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 №2а

«ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ. МОДЕЛЬ PERCEPTRON»

Виконав:
студент ІІ курсу ФІОТ
групи ІВ-91
Черних Богдан

Перевірив:
Регіда П.Г.

Київ – 2021

Мета роботи: ознайомлення з принципами машинного навчання за допомогою математичної моделі сприйняття інформації Перцептрон(Perceptron). Змоделювати роботу нейронної мережі та дослідити вплив параметрів на час виконання та точність результату.

Теоретичі відомості

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 $\delta = 0.1$. Початкові значення ваги візьмемо нульовими $W1 = 0$, $W2 = 0$. 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 y – 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 i – 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 Δ:

$\Delta = 4 - 0 = 4$

За допомогою швидкості навчання δ та минулих значень ваги, розрахуємо нові значення ваги:

$W_1 = 0 + 4 * 1 * 0,1 = 0,4$

$W_2 = 0 + 4 * 5 * 0.1 = 2$

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 рахуємо 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.

Завдання до лабораторної роботи:

Поріг спрацювання: $P = 4$

Дано точки: A(0,6), B(1,5), C(3,3), D(2,4).

Швидкості навчання: $\delta = \{0,001; 0,01; 0,05; 0.1; 0.2; 0,3\}$

Дедлайн: часовий = {0.5с; 1с; 2с; 5с}, 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.

Роздруківка тексту програми

MainActivity.java

```java
package com.example.lab2a;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    private Button button;
    private EditText editTextLimit;
    private EditText editTextLearnRate;
    private EditText editTextIterations;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button = findViewById(R.id.button);
        editTextLimit = findViewById(R.id.editTextLimit);
        editTextLearnRate = findViewById(R.id.editTextLearRate);
        editTextIterations = findViewById(R.id.editTextIterations);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                String limitText = editTextLimit.getText().toString();
                String learnRateText = editTextLearnRate.getText().toString();
                String iterationsText = editTextIterations.getText().toString();

                int limit;
                double learnRate;
                int iterations;
                if (limitText.equals("") || learnRateText.equals("") ||
iterationsText.equals("")) {
                    Toast.makeText(getApplicationContext(), "Fields can`t be empty!",
Toast.LENGTH_LONG).show();
                } else {

                    limit = Integer.parseInt(limitText);
                    learnRate = Double.parseDouble(learnRateText);
                    iterations = Integer.parseInt(iterationsText);

                    Point[] points = {
                            new Point(0, 6, limit),
                            new Point(1, 5, limit),
                            new Point(2, 4, limit),
                            new Point(3, 3, limit)
                    };

                    Network network = new Network(points, limit, learnRate,
iterations);
                    double[] weights = network.train();

                    Intent intent = new Intent(getApplicationContext(),
```

```
ChartActivity.class);
                    intent.putExtra("weights", weights);
                    intent.putExtra("limit", limit);
                    startActivity(intent);
                }
            }
        });
    }
}
```

ChartActivity.java

```
package com.example.lab2a;

import androidx.appcompat.app.AppCompatActivity;

import android.graphics.Color;
import android.os.Bundle;
import android.widget.TextView;

import com.github.mikephil.charting.charts.CombinedChart;
import com.github.mikephil.charting.charts.ScatterChart;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.CombinedData;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.data.ScatterData;
import com.github.mikephil.charting.data.ScatterDataSet;
import com.github.mikephil.charting.utils.ColorTemplate;

import java.util.ArrayList;

public class ChartActivity extends AppCompatActivity {

    private CombinedChart chart;
    private final int count = 12;
    private TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chart);

        Bundle arguments = getIntent().getExtras();
        double[] weights = (double[]) arguments.get("weights");
        int limit = (int) arguments.get("limit");

        chart = findViewById(R.id.chart);
        textView = findViewById(R.id.textResult);
        textView.setText("Result:\nw1 = " + weights[0] + "\nw2 = " + weights[1] +
"\nw_bias = " + weights[2]);

        chart.getDescription().setEnabled(false);
        chart.setBackgroundColor(Color.WHITE);
        chart.setDrawGridBackground(true);

        YAxis leftAxis = chart.getAxisLeft();
        leftAxis.setDrawGridLines(true);
```

```
        leftAxis.setAxisMinimum(0f); // this replaces setStartAtZero(true)

        XAxis xAxis = chart.getXAxis();
        xAxis.setPosition(XAxis.XAxisPosition.BOTH_SIDED);

        xAxis.setGranularity(1f);

        CombinedData data = new CombinedData();

        data.setData(generateLineData(weights, limit));
        data.setData(generateScatterData());

        xAxis.setAxisMaximum(data.getXMax() + 0.25f);
        xAxis.setAxisMinimum(-1);

        chart.setData(data);
        chart.invalidate();
    }

    private LineData generateLineData(double[] weights, int limit) {

        LineData d = new LineData();

        ArrayList<Entry> entries = new ArrayList<>();

        for (int x = 0; x < count; x++)
            entries.add(new Entry(x, y(x, limit, weights)));

        LineDataSet set = new LineDataSet(entries, "Perceptron`s guess");

        set.setColor(Color.rgb(255, 0, 0));
        set.setLineWidth(2.5f);

        set.setDrawCircles(false);
        set.setCircleColor(Color.rgb(0, 0, 255));
        set.setDrawValues(false);

        set.setAxisDependency(YAxis.AxisDependency.LEFT);
        d.addDataSet(set);

        return d;
    }

    private ScatterData generateScatterData() {

        ScatterData d = new ScatterData();

        ArrayList<Entry> entries = new ArrayList<>();
        entries.add(new Entry(0, 6));
        entries.add(new Entry(1, 5));
        entries.add(new Entry(2, 4));
        entries.add(new Entry(3, 3));

        ScatterDataSet set = new ScatterDataSet(entries, "Points: A, B, C, D");
        set.setColors(ColorTemplate.MATERIAL_COLORS);
        set.setScatterShapeSize(15.5f);
        set.setScatterShape(ScatterChart.ScatterShape.CIRCLE);
        set.setDrawValues(false);

        d.addDataSet(set);

        return d;
```

```
    }

    private float y(int x, int limit, double[] weights) {
        return (float) ((limit - weights[0] * x - weights[2]) / weights[1]);
    }
}
```

