Assignment 1

IEORE 4742 Deep Learning for OR and FE

Ahmad Shayaan[1]
as5948

{ahmad.shayaan@columbia.edu}@columbia.edu
Columbia University
October 2, 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 are given as follows.

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

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

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

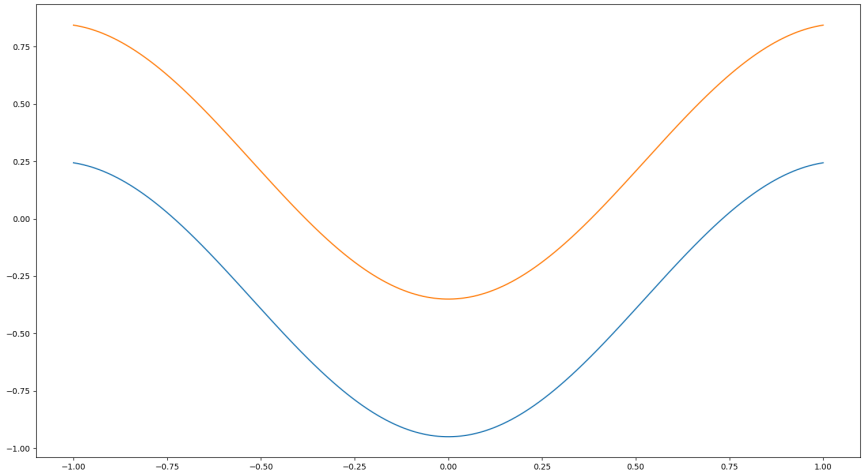

Figure 1.1: Curves Plotted in the space Ω

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.

```
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.

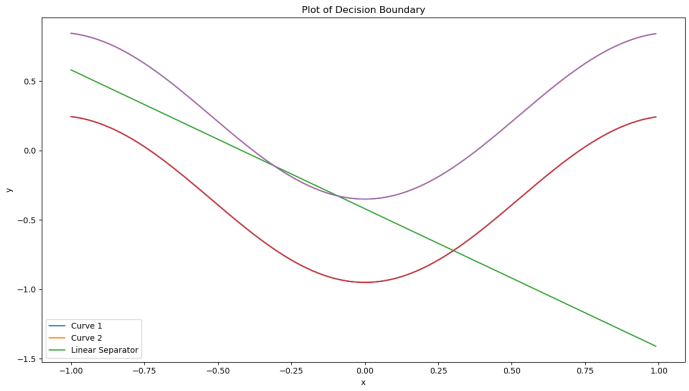

Figure 1.2: Linear Separator for the two curves

The separator almost separates the two curves in the space Ω.

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.

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

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

                    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.

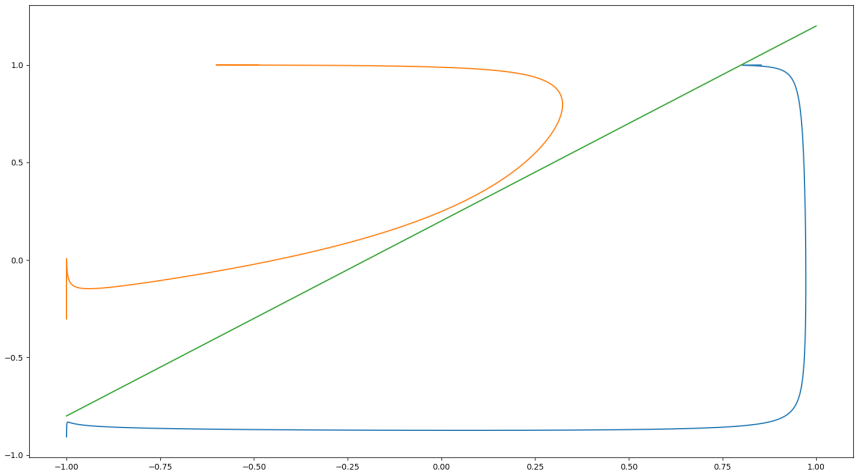

Figure 1.3: Grid Search Linear Separator

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 to stop searching for the optimal parameter set was set empirically.

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

```
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.

