Model Free RL

Ali Gooya

Reference for slides:

David Silver (Deep Mind)

(no exam content)

Model Free Reinforcement Learning

- Last Lecture
  - Planning by Dynamic Programming
  - Solves a **known** MDP
- Model free prediction
  - Estimate value function of an **unknown** MDP
- Model free control
  - Optimize the value of an **unknown** MDP

Monte-Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

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

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

First Time Monte-Carlo Policy Evaluation

- To evaluate state s
- The first time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

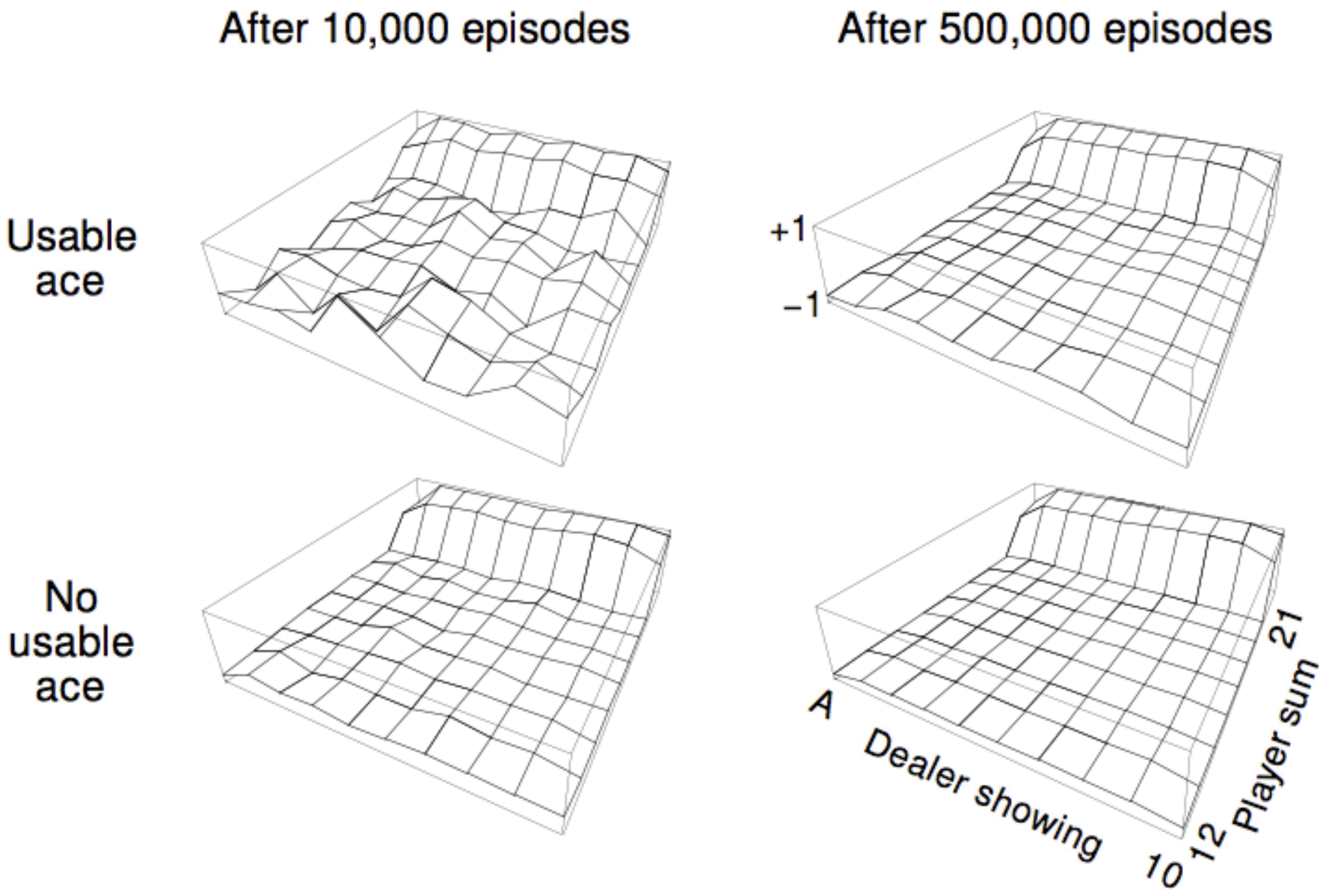
Every-Visit Monte-Carlo Policy Evalution

- To evaluate state s
- Every time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow v_{\pi}(s)$ as $N(s) \rightarrow \infty$

Blackjack Example

- States (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a "useable" ace? (yes-no)
- Action stick: Stop receiving cards (and terminate)
- Action twist: Take another card (no replacement)
- Reward for stick:
  - $+1$ if sum of cards $>$ sum of dealer cards
  - 0 if sum of cards $=$ sum of dealer cards
  - -1 if sum of cards $<$ sum of dealer cards
- Reward for twist:
  - -1 if sum of cards $>$ 21 (and terminate)
  - 0 otherwise
- Transitions: automatically twist if sum of cards $<$ 12

Value-Function Learning



Policy: stick if sum of cards ≥ 20, otherwise twist

Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally,

$$\begin{aligned}
\mu_k &= \frac{1}{k} \sum_{j=1}^{k} x_j \\
&= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\
&= \frac{1}{k} \left(x_k + (k-1)\mu_{k-1} \right) \\
&= \mu_{k-1} + \frac{1}{k} \left(x_k - \mu_{k-1} \right)
\end{aligned}$$

Incremental MC updates

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left(G_t - V(S_t)\right)$$

- In non-stationary problems, it can be useful to track a run mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

Temporal Difference learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

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

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Policy optimization

- Recall the Bellman's optimality equation for action-value function:

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} P_{ss'}^a \max_{a'} q_*(s',a')$$

- We want to learn q_* without having accessing to transition probabilities (model free). We run a stochastic version:
  - At state s take action a to arrive at random s'
  - Update using:

$$q(s,a) \leftarrow q(s,a) + \alpha\Big(\mathcal{R}_s^a + \gamma \max_{a'} q(s',a') - q(s,a)\Big)$$

Policy optimization

- In order make sure that we converge to true q_*, we want to make sure every state is visited by using some randomness.

