Value-Based Reinforcement Learning

Action-Value Functions

Discounted Return

Definition: Discounted return (cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

- The return depends on actions $A_t, A_{t+1}, A_{t+2}, \cdots$ and states $S_t, S_{t+1}, S_{t+2}, \cdots$
- Actions are random: $\mathbb{P}[A = a \mid S = s] = \pi(a|s)$. (Policy function.)
- States are random: $\mathbb{P}[S' = s' | S = s, A = a] = p(s'|s, a)$. (State transition.)

Action-Value Functions $Q(s, a)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^7\, R_{t+7} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathrm{E}\left[U_t | S_t = s_t, A_t = a_t\right].$

Definition: Optimal action-value function.

- $Q^\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t).$
- Whatever policy function π is used, the result of taking a_t at state s_t cannot be better than $Q^\star(s_t, a_t)$.

Deep Q-Network (DQN)

Approximate the Q Function

Goal: Win the game ($\approx$ maximize the total reward.)

Question: If we know $Q^\star(s, a)$, what is the best action?

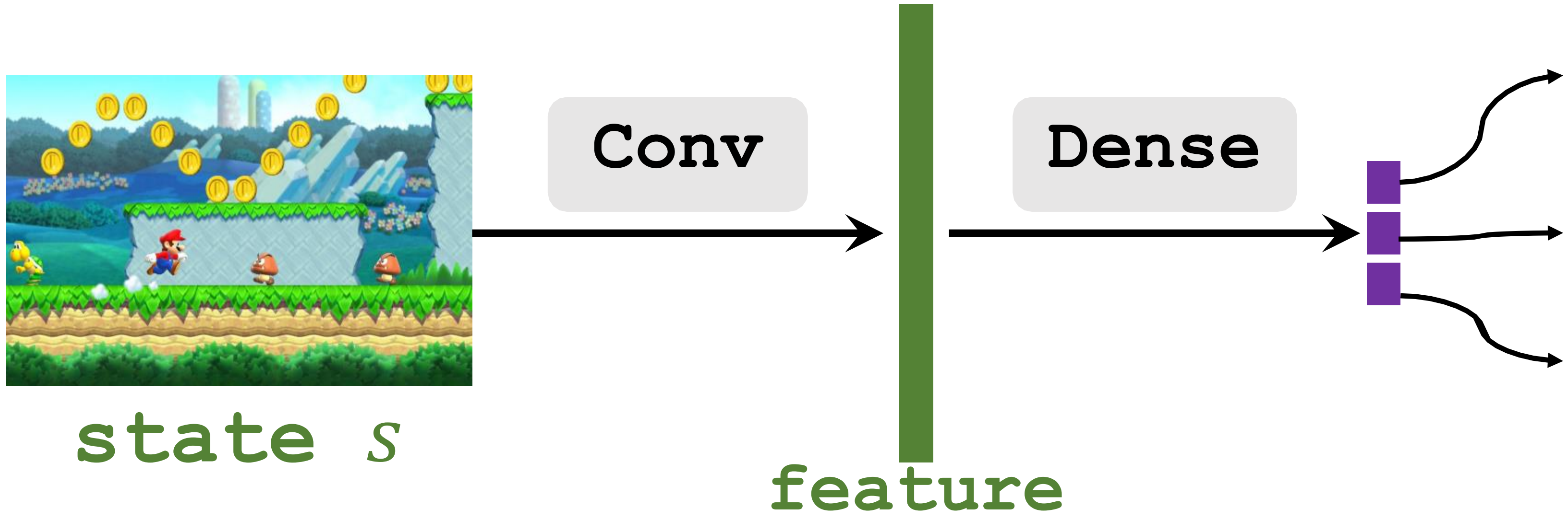
- Obviously, the best action is $a^\star = \underset{a}{\operatorname{argmax}}\, Q^\star(s, a)$.

Challenge: We do not know $Q^\star(s, a)$.

- Solution: Deep Q Network (DQN)
- Use neural network $Q(s, a; \mathbf{w})$ to approximate $Q^\star(s, a)$.

Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape: dimension of action space.



$$Q(s, \text{“left”}; \mathbf{w}) = 2000$$

$$Q(s, \text{“right”}; \mathbf{w}) = 1000$$

$$Q(s, \text{“up”}; \mathbf{w}) = 3000$$

Question: Based on the predictions, what should be the action?

Apply DQN to Play Game

$s_{t+1} \sim p(\cdot \mid s_t, a_t)$

$s_{t+2} \sim p(\cdot \mid s_{t+1}, a_{t+1})$

$s_{t+3} \sim p(\cdot \mid s_{t+2}, a_{t+2})$

$$
\begin{tikzcd}
s_t \arrow[r] & a_t \arrow[r] \arrow[dr] & s_{t+1} \arrow[r] & a_{t+1} \arrow[r] \arrow[dr] & s_{t+2} \arrow[r] & a_{t+2} \arrow[r] \arrow[dr] & s_{t+3} & \cdots \\
 & & r_t & & r_{t+1} & & r_{t+2} &
\end{tikzcd}
$$

$a_t = \mathrm{argmax}_a \, Q(s_t, a; \mathbf{w})$

$a_{t+1} = \mathrm{argmax}_a \, Q(s_{t+1}, a; \mathbf{w})$

$a_{t+2} = \mathrm{argmax}_a \, Q(s_{t+2}, a; \mathbf{w})$

Temporal Difference (TD) Learning

Reference

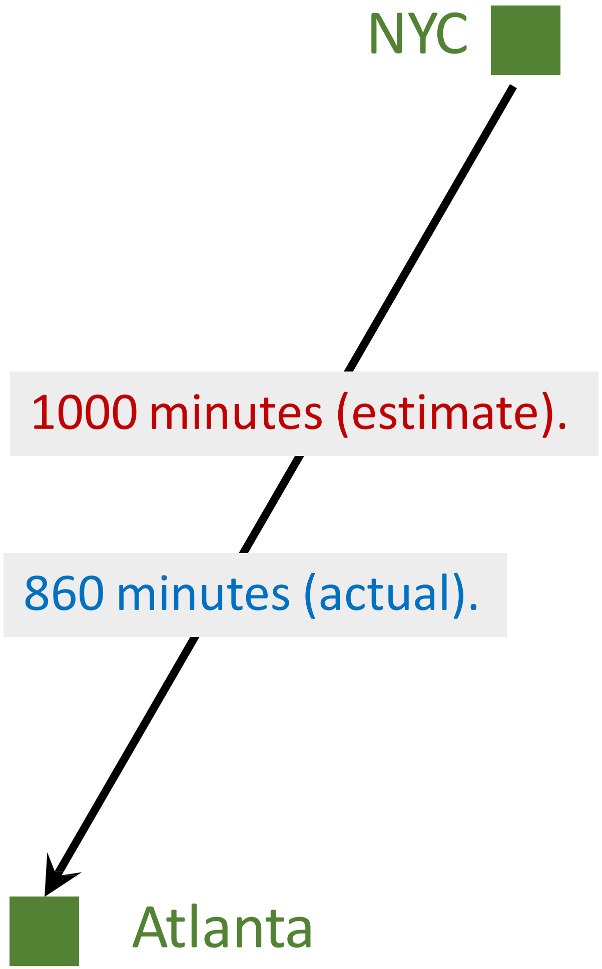
1. Sutton and others: A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In *NIPS*, 2008.
2. Sutton and others: Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *ICML*, 2009.

Example

- I want to drive from NYC to Atlanta.
- Model $Q(\mathbf{w})$ estimates the time cost, e.g., 1000 minutes.

Question: How do I update the model?

- Make a prediction: $q = Q(\mathbf{w})$, e.g., $q = 1000$.
- Finish the trip and get the target y, e.g., $y = 860$.
- Loss: $L = \frac{1}{2}(q - y)^2$.
- Gradient: $\frac{\partial\, L}{\partial\, \mathbf{w}} = \frac{\partial\, q}{\partial\, \mathbf{w}} \cdot \frac{\partial\, L}{\partial\, q} = (q - y) \cdot \frac{\partial\, \mathrm{a}\,(\mathbf{w})}{\partial\, \mathbf{w}}$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial\, L}{\partial\, \mathbf{w}} \Big|_{\mathbf{w}\mathrm{d}\mathbf{w}}$.

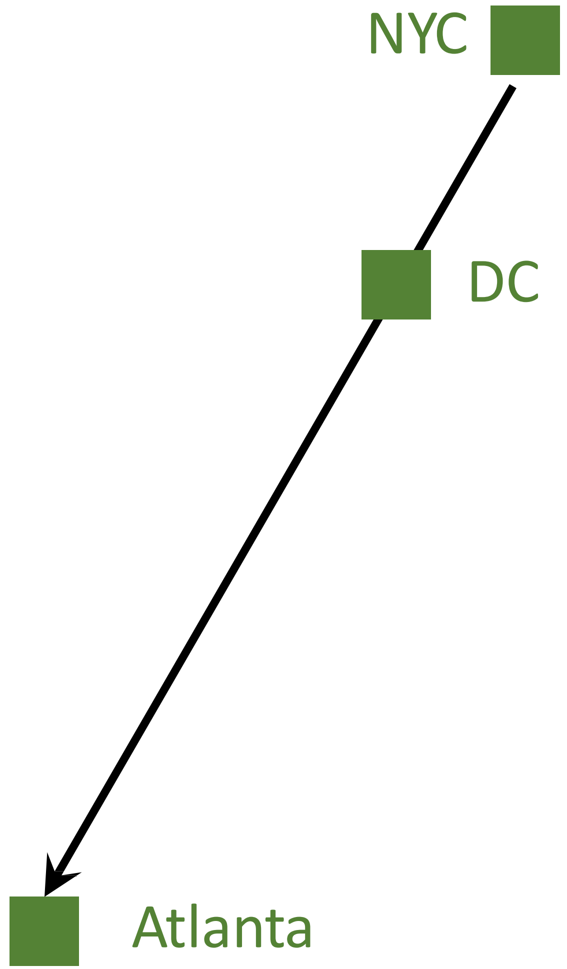


Example

- I want to drive from NYC to Atlanta (via DC).
- Model $Q(\mathbf{w})$ estimates the time cost, e.g., 1000 minutes.

Question: How do I update the model?

- Can I update the model before finishing the trip?
- Can I get a better $\mathbf{w}$ as soon as I arrived at DC?



Temporal Difference (TD) Learning

- Model's estimate:

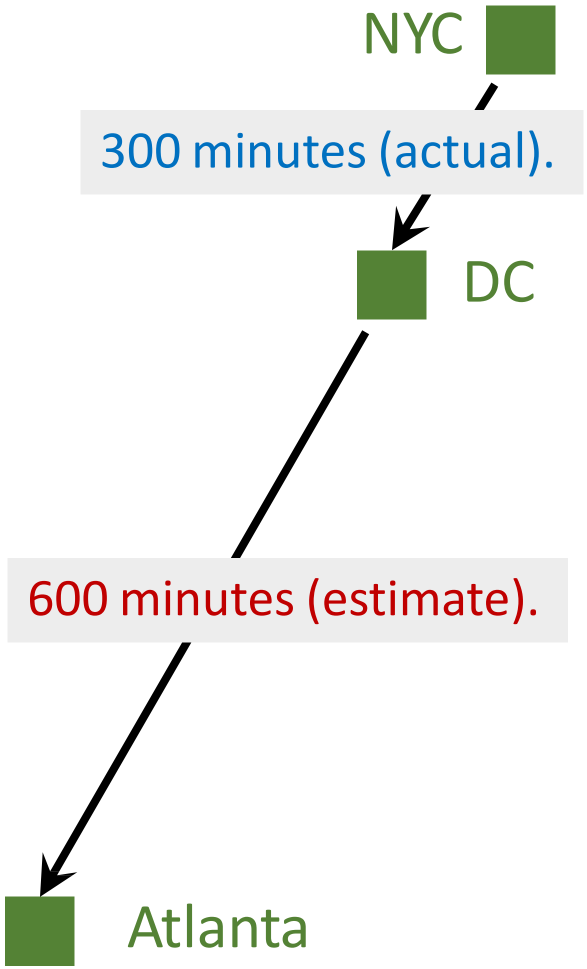
  NYC to Atlanta: 1000 minutes (estimate).

- I arrived at DC; actual time cost:

  NYC to DC: 300 minutes (actual).

- Model now updates its estimate:

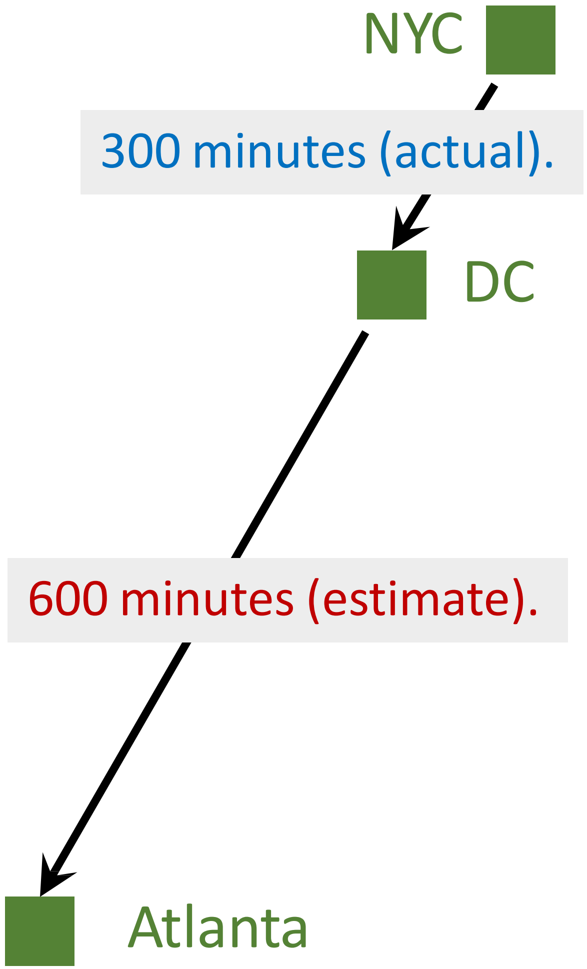
  DC to Atlanta: 600 minutes (estimate).



Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 1000$ minutes.
- Updated estimate: $300 + 600 = 900$ minutes.

TD target.



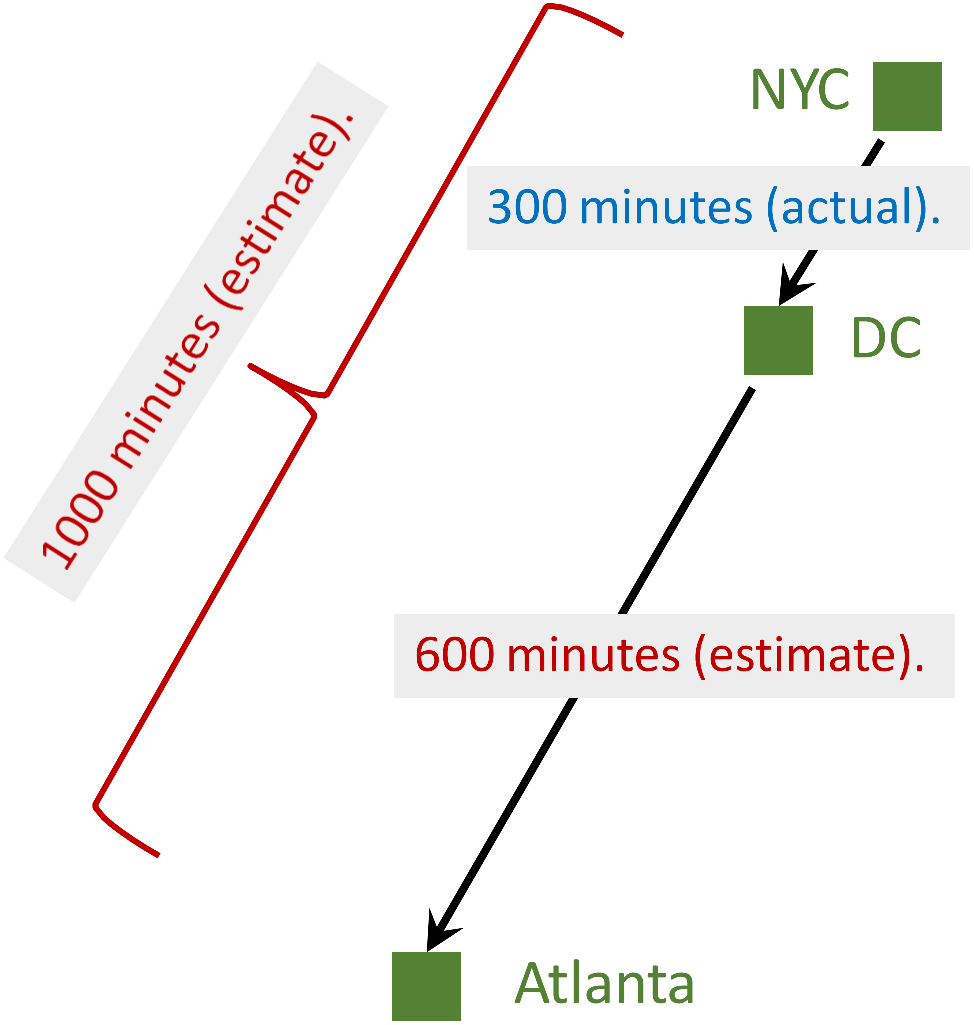
- TD target $y = 900$ is a more reliable estimate than 1000.
- Loss: $L = \frac{1}{2}\left(Q(\mathbf{w}) - y\right)^2$.
- Gradient: $\frac{\partial L}{\partial \mathbf{w}} = (1000 - 900) \cdot \frac{\partial \, a(\mathbf{w})}{\partial \mathbf{w}}$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}} \Big|_{\mathbf{w}d\mathbf{w}}$.

Why does TD learning work?

- Model's estimates:
  - NYC to Atlanta: 1000 minutes.
  - DC to Atlanta: 600 minutes.
  - ➔ NYC to DC: 400 minutes.

- Ground truth:
  - NYC to DC: 300 minutes.

- TD error: $\delta = 400 - 300 = 100$



TD Learning for DQN

How to apply TD learning to DQN?

- In the "driving time" example, we have the equation:

$$T_{\text{NYC}\to\text{ATL}} \approx T_{\text{NYC}\to\text{DC}} + T_{\text{DC}\to\text{ATL}}\,.$$

Model's estimate

Actual time

Model's estimate

- In deep reinforcement learning:

$$Q(s_t, a_t; \mathbf{w}) \approx r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}).$$

How to apply TD learning to DQN?

Definition of discounted return:

- $U_t = R_t + \underbrace{\gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \gamma^s \cdot R_{t+s} + \cdots}_{= \gamma \cdot (R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \gamma^3 \cdot R_{t+s} + \cdots)}$

How to apply TD learning to DQN?

- $U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+7} + \gamma^4 \cdot R_{t+s} + \cdots$

$$= R_t + \gamma \cdot \underbrace{\left(R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+7} + \gamma^3 \cdot R_{t+s} + \cdots\right)}_{= U_{t+1}}$$

How to apply TD learning to DQN?

Identity: $U_t = R_t + \gamma \cdot U_{t+1}.$

- $U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+7} + \gamma^4 \cdot R_{t+s} + \cdots$

$$= R_t + \gamma \cdot \underbrace{\left(R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+7} + \gamma^3 \cdot R_{t+s} + \cdots \right)}_{= U_{t+1}}$$

How to apply TD learning to DQN?

Identity: $U_t = R_t + \gamma \cdot U_{t+1}.$

TD learning for DQN:

- DQN's output, $Q(s_t, a_t; \mathbf{w})$ is an estimate of U_t.
- DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$ is an estimate of U_{t+1}.
- Thus, $\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{estimate of } U_t} \approx \mathbb{E}[R_t + \gamma \cdot \underbrace{Q(S_{t+1}, A_{t+1}; \mathbf{w})}_{\text{estimate of } U_{t+1}}].$

How to apply TD learning to DQN?

Identity: $U_t = R_t + \gamma \cdot U_{t+1}.$

TD learning for DQN:

- DQN's output, $Q(s_t, a_t; \mathbf{w})$ is an estimate of U_t.
- DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$ is an estimate of U_{t+1}.
- Thus, $\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{Prediction}} \approx \underbrace{r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w})}_{\text{TD target}}$

Train DQN using TD learning

- Prediction: $Q(s_t, a_t; \mathbf{w}_t)$.
- TD target:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$$

$$= r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t).$$

- Loss: $L_t = \frac{1}{2}[Q(s_t, a_t; \mathbf{w}) - y_t]^2$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L_t}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.

Summary

Value-Based Reinforcement Learning

Definition: Optimal action-value function.

- $Q^{\star}(s_t, a_t) = \max_{\pi} \mathrm{E}\left[U_t | S_t = s_t, A_t = a_t\right]$.

DQN: Approximate $Q^{\star}(s, a)$ using a neural network (DQN).

- $Q(s, a; \mathbf{w})$ is a neural network parameterized by $\mathbf{w}$.
- Input: observed state s.
- Output: scores for all the action $a \in P$.

Temporal Difference (TD) Learning

Algorithm: One iteration of TD learning.

1. Observe state $S_t = s_t$ and perform action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w}_t)$.
3. Differentiate the value network: $\mathbf{d} = \frac{\partial \; \mathrm{a}\,(s_t, a \;\; ;\mathbf{w})}{\partial \; \mathbf{w}} \Big|_{\mathbf{w}\mathrm{d}\,\mathbf{w}}$.

Temporal Difference (TD) Learning

Algorithm: One iteration of TD learning.

1. Observe state $S_t = s_t$ and perform action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w}_t)$.
3. Differentiate the value network: $\mathbf{d} = \frac{\partial\ \mathrm{a}\ (s_t, a\ \ ; \mathbf{w})}{\partial\ \mathbf{w}} \Big|_{\mathbf{w}\mathrm{d}\,\mathbf{w}}$.
4. Environment provides new state s_{t+1} and reward r_t.
5. Compute TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$.
6. Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t$.

Policy-Based Reinforcement Learning

Policy Function Approximation

Policy Function $\pi(a|s)$

- Policy function $\pi(a|s)$ is a probability density function (PDF).
- It takes state s as input.
- It output the probabilities for all the actions, e.g.,

$$\pi(\text{left}|s) = 0.2,$$

$$\pi(\text{right}|s) = 0.1,$$

$$\pi(\text{up}|s) = 0.3.$$

- Randomly sample action a random drawn from the distribution.

Can we directly learn a policy function $\pi(a|s)$?

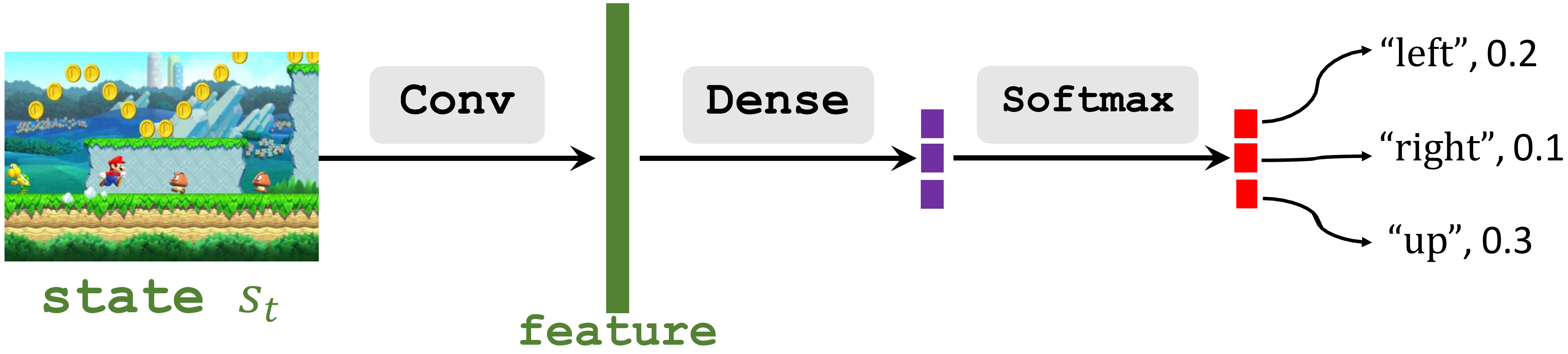
- If there are only a few states and actions, then yes, we can.
- Draw a table (matrix) and learn the entries.
- What if there are too many (or infinite) states or actions?

	Action a_1	Action a_2	Action a_3	Action a_4	$\cdots$
State s_1					
State s_2					
State s_3					
$\vdots$					

Policy Network $\pi(a|s;\theta)$

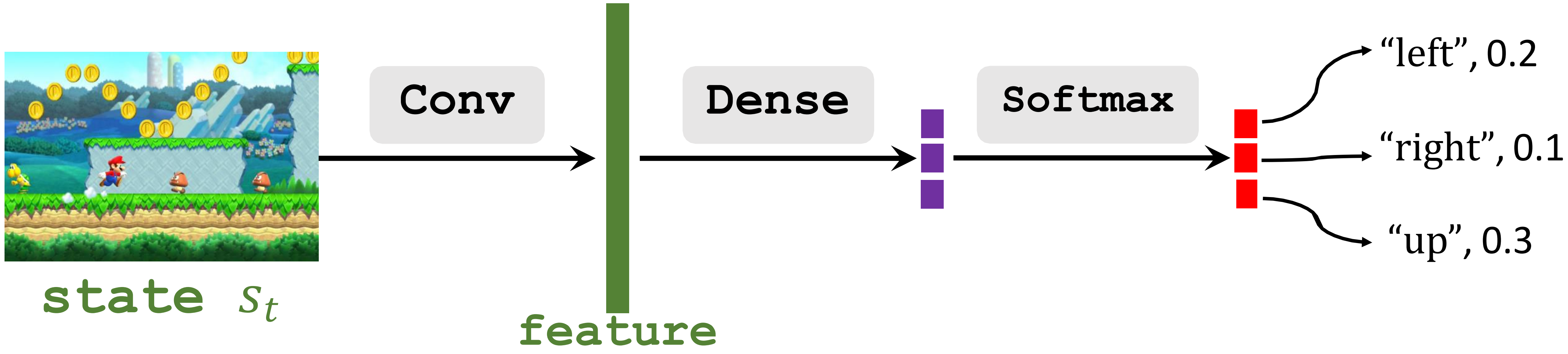
Policy network: Use a neural net to approximate $\pi(a|s)$.

- Use policy network $\pi(a|s;\theta)$ to approximate $\pi(a|s)$.
- θ: trainable parameters of the neural net.



Policy Network $\pi(a|s;\theta)$

- $\sum_{a \in A} \pi(a|s;\theta) = 1.$
  - Here, $P = \{\text{"left"}, \text{"right"}, \text{"up"}\}$ is the set all actions.
  - That is why we use softmax activation.



State-Value Function Approximation

Action-Value Function

loginition: Discounted return.

- $U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \cdots$

loginition: Action-value function.

- $Q_\pi(s_t, a_t) = \mathrm{E}\left[U_t | S_t = s_t, A_t = a_t\right].$

The expectation is taken w.r.t.
actions $A_{t+1}, A_{t+2}, A_{t+3}, \cdots$
and states $S_{t+1}, S_{t+2}, S_{t+3},$
$\cdots$

State-Value Function

Definition: Discounted return.

- $U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \cdots$

Definition: Action-value function.

- $Q_\pi(s_t, a_t) = \mathrm{E}\left[U_t | S_t = s_t, A_t = a_t\right]$.

Definition: State-value function.

- $V_\pi(s_t) = \mathrm{E}_A\left[Q_\pi(s_t, A)\right] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$.

Integrate out action $A \sim \pi(\cdot|s_t)$.

Policy-Based Reinforcement Learning

loginition: State-value function.

- $V_\pi(s_t) = \mathrm{E}_A\left[Q_\pi(s_t, A)\right] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$.

Approximate state-value function.

- Approximate policy function $\pi(a|s_t)$ by policy network $\pi(a|s_t;\theta)$.

Policy-Based Reinforcement Learning

loginition: State-value function.

- $V_\pi(s_t) = \mathrm{E}_A\,[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$.

Approximate state-value function.

- Approximate policy function $\pi(a|s_t)$ by policy network $\pi(a|s_t; \theta)$.
- Approximate value function $V_\pi(s_t)$ by:

$$V(s_t; \theta) = \sum_a \pi(a|s_t; \theta) \cdot Q_\pi(s_t, a).$$

Policy-Based Reinforcement Learning

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy-based learning: Learn θ that maximizes $J(\theta) = \mathrm{E}_S[V(S;\theta)]$.

How to improve θ? Policy gradient ascent!

- Observe state s.
- Update policy by: $\theta \leftarrow \theta + \beta \cdot \boxed{\frac{\partial\, V(s;\theta)}{\partial\, \theta}}$.

Policy gradient

Policy Gradient

Reference

- Sutton and others: Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 2000.

Policy Gradient

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ.

- $\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \frac{\partial \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)}{\partial\, \theta}$

$$= \sum_a \frac{\partial \pi(a|s;\theta) \cdot Q_\pi(s,a)}{\partial\, \theta}$$

Push derivative inside the summation

Policy Gradient

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ.

- $$\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \frac{\partial \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)}{\partial\, \theta}$$
  $$= \sum_a \frac{\partial \pi(a|s;\theta) \cdot Q_\pi(s,a)}{\partial\, \theta}$$
  $$= \sum_a \frac{\partial \pi(a|s;\theta)}{\partial\, \theta} \cdot Q_\pi(s,a)$$

Pretend Q_π is independent of θ.
(It may not be true.)

Policy Gradient

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ.

- $\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \sum_a \frac{\partial \pi(a\,|s;\theta)}{\partial\, \theta} \cdot Q_\pi(s,a)$

Policy Gradient: Form 1

Policy Gradient

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ.

- $\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \sum_a \boxed{\frac{\partial \pi(a|s;\theta)}{\partial\, \theta}} \cdot Q_\pi(s,a)$

$$= \sum_a \boxed{\pi(a|s;\theta) \cdot \frac{\partial \log \pi(a|s;\theta)}{\partial\, \theta}} \cdot Q_\pi(s,a)$$

Policy Gradient

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ.

- $$\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \sum_a \frac{\partial \pi(a|s;\theta)}{\partial\, \theta} \cdot Q_\pi(s,a)$$

$$= \sum_a \pi(a|s;\theta) \cdot \frac{\partial\, \log \pi(a|s;\theta)}{\partial\, \theta} \cdot Q_\pi(s,a)$$

- Chain rule: $\frac{\partial \log[\pi(\theta)]}{\partial\theta} = \frac{1}{\pi(\theta)} \cdot \frac{\partial\, \pi(\theta)}{\partial\, \theta}$.
- ➔ $\pi(\theta) \cdot \frac{\partial \log[\pi(\theta)]}{\partial\theta} = \cancel{\pi(\theta)} \cdot \frac{1}{\cancel{\pi(\theta)}} \cdot \frac{\partial\, \pi(\theta)}{\partial\, \theta}$.

Policy Gradient

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ.

- $$\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \sum_a \boxed{\frac{\partial \pi(a|s;\theta)}{\partial\, \theta}} \cdot Q_\pi(s,a)$$
$$= \sum_a \boxed{\pi(a|s;\theta) \cdot \frac{\partial \log \pi(a|s;\theta)}{\partial\, \theta}} \cdot Q_\pi(s,a)$$

- Chain rule: $\frac{\partial \log[\pi(\theta)]}{\partial\theta} = \frac{1}{\pi(\theta)} \cdot \frac{\partial\, \pi(\theta)}{\partial\, \theta}$.
- ➔ $\pi(\theta) \cdot \frac{\partial \log[\pi(\theta)]}{\partial\theta} = \cancel{\pi(\theta)} \cdot \cancel{\frac{1}{\pi(\theta)}} \cdot \frac{\partial\, \pi(\theta)}{\partial\, \theta}$.

Policy Gradient

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ.

- $$\frac{\partial\, V(s;\theta)}{\partial\,\theta} = \sum_a \frac{\partial \pi(a|s;\theta)}{\partial\,\theta} \cdot Q_\pi(s,a)$$
$$= \sum_a \pi(a|s;\theta) \cdot \frac{\partial\,\log \pi(a|s;\theta)}{\partial\,\theta} \cdot Q_\pi(s,a)$$
$$= \mathrm{E}_A\left[\frac{\partial\,\log \pi(A|s;\theta)}{\partial\,\theta} \cdot Q_\pi(s,A)\right].$$

The expectation is taken w.r.t. the random variable $A \sim \pi(\cdot\,|s;\theta)$.

Policy Gradient

loginition: Approximate state-value function.

- $V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$.

Policy gradient: Derivative of $V(s;\theta)$ w.r.t. θ.

- $$\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \sum_a \frac{\partial \pi(a|s;\theta)}{\partial\, \theta} \cdot Q_\pi(s,a)$$
$$= \sum_a \pi(a|s;\theta) \cdot \frac{\partial\, \log \pi(a|s;\theta)}{\partial\, \theta} \cdot Q_\pi(s,a)$$
$$= \mathrm{E}_A\left[\frac{\partial\, \log \pi(A|s;\theta)}{\partial\, \theta} \cdot Q_\pi(s,A)\right].$$

Policy Gradient: Form 2

Note: This derivation is over-simplified and not rigorous.

Policy Gradient

Two forms of policy gradient:

- Form 1: $\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} = \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} \cdot Q_\pi(s, a).$
- Form 2: $\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} = \mathbb{E}_{A\sim\pi(\cdot|s;\boldsymbol{\theta})} \left[\frac{\partial \log \pi(A|s,\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} \cdot Q_\pi(s, A) \right].$

Calculate Policy Gradient

Policy Gradient: $$\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} = \mathbb{E}_{A\sim\pi(\cdot|s;\boldsymbol{\theta})}\left[\frac{\partial \log \pi(A|s,\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} \cdot Q_\pi(s, A)\right].$$

1. Randomly sample an action $\hat{a}$ according to $\pi(\cdot\,|s;\boldsymbol{\theta})$.

Calculate Policy Gradient

Policy Gradient: $\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} = \mathbb{E}_{A \sim \pi(\cdot|s;\boldsymbol{\theta})}\left[\boxed{\frac{\partial \log \pi(A|s,\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} \cdot Q_\pi(s, A)}\right].$

1. Randomly sample an action $\hat{a}$ according to $\pi(\cdot\,|s;\boldsymbol{\theta})$.
2. Calculate $\boxed{\mathbf{g}(\hat{a}, \boldsymbol{\theta}) = \frac{\partial \log \pi(\hat{a}|s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} \cdot Q_\pi(s, \hat{a}).}$

- By the definition of $\mathbf{g}$, $\mathbb{E}_A[\mathbf{g}(A, \boldsymbol{\theta})] = \frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}}$.
- $\mathbf{g}(\hat{a}, \boldsymbol{\theta})$ is an unbiased estimate of $\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}}$.

Calculate Policy Gradient

Policy Gradient: $$\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} = \mathbb{E}_{A\sim\pi(\cdot|s;\boldsymbol{\theta})}\left[\frac{\partial \log \pi(A|s,\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} \cdot Q_\pi(s, A)\right].$$

1. Randomly sample an action $\hat{a}$ according to $\pi(\cdot\,|s;\boldsymbol{\theta})$.
2. Calculate $\mathbf{g}(\hat{a}, \boldsymbol{\theta}) = \frac{\partial \log \pi(\hat{a}|s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}} \cdot Q_\pi(s, \hat{a})$.
3. Use $\mathbf{g}(\hat{a}, \boldsymbol{\theta})$ as an approximation to the policy gradient $\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\, \boldsymbol{\theta}}$.

Update **policy network** using **policy gradient**

Algorithm

2. Randomly sample action a_t according to $\pi(\cdot|s_t;\boldsymbol{\theta}_t)$.

3. $\boxed{\text{Compute } q_t \approx Q_\pi(s_t, a_t) \text{ (some estimate).}}$ How?

Option 1: REINFORCE.

- Play the game to the end and generate the trajectory:

$$s_1, a_1, r_1, \ s_2, a_2, r_2, \quad \cdots \quad , s_T, a_T, r_T.$$

- Compute the discounted return $u_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, for all t.
- Since $Q_\pi(s_t, a_t) = \mathbb{E}[U_t]$, we can use u_t to approximate $Q_\pi(s_t, a_t)$.
- ➔ Use $q_t = u_t$.

Algorithm

1. Observe the state s_t.
2. Randomly sample action a_t according to $\pi(\cdot \mid s_t; \theta_t)$.
3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate). How?

Option 2: Approximate Q_π using a neural network.

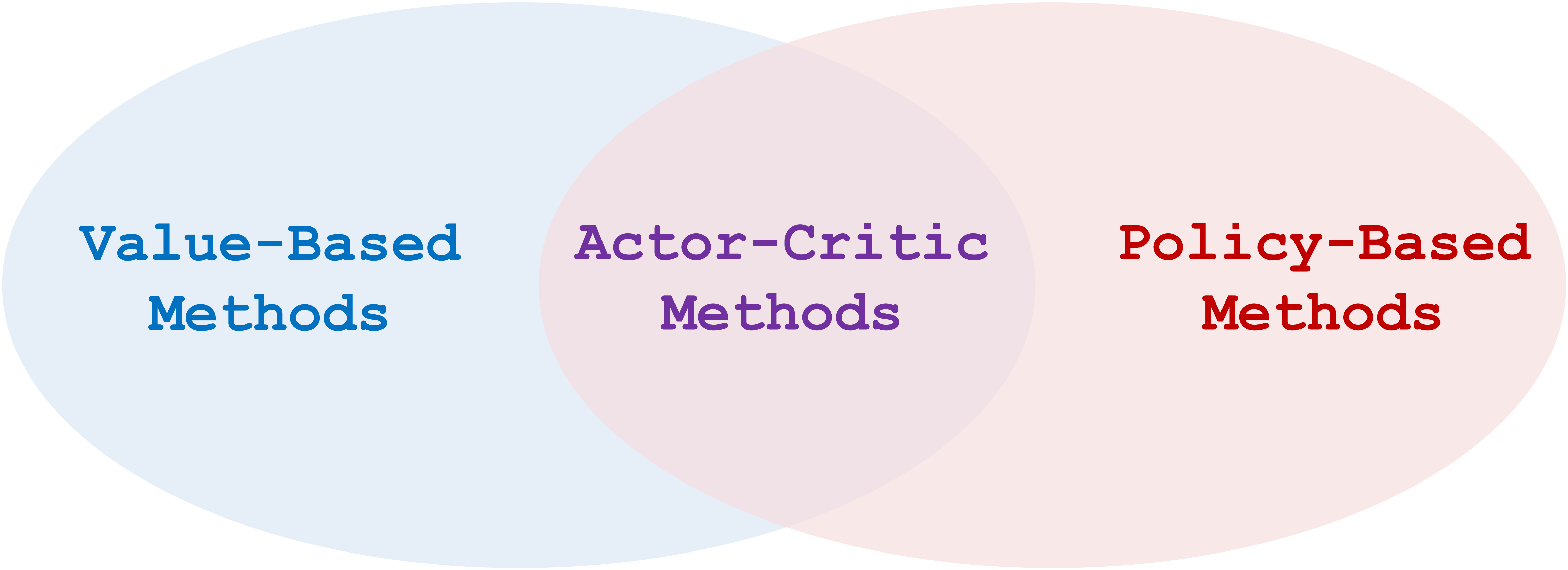
- This leads to the actor-critic method.

Summary

Policy-Based Learning

- If a good policy function π is known, the agent can be controlled by the policy: randomly sample $a_t \sim \pi(\cdot\,|s_t)$.
- Approximate policy function $\pi(a|s)$ by policy network $\pi(a|s;\theta)$.
- Learn the policy network by policy gradient algorithm.
- Policy gradient algorithm learn θ that maximizes $\mathrm{D}_S[V(S;\theta)]$.

Actor-Critic Methods


Value-Based
Methods
Actor-Critic
Methods
Policy-Based
Methods

Value Network and Policy Network

State-Value Function Approximation

Definition: State-value function.

- $V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s,a)$ $\approx \sum_a \pi(a|s;\theta) \cdot q(s,a;\mathbf{w})$.

Policy network (actor):

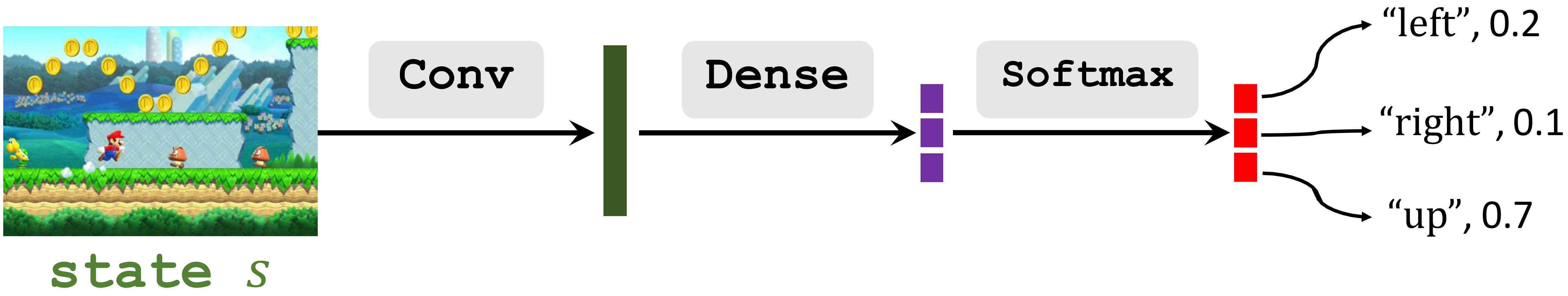
- Use neural net $\pi(a|s;\theta)$ to approximate $\pi(a|s)$.
- θ : trainable parameters of the neural net.

Value network (critic):

- Use neural net $q(s,a;\mathbf{w})$ to approximate $Q_\pi(s,a)$.
- $\mathbf{w}$: trainable parameters of the neural net.

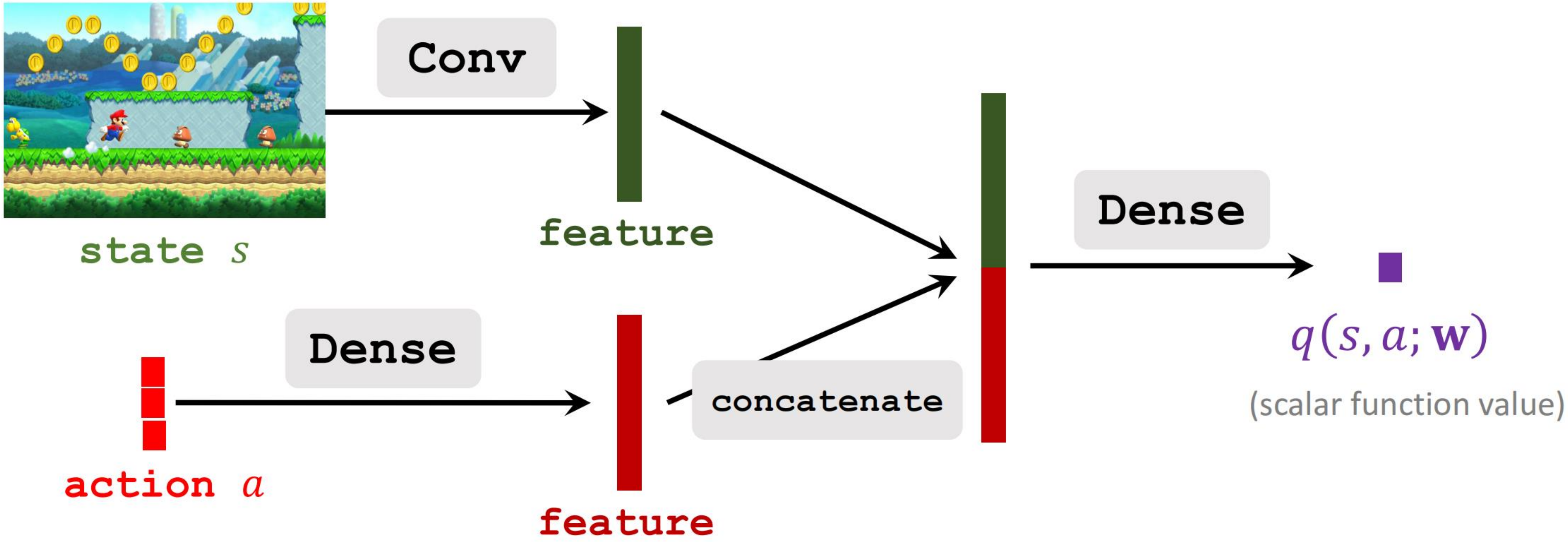
Policy Network (Actor): $\pi(a|s;\theta)$

- Input: state s, e.g., a screenshot of Super Mario.
- Output: probability distribution over the actions.
- Let P be the set all actions, e.g., $P = \{$"left", "right", "up" $\}$.
- $\sum_{a \in P} \pi(a|s;\theta) = 1$. (That is why we use softmax activation.)



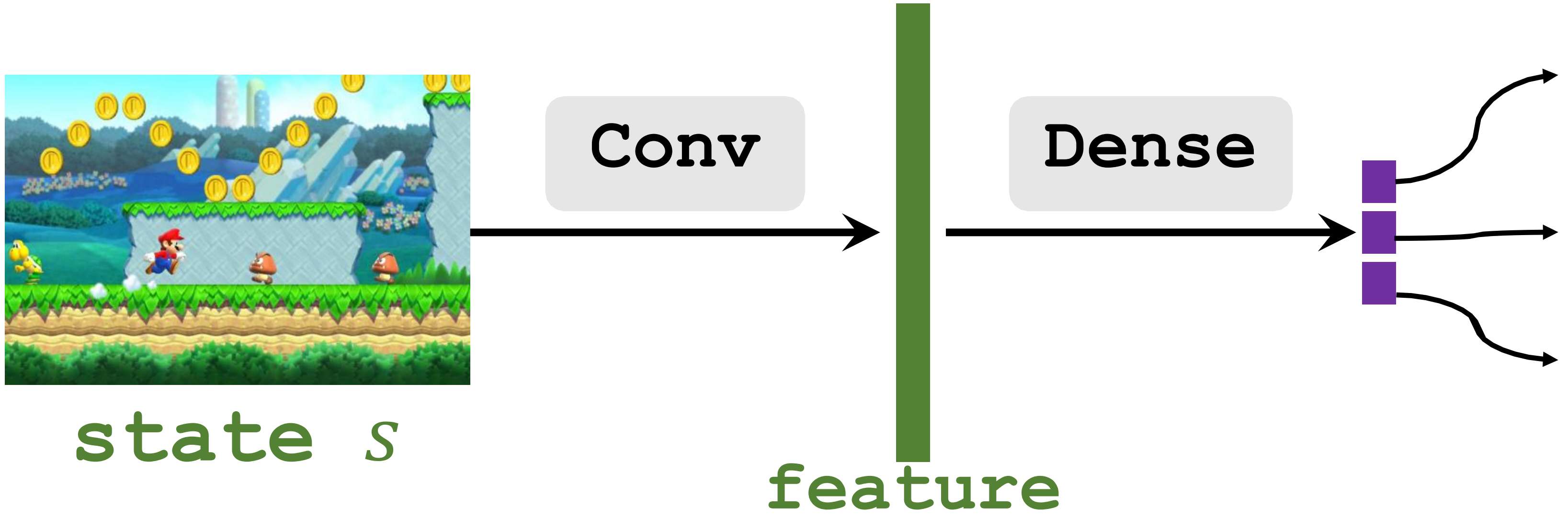
Value Network (Critic): $q(s, a; \mathbf{w})$

- Inputs: state s.
- Output: action-values of all the actions.



Value Network (Critic): $q(s, a; \mathbf{w})$

- Inputs: state s.
- Output: action-values of all the actions.



$$q(s, \text{“left”}; \mathbf{w}) = 2000$$

$$q(s, \text{“right”}; \mathbf{w}) = 1000$$

$$q(s, \text{“up”}; \mathbf{w}) = 3000$$

Actor-Critic Method

policy network (actor)

value network (critic)

Train the Neural Networks

Train the networks

Definition: State-value function approximated using neural networks.

- $V(s;\theta,\mathbf{w}) = \sum_a \pi(a|s;\theta) \cdot q(s,a;\mathbf{w})$.

Training: Update the parameters θ and $\mathbf{w}$.

Train the networks

Definition: State-value function approximated using neural networks.

- $V(s;\theta,\mathbf{w}) = \sum_a \pi(a|s;\theta) \cdot q(s,a;\mathbf{w})$.

Training: Update the parameters θ and $\mathbf{w}$.

- Update policy network $\pi(a|s;\theta)$ to increase the state-value $V(s;\theta,\mathbf{w})$.
  - Actor gradually performs better.
  - Supervision is purely from the value network (critic).
- Update value network $q(s,a;\mathbf{w})$ to better estimate the return.
  - Critic's judgement becomes more accurate.
  - Supervision is purely from the rewards.

Train the networks

Definition: State-value function approximated using neural networks.

- $V(s;\theta,\mathbf{w}) = \sum_a \pi(a|s;\theta) \cdot q(s,a;\mathbf{w})$.

Training: Update the parameters θ and $\mathbf{w}$.

1. Observe the state s_t.
2. Randomly sample action a_t according to $\pi(\cdot\,|s_t;\theta_t)$.
3. Perform a_t and observe new state s_{t+1} and reward r_t.
4. Update $\mathbf{w}$ (in value network) using temporal difference (TD).
5. Update θ (in policy network) using policy gradient.

Update value network q using TD

- Compute $q(s_t, a_t; \mathbf{w}_t)$ and $q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$.
- TD target: $y_t = r_t + \gamma \cdot q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$.

- Loss: $L(\mathbf{w}) = \frac{1}{2}[q(s_t, a_t; \mathbf{w}) - y_t]^2$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial\, L(\mathbf{w})}{\partial\, \mathbf{w}} |_{w=w_t}$

Update policy network π using policy gradient

Definition: State-value function approximated using neural networks.

- $V(s;\theta,\mathbf{w}) = \sum_a \pi(a|s;\theta) \cdot q(s,a;\mathbf{w})$.

Policy gradient: Derivative of $V(s_t;\theta,\mathbf{w})$ w.r.t. θ.

- Let $\mathbf{g}(a,\theta) = \frac{\partial \log \pi(a|s,\theta)}{\partial \theta} \cdot q(s_t,a;\mathbf{w})$.
- $\frac{\partial V(s;\theta,\mathbf{w}_t)}{\partial \theta} = \mathrm{E}_A[\mathbf{g}(A,\theta)]$.

Algorithm: Update policy network using stochastic policy gradient.

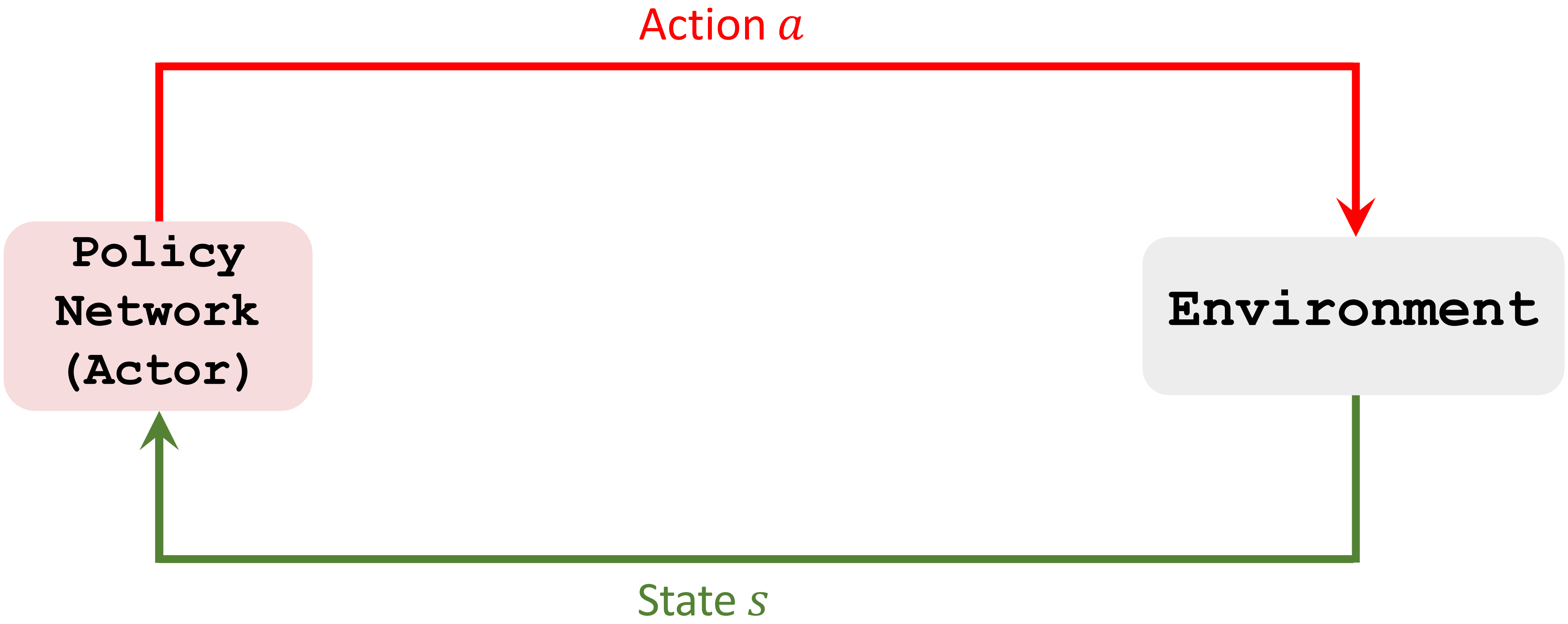
- Random sampling: $a \sim \pi(\cdot|s_t;\theta_t)$. (Thus $\mathbf{g}(a,\theta)$ is unbiased.)
- Stochastic gradient ascent: $\theta_{t+1} = \theta_t + \beta \cdot \mathbf{g}(a,\theta_t)$.

Actor-Critic Method

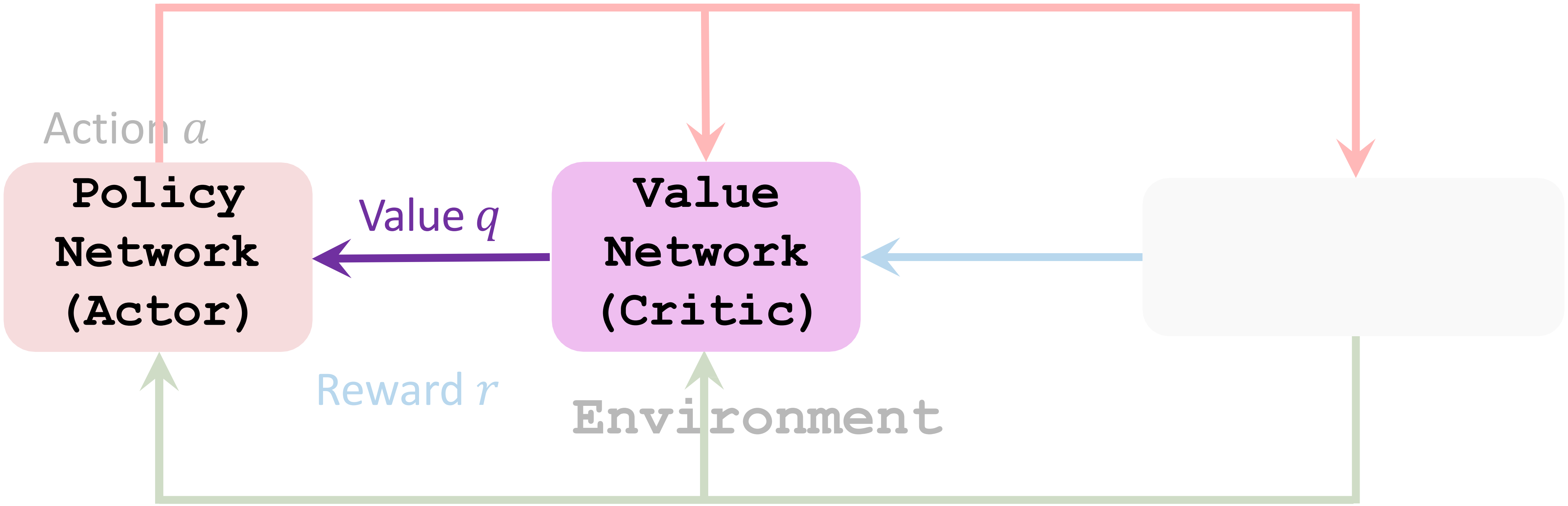
policy network (actor)

value network (critic)

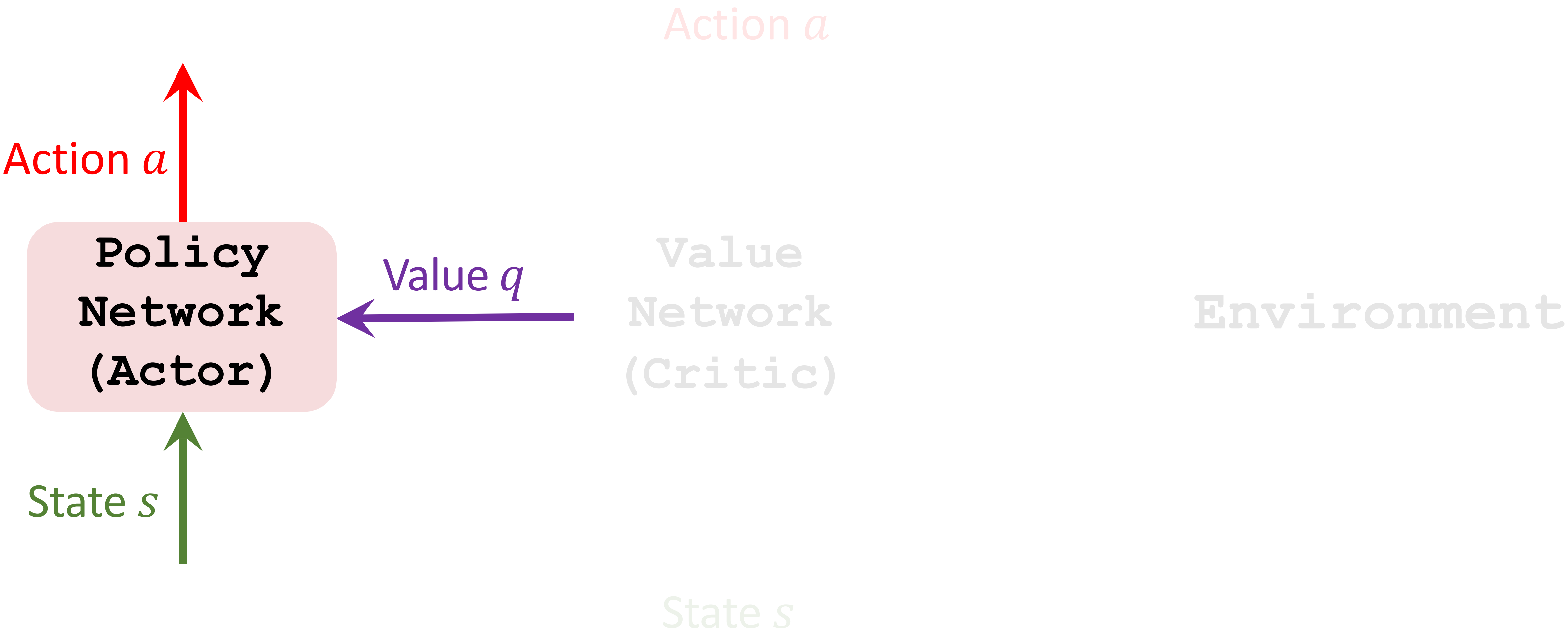
Actor-Critic Method



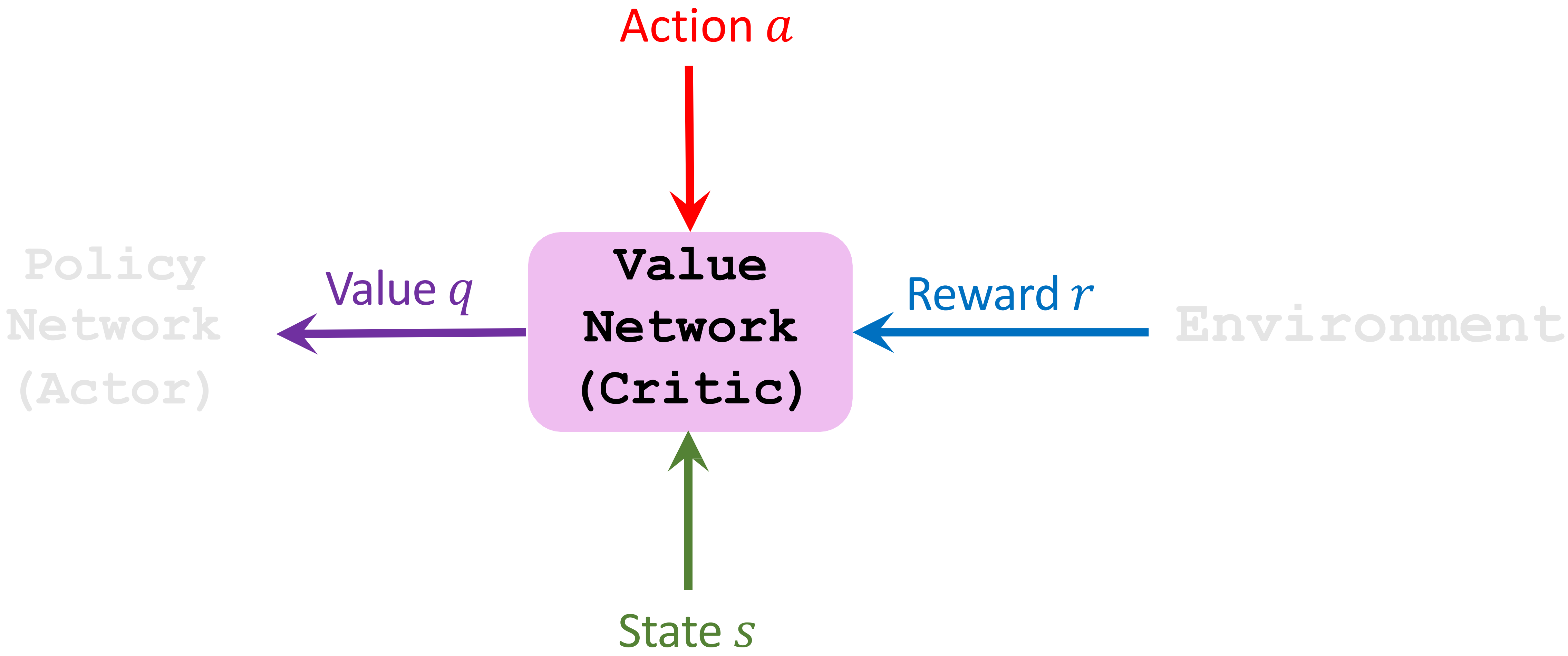
Actor-Critic Method



Actor-Critic Method: Update Actor



Actor-Critic Method: Update Critic



Summary of Algorithm

1. Observe state s_t and randomly sample $a_t \sim \pi(\cdot \mid s_t; \boldsymbol{\theta}_t)$.
2. Perform a_t; then environment gives new state s_{t+1} and reward r_t.
3. Randomly sample $\tilde{a}_{t+1} \sim \pi(\cdot \mid s_{t+1}; \boldsymbol{\theta}_t)$. (Do not perform $\tilde{a}_{t+1}$!)
4. Evaluate value network: $q_t = q(s_t, a_t; \mathbf{w}_t)$ and $q_{t+1} = q(s_{t+1}, \tilde{a}_{t+1}; \mathbf{w}_t)$.
5. Compute TD error: $\delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$.
6. Differentiate value network: $\mathbf{d}_{w,t} = \frac{\partial q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.
7. Update value network: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \delta_t \cdot \mathbf{d}_{w,t}$.
8. Differentiate policy network: $\mathbf{d}_{\theta,t} = \frac{\partial \log \pi(a_t \mid s_t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$.
9. Update policy network: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot q_t \cdot \mathbf{d}_{\theta,t}$.

Summary

Policy Network and Value Network

Definition: State-value function.

- $V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s, a)$.

Definition: function approximation using neural networks.

- Approximate policy function $\pi(a|s)$ by $\pi(a|s; \theta)$ (actor).
- Approximate value function $Q_\pi(s, a)$ by $q(s, a; \mathbf{w})$ (critic).

Roles of Actor and Critic

During training

- Agent is controlled by policy network (actor): $a_t \sim \pi(\cdot\,|s_t;\theta)$.
- Value network q (critic) provides the actor with supervision.

Roles of Actor and Critic

During training

- Agent is controlled by policy network (actor): $a_t \sim \pi(\cdot \,|s_t\,; \theta)$.
- Value network q (critic) provides the actor with supervision.

After training

- Agent is controlled by policy network (actor): $a_t \sim \pi(\cdot \,|s_t\,; \theta)$.
- Value network q (critic) will not be used.

Training

Update the policy network (actor) by policy gradient.

- Seek to increase state-value: $V(s;\theta,\mathbf{w}) = \sum_a \pi(a|s;\theta) \cdot q(s,a;\mathbf{w})$.
- Compute policy gradient: $\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \mathrm{E}_A\left[\frac{\partial \log \pi(A|s,\theta)}{\partial\, \theta} \cdot q(s,A;\mathbf{w})\right]$.
- Perform gradient ascent.

Training

Update the policy network (actor) by policy gradient.

- Seek to increase state-value: $V(s;\theta,\mathbf{w}) = \sum_a \pi(a|s;\theta)\cdot q(s,a;\mathbf{w})$.
- Compute policy gradient: $\frac{\partial\, V(s;\theta)}{\partial\, \theta} = \mathrm{D}_A\left[\frac{\partial \log \pi(A|s,\theta)}{\partial\, \theta}\cdot q(s,A;\mathbf{w})\right]$.
- Perform gradient ascent.

Update the value network (critic) by TD learning.

- Predicted action-value: $q_t = q(s_t, a_t;\mathbf{w})$.
- TD target: $y_t = r_t + \gamma\cdot q(s_{t+1}, a_{t+1};\mathbf{w})$
- Gradient: $\frac{\partial (q_t - y_t)^2/2}{\partial\, \mathbf{w}} = (q_t - y_t)\cdot\frac{\partial\, q(s_t,a_t;\mathbf{w})}{\partial\, \mathbf{w}}$.
- Perform gradient descent.

Thank you!