

Lecture 1.7

Reinforcement Learning: Policy Gradient Methods

Alexey Gruzdev
alexey.s.gruzdev@gmail.com
HSE, Winter 2019

Illustration example: Left or right?



What is $Q(s, \text{right})$? $\gamma = 0.99$



Why not using Q-learning everywhere?

$$L \approx E[Q(s_t, a_t) - (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'))]^2$$

Simple 2-state world

	True	(A)	(B)
$Q(s_0, a_0)$	1	1	2
$Q(s_0, a_1)$	2	2	1
$Q(s_1, a_0)$	3	3	3
$Q(s_1, a_1)$	100	50	100

Evolution: how humans not survived

$\operatorname{argmax}[$
Q(s,pet the tiger)
Q(s,run from tiger)
Q(s,provoke tiger)
Q(s,ignore tiger)
]



Evolution: how humans survived

$$\pi(run|s)=1$$

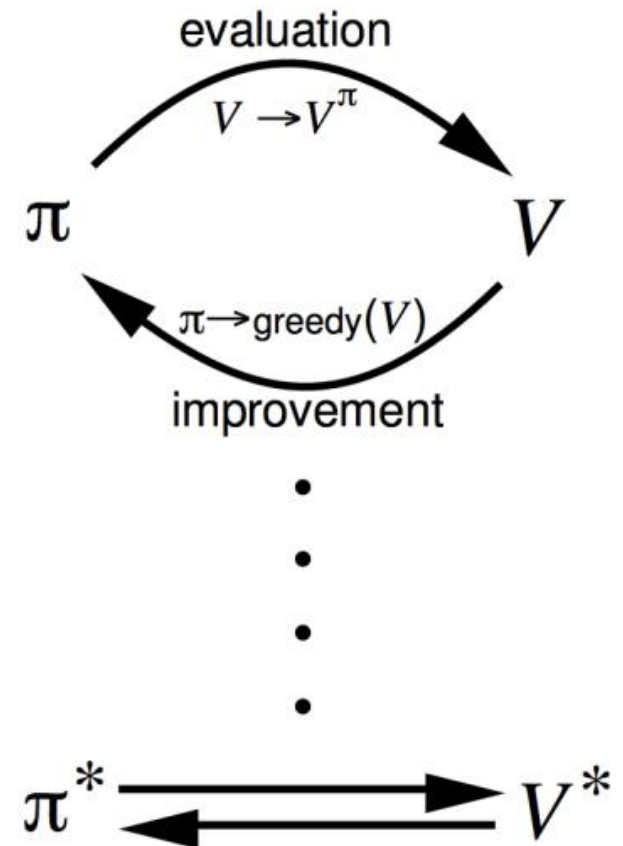
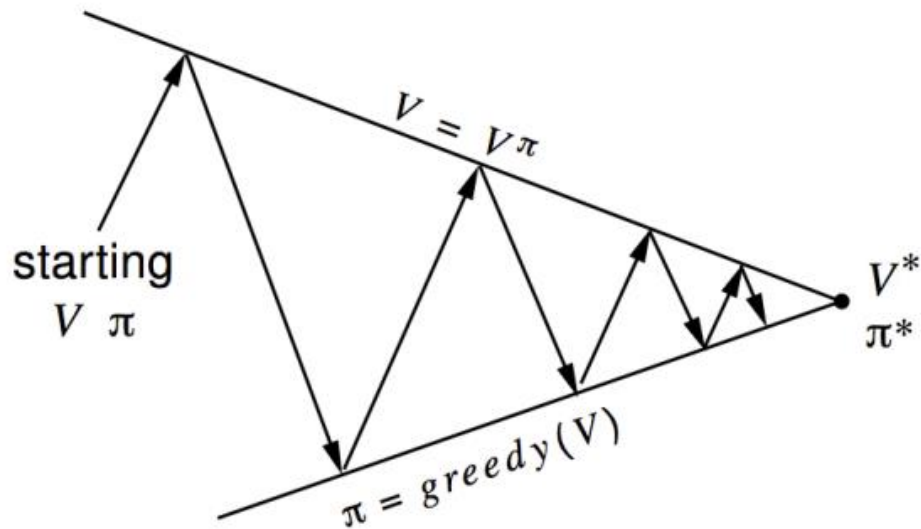


Outline

- Often computing q-values is harder than picking optimal actions!
- We could avoid learning value functions by directly learning agent's policy

Q: what algorithm works that way?

