Lecture 5: Model-Free Control

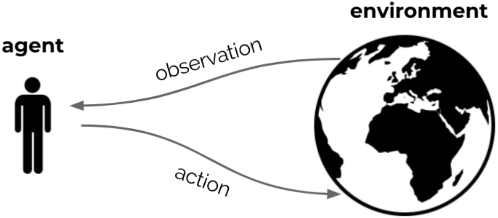
Hado van Hasselt

January 22, 2020, UCL

Background

Sutton & Barto 2018, Chapter 6

Recap

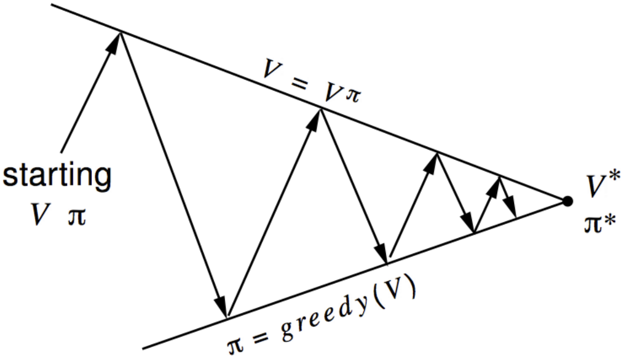


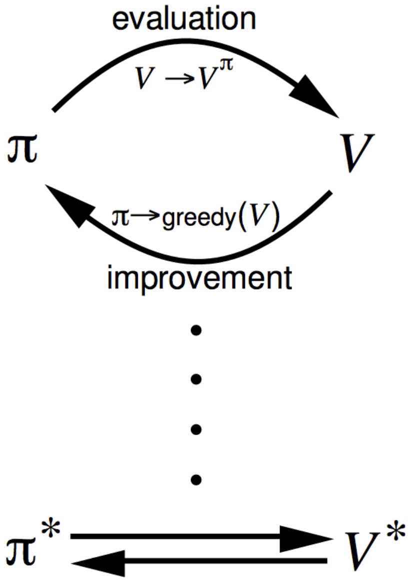
- Reinforcement learning is the science of learning to make decisions
- Agents can learn a **policy**, **value function** and/or a **model**
- The general problem involves taking into account **time** and **consequences**
- Decisions affect the **reward**, the **agent state**, and **environment state**

Model-Free Control

- Previous lecture: **Model-free prediction**
  Estimate the value function of an unknown MDP
- This lecture: **Model-free control**
  Optimise the value function of an unknown MDP

Generalized Policy Iteration (Refresher)





- **Policy evaluation**
  Estimate $v_\pi(s)$ for all s
- **Policy improvement**
  Generate π' such that $v_{\pi'}(s) \geq v_\pi(s)$ for all s

Recap: Model-Free Policy Evaluation

$$v_{n+1}(S_t) = v_n(S_t) + \alpha \left(G_t - v_n(S_t) \right)$$

- Variants:

$$\begin{aligned} G_t^{\text{MC}} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma G_{t+1}^{\text{MC}} \qquad \textbf{(MC)} \end{aligned}$$

Recap: Model-Free Policy Evaluation

$$v_{n+1}(S_t) = v_n(S_t) + \alpha\left(G_t - v_n(S_t)\right)$$

- Variants:

$$\begin{aligned} G_t^{\mathrm{MC}} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma G_{t+1}^{\mathrm{MC}} && (\textbf{MC}) \\ G_t^{(1)} &= R_{t+1} + \gamma v_t(S_{t+1}) && (\textbf{TD(0)}\text{, or } \textbf{one-step TD}) \end{aligned}$$

Recap: Model-Free Policy Evaluation

$$v_{n+1}(S_t) = v_n(S_t) + \alpha\left(G_t - v_n(S_t)\right)$$

- Variants:

$$\begin{aligned}
G_t^{\text{MC}} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\
&= R_{t+1} + \gamma G_{t+1}^{\text{MC}} && (\textbf{MC}) \\
G_t^{(1)} &= R_{t+1} + \gamma v_t(S_{t+1}) && (\textbf{TD(0)}\text{, or } \textbf{one-step TD}) \\
G_t^{(n)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n v_t(S_{t+n}) \\
&= R_{t+1} + \gamma G_{t+1}^{(n-1)} && (\boldsymbol{n}\textbf{-step TD} \text{ or } \textbf{multi-step TD})
\end{aligned}$$

