Reinforcement Learning and Autonomous Systems (CS4122)

Lecture 19 (28/09/2024)
Lecture 20 (30/09/2024)

Instructor: Gourav Saha

Lecture Content

(This lecture is the beginning of Module 3)

- Learning in MDPs.
  - Need for learning.
  - Fundamental idea behind learning in MDPs.
  - Generalized Policy Iteration.
- Monte Carlo Policy Evaluation.
  - Estimating the value function.
- Temporal Difference (TD) Policy Evaluation.
  - Estimating the value function.

Learning in MDPs

Recall that an MDP is defined by:

- States and the associate state space.
- Actions and the associated actions space.
- Reward.
- State transition probability.
- Reward probability.

Learning in MDPs : Need for Learning

Recall that an MDP is defined by:

- States and the associate state space.
- Actions and the associated actions space.
- Reward.
- State transition probability.
- Reward probability.

The need for learning in MDPs arises in two different scenarios:

- Scenario 1: **Involved probability distributions are not known**. For MDPs, state transition probability and reward probability are the involved probability distribution. Example:
  - Logistic networks (Amazon).
  - Scheduling in wireless networks.

In reality, some amount of knowledge of the involved probability distributions are known.

Learning in MDPs : Need for Learning

Recall that an MDP is defined by:

- ➢ States and the associate state space.
- ➢ Actions and the associated actions space.
- ➢ Reward.
- ➢ State transition probability.
- ➢ Reward probability.

The need for learning in MDPs arises in two different scenarios:

- ➢ Scenario 2: **The models are known but they are too complex** to the point that using any tools like value/policy iteration is impractical. The complexity arises because:
  - State space is too large (this increase the time complexity of value/policy iteration). Example: Atari games.
  - The setup does not explicitly have an "MDP friendly" model. These setups, though MDPs, are better understood in terms of ODEs (or stochastic ODEs). Example: Bicycles, quadcopters.

Learning in MDPs : Need for Learning

Recall that an MDP is defined by:

- States and the associate state space.
- Actions and the associated actions space.
- Reward.
- State transition probability.
- Reward probability.

The need for learning in MDPs arises in two different scenarios:

- **Scenario 1**: **Involved probability distributions are not known**.
- **Scenario 2**: **The models are known but they are too complex** to the point that using any tools like value/policy iteration is impractical.

For scenario 2, we can learn the optimal policy by simulating the model in computer and using this simulation as a proxy of real-world.

For scenario 1, we still need some model to simulate in computer. Otherwise, direct real-world deployment can be risky.

Learning in MDPs : Need for Learning

Given: A threshold, θ.

(S1): For all $x \in \mathcal{S}$, initialize $V(x)$ arbitrarily to any real value. Initialize $\Delta = \infty$.

(S2): while $\Delta > \theta$:

(S3): for all $x \in \mathcal{S}$:

(S4): $$V_{new}(x) = \max_{a \in \mathcal{A}(x)} \left(r(x,a) + \beta \sum_{x' \in \mathcal{S}} P[x' \mid x, a] \, V(x') \right)$$

(S5): Compute $\Delta = \max_{x \in \mathcal{S}} |V_{k+1}(x) - V_k(x)|$.

(S6): Set $V(x) = V_{new}(x)$ for all $x \in \mathcal{S}$.

(S7): For all $x \in \mathcal{S}$:

$$\pi^*(x) = \operatorname*{argmax}_{a \in \mathcal{A}(x)} \left(r(x,a) + \beta \sum_{x' \in \mathcal{S}} P[x' \mid x, a] \, V(x') \right)$$

(S8): Return π^*.

- This is the pseudocode for value iteration.
- When $r(x,a)$ and $P[x' \mid x, a]$ are not known, we can't use value iteration. This explains scenario 1 discussed in the previous slides.
- When the model is complex, i.e. too many states, this for loop which in turn is inside this while loop becomes the computational bottleneck. This explains scenario 1 discussed in the previous slides.

Learning in MDPs : Need for Learning

Given: A threshold, θ.

(S1): For all $x \in \mathcal{S}$ arbitrarily initialize: (i) a policy $\pi(x)$ to any value in $\mathcal{A}(x)$, and (ii) a value function $V(x)$ to any real value. Also, set $converged = False$.

