

**Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут ім. 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
Мазур С. В.**

**Перевірив:
асистент Регіда П.Г.**

Київ 2021

Основні теоретичні відомості

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 Перцептрон 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:

$$x1 * W1 + x2 * W2 = y$$

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:

$$W1(i+1) = W1(i) + \delta * x1$$

$$W2(i+1) = W2(i) + \delta * x2 \text{ де } i - \text{ крок, або ітерація алгоритму.}$$

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.

$$\text{Далі, рахуємо } \Delta: \Delta = 4 - 0 = 4$$

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

$$W1 = 0 + 4 * 1 * 0,1 = 0,4 \quad W2 = 0 + 4 * 5 * 0,1 = 2$$

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.

$\Delta = -5$, спростуємо результат для прикладу.

$$W1 = 0,4 + 5 * 2 * 0,1 = -0,6 \quad W2 = 2 - 5 * 4 * 0,1 = 0$$

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.

$$W2 = 0 + 5 * 5 * 0,1 = 2,5$$

По такому самому принципу рахуємо значення ваги для наступних ітерацій, поки не отримаємо значення, які задовольняють вхідним даним.

На восьмій ітерації отримуємо значення ваги $W1 = -1,8$ та $W2 = 1,5$.

$5,7 = -1,8 * 1 + 1,5 * 5$, 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

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.

Лістинг програми

Файл з класом

```
const activations = {  
  default: (x) => x,  
  // ReLu: (x) => Math.max(0, x),  
  // Sigmoid: (x) => 1 / (1 + Math.exp(-x)),  
}
```

```
class Perceptron {  
  weights = [0, 0];  
  accuracy = 0;  
  
  threshold = 1;  
  learningRate = .1;  
  
  activation = activations.default;  
  bias = 1;  
  error = 1 / 1e6;
```

```
constructor({ threshold, learningRate }) {  
    Object.assign(this, { threshold, learningRate });  
}
```

```
guess(point) {  
    return this.predict(point) > this.threshold  
}
```

```
predict(point) {  
    return this.activation(this.sum(point))  
}
```

```
train(points) {  
    let success = true  
    points.forEach((point, i) => {  
        success = this.adjustWeights(point, i) && success  
    })  
  
    return success  
}
```

```
learn(points, deadline = 100) {  
    while (true) {
```

```
if (deadline / points.length <= 0 || this.train(points))  
    break;  
deadline--  
}
```

```
return [W1 = ${this.weights[0]} \nW2= ${this.weights[1]}  
\nAccuracy = ${this.accuracy}]  
}
```

```
adjustWeights(point, i) {  
    const delta = this.delta(this.predict(point));  
  
    if (Math.abs(delta) < this.error || delta * Math.pow(-1, i) < 0)  
        return true
```

```
this.weights =  
    this.weights.map((w, i) => {  
        return w + delta * point[i] * this.learningRate  
    });
```

```
this.accuracy = 1 - delta
```

```
return false
```

```
}
```

```
sum(point) {
```

```
  return point.reduce((sum, x, i) => sum + x * this.weights[i], 0) +  
  this.bias
```

```
}
```

```
delta(y) {
```

```
  return this.threshold - y
```

```
}
```

```
}
```

```
export default Perceptron;
```

Основний файл програми з мобільним інтерфейсом

```
import React, { useState } from 'react';
```

```
import { StyleSheet, Text, View, SafeAreaView, Button } from 'react-  
native';
```

```
import RNPickerSelect from 'react-native-picker-select';
```

```
import Perceptron from './src/Perceptron';
```

```
export default function App() {
```

```
  const [learningRate, setLearningRate] = useState(0.001);
```

```
const [deadline, setDeadline] = useState(100);
```

```
const [result, setResult] = useState();
```

```
const [time, setTime] = useState();
```

```
return (
```

```
  <SafeAreaView style={{ flex: 1, alignItems: 'center' }}>
```

```
    <RNPickerSelect
```

```
      style={pickerSelectStyles}
```

```
      onChange={(value) => setLearningRate(value)}
```

```
      placeholder={{ label: 'Оберіть швидкість навчання', value: null }}
```

```
      items={[
```

```
        { label: '0.001', value: 0.001 },
```

```
        { label: '0.01', value: 0.01 },
```

```
        { label: '0.05', value: 0.05 },
```

```
        { label: '0.1', value: 0.1 },
```

```
        { label: '0.2', value: 0.2 },
```

```
        { label: '0.3', value: 0.3 },
```

```
      ]}
```

```
    />
```

```
    <RNPickerSelect
```

```
      style={pickerSelectStyles}
```

```
      onChange={(value) => setDeadline(value)}
```



```

placeholder={{ label: 'Оберіть дедлайн', value: null }}

items={[
  { label: '100', value: 100 },
  { label: '200', value: 200 },
  { label: '500', value: 500 },
  { label: '1000', value: 1000 },

]}

/>
<Text style={styles.result}>
  {[result, time]}
</Text>
<View style={styles.btn}>
  <Button
    title="Learn"
    color="#fff"
    onPress={() => {
      const p = new Perceptron({ threshold: 4, learningRate })
      let start = performance.now();
      setResult(`${p.learn([[0, 6], [3, 3], [1, 5], [2, 4]], deadline)})`
      let end = performance.now();
      setTime(`\nTime= ${end - start}`)
    }}
  >

```

```
/>
```

```
</View>
```

```
</SafeAreaView>
```

```
);
```

```
};
```

```
const styles = StyleSheet.create({
```

```
  result: {
```

```
    top: 100,
```

```
    lineHeight: 30,
```

```
    alignSelf: 'center',
```

```
    fontSize: 16,
```

```
  },
```

```
  btn: {
```

```
    justifyContent: 'center',
```

```
    alignItems: 'center',
```

```
    alignSelf: 'center',
```

```
    top: 200,
```

```
    height: 50,
```

```
    width: 150,
```

```
    backgroundColor: 'black',
```

```
    },  
  });  
  
const pickerSelectStyles = StyleSheet.create({  
  inputIOS: {  
    top: 20,  
    width: '90%',  
    alignSelf: 'center',  
    marginVertical: 10,  
    fontSize: 16,  
    paddingVertical: 12,  
    paddingHorizontal: 10,  
    borderWidth: 2,  
    borderColor: 'black',  
    borderRadius: 6,  
    color: 'black',  
  },  
  inputAndroid: {  
  
  },  
});
```

Результат роботи програми

Оберіть швидкість навчання

0.001

Оберіть дедлайн

1000

Learn

$W1 = 0.06926817725729596$

$W2 = 0.5861461662487997$

Accuracy = 0.999998982034183

Time= 0.6060280203819275

Learn

Висновки

Під час виконання лабораторної роботи було досліджено принципи машинного навчання за допомогою математичної моделі сприйняття інформації Перцептрон (Perceptron). Було реалізовано Перцептрон згідно умов завдання у вигляді мобільного додатку за допомогою фреймворку React Native та Expo. Програма реалізує роботу Перцептрона та виводить на екран отримані значення.