In [292]:

```
import numpy as np
```

In [293]:

```
# Model Parameters
input_size = 2 # no_of_features
layers = [4,3] # no of neurons in 1st and 2nd layers
output_size = 2
```

In [305]:

```
class NeuralNetwork:

    def __init__(self,input_size,layers,output_size):
        np.random.seed(0)

        model = {} # dictionary

        # First Layer
        model['W1'] = np.random.randn(input_size,layers[0])
        model['b1'] = np.zeros((1,layers[0]))

        # Second Layer
        model['W2'] = np.random.randn(layers[0],layers[1])
        model['b2'] = np.zeros((1,layers[1]))

        # Third Layer
        model['W3'] = np.random.randn(layers[1],output_size)
        model['b3'] = np.zeros((1,output_size))

        self.model = model

    def forward(self,x):

        W1,W2,W3 = self.model['W1'],self.model['W2'],self.model['W3']
        b1,b2,b3 = self.model['b1'],self.model['b2'],self.model['b3']

        z1 = np.dot(x,W1) + b1
        a1 = np.tanh(z1)

        z2 = np.dot(a1,W2) + b2
        a2 = np.tanh(z2)

        z3 = np.dot(a2,W3) + b3
        y_ = softmax(z3)

        self.activation_outputs = (a1,a2,y_)
        return y_

    def backward(self,x,y,learning_rate=0.001):
        W1,W2,W3 = self.model['W1'],self.model['W2'],self.model['W3']
        b1,b2,b3 = self.model['b1'],self.model['b2'],self.model['b3']
        a1,a2,y_ = self.activation_outputs
```

```
        m = x.shape[0]

        delta3 = y_ - y
        dw3 = np.dot(a2.T,delta3)
        db3 = np.sum(delta3,axis=0)/float(m)

        delta2 = (1-np.square(a2))*np.dot(delta3,W3.T)
        dw2 = np.dot(a1.T,delta2)
        db2 = np.sum(delta2,axis=0)/float(m)

        delta1 = (1-np.square(a1))*np.dot(delta2,W2.T)
        dw1 = np.dot(X.T,delta1)
        db1 = np.sum(delta1,axis=0)/float(m)

        # Update the model parameter using Gradient Descent
        self.model['W1'] -= learning_rate*dw1
        self.model['b1'] -= learning_rate*db1

        self.model['W2'] -= learning_rate*dw2
        self.model['b2'] -= learning_rate*db2

        self.model['W3'] -= learning_rate*dw3
        self.model['b3'] -= learning_rate*db3


    def predict(self,x):
        y_out = self.forward(x)
        return np.argmax(y_out,axis=1)


    def summary(self):
        W1,W2,W3 = self.model['W1'],self.model['W2'],self.model['W3']
        a1,a2,y_ = self.activation_outputs

        print("W1",W1.shape)
        print("A1",a1.shape)

        print("W2",W2.shape)
        print("A2",a2.shape)

        print("W3",W3.shape)
        print("Y_",y_.shape)
```

In [306]:

```
def loss(y_oht, p):
    l = -np.mean(y_oht*np.log(p))
    return l

def one_hot(y,depth):
    m = y.shape[0]
    y_oht = np.zeros((m,depth))
    y_oht[np.arange(m),y] = 1
    return y_oht

def softmax(a):
    e_pa = np.exp(a)
    ans = e_pa/np.sum(e_pa,axis=1,keepdims=True)
    return ans
```

In [307]:

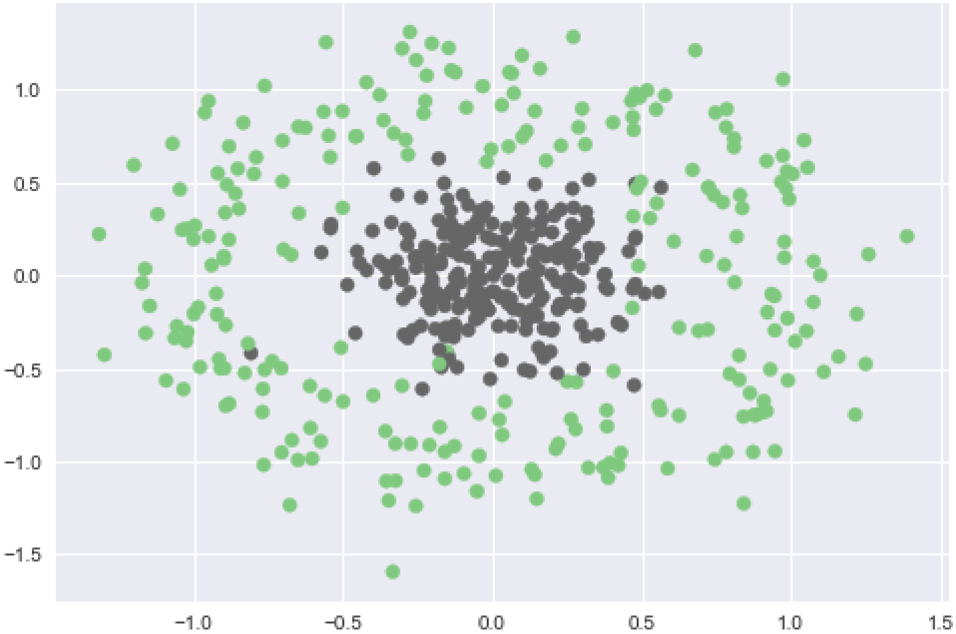
```
## Generate Dataset
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt
```

In [308]:

```
X,Y = make_circles(n_samples=500, shuffle=True,noise=0.2,random_state=1,factor=0.2)
```

In [309]:

```
plt.style.use('seaborn')
plt.scatter(X[:,0],X[:,1],c=Y,cmap=plt.cm.Accent)
plt.show()
```



In [310]:

```
print(Y.shape)
```

```
(500,)
```

In [ ]:

In [311]:

```
def train(X,Y,model,epochs,learning_rate,logs=True):
    training_loss = []

    classes = 2
    Y_OHT = one_hot(Y,classes)
    for ix in range(epochs):

        Y_ = model.forward(X)
        l = loss(Y_OHT,Y_)
        training_loss.append(l)
        model.backward(X,Y_OHT,learning_rate)

        if(logs):
            print("Epoch %d Loss %.4f"%(ix,l))

    return training_loss
```

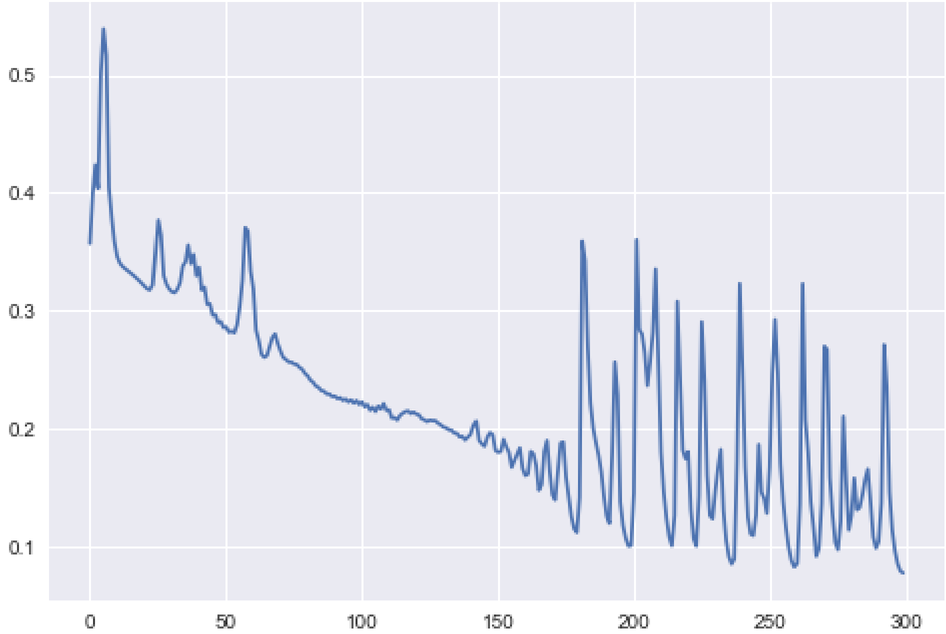
In [339]:

```
#Training Model
model = NeuralNetwork(input_size=2,layers=[10,5],output_size=2)
losses = train(X,Y,model,300,0.005,True)
```

```
Epoch 213 Loss 0.1084
Epoch 214 Loss 0.1006
Epoch 215 Loss 0.1265
Epoch 216 Loss 0.3078
Epoch 217 Loss 0.2445
Epoch 218 Loss 0.1820
Epoch 219 Loss 0.1747
Epoch 220 Loss 0.1805
Epoch 221 Loss 0.1317
Epoch 222 Loss 0.1068
Epoch 223 Loss 0.1003
Epoch 224 Loss 0.1433
Epoch 225 Loss 0.2909
Epoch 226 Loss 0.2415
Epoch 227 Loss 0.1602
Epoch 228 Loss 0.1263
Epoch 229 Loss 0.1235
Epoch 230 Loss 0.1473
Epoch 231 Loss 0.1669
Epoch 232 Loss 0.1824
Epoch 233 Loss 0.1305
Epoch 234 Loss 0.1053
Epoch 235 Loss 0.0913
Epoch 236 Loss 0.0852
```

