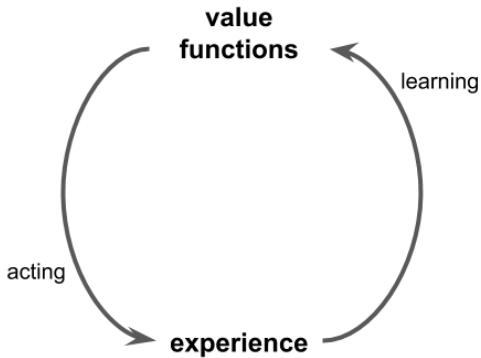
Dynamic Programming and Model-Free RL

- Dynamic Programming
  - Assume a model
  - Solve model, no need to interact with the world at all.
- Model-Free RL
  - No model
  - Learn value functions from experience.

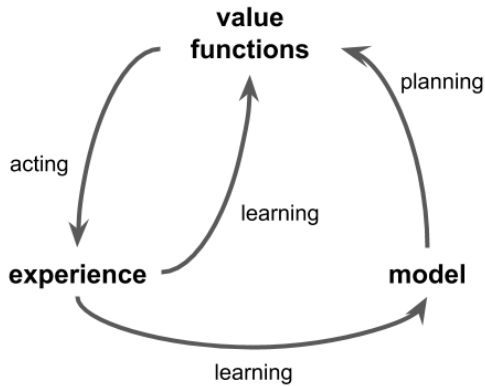
Model-Free RL



Model-Based RL

- Model-Based RL
  - Learn a model from experience
  - Plan value functions using the learned model.

Model-Based RL



Monte-Carlo Policy Evaluation

- Now we consider **sequential decision problems**
- Goal: learn v_π from episodes of experience under policy π

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

- The **return** is the total discounted reward (for an episode ending at time $T > t$):

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

- The value function is the expected return:

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

- We can just use **sample average** return instead of **expected** return
- We call this **Monte Carlo policy evaluation**

Temporal Difference Learning by Sampling Bellman Equations

- Previous lecture: Bellman equations,

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

- Previous lecture: Approximate by iterating,

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

- We can sample this!

$$v_{t+1}(S_t) = R_{t+1} + \gamma v_t(S_{t+1})$$

- This is likely quite noisy — better to take a small step (with parameter α):

$$v_{t+1}(S_t) = v_t(S_t) + \alpha_t\Bigg(\underbrace{R_{t+1} + \gamma v_t(S_{t+1})}_{\text{target}} - v_t(S_t)\Bigg)$$

(Note: tabular update)

Temporal difference learning

- **Prediction** setting: learn v_π online from experience under policy π
- Monte-Carlo
  - Update value $v_n(S_t)$ towards sampled return $\boldsymbol{G_t}$

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

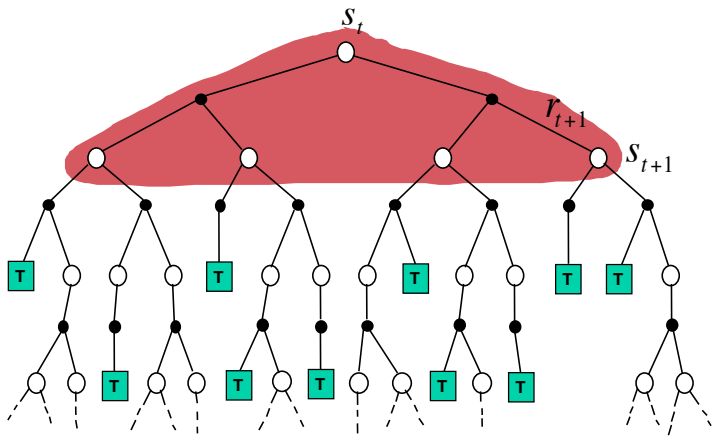
- Temporal-difference learning:
  - Update value $v_t(S_t)$ towards estimated return $\boldsymbol{R_{t+1} + \gamma v(S_{t+1})}$

$$v_{t+1}(S_t) \leftarrow v_t(S_t) + \alpha \left(\overbrace{\underbrace{\boldsymbol{R_{t+1} + \gamma v_t(S_{t+1})}}_{\text{target}} - v_t(S_t)}^{\text{TD error}} \right)$$

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

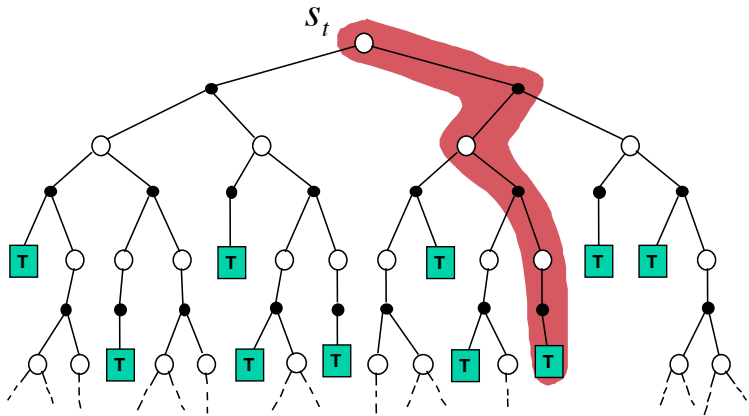
Dynamic Programming Backup

$$v(S_t) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid A_t \sim \pi(S_t)\right]$$



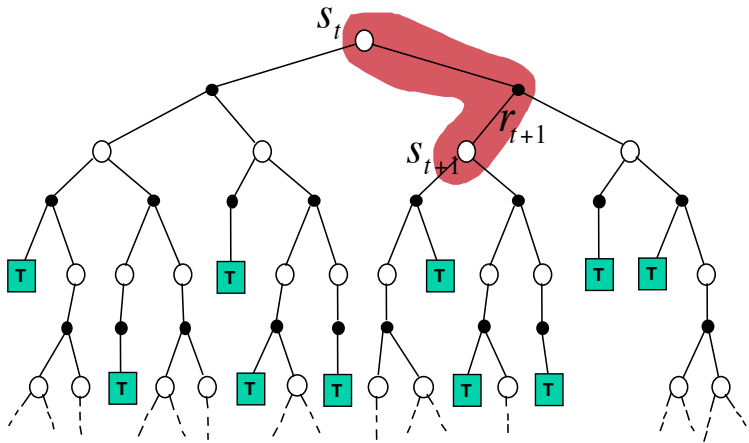
Monte-Carlo Backup

$$v(S_t) \leftarrow v(S_t) + \alpha\,(G_t - v(S_t))$$



Temporal-Difference Backup

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



Bias/Variance Trade-Off

- MC return $G_t = R_{t+1} + \gamma R_{t+2} + \dots$ is an **unbiased** estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma v_t(S_{t+1})$ is a **biased** estimate of $v_\pi(S_t)$ (unless $v_t(S_{t+1}) = v_\pi(S_{t+1})$)
- But the TD target has **lower variance**:
  - Return depends on **many** random actions, transitions, rewards
  - TD target depends on **one** random action, transition, reward

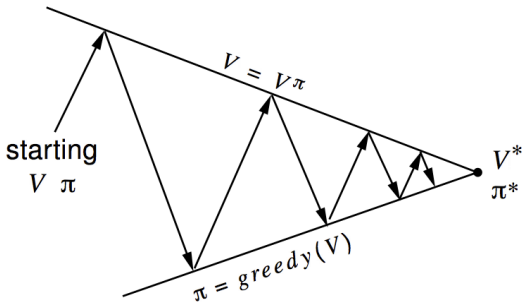
Temporal difference learning

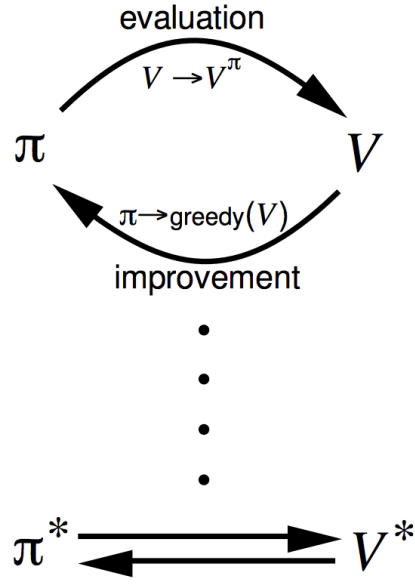
- We can apply the same idea to **action values**
- Temporal-difference learning for action values:
  - Update value $q_t(S_t, A_t)$ towards estimated return $\boldsymbol{R_{t+1} + \gamma q(S_{t+1}, A_{t+1})}$

$$q_{t+1}(S_t, A_t) \leftarrow q_t(S_t, A_t) + \alpha \left(\overbrace{\underbrace{\boldsymbol{R_{t+1} + \gamma q_t(S_{t+1}, A_{t+1})}}_{\text{target}} - q_t(S_t, A_t)}^{\text{TD error}} \right)$$

  - This algorithm is known as **SARSA**, because it uses $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

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

Model-Free Policy Iteration Using Action-Value Function

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

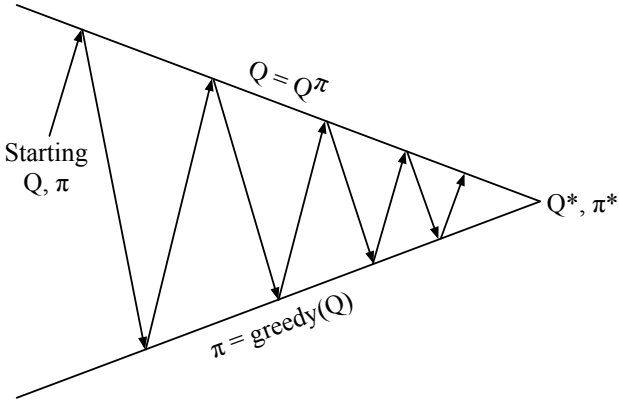
$$\pi'(s) = \underset{a}{\operatorname{argmax}}\, \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s, A_t = a\right]$$

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

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

- This makes action values convenient

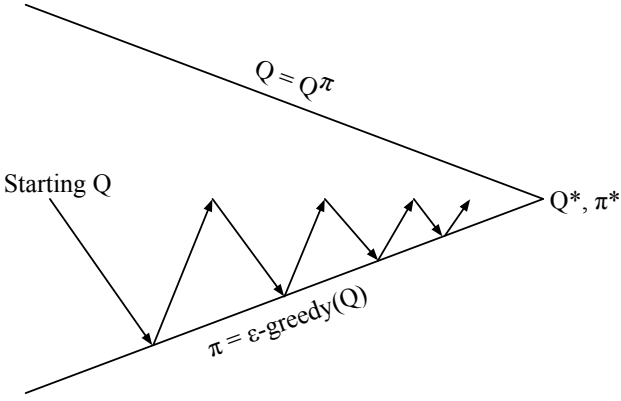
Generalised Policy Iteration with Action-Value Function



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

Policy improvement Greedy policy improvement? No exploration!
(Can't sample all s, a, when learning by interacting)

Monte-Carlo Generalized Policy Iteration



Every episode:

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

Policy improvement ϵ-greedy policy improvement

Model-free control

Repeat:

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

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t \left(G_t - q(S_t, A_t)\right)$$

- E.g.,

$$\alpha_t = \frac{1}{N(S_t, A_t)} \quad \text{of} \quad \alpha_t = 1/k$$

- Improve policy based on new action-value function

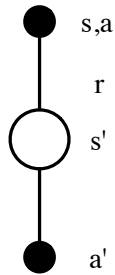
$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(q)$$

(Generalises the ϵ-greedy bandit algorithm)

MC vs. TD Control

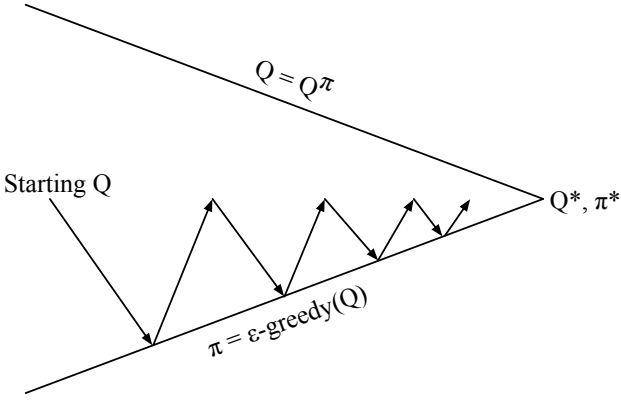
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Can learn from incomplete sequences
- Natural idea: use TD instead of MC for control
  - Apply TD to $q(s, a)$
  - Use, e.g., ϵ-greedy policy improvement
  - Update every time-step

Updating Action-Value Functions with SARSA



$$q_{t+1}(S_t, A_t) = q_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)\right)$$

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

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 **multiple** policies while following **one** policy
  - Learn about **greedy** policy while following **exploratory** policy
- **Q-learning** estimates the value of the **greedy** policy

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

Acting greedy all the time would not explore sufficiently

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. $q(s, a)$

$$\pi(S_{t+1}) = \operatorname*{argmax}_{a'} q(S_{t+1}, a')$$

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

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

Model-Based RL

- Model-Based RL
  - Learn a model from experience
  - Plan value functions using the learned model.

Why should we even consider this?

One clear disadvantage:

- First learn a model, then construct a value function
  $\Rightarrow$ two sources of approximation error
- Learn a value function directly
  $\Rightarrow$ only one source of approximation error

However:

- Models can efficiently be learned by supervised learning methods
- Reason about model uncertainty (better exploration?)
- Reduce the interactions in the real world (data efficiency? faster/cheaper?).

Model Learning - I

Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, ..., S_T\}$

- This is a supervised learning problem

$$S_1, A_1 \rightarrow R_2, S_2$$
$$\vdots$$
$$S_{T-1}, A_{T-1} \rightarrow R_T, S_T$$

- over a dataset of state transitions observed in the environment.

Dynamic Programming with a learned Model

Once learned a model $\hat{p}_\eta$ from experience:

- Solve the MDP $\langle \mathcal{S}, \mathcal{A}, \hat{p}_\eta \rangle$
- Using favourite dynamic programming algorithm
  - Value iteration
  - Policy iteration
  - ...

Sample-Based Planning with a learned Model

A simple but powerful approach to planning:

- Use the model only to generate samples
- Sample experience from model

$$S, R \sim \hat{p}_\eta(\cdot \mid s, a)$$

- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning

Limits of Planning with an Inaccurate Model - II

How can we deal with the inevitable inaccuracies of a learned model?

- Approach 1: when model is wrong, use model-free RL
- Approach 2: reason about model uncertainty over η (e.g. Bayesian methods)
- Approach 3: Combine model-based and model-free methods in a single algorithm.

Integrating Learning and Planning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
  - Learn a model from real experience
  - Plan value function (and/or policy) from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn AND plan value function (and/or policy) from real and simulated experience
  - Treat real and simulated experience equivalently. Conceptually, the updates from learning or planning are not distinguished.

Dyna-Q Algorithm

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $s \leftarrow$ current (nonterminal) state

(b) $a \leftarrow \varepsilon$-greedy(s,Q)

(c) Execute action a; observe resultant state, s', and reward, r

(d) $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)

(f) Repeat N times:

$s \leftarrow$ random previously observed state

$a \leftarrow$ random action previously taken in s

$s', r \leftarrow Model(s,a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Policy-Based Reinforcement Learning

- Previously we approximated paramateric value functions

$$v_{\boldsymbol{w}}(s) \approx v_\pi(s)$$
$$q_{\boldsymbol{w}}(s,a) \approx q_\pi(s,a)$$

- A policy can be generated from these values (e.g., greedy)
- In this lecture we directly parametrise the **policy** directly

$$\pi_{\boldsymbol{\theta}}(a|s) = p(a|s,\boldsymbol{\theta})$$

- This lecture, we focus on **model-free** reinforcement learning

Policy Objective Functions

- Goal: given **policy $\pi_\theta(s, a)$**, find best **parameters θ**
- How do we measure the quality of a policy π_θ?
- In episodic environments we can use the **average total return per episode**
- In continuing environments we can use the **average reward per step**

Policy Objective Functions: Episodic

- **Episodic-return objective**:

$$
\begin{aligned}
J_{\mathrm{G}}(\boldsymbol{\theta}) &= \mathbb{E}_{S_0 \sim d_0, \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \\
&= \mathbb{E}_{S_0 \sim d_0, \pi_{\boldsymbol{\theta}}} [G_0] \\
&= \mathbb{E}_{S_0 \sim d_0} [\mathbb{E}_{\pi_{\boldsymbol{\theta}}} [G_t \mid S_t = S_0]] \\
&= \mathbb{E}_{S_0 \sim d_0} [v_{\pi_{\boldsymbol{\theta}}}(S_0)]
\end{aligned}
$$

where d_0 is the start-state distribution This objective equals the expected value of the start state

Policy Objective Functions: Average Reward

- **Average-reward objective**

$$
\begin{aligned}
J_{\mathrm{R}}(\boldsymbol{\theta}) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[R_{t+1}\right] \\
&= \mathbb{E}_{S_t \sim d_{\pi_{\boldsymbol{\theta}}}}\left[\mathbb{E}_{A_t \sim \pi_{\boldsymbol{\theta}}(S_t)}\left[R_{t+1} \mid S_t\right]\right] \\
&= \sum_s d_{\pi_{\boldsymbol{\theta}}}(s) \sum_a \pi_{\boldsymbol{\theta}}(s,a) \sum_r p(r \mid s,a) r
\end{aligned}
$$

where $d_\pi(s) = p(S_t = s \mid \pi)$ is the probability of being in state s in the long run
Think of it as the ratio of time spent in s under policy π

Policy Optimisation

- Policy based reinforcement learning is an **optimization** problem
- Find $\boldsymbol{\theta}$ that maximises $J(\boldsymbol{\theta})$
- We will focus on **stochastic gradient ascent**, which is often quite efficient (and easy to use with deep nets)
- Some approaches do not use gradient
  - Hill climbing / simulated annealing
  - Genetic algorithms / evolutionary strategies

Policy Gradient

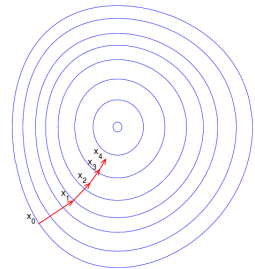
- Idea: ascent the gradient of the objective $J(\boldsymbol{\theta})$

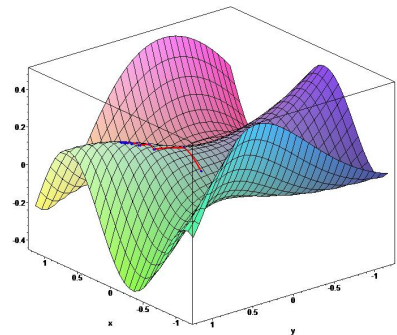
$$\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- Where $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the **policy gradient**

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_n} \end{pmatrix}$$

- and α is a step-size parameter
- Stochastic policies help ensure $J(\boldsymbol{\theta})$ is smooth (typically/mostly)





Critics

- A critic is a value function, learnt via **policy evaluation**:
  What is the value v_{π_θ} of policy π_θ for current parameters θ?
- This problem was explored in previous lectures, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - n-step TD

Actor-Critic

Critic Update parameters $\boldsymbol{w}$ of $v_{\boldsymbol{w}}$ by TD (e.g., one-step) or MC

Actor Update $\boldsymbol{\theta}$ by policy gradient

function One-step Actor Critic

Initialise s, $\boldsymbol{\theta}$

for t = 0, 1, 2, … **do**

Sample $A_t \sim \pi_{\boldsymbol{\theta}}(S_t)$

Sample R_{t+1} and S_{t+1}

$\delta_t = R_{t+1} + \gamma v_{\boldsymbol{w}}(S_{t+1}) - v_{\boldsymbol{w}}(S_t)$ [one-step TD-error, or **advantage**]

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta\, \delta_t\, \nabla_{\boldsymbol{w}} v_{\boldsymbol{w}}(S_t)$ [TD(0)]

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \delta_t\, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)$ [Policy gradient update (ignoring γ^t term)]

End of Lecture