|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| https://lh6.googleusercontent.com/QcftzNtI05T0Y6fjdSh1Rr2rt8oqZ1IvnLvbn1jLJ7CCyteVir3k-xBLv4SL1wAgWJsRhmmJSR0UW-RP63_GQenE4vVWv05BRoZTsmIcBccVTnfxwmsnNMvjg599x9SqZd8E3dkd   |  |  |  | | --- | --- | --- | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | | |  | | | |  |
|  |  |

**Институт информационных технологий (ИИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИППО)**

|  |  |
| --- | --- |
| **ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4** | |
| **по дисциплине** | |
| «Программное обеспечение интеллектуальных систем» | |
|  | |
| Выполнил студент группы ИКБО-02-17 | *Верба Д.С.* |
| Принял | *Волков М.Ю.* |

|  |  |  |
| --- | --- | --- |
| Работа выполнена | «\_\_»\_\_\_\_\_\_\_2020 г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2020 г. | *(подпись руководителя)* |

Москва 2020

**Цель:** ознакомиться с работой нейронных сетей на примере создания приложений со сторонними библиотеками Java.

**Задание:**

В рамках лабораторной работы попробуйте решить одну из следующих

задач:

1. Тренировка нейросети;

2. Решение NP-полной задачи.

**Ход работы**

Данный генетический алгоритм поможет нам найти оптимальный путь между узлами в графе для задачи коммивояжера при следующих заданных начальных данных: всего в графе имеется 30 точек.

Алгоритм позволяет найти оптимальный путь между узлами в заданном графе. В качестве одного из критериев оптимизации мы можем задать ожидаемый размер пути.

Для начала подадим на вход данные, которые будут использоваться алгоритмом в последующем решении задачи нахождения оптимального пути в заданном графе. Как было сказано ранее, зададим размер графа равный 30.

Для заполнения длин путей используем метод dist, формирующий граф в виде двумерного массива.   
 Затем установим количество итераций равное 30. Так как мы используем генетический алгоритм, то ограничим размер вычислений на итерацию до 500.

Далее зададим вероятность 0.9 смешения результатов каждых поколений друг на друга, а также вероятность 0.1 передачи результатов между разными генами в поколениях для повышения точности алгоритма.

Как мы видим исходная нейросеть тренируется в тее

Исходный код

**import io.jenetics.\*;**

**import io.jenetics.engine.Codecs;**

**import io.jenetics.engine.Engine;**

**import io.jenetics.engine.EvolutionStatistics;**

**import java.util.stream.IntStream;**

**import static io.jenetics.engine.EvolutionResult.*toBestPhenotype*;**

**import static io.jenetics.engine.Limits.*bySteadyFitness*;**

**import static java.lang.Math.\*;**

**public class TravelingSalesmanLab4 {**

**private static final int *NODES* = 100;**

**private static final double[][] *ALPHA* = *matrix*(*NODES*);**

**private static final int *iterations* = 500;**

**private static final int *sizePerIteration* = 30;**

**private static double[][] matrix(int stops) {**

**final double radius = 30.0;**

**double[][] matrix = new double[stops][stops];**

**for (int i = 0; i < stops; ++i) {**

**for (int j = 0; j < stops; ++j) {**

**matrix[i][j] = *chord*(stops, *abs*(i - j), radius);**

**}**

**}**

**return matrix;**

**}**

**private static double chord(int stops, int i, double r) {**

**return 2.0 \* r \* *abs*(*sin*(*PI* \* i / stops));**

**}**

**private static double dist(final int[] path) {**

**return IntStream.*range*(0, *NODES*)**

**.mapToDouble(i -> *ALPHA*[path[i]][path[(i + 1) % *NODES*]])**

**.sum();**

**}**

**public static void main(String[] args) {**

**final Engine<EnumGene<Integer>, Double> engine = Engine.*builder*(TravelingSalesmanLab4::*dist*, Codecs.*ofPermutation*(*NODES*))**

**.optimize(Optimize.*MINIMUM*)**

**.maximalPhenotypeAge(5)**

**.populationSize(*sizePerIteration*)**

**.alterers(new SwapMutator<>(0.1), new PartiallyMatchedCrossover<>(0.9))**

**.build();**

**final EvolutionStatistics<Double, ?> statistics = EvolutionStatistics.*ofNumber*();**

**final Phenotype<EnumGene<Integer>, Double> best = engine.stream()**

**.limit(*bySteadyFitness*(*iterations*))**

**.limit(500)**

**.peek(statistics)**

**.collect(*toBestPhenotype*());**

**System.*out*.println(best.toString().toCharArray());**

**}**

**}**

Получим три результата работы для дальнейшей обработки их в нейронной сети. На рисунке 1 представлен один из результатов работы генетического алгоритма



Затем используем полученные значения в качестве входных данных для нашей нейронной сети как показано на рисунке 2



рисунок 2 - использование результата генетического алгоритма в качестве входных данных для нейронной сети

Затем осуществим добавление нейронный слоев. После этого добавим в каждый слой необходимое количество нейронной. Из изначально поставленных данных количество входных нейронов будет ограничено 10, а количество выходных будет ограничено 1

DataSet ds = new DataSet(10, 1);

ds.add(rOne);

NeuralNetwork ann = new NeuralNetwork();

Layer inputLayer = new Layer();

Layer outputLayer = new Layer();

Layer hiddenLayer = new Layer();

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

inputLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

hiddenLayer.addNeuron(new Neuron());

outputLayer.addNeuron(new Neuron());

ann.addLayer(0, inputLayer);

ann.addLayer(1, hiddenLayer);

ann.addLayer(2, outputLayer);

ConnectionFactory.*fullConnect*(ann.getLayerAt(0), ann.getLayerAt(1));

ConnectionFactory.*fullConnect*(ann.getLayerAt(1), ann.getLayerAt(2));

ConnectionFactory.*fullConnect*(ann.getLayerAt(0), ann.getLayerAt(ann.getLayersCount() - 1), false);

ann.setInputNeurons(inputLayer.getNeurons());

ann.setOutputNeurons(ann.getLayerAt(ann.getLayersCount() - 1).getNeurons());

BackPropagation backPropagation = new BackPropagation();

backPropagation.setMaxIterations(500);

ann.setLearningRule(backPropagation);

ann.setInput(1, 0, 9, 8, 7, 6, 5, 4, 3, 2);

ann.learn(ds);

ann.calculate();

Ссылка на GitHub: https://github.com/GooDiVer/sem7/tree/main/pois/lab4

**Вывод**

В ходе выполнения лабораторной работы мы научились строить нейронную сеть на основе библиотеки neuroph, ознакомились с работой нейронной сети, а также применили знания библиотеки jenetics на практике.

**Список литературы**

1. Леонид Гладков. Генетические алгоритмы / - М.: Отдельное издание 2010

2. Петер Флах. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / - М.: Изд-во ДМК Пресс, 2019

3. Материалы лекций