Министерство образования Республики Беларусь
УО «Брестский государственный технический университет»
Кафедра ИИТ

Лабораторные работы №1-5

По дисциплине: “Методы и алгоритмы принятия решений”

Тема: “Нейронные сети”

Вариант №9

Выполнил: студент 2 курса группы ПО-7
Крупенков Михаил Дмитриевич

Проверил: Крощенко А.А.

Брест 2021

Универсальная нейронная сеть

Для выполнения всех лабораторных работ в стенах Брестского Государственного Технического Университета была разработана универсальная динамическая нейронная сеть (universal dynamic neural network – uninn). Её особенность: вариативное количество слоев с разными функциями активации, количеством нейронов.

Ссылка на git:
https://github.com/FalseR20/miapr_po7/tree/main/reports/Krupenkov/universal/src

Код универсальной нейронной сети

funsact.py

```python
import numpy as np


def sigmoid(s):  # sigmoid
    return 1.0 / (1.0 + np.exp(-s))


def d_sigmoid(y):  # sigmoid'
    return y * (1.0 - y)


def linear(s):  # linear
    return s


def d_linear(_):  # linear'
    return 1.0
```

uninn.py

```python
import os
import pickle
import numpy as np
import funsact
from numpy.typing import NDArray
from typing import Optional, Callable


class Layer:
    """Слой нейросети"""

    def __init__(
            self,
            lens: tuple[int, int],
            f_act: Callable = funsact.linear,
            d_f_act: Callable = funsact.d_linear,
            w=None, t=None,
            is_need_vectorize=False
    ):
        """
        - lens (количество нейронов этого и следующего слоя)
        - функции активации
        """
        self.lens = lens
        self.w = np.random.uniform(-0.5, 0.5, lens) if w is None else w
        self.t = np.random.uniform(-0.5, 0.5, lens[1]) if t is None else t
        if is_need_vectorize:
```

