Machine learning. Finit state machine

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)*, Renforcement learning and *Finit State Machine (FSM)*.

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.

For the exploration of the our gridworld we use *FSM*. FSM have some states. The number of state should be $2\hat{n}$, that is, like 2, 4, 8, 16, or 32 for the reason, that we might guess from below. Machine have input and output vectors. After we take input for machine, it take to us output, and may change it state.

2 EXPERIMENT

For our gridworld input are fourfold, that are, empty, cliff, sausage or border.

Output is fivefold, that is, either of go forward, turn left, turn right, turn back, or stop (when the agent reaches the goal.)

Assuming 2 state FSM, one example of the transition table will be like:

State	Input	output	next state
0	1	10	0
0	2	11	0
0	3	01	1
0	4	00	1
1	1	10	0
1	2	11	0
1	3	01	1
1	4	00	0

The first 2 column is fixed (automatic depending on the number of state & input)The 3rd and 4th columns are given at random at the beginning.

Our input have states:

```
input
  1      - empty cell
  2      - border
  3      - cliff
  4      - sausage
```

Our output have next instructions:

```
state   optput
  0       11    - go forward
  0       10    - turn left
  0       01    - turn right
  0       00    - turn back
  1       11    - go forward
  1       10    - turn right
  1       01    - turn left
  1       00    - stop
```

Thus we create 100 stupid FSM's at the beginning. Then express them with a chromosome. Our example can be (100110011100110011). Then we using GA and after lots of generations it converges one clever FSM.

For our problem we use uniform-crossover selection. We make 100x100 gridworld with cliff. We make 100 vectors with 28 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.

3 RESULT

We take input vector on this

```
state   optput
  0       11    - go forward
  0       10    - turn left
  0       01    - turn right
  0       00    - turn back
  1       11    - go forward
  1       10    - turn right
  1       01    - turn left
  1       00    - stop
```

and study our "stupid" FSMs with genetic algorithm. One of results of our studing is:

```
State  Input   output   next state
0        1       11         0
0        2       10         0
0        3       01         0
0        4       11         1
1        1       00         1
1        2       00         1
1        3       00         1
1        4       00         1
```

We study our FSMs on the standart 100x100 cliff world, with one cliff with coordinate (2,1) to (99,1). When study is end, we use our "clever" FSM on our "big" world:

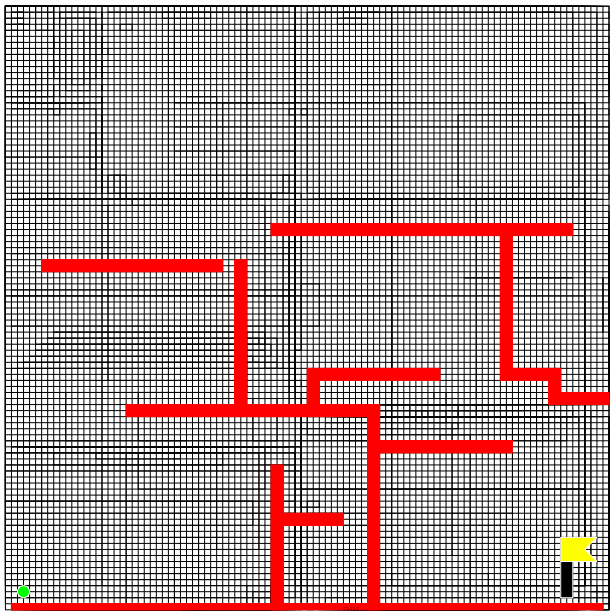

Figure 1: length of our vectors vs. quantity generations.

We make graphic for our aducational proccess:

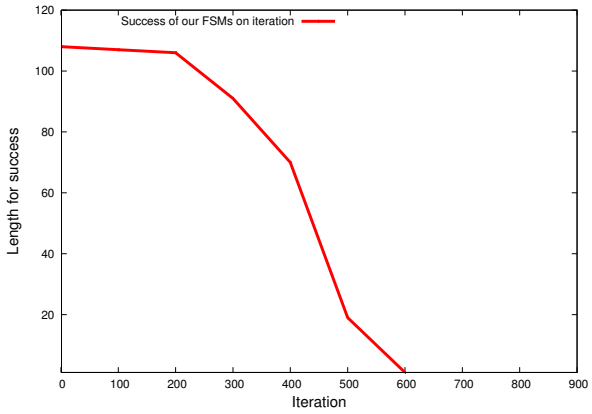

Figure 2: Success of better FSMs

3.1 CONSULTION

In our reseach we find, that if we want to study FSM with GA, we must study it on "simple" world. After study we have good FSM, that we must adapt to owr new "big" world.

4 APPENDIX: C implementation of the FSM algorithm

Listing 1: main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include "FSM.h"
5  #include "GA.h"
6
7  #define LEN 400
8
9
10
11 int whats_happens(int x, int y, struct gridworld *world) {
12     switch (world->world[x][y]) {
13         case BORDER:
14             return BORDER;
15             break;
16         case EMPTY:
17             return EMPTY;
18             break;
19         case SAUSAGE:
20             return SAUSAGE;
21             break;
22         case CLIFF:
23             return CLIFF;
24             break;
25         default:
26             break;
27     }
28     return -1;
29 }
30
31 void Go(struct FSM *agent, struct gridworld *world) {
32     int age = 0;
33     int type;
34
35     do {
36         FSM_step(agent, world);
37         type = whats_happens(agent->roboX, agent->roboY, world);
38         if (type == CLIFF || type == BORDER)
39             agent->die = TRUE;
40         else if (type == SAUSAGE)
41             agent->success = TRUE;
42         age++;
43     } while (agent->success == FALSE && agent->die == FALSE && age < 200);
44
45
46     agent->age = age;
47 }
48
49 int main()
50 {
51     int i,j;
52     char tmp;
```

```
53      int tt = 10000;
54      int epoch = 0, epoch1 = 0;
55      int startX, startY;
56      struct gridworld world;
57      struct FSM agent[100];
58      struct FSM childs[50];
59      FILE *learning;
60  /*
61    Initialization of the gridworld
62  */
63
64      srand(time(NULL));
65
66      FILE *wld;
67      learning = fopen("learning.txt", "w");
68      wld = fopen("world.txt", "r");
69      if (wld == NULL) {
70          printf("Input file not open!");
71          return 1;
72      }
73      fscanf(wld, "%d %d\n", &world.M, &world.N);
74      if (world.M > 1000 || world.M > 1000) {
75          printf("You M or N input data out of range!\n");
76          return 1;
77      }
78      for (i = 0; i < world.M; i++) {
79          for (j = 0; j < world.N; j++) {
80              do {
81                  fscanf(wld, "%c", &tmp);
82              } while (tmp != '*' && tmp != '.' && tmp != '@' && tmp != '0' && tmp != '^');
83              switch (tmp) {
84                  case '*':
85                      world.world[j][world.M - i - 1] = BORDER;
86                      break;
87                  case '.':
88                      world.world[j][world.M - i - 1] = EMPTY;
89                      break;
90                  case '@':
91                      world.world[j][world.M - i - 1] = SAUSAGE;
92                      break;
93                  case '0':
94                      startX = j;
95                      startY = world.M - i - 1;
96                      world.world[j][world.M - i - 1] = EMPTY;
97                      break;
98                  case '^':
99                      world.world[j][world.M - i - 1] = CLIFF;
100                     break;
101                 default:
102                     printf("Reading error! You must check input data!\n");
103                     break;
104             }
105         }
106         fscanf(wld, "\n");
107     }
```

```
108     fclose (wld);
109 /*
110   End of initialization of the gridworld
111 */
112
113     for (i = 0; i < 100; i++) {
114         init_FSM(&agent[i]);
115         agent[i].roboX = startX;
116         agent[i].roboY = startY;
117     }
118
119     for (i = 0; i < 100; i++) {
120         Go(&agent[i], &world);
121         value[i] = evalute(&agent[i]);
122     }
123     sort();
124     do {
125         make_childs(agent, childs);
126         generate_parents(childs, agent);
127         reset_parents(agent, startX, startY);
128         for (i = 0; i < 100; i++) {
129             Go(&agent[i], &world);
130             value[i] = evalute(&agent[i]);
131         }
132         sort();
133         epoch++;
134         fprintf(learning, "%d, %d\n", epoch, value[0]);
135         fflush(learning);
136     } while (value[0] != 0);
137
138     printf("Count of iterations: %d\n", epoch);
139
140     printf("Clever FSM:\n\n");
141     printf("state   input   output   next state\n");
142     for (i = 0; i < 8; i++) {
143         printf("%d     %d     %d%d     %d\n",
144                     agent->input[i * 2],
145                     agent->input[i * 2 + 1],
146                     agent->output[i * 3 + 1],
147                     agent->output[i * 3 + 2],
148                     agent->output[i * 3 + 3]);
149     }
150     fflush(stdout);
151
152     fclose(learning);
153
154     return 0;
155 }
```

Listing 2: FSM.h

```
1 #ifndef FSM_H_INCLUDED
2 #define FSM_H_INCLUDED
3
4 #include <stdio.h>
5 #include <stdlib.h>
```

```
6
7  #define TRUE 1
8  #define FALSE 0
9
10 #define TOP 0
11 #define BOTTOM 1
12 #define RIGHT 2
13 #define LEFT 3
14
15 #define EMPTY 1
16 #define BORDER 2
17 #define CLIFF 3
18 #define SAUSAGE 4
19
20 struct gridworld {
21     int M;
22     int N;
23     int world[102][102];
24 };
25
26 struct FSM {
27 /*
28 State   Input    output    next state
29 0         1        10          0
30 0         2        11          0
31 0         3        01          1
32 0         4        00          1
33 1         1        10          0
34 1         2        11          0
35 1         3        01          1
36 1         4        00          0
37
38 input
39   1       - empty cell
40   2       - border
41   3       - cliff
42   4       - sausage
43
44 state    optput
45   0        11    - go forward
46   0        10    - turn left
47   0        01    - turn right
48   0        00    - turn back
49   1        11    - go forward
50   1        10    - turn right
51   1        01    - turn left
52   1        00    - stop
53 */
54
55 int input[16];
56 int output[24];
57 int roboX, roboY;
58 int out[501];
59 int die, success, state, what_input, age;
60
```

```
61
62 };
63
64 void init_FSM(struct FSM *);
65 void FSM_step(struct FSM*, struct gridworld*);
66
67 #endif // GA_H_INCLUDED
```

Listing 3: FSM.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include "FSM.h"
5
6  #define BACK 1
7
8  void init_FSM(struct FSM *agent) {
9      int i, j;
10     int mass[] = {0, 1, 0, 2, 0, 3, 0, 4, 1, 1, 1, 2, 1, 3, 1, 4};
11     srand(time(NULL));
12
13     for (i = 0; i < 8; i++)
14         for (j = 0; j < 3; j++)
15             agent->output[i * 3 + j] = rand() % 2;
16     for (i = 0; i < 8; i++)
17         for (j = 0; j < 2; j++)
18             agent->input[i * 2 + j] = mass[i * 2 + j];
19     agent->what_input = rand() % 4;
20     agent->state = 0;
21     agent->success = FALSE;
22     agent->die = FALSE;
23 }
24
25 void walk(struct FSM *agent) {
26     switch (agent->what_input) {
27         case TOP:
28             agent->roboY++;
29             break;
30         case BOTTOM:
31             agent->roboY--;
32             break;
33         case RIGHT:
34             agent->roboX++;
35             break;
36         case LEFT:
37             agent->roboX--;
38             break;
39         default:
40             break;
41     }
42 }
43
44 void turn(int direction, struct FSM *agent) {
45     int tmp = agent->what_input;
46
```

```
47      switch (tmp) {
48          case TOP:
49                  agent->what_input = direction;
50              break;
51          case BOTTOM:
52              if (direction == LEFT)
53                  agent->what_input = RIGHT;
54              else if (direction == RIGHT)
55                  agent->what_input = LEFT;
56              else
57                  agent->what_input = TOP;
58              break;
59          case RIGHT:
60              if (direction == LEFT)
61                  agent->what_input = TOP;
62              else if (direction == RIGHT)
63                  agent->what_input = BOTTOM;
64              else
65                  agent->what_input = LEFT;
66              break;
67          case LEFT:
68              if (direction == LEFT)
69                  agent->what_input = BOTTOM;
70              else if (direction == RIGHT)
71                  agent->what_input = TOP;
72              else
73                  agent->what_input = RIGHT;
74              break;
75          default:
76              break;
77      }
78  }
79
80  void FSM_step(struct FSM* agent, struct gridworld* world) {
81      int input, output;
82      int k;
83
84      switch (agent->what_input) {
85          case TOP:
86              input = world->world[agent->roboX][agent->roboY + 1];
87              break;
88          case BOTTOM:
89              input = world->world[agent->roboX][agent->roboY - 1];
90              break;
91          case LEFT:
92              input = world->world[agent->roboX - 1][agent->roboY];
93              break;
94          case RIGHT:
95              input = world->world[agent->roboX + 1][agent->roboY];
96              break;
97          default:
98              break;
99      }
100     k = (input - 1) * 3;
101     output = agent->output[k + agent->state * 12] * 10 + agent->output[k + agent->state *
```

```
102     agent->state = agent->output[k + agent->state * 12 + 2];
103
104     switch (output) {
105         case 11:
106                 walk(agent);
107             break;
108         case 10:
109             if (agent->state  == 0)
110                 turn(LEFT, agent);
111             else
112                 turn(RIGHT, agent);
113             break;
114         case 01:
115             if (agent->state == 0)
116                 turn(RIGHT, agent);
117             else
118                 turn(LEFT, agent);
119             break;
120         case 00:
121             if (agent->state == 0)
122                 turn(BACK, agent);
123             break;
124         default:
125             break;
126     }
127 }
```

Listing 4: GA.h

```
1  #ifndef GA_H_INCLUDED
2  #define GA_H_INCLUDED
3  #include "FSM.h"
4
5  int ind[100];
6  int value[100];
7
8  int evalute (struct FSM *);
9  void sort(void);
10 void make_childs(struct FSM agent[], struct FSM childs[]);
11 void generate_parents(struct FSM childs[], struct FSM agent[]);
12 void reset_parents(struct FSM agent[], int x, int y);
13
14 #endif // GA_H_INCLUDED
```

Listing 5: GA.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "GA.h"
5
6  int evalute (struct FSM *agent) {
7      if (agent->die == TRUE)
8          return 10000;
9      else if (agent->success == TRUE)
10         return 0;
11     else {
```

```
12          return (100 - agent->roboX) + (agent->roboY - 1);
13      }
14  }
15
16  void sort(void) {
17      int i, j, tmp;
18
19      for (i = 0; i < 100; i++)
20          ind[i] = i;
21
22      for (i = 0; i < 100; i++)
23          for (j = i + 1; j < 100; j++)
24              if (value[ind[i]] > value[ind[j]]) {
25                  tmp = ind[i];
26                  ind[i] = ind[j];
27                  ind[j] = tmp;
28              }
29  }
30
31  void make_childs(struct FSM agent[], struct FSM childs[]) {
32      int i;
33      for (i = 0; i < 50; i++)
34          childs[i] = agent[ind[i]];
35  }
36
37  void generate_parents(struct FSM childs[], struct FSM agent[]) {
38      int i, j;
39      int mam, dad, koef;
40
41      for (i = 0; i < 100; i++) {
42  //          do {
43                mam = rand() % 50;
44  //          } while(value[mam] > 200);
45  //          do {
46                dad = rand() % 50;
47  //          } while(value[dad] > 200);
48
49          for (j = 0; j < 18; j++) {
50              koef = rand() % 2;
51              agent[i].output[j] = childs->output[mam * (koef) + dad * (koef)];
52          }
53          agent[i].output[rand() % 18] = rand() % 2;
54      }
55  }
56
57  void reset_parents(struct FSM agent[], int x, int y) {
58      int i;
59
60      for (i = 0; i < 100; i++) {
61          agent[i].what_input = rand() % 4;
62          agent[i].state = 0;
63          agent[i].success = FALSE;
64          agent[i].die = FALSE;
65          agent[i].roboX = x;
66          agent[i].roboY = y;
```

```
67          }
68  }
```