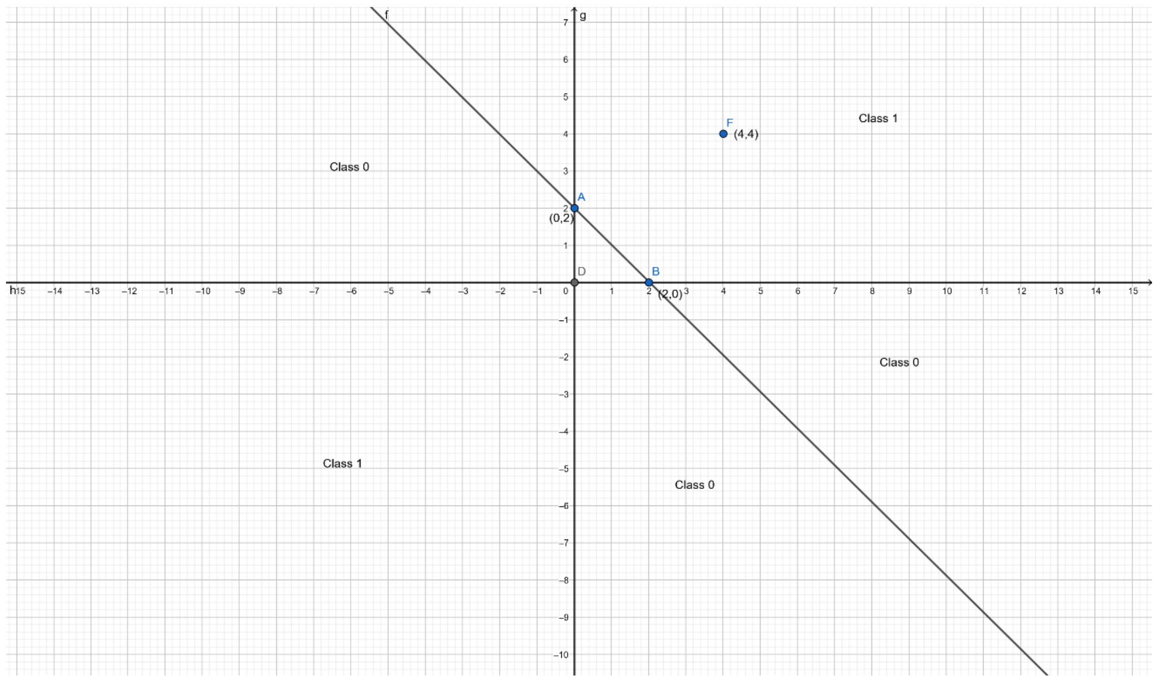
9th Homework (part B)

Thomas Saltos

Exercise 4

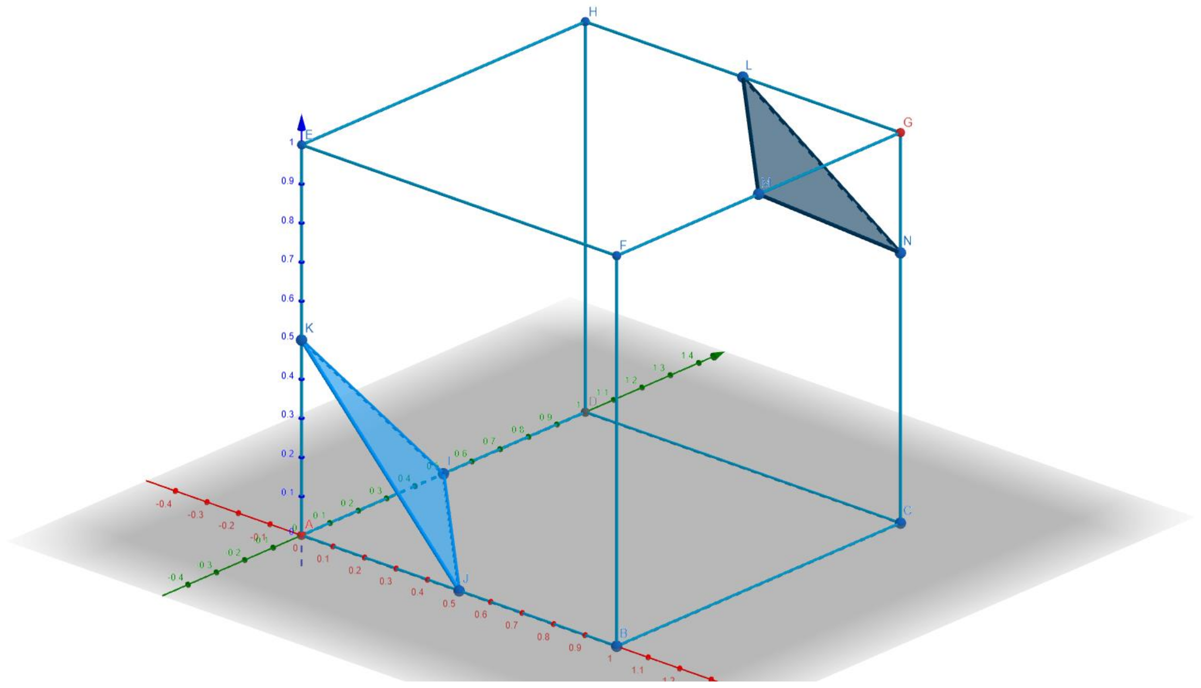
i)



