Inference Approaches to Optimal Control

Jiguang Lu(331331)
Lu Peng(329210)
Shaoyun Liu(322780)
Yangchao Liu(324738)

Projects in Machine Learning and Artificial Intelligence

WiSe10/11

Informatik Fakulität
Technische Universität zu Berlin

Abstract

Q-learning (at first was introduced by Watkins in 1989) is one of the simplest way for agents to learn how to solve the optimal problems in Controlled Markovian Domains.[1] It can amount to an incremental method for dynamic programming which imposes limited computational demands. So it can be widely used for calculating the most possible choice of particular quality actions at many fields as diverse as neuroscience, psychology, economics, computer science and control engineering.

In this paper we have discussed the basic theory and methods of Q-learning, some models and given out an example of it. We show the readers how to calculate the most efficient value with Q-learning in the optimal controlled problems, and compared with some different relevant models and also used a program to prove our main theory.

Keywords

MDPs, Q-learning, reinforcement learning, temporal differences, grid world

1. Introduction

Q-learning is a kind of reinforcement learning technique, which works with the learning of value action function and gives the expected utility of taking a given action in a given state and following a fixed policy thereafter [2]. The agent can use Q-learning to learn a mapping and with the help of value analysis to decide which action he should take when there is a state that the environment has given out.

1.1 Reinforcement learning

At first, we explain the theory of Markov decision processes (MDPs), it is very useful to solve the problems of optimization control via dynamic programming and reinforcement learning. And that is a discrete time stochastic control process. We define at each time step, the process is in some state *s*, and the decision maker may choose any action *a* that is available in state *s*. This definition will be used in our whole paper. And when there is a new action *a'*, there will be also a new reward.

Reinforcement learning is an approach to artificial intelligence that emphasizes learning by the individual from its interaction with its environment. [3]

Reinforcement learning is about learning what to do and how to choose the way in the maps of grid world for the actions. The main goal of reinforcement learning is to maximize the reward and minimize the cost in the action. Instead of telling the learner which action to take, most forms of reinforcement learning must find out the action yield of the most reward or least cost and try them.

In the typical environment form of machine learning is Markov decision process (MDP) and we related highly many reinforcement learning algorithms for the context to the dynamic programming techniques. And in my opinion, the reinforcement learning is usually a long time learning as lifelong learning and when we use it to learn MDP, we don't need to know the knowledge of MDP itself, we need only to know the methods, that means, everyone can use these methods not only the math professors.

1.2 Q-learning

Q-learning is a recent form of reinforcement learning, we can use it without the environment of this model and it is very useful and suitable for the repeated game theory under unknown opponent. And Q-learning functions basis of the estimating the values of state-action pairs.

The problem model consists of an agent, states *S* and a set of actions per state *A*. We define the value like Q(s,a) as the expected discounted sum of future profit.(And we can also calculate the sum of the future payoffs, and under this situation we must find out the minimum of the results.)When these values have been learned, it will be an optimal action from any state with the highest Q-value. Q-learning can be estimated as basis like:

From the current state *s*, select an action *a*. This will cause a receipt of an immediate payoff *r*, and arrival at a next state *s'*.

1. From the current state *s*, select an action *a*. This will cause a receipt of an immediate payoff *r*, and arrival at a next state *s'*.
2. Update Q(*s*,*a*) based upon this experience as follows:

Small changes in Q(*s*,*a*) = *x*[*r* + *y*maxQ(*s'*,*a'*)-Q(*s*,*a)*]
where *x* is the learning rate and 0 < *y* < 1 is the discount factor

1. Go to 1. [4]

1.3 Grid world

The grid world is a very important type of MDPs; it is a case study, which employs an actor class to construct objects in the grid. This actor class can control the direction and location in the grid and act the other objects. There are usually the classes like ‘‘flowers’’ (or measures), "rocks", "bugs" and "critter", this form usually hat a single start and a single end. And we can get a big value at the end of the grid world.