(S2): while not(converged):

(S3): Set Use IPE to compute the value function $V^{\pi}(x)$ for all $x \in \mathcal{S}$ corresponding to current policy π. Set the initial value function of IPE as $V(x)$ and the conversion threshold as θ.

(S4): For all $x \in \mathcal{S}$:

(S5): Compute,

$$\tilde{a} = \underset{a \in \mathcal{A}(x)}{\operatorname{argmax}} \left(r(x,a) + \beta \sum_{x' \in \mathcal{S}} P[x' \mid x, a] V^{\pi}(x') \right)$$

$$q^{\pi}(x, \tilde{a}) = r(x, \tilde{a}) + \beta \sum_{x' \in \mathcal{S}} P[x' \mid x, \tilde{a}] V^{\pi}(x')$$

(S6): If $q^{\pi}(x, \tilde{a}) > V^{\pi}(x)$, then set $\pi_{new}(x) = \tilde{a}$. Else set $\pi_{new}(x) = \pi(x)$.

(S7): If $\pi_{new}(x) = \pi(x)$ for all $x \in \mathcal{S}$, set $converged = True$.

(S8): Set $\pi(x) = \pi_{new}(x)$ and $V(x) = V^{\pi}(x)$ for all $x \in \mathcal{S}$.

(S9): Return π.

- This is the pseudocode for policy iteration.
- Similar explanation as value iteration discussed in the previous slides.

Learning in MDPs: Fundamental idea behind learning in MDPs

$$\pi_{new}(x) = \underset{a \in \mathcal{A}(x)}{\text{argmax}} \; q^{\pi}(x, a)$$

- Policy iteration shows that if we can compute the Q-function, $q^{\pi}(x, a)$, corresponding to the current policy, π, then we can use the above equation to improve the policy.
- In module 2, we had $r(x, a)$ and $P[x' \mid x, a]$ and hence we could use value and policy iteration. But we can't do that in a learning setup.
- Another way to compute the Q-function is to realize the Q-function $q(x, a)$ is just the expected value of the return G_t starting from state x and action a. **We can estimate the Q-function by finding the sample average of G_t.** Provide that the initial state is x and initial action is a.

This is the fundamental idea of learning in MDPs.

Learning in MDPs: Generalized PI

- Generalized policy iteration (GPI) for the foundation of learning in MDPs.

Generalized Policy Iteration (not rigorous but a broadly observed trend):

- $q^\pi(x, a)$ does not have to be computed exactly (few rounds of IPE; maybe even one round which is VI).

- $q^\pi(x, a)$ does not have to be updated for all (x, a) pairs.

- Policy update:

$$\pi_{new}(x) = \underset{a \in \mathcal{A}(x)}{\operatorname{argmax}}\ q^\pi(x, a)$$

does not have to happened for all x.

Learning in MDPs: Generalized PI

- Generalized policy iteration (GPI) for the foundation of learning in MDPs.

Generalized Policy Iteration (not rigorous but a broadly observed trend):

- Till the time policy evaluation to compute $q^{\pi}(x, a)$ keeps happening for all (x, a) pairs infinitely often, AND
- Till the time policy update:

$$\pi_{new}(x) = \underset{a \in \mathcal{A}(x)}{\operatorname{argmax}}\ q^{\pi}(x, a)$$

keeps happening for all x infinitely often, THEN

- **Convergence to optimal policy is guaranteed.**

MC Policy Evaluation

To compute the value function $V^{\pi}(x)$

- The main objective is to compute the optimal policy which in turn relies on computing the Q-function for a given policy.

- But to make is simpler, we will start by computing the value function for a given policy just to get an overall idea. Remember: Computing value function for a given policy is called policy evaluation.

MC Policy Evaluation: Trajectory

- Whenever we are in a "learning" setup, there is always going to be a concept if **"sampling"** from the environment. The question is, what to sample?

