Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення
Дисципліна: Алгоритми та структури даних

Лабораторна робота №6

Тема: «ШТУЧНІ НЕЙРОННІ МЕРЕЖІ. МОДЕЛЮВАННЯ ФОРМАЛЬНИХ ЛОГІЧНИХ ФУНКЦІЙ. ПРОГНОЗУВАННЯ ЧАСОВИХ РЯДІВ»

Виконав: ст. гр. КН-24
Куріщенко П. В.
Перевірив: викладач
Константинова Л. В.

Кропивницький 2026

Мета: Отримати початкові навички по створенню штучних нейронних мереж, що здатні виконувати прості логічні функції, та нейронних мереж, що здатні прогнозувати часові ряди.

Завдання: Написати програму для реалізації штучних нейронів та нейронних мереж для: - моделювання логічної функції І, - моделювання логічної функції АБО, - моделювання логічної функції НІ, - моделювання логічної функції Виключне АБО, - прогнозування часового ряду (1.88, 4.52 1.91, 5.66, 1.23, 5.50, 1.14, 5.29, 1.60, 4.31, 0.06, 5.33, 0.07, 4.62, 0.69).

Додаткове завдання: Написати програму для реалізації штучної нейронної мережі, що моделює логічну функцію, таблиця істинності якої наводиться в таблиці 7.

Таблиця 7 – Таблиця істинності логічної функції

x_1	x_2	x_2	y
0	0	0	1
0	1	0	1
1	0	0	0
1	1	1	1

Повний лістинг програми міститься в git репозиторії:
https://github.com/movavok/Algorithms-labs/tree/main/lab_6

Лістинг *perceptron.h*:

```
#ifndef PERCEPTRON_H
#define PERCEPTRON_H

#include <vector>
#include <iostream>

using namespace std;

class Perceptron {
private:
    vector<double> m_weights;
    double m_learningRate;
    double m_threshold;

public:
    Perceptron(int inputsAmount, double learnRate);
```

```
    void setWeights(const vector<double>& weights) { m_weights = weights; }
    void setThreshold(double threshold) { m_threshold = threshold; }

    void modelAND();
    void modelOR();
    void modelNOT();

    int predict(const vector<int>& inputs);
    void train(const vector<vector<int>>& inputs, const vector<int>& targets,
int epochs = 100);
    void test(const vector<vector<int>>& inputs, const vector<int>& targets);
};

#endif // PERCEPTRON_H
```

Лістинг *perceptron.cpp*:

```
#include "perceptron.h"

Perceptron::Perceptron(int inputsAmount, double learnRate)
    : m_learningRate(learnRate), m_threshold(0.5)
{
    m_weights.resize(inputsAmount, 1.0);
}

int Perceptron::predict(const vector<int>& inputs) {
    double sum = 0.0;
    for (size_t index = 0; index < inputs.size(); index++)
        sum += inputs[index] * m_weights[index];
    return sum >= m_threshold ? 1 : 0;
}

void Perceptron::train(const vector<vector<int>>& inputs, const vector<int>&
targets, int epochs) {
    for (int epoch = 0; epoch < epochs; epoch++)
        for (size_t inputIdx = 0; inputIdx < inputs.size(); inputIdx++) {
            int resultPred = predict(inputs[inputIdx]);
            int error = targets[inputIdx] - resultPred;
            for (size_t weightIdx = 0; weightIdx < m_weights.size();
weightIdx++)
                m_weights[weightIdx] += m_learningRate * error *
inputs[inputIdx][weightIdx];
        }
}

void Perceptron::test(const vector<vector<int>>& inputs, const vector<int>&
targets) {
    for (size_t index = 0; index < inputs.size(); index++) {
        int output = predict(inputs[index]);
        cout << "Вхід: ";
        for (int value : inputs[index]) cout << value << " ";
        cout << "=> Вихід: " << output << " (Очікуваний: " << targets[index]
<< ")";

        if (output == targets[index])
            cout << " ВІРНО";
        else
            cout << " НЕПРАВИЛЬНО";
```

```
            cout << endl;
        }
        cout << endl;
}

void Perceptron::modelAND() {
    m_weights = {1, 1};
    m_threshold = 1.5;

    cout << "Моделювання логічної функції І:" << endl;
    test({{0,0}, {0,1}, {1,0}, {1,1}}, {0,0,0,1});
}

void Perceptron::modelOR() {
    m_weights = {1, 1};
    m_threshold = 0.5;

    cout << "Моделювання логічної функції АБО:" << endl;
    test({{0,0}, {0,1}, {1,0}, {1,1}}, {0,1,1,1});
}

void Perceptron::modelNOT() {
    m_weights = {-1};
    m_threshold = -0.5;

    cout << "Моделювання логічної функції НЕ:" << endl;
    test({{0}, {1}}, {1,0});
}
```

