Policy Gradient

Policy-Based Reinforcement Learning

- In DQN, we approximated the value or action-value function using parameters θ

$$V_\theta(s) \approx V_\pi(s), \qquad Q_\theta(s,a) \approx Q_\pi(s,a)$$

- A policy was generated directly from the value function (Value-Based RL)

- Now search directly for the optimal policy π* (Policy-Based RL)

- We use a parameterized function(neural nets) to represent the policy directly

$$\pi_\theta(s,a) = P[a|s,\theta]$$

  - Using neural nets: policy network

- Can use any parametric supervised machine learning model to learn policies $\pi_\theta(s,a)$ where θ represents the learned parameters

Policy-based Reinforcement Learning

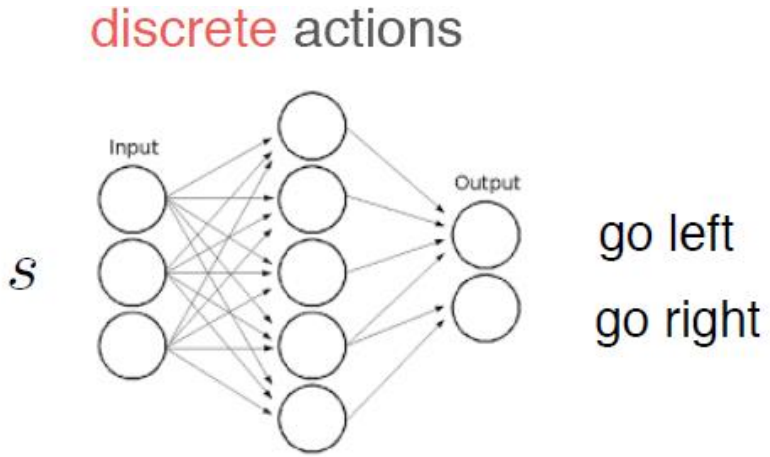
- Recall that the optimal policy is the policy that achieves maximum future return
  - Goal is to find a policy π with the highest value function V_π

- Policy gradient method is a training method in policy network/function
- Seek the policy weights which maximize performance using gradient ascent

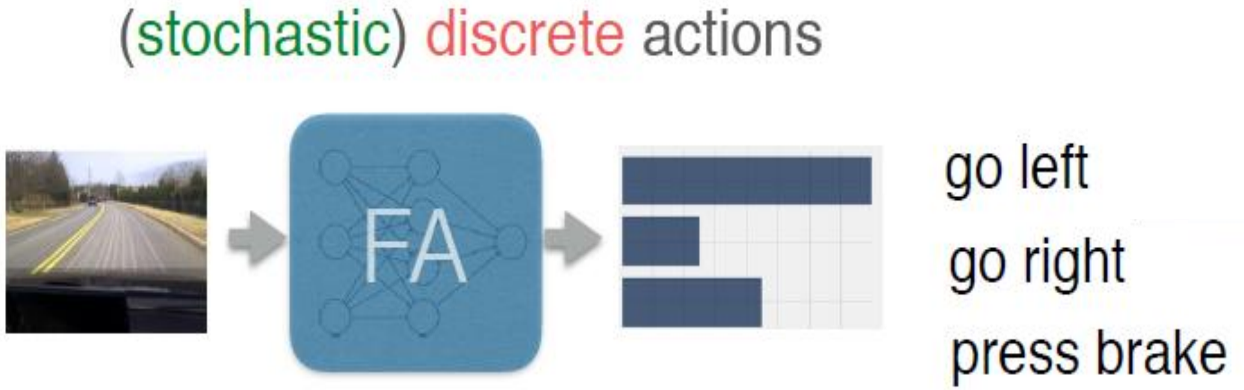
- We will focus again on model-free reinforcement learning

Policy Function Approximators

- Policy network



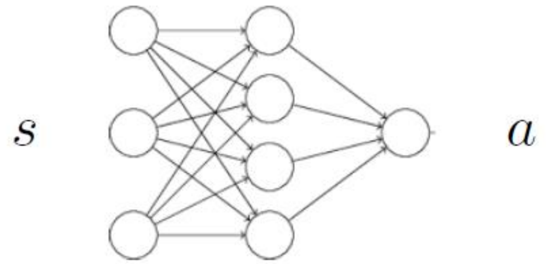
Output is a distribution over a discrete set of actions



Source: CMU 10-403, Deep Reinforcement Learning and Control, CMU

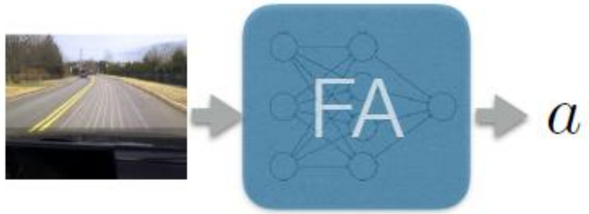
Policy Function Approximators

deterministic continuous policy



$$a = \pi_\theta(s)$$

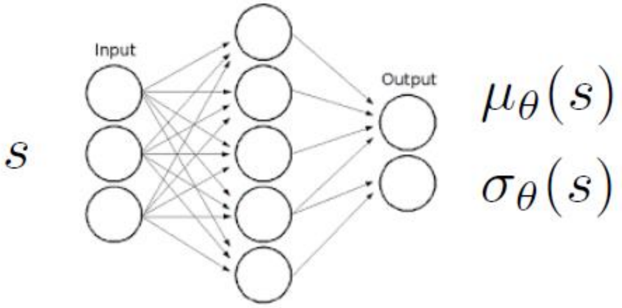
deterministic continuous policy



$$a = \pi_\theta(s)$$

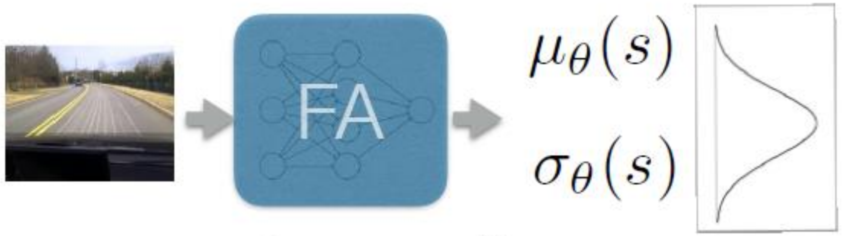
e.g. outputs a steering angle directly

stochastic continuous policy



$$a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s))$$

stochastic continuous policy

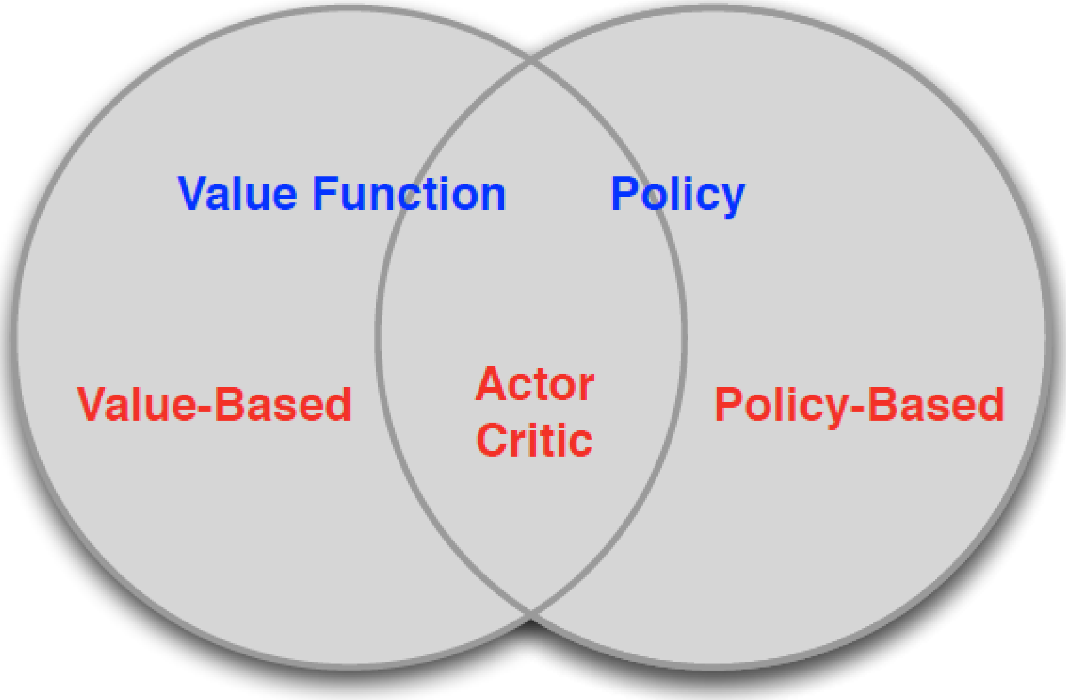


$$a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s))$$

Source: CMU 10-403, Deep Reinforcement Learning and Control, CMU

Value-Based and Policy-Based RL

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



Advantages of Policy-Based RL

- Advantages:
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Don’t need to compute all state values
    - as long as one policy is better than the other
  - Can learn stochastic policies
    - Value-based RL learns a near-deterministic policy
    - e.g. greedy or ϵ-greedy
  - Can benefit from demonstrations (imitation learning)
  - Exploration can be directly controlled
- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

Need of Stochastic Policy

- 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)

- So far have focused on deterministic policies
- Now we are thinking about direct policy search in RL, will focus heavily on stochastic policies
- Consider stochastic policy $\pi_\theta(a|s) = P(a|s;\theta)$ parameterized by θ

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?
- Objective function:
  - In episodic environments we can use the start value

$$J_0(\theta) = V_{\pi_\theta}(s_0) = E_{\pi_\theta}[v_0]$$

  - In continuing(stationary) 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

$$J_{avR}(\theta) = \sum_s d_{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R_s^a$$

where $d_{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Policy Optimization

- In policy gradient method, now policy is represented as
  1) function (policy function) or
  2) (deep) neural network (policy network)
- Policy based reinforcement learning is an optimization problem
- Find θ that maximizes $J(\theta)$

- Some approaches do not use gradient
  - Hill climbing
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient ascent/descent
  - Conjugate gradient
  - Quasi-newton
- We focus on gradient descent/ascent, many extensions possible and on methods that exploit sequential structure

Genetic Algorithm

- A **genetic algorithm** (or **GA**) is a search technique to find true or approximate solutions to optimization and search problems.
- Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).
- Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions.

GA Requirements

- A typical genetic algorithm requires two things to be defined:
  - a genetic representation of the solution domain, and
  - a fitness function to evaluate the solution domain.
- A standard representation of the solution is as an array of bits.
- The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, that facilitates simple crossover operation.
- Variable length representations may also be used, but crossover implementation is more complex in this case.

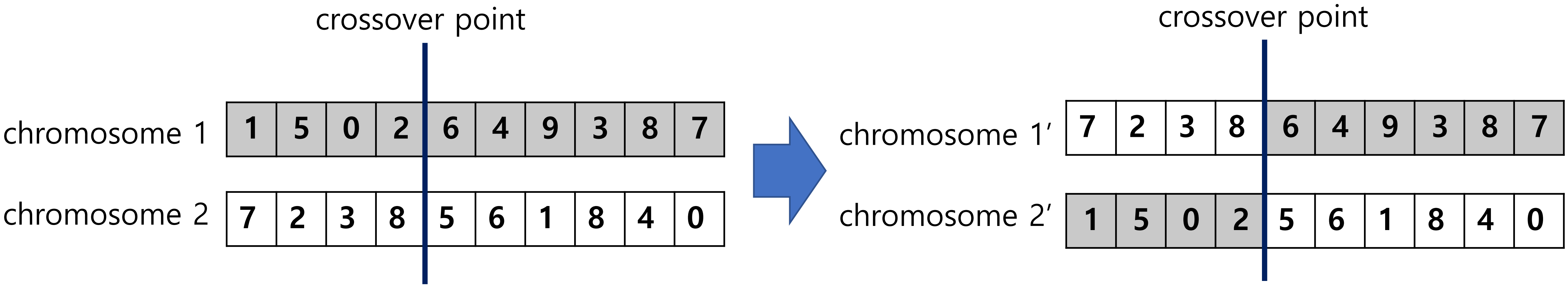
Genetic Algorithm

- The most common type of genetic algorithm works like this:

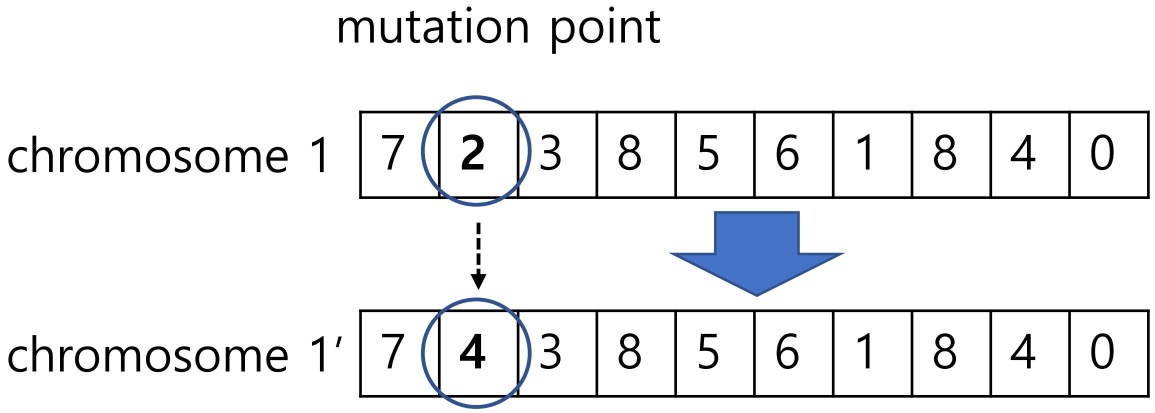
1) A population is created with a group of individuals (chromosomes) created randomly.
2) The individuals in the population are then evaluated. (fitness function)
3) The fitness function gives the individuals a score based on how well they perform at the given task.
4) Two individuals are then selected based on their fitness, the higher the fitness, the higher the chance of being selected.
   - crossover
5) These individuals then "reproduce" to create one or more offspring, after which the offspring are mutated randomly.
   - mutation
6) This continues until a suitable solution has been found or a certain number of generations have passed

Genetic Algorithm

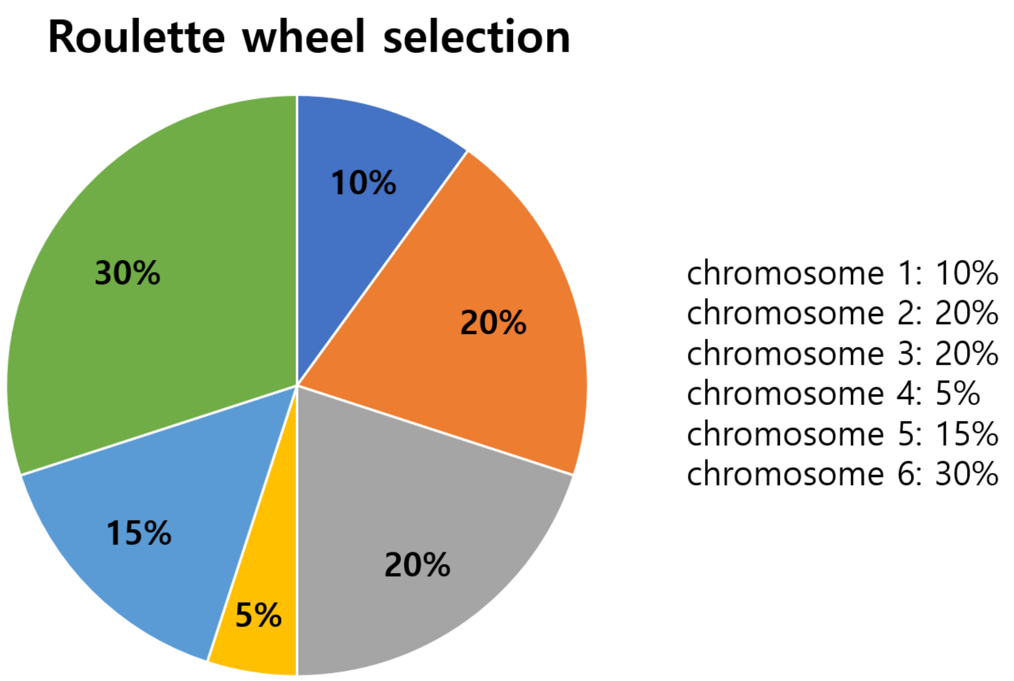
1) Crossover



2) Mutation



Genetic Algorithm



- Typical method without using gradient
- Performance of GA is known to be compatable with that of DQN and A3C [Such et al., 2018]
- Parallel processing can be done easily
- More robust in local optimal problem
- Strong alternate candidate over gradient method

Genetic Algorithm Pseudo Code

```
randomly generate N chromosome
loop (converge)
    compute fitness value of each chromosome
    for i = 1, … , p
        select two chromosomes using oulette wheel selection
        generate new chromosomes using crossover operation
    end
    for i = 1, … , k
        select one chromosome randomly
        generate new chromosome using mutation with ρ prob.
    end
if (best fitness>δ) then
    return chromosome with best fitness
end
end loop
```

Recap: Gradient Method

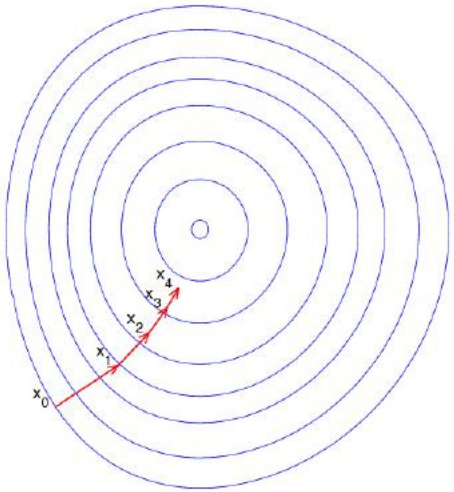
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

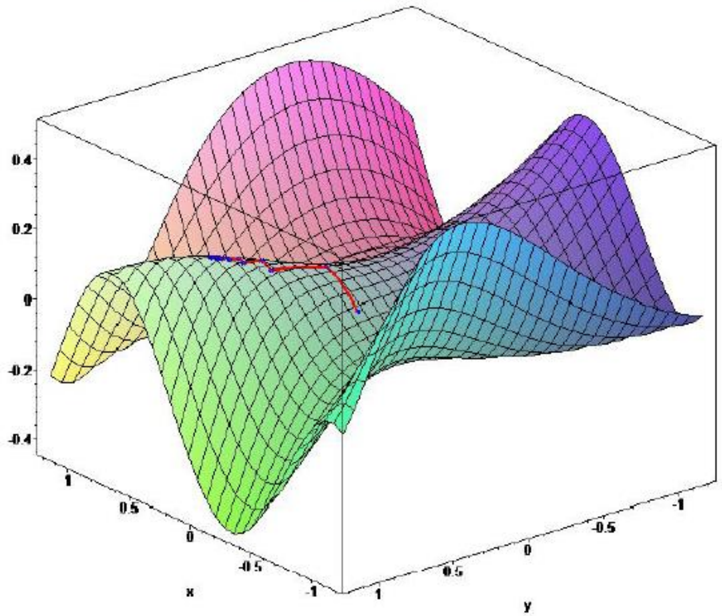
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

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 α is a step-size parameter





Types of Policy Gradient Method

- Finite Difference Policy Gradient
- Monte Carlo Policy Gradient (REINFORCE)
- Actor-Critic Policy Gradient

Computing Gradients By Finite Differences

- To evaluate policy gradient of π_θ(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

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

$$u_k = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

k-th element

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

Score Function

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

$$\begin{aligned}\nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\ &= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\end{aligned} \qquad \left(\nabla \log f(x) = \frac{\nabla f(x)}{f(x)}\right)$$

- The *score function* is $\nabla_\theta \log \pi_\theta(s,a)$
  - Score function measures how a small change of θ will affect the log-likelihood of the policy $\pi_\theta(s,a)$

Maximum Likelihood

- Consider a stochastic policy $\pi_\theta(s, a)$

- Data: state-action pairs $\{(s_1, a_1^*), (s_2, a_2^*), \dots\}$
  - Assume a_k^* are oracle values

- Maximum Likelihood learning = $\underset{\theta}{\operatorname{argmax}} \prod_i \pi_\theta\left(s_i, a_i^*\right)$
- Maximizing log likelihood of the data (use log form)

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_k \log \pi_\theta\left(s_k, a_k^*\right)$$

- Gradient update

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \log \pi_\theta\left(s_k, a_k^*\right)$$

Differentiable Policy Classes

- Many choices of differentiable policy classes including:
  - Softmax
  - Gaussian
  - Neural networks

Softmax Policy

- We will use a softmax policy as a running example

$$softmax(y_i) = e^{y_i} / \sum_j e^{y_j}$$

- Weight actions using linear combination of features $\emptyset(s,a)^T\theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) \propto e^{\emptyset(s,a)^T\theta} / \sum_{a\prime} e^{\emptyset(s,a\prime)^T\theta}$$

- Is designed as a stochastic policy but can approach deterministic behavior in the limit.

- The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \emptyset(s,a) - E_{\pi_\theta}[\emptyset(s,\cdot)]$$

Gaussian Policy

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\mu(s) = \emptyset(s)^T\theta$
- Variance may be fixed σ^2, or can also parameterized

- Policy is Gaussian, $a \sim N(\mu(s), \sigma^2)$

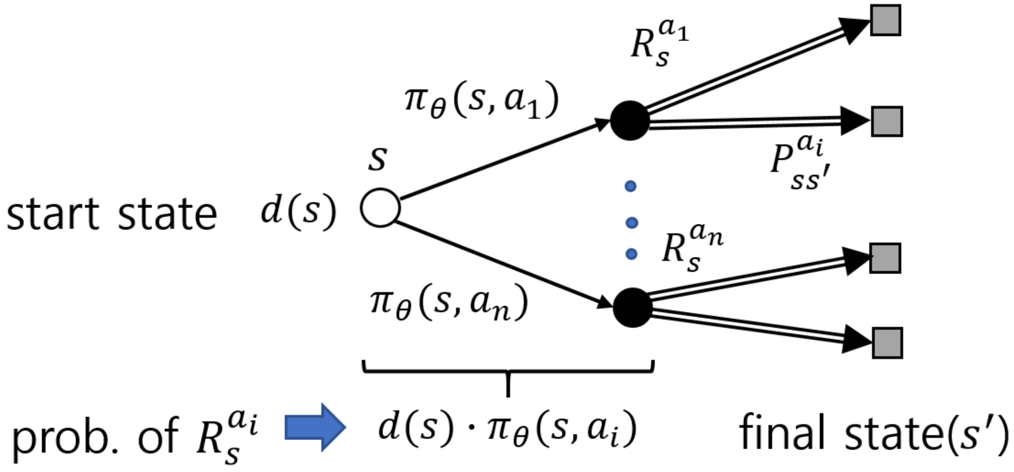
$$\pi_\theta(s,a) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(a-\mu(s))^2}{2\sigma^2}}$$

- The score function is

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

One-Step MDPs

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
    - $d(s)$: starting states of MDP
  - Terminating after one time-step with reward $r = R_s^a$



- Let's look at the objective function:
  - average reward per time-step ($J_{avR}(\theta)$)

$$J(\theta) = E_{\pi_\theta}[r] = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) R_s^a$$

$$\nabla_\theta J(\theta) = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) \, R_s^a \qquad \text{(likelihood ratios)}$$

$$= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, r]$$

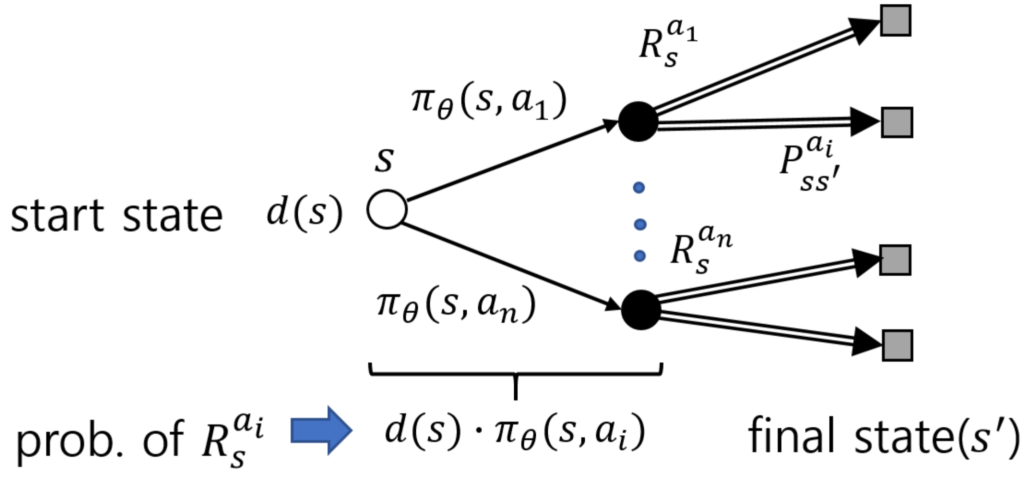
- no need to know $d(s)$

One-Step MDPs

$$\nabla_\theta J(\theta) = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)\, R_s^a$$

can be approximated by experiments:

$$\sum_{s \in S} \sum_{a \in A} d(s) \cdot \pi_\theta(s,a) \approx \frac{\text{number of } (s,a) \sim \pi_\theta}{\text{number of total actions}}$$



- Now $\nabla_\theta J(\theta)$ can be computed through experiments

$$\nabla_\theta J(\theta) = \sum_{s \in S} \sum_{a \in A} d(s) \cdot \pi_\theta(s,a) \cdot \nabla_\theta \log \pi_\theta(s,a)\, R_s^a \approx \frac{1}{N} \sum_t \nabla_\theta \log \pi_\theta(s,a)\, R_s^a$$

compute estimates through experiments

compute from network output $\pi_\theta(s,a)$

network output $\pi_\theta(s,a)$

reward from action a in state s

Policy Gradient Theorem

- How about multi-step MDP?
- 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)

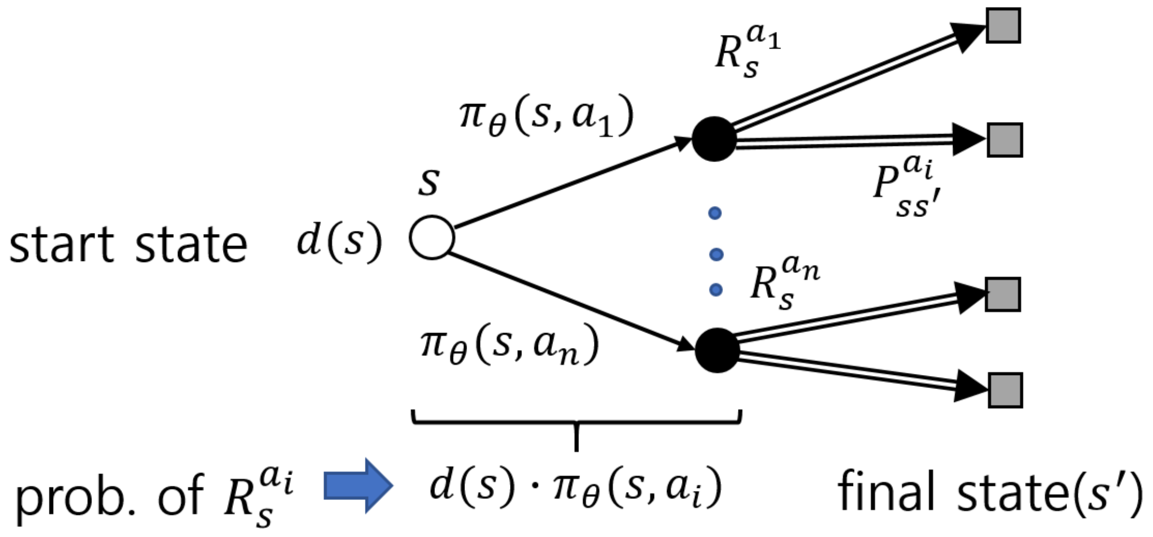
- **Policy Gradient Theorem**: For any differentiable policy $\pi_\theta(s,a)$, for any of the policy objective functions $J = J_1,\ J_{aVR},$ or $\frac{1}{1-\gamma} J_{av}V$, the policy gradient is

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q_{\pi_\theta}(s,a)]$$

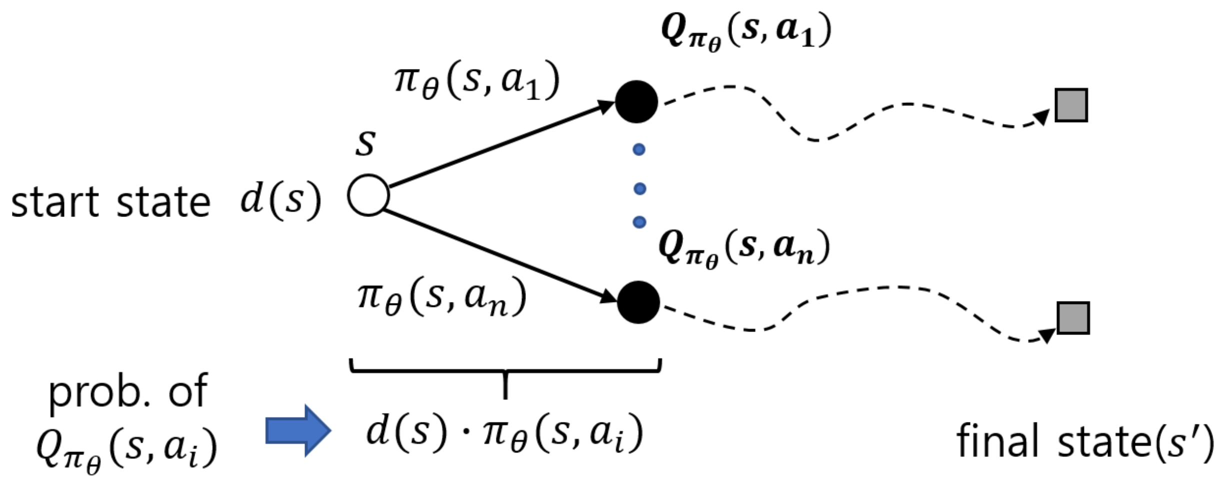
- $\nabla_\theta J(\theta)$ =(score function)*(return)
  - score function: direction of update
  - return: importance of update

One-Step MDP vs Multi-Step MDP

One-Step MDP



Multi-Step MDP



(*) Derivation of Policy Gradient Theorem (1)

- Suppose τ is generated using policy π_θ

$$\tau = (s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T)$$

- Return value of the episode is given as ($R(s_t, a_t)$ means r_{t+1}, $\gamma = 1$)

$$R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$$

- Suppose $P(\tau|\theta)$ is the prob. of episode τ

$$J(\theta) = E_{\tau\sim\pi_\theta}[R(\tau)] = \sum_{\tau} P(\tau|\theta)R(\tau)$$

- Policy objective function is defined as

$$\underset{\theta}{\operatorname{argmax}}\, J(\theta) = \underset{\theta}{\operatorname{argmax}} \sum_{\tau} P(\tau|\theta)R(\tau)$$

- Derivative of $J(\theta)$ is

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{\tau} P(\tau|\theta)R(\tau) = \sum_{\tau} P(\tau|\theta)R(\tau)\,\nabla_\theta \log P(\tau|\theta)$$
$$= E_{\tau\sim\pi_\theta}[R(\tau)\nabla_\theta \log P(\tau|\theta)]$$

(*) Derivation of Policy Gradient Theorem (2)

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau|\theta)R(\tau)\,\nabla_\theta \log P(\tau|\theta) \approx (1/m)\sum_{i=1}^{m} R\left(\tau^{[i]}\right)\nabla_\theta \log P(\tau^{[i]}|\theta) \quad \text{(A)}$$

- Because of Markov property in states

$$P\left(\tau^{[i]}\middle|\theta\right) = \prod_{t=0}^{T-1} \pi_\theta(a_t^{[i]}|s_t^{[i]})P\left(s_{t+1}^{[i]}\middle|s_t^{[i]}, a_t^{[i]}\right)$$

- Therefore,

$$\nabla_\theta \log P\left(\tau^{[i]}\middle|\theta\right) = \nabla_\theta \log\left[\prod_{t=0}^{T-1} \pi_\theta\left(a_t^{[i]}\middle|s_t^{[i]}\right) P\left(s_{t+1}^{[i]}\middle|s_t^{[i]}, a_t^{[i]}\right)\right] = \nabla_\theta\left[\sum_{t=0}^{T-1} \log \pi_\theta\left(a_t^{[i]}\middle|s_t^{[i]}\right) + \log P\left(s_{t+1}^{[i]}\middle|s_t^{[i]}, a_t^{[i]}\right)\right]$$

- Since $\nabla_\theta \sum_{t=0}^{T-1} \log P\left(s_{t+1}^{[i]}\middle|s_t^{[i]}, a_t^{[i]}\right) = 0$

$$\nabla_\theta \log P(\tau^{[i]}|\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{[i]}|s_t^{[i]}) \quad \text{(B)}$$

- This formula means we can compute $\nabla_\theta \log P\left(\tau^{[i]}\middle|\theta\right)$ without knowing $P\left(s_{t+1}^{[i]}\middle|s_t^{[i]}, a_t^{[i]}\right)$

(*) Derivation of Policy Gradient Theorem (3)

- From formula (A) and (B),

$$\nabla_\theta J(\theta) \approx \left(\frac{1}{m}\right) \sum_{i=1}^{m} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta (a_t^{[i]} | s_t^{[i]})\, R\big(\tau^{[i]}\big)$$

- Compute return value starting time-step t (instead of entire episode)

$$R\big(\tau^{[i]}\big) \leftarrow \sum_{k=t}^{H} R\left(a_k^{[i]} \middle| s_k^{[i]}\right)$$

- Therefore,

$$\nabla_\theta J(\theta) \approx \left(\frac{1}{m}\right) \sum_{i=1}^{m} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_t^{[i]} \middle| s_t^{[i]}\right) \left(\sum_{k=t}^{H} R\left(a_k^{[i]} \middle| s_k^{[i]}\right) \right)$$

$$= \left(\frac{1}{m}\right) \sum_{i=1}^{m} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_t^{[i]} \middle| s_t^{[i]}\right) G_t^{[i]}$$

- Stochastic form is

$$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta (a_t | s_t) G_t$$

Comparison of Policy Gradient and Maximum Likehood

- In Maximum Likelihood, data are given as $\{(s_0, a_0^*), (s_1, a_1^*), \ldots\}$
- Update formula of Maximum Likelihood is

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(a_t^* | s_t)$$

- Update formula of Policy Gradient is

$$\nabla_\theta \log \pi_\theta(a_t | s_t) G_t$$

Monte-Carlo Policy Gradient (REINFORCE)

- Algorithm using policy gradient theorem

- Generate an episode using a policy π_θ

$$s_0, a_0, r_1, \dots, s_{n-1}, a_{n-1}, r_n$$

- Compute return value for each step of the episode
- At time-step t, return G_t is defined as

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots + \gamma^T r_{t+T+1} = \sum_{i=0}^{T} \gamma^i r_{t+i+1}$$

- Using return G_t as an unbiased sample of $Q_{\pi_\theta}(s_t, a_t)$, policy gradient becomes

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, G_t]$$

Monte-Carlo Policy Gradient (REINFORCE)

Initialize policy network π_θ
loop
    Generate episodes $\tau \sim s_0, a_0, r_1, \dots, s_{n-1}, a_{n-1}, r_n$ with π_θ
    $\Delta\theta$=0
    for every step $t = 0,1,2,\dots,n-1$ in episode **do**

$$G_t = \sum_{i=0}^{n-t} \gamma^i r_{t+i}$$

$$\Delta\theta \leftarrow \Delta\theta + \underline{\gamma^t \nabla_\theta \log \pi_\theta(s_t, a_t)}\, G_t$$

    end for
$\theta \leftarrow \theta + \alpha\Delta\theta$
end loop
return π_θ

- On-policy method
- Monte Carlo algorithm and is only well-defined for the episodic case

Reducing Variance Using a Critic

- Monte-Carlo policy gradient has high variance and many of return v_t are zero
- Instead of using real value $Q_{\pi_\theta}(s_t, a_t)$, we use a critic to estimate the action-value function,

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

  - critic can be a value function approximation (implemented as DQN)

- Actor-critic algorithms maintain two sets of parameters
  - Actor Updates policy parameters θ, in direction suggested by critic
  - Critic Updates action-value function parameters w

Estimating the Action-Value Function

instead of $Q_{\pi_\theta}(s,a)$

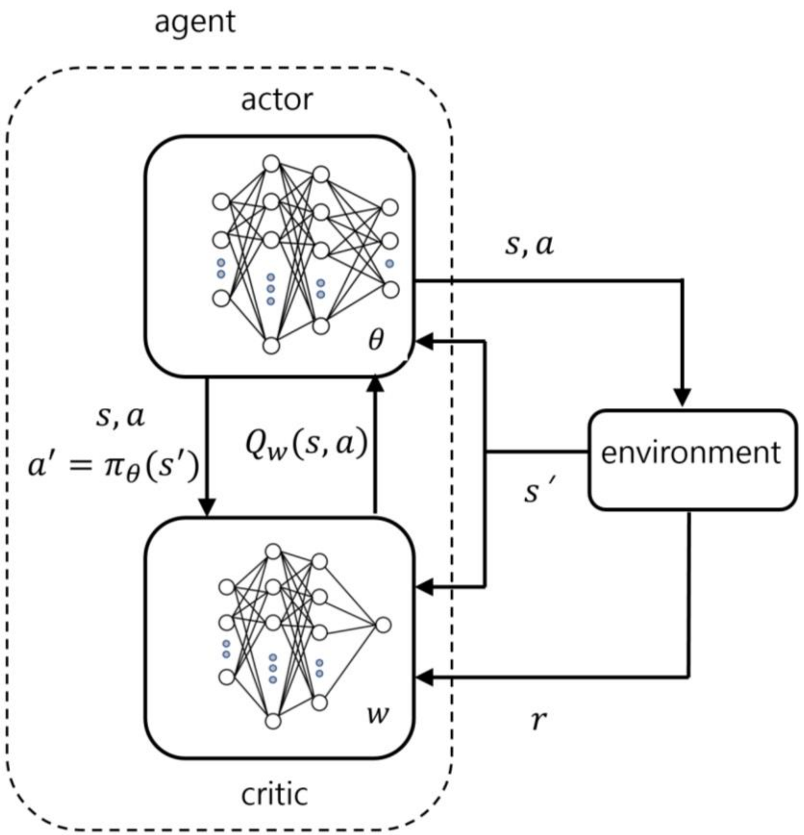
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q_w(s,a)]$$
$$\Delta\theta = \alpha\, \nabla_\theta \log \pi_\theta(s,a)\, Q_w(s,a)]$$

- The critic is solving a familiar problem: policy evaluation

- Use Temporal-Difference learning for critic network
- Critic update:

$$\Delta w = \alpha\big(r + \gamma Q_w(s',a') - Q_w(s,a)\big)\nabla_w Q_w(s,a)$$



Action-Value Actor-Critic

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

function QAC
    Initialise s, θ
    Sample $a \sim \pi_\theta$
    for each step **do**
        Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
        Sample action $a' \sim \pi_\theta(s', a')$
        $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$ → TD(0) learning
        $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$ → Policy gradient
        $w \leftarrow w + \beta \delta \boxed{\phi(s, a)}$
        $a \leftarrow a', s \leftarrow s'$
    end for
end function

$\nabla_w Q_w(s,a)$ if $Q_w(s,a)$ is neural net(non-linear)

Reducing Variance Using a Baseline

- We want to reduce the variance of Q value further
- We subtract a baseline function $B(s)$ from the policy gradient
  - $B(s)$: an arbitrary base function for a state s
  - Use $Q_w(s,a) - B(s)$ instead of $Q_w(s,a)$
  - $E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q_w(s,a)] - E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, B(s)]$

- This can reduce variance, without changing expectation

$$E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, B(s)] = \sum_{s\in S} d_{\pi_\theta}(s) \sum_{a\in A} \nabla_\theta \pi_\theta(s,a) B(s)$$

$$= \sum_{s\in S} d_{\pi_\theta}(s) B(s) \nabla_\theta \sum_{a\in A} \pi_\theta(s,a) = 0$$

  - $\nabla_\theta \sum_{a\in A} \pi_\theta(s,a)$ =0 since $\sum_{a\in A} \pi_\theta(s,a)$=1

How to Choose the Baseline?

- A good baseline is the state value function $B(s) = V_{\pi_\theta}(s)$
- Intuitively, we are happy with an action a_t in a state s if $Q_{\pi_\theta}(s, a) - V_{\pi_\theta}(s)$ is large. On the contrary, we are unhappy with an action if it's small.
- Want to push up the probability of an action from a state, if this action was better than the expected value of what we should get from that state
- So we can rewrite the policy gradient using the advantage function $A_{\pi_\theta}(s, a)$

$$A_{\pi_\theta}(s, a) = Q_{\pi_\theta}(s, a) - V_{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a)\, A_{\pi_\theta}(s, a)]$$

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,

$$V_V(s) \approx V_\pi(s)$$
$$Q_w(s, a) \approx Q_\pi(s, a)$$
$$A(s, a) = Q_w(s, a) - V_V(s)$$

- 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 δ^{π_θ}

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

  is an unbiased estimate of the advantage function

$$E_{\pi_\theta}[\delta^{\pi_\theta}|s,a] = E_{\pi_\theta}\left[r + \gamma V_{\pi_\theta}(s')\middle|s,a\right] - V_{\pi_\theta}(s)$$
$$= Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s) = A_{\pi_\theta}(s,a)$$

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

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta^{\pi_\theta}] \longrightarrow \text{Instead of } A_{\pi_\theta}(s,a)$$

- In practice we can use an approximate TD error (since we don't know the true value function $V_{\pi_\theta}(s)$)

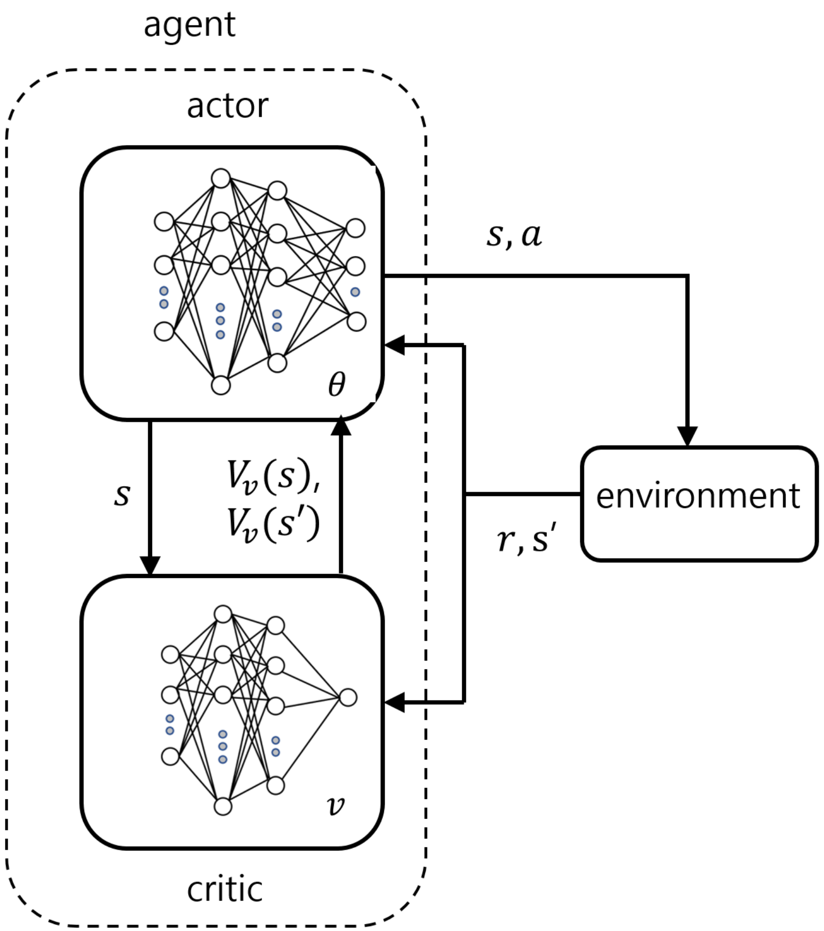
$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- This approach only requires one set of critic parameters v

Actor-Critic Algorithm

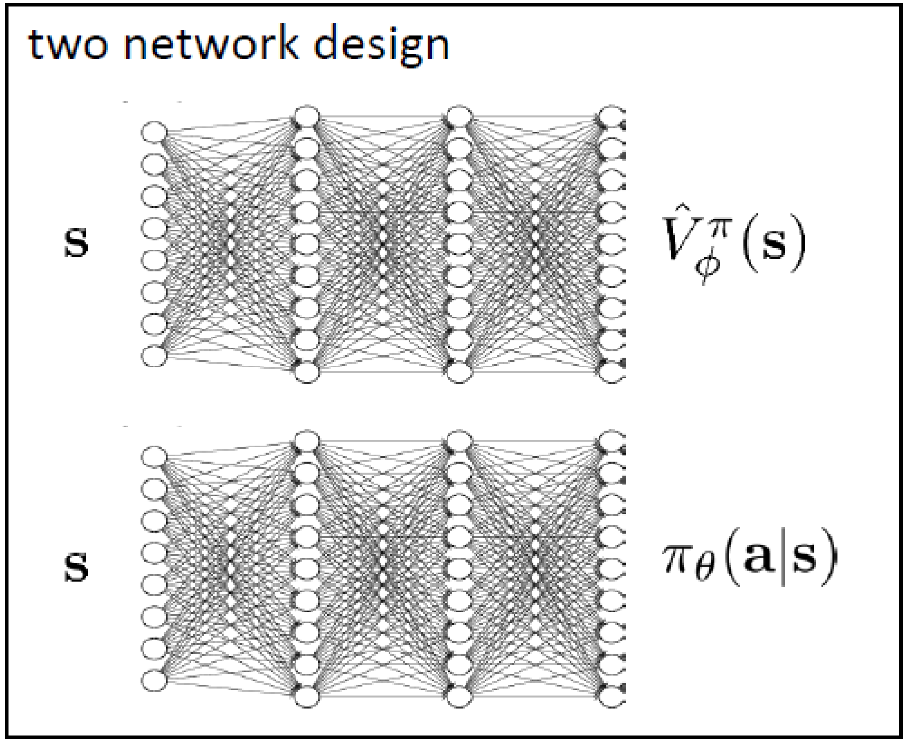
TD Actor-Critic

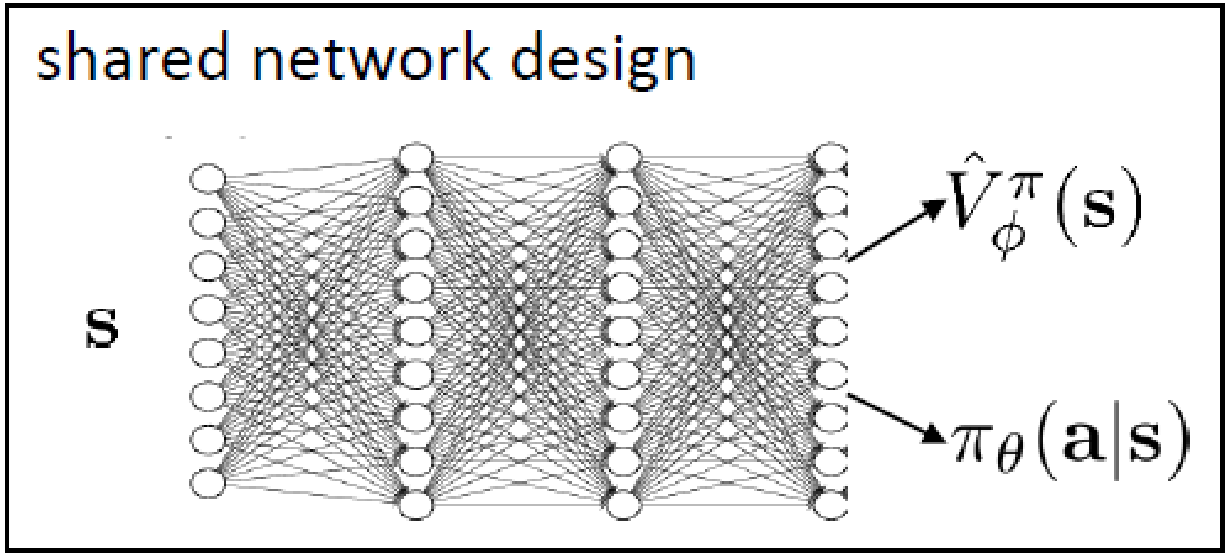
Initialize actor parameter θ critic parameter v
repeat the following with different start state s
for $i = 0, \ldots, T$ **do**
    perform action a in state s using policy π_θ // $a \sim \pi_\theta(a|s)$
    generate reward r and next state s'
    $\delta_\upsilon = r + \gamma V_v(s') - V_v(s)$ // compute TD error
        (**if** s is a final state $\delta_\upsilon = r - V_v(s)$)
    $\theta \leftarrow \theta + \alpha \delta_\upsilon \nabla \log \pi_\theta(a|s)$ // update actor
    $v \leftarrow v + \beta \delta_\upsilon \nabla V_v(s)$ // update critic
    $s \leftarrow s'$
end for



Actor-Critic Networks

Value network





Policy network

Generalized Advantage Estimation(GAE)

- Recap: n step return:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v(s_{t+n})$$

λ-return:

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- Advantage Actor-Critic

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)\, \delta_v]$$

where

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

Generalized Advantage Estimation(GAE)

- Instead of one-step, use multi-step data (similar to TD(λ))
- Define

$$A_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^n v(s_{t+n}) - v(s_t)$$

- GAE advantage estimate is

$$A_t^{GAE(\gamma\lambda)} = (1-\lambda)\left(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \cdots\right) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}^{v}$$

where

$$\delta_t^v = A_t^{(1)} = r_{t+1} + \gamma v(s_{t+1}) - v(s_t)$$

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

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

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