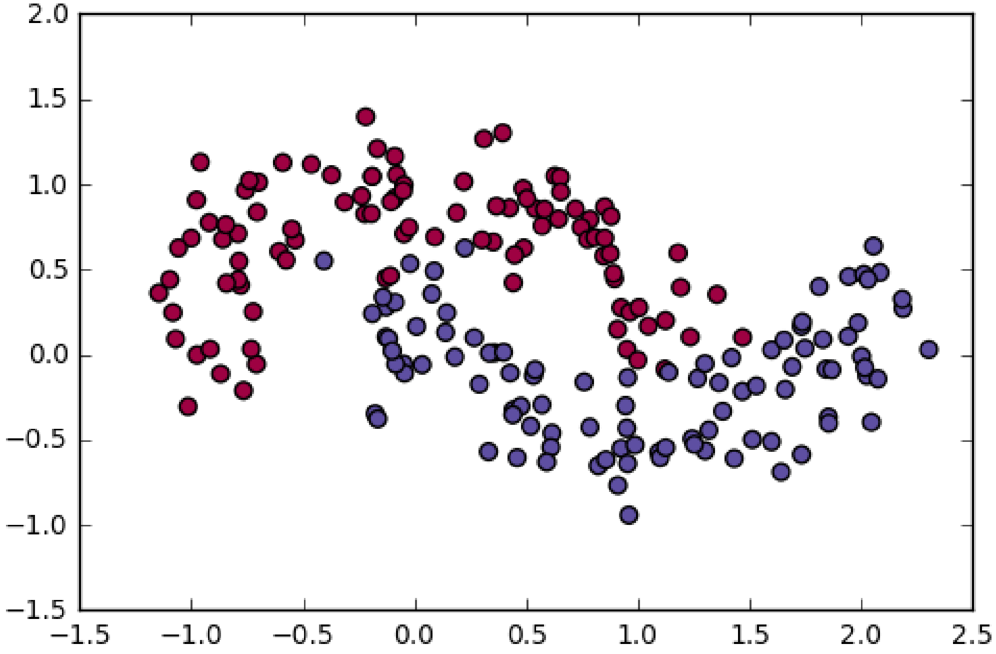
HW5

December 10, 2017

In [189]:
```
# Import packages
import numpy as np
import sklearn
from sklearn import datasets
import matplotlib.pyplot as plt
%matplotlib inline
```

In [190]:
```
# Generate a toy dataset and plot it
np.random.seed(1)
X, y = sklearn.datasets.make_moons(200, noise=0.20)
plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=plt.cm.Spectral)
```

Out[190]:
```
<matplotlib.collections.PathCollection at 0x1a10b74690>
```



In [191]:
```python
# Define helper functions

def predict(W, X):

    # Forward pass
    z_h1 = np.dot(X, W['W_input_h1']) + W['b_input_h1']
    a_h1 = np.tanh(z_h1)

    z_h2 = np.dot(a_h1, W['W_h1_h2']) + W['b_h1_h2']
    a_h2 = np.tanh(z_h2)
    output = np.dot(a_h2, W['W_h2_output']) + W['b_h2_output']

    probs = sigmoid(output)
```

```
        probs = probs.squeeze()

        predictions = np.zeros(probs.shape[0])
        predictions[probs>0.5] = 1

        return predictions

    def cross_entropy_loss(t_hat, t):
        return np.sum(-(t*np.log(t_hat) + (1-t)*np.log(1-t_hat)))

    def sigmoid(x):
        return 1/(1 + np.exp(-x))

    def sigmoid_grad(x):
        return sigmoid(x)*(1-sigmoid(x))

    def tanh_grad(x):
        return 1 - np.tanh(x)**2

    def plot_decision_boundary(W):
        # Set min and max values and give it some padding
        x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
        y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .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 = predict(W, np.c_[xx.ravel(), yy.ravel()])
        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)
```

In [192]:
```
# Create the model architecture
input_dim = 2
nodes_hidden_1 = 3
nodes_hidden_2 = 3
output_dim = 1
```

```python
W = {
    'W_input_h1' : np.random.rand(input_dim, nodes_hidden_1),
    'W_input_h1' : np.random.rand(input_dim, nodes_hidden_1),
    'b_input_h1' : np.random.rand(nodes_hidden_1),

    'W_h1_h2' : np.random.rand(nodes_hidden_1, nodes_hidden_2),
    'b_h1_h2' : np.random.rand(nodes_hidden_2),

    'W_h2_output' : np.random.rand(nodes_hidden_2, output_dim),
    'b_h2_output' : np.random.rand(output_dim)
}
```

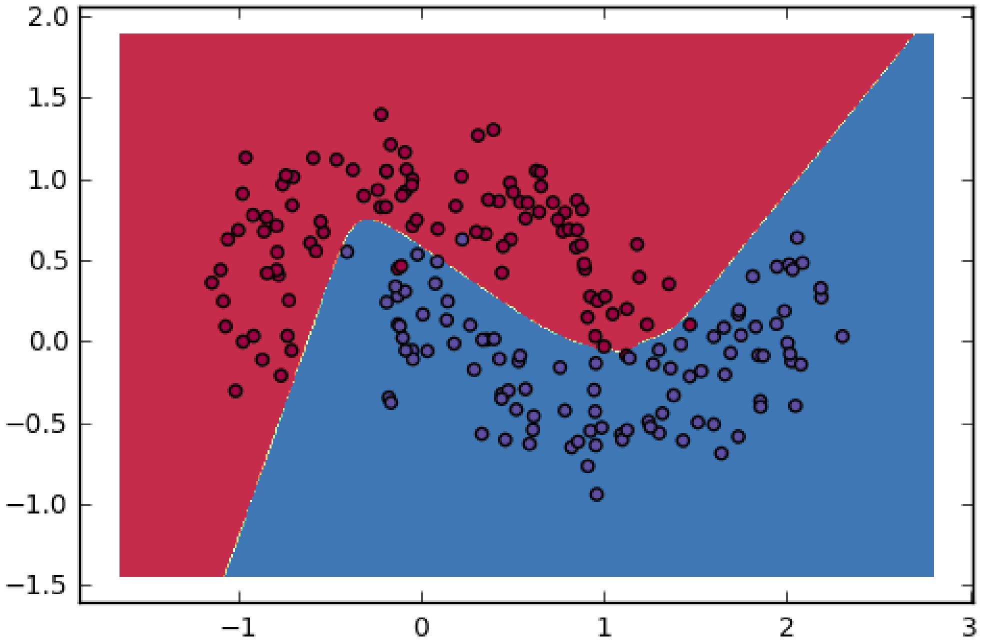
In [193]:
```python
# Train using Gradient Descent

learning_rate = 0.01
reg_lambda = 0.01
no_epochs = 2000

for epoch in range(no_epochs):

    # Forward pass
    z_h1 = np.dot(X, W['W_input_h1']) + W['b_input_h1']
    a_h1 = np.tanh(z_h1)

    z_h2 = np.dot(a_h1, W['W_h1_h2']) + W['b_h1_h2']
    a_h2 = np.tanh(z_h2)
    output = np.dot(a_h2, W['W_h2_output']) + W['b_h2_output']

    probs = sigmoid(output)
    probs = probs.squeeze()

    #~~~~~~~~~~~~ YOUR CODE BEGINS HERE ~~~~~~~~~~~~~~~~

    # Backward Pass

    # Gradient Descent
```

```python
        # Take derivative of the loss function wrt W_h2_output, a_h2, and b_h2_output
        der_probs = (probs-y)/(probs*(1-probs)) # (200,)
        der_output = der_probs.reshape(200,1) * sigmoid_grad(output)
        der_W_h2_output = np.matmul(a_h2.T, der_output) + reg_lambda * W['W_h2_output'] # (200,3)
        der_a_h2_output = np.matmul(der_output, W['W_h2_output'].T)
        der_b_h2_output = np.sum(der_output)

        # Take derivative wrt W_h1_output, a_h1, and b_h1_output
        der_z_h2 = der_a_h2_output * tanh_grad(z_h2)
        der_W_h1_h2 = np.matmul(a_h1.T, der_z_h2) + reg_lambda * W['W_h1_h2'] # (3,3)
        der_a_h1_output = np.matmul(der_z_h2, W['W_h1_h2'].T)
        der_b_h1_h2 = np.sum(der_z_h2)

        # Take derivative wrt to z_h1, W_input_h1, and b_input_h1
        der_z_h1 = der_a_h1_output * tanh_grad(z_h1)
        der_W_input_h1 = np.matmul(X.T, der_z_h1) + reg_lambda * W['W_input_h1'] #(2,3)
        der_b_input_h1 = np.sum(der_z_h1)

        # update weight
        W['W_input_h1'] = W['W_input_h1'] - learning_rate * der_W_input_h1
        W['b_input_h1'] = W['b_input_h1'] - learning_rate * der_b_input_h1
        W['W_h1_h2'] = W['W_h1_h2'] - learning_rate * der_W_h1_h2
        W['b_h1_h2'] = W['b_h1_h2'] - learning_rate * der_b_h1_h2
        W['W_h2_output'] = W['W_h2_output'] - learning_rate * der_W_h2_output
        W['b_h2_output'] = W['b_h2_output'] - learning_rate * der_b_h2_output

        #~~~~~~~~~~~ YOUR CODE ENDS HERE ~~~~~~~~~~~~~~~~~

        loss = cross_entropy_loss(probs, y)

        if epoch%100 == 0:
            print("Loss after epoch ", epoch, " : ", loss)
```

```
('Loss after epoch ', 0, ' : ', 190.71724215235707)
('Loss after epoch ', 100, ' : ', 58.673491981533616)
('Loss after epoch ', 200, ' : ', 56.44930914655167)
('Loss after epoch ', 300, ' : ', 49.900367072018071)
('Loss after epoch ', 400, ' : ', 18.811066737894912)
```

```
('Loss after epoch ', 500, ' : ', 28.596498310259687)
('Loss after epoch ', 600, ' : ', 33.146068794884094)
('Loss after epoch ', 700, ' : ', 34.07696275842622)
('Loss after epoch ', 800, ' : ', 23.295777427785708)
('Loss after epoch ', 900, ' : ', 20.65191828345888)
('Loss after epoch ', 1000, ' : ', 33.484715928022396)
('Loss after epoch ', 1100, ' : ', 15.645999476589427)
('Loss after epoch ', 1200, ' : ', 14.673187249897939)
('Loss after epoch ', 1300, ' : ', 15.874693293299806)
('Loss after epoch ', 1400, ' : ', 14.653693633529139)
('Loss after epoch ', 1500, ' : ', 25.968371880237765)
('Loss after epoch ', 1600, ' : ', 14.324316486858281)
('Loss after epoch ', 1700, ' : ', 16.005930278943442)
('Loss after epoch ', 1800, ' : ', 16.873669195372848)
('Loss after epoch ', 1900, ' : ', 15.24150726138199)
```

```
In [194]: plot_decision_boundary(W)
```




```
In [ ]:
```