Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №5

По дисциплине: «МиАПР»

Тема: «Нелинейные ИНС в задачах распознавания образов.»

Выполнил:

Студент 2 курса

Группы ПО-7(2)

Панкратов Р.С.

Проверил:

Крощенко А.А.

2021

Вариант 3

**Цель работы:**

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

**Задание:**

1. Изучить теоретические сведения, приведенные в данных методических указания и методических указаниях для работ №3 и №4.

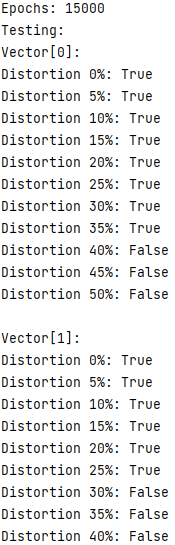
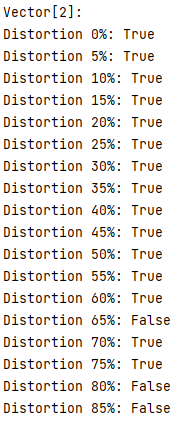
2. Написать на любом ЯВУ программу моделирования нелинейной ИНС для распознавания образов. Рекомендуется использовать сигмоидную функцию, но это не является обязательным. Количество НЭ в скрытом слое взять согласно варианту работы №3. Его можно варьировать, если сеть не обучается или некорректно функционирует.

3. 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.

**Код программы:**

from typing import Callable  
  
import numpy as np  
from numpy.typing import NDArray  
  
LEARNING\_SPEED = 0.1  
  
WeightsType = NDArray[NDArray[np.float64]]  
TType = NDArray[np.float64]  
  
  
def sigmoid(s):  
 return 1 / (1 + np.exp(-s))  
  
  
def d\_sigmoid(y):  
 return y \* (1 - y)  
  
  
def lin(s):  
 return s  
  
  
def d\_lin(s):  
 return np.full(shape=s.shape, fill\_value=1.0, dtype=np.float64)  
  
  
def calc\_error(  
 outputs: NDArray[NDArray[np.float64]], ideal\_outputs: NDArray[NDArray[np.float64]]  
) -> NDArray[np.float64]:  
 errors = (outputs - ideal\_outputs) \*\* 2  
 return errors.sum(axis=0) / len(ideal\_outputs)  
  
  
def training(  
 inputs: NDArray[np.float64],  
 outputs: NDArray[np.float64],  
 w\_output: WeightsType,  
 w\_hidden: WeightsType,  
 T\_output: TType,  
 T\_hidden: TType  
) -> None:  
 hidden\_outputs = sigmoid(np.dot(w\_hidden, inputs) - T\_hidden)  
 output\_outputs = sigmoid(np.dot(w\_output, hidden\_outputs) - T\_output)  
  
 error\_output = (output\_outputs - outputs).reshape(1, -1)  
 error\_hidden = (error\_output \* w\_output.T \* d\_sigmoid(output\_outputs)).sum(axis=1).reshape(1, -1)  
  
 learning\_speed\_output = LEARNING\_SPEED  
 learning\_speed\_hidden = LEARNING\_SPEED  
  
 w\_output -= learning\_speed\_output \* error\_output.reshape(-1, 1) \* hidden\_outputs \* d\_sigmoid(output\_outputs).reshape(-1, 1)  
 T\_output += (learning\_speed\_output \* error\_output \* d\_sigmoid(output\_outputs)).reshape(-1)  
  
 w\_hidden -= learning\_speed\_hidden \* error\_hidden.reshape(-1, 1) \* inputs.reshape(1, -1) \* d\_sigmoid(hidden\_outputs).reshape(-1, 1)  
 T\_hidden += learning\_speed\_hidden \* error\_hidden.reshape(-1) \* d\_sigmoid(hidden\_outputs)  
  
