(2010 Semester Project)

Exploration in a gridworld searching for foods

Akira Imada

Most recently renewed on

October 25, 2010

1 Introduction

In this project, we study machine learning techniques such as Genetic Algorithm (GA), artificial Neural Network (NN), Reinforcement Learning (RL), and so on, by observing an artificial agent who travels a virtual world called a gridworld. The world has the point for the agent to start from, the point for the agent to aim as a goal, and the point for the agent to avoid.

Usually we call "*agent*" who are to learn to behave intelligently, and here, agent's aim is to reach the goal. You may, however, replace the term an "agent" with a "dog," and the "goal" with the "place where dog can find a sausage." That is, a dog learns how he/she reaches the sausage efficiently, or with a shortest path.

2 The T-maze Problem

The gridworld those agents are going to explore is like a grid shown in Figure 1. An example of a trace of agent is also shown in the figure.

In T-maze, three points are specified in a T-shaped gridworld. (i) Start point at the bottom region of the gridworld where a dog start from. Let's assume the cordinate of this point is $(0, 0)$; (ii) A food-site-1 at the left wing of the top area of the gridworld where a good food like sausages are to be found; and (iii) A food-site-2 at the right wing of the top area of the gridworld where a not good food like potatos, or sometimes bad food like very old sausages, or even poisonous food are to be found.

Agent can move only one cell at a time to the neighbor cell, that is, to up, down, right and left, unless the agent touches the border. When the agent touches the border, the action that makes the agent cross the border is not performed but it must remain stopped at the point waiting until the next action. For example, if the agent in Figure 1 is at $(2, 6)$

and the action is to *right*, then agent remains at that point, and if the next action is to *up*, then it moves to $(2, 7)$, or when the next action is *left* then agent moves to $(3, 6)$.

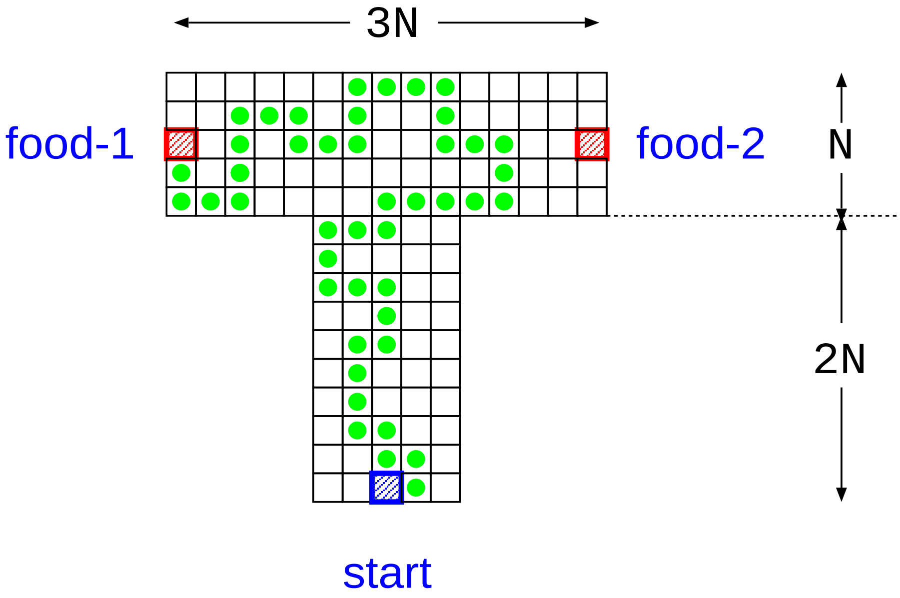


Figure 1: *A gridworld called T-maze. Also an example of trace by an agent is shown.*

Or, try to begin your experiment with more simple gridworld.

Maybe you start with much more simple gridworld – no foods, no obstacles like wall, nor border. A huge empty gridworld only with a point to start and the goal. Agent starts at the center of a huge gridworld, say 1000×1000 cells. Agents must spend one unit of energy to move to the next cell. You set the total energy when the agent starts, for example, 500 units. Then agent would never reach the border. Agent starts at $(500, 500)$ aiming the goal $(600, 600)$ for example. Then agent should look for the shortest *Manhattan distance* to the goal point. (How many such route do you guess?) See Figure 2.

3 Experiments

The result of the agent, successful or not, depends on the size of the gridworld. So experiments would be performed by changing the size of the gridworld – from small one to the large one – with M being increased, say, from 9 to 100.

