МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

ЛАБОРАТОРНА РОБОТА № 3.2

з дисципліни
“Інтелектуальні вбудовані системи”
на теми
“ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ. МОДЕЛЬ PERCEPTRON”

Виконав:
Студент групи ІП-84
Павловський В.Є.
№ ЗК: ІП-8417

Перевірив:
викладач Регіда П.Г.

Київ 2021

Основні теоретичні відомості

Важливою задачеюяку система реального часу має вирішувати є отримання необхідних для обчислень параметрів, її обробка та виведення результату у встановлений дедлайн. З цього постає проблема отримання водночас точних та швидких результатів. Модель Перцпептрон дозволяє покроково наближати початкові значення.

Розглянемо приклад: дано дві точки A(1,5), B(2,4), поріг спрацювання P = 4, швидкість навчання δ = 0.1. Початкові значення ваги візьмемо нульовими W1 = 0, W2 = 0.

Розрахунок вихідного сигналу у виконується за наступною формулою:

x1 *W1 +x2 *W2 =y

Для кожного кроку потрібно застосувати дельта-правило, формула для розрахунку похибки:

Δ=P–y

де y – значення на виході.

Для розрахунку ваги, використовується наступна формули:

W1(i+1) = W1(i) + W2 * x11

W2(i+1) = W1(i) + W2 * x12 де i – крок, або ітерація алгоритму.

Умови завдання для варіанту

Поріг спрацювання: P = 4
Дано точки: A(0,6), B(1,5), C(3,3), D(2,4).
Швидкості навчання: δ = {0,001; 0,01; 0,05; 0.1; 0.2; 0,3}
Дедлайн: часовий = {0.5c; 1c; 2c; 5c}, кількість ітерацій = {100;200;500;1000} Обрати швидкість навчання та дедлайн. Налаштувати Перцептрон для даних
точок. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрати часу та точності результату за різних параметрах навчання.

Лістинг програми із заданими умовами завдання

```
//
//  ContentView.swift
//  rts-lab-3-2
//
//  Created by Vsevolod Pavlovskyi on 24.05.2021.
//

import SwiftUI
import Combine

struct ContentView: View {

    @ObservedObject var viewModel = ViewModel()

    var body: some View {
        VStack(alignment: .leading) {
            pickers
            points
            Spacer()
            labels
            Spacer()
            button
        }
    }
}

// MARK: -Interface
private extension ContentView {

    var button: some View {
        Button(action: { viewModel.compute() }) {
            HStack {
```

```
                Spacer()
                Text("Compute")
                    .foregroundColor(.white)
                    .font(.headline)
                Spacer()
            }
            .padding()
            .background(Color.blue)
            .cornerRadius(8)
        }
        .padding()
    }

    var labels: some View {
        VStack(alignment: .leading) {
            threshold
            w1
            w2
            result
        }
        .font(.body)
        .padding()
    }

    var threshold: some View {
        Text("P: \(viewModel.threshold)")
    }

    var w1: some View {
        Text("W1: \(viewModel.w1)")
    }

    var w2: some View {
        Text("W2: \(viewModel.w2)")
    }

    var result: some View {
        Text("Result: \(viewModel.result ?? false == true ? "Corrent" : "Can't do the
calculations in a proper time/iterations number")")
            .bold()
    }
}

// MARK: -Poins
private extension ContentView {

    var points: some View {
        VStack {
            HStack {
```

