DSC 475 - Project 2

Hao Ma

1.1 (a), (b)

Note: The two plots are embedded in accuracy&loss reports.

In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [14]:

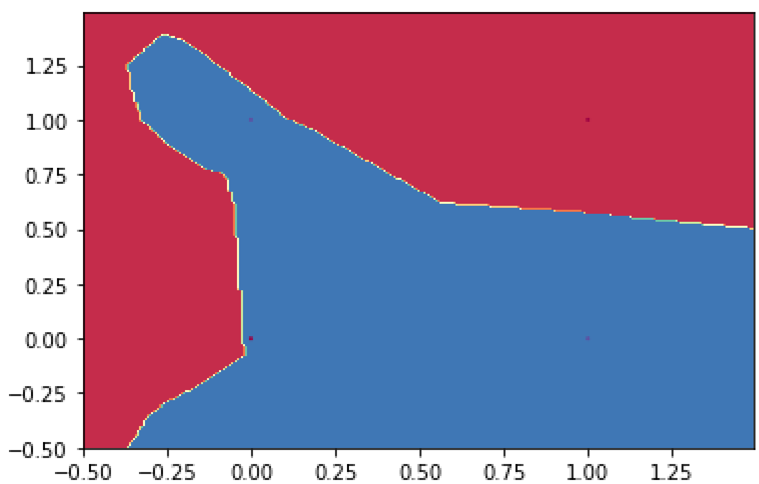
```
class Net(nn.Module):
    # network structure for 1.1
    def __init__(self):
        super(Net, self).__init__()
        # three hidden layers with 20 neurons each
        self.fc1 = nn.Linear(2, 20)
        self.fc2 = nn.Linear(20, 20)
        self.fc3 = nn.Linear(20, 20)
        self.fc4 = nn.Linear(20, 20)
        self.fc5 = nn.Linear(20, 2)

    def forward(self, x):
        # choice of activation function: relu
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)
        return F.log_softmax(x)


def plot_decision_boundary(net, X, y):
    # A function to plot the decision boundary
    x_min, x_max = -0.5, 1.5
    y_min, y_max = -0.5, 1.5
    h = 0.01
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h
))
    # Predict the function value for the whole grid
    X_out = net(torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype = torch.floa
t))
    Z = X_out.data.max(1)[1]
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)

#  XOR data (columns names have been added in the csv file)
data = pd.read_csv("XOR.csv")
X = data.values[:, 0:2]  # Take only the first two features.
X = torch.tensor(X, dtype = torch.float)
y = data.values[:, 2]
y = torch.tensor(y, dtype = torch.long)

# Training
net = Net()
# create a stochastic gradient descent optimizer
learning_rate = 0.05 # for faster convergence
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)

# create a loss function
criterion = nn.NLLLoss()

nepochs = 10000
data, target = X, y
```

```
earliest = True
# run the main training loop
for epoch in range(nepochs):
    optimizer.zero_grad()
    # forward propagate
    net_out = net(data)
    # compute loss
    loss = criterion(net_out, target)
    # backpropagate
    loss.backward()
    # update parameters
    optimizer.step()
    # print out report until 100% accuracy is achieved or
    # the loss falls below 0.0001
    print('Epoch ', epoch, 'Loss ', loss.item())
    pred = net_out.data.max(1)[1]  # get the index of the max log-probability
    correctidx = pred.eq(target.data)
    ncorrect = correctidx.sum()
    accuracy = ncorrect.item()/len(data)
    print('Training accuracy is ', accuracy)

    if accuracy == 1 and earliest:
        # As required, plot the decision boundaries of the earliest network in
the training process
        # that achieves 100% accuracy
        earliest = False
        print('**100% accuracy is achieved here**')
        plot_decision_boundary(net, X, y)
        plt.show()

    if loss.item() < 0.0001:
        # As required, plot the decision boundaries after the loss falls below
0.0001
        print('**Loss falls below 0.0001 here**')
        plot_decision_boundary(net, X, y)
        plt.show()
        break
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:19: UserWarning: Implici
t dimension choice for log_softmax has been deprecated. Change the call to in
clude dim=X as an argument.
```

```
Epoch  0 Loss  0.6945462226867676
Training accuracy is  0.5
Epoch  1 Loss  0.694356381893158
Training accuracy is  0.5
Epoch  2 Loss  0.6940532326698303
Training accuracy is  0.5
Epoch  3 Loss  0.6937108039855957
Training accuracy is  0.5
Epoch  4 Loss  0.6933308839797974
Training accuracy is  0.5
Epoch  5 Loss  0.6929678916931152
Training accuracy is  0.5
Epoch  6 Loss  0.6926510334014893
Training accuracy is  0.5
Epoch  7 Loss  0.6923943758010864
Training accuracy is  0.5
Epoch  8 Loss  0.692197859287262
Training accuracy is  0.75
Epoch  9 Loss  0.6920502185821533
Training accuracy is  0.5
Epoch  10 Loss  0.6919325590133667
Training accuracy is  0.5
Epoch  11 Loss  0.6918235421180725
Training accuracy is  0.5
Epoch  12 Loss  0.6917003393173218
Training accuracy is  0.5
Epoch  13 Loss  0.6915504336357117
Training accuracy is  0.5
Epoch  14 Loss  0.6913648247718811
Training accuracy is  0.5
Epoch  15 Loss  0.6911377310752869
Training accuracy is  0.5
Epoch  16 Loss  0.6909124255180359
Training accuracy is  0.5
Epoch  17 Loss  0.6906532049179077
Training accuracy is  0.5
Epoch  18 Loss  0.6903915405273438
Training accuracy is  0.5
Epoch  19 Loss  0.6900997161865234
Training accuracy is  0.5
Epoch  20 Loss  0.6897867918014526
Training accuracy is  0.5
Epoch  21 Loss  0.6894621849060059
Training accuracy is  0.5
Epoch  22 Loss  0.6891050934791565
Training accuracy is  0.75
Epoch  23 Loss  0.6887248158454895
Training accuracy is  0.75
Epoch  24 Loss  0.6883214116096497
Training accuracy is  0.75
Epoch  25 Loss  0.6880104541778564
Training accuracy is  0.75
Epoch  26 Loss  0.6876884698867798
Training accuracy is  0.75
Epoch  27 Loss  0.6873420476913452
Training accuracy is  0.75
Epoch  28 Loss  0.6869895458221436
```

