Лабораторная работа №4 по курсу "Нейроинформатика" на тему "Сети с радиальными базисными элементами"

Целью работы является исследование свойств некоторых видов сетей с радиальными базисными элементами, алгоритмов обучения, а также применение сетей в задачах классификации и аппроксимации функции.

Задания:

1. Создать RBF сеть для классификации точек в случае, когда классы не являются линейно разделимыми.
2. Создать генеративную RBF сеть для аппроксимации функции. Произвести обучение с помощью одного из методов первого порядка.

Вариант 10

Выполнил студент Шавандрин Фёдор
Группа М8О-408Б-19

In [11]:
```
# импортируем библиотеки
import numpy as np

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.layers import Layer
from keras import backend as back

import matplotlib.pyplot as plt
```

Константы a и b задают большую и малую полуоси эллипса. Параметры преобразования прямоугольной системы координат на плоскости: угол поворота α и координаты параллельного переноса (x_0, y_0).

In [2]:
```
ellipse1 = dict(a = 0.2, b = 0.2, alpha = 0, x0 = 0.2, y0 = 0, label = 0)
ellipse2 = dict(a = 0.7, b = 0.5, alpha = -np.pi / 3, x0 = 0, y0 = 0, label = 1)
ellipse3 = dict(a = 1, b = 1, alpha = 0, x0 = 0, y0 = 0, label = 2)
```

Задание 1

Классификация точек в случае, когда классы не являются линейно разделимыми.

Функция генерации точек, используя параметрическое уравнение линии в канонической системе координат:
$t=0:0.025:2\pi$
$x=f(t)$
$y=g(t)$

In [3]:
```
def generate_points_dataset(ellipses):
    t = np.linspace(0, 2 * np.pi, int(2 * np.pi / 0.025))

    points = np.array([
        [
            ellipses[i]['a'] * np.cos(t) * np.cos(ellipses[i]['alpha']) \
            - ellipses[i]['b'] * np.sin(t) * np.sin(ellipses[i]['alpha']) \
            + ellipses[i]['x0'],

            ellipses[i]['a'] * np.cos(t) * np.sin(ellipses[i]['alpha']) \
            + ellipses[i]['b'] * np.sin(t) * np.cos(ellipses[i]['alpha']) \
            + ellipses[i]['y0'],

            np.tile(ellipses[i]['label'], len(t)),

         ] for i in range(len(ellipses))])

    np.random.seed(66)

    class_1_points = points[0, :, np.random.choice(len(t), 60, replace=False)]
    class_2_points = points[1, :, np.random.choice(len(t), 100, replace=False)]
    class_3_points = points[2, :, np.random.choice(len(t), 120, replace=False)]

    dataset = np.vstack((class_1_points, class_2_points, class_3_points))
    np.random.shuffle(dataset)

    return dataset
```

