# Deep Reinforcement Learning

## Lecture 8: Policy Gradient Methods

Instructor: Chongjie Zhang

Tsinghua University

# Materials Used

- Much of the material and slides for this lecture were taken from Chapter 13 of Barto & Sutton textbook.
- Some slides are borrowed from Rich Sutton's RL class and David Silver's Deep RL tutorial

# RL via Policy Gradient Search

- So far all of our RL techniques have tried to learn an exact or approximate value function or Q-function
  - Learn optimal value of being in a state, or taking an action from  state.
- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
- Value functions can often be much more complex to represent than the corresponding policy
  - Do we really care about knowing Q(s, left) = 0.3554, Q(s, right) = 0.533
  - Or just that "right is better than left in state s"
- Motivates searching directly in a parameterized policy space
  - Bypass learning value function and "directly" optimize the value of a policy

# Policy Search

# Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters $\theta$,

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s, a) \approx Q^\pi(s, a)$$

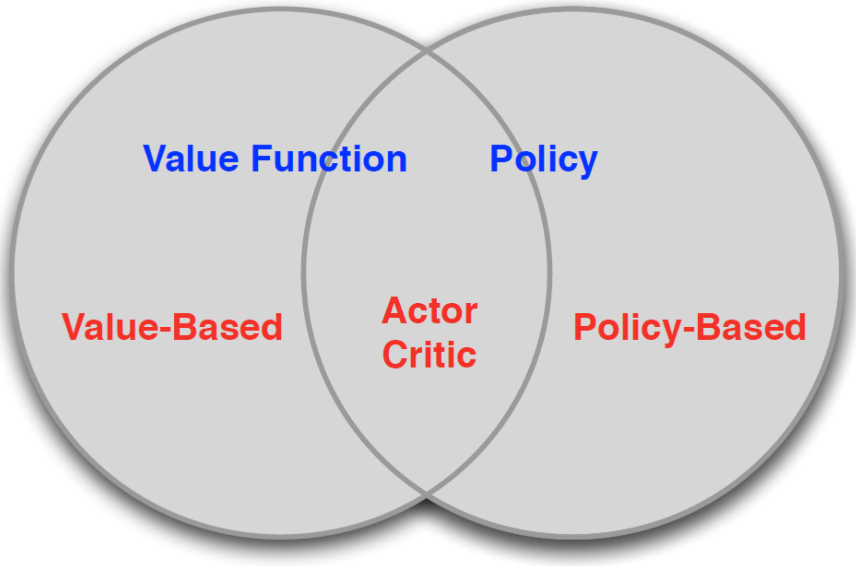
- A policy was generated directly from the value function
  - e.g. using $\epsilon$-greedy
- In this lecture we will directly parametrise the policy

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

- We will focus again on model-free reinforcement learning

# Value-Based and Policy-Based RL

- Value Based
  - Learning Value Function
  - Implicit Policy (e.g., ε-greedy)
- Policy Based
  - No Value Function
  - Directly learning policy
- Actor-Critic
  - Learning Value Function
  - Learning Policy



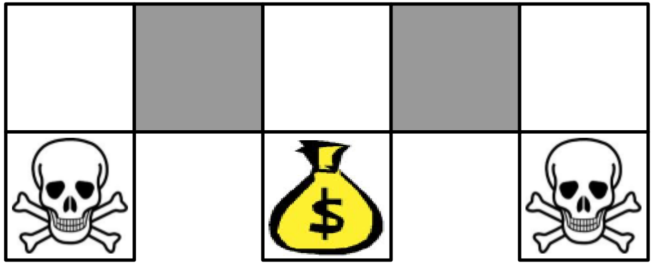
# Why Policy-Based RL?

