Introduction to Reinforcement Learning

CMPT 882

Mar. 18

Outline for the week

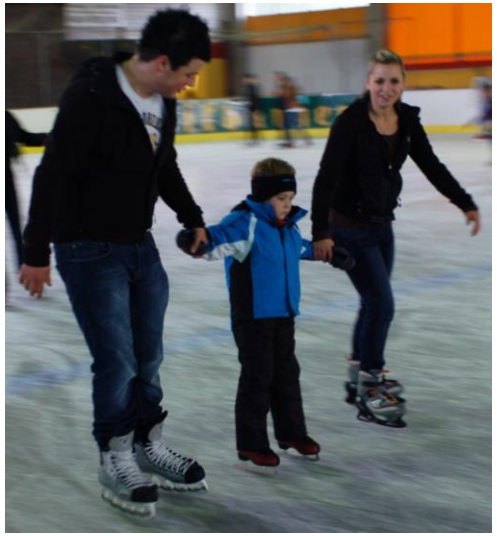
- Basic ideas in RL
  - Value functions and value iteration
  - Policy evaluation and policy improvement

- Model-free RL
  - Monte-Carlo and temporal differencing policy evaluation
  - ϵ-greedy policy improvement

- Function Approximation
  - Adaptation of supervised learning to reinforcement learning

- Policy Gradient

Reinforcement Learning



- Humans can learn without imitation
  - Given goal/task
  - Try an initial strategy
  - See how well the task is performed
  - Adjust strategy next time

- Reinforcement learning agent
  - Given goal/task in the form of reward function $r(s, a)$
  - Start with initial policy $\pi_\theta(a|s)$; execute policy
  - Obtain sum of rewards, $\sum_t r(s_t, a_t)$
  - Improve policy by updating θ, based on rewards

Reinforcement Learning Objective

- Given: an MDP with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $\mathcal{T}$, and reward function $r(s, a)$

- Objective: Maximize expected discounted sum of rewards ("return")

$$\underset{\pi_\theta}{\text{maximize}}\, \mathbb{E} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

  - $\gamma \in (0,1]$: discount factor – larger roughly means "far-sighted"
    - Prioritizes immediate rewards
    - $\gamma < 1$ avoids infinite rewards; $\gamma = 1$ is possible if all sequences are finite

- Constraints: now incorporated into the reward function
  - Only constraint (usually implicit): subject to transition matrix $\mathcal{T}$ (system dynamics)

RL vs. Other ML Paradigms

- No supervisor
  - But we will often draw inspiration from supervised learning

- Sequential data in time

- Reward feedback is obtained after a long time
  - Many actions combined together will receive reward
  - Actions are dependent on each other

- In robotics: lack of data

Reinforcement Learning Categories

- Model-based
  - Explicitly involves an MDP model
- Model-free
  - Does not explicitly involve an MDP model

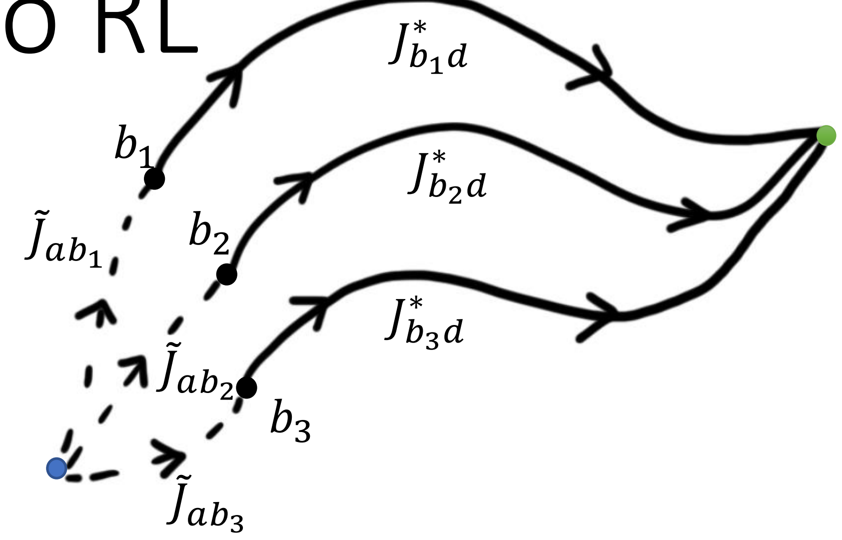
- Value based
  - Learns value function, and derives policy from value function
- Policy based
  - Learns policy without value function
- Actor critic
  - Incorporates both value function and policy

