Robot Perception and Learning

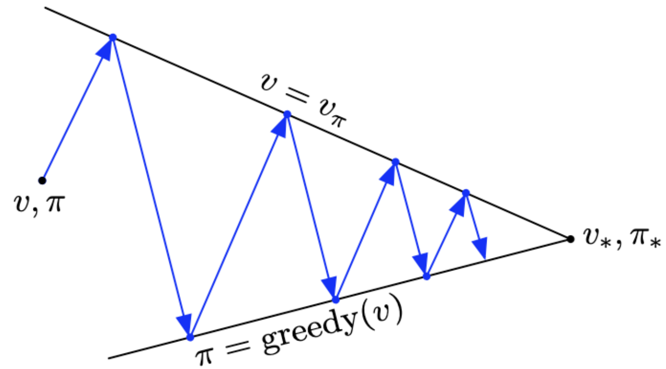
Value Function Approximation, Policy Gradient, Actor Critic

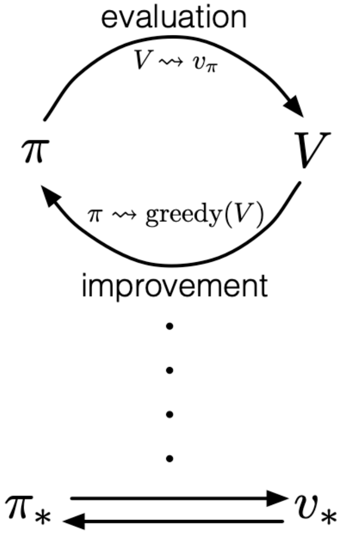
Tsung-Wei Ke

Fall 2025

Summary

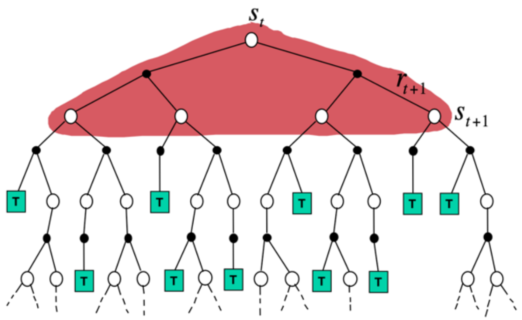
Generalized Policy Iteration



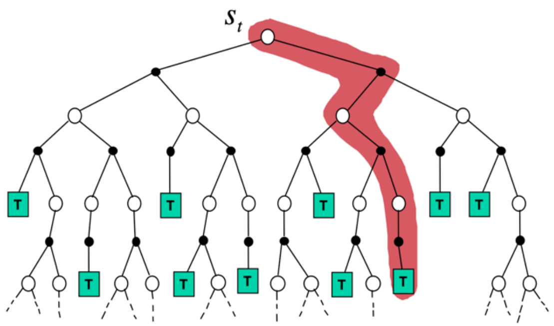


DP vs. MC vs. TD

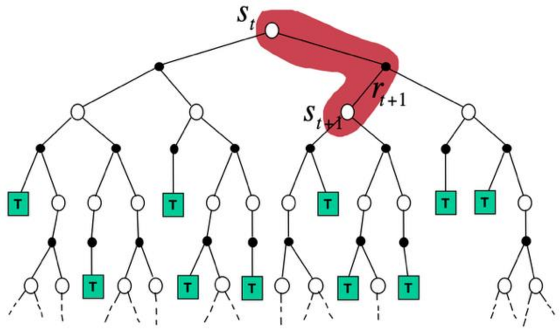
Dynamic Programming Backup



MC Backup



TD Backup



	Bootstrap	Sample
DP	✓	×
MC	×	✓
TD	✓	✓

- DP: $V(S_t) \leftarrow \sum_{A_t} \pi(A_t|S_t) \sum_{S_{t+1},R_{t+1}} p(S_{t+1}, R_{t+1}|S_t, A_t)[R_{t+1} + \gamma V(S_{t+1})]$
- MC: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$
- TD: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

- **On-policy learning**: learn value and execute with the same policy
- **Off-policy learning**: learn and execute with different policies

Importance Sampling

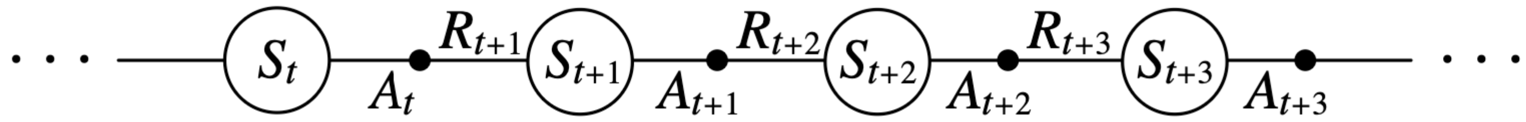
$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s)$$

(Recap) Sarsa: On-policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

- We can learn an action-value function in a similar manner as a state-value function. Instead of considering transitions from state to state, we now consider transitions from state-action pair to state-action pair



(Recap) Sarsa: On-policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Q-learning: Off-policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q $\boxed{\text{(e.g., } \varepsilon\text{-greedy)}}$
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \boxed{\max_a Q(S', a)} - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

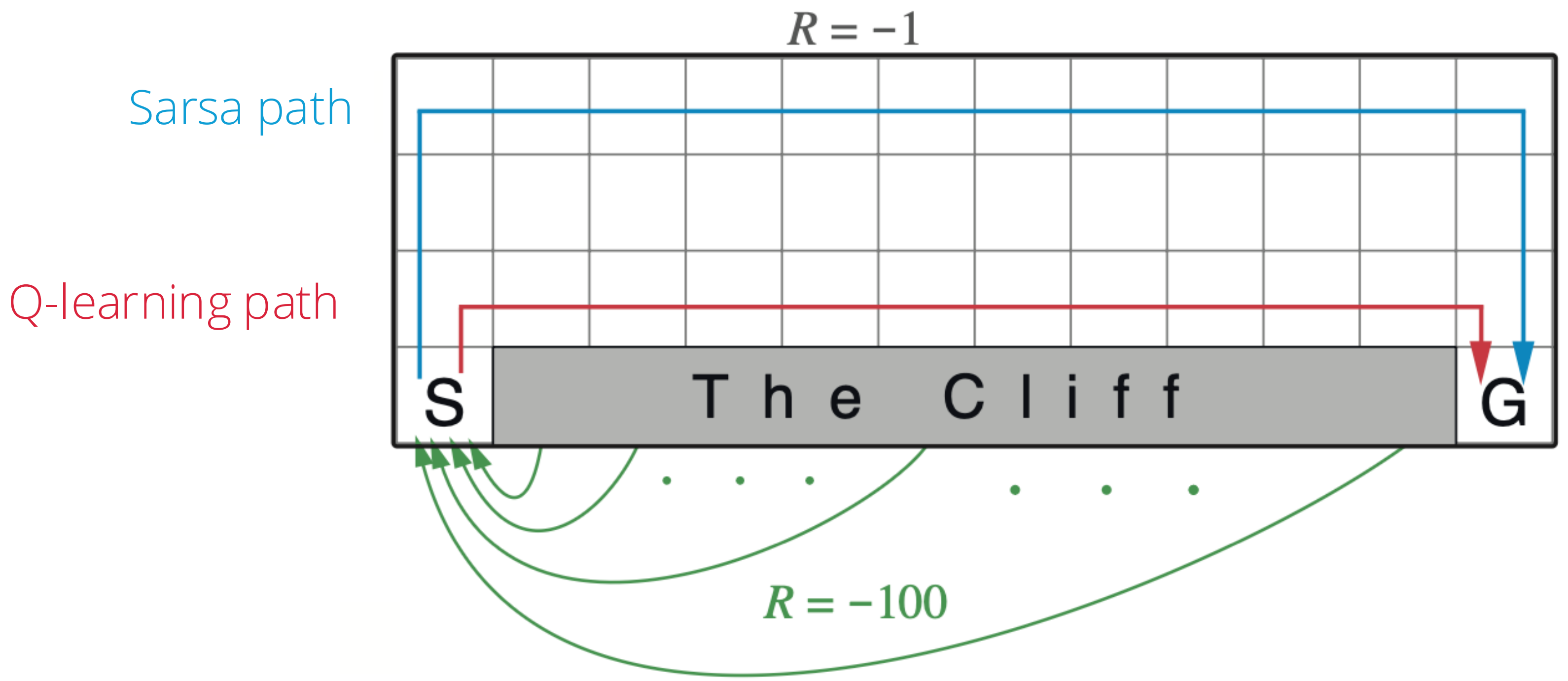
behavior policy

target policy

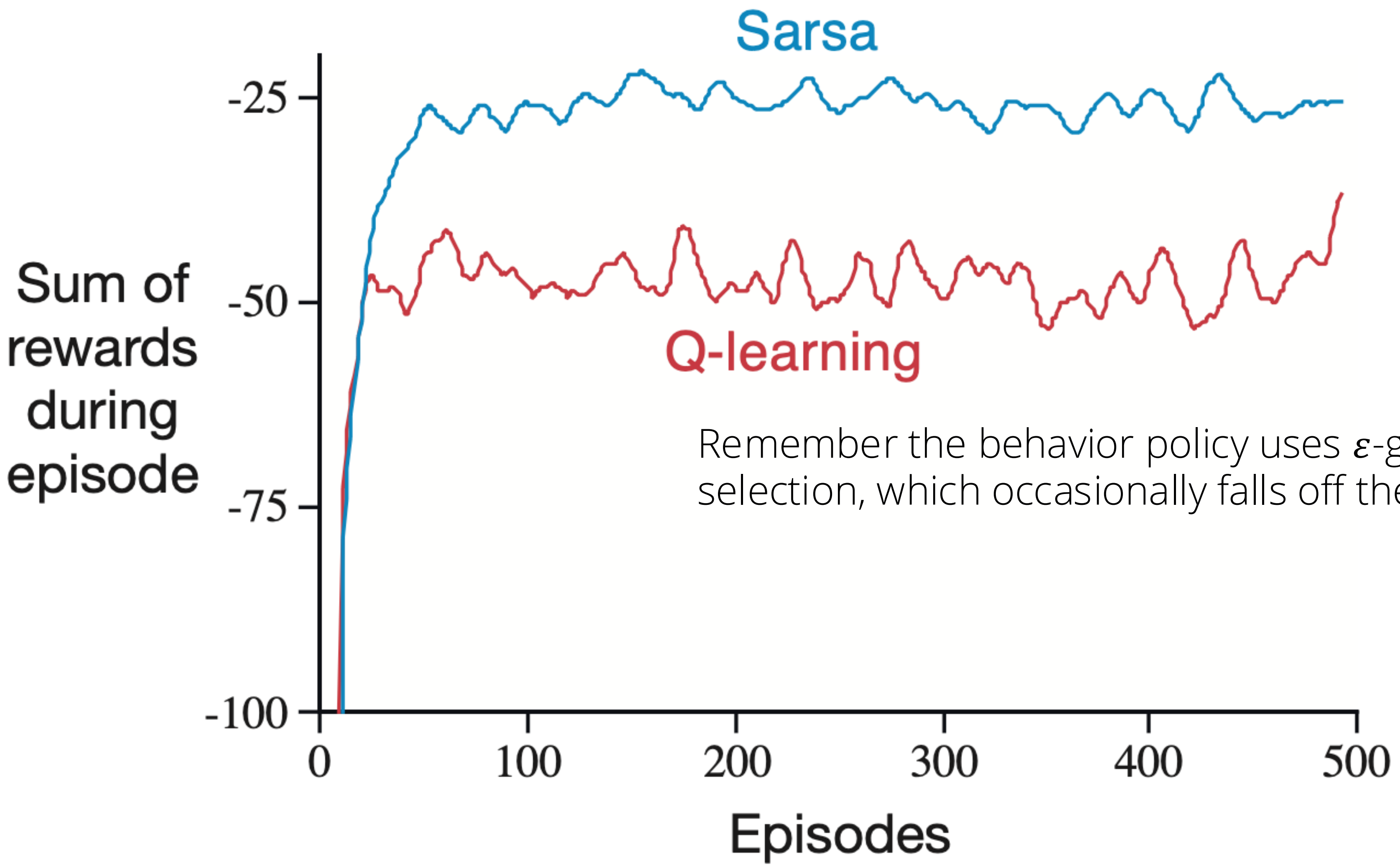
- The learned action-value function approximates q^*
- If all state-action pairs continue to be updated, Q has been shown to converge with probability 1 to q^*

Cliff Walking Example

- The behavior policy uses ε-greedy action selection, with $\varepsilon = 0.1$
- Action: up, down, left and right
- Reward is -100 at the Cliff region, otherwise, reward is -1



Cliff Walking Example



Remember the behavior policy uses ε-greedy action selection, which occasionally falls off the cliff!

Maximization Bias and Double Q-Learning

- The estimated values $Q(s,a)$ are often uncertain and distributed some above and some below zero. The maximum of estimated values induces a positive bias.

- Let say the true values of state s and many actions a are all zero, but estimated values $Q(s,a)$ has positive bias

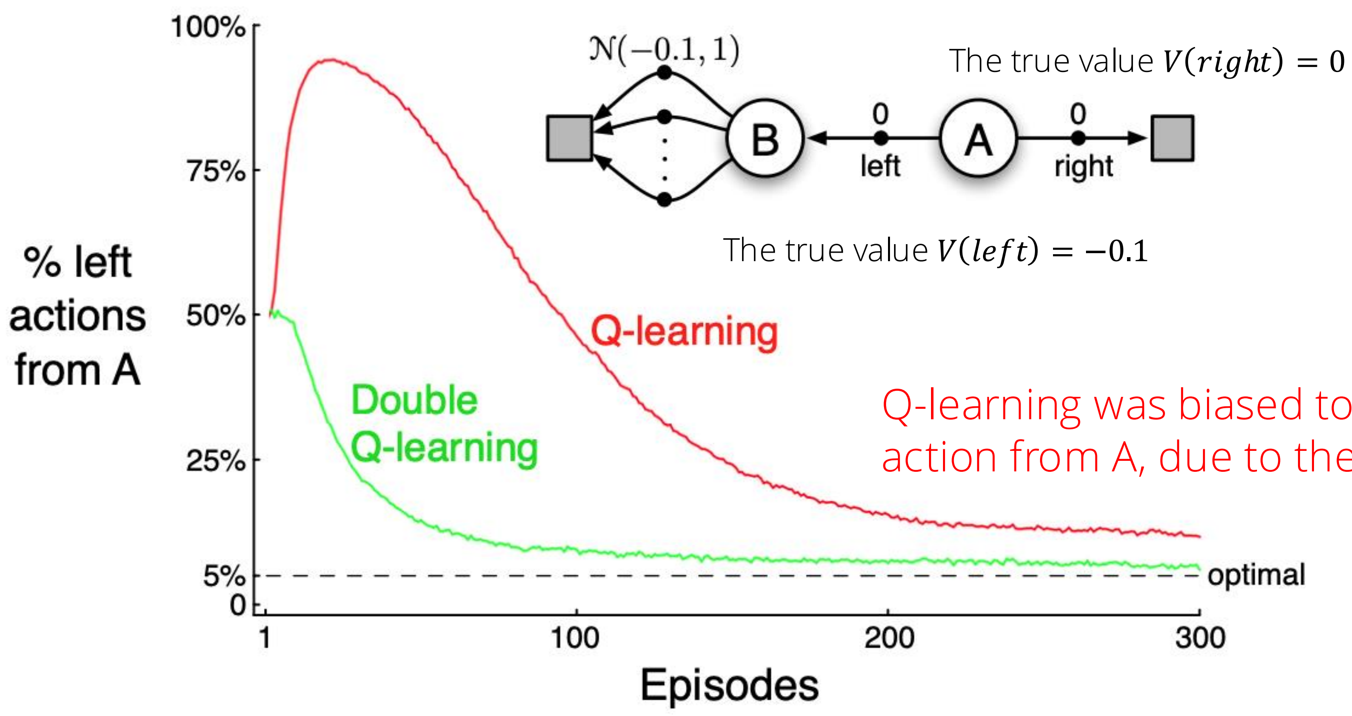
positive bias is introduced by the "maximum" operator

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \boxed{\max_a Q(S_{t+1}, a)} - Q(S_t, A_t) \Big]$$

- This is because we use the same samples to determine the maximizing action and to estimate is values!

Maximization Bias Example

- Action: left and right
- Reward is 0 when transitioning from A to B; reward is drawn from $\mathcal{N}(-0.1,1)$ when transitioning from B to left.
- Taking "left" action from A should always be worse than "right" action



Q-learning was biased toward "left" action from A, due to the positive bias!

Double Q-Learning

- The estimated values $Q(s,a)$ are often uncertain and distributed some above and some below zero. The maximum of estimated values induces a positive bias.

- This is because we use the same samples to determine the maximizing action and to estimate is values!

