Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Звіт до лабораторної роботи №1

«Перцептрон»

з дисципліни

«Програмні засоби проектування та реалізації нейромережевих систем»

Виконала:
студентка групи ІМ-13
Мартинюк Марія Павлівна

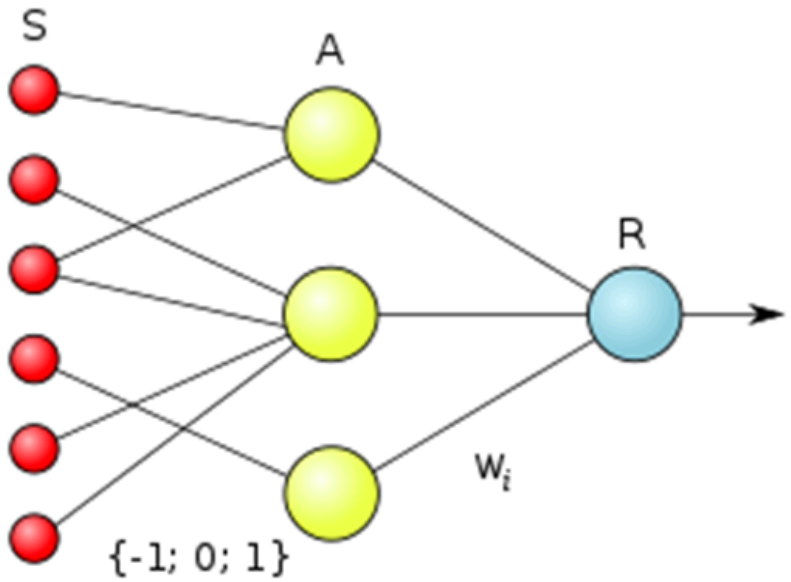
Перевірив:
Шимкович В. М.

Київ 2024

Завдання: Написати програму, що реалізує нейронну мережу Перцептрон та навчити її виконувати функцію XOR.

Хід роботи та короткі теоретичні відомості:

Механізм роботи Перцептрона чудово ілюструє наступний графік:



Для початку визначимо складові елементи перцептрону, які є окремими випадками штучного нейрона з порогової функцією передачі.

- простим S-елементом(сенсором) є чутливий елемент, який від впливу виробляє сигнал. Якщо вхідний сигнал перевищує певний поріг θ, на виході елемента отримуємо +1, в іншому випадку - 0.
- простим A-елементом (асоціативним) називається логічний вирішальний елемент, який дає вихідний сигнал +1, коли алгебраїчна сума його вхідних сигналів перевищує деяку порогову величину θ (кажуть, що елемент *активний*), в іншому випадку вихід дорівнює нулю.
- простим R-елементом(реагуючим, тобто діючим) називається елемент, який видає сигнал +1, якщо сума його вхідних сигналів є строго позитивної, і сигнал -1, якщо сума його вхідних сигналів є

строго негативною. Якщо сума вхідних сигналів дорівнює нулю, вихід вважається або рівним нулю, або невизначеним.

Сигнали від збуджених A-елементів, в свою чергу, передаються в акумулятор R, причому сигнал від *i*-го асоціативного елемента передається з коефіцієнтом w_i. Цей коефіцієнт називається *вагою* A-R зв'язку. Так само як і A-елементи, R-Елемент підраховує суму значень вхідних сигналів, помножених на ваги. Ваги зв'язків S-A (які можуть набувати значень {-1; 0; +1}) і значення порогів A-елементів вибираються випадковим чином на самому початку і потім не змінюються.

Лістинг програми:

```
import numpy as np

class Perceptron:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize random weights for input to hidden layer and hidden to output layer
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)

    def activate_func(self, x):
        # Step activation function: Returns 1 if x > 0, else returns 0
        return np.where(x > 0, 1, 0)

    def train(self, inputs, outputs, epochs, learning_rate):
        for _ in range(epochs):
            # Compute the output of hidden layer
            hidden_layer_output = self.activate_func(np.dot(inputs,
self.weights_input_hidden))
            predicted_output = self.activate_func(np.dot(hidden_layer_output,
self.weights_hidden_output))

            # Compute the errors and delta for the hidden and output layers
            output_error = outputs - predicted_output
            output_delta = output_error * 1
```

```
            hidden_layer_error = output_delta.dot(self.weights_hidden_output.T)
            hidden_layer_delta = hidden_layer_error * 1

            # Update weights
            self.weights_hidden_output += learning_rate *
hidden_layer_output.T.dot(output_delta)
            self.weights_input_hidden += learning_rate * inputs.T.dot(hidden_layer_delta)

    def predict(self, inputs):
        # Make predictions using the trained perceptron
        hidden_layer_output = self.activate_func(np.dot(inputs,
self.weights_input_hidden))
        return self.activate_func(np.dot(hidden_layer_output,
self.weights_hidden_output))

# XOR input and output for further testing
input_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
output_data = np.array([[0], [1], [1], [0]])

# Number of epochs and learning rate
epochs = 150
learning_rate = 0.1

# Size of layers
input_size=2
hidden_size=2
output_size=1

# Create and train the neural network
perceptron = Perceptron(input_size, hidden_size, output_size)
perceptron.train(input_data, output_data, epochs, learning_rate)


# Predict and print results
for test_input, target_output in zip(input_data, output_data):
    training_res = perceptron.predict(test_input)[0]
    print(f"{test_input} -> {training_res} | Target result: {target_output[0]}")
```

Результати:

```
lab1.py
[0 0] -> 0 | Target result: 0
[0 1] -> 1 | Target result: 1
[1 0] -> 1 | Target result: 1
[1 1] -> 0 | Target result: 0
```

Висновок:

Задля виконання лабораторної роботи було розроблено нейронну мережу на базі перцептрону, якого було навчено виконувати функцію XOR. Отримані результати підтверджують правильність алгоритму організації навчання нейронної мережі.