Recap: Model-Free Policy Evaluation

$$v_{n+1}(S_t) = v_n(S_t) + \alpha\left(G_t - v_n(S_t)\right)$$

- Variants:

$$\begin{aligned}
G_t^{\text{MC}} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \\
&= R_{t+1} + \gamma G_{t+1}^{\text{MC}} && (\textbf{MC}) \\
G_t^{(1)} &= R_{t+1} + \gamma v_t(S_{t+1}) && (\textbf{TD(0)}, \text{ or } \textbf{one-step TD}) \\
G_t^{(n)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v_t(S_{t+n}) \\
&= R_{t+1} + \gamma G_{t+1}^{(n-1)} && (\boldsymbol{n}\textbf{-step TD} \text{ or } \textbf{multi-step TD}) \\
G_t^{\lambda} &= R_{t+1} + \gamma[(1-\lambda)v_t(S_{t+1}) + \lambda G_{t+1}^{\lambda}] && (\textbf{TD}(\boldsymbol{\lambda}))
\end{aligned}$$

Recap: Model-Free Policy Evaluation

$$v_{n+1}(S_t) = v_n(S_t) + \alpha\left(G_t - v_n(S_t)\right)$$

- Variants:

$$\begin{aligned}
G_t^{\text{MC}} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \\
&= R_{t+1} + \gamma G_{t+1}^{\text{MC}} && (\textbf{MC}) \\
G_t^{(1)} &= R_{t+1} + \gamma v_t(S_{t+1}) && (\textbf{TD(0)}\text{, or } \textbf{one-step TD}) \\
G_t^{(n)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v_t(S_{t+n}) \\
&= R_{t+1} + \gamma G_{t+1}^{(n-1)} && (\textbf{\textit{n}-step TD} \text{ or } \textbf{multi-step TD}) \\
G_t^{\lambda} &= R_{t+1} + \gamma[(1-\lambda)v_t(S_{t+1}) + \lambda G_{t+1}^{\lambda}] && (\textbf{TD}(\lambda))
\end{aligned}$$

- In all cases, for given π goal is estimating v_π, data is generated to π

Model-Free Policy Iteration Using Action-Value Function

- Greedy policy improvement over $v(s)$ requires model of MDP

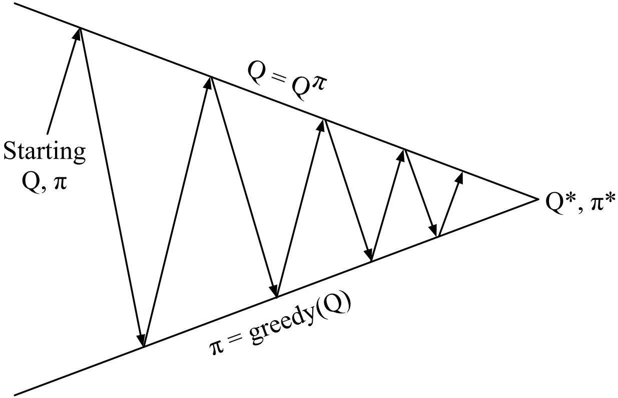
$$\pi'(s) = \operatorname*{argmax}_a \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s, A_t = a\right]$$