- Solution: use two sets of samples to learn two independent estimates Q_1 and Q_2
  - Q_1 determines the maximizing action:

$$A^* = \underset{a}{argmax}\, Q_1(s,a)$$

  - Q_2 provides the estimate of its value:

$$Q_2(s,A^*) = Q_2(s, \underset{a}{argmax}\, Q_1(s,a))$$

Double Q-Learning

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big].$$

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Quick Recap: Temporal-Difference Learning

- Temporal-Difference (TD) methods: combine Monte Carlo methods with Dynamic Programming methods that wait only until the next time step and bootstrap value functions from existing estimates

$$V(S_t) \leftarrow V(S_t) + \alpha[\boxed{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)]$$

We call this formulation 1-step TD
We can also have n-step TD

N-step TD Prediction

- n-step TD:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t)]$$

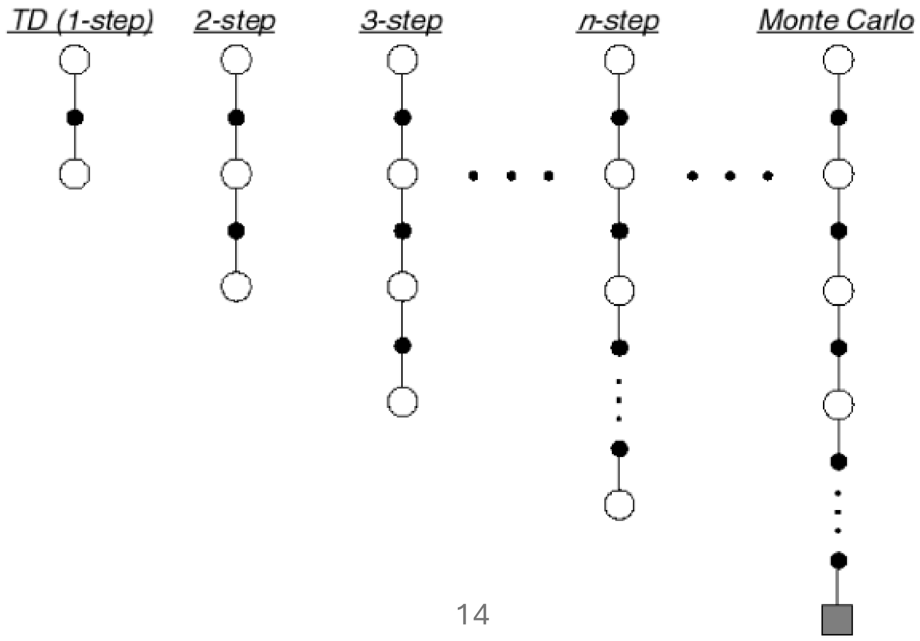
N-step TD Prediction

- n-step TD:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t)]$$

- When $n \to \infty$, n-step TD becomes an MC method:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T - V(S_t)]$$



N-step TD Prediction

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

No bootstrapping until time step $t + n$

N-step TD Prediction

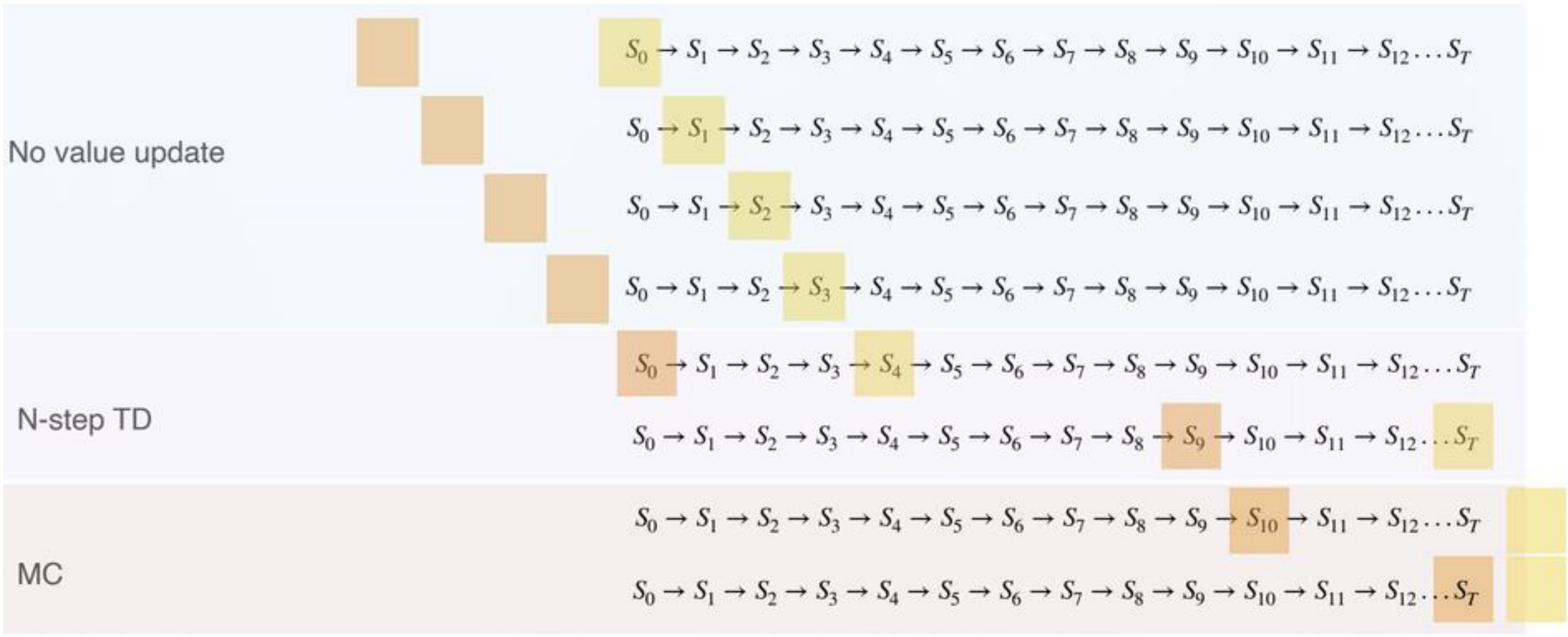


Image credit K. Fragkiadaki

On-policy n-step Action-Value Methods

- Action-value form of n-step return

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n,$$

- n-step Sarsa

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right]$$

- n-step expected Sarsa

$$G_t^{(n)} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n}) Q_{t+n-1}(S_{t+n}, a)$$

Off-policy n-step Action-Value Methods

- Importance-sampling ratio

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

- Weighted estimated value functions with importance-sampling ratio
- Off-policy n-step TD

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t) \right], \qquad 0 \le t < T,$$

- Off-policy n-step Sarsa

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right]$$

Tabular Value Function Learning

- TD state-value learning:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Q-learning:

$$\mathrm{Q}(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a \mathrm{Q}(S_{t+1}, a) - \mathrm{Q}(S_t, A_t)]$$

- We used a tabular setup: the value is retrieved from a table with a key of state/state-action pair

- What are the limitations?

Tabular Value Function Learning

- TD state-value learning:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Q-learning:

$$\mathrm{Q}(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a \mathrm{Q}(S_{t+1}, a) - \mathrm{Q}(S_t, A_t)]$$

- We used a tabular setup: the value is retrieved from a table with a key of state/state-action pair

- What are the limitations?
  - ➢ Discrete state/action space

Tabular Value Function Learning

- TD state-value learning:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Q-learning:

$$\mathrm{Q}(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a \mathrm{Q}(S_{t+1}, a) - \mathrm{Q}(S_t, A_t)]$$

- We used a tabular setup: the value is retrieved from a table with a key of state/state-action pair

- What are the limitations?
  - ➢ Discrete state/action space
  - ➢ Curse of dimensionality: too many states / state-action pairs stored in the memory

$$\mathbf{s} = 0 : V(\mathbf{s}) = 0.2$$
$$\mathbf{s} = 1 : V(\mathbf{s}) = 0.3$$
$$\mathbf{s} = 2 : V(\mathbf{s}) = 0.5$$

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

(more than atoms in the universe)

Tabular Value Function Learning

- TD state-value learning:

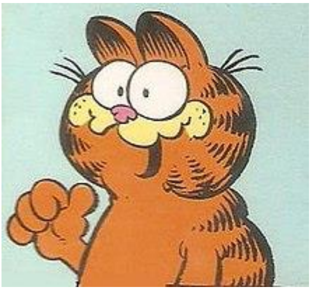
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Q-learning:

$$\mathrm{Q}(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a \mathrm{Q}(S_{t+1}, a) - \mathrm{Q}(S_t, A_t)]$$

- We used a tabular setup: the value is retrieved from a table with a key of state/state-action pair

- What are the limitations?
  - Discrete state/action space
  - Curse of dimensionality: too many states / state-action pairs stored in the memory
  - Closed world (can't generalize to unseen state/action)



Tabular Value Function Learning

- TD state-value learning:

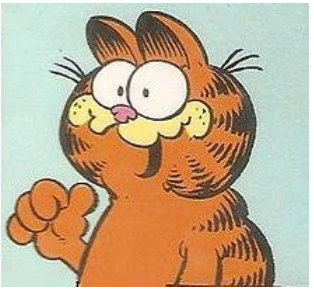
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Q-learning:

$$\mathrm{Q}(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \alpha[\overbrace{\boxed{R_{t+1} + \gamma \max_a \mathrm{Q}(S_{t+1}, a)}}^{\text{What are the labels}} - \overbrace{\boxed{\mathrm{Q}(S_t, A_t)}}^{\text{Fit an approximator}}]$$

- We used a tabular setup: the value is retrieved from a table with a key of state/state-action pair

- What are the limitations?
  - ➢ Discrete state/action space
  - ➢ Curse of dimensionality: too many states / state-action pairs stored in the memory
  - ➢ Closed world (can't generalize to unseen state/action)

Value Function Approximation

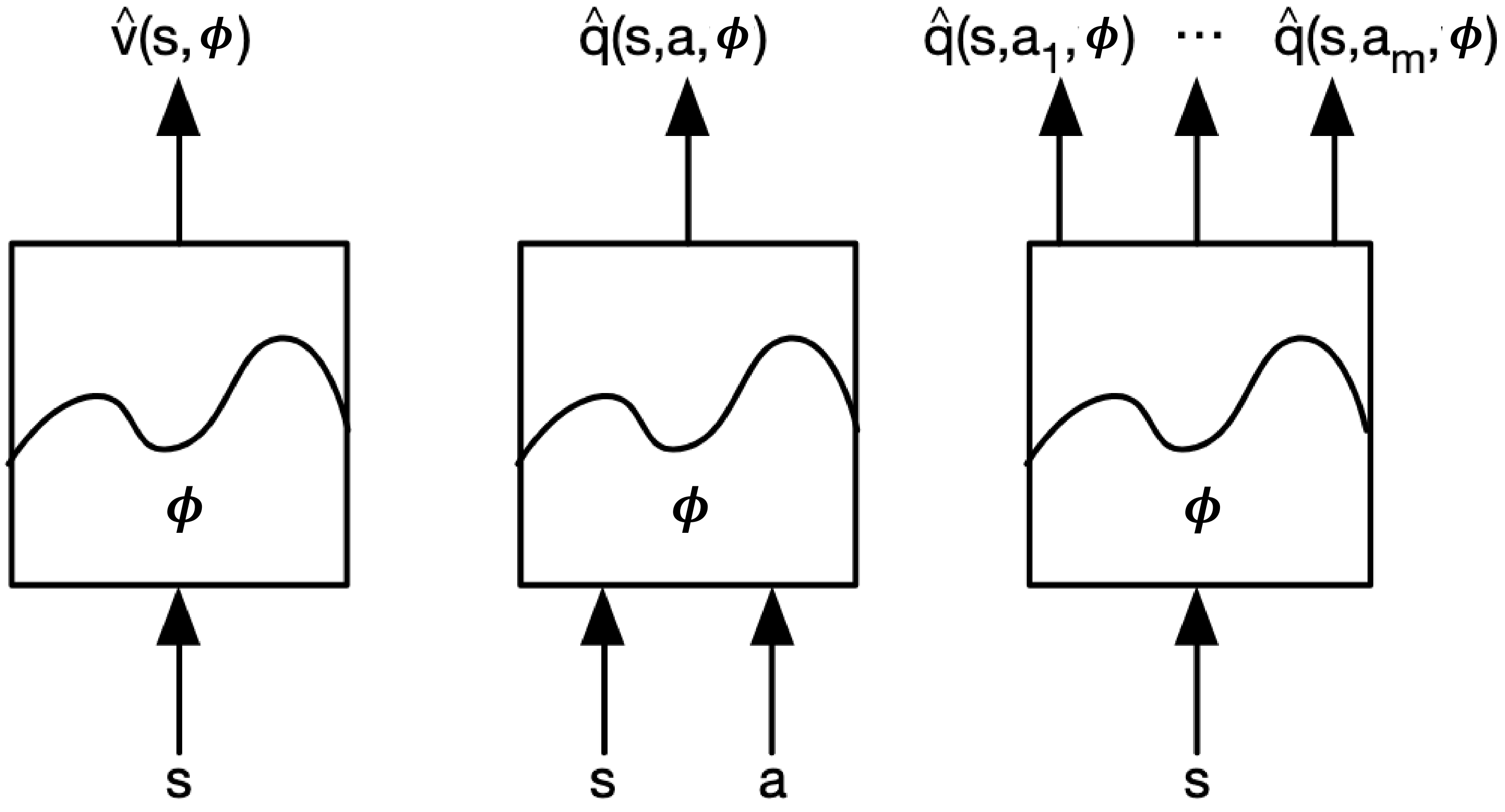
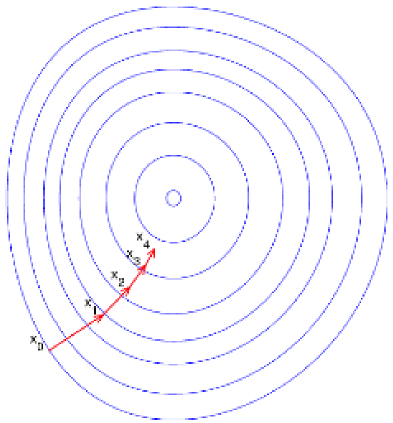


Image credit D. Silver

Gradient Descent

- Let $J(\boldsymbol{\phi})$ be a differentiable function of parameter vector $\boldsymbol{\phi}$
- Define the *gradient* of $J(\boldsymbol{\phi})$ to be

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



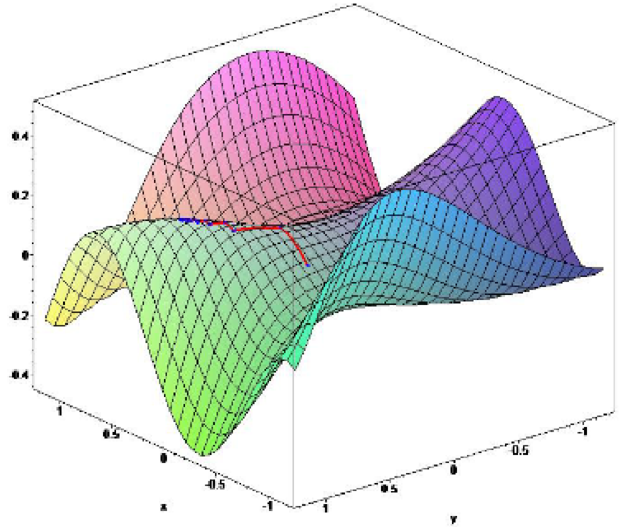


Image credit D. Silver

Gradient Descent

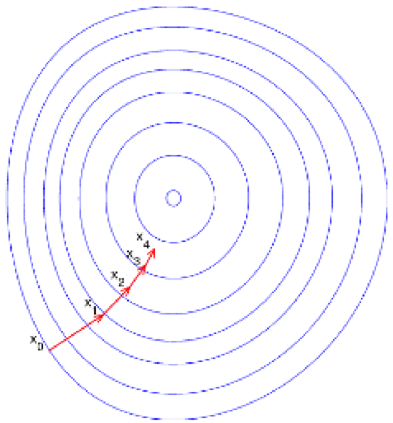
- Let $J(\boldsymbol{\phi})$ be a differentiable function of parameter vector $\boldsymbol{\phi}$
- Define the *gradient* of $J(\boldsymbol{\phi})$ to be

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

- To find a local minimum of $J(\boldsymbol{\phi})$
- Adjust $\boldsymbol{\phi}$ in direction of -ve gradient

$$\Delta \boldsymbol{\phi} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi})$$

where α is a step-size parameter



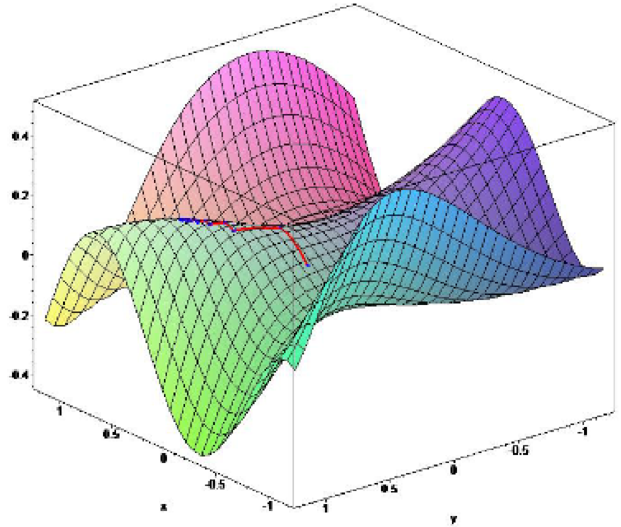


Image credit D. Silver

Value Function Approximation by Stochastic Gradient Descent

- Goal: find parameter vector $\boldsymbol{\phi}$ minimizing mean-squared error between approximate value function $\hat{V}(S, \boldsymbol{\phi})$ and true value function $V^{\pi}(S)$

$$J(\boldsymbol{\phi}) = \mathbb{E}_{\pi}\left[(V^{\pi}(S) - \hat{V}(S, \boldsymbol{\phi}))^2\right]$$