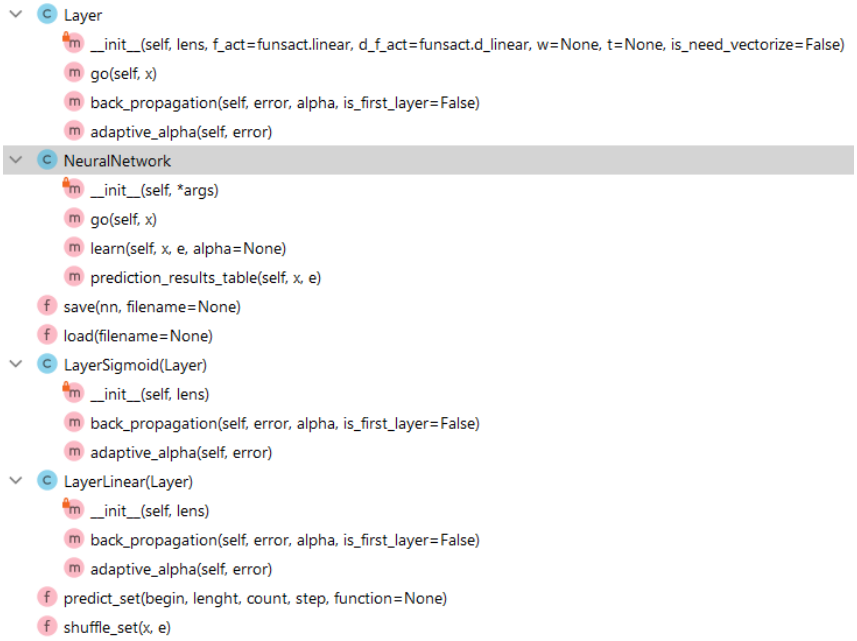
```
            self.f_act = np.vectorize(f_act)
            self.d_f_act = np.vectorize(d_f_act)
        else:
            self.f_act = f_act
            self.d_f_act = d_f_act

    def go(self, x: NDArray) -> NDArray:
        """Прохождение слоя"""
        self.x = x
        self.s: NDArray = np.dot(x, self.w) - self.t
        self.y: NDArray = self.f_act(self.s)
        return self.y

    def back_propagation(
            self,
            error: NDArray,
            alpha: Optional[float],
            is_first_layer=False
    ) -> Optional[NDArray]:
        """Обратное распространение ошибки с изменением весов, порога"""
        error_later = (
            np.dot(error * self.d_f_act(self.y), self.w.transpose())
            if not is_first_layer
            else None
        )

        if not alpha:
            alpha = self.adaptive_alpha(error)

        gamma = alpha * error * self.d_f_act(self.y)
        self.w -= np.dot(self.x.reshape(-1, 1), gamma.reshape(1, -1))
        self.t += gamma

        return error_later

    def adaptive_alpha(self, error: NDArray) -> float:
        if not hasattr(self, "d_f_act_0"):
            self.d_f_act_0 = self.d_f_act(self.f_act(0))
        alpha = (
                (error ** 2 * self.d_f_act(self.y)).sum()
                / self.d_f_act_0
                / (1 + (self.x ** 2).sum())
                / ((error * self.d_f_act(self.y)) ** 2).sum()
        )
        return alpha


class NeuralNetwork:
    def __init__(self, *args: Layer) -> None:
        """Создание нейросети с заданием массива слоев"""
        self.layers = args
        self.back_propagation_range = range(len(self.layers) - 1, 0, -1)

    def go(self, x: NDArray) -> NDArray:
        """Прохождение всех слоев нейросети"""
        for layer in self.layers:
            x = layer.go(x)
        return x

    def learn(
            self,
            x: NDArray[NDArray],
            e: NDArray[NDArray],
            alpha: Optional[float] = None
    ) -> NDArray:
        """Обучение наборами обучающих выборок

        - x: (n, len_in) ... [[1 2] [2 3]]
        - e: (n, len_out) ..... [[3] [4]]
        """
```

```
        square_error = np.zeros(self.layers[-1].lens[1])
        for i in range(len(e)):
            y: NDArray = self.go(x[i])
            error: NDArray = y - e[i]
            square_error += error ** 2 / 2
            for layer_i in self.back_propagation_range:
                error = self.layers[layer_i].back_propagation(error, alpha)
            self.layers[0].back_propagation(error, alpha, is_first_layer=True)
        return square_error / self.layers[-1].lens[1]

    def prediction_results_table(self, x: NDArray[NDArray], e: NDArray[NDArray]) -> None:
        """Красивый вывод прогона тестирующей выборки"""
        print(
            "                эталон        выходное значение                  разница
среднекв. ошибка"
        )
        y = self.go(x)
        y = y.reshape(-1)
        e = e.reshape(-1)
        delta = y - e
        square_error = delta ** 2 / 2
        for i in range(len(e)):
            print(f"{e[i] : 22}{y[i] : 25}{delta[i] : 25}{square_error[i] : 25}")
        print(f"\n-- Final testing square error: {np.average(square_error) * 2} --")


def save(nn: NeuralNetwork, filename=None) -> None:
    ans = input("Желаете сохранить? (y/n): ")
    if ans and (ans[0] == "y" or ans[0] == "н"):
        if filename is None:
            filename = input("Имя файла (*.nn): ") + ".nn"
        filename = "nn_files/" + filename
        if not os.path.exists("nn_files"):
            os.mkdir("nn_files")
        with open(filename, "wb") as file:
