AA203 Optimal and Learning-based Control

Model-free Reinforcement Learning: Value-based Methods
Roadmap

Control
- Feedback control
- Adaptive control

Adaptive optimal control
- Model-free RL
- Model-based RL

Optimal and learning control

Open-loop
- Indirect methods
- Direct methods

MPC

Closed-loop
- DP
- HJB / HJI
Markov Decision Process

State: \( x \in X \)
Action: \( u \in U \)
Transition function / Dynamics: \( T(x_t \mid x_{t-1}, u_{t-1}) = p(x_t \mid x_{t-1}, u_{t-1}) \)
Reward function: \( r_t = R(x_t, u_t) : X \times U \to \mathbb{R} \)
Discount factor: \( \gamma \in (0,1) \)

Goal: choose a policy that maximizes cumulative (discounted) reward

\[
\pi^* = \arg\max_{\pi} \mathbb{E}_p \left[ \sum_{t \geq 0} \gamma^t R(x_t, \pi(x_t)) \right]
\]
Review

In previous lectures, we made the distinction between prediction (given a policy \( \pi \), estimate \( V_\pi, Q_\pi \)) and control (learn the optimal policy \( \pi^* \))

Motivated by Dynamic Programming, we discussed exact methods for solving MDPs:
• Policy Iteration
• Value Iteration

<table>
<thead>
<tr>
<th>Problem</th>
<th>Bellman Equation</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction</td>
<td>Bellman Expectation Equation</td>
<td>Iterative Policy Evaluation</td>
</tr>
<tr>
<td>Control</td>
<td>Bellman Expectation Equation + Greedy Policy Improvement</td>
<td>Policy Iteration</td>
</tr>
<tr>
<td>Control</td>
<td>Bellman Optimality Equation</td>
<td>Value Iteration</td>
</tr>
</tbody>
</table>

**Limitation:** Update equations (i.e., Bellman equations) require access to dynamics model \( T(x_{t+1} \mid x_t, u_t) \)

We saw how to use sampling and bootstrapping to approximate the expectations in the update equations:
• Monte Carlo (MC) Learning
• Temporal-Difference (TD) Learning
• **Sampling:** define the update through samples to approximate expectations
  - MC samples
  - TD samples
  - DP does not sample

• **Bootstrapping:** define the update through an estimate
  - MC does not bootstrap
  - TD bootstraps
  - DP bootstraps

**Dynamic Programming backup**

\[ \hat{V}(x_t) \leftarrow \mathbb{E} \left[ R_t + \gamma \hat{V}(x_{t+1}) \mid x_t \right] \]

**Monte Carlo backup**

\[ \hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha \left( G_t - \hat{V}(x_t) \right) \]

**Temporal-Difference backup**

\[ \hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha \left( R_t + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t) \right) \]
• For control:
A taxonomy of RL

**Model-free**
- Policy optimization
  - do not use dynamics $T(x_{t+1} | x_t, u_t)$
  - directly maximize the RL objective
    \[ \mathbb{E}_{r \sim p_s(r)} \left[ \sum_{t=0}^{H} \gamma^t r(x_t, u_t) \right] \]

**Value-based**
- Value-based
  - policy implicitly defined via $V(x)$ or $Q(x, u)$
  - set $\pi(s_t) = \arg \max_a Q(s_t, a_t)$

**Model-based**
- Learn the model
  - use dynamics $T(x_{t+1} | x_t, u_t)$
  - estimate $f_\theta \approx T(x_{t+1} | x_t, u_t)$

- Given the model
  - $T(x_{t+1} | x_t, u_t)$ is known
Outline

Tabular methods
  • On-policy & Off-policy
    • SARSA
    • Q-learning

Value function approximation

Deep (Value-based) RL Methods & Applications
Temporal-Difference Control

- TD learning has several advantages over MC
  - Lower variance
  - Online
  - Incomplete sequences

- Natural idea: use TD instead of MC in our GPI scheme
  - Apply TD to estimate $Q(x, u)$
  - Use $\epsilon$-greedy policy improvement
  - Update every time-step
Updating action-value functions with Sarsa

