535514: Reinforcement Learning

Lecture 11 — Value Function Approximation and A2C Algorithm

Ping-Chun Hsieh

March 28, 2024

Quick Review

1. (Incremental) Monte-Carlo update:

$$V_{k+1}(s_t) \leftarrow V_k(s_t) + \alpha(\underline{\qquad\qquad} - V_k(s_t))$$

2. TD(0) update:

$$V_{k+1}(s_t) \leftarrow V_k(s_t) + \alpha(\underline{\qquad\qquad} - V_k(s_t))$$

3. n-step TD update:

$$V_{k+1}(s_t) \leftarrow V_k(s_t) + \alpha(\underline{\qquad\qquad} - V_k(s_t))$$

4. TD(λ) update:

$$V_{k+1}(s_t) \leftarrow V_k(s_t) + \alpha(\underline{\qquad\qquad} - V_k(s_t))$$

Next Question: How to Estimate $A^{\pi}(s, a)$?

Generalized Advantage Estimator (GAE): Using TD(λ) to Estimate $A^\pi(s,a)$

Let $V(s)$ be the current estimate of true value $V^\pi(s)$

$$\hat{A}_t^{(1)} := r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \qquad (= \delta_t)$$

$$\hat{A}_t^{(2)} := r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2}) - V(s_t) \qquad (= \delta_t + \gamma\delta_{t+1})$$

$$\hat{A}_t^{(3)} := r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V(s_{t+3}) - V(s_t) \qquad (= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2})$$

$$\hat{A}_t^{(k)} := r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^k V(s_{t+k}) - V(s_t) \qquad (= \sum_{\ell=0}^{k-1} \gamma^\ell \delta_{t+\ell})$$

- Fact: $\hat{A}_t^{(\infty)} = \sum_{\ell=0}^{\infty} \gamma^\ell \delta_{t+\ell} = G_t - V(s_t)$

- GAE Estimator:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots\right) = \sum_{\ell=0}^{\infty} (\gamma\lambda)^\ell \delta_{t+\ell}$$

Schulman et al., High-Dimensional Continuous Control Using Generalized Advantage Estimation, ICLR 2016

Algorithm: REINFORCE With GAE

Recall: (P5) REINFORCE with advantage

$$\nabla_\theta V^{\pi_\theta}(\mu) = \mathbb{E}_{\tau \sim P_\mu^{\pi_\theta}}\left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]$$

- REINFORCE with GAE

Step 1: Initialize θ_0 and step size η

Step 2: Sample a trajectory $\tau \sim P_\mu^{\pi_\theta}$ and make the update as

$$\theta_{k+1} = \theta_k + \eta\left(\sum_{t=0}^{\infty} \gamma^t \hat{A}_t^{GAE(\gamma,\lambda)} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right)$$

where $\hat{A}_t^{GAE(\gamma,\lambda)}$ is constructed from $V(s)$ learned by TD

(Repeat Step 2 until termination)

Some Discussions on GAE

1. Do we need to wait until the end of a trajectory to construct GAE?

2. How to efficiently calculate GAE for different t of the same trajectory?

```
def calculate_advantages(rewards, values, discount_factor, trace_decay, normalize = True):

    advantages = []
    advantage = 0
    next_value = 0

    for r, v in zip(reversed(rewards), reversed(values)):
        td_error = r + next_value * discount_factor - v
        advantage = td_error + advantage * discount_factor * trace_decay
        next_value = v
        advantages.insert(0, advantage)

    advantages = torch.tensor(advantages)
```

3. Where does $V(s)$ in GAE come from?

This Lecture:
Let's discuss 1 Fundamental Concept + 1 Algorithm

1. Value Function Approximation

2. Advantage Actor Critic (A2C)

References:

Richard Sutton and Andrew Barto, Reinforcement Learning: An Introduction, 2019

Mnih et al., Asynchronous Methods for Deep Reinforcement Learning, ICML 2016

Schulman et al., High-Dimensional Continuous Control Using Generalized Advantage Estimation, ICLR 2016

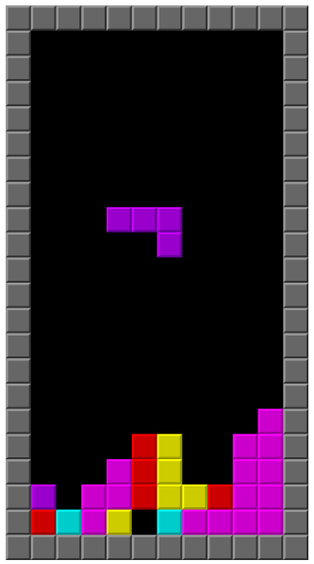
So far, we presume that $V^{\pi}(s)$ and $Q^{\pi}(s, a)$ can be predicted accurately via model-free prediction.