            pickle.dump(nn, file)
        print("Сохранено в", filename)
    else:
        print("Сохранение отклонено")


def load(filename=None) -> NeuralNetwork:
    if filename is None:
        filename = input("Имя файла (*.nn): ") + ".nn"
    filename = "nn_files/" + filename
    if os.path.exists(filename):
        with open(filename, "rb") as file:
            new_nn: NeuralNetwork = pickle.load(file)
        return new_nn
    else:
        raise FileNotFoundError


class LayerSigmoid(Layer):
    """Слой нейросети с сигмоидной функцией активации"""

    def __init__(self, lens: tuple[int, int]):
        """Слой нейросети с сигмоидной функцией активации"""
        super().__init__(lens, f_act=funsact.sigmoid, d_f_act=funsact.d_sigmoid)

    def back_propagation(
            self, error: NDArray,
            alpha: Optional[float],
            is_first_layer=False
    ) -> Optional[NDArray]:
        """Обратное распространение ошибки с изменением весов, порога"""
        error_later = (
            np.dot(error * self.y * (1 - self.y), self.w.transpose())
            if not is_first_layer
            else None
        )
```

```
        if not alpha:
            alpha = self.adaptive_alpha(error)

        gamma = alpha * error * self.y * (1 - self.y)
        self.w -= np.dot(self.x.reshape(-1, 1), gamma.reshape(1, -1))
        self.t += gamma

        return error_later

    def adaptive_alpha(self, error) -> float:
        alpha = (
                4
                * (error ** 2 * self.y * (1 - self.y)).sum()
                / (1 + (self.x ** 2).sum())
                / np.square(error * self.y * (1 - self.y)).sum()
        )
        return alpha


class LayerLinear(Layer):
    """Слой нейросети с линейной функцией активации"""

    def __init__(self, lens: tuple[int, int]):
        """Слой нейросети с линейной функцией активации"""
        super().__init__(lens, funsact.linear, funsact.d_linear)

    def back_propagation(
            self, error: NDArray,
            alpha: Optional[float],
            is_first_layer=False
    ) -> Optional[NDArray]:
        """Обратное распространение ошибки с изменением весов, порога"""
        error_later = np.dot(error, self.w.transpose()) if not is_first_layer else None

        if not alpha:
            alpha = self.adaptive_alpha(error)

        gamma = alpha * error
        self.w -= np.dot(self.x.reshape(-1, 1), gamma.reshape(1, -1))
        self.t += gamma

        return error_later

    def adaptive_alpha(self, error) -> float:
        alpha = 1 / (1 + np.square(self.x).sum())
        # alpha = np.square(error).sum() / (1 + np.square(self.x).sum()) / error.sum()
        return alpha


def predict_set(
        begin: float,
        lenght: float,
        count: int,
        step: float,
        function: Optional[Callable] = None
) -> tuple[NDArray[NDArray], NDArray[NDArray]]:
    """Набор обучающей выборки типа:

    x = [ [1 2 3] [2 3 4] [3 4 5] ]\n
    e = [ [4] [5] [6] ]"""

    base_array = np.arange(count + lenght)
    x = np.zeros(shape=(count, lenght))
    for i in range(count):
        x[i] = base_array[i: lenght + i]
    e = base_array[lenght: lenght + count].reshape(-1, 1)

    x = x * step + begin
    e = e * step + begin
    if function is None:
```

```
        return x, e
    else:
        return function(x), function(e)


def shuffle_set(
        x: NDArray[NDArray],
        e: NDArray[NDArray]
) -> tuple[NDArray[NDArray], NDArray[NDArray]]:
    """Перемешивание набора обучающей выборки"""
    randomize = np.arange(len(x))
    np.random.shuffle(randomize)
    x = x[randomize]
    e = e[randomize]
    return x, e
```

