({'X':x_hat[:],'Y':y_hat[:],'
labels':labels[:]})
              df = shuffle(df)


              data = np.array([df['X'],df['Y']]).reshape(400,2)
              labels = np.array(df['labels'])

              clf.fit(data,labels)
              lr.fit(data,labels)

              print ("SVM SCORE " + str(clf.score(data,labels))
)
              print ("LR SCORE " + str(lr.score(data,labels)))

              if (lr.score(data,labels) > 0.57):
```

```
                    return w11,w12, w21, w22, b1, b2, lr.coef_[0],
    lr.intercept_[0].
```

Logistic regression seeks to minimize the cross entropy loss where as the support vector machine uses the hinge loss function in which we seek to maximize the margins from both the curve and fit the most optimal separator. Figure 1.4 and 1.5 show the linear separators found using the two different strategies.
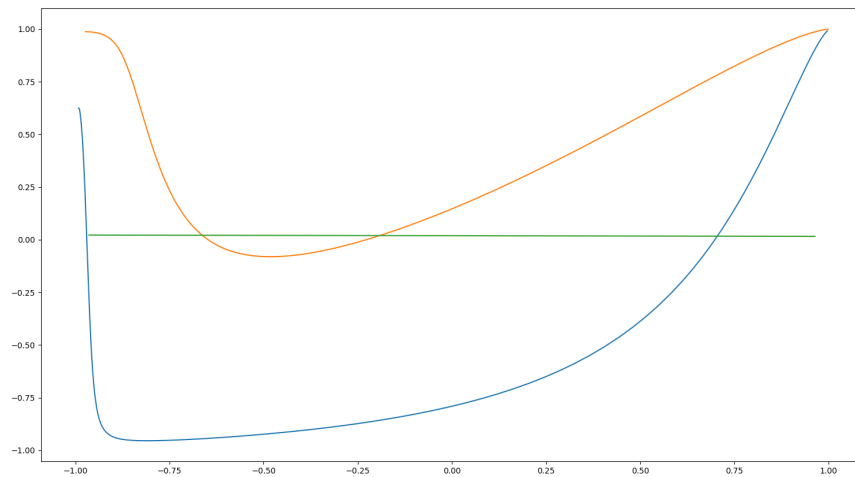

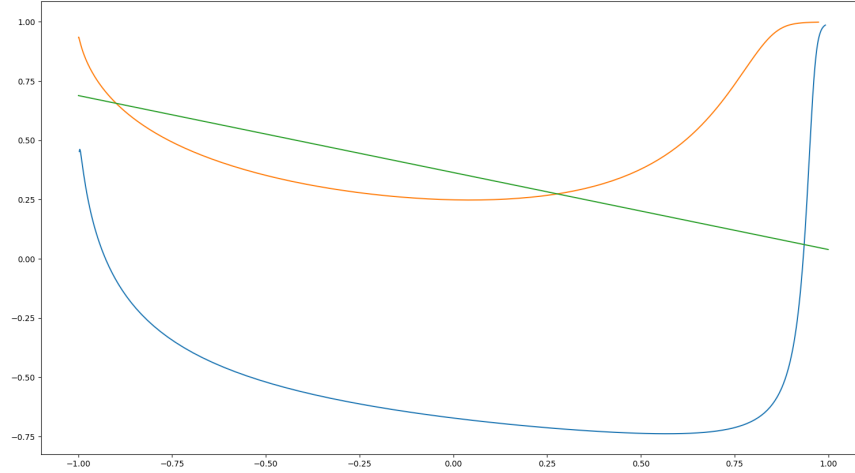
Figure 1.4: SVM Linear separator

Figure 1.5: Logistic Regression Linear separator

The linear separator are not as good as the gird search because these strategies stop when they reach a local optimum where as in grid search we stop when we reach a global optimum. The optimal parameter set found by the two strategies is shown in Table.

Table 1.2: Optimal Parameter set for LR and SVM

Parameters	Logistic Regression Values	SVM Values
W11	-3	-3
W12	-1	-2
W21	-1	-2
W22	2	3
b1	0	0
b2	1	1
a	0.12	-0.0096
b	0.369	1.015
c	-0.134	-0.0054

Chapter 2

Solution for Question 2

2.1 Question 2 Part 1

We had to plot the curves defined by the parameters given in the questions. The code to generate and plot the curves is given below.

