Лабораторная работа №3

Многослойные сети. Алгоритм обратного распространения ошибки

Выполнил Попов Матвей

Группа М8О-**408**Б-**20**

Вариант **21**

Цель работы

Исследование свойств многослойной нейронной сети прямого распространения и алгоритмов ее обучения, применение сети в задачах классификации и аппроксимации функции.

Задание 1

Использовать многослойную нейронную сеть для классификации точек в случае, когда классы не являются линейно разделимыми.

In [1]:

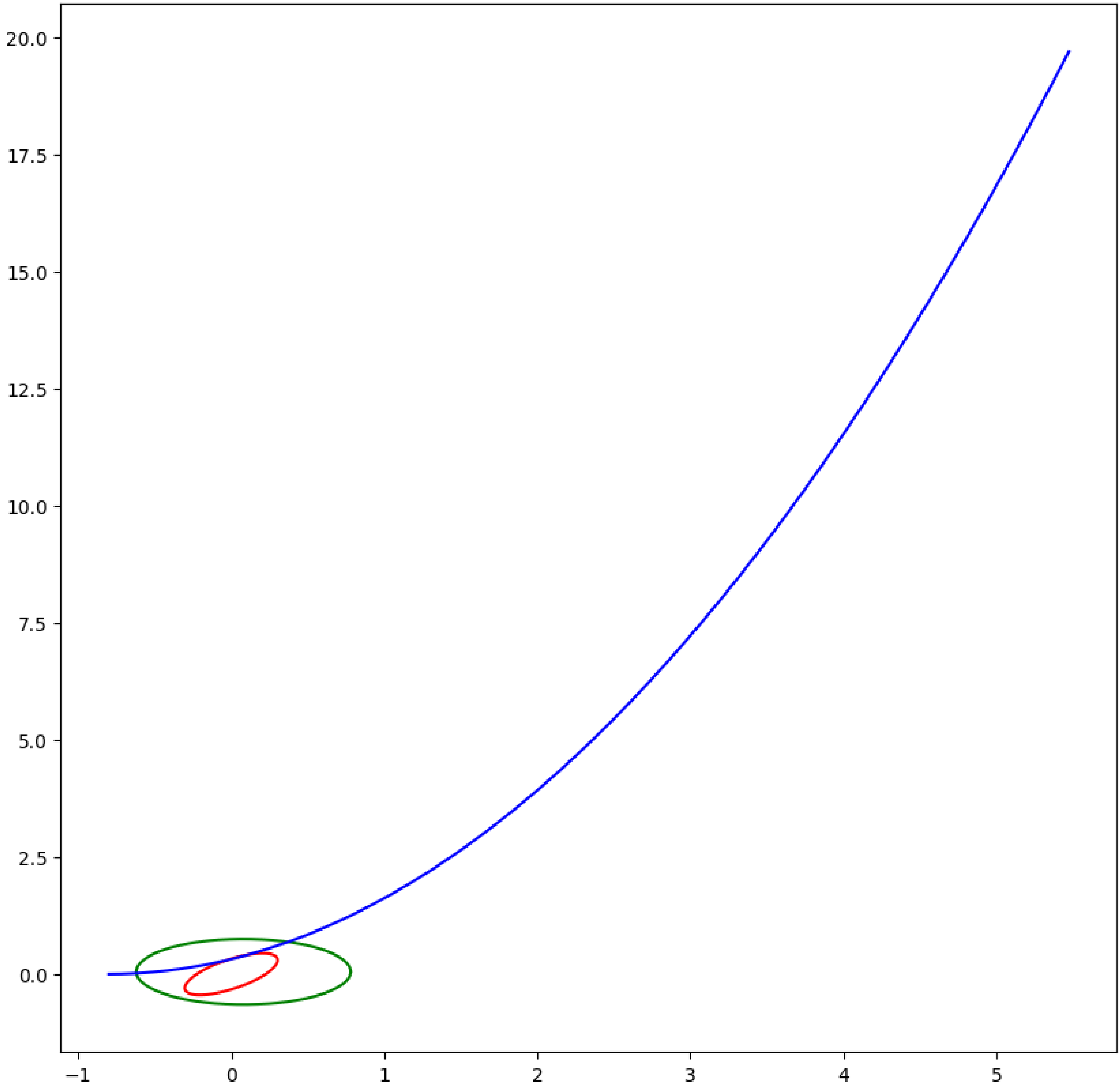
```
import math

figures = [
    {'a': 0.5, 'b': 0.2, 'alpha': math.pi / 3, 'x0': 0, 'y0': 0}, # эллипс
    {'a': 0.7, 'b': 0.7, 'alpha': 0, 'x0': 0.08, 'y0': 0.05},      # эллипс
    {'p': -1, 'alpha': -math.pi / 2, 'x0': 0, 'y0': -0.8}          # парабола
]
```