There are many games in the form of grid world, one of the most famous games is gold digger(for example pic.1). In this game you can see you have only 25 times from start at the left of the grid to find out the gold, and every time, the actor can only move one grid, and there are some grids with big gold and small gold, and there maybe also are some promotion like the shining grid, which has already gold there. And finally we must dig as the most gold as we can, in this case, all the gold will be hidden, and we don't know the whole value of this process until end of the digging, we can only promote across the shining grid, how to get some gold.

Pic.1 gold digger

And there are also some other types of grid world like Pic.2. We can see all the result, the measure, rocks and so on, but every time you can only calculate 4 actions in the grid. And every time we can choose one of the four dices under the grid, and each dice we can only have once to choose the promotional grid. And in this game we have only 10 chances for the actor to go from start to final, and we can only forecast at most 4 actions in the future from the four stochastic dices. The main goal of this grid world is not only to get the destination of the grid, but also to get as most of the value of coins; we can get some coins from the grids 1, and lost coins by thieves from the grids 2. And all the grids with the mark of the swords are combats, when you lost the battle, you must also pay some coins. So at the end of the grid we can calculate the whole value we can choose and decide play again.

Pic.2 example of grid world

Generally speaking, we can see, in this grid world, the action must be alive, it can move, and there are not some more rules but some encouragements and lost. Our goal

in the grid world is to maximize the value we can find out .We calculate the sum of rewards and make it as higher as possible with the tools like reinforcement learning.

2. Math's model

Like we have already talked in the introduction, we definite at first states *S* and a set of actions per state *A*. And we definite the function U (*i*) is a set of admissible controls at state $i \in S$, $l(i; u) \geq 0$ is a cost for being in state *i* and choosing a control as $u \in U(i)$, and P (*u*) is a stochastic matrix and its element p*ij* (u) is the transition probability from state *i* to state *j* under control *u*. A is a non-empty subset, and $A \subseteq S$ of states are absorbing and incur zero cost: $p_{ij}(u) = \delta^{j}_{i}$ and $l(i; u) = 0$ whenever $i \in A$. The results for other formulations will be summarized later.[5] And now we can get a function to calculate the value, whose name is Bellman equation.

$$v(i) = \min_{u \in \mathcal{U}(i)} \left\{ \ell(i,u) + \sum\nolimits_{j} p_{ij}(u)\, v(j) \right\} \tag{1}$$

2.1 Main model

And usually we can use this Bellman equation to solve most of MDPs problems ,that means we can use Bellman equation to solve the continuous MDPs, but when stochastic approximations of the optimal value function which can be used when a model is not available, we must use another ways to solve this problem, one of them is Q-learning.

$$Q(s,a) = x[r + y\max Q(s',a') - Q(s,a)] \tag{2}$$

We usually definite the learning rate *x* as a constant value, we can write this function like follows:

$$Q(s,a) \leftarrow r + y\max Q(s',a') - Q(s,a) \tag{3}$$

And in this function, *r* is immediate reward, we can also write r as *r*(*s*,*a*), and $y \in (0,1)$ is the discount factor, it is a relative value of delayed and Immediate rewards. *s '* is the new state after action *a*, max*Q(s',a')* is the max future value and Q(*s*,*a*) is the old value.

The learning rate *x* determines to what about the new extent acquired

information will write into the old information. When *x* is bigger, there will be more changes in the function. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information.

And the discount factor *y* influence the future rewards. A factor of 0 will make the agent "opportunistic" by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. If the discount factor meets or exceeds 1, the *Q* values will diverge.

Here we take an example to explain the whole calculate process of Q-learning.

2.2 Model for the interactions between agents and the world

We definite an actual environment and it will be influenced by some actions. It has rewards and only some parts of it have risks and lose.

