Week 5: Policy Optimization I: Policy Gradient and Actor-Critic

Bolei Zhou

UCLA

February 3, 2026

Announcement

1. Assignment 2 due by the end of this week

This Week's Plan

1. Introduction on policy function versus value function
2. Policy-based reinforcement learning
3. Monte-Carlo policy gradient
4. Actor-Critic methods

Value-based RL vs Policy-based RL

1. In previous lectures, we focused on value-based RL and had value function approximation with parameter θ

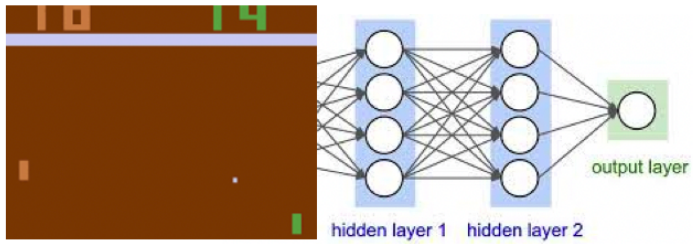
$$V_\theta(s) \approx v^\pi(s)$$
$$Q_\theta(s,a) \approx q^\pi(s,a)$$

2. A policy is generated directly from the value function using ϵ-greedy or greedy $a_t = \arg\max_a Q(a, s_t)$
3. Instead, we can also parameterize the policy function as $\pi_\theta(a|s)$ where θ is the learnable policy parameter

RL Diagram in the View of Policy Optimization

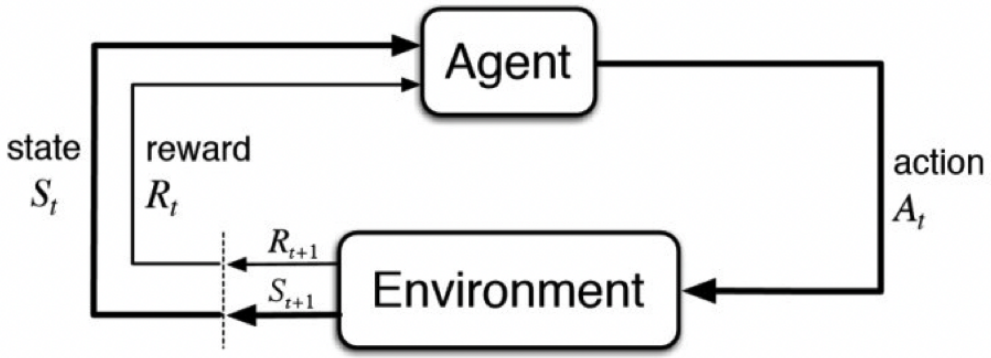
no need for a state or q value function

1. Action from the policy is all we need, then let's optimize the policy directly!



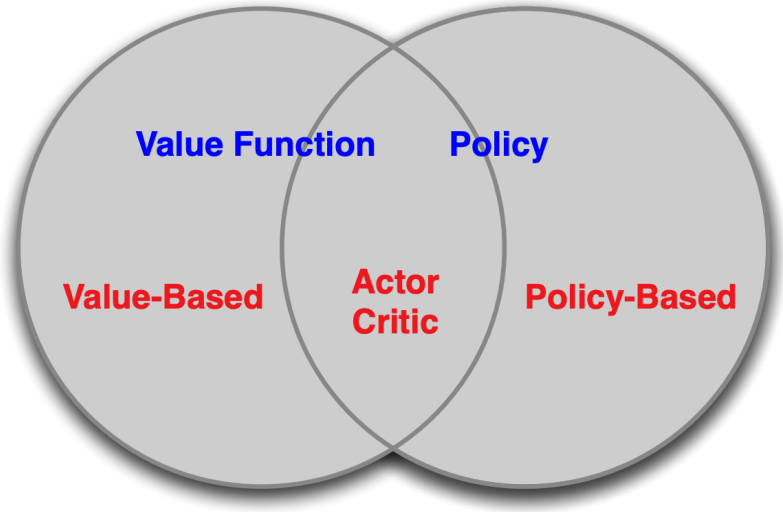
Probability of moving up

$$\pi_\theta(A_t|S_t)$$



Value-based RL versus Policy-based RL

1. Value-based RL
   1. learn value function
   2. implicit policy based on the value function
2. Policy-based RL
   1. no value function
   2. learn policy directly
3. Actor-critic
   1. learn both policy and value function



