Лабораторная работа №4

Сети с радиальными базисными элементами

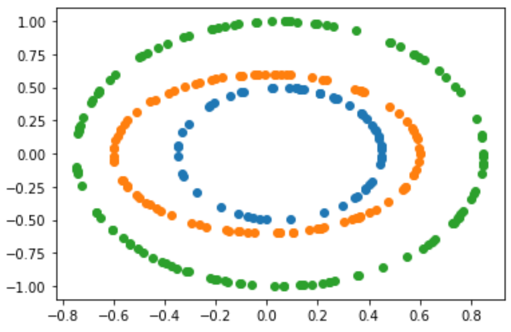
Целью работы является исследование свойств некоторых видов сетей с радиальными базисными элементами, алгоритмов обучения, а также применение сетей в задачах классификации и аппроксимации функции.

```
# import os
# os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow import keras
from tensorflow.keras import layers
from keras import backend
```

Сгенерирую псевдослучайные данные для обучения и тестирования.

```
def gen_random_ellipse_data(a, b, alpha, x_0, y_0, n):
    rand_nums = 2*np.pi*np.random.rand(n)
    x = x_0 + a * np.cos(rand_nums)
    y = y_0 + b * np.sin(rand_nums)
    xr = x * np.cos(alpha) - y * np.sin(alpha)
    yr = x * np.sin(alpha) + y * np.cos(alpha)
    return np.array([xr, yr])
```

```
fig, ax = plt.subplots(1, 1)
data1_size = 60
data2_size = 100
data3_size = 120
data1 = gen_random_ellipse_data(0.4, 0.5, 0, 0.05, 0, data1_size)
data2 = gen_random_ellipse_data(0.6, 0.6, 0, 0.00, 0, data2_size)
data3 = gen_random_ellipse_data(0.8, 1, 0, 0.05, 0, data3_size)
ax.scatter(data1[0], data1[1])
ax.scatter(data2[0], data2[1])
ax.scatter(data3[0], data3[1])

plt.show()
```



Перегруппирую и разобью данные на тестовую и обучающую выборку.

```
data = np.concatenate([data1, data2, data3], axis=1)
data = data.transpose()
labels1 = np.array([[1, 0, 0] for i in range(data1_size)])
labels2 = np.array([[0, 1, 0] for i in range(data2_size)])
labels3 = np.array([[0, 0, 1] for i in range(data3_size)])
labels = np.concatenate([labels1, labels2, labels3], axis=0)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.3, random_state=322)
```

Реализую радиально базисный слой, для построения нейросетевой модели.

```
class RBFLayer(layers.Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(RBFLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        # print(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(RBFLayer, self).build(input_shape)

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

Построю и скомпилирую и обучу модель.

```
model = keras.models.Sequential()
model.add(RBFLayer(10, input_dim = 2))
model.add(keras.layers.Dense(3, activation='sigmoid'))

model.compile(loss='mse', optimizer='adam', metrics=['mae'])
```

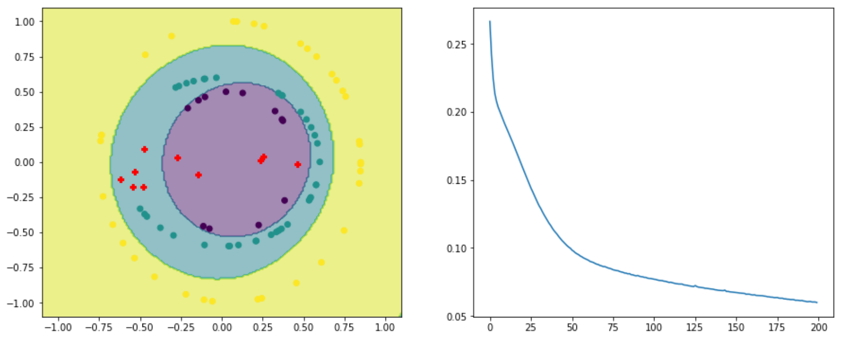
```
epochs = 200
hist = model.fit(X_train, y_train, batch_size=5, epochs=epochs)
```