Структура



Автор: Крупенков Михаил (см. фотокарточку)

Однослойный персептрон.
Линейная функция активации

Цель работы:

Изучить обучение и функционирование линейной ИНС при решении задач прогнозирования. Изучить обучение и функционирование линейной ИНС с применением адаптивного шага.

Постановка задачи:

Написать на любом ЯВУ программу моделирования прогнозирующей линейной ИНС. Для тестирования использовать функцию

$$y = a\,sin(bx) + d$$

Согласно варианту 9 в таблице:

№ варианта	a	b	d	Кол-во входов ИНС
9	1	8	0.3	5

$$y = sin(8x) + 0.3$$

Без адаптивного шага

lab1u.py

```
from uninn import *


# Функция по условию (лаб.1 вар.9)
def function_lab1_9(x: float) -> float:
    return np.sin(8 * x) + 0.3


def main():
    layer = Layer(lens=(5, 1))  # Создание слоя
    nn = NeuralNetwork(layer)  # Создание нейронной сети с этим слоем
    learn_x, learn_e = predict_set(0, 5, 30, 0.1, function=function_lab1_9)  # Набор для
предсказания функции

    for t in range(10):  # Прогон набора _ раз
        square_error = nn.learn(learn_x, learn_e, 0.08)  # Метод для обучения
        print(f"Average square error {t : 3}: {square_error}")  # Вывод на каждой итерации средней
ошибки

    test_x, test_e = predict_set(3, 5, 15, 0.1, function=function_lab1_9)  # Набор для тестирования
    nn.prediction_results_table(test_x, test_e)  # Метод для красивого вывода


if __name__ == "__main__":
    main()
```

Результаты программы

```
Average square error   0: [6.10399331]
Average square error   1: [0.09098079]
Average square error   2: [0.00181354]
Average square error   3: [3.63224762e-05]
Average square error   4: [7.27384184e-07]
Average square error   5: [1.45660164e-08]
Average square error   6: [2.91686782e-10]
Average square error   7: [5.84107308e-12]
Average square error   8: [1.16968394e-13]
Average square error   9: [2.34231026e-15]
```

эталон	выходное значение	разница	среднекв. ошибка
0.570905788307869	0.5709057879615469	-3.4632208212315163e-10	5.996949228305749e-20
-0.20178930102057407	-0.20178929901011022	2.0104638509543804e-09	2.0209824479971586e-18
-0.6701057337071854	-0.6701057313847308	2.3224545619981996e-09	2.6968975962731245e-18
-0.549969045879328	-0.5499690454789256	4.00402488942575e-10	8.016107657570446e-20
0.0857474597041123	0.08574745711430998	-2.5898023170434215e-09	3.3535380206817372e-18
0.8514266812416906	0.8514266764073488	-4.834341815751486e-09	1.1685430395761688e-17
1.2826178773641395	1.2826178723924326	-4.971706824008493e-09	1.2358934371946307e-17
1.1177662545264417	1.117766251607867	-2.9185747152382646e-09	4.259039184214059e-18
0.45686859504841	0.45686859512805966	7.964967574380921e-11	3.1720354230469748e-21
-0.29918344921426526	-0.29918344700997357	2.2042916936193535e-09	2.429450935279639e-18
-0.6917788534431157	-0.6917788512765545	2.1665612637278286e-09	2.346993854742963e-18
-0.4827745135506544	-0.4827745135613003	-1.0645873071979395e-11	5.6667306732348e-23
0.20105034244971043	0.20105033944303807	-3.006672355532203e-09	4.5200393267607825e-18
0.9448967329448674	0.944896727940707	-5.0041604193751255e-09	1.2520810751420316e-17
1.2975574189078052	1.2975574141163364	-4.791468777298746e-09	1.1479086521914372e-17

```
-- Final testing square error: 9.30860835714409e-18 --
```

С адаптивным шагом

Для того, чтобы использовать адаптивный шаг, нужно не вводить его в метод learn или вводить None.

lab2u.py

```python
from uninn import *
from lab1u import function_lab1_9


def main():
    layer = LayerLinear(lens=(5, 1))
    nn = NeuralNetwork(layer)

    learn_x, learn_e = predict_set(0, 5, 30, 0.1, function=function_lab1_9)
    for t in range(10):
        square_error = nn.learn(learn_x, learn_e)
        print(f"Average square error {t : 3}: {square_error}")

    test_x, test_e = predict_set(3, 5, 15, 0.1, function=function_lab1_9)
    nn.prediction_results_table(test_x, test_e)


if __name__ == "__main__":
    main()
```

Результаты программы

```
Average square error   0: [1.55516573]
Average square error   1: [0.002445]
Average square error   2: [3.50849982e-07]
Average square error   3: [5.56047595e-11]
Average square error   4: [1.06821082e-14]
Average square error   5: [2.78236e-18]
Average square error   6: [1.02133562e-21]
Average square error   7: [4.88539043e-25]
Average square error   8: [2.76642118e-28]
Average square error   9: [1.54552026e-29]
```

