CS 337, Fall 2023
Markov Decision Processes

Scribes: Om Godage * , Moningi Srija *, Ashwin Abraham *
Harsh Vardhan, Himanshu Devatwal, Shubham Sunil Mehroliya
Edited By: Vedang Asgaonkar

November 9, 2023

Recap

We discussed Bayesian Networks where we obtained a robust understanding of probabilistic modeling, decision-making under uncertainty, and the graphical representation of dependencies. All of which are integral elements in comprehending and approaching the challenges presented by MDPs.

1 MDPs

1.1 What are MDPs?

MDPs are a formal framework for modeling sequential decision making problems. They are a generalization of the search problems we have been studying so far.
The way MDPs differ is that the transitions are no longer deterministic. Instead, they are stochastic, i.e., they are governed by a probability distribution. This means that the agent does not have complete control over the outcome of its actions. The agent can only control the action it takes, but the outcome of that action is probabilistic. This is a more realistic model of the real world, where the agent does not have complete control over the outcome of its actions. For example, if you are driving a car, you can control the steering wheel, but you cannot control the outcome of your actions. The outcome of your actions depends on the road conditions, the weather, and other factors. MDPs are a formal framework for modeling such problems.

MDPs are defined by the following components:

1. A set of states S
2. A start state $s_{start} \in S$
3. A goal state $s_{goal} \in S$
4. A set of actions A
5. A transition function $T : S \times A \to S$ denoted by T(s' | s, a) where s represents current state, a represents the action taken at that state and s' represents the next state.

*Larger credit goes to these scribes for the final notes.

6. A reward function $R : S \times A \to \mathbb{R}$ denoted by R(s, a, s') where s represents current state, a represents the action taken at that state and s' represents the next state.

7. A discount factor $\gamma \in [0, 1]$

γ is a discount factor that determines how much the agent values future rewards. A discount factor of 0 indicates that the agent only cares about the immediate reward. A discount factor of 1 indicates that the agent cares about all future rewards equally.
Utility is a measure of the agent's happiness. The agent's goal is to maximize its utility. The utility of a state is the expected sum of rewards that the agent will receive in the future and it is given by the following equation:

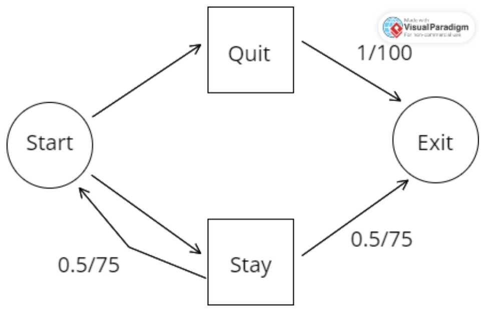
$$U(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s]$$

The model follows the Markov property, i.e., the future is independent of the past given the present. This means that the transition function T and the reward function R only depend on the current state and action, and not on the history of states and actions.

1.2 Example problem

Here we play a simple game to illustrate the concepts of MDPs.

The player chooses to either stay or quit. If the player quits, he gets \$100.
If the player stays, then a coin is flipped. If the coin comes up heads, the player leaves the game and gets \$0. If the coin comes up tails, the player gets \$75 and has to decide again whether to stay or quit.
This can be represented by the Influence Diagram. An Influence Diagram is a graphical model for decision analysis. When considering each edge, the notation 'x/y' is employed, where 'x' represents the transition probability, and 'y' indicates the corresponding reward.



If we choose to always stay, how much money (utility) do we expect to get?

$$U(\text{stay}) = \frac{75}{2} + \frac{150}{4} + \frac{225}{2^3} + \cdots$$

This forms a arithmetico-geometric series, we subtract the series from itself multiplied by $\frac{1}{2}$ to get:

$$\begin{aligned}
\frac{U(\text{stay})}{2} &= \frac{75}{2} + \frac{75}{4} + \frac{75}{2^3} + \cdots \\
&= 75\left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots\right) \\
\frac{U(\text{stay})}{2} &= 75 \cdot 1 \\
U(\text{stay}) &= 150
\end{aligned}$$

Which is the expected utility of staying.
How do we find an optimal way to choose actions in the above game? Here we get to the concept of policies.