You may report the number of successful run in which agent reaches the goal as a function of the size of gridworld.

You start your simulation with agent who walks the gridworld at random (Random Walk). Then you choose at least one of the machine learning technique and study how efficient

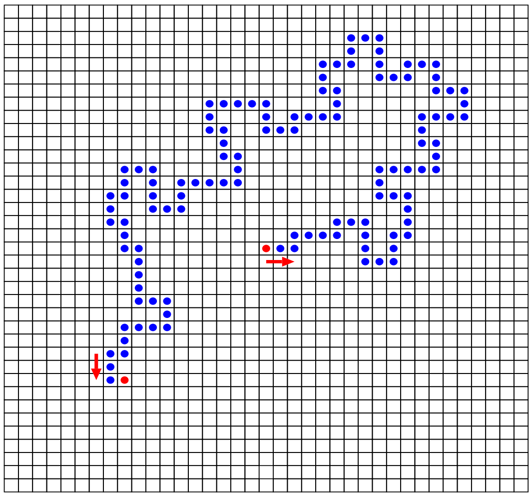


Figure 2: *An empty gridworld. Also shown an example of a random walk by an agent. Note that this is just a central region of a huge gridworld.*

the technique you chose is.

If you take only one technique, study it as deep as possible. If you take multiple techniques, you may compare them with each other as for how efficient each of techniques you chose.

3.1 Random Walk

Agent move one step upwards, downwards, to the right, or to the left, if the action is possible. If the movement is not possible due to the border of the gridworld, do nothing and decide the next action again at random, as already mentioned. This would be repeated until the agent reaches the goal, or die by spending all the energy before reacing tho the food. The maximum number of steps should be determined depending on the size of the gridworld. For example agent can move 1000 steps unless it dies during its travel.

We now look at the result of a random walk agent in an empty huge gridworld mentioned above. See Figure. 2.

Then try the gridworld of T-maze. In both cases you will summarize your results as follows.

Experiment 1 *(1) Run the algorithm by changing random number seed from run to run. (2) Count the number of agents who succeeded by repeating the run, say, 200 times. (3) Show an example of the route the successful agent follows. Or, also the route the agent failed.*

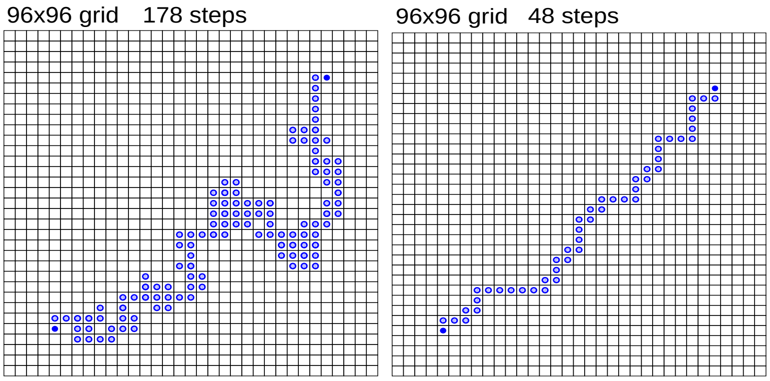


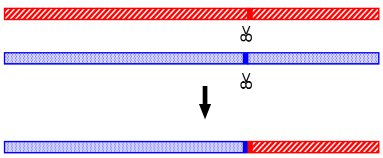
Figure 3: In the grid-world of 96×96, starting from (24,24) a robot walks aiming the goal at (72,72) of which the robot had no *a-priori* information. Left: A path chosen from 100 trials by random walk. Right: A route of the minimum Manhattan distance to the goal. (Obtained by GA). Marginal area is omitted in both figures.

3.2 Genetic Algorithm

Borrowing the idea from biological evolution, we can solve certain types of problems by an algorithm which we call *Genetic Algorithm*. In this case we express the problem we want to solve by a vector which we call *chromosome*. Chromosome is made up of 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, let's choose them at random from better half of the population. This is called *truncate selection*.[1]

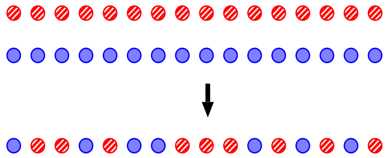
Let's call these two chromosomes parents. Then we create a child from these two parents by *crossover operation*.