эталон	выходное значение	разница	среднекв. ошибка
0.570905788307869	0.5709057883078669	-2.1094237467877974e-15	2.224834271756135e-30
-0.20178930102057407	-0.20178930102057557	-1.498010832439613e-15	1.123202343566636e-30
-0.6701057337071854	-0.6701057337071866	-1.2212453270876722e-15	7.457200744667377e-31
-0.549969045879328	-0.5499690458793286	-5.551115123125783e-16	1.5407439555097887e-31
0.0857474597041123	0.08574745970411224	-5.551115123125783e-17	1.5407439555097887e-33
0.8514266812416906	0.8514266812416901	-5.551115123125783e-16	1.5407439555097887e-31
1.2826178773641395	1.2826178773641397	2.220446049250313e-16	2.465190328815662e-32
1.1177662545264417	1.1177662545264417	0.0	0.0
0.45686859504841	0.456868595048407	-2.9976021664879227e-15	4.492809374266544e-30
-0.29918344921426526	-0.2991834492142639	1.3877787807814457e-15	9.62964972193618e-31
-0.6917788534431157	-0.6917788534431176	-1.887379141862766e-15	1.7811000125693157e-30
-0.4827745135506544	-0.4827745135506537	7.216449660063518e-16	2.603857284811543e-31
0.20105034244971043	0.2010503424497077	-2.7200464103316335e-15	3.6993262371790026e-30
0.9448967329448674	0.9448967329448696	2.220446049250313e-15	2.465190328815662e-30
1.2975574189078052	1.2975574189078043	-8.881784197001252e-16	3.944304526105059e-31

```
-- Final testing square error: 2.464574031233458e-30 --
```

Благодаря адаптивному шагу нейронная сеть достигает лучших результатов за одинаковое количество повторений.

Нелинейные ИНС в задачах прогнозирования

Цель работы:

Изучить обучение и функционирование нелинейной ИНС при решении задач прогнозирования

Постановка задачи:

Написать на любом ЯВУ программу моделирования прогнозирующей нелинейной ИНС. Для тестирования использовать функцию

$y = a\,cos(\,bx) + c\,sin(\,dx)$.

Варианты заданий приведены в следующей таблице:

№ варианта	a	b	c	d	Кол-во входов	Кол-во НЭ в скрытом слое
9	0.1	0.3	0.08	0.3	10	4

$y = 0.1\,cos(\,0.3x) + 0.08\,sin(\,0.3x)$.

Для прогнозирования использовать многослойную ИНС с одним скрытым слоем. В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного - линейную.

Без адаптивного шага

lab3u.py

```
from uninn import *
import time


def function_lab3_9(x):
    return 0.1 * np.cos(0.3 * x) + 0.08 * np.sin(0.3 * x)


def main():
    nn = NeuralNetwork(
        Layer(lens=(10, 4), f_act=funsact.sigmoid, d_f_act=funsact.d_sigmoid),
        Layer(lens=(4, 1))
    )

    learn_x, learn_e = predict_set(0, 10, 30, 0.1, function=function_lab3_9)
    test_x, test_e = predict_set(3, 10, 15, 0.1, function=function_lab3_9)
    alpha = 0.05
    times = 30
    sep = 1000
    print(f"- Learning {times * sep} times -")

    start_time = time.time()

    for thousand in range(times):
        for _ in range(sep - 1):
            nn.learn(learn_x, learn_e, alpha)
        error = nn.learn(learn_x, learn_e, alpha)[0]
```

```
            print(f"{thousand + 1 : 5d}x{sep} error: {error : .5e}")

        print(f"- Learning time: {time.time() - start_time} seconds -")

        nn.prediction_results_table(test_x, test_e)


if __name__ == "__main__":
    main()
```