- Greedy policy improvement over $q(s, a)$ is **model-free**

$$\pi'(s) = \operatorname*{argmax}_a q(s, a)$$

- This makes action values convenient

Generalised Policy Iteration with Action-Value Function

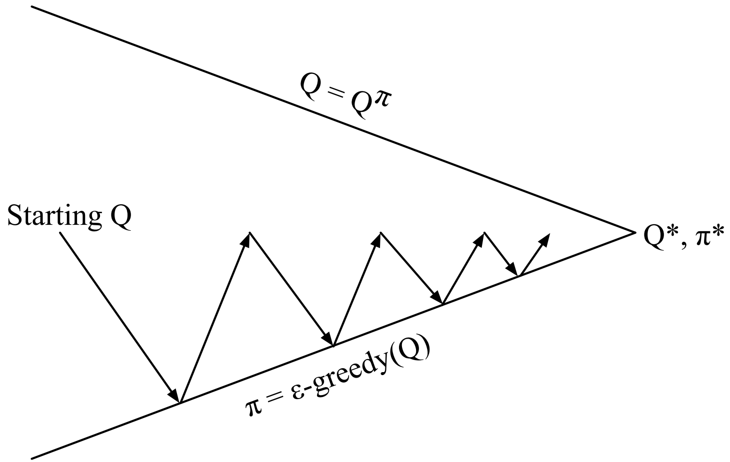


Policy evaluation Monte-Carlo policy evaluation, $\boldsymbol{q \approx q_\pi}$

Policy improvement Greedy policy improvement? No exploration!

(Can't do "for all s, a", because we sample by interacting)

Monte-Carlo Generalized Policy Iteration



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $\boldsymbol{q \approx q_\pi}$

Policy improvement ϵ-greedy policy improvement

Model-free control

Repeat:

- Sample episode $1, \ldots, k, \ldots$, using π: $\{S_1, A_1, R_2, ..., S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$
$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - q(S_t, A_t))$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(q)$$

(Generalises the ϵ-greedy bandit algorithm)

GLIE

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

$$\forall s, a \ \lim_{t \to \infty} N_t(s, a) = \infty$$

- The policy converges to a greedy policy,

$$\lim_{t \to \infty} \pi_t(a|s) = \mathcal{I}(a = \operatorname*{argmax}_{a'} q_t(s, a'))$$

- For example, ϵ-greedy with $\epsilon_k = \frac{1}{k}$

GLIE

Greedy in the Limit with Infinite Exploration (GLIE)

Definition

- All state-action pairs are explored infinitely many times,

$$\forall s, a \ \lim_{t \to \infty} N_t(s, a) = \infty$$

- The policy converges to a greedy policy,

$$\lim_{t \to \infty} \pi_t(a|s) = \mathcal{I}(a = \operatorname*{argmax}_{a'} q_t(s, a'))$$

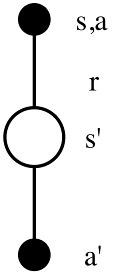
Theorem

GLIE Model-free control converges to the optimal action-value function, $q(s, a) \to q_*(s, a)$

MC vs. TD Control

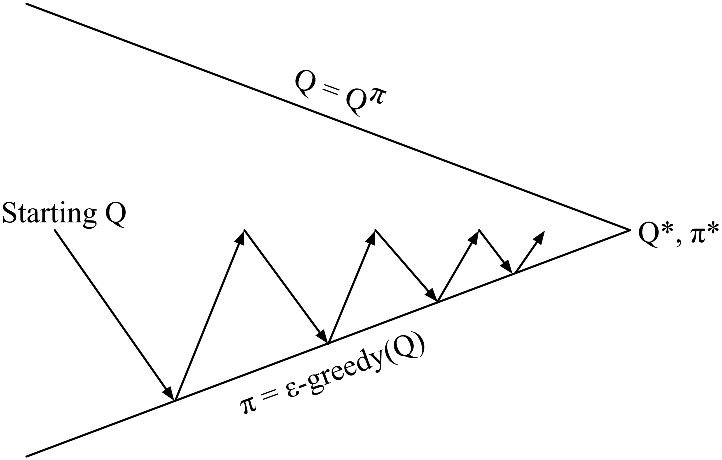
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Can learn from incomplete sequences
- Natural idea: use TD instead of MC for control
  - Apply TD to $q(s, a)$
  - Use, e.g., ϵ-greedy policy improvement
  - Update every time-step

Updating Action-Value Functions with Sarsa



$$q(s,a) \leftarrow q(s,a) + \alpha \left(r + \gamma q(s',a') - q(s,a)\right)$$

Sarsa



Every **time-step**:

Policy evaluation **Sarsa**, $q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Tabular Sarsa

Initialize $Q(s, a)$ arbitrarily
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)
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until s is terminal