```
Epoch 1/200
40/40 [==============================] - 1s 4ms/step - loss: 0.2665 - mae: 0.4880
Epoch 2/200
40/40 [==============================] - 0s 5ms/step - loss: 0.2419 - mae: 0.4696
Epoch 3/200
40/40 [==============================] - 0s 4ms/step - loss: 0.2247 - mae: 0.4550
Epoch 4/200
40/40 [==============================] - 0s 3ms/step - loss: 0.2136 - mae: 0.4424
Epoch 5/200
40/40 [==============================] - 0s 3ms/step - loss: 0.2078 - mae: 0.4330
Epoch 6/200
40/40 [==============================] - 0s 4ms/step - loss: 0.2036 - mae: 0.4274
Epoch 7/200
40/40 [==============================] - 0s 4ms/step - loss: 0.2005 - mae: 0.4228
Epoch 8/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1974 - mae: 0.4183
Epoch 9/200
40/40 [==============================] - 0s 2ms/step - loss: 0.1943 - mae: 0.4136
Epoch 10/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1913 - mae: 0.4086
Epoch 11/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1885 - mae: 0.4042
Epoch 12/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1856 - mae: 0.3998
Epoch 13/200
40/40 [==============================] - 0s 2ms/step - loss: 0.1828 - mae: 0.3953
Epoch 14/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1800 - mae: 0.3932
Epoch 15/200
40/40 [==============================] - 0s 5ms/step - loss: 0.1771 - mae: 0.3893
Epoch 16/200
40/40 [==============================] - 0s 5ms/step - loss: 0.1740 - mae: 0.3850
Epoch 17/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1710 - mae: 0.3796
Epoch 18/200
40/40 [==============================] - 0s 5ms/step - loss: 0.1680 - mae: 0.3749
Epoch 19/200
40/40 [==============================] - 0s 2ms/step - loss: 0.1650 - mae: 0.3713
Epoch 20/200
40/40 [==============================] - 0s 5ms/step - loss: 0.1619 - mae: 0.3676
Epoch 21/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1588 - mae: 0.3622
Epoch 22/200
40/40 [==============================] - 0s 4ms/step - loss: 0.1557 - mae: 0.3577
Epoch 23/200
40/40 [==============================] - 0s 5ms/step - loss: 0.1528 - mae: 0.3532
Epoch 24/200
40/40 [==============================] - 0s 4ms/step - loss: 0.1499 - mae: 0.3485
Epoch 25/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1470 - mae: 0.3447
Epoch 26/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1439 - mae: 0.3398
Epoch 27/200
40/40 [==============================] - 0s 4ms/step - loss: 0.1414 - mae: 0.3349
Epoch 28/200
40/40 [==============================] - 0s 3ms/step - loss: 0.1387 - mae: 0.3294
```

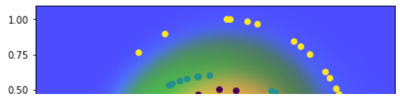
```
Epoch 29/200
```

Нарисую результаты.

```
steps = 200
x_span = np.linspace(-1.1, 1.1, steps)
y_span = np.linspace(-1.1, 1.1, steps)
xx, yy = np.meshgrid(x_span, y_span)
grid_points = np.array([xx.ravel(), yy.ravel()])
grid_points = grid_points.transpose()

grid_predictions = model(grid_points)
grid_labels = np.argmax(grid_predictions, axis=1)

predictions = model(X_test)

fig, ax = plt.subplots(1, 2, figsize=(15, 6))

grid_labels = grid_labels.reshape(xx.shape)
ax[0].contourf(xx, yy, grid_labels, alpha=0.5)
ax[0].scatter(X_test[:, 0], X_test[:, 1], c=np.argmax(y_test, axis=1))

mu_x = model.layers[0].get_weights()[0][0, :]
mu_y = model.layers[0].get_weights()[0][1, :]
ax[0].scatter(mu_x, mu_y, marker='P', color='red')

ax[1].plot(hist.history['loss'])

plt.show()
```



```
grid_predictions = model(grid_points).numpy().reshape(steps, steps, 3)

fig, ax = plt.subplots(1, 2, figsize=(15, 6))

ax[0].imshow(grid_predictions, extent=(-1.1, 1.1, -1.1, 1.1), alpha=0.7, origin='lower', aspect='auto', interpolation='bilinear')
ax[0].scatter(X_test[:, 0], X_test[:, 1], c=np.argmax(y_test, axis=1))

mu_x = model.layers[0].get_weights()[0][0, :]
mu_y = model.layers[0].get_weights()[0][1, :]
ax[0].scatter(mu_x, mu_y, marker='P', color='red')

ax[1].plot(hist.history['loss'])

plt.show()
```



Аппроксимация функции

Аппроксимирую функцию с помощью нейросетевой модели с радиально базисными элементами.

```
def f(t):
    return np.sin(t**2-6*t+3)

x = np.arange(0, 5, 0.025)
y = f(x)
```

Построю и скомпилирую и обучу модель.