Point.java

```
package com.example.lab2a;

public class Point {
    int x1;
    int x2;
    int bias = 1;
    String label;

    public Point(int x1, int x2, String label) {
        this.x1 = x1;
        this.x2 = x2;
        this.label = label;
    }

    public Point(int x1, int x2, int limit) {
        this.x1 = x1;
        this.x2 = x2;

         if (x2 > limit) label = "A";
         else label = "B";
    }
}
```

Perceptron.java

```
package com.example.lab2a;

import java.util.Arrays;

public class Perceptron {

    double[] weights = new double[3];
    int limit;
    double learnRate;
    double sum;

    public Perceptron(int limit, double learnRate) {
        this.limit = limit;
        this.learnRate = learnRate;

        Arrays.fill(weights, 0);
    }

    private String activationFunc(double sum) {
        if (sum > limit) {
            return "A";
        }
        return "B";
    }
```

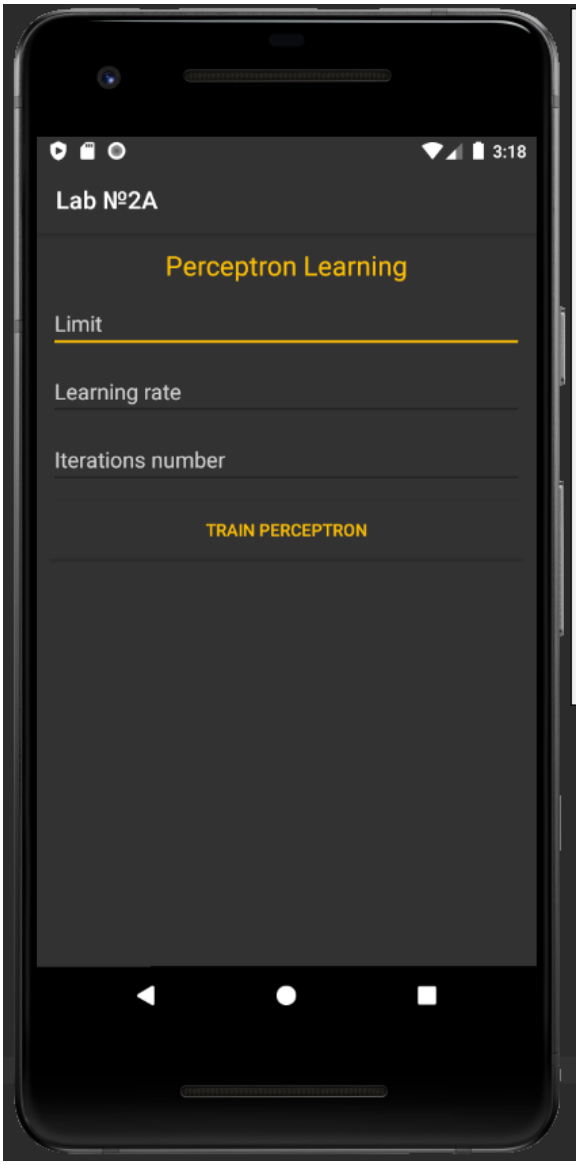
```
    public String guess(int[] input) {
        sum = 0;
        for (int i = 0; i < input.length; i++) {
            sum += input[i] * weights[i];
        }
        return activationFunc(sum);
    }

    public void train(int[] input, String target) {
        String guess = guess(input);
//        System.out.println("Expected value: " + target + ". Guessed value: " +
guess);
        if (!guess.equals(target)) {
//            System.out.println("Changing the weights...");
            double error = limit - sum;
            for (int i = 0; i < weights.length; i++) {
                weights[i] += error * input[i] * learnRate;
            }
        } else {
//            System.out.println("Right guess!");
        }
    }

    @Override
    public String toString() {
        return "Perceptron`s weights: w1 = " + weights[0] + ", w2 = " + weights[1] +
", w_bias = " + weights[2];
    }
}
```