  
def learn(  
 inputs: NDArray[NDArray[np.float64]],  
 ideal\_outputs: NDArray[NDArray[np.float64]],  
 w\_output: WeightsType,  
 w\_hidden: WeightsType,  
 T\_output: TType,  
 T\_hidden: TType,  
 epochs: int = 1000,  
 is\_print\_intermediate\_result: bool = True,  
):  
 for epoch in range(epochs + 1):  
 for r\_i in np.random.choice(3, 3, replace=False):  
 training(inputs[r\_i], ideal\_outputs[r\_i], w\_output, w\_hidden, T\_output, T\_hidden)  
  
 if is\_print\_intermediate\_result and not epoch % (epochs // 10):  
 outputs = v\_predict(inputs, w\_hidden, w\_output, T\_hidden, T\_output)  
 print(f"Epoch: {epoch}")  
 for i, (output, ideal\_output) in enumerate(zip(outputs, ideal\_outputs)):  
 max\_index = output.argmax()  
 print(f"Vector {i + 1}: {ideal\_output[max\_index] == 1}")  
  
 return w\_output, w\_hidden, T\_output, T\_hidden  
  
  
def predict(  
 inputs: NDArray[np.float64], w\_hidden: WeightsType, w\_output: WeightsType, T\_hidden: TType, T\_output: TType  
) -> NDArray[np.float64]:  
 hidden\_outputs = sigmoid(np.dot(w\_hidden, inputs) - T\_hidden)  
 return sigmoid(np.dot(w\_output, hidden\_outputs) - T\_output)  
  
  
def switch\_bit(arr: NDArray[np.int64], index: int) -> NDArray[np.int64]:  
 arr[index] ^= 1  
 return arr  
  
  
VPredictType = Callable[  
 [NDArray[NDArray[np.float64]], WeightsType, WeightsType, TType, TType], NDArray[NDArray[np.float64]]  
]  
v\_predict: VPredictType = np.vectorize(predict, signature="(n)->(m)", excluded=[1, 2, 3, 4])  
  
  
def main():  
 EPOCHS = 15\_000  
  
 nn\_inputs = 20  
 nn\_hidden = 4  
 nn\_output = 3  
  
 dataset = np.array([  
 [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0],  
 [1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1],  
 [1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1]  
 ])  
 ideal\_outputs = np.array([  
 [1, 0, 0],  
 [0, 1, 0],  
 [0, 0, 1]  
 ])  
  
 w\_hidden: WeightsType = np.random.normal(-1, 1, (nn\_hidden, nn\_inputs))  
 w\_output: WeightsType = np.random.normal(-1, 1, (nn\_output, nn\_hidden))  
 T\_hidden: TType = np.random.normal(-1, 1, nn\_hidden)  
 T\_output: TType = np.random.normal(-1, 1, nn\_output)  
  
 w\_output, w\_hidden, T\_output, T\_hidden = learn(  
 dataset, ideal\_outputs, w\_output.copy(), w\_hidden.copy(), T\_output.copy(), T\_hidden.copy(), EPOCHS, False  
 )  
  
 print(f"Epochs: {EPOCHS}")  
 print("Testing:")  
 for i, ideal\_output in enumerate(ideal\_outputs):  
 print(f"Vector[{i}]:")  
 dataset\_input = dataset[i]  
 output = predict(dataset\_input, w\_hidden, w\_output, T\_hidden, T\_output)  
 result = ideal\_output[output.argmax()] == 1  
  
 print(f"Distortion 0%: {result}")  
  
 distortion\_dataset\_input = dataset\_input.copy()  
 num\_incorrect\_results = 0  
 for j\_i, j in enumerate(np.random.choice(20, 20, replace=False)):  
 switch\_bit(distortion\_dataset\_input, j)  
 output = predict(distortion\_dataset\_input, w\_hidden, w\_output, T\_hidden, T\_output)  
 result = ideal\_output[output.argmax()] == 1  
 print(f"Distortion {(j\_i + 1) \* 5}%: {result}")  
  
 if not result:  
 num\_incorrect\_results += 1  
 if num\_incorrect\_results >= 3:  
 break  
  
 print()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Результат:**

**Вывод:**

Я изучил обучение и функционирование нелинейной ИНС при решении задач распознавания образов.