Machine learning. Genetic Algorithm

Vladimir Demin

Brest State Technical University
spas.work@gmail.com

Abstract

In this project, we test some machine learning techniques by observing how an agent explores a gridworld with a cliff. Starting with a random walk we compare it with an agent behaviors learned with *Genetic Algorithm (GA)* and Renforcement learning.

1 INTRODUCTION

Borrowing the idea from biological evolution, we can be solved certain types of problems by an algorithm which we call *GA*. In this case we express the problem we want to solve by a vector which we call *chromosome*. Chromosome is made up a number of *genes*. At the beginning we create a population of, say 100, chromosomes at random. They are not good solutions at all because they are randomly created. But some are a little better than others. So we pick up two chromosomes such that better chromosomes are more likely to be chosen. Here, lets choose them at random from better half of the population. This is called truncate selection. Also, we used selection, called uniform-crossover. Where we choose genes one by one either from parents at random.

Then by repeating this procedure, we create the next *generation*. The population of the nextgeneration includes same number of chromosomes in the previous population. Thus we can evolve the first random population of chromosomes generation by generation. We can expect those chromosomes' perfomance become better and better.

We also give a *mutation* to introduce new genes. This is avoid for individuals in the population to be trapped into a *local minimum*. The probability for mutation to occur is small - typically 1/number-of-genes.

2 EXPERIMENT

For our problem we use uniform-crossover selection. We make 100x100 gridworld with cliff. We make 100 vectors with 500 genes. We randomly create this 100 vectors by random walking algorithm. After we evaluate quality of vectors and take 50 only 'good' vectors (it means that that agent didn't die). After we make 100 childs by randomly trancate of good vectors. Repeate this actions until one of the vector will success.

We change length of our vectors from 500 downto 102 and take good result for all of them. We descibe dependence between length of our vectors and quantity generations.

3 RESULT

We submit our result on next diagram:

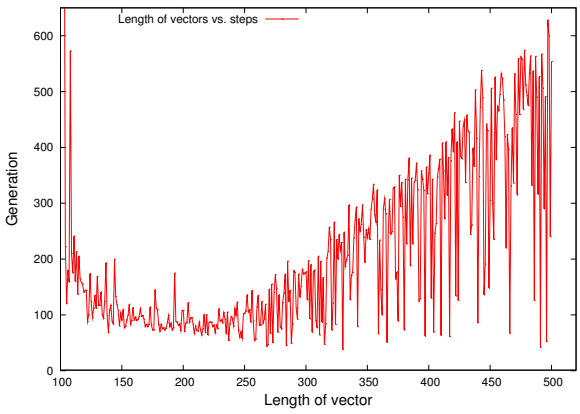

Figure 1: length of our vectors vs. quantity generations.

Also we have two success way. Next diagram show good way with 102 length vector:

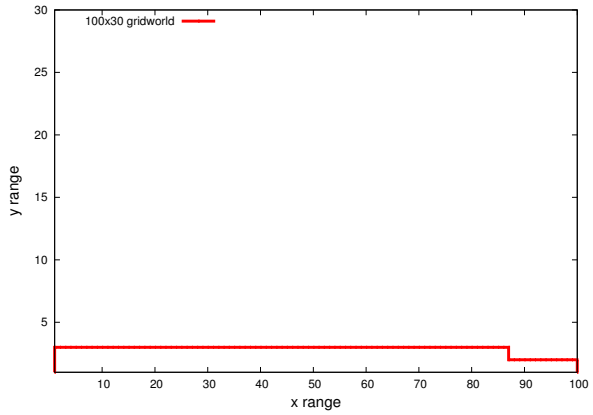

Figure 2: Success way where vector length is 102

And way where length of vectors is 500:

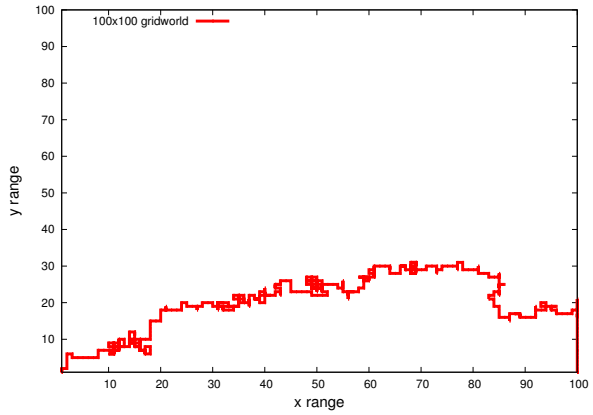

Figure 3: Success way where vector length is 500

3.1 CONSULTION

In our experiments we seen that GA have good result on machine learning and may odapt to resolv problem of walkin real robot. It may used to solve some problems like labyrinth walking.

4 APPENDIX: C implementation of the GA algorithm

Listing 1: main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include "main.h"
5
6  #define LIFES  100
7
8  #define TRUE  1
9  #define FALSE 0
10
11 #define TOP    0
12 #define BOTTOM 1
13 #define LEFT   2
14 #define RIGHT  3
15
16 int lenn;
17 int parent_vect[100][LEN];
18 int childs_vect[50][LEN];
19 int value[100], ind[100];
20 int succecc, die, roboX, roboY;
21
22 int main() {
23     int val = -100;
24     int i, step, out;
25     FILE *f;
26     FILE *rep;
27     f = fopen("way.txt", "w");
28     rep = fopen("rep.txt", "w");
29     srand(time(NULL));
30
31
32
33     for (i = 500; i >= 102; i--) { /* here you may choose what length of vectors you nee
34         lenn = i;
35
36         do {
37             step = 0;
38             val = 2 * lenn;
39             out = FALSE;
40             clean();
41             initial_generation();
42             sort();
43             while (val != 0 && out != TRUE) {
44                 if (next_generation())
45                     out = TRUE;
46                 evalute_value();
47                 val = sort();
48                 fflush(stdout);
49                 step++;
50                 //printf("%d\n", val);
51                 if (step > 10000)
52                     out = TRUE;
```

