



COL778: Principles of Autonomous Systems

Semester II, 2023-24

Policy Gradients – Actor Critic Methods

Rohan Paul

Outline

- Last Class
 - Policy Gradients
- This Class
 - Actor Critic Methods
- Reference Material
 - Please follow the notes as the primary reference on this topic.

“Reward to go” in Policy Gradients

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i) \right) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

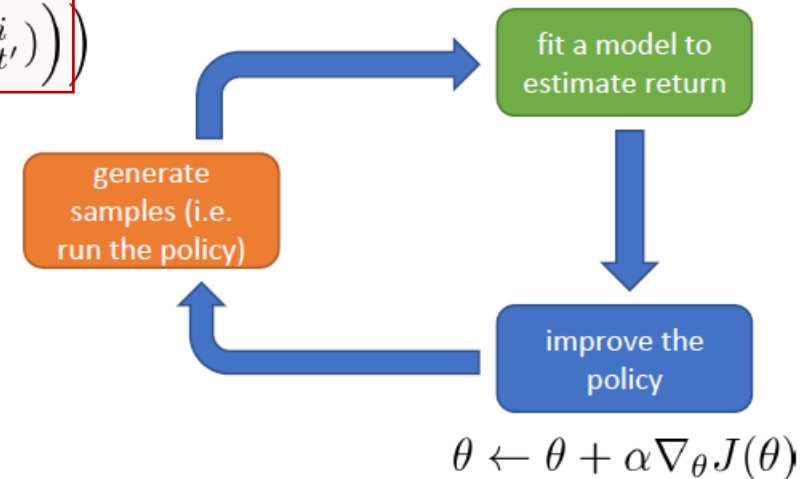
While computing rewards assigned to a roll out, consider only the future rewards (causality).

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\hat{Q}_{i,t}^\pi}_{\text{reward to go}}$$

Interpret as reward to go by taking an action a on state s, which is the Q function.

Now, estimating the Q function is part of the gradient update step

$$\hat{Q}^\pi(\mathbf{x}_t, \mathbf{u}_t) = \sum_{t'=t}^T r(\mathbf{x}_{t'}, \mathbf{u}_{t'})$$



Q-function estimation: From a single rollout

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=1}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)}_{\text{"reward to go"}}$$

$\hat{Q}_{i,t}$

$\hat{Q}_{i,t}$: estimate of expected reward if we take action $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$

Note: we interpret the second term as reward to go.

This term sums the reward from the current state till the end on “a single” roll out.

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

Better Q-function estimation from more rollouts

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=1}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)}_{\text{"reward to go"}}$$

$\hat{Q}_{i,t}$

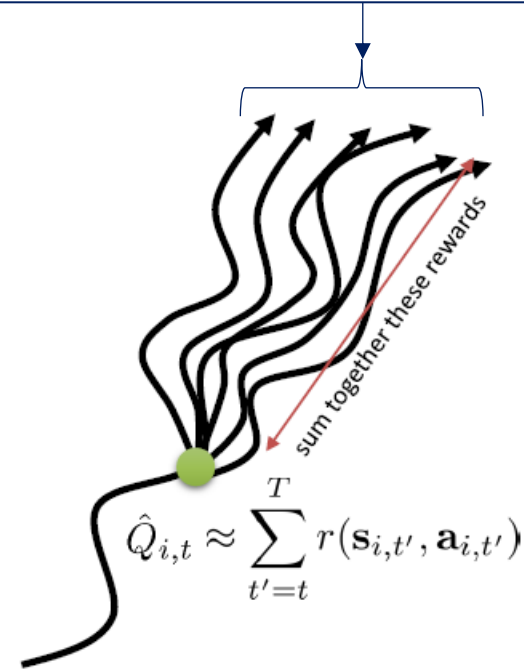
$\hat{Q}_{i,t}$: estimate of expected reward if we take action $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$

can we get a better estimate?

$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_{\theta}} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$: true *expected* reward-to-go

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

For estimating the Q function, instead of one rollout, average over the N samples to get a better estimate.