- Let $\boldsymbol{\mu}(S)$ denote how much time we spend in each state S under policy $\boldsymbol{\pi}$, then

$$J(\boldsymbol{\phi}) = \sum_{s \in \mathcal{S}}^{|\mathcal{S}|} \mu(S)\left[V^{\pi}(S) - \hat{V}(S, \boldsymbol{\phi})\right]^2 \qquad \text{s.t} \qquad \sum_{s \in \mathcal{S}} \mu(S) = 1$$

- In contrast to

$$J(\boldsymbol{\phi}) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}}^{|\mathcal{S}|} \left[V^{\pi}(S) - \hat{V}(S, \boldsymbol{\phi})\right]^2$$

Value Function Approximation by Stochastic Gradient Descent

- Goal: find parameter vector $\boldsymbol{\phi}$ minimising mean-squared error between approximate value function $\hat{V}(S, \boldsymbol{\phi})$ and true value function $V^{\pi}(S)$

$$J(\boldsymbol{\phi}) = \mathbb{E}_{\pi}\left[(V^{\pi}(S) - \hat{V}(S, \boldsymbol{\phi}))^2\right]$$

- Let $\mu(S)$ denote how much time we spend in each state S under policy π, then

$$J(\boldsymbol{\phi}) = \sum_{s \in \mathcal{S}}^{|\mathcal{S}|} \boxed{\mu(S)} \left[V^{\pi}(S) - \hat{V}(S, \boldsymbol{\phi})\right]^2 \qquad \text{s.t} \qquad \sum_{s \in \mathcal{S}} \mu(S) = 1$$

We care more about the frequently visited states, even though the total number of states is huge

- In contrast to

$$J(\boldsymbol{\phi}) = \boxed{\frac{1}{|\mathcal{S}|}} \sum_{s \in \mathcal{S}}^{|\mathcal{S}|} \left[V^{\pi}(S) - \hat{V}(S, \boldsymbol{\phi})\right]^2$$

On-policy State Distribution

- Let $h(S)$ be the initial state distribution, i.e, the probability that an episode starts at state S

- Un-normalized on-policy state probability satisfies the following recursions:

$$\eta(S) = h(S) + \sum_{S'} \eta(S') \sum_{a} \pi(a|S') p(S|S', a)$$

$$\mu(S) = \frac{\eta(S)}{\sum_{S'} \eta(S')}$$

Slide credit K. Fragkiadaki

Value Function Approximation by Stochastic Gradient Descent

- Goal: find parameter vector $\boldsymbol{\phi}$ minimising mean-squared error between approximate value fn $\hat{v}(s, \boldsymbol{\phi})$ and true value fn $v_\pi(s)$

$$J(\boldsymbol{\phi}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \boldsymbol{\phi}))^2 \right]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \boldsymbol{\phi} &= -\frac{1}{2} \alpha \nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \boldsymbol{\phi})) \nabla_{\boldsymbol{\phi}} \hat{v}(S, \boldsymbol{\phi}) \right] \end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \boldsymbol{\phi} = \alpha (v_\pi(S) - \hat{v}(S, \boldsymbol{\phi})) \nabla_{\boldsymbol{\phi}} \hat{v}(S, \boldsymbol{\phi})$$

- Expected update is equal to full gradient update

Slide credit D. Silver

What are the Targets of Value Function Approximation?

- We usually have no access to the true value function $V^{\pi}(S)$ (otherwise, the problem is solved). Let $\mathrm{y}(s)$ be the target of the value function approximator

$$J(\boldsymbol{\phi}) = \mathbb{E}_{\pi}\left[\left(y(S) - \hat{V}(S, \boldsymbol{\phi})\right)^2\right]$$

- What could be $\mathrm{y}(s)$?
  - Monte-Carlo Method?
  - Temporal Difference Method?

Slide credit D. Silver

Recap: Monte-Carlo and Temporal Difference Method

- Monte Carlo (MC) methods: must wait until the end of the episode to learn value functions (only when the return is known)

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- Temporal-Difference (TD) methods: combine Monte Carlo methods with Dynamic Programming methods that wait only until the next time step and bootstrap value functions from existing estimates

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Remember, we said in incremental method:

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} \times [\text{Target} - \text{OldEstimate}]$$

Slide credit D. Silver

Recap: Monte-Carlo and Temporal Difference Method

- Monte Carlo (MC) methods: must wait until the end of the episode to learn value functions (only when the return is known)

$$V(S_t) \leftarrow V(S_t) + \alpha[\boxed{G_t} - V(S_t)]$$

- Temporal-Difference (TD) methods: combine Monte Carlo methods with Dynamic Programming methods that wait only until the next time step and bootstrap value functions from existing estimates

This could be the target!

$$V(S_t) \leftarrow V(S_t) + \alpha[\boxed{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)]$$

- Remember, we said in incremental method:

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} \times [\text{Target} - \text{OldEstimate}]$$

Slide credit D. Silver

Monte-Carlo with Value Function Approximation

Return G_t is unbiased, noisy sample of true value $V_\pi(S_t)$

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\boldsymbol{\phi} \in \mathbb{R}^d$ arbitrarily (e.g., $\boldsymbol{\phi} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha\big[G_t - \hat{v}(S_t, \boldsymbol{\phi})\big]\nabla\hat{v}(S_t, \boldsymbol{\phi})$

Temporal Difference with Value Function Approximation

- Objective:

$$J(\boldsymbol{\phi}) = \mathbb{E}_\pi\left[(R_{t+1} + \gamma \hat{V}(S_{t+1}, \boldsymbol{\phi}) - \hat{V}(S_t, \boldsymbol{\phi}))^2\right]$$

- What are the gradients?

$$\nabla J(\boldsymbol{\phi}) = \mathbb{E}_\pi\left[(R_{t+1} + \gamma \hat{V}(S_{t+1}, \boldsymbol{\phi}) - \hat{V}(S_t, \boldsymbol{\phi}))^2\right]$$

$$= \mathbb{E}_\pi\left[\left(R_{t+1} + \gamma \hat{V}(S_{t+1}, \boldsymbol{\phi}) - \hat{V}(S_t, \boldsymbol{\phi})\right)\left(\cancel{\gamma \nabla \hat{V}(S_{t+1}, \boldsymbol{\phi})} - \nabla \hat{V}(S_t, \boldsymbol{\phi})\right)\right]$$

Ignore the dependence of the target on $\boldsymbol{\phi}$!
This is called semi-gradient method

Temporal Difference with Value Function Approximation

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\boldsymbol{\phi} \in \mathbb{R}^d$ arbitrarily (e.g., $\boldsymbol{\phi} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha\big[R + \gamma\hat{v}(S',\boldsymbol{\phi}) - \hat{v}(S,\boldsymbol{\phi})\big]\nabla\hat{v}(S,\boldsymbol{\phi})$
    $S \leftarrow S'$
  until S is terminal

How About Action-Value Function Approximation? Same Story!

- Goal: find parameter vector $\boldsymbol{\phi}$ minimizing mean-squared error between approximate action-value function $\hat{Q}(S, A, \boldsymbol{\phi})$ and true value function $Q_\pi(S, A)$

$$J(\boldsymbol{\phi}) = \mathbb{E}_\pi\left[(Q_\pi(S, A) - \hat{Q}(S, A, \boldsymbol{\phi}))^2\right]$$

- Use Stochastic Gradient Descent to optimize $\boldsymbol{\phi}$:

$$-\frac{1}{2}\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) = \left(Q_\pi(S, A) - \hat{Q}(S, A, \boldsymbol{\phi})\right) \nabla_{\boldsymbol{\phi}} \hat{Q}(S, A, \boldsymbol{\phi})$$

$$\Delta\boldsymbol{\phi} = \alpha \left(Q_\pi(S, A) - \hat{Q}(S, A, \boldsymbol{\phi})\right) \nabla_{\boldsymbol{\phi}} \hat{Q}(S, A, \boldsymbol{\phi})$$

How About Action-Value Function Approximation? Same Story!

- Goal: find parameter vector $\boldsymbol{\phi}$ minimizing mean-squared error between approximate action-value function $\hat{Q}(S, A, \boldsymbol{\phi})$ and true value function $Q_\pi(S, A)$

$$J(\boldsymbol{\phi}) = \mathbb{E}_\pi\left[\left(Q_\pi(S, A) - \hat{Q}(S, A, \boldsymbol{\phi})\right)^2\right]$$

- For MC:

$$J(\boldsymbol{\phi}) = \mathbb{E}_\pi\left[\left(G_t - \hat{Q}(S_t, A_t, \boldsymbol{\phi})\right)^2\right]$$

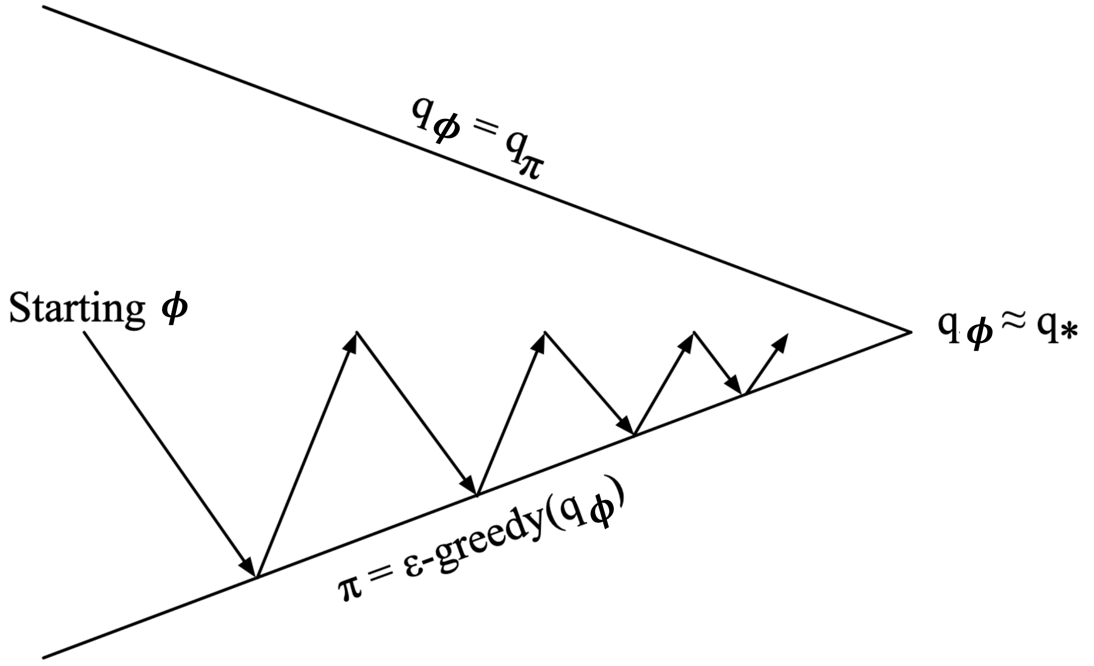
- For TD(0):

$$J(\boldsymbol{\phi}) = \mathbb{E}_\pi\left[\left(R_{t+1} + \gamma \hat{Q}(S_{t+1}, A_{t+1}, \boldsymbol{\phi}) - \hat{Q}(S_t, A_t, \boldsymbol{\phi})\right)^2\right]$$

- For Q-learning:

$$J(\boldsymbol{\phi}) = \mathbb{E}_\pi\left[\left(R_{t+1} + \gamma \max_a \hat{Q}(S_{t+1}, a, \boldsymbol{\phi}) - \hat{Q}(S_t, A_t, \boldsymbol{\phi})\right)^2\right]$$

Control with Action-Value Function Approximation



Policy evaluation Approximate policy evaluation, $\hat{q}(\cdot, \cdot, \boldsymbol{\phi}) \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

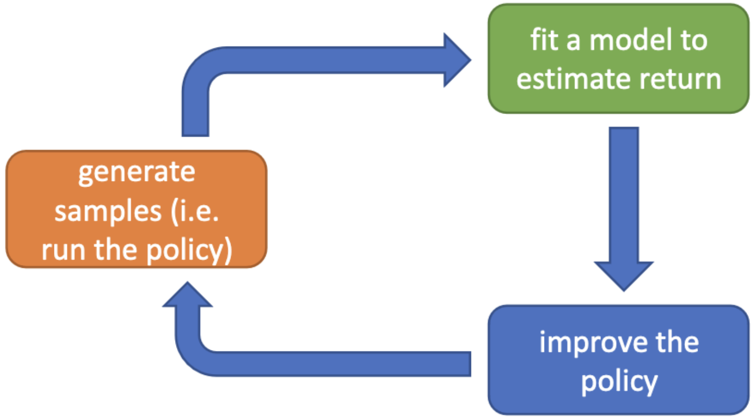
Slide credit D. Silver

So Far, We Discussed an Online Setup of 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. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

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



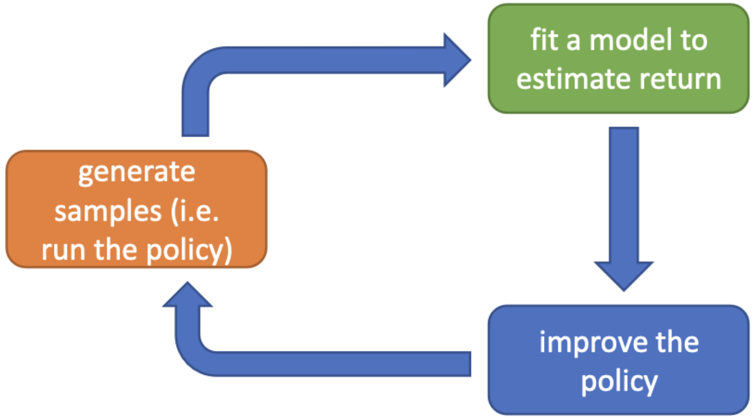
$$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$$

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

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



$$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$$

1. Sequential samples are highly correlated (little information gain)

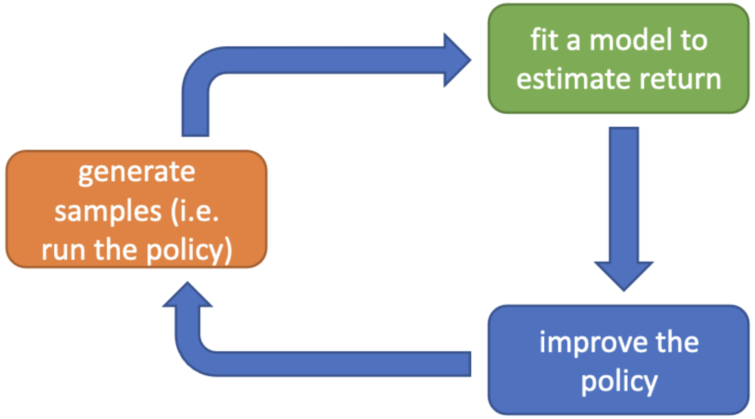
Problems in Online Q-Learning

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

1. Sequential samples are highly correlated (little information gain)
2. Target value changes whenever ϕ is updated

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



$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

Problems in Online Q-Learning

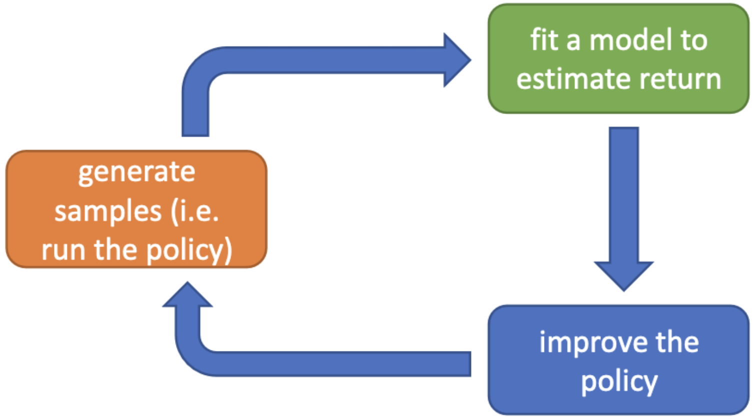
online Q iteration algorithm:

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

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

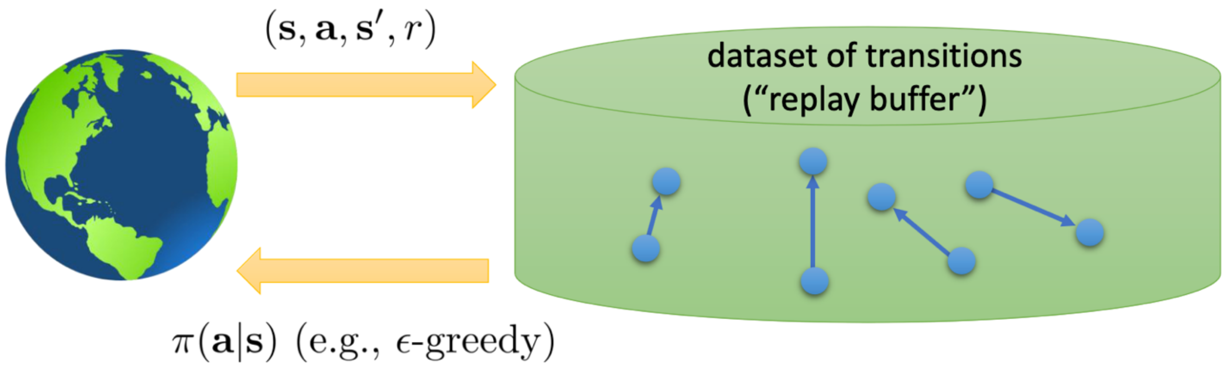
1. Sequential samples are highly correlated (little information gain)
2. Target value changes whenever ϕ is updated
3. Gradients don't pass through $Q_\phi(s'_i, a'_i)$ (but it becomes numerically unstable when propagating gradients through $Q_\phi(s'_i, a'_i)$)

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



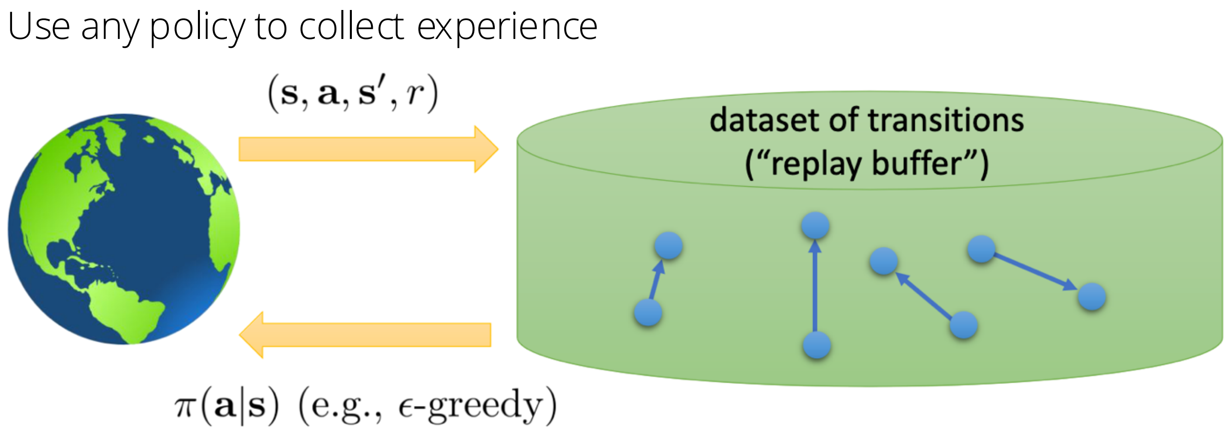
$$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$$

Solution 1: Use Replay Buffer to Decorrelate Samples



- Collect transitions data, using a random policy for exploration (find unseen states and transitions), and a deterministic policy for exploitation (obtain most promising states)

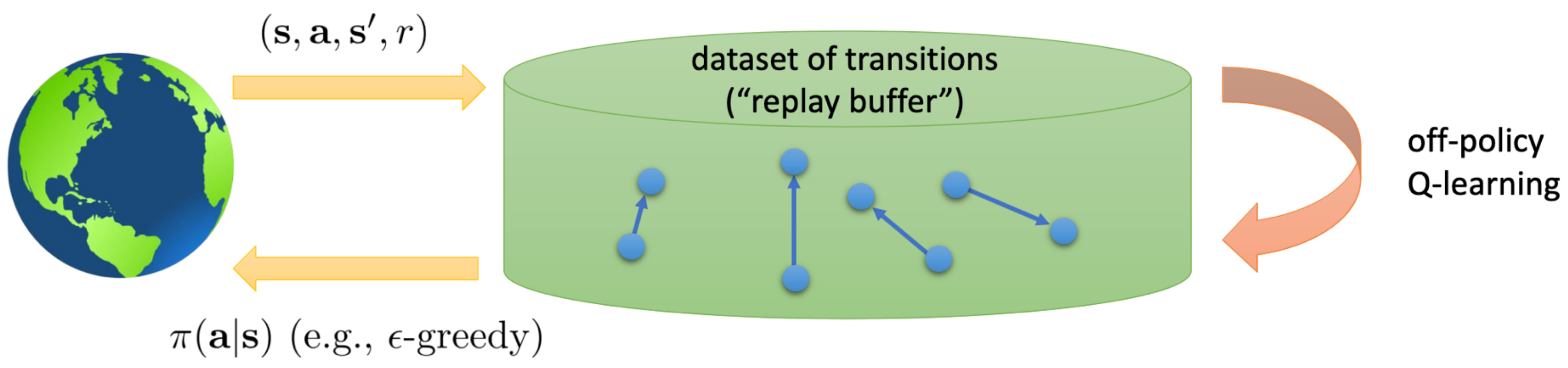
Solution 1: Use Replay Buffer to Decorrelate Samples



- Collect transitions data, using a random policy for exploration (find unseen states and transitions), and a deterministic policy for exploitation (obtain most promising states)

- Sample a batch of $\mathcal{B} = \{(s, a, s', r)\}$ from every collected (s, a, s', r), temporally neighboring (s, a, r) may not be sampled.

Solution 1: Use Replay Buffer to Decorrelate Samples



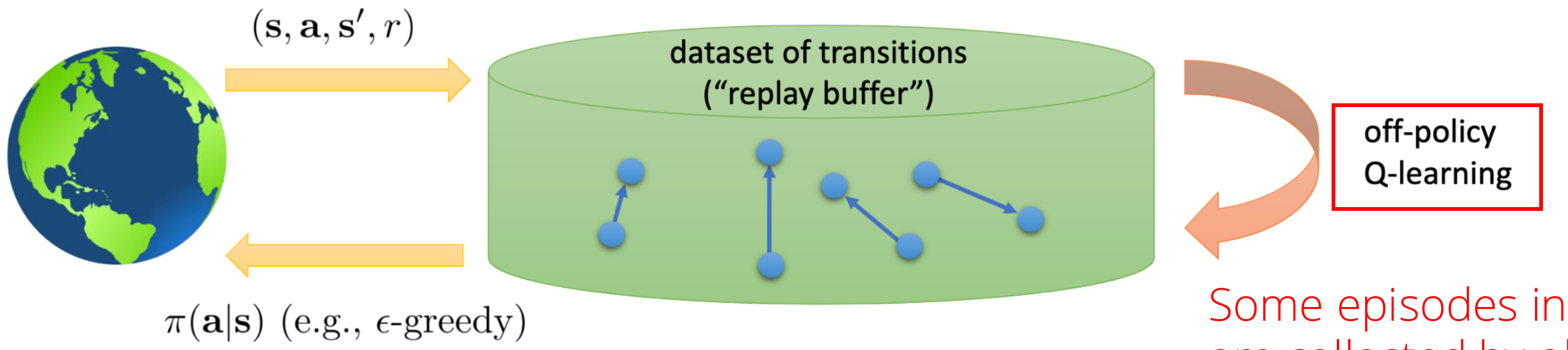
full Q-learning with replay buffer:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$

$K\times$

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

Solution 1: Use Replay Buffer to Decorrelate Samples



Some episodes in the replay buffer are collected by older policies

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 $\mathcal{B}$

$K\times$

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

Fully Fitted Q-Iteration vs. 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. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

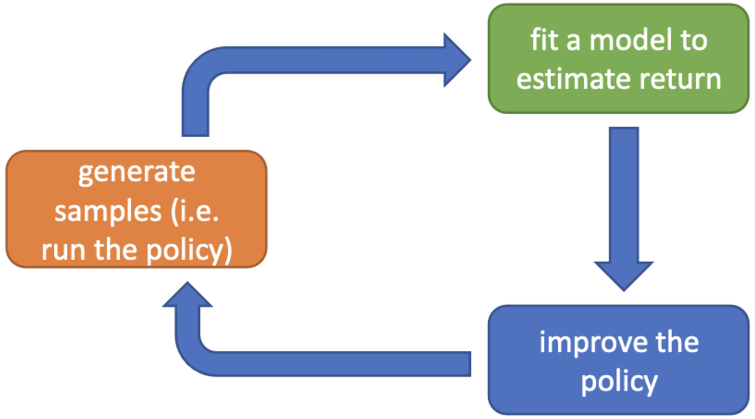
full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy

$K\times$

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 $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

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



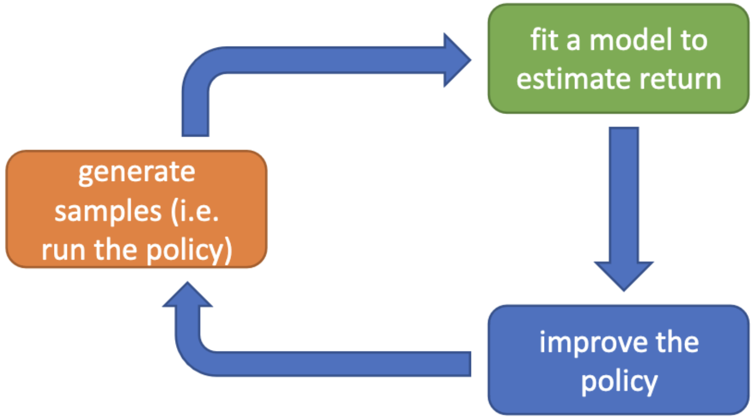
$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

Fully Fitted Q-Iteration vs. 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. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

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



$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. $\boxed{\text{set } \mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)}$

$K\times$

3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

We still have these problems:

2. Target value changes whenever ϕ is updated
3. Gradients don't pass through $Q_\phi(s'_i, a'_i)$

Solution 2 and 3: Use Target Networks

Q-learning with replay buffer and target network:

$N\times$ $K\times$

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

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: $\phi' \leftarrow \phi$

$N \times$

2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$

$K \times$

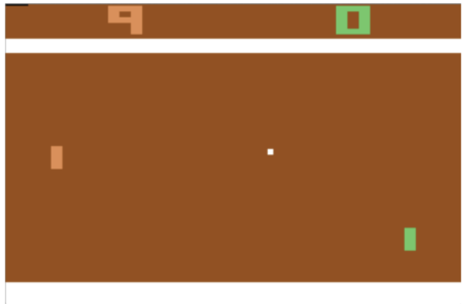
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

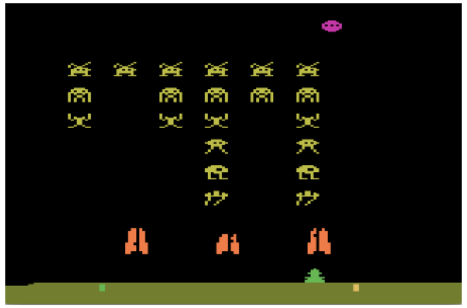
"classic" deep Q-learning algorithm:

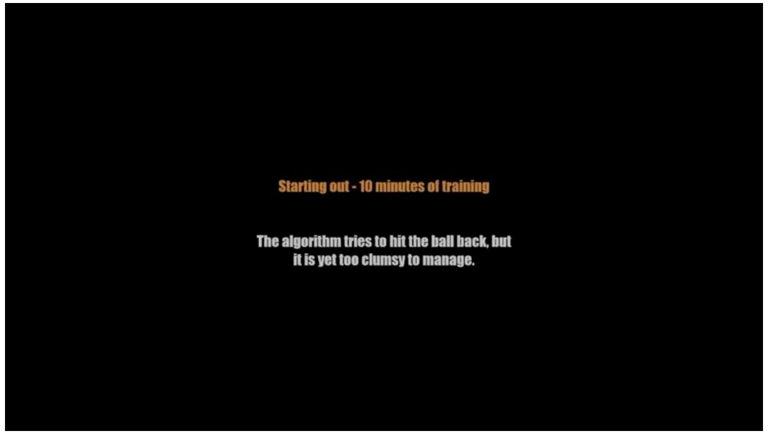
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ 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