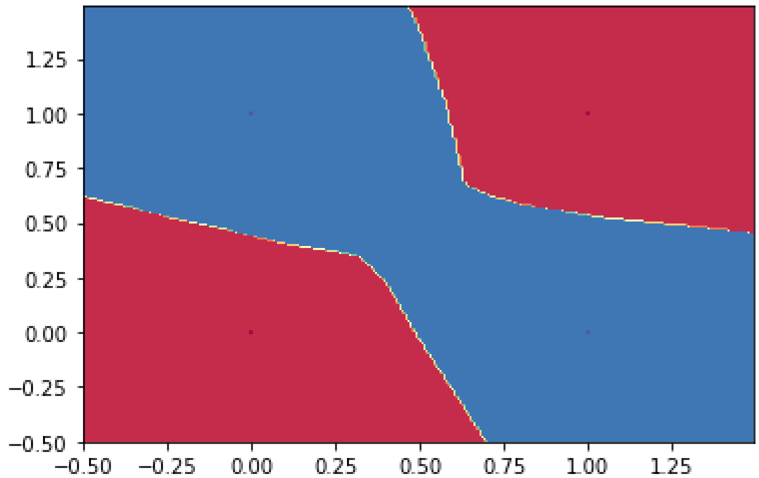
```
Training accuracy is  0.75
Epoch  29 Loss  0.6866284012794495
Training accuracy is  0.75
Epoch  30 Loss  0.6862086057662964
Training accuracy is  0.75
Epoch  31 Loss  0.6857434511184692
Training accuracy is  0.75
Epoch  32 Loss  0.6852546334266663
Training accuracy is  0.75
Epoch  33 Loss  0.6847296953201294
Training accuracy is  0.75
Epoch  34 Loss  0.6841648817062378
Training accuracy is  0.75
Epoch  35 Loss  0.6835817098617554
Training accuracy is  0.75
Epoch  36 Loss  0.6829493641853333
Training accuracy is  0.75
Epoch  37 Loss  0.6822612285614014
Training accuracy is  1.0
**100% accuracy is achieved here**
```



```
Epoch  38 Loss  0.6815095543861389
Training accuracy is  0.75
Epoch  39 Loss  0.6806870698928833
Training accuracy is  0.75
Epoch  40 Loss  0.6797855496406555
Training accuracy is  0.75
Epoch  41 Loss  0.6789296865463257
Training accuracy is  0.75
Epoch  42 Loss  0.6778762340545654
Training accuracy is  0.75
Epoch  43 Loss  0.676721453666687
Training accuracy is  0.75
Epoch  44 Loss  0.6755290627479553
Training accuracy is  0.75
Epoch  45 Loss  0.674177169799804 7
Training accuracy is  0.75
Epoch  46 Loss  0.6726700663566589
Training accuracy is  0.75
Epoch  47 Loss  0.6710194945335388
Training accuracy is  0.75
Epoch  48 Loss  0.6692386269569397
Training accuracy is  0.75
Epoch  49 Loss  0.6672821044921875
Training accuracy is  0.75
Epoch  50 Loss  0.6650102138519287
Training accuracy is  0.75
Epoch  51 Loss  0.6625635027885437
Training accuracy is  0.75
Epoch  52 Loss  0.6598827242851257
Training accuracy is  0.75
Epoch  53 Loss  0.6569311618804932
Training accuracy is  0.75
Epoch  54 Loss  0.6536464095115662
Training accuracy is  0.75
Epoch  55 Loss  0.6499893665313721
Training accuracy is  0.75
Epoch  56 Loss  0.6459165811538696
Training accuracy is  0.75
Epoch  57 Loss  0.6410896182060242
Training accuracy is  0.75
Epoch  58 Loss  0.6355979442596436
Training accuracy is  0.75
Epoch  59 Loss  0.6294361352920532
Training accuracy is  0.75
Epoch  60 Loss  0.6226579546928406
Training accuracy is  0.75
Epoch  61 Loss  0.616114616394043
Training accuracy is  0.75
Epoch  62 Loss  0.607833206653595
Training accuracy is  0.75
Epoch  63 Loss  0.5985419750213623
Training accuracy is  0.75
Epoch  64 Loss  0.5876613259315491
Training accuracy is  0.75
Epoch  65 Loss  0.5751106142997742
Training accuracy is  0.75
Epoch  66 Loss  0.560251772403717
```

```
Training accuracy is  0.75
Epoch  67 Loss  0.5439274907112122
Training accuracy is  0.75
Epoch  68 Loss  0.5251119136810303
Training accuracy is  0.75
Epoch  69 Loss  0.5049505233764648
Training accuracy is  0.75
Epoch  70 Loss  0.4819396436214447
Training accuracy is  0.75
Epoch  71 Loss  0.456208199262619
Training accuracy is  1.0
Epoch  72 Loss  0.42637455463409424
Training accuracy is  1.0
Epoch  73 Loss  0.393406480550766
Training accuracy is  1.0
Epoch  74 Loss  0.35736820101737976
Training accuracy is  1.0
Epoch  75 Loss  0.31923383474349976
Training accuracy is  1.0
Epoch  76 Loss  0.27747732400894165
Training accuracy is  1.0
Epoch  77 Loss  0.23408086597919464
Training accuracy is  1.0
Epoch  78 Loss  0.18903197348117828
Training accuracy is  1.0
Epoch  79 Loss  0.14502406120300293
Training accuracy is  1.0
Epoch  80 Loss  0.10646328330039978
Training accuracy is  1.0
Epoch  81 Loss  0.07477398216724396
Training accuracy is  1.0
Epoch  82 Loss  0.05193035304546356
Training accuracy is  1.0
Epoch  83 Loss  0.035627152770757675
Training accuracy is  1.0
Epoch  84 Loss  0.024248354136943817
Training accuracy is  1.0
Epoch  85 Loss  0.01676982454955578
Training accuracy is  1.0
Epoch  86 Loss  0.011914549395442009
Training accuracy is  1.0
Epoch  87 Loss  0.008497071452438831
Training accuracy is  1.0
Epoch  88 Loss  0.006136741489171982
Training accuracy is  1.0
Epoch  89 Loss  0.004476932343095541
Training accuracy is  1.0
Epoch  90 Loss  0.0033141691237688065
Training accuracy is  1.0
Epoch  91 Loss  0.002492073690518737
Training accuracy is  1.0
Epoch  92 Loss  0.0019027431262657046
Training accuracy is  1.0
Epoch  93 Loss  0.0014740444021299481
Training accuracy is  1.0
Epoch  94 Loss  0.001162998960353434
Training accuracy is  1.0
```

```
Epoch  95 Loss  0.0009316123323515058
Training accuracy is  1.0
Epoch  96 Loss  0.0007578188669867814
Training accuracy is  1.0
Epoch  97 Loss  0.0006256877677515149
Training accuracy is  1.0
Epoch  98 Loss  0.0005241867038421333
Training accuracy is  1.0
Epoch  99 Loss  0.00044528322177939117
Training accuracy is  1.0
Epoch  100 Loss  0.0003832040820270777
Training accuracy is  1.0
Epoch  101 Loss  0.0003339606919325888
Training accuracy is  1.0
Epoch  102 Loss  0.00029463559621945024
Training accuracy is  1.0
Epoch  103 Loss  0.0002626081695780158
Training accuracy is  1.0
Epoch  104 Loss  0.00023644912289455533
Training accuracy is  1.0
Epoch  105 Loss  0.00021484798344317824
Training accuracy is  1.0
Epoch  106 Loss  0.00019685158622451127
Training accuracy is  1.0
Epoch  107 Loss  0.0001817450684029609
Training accuracy is  1.0
Epoch  108 Loss  0.0001689326309133321
Training accuracy is  1.0
Epoch  109 Loss  0.00015808662283234298
Training accuracy is  1.0
Epoch  110 Loss  0.00014876014029141515
Training accuracy is  1.0
Epoch  111 Loss  0.0001407446397934109
Training accuracy is  1.0
Epoch  112 Loss  0.00013380181917455047
Training accuracy is  1.0
Epoch  113 Loss  0.0001277528645005077
Training accuracy is  1.0
Epoch  114 Loss  0.00012241902004461735
Training accuracy is  1.0
Epoch  115 Loss  0.00011780030035879463
Training accuracy is  1.0
Epoch  116 Loss  0.00011374773748684675
Training accuracy is  1.0
Epoch  117 Loss  0.00011017193173756823
Training accuracy is  1.0
Epoch  118 Loss  0.00010695368837332353
Training accuracy is  1.0
Epoch  119 Loss  0.00010409301467007026
Training accuracy is  1.0
Epoch  120 Loss  0.00010150053276447579
Training accuracy is  1.0
Epoch  121 Loss  9.920603042701259e-05
Training accuracy is  1.0
**Loss falls below 0.0001 here**
```