In [4]:
```
data1 = generate_points_dataset([ellipse1, ellipse2, ellipse3])
```

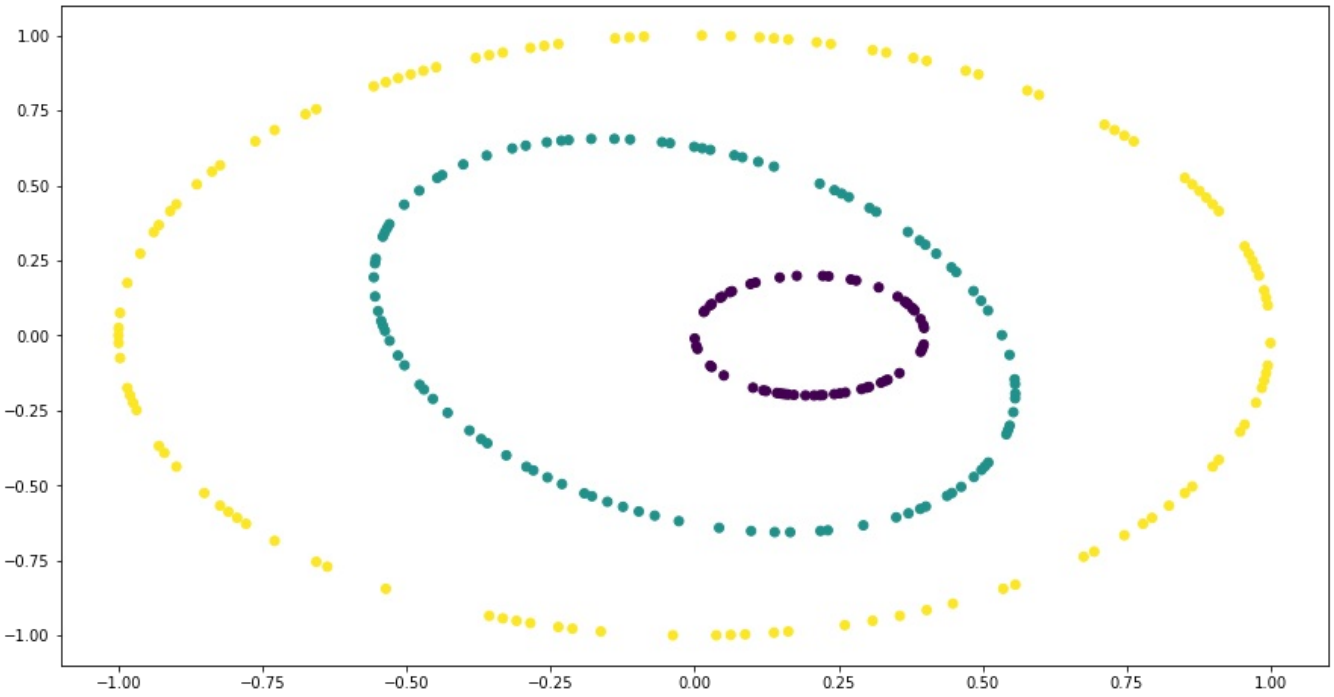
Итого имеем 280 точек в датасете.

In [5]:
```
data1.shape
```

Out[5]:
```
(280, 3)
```

Посмотрим на расположение точек каждого класса.

In [6]:
```
plt.figure(figsize=(15, 8))
plt.scatter(data1[:, 0], data1[:, 1], c=data1[:, 2])
plt.show()
```



Обучающая, тестовая и валидационная выборка

Разобьём датасет на обучающую, валидационную и тестовую выборку в соотношении $70\%-20\%-10\%$.

In [7]:
```
train, val, test = np.split(data1, [int(.7*len(data1)), int(.9*len(data1))])
```

In [8]:
```
train.shape, val.shape, test.shape
```

Out[8]:
```
((196, 3), (56, 3), (28, 3))
```

Выделяем для каждой выборке признаки (фичи) и таргеты (лейблы).

In [9]:
```
X_train = train[:, :2]
y_train = train[:, 2]

X_test = test[:, :2]
y_test = test[:, 2]

X_val = val[:, :2]
y_val = val[:, 2]
```

Реализация RBF слоя

RBF сеть использует радиальные базисные функции как функции активации. Радиальные базисные функции - это функции вида $f(x)= \phi({x^2 \over \sigma^2})$, где x - вектор входных сигналов нейрона , σ - ширина окна функции, $\phi(y)$ - убывающая функция (чаще всего, равная нулю вне некоторого отрезка).

RBF слой наследуется от линейного слоя из keras. Реализуем методы инициализации слоя, его построения и применения.

In [12]:
```
class RBF(Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(RBF, self).__init__(**kwargs)

    def build(self, input_shape):
        self.mu = self.add_weight(
            name='mu',
            shape=(input_shape[1], self.output_dim),
            initializer='uniform',
            trainable=True,
        )
```

```
        self.sigma = self.add_weight(
            name='sigma',
            shape=(self.output_dim, ),
            initializer='uniform',
            trainable=True,
        )

        super(RBF, self).build(input_shape)

    def call(self, inputs):
        diff = back.expand_dims(inputs) - self.mu
        output = back.exp(back.sum(diff**2, axis=1) * self.sigma)
        return output
```

Обучение модели

Будем использовать RBF и линейный слои. В качестве функции активации будем использовать Relu, алгоритм обучения - Adam.