Network.java

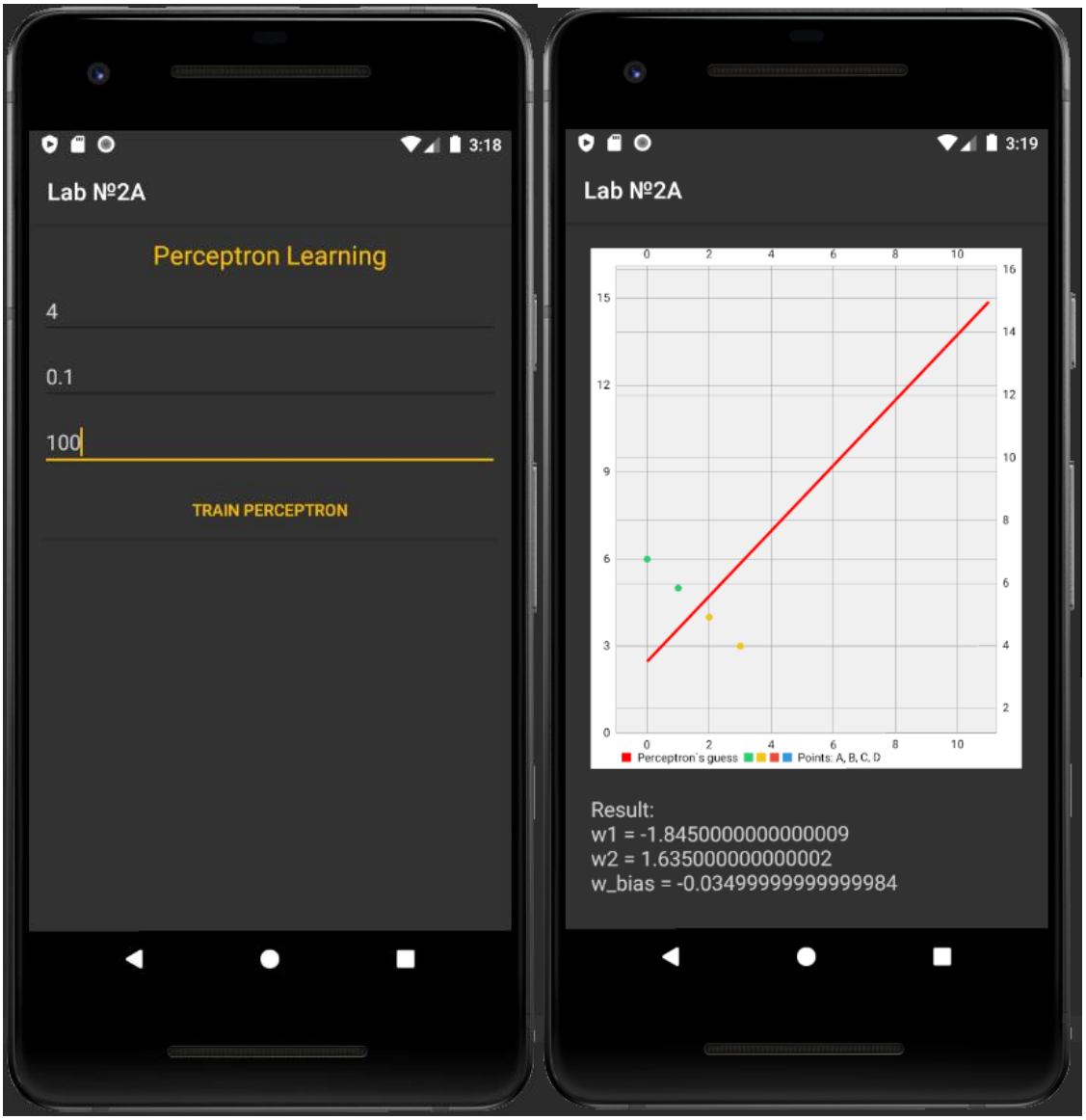
```
package com.example.lab2a;

import java.util.ArrayDeque;
import java.util.Arrays;

public class Network {

    Point[] points;
    int limit;
    double learnRate;
    int iterations;

    public Network(Point[] points, int limit, double learnRate, int iterations) {
        this.points = points;
        this.limit = limit;
        this.learnRate = learnRate;
        this.iterations = iterations;
    }

    public double[] train() {

        ArrayDeque<Point> pointArrayDeque = new
ArrayDeque<Point>(Arrays.asList(points));
        Perceptron perceptron = new Perceptron(limit, learnRate);

        Point currentPoint;
        for (int i = 0; i < iterations; i++) {
//            System.out.println("Iteration №" + (i + 1));
```