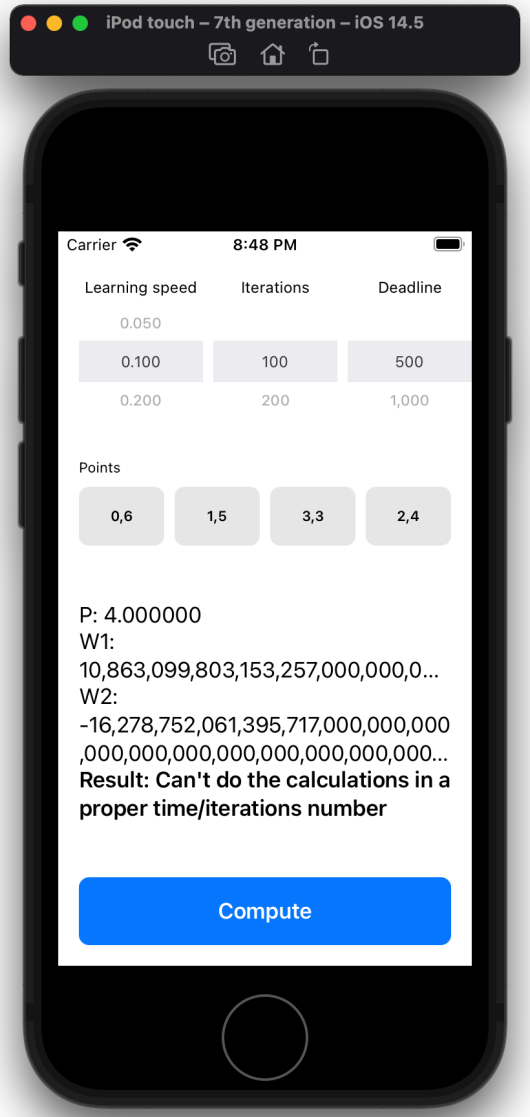
```
                Text("Points")
                    .font(.caption2)
                Spacer()
            }
            HStack {
                ForEach(viewModel.points, id:\.0.self) {
                    point(for: $0)
                }
            }
        }
        .padding()
    }

    func point(for point: Point) -> some View {
        HStack {
            Spacer()
            Text(stringRepresentation(of: point))
                .font(.caption2)
                .bold()
                .lineLimit(1)
            Spacer()
        }
        .padding()
        .background(Color.gray.opacity(0.2))
        .cornerRadius(8)
    }

    func stringRepresentation(of point: Point) -> String {
        "\(Int(point.0)),\(Int(point.1))"
    }

}

// MARK: -Pickers
private extension ContentView {

    var pickers: some View {
        GeometryReader { proxy in
            HStack {
                speedPicker
                    .frame(width: proxy.size.width / 3)
                    .clipped()
                iterationNumber
                    .frame(width: proxy.size.width / 3)
                    .clipped()
                deadline
                    .frame(width: proxy.size.width / 3)
                    .clipped()
            }
```

