Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 4

по курсу «Нейроинформатика»

Тема: Сети с радиальными базисными элементами.

Студент: Куликов А.В.
Группа: М80-408Б-17
Преподаватель: Аносова Н.П.
Дата: 19 ноября 2020
Оценка:

Москва, 2020

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

Основные этапы работы:

1. Использовать вероятностную нейронную сеть для классификации точек в случае, когда классы не являются линейно разделимыми.
2. Использовать сеть с радиальными базисными элементами (RBF) для классификации точек в случае, когда классы не являются линейно разделимыми.
3. Использовать обобщенно-регрессионную нейронную сеть для аппроксимации функции. Проверить работу сети с рыхлыми данными

Оборудование:

Процессор: AMD Ryzen 5 Mobile 3550H

Объем оперативной памяти: 8 Гб

Программное обеспечение:

Python 3.8.5

Сценарий выполнения работы:

Реализация вероятностной нейронной сети

```
import numpy as np

def radbas(x):
    return np.exp(-(x ** 2))

class pnn:
    def fit(self, x_train, y_train, spread=2):
        class_number = np.argmax(y_train, axis=1)
        y_train_sorted_index = class_number.argsort()

        self.unique_y = np.unique(y_train, axis=0)
        unique_y_sorted_index = np.argsort(np.argmax(self.unique_y, axis=1))
        self.unique_y = self.unique_y[unique_y_sorted_index]

        self.n_train = x_train.shape[0]
        self.n_classes = np.unique(class_number).shape[0]

        self.x_train = x_train[y_train_sorted_index]
        self.y_train = y_train[y_train_sorted_index]

        self.spread = spread

    def predict(self, x):
        n_samples = x.shape[0]
```

```
        res = []
        y_train_argmax = np.argmax(self.y_train, axis=1)
        for i in range(n_samples):
            dist = np.zeros(self.n_train)
            for j in range(self.n_train):
                dist[j] = np.sqrt(((x[i] - self.x_train[j]) ** 2).sum())

            a1 = radbas(0.8326 / self.spread * dist)

            max = 0
            max_index = 0
            for k in range(self.n_classes):
                cur_sum = a1[y_train_argmax == np.argmax(self.unique_y[k])].sum()
                if cur_sum > max:
                    max = cur_sum
                    max_index = k

            res.append(self.unique_y[max_index])

        return np.array(res)
```

Задание №1

```
import numpy as np
from pnn import pnn

from utils import *

import matplotlib.pyplot as plt

def rotate(point, alpha):
    c = np.cos(alpha)
    s = np.sin(alpha)

    rotation_matrix = np.array([[c, s], [-s, c]])
    return rotation_matrix @ point

def ellipse(a, b, alpha, x0, y0, n):
    t = np.linspace(0, 2*np.pi, n)

    c = np.cos(alpha)
    s = np.sin(alpha)

    points_x =  a * np.cos(t)
    points_y = b * np.sin(t)

    points_x_rotated = points_x * c + points_y * -s
    points_y_rotated = points_x * s + points_y * c

    points_x_rotated_shifted = points_x_rotated + x0
```

```
    points_y_rotated_shifted = points_y_rotated + y0

    return points_x_rotated_shifted, points_y_rotated_shifted

class1_n_points = 60
class2_n_points = 100
class3_n_points = 120

h = 0.025
n_elipse_points = int(2 * np.pi / h) + 1

# вариант 8
class1_points_x, class1_points_y = ellipse(0.2, 0.2, 0, -
0.25, 0.25, n_elipse_points)
class2_points_x, class2_points_y = ellipse(0.7, 0.5, -
np.pi/3, 0, 0, n_elipse_points)
class3_points_x, class3_points_y = ellipse(1, 1, 0, 0, 0, n_elipse_points)

class1_select = np.random.choice(n_elipse_points, class1_n_points, replace=False)
class2_select = np.random.choice(n_elipse_points, class2_n_points, replace=False)
class3_select = np.random.choice(n_elipse_points, class3_n_points, replace=False)

fig, ax = plt.subplots()

ax.scatter(class1_points_x[class1_select], class1_points_y[class1_select], c='red
', s=2)
ax.scatter(class2_points_x[class2_select], class2_points_y[class2_select], c='gre
en', s=2)
ax.scatter(class3_points_x[class3_select], class3_points_y[class3_select], c='blu
e', s=2)
ax.grid()

class1_X = np.column_stack([np.take(class1_points_x, class1_select), np.take(clas
s1_points_y, class1_select)])
class2_X = np.column_stack([np.take(class2_points_x, class2_select), np.take(clas
s2_points_y, class2_select)])
class3_X = np.column_stack([np.take(class3_points_x, class3_select), np.take(clas
s3_points_y, class3_select)])

class1_Y = np.tile(np.array([1, 0, 0]), (class1_n_points, 1))
class2_Y = np.tile(np.array([0, 1, 0]), (class2_n_points, 1))
class3_Y = np.tile(np.array([0, 0, 1]), (class3_n_points, 1))

n_samples = class1_n_points + class2_n_points + class3_n_points
permutation = np.random.permutation(n_samples)

X = np.take(np.vstack([class1_X, class2_X, class3_X]), permutation, axis=0)
Y = np.take(np.vstack([class1_Y, class2_Y, class3_Y]), permutation, axis=0)

x_train, y_train, x_test, y_test, x_valid, y_valid = split_dataset(X, Y, 0.2, 0)
```

```
classifier = pnn()
spread = 0.3
classifier.fit(x_train, y_train, spread=spread)

y_pred = classifier.predict(x_train)
correctly_classified = row_wise_equal(y_pred, y_train)
print(f'{correctly_classified}/{x_train.shape[0]} ({correctly_classified / x_trai
n.shape[0]}) points of train set are classified correctly!')

y_pred = classifier.predict(x_test)
correctly_classified = row_wise_equal(y_pred, y_test)
print(f'{correctly_classified}/{x_test.shape[0]} ({correctly_classified / x_test.
shape[0]}) points of test set are classified correctly!')

x = np.arange(-1.2, 1.2, h)
y = np.arange(-1.2, 1.2, h)

x_, y_ = np.meshgrid(x, y)

points = np.vstack([x_.flatten(), y_.flatten()]).T

pred = np.where(classifier.predict(points) > 0.5, 1, 0)

class_labels = np.argmax(pred, axis=1)

class1_grid_points = points[class_labels == 0]
class2_grid_points = points[class_labels == 1]
class3_grid_points = points[class_labels == 2]

x_class1_grid = class1_grid_points[:,0]
y_class1_grid = class1_grid_points[:,1]

x_class2_grid = class2_grid_points[:,0]
y_class2_grid = class2_grid_points[:,1]

x_class3_grid = class3_grid_points[:,0]
y_class3_grid = class3_grid_points[:,1]

ax.scatter(x_class1_grid, y_class1_grid, color=(1, 0, 0, 0.3), s=3)
ax.scatter(x_class2_grid, y_class2_grid, color=(0, 1, 0, 0.3), s=3)
ax.scatter(x_class3_grid, y_class3_grid, color=(0, 0, 1, 0.3), s=3)

ax.grid()

ax.set_title('Классификация точек плоскости')
plt.savefig('1.png')
plt.show()
```

Вывод программы при SPREAD = 0.3:

```
212/224 (0.9464285714285714) points of train set are classified correctly!

54/56 (0.9642857142857143) points of test set are classified correctly!
```

Вывод программы при SPREAD = 0.1:

```
224/224 (1.0) points of train set are classified correctly!

55/56 (0.9821428571428571) points of test set are classified correctly!
```

Реализация сети с радиальными базисными элементами

```
import numpy as np

def radbas(x):
    return np.exp(-(x ** 2))

class rbf:
    def fit(self, x_train, y_train, spread=2):
        class_number = np.argmax(y_train, axis=1)
        y_train_sorted_index = class_number.argsort()

        self.unique_y = np.unique(y_train, axis=0)
        unique_y_sorted_index = np.argsort(np.argmax(self.unique_y, axis=1))
        self.unique_y = self.unique_y[unique_y_sorted_index]

        self.n_train = x_train.shape[0]
        self.n_classes = np.unique(class_number).shape[0]

        self.x_train = x_train[y_train_sorted_index]
        self.y_train = y_train[y_train_sorted_index]

        self.spread = spread

        A = np.zeros((self.n_train, self.n_train))
        for i in range(self.n_train):
            for j in range(i+1, self.n_train):
                dist = np.sqrt(((self.x_train[i] - self.x_train[j]) ** 2).sum())
                A[i][j] = dist
                A[j][i] = dist

        A = radbas(0.8326 / self.spread * A)
        A_stacked = np.hstack([A, np.ones(self.n_train)[np.newaxis].T])
        wb = np.linalg.pinv(A_stacked) @ self.y_train

        self.W = wb[:-1:]
        self.b = wb[-1]

    def predict(self, x):
        n_samples = x.shape[0]

        res = []
        for i in range(n_samples):
            dist = np.zeros(self.n_train)
            for j in range(self.n_train):
                dist[j] = np.sqrt(((x[i] - self.x_train[j]) ** 2).sum())

            a1 = radbas(0.8326 / self.spread * dist)
            res.append(a1 @ self.W + self.b)

        return np.array(res)
```

Задание №2

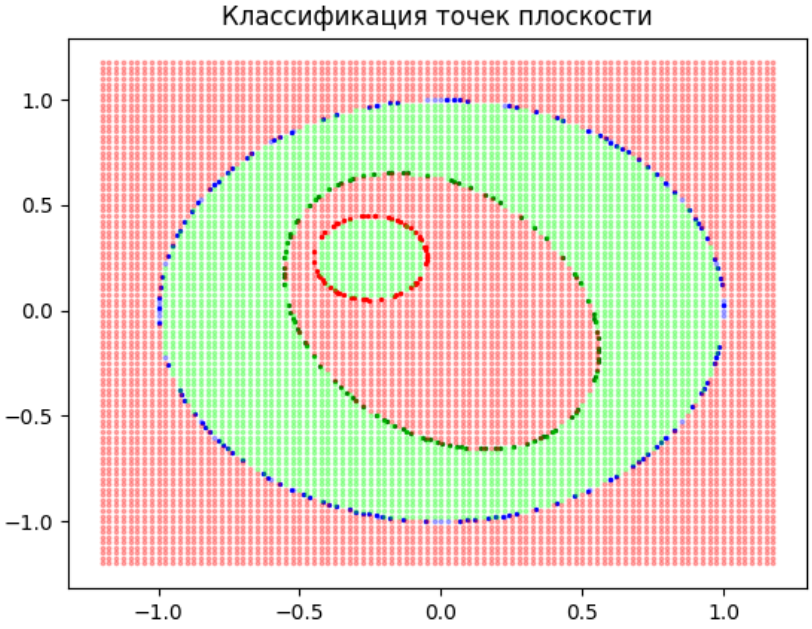
```
import numpy as np
from rbf import rbf

from utils import *

import matplotlib.pyplot as plt

def rotate(point, alpha):
    c = np.cos(alpha)
    s = np.sin(alpha)

    rotation_matrix = np.array([[c, s], [-s, c]])
    return rotation_matrix @ point

def ellipse(a, b, alpha, x0, y0, n):
    t = np.linspace(0, 2*np.pi, n)

    c = np.cos(alpha)
    s = np.sin(alpha)

    points_x =  a * np.cos(t)
    points_y = b * np.sin(t)

    points_x_rotated = points_x * c + points_y * -s
    points_y_rotated = points_x * s + points_y * c

    points_x_rotated_shifted = points_x_rotated + x0
    points_y_rotated_shifted = points_y_rotated + y0

    return points_x_rotated_shifted, points_y_rotated_shifted

class1_n_points = 60
class2_n_points = 100
class3_n_points = 120

h = 0.025
n_elipse_points = int(2 * np.pi / h) + 1

# вариант 8
class1_points_x, class1_points_y = ellipse(0.2, 0.2, 0, -
0.25, 0.25, n_elipse_points)
class2_points_x, class2_points_y = ellipse(0.7, 0.5, -
np.pi/3, 0, 0, n_elipse_points)
class3_points_x, class3_points_y = ellipse(1, 1, 0, 0, 0, n_elipse_points)

class1_select = np.random.choice(n_elipse_points, class1_n_points, replace=False)
class2_select = np.random.choice(n_elipse_points, class2_n_points, replace=False)
class3_select = np.random.choice(n_elipse_points, class3_n_points, replace=False)
```

```
fig, ax = plt.subplots()

ax.scatter(class1_points_x[class1_select], class1_points_y[class1_select], c='red
', s=2)
ax.scatter(class2_points_x[class2_select], class2_points_y[class2_select], c='gre
en', s=2)
ax.scatter(class3_points_x[class3_select], class3_points_y[class3_select], c='blu
e', s=2)
ax.grid()

class1_X = np.column_stack([np.take(class1_points_x, class1_select), np.take(clas
s1_points_y, class1_select)])
class2_X = np.column_stack([np.take(class2_points_x, class2_select), np.take(clas
s2_points_y, class2_select)])
class3_X = np.column_stack([np.take(class3_points_x, class3_select), np.take(clas
s3_points_y, class3_select)])

class1_Y = np.tile(np.array([1, 0, 0]), (class1_n_points, 1))
class2_Y = np.tile(np.array([0, 1, 0]), (class2_n_points, 1))
class3_Y = np.tile(np.array([0, 0, 1]), (class3_n_points, 1))

n_samples = class1_n_points + class2_n_points + class3_n_points
permutation = np.random.permutation(n_samples)

X = np.take(np.vstack([class1_X, class2_X, class3_X]), permutation, axis=0)
Y = np.take(np.vstack([class1_Y, class2_Y, class3_Y]), permutation, axis=0)

x_train, y_train, x_test, y_test, x_valid, y_valid = split_dataset(X, Y, 0.2, 0)

classifier = rbf()
spread = 0.3
classifier.fit(x_train, y_train, spread=spread)

y_pred = np.where(classifier.predict(x_train) > 0.5, 1, 0)
correctly_classified = row_wise_equal(y_pred, y_train)
print(f'{correctly_classified}/{x_train.shape[0]} ({correctly_classified / x_trai
n.shape[0]}) points of train set are classified correctly!')

y_pred = np.where(classifier.predict(x_test) > 0.5, 1, 0)
correctly_classified = row_wise_equal(y_pred, y_test)
print(f'{correctly_classified}/{x_test.shape[0]} ({correctly_classified / x_test.
shape[0]}) points of test set are classified correctly!')

x = np.arange(-1.2, 1.2, h)
y = np.arange(-1.2, 1.2, h)

x_, y_ = np.meshgrid(x, y)

points = np.vstack([x_.flatten(), y_.flatten()]).T

pred = classifier.predict(points)
```

```
class_labels = np.argmax(pred, axis=1)

class1_grid_points = points[class_labels == 0]
class2_grid_points = points[class_labels == 1]
class3_grid_points = points[class_labels == 2]

x_class1_grid = class1_grid_points[:,0]
y_class1_grid = class1_grid_points[:,1]

x_class2_grid = class2_grid_points[:,0]
y_class2_grid = class2_grid_points[:,1]

x_class3_grid = class3_grid_points[:,0]
y_class3_grid = class3_grid_points[:,1]

ax.scatter(x_class1_grid, y_class1_grid, color=(1, 0, 0, 0.3), s=3)
ax.scatter(x_class2_grid, y_class2_grid, color=(0, 1, 0, 0.3), s=3)
ax.scatter(x_class3_grid, y_class3_grid, color=(0, 0, 1, 0.3), s=3)

ax.grid()

ax.set_title('Классификация точек плоскости')
plt.savefig('3.png')
plt.show()
```

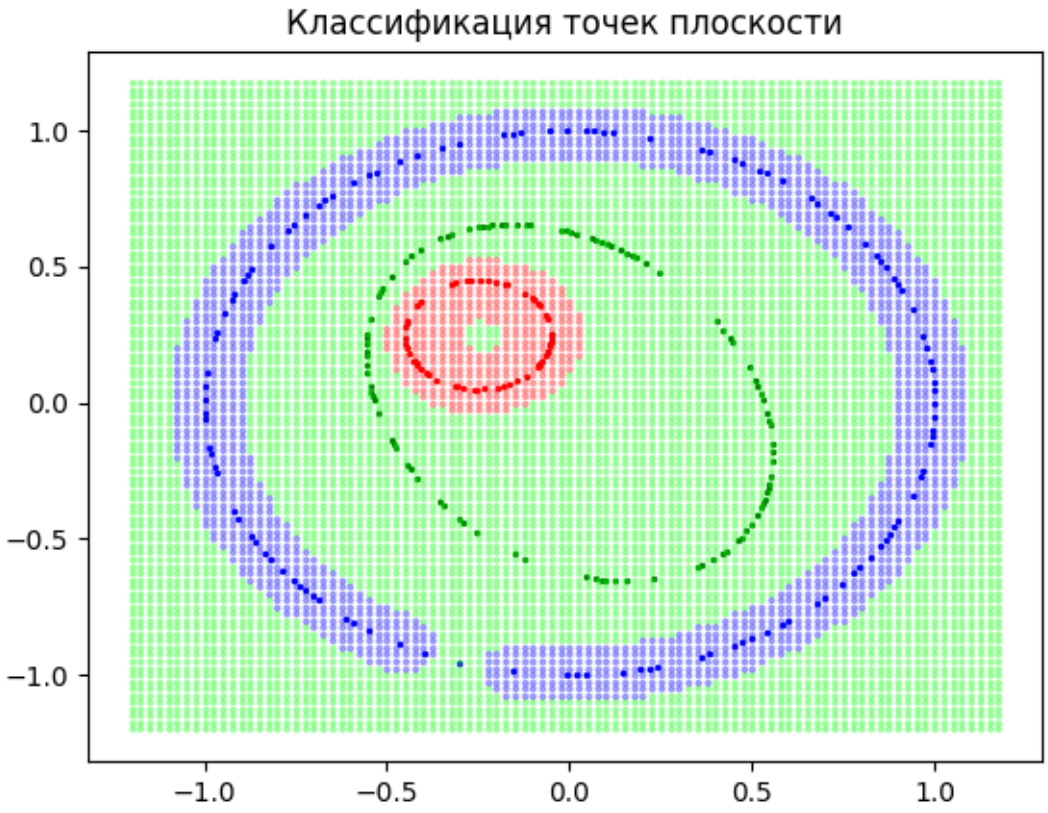
Вывод программы при SPREAD = 0.3:

```
224/224 (1.0) points of train set are classified correctly!

56/56 (1.0) points of test set are classified correctly!
```



Вывод программы при SPREAD = 0.1:

```
224/224 (1.0) points of train set are classified correctly!

55/56 (0.9821428571428571) points of test set are classified correctly!
```



Реализация обобщенно-регрессионной нейронной сети

```python
import numpy as np

def radbas(x):
    return np.exp(-(x ** 2))

eps = 1e-309

class grnn:
    def fit(self, x_train, y_train, spread=2):
        class_number = np.argmax(y_train, axis=1)
        y_train_sorted_index = class_number.argsort()

        self.unique_y = np.unique(y_train, axis=0)
        unique_y_sorted_index = np.argsort(np.argmax(self.unique_y, axis=1))
        self.unique_y = self.unique_y[unique_y_sorted_index]

        self.n_train = x_train.shape[0]
        self.n_classes = np.unique(class_number).shape[0]
```

```
        self.x_train = x_train[y_train_sorted_index]
        self.y_train = y_train[y_train_sorted_index]

        self.spread = spread

    def predict(self, x):
        n_samples = x.shape[0]

        res = []
        for i in range(n_samples):
            dist = np.zeros(self.n_train)
            for j in range(self.n_train):
                dist[j] = np.sqrt(((x[i] - self.x_train[j]) ** 2).sum())

            a1 = radbas(0.8326 / self.spread * dist)

            a1_sum = a1.sum()
            if a1_sum < eps:
                a1_sum = eps

            res.append(a1 @ self.y_train / a1_sum)

        return np.array(res)
```

Задание №3

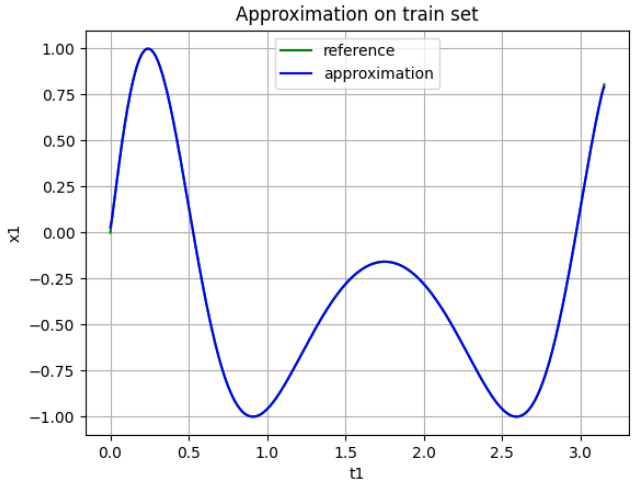
```
import numpy as np
import matplotlib.pyplot as plt

from grnn import grnn
from utils import *

def mse(A, B):
    return ((A - B) ** 2).sum() / A.shape[0]

def rmse(A, B):
    return np.sqrt(mse(A, B))

def phi(t):
    return np.sin(-2*t*t + 7*t)

t_begin = 0
t_end = 3.5
h = 0.01

n_points = int((t_end - t_begin) / h) + 1
x = np.linspace(t_begin, t_end, n_points).reshape((-1,1))

y = phi(x)
```

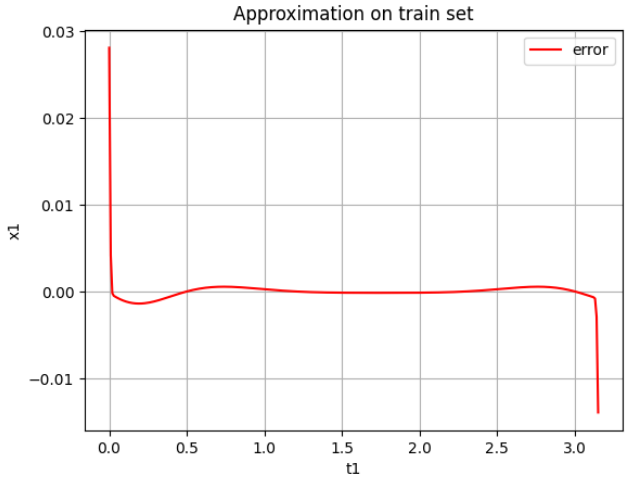
```
x_train, y_train, x_test, y_test, x_valid, y_valid = split_dataset(x, y, 0.0, 0.1
)

approximator = grnn()

approximator.fit(x_train, y_train, spread=h)

y_pred = approximator.predict(x_train)
print(f'RMSE on train set: {rmse(y_train, y_pred)}')

fig, ax = plt.subplots()
ax.plot(x_train, y_train, 'g', label='reference')
ax.plot(x_train, y_pred, 'b', label='approximation')

ax.set(xlabel='t1', ylabel='x1', title='Approximation on train set')
ax.grid()
ax.legend()

plt.savefig('5.png')
plt.show()

error = y_pred - y_train
fig, ax = plt.subplots()
ax.plot(x_train, error, 'r', label='error')

ax.set(xlabel='t1', ylabel='x1', title='Approximation on train set')
ax.grid()
ax.legend()

plt.savefig('6.png')
plt.show()

y_pred = approximator.predict(x_valid)

print(f'RMSE on validation set: {rmse(y_valid, y_pred)}')

fig, ax = plt.subplots()
ax.plot(x_valid, y_valid, 'g', label='reference')
ax.plot(x_valid, y_pred, 'b', label='approximation')

ax.set(xlabel='t1', ylabel='x1', title='Approximation on validation set')
ax.grid()
ax.legend()

plt.savefig('7.png')
plt.show()

error = y_pred - y_valid
fig, ax = plt.subplots()
ax.plot(x_valid, error, 'r', label='error')
```

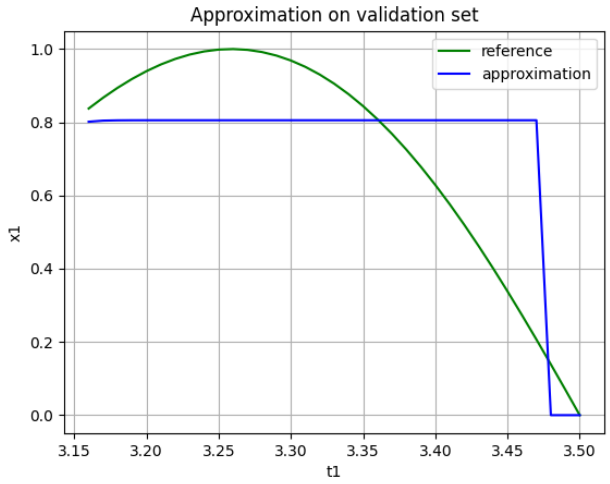
```
ax.set(xlabel='t1', ylabel='x1', title='Approximation on validation set')
ax.grid()
ax.legend()

plt.savefig('8.png')
plt.show()

# проверка для разреженных данных
x_train, y_train, x_test, y_test, x_valid, y_valid = split_dataset(x_train, y_tra
in, 0.2, 0)

approximator = grnn()

approximator.fit(x_train, y_train, spread=h)

y_pred = approximator.predict(x_train)
print(f'RMSE on train set: {rmse(y_train, y_pred)}')

error = y_pred - y_train

fig, ax = plt.subplots()
ax.plot(x_train, y_train, 'g', label='reference')
ax.plot(x_train, y_pred, 'b', label='approximation')
ax.plot(x_train, error, 'r', label='error')

ax.set(xlabel='t1', ylabel='x1', title='Approximation on train set')
ax.grid()
ax.legend()

plt.savefig('9.png')
plt.show()
```

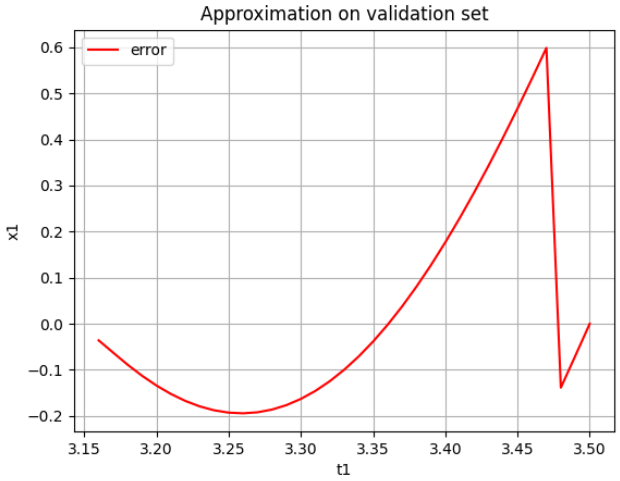
Вывод программы:

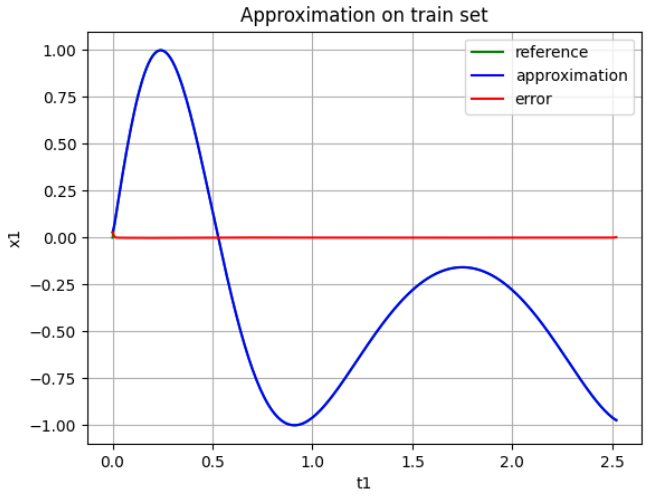
```
RMSE on train set: 0.0018475984999177665
RMSE on validation set: 0.22525446462346232
RMSE on train set with sparse data: 0.0018585687948049177
```




Approximation on train set
error
x1
t1
0.03
0.02
0.01
0.00
−0.01
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Approximation on validation set
reference
approximation
x1
t1
1.0
0.8
0.6
0.4
0.2
0.0
3.15
3.20
3.25
3.30
3.35
3.40
3.45
3.50


Approximation on validation set
error
x1
t1
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
3.15
3.20
3.25
3.30
3.35
3.40
3.45
3.50



Выводы:

Радиально-базисные сети – альтернативное направлении в нейронных сетях.

Процесс обучения в основном не использует алгоритм обратного распространения ошибки. Так, в вероятностной нейронной сети и в GRNN обучение отсутствует совсем, для обучения же RBF-сети необходимо решить СЛАУ. Таким образом процесс обучения радиально-базисных сетей занимает меньше времени, чем обучение многослойного персептрона.

Радиально-базисные сети локально-рецептивны. Поэтому изменение обучающего набора не требует полного переобучения сети.

Классификация и регрессия может занимать существенно больше времени по сравнению с многослойным персептроном т.к. для каждого примера из обучающего множества (а их может быть очень много) необходимо вычислять радиально-базисную функцию.

Так же нужно отметить, что первый из слоев любой радиально-базисной сети производит нелинейное преобразование входного вектора и повышает размерность вектора признаков, что положительно сказывается на качестве классификации и позволяет решать задачу, в которой классы линейно неразделимы.