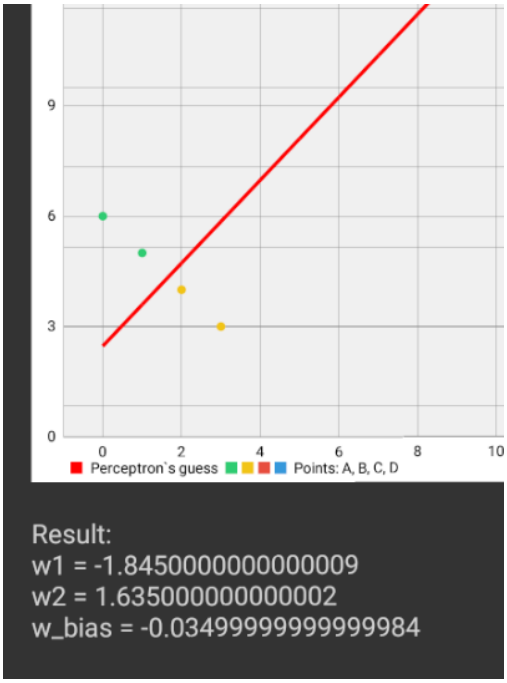
```
            currentPoint = pointArrayDeque.pop();

            int[] input = {currentPoint.x1, currentPoint.x2, currentPoint.bias};
            String target = currentPoint.label;
            perceptron.train(input, target);

            pointArrayDeque.addLast(currentPoint);
        }

//         System.out.println(perceptron);

        return perceptron.weights;
    }

}
```

