Практическая работа №2

Выполнил студент группы БВТ2003 Глазков Даниил

Задание: Необходимо дополнить следующий фрагмент кода моделью ИНС, которая способна провести бинарную классификацию по сгенерированным данным.

In [1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib.colors as mclr
from tensorflow.keras import layers
from tensorflow.keras import models
```

In [2]:
```
def genData(size=500):
    size1 = size//2
    size2 = size - size1

    t1 = np.random.rand(size1)
    x1 = np.asarray([i * math.cos(i*2*math.pi) + (np.random.rand(1)-1)/2*i for i
    y1 = np.asarray([i * math.sin(i*2*math.pi) + (np.random.rand(1)-1)/2*i for i
    data1 = np.hstack((x1, y1))
    label1 = np.zeros([size1, 1])
    div1 = round(size1*0.8)

    t2 = np.random.rand(size2)
    x2 = np.asarray([-i * math.cos(i*2*math.pi) + (np.random.rand(1)-1)/2*i for
    y2 = np.asarray([-i * math.sin(i*2*math.pi) + (np.random.rand(1)-1)/2*i for
    data2 = np.hstack((x2, y2))
    label2 = np.ones([size2, 1])
    div2 = round(size2*0.8)

    div = div1 + div2

    order = np.random.permutation(div)

    train_data = np.vstack((data1[:div1], data2[:div2]))
    test_data = np.vstack((data1[div1:], data2[div2:]))
    train_label = np.vstack((label1[:div1], label2[:div2]))
    test_label = np.vstack((label1[div1:], label2[div2:]))

    return (train_data[order, :], train_label[order, :]), (test_data, test_label
```

In [3]:
```
def drawResults(data, label, prediction):
    p_label = np.array([round(x[0]) for x in prediction])
    plt.scatter(data[:, 0], data[:, 1], s=30, c=label[:, 0], cmap=mclr.ListedCol
    plt.scatter(data[:, 0], data[:, 1], s=10, c=p_label, cmap=mclr.ListedColorma
    plt.grid()
    plt.show()
(train_data, train_label), (test_data, test_label) = genData()
```

In [4]:
```
#Создание базовой архетектуры сети(четыре слоя)
model1 = models.Sequential()
model1.add(layers.Dense(16, activation='relu'))
```

```
model1.add(layers.Dense(16, activation='relu'))
model1.add(layers.Dense(16, activation='relu'))
model1.add(layers.Dense(1, activation='sigmoid'))
```

In [5]:
```
#Инициализация параметров обучения
model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'
```

In [6]:
```
#Обучение модели через метод fit()
H1 = model1.fit(train_data, train_label, epochs=50, batch_size=10, validation_sp
```