- This is achieved with ϵ-Greedy policy (allowing continual exploration):

  - With probability ϵ choose an action in random

  - With probability $1-\epsilon$ choose $a = \arg\max_{a'} q(s', a')$

Q-Learning

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$;
    until S is terminal

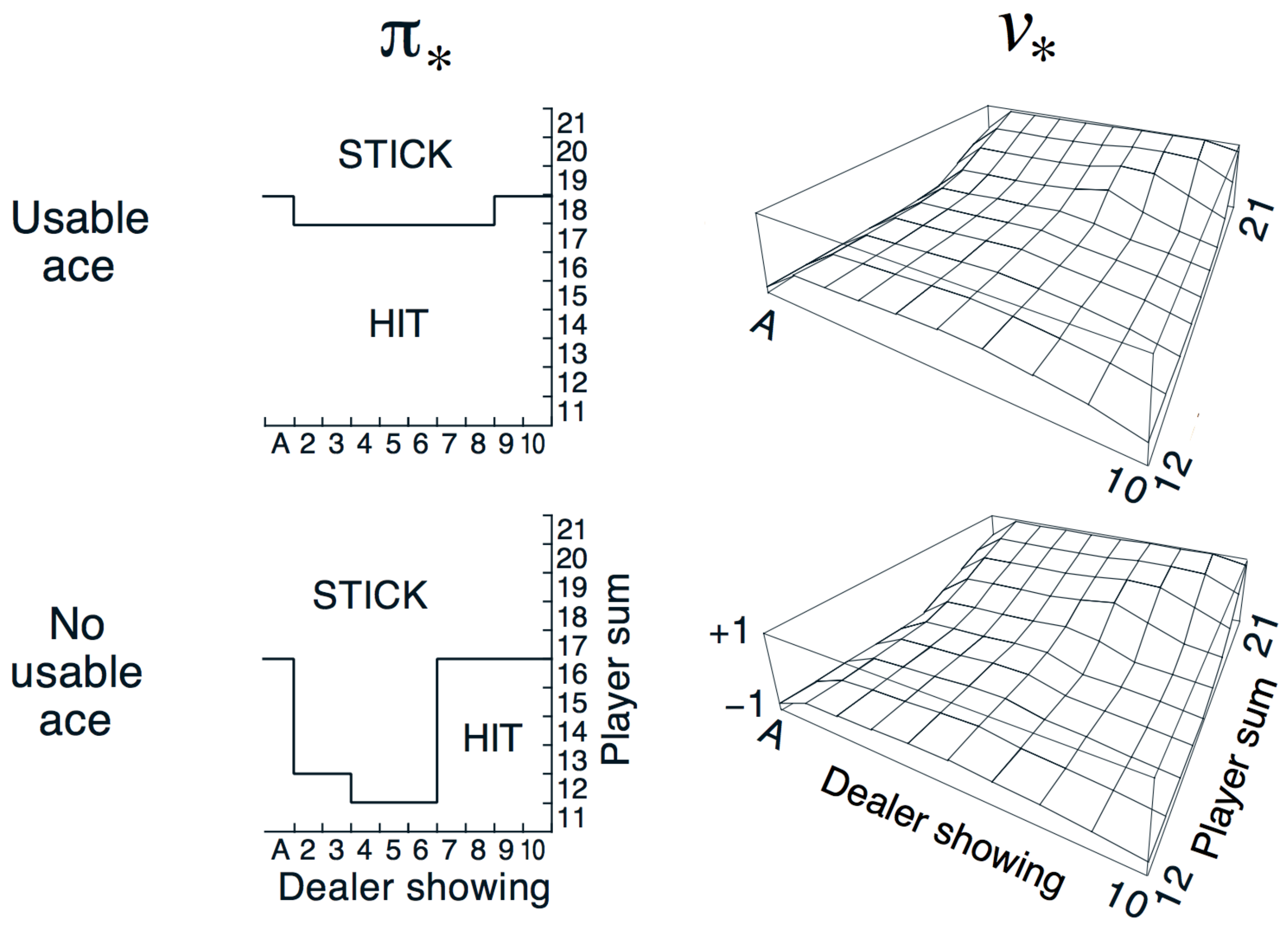
SARSA: State Action Reward State Action

- Works almost like Q-Learning with a difference in update rule:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

- Here, A' is the action that the agent ends up taking (which is not necessarily the $argmax_a\, Q(S',a)$ as the policy includes randomness.)

Back to Blackjack's example



Conclusion

- Model free policy evaluation (prediction) is possible via deployment of Monte-Carlo and TD methods.
- TD is advantageous and does not require full episode to be stored.
- Model free policy optimization (control) can be achieved via Q-Learning and SARSA methods.
- Epsilon Greedy policies make exploration possible and ensure theoretical convergence towards the true value function.