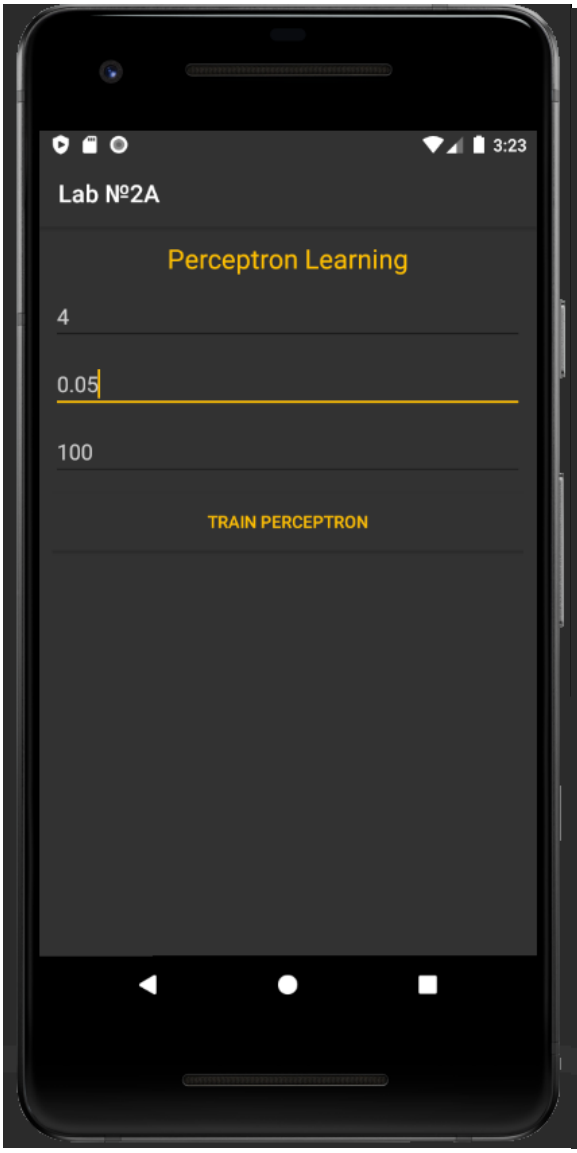
Результати роботи програми

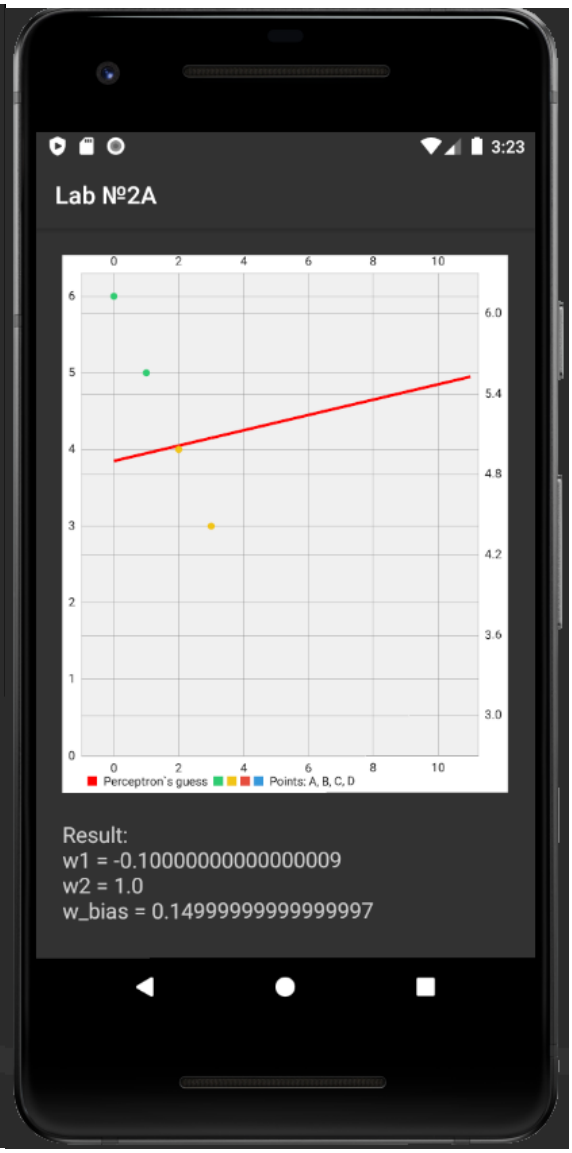


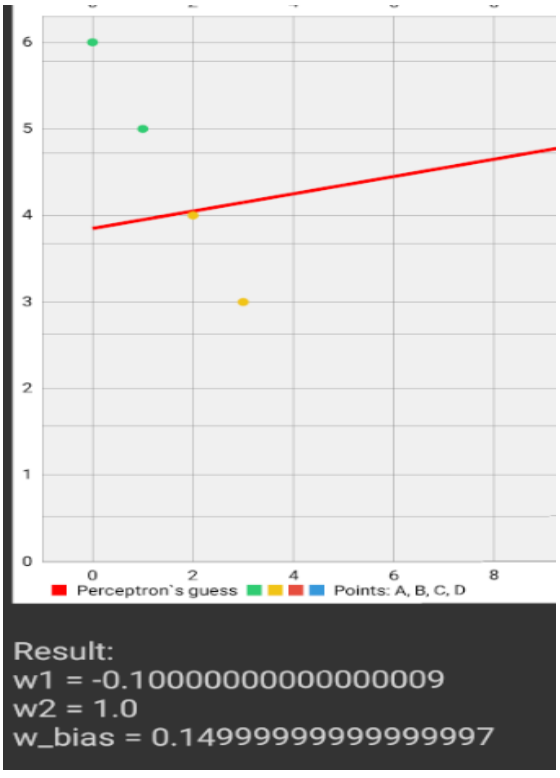