Generalized Policy Iteration



Policy evaluation Estimate v_π

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

Policy-based RL

- In previous lectures we approximated the value or action-value function using parameters:

$$\begin{aligned} V_{\theta}(s) &\approx V_{\pi}(s) \\ Q_{\theta}(s, a) &\approx Q_{\pi}(s, a) \end{aligned}$$

- A policy was generated directly from the value function e.g. using ϵ -greedy
- In this lecture we will directly parametrize the policy

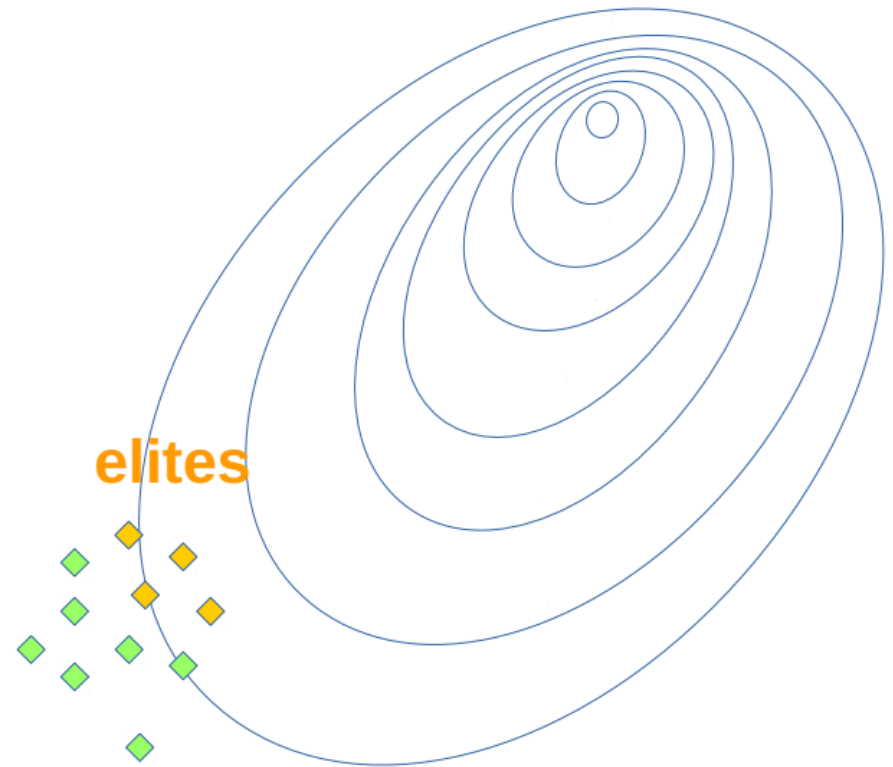
$$\pi_{\theta}(s, a) = P[a \mid s, \theta]$$

- We will focus again on **model-free** reinforcement learning

Recap: Cross Entropy Method

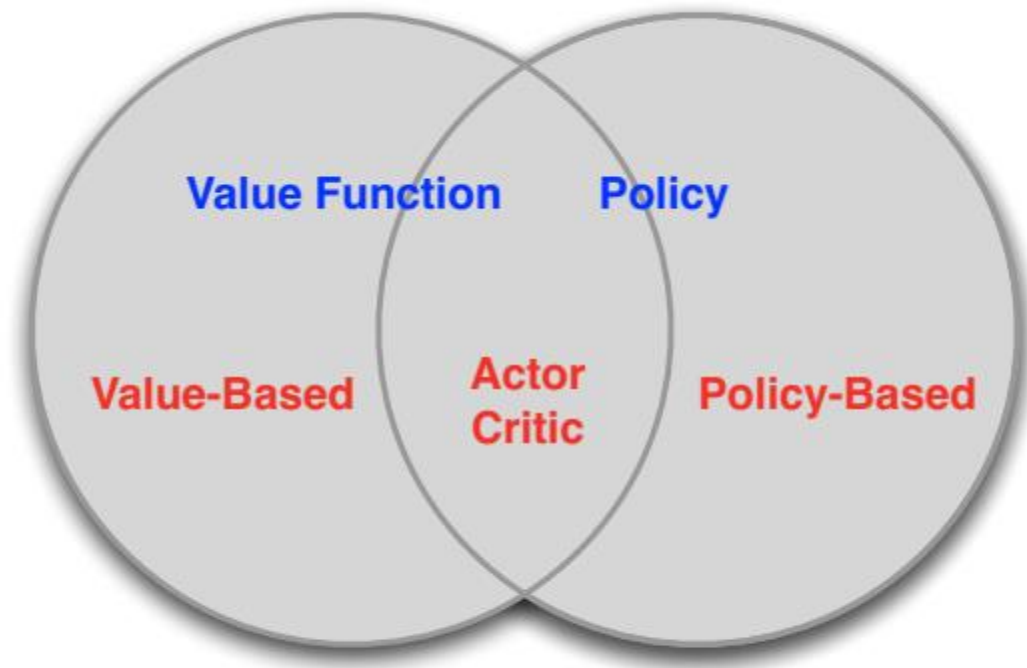
■ CEM:

- Evolutionary
- Go in the direction where elite goes.
- Easy to implement
- Black Box
- Need to play full episode to start learning



Value-based & 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



Advantages of Policy-Based RL

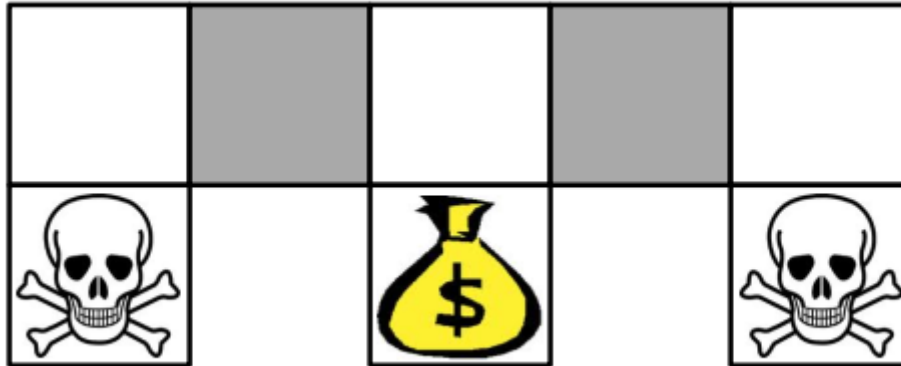
- Advantages:
 - Better convergence properties
 - Effective in high-dimensional or continuous action spaces
 - Can learn stochastic policies
- Disadvantages:
 - Typically converge to a local rather than global optimum
 - Evaluating a policy is typically inefficient and high variance

Example: Rock-Paper-Scissors



- Two-player game of rock-paper-scissors
 - Scissors beats paper
 - Rock beats scissors
 - Paper beats rock
- Consider policies for *iterated* rock-paper-scissors
 - A deterministic policy is easily exploited
 - A uniform random policy is optimal

Example #2: Aliased Grid world



- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

$$\varphi(s; a) = \mathbf{1}(\text{wall to } N, a = \text{move } E)$$

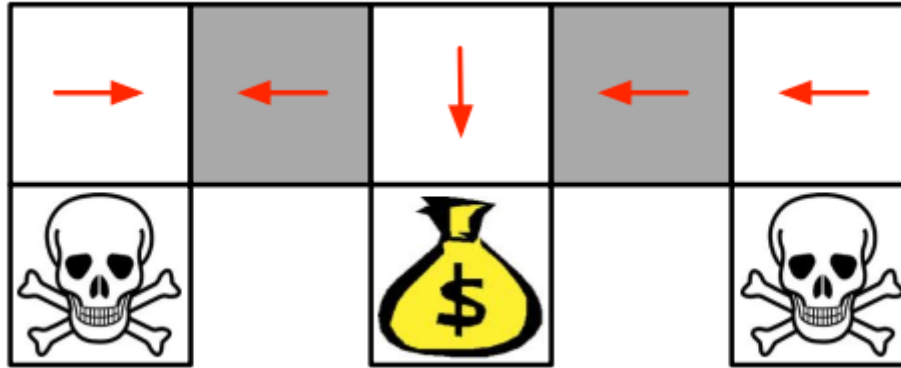
- Compare value-based RL, using an approximate value function

$$Q_{\theta}(s; a) = f(\varphi(s, a), \theta)$$

- To policy-based RL, using a parametrized policy

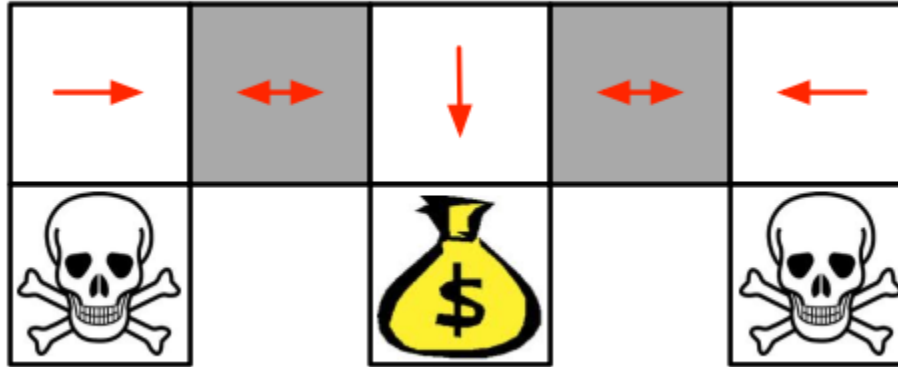
$$\pi_{\theta}(s, a) = g(\varphi(s, a), \theta)$$

