Homework 3 - Neural Networks

Ran Ju

Netid: rj133

1) Mathematical description of NN

(2.5 points)

Let's assume you have a deep neural network with 10 input neurons, one hidden layer with 50 neurons, and one output layer with 3 neurons. All neurons use the hyperbolic tangent as activation.

(a) What are the dimensions of a pair of feature and target variables $\mathbf{x_i}$ and $\mathbf{y_i}$? (*0.5 points*)

(b) What are the dimensions of the first weight matrix $\mathbf{w_1}$ and the corresponding bias vector $\mathbf{b_1}$? (*0.5 points*)

(c) What are the dimensions of the weight matrix $\mathbf{w_2}$ and the bias vector $\mathbf{b_2}$ of the output layer? (*0.5 points*)

(d) Write down the equation to compute $\mathbf{y_i}$. (*0.5 points*)

(e) How many trainable parameters does this network have? (*0.5 points*)

ANSWER

(a) The dimensions of (x_i, y_i) are $(10 \times 1, 3 \times 1)$.

(b) The dimensions of w_1 is 50×10 and the dimension of b_1 is 50×1.

(c) The dimensions of w_2 is 3×50 and the dimension of b_2 is 3×1.

(d) $z_1 = w_1 x_i + b_1, a_1 = tanh(z_1), z_2 = w_2 a_1 + b_2, y_i = tanh(z_2)$ where $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

(e) There are totally 703 features.

$50 \times (10 + 1) + 3 \times (50 + 1) = 703$

In [23]:

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
model_simple = Sequential()
model_simple.add(Dense(50, activation='relu', input_shape=(10,)))# the hidden has 5
model_simple.add(Dense(3, activation='softmax'))#the output layer has 3 neurons
model_simple.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_22 (Dense)             (None, 50)                550       
_________________________________________________________________
dense_23 (Dense)             (None, 3)                 153       
=================================================================
Total params: 703
Trainable params: 703
Non-trainable params: 0
_________________________________________________________________
```

2) Classification with a deep neural network

(5 points)

(a) Create 1000 training and 400 test data points with the make_moons function from Scikit Learn. Set the noise level to 0.2. (*0.5 points*)

(b) Design a neural network using Keras. The first hidden layer has 100 neurons with rectified linear units as activation. The second hidden layer has 25 neurons and also rectified linear units as activation. The output layer uses sigmoid activation. The loss function is binary crossentropy, the gradient descent method is Adam and the metric used for evaluation is accuracy. (*1 point*)

(c) Train the network with a batch size of 64 and early stopping if the validation loss does not change over 4 epochs. Report the test accuracy. (*1 point*)

(d) Plot the test data points together with a mesh indicating the prediction of the neural network. You can reuse the code from notebook 04_07. (*1 point*)

(e) Make two figures showing the evolution of loss and accuracy as a function of number of epochs. In both figures report training and test results.(*0.5 point*)

(f) Describe *in words* how the results change (if at all) when you change the batch size to 32 and 128. (*0.5 points*)

(g) Describe *in words* how the results change (if at all) when you change the the activation of the two hidden layers to sigmoid. (*0.5 points*)

ANSWER

In [24]:

```
#(a) create data points
from sklearn.datasets import make_moons
trainx,trainy=make_moons(n_samples=1000,noise=0.2) #training data
testx,testy=make_moons(n_samples=400,noise=0.2) #test data
```

In [25]:

```
trainx.shape
```

Out[25]:

```
(1000, 2)
```

In [26]:

```
#(b) create neural networks
from keras.optimizers import RMSprop, Adam

model=Sequential()
model.add(Dense(100,activation='relu',input_shape=(2,)))# input shape is (2,)
                                                        #because the shape of trainx is
model.add(Dense(25,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer=Adam(),metrics=['accuracy'])
```

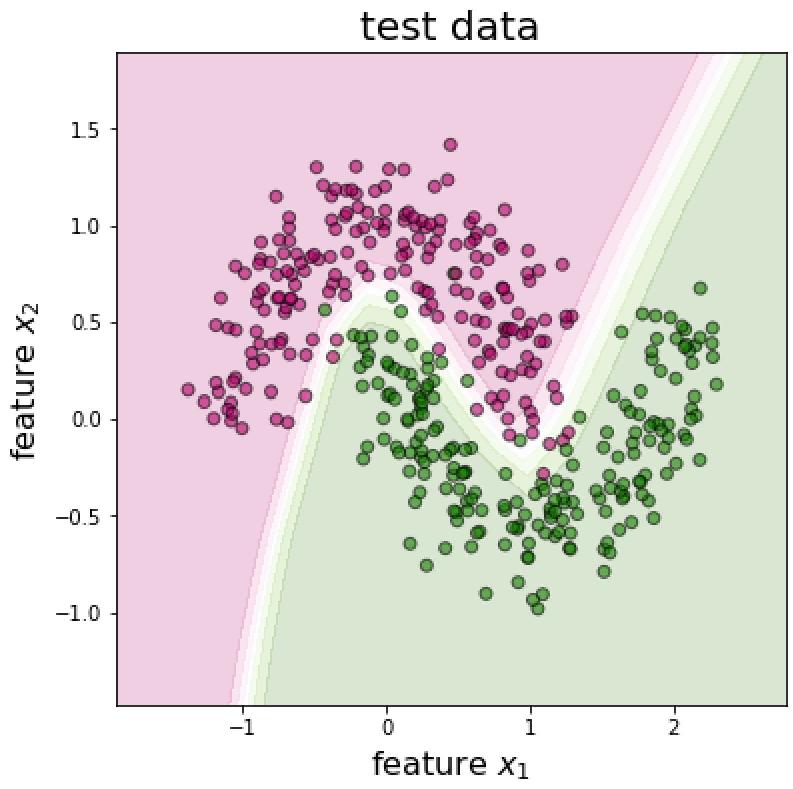
In [27]:

