Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

Отчет

по лабораторной работе №4 по теме

«Генетический алгоритм»

по дисциплине

«Математические основы искусственного интеллекта»

|  |  |
| --- | --- |
| Выполнил:  студенты гр. Б06-191-1 | Мусин Р.И.  Чапаева А.С. |
| Принял: | Коробейников А.В. |

Ижевск 2018

СОДЕРЖАНИЕ ЗАДАНИЯ

В рамках дисциплины искусственного интеллекта познакомиться с принципами работы генетического алгоритма на примере обучения нейросети распознаванию букв.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Генетический алгоритм — это в первую очередь эволюционный алгоритм, другими словами, его основная идея— скрещивание (комбинирование). Путем перебора и, самое главное, отбора получается правильная «комбинация».

Алгоритм делится на три этапа:

1. Скрещивание
2. Селекция (отбор)
3. Формирования нового поколения

Если результат не устраивает, эти шаги повторяются до тех пор, пока результат не начнет удовлетворять или произойдет одно из ниже перечисленных условий:

1. Количество поколений (циклов) достигнет заранее выбранного максимума
2. Исчерпано время на мутацию

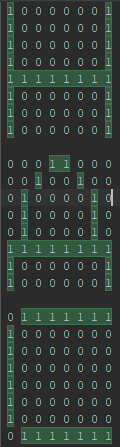
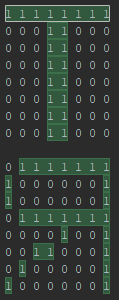
Подробнее о шагах:

1. Создание новой популяции. На этом шаге создается начальная популяция, которая, вполне возможно, окажется не кошерной, однако велика вероятность, что алгоритм эту проблему исправит. Главное, чтобы они соответствовали «формату» и были «приспособлены к размножению».
2. Размножение. Ну тут все как у людей, для получения потомка требуется два родителя. Главное, чтобы потомок (ребенок) мог унаследовать у родителей их черты. При это размножаются все, а не только выжившие (эта фраза особенно абсурдна, но так как у нас все в сферическом вакууме, то можно все), в противном случае выделится один альфа самец, гены которого перекроют всех остальных, а нам это принципиально не приемлемо.
3. Мутации. Мутации схожи с размножением, из мутантов выбирают некое количество особей и изменяют их в соответствии с заранее определенными операциями.
4. Отбор. Тут начинается самое сладкое, мы начинаем выбирать из популяции долю тех, кто «пойдет дальше». При этом долю «выживших» после нашего отбора мы определяем заранее руками, указывая в виде параметра. Как ни печально, остальные особи должны погибнуть.

ЛИСТИНГ ПРОГРАММЫ

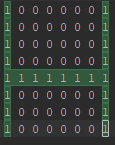
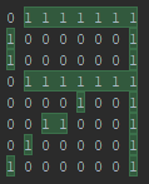
**package** main.java;  
  
**import** java.io.IOException;  
**import** java.nio.file.Paths;  
**import** java.util.Arrays;  
**import** java.util.Comparator;  
**import** java.util.Random;  
**import** java.util.Scanner;  
  
**public class** Main {  
  
 **private static final char**[] ***ALPHABET*** = **"НАСТЯ"**.toCharArray();  
 **private static final int *LETTER\_COUNT*** = ***ALPHABET***.**length**;  
 **private static final int *LETTER\_SIZE*** = 8 \* 8;  
 **private static final int *GENERATION\_SIZE*** = 100;  
 **private static final** Letter[] ***LETTERS*** = **new** Letter[***LETTER\_COUNT***];  
 **private static final** Random ***rnd*** = **new** Random(239);  
  
 **public static void** main(String[] args) **throws** IOException {  
 Scanner lettersScanner = **new** Scanner(Paths.*get*(**"teaching.txt"**));  
 Generation[] generations = **new** Generation[***LETTER\_COUNT***];  
 **for** (**int** i = 0; i < ***LETTER\_COUNT***; i++) {  
 Letter letter = Letter.*read*(***ALPHABET***[i], lettersScanner);  
 ***LETTERS***[i] = letter;  
 generations[i] = Generation.*generate*(letter);  
 }  
  
 **for** (**int** i = 0; i < ***LETTER\_COUNT***; i++)  
 generations[i] = generations[i].evaluate();  
  
 Letter letterToTest = Letter.*read*(**'\*'**, **new** Scanner(Paths.*get*(**"input.txt"**)));  
 **for** (Letter letter : ***LETTERS***)  
 **if** (letterToTest.checkWeights(letter.**weights**))  
 System.***out***.println(**"Это буква "** + letter.**name**);  
 }  
  
