

UC Berkeley · CSW182 | [Deep Learning]

Designing, Visualizing and Understanding Deep Neural Networks (2021)

CSW182 (2021) · 课程资料包 @ShowMeAI



视频
中英双语字幕



课件
一键打包下载



笔记
官方笔记翻译



代码
作业项目解析



视频 · B 站 [扫码或点击链接]
<https://www.bilibili.com/video/BV1Ff4y1n7ar>



课件 & 代码 · 博客 [扫码或点击链接]
<http://blog.showmeai.tech/berkeley-csw182>

Berkeley
Q-Learning
计算机视觉

循环神经网络
风格迁移
机器学习基础

可视化
模仿学习
生成模型

梯度策略
元学习
卷积网络

Awesome AI Courses Notes Cheatsheets 是 [ShowMeAI](#) 资料库的分支系列，覆盖最具知名度的 **TOP50+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称，跳转至课程**资料包**页面，**一键下载**课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets · 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094



微信公众号

资料下载方式 2：扫码点击**底部菜单栏**
称为 **AI 内容创作者**？回复 [添砖加瓦]

Reinforcement Learning

Designing, Visualizing and Understanding Deep Neural Networks

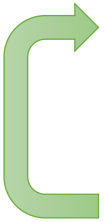
CS W182/282A

Instructor: Sergey Levine
UC Berkeley



Recap: policy gradients

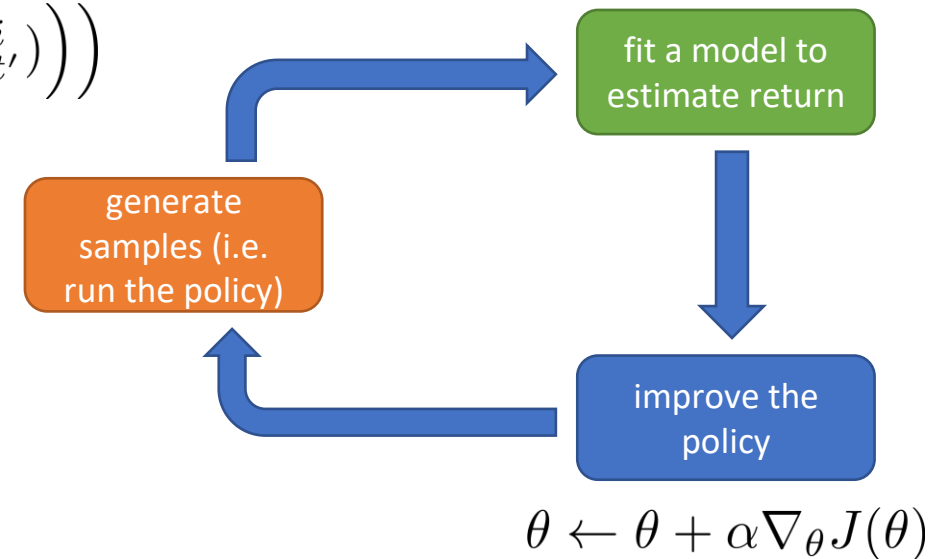
REINFORCE algorithm:

- 
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. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

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

“reward to go”

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



Improving the policy gradient

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

“reward to go”

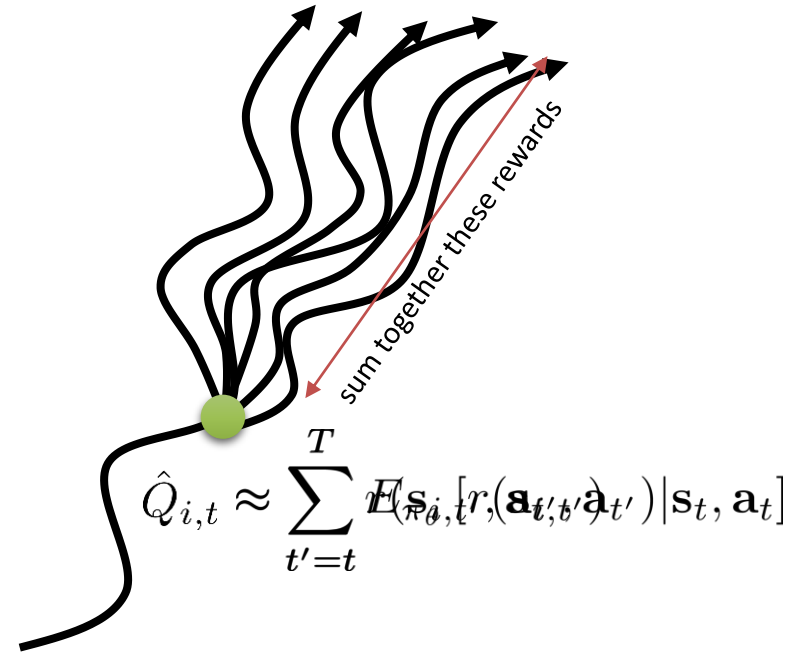
$$\hat{Q}_{i,t}$$

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

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



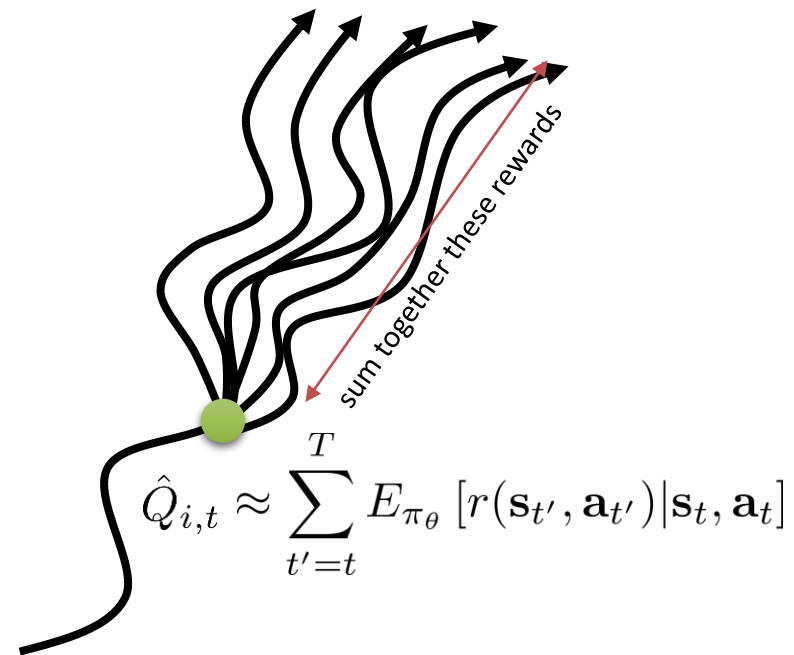
What about the baseline?

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

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

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

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



State & state-action value functions

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \sum_{t=0}^T E_{\pi_\theta} [r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^\pi(\mathbf{s}_{t+1}) | \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}]$$

total reward from taking \mathbf{a} in \mathbf{s}

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

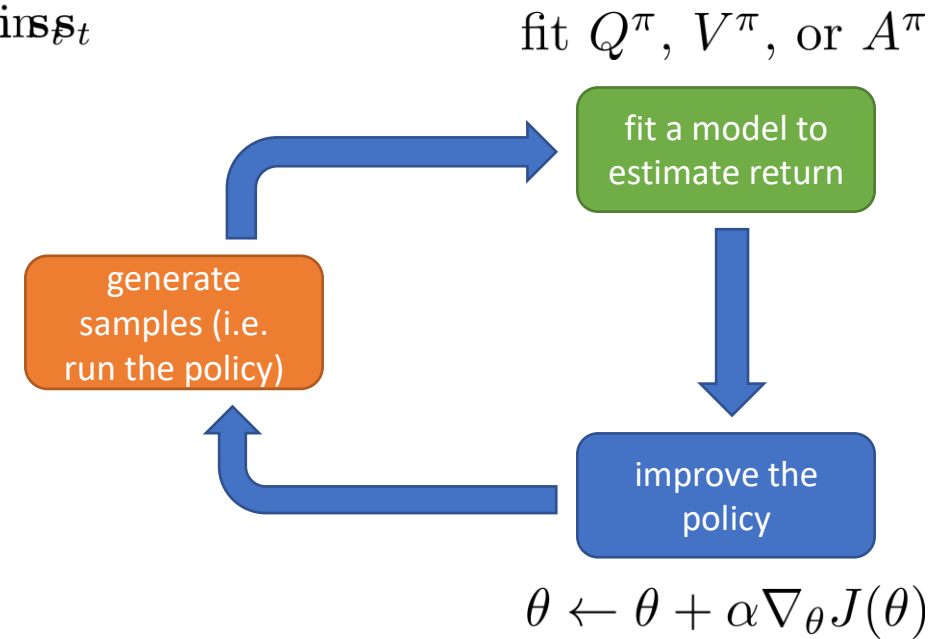
$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t): \text{how much better } \mathbf{a}_t \text{ is}$$

