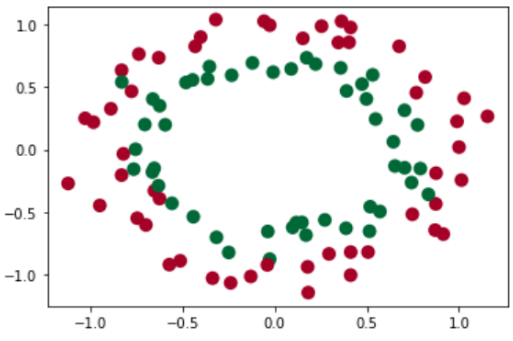
(a) Train a neural network on a dataset You don't learn from just copying :) to ≥ 98% accuracy. What is the test set performance?

```
class Net(nn.Module):
    def __init__(self, n_input, n_output):
        super(Net, self).__init__()

        self.linear = nn.Linear(n_input, 1000)
        self.linear_1 = nn.Linear(1000, 1000)
        self.relu = F.relu
        self.output = nn.Linear(1000, n_output)

    def forward(self, x):

        #x = self.pool(F.relu(self.conv(x)))
        x = self.linear(x)
        x = self.relu(x)
        x = self.relu(self.linear_1(x))
        x = self.relu(self.linear_1(x))
        x = self.relu(self.linear_1(x))
        x = self.relu(self.linear_1(x))
        x = self.output(x)
        return x
```

Result:

```
plt.scatter(data.cpu().numpy()[:, 0], data.cpu().numpy()[:, 1], c=pred_label, s=100, lw=0, cmap='RdYlGn')
#accuracy = roc_auc_score(target_label, pred_label)
accuracy = float((pred_label == target_label).astype(int).sum()) / float(target_label.size)
plt.text(1, -2, 'Acc.=%.2f' % accuracy, fontdict={'size': 20, 'color': 'red'})
plt.show()
```



We can see that after multiple tests the accuracy of our neural network increases and sometimes even reaches 1.0.

However, this neural network is not quite stable and needs improvement. Generally, it is still an efficient neural network.

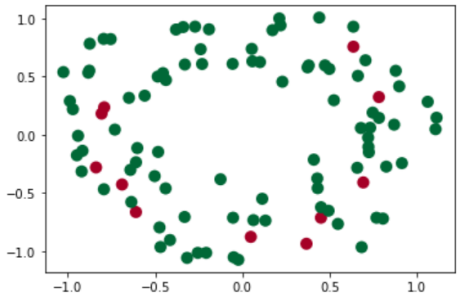
What is the test set performance?

```
In [36]: data = torch.tensor(X_test, dtype = torch.float)
         pred_label = net(data)
         pred_label = torch.max(pred_label, 1)[1]
         y_pred = pred_label.detach().numpy()

In [37]: target = torch.tensor(y_test)
         target_label = target.type(torch.LongTensor)

In [42]: accuracy = float((y_pred == y_test).astype(int).sum()) /len(target)
         accuracy

Out[42]: 0.89
```

(b) Illustrate which points in the training and test sets that are correctly classified using a scatter plot.

(b)Illustrate which points in the training and test sets that are correctly classified using a scatter plot.

```
In [43]: plt.scatter(data.cpu().numpy()[:, 0], data.cpu().numpy()[:, 1], c=(y_pred == y_test), s=100, lw=0, cmap='RdYlGn')
         plt.text(1, -2, 'Acc.=%.2f' % accuracy, fontdict={'size': 20, 'color': 'red'})
         plt.show()
```

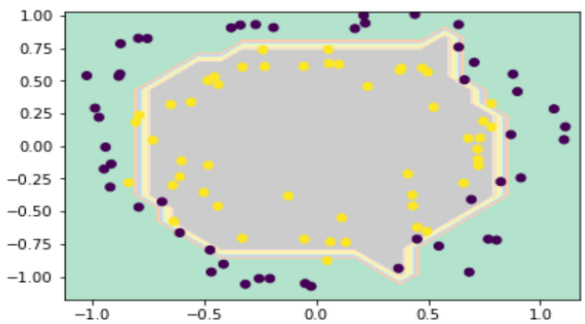


Acc.=0.89

The green spots are correctly predicted.

(c) Illustrate the decision boundary, i.e., the line separating the two predictions by your model.

```
In [51]: plt.contourf(xx, yy, zz.detach().numpy(), cmap="Pastel2")
         plt.scatter(data.cpu().numpy()[:, 0], data.cpu().numpy()[:, 1], c=target_label)

Out[51]: <matplotlib.collections.PathCollection at 0x200b4b70250>
```



(d) Can you create a better classifier manually?

In [52]:
```
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, roc_auc_score, classification_
```

In [53]:
```
svm_clf = SVC(kernel='rbf',gamma='auto')
svm_clf.fit(X_train, y_train)
q_estimate_svm = svm_clf.predict(X_test)
```

In [54]:
```
svm_auc_score = roc_auc_score(y_test, q_estimate_svm)*100
print('Training AUC: %.4f %%' % svm_auc_score)
```

```
Training AUC: 94.0000 %
```

The SVM model is a more stable and also quite accurate classifier.