Strengths of Policy-based RL

1. Strengths:
   1. Better convergence properties: we are guaranteed to converge on a local optimum (worst case) or global optimum (best case)
   2. Policy gradient is more effective in a high-dimensional action space
   3. Policy gradient can learn stochastic policies, while value function can't
      probability over actions
2. Weaknesses:
   1. typically converges to a local optimum
   2. Evaluating a policy is inefficient and has high variance

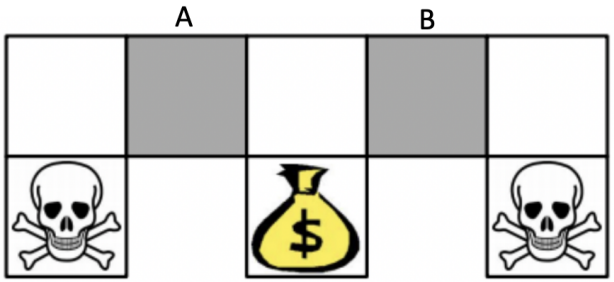
Two Types of Policies

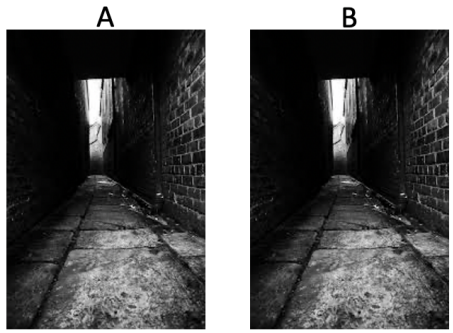
1. Deterministic: given a state, the policy returns the action to take (discrete action space) or returns a continuous value (continuous action space)
2. Stochastic: given a state, the policy returns a probability distribution of the actions (e.g., 40% chance to turn left, 60% chance to turn right), or a probability density function for continuous action

Example: Rock-Paper-Scissor

1. Two-player game
2. What is the best policy?
   1. It is easy to beat a deterministic policy
   2. A uniform random policy is optimal (Nash equilibrium)

Example: Alleyways in the Gridworld





state a and b look the same

1. The agent cannot differentiate the two grey states (the two alleyways look the same to the agent)
2. Considers the following features (for all N, E, S, W)

$$\psi(s,a) = [\mathbf{1}(\text{wall to N, a = move E}), \mathbf{1}(\text{wall to S, a = move W}), ...]$$

3. If it is value-based RL, value function approximation as:

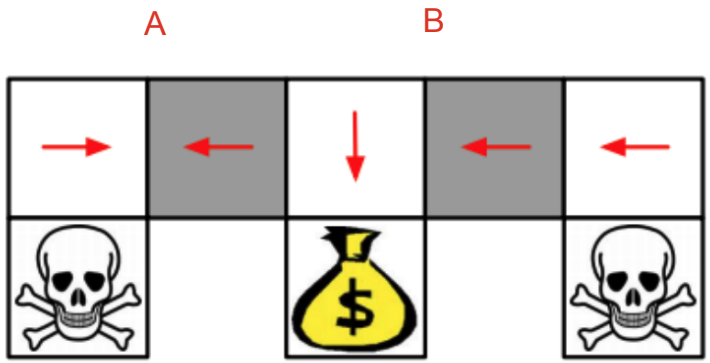
same features will represent the two states

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

4. If it is policy-based RL, policy function approximation as:

$$\pi_\theta(a|s) = g(\psi(s,a),\theta)$$

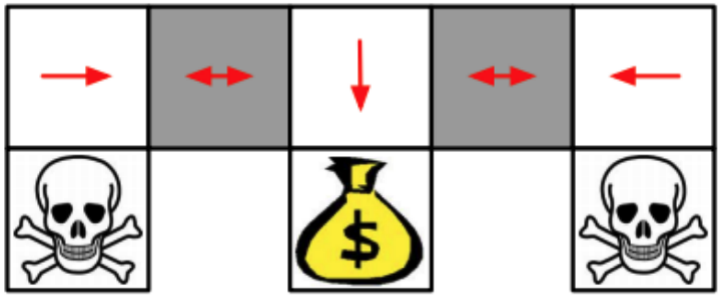
Example: Alleyways in the Gridworld



in value based, A and B must be the same policy (based on the value function?)

1. Value-based RL learns a near-deterministic policy, e.g., greedy or ϵ-greedy
2. Because of the aliasing (the agent cannot differentiate two states), an optimal deterministic policy from value-based RL will either
   1. move W in both grey states (shown by red arrows)
   2. move E in both grey states
3. Either way, 50% chance will get stuck

Example: Alleyways in the Gridworld



1. Policy-based RL can learn the optimal stochastic policy
2. An optimal stochastic policy will randomly move E or W in two grey states

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

3. For any starting point, it will reach the goal state in a few steps with high probability

Objective of Optimizing Policy

1. Objective: Given a policy approximator $\pi_\theta(a|s)$ with parameter θ, find the best θ

want to optimize cumulative reward

2. How do we measure the quality of a policy π_θ?

plug in policy in the environment and do the roll out

3. In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1)$$

s1 = initial state

4. In continuing environments
   1. we can use the average value

probability of choosing the state given the policy

value of starting at s?

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

get expected state value over all states

   2. or the average reward per time-step

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

where d^{π_θ} is stationary distribution of Markov chain for π_θ

Objective of Optimizing Policy

1. The value of the policy is defined as

use per step rewards to evaluate quality of a policy

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_t r(s_t, a_t)]$$

sampling actions from our policy

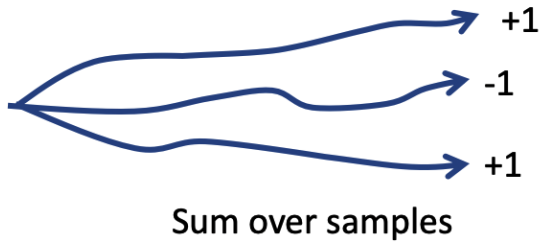
objective function measures the performance of whole trajectories sampled from our policy

run m times

$$\approx \frac{1}{m} \sum_m \sum_t r(s_{t,m}, a_{t,m})$$

m indexes each trajectory

It is the same as the value function we defined in the value-based RL



   1. τ is a trajectory sampled from the policy function π_θ
2. The goal of policy-based RL

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim \pi_\theta}[\sum_t r(s_t, a_t)]$$

tau = [a1, a2, a3, ...] (whole trajectories)

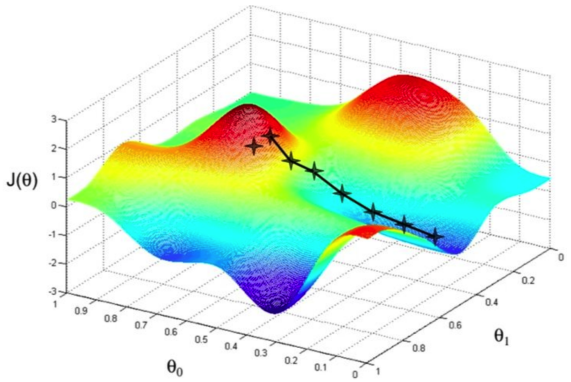
tau are the actions sampled from our policy

Objective of Optimizing Policy

1. Policy-based RL is an optimization problem that finds θ that maximizes $J(\theta)$
2. If $J(\theta)$ is differentiable, we can use gradient-based methods:
   1. gradient ascend
   2. conjugate gradient
   3. quasi-newton
3. If $J(\theta)$ is non-differentiable or hard to compute the derivative, some derivative-free black-box optimization methods:
   1. Cross-entropy method (CEM)
   2. Evolution strategy
   3. Finite difference

can't calculate the derivative df(x)/dx

Policy Optimization using Gradient Ascend



1. Consider a function $J(\theta)$ to be any policy objective function
2. Goal is to find parameter θ^* that maximizes $J(\theta)$ by ascending the gradient of the policy,w.r.t parameter θ

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

3. Adjust θ in the direction of the gradient, where α is step-size
4. Define the gradient of $J(\theta)$ to be

$$\nabla_\theta J(\theta) = (\frac{\partial J(\theta)}{\partial\theta_1}, \frac{\partial J(\theta)}{\partial\theta_2}, ..., \frac{\partial J(\theta)}{\partial\theta_n})^T$$

Policy Optimization using Derivative-free Methods

1. Sometimes we cannot compute the derivative, i.e., $\nabla_\theta J(\theta)$
2. Derivative free methods:
    1. Evolution Strategy (ES)
    2. Cross Entropy Method (CEM)
    3. Finite Difference

Derivative-free Method: Evolution Strategy (ES)

1. $\theta^* = \arg\max J(\theta)$ \pi_\theta is differentiable?
2. Treat $J(\theta)$ as a black box score function (not differentiable)

this helps maximize J(\theta) based on changing \theta

Algorithm 1 Evolution Strategy

1: **for** iter $t = 0, 1, 2, \ldots$ **do** generate pertubation (noise), add to policy parameters
2: sample $\epsilon_1, \ldots, \epsilon_n \sim \mathcal{N}(0, I)$ to generate perturbed $\theta_i = \theta_t + \delta\epsilon_i$
3: execute roll-out under θ_i for $i = 1, \ldots, n$ to compute $J(\theta_i)$ play out
4: set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\delta} \sum_{i=1}^{n} J(\theta_i)\epsilon_i$ and get the sample return
5: **end for**

use a weighted suim with noise to update parameters

3. ES is easy to implement and scale. Running on a computing cluster of 80 machines and 1,440 CPU cores, the implementation is able to train a 3D MuJoCo humanoid walker in only 10 minutes
   1. `https://openai.com/blog/evolution-strategies/`
4. Example code: `https://github.com/ucla-rlcourse/RLexample/blob/master/derivativefree/es_cem_agent.py`

Derivative-free Method: Cross-Entropy Method

1. $\theta^* = \arg\max J(\theta)$
2. Treat $J(\theta)$ as a black box score function (not differentiable)

Algorithm 2 CEM

for iter $t = 1, 2, 3, \ldots$ **do**
  $\mathcal{C} = \{\}$
  for parameter set $e = 1$ to n **do**
    sample $\theta^{(e)} \sim P_{\mu^{(t)}}(\theta)$
    execute roll-outs under $\theta^{(e)}$ to evaluate $J(\theta^{(e)})$
    store $(\theta^e, J(\theta^{(e)}))$ in $\mathcal{C}$
  end for
  $\mu^{(i+1)} = \arg\max_\mu \sum_{k \in \hat{\mathcal{C}}} \log P_\mu(\theta^{(k)})$
  where $\hat{\mathcal{C}}$ are the top 10% of $\mathcal{C}$ ranked by $J(\theta^{(e)})$
end for

think N(\mu, \sigma^2)

probability distribution over parameters

cimplete the roll outs

update mu to increase log likelihood of generating a good set

based on choosing only the \theta params with the best performance

3. Example code: `https://github.com/ucla-rlcourse/RLexample/blob/master/derivativefree/es_cem_agent.py`

Approximate Gradients by Finite Difference

1. To evaluate policy gradient of $\pi_\theta(a|s)$
2. For each dimension $k \in [1, n]$
   1. estimate *kth* partial derivative of objective function by perturbing θ by a small amount δ in *kth* dimension

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

   where u_k is unit vector with 1 in *kth* component, 0 elsewhere
3. use n evaluations to compute policy gradient in total n dimensions
4. Though noisy and inefficient, it works for arbitrary policies, even if policy is not differentiable.

Compute the Policy Gradient Analytically

REVIEW THIS SLIDE GANG

1. π_θ is a probability function or density function with parameter θ
2. Assume policy π_θ is differentiable whenever it is no-zero
3. We can compute the gradient $\nabla_\theta \pi_\theta(a|s)$
4. Likelihood ratio exploits the following trick

gradient over policy function

why is this equivalent ot the log?, just think of the chain rule

$$\nabla_\theta \pi_\theta(a|s) = \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$$

d[log (f(x))]/dx = df(x) / f(x)

$$= \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)$$

this is like maximum likelihood estimation

5. The score function is $\nabla_\theta \log \pi_\theta(a|s)$ this is the gradient of our log likelihood function, where we use samples from MC
   1. Score indicates the steepness of the log-likelihood function
   2. `https://en.wikipedia.org/wiki/Score_(statistics)`

Policy Gradient for One-Step MDPs

one step to finish the game

1. Consider a simple class of one-step MDPs
   1. Start in state $s \sim d(s)$ generate a starting s, based on the distribution of starting states
   2. Terminate after one time-step with reward $r = R(s, a)$ terminate immediately
2. The objective is as follows \theta are the parameters of the policy network

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r]$$

this equation: we sum up the reward for every possible action in every possible state, weighted by how likely that action is going ot happen d(s) and how likely the agent is to pick the action \pi_\theta(a|s)

$$= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) r$$

REVIEW HOW THIS IS RELATED TO MLE

3. Use likelihood ratios to compute the policy gradient, then the gradient is as

taking the \grad, since its with respect to the policy, we can take it inside

$$\nabla_\theta J(\theta) = \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) r$$

then we use the MLE to get the log

this can be estimated through sampling

$$= \mathbb{E}_{\pi_\theta}[r \nabla_\theta \log \pi_\theta(a|s)]$$

expectation over cum. rewards

observations are state s

Discrete Policy Function: Softmax Policy

this slide is about how we use the features and weights to calculate a score we can softmax to get action probabilities

1. Simple policy model: weight actions using a linear combination of features $\phi(s,a)^T\theta$
2. Probability of action is proportional to the exponentiated weight

$$\pi_\theta(a|s) = \frac{\exp^{\phi(s,a)^T\theta}}{\sum_{a'} \exp^{\phi(s,a')^T\theta}}$$

how do we derive this?
its just basic calc and log rules

3. The score function is

something is a neural network?

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

features of action we actually took

average features of all actions, weighted by how likely the policy thinks they are

Continuous Policy Function: Gaussian Policy

1. In continuous action spaces, a Gaussian policy can be naturally defined
2. Mean is a linear combination of state features $\mu(s) = \phi(s)^T \theta$
3. Variance may be fixed σ^2 or can also be parameterized
4. Policy is Gaussian value $a \sim \mathcal{N}(\mu(s), \sigma^2)$
5. The score function is

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

Policy Gradient for Multi-step MDPs

1. Denote a sampled trajectory from π_θ from the environment as $\tau = (s_0, a_0, r_1, ...s_{T-1}, a_{T-1}, r_T, s_T) \sim (\pi_\theta, P(s_{t+1}|s_t, a_t))$
   1. Sometimes we just simplify the notation as $\tau \sim \pi_\theta$
2. Use $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ to be the sum of rewards for a trajectory τ
3. The policy objective is

these are rollouts from the environment

$$J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T} R(s_t, a_t)\Big] = \sum_{\tau} P(\tau;\theta)R(\tau)$$

   where $P(\tau;\theta)$ denotes the probability over trajectories when executing the policy π_θ
4. Then our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_{\tau} P(\tau;\theta)R(\tau)$$

Policy Gradient for Multi-step MDPs

1. Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

2. Take the gradient with respect to θ:

$$\begin{aligned}
\nabla_\theta J(\theta) =& \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\
=& \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) \\
=& \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
=& \sum_\tau P(\tau;\theta)R(\tau)\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} \\
=& \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)
\end{aligned}$$

Policy Gradient for Multi-step MDPs

1. Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

2. Take the gradient with respect to θ:

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)$$

3. Approximate the gradient of objective with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau_i)\nabla_\theta \log P(\tau_i;\theta)$$

\theta_{t+1} = \theta_t + \nabla_\theta J(\theta)

Decomposing a Trajectory

1. Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i; \theta)$$

\tau = {s_0, a_0, r_1, s_1, ...}

2. Decompose $\nabla_\theta \log P(\tau; \theta)$

\mu(s_0) is the initial distribution

this is decided by the environment

$$\nabla_\theta \log P(\tau; \theta) = \nabla_\theta \log \left[\mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right]$$

might not be available

im guessing that log \mu(s_0) doesn't depend on \theta so the gradient just takes it out

$$= \nabla_\theta \left[\log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \right]$$

this doesn't depend on the policy so we can just take it out

$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Likelihood Ratio Policy Gradient

1. Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta) R(\tau)$$

2. Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i;\theta)$$

3. Thus we have $\nabla_\theta \log P(\tau_i;\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

    1. It does not need to know the transition dynamics!

Understand Score Function Gradient Estimator

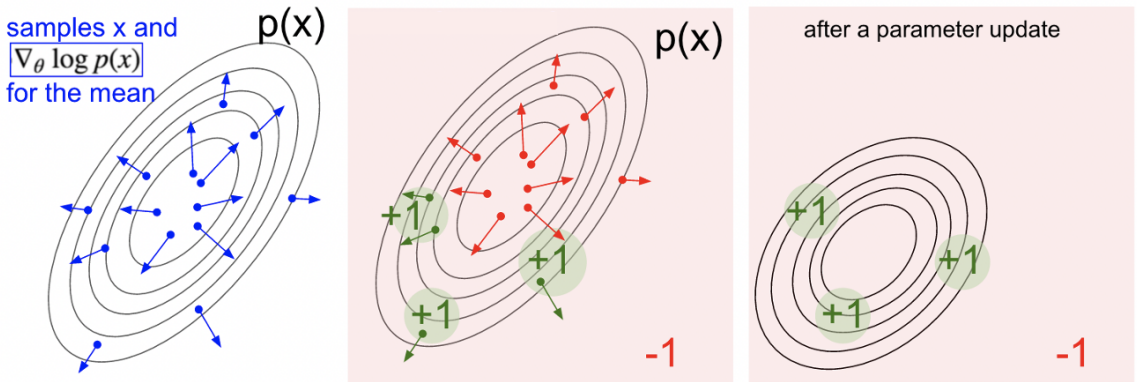
1. Consider the generic form of $E_{\tau\sim\pi_\theta}[R(\tau)]$ as

$$\nabla_\theta \mathbb{E}_{x\sim p(x;\theta)}[f(x)] = \mathbb{E}_x[f(x)\nabla_\theta \log p(x;\theta)]$$

x = trajectory

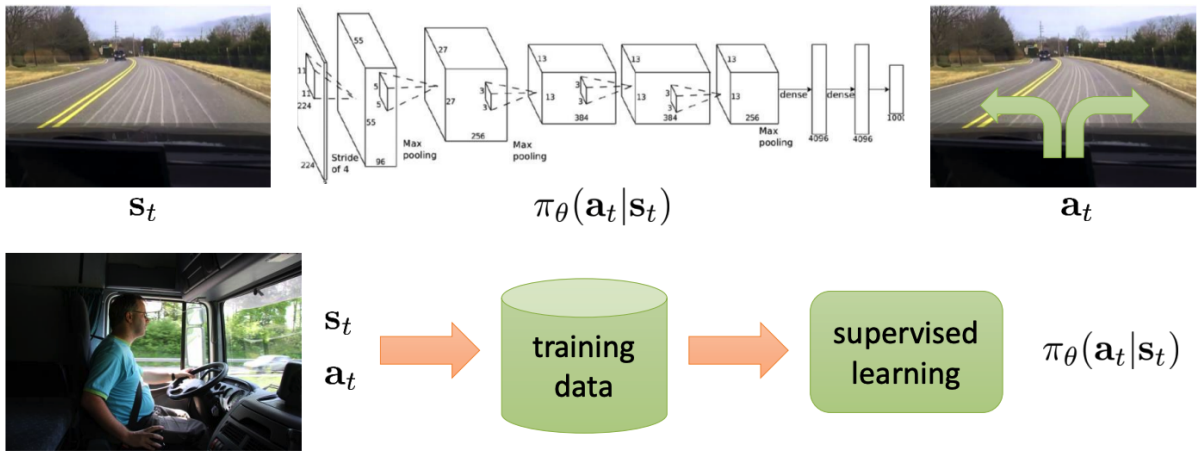
p(x; \theta) = probability distribution of a trajectory happening