• Uses every element of the quintuple of events, \((x_t, u_t, r_t, x_{t+1}, u_{t+1})\), that make up a transition from one state–action pair to the next through the following update rule

\[
Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left( r_t + \gamma Q(x_{t+1}, u_{t+1}) - Q(x_t, u_t) \right)
\]

• In RL literature, \((x_t, u_t, r_t, x_{t+1}, u_{t+1})\) is often expressed as \((s_t, a_t, r_t, s_{t+1}, a_{t+1})\) : hence the name

\[
\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha \left( R_t + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t) \right)
\]
Sarsa algorithm for on-policy control

Initialize $Q(x, u), \forall x \in X, \forall u \in U$, arbitrarily, and $Q(\text{terminal-state, } \cdot) = 0$

Repeat (for each episode):
  Initialize $x_t$
  Choose $u_t$ from $x_t$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)

Repeat (for each step of episode):
  Take action $u_t$, observe $r_t, x_{t+1}$
  Choose $u_{t+1}$ from $u_{t+1}$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
  $Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left( r_t + \gamma Q(x_{t+1}, u_{t+1}) - Q(x_t, u_t) \right)$

$x_t \leftarrow x_{t+1}, u_t \leftarrow u_{t+1}$
until $x_t$ is terminal

**On-policy**: evaluate or improve the policy that is used to make decisions

**Off-policy**: evaluate or improve a policy different from that used to generate the data
Windy Gridworld example

- Reward -1 until goal is reached
- $\epsilon = 0.1$
- $\alpha = 0.5$
- $\gamma = 1$
Windy Gridworld example

Question:
Would MC methods easily apply to this problem? And why?
Off-policy learning

- Evaluate target policy $\pi(u \mid x)$ to compute $V_\pi(x)$ or $Q_\pi(x, u)$ while following behavior policy $\mu(u \mid x)$, i.e.,

$$\{x_1, u_1, r_1, \ldots, x_T\} \sim \mu, \text{ “the data we observe is obtained under policy } \mu\text{”}$$

Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
- Learn about optimal policy while following exploratory policy
Off-policy learning of action-values

• We consider off-policy learning of action-values $Q(x, u)$

• As in Sarsa, we use the behavior policy $\mu$ to obtain $(x_t, u_t, r_t, x_{t+1}, u'_{t+1})$, but we consider an alternative successor action $u'_{t+1} \sim \pi(u'_{t+1} | x_{t+1})$

• And update $Q(x, u)$ towards value of alternative action

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left( r_t + \gamma Q(x_{t+1}, u'_{t+1}) - Q(x_t, u_t) \right)$$
Q-learning

Specifically, in Q-learning

- The target policy $\pi$ is chosen as the greedy policy w.r.t. $Q(x, u)$

$$\pi(x_{t+1}) = \arg\max_{u'_{t+1}} Q(x_{t+1}, u'_{t+1})$$

- The behavior policy $\mu$ is chosen as the $\epsilon$-greedy policy w.r.t. $Q(x, u)$

Which leads to the following Q-learning target and update:

$$r_{t+1} + \gamma Q(x_{t+1}, u'_{t+1})$$

$$= r_{t+1} + \gamma Q\left(x_{t+1}, \arg\max_{u'_{t+1}} Q(x_{t+1}, u'_{t+1})\right)$$

$$= r_{t+1} + \gamma \max_{u'_{t+1}} Q(x_{t+1}, u'_{t+1})$$

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left( r_t + \gamma \max_{u'_{t+1}} Q(x_{t+1}, u'_{t+1}) - Q(x_t, u_t) \right)$$
Q-learning algorithm for off-policy control

Initialize $Q(x, u), \forall x \in X, \forall u \in U$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):
  Initialize $x_t$

Repeat (for each step of episode):
  Choose $u_t$ from $x_t$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
  Take action $u_t$, observe $r_t, x_{t+1}$