Value Functions

- "**State-value function**": $V_\pi(s)$ -- expected return starting from state s and following policy π
  - $V_\pi(s) = \mathbb{E}_{a_t \sim \pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|s_0 = s]$
  - Expectation is on the random sequence $\{s_0, a_0, s_1, a_1, \dots\}$

- "**Action-value function**", or "$\boldsymbol{Q}$ **function**": $Q_\pi(s, a)$ -- expected return starting from state s, taking action a, and then following policy π
  - $Q_\pi(s, a) = \mathbb{E}_{a_t \sim \pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|s_0 = s, a_0 = a]$

Principal of Optimality Applied to RL



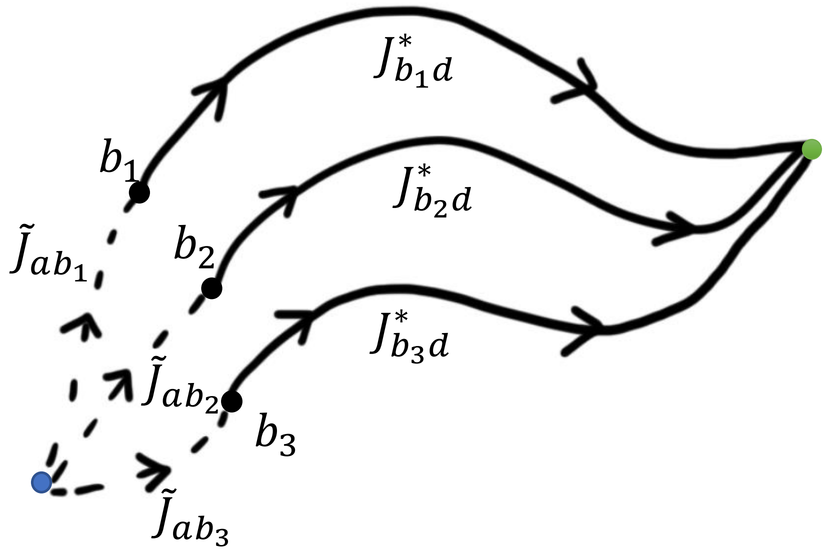
- Optimal discounted sum of rewards:
  - $V_{\pi^*}(s) = \max_{\pi} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|s_0 = s]$
- Dynamic programming:
  - $V_{\pi^*}(s) = \max_{a_t} \mathbb{E}[r(s_t, a_t) + \gamma V_{\pi^*}(s_{t+1}) | s_t = s]$
  - $Q_{\pi^*}(s, a) = \mathbb{E}[r(s_t, a_t) + \gamma V_{\pi^*}(s_{t+1}) | s_t = s, a_t = a]$

- Actually, recurrence is true even without maximization
  - $V_{\pi}(s) = \mathbb{E}[r(s_t, a_t) + \gamma V_{\pi}(s_{t+1}) | s_t = s]$
  - $Q_{\pi}(s, a) = \mathbb{E}[r(s_t, a_t) + \gamma V_{\pi}(s_{t+1}) | s_t = s, a_t = a]$

