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 тему
«Дослідження нейронних мереж. Модель perceptron»

Виконала:

студентка

групи ІП-83

Гомілко Діана Володимирівна

Перевірив:

Регіда Павло Геннадійович

Київ 2021

Основні теоретичні відомості, необхідні для виконання лабораторної роботи

Важливою задачею, яку 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:

$$\Delta = P - y$$

де y – значення на виході.

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

MainActivity.kt

```
package com.example.modelperceptron

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.SystemClock
import android.view.View
import android.widget.*

class MainActivity : AppCompatActivity() {
    private lateinit var submitBtn: Button
    private lateinit var resultArea: TextView
```

```
    private var timeDeadline = 0.5
    private var iterationsDeadline = 100
    private var learningRate = 0.001
    private lateinit var perceptron: Perceptron

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val points = listOf(
                listOf(0, 6),
                listOf(1, 5),
                listOf(3, 3),
                listOf(2, 4)
        )
        perceptron = Perceptron(points, 4, 2)

        val spinnersIds = listOf(R.id.spinnerLearningRate, R.id.spinnerIterDeadline,
R.id.spinnerTimeDeadline)
        val lrSpinnerClb: SpinnerClb = { parent, _, p, _ ->
            learningRate = parent!!.getItemAtPosition(p).toString().toDouble() }
        val iterSpinnerClb: SpinnerClb = { parent, _, p, _ ->
            iterationsDeadline = parent!!.getItemAtPosition(p).toString().toInt() }
        val timeSpinnerClb: SpinnerClb = { parent, _, p, _ ->
            timeDeadline = parent!!.getItemAtPosition(p).toString().toDouble() }
        val spinnersData = listOf(
                Pair(R.array.learning_rates, lrSpinnerClb),
                Pair(R.array.iteration_deadlines, iterSpinnerClb),
                Pair(R.array.time_deadlines, timeSpinnerClb)
        )
        spinnersIds.mapIndexed { i, id ->
            setUpSpinner(id, spinnersData[i].first, spinnersData[i].second)
        }

        submitBtn = findViewById(R.id.btn)
        resultArea = findViewById(R.id.resultArea)
        submitBtn.setOnClickListener { handleSubmitBtn() }
    }

    fun onRadioButtonClicked(view: View) {
        val checked = (view as RadioButton).isChecked
        if (view.getId() == R.id.radioTime && checked)
perceptron.setDeadline(timeDeadline)
        if (view.getId() == R.id.radioIterations && checked)
perceptron.setDeadline(iterationsDeadline)
    }

    private fun setUpSpinner(id: Int, res: Int, clb: SpinnerClb) {
        val spinner = findViewById<Spinner>(id)
        val resource = resources.getStringArray(res)
        val adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item,
resource)
        spinner.adapter = adapter
        spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>?, view: View?, position:
Int, id: Long) {
                clb(parent, view, position, id)
            }
            override fun onNothingSelected(parent: AdapterView<*>?) { }
        }
    }

    private fun handleSubmitBtn() {
```

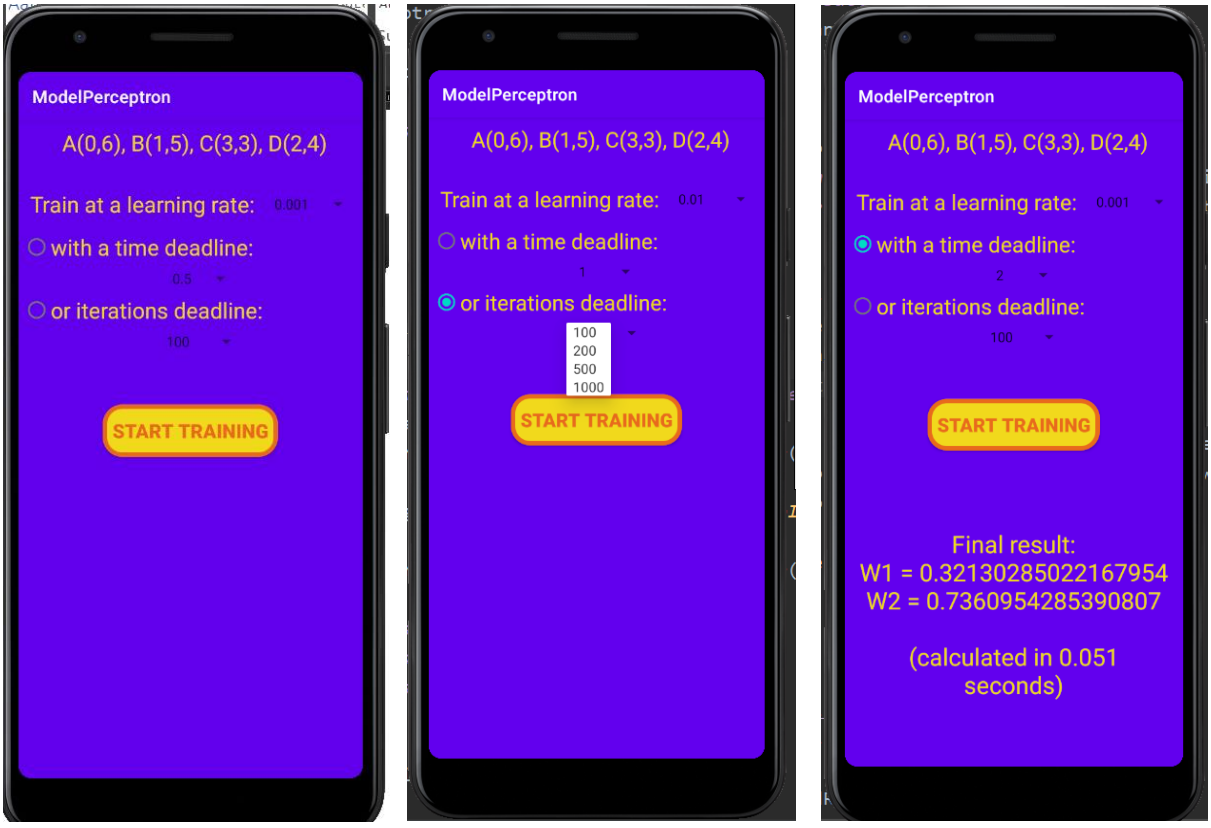
```
        perceptron.setLearningRate(learningRate)
        val startTime = SystemClock.elapsedRealtime()
        val result = perceptron.train()
        val endTime = SystemClock.elapsedRealtime()
        val elapsedMilliSeconds = endTime - startTime
        val elapsedSeconds = elapsedMilliSeconds / 1000.0
        val isOptimal = result.second
        val resultStr = "W1 = ${result.first[0]}\nW2 =
${result.first[1]}\n\n(calculated in $elapsedSeconds seconds)"
        val prefix = if (isOptimal) "Final result:\n" else "Best fit (missed
deadline):\n"
        resultArea.text = prefix + resultStr
    }
}

private typealias SpinnerClb = (parent: AdapterView<*>?, view: View?, pos: Int, id:
Long) -> Unit
```

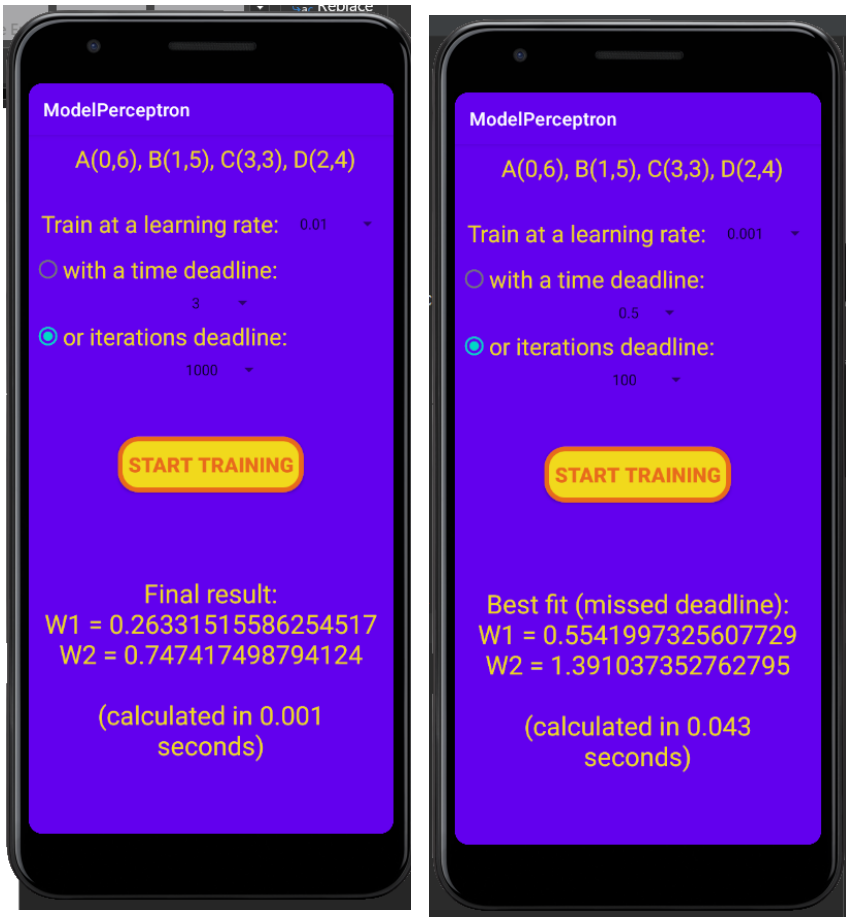
Perceptron.kt

```
package com.example.modelperceptron

import android.os.SystemClock

class Perceptron(
        private val points: List<List<Int>>,
        private val threshold: Int,
        private val border: Int
) {
    private val weights = mutableListOf(0.0, 0.0)
    private var learningRate = 0.01
    private var timeDeadline = 0.0
    private var iterationsDeadline = 0

    private fun activation(point: List<Int>): Double =
            weights.mapIndexed { i, w -> w * point[i] }.sum()

    private fun checkFitness(): Boolean {
        val size = points.size
        return points
                    .slice(0 until border)
                    .all { point -> activation(point) > threshold } &&
                points
                    .slice(border until size)
                    .all { point -> activation(point) < threshold }
    }

    private fun updateWeights(point: List<Int>, activationRes: Double) {
        val delta = threshold - activationRes
        weights.mapIndexed { i, w -> weights[i] = w + delta * point[i] * learningRate }
    }

    fun setDeadline(iterations: Int) {
        iterationsDeadline = iterations
        timeDeadline = 0.0
    }
    fun setDeadline(time: Double) {
        timeDeadline = time * 1000
        iterationsDeadline = 0
    }
    fun setLearningRate(rate: Double) { learningRate = rate }
```

```
    private fun checkDeadline(count: Long): Boolean =
            if (timeDeadline != 0.0) count < timeDeadline else count <
iterationsDeadline

    fun train(): Pair<List<Double>, Boolean> {
        weights.mapIndexed { i, _ -> weights[i] = 0.0 }
        var curCount = 0L
        val startTime = SystemClock.elapsedRealtime()
        println("$timeDeadline, $iterationsDeadline")
        while(checkDeadline(curCount)) {
            points.map { point ->
                if (checkFitness()) return Pair(weights, true)
                val y = activation(point)
                updateWeights(point, y)
                if (timeDeadline != 0.0) curCount = SystemClock.elapsedRealtime() -
startTime
                else curCount++
            }
        }
        return Pair(weights, false)
    }
}
```

Результати виконання кожної програми





Висновки щодо виконання лабораторної роботи

Під час виконання даної лабораторної роботи ми ознайомилися з принципами машинного навчання за допомогою математичної моделі сприйняття інформації Перцептрон(Perceptron) та змоделювали роботу нейронної мережі та дослідити вплив параметрів на час виконання та точність результату. Було створено програму у вигляді маобільного застосунку з користувацьким інтерфейсом, що дозволяє задавати швидкість навчання та різні варіанти дедлайну.