```
model = keras.models.Sequential()
model.add(RBFLayer(20, input_dim = 1))
model.add(keras.layers.Dense(20, activation='tanh'))
model.add(keras.layers.Dense(1, activation='linear'))

model.compile(loss='mse', optimizer='adam', metrics=['mae'])
```

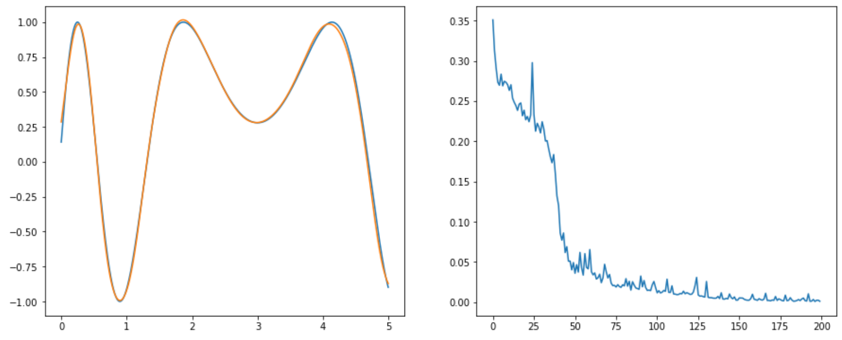
```
epochs = 200
hist = model.fit(x, y, batch_size=1, epochs=epochs)
```

```
Epoch 1/200
200/200 [==============================] - 1s 2ms/step - loss: 0.3509 - mae: 0.4588
Epoch 2/200
200/200 [==============================] - 0s 2ms/step - loss: 0.3126 - mae: 0.4416
Epoch 3/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2908 - mae: 0.4381
Epoch 4/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2734 - mae: 0.4269
Epoch 5/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2696 - mae: 0.4230
Epoch 6/200
200/200 [==============================] - 0s 1ms/step - loss: 0.2834 - mae: 0.4360
Epoch 7/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2689 - mae: 0.4322
Epoch 8/200
200/200 [==============================] - 0s 1ms/step - loss: 0.2746 - mae: 0.4347
Epoch 9/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2733 - mae: 0.4335
Epoch 10/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2703 - mae: 0.4273
Epoch 11/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2633 - mae: 0.4249
Epoch 12/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2702 - mae: 0.4266
Epoch 13/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2535 - mae: 0.4093
Epoch 14/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2484 - mae: 0.4060
Epoch 15/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2444 - mae: 0.4012
Epoch 16/200
200/200 [==============================] - 0s 1ms/step - loss: 0.2384 - mae: 0.3930
Epoch 17/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2461 - mae: 0.3893
Epoch 18/200
200/200 [==============================] - 0s 1ms/step - loss: 0.2477 - mae: 0.4029
Epoch 19/200
200/200 [==============================] - 0s 1ms/step - loss: 0.2317 - mae: 0.3724
Epoch 20/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2386 - mae: 0.3884
Epoch 21/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2268 - mae: 0.3624
Epoch 22/200
200/200 [==============================] - 1s 3ms/step - loss: 0.2308 - mae: 0.3687
Epoch 23/200
200/200 [==============================] - 1s 3ms/step - loss: 0.2243 - mae: 0.3664
Epoch 24/200
200/200 [==============================] - 0s 1ms/step - loss: 0.2324 - mae: 0.3715
Epoch 25/200
200/200 [==============================] - 0s 1ms/step - loss: 0.2978 - mae: 0.4448
Epoch 26/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2334 - mae: 0.3624
Epoch 27/200
200/200 [==============================] - 0s 2ms/step - loss: 0.2125 - mae: 0.3402
Epoch 28/200
200/200 [==============================] - 0s 1ms/step - loss: 0.2223 - mae: 0.3393
```

```
Epoch 29/200
```

Нарисую результаты.

```
x = np.arange(0, 5, 0.025/4)

predictions = model(x)

fig, ax = plt.subplots(1, 2, figsize=(15, 6))
ax[0].plot(x, f(x))
ax[0].plot(x, predictions)

ax[1].plot(hist.history['loss'])

plt.show()
```



Вывод: в ходе выполнения лабораторной работы я реализовал радиально базисный слой и с помощью него построил нейросетевые модели для классификации объектов и аппроксимации функции. Обучил обе модели и убедился, что сети с радиально базисными элементами могут быть продуктивными для решения задач классификации при линейной неразделимости выборки и аппроксимации функции.