Лістинг *neuronnet.h*:

```
#ifndef NEURONNET_H
#define NEURONNET_H

#include <ctime>

#include "perceptron.h"

class NeuronNet {
public:
    NeuronNet(int inputsAmount);

    void modelXOR();

    void trainAndTestTS(vector<double>& series, int epochs = 50000);


private:
    vector<double> m_weights;
    double m_learningRate;

    // XOR members/helper
    Perceptron pNAND;
    Perceptron pOR;
    Perceptron pAND;
    void initXOR();

    // Time series helper
    double sigmoid(double x) { return 1.0 / (1.0 + exp(-x)) * 10; }
```

```
    double predictTimeSeries(const vector<double>& input);
    void trainTimeSeries(const vector<double>& series, int epochs, size_t
trainCount = 13);
    void testTimeSeries(const vector<double>& series, size_t trainCount =
13);
};

#endif // NEURONNET_H
```

Лістинг *neuronnet.cpp*:

```
#include "neuronnet.h"

NeuronNet::NeuronNet(int inputsAmount)
    : m_learningRate(0.004), pNAND(2, 0.1), pOR(2, 0.1), pAND(2, 0.1)
{
    m_weights.resize(inputsAmount, 1.0);
    srand(time(0));
}

void NeuronNet::initXOR() {
    pNAND.setWeights({-1, -1});
    pNAND.setThreshold(-1.5);

    pOR.setWeights({1, 1});
    pOR.setThreshold(0.5);

    pAND.setWeights({1, 1});
    pAND.setThreshold(2);
}

void NeuronNet::modelXOR() {
    initXOR();

    vector<vector<int>> inputs = {{0,0},{0,1},{1,0},{1,1}};
    vector<int> targets = {0,1,1,0};

    cout << "Моделювання логічної функції Виключне АБО:" << endl;

    for (size_t index = 0; index < inputs.size(); index++) {
        int firstResult = pNAND.predict(inputs[index]);
        int secondResult = pOR.predict(inputs[index]);
        int finalResult = pAND.predict({firstResult, secondResult});

        cout << "Вхід: " << inputs[index][0] << " " << inputs[index][1]
             << " => Вихід: " << finalResult << " (Очікуваний: " <<
targets[index] << ")";

        if (finalResult == targets[index])
            cout << " ВІРНО";
        else
            cout << " НЕПРАВИЛЬНО";

        cout << endl;
    }
    cout << endl;
}

double NeuronNet::predictTimeSeries(const vector<double>& input) {
```

```
    double result = input[0] * m_weights[0] + input[1] * m_weights[1] +
input[2] * m_weights[2];
    return sigmoid(result);
}

void NeuronNet::trainTimeSeries(const vector<double>& series, int epochs,
size_t trainCount) {
    for (double &weigth : m_weights)
        weigth = static_cast<double>(rand()) / RAND_MAX;

    for (int epoch = 0; epoch < epochs; epoch++) {
        for (size_t inputIdx = 0; inputIdx + 3 < trainCount; inputIdx++) {
            vector<double> input = { series[inputIdx], series[inputIdx+1],
series[inputIdx+2] };
            double predicted = predictTimeSeries(input);
            double error = predicted - series[inputIdx+3];

            for (size_t weightIdx = 0; weightIdx < m_weights.size();
weightIdx++)
                m_weights[weightIdx] -= m_learningRate * error *
input[weightIdx];
        }
    }
}

void NeuronNet::testTimeSeries(const vector<double>& series, size_t
trainCount) {
    cout << "Прогнозування часового ряду:" << endl;
    for (size_t index = trainCount - 3; index + 3 < series.size(); index++) {
        vector<double> input = { series[index], series[index+1],
series[index+2] };
        double predict = predictTimeSeries(input);
        double expected = series[index+3];

        double accuracy = 100.0 * (1.0 - fabs(predict - expected) /
expected);
        if (accuracy < 0) accuracy = 0;

        cout << "Вхід: " << input[0] << ", " << input[1] << ", " << input[2]
             << " => Вихід: " << predict << " (Очікуваний: " << expected <<
")"
             << " Точність: " << accuracy << "%" << endl;
    }
    cout << endl;
}

void NeuronNet::trainAndTestTS(vector<double>& series, int epochs) {
    trainTimeSeries(series, epochs);
    testTimeSeries(series);
}
```