Результаты программы

```
- Learning 30000 times -
    1x1000 error:  1.73608e-04
    2x1000 error:  1.72640e-04
    3x1000 error:  1.71680e-04
    4x1000 error:  1.70726e-04
    5x1000 error:  1.69778e-04
    6x1000 error:  1.68834e-04
    7x1000 error:  1.67892e-04
    8x1000 error:  1.66952e-04
    9x1000 error:  1.66012e-04
   10x1000 error:  1.65072e-04
   11x1000 error:  1.64130e-04
   12x1000 error:  1.63186e-04
   13x1000 error:  1.62239e-04
   14x1000 error:  1.61288e-04
   15x1000 error:  1.60333e-04
   16x1000 error:  1.59374e-04
   17x1000 error:  1.58409e-04
   18x1000 error:  1.57439e-04
   19x1000 error:  1.56464e-04
   20x1000 error:  1.55481e-04
   21x1000 error:  1.54493e-04
   22x1000 error:  1.53497e-04
   23x1000 error:  1.52495e-04
   24x1000 error:  1.51485e-04
   25x1000 error:  1.50468e-04
   26x1000 error:  1.49443e-04
   27x1000 error:  1.48410e-04
   28x1000 error:  1.47370e-04
   29x1000 error:  1.46322e-04
   30x1000 error:  1.45267e-04
- Learning time: 25.220435619354248 seconds -
```

эталон	выходное значение	разница	среднекв. ошибка
0.11079890232504547	0.11811872177725319	0.007319819452207721	2.678987840645927e-05
0.10882287686698608	0.117481914692635	0.008659037825648919	3.7489468033009375e-05
0.1067489181650702	0.11680683597927152	0.010057917814201317	5.058085537861 41e-05
0.10457889264214158	0.11609404920177163	0.01151515655963005	6.629941529639548e-05
0.10231475317469851	0.11534415102885515	0.013029397854156632	8.488260422095073e-05
0.0999585373354369	0.11455777095953798	0.01459923362410108	0.00010656881120554177
0.09751236555957221	0.11373557102235421	0.016223205462781998	0.00013159619774381982
0.09497843923659116	0.11287824544613592	0.01789980620954476	0.00016020153116962858
0.09235903872914948	0.11198652030088907	0.01962748157173959	0.00019261901642448858
0.08965652132089966	0.11106115310733791	0.021404631786438255	0.00022907913095650145
0.08687331909509466	0.11010293241375135	0.023229613318656694	0.00026980746746715624
0.0840119367458772	0.1091126773387282	0.025100740592851	0.000315023589154799
0.08107494932422404	0.10809123707868865	0.027016287754464602	0.000364939902016017
0.078064999920574	0.10703949037890334	0.02897449045832934	0.00041976054865990893
0.07498479728622486	0.10595834496699472	0.03097354768076986	0.000479680327966462

```
 -- Final testing square error: 0.00039137583254663364 --
```