```
#(c) train the network
batch_size=64
epochs=100# set it to 100 because I tried several numbers and I find 100 is the safe
# the model should stop training when it won't improve anymore
from keras.callbacks import EarlyStopping
early_stopping_monitor=EarlyStopping(monitor='val_loss', patience=4)# not change ove
history_simple=model.fit(trainx,trainy,batch_size=batch_size,epochs=epochs,verbose=
                    validation_data=(testx,testy))

# evaluate model performance
score = model.evaluate(testx,testy,verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Epoch 57/100
1000/1000 [==============================] - 0s 46us/step - loss: 0.1
044 - accuracy: 0.9630 - val_loss: 0.0866 - val_accuracy: 0.9700
Epoch 58/100
1000/1000 [==============================] - 0s 43us/step - loss: 0.1
046 - accuracy: 0.9620 - val_loss: 0.0833 - val_accuracy: 0.9725
Epoch 59/100
1000/1000 [==============================] - 0s 44us/step - loss: 0.1
039 - accuracy: 0.9630 - val_loss: 0.0865 - val_accuracy: 0.9700
Epoch 60/100
1000/1000 [==============================] - 0s 46us/step - loss: 0.1
006 - accuracy: 0.9660 - val_loss: 0.0840 - val_accuracy: 0.9725
Epoch 61/100
1000/1000 [==============================] - 0s 48us/step - loss: 0.1
011 - accuracy: 0.9620 - val_loss: 0.0856 - val_accuracy: 0.9700
Epoch 62/100
1000/1000 [==============================] - 0s 45us/step - loss: 0.0
992 - accuracy: 0.9630 - val_loss: 0.0836 - val_accuracy: 0.9725
Test loss: 0.08363314956426621
Test accuracy: 0.9725000262260437
```

In [28]:

```
#(d) plot the test data

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
# 1) determine boundaries
# compute the decision boundary on a grid, for later visualization

X=testx
x1_min,x1_max=X[:,0].min()-.5,X[:,0].max()+.5
x2_min,x2_max=X[:,1].min()-.5,X[:,1].max()+.5

# 2) create a mesh of size [x1_min, x1_max] x [x2_min, x2_max].
h=.02  # step size in the mesh
xx,yy=np.meshgrid(np.arange(x1_min,x1_max,h), np.arange(x2_min,x2_max,h))

# 3) assign logistic regression prediction to each mesh point
Z=model.predict(np.c_[xx.ravel(),yy.ravel()])

# prepare colormaps
cm=plt.cm.PiYG
cm_bright=ListedColormap(['#b30065','#178000'])

plt.figure(figsize=(6,6))
# plot the prediction result using the mesh
Z=Z.reshape(xx.shape)
plt.contourf(xx,yy,Z,cmap=cm, alpha=.2)

# and test points
plt.scatter(testx[:,0],testx[:,1],c=testy,cmap=cm_bright,edgecolors='k',alpha=0.6)

plt.xlabel('feature $x_1$',size=16)
plt.ylabel('feature $x_2$',size=16)
plt.title('test data',size=20)

plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
plt.show()
```



test data
feature x_2
feature x_1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2

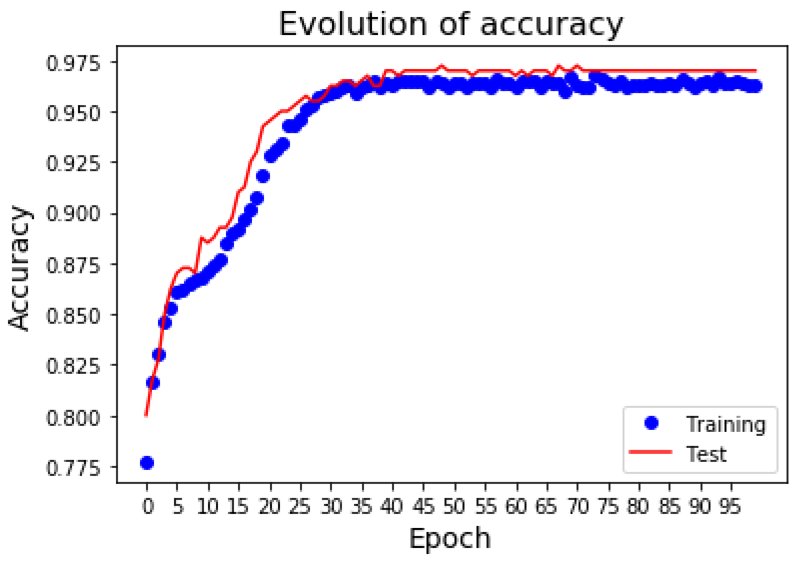
In [29]:

```
#(e)show the learning process
model=Sequential()
model.add(Dense(100,activation='relu',input_shape=(2,)))# input shape is (2,)
                                                    #because the shape of trainx is
model.add(Dense(25,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer=Adam(),metrics=['accuracy'])
batch_size=64
epochs=100

history=model.fit(trainx, trainy,batch_size=batch_size,epochs=epochs,verbose=1,valic

# evaluate model performance
score=model.evaluate(testx,testy,verbose=0)
print('Test loss:',score[0])
print('Test accuracy:',score[1])
```