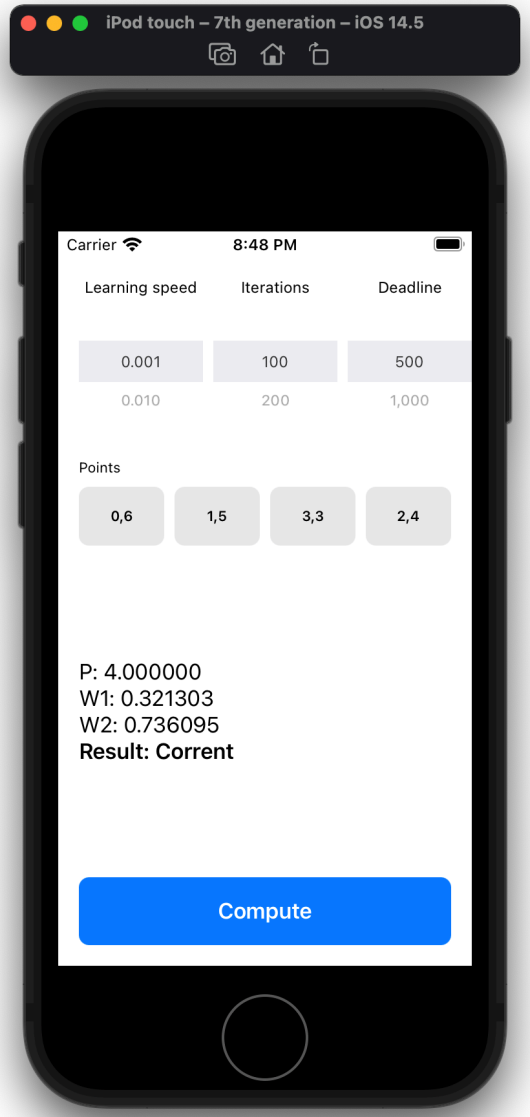
```
                .font(.caption)
            }
            .frame(height: 100)
            .padding()
        }

        var speedPicker: some View {
            VStack {
                Text("Learning speed")
                Picker("Learning speed", selection: $viewModel.speed) {
                    ForEach(viewModel.speeds, id:\.self) {
                        Text("\($0, specifier: "%.3f")")
                            .font(.caption)
                    }
                }
                .frame(height: 80)
                .clipped()
            }
        }

        var iterationNumber: some View {
            VStack {
                Text("Iterations")
                Picker("Iterations", selection: $viewModel.iterationNumber) {
                    ForEach(viewModel.iterationNumbers, id:\.self) {
                        Text("\($0)")
                            .font(.caption)
                    }
                }
                .frame(height: 80)
                .clipped()
            }
        }

        var deadline: some View {
            VStack {
                Text("Deadline")
                Picker("Deadline", selection: $viewModel.deadline) {
                    ForEach(viewModel.deadlines, id:\.self) {
                        Text("\($0)")
                            .font(.caption)
                    }
                }
                .frame(height: 80)
                .clipped()
            }
        }

}
```

```
typealias Point = (Double, Double)

class ViewModel: ObservableObject {

    @Published var speed: Double
    @Published var iterationNumber: Int
    @Published var deadline: Int

    @Published var w1: Double = 0
    @Published var w2: Double = 0

    @Published var result: Bool?

    public var threshold: Double = 4
    public var points: [Point] = [
        (0, 6),
        (1, 5),
        (3, 3),
        (2, 4)
    ]

    public var speeds = [0.001, 0.01, 0.05, 0.1, 0.2, 0.3]
    public var iterationNumbers = [100, 200, 500, 1000]
    public var deadlines = [500, 1000, 2000, 5000]

    private var cancellable = Set<AnyCancellable>()

    init() {
        speed = speeds[0]
        iterationNumber = iterationNumbers[0]
        deadline = deadlines[0]
    }

    public func compute() {

        let startTime = DispatchTime.now()

        result = nil
        w1 = 0
        w2 = 0

        for _ in 0..<iterationNumber {
            for point in points {
                // Time elapsed in nanosec
                let timeElapsed = DispatchTime.now().uptimeNanoseconds -
startTime.uptimeNanoseconds
                print(timeElapsed)
                if timeElapsed > deadline * 1_000_000 { return }
```

```
                let y = calculateSignal(point: point, w1: w1, w2: w2)
                let delta = threshold - y
                refreshWeights(w1: &w1, w2: &w2, point: point, speed: speed, delta:
delta)
                if validate(points: points, w1: w1, w2: w2, threshold: threshold) {
                    self.result = true
                    return
                }
            }
        }
        self.result = false
    }

}

private extension ViewModel {

    func calculateSignal(point: Point, w1: Double, w2: Double) -> Double {
        point.0 * w1 + point.1 * w2
    }

    func refreshWeights(w1: inout Double,
                        w2: inout Double,
                        point: Point,
                        speed: Double,
                        delta: Double) {
        w1 += delta * point.0 * speed
        w2 += delta * point.1 * speed
    }

    func validate(points: [Point], w1: Double, w2: Double, threshold: Double) -> Bool
{
        let middle = Int(points.count / 2)
        return points
            .map {
                calculateSignal(point: $0, w1: w1, w2: w2)

            }
            .enumerated()
            .filter { (index, value) in
                (index < middle && value < threshold) ||
                    (index ⩾ middle && value > threshold)
            }
            .count == 0
    }

}
```

Результати виконання програми





Висновки

Під час лабораторної роботи ми ознайомлись з принципами машинного навчання за допомогою математичної моделі сприйняття інформації Перцептрон(Perceptron), змоделювали роботу нейронної мережі та дослідили вплив параметрів на час виконання та точність результату