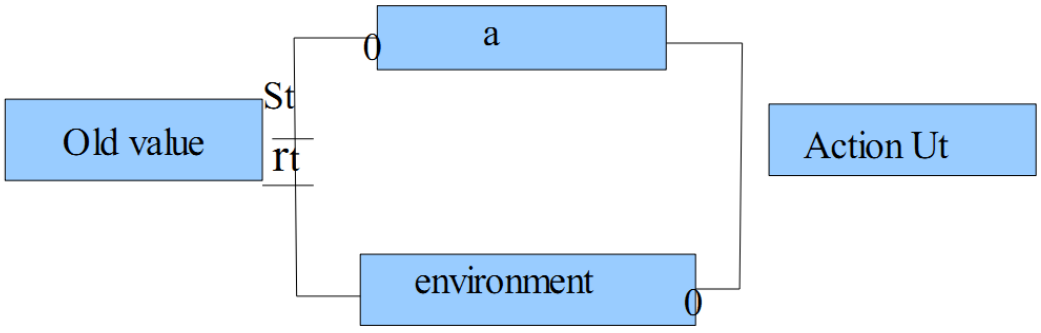
We explain the dynamic description in a discrete world.[6]

The state and rewards are changed through events and the agent influences this world with events. So there are some strategies for these events. And we use e_t as the event at the time of t.

$$s_0 \xrightarrow{e_0} r_1 s_1 \xrightarrow{e_1} r_2 s_2 \xrightarrow{e_2} \ldots.$$

Or we can also use action u_t of the agent.

$$s_0 \xrightarrow{u_0} r_1 s_1 \xrightarrow{u_1} r_2 s_2 \xrightarrow{u_2} \ldots.$$



And then we can get the sequence like：

$$s_0 \xrightarrow{u_0} r_1 s_1 \xrightarrow{u_1} r_2 s_2 \xrightarrow{u_2} \ldots. \xrightarrow{u_{t-1}} r_t s_t \xrightarrow{u_t}$$

And the new reward at the time of t+1 is $r_{t+1}s_{t+1}$

Depending on the old reward:

$$P(s_{t+1} = s', r_{t+1}=r \mid s_0\, u_0 r_1 s_1 u_1 r_2 s_2 u_2 \ldots u_{t-1} r_t s_t u_t)$$

Markov-Conditions: $$= P(s_{t+1} = s', r_{t+1}=r \mid s_t u_t)$$

That means, only the last action we can calculate.

We describe the dynamics of the system of stochastic world with Markov condition:

$$F: S \times U \rightarrow \text{Verteilungen über } S \times \mathbb{R}$$

$$P(s' = s_{t+1}, r = r_{t+1} \mid s_t u_t = su) = F(s,u)(s',r)$$

And the Transition probability: $P_{sus'} := P(s' = s_{t+1} \mid s_t u_t = su)$

Reward probability: $P_{sus'r} := P(r = r_{t+1} \mid s_t u_t s_{t+1} = sus')$

Expected value for rewards: $r_{sus'} := E(r_{t+1} \mid s_t u_t s_{t+1} = sus')$

For an agent at time t: r_t or $r_{sus'}$

Cumulative value of fixed sequence: $s_0 -u_0\rightarrow r_1 s_1 -u_1\rightarrow r_2 s_2 -u_2\rightarrow \ldots.$

and the value from the time t: $R_t := \sum_{i=0\ldots\infty} \gamma^i r_{t+1+i}$

With Discount-Factor $0 < \gamma <= 1$

we definite in a stochastic world an expected value Rt for a given strategy π. Then we can get a deterministic strategy for the deterministic MDP.

$$\pi: S \rightarrow U$$

And we definite S as the number of states and U is the number of the actions.