Introducing baseline to reduce variance

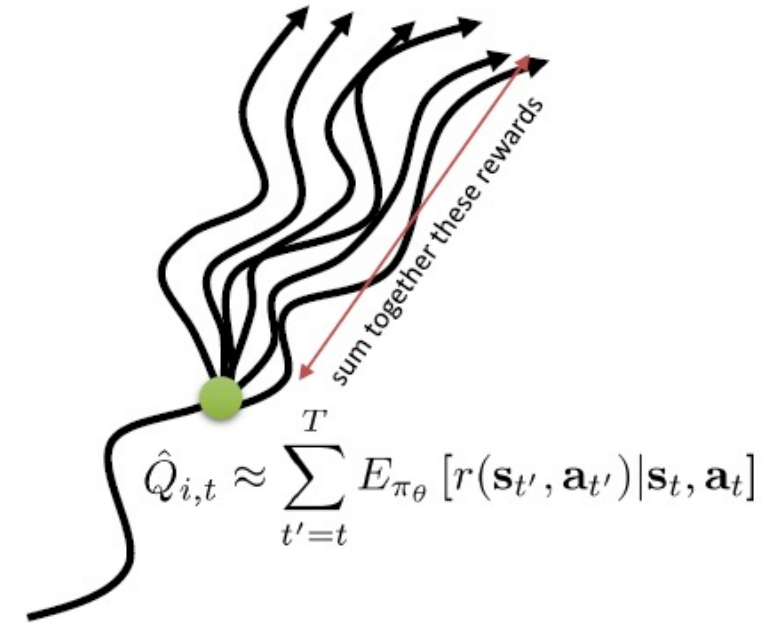
We know that variance reduction using a baseline helps policy gradient method. Hence, introduce a baseline here:

$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$: true *expected* reward-to-go

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) (Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t}))$$

$$b_t = \frac{1}{N} \sum_i Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

$$V(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} [Q(\mathbf{s}_t, \mathbf{a}_t)]$$



Advantage Function

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$: total reward from taking \mathbf{a}_t in \mathbf{s}_t

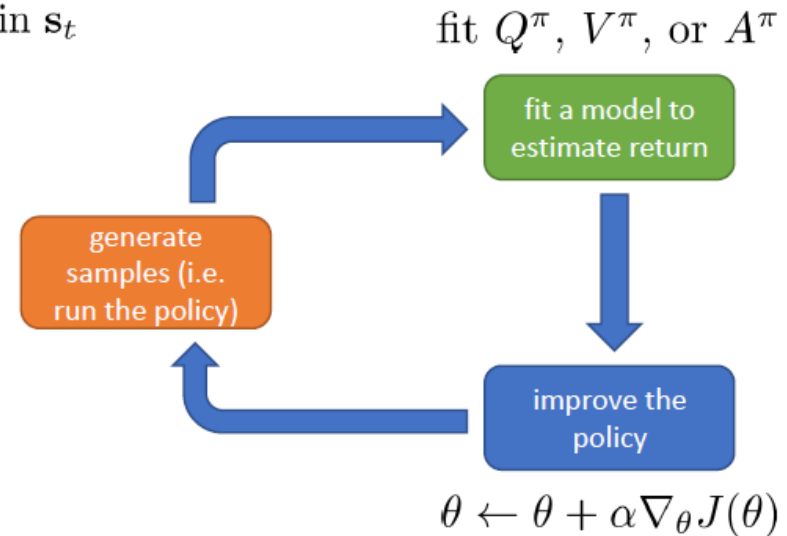
$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} [Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$: total reward from \mathbf{s}_t

$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$: how much better \mathbf{a}_t is

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$



$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=1}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) - b \right)$$



The term captures the advantage of taking an action a over the baseline.

Advantage Function:

Now there is a Q function **and** a V() function to be estimated! Do we really need two quantities?

$$Q^\pi(s_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(s_{t'}, \mathbf{a}_{t'}) | s_t, \mathbf{a}_t]$$

$$V^\pi(s_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | s_t)} [Q^\pi(s_t, \mathbf{a}_t)]$$

$$A^\pi(s_t, \mathbf{a}_t) = Q^\pi(s_t, \mathbf{a}_t) - V^\pi(s_t)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

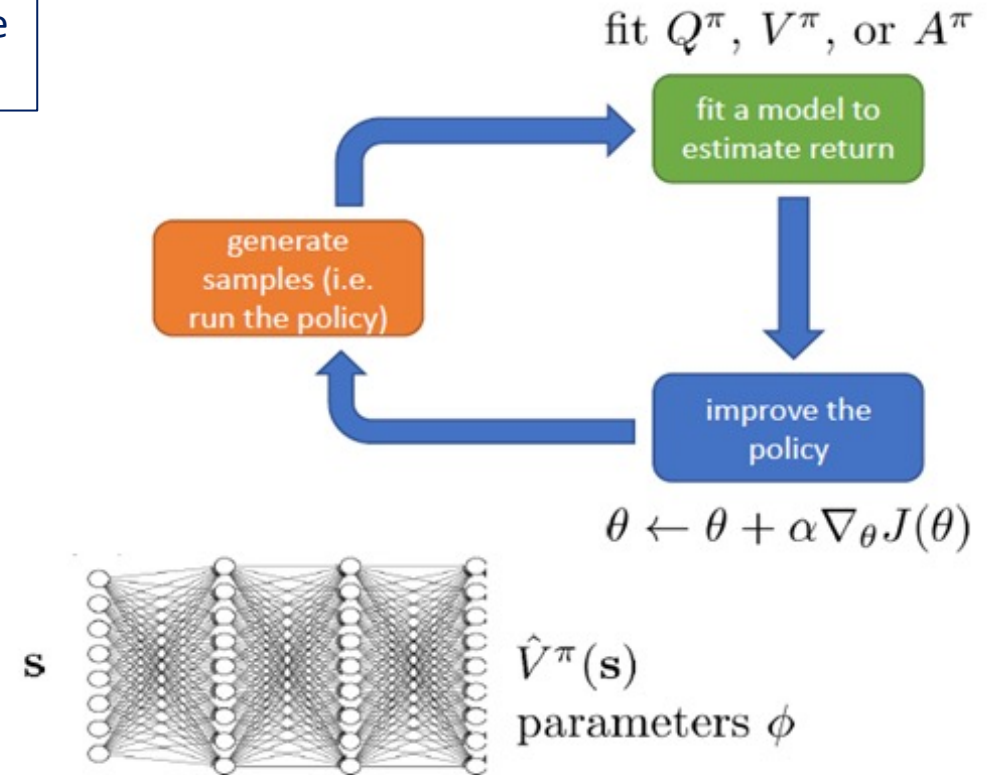
fit *what* to *what*?

Q^π, V^π, A^π ?

$$Q^\pi(s_t, \mathbf{a}_t) = r(s_t, \mathbf{a}_t) + \sum_{t'=t+1}^T E_{\pi_\theta} [r(s_{t'}, \mathbf{a}_{t'}) | s_t, \mathbf{a}_t]$$

$$A^\pi(s_t, \mathbf{a}_t) \approx r(s_t, \mathbf{a}_t) + V^\pi(s_{t+1}) - \underbrace{V^\pi(s_t)}$$

let's just fit $V^\pi(s)$!



Core Idea:

- Estimation will be using a neural network that predicts the value.
- Express estimation of the advantage function as estimating the value function for a policy (which is policy evaluation). Use Bellman relationship.

Policy Evaluation: By Averaging

Note: we are to compute the value of a particular state s under the given policy (this is policy evaluation)

$$V^\pi(s_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(s_{t'}, \mathbf{a}_{t'}) | s_t]$$

$$J(\theta) = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} [V^\pi(\mathbf{s}_1)]$$

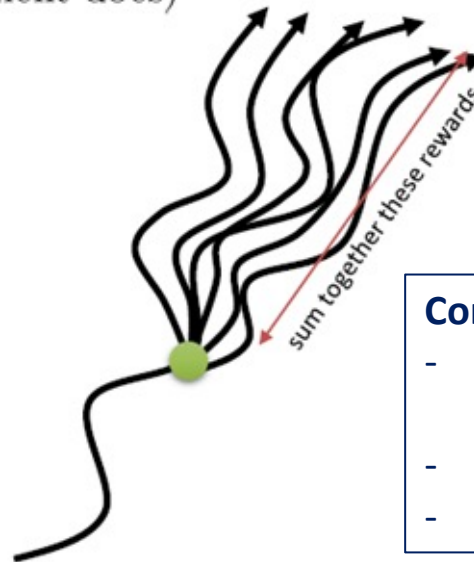
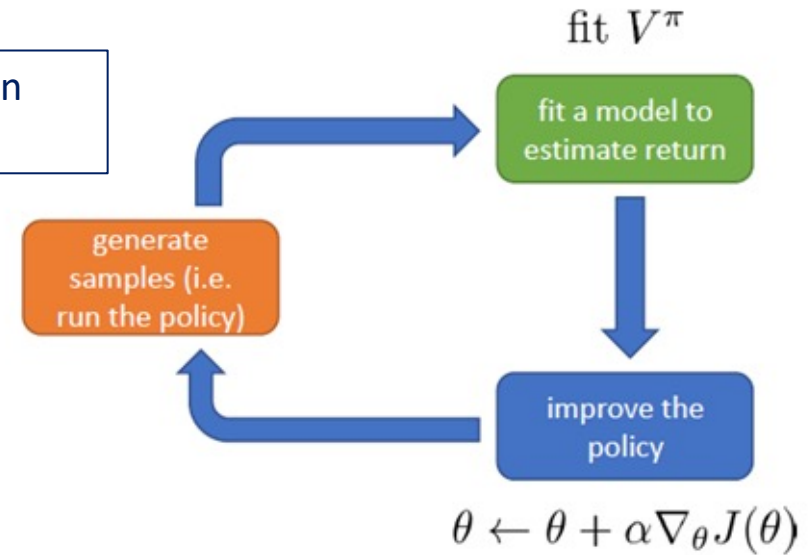
how can we perform policy evaluation?

Monte Carlo policy evaluation (this is what policy gradient does)

$$V^\pi(s_t) \approx \sum_{t'=t}^T r(s_{t'}, \mathbf{a}_{t'})$$

$$V^\pi(s_t) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t'=t}^T r(s_{t'}, \mathbf{a}_{t'})$$

(requires us to reset the simulator)

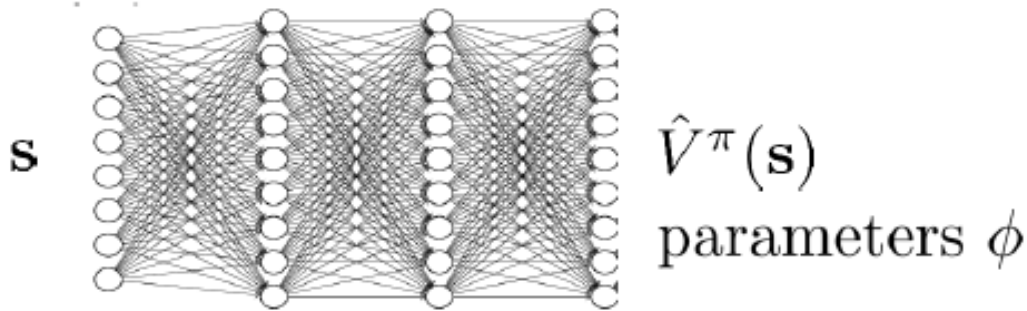


Core Idea:

- Estimation of the value function can be over a single rollout or averaged over multiple rollouts.
- Averaging -> monte carlo and a better estimate.
- Note: requires resetting the start state.

Can we do the same estimation via a neural network?

- Represent the value function as a neural network that estimates (predicts) the value for a state under policy.
- The network needs training. How to train?
- The reward observed for a rollout is the total reward the agent experiences during the rollout.
- Use the “observed” reward to supervise the network. That is, treat the observed reward as the regression target.
- Note: this is for a “single” rollout.



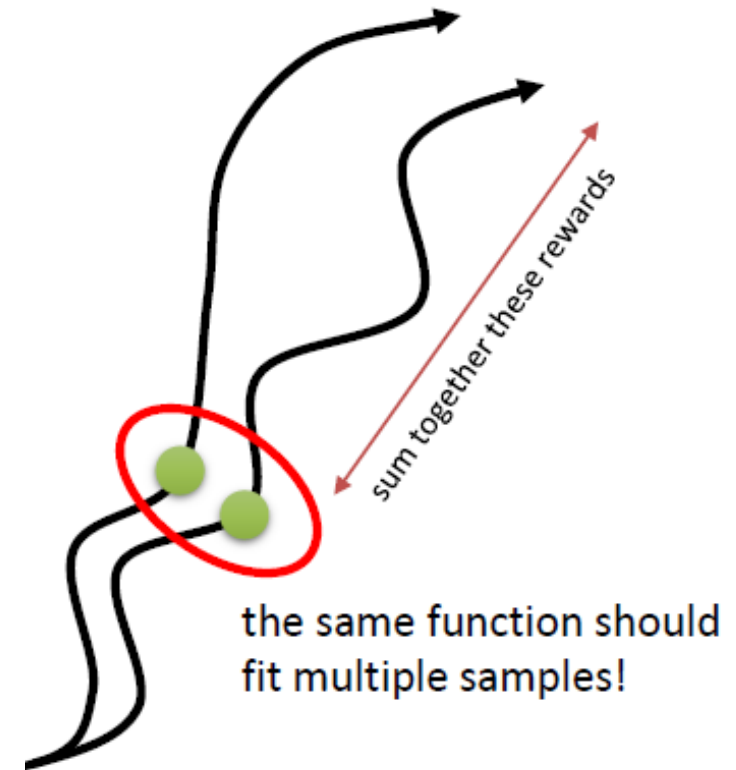
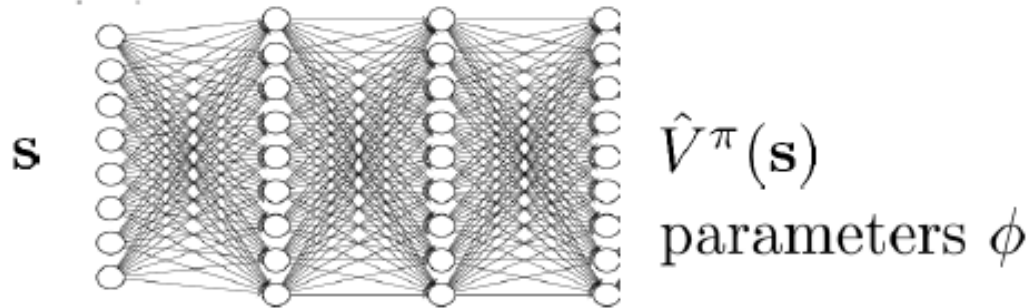
$$V^{\pi}(\mathbf{s}_t) \approx \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

$$\text{training data: } \left\{ \left(\mathbf{s}_{i,t}, \underbrace{\sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})}_{y_{i,t}} \right) \right\}$$

$$\text{supervised regression: } \mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_{\phi}^{\pi}(\mathbf{s}_i) - y_i \right\|^2$$

Let us just learn one network for predicting V

- The network is trained using “a single” rollout.
- But, it is the “same” network across rollouts.
- Finally, when trained, it is the same network that takes a state in and produces a value function output.
- Internally, the network will learn common features to generalize.



From batch data to incremental (bootstrapping)

$$\text{ideal target: } y_{i,t} = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_{i,t}] \approx \boxed{r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + V^\pi(\mathbf{s}_{i,t+1})} \approx \boxed{r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \underbrace{\hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})}_{\text{directly use previous fitted value function!}}}$$

$$\text{Monte Carlo target: } y_{i,t} = \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$$

directly use previous fitted value function!

$$\text{training data: } \left\{ \underbrace{\left(\mathbf{s}_{i,t}, r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \boxed{\hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})} \right)}_{y_{i,t}} \right\}$$

$$\text{supervised regression: } \mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

sometimes referred to as a “bootstrapped” estimate

Core Idea:

- Till now, we computed the value by summing up the reward for the states encountered.
- Alternatively, we can invoke the Bellman relationship that connects the value function for two states.
- Bootstrapping
 - Use the same network to output the value for the subsequent state.

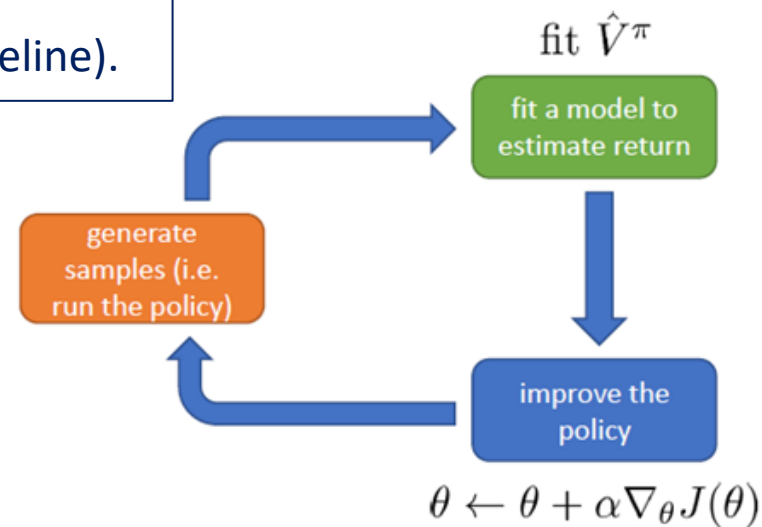
A Basic Actor-Critic algorithm

Actor: refers to the aspect of proposing an action on a state.

Critic: refers to the aspect of criticizing the proposed action (baseline).

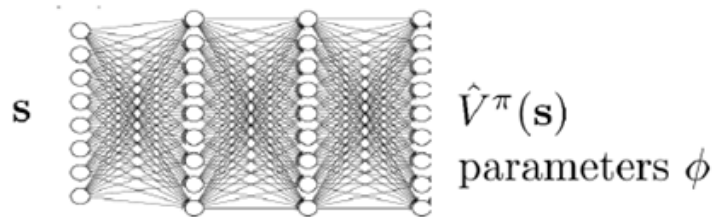
batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



$$y_{i,t} \approx \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$$


$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$



$$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t]$$

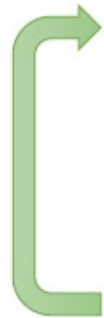
Actor-Critic Algorithm with Discounts

batch actor-critic algorithm:

- 
1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
 2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
 3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
 4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Till now, we have simply added the rewards without discounting. But we do need to incorporate the discounting.

online actor-critic algorithm:

- 
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
 2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$
 3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
 4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

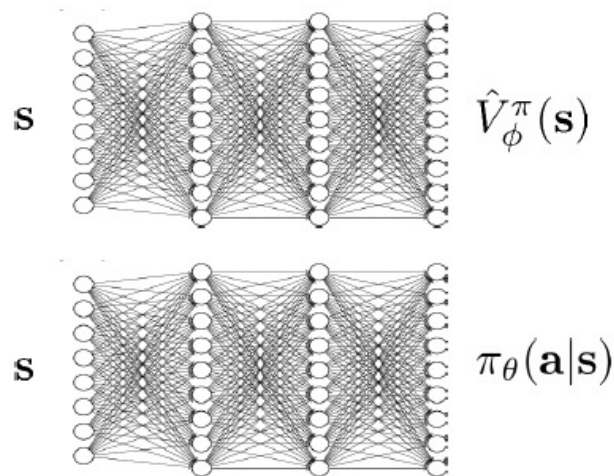
Sharing the network between the actor and the critic

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_{\theta}(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update \hat{V}_{ϕ}^{π} using target $r + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}')$
3. evaluate $\hat{A}^{\pi}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}') - \hat{V}_{\phi}^{\pi}(\mathbf{s})$
4. $\nabla_{\theta} J(\theta) \approx \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s}) \hat{A}^{\pi}(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

Both can use some part of the network – for example encoding of the state. This is like a prior injected into the network.

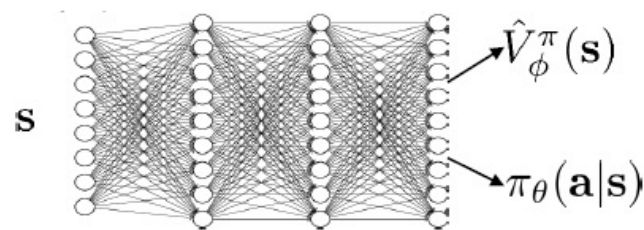
two network design



+ simple & stable

- no shared features between actor & critic

shared network design



Other Policy Gradient Successes

TD-Gammon, Gerald Tesauro 1992

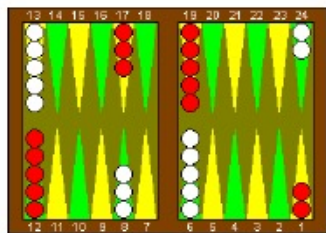


Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference, 13-9, 24-23. TD-Gammon's analysis is given in Table 2.

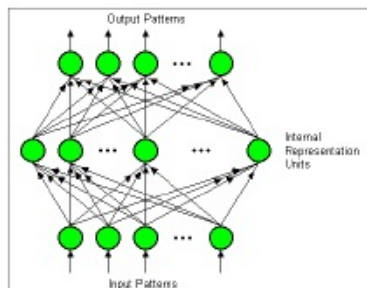


Figure 3. An illustration of the multilayer perceptron architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:

expected outcome given board state

AlphaGo, Silver et al. 2016



reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:

expected outcome given board state

Other Policy Gradient Algorithms

- A2C
- Soft Actor Critic
- A3C
- Natural Policy Gradients

Acknowledgements

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Nicholas Roy, Wolfram Burgard, Dieter Fox, Sebastian Thrun, Siddharth Srinivasa, Dan Klein, Pieter Abbeel, Max Likhachev, Alexander Amini (MIT Introduction to Deep Learning) and others. This lecture builds on material from Sergey Levine's course on Deep RL.