Standard crossover is what we call *one-point-crossover*. We cut two parent chromosome at a same location at random. That is, the location to be cut is chosen by creating a random integer from one to N — the total number of genes in one chromosome. Then we take the first part from one parent and second part from the other parent and unite the two parts. See the following Figure.



[1]We have a couple of different types of selection such as *roulette-wheel-selection* and *tournament-selection*.

Or, yet another crossover is called *uniform-crossover* where we choose genes one by one either from parents at random. See the following figure.



Then by repeating this procedure (select two parent chromosomes and create one child chromosome), say 100 times, we create the next *generation*. The population of the next generation includes the same number of chromosomes in the population of the previous generation. Thus we can evolve the first random population of chromosomes generation by generation. We can expect those chromosomes' performance becomes better and better gradually.

We also give a *mutation* to introduce new genes. This is to 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.

Agent in GA behaves by following its chromosome

What we assume here is an evolution in an agent's brain in advance the action. That is a population of chromosomes in the brain will be evolved. Then after a convergence, the agent acts following the best chromosome.

Our chromosome in this study of exploring gridworld is made up of 4 different genes move (i) up, (ii) down, (iii) to right , and (iv) to left. See an example below.

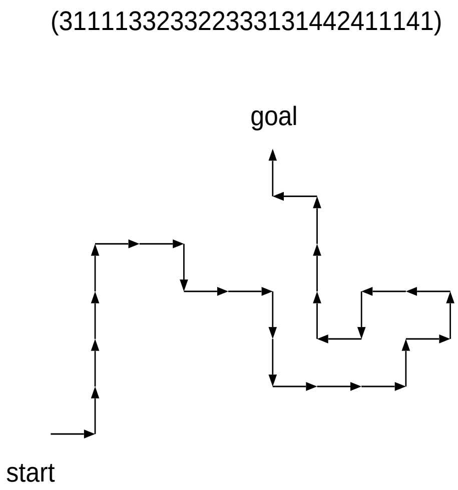


Figure 4: An example of a chromosome and the path represented by it.

A GA implementation

Assuming now the total energy when the agent starts is 500 units, as an example, all chromosome has 500 genes whose value is 1, 2, 3, or 4. Then (1) Create 100 such chromosomes at random. This is the initial population; (2) Evaluate fitness – how good is the chromosome – by the Manhattan distance when the route approaches to the goal with a closest distance. Note that the smaller the fitness, the better the chromosome; (3) Sort the population according to the fitness. With the best at the top and with the worst at the bottom; (4) Pick up two parent chromosomes randomly from the upper half of the population and create one child chromosome by uniform crossover; (5) Mutate the child chromosome by replacing the gene with 1, 2, 3, or 4 at random. This should be done by selecting the gene to be mutated with a probability of 1/500. Not all the genes; (6) Repeat (4) and (5) 100 times to create the next generation; (7) Record the average fitness and maximum fitness in each generation; (8) Stop the iteration when fitness of any chromosome becomes 0, which means the chromosome is successful in reaching to the goal, then stop the run.

The following Figure is the maximum fitness versus iteration of such a run.

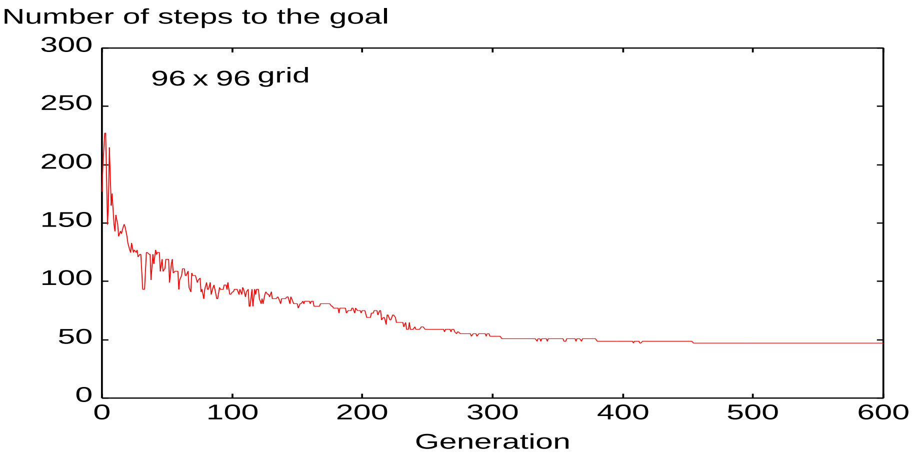


Figure 5: Random walk evolved to be minimized by GA.