Изобразим заданные фигуры на графике

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt

def ellipse(t, a, b, x0, y0):
    x = x0 + a * np.cos(t)
    y = y0 + b * np.sin(t)
    return x, y

def parabola(t, p, x0, y0):
    x = x0 + t ** 2 / (2. * p)
    y = y0 + t
    return x, y

def rotate(x, y, alpha):
    xr = x * np.cos(alpha) - y * np.sin(alpha)
    yr = x * np.sin(alpha) + y * np.cos(alpha)
    return xr, yr

t = np.arange(0, 2 * math.pi, 0.025)

points = [(), (), ()]

fig1x, fig1y = ellipse(t, figures[0]['a'], figures[0]['b'], figures[0]['x0'], figures[0]
['y0'])
points[0] = rotate(fig1x, fig1y, figures[0]['alpha'])

fig2x, fig2y = ellipse(t, figures[1]['a'], figures[1]['b'], figures[1]['x0'], figures[1]
['y0'])
```

```
points[1] = rotate(fig2x, fig2y, figures[1]['alpha'])

fig3x, fig3y = parabola(t, figures[2]['p'], figures[2]['x0'], figures[2]['y0'])
points[2] = rotate(fig3x, fig3y, figures[2]['alpha'])

figure = plt.figure(figsize = (10, 10))

plt.plot(*points[0], c = 'r')
plt.plot(*points[1], c = 'g')
plt.plot(*points[2], c = 'b')
plt.show()
```



Создадим датасет на основе полученных точек

In [3]:

```
datax = np.concatenate((points[0][0], points[1][0], points[2][0]), axis=0)
datay = np.concatenate((points[0][1], points[1][1], points[2][1]), axis=0)


data = np.array([datax, datay])

l1 = [[1, 0, 0] for _ in range(len(fig1x))]
l2 = [[0, 1, 0] for _ in range(len(fig2x))]
l3 = [[0, 0, 1] for _ in range(len(fig3x))]
```

```
labels = np.array(l1 + l2 + l3)

data = data.transpose()
```

Разделим датасет на обучающую и тестовую выборки

In [5]:

```
from sklearn.model_selection import train_test_split

train, test, train_labels, test_labels = train_test_split(data, labels, test_size = 0.2,
random_state = 10,shuffle = True)
```

Создадим и обучим нейросеть

In [6]:

```
import keras
import tensorflow as tf
from keras.layers import *

model = keras.models.Sequential([
    Dense(10, input_dim = 2, activation = "tanh",  kernel_initializer = keras.initialize
rs.RandomNormal(stddev = 0.01), bias_initializer = keras.initializers.Zeros()),
    Dense(20, activation = "tanh"),
    Dense(10, activation = "tanh"),
    Dense(3, activation = "sigmoid")
])

model.compile(tf.keras.optimizers.SGD(0.05), 'mse')

hist = model.fit(train, train_labels, batch_size = 1, epochs = 100)
```

```
2023-10-26 19:08:15.470458: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda
drivers on your machine, GPU will not be used.
2023-10-26 19:08:15.608393: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda
drivers on your machine, GPU will not be used.
2023-10-26 19:08:15.610696: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Ten
sorFlow binary is optimized to use available CPU instructions in performance-critical ope
rations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow w
ith the appropriate compiler flags.
2023-10-26 19:08:16.881228: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TR
T Warning: Could not find TensorRT
```