In [13]:
```
model1 = keras.Sequential([
    RBF(input_dim=2, output_dim=10),
    keras.layers.Dense(3, activation='softmax'),
])
```

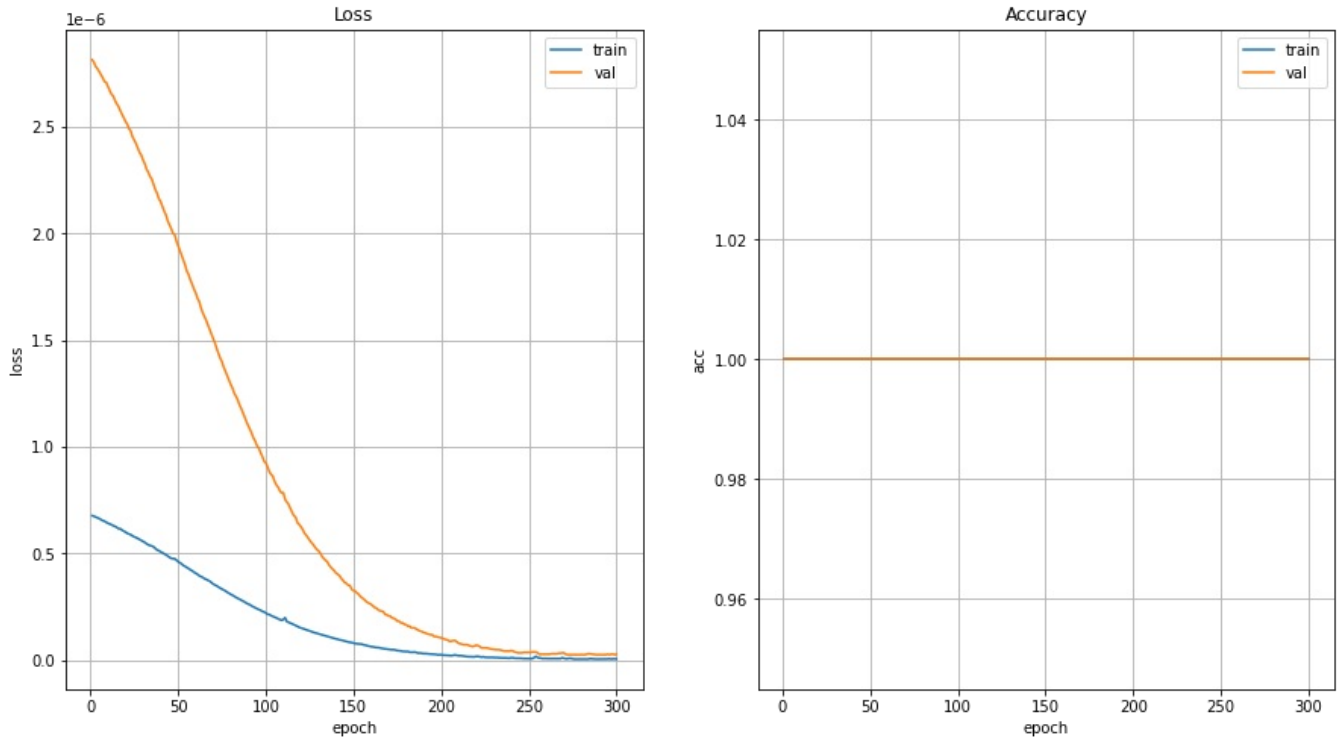
In [14]:
```
model1.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='Adam',
    metrics='accuracy'
)
```

In [28]:
```
train_info1 = model1.fit(X_train, y_train, batch_size=4, epochs=300, validation_data=(X_val, y_val), shuffle=Tr
```

Функция построения графиков лосса и accuracy на обучающей и валидационной выборке.

In [29]:
```
def create_metrics_plot(train_info):
    plt.figure(figsize=(15, 8))

    plt.subplot(1, 2, 1)
    loss_history = train_info.history['loss']
    val_loss_history = train_info.history['val_loss']
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.plot(range(1, len(loss_history) + 1), loss_history, label='train')
    plt.plot(range(1, len(loss_history) + 1), val_loss_history, label='val')
    plt.grid()
    plt.legend()
    plt.title('Loss')

    plt.subplot(1, 2, 2)
    acc_history = train_info.history['accuracy']
    val_acc_history = train_info.history['val_accuracy']
    plt.xlabel('epoch')
    plt.ylabel('acc')
    plt.plot(range(1, len(acc_history) + 1), acc_history, label='train')
    plt.plot(range(1, len(val_acc_history) + 1), val_acc_history, label='val')
    plt.grid()
    plt.legend()
    plt.title('Accuracy')

    plt.show()
```