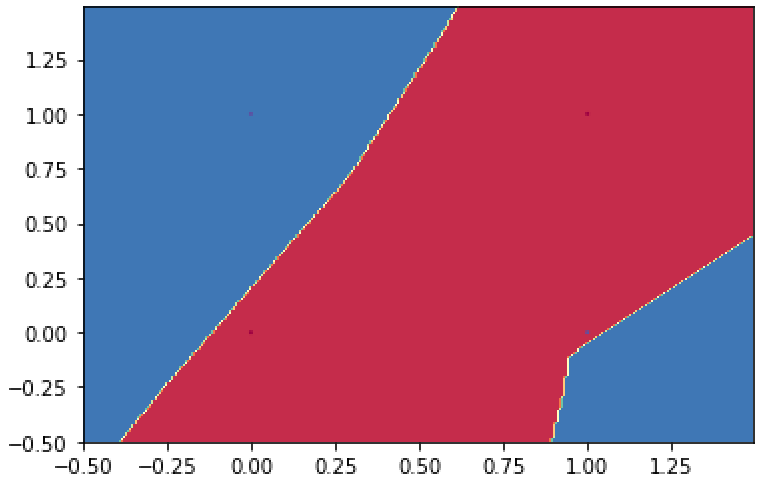


1.2 (a), (b)