```
1000/1000 [==============================] - 0s 53us/step - loss: 0.0
950 - accuracy: 0.9630 - val_loss: 0.0806 - val_accuracy: 0.9700
Epoch 88/100
1000/1000 [==============================] - 0s 55us/step - loss: 0.0
947 - accuracy: 0.9660 - val_loss: 0.0770 - val_accuracy: 0.9700
Epoch 89/100
1000/1000 [==============================] - 0s 57us/step - loss: 0.0
960 - accuracy: 0.9640 - val_loss: 0.0763 - val_accuracy: 0.9700
Epoch 90/100
1000/1000 [==============================] - 0s 54us/step - loss: 0.0
953 - accuracy: 0.9620 - val_loss: 0.0785 - val_accuracy: 0.9700
Epoch 91/100
1000/1000 [==============================] - 0s 54us/step - loss: 0.0
940 - accuracy: 0.9640 - val_loss: 0.0777 - val_accuracy: 0.9700
Epoch 92/100
1000/1000 [==============================] - 0s 54us/step - loss: 0.0
943 - accuracy: 0.9650 - val_loss: 0.0774 - val_accuracy: 0.9700
Epoch 93/100
1000/1000 [==============================] - 0s 52us/step - loss: 0.0
942 - accuracy: 0.9630 - val_loss: 0.0780 - val_accuracy: 0.9700
```

In [30]:

```
accuracy=history.history['accuracy']
val_accuracy=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(accuracy))
#accuracy
plt.plot(epochs,accuracy,'bo',label='Training')#training data
plt.plot(epochs,val_accuracy,'r',label='Test')#test data
plt.xlabel('Epoch',size=14)
plt.ylabel('Accuracy',size=14)
plt.title('Evolution of accuracy',size=16)
plt.xticks(np.arange(0,100,step=5))#because there are 100 epochs
plt.legend()
plt.show()
#loss
plt.figure()
plt.plot(epochs,loss,'bo',label='Training')#training
plt.plot(epochs,val_loss,'r',label='Test')#test
plt.xlabel('Epoch',size=14)
plt.ylabel('Loss',size=14)
plt.title('Evolution of loss function',size=16)
plt.xticks(np.arange(0,100,step=5))
plt.legend()
plt.show()
```



Evolution of loss function

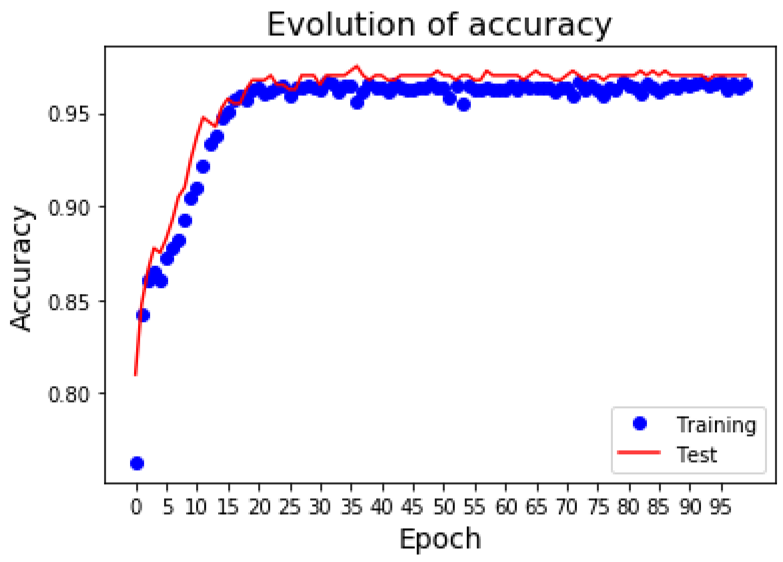
In [31]:

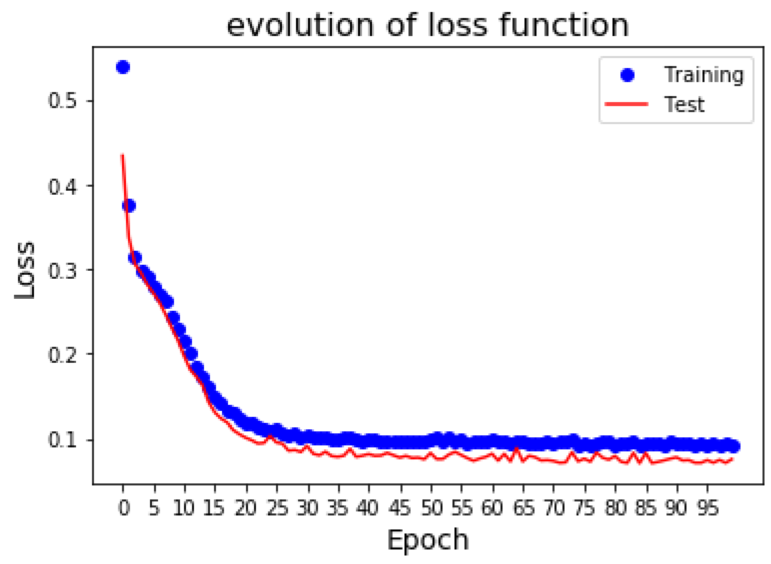
```
#(f) batch size change
#First draw the figures
model=Sequential()
model.add(Dense(100,activation='relu',input_shape=(2,)))# input shape is (2,)
                                                      #because the shape of trainx is
model.add(Dense(25,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer=Adam(),metrics=['accuracy'])
batch_size=32
epochs=100
history=model.fit(trainx,trainy,batch_size=batch_size,epochs=epochs,verbose=1,  val
# evaluate model performance
score=model.evaluate(testx,testy,verbose=0)
print('Test loss:',score[0])
print('Test accuracy:',score[1])
```

```
1000/1000 [==============================] - 0s 94us/step - loss: 0.0
942 - accuracy: 0.9660 - val_loss: 0.0713 - val_accuracy: 0.9700
Epoch 96/100
1000/1000 [==============================] - 0s 93us/step - loss: 0.0
920 - accuracy: 0.9670 - val_loss: 0.0745 - val_accuracy: 0.9700