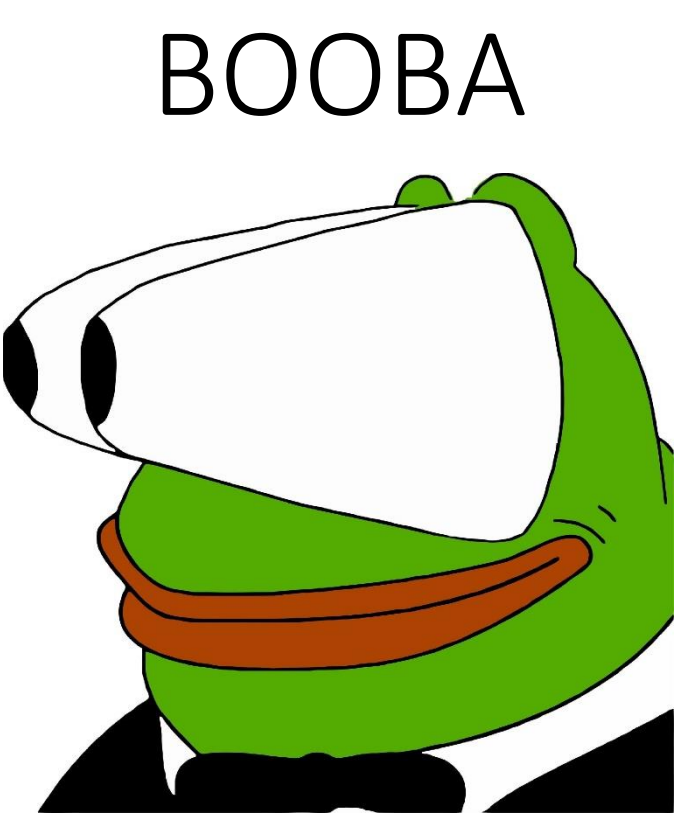
С адаптивным шагом

lab4u.py

```python
from uninn import *
import time
from lab3u import function_lab3_9


def main():
    nn = NeuralNetwork(
        LayerSigmoid(lens=(10, 4)),
        LayerLinear(lens=(4, 1))
    )

    learn_x, learn_e = predict_set(0, 10, 30, 0.1, function=function_lab3_9)
    test_x, test_e = predict_set(3, 10, 15, 0.1, function=function_lab3_9)
    times = 30
    sep = 1000
    print(f"- Learning {times * sep} times -")

    start_time = time.time()

    for thousand in range(times):
        for _ in range(sep - 1):
            nn.learn(learn_x, learn_e)
        error = nn.learn(learn_x, learn_e)[0]
        print(f"{thousand + 1 : 5d}x{sep} error: {error : .5e}")

    print(f"- Learning time: {time.time() - start_time} seconds -")

    nn.prediction_results_table(test_x, test_e)


if __name__ == "__main__":
    main()
```

Результаты программы

```
- Learning 30000 times -
    1x1000 error:  1.69581e-05
    2x1000 error:  6.84900e-05
    3x1000 error:  1.28096e-04
    4x1000 error:  1.42925e-04
    5x1000 error:  1.33529e-04
    6x1000 error:  1.20412e-04
    7x1000 error:  1.08578e-04
    8x1000 error:  9.80671e-05
    9x1000 error:  8.82054e-05
   10x1000 error:  7.84643e-05
   11x1000 error:  6.85393e-05
   12x1000 error:  5.83208e-05
   13x1000 error:  4.78774e-05
   14x1000 error:  3.74564e-05
   15x1000 error:  2.74780e-05
   16x1000 error:  1.84914e-05
   17x1000 error:  1.10643e-05
   18x1000 error:  5.60758e-06
   19x1000 error:  2.20395e-06
   20x1000 error:  5.52853e-07
   21x1000 error:  9.11510e-08
   22x1000 error:  2.18113e-07
   23x1000 error:  4.82484e-07
   24x1000 error:  6.47858e-07
   25x1000 error:  6.54252e-07
   26x1000 error:  5.40039e-07
   27x1000 error:  3.73946e-07
   28x1000 error:  2.14964e-07
   29x1000 error:  9.71662e-08
   30x1000 error:  3.00148e-08
- Learning time: 42.87540006637573 seconds -
```



эталон	выходное значение	разница	среднекв. ошибка
0.11079890232504547	0.11080894773317745	1.0045408131986733e-05	5.045511226909259e-11
0.10882287686698608	0.10884484035596881	2.196348898272915e-05	2.4119742414723243e-10
0.1067489181650702	0.10677869227040565	2.9774105335445e-05	4.432486742630872e-10
0.10457889264214158	0.10461182030069155	3.29276585499666e-05	5.421153487915942e-10
0.10231475317469851	0.1023456426313547	3.088945665619125e-05	4.77079266257359e-10
0.0999585373354369	0.09998168249645362	2.3145161016716043e-05	2.67849239244856e-10
0.09751236555957221	0.09752157172672454	9.206167152331357e-06	4.2376756818332424e-11
0.09497843923659116	0.09496705410924072	-1.1385127350438395e-05	6.48105623928502e-11
0.09235903872914948	0.0923199885135762	-3.9050215573280256e-05	7.624596681598299e-10
0.08965652132089966	0.08958235173827395	-7.41695826257105e-05	2.750563493436048e-09
0.08687331909509466	0.08675624103167467	-0.00011707806341998417	6.853636467086917e-09
0.0840119367458772	0.08384387624186446	-0.00016806050401274586	1.4122166504509083e-08
0.08107494932422404	0.08084760155171322	-0.00022734777251082194	2.5843504832816224e-08
0.078064999920574	0.07776988675670377	-0.00029511316387023645	4.3545889744750515e-08
0.07498479728622486	0.07461332804551968	-0.0003714692407051773	6.899469839504048e-08

```
-- Final testing square error: 2.20002735319978e-08 --
```