ii) The first hidden layer will have 3 nodes corresponding to each of the line $(\varepsilon_1), (\varepsilon_2), (\varepsilon_3)$. Each region of each class is assigned to a vertex of the 3-dimensional hypercube. In this case the vertices of the hypercube that correspond to class 1 are not linearly separable from the remaining vertices so we will add a second hidden layer with two nodes, each one corresponding to a plane separating a vertex corresponding to a region of class 1. In order to identify the plane, we need 3 points. For h1 that separate the vertex (000) from all remaining ones we use the points $\left(\frac{1}{2}, 0,0\right), \left(0, \frac{1}{2}, 0\right), \left(0,0, \frac{1}{2}\right)$. Thus, we need to solve the following system of linear equations in order to find the parameters of h1

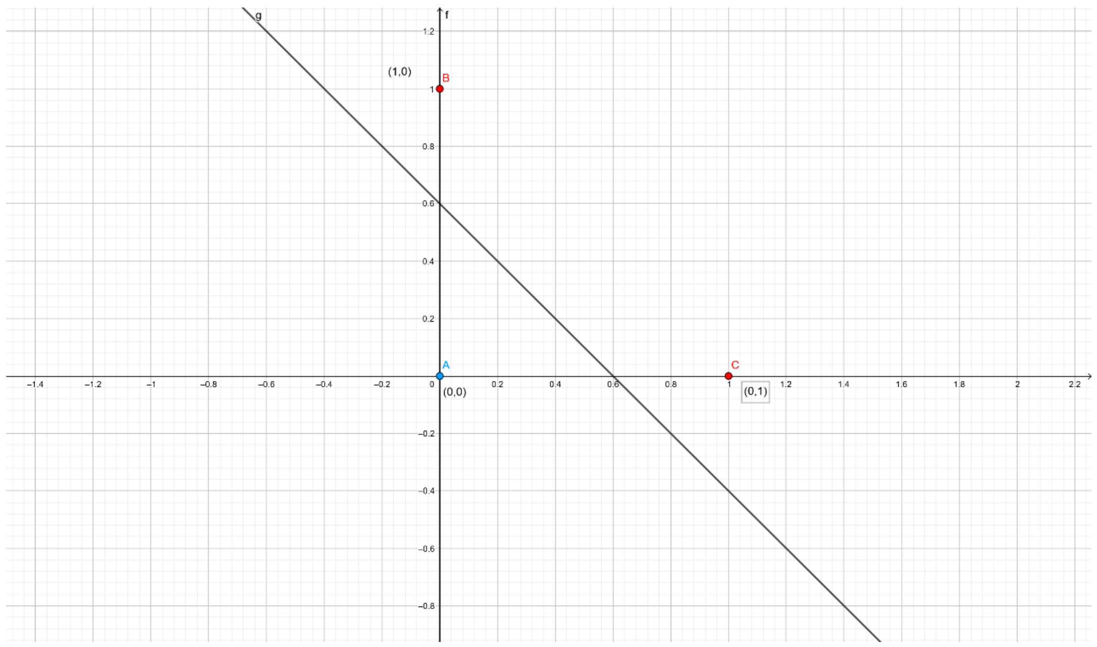
$$\frac{1}{2}\theta_{11} + \theta_{10} = 0$$
$$\frac{1}{2}\theta_{12} + \theta_{10} = 0$$
$$\frac{1}{2}\theta_{13} + \theta_{10} = 0$$

One solution of this system is $\theta_1 = \left[1,1,1,-\frac{1}{2}\right]^T$. For h2 we have the points $\left(\frac{1}{2},1,1\right),\left(1,\frac{1}{2},1\right),\left(1,1,\frac{1}{2}\right)$. A solution of this system is $\theta_2 = \left[1,1,1,-\frac{5}{2}\right]^T$

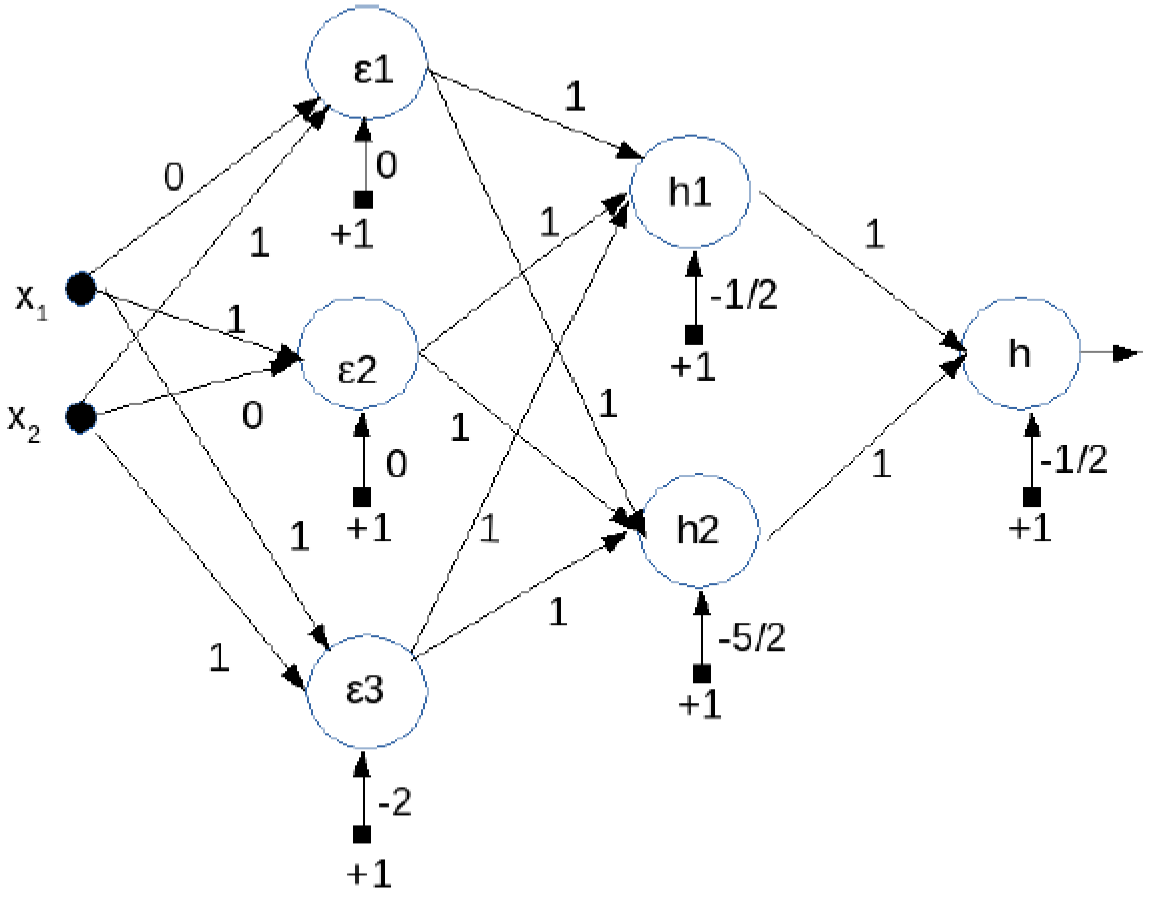


The second hidden layer maps the regions of class 1 to vertices of the square, which are linearly separable from the vertex (0,0) where the regions of class 0 are mapped. In this case we add a node in the third layer that corresponds to a line that separates the two classes. To identify this line, we use the points $\left(\frac{1}{2},0\right),\left(0,\frac{1}{2}\right)$.

A solution to that system is $\theta = \left[1,1,-\frac{1}{2}\right]^T$



The full architecture of the network along with the weights and the thresholds is the following:

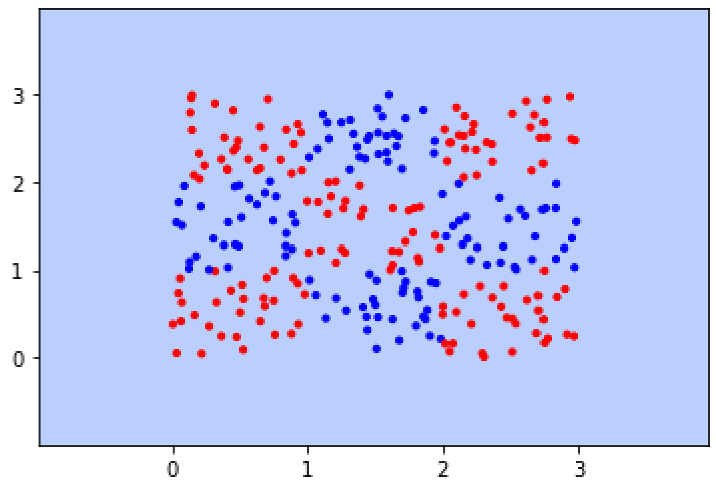


Exercise 5

```
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt


Dataset_a = sio.loadmat('HW9a.mat')

train_x_a = Dataset_a['train_X']
train_y_a = Dataset_a['train_y']
test_x_a = Dataset_a['test_X']
test_y_a = Dataset_a['test_y']



from sklearn import svm


def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

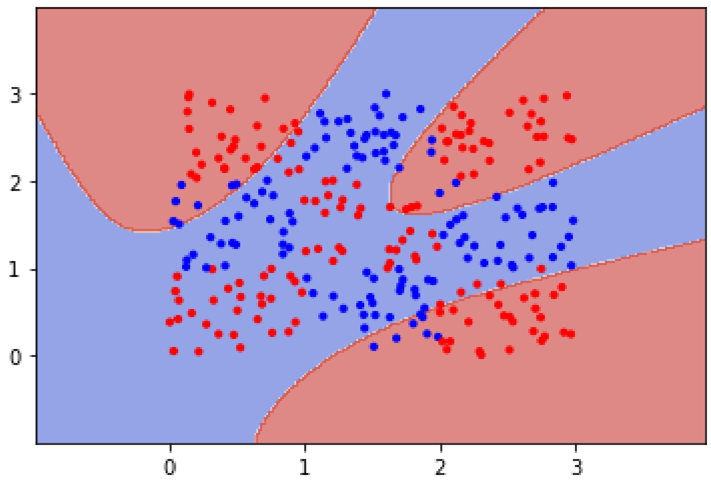
a) Linear kernel

```
clf = svm.SVC(kernel='linear',C=10,gamma=1,degree=2)
clf.fit(train_x_a, train_y_a.reshape(270))

X00, X11 = train_x_a[:,0], train_x_a[:,1]
xx, yy = make_meshgrid(X00, X11)
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in train_y_a.reshape(270)]
plot_contours(ax, clf, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()

clf.predict(test_x_a)
```



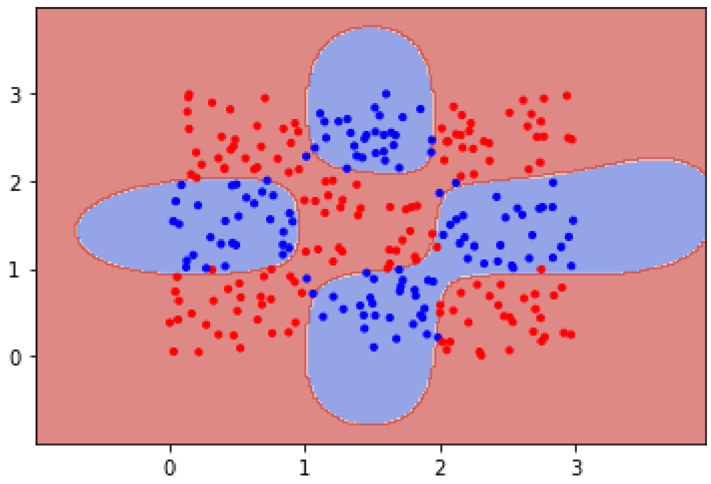
b) Polynomial Kernel

```
clf = svm.SVC(kernel='poly',C=10,gamma=1,degree=5)
clf.fit(train_x_a, train_y_a.reshape(270))

X00, X11 = train_x_a[:,0], train_x_a[:,1]
xx, yy = make_meshgrid(X00, X11)
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in train_y_a.reshape(270)]
plot_contours(ax, clf, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()

clf.predict(test_x_a)
```



c) Rbf kernel

```
clf = svm.SVC(kernel='rbf',C=10,gamma=1,degree=2)
clf.fit(train_x_a, train_y_a.reshape(270))

X00, X11 = train_x_a[:,0], train_x_a[:,1]
xx, yy = make_meshgrid(X00, X11)
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in train_y_a.reshape(270)]
plot_contours(ax, clf, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()

clf.predict(test_x_a)
```



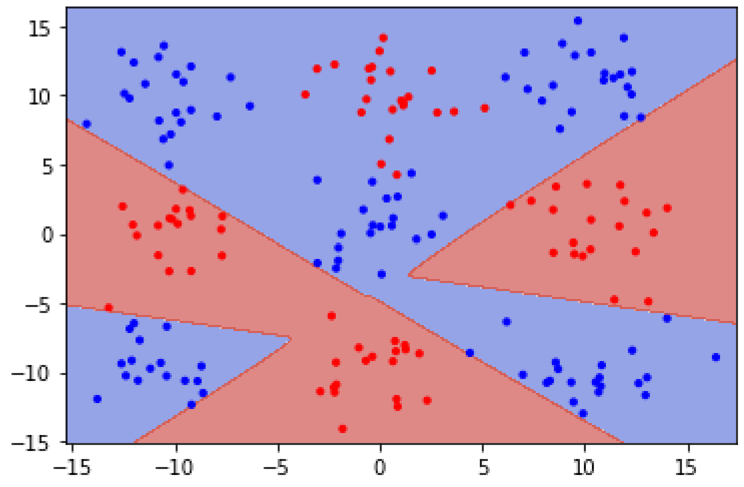
Performing the experiments, it can be noticed that the SVM with the linear kernel always fail to classify the classes since the problem is nonlinear. In the case of SVM with polynomial kernel we have better results than the linear case. Finally, the best results are obtained for the radial basis function kernel.

Exercise 6

```
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt


Dataset_b = sio.loadmat('HW9b.mat')

train_x_b = Dataset_b['train_X']
train_y_b = Dataset_b['train_y']
test_x_b = Dataset_b['test_X']
test_y_b = Dataset_b['test_y']

from sklearn.neural_network import MLPClassifier

def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

a) 3 nodes

```
clf = MLPClassifier(activation='tanh',solver='lbfgs', alpha=1e-3,hidden_layer_sizes=3, random_state=1)
clf.fit(train_x_b, train_y_b.reshape(180))

X00, X11 = train_x_b[:,0], train_x_b[:,1]
xx, yy = make_meshgrid(X00, X11)
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in train_y_b.reshape(180)]
plot_contours(ax, clf, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()


z = (clf.predict(test_x_b))
z = z.reshape(180,1)
Error = 1 - (np.sum(i==1 for i in z == test_y_b)/len(test_y_b))
print('Error - 3 nodes : ',Error)
```

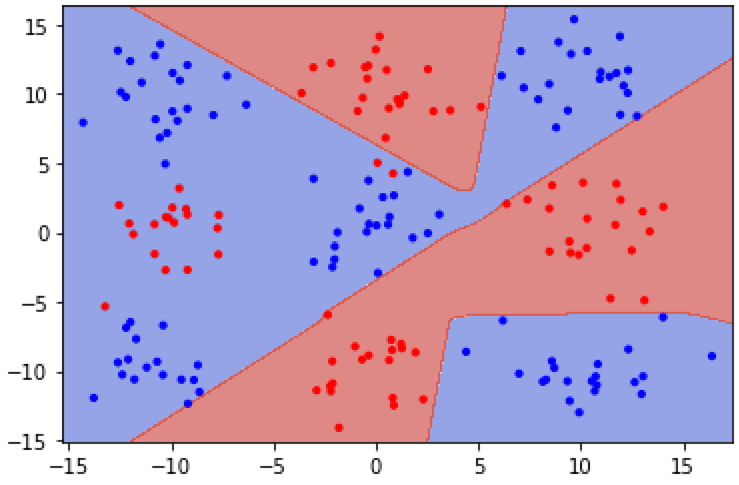


```
Error - 3 nodes : [0.18333333]
```

b) 4 nodes

```
clf = MLPClassifier(activation='tanh', solver='lbfgs', alpha=1e-3,hidden_layer_sizes=4, random_state=1)
clf.fit(train_x_b, train_y_b.reshape(180))

X00, X11 = train_x_b[:,0], train_x_b[:,1]
xx, yy = make_meshgrid(X00, X11)
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in train_y_b.reshape(180)]
plot_contours(ax, clf, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()


z = (clf.predict(test_x_b))
```

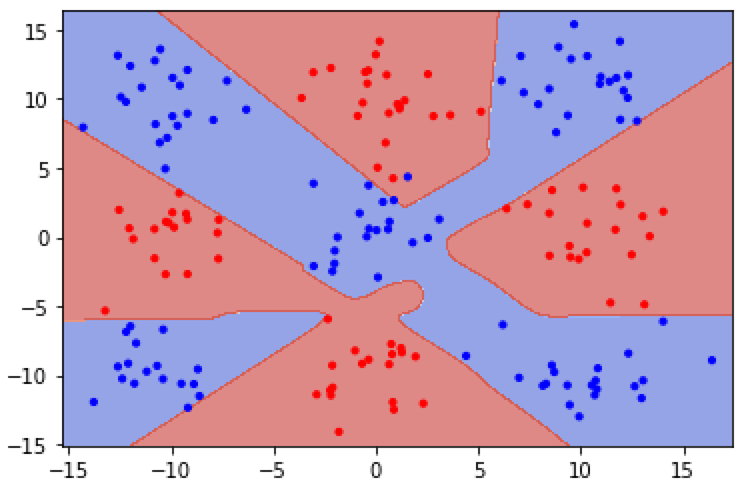
```
z = z.reshape(180,1)
Error = 1 - (np.sum(i==1 for i in z == test_y_b)/len(test_y_b))
print('Error - 4 nodes : ',Error)
```



```
Error - 4 nodes : [0.2]
```

c) 10 nodes

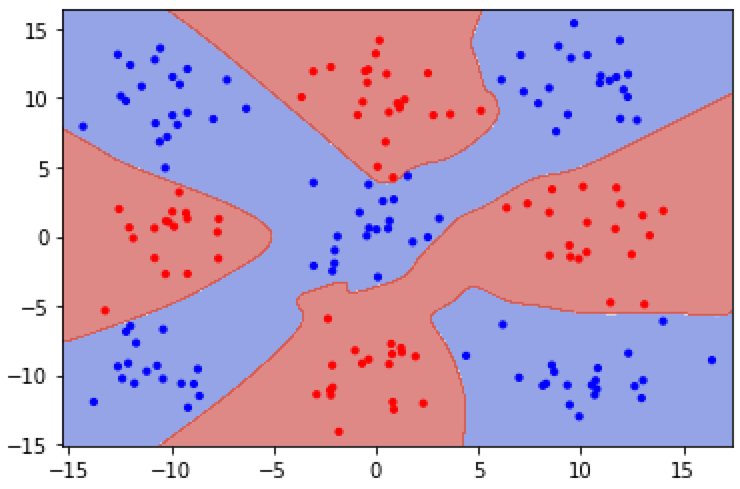
```
clf = MLPClassifier(activation='tanh', solver='lbfgs', alpha=1e-3,hidden_layer_sizes=10, random_state=1)
clf.fit(train_x_b, train_y_b.reshape(180))

X00, X11 = train_x_b[:,0], train_x_b[:,1]
xx, yy = make_meshgrid(X00, X11)
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in train_y_b.reshape(180)]
plot_contours(ax, clf, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()


z = (clf.predict(test_x_b))
z = z.reshape(180,1)
Error = 1 - (np.sum(i==1 for i in z == test_y_b)/len(test_y_b))
print('Error - 10 nodes : ',Error)
```



```
Error - 10 nodes : [0.08888889]
```

d) 50 nodes

```
clf = MLPClassifier(activation='tanh', solver='lbfgs', alpha=1e-3,hidden_layer_sizes=50, random_state=1)
clf.fit(train_x_b, train_y_b.reshape(180))

X00, X11 = train_x_b[:,0], train_x_b[:,1]
xx, yy = make_meshgrid(X00, X11)
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in train_y_b.reshape(180)]
plot_contours(ax, clf, xx, yy,
              cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()


z = (clf.predict(test_x_b))
z = z.reshape(180,1)
Error = 1 - (np.sum(i==1 for i in z == test_y_b)/len(test_y_b))
print('Error - 50 nodes : ',Error)
```



```
      Error - 50 nodes : [0.06111111]
```

It can be noticed that for 10 nodes and above we can see that the training samples of the two classes are well separated. This means that 10 nodes in the hidden layer are enough to solve this problem. Finally, the error on the test set reduces as the number of the nodes in the hidden layer increases.