P2_Q1_A

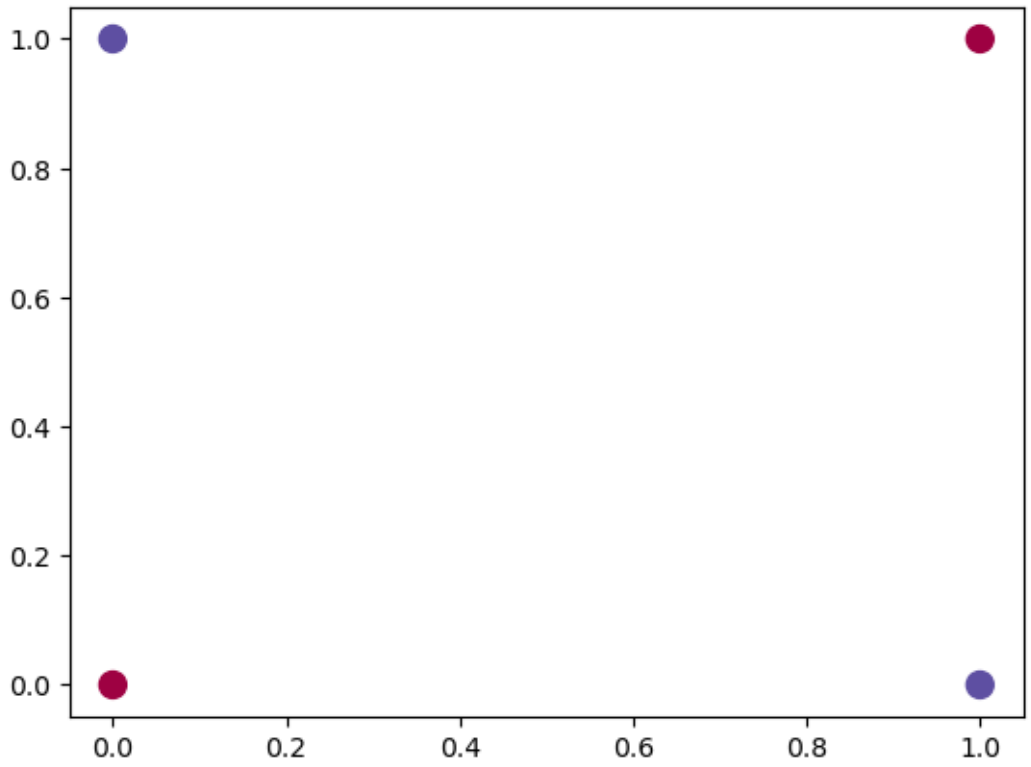
November 15, 2022

[43]:
```
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

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 2)
        self.fc2 = nn.Linear(2, 2)
        self.fc3 = nn.Linear(2, 2)
        #self.fc4 = nn.Linear(3, 3)
        self.fc5 = nn.Linear(2, 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 = self.fc5(x)
        return F.log_softmax(x)
        #return F.softmax(x)
```

[44]:
```
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()

def plot_decision_boundary(clf, X, y, filename):
    # Set min and max values and give it some padding
    #x_min, x_max = X[:, 0].min() - .1, X[:, 0].max() + .1
    #y_min, y_max = X[:, 1].min() - .1, X[:, 1].max() + .1
    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.
 ↪float))
    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.savefig(filename)
    plt.close()
data = pd.read_csv("XOR.csv") # UPDATE THE FILE NAME AND PATH TO MATCH YOUR␣
 ↪REQUIREMENT
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)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 100)
```

[44]: <matplotlib.collections.PathCollection at 0x11e547c4f70>



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

# create a stochastic gradient descent optimizer
learning_rate = 0.01
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)
#optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)


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

#nepochs = 600
nepochs = 3000
#10000
data, target = X, y

for epoch in range(nepochs):
#    adjust learning rate if desired
#    if epoch % 3000 == 0 and epoch <= 24000:
#        for g in optimizer.param_groups:
#            g['lr'] = g['lr']/2
    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

    if epoch % 10 == 0:
        print('Epoch ', epoch, 'Loss ', loss.item())
        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)
        if (accuracy==1):
```

```
            break
```

```
Epoch  0 Loss  0.6987584829330444
Training accuracy is  0.5
Epoch  10 Loss  0.6936651468276978
Training accuracy is  0.5
Epoch  20 Loss  0.6926511526107788
Training accuracy is  0.5
Epoch  30 Loss  0.6917334198951721
Training accuracy is  0.5
Epoch  40 Loss  0.6888456344604492
Training accuracy is  0.5
Epoch  50 Loss  0.6809083819389343
Training accuracy is  0.75
Epoch  60 Loss  0.6620653867721558
Training accuracy is  0.75
Epoch  70 Loss  0.6267451047897339
Training accuracy is  0.75
Epoch  80 Loss  0.5702000260353088
Training accuracy is  0.75
Epoch  90 Loss  0.5063083171844482
Training accuracy is  0.75
Epoch  100 Loss  0.4361491799354553
Training accuracy is  0.75
Epoch  110 Loss  0.35086917877197266
Training accuracy is  1.0
```

```
C:\Users\aradh\AppData\Local\Temp\ipykernel_42320\2674906490.py:24: UserWarning:
Implicit dimension choice for log_softmax has been deprecated. Change the call
to include dim=X as an argument.
  return F.log_softmax(x)
```

[ ]:

[46]:
```
plt.scatter(X[:, 0], X[:, 1], c=pred, cmap=plt.cm.Spectral, s = 1)
plot_decision_boundary(net, X, y, 'P2_Q1_A.pdf')
```

```
C:\Users\aradh\AppData\Local\Temp\ipykernel_42320\2674906490.py:24: UserWarning:
Implicit dimension choice for log_softmax has been deprecated. Change the call
to include dim=X as an argument.
  return F.log_softmax(x)
```

[ ]: