ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное бюджетное образовательное учреждение высшего образования "Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича"

Факультет «Инфокоммуникационных сетей и систем»

Кафедра «Программной инженерии и вычислительной техники»

Отчет по лабораторной работе №2

*Дисциплина: «Системы искусственных интеллектов»*

Выполнил студент гр. ИКПИ-84

Василец Павел

Приняла: Белая Т.И.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Санкт-Петербург**

**2021 г.**

**«Классификация с помощью перцептрона»**

**Постановка задачи:**

Требуется построить нейронную сеть, которая производит классификацию точек на координатной плоскости на 4 класса.

|  |  |
| --- | --- |
| ***Кол-во классов*** | ***Координаты точек проверочного множества и номер класса, к которому принадлежит каждая точка*** |
| 4 | [2, 0] – 1; [1, 0] – 2; [0, 1] – 3; |

*Таблица 1 – Исходные данные*

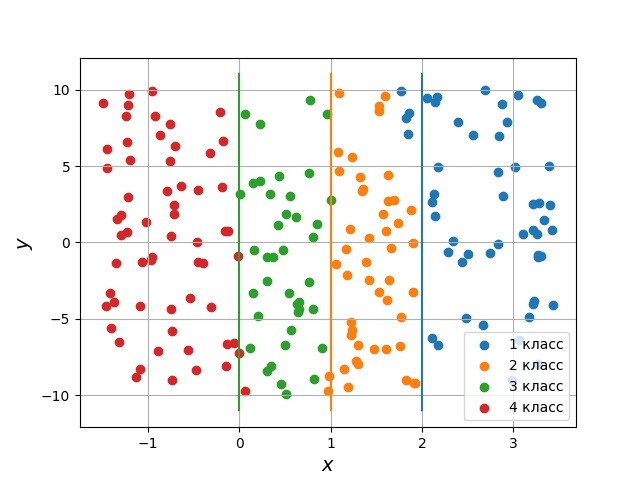
**Ход работы:**

Параметры нейронной сети:

* Входной слой;
* Скрытый слой (10 нейронов)
* Выходной слой;
* В качестве функции потерь используется дискретная перекрестная энтропия;
* В качестве функции активации используется выпрямленный линейный блок (ReLU);
* В качестве оптимизатора используется аппроксимация градиентного спуска (стохастический градиентный спуск);

Наибольше показатели точности были достигнуты при следующих параметрах обучения:

* Скорость обучения: 0.01;
* Размер пакетов: 70 точек;
* Количество эпох (итераций обучения): 7 500;



*Рисунок 1 – Классифицированные моделью точки*

Точность предсказаний модели: 0.985

**Листинг:**

import matplotlib.pyplot as plt

import numpy as np

import torch

def get\_data(batch\_size):

    data = []

    targets = np.zeros(batch\_size)

    for batch\_i in range(batch\_size):

        x\_point = [np.random.uniform(-1.5, 3.5), np.random.uniform(-10., 10.)]

        data.append(x\_point)

        if x\_point[0] >= 2:

            targets[batch\_i] = 0

        elif 1 <= x\_point[0] < 2:

            targets[batch\_i] = 1

        elif 0 <= x\_point[0] < 1:

            targets[batch\_i] = 2

        elif x\_point[0] < 0:

            targets[batch\_i] = 3

    return torch.FloatTensor(data), torch.LongTensor(targets)

def get\_x\_y\_from\_tensor(tensor):

    x\_points = []

    y\_points = []

    for i in range(tensor.size()[0]):

        x\_points.append(tensor[i][0].item())

        y\_points.append(tensor[i][1].item())

    return x\_points, y\_points

def accuracy\_score(preds, targets):

    preds = preds.data.numpy()

    preds = np.argmax(preds, axis=1)

    total = len(targets.data.numpy())

    correct = 0

    for i in range(len(targets)):

        if preds[i] == targets[i]:

            correct += 1

    # print("Целевые переменные: ", targets.data.numpy())

    # print("Предсказания модели:", preds)

    print("Точность: ", correct / total)

def point\_classification(test\_data, preds):

    first\_class\_x\_points = []

    first\_class\_y\_points = []

    second\_class\_x\_points = []

    second\_class\_y\_points = []

    third\_class\_x\_points = []

    third\_class\_y\_points = []

    fourth\_class\_x\_points = []

    fourth\_class\_y\_points = []

    preds = preds.data.numpy()

    preds = np.argmax(preds, axis=1)

    for i in range(test\_data.size()[0]):

        if preds[i] == 0:

            first\_class\_x\_points.append(test\_data[i][0].item())

            first\_class\_y\_points.append(test\_data[i][1].item())

        elif preds[i] == 1:

            second\_class\_x\_points.append(test\_data[i][0].item())

            second\_class\_y\_points.append(test\_data[i][1].item())

        elif preds[i] == 2:

            third\_class\_x\_points.append(test\_data[i][0].item())

            third\_class\_y\_points.append(test\_data[i][1].item())

        elif preds[i] == 3:

            fourth\_class\_x\_points.append(test\_data[i][0].item())

            fourth\_class\_y\_points.append(test\_data[i][1].item())

    return first\_class\_x\_points, first\_class\_y\_points, second\_class\_x\_points, second\_class\_y\_points, third\_class\_x\_points, third\_class\_y\_points, fourth\_class\_x\_points, fourth\_class\_y\_points

def visualize\_data(train\_data, preds):

    x1\_points, y1\_points, x2\_points, y2\_points, x3\_points, y3\_points, x4\_points, y4\_points = point\_classification(

        train\_data, preds)

    classes = preds.data.numpy()

    classes = np.argmax(classes, axis=1)

    x, y = get\_x\_y\_from\_tensor(train\_data)

    plt.figure(1)

    plt.grid()

    plt.xlabel("$x$", fontsize=14)

    plt.ylabel("$y$", fontsize=14)

    plt.plot([2, 2], [-11, 11])

    plt.plot([1, 1], [-11, 11])

    plt.plot([0, 0], [-11, 11])

    plt.plot([-1.5, -1.5], [-11, 11])

    plt.scatter(x, y)

    plt.figure(2)

    plt.plot([2, 2], [-11, 11])

    plt.plot([1, 1], [-11, 11])

    plt.plot([0, 0], [-11, 11])

    plt.plot([-1.5, -1.5], [-11, 11])

    plt.scatter(x1\_points, y1\_points, label="1 класс")

    plt.scatter(x2\_points, y2\_points, label="2 класс")

    plt.scatter(x3\_points, y3\_points, label="3 класс")

    plt.scatter(x4\_points, y4\_points, label="4 класс")

    plt.legend()

    plt.grid()

    plt.xlabel("$x$", fontsize=14)

    plt.ylabel("$y$", fontsize=14)

    plt.show()

batch\_size = 70

input\_dim = 2

hidden\_dimension = 10

output\_dimension = 4

learning\_rate = 0.01

epochs = 7500

two\_layer\_net = torch.nn.Sequential(

    torch.nn.Linear(input\_dim, hidden\_dimension),

    torch.nn.ReLU(),

    torch.nn.Linear(hidden\_dimension, output\_dimension)

)

ce\_loss = torch.nn.CrossEntropyLoss(size\_average=False)

optimizer = torch.optim.SGD(two\_layer\_net.parameters(), lr=learning\_rate)

print("Обучение запущено")

for epoch in range(epochs):

    train\_data, train\_targets = get\_data(batch\_size)

    preds = two\_layer\_net(train\_data)

    loss = ce\_loss(preds, train\_targets)

    optimizer.zero\_grad()

    # print(loss.item())

    loss.backward()

    optimizer.step()

    if epoch % 500 == 0:

        print(epoch, "эпоха")

print("Обучение окончено")

test\_data, test\_targets = get\_data(200)

check = torch.FloatTensor([[0.99, 10]])

preds = two\_layer\_net(test\_data)

accuracy\_score(preds, test\_targets)

visualize\_data(test\_data, preds)

**«Аппроксимация функции»**

**Постановка задачи:**

Для заданной функции построить ее табличные значения (количество значений должно быть достаточным для того, чтобы аппроксимированная функция визуально совпадала с табличными значениями).