Нелинейные ИНС в задачах распознавания образов

Цель работы

Изучить обучение и функционирование нелинейной ИНС при решении задач распознавания образов.

Постановка задачи

Написать на любом ЯВУ программу моделирования нелинейной ИНС для распознавания образов. Рекомендуется использовать сигмоидную функцию, но это не является обязательным. Количество НЭ в скрытом слое взять согласно варианту работы №3. Его можно варьировать, если сеть не обучается или некорректно функционирует.

Провести исследование полученной модели. При этом на вход сети необходимо подавать искаженные образы, в которых инвертированы некоторые биты. Критерий эффективности процесса распознавания - максимальное кодовое расстояние (количество искаженных битов) между исходным и поданным образом.

Вариант	Вектор 1	Вектор 2	Вектор 3
9	2	4	8

№	Данные вектора																			
2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
4	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
8	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1

lab5u.py

```
from uninn import *
import time


def switch_bit(row: np.ndarray, j) -> np.ndarray:
    row[j] ^= 1
    return row


def main():
    nn = NeuralNetwork(
        LayerSigmoid(lens=(20, 4)),
        LayerLinear(lens=(4, 3))
    )

    learn_x = np.array(
        [
            [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0],
            [1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0],
```

```
            [1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1]
        ]
    )
    learn_e = np.eye(3)
    times = 2
    print(f"- Learning {times} times -")

    start_time = time.time()

    for t in range(times):
        nn.learn(learn_x, learn_e)
        error = nn.learn(learn_x, learn_e).sum()
        print(f"{t + 1 : 5d}/{times} error: {error : .5e}")
    print(f"- Learning time: {time.time() - start_time} seconds -")

    print("\nГлубокая проверка с заменой битов №\n[i]: - | 0 1 2 3 ...")
    correct_amount = 0
    for i in range(3):
        row = learn_x[i]
        out = (nn.go(row)).argmax()
        print(f"[{i}]: {out} | ", end="")
        if out == i:
            correct_amount += 1

        for j in range(20):
            out = (nn.go(switch_bit(row, j))).argmax()
            if out == i:
                correct_amount += 1
            print(out, end=" ")
        print()
    print(f"Правильно {correct_amount} / 63: {correct_amount / 0.63 : .1f}%")


if __name__ == "__main__":
    main()
```

Результаты программы

```
- Learning 3 times -
    1/3 error:  9.69309e-01
    2/3 error:  6.67125e-01
    3/3 error:  2.41185e-01
- Learning time: 0.0009953975677490234 seconds -

Глубокая проверка с заменой битов №
[i]: - | 0 1 2 3 ...
[0]: 0 | 0 0 0 0 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1
[1]: 1 | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 2 2 2 2
[2]: 2 | 2 2 2 2 2 2 2 2 0 1 1 1 1 1 1 1 1 1 1 1
Правильно 29 / 63:  46.0%
```

```
- Learning 2 times -
    1/2 error:  6.12954e-01
    2/2 error:  3.24473e-01
- Learning time: 0.0009703636169433594 seconds -

Глубокая проверка с заменой битов №
[i]: - | 0 1 2 3 ...
[0]: 0 | 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2
[1]: 1 | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 2 2
[2]: 2 | 2 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1
Правильно 32 / 63:  50.8%
```

Вывод

Я преисполнился в своих познаниях о нейронных сетях.