DSC 475: Time Series Analysis and Forecasting

Project 2 – Feedforward Neural Networks

Jingwen Zhong

In this project, you will create and train feedforward neural networks on synthetic, separable data. Although traditional machine learning tasks involve measuring performance on sets of data not seen by the network during training (e.g., test data), in this project we will mostly be concerned with architecture, hyperparameter and learning parameter tuning to maximize performance on the training set. This is an often-overlooked aspect of machine learning: practitioners seldom verify their algorithms have the right capacity for the data by examining their behavior on the training set. In this project, all performance metrics of interest will be computed on the training data.

In order to construct networks that are capable of performing the tasks being posed, you will need to adjust parameters such as learning rate, type of network activation function, number of layers and number of neurons per layer. Lastly, you will need to monitor the loss and the accuracy on the training set as the learning takes place.

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
import math
```

1. Classification of XOR data

At first sight, it may seem as if separating the XOR data is a simple task. However, due to the fact that the data is not linearly separable and that fitting the data requires non-trivial learning, the XOR problem has been a case study of interest on many topics related to training of feedforward networks. In the 1960s, Minsky and Papert's observations that perceptrons (neural network ancestors) were unable to fit the XOR data contributed to the rise of the first AI winter. XOR data makes for such an interesting case study that papers describing learning properties of networks trained on it are still being published to this day.

1.1. Create arrays containing the input data and the corresponding output labels for the XOR operator. Recall that XOR takes as input two binary variables, and outputs a 0/1 if they have the same/different value. Your XOR data should look like the following table:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

In other words, the XOR training includes four, two-dimensional data samples with labels. Create and train a network with at least three hidden layers that separates the XOR data, that is, a network that gets 100% performance on the four training samples above. Monitor the accuracy on the training set as the training progresses.

In [2]:
```
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)
```

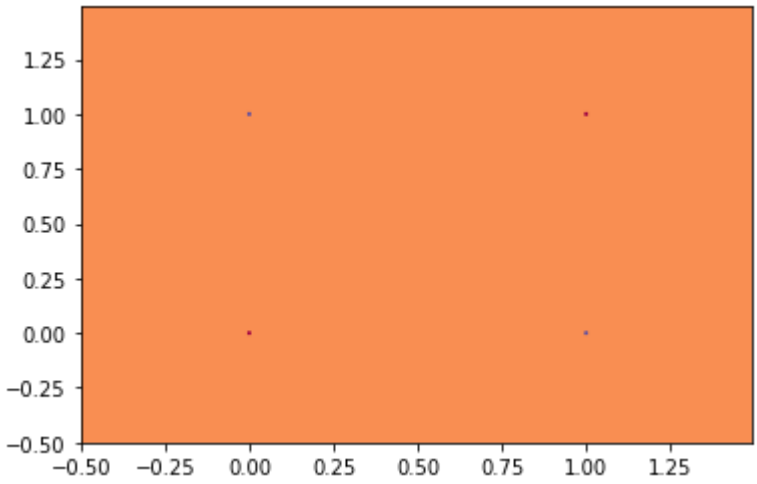
In [3]:
```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        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, 2)

    def forward(self, x):
        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 = self.fc6(x)
        return F.log_softmax(x)
        #return F.softmax(x)

#%% plot function

def plot_data(X, y, filename):
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
    plt.savefig(filename)
    plt.close()
```

(a) Plot the decision boundaries of the earliest network in the training process that achieves 100% accuracy by plotting the network outputs in a densely sampled region around $[-0.5, 1.5] \times [-0.5, 1.5]$. ***(5 points)***

In [4]:
```
def plot_decision_boundary(clf, X, y):
    # Set min and max values and give it some padding
    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 gid
    #Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    X_out = net(torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype = torch.floa
t))
    Z = X_out.data.max(1)[1]
    # Z.shape
    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)
    plt.show()
```

In [5]:
```
net = Net()
plot_decision_boundary(net, X, y)
```

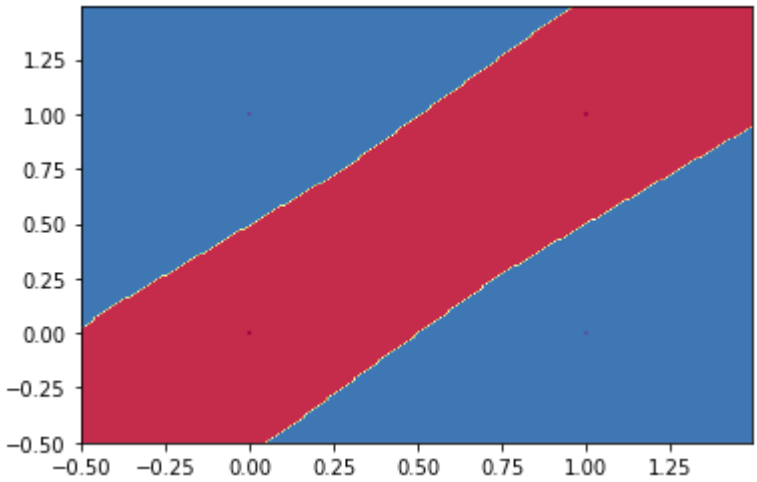
```
C:\Users\Jingwen\anaconda3\lib\site-packages\ipykernel_launcher.py:20: UserWa
rning: Implicit dimension choice for log_softmax has been deprecated. Change
the call to include dim=X as an argument.
```



(b) Plot the decision boundaries of a network after the loss falls below 1×10^{-4}***. (5 points)***

In [6]:
```
net = Net()

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

learning_rate = .01
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)

nepochs = 3000
data, target = X, y
# 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()

    if loss < 1*10**(-4):
        break

plot_decision_boundary(net, X, y)
```

```
C:\Users\Jingwen\anaconda3\lib\site-packages\ipykernel_launcher.py:20: UserWa
rning: Implicit dimension choice for log_softmax has been deprecated. Change
the call to include dim=X as an argument.
```



1.2. Gradually decrease the capacity of the network above. Find the smallest network that can still separate the data.

(a) What are the number of hidden layers and neurons of the smallest network that still can separate the data, i.e. which produces an accuracy of 1 on the training set? (5 points)

```
hidden layers = 1
neurons = 6
```

(b) Plot the decision boundaries of the network as before, that is, right after the network achieves perfect accuracy and after the loss falls below 1×10^{-4}***. (5 points)***

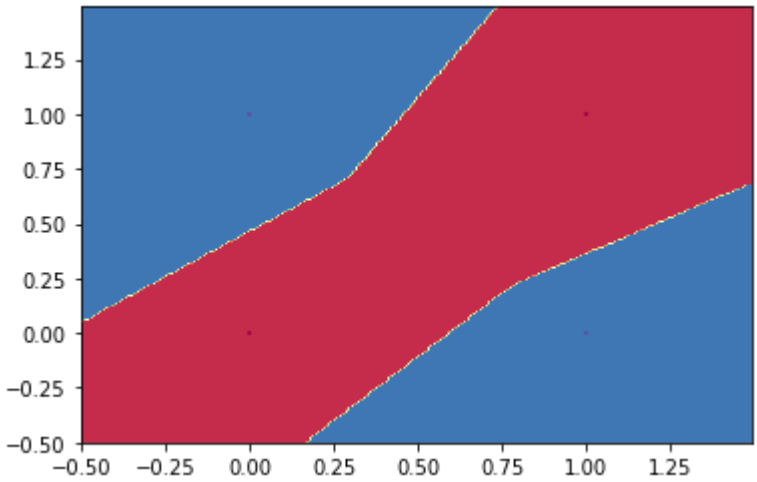
In [7]:
```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 6)
        self.fc2 = nn.Linear(6, 6)
        self.fc6 = nn.Linear(6, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc6(x)
        return F.log_softmax(x)
        #return F.softmax(x)
```

In [8]:
```
#%% train
net = Net()

# create a stochastic gradient descent optimizer
learning_rate = .01
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
#optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)


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

nepochs = 3000
data, target = X, y
# 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

#%% compute accuracy on training data

net_out = net(data)
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)
```

```
C:\Users\Jingwen\anaconda3\lib\site-packages\ipykernel_launcher.py:13: UserWa
rning: Implicit dimension choice for log_softmax has been deprecated. Change
the call to include dim=X as an argument.
  del sys.path[0]