Basic Properties of Value Functions

- $V_{\pi^*}(s) = \max_{\pi} V_{\pi}(s)$
- $Q_{\pi^*}(s, a) = \max_{\pi} Q_{\pi}(s, a)$
- $V_{\pi^*}(s) = \max_{a} Q_{\pi^*}(s, a)$
- For now, value functions are stored in multi-dimensional arrays
- DP leads to deterministic policies – we will come back to stochastic policies

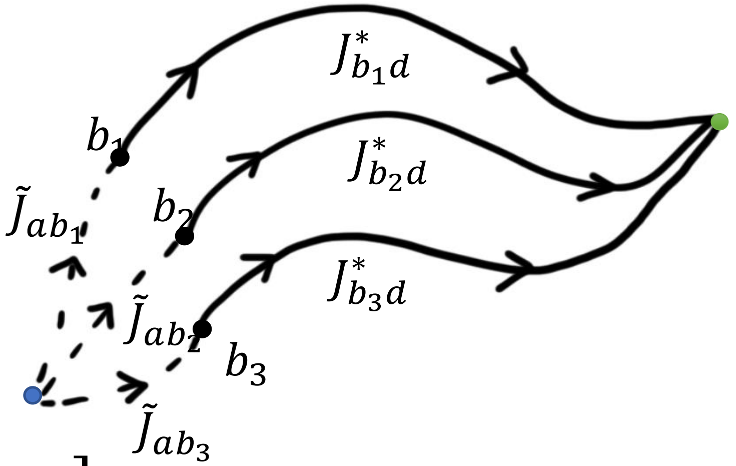
Optimizing the RL Objective via DP



- State-value function
  - $V_{\pi^*}(s) = \max_{a_t} \mathbb{E}[r(s_t, a_t) + \gamma V(s_{t+1}) | s_t = s]$
  - $V_{\pi^*}(s) = \max_{a} \{r(s, a) + \gamma \mathbb{E}[V(s_{t+1}) | s_t = s]\}$
  - $V_{\pi^*}(s) = \max_{a} \{r(s, a) + \gamma \sum_{s'} [p(s' | s, a) V_{\pi^*}(s')]\}$
  - "**Bellman backup**": $V(s) \leftarrow \max_{a} \{r(s, a) + \gamma \sum_{s'} [p(s' | s, a) V(s')]\}$
    - This is done for all s
    - Iterate until convergence

- Optimal policy: $a = \arg\max_{a'} \{r(s, a') + \gamma \sum_{s'} [p(s' | s, a') V(s')]\}$
  - Deterministic

Optimizing the RL Objective via DP



- Action-value function
  - $Q_{\pi^*}(s,a) = \mathbb{E}\left[r(s_t,a_t) + \gamma \max_{a_{t+1}} Q_{\pi^*}(s_{t+1},a_{t+1})\,|s_t = s, a_t = a\right]$
  - $Q_{\pi^*}(s,a) = r(s,a) + \gamma\mathbb{E}\left[\max_{a_{t+1}} Q_{\pi^*}(s_{t+1},a_{t+1})\,|s_t = s, a_t = a\right]$
  - $Q_{\pi^*}(s,a) = r(s,a) + \gamma \sum_{s'}[p(s'|s,a)V_{\pi^*}(s')]$
  - “Bellman backup”:
    - $V(s) \leftarrow \max_{a} Q(s,a)$
    - $Q(s,a) \leftarrow r(s,a) + \gamma \sum_{s'}[p(s'|s,a)V(s')]$
    - This is done for all s and all a
    - Iterate until convergence

