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"

Виконав:

Студент групи ІП-83

Карпюк І.В.

№ 3K: IП-8311

Перевірив:

викладач Регіда

П.Г.

Основні теоретичні відомості

Шора. 🔛

Метод перебору можливих дільників.

```
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 \delta = 0.1. Початкові значення ваги візьмемо нульовими
W2 = 0. Розрахунок вихідного сигналу у виконується за наступною формулою:
x 1 * W 1 + x 2 * W 2 = y
Для кожного кроку потрібно застосувати дельта-правило, формула для
розрахунку похибки:
\Delta = P - y
де у - значення на виході.
 Для розрахунку ваги, використовується наступна формули:
W \ 1 \ (i+1) = W \ 1 \ (i) + W \ 2 * x \ 11
W \ 2 \ (i+1) = W \ 1 \ (i) + W \ 2 * x \ 12
де і - крок, або ітерація алгоритму.
Розпочнемо обробку:
1 ітерація:
Використовуємо формулу обрахунку вихідного сигналу:
     0 = 0 * 1 + 0 * 5 значення не підходить, оскільки воно менше зазначеного
порогу. Вихідний сигнал повинен бути строго більша за поріг.
Далі, рахуємо \Delta:
 \Delta = 4 - 0 = 4
За допомогою швидкості навчання \delta та минулих значень ваги, розрахуємо нові
значення ваги:
 W 1 = 0 + 4 * 1 * 0,1 = 0,4
 W~2~=~0~+~4~\star~5~\star~0.1~=~2 Таким чином ми отримали нові значення ваги. Можна
побачити, що результат
змінюється при зміні порогу.
2 ітерація:
Виконуємо ті самі операції, але з новими значеннями ваги та для іншої точки.
  8,8 = 0,4 * 2 + 2 * 4 , не підходить, значення повинно бути менше порогу.
  \Delta = -5 , спрощуємо результат для прикладу.
 W 1 = 0,4 + 5 * 2 * 0,1 = -0,6
  W 2 = 2 - 5 * 4 * 0.1 = 0
3 ітерація:
Дано тільки дві точки, тому повертаємось до першої точки та нові значення ваги
розраховуємо для неї.
```

-0,6 = -0,6 * 1 + 0 * 5 , не підходить, значення повинно бути більше порогу.

 $\Delta = 5$, спрощуємо результат для прикладу.

W 1 = -0.6 + 5 * 1 * 0.1 = -0.1

```
W 2 = 0 + 5 * 5 * 0.1 = 2,5
По такому самому принципу рахуемо значення ваги для наступних ітерацій, поки не отримаємо значення, які задовольняють вхідним даним.
На восьмій ітерації отримуємо значення ваги W 1 = -1,8 та W 2 = 1,5.

5,7 = -1,8 * 1 + 1,5 * 5, більше за поріг, задовольняє
2,4 = -1,8 * 2 + 1,5 * 4, менше за поріг, задовольняє
Отже, бачимо, що для заданого прикладу, отримано значення ваги за 8 ітерацій.
При розрахунку значень, потрібно враховувати дедлайн. Дедлайн може бути в вигляді максимальної кількості ітерацій або часовий.
```

Лістинг програми із заданими умовами завдання

```
} package ua.kpi.comsys.factorio
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.*
import androidx.fragment.app.Fragment
import kotlin.concurrent.timer
import kotlin.system.measureTimeMillis
data class Point(val x: Double, val y: Double)
class Perceptron : Fragment () {
    override fun onCreateView(
            inflater: LayoutInflater,
            container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.perceptron layout, container, false)
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val threshold: Int = 4
        val points = listOf(Point(0.0, 6.0), Point(1.0, 5.0), Point(3.0, 3.0),
Point(2.0, 4.0))
        val speeds = listof(0.001, 0.01, 0.05, 0.1, 0.2, 0.3)
        val deadlines = listOf(0.5, 1.0, 2.0, 5.0)
        val iterations = listOf(100, 200, 500, 1000)
        val illegalValues = listOf(
           Double.NaN,
           Double.NEGATIVE_INFINITY,
           Double.POSITIVE INFINITY
        val speedSpinner: Spinner = view.findViewById(R.id.speed)
        val deadlineSpinner: Spinner = view.findViewById(R.id.deadline)
```

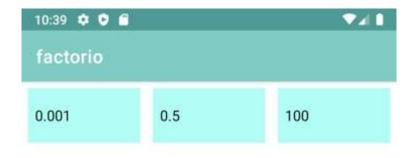
```
val iterationsSpinner: Spinner = view.findViewById(R.id.iterations)
        val textOutput: TextView = view.findViewById(R.id.output)
        val speedArray = ArrayAdapter(context,
R.layout.support simple spinner dropdown item, speeds).also { speedArray ->
speedArray.setDropDownViewResource(android.R.layout.simple spinner dropdown item)
            speedSpinner.adapter = speedArray
        val deadlineArray = ArrayAdapter(context,
R.layout.support simple spinner dropdown item, deadlines).also { deadlineArray ->
deadlineArray.setDropDownViewResource(android.R.layout.simple spinner dropdown item)
            deadlineSpinner.adapter = deadlineArray
        }
        val iterationsArray = ArrayAdapter(context,
R.layout.support_simple_spinner_dropdown_item, iterations).also { iterationsArray ->
iterationsArray.setDropDownViewResource(android.R.layout.simple spinner dropdown item
            iterationsSpinner.adapter = iterationsArray
        val time: Int = 3000
        view.findViewById<Button>(R.id.cycle).setOnClickListener {
            var counter: Int = 0
            val time in = System.currentTimeMillis()
                val accuracy = percept(speeds.random(), deadlines.random(),
iterations.random())
                if (!(accuracy.first in illegalValues || accuracy.second in
illegalValues)) {
                    counter++
                if ((System.currentTimeMillis() - time_in) > time) {
                    break
            textOutput.text = counter.toString()
        view.findViewById<Button>(R.id.calc).setOnClickListener { ->
            val speed = speedSpinner.selectedItem.toString().toDouble()
            val deadline = deadlineSpinner.selectedItem.toString().toDouble()
            val iterations = iterationsSpinner.selectedItem.toString().toInt()
            val accuracy = percept(speed, deadline, iterations)
```

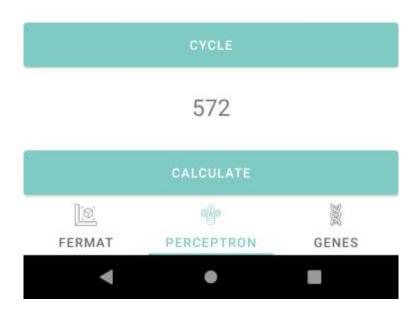
```
if (accuracy.first in illegalValues || accuracy.second in illegalValues)
                textOutput.text = "No solution found"
                textOutput.text = accuracy.toString()
fun percept (speed: Double, deadline: Double, iterations: Int): Pair<Double, Double>
    var W1 = 0.00
    var W2 = 0.00
    var P = 4.00
    var points = arrayListOf(Pair(0.00, 6.00), Pair(1.00, 5.00), Pair(3.00, 3.00),
Pair(2.00, 4.00))
    fun validate(): Boolean {
        val y1 = W1 * points[0].first + W2 * points[0].second
        val y2 = W1 * points[1].first + W2 * points[1].second
        val y3 = W1 * points[2].first + W2 * points[2].second
        val y4 = W1 * points[3].first + W2 * points[3].second
        if ((y1 > P) \&\& (y2 > P) \&\& (y3 < P) \&\& (y4 < P)) {
            return true
    val time_in = System.currentTimeMillis()
    for (i in 0..iterations) {
        if ((System.currentTimeMillis() - time_in) <= deadline * 1000) {</pre>
            for (k in 0 until points.size) {
                val y = W1 * points[k].first + W2 * points[k].second
                val delta = P - y
                W1 += delta * points[k].first * speed
                W2 += delta * points[k].second * speed
                if (validate()) {
                    return Pair(W1, W2)
    return Pair(W1, W2)
```

Результати виконання програми









Висновки

Вивчив роботу Перцептрону.