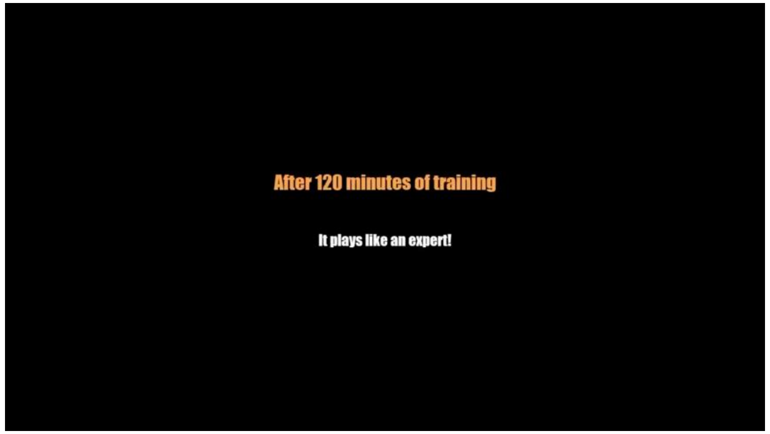
$K = 1$

DQN Surpasses Human Experts on (some) Atari Games

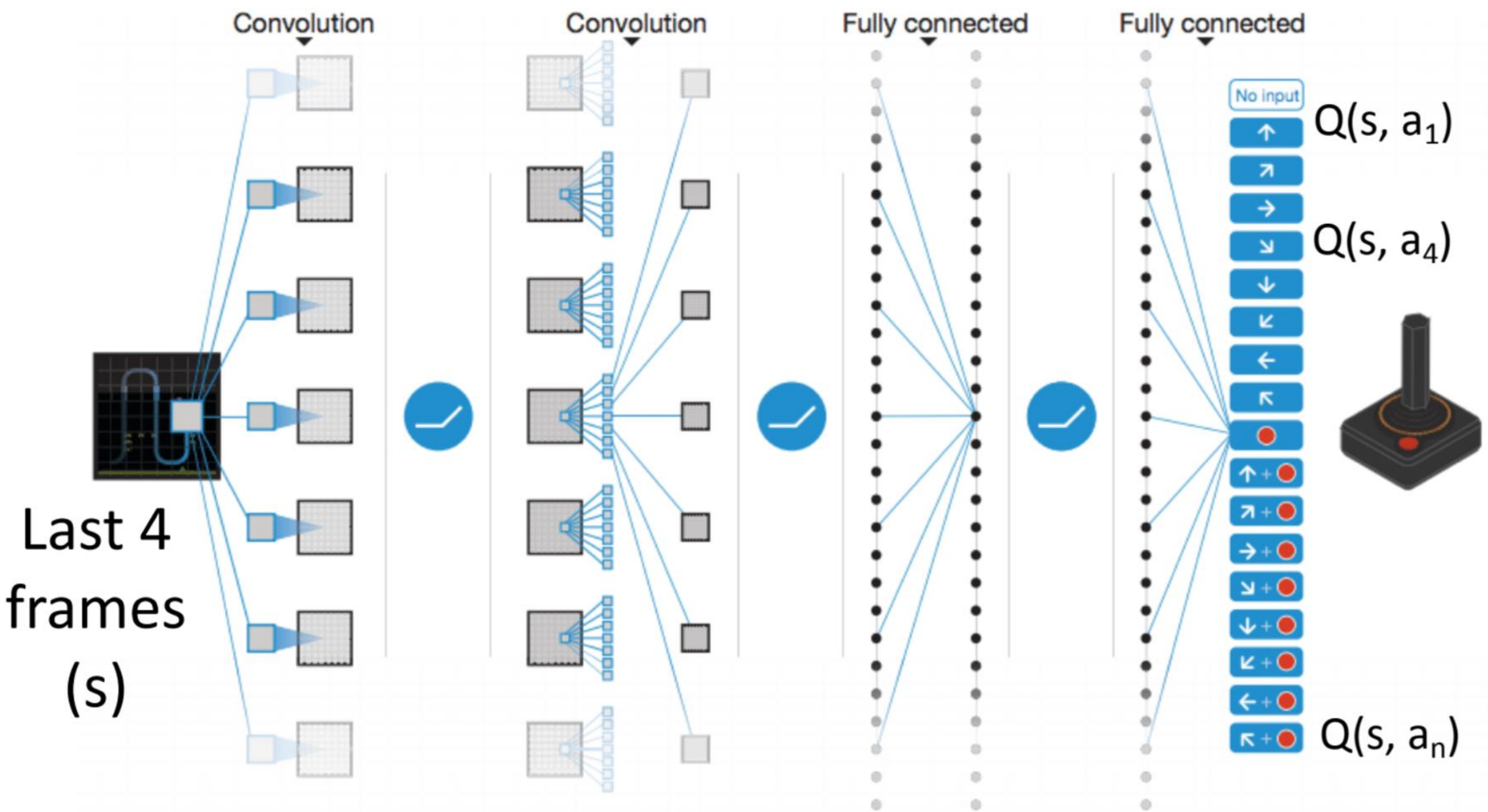






DQN Estimates State-Action Values

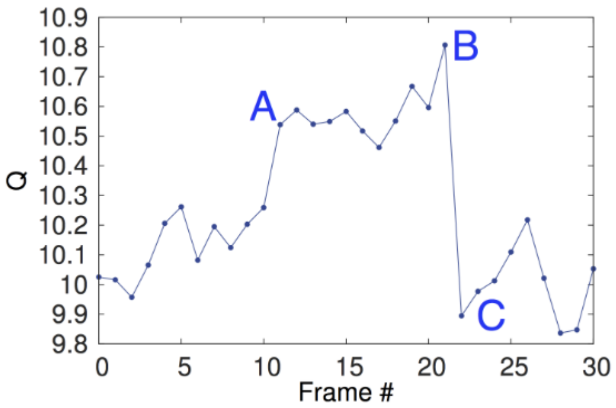


With probability $(1 - \varepsilon)$ → execute $\max_a Q(s,a)$

With probability ε → execute random action

Image credit P. Agrawal

DQN Captures High-Value State-Action Pairs

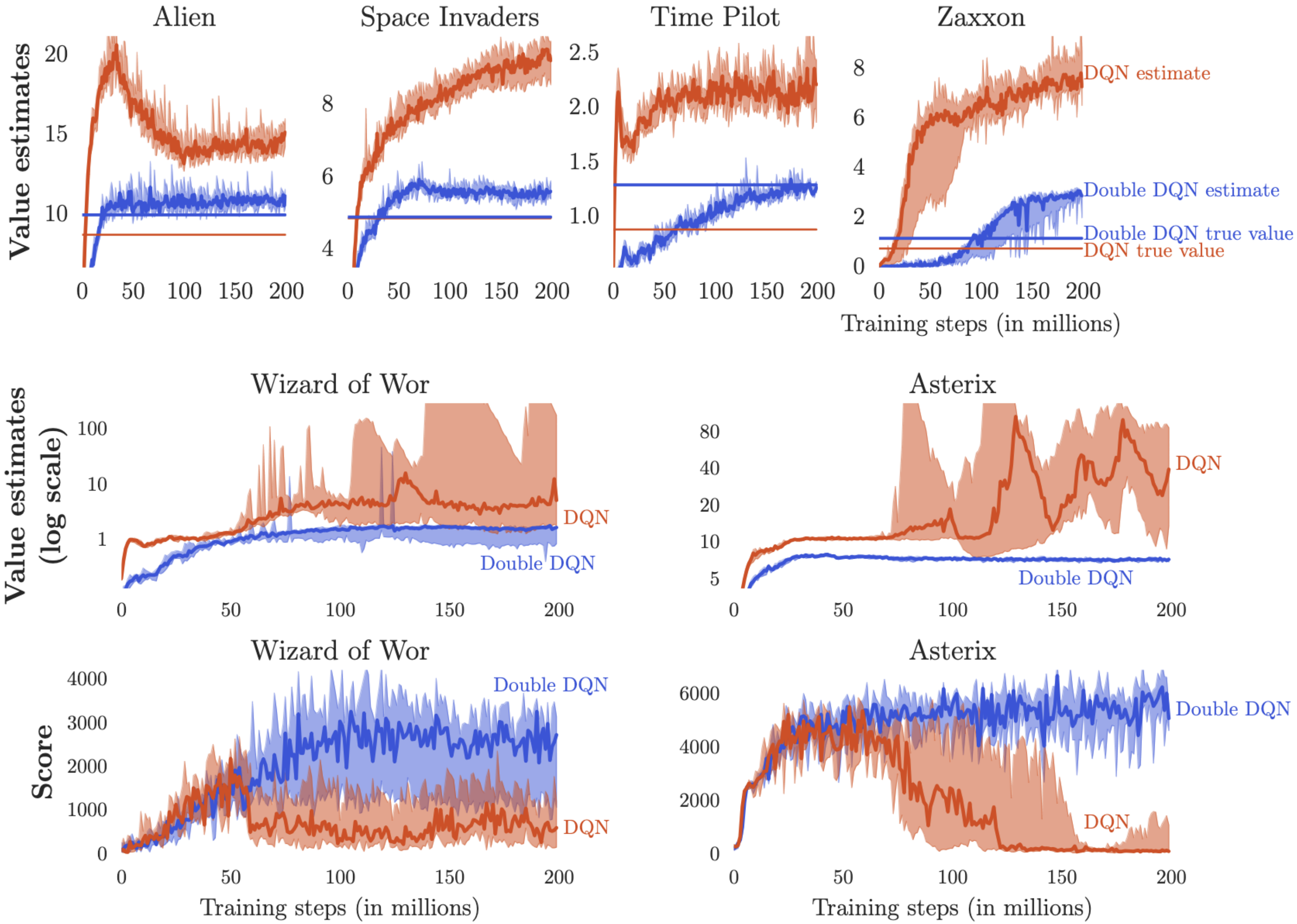


A

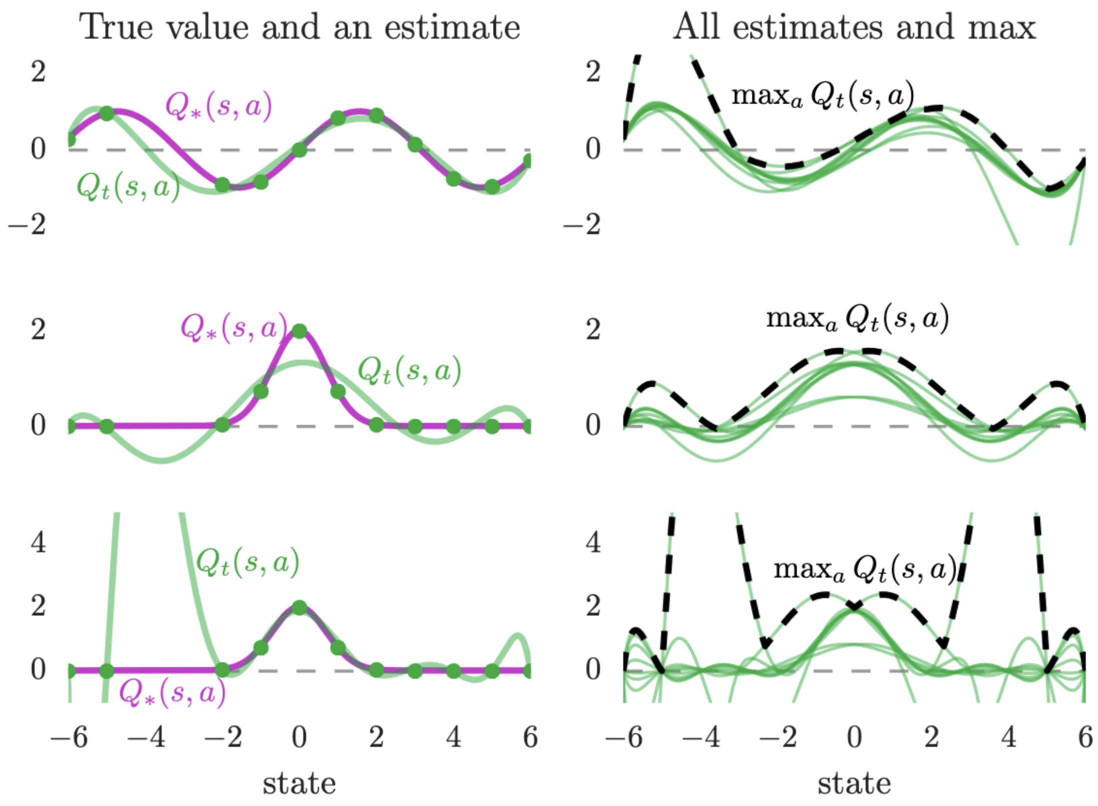
B

C

DQN Has a Problem in Overestimating Values

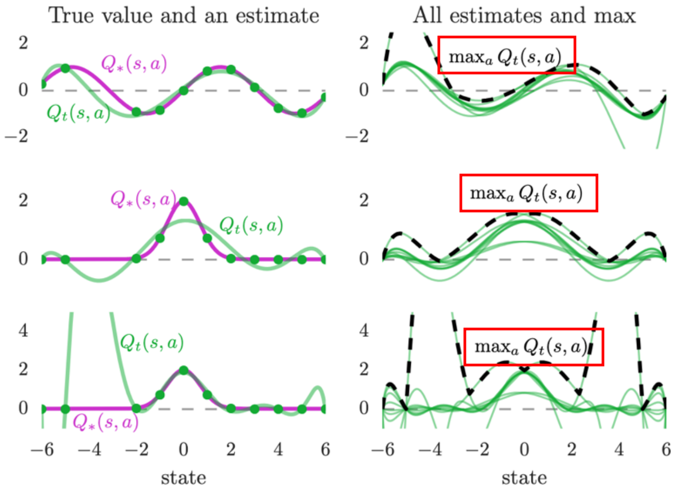


Why does DQN Overestimate Values?



$Q_*(s,a)$ is the true value function
$Q_t(s,a)$ is the estimated value function

Why does DQN Overestimate Values?



$Q_*(s,a)$ is the true value function
$Q_t(s,a)$ is the estimated value function

$$\hat{Q}(s,a_1) = Q(s,a_1) + \epsilon_1(s)$$

$$\hat{Q}(s,a_2) = Q(s,a_2) + \epsilon_2(s)$$

$$\vdots$$

$$\hat{Q}(s,a_N) = Q(s,a_N) + \epsilon_N(s)$$

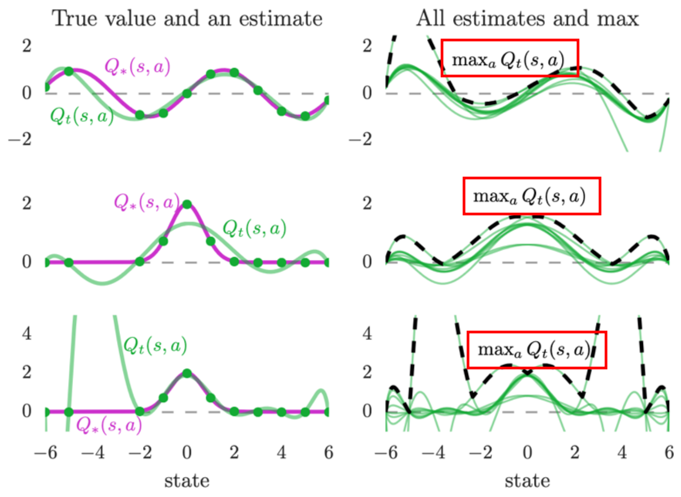
$$\max_a \hat{Q}(s,a) = \max_i \left(Q(s,a_i) + \epsilon_i(s)\right)$$

assume $V(s) = Q(s,a_i)\ \forall i$

$$\max_a \hat{Q}(s,a) = V(s) + \max_i(\epsilon_i(s))$$

$$\max_a \hat{Q}(s,a) \geq V(s)$$

Why does DQN Overestimate Values?



$Q_*(s,a)$ is the true value function
$Q_t(s,a)$ is the estimated value function

See the proof in "Deep Reinforcement Learning with Double Q-learning"

if $\Sigma_a\left(\hat{Q}(s,a) - V(s)\right) = 0$ unbiased estimator

but, $\Sigma_a\left(\hat{Q}(s,a) - V(s)\right)^2 = C$ estimation error

then,

$$\max \hat{Q}(s,a) \geq V(s) + \sqrt{\frac{C}{m-1}}$$ (m: number of actions)

$$\hat{Q}(s,a_1) = Q(s,a_1) + \epsilon_1(s)$$

$$\hat{Q}(s,a_2) = Q(s,a_2) + \epsilon_2(s)$$

$$\vdots$$

$$\hat{Q}(s,a_N) = Q(s,a_N) + \epsilon_N(s)$$

$$\max_a \hat{Q}(s,a) = \max_i \left(Q(s,a_i) + \epsilon_i(s)\right)$$

assume $V(s) = Q(s,a_i)\ \forall i$

$$\max_a \hat{Q}(s,a) = V(s) + \max_i(\epsilon_i(s))$$

$$\max_a \hat{Q}(s,a) \geq V(s)$$

Recap: Tabular Double Q-Learning

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big].$$

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Double Deep Q-Learning

- Standard Q-learning:

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

- Double Q-learning:

$$y = r + \gamma Q_{\phi'}(s', \max_{a'} Q_{\boldsymbol{\phi}}(s', a'))$$

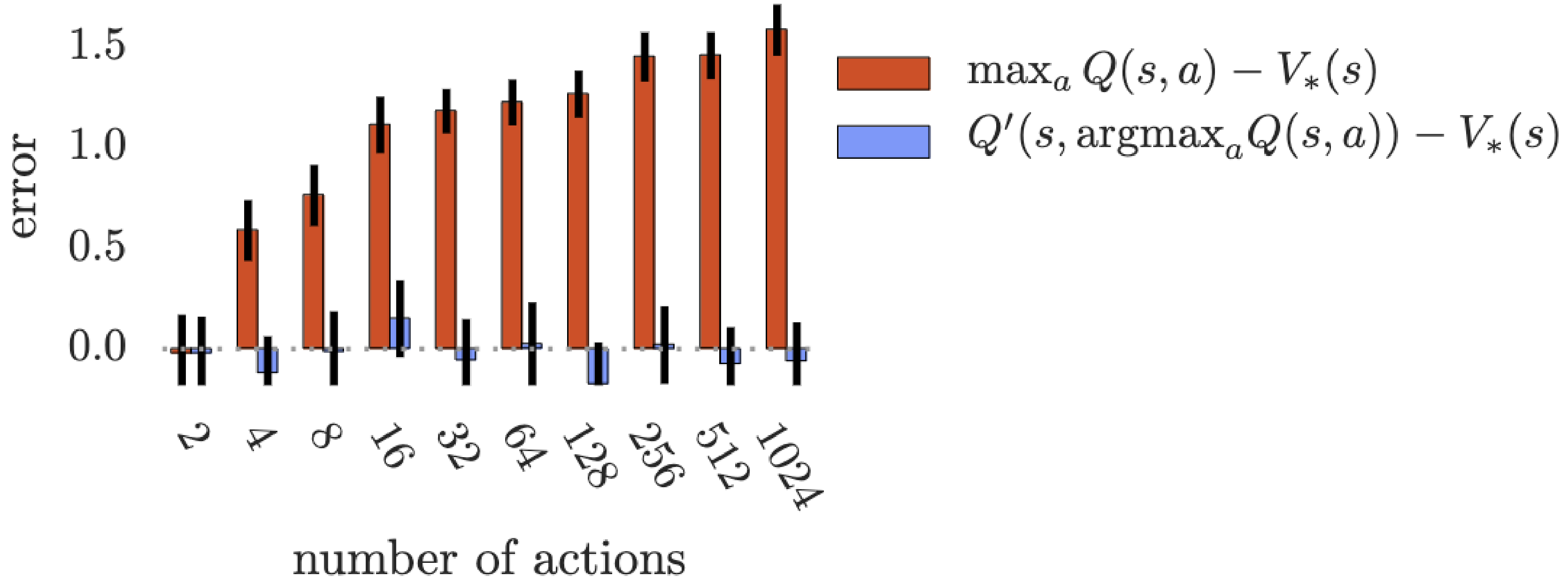
- Using different value function to reduce overestimation: $\boldsymbol{Q_{\phi}}$ for selecting actions and $\boldsymbol{Q_{\phi'}}$ for evaluating actions.
  - ➢ a' that induces overestimated $Q_{\phi'}(s', a')$ is less frequently selected

Double Deep Q-Learning

- Double Q-learning:

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

- Using different value function to reduce overestimation: Q_{ϕ} for selecting actions and $Q_{\phi'}$ for evaluating actions.



Clipped Double Deep Q-Learning

- Unfortunately, ue to the slow-changing policy in an actor-critic setting, the current and target value estimates remain too similar to avoid maximization bias

- Clipped Double Q-learning:

$$y = r + \gamma \min_{\breve{\phi} \in \{\phi', \boldsymbol{\phi}\}} Q_{\breve{\phi}}(s', \max_{a'} Q_{\boldsymbol{\phi}}(s', a'))$$

- Using different value function to reduce overestimation: $\boldsymbol{Q_{\phi}}$ for selecting actions and the minimum of the two values provided by$\boldsymbol{Q_{\phi}}$ and $Q_{\phi'}$ for evaluating actions.

Addressing Function Approximation Error in Actor-Critic Methods, Fujimoto et al. 2018

Can We Do Better than MC / TD methods?

- Idea: Obtain more precise target by searching.
- Use Monte-Carlo Tree Search (MCTS) for Q value estimation and action selection at training time instead of the Q learning update rule.
- At test time just use the reactive policy network, without any look-ahead planning. In other words, imitate the MCTS planner.

Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning

Xiaoxiao Guo
Computer Science and Eng.
University of Michigan
guoxiao@umich.edu

Satinder Singh
Computer Science and Eng.
University of Michigan
baveja@umich.edu

Honglak Lee
Computer Science and Eng.
University of Michigan
honglak@umich.edu

Richard Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Xiaoshi Wang
Computer Science and Eng.
University of Michigan
xiaoshiw@umich.edu

Monte-Carlo Tree Search

1. **Selection**
    - Used for nodes we have seen before
    - Pick actions according to UCB
2. **Expansion**
    - Used when we reach the frontier
    - Add one node per rollout
3. **Simulation**
    - Used beyond the search frontier
    - Don't bother with UCB, just pick actions randomly
4. **Back-propagation**
    - After reaching a terminal node
    - Update value and visits for states expanded in selection and expansion

Bandit based Monte-Carlo Planning, *Kocsis and Szepesvari, 2006*

Check this video:
https://youtu.be/UXW2yZndl7U?si=S0WHGlC1B2UVAYEA

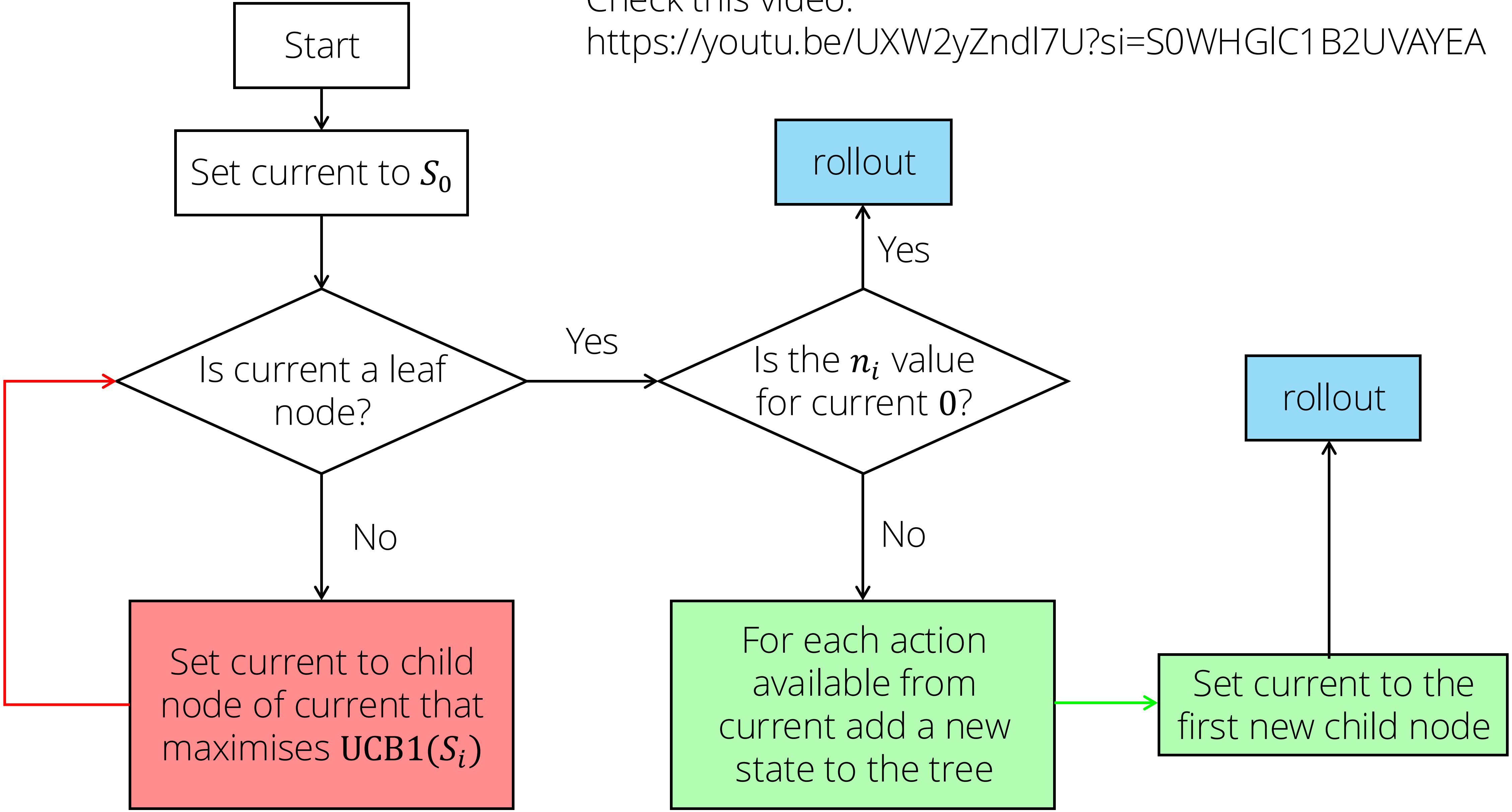
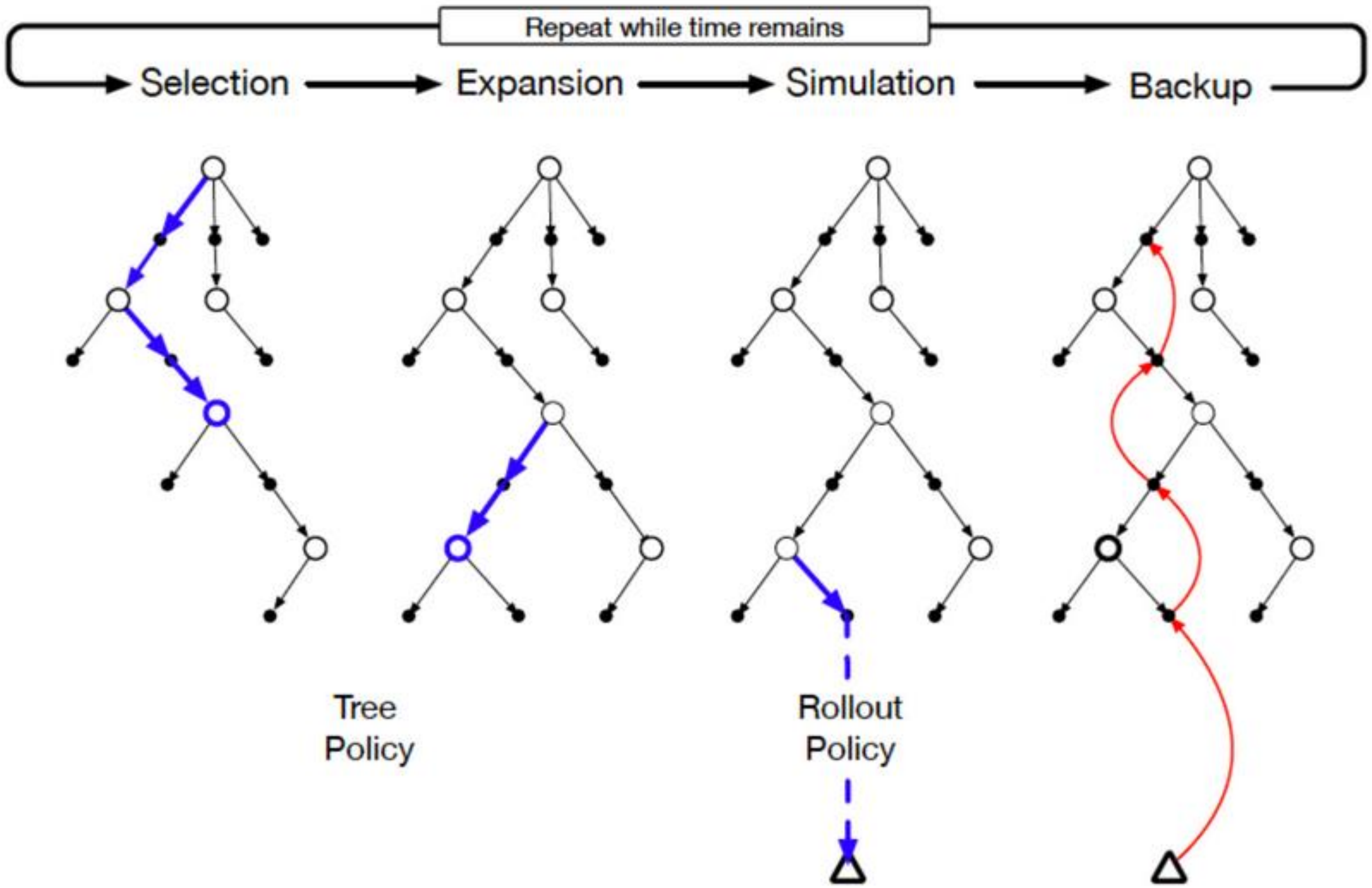


Image credit S. Levine

Monte-Carlo Tree Search



Slide credit K. Fragkiadaki

Results

Agent	B.Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S.Invaders
DQN	4092	168	470	20	1952	1705	581
-best	5184	225	661	21	4500	1740	1075
UCC	5342 (20)	175(5.63)	558(14)	19(0.3)	11574(44)	2273(23)	672(5.3)
-best	10514	351	942	21	29725	5100	1200
-greedy	5676	269	692	21	19890	2760	680
UCC-I	5388(4.6)	215(6.69)	601(11)	19(0.14)	13189(35.3)	2701(6.09)	670(4.24)
-best	10732	413	1026	21	29900	6100	910
-greedy	5702	380	741	21	20025	2995	692
UCR	2405(12)	143(6.7)	566(10.2)	19(0.3)	12755(40.7)	1024 (13.8)	441(8.1)

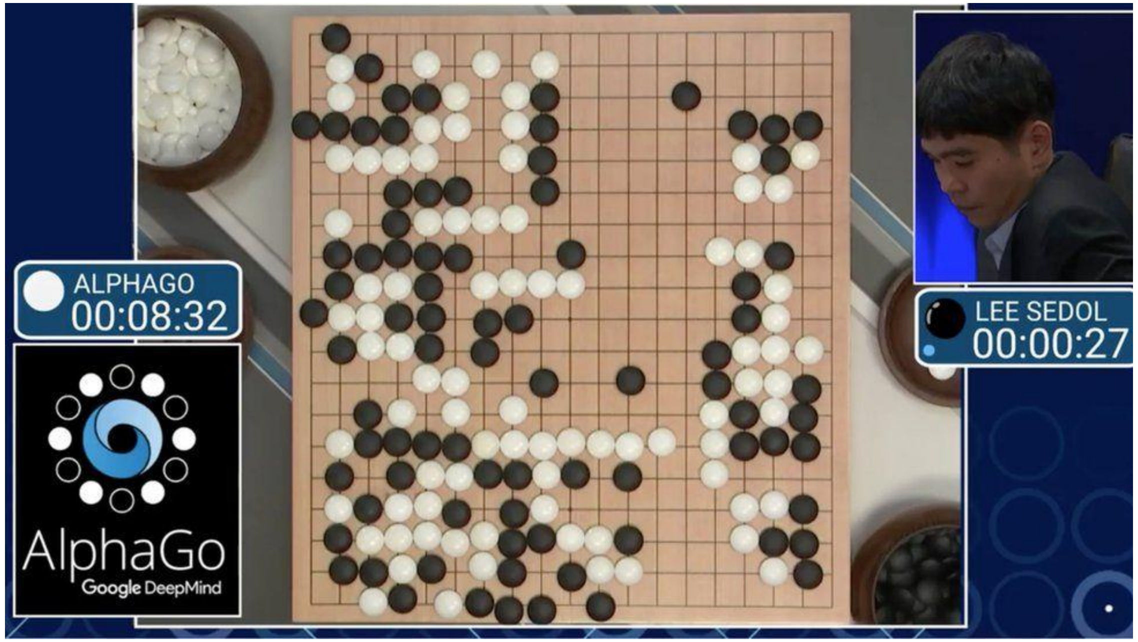
Table 2: Performance (game scores) of the off-line UCT game playing agent.

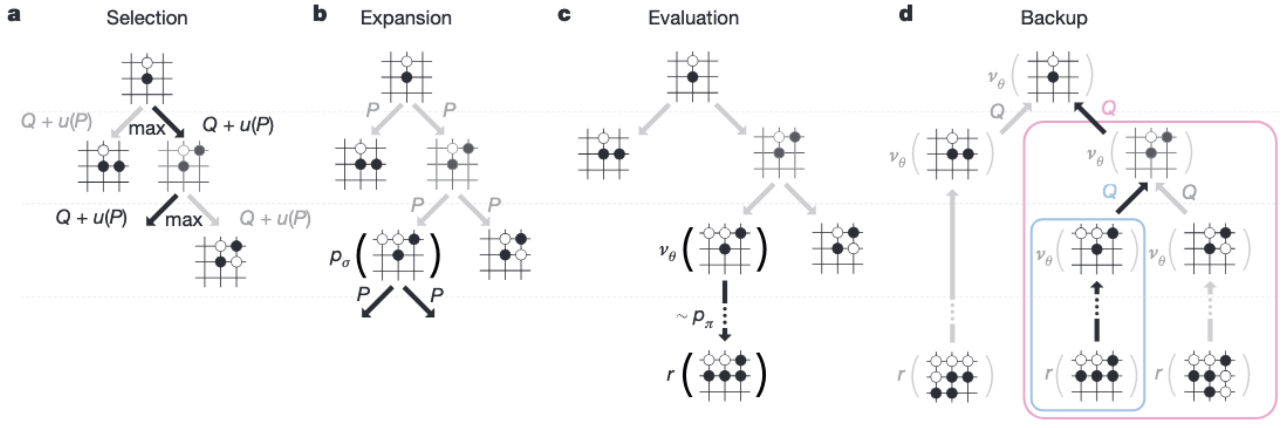
Agent	B.Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S.Invaders
UCT	7233	406	788	21	18850	3257	2354

MCTS planning discovers better actions than deep Q learning. It takes though "a few days on a recent multicore computer to play for each game"

Slide credit K. Fragkiadaki

Similar Idea is Used in Alpha Go / Super-human Agents





- Check different implementation of MCTS
- They also use searching during testing
  - ➢ An awesome video by Noah Brown about test-time planning: https://youtu.be/eaAonE58sLU?si=wZmo8gQFJXCLU39Y
  - ➢ We'll talk more about learning with planning in this class

Mastering the game of Go with deep neural networks and tree search. Silver et al.

Wait! DQN Still Has a Problem...

- Standard Q-learning:

$$y = r + \gamma \max_{a'} Q_{\phi'}(s', a') = r + \gamma Q_{\phi'}(s', \boxed{\max_{a'}} Q_{\phi'}(s', a'))$$

- Double Q-learning:

$$y = r + \gamma Q_{\phi'}(s', \boxed{\max_{a'}} Q_{\color{red}{\phi}}(s', a'))$$

- Using different value function $Q_{\color{red}{\phi}}$ for selecting action reduces overestimation.
  - ➢ a' that induces overestimated $Q_{\phi'}(s', a')$ is less frequently selected

What if the action is continuous?

Wait! DQN Still Has a Problem…

- Standard Q-learning:

$$y = r + \gamma \max_{a'} Q_{\phi'}(s', a') = r + \gamma Q_{\phi'}(s', \boxed{\max_{a'}} Q_{\phi'}(s', a'))$$

- Double Q-learning:

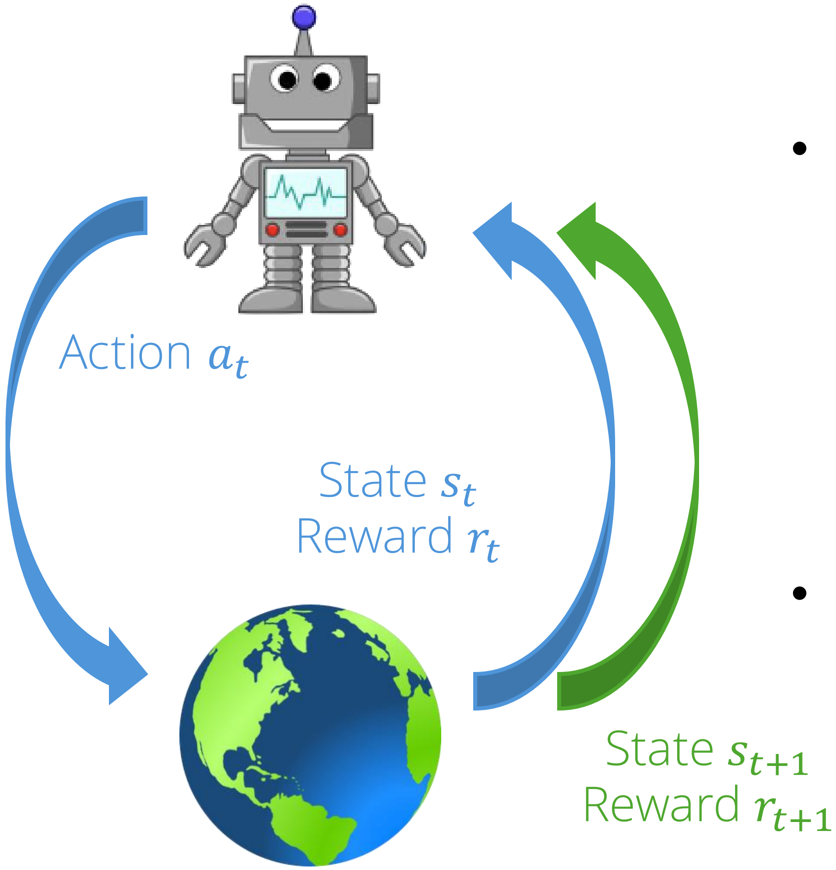
$$y = r + \gamma Q_{\phi'}(s', \boxed{\max_{a'}} Q_{\phi}(s', a'))$$

- Using different value function Q_{ϕ} for selecting action reduces overestimation.
  - ➢ a' that induces overestimated $Q_{\phi'}(s', a')$ is less frequently selected

What if the action is continuous?

There're many solutions, but the most obvious one may be learning an action policy for maximizing the value…

Recap: Reinforcement Learning Aims to Maximize the Total Reward of an Episode of Interaction



- A trajectory of interaction in the environment

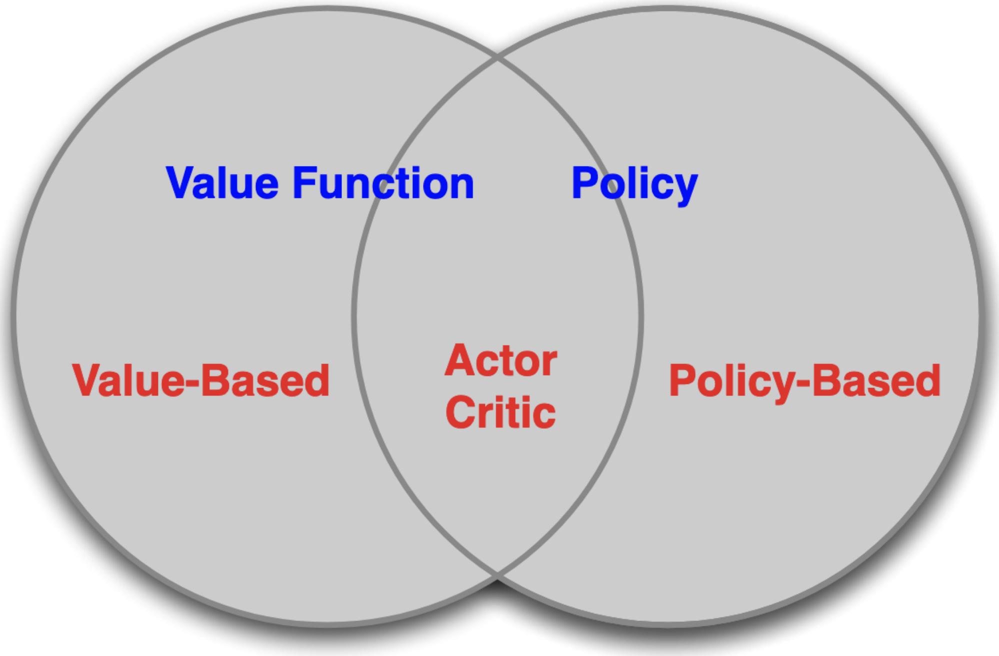
$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \overbrace{\boxed{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}}^{\text{The policy}} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

- Maximize the expected value of the cumulative sum of reward

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Valued-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Slide credit D. Silver

Policy Gradient

- Let $r(\tau) = \sum_t r(s_t, a_t)]$
- Let $J(\theta) = \sum_{\tau\sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] = \int p_\theta(\tau) r(\tau) d\tau$
- Compute gradients of θ w.r.t to $J(\theta)$

$$\nabla J(\theta) = \int \boxed{\nabla p_\theta(\tau)} r(\tau) d\tau = \int \boxed{p_\theta(\tau) \nabla \log p_\theta(\tau)} r(\tau) d\tau = E_{\tau\sim p_\theta(\tau)}[\nabla \log p_\theta(\tau) r(\tau)]$$

$$\boxed{\nabla \log p_\theta = \frac{1}{p_\theta} \nabla p_\theta}$$

Policy Gradient

- Let $r(\tau) = \sum_t r(s_t, a_t)]$
- Let $J(\theta) = \sum_{\tau\sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] = \int p_\theta(\tau) r(\tau) d\tau$
- Compute gradients of θ w.r.t to $J(\theta)$

$$\nabla J(\theta) = \int \nabla p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla \log p_\theta(\tau)\, r(\tau) d\tau = E_{\tau\sim p_\theta(\tau)}[\nabla \log p_\theta(\tau) r(\tau)]$$

$$\nabla \log p_\theta(\tau) = \nabla \log[p(s_0) \times \prod_{t=0}^{T} \pi_\theta(a_t\,|s_t) \times p(s_{t+1}|s_t, a_t)]$$

$$= \nabla[\log p(s_0) + \sum_{t=0}^{T} \log \pi_\theta(a_t\,|s_t) + \sum_{t=0}^{T} \log(s_{t+1}|s_t, a_t)]$$

Policy Gradient

- Let $r(\tau) = \sum_t r(s_t, a_t)]$
- Let $J(\theta) = \sum_{\tau \sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] = \int p_\theta(\tau) r(\tau) d\tau$
- Compute gradients of $\boldsymbol{\theta}$ w.r.t to $J(\theta)$

$$\nabla J(\theta) = \int \nabla p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla \log p_\theta(\tau)\, r(\tau) d\tau = E_{\tau \sim p_\theta(\tau)}[\nabla \log p_\theta(\tau) r(\tau)]$$

$$\nabla \log p_\theta(\tau) = \nabla \log[p(s_0) \times \prod_{t=0}^{T} \pi_\theta(a_t \,|s_t) \times p(s_{t+1}|s_t, a_t)]$$

$$= \nabla[\cancel{\log p(s_0)} + \sum_{t=0}^{T} \log \pi_\theta(a_t \,|s_t) + \cancel{\sum_{t=0}^{T} \log(s_{t+1}|s_t, a_t)}]$$

$$\nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t \,|s_t)\right)\left(\sum_{t=0}^{T} r(s_t, a_t)\right)\right]$$

Policy Gradient

- Let $r(\tau) = \sum_t r(s_t, a_t)]$
- Let $J(\theta) = \sum_{\tau \sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] = \int p_\theta(\tau) r(\tau) d\tau$
- Compute gradients of θ w.r.t to $J(\theta)$

$$\nabla J(\theta) = \int \nabla p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla \log p_\theta(\tau)\, r(\tau) d\tau = E_{\tau \sim p_\theta(\tau)}[\nabla \log p_\theta(\tau) r(\tau)]$$

$$\nabla \log p_\theta(\tau) = \nabla \log[p(s_0) \times \prod_{t=0}^{T} \pi_\theta(a_t\,|s_t) \times p(s_{t+1}|s_t, a_t)]$$

$$= \nabla[\log \cancel{p}(s_0) + \sum_{t=0}^{T} \log \pi_\theta(a_t\,|s_t) + \sum_{t=0}^{T} \log(\cancel{s_{t+1}}|s_t, a_t)]$$

$$\nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t\,|s_t)\right)\left(\sum_{t=0}^{T} r(s_t, a_t)\right)\right]$$

The transition function is not needed.
We just need experience!

Maximizing Action Trajectories using Returns

- Policy Gradients weights gradients with a sum of rewards of the trajectory:

$$\nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t \mid s_t)\right)\boxed{\left(\sum_{t=0}^{T} r(s_t, a_t)\right)}\right]$$

The policy is optimized to follow the trajectory of high-reward episodes

- Policy Gradients learn from "trial and error"

Slide adapted from S. Levine

Maximizing Action Trajectories using Returns

- Policy Gradients weights gradients with a sum of rewards of the trajectory:

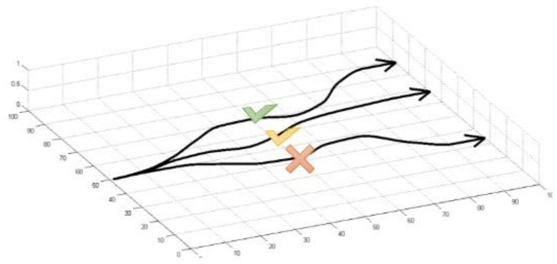
$$\nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t \mid s_t)\right)\left(\sum_{t=0}^{T} r(s_t, a_t)\right)\right]$$

- Idea: approximate expected value by sampling

$$\nabla J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t \mid s_t)\right)\left(\sum_{t=0}^{T} r(s_t, a_t)\right)\right]$$

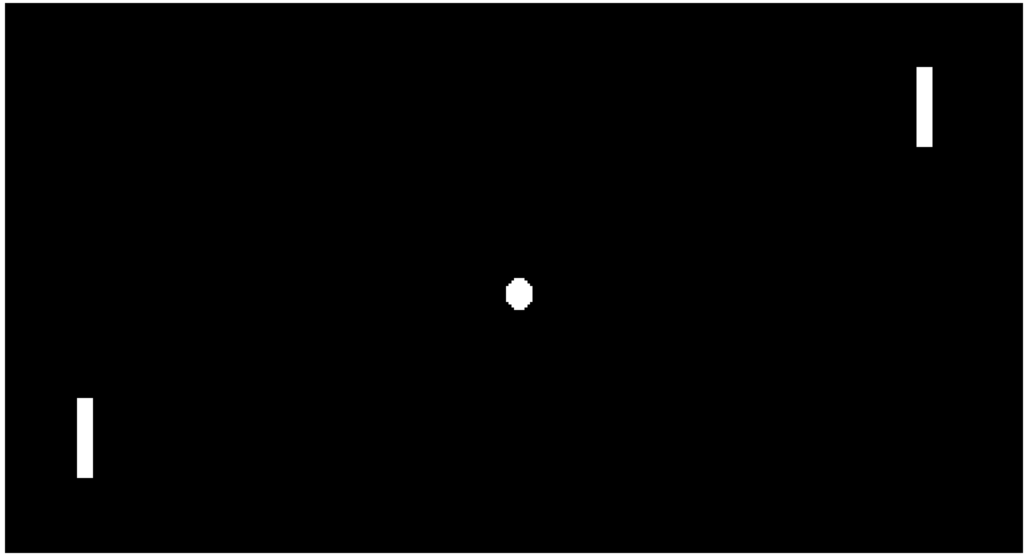
<u>REINFORCE algorithm (Monte-Carlo Policy Gradient)</u>

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

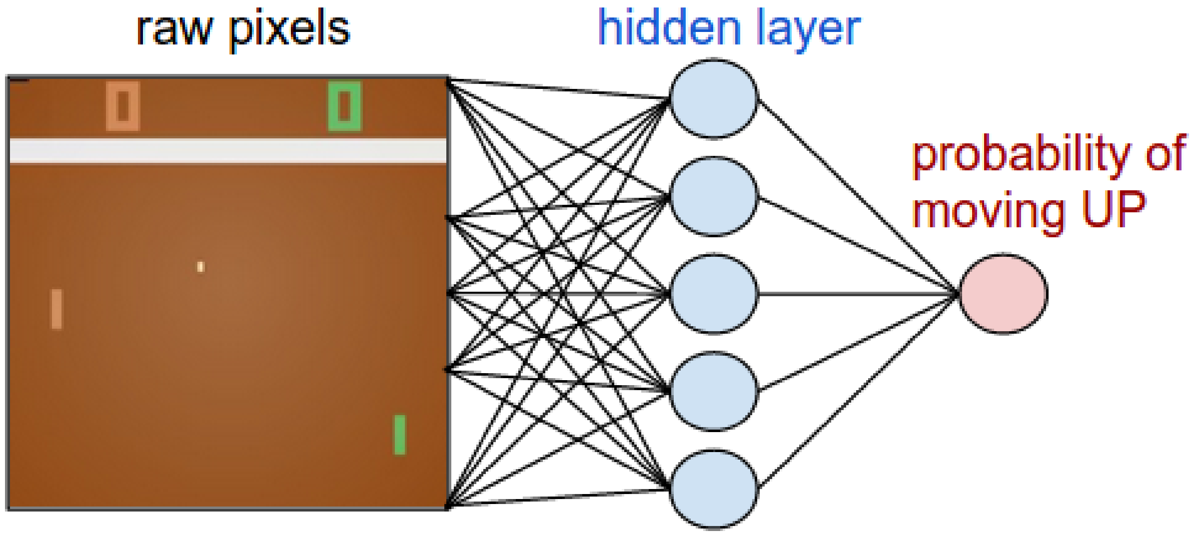


Slide adapted from S. Levine

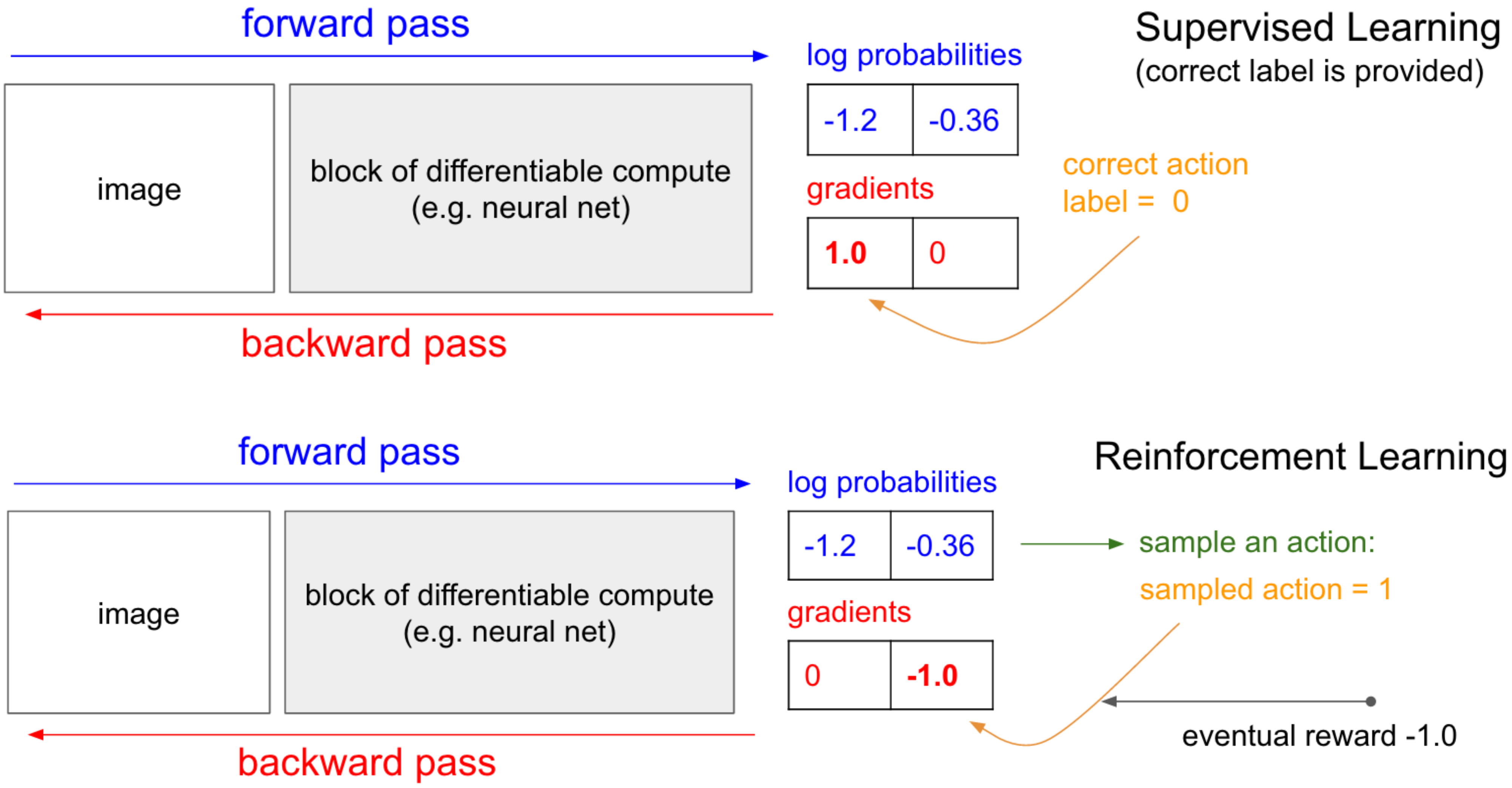
An Example of Pong



Policy network

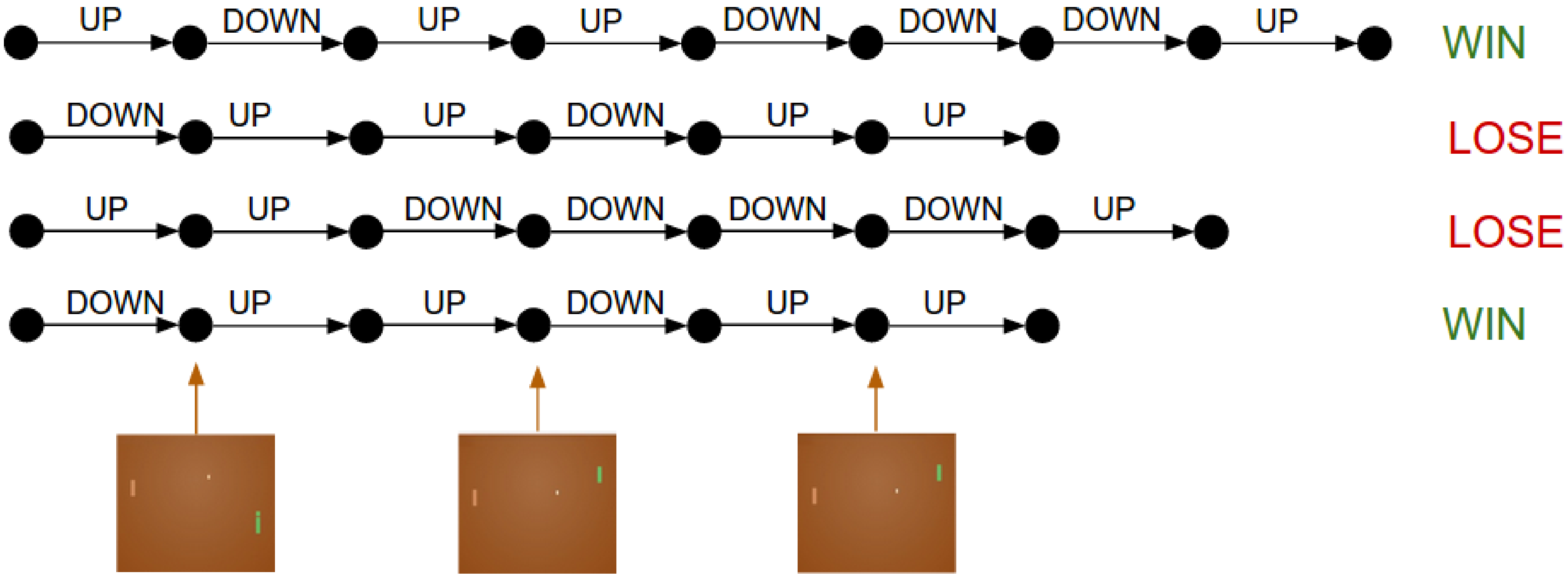


Supervised Learning vs. Reinforcement Learning



Reinforcement Learning: Optimize the Policy Based on Trial and Error

- Rollout the policy and collect episodes...



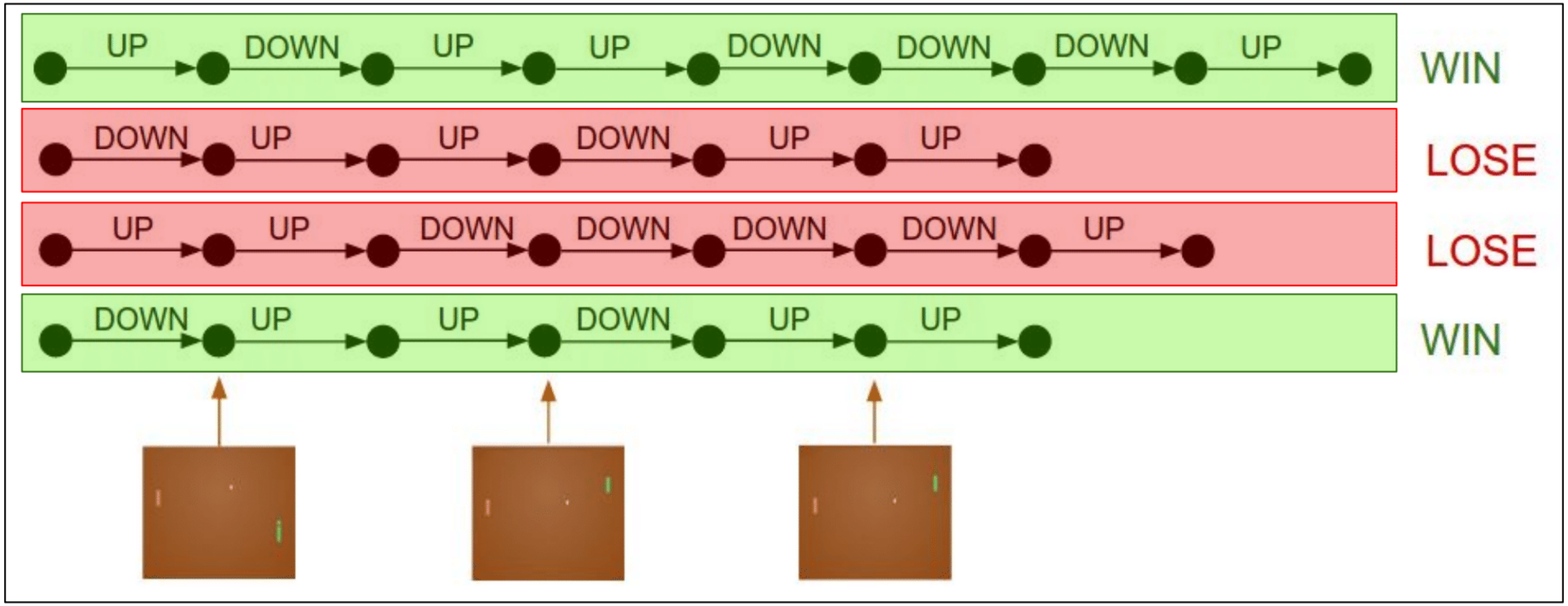
Material Source: https://karpathy.github.io/2016/05/31/rl/

Pretend every action we took here was the correct label.

maximize: $\log p(y_i \mid x_i)$

Pretend every action we took here was the wrong label.

maximize: $(-1) * \log p(y_i \mid x_i)$



We don't need any action labels!

$$\nabla_\theta U(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\alpha_t^{(i)} \mid s_t^{(i)}) R(\tau^{(i)})$$

Slide adapted from K. Fragkiadaki
Material Source: https://karpathy.github.io/2016/05/31/rl/

Wait! Policy Gradient Also Has Problems...

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Wait! Policy Gradient Also Has Problems...

REINFORCE algorithm:

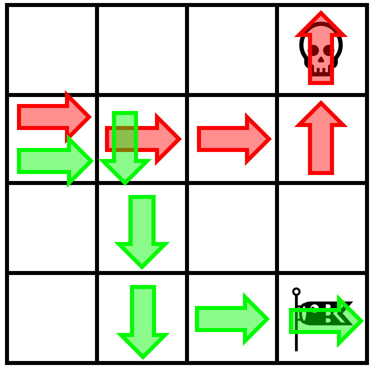
1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \boxed{\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)}$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Gradients of each action in the trajectory
are weighted by the sum of rewards

Wait! Policy Gradient Also Has Problems…

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

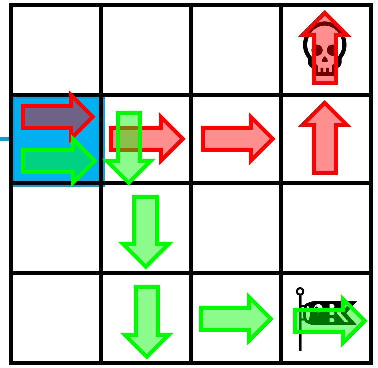


- The gradient estimator is unbiased, but requires a very large number of samples to accurately approximate the true gradient
- When the number of sample is small, the variance is high…

Gradients Have High Variance

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

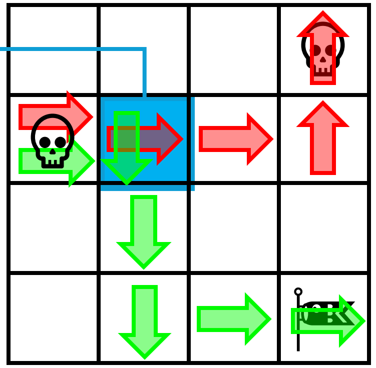


Gradients of the same action has high variance!

Gradients Have High Variance

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

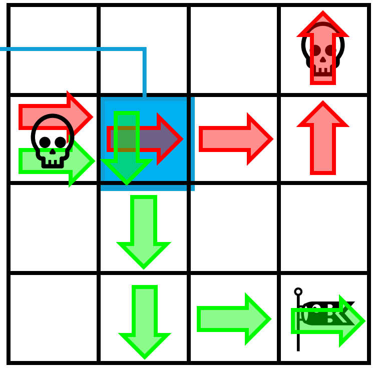


Gradients of current actions forced to account for previous success/failure

Gradients Have High Variance

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



Gradients of current actions forced to account for previous success/failure

Solution:

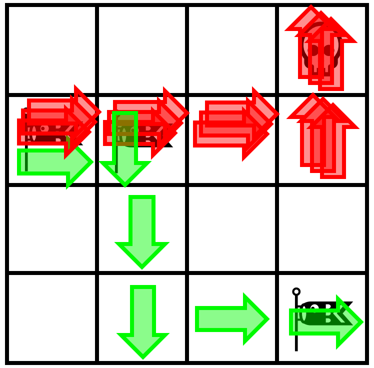
- Causality: let current actions only account for future success/failure

$$\nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t\,|s_t)\right)\left(\sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}, a_{t'})\right)\right]$$

Gradients Have High Variance

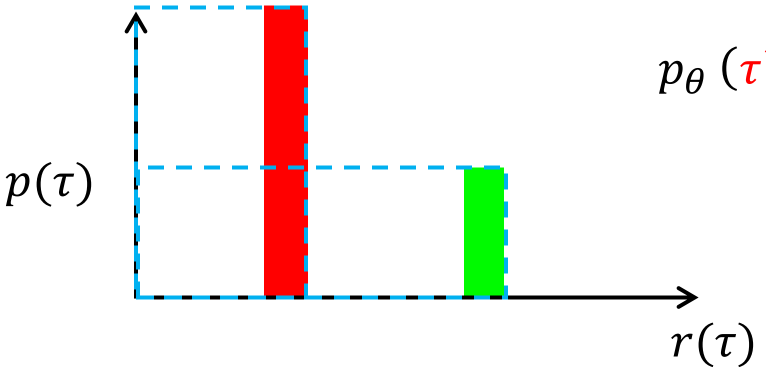
REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



$$\nabla J(\theta) = \int p_\theta(\tau) \nabla \log p_\theta(\tau)\, r(\tau) d\tau$$

$$p_\theta(\tau^r) r(\tau^r) = p_\theta(\tau^g) r(\tau^g)$$



Common sub-optimal trajectory receives same weightings as rare optimal trajectory

A More General Solution to Reduce Variance

- Subtract a baseline $\boldsymbol{b}$

$$\nabla J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla \log p_\theta(\tau)\,[r(\tau) - b]$$

- The gradients don't change

$$E[\nabla \log p_\theta(\tau) b] = \int p_\theta(\tau) \nabla \log p_\theta(\tau)\, b d\tau = \int \nabla p_\theta(\tau) b d\tau = b\nabla \int p_\theta(\tau) b d\tau = 0$$

- Subtracting a baseline is unbiased in expectation!

What about variance?

$$\text{Var}[x] = E[x^2] - E[x]^2$$

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]$$

$$\text{Var} = E_{\tau \sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau)(r(\tau) - b))^2] - \underbrace{E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]^2}$$

this bit is just $E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$

(baselines are unbiased in expectation)

What about variance?

$$\text{Var}[x] = E[x^2] - E[x]^2$$

$$\nabla_\theta J(\theta) = E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]$$

$$\text{Var} = E_{\tau\sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau)(r(\tau) - b))^2] - \underbrace{E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]^2}$$

this bit is just $E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$

(baselines are unbiased in expectation)

$$\frac{d\text{Var}}{db} = \frac{d}{db} E[g(\tau)^2(r(\tau) - b)^2] = \frac{d}{db}\left(\cancel{E[g(\tau)^2 r(\tau)^2]} - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2]\right)$$

$$= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$$

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]} \longleftarrow$$

This is just expected reward, but weighted by gradient magnitudes!

Slide credit S. Levine

Other Options of Baselines?

- Constant baseline $b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$

$$\nabla J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla \log p_\theta(\tau)\,[r(\tau) - b]$$

- Time-dependent baseline $b_t = \frac{1}{N}\sum_{i=1}^{N} r_t(\tau)$

$$\nabla J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla \log p_\theta(a_t|s_t)\,[r_t(\tau_t) - b_t]$$

- State-dependent baseline $b_s = V^\pi(s)$

$$\nabla J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla \log p_\theta(a_t|s_t)\,\left[r_t(\tau_t) - b_{s_t}\right]$$

Slide adapted from K. Fragkiadaki

Off-Policy Policy Gradient with Importance Sampling

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \underbrace{\left(\prod_{t'=1}^{t} \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})} \right)}_{\text{exponential in } T\ldots} \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

let's write the objective a bit differently...

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

$(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \sim \pi_\theta(\mathbf{s}_t, \mathbf{a}_t)$

off-policy policy gradient: $\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \underbrace{\cancel{\frac{\pi_{\theta'}(\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t})}}}_{\text{ignore this part}} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

Slide credit S. Levine

Actor Critic

- Policy Gradient:

$$\nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t \,|s_t)\right)\left(\sum_{t=0}^{T} r(s_t, a_t) - b\right)\right]$$

- Re-write with action value $Q_\pi(s, a)$:

$$\nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t \,|s_t)\right)(Q^\pi(s_t, a_t) - b)\right]$$

Actor Critic

- Re-write with action value $Q_\pi(s,a)$:

$$\nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t \mid s_t)\right)(Q^\pi(s_t, a_t) - b)\right]$$

Let $b = V^\pi(s)$ $\Rightarrow \nabla J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t \mid s_t)\right)\underbrace{(Q^\pi(s_t, a_t) - V^\pi(s))}_{\text{Advantage } A^\pi(s_t, a_t)}\right]$

- Advantage $A^\pi(s_t, a_t)$ measures how much better action a_t is than other actions

- If we learn a value estimator $V_\phi^\pi(s)$ parameterized by ϕ, we obtain:
  - Estimated action value $Q^\pi(s_t, a_t) = R(s_t, a_t) + \gamma V_\phi^\pi(s_{t+1})$
  - Estimated advantage $A^\pi(s_t, a_t) = R(s_t, a_t) + \gamma V_\phi^\pi(s_{t+1}) - V_\phi^\pi(s_t)$

On-policy Advantage Actor Critic

0. Initialize policy parameters θ and critic parameters ϕ .

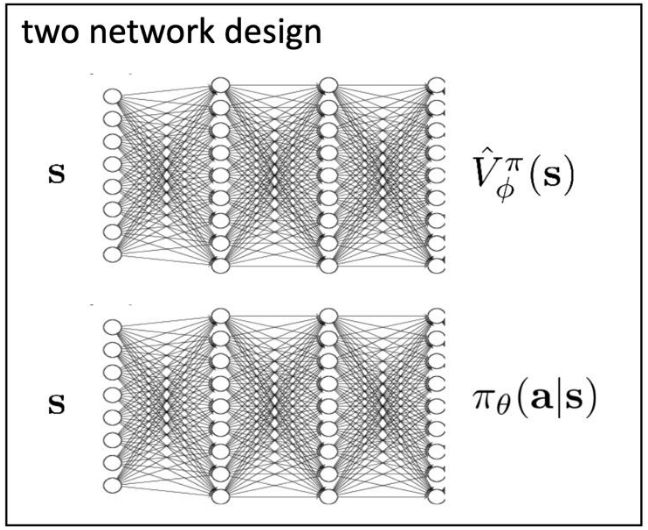
1. Sample trajectories $\{\tau_i = \{s_t^i, a_t^i\}_{t=0}^T\}$ by deploying the current policy $\pi_\theta(a_t \mid s_t)$.

2. Fit value function $V_\phi^\pi(s)$ by MC or TD estimation (update ϕ)

3. Compute action advantages $A^\pi(s_t^i, a_t^i) = R(s_t^i, a_t^i) + \gamma V_\phi^\pi(s_{t+1}^i) - V_\phi^\pi(s_t^i)$

4. $\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha_t^i \mid s_t^i) A^\pi(s_t^i, a_t^i)$

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



Slide credit K. Fragkiadaki

On-policy Advantage Actor Critic

0. Initialize policy parameters θ and critic parameters ϕ.

1. Sample trajectories $\{\tau_i = \{s_t^i, a_t^i\}_{t=0}^{T}\}$ by deploying the current policy $\pi_\theta(a_t \mid s_t)$.

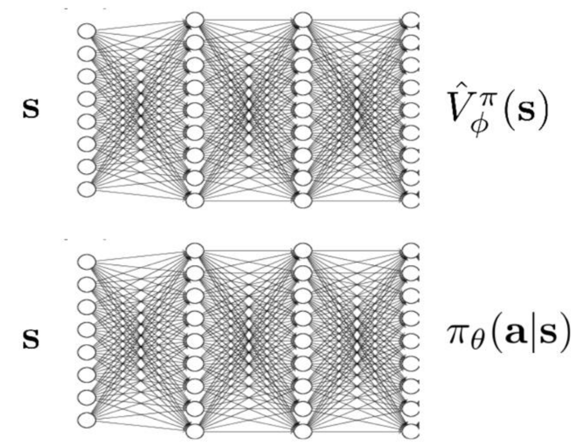
2. Fit value function $V_\phi^\pi(s)$ by MC or TD estimation (update ϕ)

3. Compute action advantages $A^\pi(s_t^i, a_t^i) = R(s_t^i, a_t^i) + \gamma V_\phi^\pi(s_{t+1}^i) - V_\phi^\pi(s_t^i)$

4. $\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha_t^i \mid s_t^i) A^\pi(s_t^i, a_t^i)$

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

two network design



Off-policy Actor Critic with Importance Sampling

- Policy gradient objective:

$$J(\theta) = \sum_{\tau \sim p_\theta(\tau)} \sum_t r(s_t, a_t)$$

- Actor-critic objective:

$$J(\theta) = \sum_{\tau \sim p_\theta(\tau)} A^\pi(s_t, a_t)$$

- Off-policy Actor-critic objective with importance sampling:

$$J(\theta) = \sum_{\tau \sim p_{\theta_{old}}(\tau)} \boxed{\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}} A^\pi(s_t, a_t)$$

Often ends up in huge change in the policy