1.3 Policies and Value Functions

There are two important concepts in MDPs: policies and value functions.
Policies are functions that map states to actions. They tell the agent what action to take in each state.
Value functions are functions that map states to values. They tell the agent how good it is to be in each state.

1. $V_\pi(s)$: Expected utility of being in state s and following policy π. Also called the state-value function.
2. $Q_\pi(s,a)$: Expected utility of taking action a in state s and then following policy π. Also called the action-value function.

We can define the value functions recursively in terms of each other.

$$V_\pi(s) = \begin{cases} 0 & \text{if } s \text{ is in a goal state} \\ Q_\pi(s,\pi(s)) & \text{otherwise} \end{cases}$$

$$Q_\pi(s,a) = \sum_{s'} T(s' \mid s,a)\left(R(s,a,s') + \gamma \cdot V_\pi(s')\right)$$

Continuing with our example, we can define the value functions for the game.
For the policy π that always stays:

$$\begin{aligned}
V_\pi(\text{quit}) &= 0 \\
V_\pi(\text{stay}) &= \frac{1}{2}(75 + V_\pi(start)) + \frac{1}{2}(75 + \cancelto{0}{V_\pi(quit)}) \\
V_\pi(\text{stay}) &= 150
\end{aligned}$$

This gives us a way to find utilities for a certain policy. But how do we find the optimal policy? And thus the optimal utilities?

1.4 Policy evaluation/iteration

We can find the optimal policy by using policy iteration.

Algorithm 1 Policy iteration

```
Initialize $V_\pi^{(0)}(s) \leftarrow 0$
for $t \leftarrow 1$ to $T$ do
    for every state $s$ do
        $V_\pi^t(s) \leftarrow \sum_{s'} T(s' \mid s,a)\left(R(s,a,s') + \gamma V_\pi^{t-1}(s')\right)$
    end
end
```

The horizon T is the number of iterations we run the algorithm for, which can either pre-defined or we can run the algorithm until the values converge to a certain threshold.

Given a policy, we now know how to compute its value using policy iteration. How do we find the optimal policy efficiently?

1.5 Value iteration

Value iteration is an efficient algorithm to find the optimal policy. It relies on the following Bellman optimality equation.

$$V^*(s) = \max_a \sum_{s'} T(s' \mid s,a)\left(R(s,a,s') + \gamma V^*(s')\right)$$

Algorithm 2 Value iteration

```
Result: Optimal policy
Initialize $V^0 \leftarrow 0$ for $t \leftarrow 1$ to $T$ do
    for every state $s$ do
        $V^t(s) \leftarrow \underset{a\in A}{max} \sum_{s'} T(s' \mid s,a)\left(R(s,a,s') + \gamma V^{t-1}(s')\right)$
    end
end
```

We can find the optimal policy from the optimal state-value function using the following equation:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

This algorithm is guaranteed to converge to the optimal policy, given two assumptions:

1. The discount factor $\gamma < 1$
2. The MDP graph is acyclic

The proof of which is beyond the scope of this course.

1.6 Glimpse of reinforcement learning

What if we don't know the transition probabilities and rewards? This is exactly the birth ground of reinforcement learning.
There are 2 problems reinforcement learning tries to solve:

1. **Prediction**: Given a policy π, find the state-value function V_π

2. **Control**: Find the optimal policy π^* and the optimal state-value function V^*

Conclusion

In this course, we began with Linear Regression and then to address the problems of overfitting and underfitting we used Ridge and Lasso regression. We understood the Bias-Variance tradeoff. Then we stepped into classification, starting with Logistic Regression where we used Linear Regression and sigmoid function. We then dived into Decision Trees and Random Forests. The Perceptron Classifier shed light on neuron-like models. SVM entered the scene, and we used kernels for non-linear decision boundaries. Neural Networks took center stage, from McCulloch-Pitts neurons to advanced architectures, paving the way for Convolutional and Recurrent Neural Networks. Then we got ourselves into clustering with K-Means, stable marriage problems, and Expectation-Maximization. Techniques like PCA and Autoencoders simplified complex data. Boosting, bagging, and AI search algorithms enriched our toolkit, while logic and reasoning led us to Bayesian Networks and Markov Decision Problems.
In summary, this course provided a holistic exploration of Machine Learning and Artificial Intelligence.