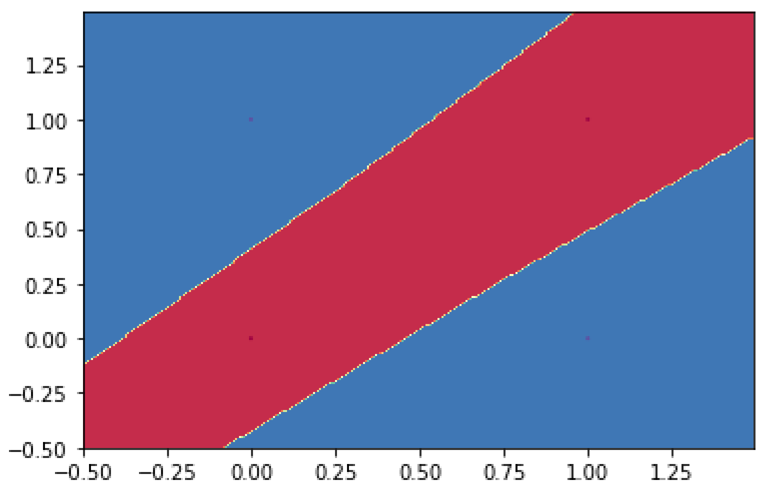
In [16]:
```
class Net(nn.Module):
    # network structure for 1.2: 1 hidden layer with 5 neurons
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 5)
        self.fc2 = nn.Linear(5, 5)
        self.fc3 = nn.Linear(5, 2)

    def forward(self, x):
        # choice of activation function: relu
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x)


net = Net()
learning_rate = 0.05 # for faster convergence
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
criterion = nn.NLLLoss()

nepochs = 30000
data, target = X, y
earliest = True
for epoch in range(nepochs):
    optimizer.zero_grad()
    # forward propagate
    net_out = net(data)
    # compute loss
    loss = criterion(net_out, target)
    # backpropagate
    loss.backward()
    # update parameters
    optimizer.step()

    pred = net_out.data.max(1)[1]
    correctidx = pred.eq(target.data)
    ncorrect = correctidx.sum()
    accuracy = ncorrect.item()/len(data)

    if accuracy == 1 and earliest:
        # As required, plot the decision boundaries of the earliest network in
the training process
        # that achieves 100% accuracy
        earliest = False
        print('**100% accuracy is achieved here**')
        print('Epoch ', epoch, 'Loss ', loss.item())
        print('Training accuracy is ', accuracy)
        plot_decision_boundary(net, X, y)
        plt.show()

    if loss.item() < 0.0001:
        # As required, plot the decision boundaries after the loss falls below
0.0001
        print('**Loss falls below 0.0001 here**')
        print('Epoch ', epoch, 'Loss ', loss.item())
```

```
        print('Training accuracy is ', accuracy)
        plot_decision_boundary(net, X, y)
        plt.show()
        break
```

```
**100% accuracy is achieved here**
Epoch  24 Loss  0.6558001041412354
Training accuracy is  1.0

C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
```




```
**Loss falls below 0.0001 here**
Epoch  1730 Loss  9.994937136070803e-05
Training accuracy is  1.0
```

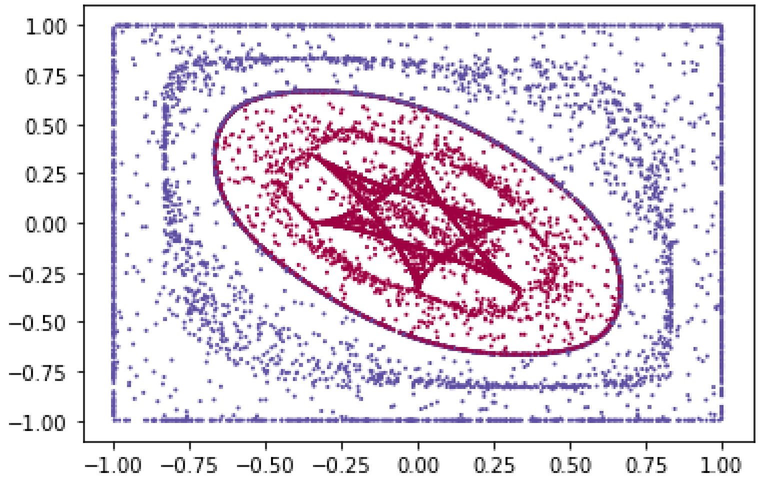


The smallest network that can achieve 100% accuracy (with reasonable number of iterations) contains 1 hidden layer with 5 neurons.

2.1

In [22]:
```
mydata = pd.read_csv('FeedForward_Data_ellipse.csv', header=None)
X = mydata.values[:, 0:2] # features
X = torch.tensor(X, dtype = torch.float)
y = mydata.values[:, 2] # labels
y = torch.tensor(y, dtype = torch.long)

# plot the original data
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
```