- Observation: often the feature-based policies that work well (win games, maximize utilities) aren’t the ones that approximate V / Q best
- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

# 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 (i.e.  Nash equilibrium)

# Example: Aliased Gridworld (1)



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

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

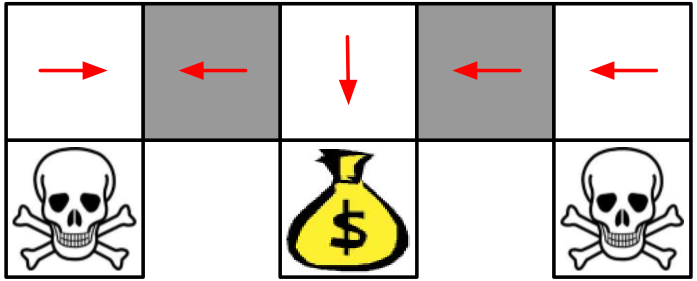
- Compare value-based RL, using an approximate value function

$$Q_\theta(s, a) = f(\phi(s, a), \theta)$$

- To policy-based RL, using a parametrised policy

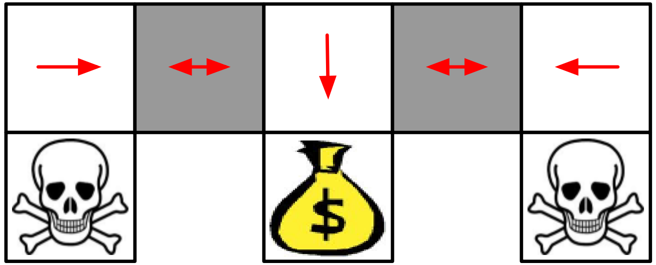
$$\pi_\theta(s, a) = g(\phi(s, a), \theta)$$

# Example: Aliased Gridworld (2)



- 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 $\epsilon$-greedy
- So it will traverse the corridor for a long time

# Example: Aliased Gridworld (3)



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

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

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

# RL via Policy Gradient Ascent

- The policy gradient approach has the following schema:
  1. Select a space of parameterized policies
  2. Compute the gradient of the value of current policy wrt parameters
  3. Move parameters in the direction of the gradient
  4. Repeat these steps until we reach a local maxima
  5. Possibly also add in tricks for dealing with bad local maxima (e.g. random restarts)
- So we must answer the following questions:
  - How should we represent and evaluate parameterized policies?
  - How can we compute the gradient?

# What is Policy Learning Objective?

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

# Policy Optimization

- Policy based reinforcement learning is an optimisation problem
- Find $\theta$ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

# Policy Gradient

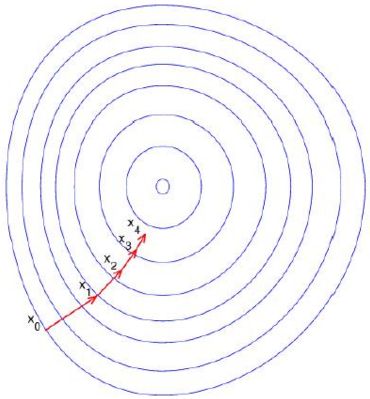
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a *local* maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters $\theta$

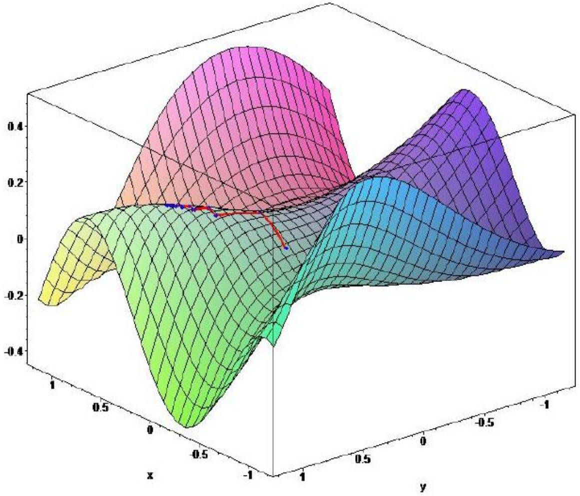
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- Where $\nabla_\theta J(\theta)$ is the policy gradient

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

- and $\alpha$ is a step-size parameter



# Today’s Lecture

- Introduction to Policy Learning

- Policy Gradient Methods
  - **Finite Difference Policy Gradient**
  - Monte-Carlo Policy Gradient (REINFORCE)
- Actor-Critic Methods
  - Q Actor-Critic
  - Advantage Actor-Critic

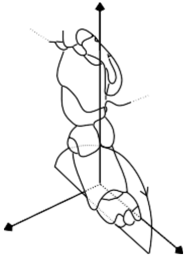
# Computing Gradients By Finite Differences

- 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. $\theta$
  - By perturbing $\theta$ by small amount $\epsilon$ 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

# Training AIBO to Walk by Finite Difference Policy Gradient



- Goal: learn a fast AIBO walk (useful for Robocup)
- AIBO walk policy is controlled by 12 numbers (elliptical loci)
- Adapt these parameters by finite difference policy gradient
- Evaluate performance of policy by field traversal time

# Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

# Example: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

# Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

# Today's Lecture

- Introduction to Policy Learning

- Policy Gradient Methods
  - Finite Difference Policy Gradient
  - **Monte-Carlo Policy Gradient (REINFORCE)**
- Actor-Critic Methods
  - Q Actor-Critic
  - Advantage Actor-Critic

# Score Function

- We now compute the policy gradient *analytically*
- Assume policy $\pi_\theta$ is differentiable whenever it is non-zero
- and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Likelihood ratios exploit the following identity

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

- The score function is $\nabla_\theta \log \pi_\theta(s, a)$

# Typical Parameterized Differential Policy: Softmax

- We will use a softmax policy as a running example
- Weight actions using linear combination of features $\phi(s, a)^\top \theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s, a) \propto e^{\phi(s,a)^\top \theta}$$

- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \phi(s, a) - \mathbb{E}_{\pi_\theta}\left[\phi(s, \cdot)\right]$$

This is because:

SOFTMAX $\Rightarrow$ $\pi(s,a;\theta) = \exp(\phi(s,a)^t\theta) \Big/ \sum_b \exp(\phi(s,b)^t\theta)$

$$\therefore \log \pi_\theta = \phi(s,a)^t\theta - \log \sum_b \exp(\phi(s,b)^t\theta)$$

$$\therefore \nabla_\theta \log \pi_\theta = \phi(s,a) - \frac{1}{\sum_b \exp(\phi(s,b)^t\theta)} \sum_b \phi(s,b)\exp(\phi(s,b)^t\theta)$$

$$= \phi(s,a) - \frac{1}{\underbrace{\sum_b \pi_\theta(s,b)}_{=1}} \sum_b \phi(s,b)\,\pi_\theta(s,b)$$

$$= \phi(s,a) - \mathbb{E}_{\pi_\theta}(\phi(s,\cdot))$$

# Typical Parameterized Differential Policy: Gaussian

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\mu(s) = \phi(s)^\top \theta$
- Variance may be fixed $\sigma^2$, or can also parametrised
- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

# 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 $r = \mathcal{R}_{s,a}$
- Use likelihood ratios to compute the policy gradient

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

likelihood ratio

# Policy Gradient Theorem

- The policy gradient theorem generalises 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*

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

# Policy Gradient Theorem

- $\nabla_\theta V^\pi(s)$
- $= \nabla_\theta \left(\sum_{a \in A} \pi_\theta(a|s) Q^\pi(s,a)\right)$
- $= \sum_{a \in A} (\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \nabla_\theta Q^\pi(s,a))$
- $= \sum_{a \in A} (\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \nabla_\theta \sum_{s',r} P(s',r|s,a)(r + V^\pi(s')))$
- $= \sum_{a \in A} (\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \sum_{s',r} P(s',r|s,a) \nabla_\theta V^\pi(s'))$
- $= \sum_{a \in A} (\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V^\pi(s'))$

- Notation:
- $\phi(s) = \sum_{a \in A} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a)$

# Policy Gradient Theorem

- $\nabla_\theta V^\pi(s)$
- $= \phi(s) + \sum_{a\in A} \pi_\theta(a|s) \sum_{s'} P(s'|s,a)\nabla_\theta V^\pi(s')$
- $= \phi(s) + \sum_{s'} \sum_a \pi_\theta(a|s) P(s'|s,a)\nabla_\theta V^\pi(s')$
- $= \phi(s) + \sum_{s'} \rho^\pi(s \to s', 1)\nabla_\theta V^\pi(s')$
- $= \phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \,[\phi(s') + \sum_{s''} \rho^\pi(s' \to s'', 1)\nabla_\theta V^\pi(s'')]$
- $= \phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \,\phi(s') + \sum_{s''} \rho^\pi(s \to s'', 2)\nabla_\theta V^\pi(s'')]$
- $= \cdots$
- $= \sum_{x\in S} \sum_{k=0}^{\infty} \rho^\pi(s \to x, k)\phi(x)$

$$\phi(s) = \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a)$$

# Policy Gradient Theorem

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta V^\pi(s_0) && \text{; Starting from a random state } s_0 \\
&= \sum_s \sum_{k=0}^{\infty} \rho^\pi(s_0 \to s, k)\phi(s) && \text{; Let } \eta(s) = \textstyle\sum_{k=0}^{\infty} \rho^\pi(s_0 \to s, k) \\
&= \sum_s \eta(s)\phi(s) && \\
&= \Big(\sum_s \eta(s)\Big) \sum_s \frac{\eta(s)}{\sum_s \eta(s)}\phi(s) && \text{; Normalize } \eta(s), s \in \mathcal{S} \text{ to be a probability distribution.} \\
&\propto \sum_s \frac{\eta(s)}{\sum_s \eta(s)}\phi(s) && \textstyle\sum_s \eta(s) \text{ is a constant} \\
&= \sum_s d^\pi(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) && d^\pi(s) = \frac{\eta(s)}{\sum_s \eta(s)} \text{ is stationary distribution.}
\end{aligned}
$$

# Policy Gradient Theorem

**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*

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

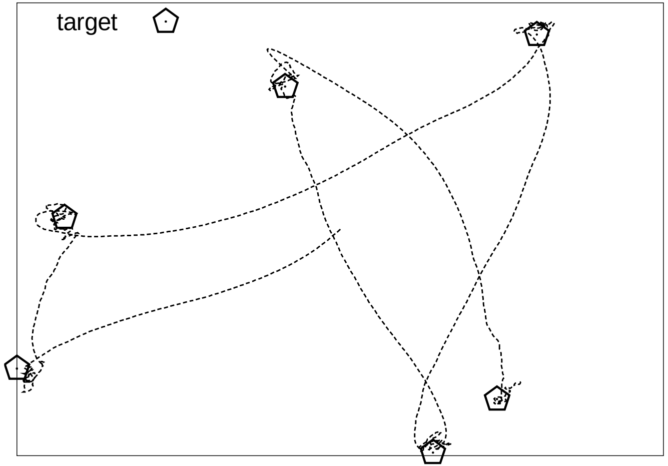
# Monte-Carlo Policy Gradient (REINFORCE)

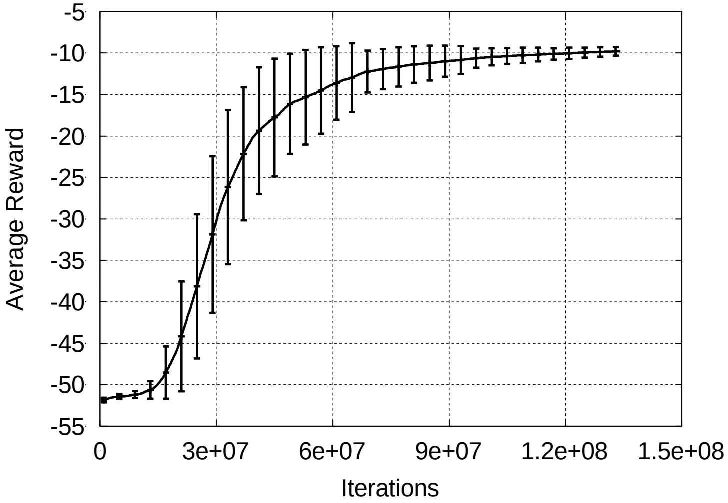
- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return $v_t$ as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

**function REINFORCE**
    Initialise $\theta$ arbitrarily
    **for** each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        **for** $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        **end for**
    **end for**
    **return** $\theta$
**end function**

# Puck World Example




- Continuous actions exert small force on puck
- Puck is rewarded for getting close to target
- Target location is reset every 30 seconds
- Policy is trained using variant of Monte-Carlo policy gradient

# 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

# Today's Lecture

- Introduction to Policy Learning

- Policy Gradient Methods
  - Finite Difference Policy Gradient
  - Monte-Carlo Policy Gradient (REINFORCE)

- **Actor-Critic Methods**
  - **Q Actor-Critic**
  - Advantage Actor-Critic

# Reducing Variance Using a 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 $\theta$, in direction suggested by critic

- Actor-critic algorithms follow an *approximate* policy gradient

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

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

# Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- How good is policy $\pi_\theta$ for current parameters $\theta$?
- This problem was explored in the previous lecture, e.g.
  - TD-based approach
- Could also use methods such as least-squares policy evaluation

# 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 $\theta$ by policy gradient

**function** $\mathrm{QAC}$
    Initialise $s$, $\theta$
    Sample $a \sim \pi_\theta$
    **for** each step **do**
        Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
        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**

# Today’s Lecture

- Introduction to Policy Learning
- Policy Gradient Methods
  - Finite Difference Policy Gradient
  - Monte-Carlo Policy Gradient (REINFORCE)
- Actor-Critic Methods
  - Q Actor-Critic
  - **Advantage Actor-Critic**

# Reducing Variance Using a Baseline

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance, without changing expectation

$$
\begin{aligned}
\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) B(s)\right] &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a) B(s) \\
&= \sum_{s\in\mathcal{S}} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(s,a) \\
&= 0
\end{aligned}
$$

- A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- So we can rewrite the policy gradient using the advantage function $A^{\pi_\theta}(s,a)$

$$
\begin{aligned}
A^{\pi_\theta}(s,a) &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \; A^{\pi_\theta}(s,a)\right]
\end{aligned}
$$

# Estimating the Advantage Function (1)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating *both* $V^{\pi_\theta}(s)$ *and* $Q^{\pi_\theta}(s,a)$
- Using two function approximators and two parameter vectors,

$$
\begin{aligned}
V_v(s) &\approx V^{\pi_\theta}(s) \\
Q_w(s,a) &\approx Q^{\pi_\theta}(s,a) \\
A(s,a) &= Q_w(s,a) - V_v(s)
\end{aligned}
$$

- And updating *both* value functions by e.g. TD learning

# Estimating the Advantage Function (2)

- For the true value function $V^{\pi_\theta}(s)$, the TD error $\delta^{\pi_\theta}$

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

- is an unbiased estimate of the advantage function

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta}|s,a\right] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V^{\pi_\theta}(s')|s,a\right] - V^{\pi_\theta}(s)\\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)\\ &= A^{\pi_\theta}(s,a)\end{aligned}$$

- So we can use the TD error to compute the policy gradient

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

- In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- This approach only requires one set of critic parameters $v$

# Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

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

- 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 $V^\pi(s)$

But will it converge if we use function approximation??

Under what conditions??

# Bias in Actor-Critic Algorithms

- Approximating the policy gradient introduces bias

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

- A biased policy gradient may not find the right solution
- 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

# Compatible Function Approximation

- If the following two conditions are satisfied:

  1. Value function approximator is compatible with the policy

$$\boxed{\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)}$$

  2 Value function parameters w minimize the mean-squared error

$$\boxed{\varepsilon = \mathbb{E}_{\pi_\theta}\left[(Q^{\pi_\theta}(s,a) - Q_w(s,a))^2\right]}$$

- Then the policy gradient is exact,

$$\boxed{\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q_w(s,a)\right]}$$

- Remember:

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

# Proof of Compatible Function Approximation Theorem

If $w$ is chosen to minimise mean-squared error, gradient of $\varepsilon$ w.r.t. $w$ must be zero,

$$\begin{aligned}
\nabla_w \varepsilon &= 0 \\
\mathbb{E}_{\pi_\theta}\left[(Q^\theta(s,a) - Q_w(s,a))\nabla_w Q_w(s,a)\right] &= 0 \\
\mathbb{E}_{\pi_\theta}\left[(Q^\theta(s,a) - Q_w(s,a))\nabla_\theta \log \pi_\theta(s,a)\right] &= 0 \\
\mathbb{E}_{\pi_\theta}\left[Q^\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\right] &= \mathbb{E}_{\pi_\theta}\left[Q_w(s,a)\nabla_\theta \log \pi_\theta(s,a)\right]
\end{aligned}$$

So $Q_w(s,a)$ can be substituted directly into the policy gradient,

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

# Compatible Function Approximation

- If the following two conditions are satisfied:

  1. Value function approximator is compatible with the policy

  $$\boxed{\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)}$$

  2 Value function parameters w minimize the mean-squared error

  $$\boxed{\varepsilon = \mathbb{E}_{\pi_\theta}\left[\left(Q^{\pi_\theta}(s,a) - Q_w(s,a)\right)^2\right]}$$

- Then the policy gradient is exact,

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

- Remember:

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

note error ε need not be zero, just needs to be minimized!

note we only need

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

to within a constant!

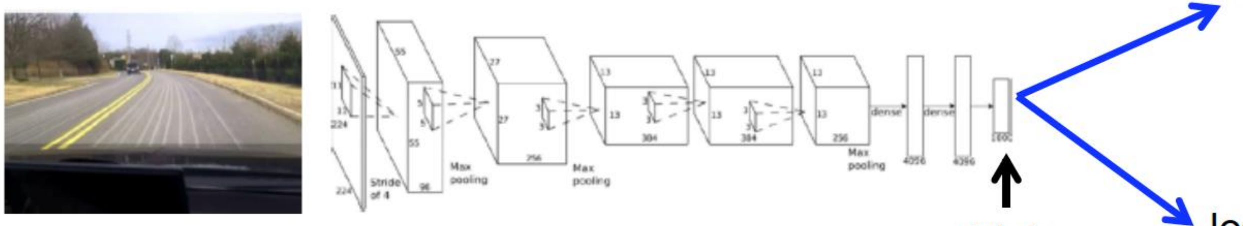
# Compatible Function Approximation

$$\boxed{\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)}$$

How can we achieve this??

One way: make $Q_w$ and $\pi_\theta$ both be linear functions of same features of s,a

- let $\Phi(s,a)$ be a vector of features describing the pair (s,a)
- let $Q_w(s,a) = \mathbf{w}^T\Phi(s,a)$ . let $\log \pi_\theta(s,a) = \boldsymbol{\theta}^T\Phi(s,a)$
- then $\nabla_w Q_w(s,a) = \phi(s,a) = \nabla_\theta \pi_\theta(s,a)$



$Q_w(s,a) = \mathbf{w}_a^T\,\Phi(s)$

$\Phi(s)$

$\log \pi_\theta(s,a) = \theta_a^T\,\Phi(s)$

# Challenges with Policy Gradient Methods

- Data Inefficiency
  - On-policy method: for each new policy, we need to generate a completely new trajectory
  - The data is thrown out after just one gradient update
  - As complex neural networks need many updates, this makes the training process very slow
- Unstable update：  step size is very important
  - If step size is too large:
    - Large step → bad policy
    - Next batch is generated from current bad policy → collect bad samples
    - Bad samples → worse policy
      (compare to supervised learning: the correct label and data in the following batches may correct it)
  - If step size is too small: the learning process is slow