The reward functions are:

$$u_t = \pi(s_t)$$

$$r_{t+1} = f_r(s_t,u_t)$$

$$s_{t+1} = f_s(s_t,u_t)$$

And the we calculate the value function in deterministic MDP with a given strategy π for each state s: $V_\pi: S \rightarrow \mathbb{R}$;

and $V\pi(s)$ is the value that is achieved when it starts in s, and works for all π:

$V_\pi(s_t) := R_t = \sum_{i=0\ldots\infty} \gamma^i r_{t+1+i}$ and $0 \leq \pi \leq 1$.

And in the case the strategy π is always the same, so $V\pi$: $S \rightarrow R$, and its expected value is:

$$V_\pi(s) := E(R_t \mid s = s_t, \pi) = E\left(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s = s_t, \pi\right)$$

And the Bellman equation is:

$$\begin{aligned}
&V_\pi(s)\\
&:= E(R_t \mid s = s_t, \pi)\\
&= E\left(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s = s_t, \pi\right)\\
&= E\left(r_{t+1} + \gamma \sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s = s_t, \pi\right)\\
&= \sum_u \pi(s,u) \sum_{s'} P_{sus'} \left(r_{sus'} + \gamma E\left(\sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s' = s_{t+1}, \pi\right)\right)\\
&= \sum_u \pi(s,u) \sum_{s'} P_{sus'} \left(r_{sus'} + \gamma V_\pi(s')\right)
\end{aligned}$$

And we can also get the Q-learning function as the action value function:

$$Q_\pi: S \times U \rightarrow \mathbb{R}$$

And the expected value

$$\begin{aligned}
&Q_\pi(s, u)\\
&:= E(R_t \mid s=s_t, u=u_t, \pi) = E\left(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s=s_t, u=u_t, \pi\right)
\end{aligned}$$

Compare it to Bellman equation:

$$\begin{aligned}
&Q_\pi(s,u) := E(R_t \mid s = s_t, u=u_t, \pi)\\
&= E\left(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s = s_t, u=u_t, \pi\right)\\
&= E\left(r_{t+1} + \gamma \sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s = s_t, u=u_t, \pi\right)\\
&= \sum_{s'} P_{sus'} \left(r_{sus'} + \gamma \sum_{u'} \pi(s',u') E\left(\sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s'=s_{t+1}, u'=u_{t+1}, \pi\right)\right)\\
&= \sum_{s'} P_{sus'} \left(r_{sus'} + \gamma \sum_{u'} \pi(s',u') Q_\pi(s',u')\right)\\
&= \sum_{s'} P_{sus'} \left(r_{sus'} + \gamma V_\pi(s')\right)
\end{aligned}$$

For maximum strategy π^* is : $Q^*(s,u) := \max_\pi Q_\pi(s,u)$

$$Q_{\pi^*}(s,u) = \sum_{s'} P_{sus'} \left(r_{sus'} + \gamma \max_{u'} Q_{\pi^*}(s',u')\right)$$

And

$$Q_{\pi^*}(s,u) \rightarrow r_{sus'} + \gamma \max_{u'} Q_{\pi^*}(s',u')$$

3. Program Analysis

In order to understand how the Q learning algorithm works, we will take a simple example to explain it.
Gama can be any value between 0 and 1, and here we set the value of learning parameter $\gamma = 0.8$.

First we set matrix **Q** as a zero matrix.

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Second we set up a random matrix, the Column represents the current state ,and Every Row means possible actions.

$$\mathbf{R} = \begin{array}{c} state \setminus action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 0 & - \\ - & - & - & 0 & - & 100 \\ - & - & - & 0 & - & - \\ - & 0 & 0 & - & 0 & - \\ 0 & - & - & 0 & - & 100 \\ - & 0 & - & - & 0 & 100 \end{bmatrix} \end{array}$$

Look at the second row (state B) of matrix **R.** There are two possible actions for the current state B, that is to go to state D, or go to state F. Here, we select to go to F as our action.

Now we suppose that we are in state F. Look at the sixth row of reward matrix **R** (i.e. state F). There are 3 possible actions to go to state B, E or F.