In [30]:
```
create_metrics_plot(train_info1)
```

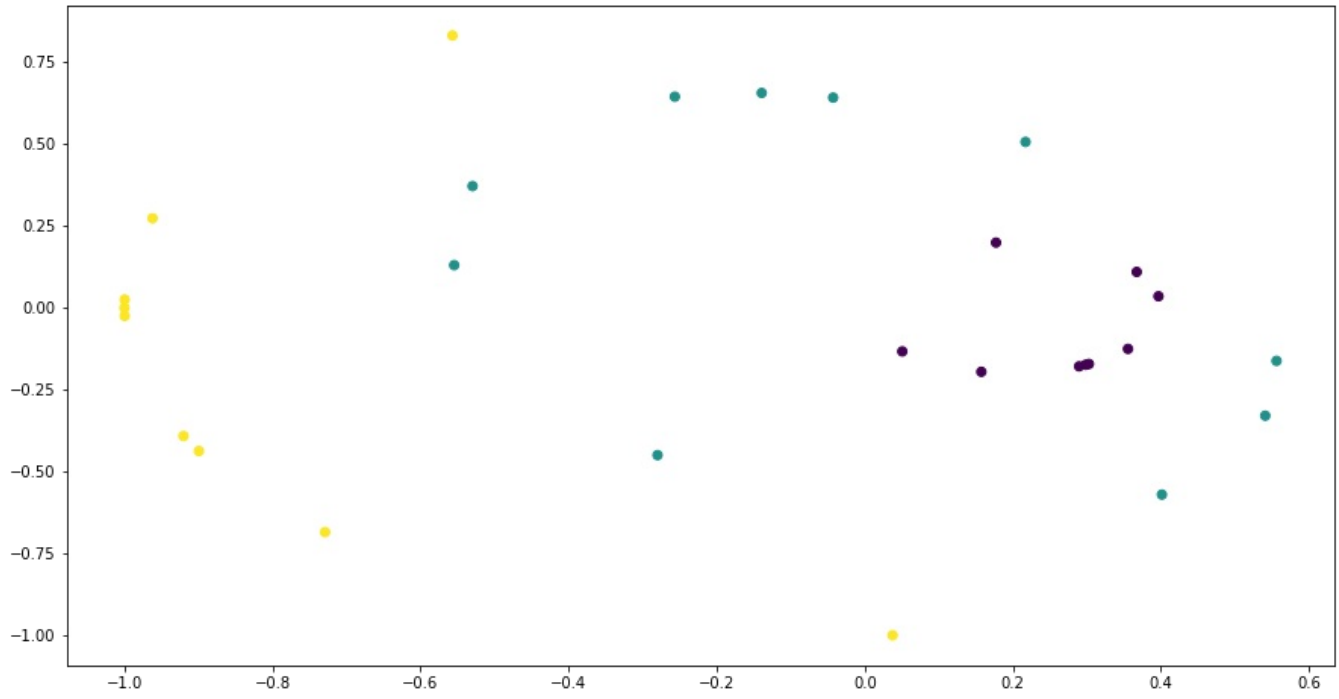


In [31]:
```
print("Точность на валидационной выборке =", train_info1.history['val_accuracy'][-1])
print("Лосс на валидационной выборке =", train_info1.history['val_loss'][-1])
```

```
Точность на валидационной выборке = 1.0
Лосс на валидационной выборке = 2.7673580760279037e-08
```

Результаты работы модели

In [32]:
```
plt.figure(figsize=(15, 8))
plt.scatter(X_test[:, 0], X_test[:, 1], c=np.argmax(model1.predict(X_test, verbose=0), axis=1))
plt.show()
```



In [17]:
```
print("Точность на тестовой выборке =", (np.argmax(model1.predict(X_test, verbose=0), axis=1) == y_test).mean()
```

```
Точность на тестовой выборке = 1.0
```

Все точки тестовой выборки модель предсказала верно.

Произведём классификацию точек области $[−1.2, 1.2]×[−1.2, 1.2]$. Для этого зададим сетку для указанной области с шагом $h = 0.025$. Рассчитаем выход сети для всех узлов сетки.