```python
def generatingSpace(t):
  r1 = 50 + 0.2 * t
  r2 = 30 + 0.4 * t
  phi1 = -0.06*t + 3
  phi2 = -0.08*t + 2

  #curve1
  x1 = r1 * np.cos(phi1)
  y1 = r1 * np.sin(phi1)

  #curve2
  x2 = r2 * np.cos(phi2)
  y2 = r2 * np.sin(phi2)

  plt.plot(x1,y1)
  plt.plot(x2,y2)
  plt.show()

  return np.array([x1,y1,np.ones(x1.shape)]), np.array([x2,y2,
  np.zeros(x2.shape)])
```
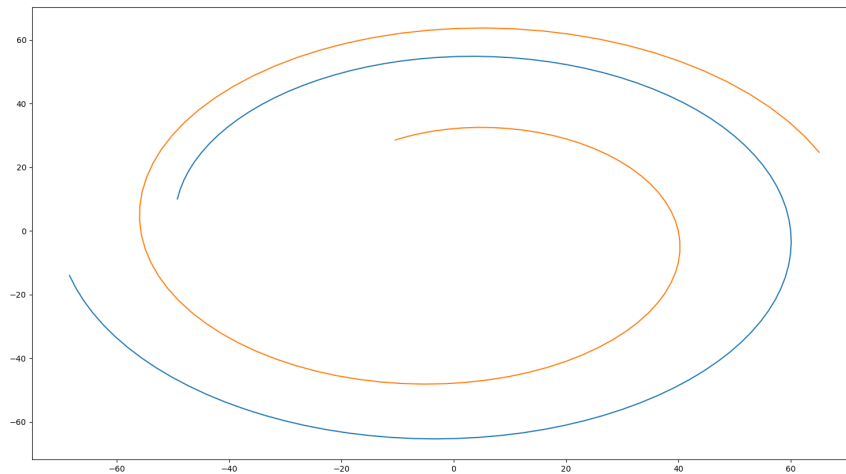
Figure shows the two curves.

Figure 2.1: The given curves

2.2 Question 2 Part 2

We had to code a feed forward neural network to linear classification of the given curves. I used python to code the neural network. My solution consisted of two parts. I used the neural network to transform the topology of the curves and then trained a linear classifier to classify the curves in the transformed space. The Class which defines the neural network and all it's parameters is shown below.

```python
import torch
import torch.nn as nn
import pandas as pd
import numpy as np

class Network(nn.Module):
    """docstring for Network"""
    def __init__(self):
        super(Network, self).__init__()
        self.relu = nn.ReLU()
        self.Sigmoid = nn.Sigmoid()

        self.fc1 = nn.Linear(2,12)
        self.fc2 = nn.Linear(12,18)
        self.fc3 = nn.Linear(18,8)
        self.fc4 = nn.Linear(8,4)
        self.fc5 = nn.Linear(4,2)

    def forward(self,data):
```

11

```
            out = self.relu(self.fc1(data))
21          out = self.relu(self.fc2(out))
            out = self.relu(self.fc3(out))
23          out = self.relu(self.fc4(out))
            out = self.relu(self.fc5(out))
25          return out
```

The code to train the neural network and the linear separator is given below. I used the adam optimizer provided by the PyTorch library to find the optimal. The linear separator and the weights of the neural network are adjusted by computing the binary cross entropy loss in classification

```
1   class Train(object):
      """docstring for Train"""
3     def __init__(self,epochs,learning_rate):
        super(Train, self).__init__()
5       self.learning_rate = learning_rate
        self.net = Network()
7       self.W = torch.ones(2,requires_grad=True)
        self.b = torch.ones(1,requires_grad=True)
9       self.optimizer = torch.optim.Adam(itertools.chain(self.net.
    parameters(), [self.W], [self.b]), lr = self.learning_rate)
        self.loss_function =  torch.nn.BCELoss(reduction='mean')
11
    def trainModel(model,df):
13      total_loss = 0
        for i in range(len(df)):
15      data = torch.tensor((df.iloc[i]['X'],df.iloc[i]['Y'])).float
    ()
        label = torch.tensor(df.iloc[i]['label'])
17      out = model.net.forward(data)
        out = torch.sigmoid(torch.matmul(model.W,out) + model.b )
19      loss = model.loss_function(out,label)

21      model.optimizer.zero_grad()
        loss.backward()
23      model.optimizer.step()

25      total_loss += loss.item()

27      return model,total_loss/len(df)
```

Figure 2.2: Loss per epoch

The minimum loss that we attain is 0.4269.