Epoch 97/100
1000/1000 [==============================] - 0s 91us/step - loss: 0.0
946 - accuracy: 0.9620 - val_loss: 0.0714 - val_accuracy: 0.9700
Epoch 98/100
1000/1000 [==============================] - 0s 91us/step - loss: 0.0
916 - accuracy: 0.9660 - val_loss: 0.0748 - val_accuracy: 0.9700
Epoch 99/100
1000/1000 [==============================] - 0s 93us/step - loss: 0.0
951 - accuracy: 0.9640 - val_loss: 0.0712 - val_accuracy: 0.9700
Epoch 100/100
1000/1000 [==============================] - 0s 95us/step - loss: 0.0
920 - accuracy: 0.9660 - val_loss: 0.0756 - val_accuracy: 0.9700
Test loss: 0.07557605437934399
Test accuracy: 0.9700000286102295
```

In [32]:

```
accuracy=history.history['accuracy']
val_accuracy=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(accuracy))
plt.plot(epochs,accuracy,'bo',label='Training')
plt.plot(epochs,val_accuracy,'r',label='Test')
plt.xlabel('Epoch',size=14)
plt.ylabel('Accuracy',size=14)
plt.title('Evolution of accuracy',size=16)
plt.xticks(np.arange(0,100,step=5))
plt.legend()
plt.show()
plt.figure()
plt.plot(epochs,loss,'bo',label='Training')
plt.plot(epochs,val_loss,'r',label='Test')
plt.xlabel('Epoch',size=14)
plt.ylabel('Loss',size=14)
plt.title('evolution of loss function',size=16)
plt.xticks(np.arange(0,100,step=5))
plt.legend()
plt.show()
```





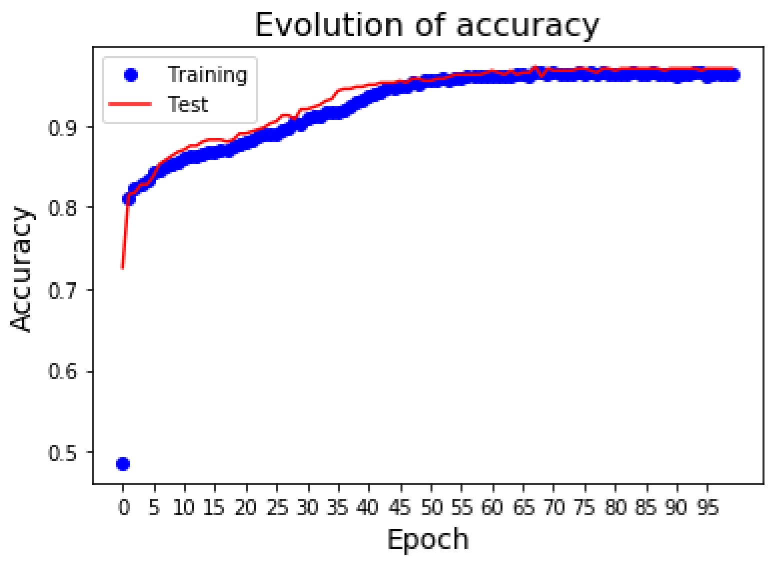
In [33]:

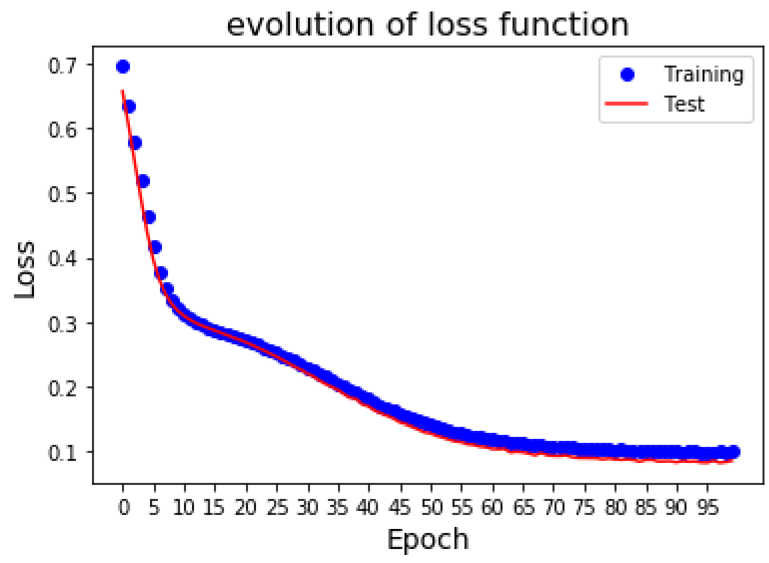
```
model=Sequential()
model.add(Dense(100,activation='relu',input_shape=(2,)))# input shape is (2,)
                                                #because the shape of trainx is
model.add(Dense(25,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer=Adam(),metrics=['accuracy'])

batch_size=128
epochs=100

history=model.fit(trainx,trainy,batch_size=batch_size,epochs=epochs,verbose=1,valida

# evaluate model performance
score=model.evaluate(testx,testy,verbose=0)
print('Test loss:',score[0])
print('Test accuracy:',score[1])
```

```
1000/1000 [==============================] - 0s 32us/step - loss: 0.0
985 - accuracy: 0.9660 - val_loss: 0.0826 - val_accuracy: 0.9675
Epoch 96/100
1000/1000 [==============================] - 0s 32us/step - loss: 0.0
985 - accuracy: 0.9610 - val_loss: 0.0823 - val_accuracy: 0.9700