Out[22]: <matplotlib.collections.PathCollection at 0x29df2874508>



In [24]:
```
class Net(nn.Module):
    # network structure for 2.1
    def __init__(self):
        super(Net, self).__init__()
        # 10 hidden layers with 20 neurons each
        self.fc1 = nn.Linear(2, 20)
        self.fc2 = nn.Linear(20, 20)
        self.fc3 = nn.Linear(20, 20)
        self.fc4 = nn.Linear(20, 20)
        self.fc5 = nn.Linear(20, 20)
        self.fc6 = nn.Linear(20, 20)
        self.fc7 = nn.Linear(20, 20)
        self.fc8 = nn.Linear(20, 20)
        self.fc9 = nn.Linear(20, 20)
        self.fc10 = nn.Linear(20, 20)
        self.fc11 = nn.Linear(20, 20)
        self.fc12 = nn.Linear(20, 2)


    def forward(self, x):
        # choice of activation function: relu
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = F.relu(self.fc5(x))
        x = F.relu(self.fc6(x))
        x = F.relu(self.fc7(x))
        x = F.relu(self.fc8(x))
        x = F.relu(self.fc9(x))
        x = F.relu(self.fc10(x))
        x = F.relu(self.fc11(x))
        x = self.fc12(x)
        return F.log_softmax(x)


def plot_decision_boundary(net, X, y):
    # A function to plot the decision boundary
    x_min, x_max = -1.0, 1.0
    y_min, y_max = -1.0, 1.0
    h = 0.01
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h
))
    # Predict the function value for the whole grid
    X_out = net(torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype = torch.floa
t))
    Z = X_out.data.max(1)[1]
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)

# Training
net = Net()
learning_rate = 0.1
```

```
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
criterion = nn.NLLLoss()

nepochs = 5001
data, target = X, y
for epoch in range(nepochs):
    optimizer.zero_grad()
    # forward propagate
    net_out = net(data)
    # compute loss
    loss = criterion(net_out, target)
    # backpropagate
    loss.backward()
    # update parameters
    optimizer.step()

    pred = net_out.data.max(1)[1]
    correctidx = pred.eq(target.data)
    ncorrect = correctidx.sum()
    accuracy = ncorrect.item()/len(data)

    if epoch % 50 == 0:
        # print out report every 50 iterations
        print('Epoch ', epoch, 'Loss ', loss.item())
        print('Training accuracy is ', accuracy)
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:34: UserWarning: Implici
t dimension choice for log_softmax has been deprecated. Change the call to in
clude dim=X as an argument.
```

```
Epoch  0 Loss  0.7107270956039429
Training accuracy is  0.4434344951923077
Epoch  50 Loss  0.6867424845695496
Training accuracy is  0.5565655048076923
Epoch  100 Loss  0.6867023706436157
Training accuracy is  0.5565655048076923
Epoch  150 Loss  0.686763346195221
Training accuracy is  0.5565655048076923
Epoch  200 Loss  0.6866981983184814
Training accuracy is  0.5565655048076923
Epoch  250 Loss  0.686698853969574
Training accuracy is  0.5565655048076923
Epoch  300 Loss  0.6866989135742188
Training accuracy is  0.5565655048076923
Epoch  350 Loss  0.686698853969574
Training accuracy is  0.5565655048076923
Epoch  400 Loss  0.686698853969574
Training accuracy is  0.5565655048076923
Epoch  450 Loss  0.686698853969574
Training accuracy is  0.5565655048076923
Epoch  500 Loss  0.6866987943649292
Training accuracy is  0.5565655048076923
Epoch  550 Loss  0.6866986751556396
Training accuracy is  0.5565655048076923
Epoch  600 Loss  0.6866986751556396
Training accuracy is  0.5565655048076923
Epoch  650 Loss  0.6866985559463501
Training accuracy is  0.5565655048076923
Epoch  700 Loss  0.6866984963417053
Training accuracy is  0.5565655048076923
Epoch  750 Loss  0.6866984963417053
Training accuracy is  0.5565655048076923
Epoch  800 Loss  0.6866983771324158
Training accuracy is  0.5565655048076923
Epoch  850 Loss  0.6866983771324158
Training accuracy is  0.5565655048076923
Epoch  900 Loss  0.686698317527771
Training accuracy is  0.5565655048076923
Epoch  950 Loss  0.686698317527771
Training accuracy is  0.5565655048076923
Epoch  1000 Loss  0.686698317527771
Training accuracy is  0.5565655048076923
Epoch  1050 Loss  0.6866982579231262
Training accuracy is  0.5565655048076923
Epoch  1100 Loss  0.6866981983184814
Training accuracy is  0.5565655048076923
Epoch  1150 Loss  0.6866981983184814
Training accuracy is  0.5565655048076923
Epoch  1200 Loss  0.6866980791091919
Training accuracy is  0.5565655048076923
Epoch  1250 Loss  0.6866980791091919
Training accuracy is  0.5565655048076923
Epoch  1300 Loss  0.6866980791091919
Training accuracy is  0.5565655048076923
Epoch  1350 Loss  0.6866980791091919
Training accuracy is  0.5565655048076923
Epoch  1400 Loss  0.6866980791091919
```

```
Training accuracy is  0.5565655048076923
Epoch  1450 Loss  0.6866980195045471
Training accuracy is  0.5565655048076923
Epoch  1500 Loss  0.6866980195045471
Training accuracy is  0.5565655048076923
Epoch  1550 Loss  0.6866980195045471
Training accuracy is  0.5565655048076923
Epoch  1600 Loss  0.6866980195045471
Training accuracy is  0.5565655048076923
Epoch  1650 Loss  0.686698317527771
Training accuracy is  0.5565655048076923
Epoch  1700 Loss  0.6866986155509949
Training accuracy is  0.5565655048076923
Epoch  1750 Loss  0.686698853969574
Training accuracy is  0.5565655048076923
Epoch  1800 Loss  0.6866990327835083
Training accuracy is  0.5565655048076923
Epoch  1850 Loss  0.6866992712020874
Training accuracy is  0.5565655048076923
Epoch  1900 Loss  0.6866998076438904
Training accuracy is  0.5565655048076923
Epoch  1950 Loss  0.6867008805274963
Training accuracy is  0.5565655048076923
Epoch  2000 Loss  0.6867018342018127
Training accuracy is  0.5565655048076923
Epoch  2050 Loss  0.6867029666900635
Training accuracy is  0.5565655048076923
Epoch  2100 Loss  0.6867037415504456
Training accuracy is  0.5565655048076923
Epoch  2150 Loss  0.68670654296875
Training accuracy is  0.5565655048076923
Epoch  2200 Loss  0.6867085099220276
Training accuracy is  0.5565655048076923
Epoch  2250 Loss  0.6867102980613708
Training accuracy is  0.5565655048076923
Epoch  2300 Loss  0.6867122650146484
Training accuracy is  0.5565655048076923
Epoch  2350 Loss  0.686714231967926
Training accuracy is  0.5565655048076923
Epoch  2400 Loss  0.6867154240608215
Training accuracy is  0.5565655048076923
Epoch  2450 Loss  0.6867167353630066
Training accuracy is  0.5565655048076923
Epoch  2500 Loss  0.6867179870605469
Training accuracy is  0.5565655048076923
Epoch  2550 Loss  0.6867192983627319
Training accuracy is  0.5565655048076923
Epoch  2600 Loss  0.6867207884788513
Training accuracy is  0.5565655048076923
Epoch  2650 Loss  0.6867217421531677
Training accuracy is  0.5565655048076923
Epoch  2700 Loss  0.6867228746414185
Training accuracy is  0.5565655048076923
Epoch  2750 Loss  0.6867242455482483
Training accuracy is  0.5565655048076923
Epoch  2800 Loss  0.6867255568504333
Training accuracy is  0.5565655048076923
```

```
Epoch  2850 Loss  0.6867268085479736
Training accuracy is  0.5565655048076923
Epoch  2900 Loss  0.6867278814315796
Training accuracy is  0.5565655048076923
Epoch  2950 Loss  0.6867291331291199
Training accuracy is  0.5565655048076923
Epoch  3000 Loss  0.6867303252220154
Training accuracy is  0.5565655048076923
Epoch  3050 Loss  0.6867312788963318
Training accuracy is  0.5565655048076923
Epoch  3100 Loss  0.6867325901985168
Training accuracy is  0.5565655048076923
Epoch  3150 Loss  0.6867333650588989
Training accuracy is  0.5565655048076923
Epoch  3200 Loss  0.6867353916168213
Training accuracy is  0.5565655048076923
Epoch  3250 Loss  0.6867369413375854
Training accuracy is  0.5565655048076923
Epoch  3300 Loss  0.6867368221282959
Training accuracy is  0.5565655048076923
Epoch  3350 Loss  0.6867377161979675
Training accuracy is  0.5565655048076923
Epoch  3400 Loss  0.6867396235466003
Training accuracy is  0.5565655048076923
Epoch  3450 Loss  0.6867396831512451
Training accuracy is  0.5565655048076923
Epoch  3500 Loss  0.6867386102676392
Training accuracy is  0.5565655048076923
Epoch  3550 Loss  0.686737596988678
Training accuracy is  0.5565655048076923
Epoch  3600 Loss  0.6867360472679138
Training accuracy is  0.5565655048076923
Epoch  3650 Loss  0.6867346167564392
Training accuracy is  0.5565655048076923
Epoch  3700 Loss  0.6867259740829468
Training accuracy is  0.5565655048076923
Epoch  3750 Loss  0.6867111921310425
Training accuracy is  0.5565655048076923
Epoch  3800 Loss  0.6866973042488098
Training accuracy is  0.5565655048076923
Epoch  3850 Loss  0.6866486072540283
Training accuracy is  0.5565655048076923
Epoch  3900 Loss  0.6865653395652771
Training accuracy is  0.5565655048076923
Epoch  3950 Loss  0.6864850521087646
Training accuracy is  0.5565655048076923
Epoch  4000 Loss  0.6862711310386658
Training accuracy is  0.5565655048076923
Epoch  4050 Loss  0.6855273842811584
Training accuracy is  0.5565655048076923
Epoch  4100 Loss  0.6563113331794739
Training accuracy is  0.5565655048076923
Epoch  4150 Loss  0.336140900850296
Training accuracy is  0.8275240384615384
Epoch  4200 Loss  0.19704784452915192
Training accuracy is  0.8982872596153846
Epoch  4250 Loss  0.3086509704589844
```

```
Training accuracy is  0.8722956730769231
Epoch  4300 Loss  0.2222747653722763
Training accuracy is  0.8900240384615384
Epoch  4350 Loss  0.2006864696741104
Training accuracy is  0.8977614182692307
Epoch  4400 Loss  0.2146114706993103
Training accuracy is  0.8958834134615384
Epoch  4450 Loss  0.1951487958431244
Training accuracy is  0.8981370192307693
Epoch  4500 Loss  0.19641393423080444
Training accuracy is  0.8979867788461539
Epoch  4550 Loss  0.2204727977514267
Training accuracy is  0.8942307692307693
Epoch  4600 Loss  0.1958731710910797
Training accuracy is  0.8974609375
Epoch  4650 Loss  0.20922620594501495
Training accuracy is  0.8915264423076923
Epoch  4700 Loss  0.21865598857402802
Training accuracy is  0.8641826923076923
Epoch  4750 Loss  0.23160573840141296
Training accuracy is  0.8965594951923077
Epoch  4800 Loss  0.1952981799840927
Training accuracy is  0.8982121394230769
Epoch  4850 Loss  0.20291200280189514
Training accuracy is  0.8958834134615384
Epoch  4900 Loss  0.19397442042827606
Training accuracy is  0.8997145432692307
Epoch  4950 Loss  0.19692093133926392
Training accuracy is  0.8975360576923077
Epoch  5000 Loss  0.20486140251159668
Training accuracy is  0.8934795673076923
```