Updating Action-Value Functions with Sarsa

$$q(s,a) \leftarrow q(s,a) + \alpha \left(r + \gamma q(s',a') - q(s,a) \right)$$

Theorem

Tabular Sarsa converges to the optimal action-value function, $q(s,a) \rightarrow q_*(s,a)$, if the policy is GLIE

Dynamic programming

- We discussed several dynamic programming algorithms

$$
\begin{aligned}
v_{k+1}(s) &= \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)\right] && \text{(policy evaluation)} \\
v_{k+1}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a\right] && \text{(value iteration)} \\
q_{k+1}(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma q_k(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] && \text{(policy evaluation)} \\
q_{k+1}(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') \mid S_t = s, A_t = a\right] && \text{(value iteration)}
\end{aligned}
$$

TD learning

- Analogous model-free TD algorithms

$$v_{t+1}(S_t) = v_t(S_t) + \alpha_t \left(R_{t+1} + \gamma v_t(S_{t+1}) - v_t(S_t)\right) \qquad \textbf{(TD)}$$

$$q_{t+1}(s, a) = q_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma q_t(S_{t+1}, A_{t+1}) - q_t(S_t, A_t)\right) \qquad \textbf{(Sarsa)}$$

$$q_{t+1}(s, a) = q_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t)\right)$$

(Q-learning)

- Note, no trivial analogous version of value iteration

$$v_{k+1}(s) = \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a\right]$$

Can you explain why?

On and Off-Policy Learning

- **On-policy** learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- **Off-policy** learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While using behaviour policy $\mu(a|s)$ to generate actions
- Why is this important?
  - Learn from observing humans or other agents (e.g., from logged data)
  - Re-use experience from old policies (e.g., from your own past experience)
  - Learn about **greedy** policy while following **exploratory** policy
  - Learn about **multiple** policies while following **one** policy
- **Q-learning** estimates the value of the **greedy** policy

$$q_{t+1}(s, a) = q_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t) \right)$$

But just being greedy would not explore

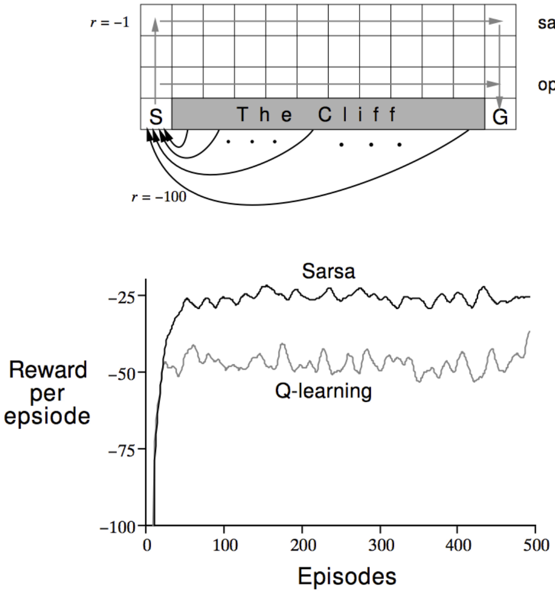
Q-Learning Control Algorithm

Theorem

Q-learning control converges to the optimal action-value function, $q \to q^*$, as long as we take each action in each state infinitely often.

Note: no need for greedy behaviour!
Works for **any** policy that eventually selects all actions sufficiently often
(Does require appropriately decaying step sizes $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$,
E.g., $\alpha = 1/t^\omega$, with $\omega \in (0.5, 1)$)

Cliff Walking Example



Q-learning overestimation

- Classical Q-learning has potential issues
- Recall

$$\max_a q_t(S_{t+1}, a) \;=\; q_t(S_{t+1}, \operatorname*{argmax}_a q_t(S_{t+1}, a))$$

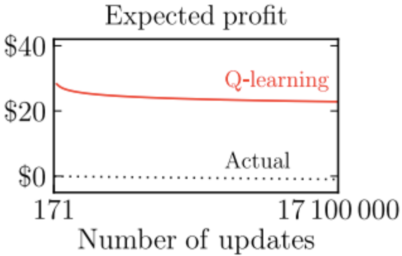
- Uses same values to **select** and to **evaluate**
- ... but values are approximate
  - **more** likely to select **overestimated values**
  - **less** likely to select **underestimated values**
- This causes upward bias