```
Epoch 1/50
36/36 [==============================] - 1s 8ms/step - loss: 0.6879 - accuracy:
0.4639 - val_loss: 0.6775 - val_accuracy: 0.6000
Epoch 2/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6782 - accuracy:
0.5361 - val_loss: 0.6642 - val_accuracy: 0.6250
Epoch 3/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6673 - accuracy:
0.5528 - val_loss: 0.6513 - val_accuracy: 0.6250
Epoch 4/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6552 - accuracy:
0.5972 - val_loss: 0.6328 - val_accuracy: 0.6000
Epoch 5/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6375 - accuracy:
0.6389 - val_loss: 0.6078 - val_accuracy: 0.6500
Epoch 6/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6141 - accuracy:
0.6694 - val_loss: 0.5778 - val_accuracy: 0.7250
Epoch 7/50
36/36 [==============================] - 0s 3ms/step - loss: 0.5862 - accuracy:
0.6806 - val_loss: 0.5492 - val_accuracy: 0.7000
Epoch 8/50
36/36 [==============================] - 0s 3ms/step - loss: 0.5569 - accuracy:
0.7389 - val_loss: 0.5218 - val_accuracy: 0.6500
Epoch 9/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5236 - accuracy:
0.7250 - val_loss: 0.4952 - val_accuracy: 0.6500
Epoch 10/50
36/36 [==============================] - 0s 3ms/step - loss: 0.4893 - accuracy:
0.7528 - val_loss: 0.4746 - val_accuracy: 0.7500
Epoch 11/50
36/36 [==============================] - 0s 3ms/step - loss: 0.4590 - accuracy:
0.7778 - val_loss: 0.4541 - val_accuracy: 0.7500
Epoch 12/50
36/36 [==============================] - 0s 3ms/step - loss: 0.4247 - accuracy:
0.8056 - val_loss: 0.4181 - val_accuracy: 0.7750
Epoch 13/50
36/36 [==============================] - 0s 3ms/step - loss: 0.3901 - accuracy:
0.8500 - val_loss: 0.3861 - val_accuracy: 0.7750
Epoch 14/50
36/36 [==============================] - 0s 3ms/step - loss: 0.3598 - accuracy:
0.8861 - val_loss: 0.3556 - val_accuracy: 0.7750
Epoch 15/50
36/36 [==============================] - 0s 3ms/step - loss: 0.3321 - accuracy:
0.8667 - val_loss: 0.3405 - val_accuracy: 0.8250
Epoch 16/50
36/36 [==============================] - 0s 3ms/step - loss: 0.3006 - accuracy:
0.9083 - val_loss: 0.2982 - val_accuracy: 0.8500
Epoch 17/50
36/36 [==============================] - 0s 3ms/step - loss: 0.2731 - accuracy:
0.9111 - val_loss: 0.2852 - val_accuracy: 0.9000
Epoch 18/50
36/36 [==============================] - 0s 3ms/step - loss: 0.2516 - accuracy:
0.9333 - val_loss: 0.2559 - val_accuracy: 0.9000
Epoch 19/50
36/36 [==============================] - 0s 3ms/step - loss: 0.2269 - accuracy:
0.9500 - val_loss: 0.2231 - val_accuracy: 0.9250
Epoch 20/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1991 - accuracy:
0.9556 - val_loss: 0.2012 - val_accuracy: 0.9250
```

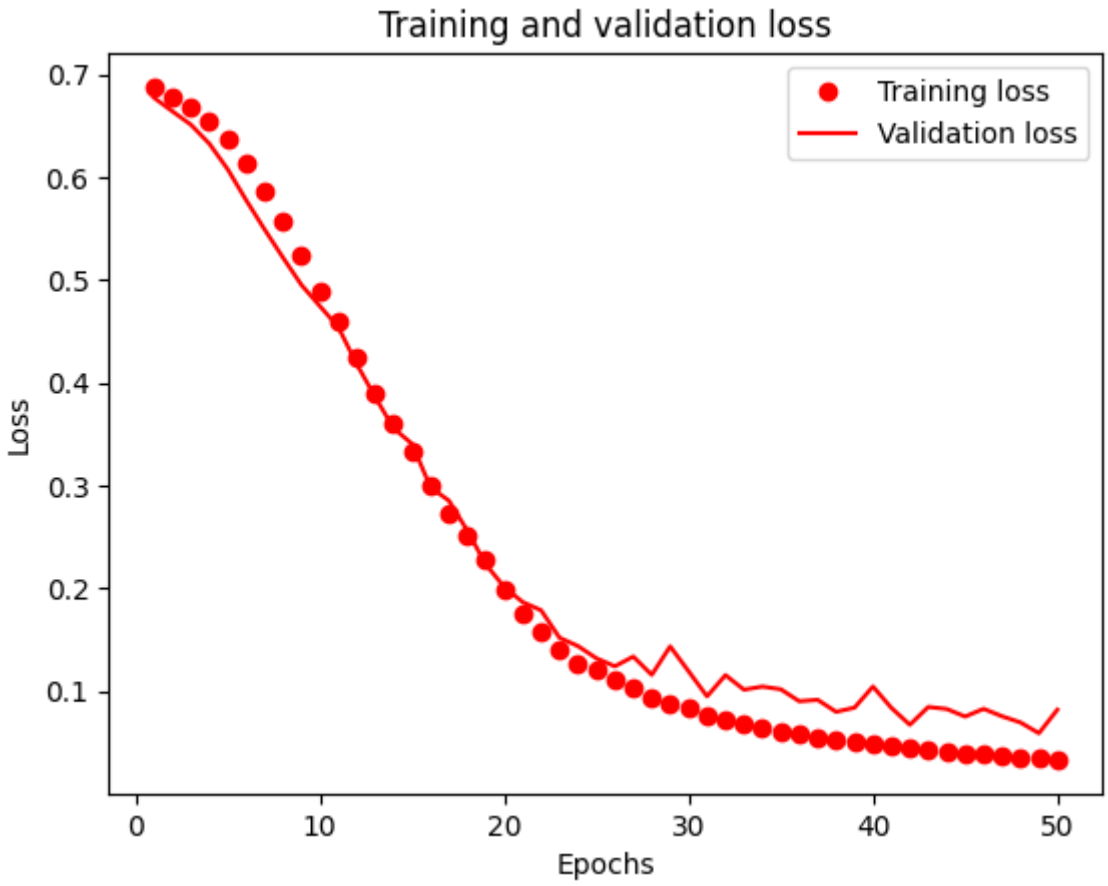
```
Epoch 21/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1753 - accuracy:
0.9806 - val_loss: 0.1864 - val_accuracy: 0.9750
Epoch 22/50
36/36 [==============================] - 0s 2ms/step - loss: 0.1573 - accuracy:
0.9806 - val_loss: 0.1785 - val_accuracy: 0.9750
Epoch 23/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1410 - accuracy:
0.9806 - val_loss: 0.1517 - val_accuracy: 0.9750
Epoch 24/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1271 - accuracy:
0.9750 - val_loss: 0.1436 - val_accuracy: 0.9750
Epoch 25/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1204 - accuracy:
0.9750 - val_loss: 0.1317 - val_accuracy: 0.9750
Epoch 26/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1102 - accuracy:
0.9861 - val_loss: 0.1239 - val_accuracy: 0.9750
Epoch 27/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1021 - accuracy:
0.9667 - val_loss: 0.1334 - val_accuracy: 0.9750
Epoch 28/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0933 - accuracy:
0.9889 - val_loss: 0.1156 - val_accuracy: 0.9750
Epoch 29/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0879 - accuracy:
0.9833 - val_loss: 0.1433 - val_accuracy: 0.9750
Epoch 30/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0834 - accuracy:
0.9806 - val_loss: 0.1191 - val_accuracy: 0.9750
Epoch 31/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0758 - accuracy:
0.9861 - val_loss: 0.0947 - val_accuracy: 0.9750
Epoch 32/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0710 - accuracy:
0.9917 - val_loss: 0.1154 - val_accuracy: 0.9750
Epoch 33/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0682 - accuracy:
0.9944 - val_loss: 0.1012 - val_accuracy: 0.9750
Epoch 34/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0640 - accuracy:
0.9833 - val_loss: 0.1042 - val_accuracy: 0.9750
Epoch 35/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0600 - accuracy:
0.9889 - val_loss: 0.1016 - val_accuracy: 0.9750
Epoch 36/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0577 - accuracy:
0.9889 - val_loss: 0.0899 - val_accuracy: 0.9750
Epoch 37/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0550 - accuracy:
0.9889 - val_loss: 0.0913 - val_accuracy: 0.9750
Epoch 38/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0532 - accuracy:
0.9917 - val_loss: 0.0796 - val_accuracy: 0.9750
Epoch 39/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0505 - accuracy:
0.9917 - val_loss: 0.0837 - val_accuracy: 0.9750
Epoch 40/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0481 - accuracy:
0.9917 - val_loss: 0.1043 - val_accuracy: 0.9750
```