Training accuracy is  1.0
```

In [9]:
```
net = Net()
# create a loss function
criterion = nn.NLLLoss()

learning_rate = .01
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)

nepochs = 3000
data, target = X, y
# 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()

    if loss < 1*10**(-4):
        break

plot_decision_boundary(net, X, y)
```

```
C:\Users\Jingwen\anaconda3\lib\site-packages\ipykernel_launcher.py:13: UserWa
rning: Implicit dimension choice for log_softmax has been deprecated. Change
the call to include dim=X as an argument.
  del sys.path[0]
```



2. Classification of Separable, Synthetic data (15 +15 points)

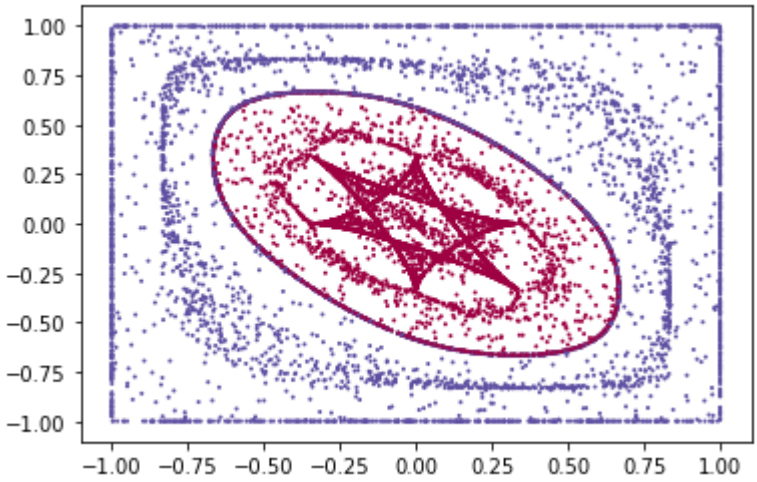
In this section, you will attempt to design a network that is able to classify synthetically created data that is still separable, that is, where no overlap exists between the classes in the native feature space. In this case, however, the tolerances (i.e., the smallest separation between samples in different classes) are much tighter than those seen in the XOR case.

2.1. File Feedforward_Data_ellipse.csv contains 13312 two-dimensional data points (feature values located in columns A and B) and their respective binary label (labels located in column C). Create and train a network that separates the data. Report your best loss and accuracy values. Plot the decision boundaries of your best network by plotting the network outputs in a densely sampled region around $[-1.0, 1.0] \times [-1.0, 1.0]$. Report the number of hidden layers, type of activation function and number of neurons per layer used. (15 points).

```
hidden layers = 15
activation function: relu
number of neurons per layer used: 50
```

In [55]:
```
#%% read data
data=pd.read_csv("FeedForward_Data_ellipse.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)

#plot_data(X,y,'data.pdf')
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
```

Out[55]: <matplotlib.collections.PathCollection at 0x1d72e873088>



In [56]:
```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 50)
        self.fc2 = nn.Linear(50, 50)
        self.fc3 = nn.Linear(50, 50)
        self.fc4 = nn.Linear(50, 50)
        self.fc5 = nn.Linear(50, 50)
        self.fc6 = nn.Linear(50, 50)
        self.fc7 = nn.Linear(50, 50)
        self.fc8 = nn.Linear(50, 50)
        self.fc9 = nn.Linear(50, 50)
        self.fc10 = nn.Linear(50, 50)
        self.fc11 = nn.Linear(50, 50)
        self.fc12 = nn.Linear(50, 50)
        self.fc13 = nn.Linear(50, 50)
        self.fc14 = nn.Linear(50, 50)
        self.fc15 = nn.Linear(50, 50)
        self.fc16 = nn.Linear(50, 2)

    def forward(self, x):
        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 = F.relu(self.fc12(x))
        x = F.relu(self.fc13(x))
        x = F.relu(self.fc14(x))
        x = F.relu(self.fc15(x))
        x = self.fc16(x)
        return F.log_softmax(x)
        #return F.softmax(x)

#%% plot function

def plot_data(X, y):
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
    plt.show()

def plot_decision_boundary(clf, X, y):
    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 gid
    X_out = net(torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype = torch.floa
```

```
t))
    Z = X_out.data.max(1)[1]
    # Z.shape
    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)
    plt.show()
```

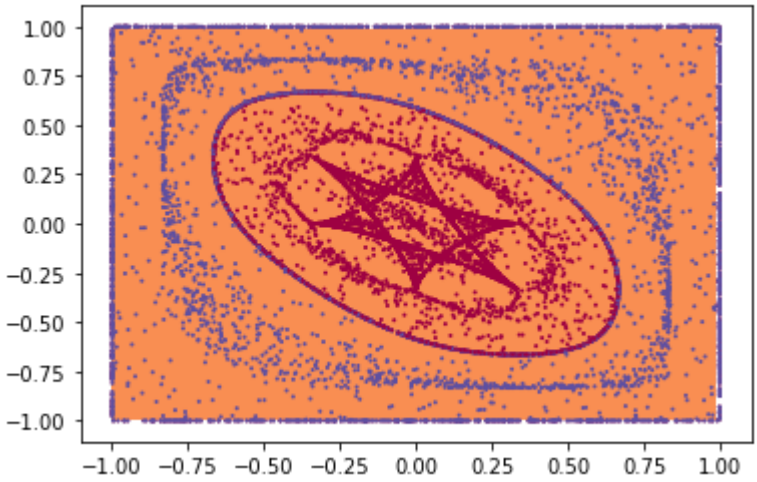
In [ ]:

```
#%% train
net = Net()

# create a stochastic gradient descent optimizer
learning_rate = .01
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
#optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)


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

nepochs = 8000
data, target = X, y
# 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
```

```
C:\Users\Jingwen\anaconda3\lib\site-packages\ipykernel_launcher.py:39: UserWa
rning: Implicit dimension choice for log_softmax has been deprecated. Change
the call to include dim=X as an argument.
```

In [54]:
```
#%% compute accuracy on training data

net_out = net(data)
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)
plt.scatter(X[:, 0], X[:, 1], c=pred, cmap=plt.cm.Spectral, s = 1)


#%%  plot outputs
plot_decision_boundary(net, X, y)
```

```
C:\Users\Jingwen\anaconda3\lib\site-packages\ipykernel_launcher.py:39: UserWa
rning: Implicit dimension choice for log_softmax has been deprecated. Change
the call to include dim=X as an argument.

Training accuracy is  0.5565321914206296
```

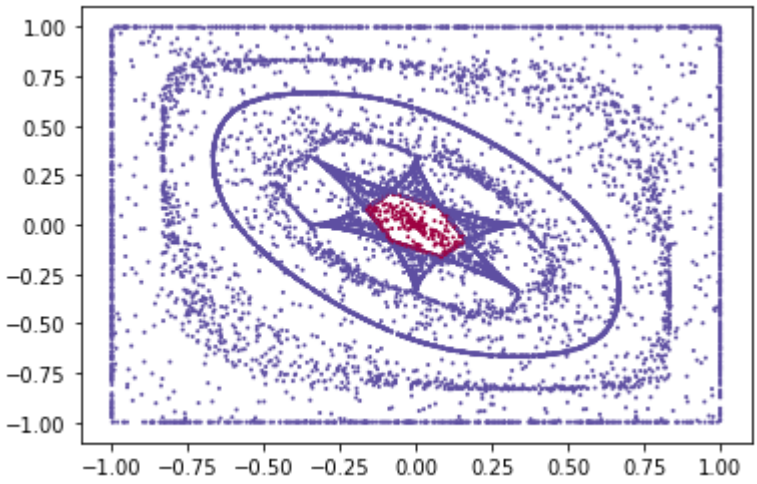


2.2. File Feedforward_Data_hexa.csv contains 13312 two-dimensional data points (feature values located in columns A and B) and their respective binary label (labels located in column C). Create and train a network that separates the data. Report your best loss and accuracy values. Plot the decision boundaries of your best network by plotting the network outputs in a densely sampled region around $[-1.0, 1.0] \times [-1.0, 1.0]$. Report the number of hidden layers, type of activation function and number of neurons per layer used. (15 points).

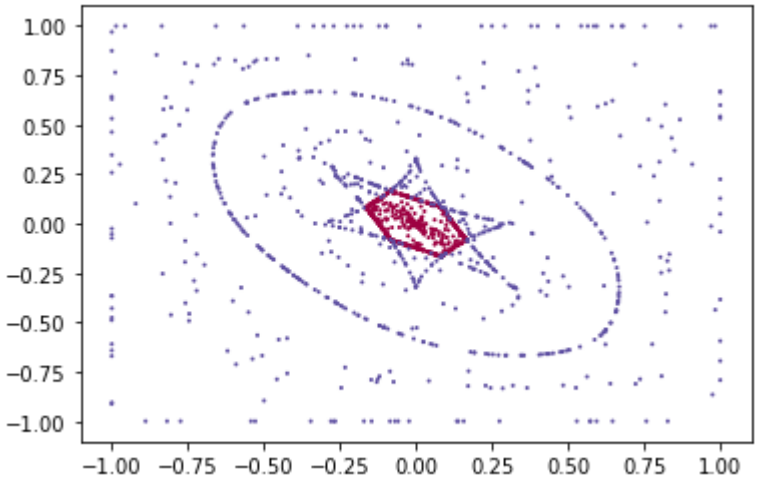
```
hidden layers = 15
activation function: relu
number of neurons per layer used: 50
```

In [16]:
```
#%% read data
data=pd.read_csv("FeedForward_Data_hexa.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)