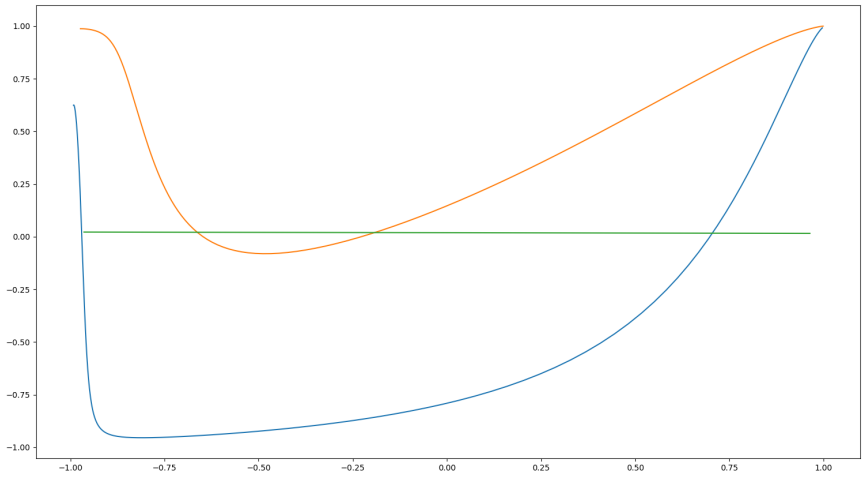

Figure 1.4: SVM Linear separator

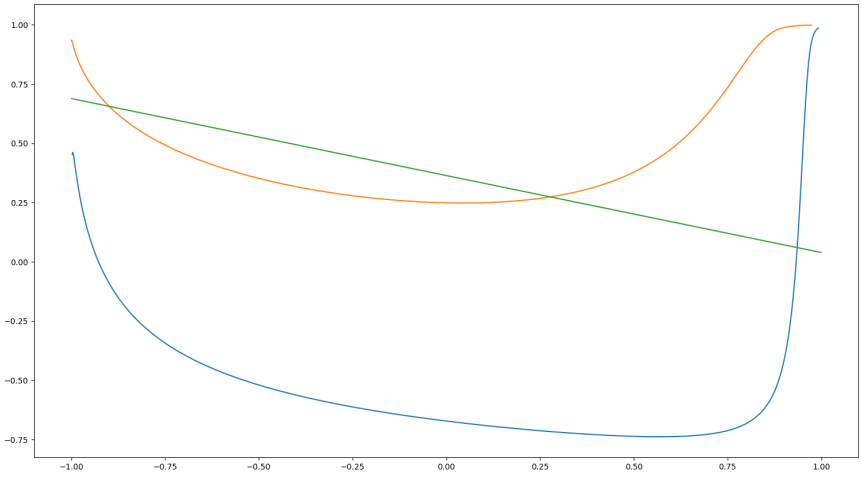

Figure 1.5: Logistic Regression Linear separator

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.

```
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 2.1 shows the two curves.

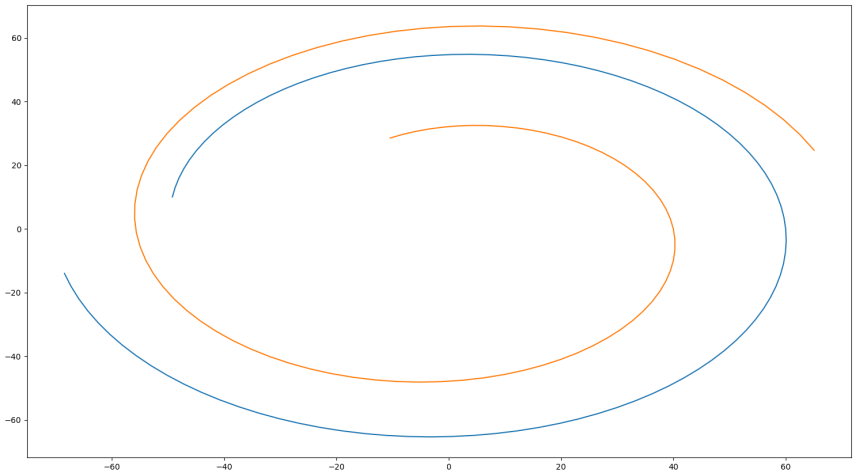

Figure 2.1: The given curves

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.

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

    self.fc1 = nn.Linear(2,100,bias=True)
    self.fc2 = nn.Linear(100,50,bias=True)
    self.fc3 = nn.Linear(50,2)

  def forward(self,data):
    out = self.tanh(self.fc1(data))
    out = self.tanh(self.fc2(out))
    out = self.tanh(self.fc3(out))
    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

```
class Train(object):
  """docstring for Train"""
  def __init__(self,epochs,learning_rate):
    super(Train, self).__init__()
    self.learning_rate = learning_rate
    self.net = Network()
    self.W = torch.ones(2,requires_grad=True)
    self.b = torch.ones(1,requires_grad=True)
    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')

def trainModel(model,df):
  total_loss = 0
  for i in range(len(df)):
  data = torch.tensor((df.iloc[i]['X'],df.iloc[i]['Y'])).float
 ()
  label = torch.tensor(df.iloc[i]['label'])
  out = model.net.forward(data)
  out = torch.sigmoid(torch.matmul(model.W,out) + model.b )
  loss = model.loss_function(out,label)

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

  total_loss += loss.item()

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

The transformed curves and the linear separators are shown in figure 2.2

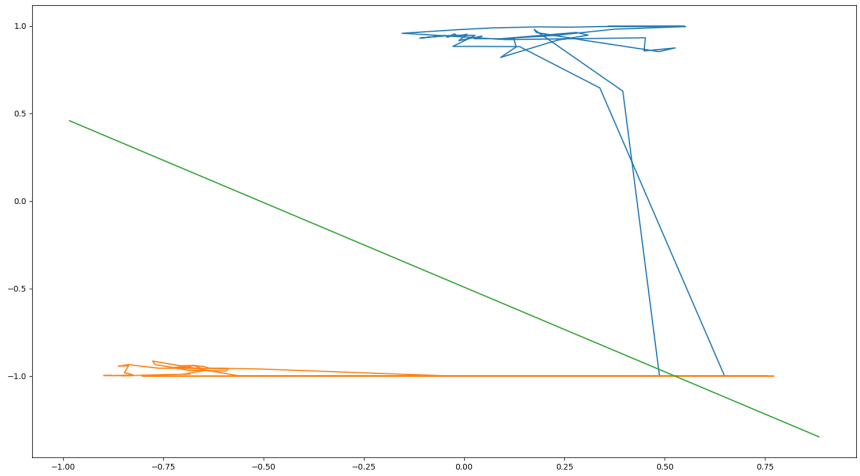

Figure 2.2: The curves and linear separator in the spcae transformed by the neural network

It can be seen from figure 2.2 that the neural network is effectively able to learn the transformations such that the curves become linearly separable. Figure 2.3 shows the loss per epoch.

```
Epoch: 78 Loss: 0.4481905805313903
Epoch: 79 Loss: 0.45223833550169895
Epoch: 80 Loss: 0.45111404864395016
Epoch: 81 Loss: 0.4536170519657922
Epoch: 82 Loss: 0.4337056218846372
Epoch: 83 Loss: 0.44502608868942567
Epoch: 84 Loss: 0.43994980979231046
Epoch: 85 Loss: 0.441245640598642
Epoch: 86 Loss: 0.4404083667405307
Epoch: 87 Loss: 0.4374852302658249
Epoch: 88 Loss: 0.44138785308306905
Epoch: 89 Loss: 0.43953079279427504
Epoch: 90 Loss: 0.533885301733707
Epoch: 91 Loss: 0.43385462680769227
Epoch: 92 Loss: 0.4242095802951783
Epoch: 93 Loss: 0.4425386663385481
Epoch: 94 Loss: 0.43808660211189737
Epoch: 95 Loss: 0.446521971296258
Epoch: 96 Loss: 0.43138106302693346
Epoch: 97 Loss: 0.45042909487448113
Epoch: 98 Loss: 0.42743681188828003
Epoch: 99 Loss: 0.42692896999028335
Epoch: 100 Loss: 0.43509291808798234
```

Figure 2.3: Loss per epoch

The minimum loss that we attain is 0.4269.

Chapter 3

Solution for Question 3

To show neural network with linear activation is with many layer is equivalent to linear neural neural network with no hidden layer.
Proof.

Let a output of a feed forward network with n layers and activation function σ_i at the i^{th} layer be given by

$$Network = W_n(\sigma_n(W_{n-1}(\sigma_{n-2}\dots(W_2(\sigma_1(W_1 * X + b_1)) + b_2)\dots))$$

give that $\sigma(x) = x$

$$Network = W_n((W_{n-1}(\dots(W_2((W_1 * X + b_1)) + b_2)\dots))$$

The product of all the weights can be written as written as a single weight i.e. $\Pi_i^n W_i = W$ and similarly the product of the weights and biases can be replaced by a single bias i.e. $\Pi_{i=2}^n W_i \times b_{i-1} = B$. Therefore we can write.

$$W_n((W_{n-1}(\dots(W_2((W_1 * X + b_1)) + b_2)\dots)) = WX + B$$
$$Network = WX + B$$

That is the network can be treated as a linear neural neural network with no hidden layer.