Q-learning overestimation: roulette example

- Roulette: gambling game
- Here, 171 actions: bet \$1 on one of 170 options, or 'stop'
- 'Stop' ends the episode, with \$0
- All other actions have high variance reward, with negative expected value
- Betting actions do not end the episode, instead can bet again

Q-learning overestimation: roulette example

- Roulette: gambling game
- Here, 171 actions: bet \$1 on one of 170 options, or ‘stop’
- ‘Stop’ ends the episode, with \$0
- All other actions have high variance reward, with negative expected value
- Betting actions do not end the episode, instead can bet again



Q-learning overestimation

- Q-learning overestimates because it uses the same values to **select** and to **evaluate**

$$\max_a q_t(S_{t+1}, a) = q_t(S_{t+1}, \operatorname*{argmax}_a q_t(S_{t+1}, a))$$

  - Roulette: quite likely that some actions have won, on average
  - Q-learning will updates if the state actually has high value
- Solution: decouple selection from evaluation

Double Q-learning

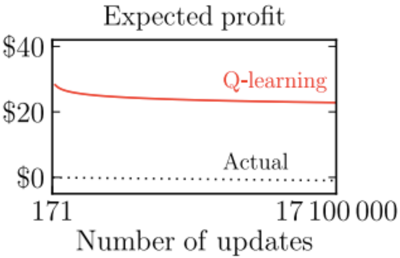
- **Double Q-learning**:
  - Store two action-value functions: q and q'

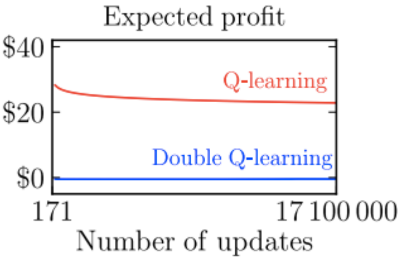
$$R_{t+1} + \gamma \boldsymbol{q'_t}(S_{t+1}, \underset{a}{\operatorname{argmax}}\, q_t(S_{t+1}, a)) \tag{1}$$

$$R_{t+1} + \gamma q_t(S_{t+1}, \underset{a}{\operatorname{argmax}}\, \boldsymbol{q'_t}(S_{t+1}, a)) \tag{2}$$

  - Each t, pick q or q' (e.g., randomly) and update using (1) for q or (2) for q'
  - Can use both to act (e.g., use policy based on $(q + q')/2$)
- Double Q-learning also converges to the optimal policy under the same conditions as Q-learning

Roulette example



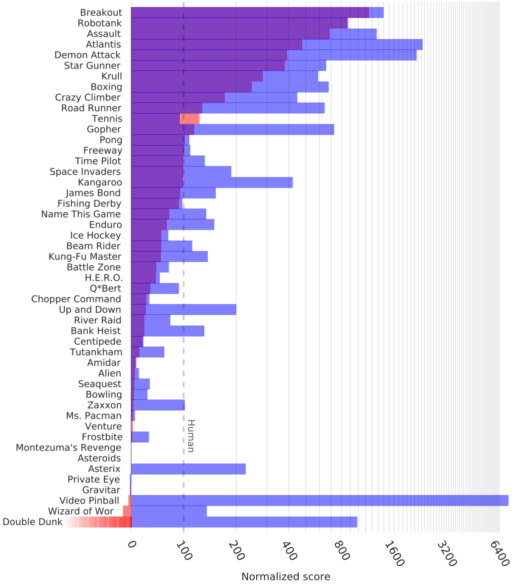


Double DQN on Atari

DQN

Double DQN

(This used a ‘target network’, to be explained later)



Double learning

- The idea of double Q-learning can be generalised to other updates
  - E.g., if you are (soft-) greedy (e.g., ϵ-greedy), then Sarsa can also overestimate
  - The same solution can be used
  - $\implies$ double Sarsa