$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left( r_t + \gamma \max_{u'_{t+1}} Q(x_{t+1}, u'_{t+1}) - Q(x_t, u_t) \right)$

until $x_t$ is terminal

Theorem

$Q$-learning control converges to the optimal action-value function, $Q(s, a) \to q_*(s, a)$
**Differences between Sarsa and Q-learning**

- Reward -1 until goal is reached, -100 if on “The Cliff”
- $\epsilon = 0.1$
- $\alpha = 0.5$
- $\gamma = 1$

- Sarsa converges to the *optimal $\epsilon$-greedy policy*
- Q-learning converges to the *optimal policy $\pi^*$ / value function $Q^*$*
Outline

Tabular methods

• On-policy & Off-policy
  • SARSA
  • Q-learning

Value function approximation

Deep (Value-based) RL Methods & Applications
Solving large-scale problems with RL

- Reinforcement learning can be used to solve large problems, e.g.,

  Backgammon: $10^{20}$ states
  Go: $10^{170}$ states

All those problems where we have a continuous state space

How can we scale the methods for model-free RL we developed over the last lectures?
Value function approximation

• So far we used lookup tables to represent value functions:
  • One entry for every state $x$ in $V(x)$
  • One entry for every state-action pair $(x, u)$ in $Q(x, u)$

• In large MDPs, lookup table might be prohibitive. For two main reasons:
  • Memory: too many actions/states to store
  • Sparsity/Curse of dimensionality: learning the value of each state/action pair individually might take too long

Solution:
• Estimate the value function through function approximation, i.e., define a parametric function with parameters $\theta$

  $\hat{Q}_\theta(x, u) \approx Q(x, u)$
  $\hat{V}_\theta(x) \approx V(x)$

→ Represent the value function compactly (depends only on parameters $\theta$)
→ Generalize across states (avoid having to visit the entire state-action space by generalizing from seen to unseen states)
Different types of value function approximations

There are many possible function approximators
- Linear regression, Neural network, Random forest, Nearest neighbor, etc.
Approximating value fn. by (stochastic) gradient descent

Goal: find the parameter vector $\theta$ that minimizes the mean-squared error between the estimated value $\hat{V}_\theta(x)$ and the true value $V_\pi(x)$

$$J(\theta) = \mathbb{E}_\pi \left[ \left( V_\pi(x) - \hat{V}_\theta(x) \right)^2 \right]$$

Gradient descent converges to a local minimum

$$\Delta \theta = -\frac{1}{2} \alpha \nabla_\theta J(\theta)$$

$$= \alpha \mathbb{E}_\pi \left[ \left( V_\pi(x) - \hat{V}_\theta(x) \right) \nabla_\theta \hat{V}_\theta(x) \right]$$

Stochastic GD samples the gradient

$$\Delta \theta = \alpha \left( V_\pi(x) - \hat{V}_\theta(x) \right) \nabla_\theta \hat{V}_\theta(x)$$
Approximating value fn. by (stochastic) gradient descent

In the previous slide, we assumed to know the true value function $V_\pi \rightarrow$ in RL there is no supervisor, only reward

In practice, we use a target for $V_\pi$

- Monte-Carlo: the target is the return
  \[
  \Delta \theta = \alpha \left( G_t - \hat{V}_\theta(x_t) \right) \nabla_\theta \hat{V}_\theta(x_t)
  \]

- Temporal-Difference: the target is the TD target
  \[
  \Delta \theta = \alpha \left( r_t + \gamma \hat{V}_\theta(x_{t+1}) - \hat{V}_\theta(x_t) \right) \nabla_\theta \hat{V}_\theta(x_t)
  \]
Intuition

1) Compute return \( G_t = r_t + \gamma r_{t+1} + \ldots \), \( \forall t \)
2) Update estimate

\[
\hat{V}(x_t) = \left[ \hat{V}(x_t) + \alpha(G_t - \hat{V}(x_t)) \right]
\]

MC

1) Collect dataset \( \mathcal{D} = \{(x_t, G_t)\} \)
2) Update \( \theta \)

\[
\theta = \theta + \alpha \left( G_t - \hat{V}_\theta(x_t) \right) \nabla_\theta \hat{V}_\theta(x_t)
\]