3.3 Reinforcement Learning

In Reinforcement Learning (RL), an agent takes one *state* at a time chosen from predefined all possible *states*. In each of those state, also all possible actions are given one by one with a probability of how likely the action will be made.

It might be helpful to understand this if we imagine that we sometimes in a *state of being happy* or in a *state of being sad*. Then assume now we see a cat. Also assume the possible actions will be (i) hug the cat; (ii) neglect; (iii) kick the cat. When we are in a happy state, most likely action will be hug the cat, while when we are in a state of sad, we might neglect the cat or might kick the cat.

Agent in RL behaves by following its policy

Thus, RL is defined with a set of *states* and a set of *actions*. Then, we have a table called *policy* in which each of all possible pairs of state and action corresponds to its probability. Starting with a random assignment of this probability at the beginning, the policy will be renewed according to the agent's experience.

Table 1: *A toy example of policy. Agent action depends on its state.*

	hug and caress	neglect	kick
Happy	0.80	0.15	0.05
Normal	0.20	0.70	0.10
Unhappy	0.10	0.50	0.40

Q-learning

States of our current study of agent's exploration in a gridworld is the cell in which the agent is located. So we have $5M^2$ different states. The possible actions is either up, down, right and left.

To choose the action in one state, we employ what we call ϵ-greedy strategy. That is, we choose an action at random with probability ϵ (a pre-fixed small value such as 0.1) and the action with the highest value in the Q-table with probability $1 - \epsilon$.

One method of renewing the policy table is called *Q-learning*. Assume now the state of the agent at time t is s_t and the action policy table gives is a_t.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\{r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\},$$

where $\max_a Q(s_{t+1}, a)$ is the action with the highest value in the state of s_{t+1} in the Q-table, α is called *learning ratio* which determines how this renewal influences the over all learning, and γ is called *discount rate* which determines the importance of the reward received in the future. Anyway both value takes a value between 0 and 1. Try your experiment by changing these two parameters. Reward at time t is expressed by r_t. In our case, $r - t$ is -10 when the agent tried to cross border, -100 when the agent reaches to the not good food, if it's a poison -500, $+500$ when agents reaches to the goal, and all other empty cell gives the agent -1.

Note that s_{t+1} is automatically determined if an action a_t is performed at the state s_t. For example, if the action is *up* at the state of $(4, 3)$, then the next state will be $(4, 4)$.

This process of selection and update the Q-table is repeated until the goal state is reached. This is called *epoch*. This epoch is repeated from one agent to the next until the Q-table will not be changed any more, that is, until the Q-table becomes stable.

3.4 Neural Network

In order to challenge agent with Neural Network might be very tough. So, just an observation of the agent moving with neural networks but before learning procedure in the gridworld with out border, obstacle, nor cliff – a huge empty gridworld starting from the center of the world.

3.4.1 Sensor for smell

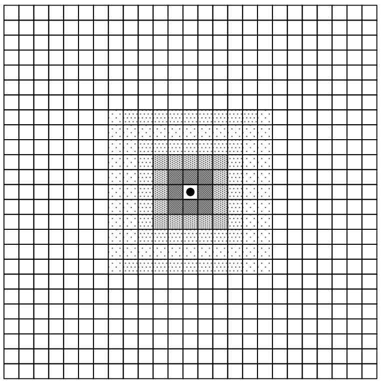


Figure 6: *An example how smell is diffused, in general. The closer to the food, the stronger the smell.*

