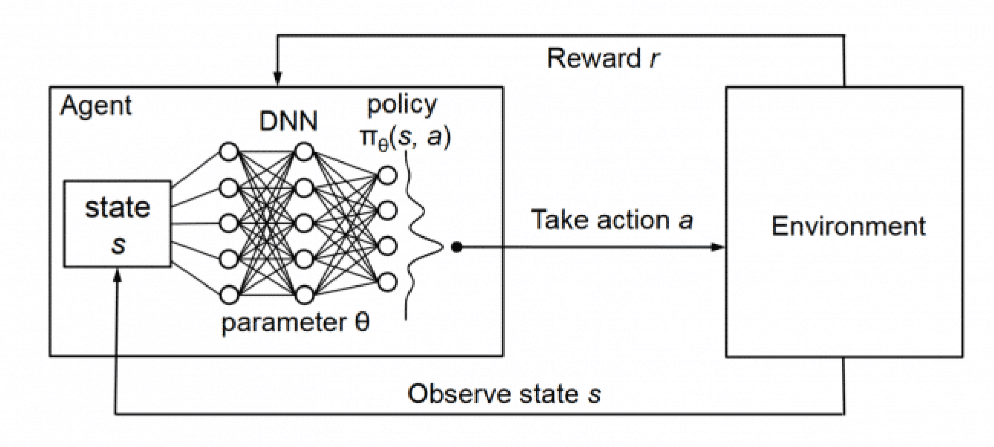
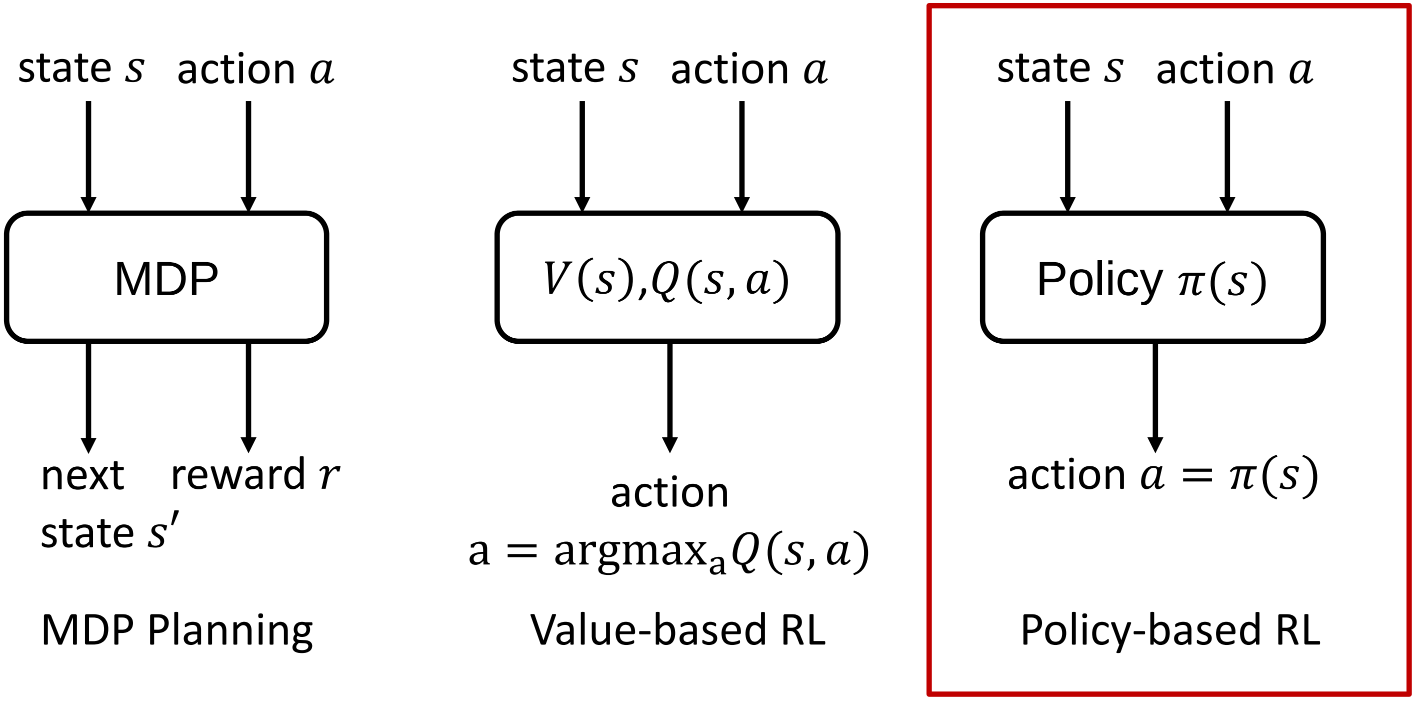
L7.3 Policy-based RL

Zonghua Gu 2021



Acknowledgement: slides based on https://www.coursera.org/specializations/reinforcement-learning
And textbook by Sutton and Barto http://incompleteideas.net/book/the-book-2nd.html

Policy-based RL



state s action a

MDP

next state s' reward r

MDP Planning

state s action a

$V(s)$,$Q(s,a)$

action $\mathrm{a} = \mathrm{argmax_a} Q(s,a)$

Value-based RL

state s action a

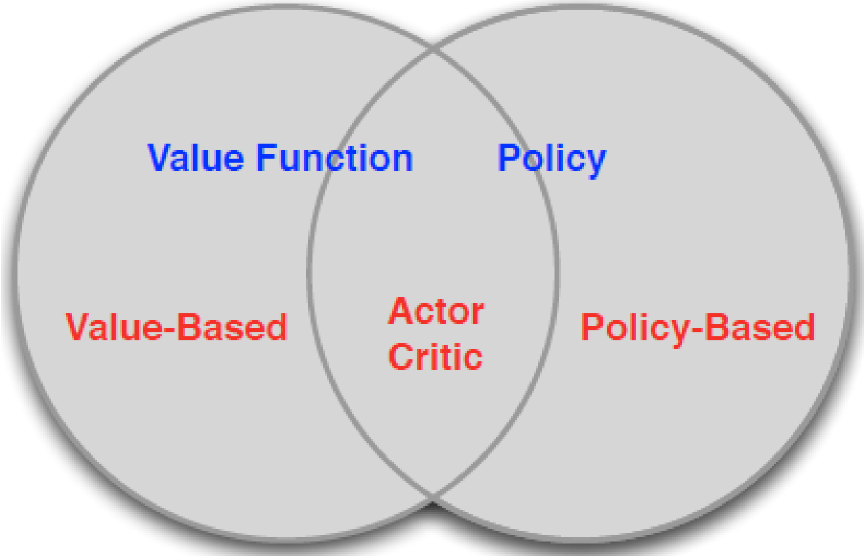
Policy $\pi(s)$

action $a = \pi(s)$

Policy-based RL

CH13 Policy Gradient Methods

- 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

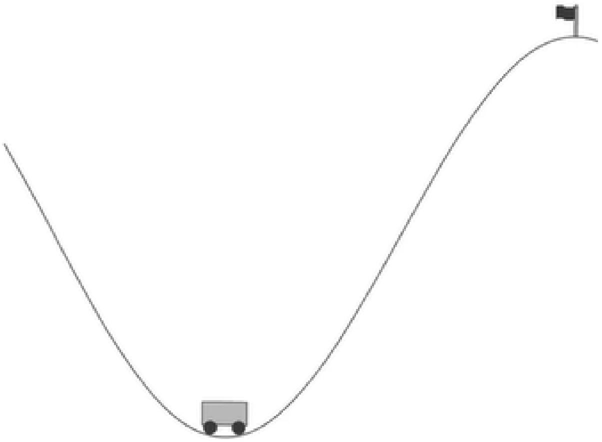


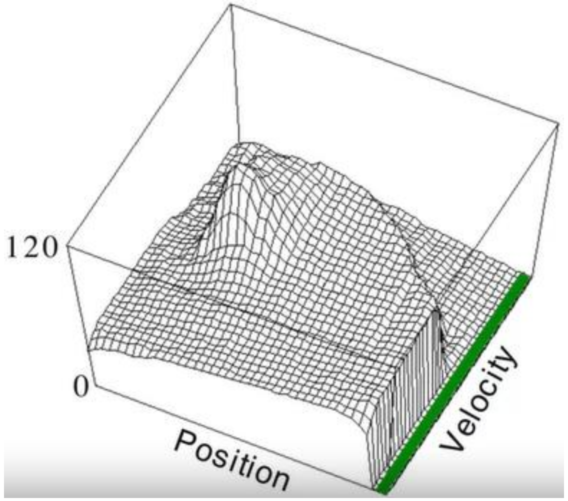
Policy-based RL Pros and Cons

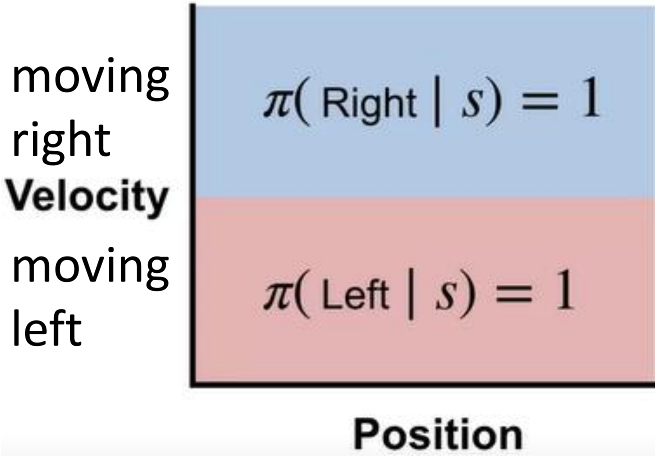
- Pros:
  - Effective in high-dimensional or continuous action space.
    - Value-based RL is only applicable to discrete action space; inefficient to discretize continuous actions for high-dim action space, as taking $\mathrm{argmax}_{\mathrm{a}}\, Q(s, a)$ may be expensive.
  - Can learn stochastic policies
    - Value-based RL learns a near-deterministic policy (greedy or ϵ-greedy).
  - Policy typically converges faster than value functions.
- Disadvantages:
  - Typically converges to a local rather than global optimum.
  - Evaluating a policy is typically inefficient and high variance.

Mountain Car Example

- An under-powered car situated in a valley wants to drive up a steep hill to reach the goal at the top of the rightmost hill. Due to gravity, the car cannot simply accelerate up the steep slope. It must learn to leverage potential energy by driving up the opposite hill before the car is able to make it to the goal.
- Middle: a complex value function
- Right: a simple policy that works well: accelerate in the direction of current velocity.

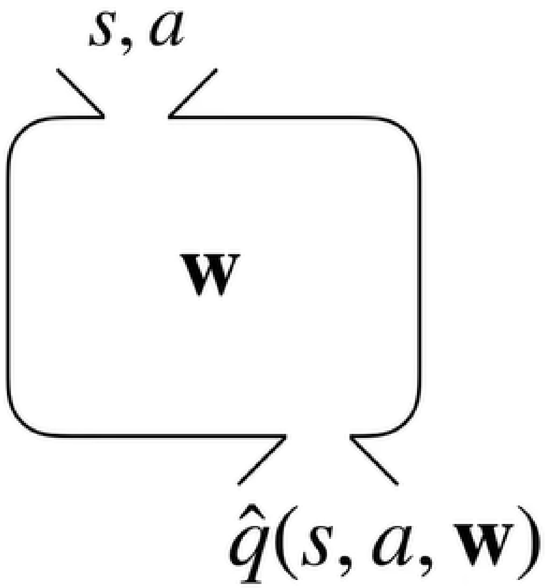


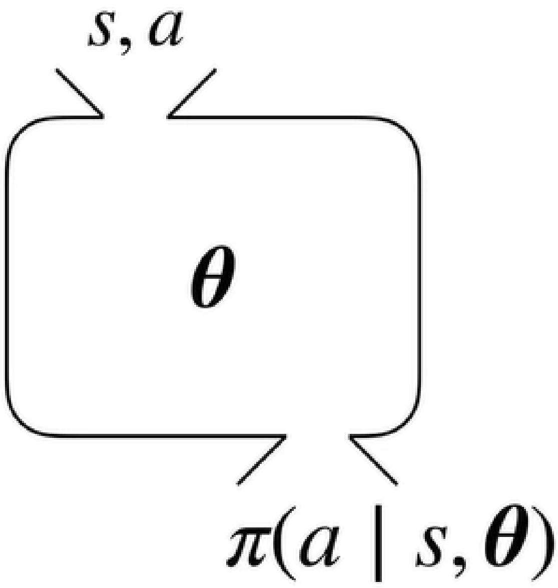




Function Approximation for Action Value Function vs. Policy

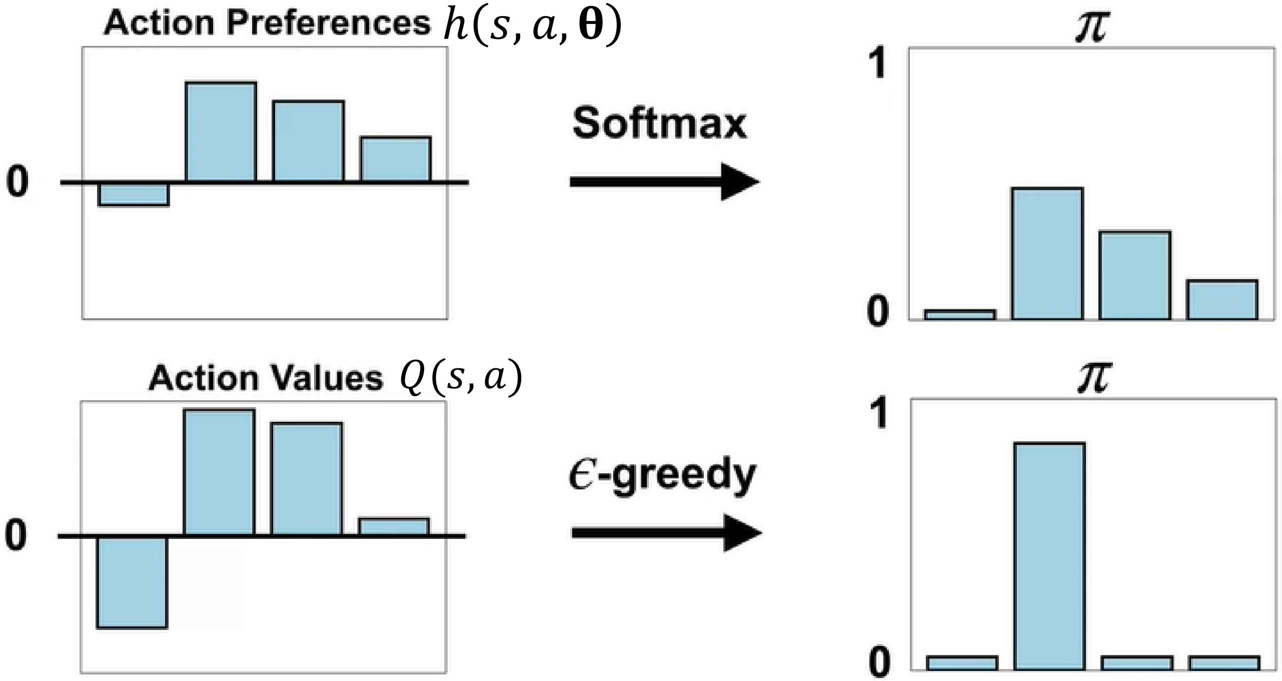
- Value-based RL learns a function approximation for action value function $\hat{q}(s, a, \boldsymbol{w})$.
  - Deterministic policy $a = \operatorname{argmax} \hat{q}(s, a, \boldsymbol{w})$ (may be ϵ-greedy during training)
- Policy-based RL learns a function approximation for stochastic policy $\pi(a|s, \boldsymbol{\theta})$:
  - Probability that action a is taken in state s, with parameter $\boldsymbol{\theta}$. The actual action taken is sampled from the probability distribution $A \sim \pi(a|s, \boldsymbol{\theta})$
  - Probability must be non-negative: $\pi(a|s, \boldsymbol{\theta}) \geq 0, \forall a \in \mathcal{A} \wedge \forall s \in \mathcal{S}$
  - Probabilities must sum to 1: $\sum_{a \in \mathcal{A}} \pi(a|s, \boldsymbol{\theta}) = 1, \forall s \in \mathcal{S}$





SoftMax vs. ϵ-greedy for Discrete Actions

- SoftMax policy: $\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{a\prime\in\mathcal{A}} e^{h(s,a\prime,\boldsymbol{\theta})}}$
  - $h(s,a,\boldsymbol{\theta})$ is action preference, which may be a linear function $\boldsymbol{\theta}^T\mathbf{x}(s,a)$, or the logit from the SoftMax layer of a DNN
  - A bad action with very negative $h(s,a,\boldsymbol{\theta})$ will be very unlikely to be selected
  - Action probabilities change smoothly as a function of $h(s,a,\boldsymbol{\theta})$
- ϵ-greedy: select the greedy action $\underset{a}{\operatorname{argmax}}\, Q(s,a)$ with prob $1-\epsilon+\frac{\epsilon}{|\mathcal{A}(s)|}$
  - No distinction between policies that are not the optimal one; A bad action with very low $Q(s,a)$ will be selected with equal prob as all other non-optimal policies
  - Action probabilities may change dramatically for an arbitrarily small change in $Q(s,a)$, if that change results in a different optimal action

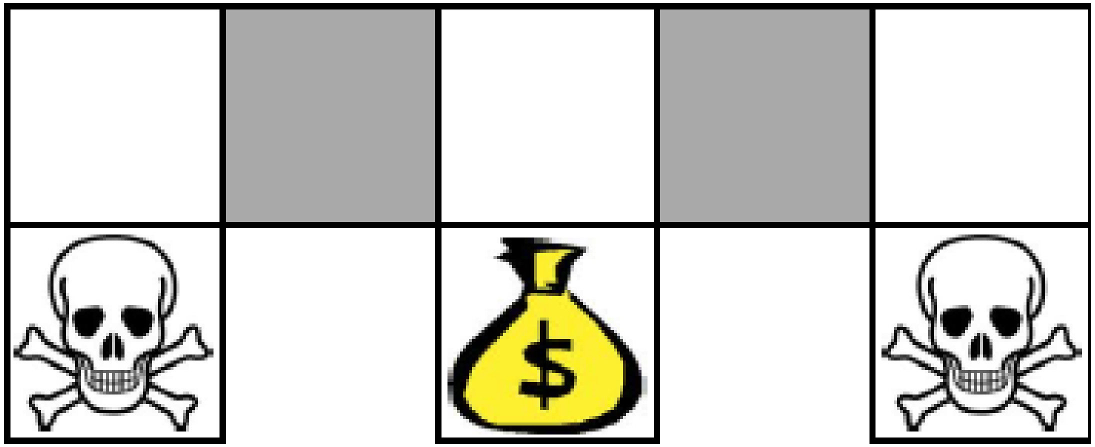


Stochastic Policy vs. Deterministic Policy

- Example 1: two-player game of rock-paper-scissors
  - Scissors beat paper; paper beats rock; rock beats scissors
  - For iterated game, a deterministic policy is easily exploited by the opponent; a uniform random policy is optimal, and achieves Nash equilibrium
- Example 2: Aliased Grid World (POMDP)

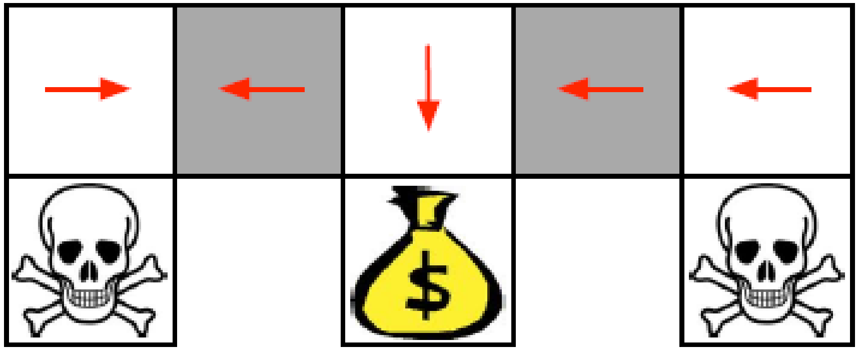
Aliased Grid World (POMDP)

- Env has 3 terminal states: 1 with high positive reward and 2 with high negative reward. It is a Partially Observable MDP (POMDP): Agent cannot observe its position directly; it can only observe features of the following form (for all directions $d_1, d_2, \ldots \in (N, E, S, W)$):
  - $\phi(s) = \mathbf{1}(\text{wall to } d_1, d_2 \ldots)$ (it can detect walls, e.g., w. Radar or Lidar)
    - $\mathbf{1}(\boldsymbol{x})$ is indicator function: $\mathbf{1}(\boldsymbol{x}) = \mathbf{1}$ if $x = \text{true}$
  - Agent cannot differentiate between the 2 grey states
- Value-based RL learns a deterministic policy: $a = \operatorname{argmax} \hat{q}(s, a, \boldsymbol{w}) = \operatorname{argmax} f(\phi(s), \boldsymbol{w})$
- Policy-based RL learns a stochastic policy: $\pi(a|s, \boldsymbol{\theta}) = g(\phi(s), \boldsymbol{\theta})$

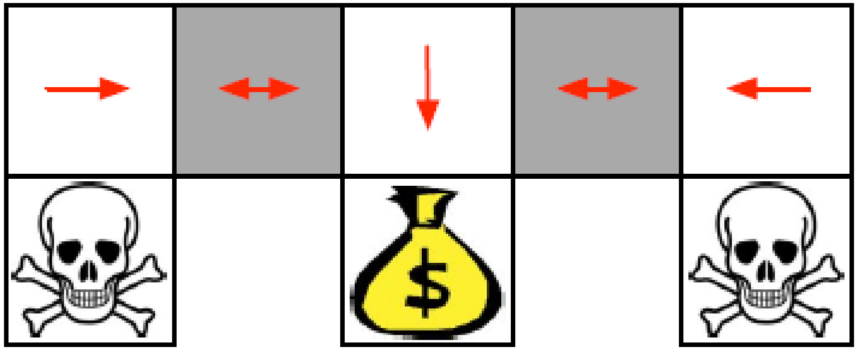


Aliased Grid World (POMDP)

- Left: an optimal deterministic policy:
  - Either move W in both grey states (red arrows), or move E in both grey states; Either way, agent can get stuck and never reach the money
- Right: the optimal stochastic policy:
  - Randomly move E or W in grey states: $\pi(\text{move } E|\text{wall to N and S}, \boldsymbol{\theta}) = \pi(\text{move } W|\text{wall to N and S}, \boldsymbol{\theta}) = 0.5$
  - Agent will likely reach the goal state quickly
- How about adding ϵ-greedy on top of the opt det policy?
  - Agent may get into one of the 2 bad states, since each non-optimal action is given equal probability in every state



Opt det policy

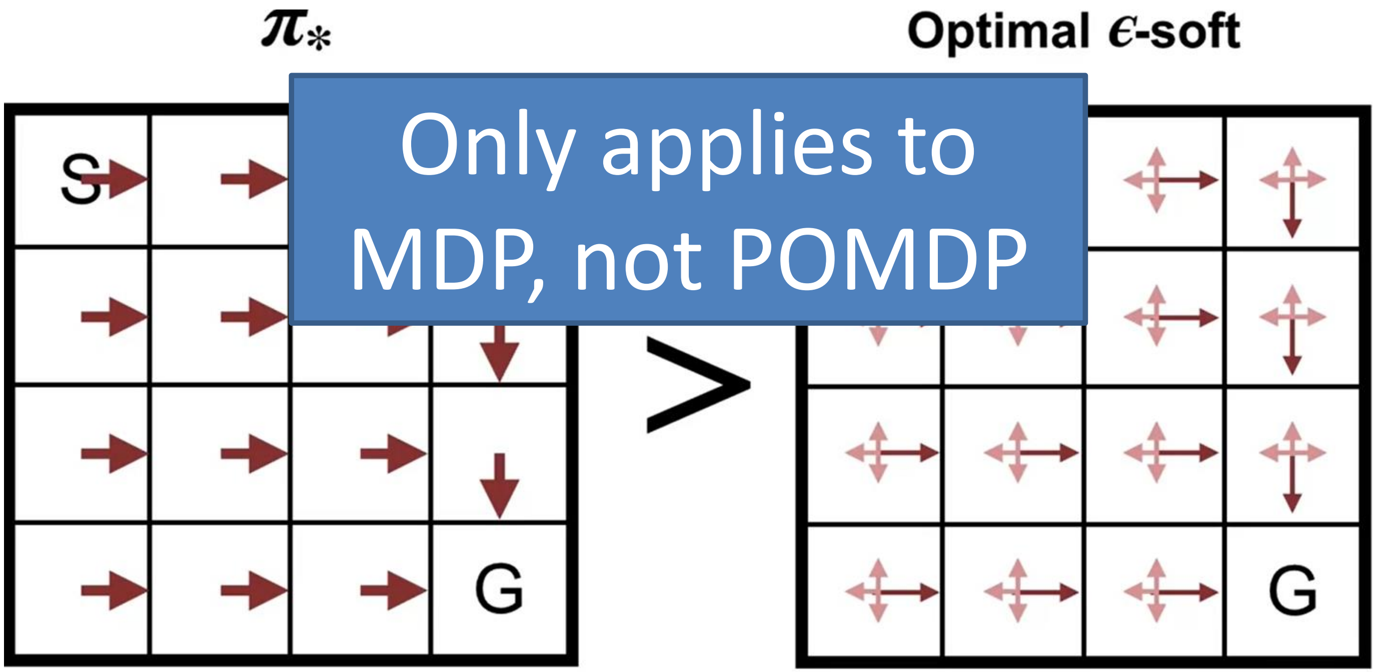


Opt Sto policy

Recall

Optimal ϵ-Soft Policy

- The optimal ϵ-soft policy is the policy with the highest value in each state among all ϵ-soft policies. It performs worse than the optimal greedy deterministic policy π_* in general.
- But it often performs reasonably well, and avoids exploring starts.



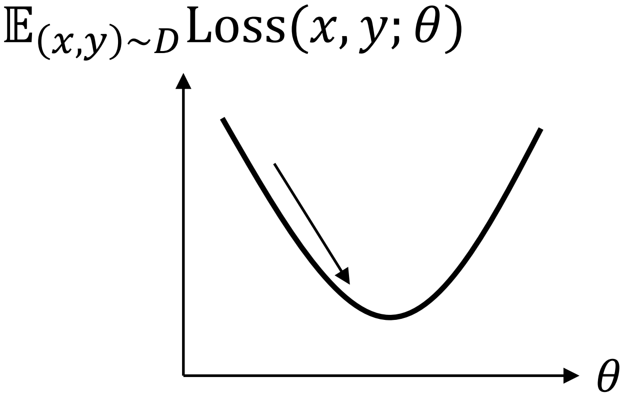
Only applies to MDP, not POMDP

Optimization Objective

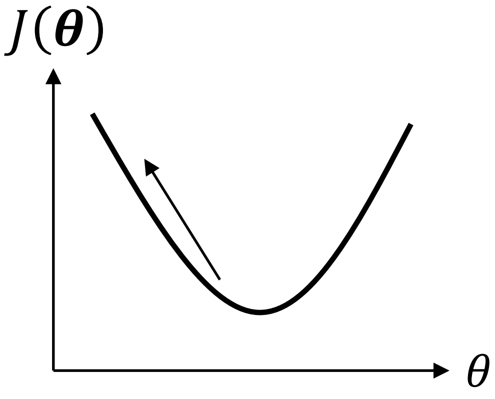
- We consider the episodic case, and would like to optimize the expected value of the start state of each episode, with start state s_0:
- $J(\boldsymbol{\theta}) \doteq v_\pi(s_0)$
- where $v_\pi(s_0)$ is the true value function for policy π, parametrized by $\boldsymbol{\theta}$: $\pi(a|s, \boldsymbol{\theta})$

Model Training in Supervised Learning vs. Policy Gradient in RL

- SL: to solve $\min_{\theta} \mathbb{E}_{(x,y)\sim D} \mathrm{Loss}(x, y; \theta)$ for model training: gradient descent $\theta \leftarrow \theta - \alpha \nabla_{\theta} \mathrm{Loss}(x, y; \theta)$
  - Update model params θ by following the gradient downhill, in order to decrease $\mathrm{Loss}(x, y; \theta)$. ($\alpha$ is the Learning Rate)



- RL: to solve $\max_{\theta} J(\boldsymbol{\theta})$ in Policy Gradient: gradient ascent $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\boldsymbol{\theta})$
  - Update policy model params θ by following the gradient uphill, in order to increase $J(\boldsymbol{\theta})$



- We use ∇ as shorthand for ∇_{θ}

Policy Gradient Theorem

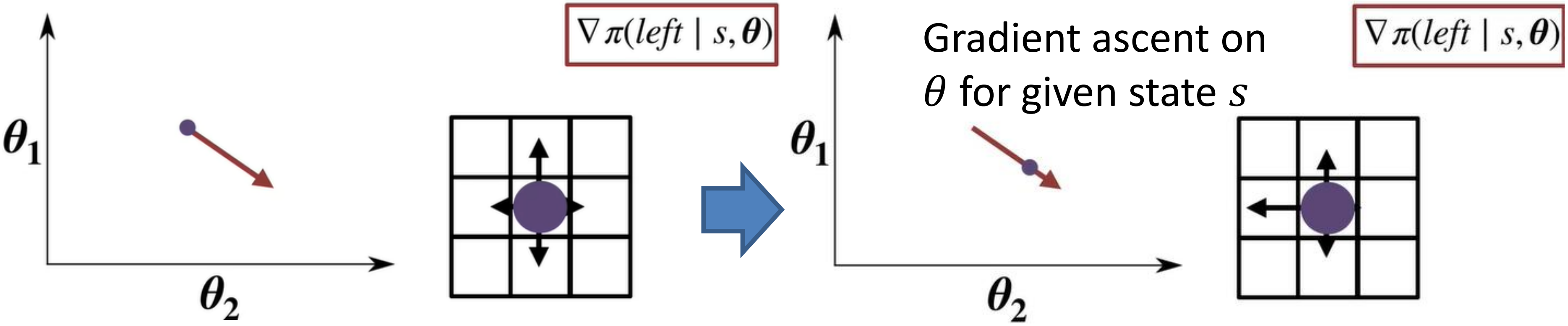
- We consider episodic instead of continuous environments in this lecture
- $\nabla J(\boldsymbol{\theta}) = \nabla v_\pi(s_0)$
- $= \nabla[\sum_a \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)]$
- $= \sum_a [\nabla \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a) + \pi(a|s, \boldsymbol{\theta}) \nabla q_\pi(s, a)]$
- Policy Gradient Theorem (proof in RLBook p. 325; assuming discount factor $\gamma = 1$):
- $\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)$
  - $\mu(s)$: on-policy distribution under policy π, $\sum_{s \in \mathcal{S}} \mu(s) = 1$. A larger $\mu(s)$ denotes state s is visited more frequently, hence the policy gradient term for state s is given more weight (i.e., we care more about frequently-visited states than rarely-visited states).
  - In the episodic case, the constant of proportionality is the average length of an episode; in the continuing case it is 1.

Policy Gradient with Baseline

- $\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s, \boldsymbol{\theta})(q_\pi(s,a) - b(s))$
  - Baseline $b(s)$ can be any function that is independent of action a. Subtracting $b(s)$ does not affect the computed expectation since: $\sum_a \nabla \pi(a|s,\boldsymbol{\theta}) b(s) = b(s) \sum_a \nabla \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla \sum_a \pi(a|s,\boldsymbol{\theta}) = b(s)\nabla 1 = 0$
  - Subtracting $b(s)$ helps to reduce variance and speed up convergence, e.g., consider 3 samples with $\nabla \pi(a|s,\boldsymbol{\theta}) = \{0.5, 0.2, 0.3\}, q_\pi(s,a) = \{1000, 1001, 1002\}, var(0.5 \cdot 1000{,}0.2 \cdot 1001{,}0.3 \cdot 1002) \approx 23287$; After subtracting a baseline of $1001, var(0.5 \cdot (-1), 0.2 \cdot 0, .3 \cdot 1) \approx 0.16$
- Advantage function:
  - $A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$ (using state value function $v_\pi(s)$ as baseline)
  - $A_\pi(s,a)$ captures the advantage of taking action a in state s then follow policy π, compared to the baseline of following policy π from state s, i.e., we care more about the relative ranking of different actions in state S_t than absolute values of $q_\pi(s,a)$

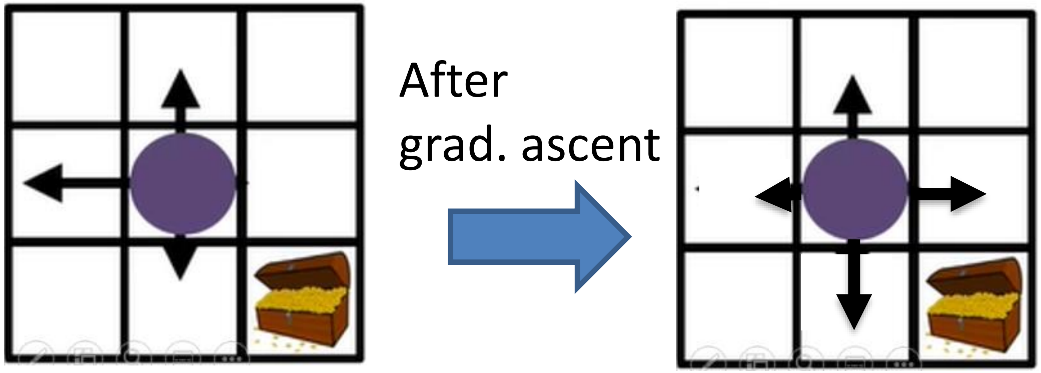
Policy Gradient Example

- The gradient $\nabla\pi(left|s,\boldsymbol{\theta})$ tells us how to change the policy parameters $\boldsymbol{\theta}$ to make action $left$ more likely to be selected in state s. By gradient ascent on $\boldsymbol{\theta}$, we increase the probability for taking action $left$ in state s.



Gradient ascent on θ for given state s

$$q_\pi(s,up) = -1.3$$
$$q_\pi(s,left) = -0.9$$
$$q_\pi(s,down) = 0.7$$
$$q_\pi(s,right) = 1.5$$



After grad. ascent

MC REINFORCE

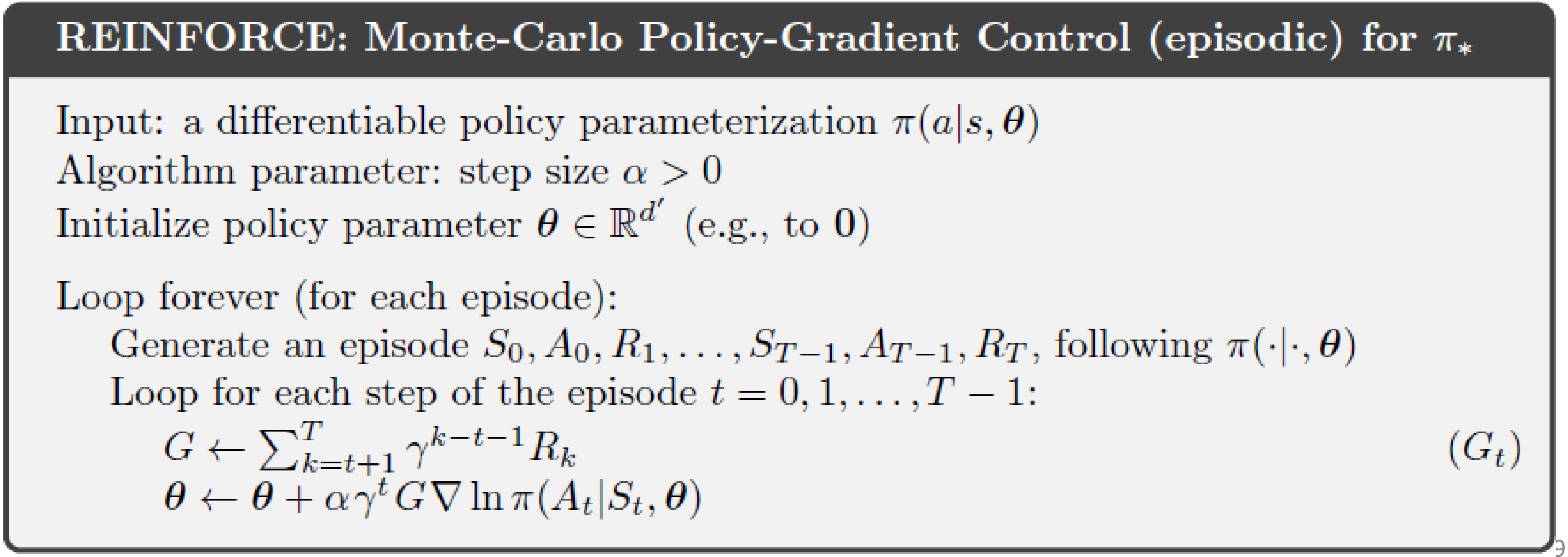
- $\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \nabla\pi(a|s,\boldsymbol{\theta}) q_\pi(s,a)$
- $= \mathbb{E}_\pi[\sum_a \nabla\pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a)]$
  - Outer exp over policy π: average over all experienced states under policy π, i.e., $S_t \sim s$
- $= \mathbb{E}_\pi\left[\sum_a \pi(a|S_t,\boldsymbol{\theta}) \frac{\nabla\pi(a|S_t,\theta)}{\pi(a|S_t,\theta)} q_\pi(S_t,a)\right]$
- $= \mathbb{E}_\pi[\sum_a \pi(a|S_t,\boldsymbol{\theta})\nabla\log \pi(A_t|S_t,\boldsymbol{\theta}) q_\pi(S_t,a)]$ (since $\nabla \log f(x) = \frac{\nabla f(x)}{f(x)}$)
- $= \mathbb{E}_\pi[\mathbb{E}_\pi[\nabla\log \pi(A_t|S_t,\boldsymbol{\theta}) q_\pi(S_t,A_t)]]$
  - Inner exp over policy π: average over all experienced actions A_t from state S_t under policy π, i.e., $A_t \sim \pi(a|S_t,\boldsymbol{\theta})$
- $= \mathbb{E}_\pi[\nabla\log \pi(A_t|S_t,\boldsymbol{\theta}) q_\pi(S_t,A_t)]$
  - Outer and inner exp over policy π combined: execute policy π and average over all experienced states S_t and actions A_t from state S_t, i.e., $S_t \sim s, A_t \sim \pi(a|S_t,\theta)$
- $= \mathbb{E}_\pi[\nabla\log \pi(A_t|S_t,\boldsymbol{\theta}) G_t]$
  - Use return $G_t \doteq \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$ as unbiased estimate of $q_\pi(S_t,A_t)(\mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t))$
  - $\nabla\log \pi(A_t|S_t,\boldsymbol{\theta})$ is called the score function
- SGD update ($\gamma = 1$): $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\nabla\log \pi(A_t|S_t,\boldsymbol{\theta_t})G_t$
- SGD update ($\gamma < 1$): $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\gamma^t\nabla\log \pi(A_t|S_t,\boldsymbol{\theta_t})G_t$

MC REINFORCE Example

- Consider a given state S with two possible actions A_1 and A_2. Initially $\pi(A_1|S, \boldsymbol{\theta}_t) = .9, \pi(A_2|S, \boldsymbol{\theta}_t) = .1$. Consider one episode with 9 occurrences of (S, A_1) at steps $t1, t2, \ldots, t9$ with return $G_{t1} = \cdots = G_{t9} = 1$, and 1 occurrence of (S, A_2) at step $t10$ with return $G_{t10} = 2$. SGD update: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \nabla \log \pi(A_t|S_t, \boldsymbol{\theta}_t) G_t$. Assume that the i-th update to $\boldsymbol{\theta}$ results in a small update ϵi to $\pi(a|S_t, \boldsymbol{\theta})$.
- Correct update sequence with the log:
  - 1st update at step $t1$: $\boldsymbol{\theta}_{t1+1} = \boldsymbol{\theta}_{t1} + \alpha\gamma^{t1} \nabla \log \pi(A_{t1}|S_{t1}, \boldsymbol{\theta}_{t1}) G_{t1} = \boldsymbol{\theta}_{t1} + \frac{1}{.9} \alpha\gamma^{t1} \nabla \pi(A_1|S, \boldsymbol{\theta}_{t1}) \cdot 1$
  - 2nd update at step $t2$: $\boldsymbol{\theta}_{t2+1} = \boldsymbol{\theta}_{t2} + \alpha\gamma^{t2} \nabla \log \pi(A_{t2}|S_{t2}, \boldsymbol{\theta}_{t2}) G_{t2} = \boldsymbol{\theta}_{t2} + \frac{1}{.9+\epsilon 1} \alpha\gamma^{t2} \nabla \pi(A_1|S, \boldsymbol{\theta}_{t2}) \cdot 1$
  - …
  - 9th update at step $t9$: $\boldsymbol{\theta}_{t9+1} = \boldsymbol{\theta}_{t9} + \alpha\gamma^{t9} \nabla \log \pi(A_{t9}|S_{t9}, \boldsymbol{\theta}_{t9}) G_{t9} = \boldsymbol{\theta}_{t9} + \frac{1}{.9+\epsilon 1+\cdots+\epsilon 8} \alpha\gamma^{t9} \nabla \pi(A_1|S, \boldsymbol{\theta}_{t9}) \cdot 1$
  - 10th update at step $t10$: $\boldsymbol{\theta}_{t10+1} = \boldsymbol{\theta}_{t10} + \alpha\gamma^{t10} \nabla \log \pi(A_{t10}|S_{t10}, \boldsymbol{\theta}_{t10}) G_{t10} = \boldsymbol{\theta}_{t10} + \frac{1}{1-(.9+\epsilon 1+\cdots+\epsilon 9)} \alpha\gamma^{t10} \nabla \pi(A_2|S, \boldsymbol{\theta}_{t10}) \cdot 2$
- Incorrect update sequence without the log:
  - 1st update at step $t1$: $\boldsymbol{\theta}_{t1+1} = \boldsymbol{\theta}_{t1} + \alpha\gamma^{t1} \nabla \log \pi(A_{t1}|S_{t1}, \boldsymbol{\theta}_{t1}) G_{t1} = \boldsymbol{\theta}_{t1} + \alpha\gamma^{t1} \nabla \pi(A_1|S, \boldsymbol{\theta}_{t1}) \cdot 1$
  - 2nd update at step $t2$: $\boldsymbol{\theta}_{t2+1} = \boldsymbol{\theta}_{t2} + \alpha\gamma^{t2} \nabla \log \pi(A_{t2}|S_{t2}, \boldsymbol{\theta}_{t2}) G_{t2} = \boldsymbol{\theta}_{t2} + \alpha\gamma^{t2} \nabla \pi(A_1|S, \boldsymbol{\theta}_{t2}) \cdot 1$
  - …
  - 9th update at step $t9$: $\boldsymbol{\theta}_{t9+1} = \boldsymbol{\theta}_{t9} + \alpha\gamma^{t9} \nabla \log \pi(A_{t9}|S_{t9}, \boldsymbol{\theta}_{t9}) G_{t9} = \boldsymbol{\theta}_{t9} + \alpha\gamma^{t9} \nabla \pi(A_1|S, \boldsymbol{\theta}_{t9}) \cdot 1$
  - 10th update at step $t10$: $\boldsymbol{\theta}_{t10+1} = \boldsymbol{\theta}_{t10} + \alpha\gamma^{t10} \nabla \log \pi(A_{t10}|S_{t10}, \boldsymbol{\theta}_{t10}) G_{t10} = \boldsymbol{\theta}_{t10} + \alpha\gamma^{t10} \nabla \pi(A_2|S, \boldsymbol{\theta}_{t10}) \cdot 2$
- Assuming each ϵi is small. The correct update sequence gives roughly equal weight to the two action choices with return of 1 (action A_1 in state S, experienced 9 times) and return of 2 (action A_2 in state S, experienced once), so $\boldsymbol{\theta}$ will be updated towards preferring action A_2 in state S
- The incorrect update sequence gives roughly 9 times the weight to the action choice with return of 1 (action A_1 in state S, experienced 9 times) than the action choice with return of 2 (action A_2 in state S, experienced once), so $\boldsymbol{\theta}$ will be incorrectly updated towards preferring action A_1 in state S
- “The update increases the parameter vector in this direction proportional to the return, and inversely proportional to the action probability. The former makes sense because it causes the parameter to move most in the directions that favor actions that yield the highest return. The latter makes sense because otherwise actions that are selected frequently are at an advantage (the updates will be more often in their direction) and might win out even if they do not yield the highest return.” – RLBook p. 327

MC REINFORCE Pseudo-Code

- At the end of each episode, for each timestep $0 \leq t < T$:
  - Calculate the return G_t from each timestep t
  - Update policy params $\boldsymbol{\theta}$ with SGD



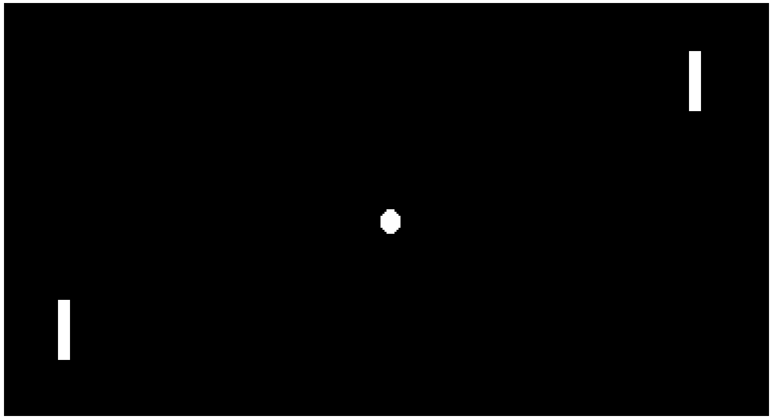
REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

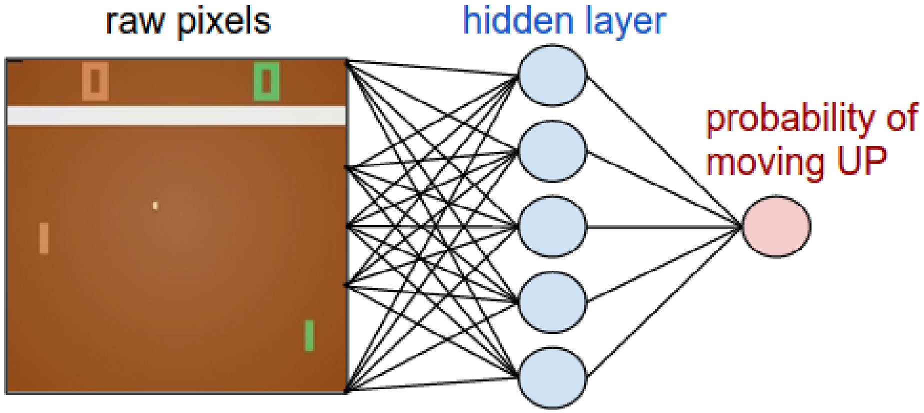
```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$                    $(G_t)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$
```

Example: Game of Pong

- The agent plays one of the paddles (the other is controlled by a decent AI) and it has to bounce the ball past the other player. For each input image, agent decides if it wants to move the paddle UP or DOWN (2 discrete actions). At the end of the game the agent either wins (reward $R_T > 0$) or loses ($R_T < 0$), i.e., the reward is sparse.
  - (In practice we may stack multiple input images as input to the agent.)
- The agent implements a policy network, which maps from input image to two possible actions (UP or DOWN) with a stochastic SoftMax policy.

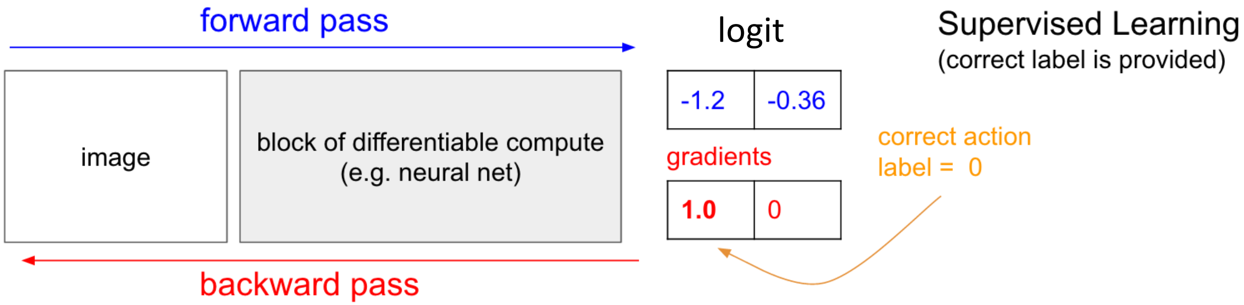


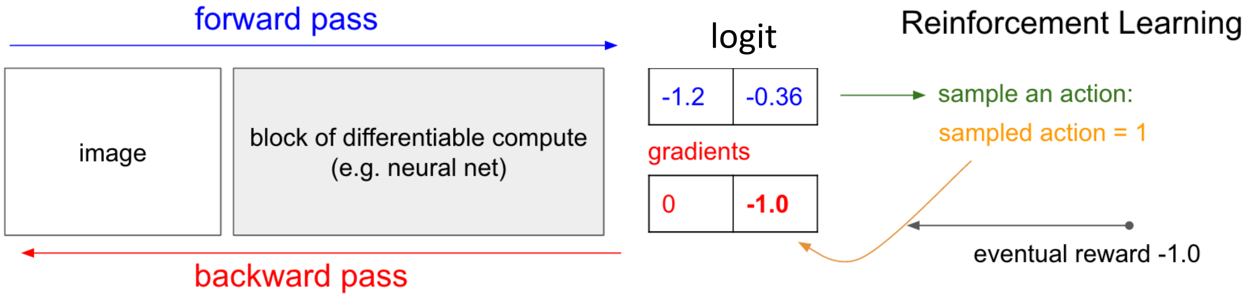


http://karpathy.github.io/2016/05/31/rl/

SL vs. RL

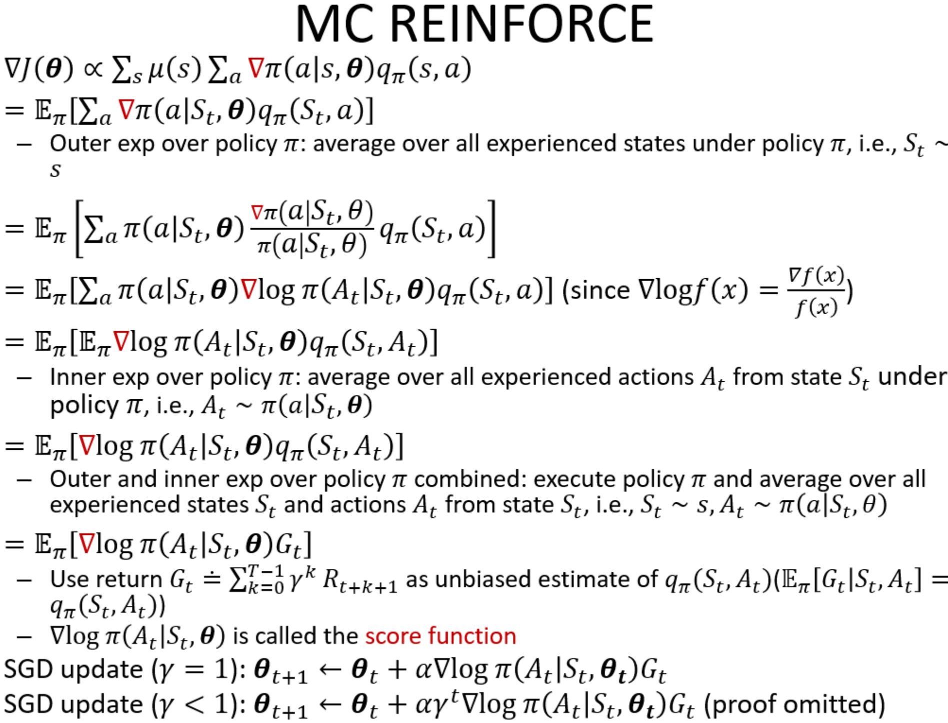
- With SL: if the NN predicts $y = UP$ for input x, and it is the correct ground truth label (e.g., the expert action in Imitation Learning), then gradient descent ($\nabla_\theta \log p(y = UP|x)$) will make the NN more likely to predict $y = UP$ for input x
- With RL: if the NN predicts $y = DOWN$ for input x, but we don’t have the ground truth label in the middle of the game, so we must wait until the end of the game. If agent loses, then gradient descent ($\nabla_\theta \log p(y = DOWN|x)$) will make the NN less likely to predict $y = DOWN$ for input x
    - Credit assignment problem: in an episode of many steps, which step contributed the most to the final outcome?





http://karpathy.github.io/2016/05/31/rl/

MC REINFORCE for Pong I



MC REINFORCE

$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \nabla\pi(a|s,\boldsymbol{\theta}) q_\pi(s,a)$

$= \mathbb{E}_\pi[\sum_a \nabla\pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a)]$

- Outer exp over policy π: average over all experienced states under policy π, i.e., $S_t \sim s$

$= \mathbb{E}_\pi\left[\sum_a \pi(a|S_t,\boldsymbol{\theta}) \frac{\nabla\pi(a|S_t,\theta)}{\pi(a|S_t,\theta)} q_\pi(S_t,a)\right]$

$= \mathbb{E}_\pi[\sum_a \pi(a|S_t,\boldsymbol{\theta}) \nabla\log\pi(A_t|S_t,\boldsymbol{\theta}) q_\pi(S_t,a)]$ (since $\nabla\log f(x) = \frac{\nabla f(x)}{f(x)}$)

$= \mathbb{E}_\pi[\mathbb{E}_\pi \nabla\log\pi(A_t|S_t,\boldsymbol{\theta}) q_\pi(S_t,A_t)]$

- Inner exp over policy π: average over all experienced actions A_t from state S_t under policy π, i.e., $A_t \sim \pi(a|S_t,\boldsymbol{\theta})$

$= \mathbb{E}_\pi[\nabla\log\pi(A_t|S_t,\boldsymbol{\theta}) q_\pi(S_t,A_t)]$

- Outer and inner exp over policy π combined: execute policy π and average over all experienced states S_t and actions A_t from state S_t, i.e., $S_t \sim s, A_t \sim \pi(a|S_t,\theta)$

$= \mathbb{E}_\pi[\nabla\log\pi(A_t|S_t,\boldsymbol{\theta}) G_t]$

- Use return $G_t \doteq \sum_{k=0}^{T-1} \gamma^k R_{t+k+1}$ as unbiased estimate of $q_\pi(S_t,A_t)(\mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t))$
- $\nabla\log\pi(A_t|S_t,\boldsymbol{\theta})$ is called the score function

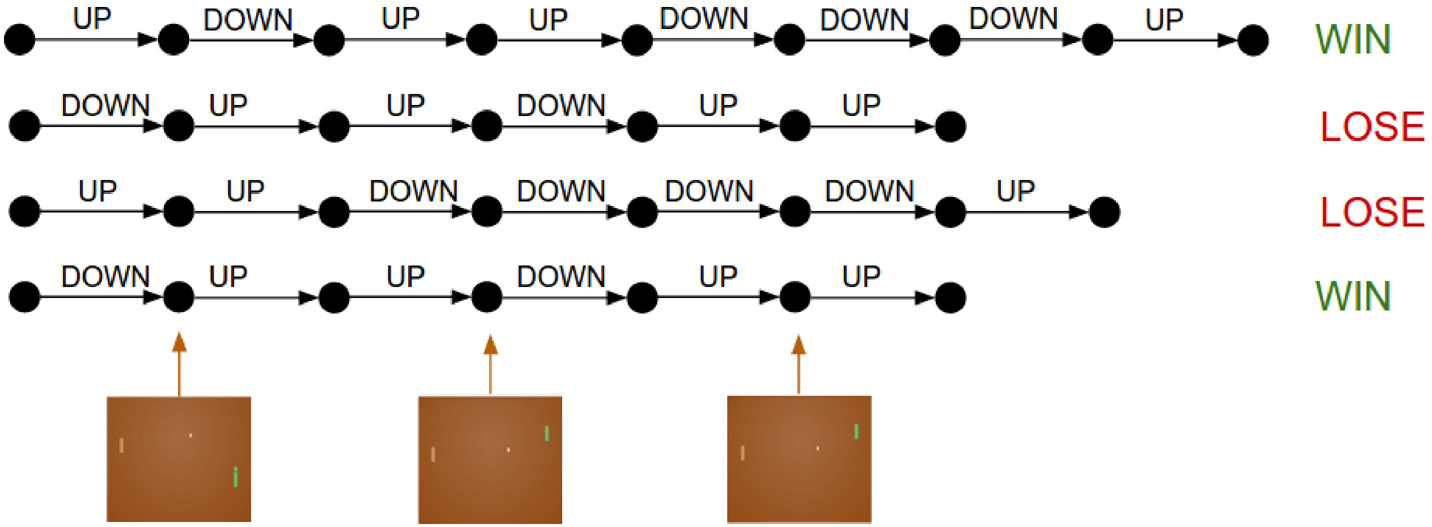
SGD update ($\gamma = 1$): $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\nabla\log\pi(A_t|S_t,\boldsymbol{\theta_t}) G_t$

SGD update ($\gamma < 1$): $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\gamma^t\nabla\log\pi(A_t|S_t,\boldsymbol{\theta_t}) G_t$ (proof omitted)

- $J(\boldsymbol{\theta}) = \sum_s \mu_\pi(s) \sum_a \pi(a|s,\theta) q_\pi(s,a)$
- Consider agent in a given state S_t at time step t, and we want to select policy params $\boldsymbol{\theta}$ to maximize
- $v_\pi(S_t) = \sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a)$
- $= \pi(U|S_{\mathrm{t}},\boldsymbol{\theta}) q_\pi(S_t,U) + \pi(D|S_t,\boldsymbol{\theta}) q_\pi(S_t,D)$
- Taking derivative w.r.t policy params $\boldsymbol{\theta}$
- $\nabla_\theta v_\pi(S_t) = \sum_a \nabla_\theta\pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a)$
- $= \nabla_\theta\pi(U|S_t,\boldsymbol{\theta}) q_\pi(S,U) + \nabla_\theta\pi(D|S_t,\boldsymbol{\theta}) q_\pi(S_t,D)$
- $= \pi(U|S_t,\boldsymbol{\theta}) \nabla_\theta \log\pi(U|S_t,\boldsymbol{\theta})\, q_\pi(S,U) + \pi(D|S_t,\boldsymbol{\theta}) \nabla_\theta \log\pi(D|S_t,\boldsymbol{\theta})\, q_\pi(S_t,D)$
- $= \mathbb{E}_\pi[\nabla_\theta \log\pi(a|S_t,\boldsymbol{\theta})\, q_\pi(S_t,a)]$
- $= \mathbb{E}_\pi[\nabla_\theta \log\pi(a|S_t,\boldsymbol{\theta})\, G_t]$
- Suppose $q_\pi(S_t,U) > 0, q_\pi(S_t,D) < 0.$
  - For the UP action in state S_t, the policy update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\nabla_\theta \log\pi(U|S_t,\theta)\, q_\pi(S_t,U)$ will push up $\pi(U|S_t,\theta)$, i.e, make it more likely to move UP in state S_t
  - For the DOWN action in state S_t, the policy update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\nabla_\theta \log\pi(U|S_t,\theta)\, q_\pi(S_t,D)$ will push down $\pi(U|S_t,\theta)$, i.e, make it less likely to move DOWN in state S_t

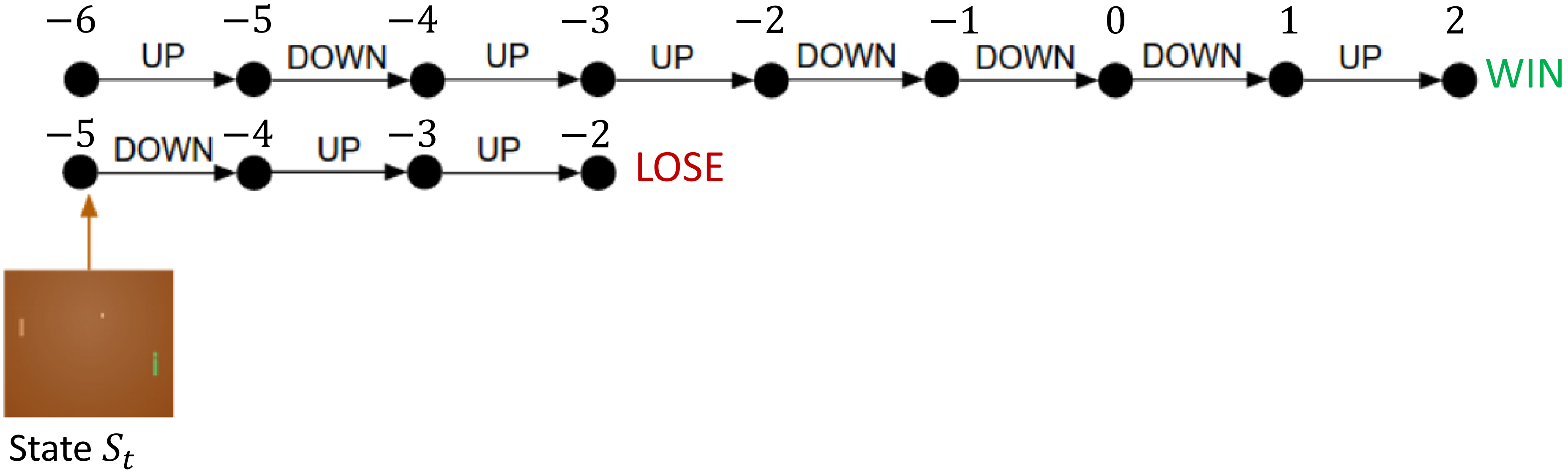
MC REINFORCE for Pong II

- Agent plays 4 rollouts (episodes), and won 2 episodes and lost 2. Assume that each episode lasts 200 steps, so agent made 200 decisions of UP or DOWN in each episode. Each step has reward of $R_t = 0$, i.e., we don't care how long each episode lasts. $\forall t, G_t = \gamma^{T-t-1} R_T$, i.e., return G_t at any step t of each episode is equal to the discounted reward R_T at the end of the episode, hence G_t has the same sign as R_T (recall "MC Prediction" in L7.2 Value-based RL).
- For each of the 2 winning episodes ($\forall t, G_t > 0$), NN params $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \log \pi(A_t|S_t, \boldsymbol{\theta}) G_t$ are updated to encourage all taken actions in the 200 steps (push up $\pi(A_t|S_t, \boldsymbol{\theta})$).
- For each of the 2 losing episodes ($\forall t, G_t < 0$), NN params $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \log \pi(A_t|S_t, \boldsymbol{\theta}) G_t$ are updated to discourage all taken actions in the 200 steps (push down $\pi(A_t|S_t, \boldsymbol{\theta})$).
- The NN will now become slightly more likely to repeat actions that worked, and slightly less likely to repeat actions that didn't work.
- If discount factor $\gamma = 1$, then we assign equal credit to all actions in the same episode; if $\gamma < 1$, then we assign more credit to actions taken in later steps of each episode than earlier steps.



MC REINFORCE for Pong III

- Consider the case when each step has reward of $R_t = -1$, to minimize the length of each episode, e.g., we may prefer a short losing episode to a long winning episode. Assuming discount factor $\gamma = 1$. Suppose reward at the end of each winning episode is $R_T = 2$; reward at the end of each losing episode is $R_T = -2$. The return G_t at each step is shown below. Consider state S_t.
- After the winning episode, NN params $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \log \pi(UP|S_t, \boldsymbol{\theta})(-6)$ are updated to discourage the UP action in state S_t (push down $\pi(UP|S_t, \boldsymbol{\theta})$).
- After the losing episode, NN params $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \log \pi(DOWN|S_t, \boldsymbol{\theta})(-5)$ are updated to discourage the DOWN action in state S_t (push down $\pi(DOWN|S_t, \boldsymbol{\theta})$).
- Both will pushdown $\pi(DOWN|S_t, \boldsymbol{\theta})$, but since the return $G_t = -6$ in the winning episode causes a larger update magnitude than $G_t = -5$ in the losing episode, agent is slightly more likely to select the DOWN action in state S_t.
- Subtracting a baseline, e.g., use $G_t - \hat{v}_\pi(S_t, \boldsymbol{w})$ to replace G_t helps to reduce variance and speed up convergence, e.g., suppose $\hat{v}_\pi(S_t, \boldsymbol{w}) = -5.5$, then updates to $\boldsymbol{\theta}$ will be in different directions for the two episodes.



PG Variants

- MC REINFORCE (no bias, high variance):
  - $\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi[\nabla \log \pi(A_t|S_t, \boldsymbol{\theta}) q_\pi(S_t, A_t)] = \mathbb{E}_\pi[\log \pi(A_t|S_t, \boldsymbol{\theta}) G_t]$
- MC REINFORCE with a baseline of estimated state value function:
  - $\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi[\log \pi(A_t|S_t, \boldsymbol{\theta})(G_t - \hat{v}_\pi(S_t, \boldsymbol{w}))]$
  - $\hat{v}_\pi(S_t, \boldsymbol{w})$ is a function approximation of the true $v_\pi(S_t, \boldsymbol{w})$, e.g., a value network
- Actor-Critic (high bias, low variance)
  - $\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi[\log \pi(A_t|S_t, \boldsymbol{\theta}) \delta_t]$
  - Q Actor-Critic: TD error for action value function $\delta_t = R_{t+1} + \gamma \hat{q}_\pi(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}_\pi(S_t, A_t, \boldsymbol{w})$
  - Advantage Actor-Critic (A2C): TD error for state value function $\delta_t = R_{t+1} + \gamma \hat{v}_\pi(S_{t+1}, \boldsymbol{w}) - \hat{v}_\pi(S_t, \boldsymbol{w})$ (sample-based estimate of the advantage function $A_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$)
- MC REINFORCE is a special case of A2C:
  - 1-step TD target (A2C): $R_{t+1} + \gamma \hat{v}_\pi(S_{t+1}, \boldsymbol{w})$
  - m-step TD target: $G_t^\lambda \doteq \sum_{k=0}^{m-1} \gamma^k R_{t+k+1} + \gamma^m \hat{v}_\pi(S_{t+m})$
  - MC target (MC REINFORCE): $G_t \doteq \sum_{k=0}^{T-1} \gamma^k R_{t+k+1}$

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

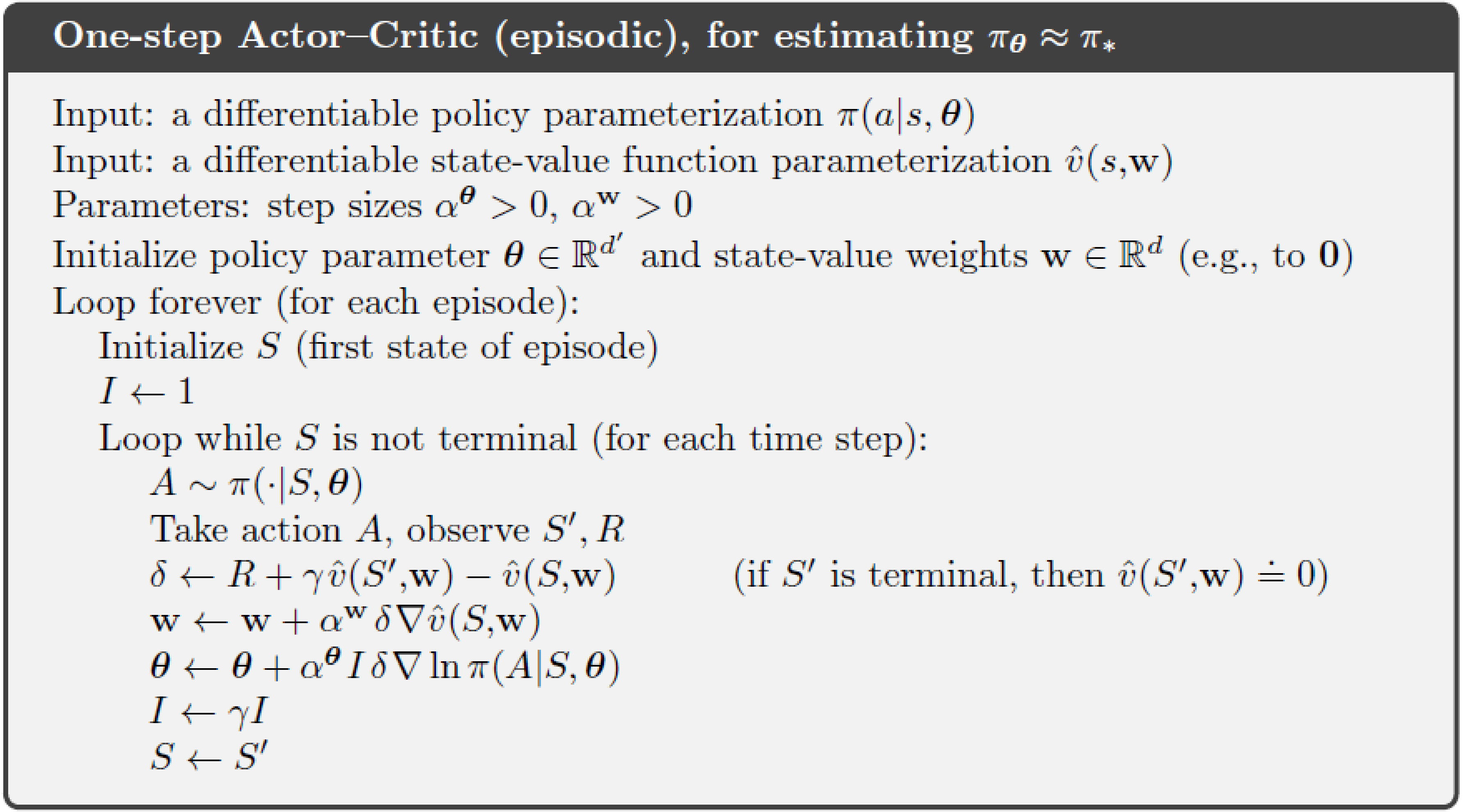
Further Explanations

- MC REINFORCE:
  - Trajectory from time t:
  - $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \dots, S_{T-1}, A_{T-1}, R_T, S_T$
  - $G_t \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$
  - $\mathbb{E}_\pi[G_t | S_t, A_t] = q_\pi(S_t, A_t)$
- Q Actor-Critic:
  - Bellman Exp Equation for Action Value Function:
  - $q_\pi(s,a) = \sum_{r,s'} p(r,s'|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s')\, q_\pi(s',a')\right]$
  - Sample-based estimate (TD target for action value function):
  - $q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$
- A2C
  - Bellman Exp Equation for State Value Function:
  - $v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r,s'|s,a) \left[r + \gamma v_\pi(s')\right]$
  - Sample-based estimate (TD target for state value function) :
  - $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$

https://www.cellstrat.com/2020/03/19/rl-with-actor-critic-methods/

One-Step A2C Pseudo-Code

- For update to critic params $\boldsymbol{w}$, refer to L7.2 Value-based RL, p 75 “Semi-Gradient TD(0) for Estimating $\hat{v} \approx v_\pi$”

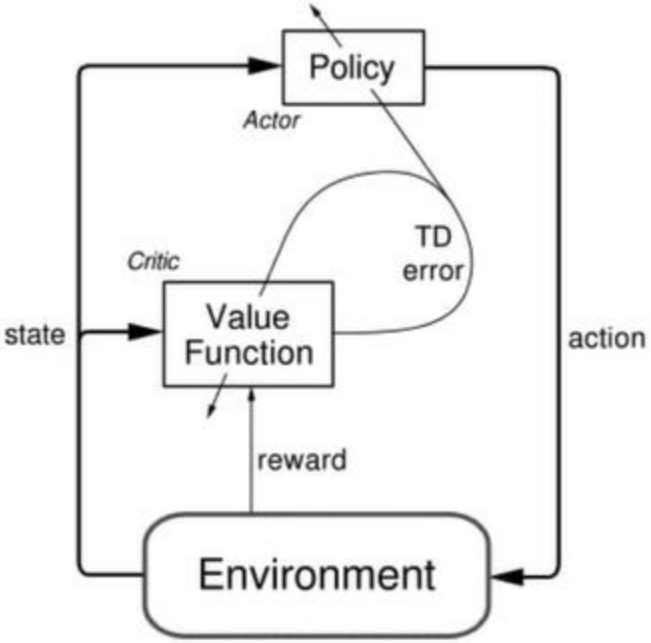


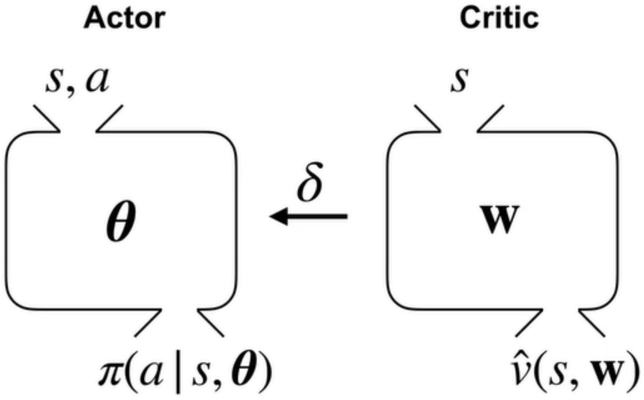
One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla\ln\pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

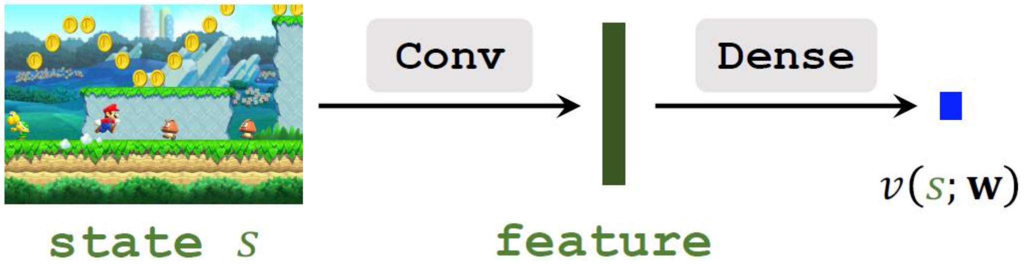
A2C Explanations

- After each step of taking action A_t in state S_t:
- Critic computes TD error $\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})$, and updates its params with semi-gradient TD(0) $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^w \delta_t \nabla_w \hat{v}(S_t, \boldsymbol{w})$ (learning rate α^w)
- Actor updates its params with Policy Gradient $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha^\theta \gamma^t \delta_t \nabla_\theta \log \pi(A_t|S_t, \boldsymbol{\theta_t})$. If $\delta_t > 0$, then it means A_t resulted in a higher (one-step estimate) value than the expected $\hat{v}(S_t, \boldsymbol{w})$, so probability of A_t in state S_t is increased; if $\delta_t < 0$, it is decreased (learning rate α^θ)
- Actor and Critic learn at the same time, constantly interacting. The actor is continually changing the policy params $\boldsymbol{\theta}$ to exceed the critic's expectation, and the critic is constantly updating its value function params $\boldsymbol{w}$ to evaluate the actor's changing policy.

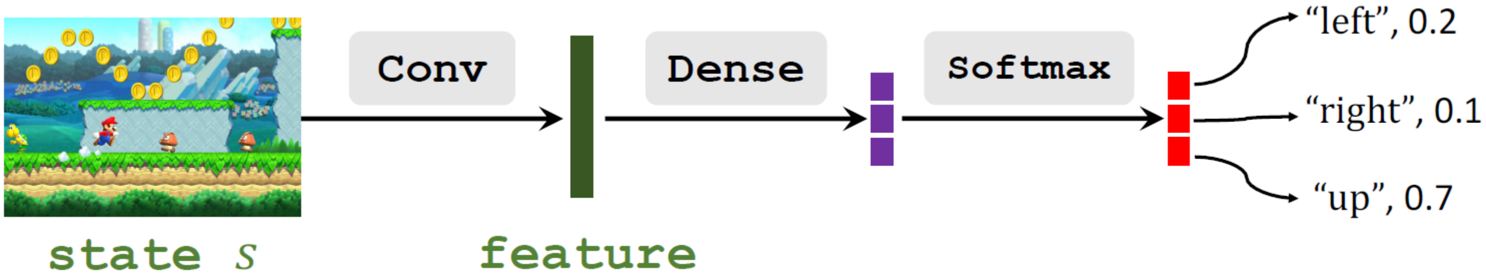




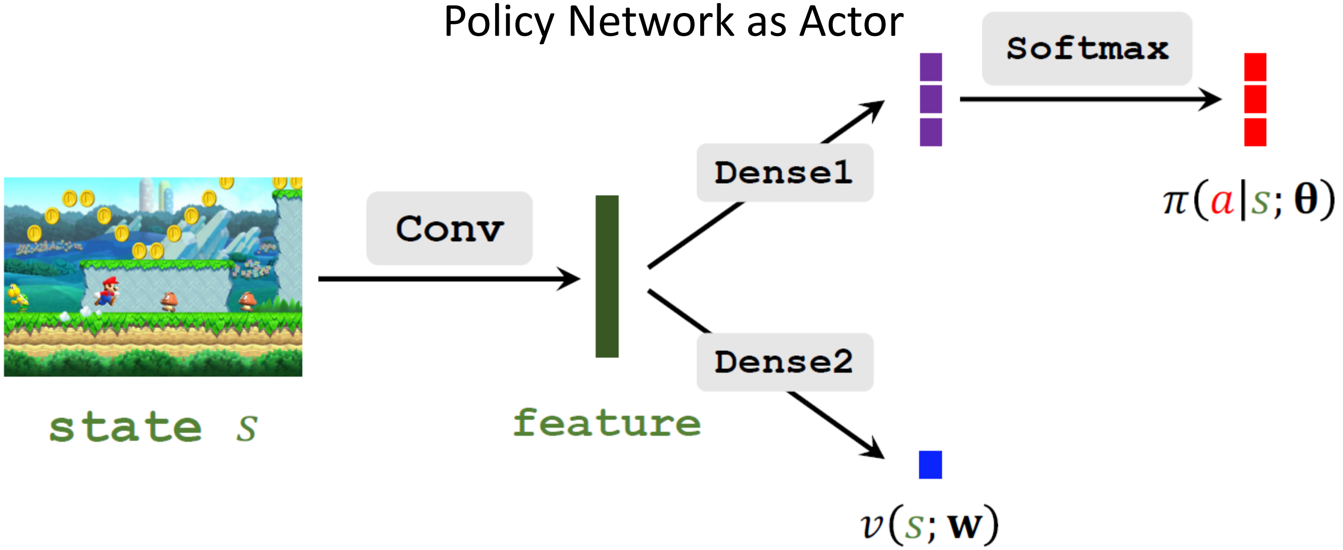
Function Approximations for Critic and Actor



Value Network as Critic



Policy Network as Actor

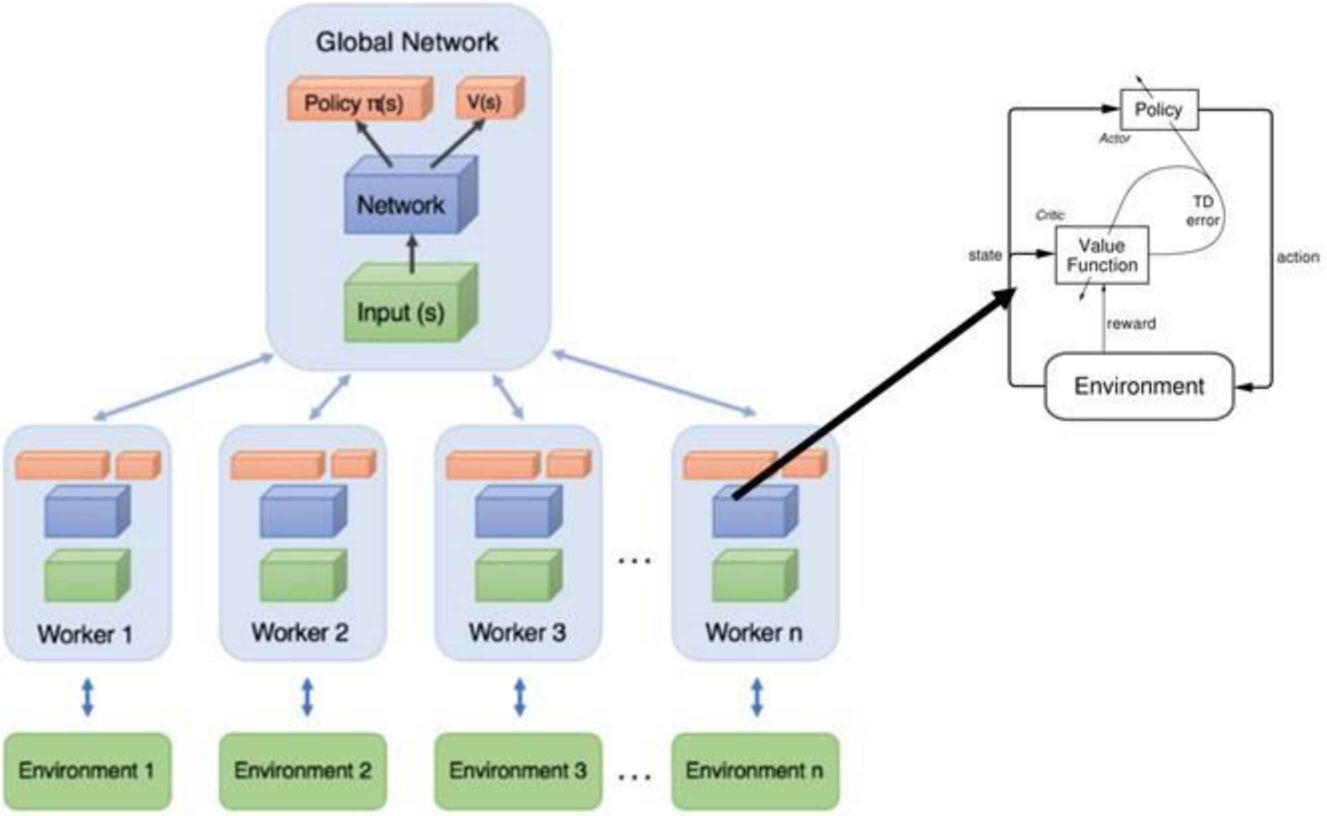


Parameter Sharing between Value and Policy Networks

https://github.com/wangshusen/DRL

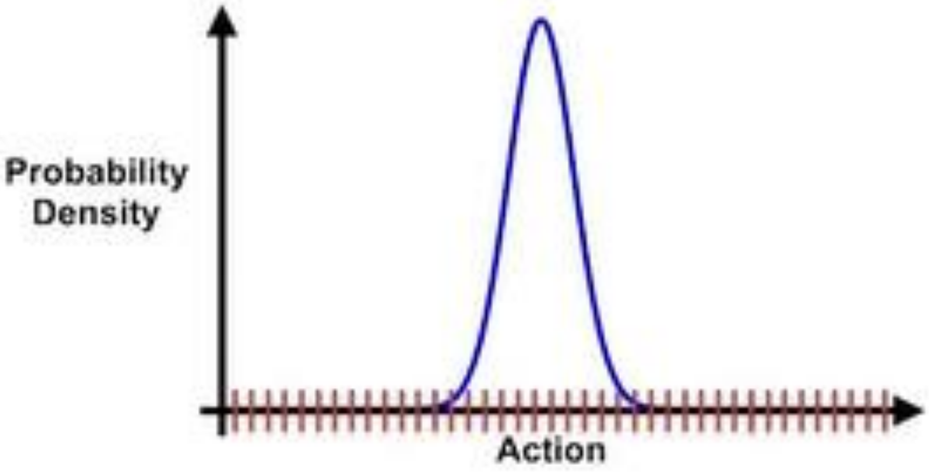
Asynchronous Advantage Actor Critic (A3C)

- A3C implements parallel training where multiple workers in parallel environments independently update the global value and policy networks, for effective and efficient exploration of the state space.



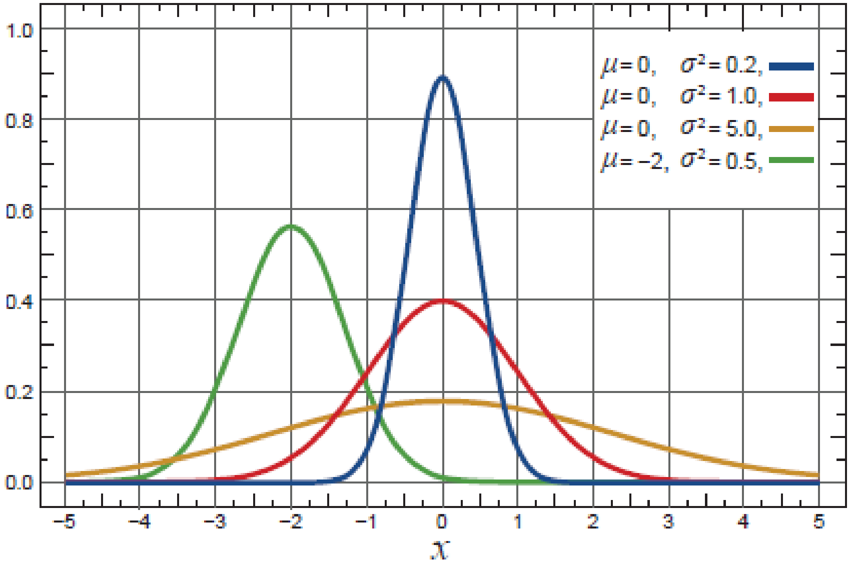
Continuous Actions

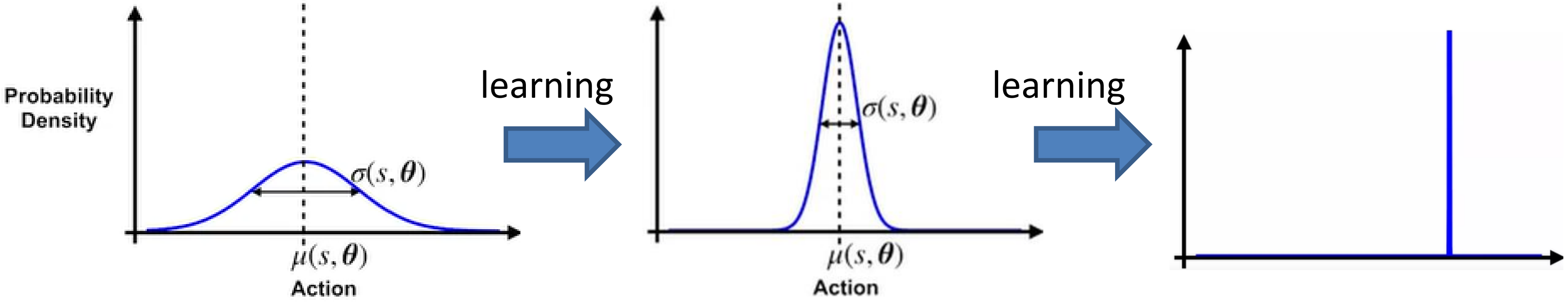
- It might not be straightforward to choose a proper discrete set of actions
- Continuous actions allow us to generalize over actions
  - If an action is good, its neighboring actions are also likely to be good
  - Discrete actions lack generalization: each action is independent of others, including its neighbors (similar to value functions for discrete states)



Gaussian Policy for Continuous Actions

- Gaussian Policy $\pi(a|s,\boldsymbol{\theta}) \doteq \frac{1}{\sigma(s,\boldsymbol{\theta})\sqrt{2\pi}}\exp(-\frac{(a-\mu(s,\boldsymbol{\theta}))^2}{2\sigma(s,\boldsymbol{\theta})^2})$
  - Mean $\mu(s,\theta)$ is the most likely action
  - Variance $\sigma(s,\theta)^2$ controls the degree of exploration.




Probability Density
$\sigma(s,\boldsymbol{\theta})$
$\mu(s,\boldsymbol{\theta})$
Action
learning
$\sigma(s,\boldsymbol{\theta})$
$\mu(s,\boldsymbol{\theta})$
Action
learning

Policy variance initially large, more exploration

Variance gradually reduced during learning w. PG, converging towards deterministic policy $a=\mu(s,\theta)$