Epoch 97/100
1000/1000 [==============================] - 0s 32us/step - loss: 0.0
979 - accuracy: 0.9640 - val_loss: 0.0853 - val_accuracy: 0.9700
Epoch 98/100
1000/1000 [==============================] - 0s 32us/step - loss: 0.0
991 - accuracy: 0.9640 - val_loss: 0.0821 - val_accuracy: 0.9700
Epoch 99/100
1000/1000 [==============================] - 0s 32us/step - loss: 0.0
981 - accuracy: 0.9640 - val_loss: 0.0833 - val_accuracy: 0.9700
Epoch 100/100
1000/1000 [==============================] - 0s 32us/step - loss: 0.0
993 - accuracy: 0.9640 - val_loss: 0.0854 - val_accuracy: 0.9700
Test loss: 0.08537426248192787
Test accuracy: 0.9700000286102295
```

In [34]:

```
accuracy=history.history['accuracy']
val_accuracy=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(accuracy))
plt.plot(epochs,accuracy,'bo',label='Training')
plt.plot(epochs,val_accuracy,'r',label='Test')
plt.xlabel('Epoch',size=14)
plt.ylabel('Accuracy',size=14)
plt.title('Evolution of accuracy',size=16)
plt.xticks(np.arange(0,100,step=5))
plt.legend()
plt.show()
plt.figure()
plt.plot(epochs,loss,'bo',label='Training')
plt.plot(epochs,val_loss,'r',label='Test')
plt.xlabel('Epoch',size=14)
plt.ylabel('Loss',size=14)
plt.title('evolution of loss function',size=16)
plt.xticks(np.arange(0,100,step=5))
plt.legend()
plt.show()
```





Note: everytime we need to rebuild the model to avoid the model is based on the training results before.

[illegible]

The results of the accuracy and loss are almost the same but the figure with the batch size 128 seems to be more flat and in the case of batch size as 32, 64 and 128, when the batch size gets larger, the learning process curve seems more smooth. The mesh figures seem almost the same.

From the Internet, I find that batch size will also have effect in the time. When the batch size is too small it will take a lot of time, but here it seems there is no much difference. Meanwhile, the gradient oscillation will be serious, which is not conducive to convergence. If the batch size is too large, the gradient direction of different batches does not change at all, so it is easy to fall into local minimum value.

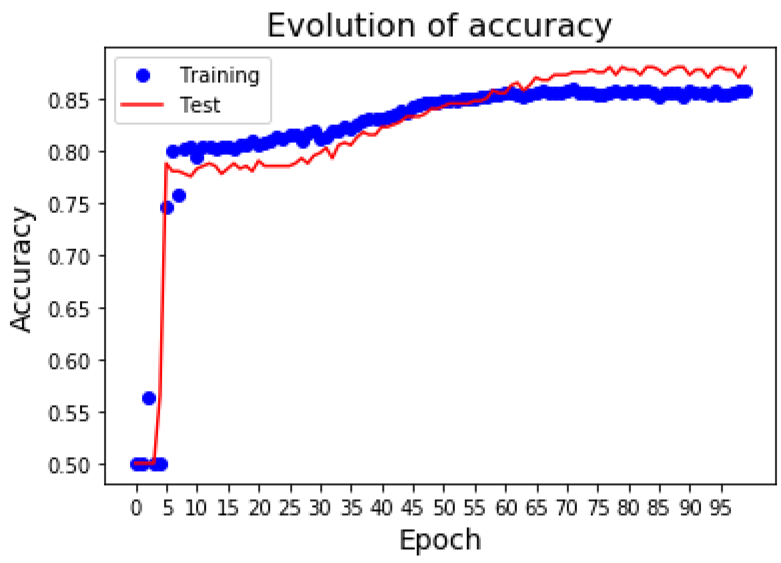
In [35]:

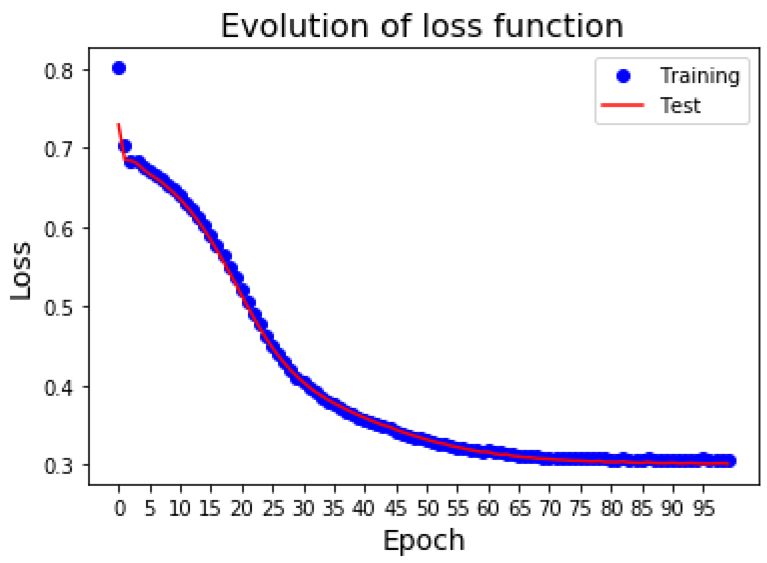
```
# (g) activation change
modelsig=Sequential()
modelsig.add(Dense(100,activation='sigmoid',input_shape=(2,)))
modelsig.add(Dense(25,activation='sigmoid'))
modelsig.add(Dense(1,activation='sigmoid'))
modelsig.compile(loss='binary_crossentropy',optimizer=Adam(),metrics=['accuracy'])
batch_size=128# compare it with above 128 batch size 100 epochs with relu activatio
epochs=100
history=modelsig.fit(trainx,trainy,batch_size=batch_size,epochs=epochs,verbose=1,val
# evaluate modelsig performance
score=modelsig.evaluate(testx,testy,verbose=0)
print('Test loss:',score[0])
print('Test accuracy:',score[1])
```