In [26]:
```
# Further training is needed
# Also, the learning rate will be changed to 0.05
# Based on the outputs above, stop training when accuracy reaches 0.899
learning_rate = 0.05
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
for epoch in range(nepochs):
    optimizer.zero_grad()
    net_out = net(data)
    loss = criterion(net_out, target)
    loss.backward()
    optimizer.step()

    pred = net_out.data.max(1)[1]
    correctidx = pred.eq(target.data)
    ncorrect = correctidx.sum()
    accuracy = ncorrect.item()/len(data)
    if accuracy >= 0.899:
        print('Epoch ', epoch, 'Loss ', loss.item())
        print('Training accuracy is ', accuracy)
        break
```

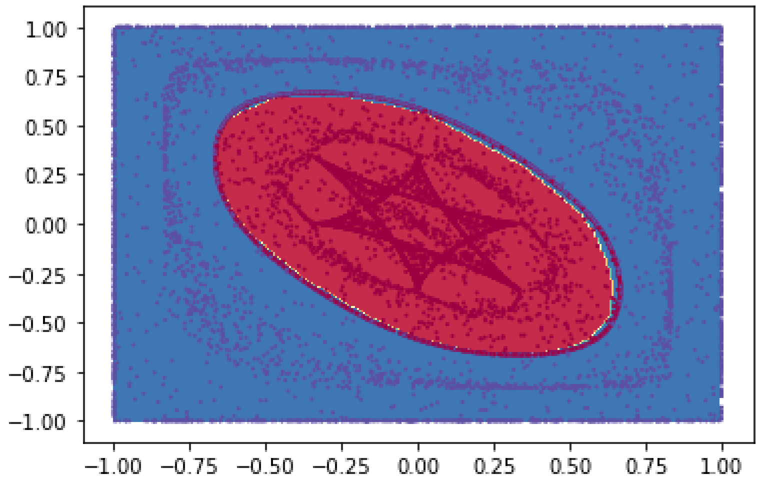
```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:34: UserWarning: Implici
t dimension choice for log_softmax has been deprecated. Change the call to in
clude dim=X as an argument.

Epoch  45 Loss  0.19511112570762634
Training accuracy is  0.8993389423076923
```