Использовать нейронную сеть для аппроксимации этих табличных значений и нахождения аппроксимированной функции. Вывести на график табличные значения (точками) и аппроксимированную функцию (линией).

|  |  |
| --- | --- |
| ***Вид функции*** | ***Промежуток нахождения решения*** |
| cos(exp(x))/sin(ln(x)) | x∈[2,4] |

*Таблица 2 – Исходные данные*

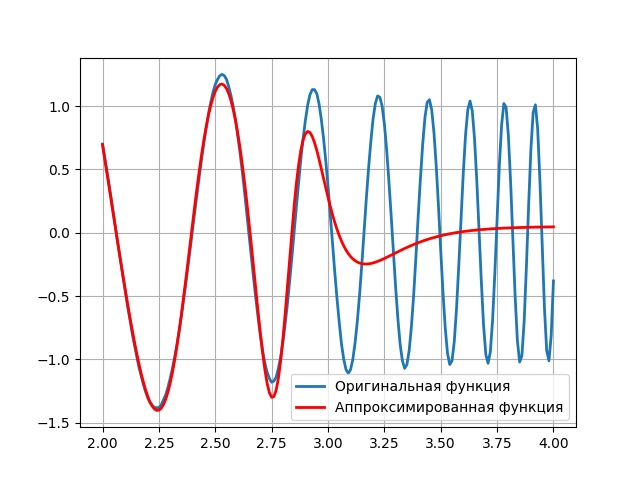
**Ход работы:**

Параметры нейронной сети:

* Входной слой;
* Скрытый слой (5 нейронов)
* Выходной слой;
* В качестве функции потерь используется среднеквадратичных потерь (MSELoss);
* В качестве функции активации используется сигмоида (Sigmoid);
* В качестве оптимизатора используется Adam.

Наибольше показатели точности были достигнуты при следующих параметрах обучения:

* Скорость обучения: 0.01;
* Обучение по всем имеющимся данным сразу без разделения на пакеты;
* Количество эпох (итераций обучения): 20 000;



*Рисунок 2 – График аппроксимированной моделью функции*

**Листинг:**

import math

import random

import numpy as np

import torch as torch

import torch.nn as nn

import torch.optim as optim

import matplotlib.pyplot as plt

def get\_data():

    data = []

    targets = []

    i = 0

    for batch\_i in np.arange(2.0, 4.01, 0.01):

        data.append([batch\_i])

        targets.append([round(math.cos(math.exp(data[i][0]))/math.sin(math.log(data[i][0], math.e)), 2)])

        i += 1

    return torch.FloatTensor(data), torch.FloatTensor(targets)

def get\_x\_y\_from\_tensor(tensor):

    x\_points = []

    y\_points = []

    ic = 2.0

    for i in range(tensor.size()[0]):

        x\_points.append(ic)

        y\_points.append(tensor[i].item())

        ic += 0.01

    return x\_points, y\_points

def shuffle\_data(train\_data, train\_targets):

    dat\_n\_tar = list()

    shuffled\_data = list()

    shuffled\_targets = list()

    for i in range(len(train\_targets)):

        dat\_n\_tar.append([train\_data[i].item(), train\_targets[i].item()])

    random.shuffle(dat\_n\_tar)

    for data in dat\_n\_tar:

        shuffled\_data.append([data[0]])

        shuffled\_targets.append([data[1]])

    return torch.FloatTensor(shuffled\_data), torch.FloatTensor(shuffled\_targets)

learning\_rate = 0.01

input\_dimension = 1

hidden\_dimension = 5

output\_dimension = 1

epochs = 20000

three\_layer\_net = nn.Sequential(

    nn.Linear(input\_dimension, hidden\_dimension),

    nn.Sigmoid(),

    nn.Linear(hidden\_dimension, output\_dimension)

