Notes CME241

Raphael Abbou

March 20, 2020

1 Markov Decision Processes

1.1 Definitions

A state S_t represent all the information at time t.

$$\mathbb{P}(S_{t+1}|S_t) = \mathbb{P}(S_{t+1}|S_1, ...S_t)$$

We define the state transition matrix by $\mathcal{P}_{ss'} = \mathbb{P}(S_{t+1} = s'|S_t = s)$

Definition 1.1. *Markov Process (MP):* $< \mathcal{S}, \mathcal{P} >$
Markov Reward Process (MRP): $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$
Markov Decision Process (MDP): $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

For a MRP:
- $\mathcal{R}$ is a reward function: $\mathcal{R}_s = \mathbb{E}[R_{t+1}|S_t = s]$
- $\gamma \in [0, 1]$ is the discount factor
- $G_t = \sum_k \gamma^k T_{t+k+1}$ is the return

For a MDP:
- $\mathcal{A}$ is the set of actions
- The probabilities of transition depends on the action: $\mathbb{P}^a_{ss'}, \mathcal{R}^a_s$

Definition 1.2. *For MDP, a policy π is a distribution over actions given a state:*

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

Definition 1.3. *For a MDP, we define:*
- *State-value function:*

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

- *Action-value function:*

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

A MDP with a policy π gives:
- A MP $< S, \mathcal{P}^\pi >$
- A MRP $< S, \mathcal{P}^\pi, \mathcal{R}^\pi >$

1.2 Bellman's Equations

We define Bellman's equation:

For a MRP:

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \ inS} \mathcal{P}_{ss'} v(s')$$

Or in its matrix form:

$$v = \mathcal{R} + \gamma \mathcal{P} v, v \ in \mathbb{R}^n$$

The resolution of this equation takes $O(n^3)$ (matrix inversion). For large MDP, it can be solved by DP, MC evaluation, or Temporal Difference learning.

For a MDP:

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

We are looking for:
- optimal state value $v^*(s) = \max_\pi v_\pi(s)$
- optimal action value $q_*(s,a) = \max_\pi q_\pi(s,a)$

How to find the associated optimal policy π^* ? By $a = argmax_{a \in \mathcal{A}} q^*(s,a)$. But we have to know $q^*(s,a)$...

$\Rightarrow$ Obtain it recursively by Bellman Equations

Definition 1.4 (Bellman's Optimality Equations)**.**

$$v^*(s) = \max_a \mathcal{R}_s^a + \sum_{s' \ inS} \mathcal{P}_{ss'}^a v^*(s')$$

$$q^*(s,a) = \mathcal{R}_s^a + \sum_{s' \in S} \mathcal{P}_{ss'}^a \max_{a'} q^*(s',a')$$

Iterative solution methods: Value Iteration, Policy Iteration, Q-learning, SARSA

2 Dynamic Programming Algorithms

Dynamix Programming Algorithms are useful to compute (optimal) Policies and Values Functions for a MDP for which we know transition probabilities and rewards.

Definition 2.1 (Policy Evaluation)**.** *Gets v_π from iterative application of Bellman's equation. We iterate over $v_1 \to v_2 \to ... \to v_k$. This converges to v_π. We use at each step:*

$$v_{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_k$$

Definition 2.2 (Policy Iteration)**.** *Once we have v_π, we choose a π' which greedily improves $v\pi$. We thus generate $\pi' \geq \pi$ with:*

$$\forall s', \pi'(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s,a)$$

In Policy Iteration, we alternate between getting v_π through Policy Evaluation, given a policy π, then finding a better policy π' given those state-values. We then compute $v_{\pi'}$, and iterate... until we get v^*, π^*.

Definition 2.3 (Value Iteration)**.** *Same as Policy Evaluation, but we stop at the first step ($k = 1$), and we use at each step the Bellman's Optimality Equation to get $v_{k+1}(s)$ from the $v_k(s)$.*
We get $v_1 \to v_2 \to ... \to v_k$, which converges to v^*. We use at each step:

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \mathcal{P}_s^a v_k$$

We also define Asynchronous Dynamic Programming, which will update inplace the state-values, using most up-to-date values for other states:

$$v_{new}(s) \leftarrow \max_{a \in ac}(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{new}(s'))$$

3 Model Free Prediction

Model Free Prediction aims at estimating Value Functions for MDP whose transitions and rewards are unknown. It samples episodes from the MDP to establish those estimates.

3.1 Monte Carlo Policy Evaluation

A Monte Carlo simulation update $V(S_t)$ towards actual return G_t of the episode:

1. Sample an episode $t = 0, ...t = T$ until we arrive to an end state
2. For each time step t, $G_t = R_{t+1} + \gamma R_{t+2} ... + \gamma_{T-1} R_T$
3. Increment Counter and Total Reward for each time step (*):
   $N(S_t) \leftarrow N(S_t) + 1$
   $TotalReward(S_t) \leftarrow TotalReward(S_t) + G_t$
4. Update the value function $V(S_t) \leftarrow V(S_t)/N(S_t)$

We can distinguish Fist-Visit MC which update the Value Function for state s only for the first t in which we step in s, while Every-Visit Monte Carlo will update $V(s)$ each time we step into s, with the forward cumulative reward G_t if we step in s at time t.

Instead of (*), we can directly update the Value Function by Incremental Monte-Carlo Updates:
$N(S_t) \leftarrow N(S_t) + 1$
$V(S_t) \leftarrow V(S_t) + \frac{1}{N_t}(G_t - V(S_t))$

3.2 Temporal Difference Learning

TD can learn from incomplete sequence, and update the Value Funtion at each step, before arriving at the end of the sequence (in contrats to MC). TD has way lower variance than MC, but some bias.