```
1000/1000 [==============================] - 0s 31us/step - loss: 0.3
066 - accuracy: 0.8570 - val_loss: 0.3017 - val_accuracy: 0.8775
Epoch 96/100
1000/1000 [==============================] - 0s 32us/step - loss: 0.3
076 - accuracy: 0.8530 - val_loss: 0.3015 - val_accuracy: 0.8800

Epoch 97/100
1000/1000 [==============================] - 0s 32us/step - loss: 0.3
063 - accuracy: 0.8540 - val_loss: 0.3016 - val_accuracy: 0.8775
Epoch 98/100
1000/1000 [==============================] - 0s 32us/step - loss: 0.3
061 - accuracy: 0.8560 - val_loss: 0.3019 - val_accuracy: 0.8775
Epoch 99/100
1000/1000 [==============================] - 0s 31us/step - loss: 0.3
067 - accuracy: 0.8580 - val_loss: 0.3023 - val_accuracy: 0.8700
Epoch 100/100
1000/1000 [==============================] - 0s 30us/step - loss: 0.3
061 - accuracy: 0.8580 - val_loss: 0.3014 - val_accuracy: 0.8800
Test loss: 0.30141878545284273
Test accuracy: 0.8799999952316284
```

In [36]:

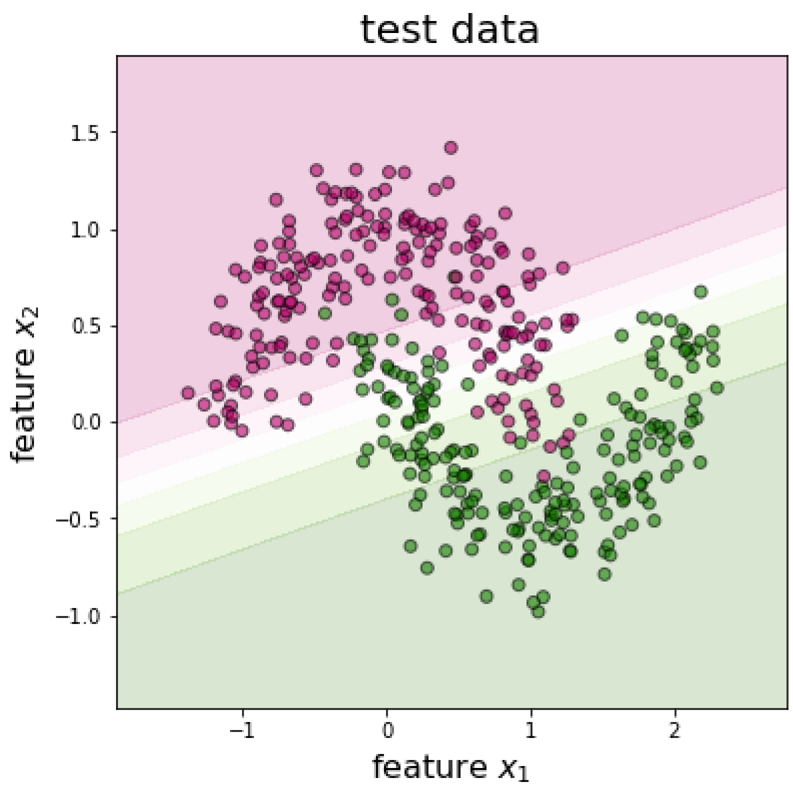
```
accuracy=history.history['accuracy']
val_accuracy=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(accuracy))
plt.plot(epochs,accuracy,'bo',label='Training')
plt.plot(epochs,val_accuracy,'r',label='Test')
plt.xlabel('Epoch',size=14)
plt.ylabel('Accuracy',size=14)
plt.title('Evolution of accuracy',size=16)
plt.xticks(np.arange(0, 100, step=5))
plt.legend()
plt.show()
plt.figure()
plt.plot(epochs,loss, 'bo',label='Training')
plt.plot(epochs,val_loss,'r',label='Test')
plt.xlabel('Epoch',size=14)
plt.ylabel('Loss',size=14)
plt.title('Evolution of loss function',size=16)
plt.xticks(np.arange(0,100,step=5))
plt.legend()
plt.show()
```




Evolution of loss function
Training
Test
Loss
Epoch
0.3
0.4
0.5
0.6
0.7
0.8
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95

In [38]:

```
X=testx
x1_min,x1_max=X[:,0].min()-.5,X[:,0].max()+.5
x2_min,x2_max=X[:,1].min()-.5,X[:,1].max()+.5

# 2) create a mesh of size [x1_min, x1_max] x [x2_min, x2_max].
h=.02   # step size in the mesh
xx,yy=np.meshgrid(np.arange(x1_min,x1_max,h), np.arange(x2_min,x2_max,h))

# 3) assign logistic regression prediction to each mesh point
Z=modelsig.predict(np.c_[xx.ravel(),yy.ravel()])

# prepare colormaps
cm=plt.cm.PiYG
cm_bright=ListedColormap(['#b30065','#178000'])

plt.figure(figsize=(6,6))
# plot the prediction result using the mesh
Z=Z.reshape(xx.shape)
plt.contourf(xx,yy,Z,cmap=cm, alpha=.2)