Off-policy learning

- Recall: off-policy learning means learning about one policy π from experience generated according to a different policy μ
- Q-learning is one example, but there are other options
- Fortunately, there are general tools to help with this
- Caveat: you can't expect to learn about things you **never** do

Importance sampling corrections

- Goal: given some function f with random inputs X, and a distribution d', estimate the expectation of $f(X)$ under a different (target) distribution d
- Solution: weight the data by the ration d/d'

$$\begin{aligned}
\mathbb{E}_{x\sim d}[f(x)] &= \sum d(x)f(x) \\
&= \sum d'(x)\frac{d(x)}{d'(x)}f(x) \\
&= \mathbb{E}_{x\sim d'}\left[\frac{d(x)}{d'(x)}f(x)\right]
\end{aligned}$$

- Intuition:
  - scale up events that are rare under d', but common under d
  - scale down events that are common under d', but rare under d

Importance sampling corrections

- Example: estimate one-step reward
- Behaviour is $\mu(a|s)$

$$
\begin{aligned}
\mathbb{E}\left[R_{t+1} \mid S_t = s, A_t \sim \pi\right] &= \sum_a \pi(a|s) r(s,a) \\
&= \sum \mu(a|s) \frac{\pi(a|s)}{\mu(a|s)} r(s,a) \\
&= \mathbb{E}\left[\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} R_{t+1} \mid S_t = s, A_t \sim \mu\right]
\end{aligned}
$$

- Ergo, when following policy μ, can use $\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} R_{t+1}$ as unbiased sample

Importance Sampling for Off-Policy Monte-Carlo

- Goal: estimate v_π
- Data: trajectory $\tau_t = \{S_t, A_t, R_{t+1}, S_{t+1}, \ldots\}$ generated with μ
- Solution: use return $G(\tau_t) = G_t = R_{t+1} + \gamma R_{t+2} + \ldots$, and correct:

$$\frac{p(\tau_t|\pi)}{p(\tau_t|\mu)} G(\tau_t)$$

- But what are $p(\tau_t|\pi)$ and $p(\tau_t|\mu)$?
- Seems unwieldy...?

Importance Sampling for Off-Policy Monte-Carlo

- Goal: estimate v_π
- Data: trajectory $\tau_t = \{S_t, A_t, R_{t+1}, S_{t+1}, \ldots\}$ generated with μ
- Solution: use return $G(\tau_t) = G_t = R_{t+1} + \gamma R_{t+2} + \ldots$, and correct:

$$\frac{p(\tau_t|\pi)}{p(\tau_t|\mu)} G(\tau_t) = \frac{p(A_t|S_t,\pi)p(R_{t+1},S_{t+1}|S_t,A_t)p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)p(R_{t+1},S_{t+1}|S_t,A_t)p(A_{t+1}|S_{t+1},\mu)\cdots} G_t$$

Importance Sampling for Off-Policy Monte-Carlo

- Goal: estimate v_π
- Data: trajectory $\tau_t = \{S_t, A_t, R_{t+1}, S_{t+1}, \ldots\}$ generated with μ
- Solution: use return $G(\tau_t) = G_t = R_{t+1} + \gamma R_{t+2} + \ldots$, and correct:

$$\frac{p(\tau_t|\pi)}{p(\tau_t|\mu)} G(\tau_t) = \frac{p(A_t|S_t,\pi)p(R_{t+1},S_{t+1}|S_t,A_t)p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)p(R_{t+1},S_{t+1}|S_t,A_t)p(A_{t+1}|S_{t+1},\mu)\cdots} G_t$$

$$= \frac{p(A_t|S_t,\pi)\cancel{p(R_{t+1},S_{t+1}|S_t,A_t)}p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)\cancel{p(R_{t+1},S_{t+1}|S_t,A_t)}p(A_{t+1}|S_{t+1},\mu)\cdots} G_t$$