But, is this always true?

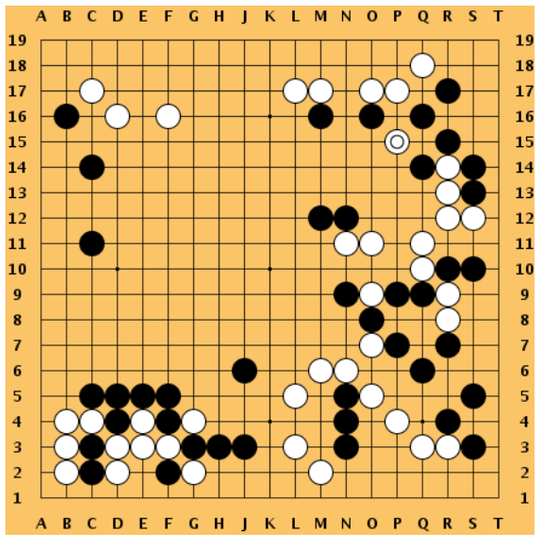
Tabular Methods for Large RL Problems?

- So far: Value functions ($V(s)$ or $Q(s,a)$) are represented by a lookup table
- Such tabular methods may be impractical for large RL problems
  - Too many states (or actions) to store in memory
  - It is slow to learn $V(s)$ or $Q(s,a)$ for each state separately

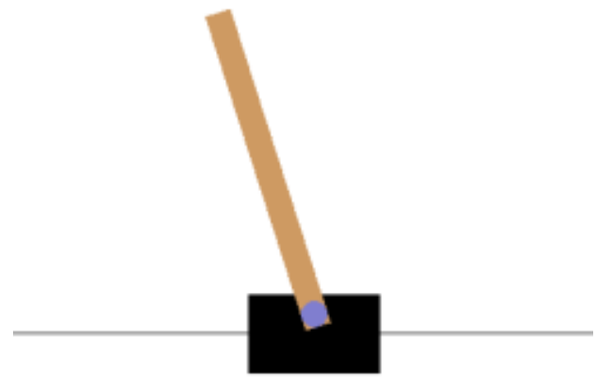
- Examples:



Tetris: $\sim 2^{200}$ states



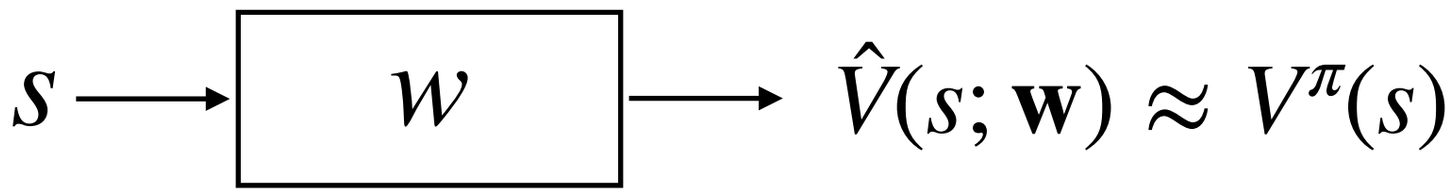
Go: $\sim 10^{170}_{9}$ states

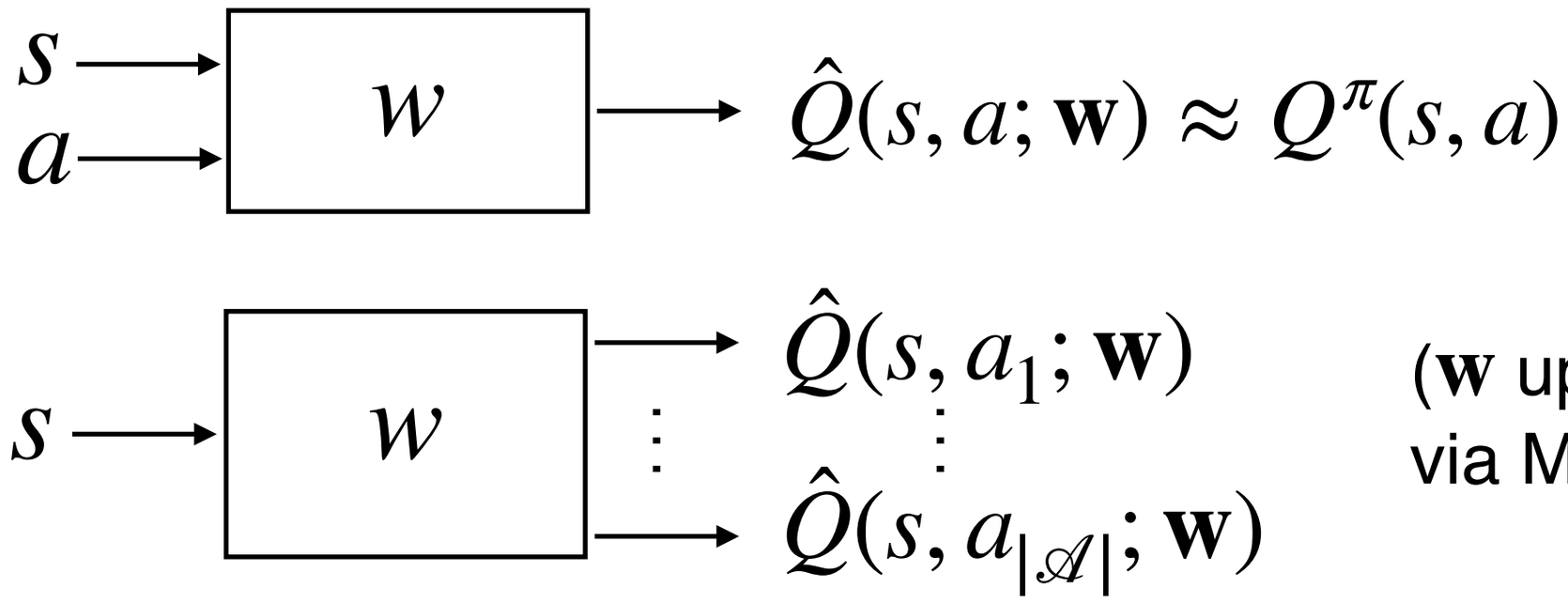


Cartpole: continuous state space

Value Function Approximation (VFA)

- To scale up the model-free methods, function approximation is commonly used to learn value functions
- Idea: Approximate a value function by a parametric function





($\mathbf{w}$ updated via MC or TD)

- Motivation: Generalize from seen states to unseen states

2 Commonly-Used Function Approximators

1. Linear combinations of features
2. Neural networks

- Both are differentiable function approximators (Why?)

How to Quantify the Accuracy of VFA?

- For each state s, the squared error between $V^{\pi}(s)$ and $\hat{V}(s;\mathbf{w})$:

$$\left(V^{\pi}(s) - \hat{V}(s;\mathbf{w})\right)^2$$

- To jointly consider all states, we use mean squared error (MSE):

$$F(\mathbf{w}) := \sum_{s \in \mathcal{S}} \rho(s)\left(V^{\pi}(s) - \hat{V}(s;\mathbf{w})\right)^2$$

- $\rho(s)$ are the weights for mixing the MSE of the states
- If $\rho(s)$ is a probability distribution, then the objective becomes:

$$F(\mathbf{w}) := \mathbb{E}_{s \sim \rho(s)}\left[\left(V^{\pi}(s) - \hat{V}(s;\mathbf{w})\right)^2\right]$$

Some Natural Choices of $\rho(s)$

- For continuing environments:

  1. choose $\rho(s) \equiv d_\mu^\pi(s) := \mathbb{E}_{s_0 \sim \mu}\left[(1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s \mid s_0, \pi)\right]$
  2. choose $\rho(s) \equiv$ undiscounted stationary state distribution

- For episodic environments:

  1. choose $\rho(s) \equiv d_\mu^\pi(s) := \dfrac{\mathbb{E}_{s_0 \sim \mu}\left[\sum_{t=0}^{T}\gamma^t P(s_t = s \mid s_0, \pi)\right]}{\text{normalization constant}}$

- Remark: $\rho(s)$ usually corresponds to the sampling strategy (will be discussed momentarily)

Value Function Approximation via GD

- Goal: find $\mathbf{w}$ that minimizes MSE between $V^\pi(s)$ and $\hat{V}(s;\mathbf{w})$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{\mathbb{E}_{s\sim\rho(s)}\left[\left(V^\pi(s) - \hat{V}(s;\mathbf{w})\right)^2\right]}_{=:F(\mathbf{w})}$$

- Suppose: We are given an oracle for querying $V^\pi(s)$
- Iterative GD update:

$$\begin{aligned}\mathbf{w}_{k+1} &= \mathbf{w}_k + \alpha_k \nabla_{\mathbf{w}} F(\mathbf{w}) \\ &= \mathbf{w}_k + \alpha_k \mathbb{E}_{s\sim\rho(s)}\left[\left(V^\pi(s) - \hat{V}(s;\mathbf{w})\right) \nabla_{\mathbf{w}} \hat{V}(s;\mathbf{w})\right]\end{aligned}$$

- GD finds a local minimum under proper step sizes α_k (Why?)

Value Function Approximation via SGD

- Iterative GD update:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbb{E}_{s\sim\rho(s)}\Big[\big(V^\pi(s) - \hat{V}(s;\mathbf{w}_k)\big)\,\nabla_{\mathbf{w}}\hat{V}(s;\mathbf{w}_k)\Big]$$

- Iterative SGD update by the sampled state $S \sim \rho$:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k\Big[\big(V^\pi(S) - \hat{V}(S;\mathbf{w}_k)\big)\,\nabla_{\mathbf{w}}\hat{V}(S;\mathbf{w}_k)\Big]$$

- The SGD update converges to a local minimum under stochastic approximation conditions of α_k (even for NN)

In practice, we don't have an oracle for $V^{\pi}(s)$.

What shall we do?

Monte-Carlo Value Function Approximation

- Recall: Iterative SGD update by the sampled state $S \sim \rho$:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \Big[\big(V^\pi(S) - \hat{V}(S; \mathbf{w}_k) \big) \nabla_{\mathbf{w}} \hat{V}(S; \mathbf{w}_k) \Big]$$

- Idea: Use MC, i.e. sample return G_t to estimate $V^\pi(S)$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \Big[\big(G_t - \hat{V}(S; \mathbf{w}_k) \big) \nabla_{\mathbf{w}} \hat{V}(S; \mathbf{w}_k) \Big]$$

G_t is an unbiased estimate of $V^\pi(S)$

- Equivalent to *supervised learning* with (noisy) training data as

$$(s_0, G_0), (s_1, G_1), \cdots, (s_t, G_t) \cdots$$

Monte-Carlo Value Function Approximation (Cont.)

- First-Visit MC Value Function Approximation:

Step 1: Initialize $\mathbf{w} = 0$ and $k = 1$

Step 2: Sample $\tau_k = (s_0, a_0, r_1, \cdots, s_{L_k-1}, a_{L_k-1}, r_{L_k}) \sim P_\mu^{\pi_\theta}$

For each step of the current episode $t = 0,1,\cdots L_k - 1$

If First visit to state s in episode k

$$G_t(s) = \sum_{i=t+1}^{L_k} r_i$$

$$\Delta \mathbf{w}_k \leftarrow \Delta \mathbf{w}_k + \alpha_k \left[\left(G_t(s) - \hat{V}(S; \mathbf{w}_k) \right) \nabla_{\mathbf{w}} \hat{V}(S; \mathbf{w}_k) \right]$$

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \Delta \mathbf{w}_k, \; k \leftarrow k + 1$$

- Convergence to a local minimum can be achieved with proper step sizes α_k, for general function approximators (Why?)

Temporal-Difference Value Function Approximation

- Recall: Iterative SGD update by the sampled state $S \sim \rho$:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \Big[\big(V^\pi(S) - \hat{V}(S; \mathbf{w}_k) \big) \nabla_{\mathbf{w}} \hat{V}(S; \mathbf{w}_k) \Big]$$

- Idea: Use bootstrapping (e.g.TD(0)) to estimate $V^\pi(S)$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \Big[\big(r + \gamma \hat{V}(S'; \mathbf{w}_k) - \hat{V}(S; \mathbf{w}_k) \big) \nabla_{\mathbf{w}} \hat{V}(S; \mathbf{w}_k) \Big]$$

$r + \gamma \hat{V}(S'; \mathbf{w}_k)$ is a biased estimate of $V^\pi(S)$

- Equivalent to *supervised learning* with (noisy) training data as

$$(s_0, r_1 + \gamma \hat{V}(s_1; \mathbf{w})), \cdots, (s_t, r_{t+1} + \gamma \hat{V}(s_{t+1}; \mathbf{w})) \cdots$$

- This can be easily extended to the more general TD(λ)

TD Value Function Approximation (Cont.)

- TD(0) Value Function Approximation:

Step 1: Initialize $\mathbf{w} = 0$ and $k = 1$

Step 2: Sample $\tau_k = (s_0, a_0, r_1, \cdots, s_{L_k-1}, a_{L_k-1}, r_{L_k}) \sim P_\mu^{\pi_\theta}$

For each step of the current episode $t = 0,1,\cdots L_k - 1$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left[\left(r_{t+1} + \gamma\hat{V}(s_{t+1};\mathbf{w}) - \hat{V}(s_t;\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{V}(s_t;\mathbf{w})\right]$$

Put Everything Together:

PG + TD + VFA = Advantage Actor-Critic (A2C)

Recall: Advantage Actor-Critic (A2C) via TD

- An example of actor-critic algorithm:
  - Critic: estimate Q^{π_θ} by TD(0) bootstrapping
  - Actor: updates policy parameters θ by policy gradient
- Advantage Actor-Critic (A2C):

Step 1: Initialize θ_0 and step size α

Step 2: Sample a trajectory $\tau = (s_0, a_0, r_1, \cdots) \sim P_\mu^{\pi_\theta}$

For each step of the current trajectory $t = 0,1,2,\cdots$

$$\Delta\theta_k \leftarrow \Delta\theta_k + \alpha\gamma^t \left(r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) \right) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

Update value function $\hat{V}(s_t)$ by TD(0)

$$\theta_{k+1} \leftarrow \theta_k + \Delta\theta_k$$

A2C With Value Function Approximation

- $\hat{V}_w(s)$ is learned by value function approximation (e.g. by using a neural network or linear combinations of features)

- Advantage Actor-Critic (A2C) with Value Function Approximation:

Step 1: Initialize θ_0, w_0 and step sizes α_θ, α_w

Step 2: Sample a trajectory $\tau = (s_0, a_0, r_1, \cdots) \sim P_\mu^{\pi_\theta}$

For each step of the current trajectory $t = 0,1,2,\cdots$

$$\Delta\theta_k \leftarrow \Delta\theta_k + \alpha_\theta \gamma^t \left(r_t + \gamma \hat{V}_{w_k}(s_{t+1}) - \hat{V}_{w_k}(s_t) \right) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

$$\Delta w_k \leftarrow \Delta w_k + \alpha_w \left(r_t + \gamma \hat{V}_{w_k}(s_{t+1}) - \hat{V}_{w_k}(s_t) \right) \nabla_w \hat{V}_w(s_t)|_{w=w_k}$$

$$\theta_{k+1} \leftarrow \theta_k + \Delta\theta_k, w_{k+1} \leftarrow w_k + \Delta w_k$$

Mnih et al., “Asynchronous Methods for Deep Reinforcement Learning”, ICML 2016

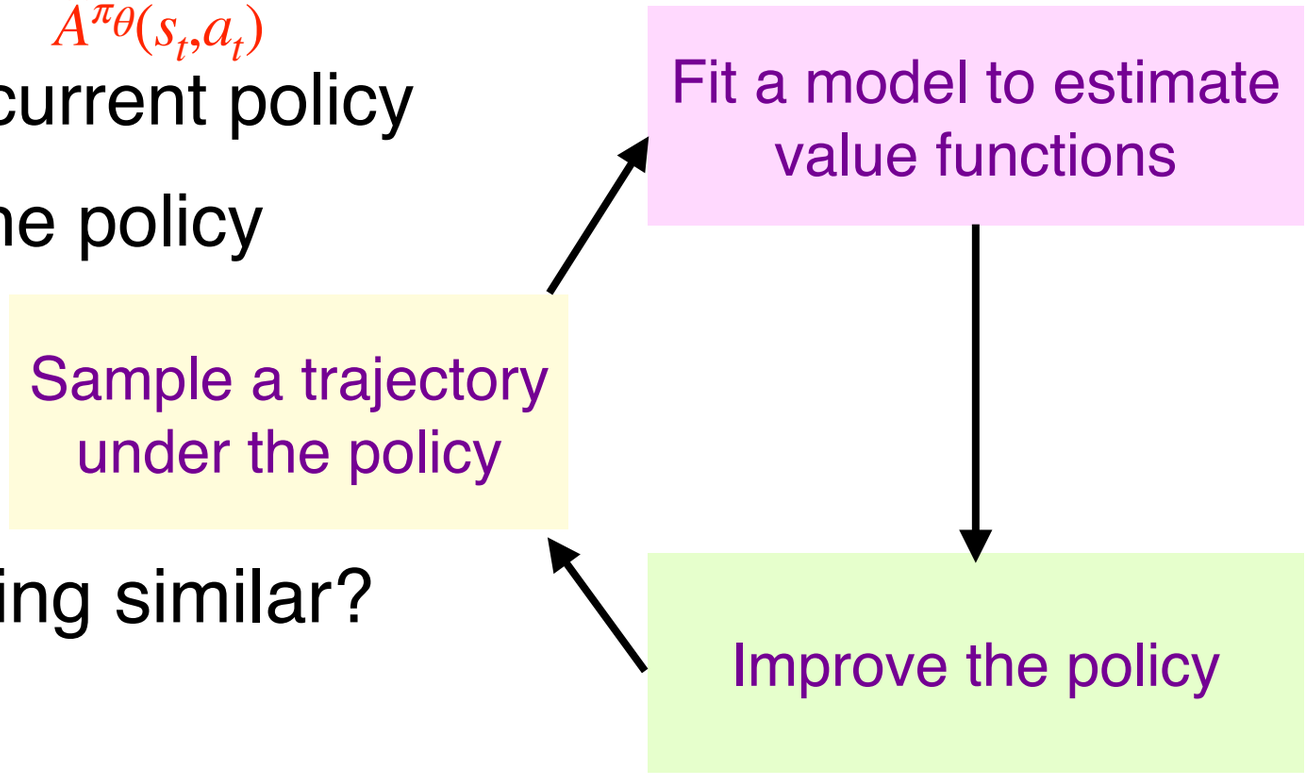
Another Interpretation of A2C

- In A2C, the policy parameters θ are updated as:

$$\Delta\theta_k \leftarrow \Delta\theta_k + \alpha_\theta \gamma^t \underbrace{\overbrace{\left(r_t + \gamma \hat{V}_{w_k}(s_{t+1})\right)}^{\hat{Q}^{\pi_\theta}(s_t,a_t)} - \hat{V}_{w_k}(s_t)}_{\hat{A}^{\pi_\theta}(s_t,a_t)} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

Step 1: Estimate $A^{\pi_\theta}(s_t, a_t)$ for the current policy

Step 2: Use $\hat{A}^{\pi_\theta}(s_t, a_t)$ to improve the policy



- Question: Have we seen anything similar?

  Policy iteration!

Step 1: Policy evaluation for the current policy (i.e. find $Q^{\pi}(s, a)$)

Step 2: Policy improvement based on Bellman optimality equations

(Slide Credit: Sergey Levine)

A2C ≈ Policy Iteration, But With Slight Difference

Policy iteration:

Step 1: Policy evaluation for the current policy (i.e. find $Q^{\pi}(s, a)$)

Step 2: Policy improvement based on Bellman optimality equations

A2C:

Step 1: Estimate $A^{\pi_\theta}(s_t, a_t)$ for the current policy

Step 2: Use $\hat{A}^{\pi_\theta}(s_t, a_t)$ to improve the policy

- Question: Is policy guaranteed to be improved in Step 2 of A2C?