- Optimal policy: $a = \arg\max_{a'} Q(s,a')$
  - Deterministic

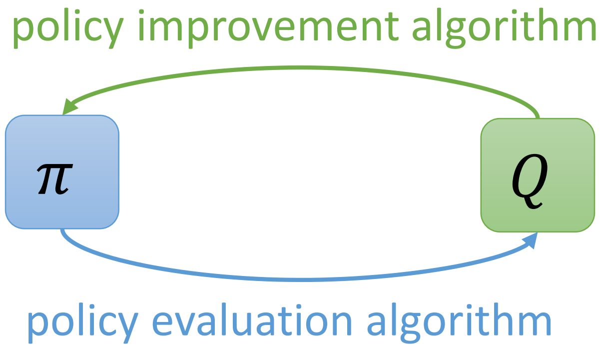
Approximate Dynamic Programming

- Use a function approximator (eg. neural network) $\hat{V}(s; w)$, where w are weights, to approximate V
  - $V(s)$ is no longer stored at every state
  - Weights w are updated using Bellman backups
- Basic algorithm: (We will learn about other variants too)
  - Sample some states, $\{s_i\}$
  - For each s_i, generate $\tilde{V}(s_i) = \max_a \left\{ r(s, a) + \gamma \sum_{s'} \left[p(s' | s_t, a) \hat{V}(s'; w) \right] \right\}$
  - Using $\{s_i, \tilde{V}(s_i)\}$, update weights w via regression (supervised learning)

Generalized Policy Evaluation and Policy Improvement

- Start with initial policy π and value function V or Q
- Use policy π to update V: $a = \pi(s)$

DP
  - $V(s) \leftarrow r(s,a) + \gamma \sum_{s'} [p(s'|s,a)V(s')]$
  - $Q(s,a) \leftarrow r(s,a) + \gamma \sum_{s'} [p(s'|s,a)V(s')]$
  - In general, any policy evaluation algorithm

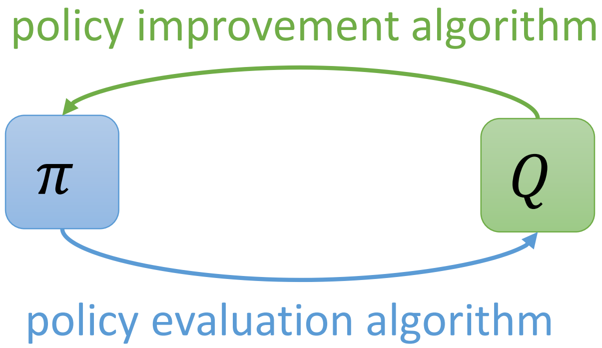


- Use V or Q to update policy π:

DP
  - Given $V(\mathrm{s}), \pi(s) = \arg\max_{a} \{r(s,a) + \gamma \sum_{s'} [p(s'|s,a)V(s')]\}$
  - Given $Q(s,a), \pi(s) = \arg\max_{a} Q(s,a)$
  - In general, any policy improvement algorithm

Convergence

- At convergence, the following are simultaneously satisfied:
  - $V(s) = r(s,a) + \gamma \sum_{s'}[p(s'|s_t,a)V(s')]$
  - $\pi(s) = \arg\max_{a'}\{r(s,a') + \gamma \sum_{s}[p(s|s_t,a')V(s)]\}$



- This is the principle of optimality
- Therefore, the value function and policy are optimal

Terminology

- "**Value iteration**": The process of iteratively updating value function
  - With DP, we only need to keep track of value function V or Q, and the policy π is implicit – determined from value function
- "**Policy iteration**": The process of iteratively updating policy
  - This is done implicitly with Bellman backups
- "**Greedy policy**": the policy obtained from choosing the best action based on the current value function
  - If the value function is optimal, the greedy policy is optimal

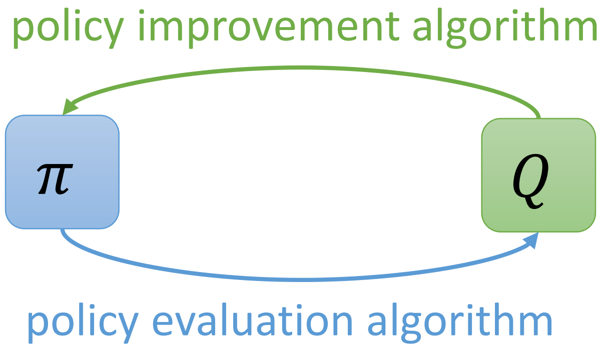
Towards Model-Free Learning

- Policy evaluation
  - Monte-Carlo (MC) Sampling
  - Temporal-difference (TD)