```
Epoch 41/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0470 - accuracy:
0.9889 - val_loss: 0.0834 - val_accuracy: 0.9750
Epoch 42/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0443 - accuracy:
0.9972 - val_loss: 0.0670 - val_accuracy: 0.9750
Epoch 43/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0429 - accuracy:
0.9889 - val_loss: 0.0844 - val_accuracy: 0.9750
Epoch 44/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0411 - accuracy:
0.9889 - val_loss: 0.0824 - val_accuracy: 0.9750
Epoch 45/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0391 - accuracy:
0.9972 - val_loss: 0.0750 - val_accuracy: 0.9750
Epoch 46/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0387 - accuracy:
0.9917 - val_loss: 0.0826 - val_accuracy: 0.9750
Epoch 47/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0368 - accuracy:
0.9972 - val_loss: 0.0753 - val_accuracy: 0.9750
Epoch 48/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0350 - accuracy:
0.9917 - val_loss: 0.0693 - val_accuracy: 0.9750
Epoch 49/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0348 - accuracy:
0.9889 - val_loss: 0.0588 - val_accuracy: 1.0000
Epoch 50/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0331 - accuracy:
0.9944 - val_loss: 0.0817 - val_accuracy: 0.9750
```

In [7]:
```
loss1 = H1.history['loss']
val_loss1 = H1.history['val_loss']
acc1 = H1.history['accuracy']
val_acc1 = H1.history['val_accuracy']
epochs1 = range(1, len(loss1) + 1)
```

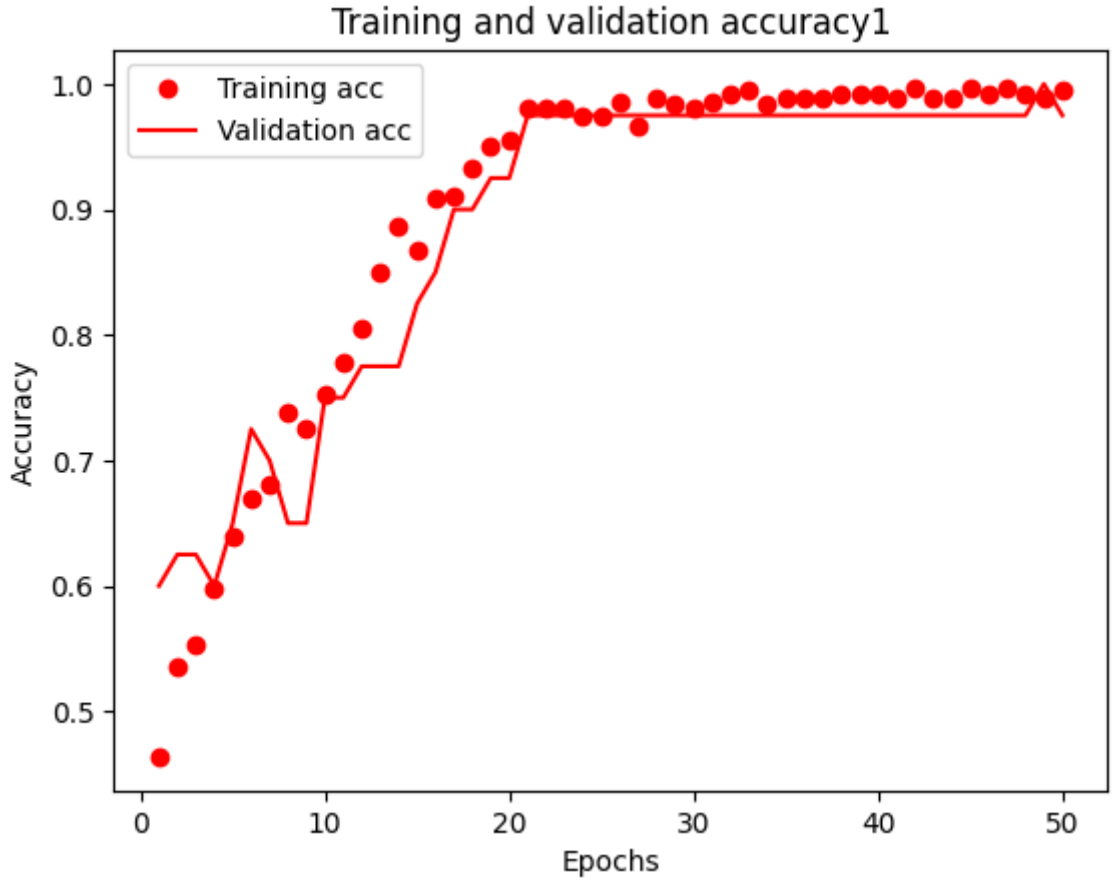
In [8]:
```
#Построение графика ошибок
plt.plot(epochs1, loss1, 'ro', label='Training loss')
plt.plot(epochs1, val_loss1, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



