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
```

```python
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:

```
Epochs: 15000
Testing:
Vector[0]:
Distortion 0%: True
Distortion 5%: True
Distortion 10%: True
Distortion 15%: True
Distortion 20%: True
Distortion 25%: True
Distortion 30%: True
Distortion 35%: True
Distortion 40%: False
Distortion 45%: False
Distortion 50%: False

Vector[1]:
Distortion 0%: True
Distortion 5%: True
Distortion 10%: True
Distortion 15%: True
Distortion 20%: True
Distortion 25%: True
Distortion 30%: False
Distortion 35%: False
Distortion 40%: False
```

```
Vector[2]:
Distortion 0%: True
Distortion 5%: True
Distortion 10%: True
Distortion 15%: True
Distortion 20%: True
Distortion 25%: True
Distortion 30%: True
Distortion 35%: True
Distortion 40%: True
Distortion 45%: True
Distortion 50%: True
Distortion 55%: True
Distortion 60%: True
Distortion 65%: False
Distortion 70%: True
Distortion 75%: True
Distortion 80%: False
Distortion 85%: False
```

Вывод:

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