- Policy improvement
  - ϵ-greedy policies

Monte-Carlo Policy Evaluation

- Start with initial policy π and value function V or Q
- Use policy π to update V: $a = \pi(s)$
  - Apply π to obtain trajectory $\{s_0, a_0, s_1, a_1, \ldots\}$
  - Compute return: $R \coloneqq \sum \gamma^t r(s_t, a_t)$
  - Repeat for many episodes to obtain empirical mean
    - "**Episode**": a single "try" that produces a single trajectory



- Use V or Q to update policy π

Monte-Carlo Policy Evaluation

- To obtain empirical mean, we record $N(s)$, # of times s is visited for every state
  - Start at $N(s) = 0$ for all s
  - Note that this means storing N (and S below) at every state

- First-visit MC Policy Evaluation:
  - At the first time t that s is visited in an episode,
    - Increment $N(s) \leftarrow N(s) + 1$
    - Record return $S(s) \leftarrow S(s) + \sum \gamma^t r(s_t, a_t)$
    - Repeat for many episodes
  - Estimate value: $V(s) = \frac{S(s)}{N(s)}$

Monte-Carlo Policy Evaluation

- To obtain empirical mean, we record $N(s)$, # of times s is visited for every state
  - Start at $N(s) = 0$ for all s
  - Note that this means storing N (and S below) at every state

- Every-visit MC Policy Evaluation:
  - Every time t that s is visited in an episode,
    - Increment $N(s) \leftarrow N(s) + 1$
    - Record return $S(s) \leftarrow S(s) + \sum \gamma^t r(s_t, a_t)$
    - Repeat for many episodes
  - Estimate value: $V(s) \approx \frac{S(s)}{N(s)}$

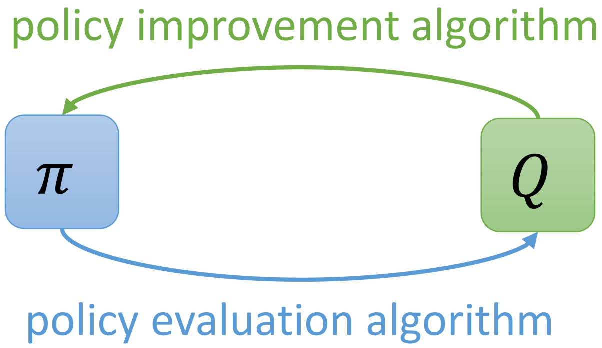
Incremental Updates

- Instead of estimating $V_\pi(s)$ after many episodes, we can update it incrementally after every episode after receiving return R
  - $N(s) \leftarrow N(s) + 1$
  - $V(s) \leftarrow V(s) + \frac{1}{N(s)}\big(R - V(s)\big)$

- More generally, we can weight the second term differently
  - $V(s) \leftarrow V(s) + \alpha\big(R - V(s)\big)$

Monte-Carlo Policy Evaluation

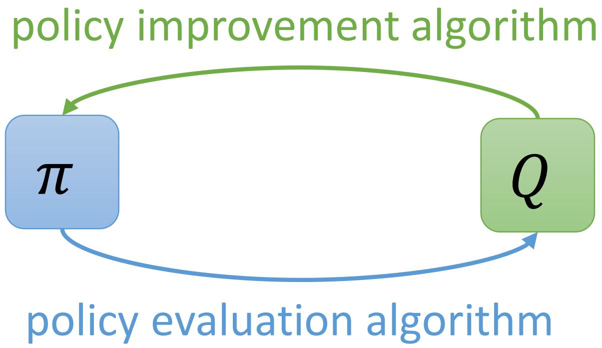
- Start with initial policy π and value function V or Q
- Use policy π to update V: $a = \pi(s)$
  - MC policy evaluation provides estimate of V_π
  - Many episodes are needed to obtain accurate estimate
  - Model-free with MC!