In [9]:
```
#Построение графика точности
plt.plot(epochs1, acc1, 'ro', label='Training acc')
plt.plot(epochs1, val_acc1, 'r', label='Validation acc')
plt.title('Training and validation accuracy1')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

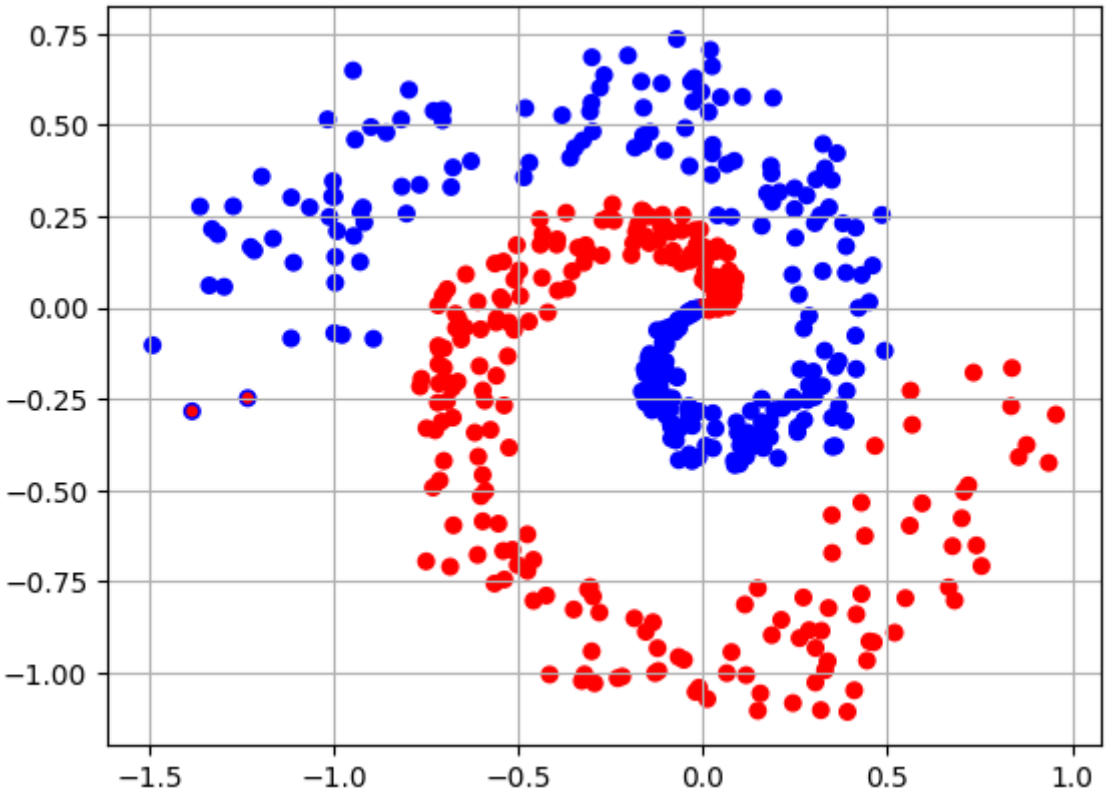


In [10]:
```
#Получение и вывод результатов на тестовом наборе
results1 = model1.evaluate(test_data, test_label)
print(results1)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.0560 - accuracy: 0.9
900
[0.05601692199707031, 0.9900000095367432]
```

In [11]:
```
#Вывод результатов бинарной классификации
all_data = np.vstack((train_data, test_data))
all_label = np.vstack((train_label, test_label))
pred1 = model1.predict(all_data)
drawResults(all_data, all_label, pred1)
```

```
16/16 [==============================] - 0s 1ms/step
```


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.5
−1.0
−0.5
0.0
0.5
1.0