```
Epoch 1/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1629
Epoch 2/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1499
Epoch 3/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1481
Epoch 4/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1479
Epoch 5/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1473
Epoch 6/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1460
Epoch 7/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1458
Epoch 8/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1451
Epoch 9/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1444
Epoch 10/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1433
Epoch 11/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1431
Epoch 12/100
604/604 [==============================] - 2s 3ms/step - loss: 0.1419
Epoch 13/100
```

```
Epoch 13/100
604/604 [==============================] - 1s 2ms/step - loss: 0.1400
Epoch 14/100
604/604 [==============================] - 1s 2ms/step - loss: 0.1378
Epoch 15/100
604/604 [==============================] - 1s 2ms/step - loss: 0.1339
Epoch 16/100
604/604 [==============================] - 1s 2ms/step - loss: 0.1296
Epoch 17/100
604/604 [==============================] - 1s 2ms/step - loss: 0.1254
Epoch 18/100
604/604 [==============================] - 1s 2ms/step - loss: 0.1199
Epoch 19/100
604/604 [==============================] - 1s 2ms/step - loss: 0.1126
Epoch 20/100
604/604 [==============================] - 1s 2ms/step - loss: 0.1043
Epoch 21/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0967
Epoch 22/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0886
Epoch 23/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0847
Epoch 24/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0819
Epoch 25/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0795
Epoch 26/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0769
Epoch 27/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0749
Epoch 28/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0733
Epoch 29/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0703
Epoch 30/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0702
Epoch 31/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0679
Epoch 32/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0664
Epoch 33/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0651
Epoch 34/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0625
Epoch 35/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0615
Epoch 36/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0596
Epoch 37/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0573
Epoch 38/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0555
Epoch 39/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0549
Epoch 40/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0537
Epoch 41/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0522
Epoch 42/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0519
Epoch 43/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0489
Epoch 44/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0463
Epoch 45/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0453
Epoch 46/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0446
Epoch 47/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0432
Epoch 48/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0404
Epoch 49/100
```

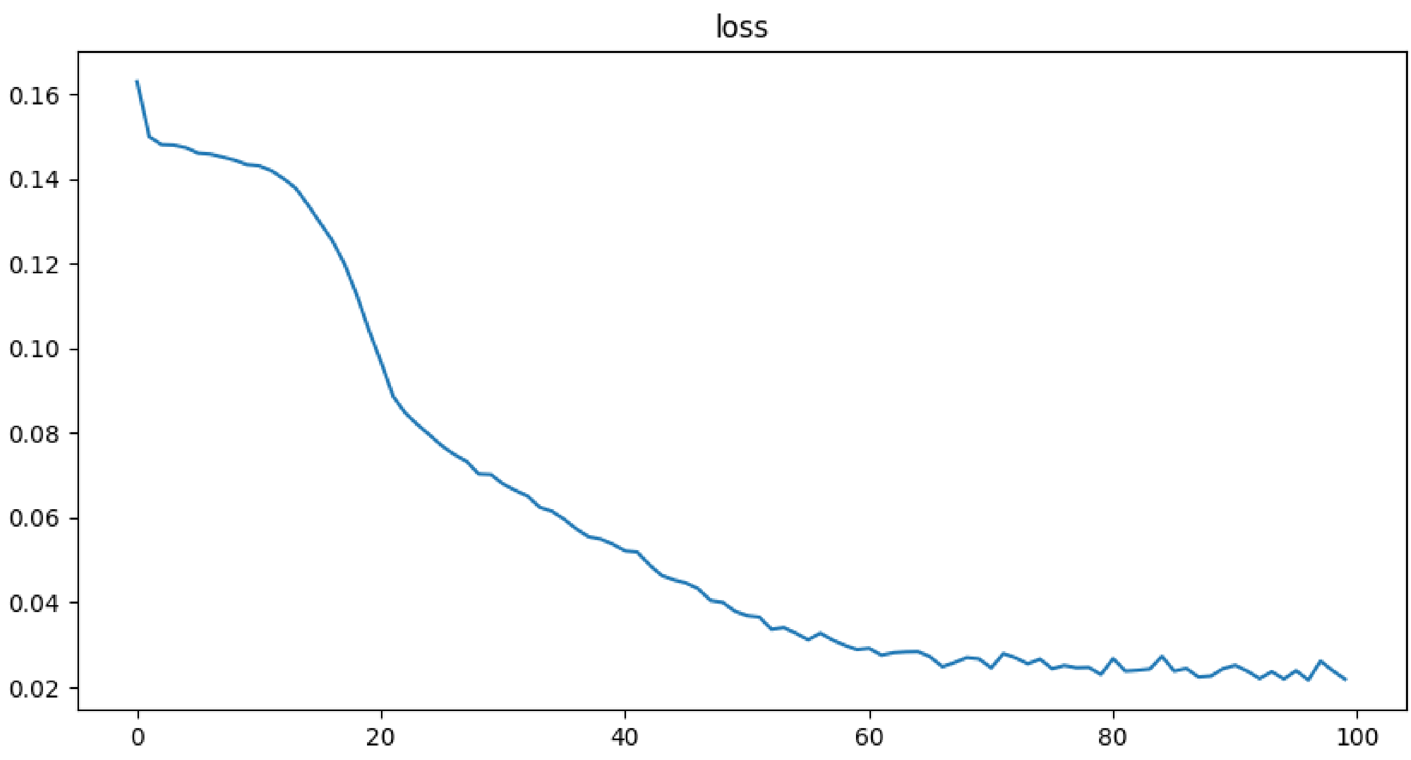
```
Epoch 49/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0400
Epoch 50/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0379
Epoch 51/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0369
Epoch 52/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0365
Epoch 53/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0336
Epoch 54/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0340
Epoch 55/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0327
Epoch 56/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0312
Epoch 57/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0327
Epoch 58/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0311
Epoch 59/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0299
Epoch 60/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0289
Epoch 61/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0292
Epoch 62/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0275
Epoch 63/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0281
Epoch 64/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0283
Epoch 65/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0284
Epoch 66/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0272
Epoch 67/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0248
Epoch 68/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0258
Epoch 69/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0270
Epoch 70/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0267
Epoch 71/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0245
Epoch 72/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0279
Epoch 73/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0269
Epoch 74/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0255
Epoch 75/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0266
Epoch 76/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0243
Epoch 77/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0251
Epoch 78/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0246
Epoch 79/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0246
Epoch 80/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0230
Epoch 81/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0267
Epoch 82/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0238
Epoch 83/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0240
Epoch 84/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0243
Epoch 85/100
```