  - We have to sample a **trajectory**. A trajectory is denoted using τ. A trajectory is a **sequence** of **state, action, and reward pairs** obtained in an episode.

$$\underbrace{(x_0, a_0, r_0)}_{t=0}, \underbrace{(x_1, a_1, r_1)}_{t=1}, \underbrace{(x_2, a_2, r_2)}_{t=2}, \ldots, \underbrace{(x_T, a_T, r_T)}_{t=T}$$

  - In the trajectory shown above, T is the last time slot of the episode. Since a trajectory is a sequence, it's short hand notation is,

$$\{(x_t, a_t, r_t)\}_{t=0}^{T}$$

  - A trajectory τ is a random variable (that's why we are sampling it) which itself consists of other random variables. So essentially, τ can be characterized by a joint distribution. This joint distribution is dependent on the policy, reward probability, and state transition probability.

  - A pseudocode to collect the trajectory is shown in the next slide.

MC Policy Evaluation: Psuedocode to Generate Trajectory

Given: A policy, π.

(S1): Reset the environment to get the initial state x_0. Initialize time $t = 0$, and an empty list τ that will contain the trajectory for the current episode.

(S2): while episode did not end:

(S3): Use policy π for the current state x_t to choose action a_t.

(S4): Take action a_t. Environment will return reward r_t and transition to next state x_{t+1}.

(S5): Append the state, action, reward pair (x_t, a_t, r_t) to τ. Set $t = t + 1$.

(S6): Return trajectory τ.

MC Policy Evaluation

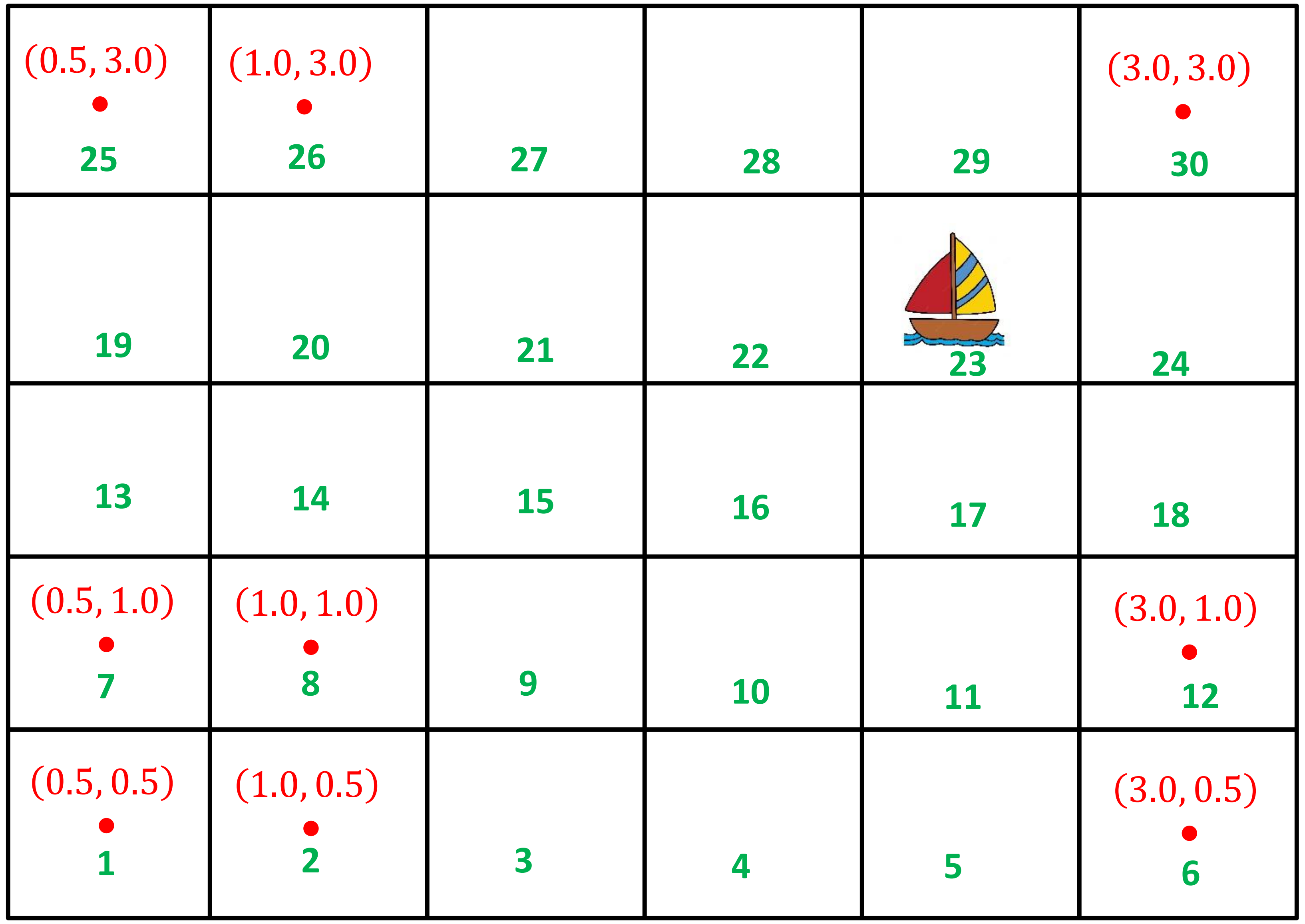
To compute the value function $V^{\pi}(x)$