```
53              }
54          } while(out != FALSE && val != 0);
55
56          fprintf(rep, "%d, %d\n", lenn, step);
57          printf("%d, %d\n", lenn, step);
58          fflush(stdout);
59      }
60
61      roboX = 1; roboY = 1;
62      die = FALSE; succecc = FALSE;
63      for (i = 0; i < LEN; i++){
64          do_shag(childs_vect[ind[0]][i]);
65          fprintf(f, "%d, %d\n", roboX, roboY);
66      }
67      fclose(f);
68
69      return 0;
70  }
```

Listing 2: main.h

```
1  #ifndef MAIN_H_INCLUDED
2  #define MAIN_H_INCLUDED
3
4  #define POINTS 500
5  #define LIFES  100
6
7  #define TRUE  1
8  #define FALSE 0
9
10 #define TOP    0
11 #define BOTTOM 1
12 #define LEFT   2
13 #define RIGHT  3
14
15 #define LEN 500
16
17 extern int lenn;
18 extern int parent_vect[100][LEN];
19 extern int childs_vect[50][LEN];
20 extern int value[100], ind[100];
21 extern int succecc, die, roboX, roboY;
22
23 int move() {
24     return rand() % 4;
25 }
26
27 int take_value(int x, int y) {
28     return (100 - x) + (y - 1);
29 }
30
31 void clean() {
32     int i, j;
33
34     for (j = 0; j < lenn; j++) {
35         for (i = 0; i < 100; i++) {
```

```
36              parent_vect[i][j] = 0;
37              value[i] = 0;
38              ind[i] = 0;
39          }
40          for (i = 0; i < 50; i++)
41              childs_vect[i][j] = 0;
42      }
43  }
44
45  void do_shag(int shag) {
46      int k = 100;
47              switch (shag) {
48              case TOP:
49                  if (roboY < k)
50                  roboY++;
51              break;
52
53              case BOTTOM:
54                  if (roboY > 1) {
55                      roboY--;
56                      if (roboY == 1) {
57                          if (roboX > 1 && roboX < k) {
58                              die = TRUE;
59                          }else if (roboX == k)
60                      succecc = TRUE;
61                      }
62                  }
63              break;
64
65              case LEFT:
66                  if (roboX > 1)
67                  roboX--;
68              break;
69
70              case RIGHT:
71                  if (roboX < k) {
72                      roboX++;
73                      if (roboY == 1)
74                          die = TRUE;
75                  }
76              break;
77
78              default:
79                  break;
80              }
81  }
82
83
84  void initial_generation() {
85      int i, age, shag;
86
87      for (i = 0; i < 100; i++) {
88
89          age = 0; die = FALSE; succecc = FALSE;
90          roboX = 1; roboY = 1;
```

```
91          while (age <= lenn && die == FALSE && succecc == FALSE) {
92              shag = move();
93              parent_vect[i][age] = shag;
94              do_shag(shag);
95              age++;
96          }
97
98          if (die == FALSE && succecc == FALSE)
99              value[i] = take_value(roboX, roboY);
100         else if (die == TRUE)
101             value[i] = 10000;
102         else
103             value[i] = 0;
104     }
105 }
106
107 int sort() {
108     int i, j, tmp;
109
110     for (i = 0; i < 100; i++)
111         ind[i] = i;
112
113     for (i = 0; i < 100; i++)
114         for (j = i + 1; j < 100; j++)
115             if (value[ind[i]] > value[ind[j]]) {
116                 tmp = ind[i];
117                 ind[i] = ind[j];
118                 ind[j] = tmp;
119             }
120     return value[0];
121 }
122
123 void mutate(int i) {
124     int mutate_gene;
125
126     mutate_gene = rand() % lenn;
127     parent_vect[i][mutate_gene] = rand() % 4;
128 }
129
130 int next_generation() {
131     int i, j;
132     int mam, dad, koef;
133     int k, kol;
134
135     for (i = 0; i < 50; i++)
136         for (j = 0; j < lenn; j++)
137             childs_vect[i][j] = parent_vect[ind[i]][j];
138
139     kol = 0;
140     for (k = 0; k < 50; k++) {
141         if (value[k] < 201)
142             kol++;
143     }
144     if (kol < 3)
145         return 1;
```

```
146
147     for (i = 0; i < 100; i++) {
148         do {
149             mam = rand() % 50;
150         } while(value[mam] > 201);
151
152         do {
153             dad = rand() % 50;
154         } while(value[dad] > 201);
155
156         for (j = 0; j < lenn; j++) {
157             koef = rand() % 2;
158             parent_vect[i][j] = childs_vect[mam * (koef) + dad * (1 - koef)][j];
159         }
160
161         mutate(i);
162     }
163     return 0;
164 }
165
166 void evalute_value() {
167     int i, age, shag;
168
169     for (i = 0; i < 100; i++) {
170
171         die = FALSE; succecc = FALSE; age = 0;
172         roboX = 1; roboY = 1;
173         while (age <= lenn && die == FALSE && succecc == FALSE) {
174             shag = parent_vect[i][age];
175             do_shag(shag);
176             age++;
177         }
178
179         if (die == FALSE && succecc == FALSE)
180             value[i] = take_value(roboX, roboY);
181         else if (die == TRUE)
182             value[i] = 10000;
183         else
184             value[i] = 0;
185     }
186 }
187
188 #endif // MAIN_H_INCLUDED
```