In [34]:
```
h = 0.025

grid = [model1.predict(np.array([[i, j]]), verbose=0).round(1)
        for i in np.arange(-1.2, 1.2 + h, h)
```

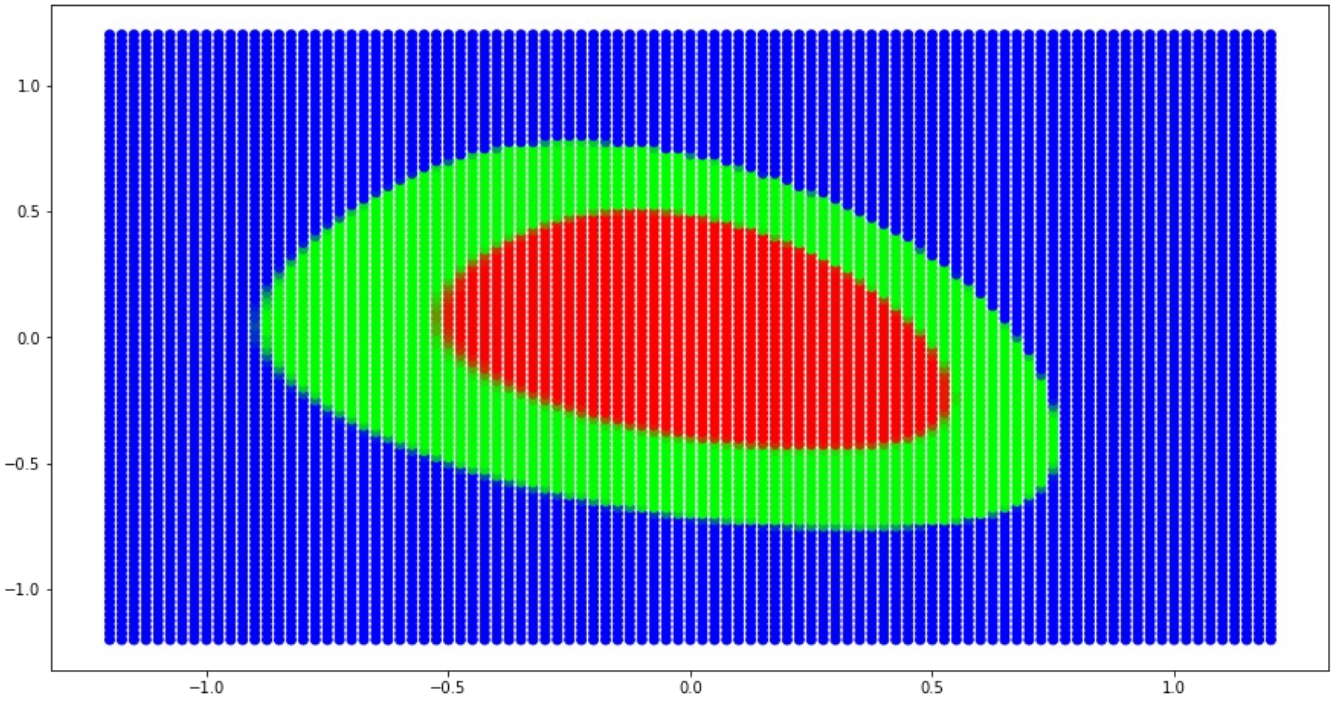
```
        for j in np.arange(-1.2, 1.2 + h, h)]
```

In [35]:
```
x_coords = np.arange(-1.2, 1.2 + h, h)
y_coords = np.arange(-1.2, 1.2 + h, h)

xx, yy = np.meshgrid(x_coords, y_coords)
```

In [36]:
```
colors = np.array(grid).reshape((len(grid), 3))
colors.shape
```

Out[36]:
```
(9409, 3)
```

In [37]:
```
plt.figure(figsize=(15, 8))
plt.scatter(xx, yy, c=colors)
plt.show()
```



Как видно, получилось первоначальное изображение эллипсов.

Задание 2.

Аппроксимировать функцию. Произвести обучение с помощью одного из методов первого порядка. Использовать генеративную RBF сеть

In [38]:
```
def X(t):
    return np.sin(t**2 - 7 * t)

t_start = 0
t_end = 5
h = 0.025
```

Создадим датасет и поделим его на обучающую и валидационную выборку в соотношении $90\%-10\%$.