 **static class** Letter {  
 **final char name**;  
 **final int**[] **representation**;  
 **double**[] **weights**;  
  
 Letter(**char** name, **int**[] representation) {  
 **this**.**name** = name;  
 **this**.**representation** = representation;  
 }  
  
 **static** Letter read(**char** name, Scanner in) {  
 **int**[] representation = **new int**[***LETTER\_SIZE***];  
 **for** (**int** i = 0; i < ***LETTER\_SIZE***; i++) representation[i] = in.nextInt();  
 **return new** Letter(name, representation);  
 }  
  
 **boolean** checkWeights(**double**[] weights) {  
 **double** sum = 0;  
 **for** (**int** i = 0; i < ***LETTER\_SIZE***; i++) sum += **representation**[i] \* weights[i];  
 **return** sum >= 0;  
 }  
 }  
  
 **static class** Generation {  
 **final** Letter **letter**;  
 **final double**[][] **variants**;  
  
 Generation(Letter letter, **double**[][] variants) {  
 **this**.**letter** = letter;  
 **this**.**variants** = variants;  
 }  
  
 **static** Generation generate(Letter letter) {  
 **double**[][] table = **new double**[***GENERATION\_SIZE***][***LETTER\_SIZE***];  
 **for** (**int** i = 0; i < ***GENERATION\_SIZE***; i++)  
 **for** (**int** j = 0; j < ***LETTER\_SIZE***; j++)  
 table[i][j] = *randomWeight*();  
 **return new** Generation(letter, table);  
 }  
  
 **static** Generation evaluate(Generation generation) {  
 **for** (**double**[] weights : generation.**variants**) {  
 **boolean** ok = **true**;  
 **for** (Letter letter : ***LETTERS***) {  
 **if** (letter.checkWeights(weights) != (generation.**letter** == letter)) {  
 ok = **false**;  
 **break**;  
 }  
 }  
 **if** (ok) {  
 generation.**letter**.**weights** = weights;  
 **return** generation;  
 }  
 }  
 **return** *evaluate*(generation.next());  
 }  
  
 **static double** randomWeight() {  
 **return *rnd***.nextDouble() \* 10 - 5;  
 }  
  
 **void** sort() {  
 Arrays.*sort*(**variants**, Comparator.*comparingInt*(weights -> {  
 **int** countBad = 0;  
 **for** (Letter cur : ***LETTERS***)  
 **if** ((**letter** == cur) != (cur.checkWeights(weights)))  
 countBad++;  
 **return** countBad;  
 }));  
 }  
  
 **void** mutate(**int** at) {  
 **variants**[at][***rnd***.nextInt(***LETTER\_SIZE***)] = *randomWeight*();  
 }  
  
 Generation next() {  
 Generation intermediateGeneration = **new** Generation(**letter**, **new double**[***GENERATION\_SIZE*** \* 2][]);  
 */\* Скрещивание\*/* **for** (**int** i = 0; i < ***GENERATION\_SIZE***; i++)  
 intermediateGeneration.**variants**[i] = crossWeights(  
 **variants**[***rnd***.nextInt(***GENERATION\_SIZE***)],  
 **variants**[***rnd***.nextInt(***GENERATION\_SIZE***)]);  
 **for** (**int** i = 0; i < ***GENERATION\_SIZE***; i++)  
 intermediateGeneration.**variants**[***GENERATION\_SIZE*** + i] = **variants**[i].clone();  
 */\* Мутации \*/* **for** (**int** i = 0; i < ***GENERATION\_SIZE*** / 4; i++)  
 intermediateGeneration.mutate(***rnd***.nextInt(***GENERATION\_SIZE***));  
 intermediateGeneration.sort();  
 **return new** Generation(**letter**, Arrays.*copyOfRange*(intermediateGeneration.**variants**, 0, ***GENERATION\_SIZE***));  
 }  
  
 **double**[] crossWeights(**double**[] weights1, **double**[] weights2) {  
 **int** length = weights1.**length**;  
 **double**[] result = **new double**[length];  
 **int** position = ***rnd***.nextInt(length + 1);  
 System.*arraycopy*(weights1, 0, result, 0, position);  
 System.*arraycopy*(weights2, position, result, position, length - position);  
 **return** result;  
 }  
  
 Generation evaluate() {  
 **return** *evaluate*(**this**);  
 }  
 }  
}

ПРИМЕРЫ ДЛЯ ОБУЧЕНИЯ

ПРИМЕР РАБОТЫ

Входные данные и результат работы:

ВЫВОД

В ходе работы были изучены принципы работы генетического алгоритма на примере распознавания букв при помощи персептронов. На языке высокого уровня была написана программа, позволяющая распознавать буквы.