- The notes are not complete. You have to read the book. **That said, all the psuedocodes are there in the slides.** You have to refer the books to understand these psuedocodes.

- The following are the important concepts:
  - **First-visit and every-visit Monte Carlo:** Read chapter 5. Start from the beginning and read section 5.1. You don't have to read example 5.2 (soap bubble) is you don't want to. The psuedocode of every-visit Monte Carlo is not there in the book however it is there in this slide.

MC Policy Evaluation: Trajectory

Fishing in Grid World



I did an example of collecting trajectories for first visit and every visit monte carlo during lecture. You can get the lecture notes from some one else.

Trajectory 1:

Trajectory 2:

MC Policy Evaluation: Psuedocode for First-Visit MC

Given: A policy, π.

(S1): For all $x \in \mathcal{S}$ arbitrarily initialize: *(i)* $V(x)$ to any real value, and *(ii)* $N(x)$ to zero. $V(x)$ and $N(x)$ are the value function and the number of samples corresponding to state x respectively.

(S2): For every episode:

For step (S3), we use the psuedocode to generate a trajectory from this slide.

(S3): Use policy π to generate a trajectory τ. Let the trajectory be as follows with the last time slot as T:

$$(x_0, a_0, r_0)\,,\ (x_1, a_1, r_1)\,,\ (x_2, a_2, r_2)\,, \ldots,\ (x_T, a_T, r_T)$$

(S4): For all $x \in \mathcal{S}$, set $visited(x)$ to $False$. Set the return corresponding to time $\mathrm{t} = 0$ as follows,

$$G_0 = \sum_{t=0}^{T} \beta^t r_t$$

(S5): For $t = 0, 1, 2, \cdots, T$:

(S6): If not$\big(visited(x_t)\big)$:

(S7): Update $V(x_t)$ as follows:

$$V(x_t) = V(x_t) + \frac{1}{N(x_t) + 1}\left(G_t - V(x_t)\right)$$

(S8): Update $N(x_t) = N(x_t) + 1$. Also set, $visited(x_t)$ to True.

(S9): Set $G_{t+1} = (G_t - r_t)/\beta$.

MC Policy Evaluation: Psuedocode for Every-Visit MC

Given: A policy, π.

(S1): For all $x \in \mathcal{S}$ arbitrarily initialize: *(i)* $V(x)$ to any real value, and *(ii)* $N(x)$ to zero. $V(x)$ and $N(x)$ are the value function and the number of samples corresponding to state x respectively.