In [39]:
```
t = np.linspace(t_start, t_end, int((t_end - t_start) / h))
x = X(t)
```

In [40]:
```
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:]
```

In [41]:
```
t_train = np.expand_dims(t_train, 1)
t_val = np.expand_dims(t_val, 1)
```

Реализация генеративного RBF слоя

In [42]:
```
class RBF_gen(Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(RBF_gen, self).__init__(**kwargs)
```

```
    def build(self, input_shape):
        self.mu = self.add_weight(
            name='mu',
            shape=(input_shape[1], self.output_dim),
            initializer='uniform',
            trainable=True,
        )

        self.sigma = self.add_weight(
            name='sigma',
            shape=(self.output_dim, ),
            initializer='uniform',
            trainable=True,
        )

        self.sw = self.add_weight(
            name='sw',
            shape=(self.output_dim, ),
            initializer='uniform',
            trainable=True,
        )

        super(RBF_gen, self).build(input_shape)

    def call(self, inputs):
        diff = back.expand_dims(inputs) - self.mu
        output = back.exp(back.sum(diff**2, axis=1) * self.sigma)
        output = output * self.sw
        return output
```

Обучение модели

В качестве алгоритма обучения возьмем Adam (метод оптимизации 1 порядка), функцию активацию Tanh, количество слоёв - 3.

In [69]:
```
model2 = keras.Sequential([
    RBF_gen(input_dim=1, output_dim=64),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1),
])
```

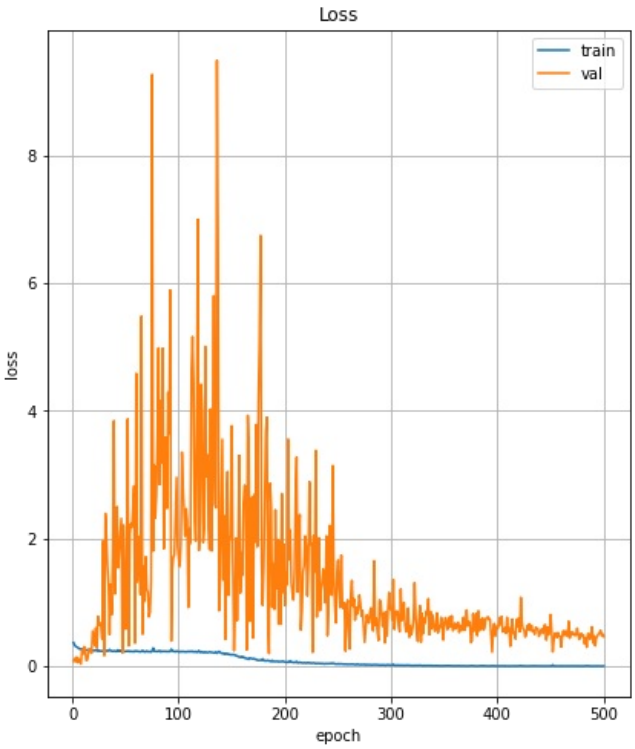
In [70]:
```
model2.compile(
    loss='mse',
    optimizer='Adam',
    metrics=tf.keras.metrics.RootMeanSquaredError(),
)
```

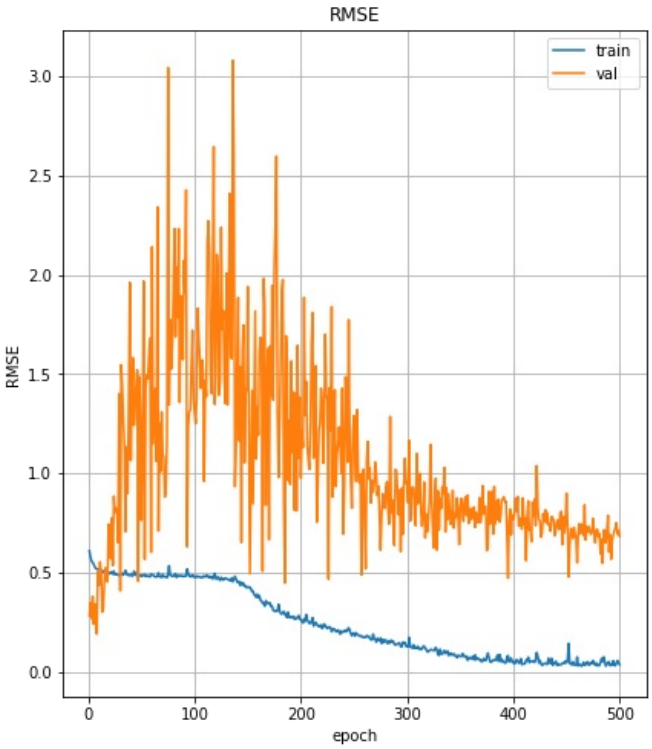
In [71]:
```
train_info2 = model2.fit(t_train, x_train, batch_size=2, epochs=500, validation_data=(t_val, x_val), verbose=0)
```

Функция построения графиков лосса и accuracy на обучающей и валидационной выборке.