Example #2: Aliased Grid world



- Under aliasing, an optimal deterministic policy will either
 - move W in both grey states (shown by red arrows)
 - move E in both grey states
- Either way, it can get stuck and *never* reach the money
- Value-based RL learns a near-deterministic policy
 - e.g. greedy or ϵ -greedy
- So it will traverse the corridor for a long time

Example #2: Aliased Grid world



- An optimal stochastic policy will randomly move E or W in grey states:

$$\begin{aligned}\pi_{\theta}(\text{wall to N and S, move E}) &= 0.5 \\ \pi_{\theta}(\text{wall to N and S, move W}) &= 0.5\end{aligned}$$

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ , find best θ
But how do we measure the quality of a policy π_θ ?

- In episodic environments we can use the **start value**:

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta} [v_1]$$

- In continuing environments we can use the **average value**:

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the **average reward per time-step**:

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d\pi_\theta(s)$ is stationary distribution of Markov chain for π_θ

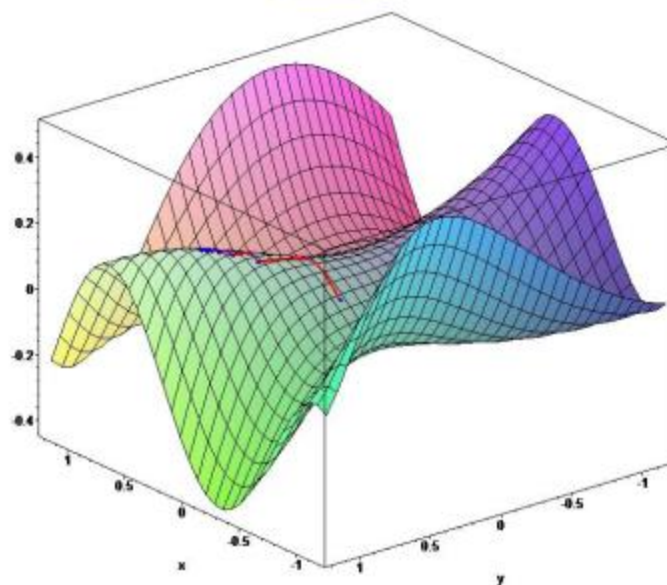
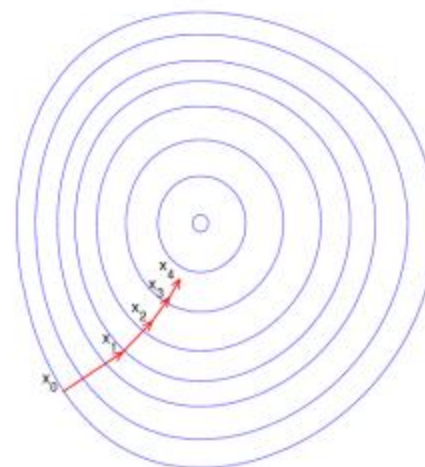
Policy Optimization

- Policy based reinforcement learning is an **optimization** problem
- Find Θ that maximizes $J(\Theta)$
- Some approaches do not use gradient
 - Hill climbing
 - Genetic algorithms
 - Simplex
- Greater efficiency often possible using gradient
 - Gradient descent
 - Conjugate gradient
 - Quasi-newton
- We focus on gradient descent, many extensions possible

Policy Gradient

- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a *local* maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\nabla_{\theta} J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$



Finite Differences approach

- To evaluate policy gradient of $\pi_{\theta}(s, a)$
- For each dimension $k \in [1, n]$
 - Estimate k th partial derivative of objective function w.r.t. θ
 - By perturbing θ by small amount ϵ in k th dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- where u_k is unit vector with 1 in k th component, 0 elsewhere
- Uses n evaluations to compute policy gradient in n dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable

Analytic Gradient Computation

- We now compute the policy gradient *analytically*
- Assume policy π_θ is differentiable whenever it is non-zero
- Log Derivative trick:

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

- We can express gradient of π_θ as function of π_θ .

One-step MDPs

- Consider a simple class of one-step MDPs
Starting in state $s \sim d(s)$
- Terminating after one time-step with reward $r = R_{s,a}$
- Use log-derivative trick to compute the policy gradient

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_{\theta}} [r] \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(s, a) \mathcal{R}_{s,a} \\ \nabla_{\theta} J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(s, a) \nabla_{\theta} \log \pi_{\theta}(s, a) \mathcal{R}_{s,a} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) r] \end{aligned}$$

Policy Gradient Theorem

- The policy gradient theorem generalizes the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q_\pi(s, a)$ 😊
Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

*For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

REINFORCE algorithm

- Update parameters by stochastic gradient **ascent** (!!)
- Using policy gradient theorem
- Using return G_t as an unbiased sample of $Q\pi_\theta(s_t, a_t)$

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

 Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$

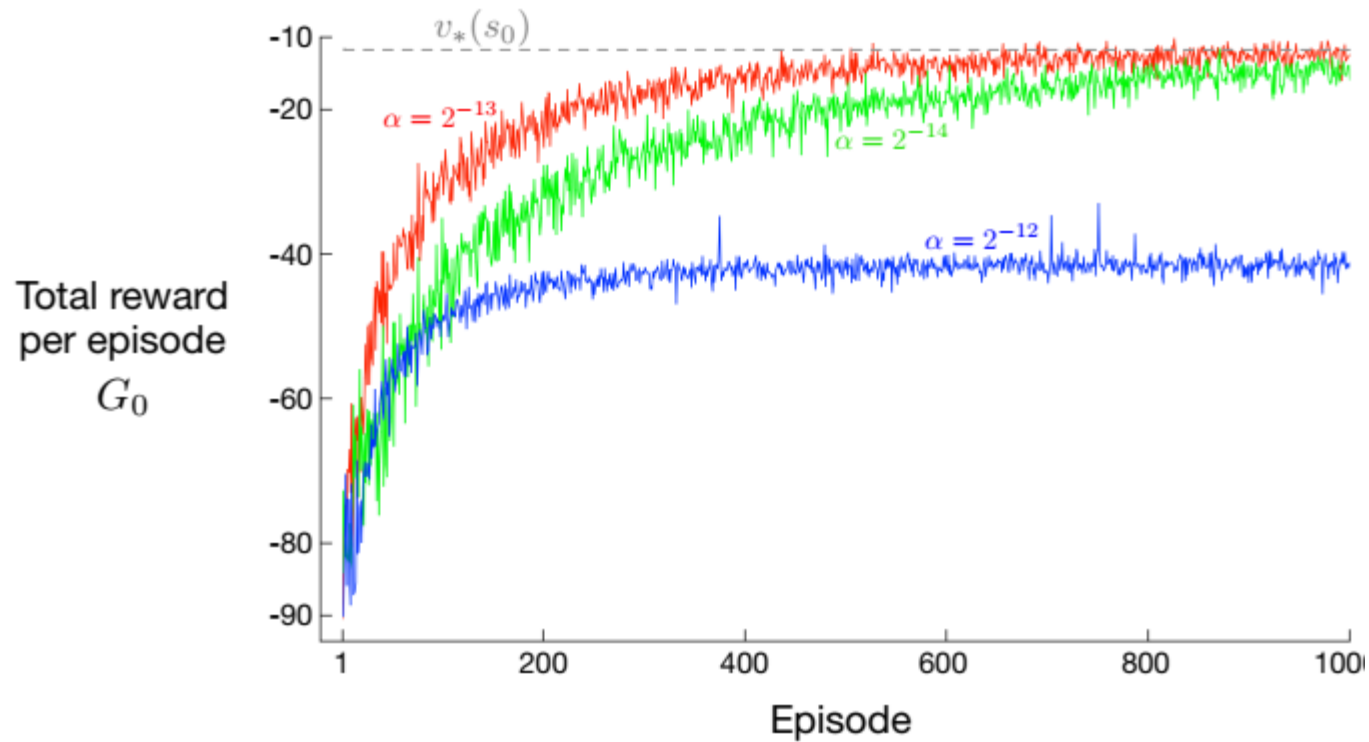
 Loop for each step of the episode $t = 0, \dots, T - 1$:

$G \leftarrow$ return from step t (G_t)

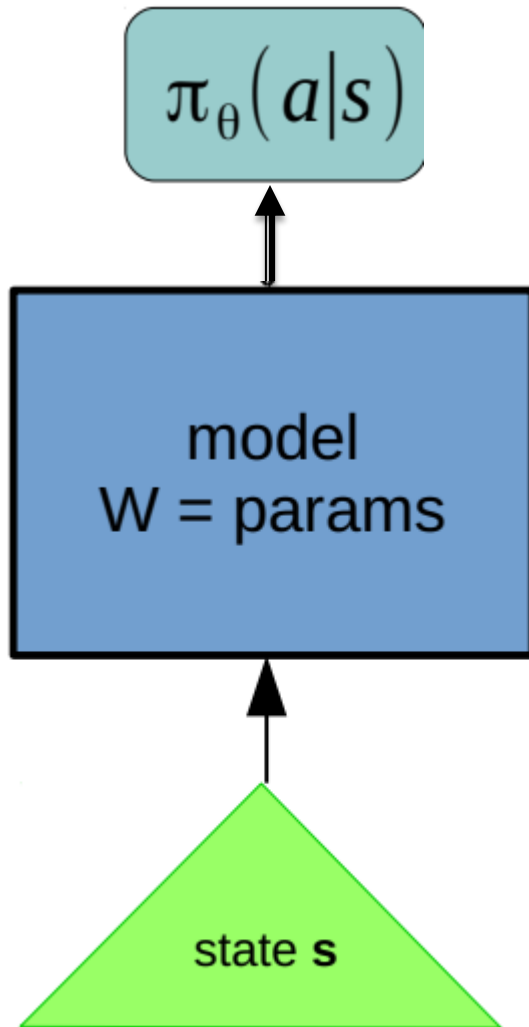
$\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \ln \pi(A_t|S_t, \theta)$

Q: Is it on-policy or off-policy?

REINFORCE training

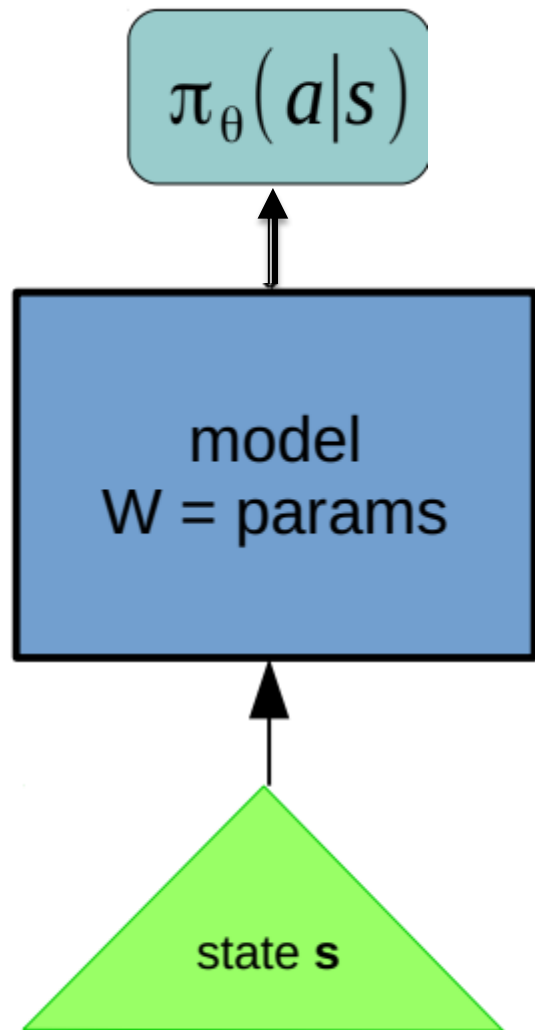


REINFORCE as NN



- Initialize NN weights
 - Θ – *random init*
- Cycle:
 - Sample N sessions \mathbf{z} under current policy $\pi_{\Theta}(s_{t'}, a_t)$
 - Evaluate policy gradient
$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_{\theta}(a|s) \cdot Q(s, a)$$
 - Apply gradient update

REINFORCE as NN with a baseline



- Initialize NN weights
 - Θ – random init
- Cycle:
 - Sample N sessions \mathbf{z} under current policy $\pi_{\Theta}(s_{t'}, a_t)$
 - Evaluate policy gradient
$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_{\theta}(a|s) \cdot (Q(s, a) - b(s))$$
 - Apply gradient update

REINFORCE with a baseline

REINFORCE with Baseline (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Algorithm parameters: step sizes $\alpha^{\theta} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

 Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$

 Loop for each step of the episode $t = 0, \dots, T - 1$:

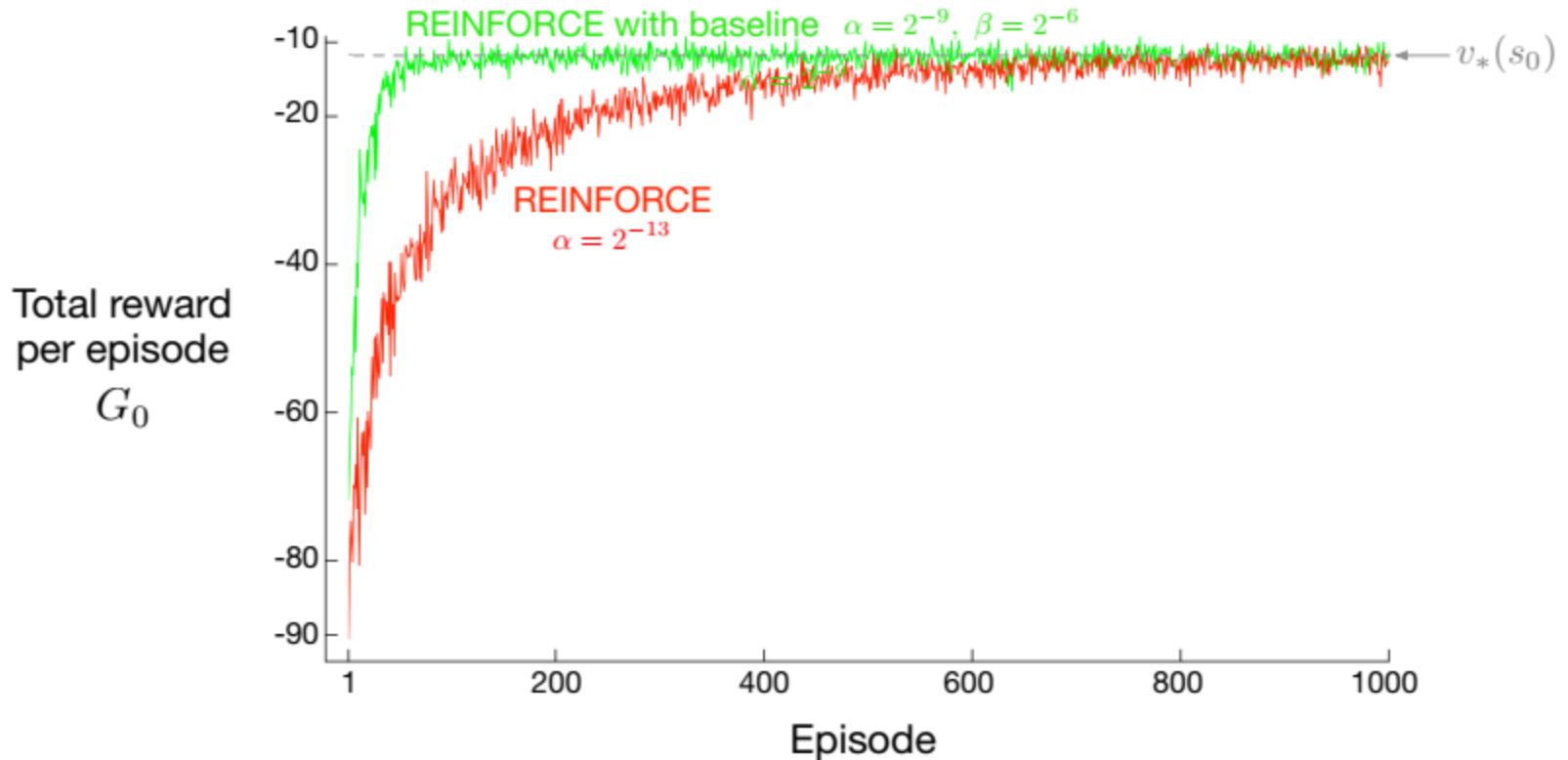
$G \leftarrow$ return from step t (G_t)

$\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$

$\theta \leftarrow \theta + \alpha^{\theta} \gamma^t \delta \nabla_{\theta} \ln \pi(A_t|S_t, \theta)$

REINFORCE with a baseline training



Next Method Name?



Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function

$$Q_w(s, a) \approx Q\pi_\theta(s, a)$$

- Actor-critic algorithms maintain *two* sets of parameters
 - Critic: Updates action-value function parameters w
 - Actor: Updates policy parameters θ , in direction suggested by critic
- Actor-critic algorithms follow an *approximate* policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$$

Estimating the action-value function

- The critic is solving a familiar problem: policy evaluation
- How good is policy π_{θ} for current parameters θ ?
- This problem was explored in previous two lectures, e.g.
 - Monte-Carlo policy evaluation
 - Temporal-Difference learning

Action-Value Actor-Critic

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s, a) = \phi(s, a)^\top w$
 - Critic Updates w by linear TD(0)
 - Actor Updates θ by policy gradient

function QAC

 Initialise s, θ

 Sample $a \sim \pi_\theta$

for each step **do**

 Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s'}^a$.

 Sample action $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

$w \leftarrow w + \beta \delta \phi(s, a)$

$a \leftarrow a', s \leftarrow s'$

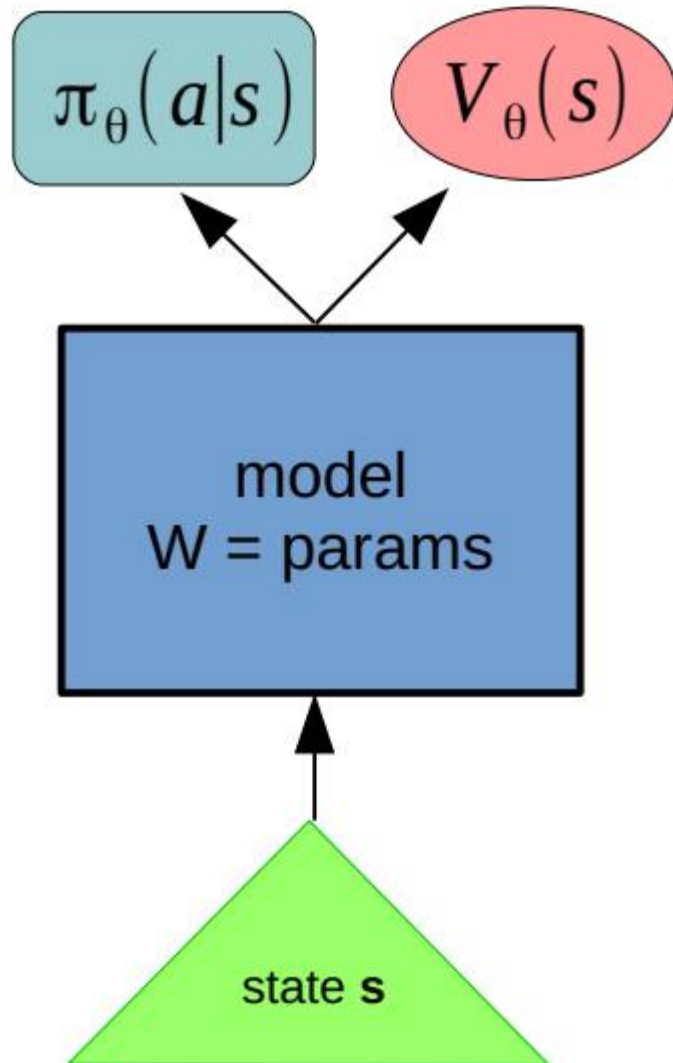
end for

end function

Bias in Actor-Critic Algorithms

- Approximating the policy gradient introduces bias
- A biased policy gradient may not find the right solution
 - e.g. if $Q_w(s, a)$ uses aliased features, can we solve gridworld example?
- Luckily, if we choose value function approximation carefully
- Then we can avoid introducing any bias
- i.e. We can still follow the *exact* policy gradient

Advantage Actor-Critic (aka A2C)



$$A(s,a) = Q(s,a) - V(s) = r + \gamma \cdot V(s') - V(s)$$

Improve policy:

$$\nabla J_{actor} \approx \frac{1}{N} \sum_{i=0}^N \sum_{s,a \in z_i} \nabla \log \pi_{\theta}(a|s) \cdot A(s,a)$$

Improve value:

$$L_{critic} \approx \frac{1}{N} \sum_{i=0}^N \sum_{s,a \in z_i} (V_{\theta}(s) - [r + \gamma \cdot V(s')])^2$$

Duct tape slide

- $V(s)$ errors less important than in Q-learning
 - actor still learns even if critic is random, just slower
- Regularize with entropy (hello exploration)
 - to prevent premature convergence
- learn on parallel sessions
 - or super-small experience replay
- Use `F.log_softmax()` for numerical stability

Summary of Policy Gradient methods

- The policy gradient has many equivalent forms:

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{g}_t] && \text{REINFORCE} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) Q^w(s, a)] && \text{Q Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) A^w(s, a)] && \text{Advantage Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta] && \text{TD Actor-Critic}\end{aligned}$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q_{\pi}(s, a)$, $A_{\pi}(s, a)$ or $V_{\pi}(s)$

Questions?