(S2): For every episode:

For step (S3), we use the psuedocode to generate a trajectory from this slide.c

(S3): Use policy π to generate a trajectory τ. Let the trajectory be as follows with the last time slot as T:

$$(x_0, a_0, r_0)\,,\ (x_1, a_1, r_1)\,,\ (x_2, a_2, r_2)\,, \ldots,\ (x_T, a_T, r_T)$$

(S4): ~~For all $x \in \mathcal{S}$, set $visited(x)$ to $False$~~. Set the return corresponding to time $t = 0$ as follows,

$$G_0 = \sum_{t=0}^{T} \beta^t r_t$$

(S5): For $t = 0, 1, 2, \cdots, T$:

(S6): ~~If not$(visited(x_t))$:~~

(S7): Update $V(x_t)$ as follows:

$$V(x_t) = V(x_t) + \frac{1}{N(x_t) + 1}\left(G_t - V(x_t)\right)$$

(S8): Update $N(x_t) = N(x_t) + 1$. ~~Also set, $visited(x_t)$ to True.~~

(S9): Set $G_{t+1} = (G_t - r_t)/\beta$.

MC Policy Evaluation: Psuedocode for Every-Visit MC

Given: A policy, π.

(S1): For all $x \in \mathcal{S}$ arbitrarily initialize: *(i)* $V(x)$ to any real value, and *(ii)* $N(x)$ to zero. $V(x)$ and $N(x)$ are the value function and the number of samples corresponding to state x respectively.

(S2): For every episode:

For step (S3), we use the psuedocode to generate a trajectory from this slide.

(S3): Use policy π to generate a trajectory τ. Let the trajectory be as follows with the last time slot as T:

$$(x_0, a_0, r_0)\,,\ (x_1, a_1, r_1)\,,\ (x_2, a_2, r_2)\,, \ldots,\ (x_T, a_T, r_T)$$

(S4): Set the return corresponding to time $\mathrm{t} = 0$ as follows,

$$G_0 = \sum_{t=0}^{T} \beta^t r_t$$

(S5): For $t = 0, 1, 2, \cdots, T$:

(S6): Update $V(x_t)$ as follows:

$$V\left(x_t\right) = V\left(x_t\right) + \frac{1}{N\left(x_t\right) + 1}\left(G_t - V\left(x_t\right)\right)$$

(S7): Update $N(x_t) = N(x_t) + 1$.

(S8): Set $G_{t+1} = (G_t - r_t)/\beta$.

MC Policy Evaluation

Gradient Descent Viewpoint

- Recorded a small video (10 minutes) to explain this idea:

https://youtu.be/UtsV835_ri4

MC Policy Evaluation: Psuedocode for Every-Visit MC

Given: A policy, π.

(S1): For all $x \in \mathcal{S}$, arbitrarily initialize $V(x)$ to any real value. $V(x)$ is the value function corresponding to state x.

(S2): For every episode:

(S3): Use policy π to generate a trajectory τ. Let the trajectory be as follows with the last time slot as T:

$$(x_0, a_0, r_0), (x_1, a_1, r_1), (x_2, a_2, r_2), \ldots, (x_T, a_T, r_T)$$

(S4): Set the return corresponding to time $t = 0$ as follows,

$$G_0 = \sum_{t=0}^{T} \beta^t r_t$$

(S5): For $t = 0, 1, 2, \cdots, T$:

(S6): Update $V(x_t)$ as follows:

$$V(x_t) = V(x_t) + \alpha (G_t - V(x_t))$$

(S7): Set $G_{t+1} = (G_t - r_t)/\beta$.

α can vary with episode and time slot.

Temporal Difference Policy Evaluation

- The notes are not complete. You have to read the book. **That said, all the psuedocodes are there in the slides.** You have to refer the books to understand these psuedocodes.

- The following are the important concepts:
  - **Temporal Difference (TD) Policy Evaluation:** Read chapter 6. Start from the beginning and read sections 6.1 and 6.2 completely.

TD Policy Evaluation: Psuedocode

Given: A policy, π.

(S1): For all $x \in \mathcal{S}$, arbitrarily initialize $V(x)$ to any real value. $V(x)$ is the value function corresponding to state x.

(S2): For every episode:

(S3): Reset the episode. This will give the current state x.

(S4): while episode did not end:

(S5): Use policy π for the current state x to choose action a.

(S6): Take action a. Environment will return reward r and transition to next state x'.

(S7): Update $V(x)$ as follows:

$$V(x) = V(x) + \alpha\left(r + \beta V\left(x'\right) - V(x)\right)$$

α can vary with episode and time slot.

(S8): Set $x = x'$, i.e. set current state equal to next state.

Thank you