In [53]:
```
def create_metrics_plot2(train_info):
    plt.figure(figsize=(15, 8))

    plt.subplot(1, 2, 1)
    loss_history = train_info.history['loss']
    val_loss_history = train_info.history['val_loss']
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.plot(range(1, len(loss_history) + 1), loss_history, label='train')
    plt.plot(range(1, len(loss_history) + 1), val_loss_history, label='val')
    plt.grid()
    plt.legend()
    plt.title('Loss')

    plt.subplot(1, 2, 2)
    acc_history = train_info.history['root_mean_squared_error']
    val_acc_history = train_info.history['val_root_mean_squared_error']
    plt.xlabel('epoch')
    plt.ylabel('RMSE')
    plt.plot(range(1, len(acc_history) + 1), acc_history, label='train')
    plt.plot(range(1, len(val_acc_history) + 1), val_acc_history, label='val')
    plt.grid()
    plt.legend()
    plt.title('RMSE')

    plt.show()
```

In [72]:
```
create_metrics_plot2(train_info2)
```





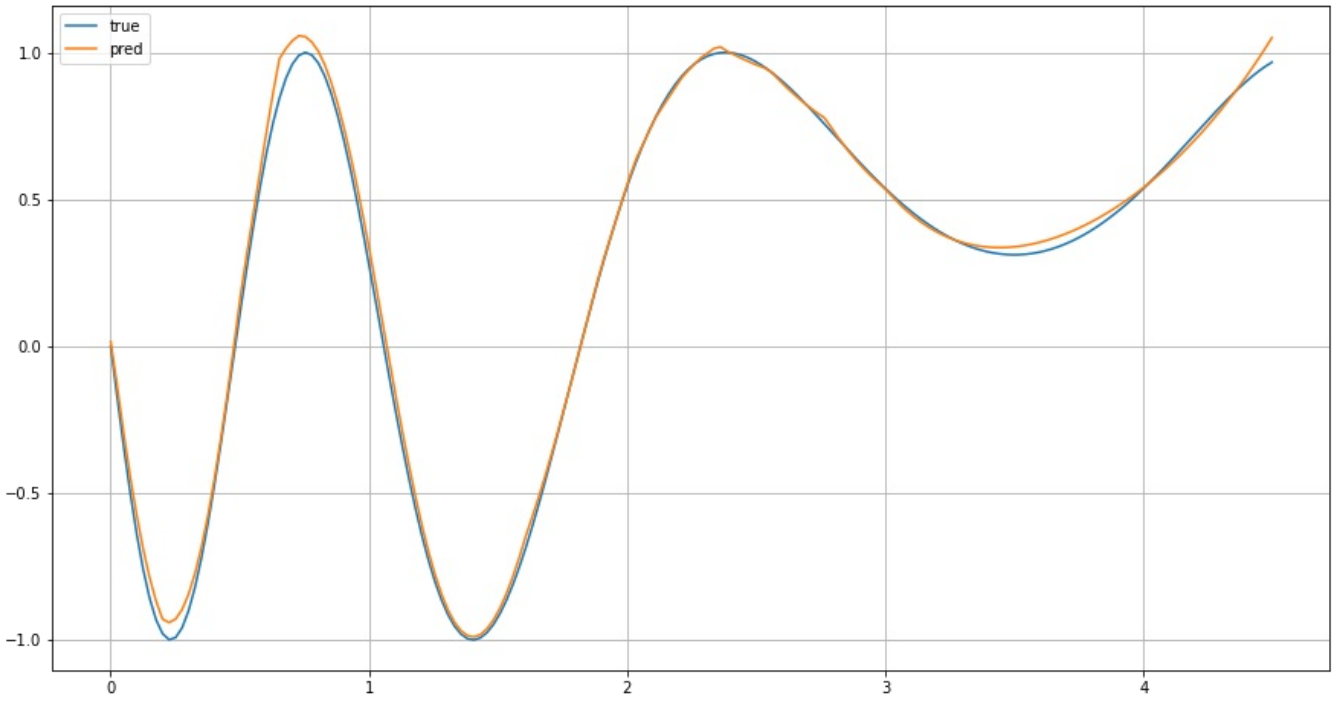
In [73]:
```
print("RMSE на валидационной выборке =", train_info2.history['val_root_mean_squared_error'][-1])
```

```
RMSE на валидационной выборке = 0.6806167364120483
```