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

the better this estimate, the lower the variance

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

unbiased, but high variance single-sample estimate



Value function fitting

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

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

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

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

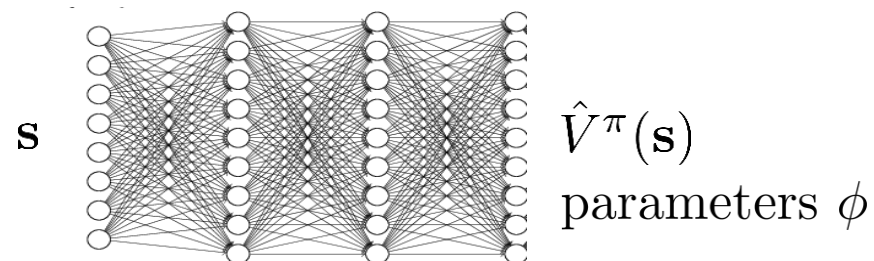
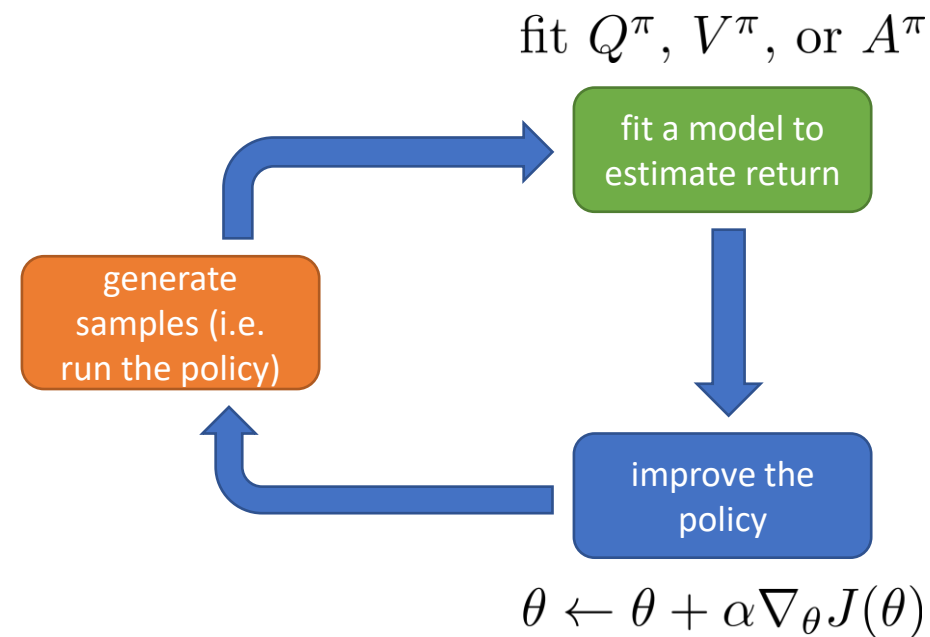
fit *what* to *what*?

Q^π, V^π, A^π ?

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

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

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



Policy evaluation

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

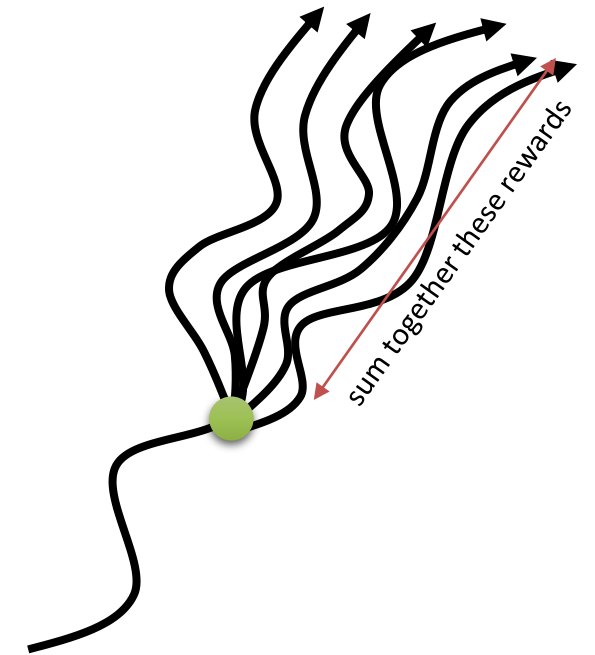
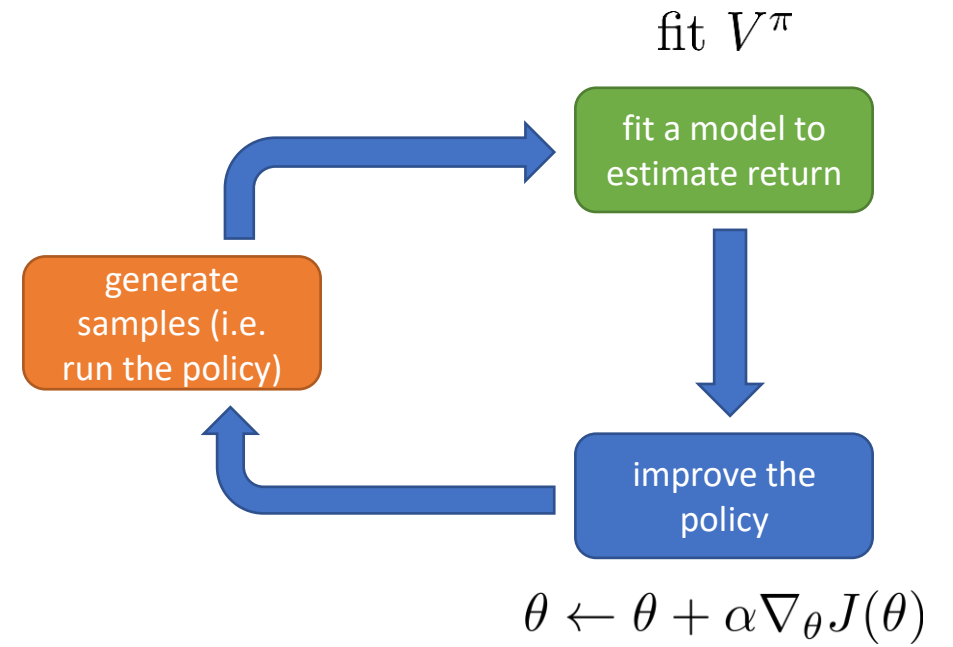
$$J(\theta) = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} [V^\pi(\mathbf{s}_1)]$$

how can we perform policy evaluation?

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

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

$$V^\pi(\mathbf{s}_t) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \quad (\text{requires us to reset the simulator})$$



Monte Carlo evaluation with function approximation

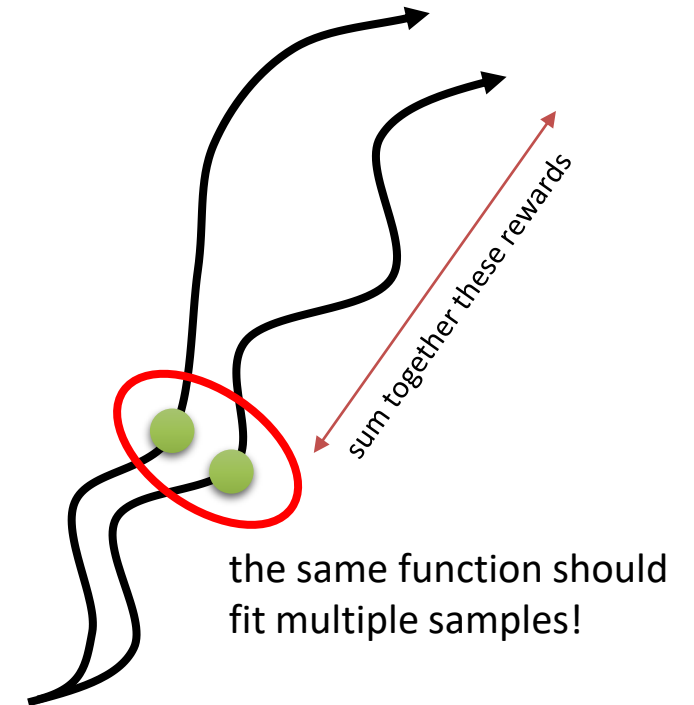
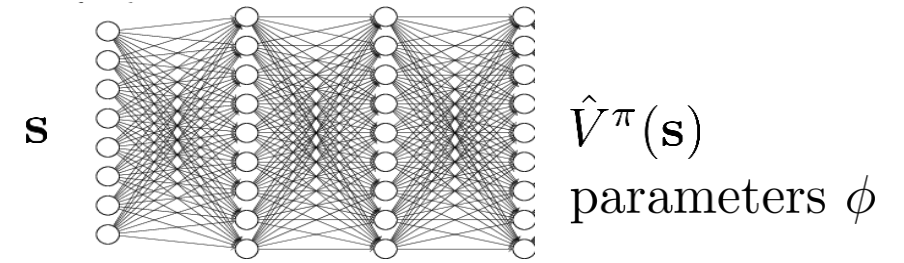
$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

not as good as this: $V^\pi(\mathbf{s}_t) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$

but still pretty good!