```
Epoch 85/100
604/604 [==============================] - 2s 4ms/step - loss: 0.0272
Epoch 86/100
604/604 [==============================] - 2s 4ms/step - loss: 0.0238
Epoch 87/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0244
Epoch 88/100
604/604 [==============================] - 1s 2ms/step - loss: 0.0224
Epoch 89/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0226
Epoch 90/100
604/604 [==============================] - 2s 4ms/step - loss: 0.0243
Epoch 91/100
604/604 [==============================] - 3s 4ms/step - loss: 0.0251
Epoch 92/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0238
Epoch 93/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0220
Epoch 94/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0236
Epoch 95/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0219
Epoch 96/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0239
Epoch 97/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0217
Epoch 98/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0261
Epoch 99/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0239
Epoch 100/100
604/604 [==============================] - 2s 3ms/step - loss: 0.0219
```

Посмотрим на график **loss**

In [7]:

```python
figure = plt.figure(figsize = (10, 5))
histx = []
for i in range(len(hist.history['loss'])):
    histx.append(i)


plt.plot(histx, hist.history['loss'])
plt.title("loss")
plt.show()
```

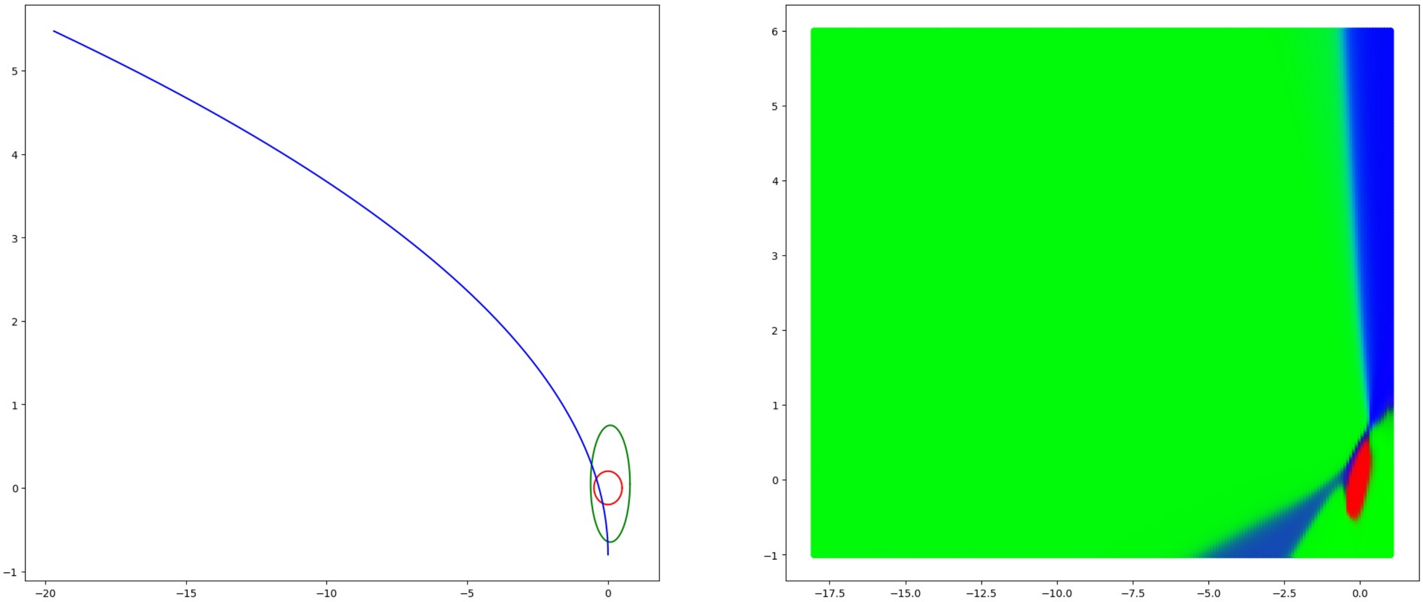


Построим классификацию точек

In [8]:

```
import itertools

x = np.linspace(-18, 1, 200)
y = np.linspace(-1, 6, 200)

figure = plt.figure(figsize = (24, 10))

ax1 = figure.add_subplot(1, 2, 1)
ax2 = figure.add_subplot(1, 2, 2)

ax1.plot(fig1x, fig1y, c = 'r')
ax1.plot(fig2x, fig2y, c = 'g')
ax1.plot(fig3x, fig3y, c = 'b')

data = np.array(list(itertools.product(x, y)))

xy = data.transpose()

pred = model.predict(data)


ax2.scatter(xy[0], xy[1], c = pred)

plt.show()
```

```
1250/1250 [==============================] - 3s 3ms/step
```



Задание 2

Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов первого порядка.

In [9]:

```
def f(t):
    return np.sin(-np.sin(t)*(t**2) + t)

t_range = (0.5, 4)
h = 0.025
```

На основе заданных функции и промежутка подготовим датасет для обучения, а также разделим датасет на обучающую и тестовую выборки

In [10]:

```
t = np.linspace(t_range[0], t_range[1], int((t_range[1] - t_range[0]) / h))
x = f(t)

train_len = int(t.shape[0] * 0.9)
t_train = t[:train_len]
t_val = t[train_len:]
x_train = x[:train_len]
x_val = x[train_len:]