Лістинг *main.cpp*:

```
#include "neuronnet.h"

int main() {
    Perceptron neuron(3, 0.1);
    neuron.modelAND();
    neuron.modelOR();
```

```
    neuron.modelNOT();

    NeuronNet net(3);
    net.modelXOR();

    vector<double> series = {
        1.88, 4.52, 1.91, 5.66, 1.23, 5.50, 1.14, 5.29,
        1.60, 4.31, 0.06, 5.33, 0.07, 4.62, 0.69
    };

    net.trainAndTestTS(series, 50000);

    vector<vector<int>> inputs = {
        {0, 0, 0},
        {0, 1, 0},
        {1, 0, 0},
        {1, 1, 1}
    };

    vector<int> targets = {1, 1, 0, 1};

    cout << "Додаткове завдання: " << endl;
    neuron.train(inputs, targets);
    neuron.test(inputs, targets);

    return 0;
}
```

Результат виконання:

```
Моделювання логічної функції І:
Вхід: 0 0 => Вихід: 0 (Очікуваний: 0) ВІРНО
Вхід: 0 1 => Вихід: 0 (Очікуваний: 0) ВІРНО
Вхід: 1 0 => Вихід: 0 (Очікуваний: 0) ВІРНО
Вхід: 1 1 => Вихід: 1 (Очікуваний: 1) ВІРНО

Моделювання логічної функції АБО:
Вхід: 0 0 => Вихід: 0 (Очікуваний: 0) ВІРНО
Вхід: 0 1 => Вихід: 1 (Очікуваний: 1) ВІРНО
Вхід: 1 0 => Вихід: 1 (Очікуваний: 1) ВІРНО
Вхід: 1 1 => Вихід: 1 (Очікуваний: 1) ВІРНО

Моделювання логічної функції НЕ:
Вхід: 0 => Вихід: 1 (Очікуваний: 1) ВІРНО
Вхід: 1 => Вихід: 0 (Очікуваний: 0) ВІРНО

Моделювання логічної функції Виключне АБО:
Вхід: 0 0 => Вихід: 0 (Очікуваний: 0) ВІРНО
Вхід: 0 1 => Вихід: 1 (Очікуваний: 1) ВІРНО
Вхід: 1 0 => Вихід: 1 (Очікуваний: 1) ВІРНО
Вхід: 1 1 => Вихід: 0 (Очікуваний: 0) ВІРНО

Прогнозування часового ряду:
Вхід: 0.06, 5.33, 0.07 => Вихід: 6.46526 (Очікуваний: 4.62) Точність: 60.0593%
Вхід: 5.33, 0.07, 4.62 => Вихід: 0.689807 (Очікуваний: 0.69) Точність: 99.972%

Додаткове завдання:
Вхід: 0 0 0 => Вихід: 1 (Очікуваний: 1) ВІРНО
Вхід: 0 1 0 => Вихід: 1 (Очікуваний: 1) ВІРНО
Вхід: 1 0 0 => Вихід: 0 (Очікуваний: 0) ВІРНО
Вхід: 1 1 1 => Вихід: 1 (Очікуваний: 1) ВІРНО
```

Висновки: У ході виконання лабораторної роботи №6 було отримано практичні навички створення та навчання штучних нейронних мереж для моделювання простих логічних функцій та прогнозування часових рядів.