- Use V or Q to update policy π
  - Greedy policy?

Monte-Carlo Policy Evaluation

- Start with initial policy π and value function V or Q
- Use policy π to update V: $a = \pi(s)$
  - MC policy evaluation provides estimate of V_π
  - Many episodes are needed to obtain accurate estimate
  - Model-free with MC!



- Use V or Q to update policy π
  - ~~Greedy policy?~~
    - Greedy policy lacks exploration, so V_π is not estimated at many states
  - ϵ-greedy policy

ϵ-Greedy Policy

- Also known as ϵ-greedy exploration

- Choose random action with probability ϵ
  - Typically uniformly random
  - If a takes on discrete values, then all actions will be chosen eventually

- Choose action from greedy policy with probability $1 - \epsilon$
  - $a = \arg\max_{a'}\{r(s, a') + \gamma \sum_s [p(s|s_t, a')V(s)]\}$
  - Still requires model, $p(s|s_t, a)$…
  - Solution: Q function

Monte-Carlo Policy Evaluation

- To obtain empirical mean, we record $N(s, a)$, # of times s is visited for every state
  - Start at $N(s, a) = 0$ for all s and a
  - Note that this means N (and S below) must be stored for every s and a

- First-visit MC Policy Evaluation:
  - At the first time t that s is visited in an episode,
    - Increment $N(s, a) \leftarrow N(s, a) + 1$
    - Record return $S(s, a) \leftarrow S(s, a) + \sum \gamma^t r(s_t, a_t)$
    - Repeat for many episodes
  - Estimate action-value function: $Q(s, a) = \frac{S(s,a)}{N(s,a)}$

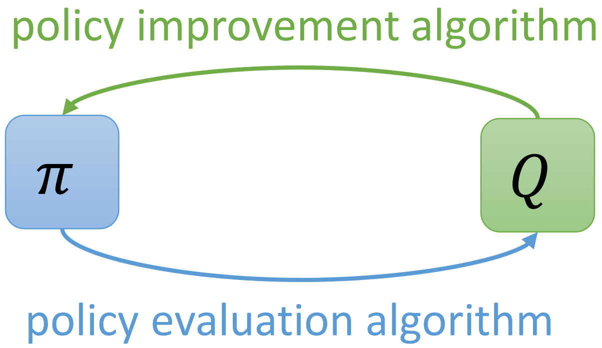
Incremental Updates

- Instead of estimating $V(s)$ after many episodes, we can update it incrementally after every episode after receiving return R
  - $N(s,a) \leftarrow N(s,a) + 1$
  - $Q(s,a) \leftarrow Q(s,a) + \frac{1}{N(s,a)}\big(R - Q(s,a)\big)$

- More generally, we can weight the second term differently
  - $Q(s,a) \leftarrow Q(s,a) + \alpha\big(R - Q(s,a)\big)$

Monte-Carlo Policy Evaluation

- Start with initial policy π and value function V or Q
- Use policy π to update Q: $a = \pi(s)$
  - MC policy evaluation provides estimate of Q_π
  - Many episodes are needed to obtain accurate estimate
  - Model-free with MC!



- Use ~~V or~~ Q to update policy π
  - ~~Greedy policy?~~
    - Greedy policy lacks exploration, so V is not estimated at many states
  - ϵ-greedy policy

ϵ-Greedy Policy

- Also known as ϵ-greedy exploration

- Choose random action with probability ϵ
  - Typically uniformly random
  - If a takes on discrete values, then all actions will be chosen eventually

- Choose action from greedy policy with probability $1 - \epsilon$
  - $a = \arg\max_{a'}\{Q(s, a')\}$
  - Model-free!