Report:

In [28]:
```
print('Accuracy:', accuracy, '\n' + 'Loss:', loss.item())
print('Number of hidden layers: 10; type of activation function: relu; number
 of neurons per layer: 20')

# plot the decision boundary of the network
plot_decision_boundary(net, X, y)
```

```
Accuracy: 0.8993389423076923
Loss: 0.19511112570762634
Number of hidden layers: 10; type of activation function: relu; number of neu
rons per layer: 20
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:34: UserWarning: Implici
t dimension choice for log_softmax has been deprecated. Change the call to in
clude dim=X as an argument.
```

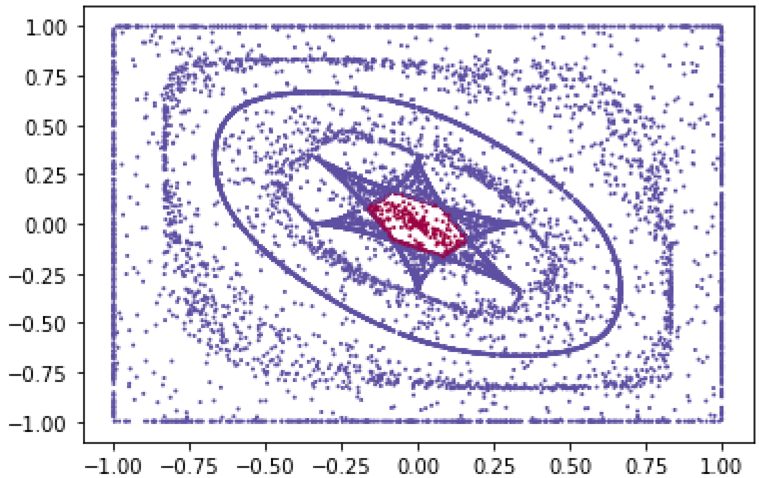


2.2

Important note: This dataset is class-imbalanced, so I am using the oversampling method to prevent the classification result from being dominated by the majority class.

In [3]:
```
mydata2 = pd.read_csv('FeedForward_Data_hexa.csv', header=None)
X = mydata2.values[:, 0:2] # features
y = mydata2.values[:, 2] # labels

# plot the original data
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
```

Out[3]: <matplotlib.collections.PathCollection at 0x1d1043a6e48>



In [4]:
```
# Oversampling the rare cases (y=0) with replacement
# Note that the number of majority cases = 12521 while the number of rare case
s = 791
mydata2.columns = ['x1','x2','y']
sample = mydata2[mydata2['y'] == 0].sample(n=11730, replace=True)
# add the new samples to X and y
new_X = np.concatenate((X, sample.values[:, 0:2]))
new_y = np.concatenate((y, sample.values[:, 2]))
new_X = torch.tensor(new_X, dtype = torch.float)
new_y = torch.tensor(new_y, dtype = torch.long)
```