#plot_data(X,y,'data.pdf')
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
```

Out[16]: <matplotlib.collections.PathCollection at 0x1d72d88f7c8>



In [29]:
```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=0)
X_resampled, y_resampled = rus.fit_sample(X, y)
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=y_resampled, cmap=plt.cm.S
pectral, s = 1)
X_resampled = torch.tensor(X_resampled, dtype = torch.float)
y_resampled = torch.tensor(y_resampled, dtype = torch.long)
```



In [44]:
```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 50)
        self.fc2 = nn.Linear(50, 50)
        self.fc3 = nn.Linear(50, 50)
        self.fc4 = nn.Linear(50, 50)
        self.fc5 = nn.Linear(50, 50)
        self.fc6 = nn.Linear(50, 50)
        self.fc7 = nn.Linear(50, 50)
        self.fc8 = nn.Linear(50, 50)
        self.fc9 = nn.Linear(50, 50)
        self.fc10 = nn.Linear(50, 50)
        self.fc11 = nn.Linear(50, 50)
        self.fc12 = nn.Linear(50, 50)
        self.fc13 = nn.Linear(50, 50)
        self.fc14 = nn.Linear(50, 50)
        self.fc15 = nn.Linear(50, 50)
        self.fc16 = nn.Linear(50, 2)

    def forward(self, x):
        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 = F.relu(self.fc12(x))
        x = F.relu(self.fc13(x))
        x = F.relu(self.fc14(x))
        x = F.relu(self.fc15(x))
        x = self.fc16(x)
        return F.log_softmax(x)
        #return F.softmax(x)

#%% plot function

def plot_data(X, y):
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
    plt.show()

def plot_decision_boundary(clf, X, y):
    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 gid
    X_out = net(torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype = torch.floa
```

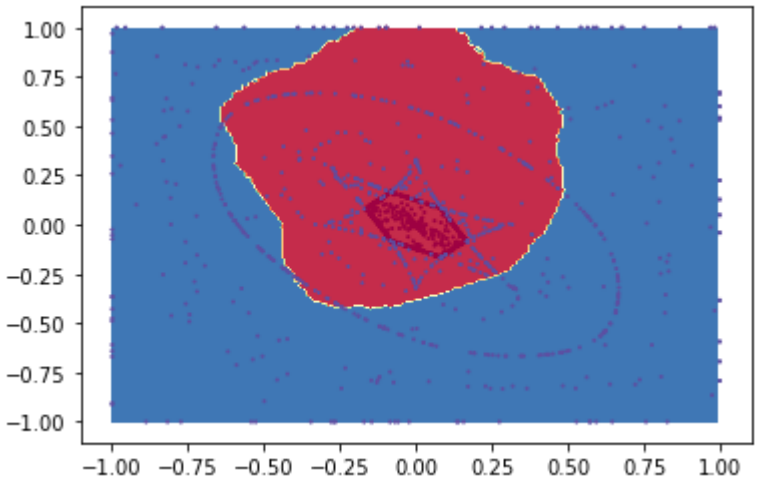
```
t))
    Z = X_out.data.max(1)[1]
    # Z.shape
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
    plt.show()
```

In [45]:
```
#%% train
net = Net()

# create a stochastic gradient descent optimizer
learning_rate = .01
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
#optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)


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

nepochs = 15000
data, target = X_resampled, y_resampled
# 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
```

```
C:\Users\Jingwen\anaconda3\lib\site-packages\ipykernel_launcher.py:39: UserWa
rning: Implicit dimension choice for log_softmax has been deprecated. Change
the call to include dim=X as an argument.
```

In [46]:
```
#%% compute accuracy on training data

net_out = net(data)
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)
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=pred, cmap=plt.cm.Spectral
, s = 1)

#%% if need to train further

# for epoch in range(nepochs):
#     # resize data from (batch_size, 1, 28, 28) to (batch_size, 28*28)
#     optimizer.zero_grad()
#     net_out = net(data)
#     loss = criterion(net_out, target)
#     loss.backward()
#     optimizer.step()
#     if epoch % 100 == 0:
#         print('Epoch ', epoch, 'Loss ', loss.item())


#%%  plot outputs
plot_decision_boundary(net, X_resampled, y_resampled)
```

```
C:\Users\Jingwen\anaconda3\lib\site-packages\ipykernel_launcher.py:39: UserWa
rning: Implicit dimension choice for log_softmax has been deprecated. Change
the call to include dim=X as an argument.

Training accuracy is  0.7338811630847029
```



In [ ]:
```
```