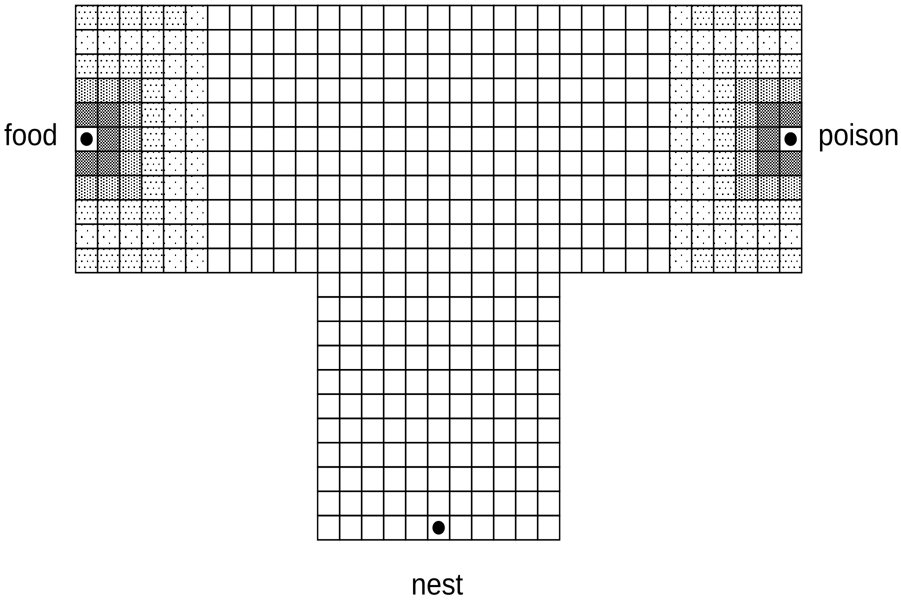


Figure 7: *An example of smell diffusion in the T-maze promlem.*

3.5 Implementation

Neural network has three sensor inputs which detect smell of its front-left, front, front-right, which allow the neural network also detect which direction it is going to. And the number of output is four and follow the winner-take-all rule, for example. Each output corresponds to up, down, right and left. Also neural net work has a hidden layer with three neurons.

In both of the good smell and bad smell, the closer to the food the stronger the semll. Starting with 100 stupid neural networks with random 21 weights specified by its cromosome, evolve them by genetic algorithm.

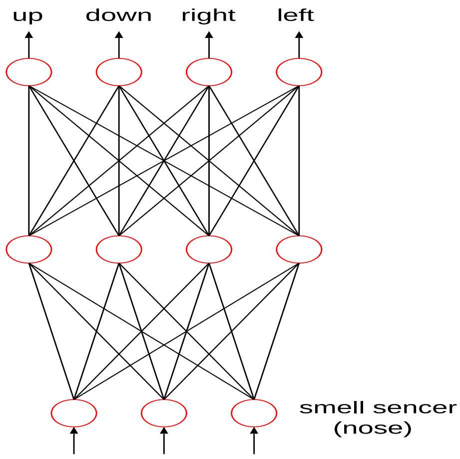


Figure 8: *A possible sturcture of the neural network for T-maze problem.*

3.5.1 Spiking Neurons

Integrate & Fire Model

Amang others, we use an *Integrate-and-Fire* model of spiking neurons following Florian (2007) in which dynamics of neurons is according to:

$$u_i(t) = u_r + \{u_i(t-\delta t) - u_r\}\exp(-\delta t/\tau) + \sum_j w_{ij} f_j(t-\delta t),$$

assuming we simulate the dynamics in discrete time with a time step δt for the sake of simplicity, where $u_i(t)$ is membrane potential of the i-th neuron at time step t, u_r is resting potential, τ is decay time constant, and $f_j(t)$ is 1 if j-th neuron has fired at time step t or 0 otherwise. When membrane potential exceeds firing threshold θ it is reset to the reset potential which is equal to the resting potential u_r here.[2]

[2]Florian's setting is u_r = -70 mV, θ = -54 mV, τ = 20 ms and δt = 1 ms.

3.6 Ants' Exploration

Here, sugers are at site-1 and poisons (ant-killer) are at site-2. An infinite number of ants repeatedly start from its nest located at the start points. In each of the grid where ant steps in, ant put a chemical called phelomone. A probability of movement of one ant is proportinal of total amount of phelomones so far accumulated. That is, if the amount of pheolomone of up, down, right, and left is 20, 40, 10, and 30 mg, respectively. the probability of the movement to up, down, right, and left will be 0.2, 0.4, 0.1 and 0.3. So, most likely to down. (At the beginning, there's no phelomon so equal probability at random.) A happy ant try to go back to the nest with the same rule, but ants failed to find the food or eat poison will die. Pheromons also evaporates at a rate, say, 1mg every 100 time-steps. After enough time passed, you may observe all ants reach the sugare site and go back to the nest efficiently.

I have never seen, that artificial ant colony challenge this cliff walking problem. Hence if your simulation with artificial ant colony successfully, you may be the first in the world.

REFERENCE

Original proposal of GA and GP.
You do not necessarily read these two books but you should cite the both.

Holland, J. H. (1975) "Adaptation in Natural and Artificial Systems." University of Michigan Press, Ann Arbor.

Sutton, R. S and A. G. Barto (2005) "Reinforcement Learning: An Introduction." A Bradford Book. The MIT Press. pp. 185-186

To understand the methods

To challange further