In [8]:
```
class Net(nn.Module):
    # network structure for 2.2
    def __init__(self):
        super(Net, self).__init__()
        # 10 hidden layers with 30 neurons each
        self.fc1 = nn.Linear(2, 30)
        self.fc2 = nn.Linear(30, 30)
        self.fc3 = nn.Linear(30, 30)
        self.fc4 = nn.Linear(30, 30)
        self.fc5 = nn.Linear(30, 30)
        self.fc6 = nn.Linear(30, 30)
        self.fc7 = nn.Linear(30, 30)
        self.fc8 = nn.Linear(30, 30)
        self.fc9 = nn.Linear(30, 30)
        self.fc10 = nn.Linear(30, 30)
        self.fc11 = nn.Linear(30, 30)
        self.fc12 = nn.Linear(30, 2)

    def forward(self, x):
        # choice of activation function: relu, tanh
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = torch.relu(self.fc5(x))
        x = torch.relu(self.fc6(x))
        x = torch.tanh(self.fc7(x))
        x = torch.tanh(self.fc8(x))
        x = torch.tanh(self.fc9(x))
        x = torch.tanh(self.fc10(x))
        x = torch.tanh(self.fc11(x))
        x = self.fc12(x)
        return F.log_softmax(x)

# Training
net = Net()
learning_rate = 0.05
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
criterion = nn.NLLLoss()
nepochs = 10001
data, target = new_X, new_y
for epoch in range(nepochs):
    optimizer.zero_grad()
    # forward propagate
    net_out = net(data)
    # compute loss
    loss = criterion(net_out, target)
    # backpropagate
    loss.backward()
    # update parameters
    optimizer.step()

    pred = net_out.data.max(1)[1]
    correctidx = pred.eq(target.data)
    ncorrect = correctidx.sum()
    accuracy = ncorrect.item()/len(data)
```

```
    if accuracy >= 0.95:
        print('Epoch ', epoch, 'Loss ', loss.item())
        print('Training accuracy is ', accuracy)
        break
    if epoch % 100 == 0:
        print('Epoch ', epoch, 'Loss ', loss.item())
        print('Training accuracy is ', accuracy)
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:33: UserWarning: Implici
t dimension choice for log_softmax has been deprecated. Change the call to in
clude dim=X as an argument.
```

```
Epoch  0 Loss  0.6978422999382019
Training accuracy is  0.5
Epoch  100 Loss  0.6930481195449829
Training accuracy is  0.5
Epoch  200 Loss  0.693304181098938
Training accuracy is  0.6009504033224183
Epoch  300 Loss  0.6933044791221619
Training accuracy is  0.5792668317227059
Epoch  400 Loss  0.6933032870292664
Training accuracy is  0.592364827090488
Epoch  500 Loss  0.6933022141456604
Training accuracy is  0.5911668397092884
Epoch  600 Loss  0.6933016777038574
Training accuracy is  0.5909671751457551
Epoch  700 Loss  0.6932948231697083
Training accuracy is  0.6063812794505231
Epoch  800 Loss  0.6932891011238098
Training accuracy is  0.6241913585176903
Epoch  900 Loss  0.6932847499847412
Training accuracy is  0.6409631818544844
Epoch  1000 Loss  0.6932802200317383
Training accuracy is  0.6423209008865106
Epoch  1100 Loss  0.6932776570320129
Training accuracy is  0.6558980912067727
Epoch  1200 Loss  0.6932759881019592
Training accuracy is  0.6766632058142321
Epoch  1300 Loss  0.6932615637779236
Training accuracy is  0.6968692596437984
Epoch  1400 Loss  0.6932504177093506
Training accuracy is  0.7015414104304768
Epoch  1500 Loss  0.6930919885635376
Training accuracy is  0.7129622234645795
Epoch  1600 Loss  0.6928747892379761
Training accuracy is  0.739357878763677
Epoch  1700 Loss  0.6926199197769165
Training accuracy is  0.7467454676144077
Epoch  1800 Loss  0.6909545660018921
Training accuracy is  0.7634773580384954
Epoch  1900 Loss  0.7013331055641174
Training accuracy is  0.5
Epoch  2000 Loss  0.6931933164596558
Training accuracy is  0.5
Epoch  2100 Loss  0.6932984590530396
Training accuracy is  0.8106381279450523
Epoch  2200 Loss  0.6932749152183533
Training accuracy is  0.7208290072677901
Epoch  2300 Loss  0.693254292011261
Training accuracy is  0.6902004632217874
Epoch  2400 Loss  0.6932433247566223
Training accuracy is  0.7002635572238639
Epoch  2500 Loss  0.6932215094566345
Training accuracy is  0.7522562095679259
Epoch  2600 Loss  0.6931699514389038
Training accuracy is  0.7896733487740596
Epoch  2700 Loss  0.6928461790084839
Training accuracy is  0.8253733727338072
Epoch  2800 Loss  0.6914950609207153
```

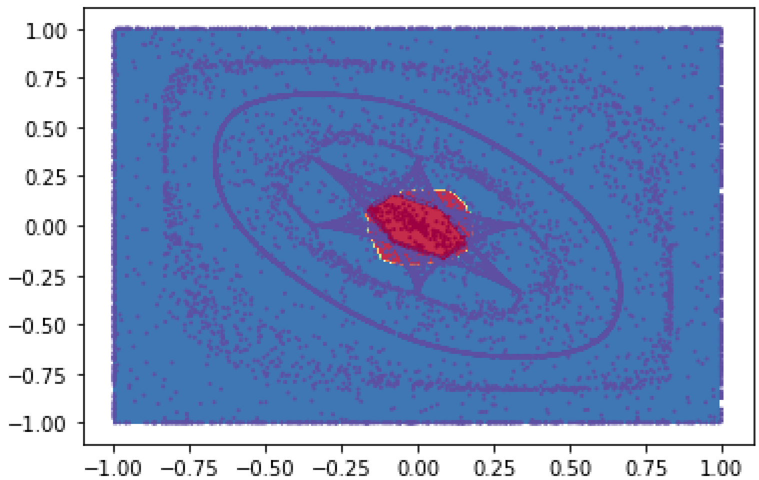
```
Training accuracy is  0.8367941857679099
Epoch  2900 Loss  0.7084758281707764
Training accuracy is  0.5
Epoch  3000 Loss  0.7805532217025757
Training accuracy is  0.5
Epoch  3100 Loss  0.693059504032135
Training accuracy is  0.5
Epoch  3200 Loss  0.6932235956192017
Training accuracy is  0.694273620317866
Epoch  3300 Loss  0.6929205060005188
Training accuracy is  0.7138806804568325
Epoch  3400 Loss  0.6925575733184814
Training accuracy is  0.7237840428080824
Epoch  3500 Loss  0.6717435121536255
Training accuracy is  0.7238639086334957
Epoch  3600 Loss  0.4430851340293884
Training accuracy is  0.7966216755850172
Epoch  3700 Loss  0.5165365934371948
Training accuracy is  0.8292069323536458
Epoch  3800 Loss  0.2585148811340332
Training accuracy is  0.9193754492452679
Epoch  3811 Loss  0.16301041841506958
Training accuracy is  0.95204057183931
```

Report:

In [9]:
```python
print('Accuracy:', accuracy, 'Loss:', loss.item())
print('I used 10 hidden layers with 30 neurons per layer.')
print('Type of activation functions: relu and tanh.')

# plot the decision boundary of the network (using the original X and y)
plot_decision_boundary(net, X, y)
```

```
Accuracy: 0.95204057183931 Loss: 0.16301041841506958
I used 10 hidden layers with 30 neurons per layer.
Type of activation functions: relu and tanh.
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:33: UserWarning: Implici
t dimension choice for log_softmax has been deprecated. Change the call to in
clude dim=X as an argument.
```



At first, my network could achieve 94% accuracy very quickly, and that was 'fake' since the network did nothing but classified all the points as '1'. Then I applied oversampling method and an accuracy value significantly higher than 50% should be acceptable because it is actually separating the data points. Based on the plot above, I think it did a good job.