# and test points
plt.scatter(testx[:,0],testx[:,1],c=testy,cmap=cm_bright,edgecolors='k',alpha=0.6)

plt.xlabel('feature $x_1$',size=16)
plt.ylabel('feature $x_2$',size=16)
plt.title('test data',size=20)

plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
plt.show()
```



When other things remain the same, the accuracy will decrease and the loss will get larger (I tried it with batch size 32, 64 and 128, for convenience here I only show the result of 128). The trend of loss and accuracy of the test data with epoch seems to be more alikely to the training data. And the mesh with sigmoid function becomes straight lines.

I think it is becuase the sigmoid function compress the data into the interval $(0, 1)$ and the ReLu function drop some data.

3) Demonstration of the universal approximation theorem

(2.5 points total)

(a) Write a function which computes $f(x) = 0.2 + 0.4\ x^2 + 0.3\ x\ sin(9x)$. Create a vector x_train which contains 10000 evenly spaced points between 0 and 1. Compute the vector y_train = f(x_train). Plot y_train versus x_train. This is the function we want to approximate with a neural network containing one hidden layer. (*0.5 points*)

(b) Create a neural network with one input neuron, a hidden layer with 50 neurons and sigmoid activation and one output neuron with linear activation. Choose Mean Squared Error as loss function and Adam(lr=0.005) as gradient descent method. Train the model with a batch size of 2000 for 4000 epochs. (We do not need test data in this demonstration.)

After training the network make a prediction using x_train and plot this prediction together with y_train (i.e. the function the network tries to approximate). (*1.5 points*)

(c) Plot the evolution of the loss function with a logarithmic y-axis. Describe *in words* how this curve changes when you change the hyperparameter learning rate for Adam to lr=0.002. (*0.5 points*)

ANSWER

In [15]:

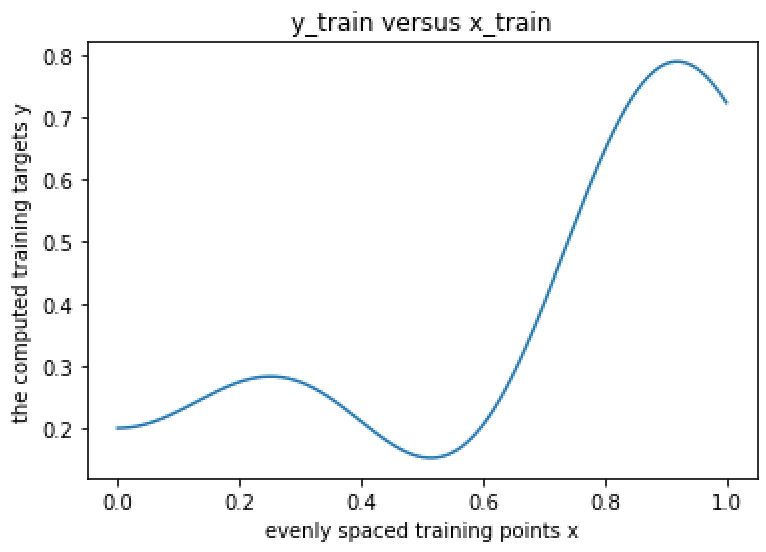
```
#(a) write the function and build training data
import math
def f(x):
    fx=0.2+.4*x*x+0.3*x*math.sin(9*x)
    return fx
```

In [16]:

```
%config InlineBackend.figure_format = 'retina'
x_train=np.linspace(0,1,num=10000)#crate training data
y_train=[]
for i in x_train:
    y_train.append(f(i))

plt.plot(x_train,y_train)
plt.xlabel('evenly spaced training points x')
plt.ylabel('the computed training targets y')
plt.title('y_train versus x_train')
```

Out[16]:

```
Text(0.5, 1.0, 'y_train versus x_train')
```



In [17]:

```
#(b) build neural netwoek
model=Sequential()
model.add(Dense(50,activation='sigmoid',input_shape=(1,)))
model.add(Dense(1,activation='linear'))
model.compile(loss='mean_squared_error',optimizer=Adam(lr=0.005))
# train the model
batch_size=2000
epochs=4000

# the model should stop training when it won't improve anymore
from keras.callbacks import EarlyStopping
early_stopping_monitor=EarlyStopping(monitor='val_loss', patience=4)

history=model.fit(x_train,y_train,batch_size=batch_size,epochs=epochs)
```

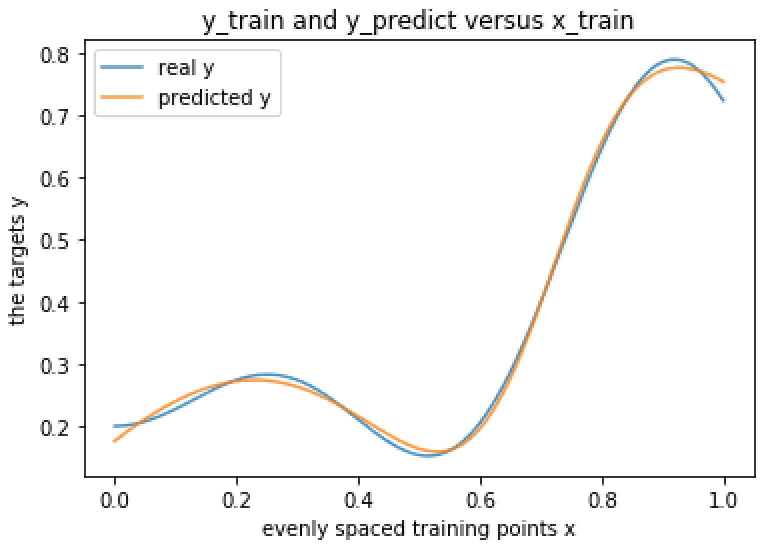
```
10000/10000 [==============================] - 0s 3us/step - loss: 9.
8831e-05
Epoch 3466/4000
10000/10000 [==============================] - 0s 2us/step - loss: 1.
0936e-04
Epoch 3467/4000
10000/10000 [==============================] - 0s 3us/step - loss: 9.
```

In [18]:

```
y_predict=model.predict(x_train)
plt.plot(x_train,y_train,label='real y',alpha=0.8)
plt.plot(x_train,y_predict,label='predicted y',alpha=0.8)
plt.xlabel('evenly spaced training points x')
plt.ylabel('the targets y')
plt.title('y_train and y_predict versus x_train')
plt.legend()
```

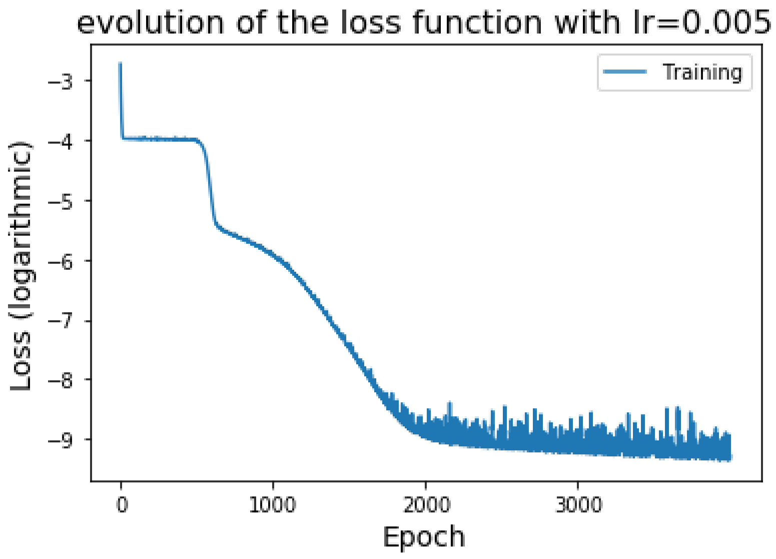
Out[18]:

```
<matplotlib.legend.Legend at 0x1a4ef0b410>
```



In [19]:

```
#(c) the evolution
loss = history.history['loss']
epochs = range(len(loss))

plt.plot(epochs, np.log(loss),label='Training')
plt.xlabel('Epoch',size=14)
plt.ylabel('Loss (logarithmic)',size=14)
plt.title('evolution of the loss function with lr=0.005',size=16)
plt.xticks(np.arange(0,4000,step=1000))
plt.legend()
plt.show()
```



In [20]:

```
modelc=Sequential()
modelc.add(Dense(50,activation='sigmoid',input_shape=(1,)))
modelc.add(Dense(1,activation='linear'))
modelc.compile(loss='mean_squared_error',optimizer=Adam(lr=0.002))# change the lr to
```

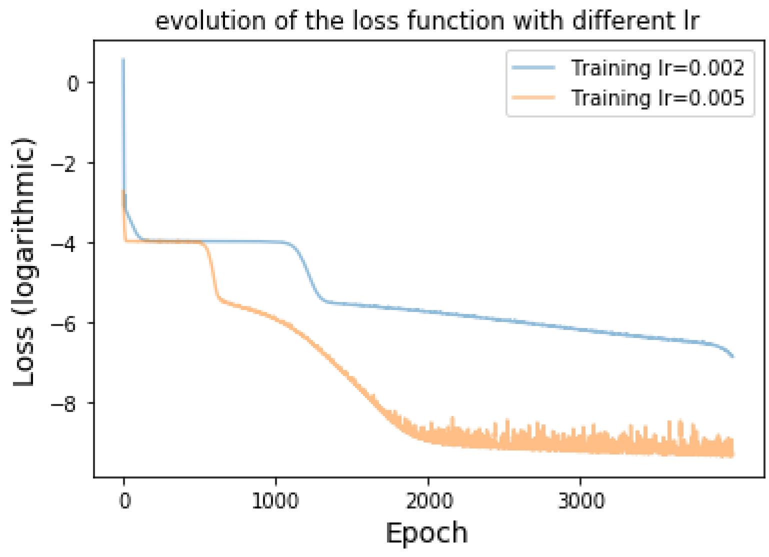
In [21]:

```
# train the model
batch_size=2000
epochs=4000
historyc=modelc.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1)
```

```
Epoch 3545/4000
10000/10000 [==============================] - 0s 2us/step - loss: 0.
0016
Epoch 3546/4000
10000/10000 [==============================] - 0s 2us/step - loss: 0.
0016
Epoch 3547/4000
10000/10000 [==============================] - 0s 2us/step - loss: 0.
0016
Epoch 3548/4000
10000/10000 [==============================] - 0s 2us/step - loss: 0.
0016
Epoch 3549/4000
10000/10000 [==============================] - 0s 2us/step - loss: 0.
0016
Epoch 3550/4000
10000/10000 [==============================] - 0s 2us/step - loss: 0.
0016
Epoch 3551/4000
10000/10000 [==============================] - 0s 2us/step - loss: 0.
```

In [22]:

```
lossc=historyc.history['loss']
epochs=range(len(loss))
plt.plot(epochs, np.log(lossc),label='Training lr=0.002',alpha=0.5)
plt.plot(epochs, np.log(loss),label='Training lr=0.005',alpha=0.5)
plt.xlabel('Epoch',size=14)
plt.ylabel('Loss (logarithmic)', size=14)
plt.title('evolution of the loss function with different lr')
plt.xticks(np.arange(0,4000,step=1000))
plt.legend()
plt.show()
```



It seems that when lr=0.002, the loss seems larger and the curve are centered when the number of epoches gets larger and the loss value seems decrease faster when lr=0.005.

I think it is because the learing rate 0.005 is too big so it causes the float when theremare more epochs and the learning rate 0.002 is too small so the loss decreases slowly and it has no chance to float.