1. **Моделювання логічних функцій**
   - Було реалізовано перцептрон, який успішно моделював функції І (AND), АБО (OR), НЕ (NOT) та Виключне АБО (XOR).
   - Результати моделювання показали, що вихідні значення мережі повністю відповідають очікуваним результатам із таблиць істинності, що свідчить про правильне налаштування ваг та порогів нейронів.
   - Для функції XOR, яка є нелінійно роздільною, було застосовано багатошарову мережу (з прихованим шаром), що дозволило досягти коректного моделювання.
2. **Прогнозування часового ряду**
   - Була реалізована мережа з трьома входами для прогнозування наступного значення ряду на основі трьох попередніх.
   - Мережа показала різний рівень точності для різних точок ряду: відносна точність коливалась від ~60% до ~100%.
   - Використання масштабованої сигмоїдної функції активації дозволило мережі адаптуватися до різних значень ряду та покращити якість прогнозу.
3. **Додаткове завдання**
   - Мережа з трьома входами успішно моделювала задану логічну функцію з трьома змінними, демонструючи правильний вихід для всіх комбінацій.
   - Це підтвердило можливість розширення перцептрона для більш складних функцій за рахунок збільшення кількості входів та відповідної корекції ваг.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. З яких елементів складається штучний нейрон? Яке призначення цих елементів?

Штучний нейрон складається з:

- **Входи (inputs)** – отримують сигнали від попередніх нейронів або зовнішнього середовища.
- **Ваги (weights)** – показують "важливість" кожного входу.
- **Суматор (summation function)** – підсумовує добутки входів на ваги.
- **Поріг (threshold)** – визначає мінімальну суму для активації нейрона.
- **Функція активації (activation function)** – вирішує, скільки вихідного сигналу нейрон передасть далі.

Призначення: нейрон приймає сигнали, обробляє їх з урахуванням ваг і порога, і передає результат далі.

2. Що таке функція активації? Які існують функції активації?

- **Функція активації** визначає вихід нейрона на основі його зваженої суми входів.
- Вона "не лінійно трансформує" сигнал, щоб мережа могла моделювати складні закономірності.

Основні функції активації:

- **Порогова (step)** – 0 або 1, залежно від порога.
- **Сигмоїдна (sigmoid)** – гладка S-подібна функція: вихід 0–1.
- **Лінійна** – вихід = вхід (для прогнозування чисел).

3. З яких основних блоків складається алгоритм навчання нейронних мереж? Як ці блоки пов'язані між собою?

Основні блоки:

1. **Пряме поширення (forward propagation)** – сигнал проходить від входів до виходів.
2. **Обчислення помилки (error calculation)** – порівняння виходу мережі з очікуваним значенням.
3. **Зворотне поширення (backpropagation)** – помилка передається назад і коригує ваги.
4. **Оновлення ваг (weight update)** – застосування градієнтного спуску або іншого правила для навчання.

Взаємозв'язок: сигнал → помилка → корекція → повторення (епохи) → навчена мережа.

4. В чому заключається алгоритм зворотного поширення помилки?

- Спочатку обчислюють помилку на виході мережі.
- Потім ця помилка передається назад через всі шари, і для кожної ваги обчислюється градієнт помилки.
- Ваги змінюють так, щоб зменшити помилку.
- Повторюють процес багато разів (епох) до прийнятної точності.

5. Яким чином можна налаштувати нейронну мережу на прогнозування значень величин часового ряду?

- Вхідні дані – це кілька попередніх значень ряду.
- Вихід – прогноз наступного значення.
- Використовують **один шар або мережу з одним прихованим шаром**, функцію активації для нормалізованих даних (sigmoid або tanh).
- Навчання – методом градієнтного спуску з мінімізацією середньоквадратичної помилки.
- Після навчання мережа може передбачати нові значення на основі останніх спостережень.

6. Яку структуру має нейронна мережа, що моделює логічну функцію XOR?

- Один перцептрон **не навчиться XOR**, бо він **не лінійно роздільний**.
- Для XOR потрібна **мала мережа з прихованим шаром**:
  - **Вхідний шар:** 2 нейрони (x1, x2).
  - **Прихований шар:** 2 нейрони, які реалізують проміжні функції (наприклад, NAND і OR).
  - **Вихідний шар:** 1 нейрон, який об’єднує результати прихованого шару (наприклад, AND).