)

loss\_fn = nn.MSELoss(size\_average=False)

optimizer = optim.Adam(three\_layer\_net.parameters(), lr=learning\_rate)

train\_data, train\_targets = get\_data()

train\_data = torch.round(train\_data \* 10\*\*2) / (10\*\*2)

for epoch in range(epochs):

    preds = three\_layer\_net(train\_data)

    loss = loss\_fn(preds, train\_targets)

    optimizer.zero\_grad()

    loss.backward()

    optimizer.step()

    if epoch % 500 == 0:

        print(epoch, "эпоха")

print("Обучение окончено")

train\_data, train\_targets = get\_data()

x\_original\_points, y\_original\_points = get\_x\_y\_from\_tensor(train\_data)

preds = three\_layer\_net(train\_data)

x\_approximation\_points, y\_approximation\_points = get\_x\_y\_from\_tensor(preds)

plt.plot(x\_original\_points, train\_targets, linewidth=2, label="Оригинальная функция")

plt.plot(x\_approximation\_points, y\_approximation\_points, color="red", linewidth=2, label="Аппроксимированная функция")

plt.grid()

plt.legend()

plt.show()

**«Классификация с помощью слоя Кохонена»**

**Постановка задачи:**

Требуется построить нейронную сеть, которая производит классификацию точек на координатной плоскости на 4 класса на основе слоя Кохонена.

|  |  |
| --- | --- |
| ***Кол-во классов*** | ***Координаты точек проверочного множества и номер класса, к которому принадлежит каждая точка*** |
| 4 | [2, 0] – 1; [1, 0] – 2; [0, 1] – 3; |

*Таблица 3 – Исходные данные*

**Ход работы:**

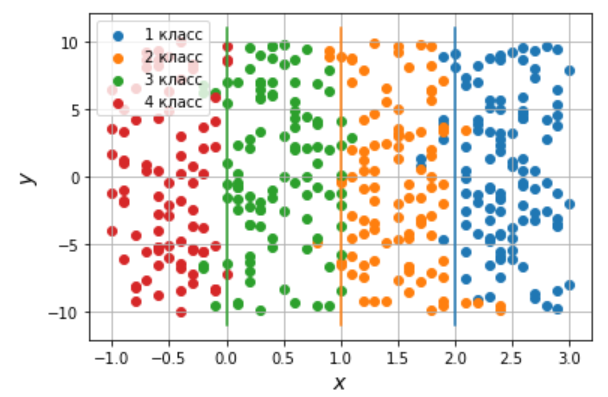
Было решено использовать алгоритмы машинного обучения, реализуемые в пакете «minisom».

Параметры нейронной сети:

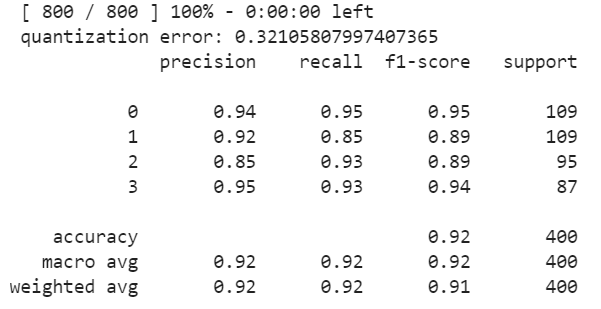
* Размер карты Кохонена 21х21;
* Функция соседства Гауссовская;

Наибольше показатели точности были достигнуты при следующих параметрах обучения:

* Скорость обучения: 0.2;
* Обучение по всем имеющимся данным сразу без разделения на пакеты;
* Количество эпох (итераций обучения): 800;

****

*Рисунок 3 – Классифицированные моделью точки*

****

*Рисунок 4 – Распределение точности модели относительного каждого класса*

**Листинг:**

import numpy as np

import matplotlib.pyplot as plt

from minisom import MiniSom

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

def get\_data(batch\_size):

    data = np.empty(shape=[0, 2])

    targets = np.zeros(batch\_size)

    for batch\_i in range(batch\_size):

        x\_point = [round(np.random.uniform(-1.0, 3.0), 1), round(np.random.uniform(-10., 10.), 1)]

        data = np.append(data, [x\_point], axis=0)

        if x\_point[0] >= 2:

            targets[batch\_i] = 0

        elif x\_point[0] >= 1 and x\_point[0] < 2:

            targets[batch\_i] = 1

        elif x\_point[0] >= 0 and x\_point[0] < 1:

            targets[batch\_i] = 2

        elif x\_point[0] < 0:

            targets[batch\_i] = 3

    targets = targets.astype(int)

    return data, targets

def classify(som, data):

    winmap = som.labels\_map(train\_data, train\_targets)

    default\_class = np.sum(list(winmap.values())).most\_common()[0][0]

    result = []

    for d in data:

        win\_position = som.winner(d)

        if win\_position in winmap:

            result.append(winmap[win\_position].most\_common()[0][0])

        else:

            result.append(default\_class)

    return result

def point\_classification(test\_data, preds):

    first\_class\_x\_points = []

    first\_class\_y\_points = []

    second\_class\_x\_points = []

    second\_class\_y\_points = []

    third\_class\_x\_points = []

    third\_class\_y\_points = []

    fourth\_class\_x\_points = []

    fourth\_class\_y\_points = []

    for i in range(len(test\_data)):

        if preds[i] == 0:

            first\_class\_x\_points.append(test\_data[i][0])

            first\_class\_y\_points.append(test\_data[i][1])

        elif preds[i] == 1:

            second\_class\_x\_points.append(test\_data[i][0])

            second\_class\_y\_points.append(test\_data[i][1])

        elif preds[i] == 2:

            third\_class\_x\_points.append(test\_data[i][0])

            third\_class\_y\_points.append(test\_data[i][1])

        elif preds[i] == 3:

            fourth\_class\_x\_points.append(test\_data[i][0])

            fourth\_class\_y\_points.append(test\_data[i][1])

    return first\_class\_x\_points, first\_class\_y\_points, second\_class\_x\_points, second\_class\_y\_points, third\_class\_x\_points, third\_class\_y\_points, fourth\_class\_x\_points, fourth\_class\_y\_points

train\_data, train\_targets = get\_data(5000)

zero, one, two, three = train\_targets.tolist().count(0), train\_targets.tolist().count(1), train\_targets.tolist().count(2), train\_targets.tolist().count(3)

print(f'zero:{zero}\none:{one}\ntwo:{two}\nthree:{three}')

test\_data, test\_targets = get\_data(400)

som = MiniSom(21, 21, 2, sigma=4, learning\_rate=0.2,

             neighborhood\_function='gaussian', random\_seed=15)

som.pca\_weights\_init(train\_data)

som.train\_random(train\_data, 800, verbose=True)

print(classification\_report(test\_targets, classify(som, test\_data)))

preds = classify(som, test\_data)

preds = np.array(preds)

fi\_class\_x, fi\_class\_y, s\_class\_x, s\_class\_y, t\_class\_x, t\_class\_y, fo\_class\_x, fo\_class\_y = point\_classification(test\_data, preds)

plt.figure(1)

plt.grid()

plt.xlabel("$x$", fontsize=14)

plt.ylabel("$y$", fontsize=14)

plt.plot([2, 2], [-11, 11])

plt.plot([1, 1], [-11, 11])

plt.plot([0, 0], [-11, 11])

plt.scatter(fi\_class\_x, fi\_class\_y, label="1 класс")

plt.scatter(s\_class\_x, s\_class\_y, label="2 класс")

plt.scatter(t\_class\_x, t\_class\_y, label="3 класс")

plt.scatter(fo\_class\_x, fo\_class\_y, label="4 класс")

plt.legend()

plt.show()

**«Прогнозирование временных рядов с использованием нейронных сетей**

**Постановка задачи:**

Сформировать на основе заданной функции обучающие множества. Построить нейронную сеть и обучить ее на полученных обучающих множествах прогнозировать поведение функции при смещении по оси абсцисс с заданным окном.

|  |  |
| --- | --- |
| ***Вид функции*** | ***Промежуток нахождения решения*** |
| cos(exp(t)) | x∈[2,4] |