TD(0): Update $V(S_t)$ toward estimated return (TD target) $R_{t+1} + \gamma V(S_t)$:

$$V_(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

n-step TD: Update towards the n-step estimated return $G_t^{(n)} = G_{t:t+n}$ for each t in the episode:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^n R_{t+n} + \gamma^{n+1} V(S_{t+m})$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

Forward TD(λ): Update toward the exponentially weighted estimated return: $G_T^{(\lambda)}$:

$$G_t^{(\lambda)} = (1-\lambda) \sum_{n=1}^{T-t-1} G_{t:t+n} + \lambda^{T-t-1} G_{t:T}$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(\lambda)} - V(S_t))$$

Backward TD(λ): Same principle but use a computation trick, the Eligibility Trace $E_t(s)$. At each time step t in the episode, and for each state s, we update the Eligibility Trace:

$$E_{t=0}(s) = 0$$
$$E_t(s) = \gamma * \lambda * E_{t-1}(s) + \mathbb{1}_{S_t=s}$$

We then update the Value Function for the state visited at time t:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t E_t(S_t)$$

The Eligibility Trace can be seen how how much to blame a given state for why we currently receive the current expected reward.

Theorem 3.1. *For each state s, the sum of offline updates is identical for forward and backward views TD(λ)*

Offline: Updates are accumulated at each episode and applied at the end.

Online: Updates are applied at each time step within the episode.

4 Model Free Control

Greedy policy improvement on Value Function requires a know MDP (we use $\mathcal{P}_{ss'}^a$ in its computation), while greedy improvement over $Q(s,a)$ is model-free $\implies$ we will focus on Q-values in the following algorithms.

4.1 On-Policy Learning

On-Policy learns about π by sampling from this policy.

Monte Carlo Policy Evaluation: Find optimal Q^*, π^*

1. Policy Evaluation: Evaluate Q-Values with Monte-Carlo simulations
2. Policy Improvement: ϵ-greedy policy improvement
3. Repeat until convergence

Monte Carlo Policy Control: Same as above, but the Monte-Carlo simulation only lasts one episode $\rightarrow$ faster convergence.

On Policy improvement: we follow a policy π (on-policy) to compute values $Q_\pi(s,a)$, but we allow random actions at each step for exploration purposes:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{m}, & \text{if } a = \arg\max_{a \in \mathcal{A}} Q_\pi(s,a) \\ \frac{\epsilon}{m}, & \text{otherwise} \end{cases}$$

Theorem 4.1. *For any policy π, the ϵ-greedy related policy π' with respect to Q_π is an improvement: $V_{\pi'}(s) \geq V_\pi(s)$*

SARSA: Find optimal Q^*, π^*, Q-values and policy are update at the same time. For each episode, start at state S and:

1. Sample A from ϵ-greedy, observe R, S' and sample A' in that new state
2. Update Q-value: $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S', A') - Q(S,A))$
3. $S \leftarrow S'$, $A \leftarrow A'$, repeat until end of episode

With analogy with Model Free Prediction, we define n-step SARSA, Foward View SARSA(λ), Backward SARSA(λ) with Eligibility Trace...

4.2 Off-Policy Learning

Off-Policy learns about π by sampling episodes from an 'experienced' policy μ. We can use Importance Sampling but it is not stable and used in practice.

Q-learning: We want to evaluate Q for actions taken by our target policy π, but the episodes are generated by a behavioral policy μ:

1. Sample A_{t+1} from $\mu(\cdot|S_{t+1})$ and land in S_{t+2}, and observe R_{t+1}
2. Update the Q-Value with hypothetical action A' drown from $\pi(\cdot|S_{t+1})$
   $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A))$

The well-known Q-value update comes when we consider the target policy π to be greedy with respect to Q, and the behavioral policy μ to be ϵ-greedy with respect to μ.

We get the update for each time step t generate with ϵ-greedy:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \max_{A' \in \mathcal{A}} Q_\pi(S_{t+1}, A') - Q(S_t, A_t))$$

5 Utility Theory and Applications

5.1 Utility Theory

Utility functions are concave, and their concavity represent the Risk-Aversion of agents.

Definition 5.1 (Certainty Equivalent Value)**.**

$$x_{CE} = U^{-1}(\mathbb{E}[U(x)])$$

Absolute Risk Premium:

$$\pi_A = \mathbb{E}[x] - x_{CE}$$

Relative Risk Premium:

$$\pi_A = \frac{\mathbb{E}[x] - x_{CE}}{\mathbb{E}[x]}$$

Definition 5.2 (CARA)**.** *Constant Absolute Risk Aversion*
Utility: $U(x) = \frac{e^{-ax}}{a}$
Absolute Risk Aversion: $A(x) = -\frac{U''(x)}{U(x)} = a$

With CARA and $x \sim \mathcal{N}(\mu, \sigma^2)$:

$$x_{CE} = \mu - \frac{a\sigma^2}{2}$$

Definition 5.3 (CRRA)**.** *Constant Relative Risk Aversion*
Utility: $U(x) = \frac{x^{1-\gamma}}{1-\gamma}$
Relative Risk Aversion: $A(x) = -\frac{U''(x)x}{U(x)} = \gamma$

With CRRA and $x \sim \mathcal{N}(\mu, \sigma^2)$:

$$x_{CE} = \frac{\mu - r}{a\sigma^2}$$