Вдалі тренування



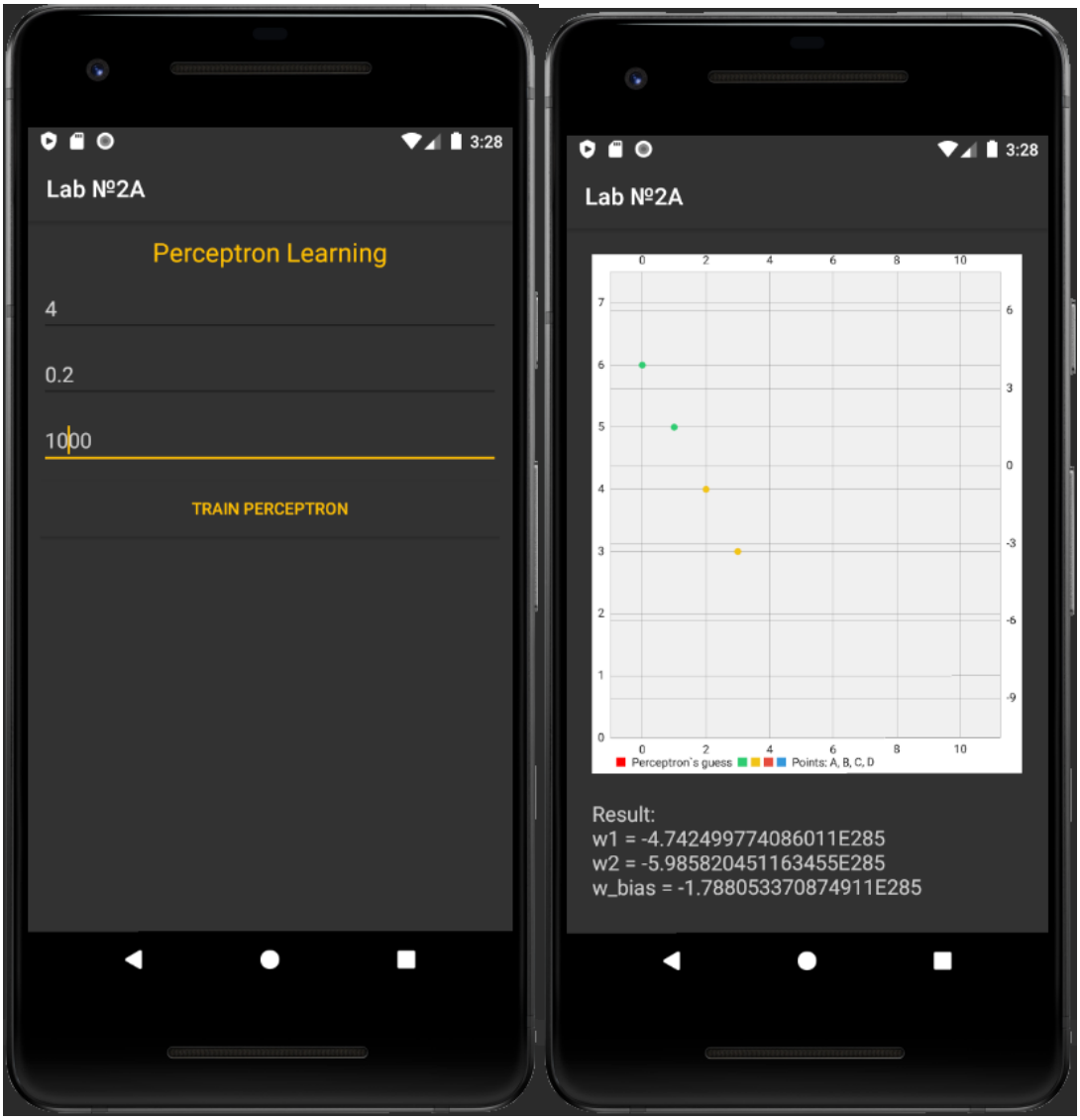


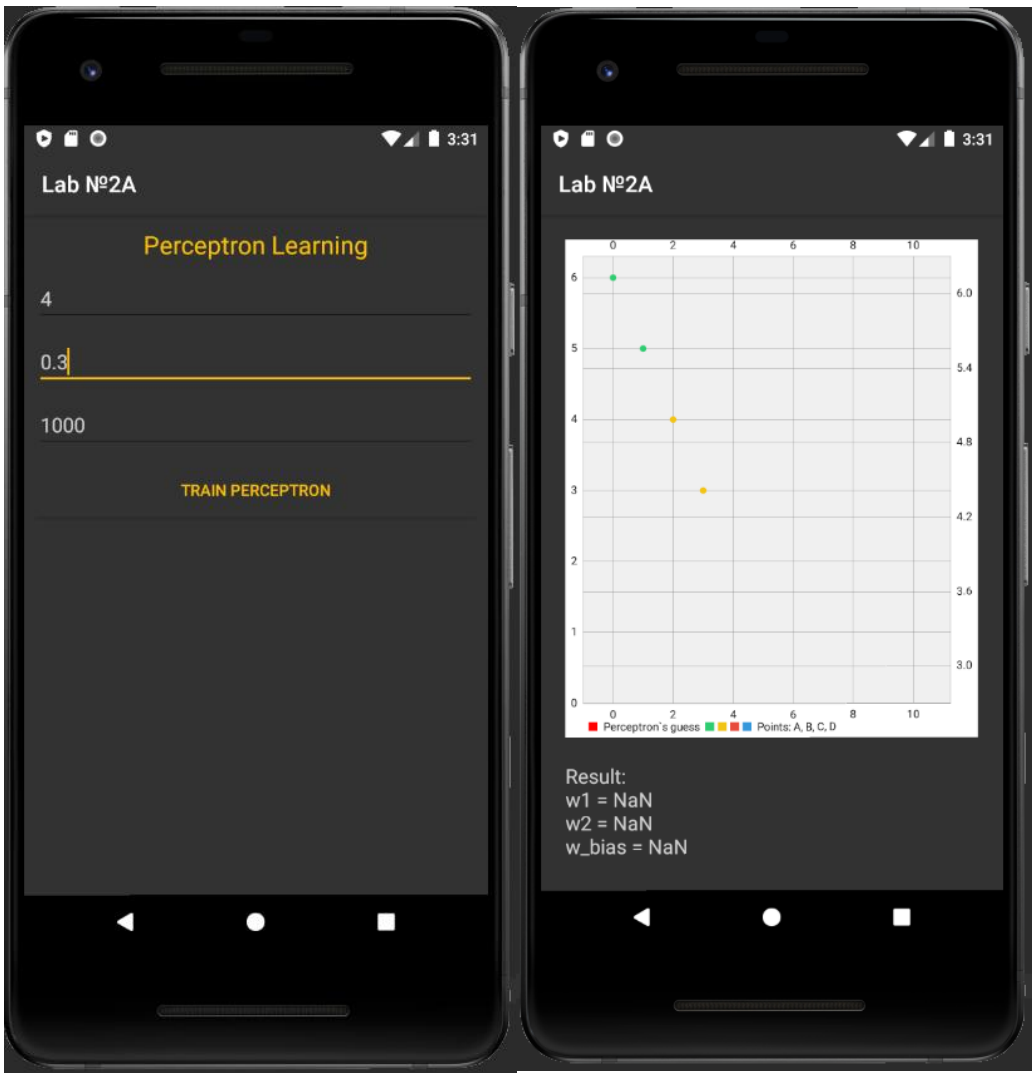

3:23
Lab №2A
Perceptron Learning
4
0.05
100
TRAIN PERCEPTRON


3:23
Lab №2A
Perceptron`s guess
Points: A, B, C, D
Result:
w1 = -0.10000000000000009
w2 = 1.0
w_bias = 0.14999999999999997