Результаты работы модели на обучающей выборке

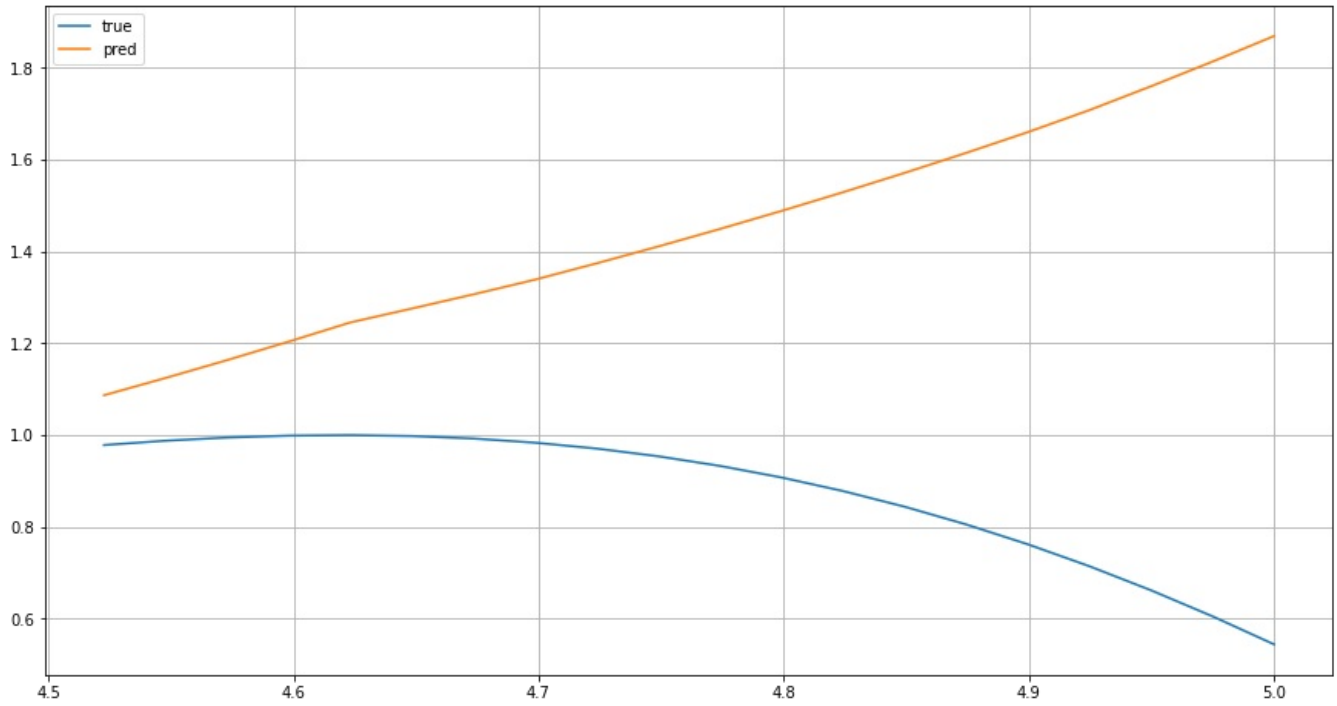
In [74]:
```
def build_results_plot(model, t, X):
    plt.figure(figsize=(15, 8))

    plt.plot(t, X(t), label='true')
    plt.plot(t, model.predict(t), label='pred')

    plt.grid()
    plt.legend()
    plt.show()
```

In [75]:
```
build_results_plot(model2, t_train, X)
```



Результаты работы модели на тестовой выборке

In [76]:
```
build_results_plot(model2, t_val, X)
```



Как видно из графика, на валидационной выборке модель плохо справляется.

Выводы

В ходе данной лабораторной работы попробовал самостоятельно реализовать RBF сеть для классификации точек с линейно не разделяемыми классами, изучил особенности RBF-слоев, одним из которых является использование другой нелинейной функции с обучаемыми параметрами.

Также обучил генеративную RBF сеть для аппроксимации заданной вариантом ЛР функции с помощью метода первого порядка Adam.

Полученные результаты модели аппроксимировать нелинейную функцию можно проанализировать следующим образом: наша многослойная нейронная сеть не очень хорошо справилась с закруглением функции - модель продолжала строить функцию по прямой.

RMSE на валидационной выборке: 0.681