TD

1) Compute target \( r_t + \gamma \hat{V}(x_{t+1}), \forall t \)
2) Update estimate

\[
\hat{V}(x_t) = \left[ \hat{V}(x_t) + \alpha(r_t + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t)) \right]
\]

1) Collect dataset \( \mathcal{D} = \{(x_t, r_t + \gamma \hat{V}_\theta(x_t))\} \)
2) Update estimate

\[
\theta = \theta + \alpha \left( r_t + \gamma \hat{V}_\theta(x_t) - \hat{V}_\theta(x_t) \right) \nabla_\theta \hat{V}_\theta(x_t)
\]
Control with function approximation

Policy evaluation  Approximate policy evaluation, $\hat{q}(\cdot, \cdot, w) \approx q_\pi$
Policy improvement  $\epsilon$-greedy policy improvement
Action-value function approximation

Exactly the same intuitions apply when we try to approximate the action value function:

- Minimize the mean-squared error between the estimated value $\hat{Q}_\theta(x, u)$ and the true value $Q_\pi(x, u)$

$$J(\theta) = \mathbb{E}_\pi \left[ Q_\pi(x, u) - \hat{Q}_\theta(x, u) \right]$$

- Use stochastic gradient descent to find a local minimum

$$\Delta \theta = \alpha \left( Q_\pi(x, u) - \hat{Q}_\theta(x, u) \right) \nabla_{\theta} \hat{Q}_\theta(x, u)$$

**Fitted Q-Iteration**: update $\theta$ via stochastic gradient descent on TD target

$$\Delta \theta = \alpha \left( r_t + \gamma \max_{u'_{t+1}} Q_\theta(x_{t+1}, u'_{t+1}) - \hat{Q}_\theta(x_t, u_t) \right) \nabla_{\theta} \hat{Q}_\theta(x_t, u_t)$$
The skeleton of fitted Q-learning

Generate samples

Fit a model / estimate return

Set target

$y_t \leftarrow r_t + \gamma \max_u Q_\theta(x_{t+1}, u)$

Update $\theta$ to minimize

$J(\theta) = \mathbb{E}_\pi [y_t - Q_\theta(x_t, u_t)]$

Run the policy and observe $(x_t, u_t, r_t, x_{t+1})$

Improve the policy

Set target

$\pi (x_t) = \arg \max_a Q_\theta (x_t, u_t)$
Deep Q-Networks (DQN)

One of the most popular Deep RL algorithms and arguably one of the first successes of RL with neural networks

(1) Use **deep neural nets** to represent $Q_\theta$ in Q-learning

(2) Uses **experience replay** and **fixed Q-targets**
Deep Q-Networks (DQN)

(2) Uses experience replay and fixed Q-targets

• These two ideas turned out to be very important to stabilize training. Specifically, these concepts attempt to solve two issues:

  i) Samples within a trajectory are highly correlated → makes supervised learning unstable
  ii) The target \( r_t + \gamma \max_{u_{t+1}} Q_\theta (x_{t+1}, u_{t+1}) \) is a moving target (i.e., as we update \( \theta \), the target of our regression also changes)

Intuitively:

• Take action \( u_t \) according to \( \epsilon \)-greedy policy
• Store transition \((x_t, u_t, r_t, x_{t+1})\) in replay memory \( \mathcal{D} \)
• Sample batch of transitions \( \{ (x_t, u_t, r_t, x_{t+1}) \}_{i=1}^{B} \) from \( \mathcal{D} \) (Experience replay decorrelates data)
• Compute Q-learning targets w.r.t. old, fixed parameters \( \phi \)
• Optimize MSE between Q-network prediction and Q-learning targets (Fixed targets stabilize the objective)

\[
J(\theta) = \mathbb{E}_{(x_t, u_t, r_t, x_{t+1}) \sim \mathcal{D}} \left[ r_t + \gamma \max_u Q_\phi(x_{t+1}, u) - Q_\theta(x_t, u_t) \right]
\]
Next time

• Model-free RL: policy optimization methods