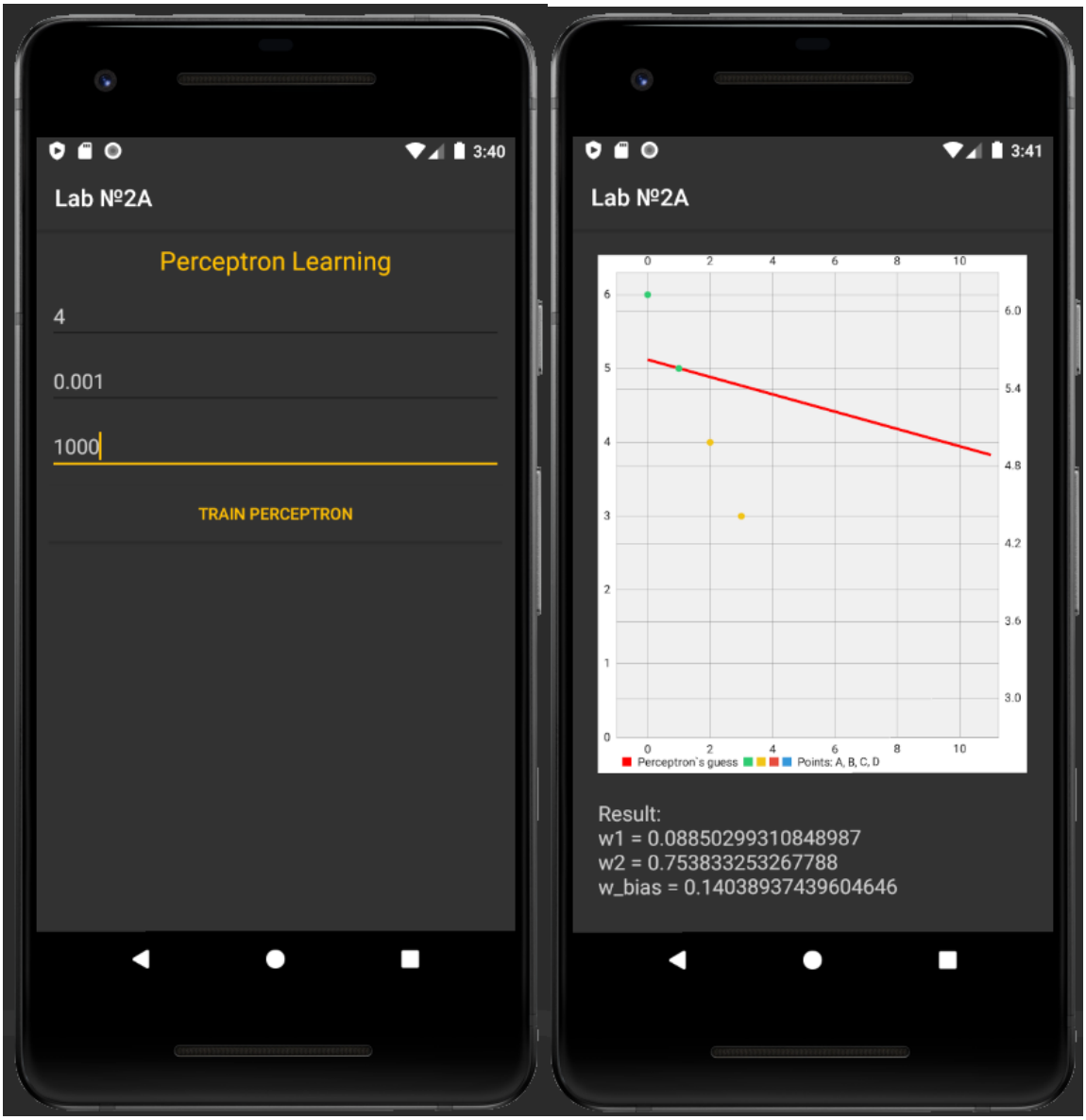

Perceptron`s guess
Points: A, B, C, D
Result:
w1 = -0.10000000000000009
w2 = 1.0
w_bias = 0.14999999999999997

Невдалі тренування




3:31
Lab №2A
Perceptron Learning
4
0.3
1000
TRAIN PERCEPTRON
3:31
Lab №2A
0
2
4
6
8
10
6.0
5.4
4.8
4.2
3.6
3.0
Perceptron`s guess
Points: A, B, C, D
Result:
w1 = NaN
w2 = NaN
w_bias = NaN


3:37
Lab №2A
Perceptron Learning
4
0.01
1000
TRAIN PERCEPTRON
3:39
Lab №2A
Perceptron`s guess
Points: A, B, C, D
Result:
w1 = 0.0754343198007248
w2 = 0.7571599923030621
w_bias = 0.1387657186839643



Висновок

У ході лабораторної роботи я ознайомився з найпростішою реалізацією нейронної мережі за допомогою моделі перцептрона. В даній лабораторній роботі нейронна мережа являє собою фактично один формальний нейрон або ж одношаровий перцептрон, який є окремим випадком класичного перцептрону.

Цей нейрон має 3 входи: X та Y координати точки та зміщення, що завжди ріне одиниці. Кожному з входів відповідає своя вага, яку ми будемо змінювати в ході навчання нейрону. При ініціалізації нейрона його вага рівна нулям. В якості функції активації цей нейрон має порогову функцію з порогом, який задасть користувач – в нашому випадку число 4.

Алгоритмом навчання нейрону є так зване навчання з вчителем – коли наш датасет відразу містить правильну відповідь, що має бути отримана на виході нейрону. Принцип навчання наступний: нейрон рахує лінійну комбінацію з входів

та ваги, це число подається на функцію активації і вона повертає значення, в даному випадку “A” або “B”, що є гіпотезою нейрону – те, що він “вважає” на даний момент. Далі ми порівнюємо гіпотезу нейрону і дані, що ми очікуємо отримати на виході, які також являють собою “A” або “B”. Якщо відповідь нейрону співпадає з очікуваним результатом, то ми нічого не робимо. Якщо ж відповідь не співпала – змінюємо ваги нейрону за дельта правилом та пробуємо знову.

Проаналізувавши роботу даної нейронної мережі стало зрозуміло, що мережа навчається як слід, коли швидкість навчання виставлена як 0.1 або 0.05, а кількість ітерацій рівна 100 – цього більш ніж достатньо, навіть 15-20 ітерацій вистачає, бо після цього числа ітерацій жодних змін до ваги не надходить. Адже як було зазначено раніше, вага входів нейрону змінюється лише коли нейрон “помиляється”. При інших запропонованих значеннях для навчання мережа не може досягти значень ваги, що задовольняла би систему. Це відбувається в силу занадто великої або занадто малої швидкості навчання і в таких випадках навіть 1000 ітерацій не дає потрібного результату.

Підтвердження зробленим висновкам можна спостерігати на представлених графіках в мобільному додатку, що відображають гіпотезу нейрону, щодо розподілу даних за результатами навчання.