Importance Sampling for Off-Policy Monte-Carlo

- Goal: estimate v_π
- Data: trajectory $\tau_t = \{S_t, A_t, R_{t+1}, S_{t+1}, \ldots\}$ generated with μ
- Solution: use return $G(\tau_t) = G_t = R_{t+1} + \gamma R_{t+2} + \ldots$, and correct:

$$
\begin{aligned}
\frac{p(\tau_t|\pi)}{p(\tau_t|\mu)} G(\tau_t) &= \frac{p(A_t|S_t,\pi)p(R_{t+1},S_{t+1}|S_t,A_t)p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)p(R_{t+1},S_{t+1}|S_t,A_t)p(A_{t+1}|S_{t+1},\mu)\cdots} G_t \\
&= \frac{p(A_t|S_t,\pi)\cancel{p(R_{t+1},S_{t+1}|S_t,A_t)}p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)\cancel{p(R_{t+1},S_{t+1}|S_t,A_t)}p(A_{t+1}|S_{t+1},\mu)\cdots} G_t \\
&= \frac{p(A_t|S_t,\pi)p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)p(A_{t+1}|S_{t+1},\mu)\cdots} G_t
\end{aligned}
$$

Importance Sampling for Off-Policy Monte-Carlo

- Goal: estimate v_π
- Data: trajectory $\tau_t = \{S_t, A_t, R_{t+1}, S_{t+1}, \ldots\}$ generated with μ
- Solution: use return $G(\tau_t) = G_t = R_{t+1} + \gamma R_{t+2} + \ldots$, and correct:

$$\begin{aligned}
\frac{p(\tau_t|\pi)}{p(\tau_t|\mu)} G(\tau_t) &= \frac{p(A_t|S_t,\pi)p(R_{t+1},S_{t+1}|S_t,A_t)p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)p(R_{t+1},S_{t+1}|S_t,A_t)p(A_{t+1}|S_{t+1},\mu)\cdots} G_t \\
&= \frac{p(A_t|S_t,\pi)\cancel{p(R_{t+1},S_{t+1}|S_t,A_t)}p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)\cancel{p(R_{t+1},S_{t+1}|S_t,A_t)}p(A_{t+1}|S_{t+1},\mu)\cdots} G_t \\
&= \frac{p(A_t|S_t,\pi)p(A_{t+1}|S_{t+1},\pi)\cdots}{p(A_t|S_t,\mu)p(A_{t+1}|S_{t+1},\mu)\cdots} G_t \\
&= \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \cdots G_t
\end{aligned}$$

Importance Sampling for Off-Policy TD Updates

- Use TD targets generated from μ to evaluate π
- Weight TD target $r + \gamma v(s')$ by importance sampling
- Only need a single importance sampling correction

$$v(S_t) \leftarrow v(S_t) \; + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \left(R_{t+1} + \gamma v(S_{t+1})\right) - v(S_t) \right)$$

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Expected Sarsa

- We now consider off-policy learning of action-values $q(s, a)$
- No importance sampling is required
- Next action may be chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_{t+1})$
- But we consider probabilities under $\pi(\cdot|S_t)$
- Update $q(S_t, A_t)$ towards value of alternative action

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \left(\boldsymbol{R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) q(S_{t+1}, a)} - q(S_t, A_t) \right)$$

- Called **Expected Sarsa** (sometimes called 'General Q-learning')
- Q-learning is a special case with greedy target policy π

Off-Policy Control with Q-Learning

- We want behaviour and target policies to **improve**
- E.g., the target policy π is **greedy** w.r.t. $q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\text{argmax}}\, q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. **ϵ-greedy** w.r.t. $q(s, a)$
- The Q-learning target is:

$$\begin{aligned} & R_{t+1} + \gamma \sum_a \pi^{\text{greedy}}(a|S_{t+1}) q(S_{t+1}, a) \\ = & R_{t+1} + \gamma \max_a q(S_{t+1}, a) \end{aligned}$$

On-Policy Control with Sarsa

- In Sarsa, the target and behaviour policies are the same

$$\textit{target} = R_{t+1} + \gamma q(S_{t+1}, A_{t+1})$$

- Then, for convergence, we need the addition requirement that π becomes greedy
- For instance, ϵ-greedy, or softmax with decreasing exploration

Questions?

The only stupid question is the one you were afraid to ask but never did.
-Rich Sutton

For questions that arise outside of class, please use Moodle