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

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



Can we do better?

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

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!

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

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

Policy evaluation examples

TD-Gammon, Gerald Tesauro 1992

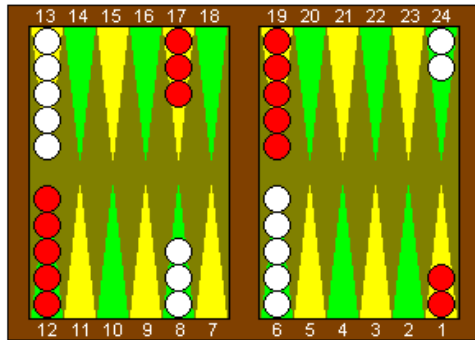


Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference,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

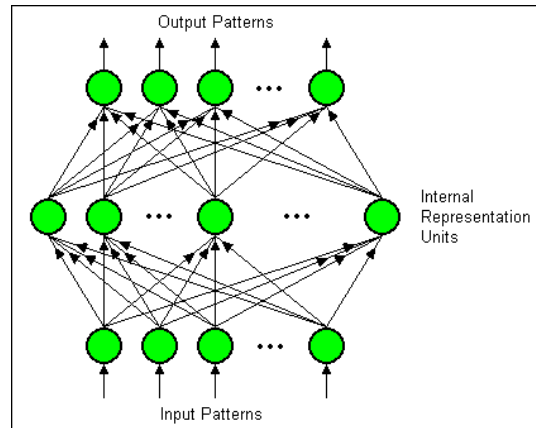


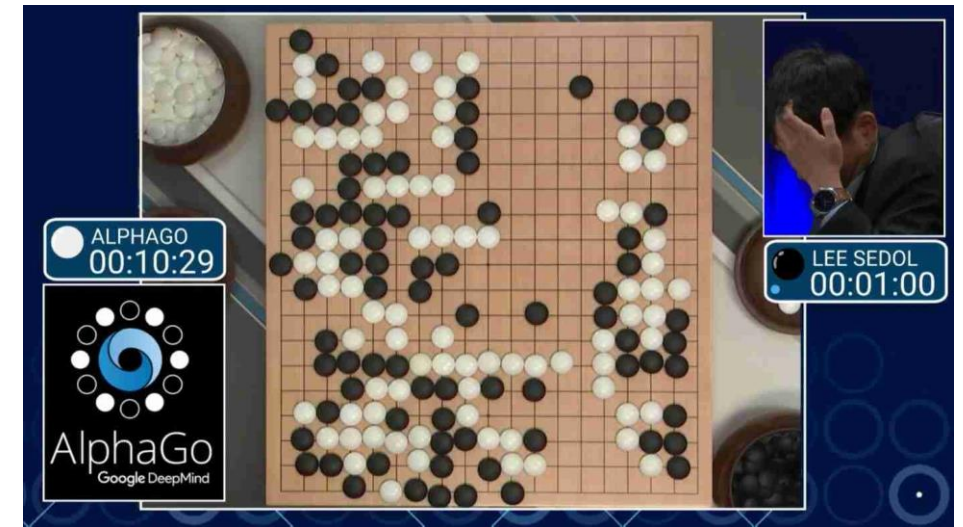
Figure 1. An illustration of the multilayer perceptron architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

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

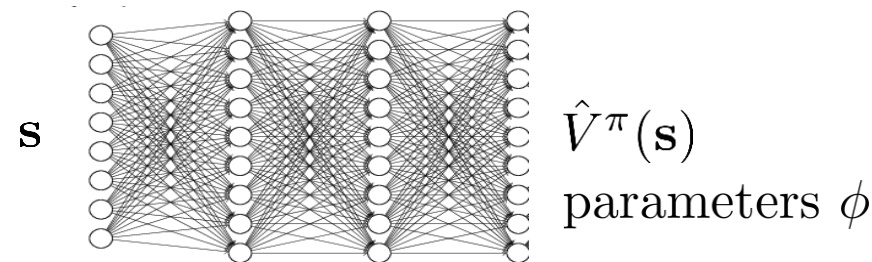
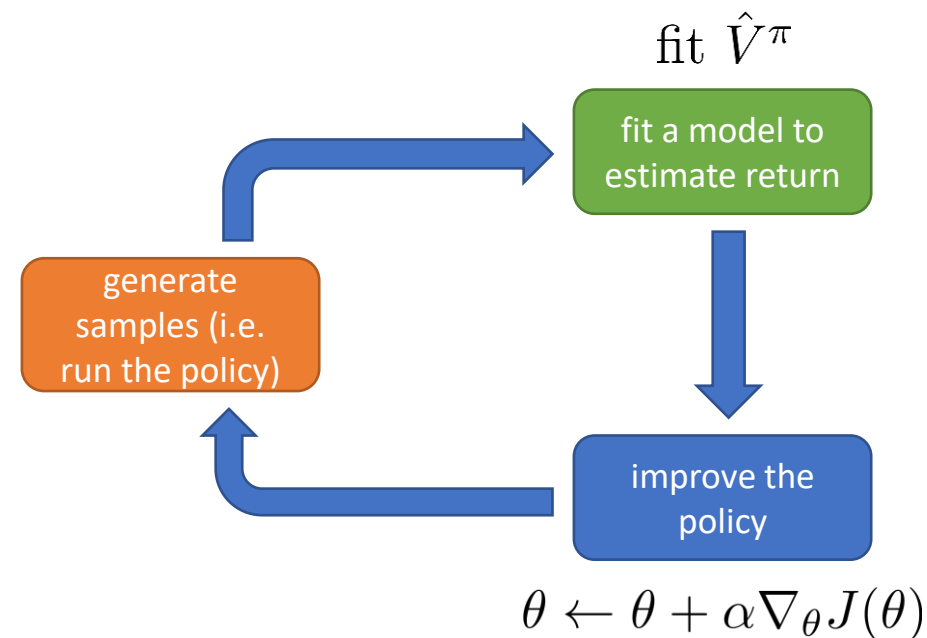
An actor-critic algorithm

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. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

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

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



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

Aside: discount factors

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t+1})$$

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

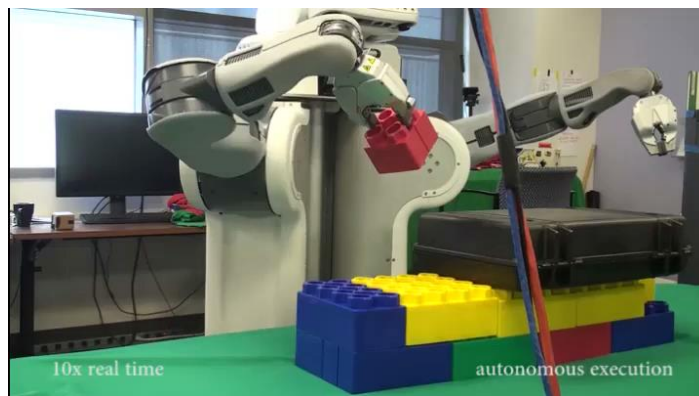
what if T (episode length) is ∞ ?

\hat{V}_{ϕ}^{π} can get infinitely large in many cases

simple trick: better to get rewards sooner than later

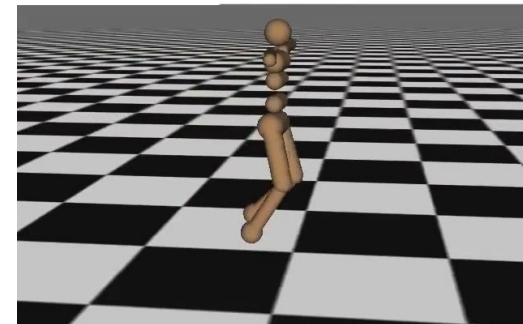
$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t+1})$$

discount factor $\gamma \in [0, 1]$ (0.99 works well)



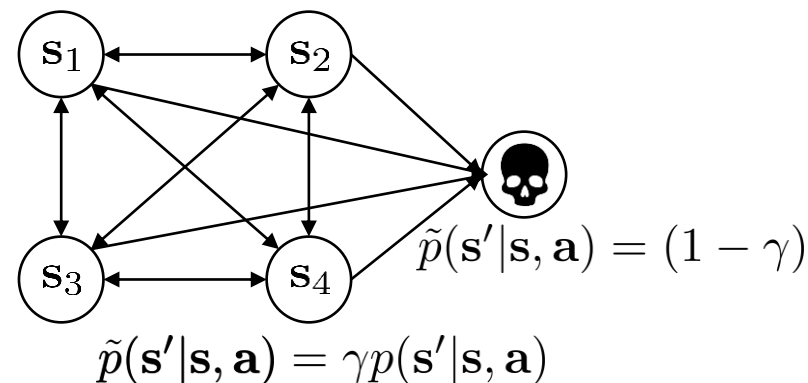
episodic tasks

Iteration 2000



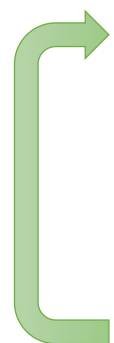
continuous/cyclical tasks

γ changes the MDP:

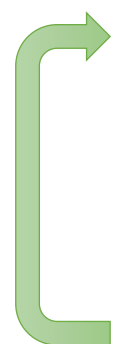


Actor-critic algorithms (with discount)

batch actor-critic algorithm:

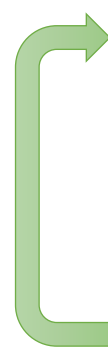
- 
1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
 2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
 3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$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

online actor-critic algorithm:

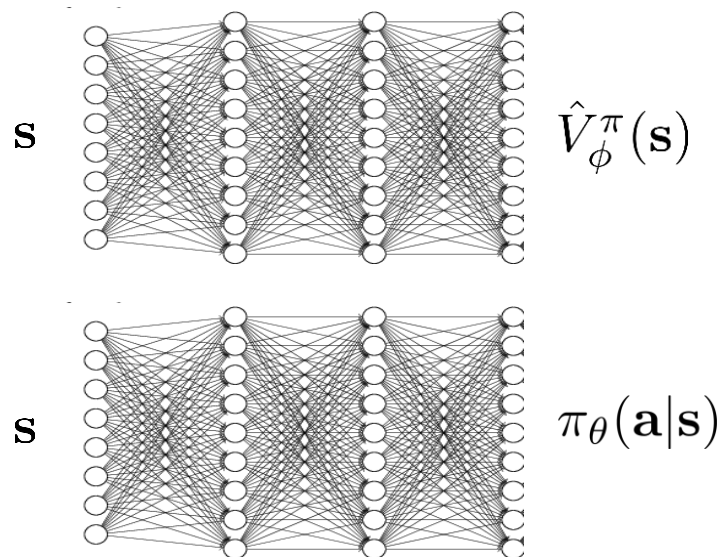
- 
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
 2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$
 3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}')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

Architecture design

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. $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

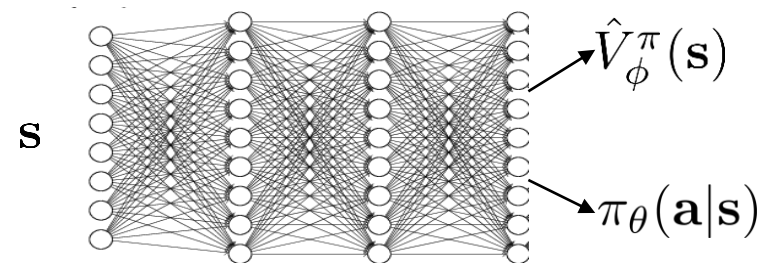
two network design



+ simple & stable

- no shared features between actor & critic

shared network design



Can we use **just** a value function?

Can we omit policy gradient completely?

$A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: how much better is \mathbf{a}_t than the average action according to π

$\arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: best action from \mathbf{s}_t , if we then follow π

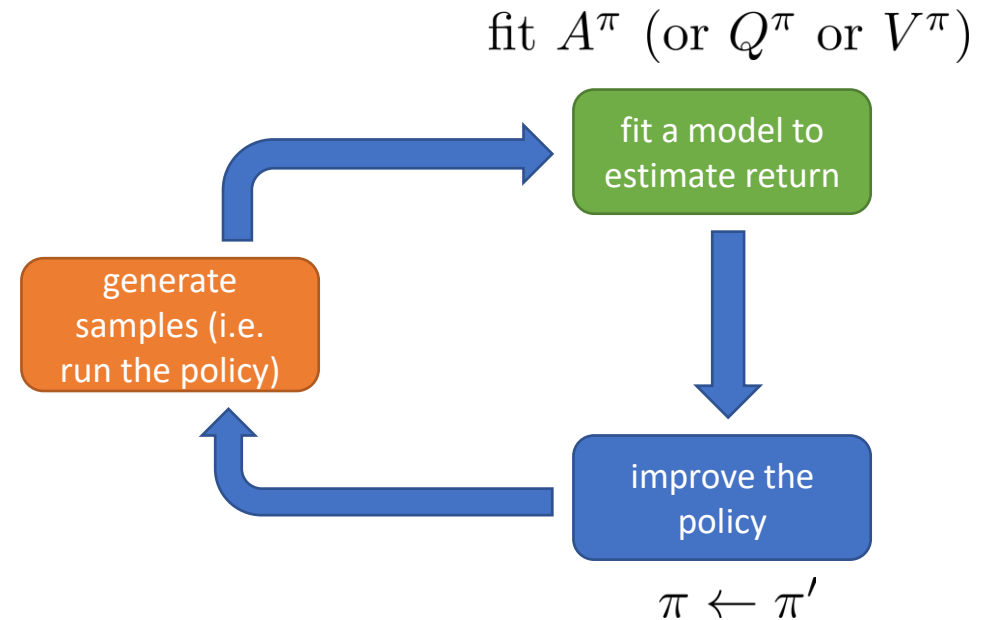
at *least* as good as any $\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)$

regardless of what $\pi(\mathbf{a}_t | \mathbf{s}_t)$ is!

forget policies, let's just do this!

$$\pi'(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

as good as π
(probably better)



Policy iteration

High level idea:

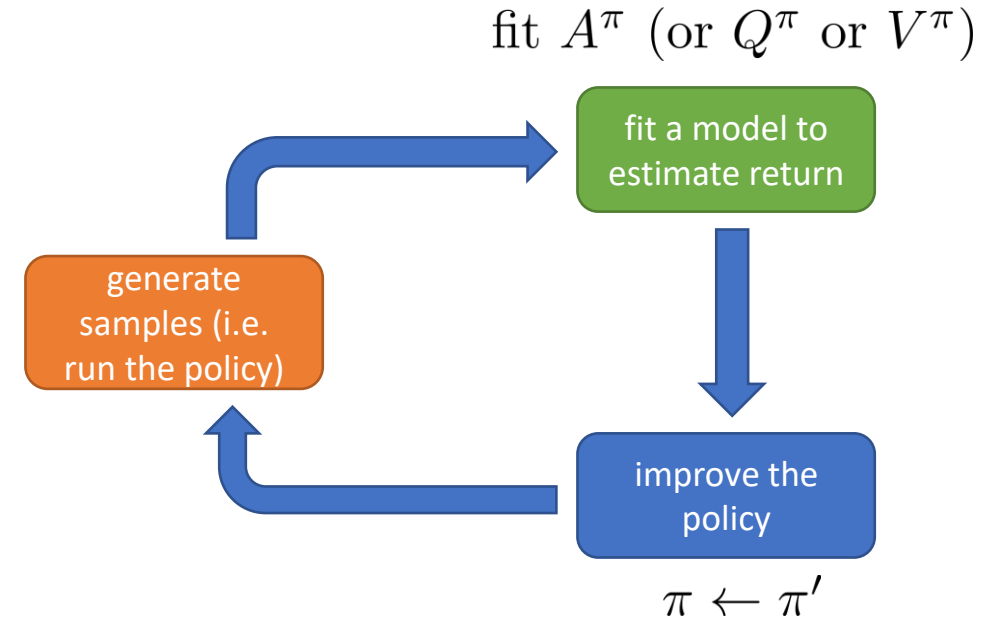
policy iteration algorithm:

- ➡ 1. evaluate $A^\pi(\mathbf{s}, \mathbf{a})$ ← how to do this?
2. set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

as before: $A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] - V^\pi(\mathbf{s})$

let's evaluate $V^\pi(\mathbf{s})$!



Dynamic programming

Let's assume we know $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, and \mathbf{s} and \mathbf{a} are both discrete (and small)

0.2	0.3	0.4	0.3
0.3	0.3	0.5	0.3
0.4	0.4	0.6	0.4
0.5	0.5	0.7	0.5

16 states, 4 actions per state

can store full $V^\pi(\mathbf{s})$ in a table!

\mathcal{T} is $16 \times 16 \times 4$ tensor

bootstrapped update: $V^\pi(\mathbf{s}) \leftarrow E_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} [V^\pi(\mathbf{s}')]]$




just use the current estimate here

$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases} \longrightarrow \text{deterministic policy } \pi(\mathbf{s}) = \mathbf{a}$

simplified: $V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s}))} [V^\pi(\mathbf{s}')]]$


Policy iteration with dynamic programming

policy iteration:

- 
1. evaluate $V^\pi(\mathbf{s})$
 2. set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

policy evaluation:

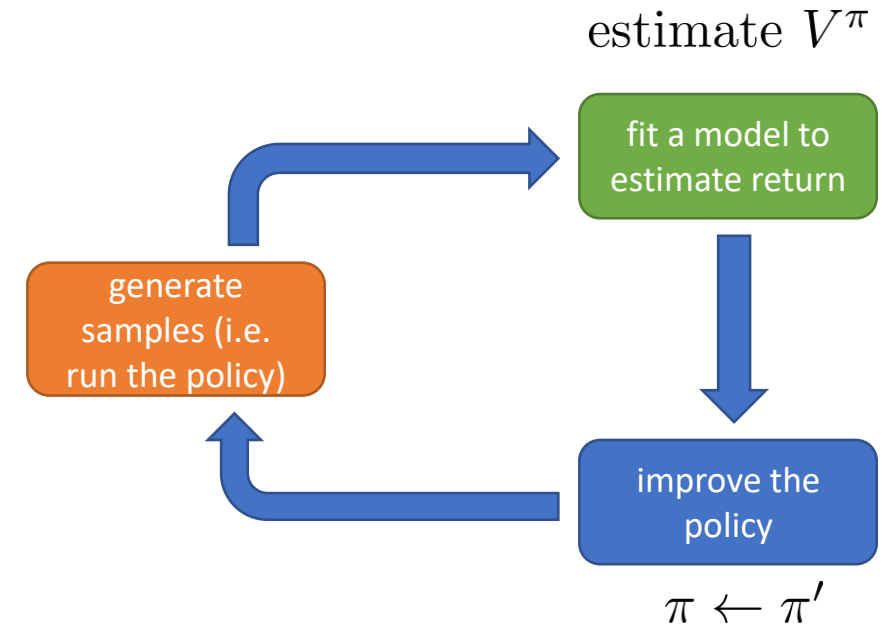

$$V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s}))} [V^\pi(\mathbf{s}')]]$$

0.2	0.3	0.4	0.3
0.3	0.3	0.5	0.3
0.4	0.4	0.6	0.4
0.5	0.5	0.7	0.5

16 states, 4 actions per state

can store full $V^\pi(\mathbf{s})$ in a table!

\mathcal{T} is $16 \times 16 \times 4$ tensor



Even simpler dynamic programming

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

$$A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] - V^\pi(\mathbf{s})$$

$$\arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) = \arg \max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] \quad (\text{a bit simpler})$$

skip the policy and compute values directly!

value iteration algorithm:

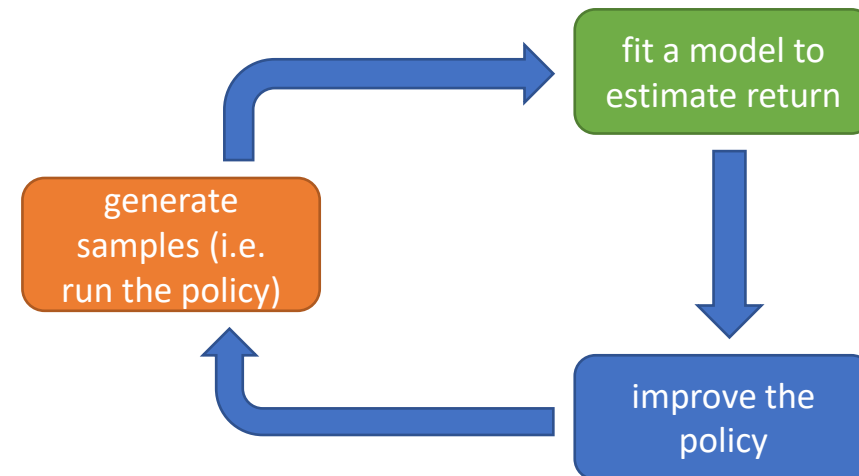
1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]$
2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

	a			
s	$Q(\mathbf{s}, \mathbf{a})$		$Q(\mathbf{s}, \mathbf{a})$	$Q(\mathbf{s}, \mathbf{a})$
	$Q(\mathbf{s}, \mathbf{a})$	$Q(\mathbf{s}, \mathbf{a})$		$Q(\mathbf{s}, \mathbf{a})$
		$Q(\mathbf{s}, \mathbf{a})$	$Q(\mathbf{s}, \mathbf{a})$	$Q(\mathbf{s}, \mathbf{a})$
	$Q(\mathbf{s}, \mathbf{a})$	$Q(\mathbf{s}, \mathbf{a})$	$Q(\mathbf{s}, \mathbf{a})$	
	$Q(\mathbf{s}, \mathbf{a})$		$Q(\mathbf{s}, \mathbf{a})$	$Q(\mathbf{s}, \mathbf{a})$
	$Q(\mathbf{s}, \mathbf{a})$	$Q(\mathbf{s}, \mathbf{a})$		$Q(\mathbf{s}, \mathbf{a})$

$$\arg \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) \rightarrow \text{policy}$$

approximates the new value!

$$Q^\pi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} [V^\pi(\mathbf{s}')]$$



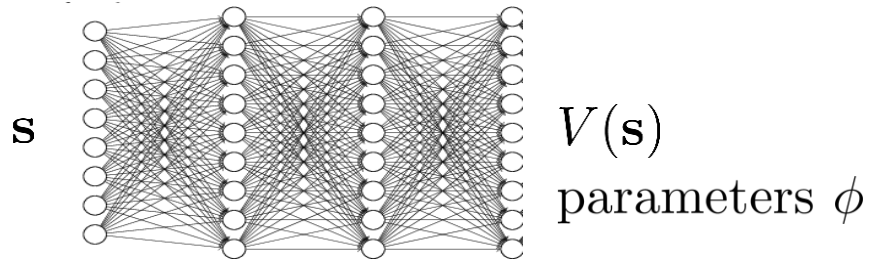
$$V^\pi(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$$

Fitted value iteration

how do we represent $V(\mathbf{s})$?

big table, one entry for each discrete \mathbf{s}

neural net function $V : \mathcal{S} \rightarrow \mathbb{R}$



$$\mathbf{s} = 0 : V(\mathbf{s}) = 0.2$$

$$\mathbf{s} = 1 : V(\mathbf{s}) = 0.3$$

$$\mathbf{s} = 2 : V(\mathbf{s}) = 0.5$$



curse of
dimensionality

$$|\mathcal{S}| = (255^3)^{200 \times 200}$$

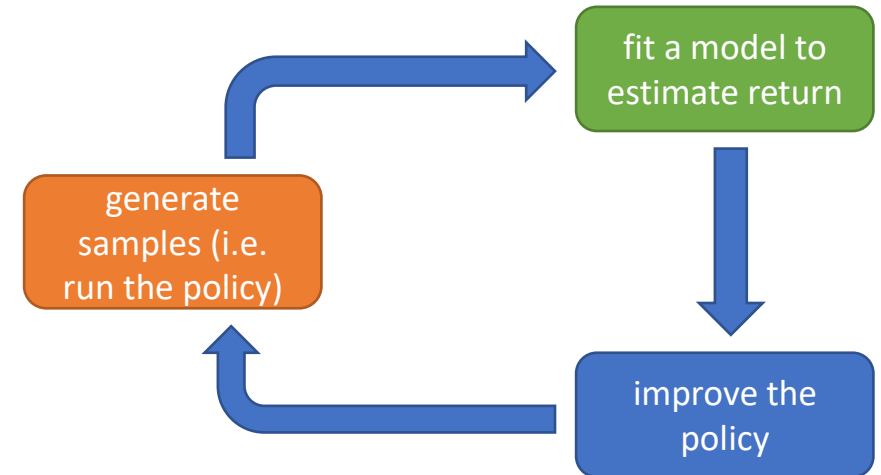
(more than atoms in the universe)

$$Q^\pi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} [V^\pi(\mathbf{s}')]]$$

$$\mathcal{L}(\phi) = \frac{1}{2} \left\| V_\phi(\mathbf{s}) - \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a}) \right\|^2$$

fitted value iteration algorithm:


- 1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)])$
- 2. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$



$$V^\pi(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$$


What if we don't know the transition dynamics?

fitted value iteration algorithm:

- 
1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)])$
 2. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$
- need to know outcomes for different actions!


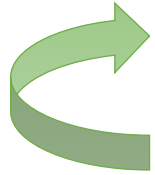
Back to policy iteration...

policy iteration:

- 
1. evaluate $Q^\pi(\mathbf{s}, \mathbf{a})$
 2. set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

policy evaluation:


$$V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}' | \mathbf{s}, \pi(\mathbf{s}))} [V^\pi(\mathbf{s}')]$$

$$Q^\pi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a})} [Q^\pi(\mathbf{s}', \pi(\mathbf{s}'))]$$

can fit this using samples



Can we do the “max” trick again?

policy iteration:



1. evaluate $V^\pi(\mathbf{s})$
2. set $\pi \leftarrow \pi'$



fitted value iteration algorithm:



1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)])$
2. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$

forget policy, compute value directly

can we do this with Q-values **also**, without knowing the transitions?

fitted Q iteration algorithm:



1. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)]$
2. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

approximate $E[V(\mathbf{s}'_i)] \approx \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$

doesn't require simulation of actions!

+ works even for off-policy samples (unlike actor-critic)

+ only one network, no high-variance policy gradient

- no convergence guarantees for non-linear function approximation (more on this later)

Fitted Q-iteration

full fitted Q-iteration algorithm:

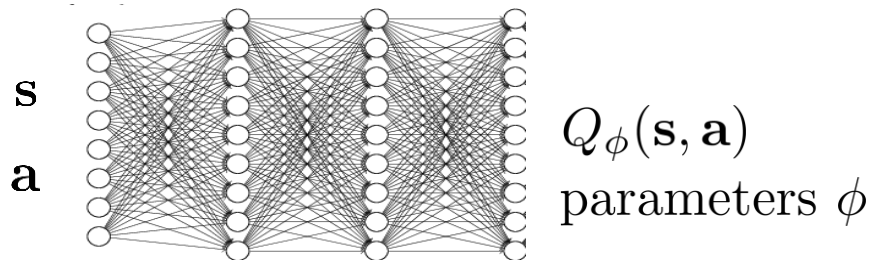
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$
- $K \times$

parameters

dataset size N , collection policy

iterations K

gradient steps S



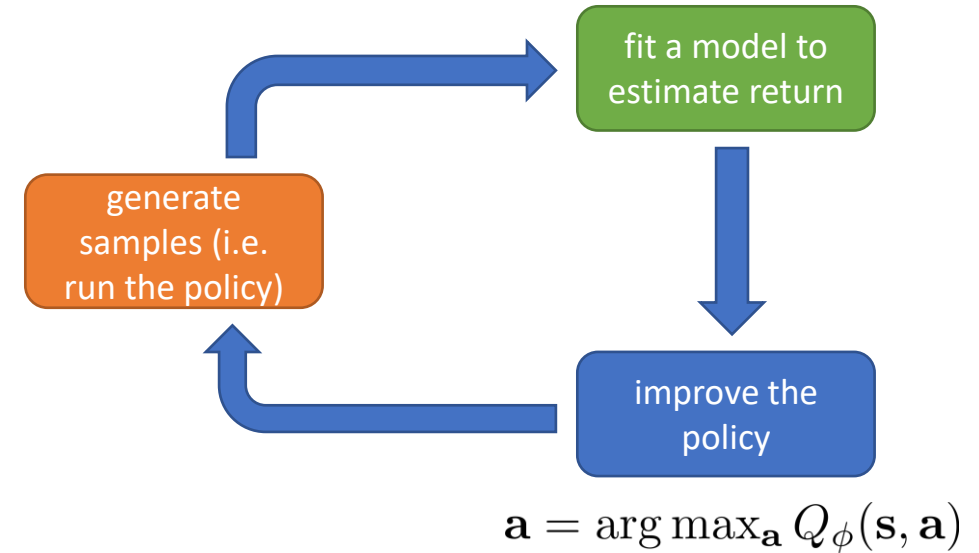
Q-Learning

Online Q-learning algorithms

full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

$$Q_\phi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$$

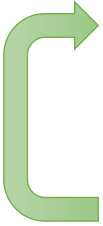


online Q iteration algorithm: off policy, so many choices here!

1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

Exploration with Q-learning

online Q iteration algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 - \epsilon & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ \epsilon / (|\mathcal{A}| - 1) & \text{otherwise} \end{cases}$$

$$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t))$$

final policy:

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

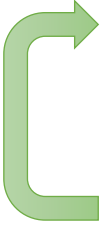
why is this a bad idea for step 1?

“epsilon-greedy”

“Boltzmann exploration”

What's wrong?

online Q iteration algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
- these are correlated!
- isn't this just gradient descent? that converges, right?

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$$

no gradient through target value

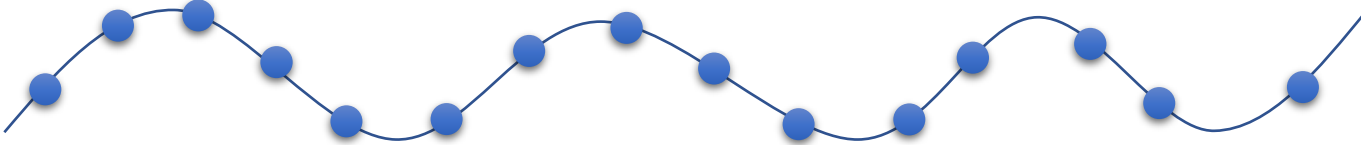
Correlated samples in online Q-learning

online Q iteration algorithm:

1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}')])$

- sequential states are strongly correlated


- target value is always changing






Replay buffers

online Q iteration algorithm:

special case with $K = 1$, and one gradient step

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}')])$

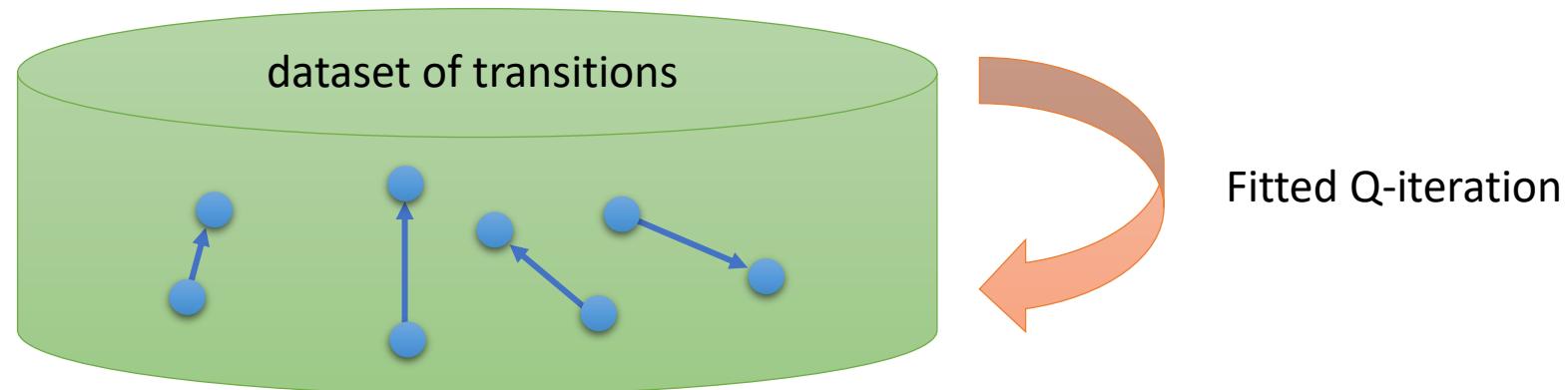
full fitted Q-iteration algorithm:

- 
- ~~1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy~~
 -  2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}')$
 - $K \times$  3. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

any policy will work! (with broad support)

just load data from a buffer here

still use one gradient step



Replay buffers

Q-learning with a replay buffer:

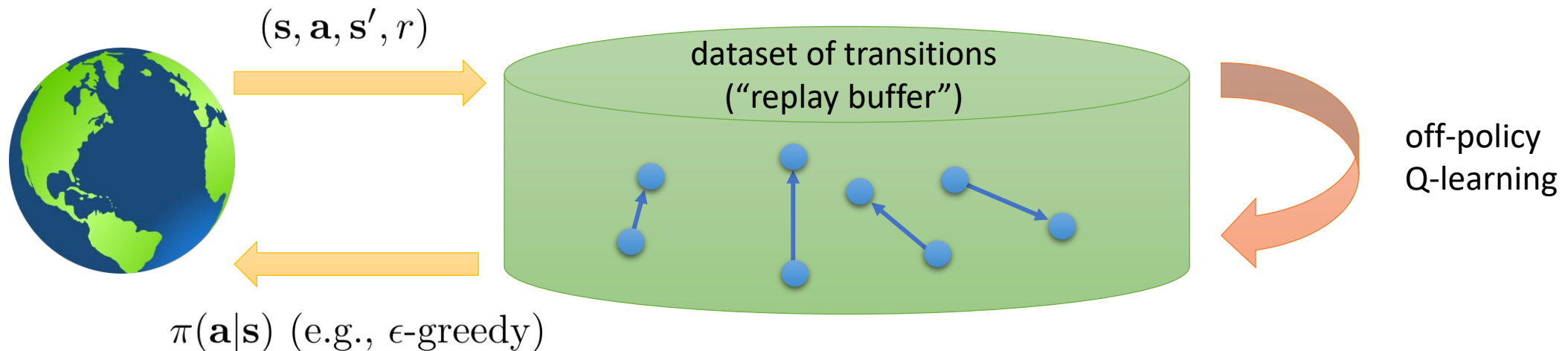
1. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
2. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i) (Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

+ samples are no longer correlated

+ multiple samples in the batch (low-variance gradient)

but where does the data come from?

need to periodically feed the replay buffer...

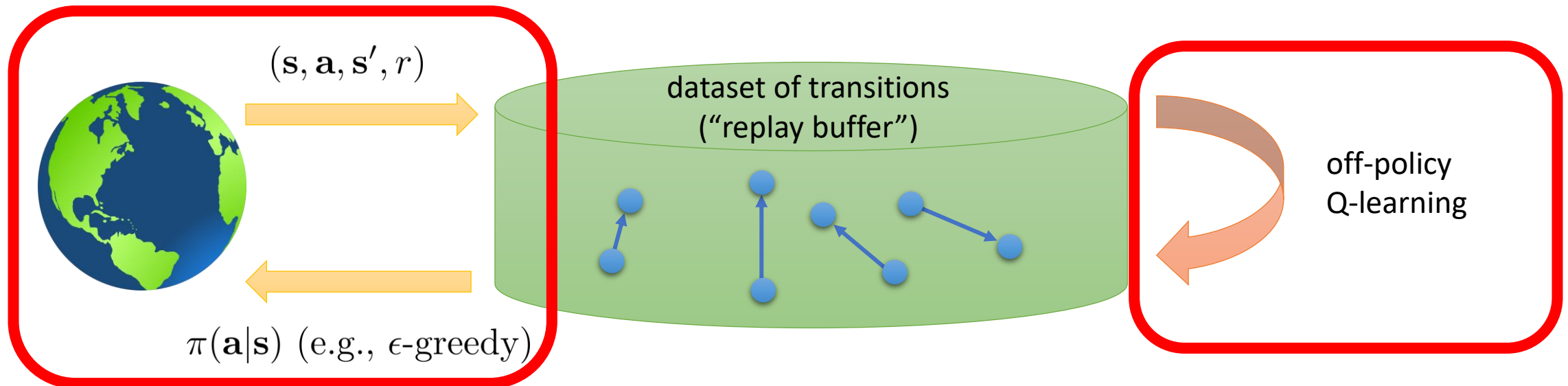


Putting it together

full Q-learning with replay buffer:

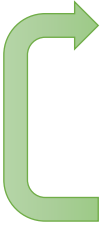
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

**K = 1 is common, though
larger K more efficient**



What's wrong?

online Q iteration algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
- ~~these are correlated!~~
use replay buffer

Q-learning is *not* gradient descent!

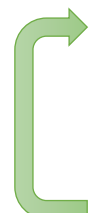
$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)))$$

no gradient through target value

This is still a problem!


Q-Learning and Regression

full Q-learning with replay buffer:

- 
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i) (Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

one gradient step, moving target

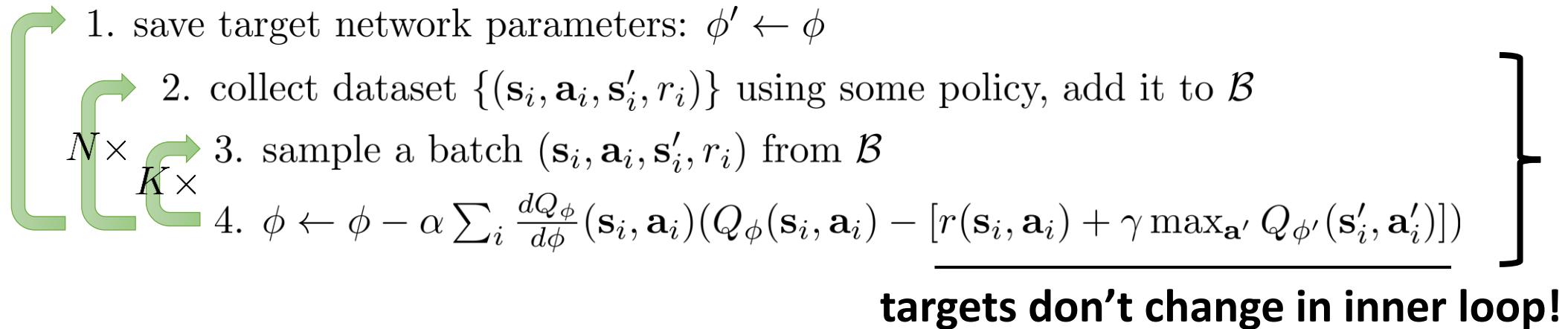
full fitted Q-iteration algorithm:

- 
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

perfectly well-defined, stable regression

Q-Learning with target networks

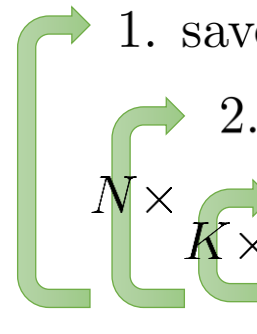
Q-learning with replay buffer and target network:

- 
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i) (Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \underbrace{[r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)]}_{\text{targets don't change in inner loop!}})$

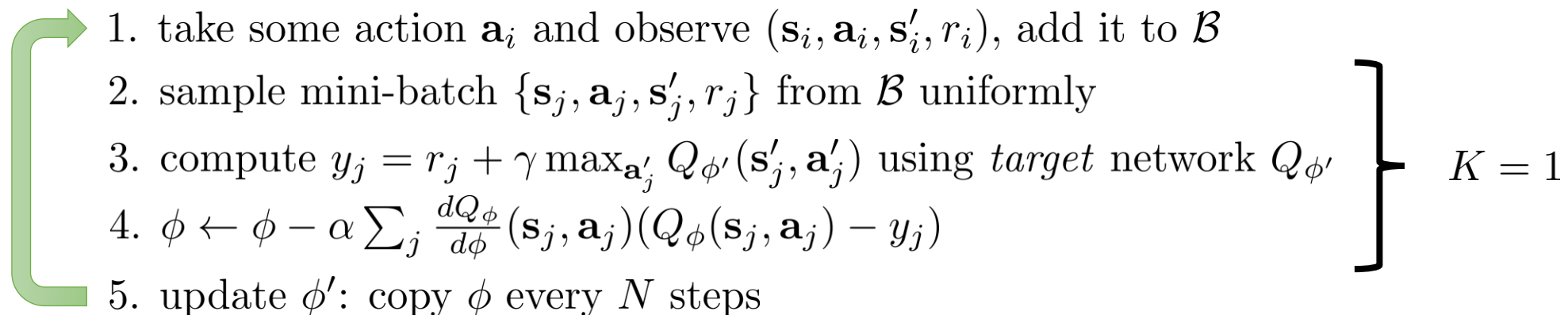
supervised regression

“Classic” deep Q-learning algorithm (DQN)

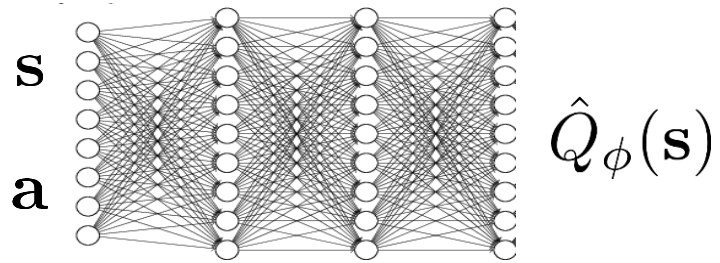
Q-learning with replay buffer and target network:

- 
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 - $N \times$ 3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 - $K \times$ 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

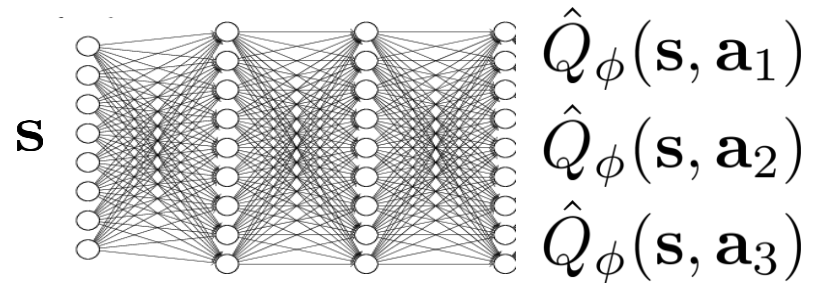
“classic” deep Q-learning algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{B}
 2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{B} uniformly
 3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
 4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
 5. update ϕ' : copy ϕ every N steps

Representing the Q-function



more common with continuous actions

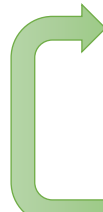


more common with discrete actions

Back to actor-critic

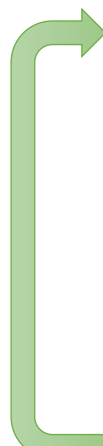
online Q iteration algorithm:

off policy, so many choices here!

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

with **continuous actions**, this is very inconvenient (but not impossible)

Idea: use actor-critic, but with Q-functions (to train off-policy)

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{B}
 2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{B} uniformly
 3. compute $y_j = r_j + \gamma E_{\mathbf{a}'_j \sim \pi_{\theta'}(\mathbf{a}'_j | \mathbf{s}'_j)}[Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)]$ using *target* ϕ' and θ'
 4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
 5. $\theta \leftarrow \theta + \beta \sum_j \nabla_{\theta} E_{\mathbf{a} \sim \pi_{\theta}(\mathbf{a} | \mathbf{s}_j)}[Q_\phi(\mathbf{s}_j, \mathbf{a})]$ \longleftarrow policy gradient
 6. update ϕ' and θ' every N steps

Simple practical tips for Q-learning

- Q-learning takes some care to stabilize
 - Test on easy, reliable tasks first, make sure your implementation is correct

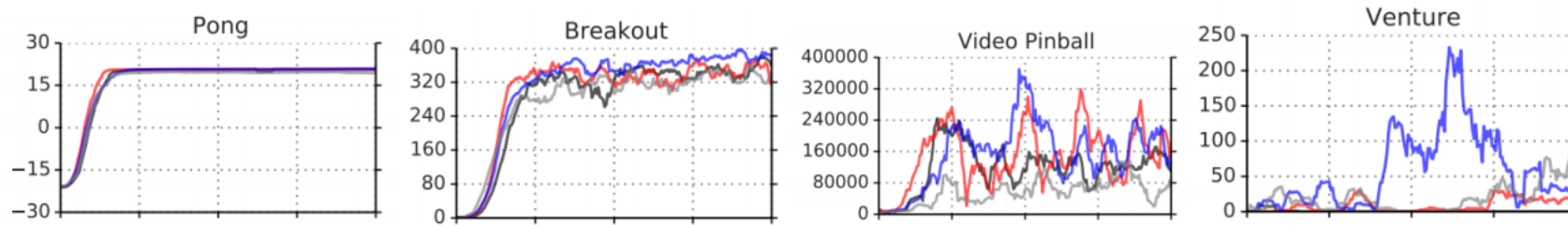


Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. “Prioritized experience replay”. *arXiv preprint arXiv:1511.05952* (2015), Figure 7

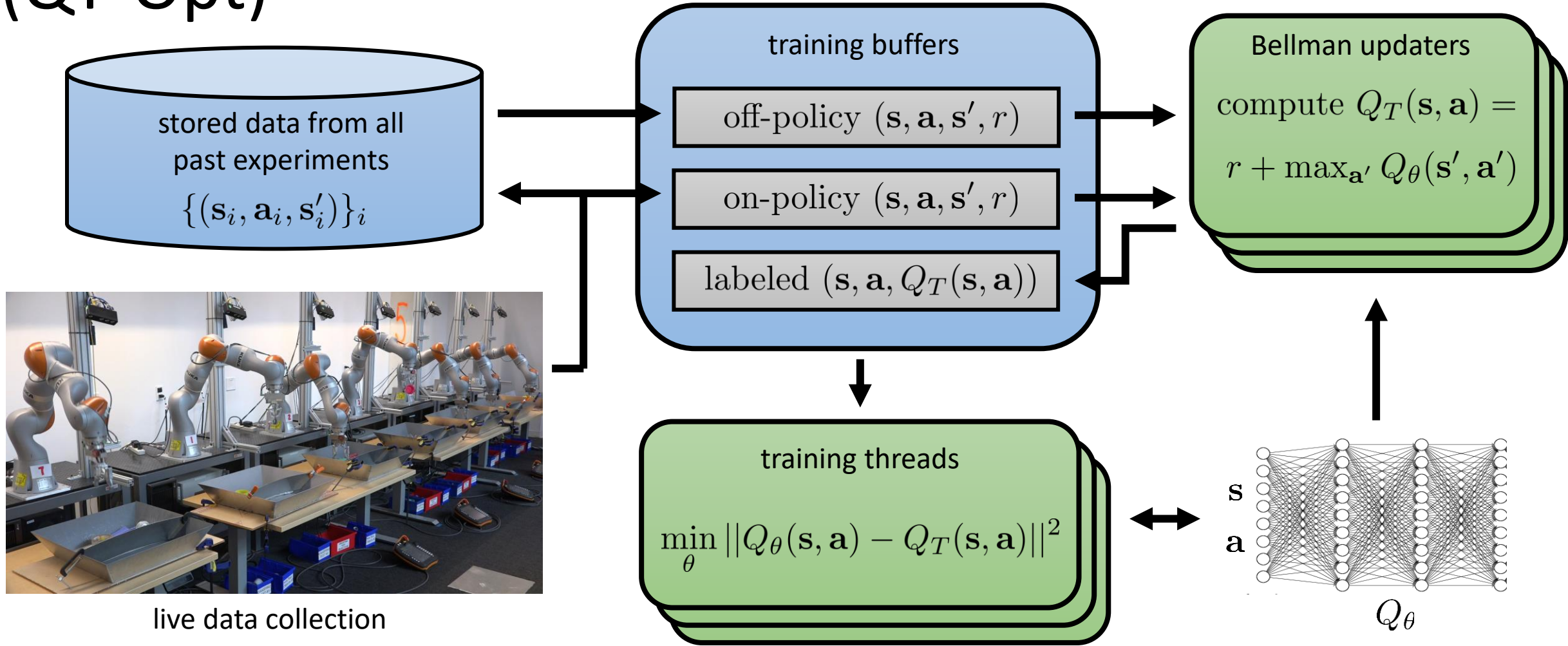
- Large replay buffers help improve stability
 - Looks more like fitted Q-iteration
- It takes time, be patient – might be no better than random for a while
- Start with high exploration (epsilon) and gradually reduce

Q-learning with convolutional networks

- “Human-level control through deep reinforcement learning,” Mnih et al. ‘13
- Q-learning with convolutional networks
- Uses replay buffer and target network
- One-step backup
- One gradient step
- Can be improved a lot with double Q-learning (and other tricks)



Large-scale Q-learning with continuous actions (QT-Opt)



live data collection

$$\text{minimize } \sum_i (Q(s_i, a_i) - [r(s_i, a_i) + \max_{a'} Q(s'_i, a'_i)])^2$$

Q-learning suggested readings

- Classic papers
 - Watkins. (1989). Learning from delayed rewards: introduces Q-learning
 - Riedmiller. (2005). Neural fitted Q-iteration: batch-mode Q-learning with neural networks
- Deep reinforcement learning Q-learning papers
 - Lange, Riedmiller. (2010). Deep auto-encoder neural networks in reinforcement learning: early image-based Q-learning method using autoencoders to construct embeddings
 - Mnih et al. (2013). Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari.
 - Van Hasselt, Guez, Silver. (2015). Deep reinforcement learning with double Q-learning: a very effective trick to improve performance of deep Q-learning.
 - Lillicrap et al. (2016). Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.
 - Gu, Lillicrap, Stuskever, L. (2016). Continuous deep Q-learning with model-based acceleration: continuous Q-learning with action-quadratic value functions.
 - Wang, Schaul, Hessel, van Hasselt, Lanctot, de Freitas (2016). Dueling network architectures for deep reinforcement learning: separates value and advantage estimation in Q-function.

UC Berkeley · CSW182 | [Deep Learning]

Designing, Visualizing and Understanding Deep Neural Networks (2021)

CSW182 (2021) · 课程资料包 @ShowMeAI



视频
中英双语字幕



课件
一键打包下载



笔记
官方笔记翻译



代码
作业项目解析



视频 · B 站 [扫码或点击链接]
<https://www.bilibili.com/video/BV1Ff4y1n7ar>



课件 & 代码 · 博客 [扫码或点击链接]
<http://blog.showmeai.tech/berkeley-csw182>

Berkeley
Q-Learning
计算机视觉

循环神经网络
风格迁移
机器学习基础

可视化
模仿学习
生成模型

梯度策略
元学习
卷积网络

Awesome AI Courses Notes Cheatsheets 是 [ShowMeAI](#) 资料库的分支系列，覆盖最具知名度的 **TOP50+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称，跳转至课程**资料包**页面，**一键下载**课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets · 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094



微信公众号

资料下载方式 2：扫码点击**底部菜单栏**
称为 **AI 内容创作者**？回复 [添砖加瓦]