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 №3.1-3.3

з дисципліни
«Інтелектуальні вбудовані системи»

на тему
“ДОСЛІДЖЕННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ ", “РЕАЛІЗАЦІЯ ЗАДАЧІ РОЗКЛАДАННЯ ЧИСЛА НА ПРОСТІ МНОЖНИКИ”, “ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ. МОДЕЛЬ PERCEPTRON”

Виконав:

студент групи ІП-83

Кухаренко Олександр Олександрович
номер залікової книжки: 8312

Перевірив:

ас. Регіда П. Г.

Київ 2021

Завдання:

1. Розробити програма для факторизації заданого числа методом Ферма. Реалізувати користувацький інтерфейс з можливістю вводу даних.
2. 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.
3. Налаштувати генетичний алгоритм для знаходження цілих коренів діофантового рівняння ax1+bx2+cx3+dx4=y. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрат часу на розрахунки.

Лістинг програми:

1)

```
import 'dart:math';

class FermatFactor {
  double n;
  FermatFactor(this.n);

  calculate() {
    int foundIn = 1;

    if (n <= 0) {
      return [
        [n, null],
        foundIn
      ];
    }

    if (!(n % 2 == 1)) {
      return [
        [n / 2, 2],
        foundIn
      ];
    }

    var a = sqrt(n).ceil();
```

```
    var b = 0;
    for (; foundIn > 0; foundIn++) {
      var c = a * a - n;
      b = sqrt(c).floor();

      if (b * b == c)
        break;
      else
        a += 1;
    }

    return [
      [a - b, a + b],
      foundIn
    ];
  }
}
```

2)

```
import 'dart:math';

class Genetics {
  getRandomInt(int max) {
    Random random = new Random();
    return random.nextInt(max);
  }

  getStartPopulation(int pSize, int gSize, int maxGene) {
    return new List<int>.filled(pSize, 0).map((e) =>
        new List<int>.filled(gSize, 0).map((_) => this.getRandomInt(maxGene)));
  }

  int fitnes(var genes, var d, int y) {
    int index = 0;
    var value = 0;
    genes.forEach((element) {
      value += (element * d[index]);
      index++;
      return value;
    });
    return (y - value).abs();
  }

  calculateProbability(var deltas) {
    var invSumDeltas =
        deltas.fold(0, (value, element) => value + (1 / element));
    return deltas.map((e) => (1 / e) / invSumDeltas);
  }

  weightedRandom(var participants) {
```

```
  Random randomIns = new Random();
  var random = randomIns.nextDouble();
  var result = participants.map((e) {
    var gene = e['gene'];
    var probability = e['probability'];
    random -= probability;
    if (random < 0) {
      return gene;
    }
    return null;
  }).where((e) => e != null);
  return result.length != 0 ? result.toList()[0] : null;
}

roulette(participants, numWins) {
  return new List.filled(numWins, null)
      .map((e) => this.weightedRandom(participants));
}

mixGene(List<dynamic> nodes) {
  var parentA = nodes[0];
  var parentB = nodes[1];
  var mid = (parentA.toList().length / 2).floor();
  return [
    [...parentA.toList().sublist(0, mid), ...parentB.toList().sublist(mid)],
    [...parentB.toList().sublist(0, mid), ...parentA.toList().sublist(mid)]
  ];
}

mutation(var gene, value, double probability) {
  Random randomIns = new Random();
  var r = randomIns.nextDouble();
  var index = this.getRandomInt(gene.length);

  if (probability >= r) {
    int i = -1;
    return gene.map((element) {
      i++;
      return i == index ? value : element;
    });
  }
  return gene;
}

diophantineEquation(
    {List<int> equation, populationSize, maxIterations = 20}) {
  var y = equation.removeLast();
  var maxGene = (y / 2).ceil();
  print('Mx Gene: $maxGene');
  var population =
```

```
      this.getStartPopulation(populationSize, equation.length, maxGene);
    while (maxIterations > 0) {
      maxIterations--;
      var deltas = population.map((gene) => this.fitnes(gene, equation, y));
      var result = deltas.where((d) => d == 0).toList();
      if (result.length != 0) {
        return population.toList()[result[0]];
      }
      var probabilities = this.calculateProbability(deltas).toList();
      var pi = -1;
      var rouletteParticipants = new List();
      population.forEach((e) {
        var map = new Map();
        pi++;
        map['gene'] = e;
        map['probability'] = probabilities[pi];
        rouletteParticipants.add(map);
      });

      population = [];
      for (var i = 0; i < populationSize / 2; i++) {
        var bestGenes = this.roulette(rouletteParticipants, 2);
        var mixedGenes = this.mixGene(bestGenes.toList());
        var mutatedGenes = mixedGenes.map(
            (gene) => this.mutation(gene, this.getRandomInt(maxGene), 0.1));
        population..addAll(mutatedGenes);
      }
    }
    return population;
  }
}
```

3)

```
class Perception {
  int p;
  double r;
  Perception(this.p, this.r);

  double w1 = 0;
  double w2 = 0;

  calculateSignal(point) {
    double x1 = point[0].toDouble();
    double x2 = point[1].toDouble();
    return this.w1 * x1 + w2 * x2;
  }

  getDelta(y) {
```

```
    double delta = this.p - y;
    if (delta > 0) {
      return delta;
    }
    return 0;
  }

  weightAdjustment(point, delta) {
    double x1 = point[0].toDouble();
    double x2 = point[1].toDouble();
    this.w1 += delta * x1 * this.r;
    this.w2 += delta * x2 * this.r;
  }

  learn(input, maxIterations, maxTime) {
    double time = 0;
    int iterations = 0;

    while (maxIterations > iterations && maxTime > time) {
      var startDate = DateTime.now().microsecondsSinceEpoch;
      for (final value in input) {
        var y = this.calculateSignal(value);
        var delta = this.getDelta(y);

        this.weightAdjustment(value, delta);
      }
      var endDate = DateTime.now().microsecondsSinceEpoch;
      time += (endDate - startDate) / 1000;
      iterations++;
    }
    return [w1, w2, time, iterations];
  }
}
```

Отримані результати:

1) https://gyazo.com/fe035bf37005aa3aeb6ffd203d6e4e60
   https://gyazo.com/2de635677275f886975dc6d6625a4160
2) https://gyazo.com/221c590b3b78fdd4843f84bc65fe7f18
   https://gyazo.com/39491af16f0d5aa62146a0447af5003d
3) https://gyazo.com/1bb065d2b4ef6ea57ccb3e356e8863f2
   https://gyazo.com/ec0d9068d92e5f13e45aa4d8a2355434

Висновки:

Під час виконання лабораторних робіт 3.1, 3.2 та 3.3 ми ознайомились з основними принципами розкладання числа на прості множники з використанням різних алгоритмів факторизаці, з принципами машинного навчання за допомогою математичної моделі сприйняття інформації Перцептрон(Perceptron) та з принципами реалізації генетичного алгоритму, вивчення та дослідження особливостей даного алгоритму з використанням засобів моделювання і сучасних програмних оболонок.