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

$$\mathbf{Q}(B,F) = \mathbf{R}(B,F) + 0.8 \cdot Max\left\{\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)\right\} = 100 + 0.8 \cdot 0 = 100$$

Since matrix **Q** that is still zero, $\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)$ are all zero. The result of computation $\mathbf{Q}(B,F)$ is also 100 because of the instant reward.

The next state is F, now become the current state. Because F is the goal state, we finish one episode. Our agent's brain now contain updated matrix **Q** as

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

For the next episode, we start with initial random state. This time for instance we have state D as our initial state. Look at the fourth row of matrix **R;** it has 3 possible actions that is to go to state B, C and E. By random selection, we select to go to state B as our action.
Now we imagine that we are in state B. Look at the second row of reward matrix **R** (i.e. state B). It has 2 possible actions to go to state D or state F. Then, we compute the **Q** value

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

$$\mathbf{Q}(D,B) = \mathbf{R}(D,B) + 0.8 \cdot Max\left\{\mathbf{Q}(B,D), \mathbf{Q}(B,F)\right\} = 0 + 0.8 \cdot Max\left\{0, 100\right\} = 80$$

We use the updated matrix **Q** from the last episode. $\mathbf{Q}(B,D) = 0$ and $\mathbf{Q}(B,F) = 100$. The result of computation $\mathbf{Q}(D,B) = 80$ because of the reward is zero. The **Q** matrix becomes

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The next state is B, now become the current state. We repeat the inner loop in Q learning algorithm because state B is not the goal state.

For the new loop, the current state is state B. There are two possible actions from the current state B, that is to go to state D, or go to state F. By lucky draw, our action selected is state F.

Now we think of state F that has 3 possible actions to go to state B, E or F. We compute the Q value using the maximum value of these possible actions.

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

$$\mathbf{Q}(B,F) = \mathbf{R}(B,F) + 0.8 \cdot Max\left\{\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)\right\}$$

$$= 100 + 0.8 \cdot Max\left\{0,0,0\right\} = 100$$

The entries of updated Q matrix contain $\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)$ are all zero. The result of computation $\mathbf{Q}(B,F)$ is also 100 because of the instant reward, see figure 1. This result does not change the Q matrix.

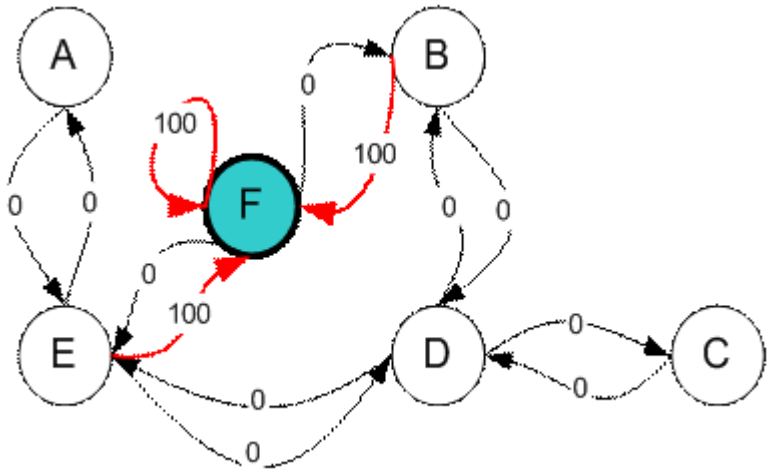


figure 1

Because F is the goal state, we finish this episode. Our agent's brain now contain updated matrix **Q** as

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

If our agent learns more and more experience through many episodes, it will finally reach convergence values of Q matrix as

$$\mathbf{Q} = \begin{array}{c} state \backslash action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 400 & - \\ - & - & - & 320 & - & 500 \\ - & - & - & 320 & - & - \\ - & 400 & 256 & - & 400 & - \\ 320 & - & - & 320 & - & 500 \\ - & 400 & - & - & 400 & 500 \end{bmatrix} \end{array}$$

This **Q** matrix, then can be normalized into a percentage by dividing all valid entries with the highest number (divided by 500 in this case) becomes

$$\tilde{\mathbf{Q}} = \begin{array}{c} state \backslash action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 80 & - \\ - & - & - & 64 & - & 100 \\ - & - & - & 64 & - & - \\ - & 80 & 51 & - & 80 & - \\ 64 & - & - & 64 & - & 100 \\ - & 80 & - & - & 80 & 100 \end{bmatrix} \end{array}$$

Once the Q matrix reaches almost the convergence value, our agent can reach the goal in an optimum way. To trace the sequence of states, it can easily compute by finding action that makes maximum Q for this state.
For example from initial State C, it can use the Q matrix as follow:
From State C the maximum Q produces action to go to state D
From State D the maximum Q has two alternatives to go to state B or E. Suppose we choose arbitrary to go to B
From State B the maximum value produces action to go to state F
Thus the sequence is C – D – B – F, see figure 2.

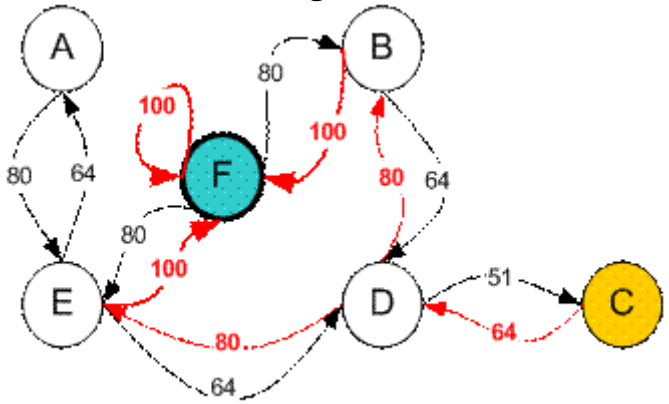


figure 2

4. Program Description

The environment of program is Matlab 7.5.0. There are two program files: Q_learning.m and RandomPermutation.m.

Q_learning.m:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% input A matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% room:1, wall:-1, start:0, goal: 2
n1 = input('please input the row number:');
m1 = input('please input the column number:');

A = zeros(n1,m1)

for i=1:n1
    for j=1:m1
        A(i,j)=input('please input element a_ij:')
    end
end

gamma = input('please input gamma:');  % learning parameter, 0 < gramma < 1



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Q learning : 1. generate R matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%get the sizes of A matrix,
p=size(A);
n=p(1);    % row = n
m=p(2);    % column = m

% generate R matrix:
% row = actions and column = states; -Inf = no door between room
R = zeros(n*m,n*m);

for i=1:n*m
    for j=1:n*m
        %if i == j
            R(i,j)= -inf;
        %elseif
        %end

    end

end

for i=1:n
    for j=1:m
        if i < n && A(i,j) ~= -1 && A(i+1,j) ~= -1
            R((i-1)*m+j,(i-1+1)*m+j)= 0;
            R((i-1+1)*m+j,(i-1)*m+j)= 0;
        end
```

```
        if j < m && A(i,j) ~= -1 && A(i,j+1) ~= -1
            R((i-1)*m+j,(i-1)*m+j+1)= 0;
            R((i-1)*m+j+1,(i-1)*m+j)= 0;
        end

    end

end

 initial_i = 0;
 initial_j = 0;

% The doors that lead immediately to the goal have instant reward of 100
for i=1:n
    for j=1:m
        if A(i,j) == 2
            for k = 1: n*m
                if R(k,(i-1)*m+j) ==0
                    R(k,(i-1)*m+j) =100;
                end
            end
        end

    end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Q learning : 2. generate Q matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    %gamma=0.80;          % learning parameter, 0 < gramma < 1

    q=zeros(size(R));     % initialize Q as zero
    q1=ones(size(R))*inf;  % initialize previous Q as big number
    count=0;             % counter

for episode=0:50000
   % random initial state
      y=randperm(size(R,1));
      state=y(1);

      % select any action from this state
      x=find(R(state,:)>=0);       % find possible action of this state

      if size(x,1)*size(x,2)>0,
        x1=RandomPermutation(x);   % randomize the possible action %%%%%%%%%
function y=RandomPermutation(A)
        x1=x1(1);               % select an action
        qMax=max(q,[],2);
      q(state,x1)= R(state,x1)+gamma*qMax(x1);   % get max of all actions
      state=x1;

      % break if convergence: small deviation on q for 1000 consecutive
```

```
      if sum(sum(abs(q1-q)))<0.0001 & sum(sum(q >0))
        if count>1000,
          episode       % report last episode
          break         % for
        else
          count=count+1; % set counter if deviation of q is small
        end
      else
        q1=q;
        count=0; % reset counter when deviation of q from previous q is large
      end
      end


   end

   %normalize q
   g=max(max(q));
   if g>0,
     q=100*q/g;
     q  % input Q matrix
   end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Find road
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


   initial_i = 0;
   initial_j = 0;
   end_i = 0;
   end_j = 0;


   for i=1:n
       for j=1:m
           if A(i,j) == 0  % find start in A matrix
               initial_i = i;
               initial_j = j;
           elseif A(i,j) ==2  % find goal in A matrix
               end_i = i;
               end_j = j;
           end
       end
   end

   state = (initial_i-1)*m + initial_j; % start
   door = (end_i-1)*m + end_j;  % goal
   road = zeros(n*m,1);  % initialize road as zero
   road(1) = state;
   k = 1;
   while(state ~= door && k < (n*m+1))
       k=k+1;
       temp = find(q(state,:)==max(q(state,:))); % find action that makes
maximum Q for this state
       road(k) = temp(1);  % find one of the actions that makes maximum Q for
```

```
this state
       state = road(k);
   end
road
```

RandomPermutation.m:

```
function y=RandomPermutation(A)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% return random permutation of matrix A
% unlike randperm(n) that give permutation of integer 1:n only,
% RandomPermutation rearrange member of matrix A randomly
% This function is useful for MonteCarlo Simulation,
%  Bootstrap sampling, game, etc.
%
% Copyright Kardi Teknomo(c) 2005
% (http://people.revoledu.com/kardi/)
%
% example: A = [ 2, 1, 5, 3]
% RandomPermutation(A) may produce [ 1, 5, 3, 2] or [ 5, 3, 2, 3]
%
% example:
% A=magic(3)
% RandomPermutation(A)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  [r,c]=size(A);
  b=reshape(A,r*c,1);      % convert to column vector
  x=randperm(r*c);         % make integer permutation of similar array as key
  w=[b,x'];               % combine matrix and key
  d=sortrows(w,2);         % sort according to key
  y=reshape(d(:,1),r,c);    % return back the matrix
```

5. Program Demo

```
please input the row number:4
please input the column number:4

A =

     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0

please input element a_ij:1
```

Input the number of rows and columns, here we enter a 4 * 4 matrix， That also

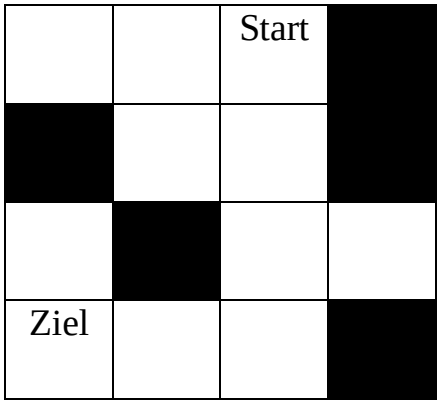
means a 4 * 4 grid world.

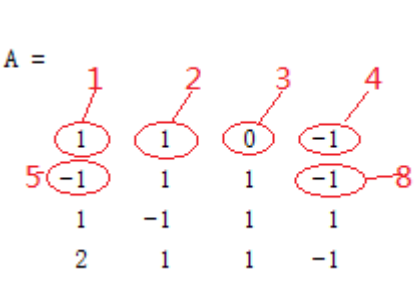
```
please input gamma:0.8
```

Input the Gamma Value, gamma value can be Random, usually we choose 0.8

```
A =

     1     1     0    -1
    -1     1     1    -1
     1    -1     1     1
     2     1     1    -1
```

From left to right from top to bottom，input the values one by one. 1, 0,-1, 2 corresponding to Space, Start, Wand, Ziel in the grid world.





Give every value a serial number one by one, look at the Picture above. And Result of the computing is also outputting by the form of the serial number.

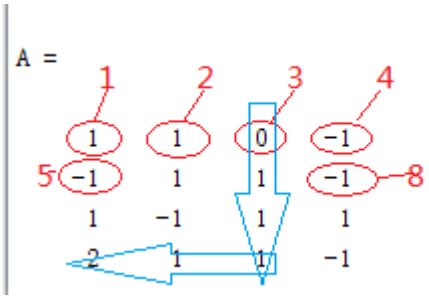
Our agent learns more and more experience through 2847 episodes; it will finally reach convergence values of matrix as following.

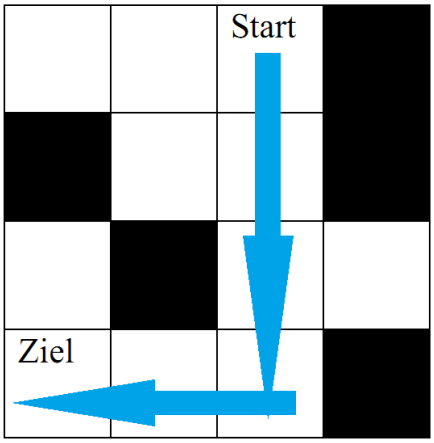
```
episode =

        2847


q =

         0   26.2144         0         0         0         0         0         0         0         0         0         0         0         0         0         0
   20.9715         0   32.7680         0         0   32.7680         0         0         0         0         0         0         0         0         0         0
         0   26.2144         0         0         0         0   40.9600         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0   26.2144         0         0         0         0   40.9600         0         0         0         0         0         0         0         0         0
         0         0   32.7680         0         0   32.7680         0         0         0         0   51.2000         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0  100.0000         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0   40.9600         0         0         0         0   40.9600         0         0   64.0000         0
         0         0         0         0         0         0         0         0         0         0   51.2000         0         0         0         0         0
         0         0         0         0         0         0         0         0   80.0000         0         0         0         0   80.0000         0         0
         0         0         0         0         0         0         0         0         0         0         0         0  100.0000         0   64.0000         0
         0         0         0         0         0         0         0         0         0         0   51.2000         0         0   80.0000         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
```

There are 4 actions available: up, down, left and right for every square. So our Computing Matrix expands to 16*16 Matrix.

```
road =

     3
     7
    11
    15
    14
    13
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
```

The Computing result is the path of serial number: 3,7,11,15,14,13.





The blue arrows show the optimal action based on the current value function.

Resources

[1]Watkins, C.J.C.H. (1989). Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England.
[2] Definition of Q-learning in Wikipedia: http://en.wikipedia.org/wiki/Q-learning
[3]Richard S. Sutton (January 28, 1999).Reinforcement learning,
[4]Zarul Hamzah(1992)Are we learning now?
[5]Emanuel Todorov(2007)Linearly-solvable Markov decision problems,page1
[6]Prof. Dr. sc. Hans-Dieter Burkhard(2008).Reinforcement Learning