t_train = np.expand_dims(t_train, 1)
t_val = np.expand_dims(t_val, 1)
```

Создадим и обучим нейросеть

In [11]:

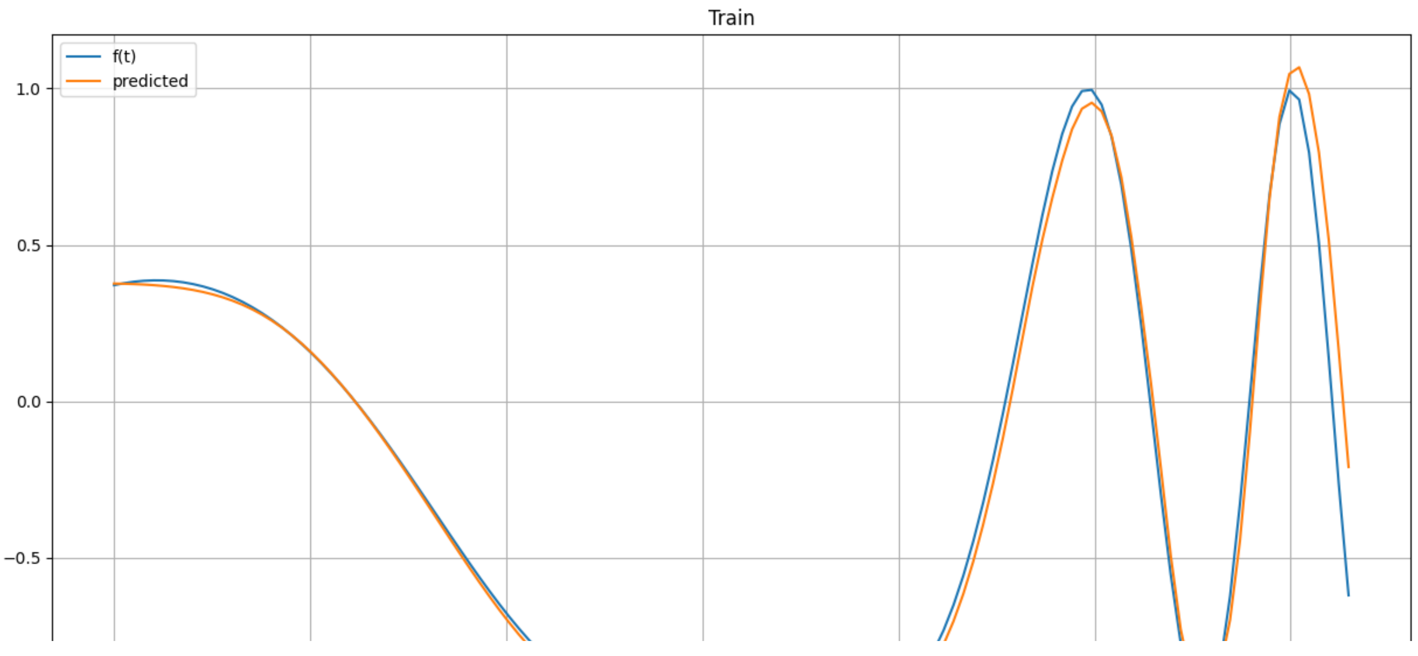
```
model = keras.Sequential([
    Dense(64, activation='tanh'),
    Dense(32, activation='tanh'),
    Dense(1),
])

model.compile(loss='mse', optimizer='Adam', metrics=tf.keras.metrics.RootMeanSquaredErro
r())
train_info = model.fit(
    t_train,
    x_train,
    batch_size=4,
    epochs=2000,
    validation_data=(t_val, x_val),
    verbose=0
)
```

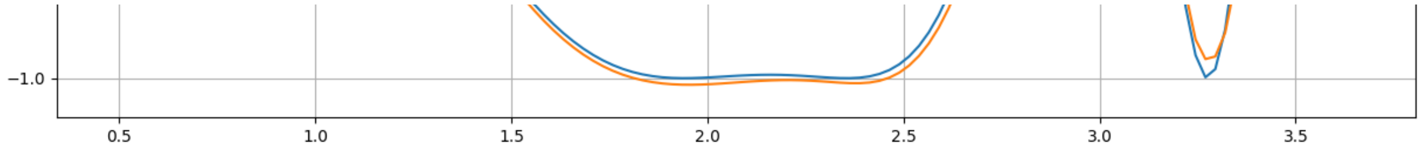
Проверим результаты обучения

In [12]:

```
plt.figure(figsize=(15, 8))

plt.plot(t_train, f(t_train), label='f(t)')
plt.plot(t_train, model.predict(t_train), label='predicted')
plt.title('Train')
plt.grid()
plt.legend()
plt.show()
```

```
4/4 [==============================] - 0s 2ms/step
```





Выводы

Проделав лабораторную работу, я создал многослойные нейросети, которые способны решать задачи классификации и аппроксимации.