*Таблица 4 – Исходные данные*

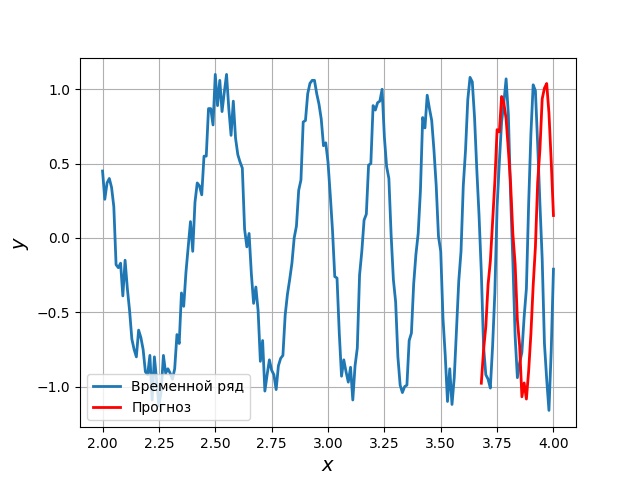
**Ход работы:**

Параметры нейронной сети:

* Входной слой;
* Скрытый слой (100 нейронов)
* Выходной слой;
* В качестве функции потерь используется Smooth L1 Loss;
* В качестве функции активации используется тангенс (Tahn);
* В качестве оптимизатора используется усредненный стохастический градиентный спуск (ASGD).

Наибольше показатели точности были достигнуты при следующих параметрах обучения:

* Скорость обучения: 0.0001;
* Обучение по всем имеющимся данным сразу без разделения на пакеты;
* Количество эпох (итераций обучения): 10 000;



*Рисунок 5 – График прогнозируемого моделью поведения функции*

**Листинг:**

import math

import numpy as np

import torch as torch

import matplotlib.pyplot as plt

func = "cos(exp(x))"

def get\_data():

    x\_points = np.array([])

    y\_points = np.array([])

    i = 0

    for batch\_i in np.arange(2., 4.01, 0.01):

        x\_points = np.append(x\_points, round(batch\_i, 2))

        y\_points = np.append(y\_points, round(math.cos(math.exp(x\_points[i])) + np.random.uniform(-0.2, 0.2), 2))

        i += 1

    return x\_points, y\_points

points\_count = 101

x, y = get\_data()

const = len(y) - (points\_count + 1)

samples\_data = list()

samples\_targets = list()

for i in range(const):

    samples\_data.append(y[i:i + points\_count])

    samples\_targets.append(y[i + points\_count+1:i + points\_count+2])

train\_data = samples\_data[len(samples\_targets) // 3:]

train\_targets = samples\_targets[len(samples\_targets) // 3:]

x\_train = torch.FloatTensor(train\_data)

y\_train = torch.FloatTensor(train\_targets)

test\_data = samples\_data[len(samples\_targets) // 3:len(samples\_targets) - (len(samples\_targets) // 3)]

test\_targets = samples\_targets[len(samples\_targets) // 3:len(samples\_targets) - (len(samples\_targets) // 3)]

x\_test = torch.FloatTensor(test\_data)

y\_test = torch.FloatTensor(test\_targets)

learning\_rate = 0.0001

epochs = 10000

three\_layer\_net = torch.nn.Sequential(

    torch.nn.Linear(points\_count, 100),

    torch.nn.Tanh(),

    torch.nn.Linear(100, 1),

)

loss\_fn = torch.nn.SmoothL1Loss(reduction='sum')

optim = torch.optim.ASGD(three\_layer\_net.parameters(), lr=learning\_rate)

for epoch in range(epochs):

    y\_pred = three\_layer\_net(x\_train)

    loss = loss\_fn(y\_pred, y\_train)

    optim.zero\_grad()

    loss.backward()

    optim.step()

    if epoch % 1000 == 0:

        print(epoch, "эпоха")

y\_preds\_test = three\_layer\_net(x\_test)

x\_graph\_preds = np.array([])

for i in range(len(x) - len(samples\_targets) // 3, len(x)):

    x\_graph\_preds = np.append(x\_graph\_preds, x[i])

y\_preds\_test = y\_preds\_test.data.numpy()

y\_preds\_test = y\_preds\_test.reshape(x\_graph\_preds.shape)

plt.plot(x, y, label='Временной ряд', linewidth=2)

plt.plot(x\_graph\_preds, y\_preds\_test, label='Прогноз', color='red', linewidth=2)

plt.xlabel('$x$', fontsize=14)

plt.ylabel('$y$', fontsize=14)

plt.grid()

plt.legend()

plt.show()