In [340]:

```
plt.plot(losses)
plt.show()
```



In [341]:

```
model.summary()
```

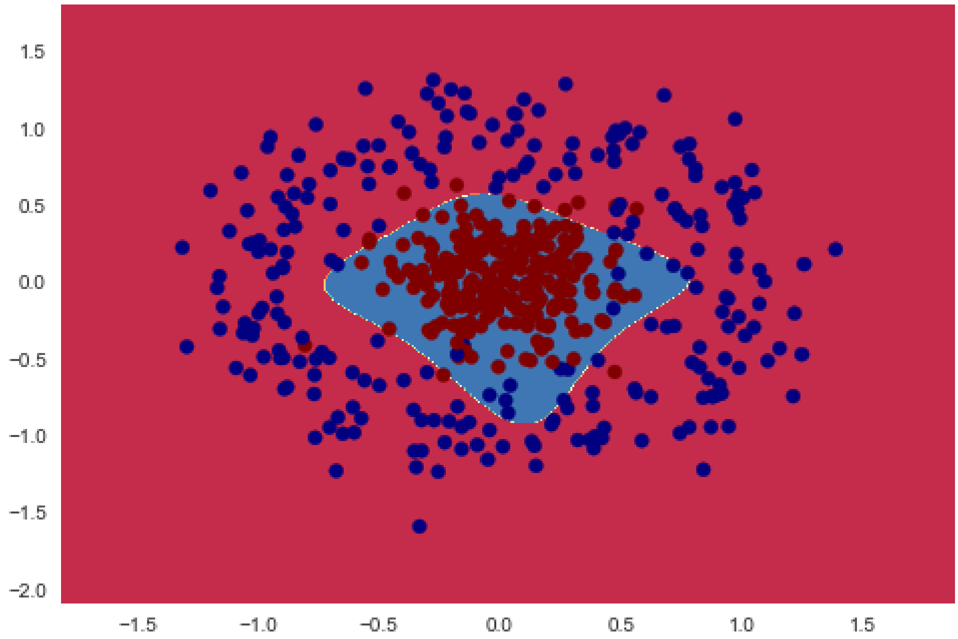
```
W1 (2, 10)
A1 (500, 10)
W2 (10, 5)
A2 (500, 5)
W3 (5, 2)
Y_ (500, 2)
```

In [342]:

```
from visualize import plot_decision_boundary
```

In [343]:

```
plot_decision_boundary(lambda x:model.predict(x),X,Y)
```



In [344]:

```
outputs = model.predict(X)
np.sum(outputs==Y)/Y.shape[0]
```

```
0.952
```

In [ ]: