ISSN:0975-9646

Finite Markov Decision Process Frame Work Algorithm

P.Sushma [#1], Dr.Yogesh Kumar Sharma [*2], Dr.S. Naga Prasad [#3]

1 Ph.D Scholar, Dept of CS, JJTU University, Jhunjhunu, Churela, Rajasthan,India
2.Associate Professor, Dept of CS, JJTU University, Jhunjhunu, Churela, Rajasthan,India
3 .Lecturer , Dept of CS, Tara Degree College,Sangareddy,Telangana,India

Abstract:- In machine learning, the environment is formulated as a Markov decision process (MDP), as many reinforcement learning algorithms for this context utilize dynamic programming techniques. The supervised learning and unsupervised learning, the paradigm of reinforcement learning deals with learning in sequential decision making problems in which there is limited feedback. RL is a general class of algorithms in the field of machine learning that aims at allowing an agent to learn how to behave in an environment, where the only feedback consists of a scalar reward signal. This text introduces the intuitions and concepts behind Markov decision processes and two classes of algorithms for computing optimal behaviors: reinforcement learning and dynamic programming. First the formal framework of Markov decision process is defined, accompanied by the definition of value functions and policies. The main part of this text deals with introducing foundational classes of algorithms for learning optimal behaviors, based on several of optimality with respect to the goal of learning sequential decisions. Additionally, it surveys efficient extensions of the foundational algorithms, differing mainly in the way feedback given by the environment is used to speed up learning, and in the way they concentrate on relevant parts of the problem. For both model-based and model-free settings these efficient extensions have shown useful in scaling up to larger problems.

1.INTRODUCTION

An MDP is a discrete time-state transition system. MDP is actually a reinforcement learning task and it satisfies all the requirements of a Markov property. Furthermore, finite MDPs or finite MDPs having the finite actions and state fulfill the requirement of a Markov property. Finite MDPs are mainly important in reinforcement learning. An MDP can be described formally with four components:

- A set of possible world states:
- S A set of possible actions: A(s) or A
- Model: T(s, a, s') ~ Probability(s'| s, a)
- A real-valued reward function: R(s) ~ R(s, a) ~ R(s, a, s')
- To understand this framework, we will use the Grid World example, as depicted in the following

Table Grid World

(1,3)	(2,3)	(3,3)	
(1,2)		(3,2)	
Smart	(2,1)	(3,1)	(4,1)

MDP has a set of states; it represents all the states that one can be in. In the Grid World example, it has 12 states and we can represent them in X and Y coordinates; say, the start state is (1,1) or the goal state is (4,4). Actually it doesn't matter whether we call these as 1, 2, 3 ..... 12 or A,B,C ... L. The point is that there are states that represent something and we should know which state we happen to be in. We can represent state as s. Action Next are actions--things that you can do in a particular state. Actions are the things I am allowed to execute when I am in a given state. What will be the actions in the Grid World? In the Grid World, we can take four types of actions: UP, DOWN, LEFT, and RIGHT. The point is that your action set will represent all the things that the agent, robot, or person we are trying to model is allowed to do. Now, in its generalized form, you can think of the set of actions one can take as the function state A(s). However, most of the time, people just treat it as set of actions or actions that are allowed on the particular state and represent it as model.

The third part of our framework is the model, sometime called transition model. It describes the rules of the games that apply, or is rather the physics of the world. It's basically a function of three variables: state, action, and another state. It produces the probability that you end up transitioning s' given that you were in state s and you took action a. Here, s' is the state where you end up and s and a are the given state and action, respectively. In our Grid World example, we are at the start state. The probability of going up is 0.8, the probability of going right is 0.1, and the probability that we end up where we started is 0.1. If we sum up all the probabilities, it becomes 1, and that's the way it works. The model is really an important thing and the reason for its importance is that it describes the rules of the game. It tells us what will happen if we do something in a particular place. It captures everything we can know about the transition: T(s, a, s') ~ Probability(s'| s, a) these processes are called Markov, because they have what is known as the Markov property. That is, given the current state and action, the next state is independent of all preceding actions and states. The current state captures all that is relevant about the world in order to predict what the next state will be. The effects of an action taken in a state depend only on that state and not on the prior history.

Reward:- It is a scalar value that you get from being in a state. There are three different definitions of rewards (R); sometimes it will be very useful to think about them in different ways. R(s) means we get a reward when entering into the state. R(s, a) means we get a reward when being in a state and taking an action. R(s, a, s') means we get a reward when being in a state, taking an action, and ending up in a new state. These are all mathematically equivalent

but it is easier to think about one form or another: R(s) ~ R(s, a) ~ R(s, a, s') the preceding four components define a problem; now, we'll look into the solution. The solution to the MDP is called policy.

Policy:-A plan or a result of classical planning can be either an ordered list of actions or a partially ordered set of actions meant to be executed without reference to the state of the environment. When we looked at conditional planning, we considered building plans with branches in them that observed something about the state of the world and acted differently depending on the observation. In an MDP, we can assume that it takes only one step to go from any one state to another. Hence, in order to be prepared, it is typical to compute a whole policy rather than a simple plan. A policy is a mapping from states to actions. It says, no matter what state you happen to find yourself in, here is the action that it's best to take now. Because of the Markov property, we'll find that the choice of action needs to depend only on the current state (possibly the current time as well) and not on any of the previous states. A policy is a function that takes state and returns an action. In other words, for any given state you are in, it tells you the action you should take: π(s) -> a MDP - more about rewards

Generally, in a reinforcement learning problem, the actions of the agent will give not only the immediate rewards but also the next state of the environment. The agent actually gets the immediate reward and the next state, and then agent needs to decide on further actions. Furthermore, the agent normally determines how it should take the future value into account; it's called model of long-run optimality. The agent also has to learn from the delayed rewards; it sometimes takes a long sequence of actions to retrieve an irrelevant reward and, after some time, it reaches a state with a high reward. The agent should also learn which of its actions give it a high reward based on the past history. This will help decide future actions. Let's take an example of a chess game. I played a long game of chess and it took 100 moves and at the end I lost the game. However, I actually lost the game on the 8th move; I made a mistake and reversed two moves because it was a new opening that I was just learning. From that point on, I played a beautiful game, but the truth is that the other player had an advantage that I never overcame. I lost the game, not because I played poorly but because I made one bad move and that move happened very early. This is the notion of a late reward. I played this long game of chess and maybe I played well and screwed up in the end. Or maybe I played a mediocre game but I had a couple of brilliant moves and that's why I won. Or maybe I played very well in the beginning and poorly at the end. Or the other way round! The truth is that you don't really know; all you know is that you take a bunch of actions and you get the reward signal back from the environment such as I won the game or I lost the game. The action Tuples and ultimately we have to figure out what action we took for the given state we were in. It helps to determine the ultimate sequence of rewards that we saw. This problem is called temporal credit assignment.

Now, we will look into the Grid World example from Chapter 1, Reinforcement Learning. Think about how we learn to get from the start state to the goal (+1) or failure (-1) depending on the kind of reward we see. In the Grid World example, the only change is in the rewards we receive for all states other than the goal (green) and failure (red) state. Let's say we give the reward for all the states as +2, and goal and failure have rewards of +1 and -1. Just to remind you of the rule, the game continues until we reach the goal or failure state: R(s) = +2 As the reward is +2, which is very high, and the target is to get the maximum rewards, in this case we will never get to the goal or failure state because the game ends as soon as it has reached to these states and that's the end of our treasure gathering:

The framework of the MDP has the following elements:

1. State of the system,
2. Actions
3. Transition probabilities
4. Transition rewards
5. A policy
6. A performance metric.

We assume that the system is modelled by a so-called abstract stochastic process called the Markov chain.

State: The “state” of a system is a parameter or a set of parameters that can be used to describe a system. For example the geographical coordinates of a robot can be used to describe its “state.” A system whose state changes with time is called a dynamic system. Then it is not hard to see why a moving robot produces a dynamic system. Another example of a dynamic system is the queue that forms in a supermarket in front of the counter. Imagine that the state of the queuing system is defined by the number ofpeople in the queue. Then, it should be clear that the state with time, and then this is dynamic system. It is to be understood that the transition from one state to another in an MDP is usually a random affair. Consider a queue in which there is one server and one waiting line. In this queue, the state x, Defined by the number of people in the queue, transitions to x + 1 with some probability and to x−1 with the remaining probability. The former type of transition occurs when a new customer arrives, while the latter event occurs when one customer departs from the system because of service completion.

Actions: Now, usually, the motion of the robot can be controlled, and in fact we are interested in controlling it in an optimal manner. Assume that the robot can move in discrete steps, and that after every step the robot takes, it can go North, go South, go East, or go West. These four options are called actions or controls allowed for the robot. For the queuing system discussed above, an action could be as follows: when the number of customers in a line exceeds some Prefixed number, (say 10), the remaining customers are diverted to a new counter that is opened. Hence, two actions for this system can be described as: (1) Open a new counter (2) Do not open a new counter.

Transition Probability: Assume that action a is selected in state i. Let the next state be j. Let p(i,a,j) denote the probability of going from state i to state j under the Influence of action a in one step. This quantity is also called the transition probability. If an MDP has 3 states

and 2 actions, there are 9 transition probabilities per action.

Immediate Rewards: Usually, the system receives an immediate reward (which could be positive or negative) when it transitions from one state to another. This is denoted by r(i,a,j).

Policy: The policy defined the action to be chosen in every state visited by the system. Note that in some states, no actions are to be chosen. States in which decisions are to be made, i.e., actions are to be chosen, are called decision- making states. In this tutorial, by states, we will mean decision-making states.

Performance Metric: Associated with any given policy, there exists a so-called performance metric — with which the performance of the policy is judged. Our goal is to select the policy that has the best performance metric. We will first consider the metric called the average reward of a policy. We will later discuss the metric called average reward. We will assume that the system is run for a long time and that we are interested in a metric measured over what is called the Infinite time horizon. Time of transition: We will assume for the MDP that the time of transition is unity (1), which means it is the same for every transition. Hence clearly 1 here does not have to mean 1 hour or minute or second. It is some fixed quantity fixed by the analyst. For the SMDP, this quantity is not fixed as we will see late

Bellman equation

Now that we have the utility and π*, we can actually do an even better job of writing out π*: π*(s) = argmaxa ∑s' T(s, a, s') Uπ*(s')

So, the optimal policy of a state is actually to look over all the actions and sum up the next state's transaction probability so that the probability ends up in the state's'. Now we have the utility of s' following the optimal policy. The preceding equation says that the optimal policy is one that, for every state, returns the action that maximizes my expected utility.

Now, this is rather circular, so it's basically a recursion. We will go through the exercise later in this chapter where we figured out the geometric series by effectively doing recursion. Now, I will write one more equation and we'll be one step closer to actually seeing it. Of course, if we are in the infinite horizon with a discounted state, even though we are one step closer, we won't actually be any closer:

The true utility of the state's then is the reward that I get for being in the state; plus, I am now going to discount all of the reward that I get from that point on.

Once we go to our new state's', we are going to look at the utility of that state. It's sort of modular recursion. We are going to look at overall actions and which action gives us the highest value of the state; it's kind of like the π* expression. Once we figure that out, we know what actions we are going to take in state's' and we are going to discount that because it just ups the gamma factor in all the rewards in the future. Then, we are going to add it to our immediate reward. In some sense, all I have done is kept substituting pieces back into one another. So, the true utility of being in a state is the reward you get in that state plus the discount of all the rewards you are going to get at that point, which, of course, is defined as the utility you are going to get for the states that you see; but each one of those is defined similarly. So, the utility you will get for s" will also be further discounted, but since it's multiplied by gamma that will be gamma squared. Then s"' will be gamma cubed, so that's just unrolling this notion of utility.

This is a very important equation, called the Bellman equation. This equation was invented by a Bellman, and in some scenes it turns out to be the key equation for solving MDPs and reinforcement learning. It is the fundamental recursive equation that defines the true value of being in some particular state. It accounts for everything that we care about in MDPs. The utilities themselves deal with the policy that we want, the gammas are discounted, and all the rewards are here. The transaction matrix is here representing the actions or all the actions we are going to take. So basically, the whole MDP is referenced inside of this and allows us, by determining utilities, to always know what the best action is of to take. If we can figure out the answer of the Bellman equation, the utilities of all the states, as perforce know what the optimal policy is. It becomes very easy.

Bellman was a very smart guy who took all the neat stuff of MDPs and put it in a single equation. Let's try to solve this equation, since it is clearly the key equation, the most important equation we are going to solve:

Uπ*(s) = R(s) + γ maxa ∑s' T(s, a, s') Uπ*(s')

We wrote this down as the utility of s. We have N states, which mean this isn't really one equation. y unknowns are there in the Bellman equation. The R's are known, the T's are known, so the only things missing are the Us. There are N equations in N unknowns. If the equation is linear, then we know how to solve N equations in N unknowns.

We will further look into the solutions of the Bellman equation in Chapter 3, Dynamic Programming, in the Value iteration and Policy iteration section.

MDP Framework

• S : states
• A : actions
• Pr(st+1 | st, at) : transition probabilities = Pr(st+1 | s0 … st, a0 … at) Markov property
• R(s) : real-valued reward

Find a policy: ∏: S → A

Maximize

• Myopic: E[r t | ∏, st] for all s
• Finite horizon: E[∑kt=0 rt | ∏, s0] – Non-stationary policy: depends on time
• Infinite horizon: E[∑∞t=0 γtrt | ∏, s0] – 0 < γ < 1 is discount factor – Optimal policy is stationary

This model has the very convenient property that the optimal policy is stationary. It's independent of how long the agent has run or will run in the future (since nobody knows that exactly). Once you've survived to live another day, in this model, the expected length of your life is the same as it was on the previous step, and so your behavior is the same

Markov Chain

• Markov Chain
• States

• Transitions
• Rewards
• No actions
• Value of a state, using infinite

If we set gamma to 0, then the values of the nodes would be the same as their rewards. If gamma were small but non-zero, then the values would be smaller than in this case and their differences more pronounced.

Value Iteration

The value function estimates the expected outcome from any given state, after any given action. The value function can be a crucial component of efficient decision-making, as it summarizes the long-term effects of the agent's decisions into a single number. The best action can then be selected by simply maximizing the value function

Initialize V0(s)=0, for all s

Loop for a while [until kVt –Vt+1k <ε(1-γ)/γ]

Loop for all

s Vt+1(s) = R(s) + maxa γ ∑s0 P(s0 | s, a) Vt(s)

•Converges to V *

•No need to keep VtvsVt+1

•Asynchronous (can do random state updates)

•Assume we want

•Gets to optimal policy in time polynomial in |A|, |S|, 1/(1-γ)

State Abstraction

In large worlds,

It is not possible to store distinct value for every individual state. State abstraction compresses the state in to a smaller number of features, which are the use in place of the complete state. Using state abstraction, the value function can be approximated by a parameterized function of the features, using many fewer parameters than there are states. Furthermore, state abstraction enables the agent to generalize between related states, so that as ingle out come can update the value of many states.

Temporality

In very large worlds state abstraction cannot usually provide accurate approximation to the value function. For example, there are10^{170} states in 19 × 19 Go. Even if the agent can store 10^{10} parameters, It is compressing the values of 10^{160}statesintoeveryparameter. The dea of temporality is to focus the agent's representation on the current region of the state space – the sub problem it is facing right now – rather than attempting to approximate the entire state space

Bootstrapping

Large problems typically entail making decisions with long-term consequences. Hundreds or thousands of time-steps may elapse before the final outcome is known. These outcomes depend on all of the agent's decisions, and on the world's uncertain responses to those decisions, throughout all of these time-steps. Bootstrapping provides a mechanism for reducing the variance of the agent's evaluation. Rather than waiting until the final outcome is reached, the idea of bootstrapping is to make can evaluation based on subsequent valuations. For example The temporal-difference learning algorithm estimates the current value from the estimated value at the next time-step.

Sample-Based Planning

The agent's experience with its world is limited, and may not be sufficient to achieve good performance in the world. The idea of sample-based planning is to simulate hypothetical experience, using a model of the world. The agent can use this simulated experience, in place of or in addition to its real experience, to learn to achieve better performance.

CONCLUSION

The MDP provided you have the simulator of the system or if you can actually experiment in the real-world system. Transition probabilities of the state transitions were not needed in this approach; this is the most attractive feature of this approach.

We did not discuss what is to be done for large-scale problems. That is beyond the scope of this tutorial. What was discussed above is called the lookup-table approach in which each Q-factor is stored explicitly (separately). For large-scale problems, clearly it is not possible to store the Q-factors explicitly because there is too many of them. Instead one stores a few scalars, called basis functions, which on demand can generate the Q-factor for any state-action pair. Function approximation when done improperly can become unstable

REFERENCES

[1] Chang, H., Fu, M., Hu, J., and Marcus, S. (2005). An adaptive sampling algorithm for solving Markov decision processes. Operations Research, 53(1):126–139.

[2] Chaslot, G., Chatriot, L., Fiter, C., Gelly, S., Hoock, J., Perez, J., Rimmel, A., and Teytaud, O. (2008a). Combining expert, online, transient and online knowledge in Monte-Carlo exploration.In 8th European Workshop on Reinforcement Learning.

[3] Chaslot, G., Winands, M., Szita, I., and van den Herik, (2008b). Parameter tuning by the cross entropy method. In 8th European Workshop on Reinforcement Learning.

[4]. Enzenberger, M. (2003). Evaluation in Go by a neural network using soft segmentation. In 10th Advances in Computer Games Conference, pages 97–108.

[5] Ernst, D., Glavic, M., Geurts, P., and Wehenkel, L.(2005). Approximate value iteration in the reinforcement learning context. application to electrical power system control. International Journal of Emerging Electric Power Systems, 3(1).

[6]. Graepel, T., Kruger, M., and Herbrich, R. (2001). Learning on graphs in the game of Go. In International Conference on Artificial Neural Networks, pages 347–352. Springer

[7]. Kearns,M.,Mansour,Y.,andNg,A.(2002). As parse sampling algorithm fornear-optimal planning in large Markova decision processes. Machine Learning, 49(2- 3):193–208.

[8] Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2004). Autonomous inverted helicopter flight via Reinforcement learning. In 9th International Symposium on Experimental Robotics, pages 363–372.

[9] P´eret, L. and Garcia, F. (2004). On-line search for solving Markov decision processes via heuristic sampling. In 16th European Conference on Artificial Intelligence, pages 530–534.

[10] S. J. Bradtke and M. Duff. Reinforcement learning methods for continuous- time Markov decision problems. In Advances in Neural Information Processing Systems 7.MIT Press, Cambridge, MA, USA, 1995.