2. The above gradient can be understood as:
   1. Shift the distribution p through its parameter θ to let its future samples x achieve higher scores as judged by $f(x)$
   2. The direction of $f(x)\nabla_\theta \log p(x;\theta)$ pushes up the log probability of the sample, in proportion to how good it is



Comparison to Maximum Likelihood

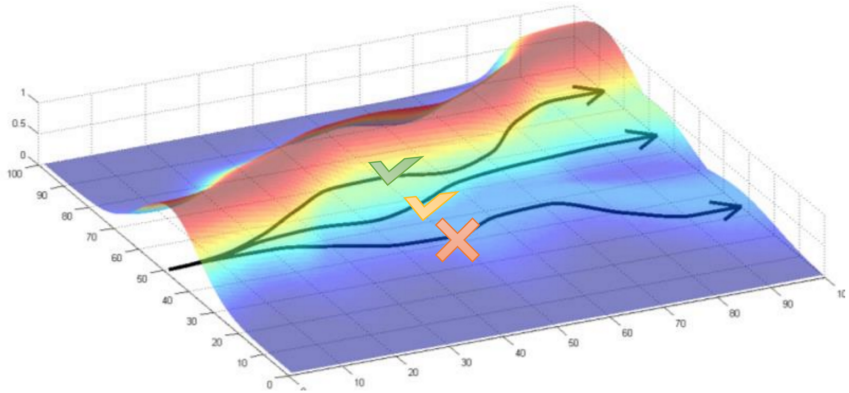
1. Policy gradient estimator: $\nabla_\theta J(\theta) \approx$ total reward for a trajectory
   $\frac{1}{M}\sum_{m=1}^{M}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{t,m}, s_{t,m})\right)\left(\sum_{t=1}^{T} r(s_{t,m}, a_{t,m})\right)$
   use a weighted sum of the gradient based on the reward
2. Maximum likelihood estimator:
   $\nabla_\theta J_{ML}(\theta) \approx \frac{1}{M}\sum_{m=1}^{M}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{t,m}, s_{t,m})\right)$
3. good action is made more likely, bad action is made less likely



Comparison to Maximum Likelihood Estimation

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i; \theta)$$

1. If going up the hill leads to higher reward, change the policy parameters to increase the likelihood of trajectories that move higher



Large Variance of Policy Gradient

1. We have the following approximate update

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)$$

2. Unbiased but having high variance as we use the MC method
3. Two fixes:
   1. Use temporal causality
   2. Include a baseline

Reduce Variance of Policy Gradient using Causality

in this, we go through an entire trajectory, then we use the score to reinforce every single action taken

this gives credit to actions that didn't actually cause the reward

1. Previously $\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\left(\sum_{t=0}^{T-1} r_t \right) \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \right]$
2. Causality: the policy cannot affect rewards in the past. This yields the following modified objective, where the sum of rewards here does not include the rewards achieved before the time step at which the policy is being queried: policy at time t' cannot affect reward at time t when $t < t'$

r1 is irrelevant to the actions and states s_{1+}, a_{1+}

summation now starts at t and not 0

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} r_{t'} \right]$$

$$= \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

ignore past rewrads

we switch to Future rewards

Reduce Variance of Policy Gradient using Causality

1. Therefore we have

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

2. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is reward-to-go of a trajectory at step t, or a sample estimate of the Q function,
3. Then we can have the following estimated update

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T-1} G_t^{(i)} \cdot \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

REINFORCE: a Monte-Carlo policy gradient algorithm

1. The algorithm simply samples multiple trajectories following the policy π_θ while updating θ using the estimated gradient

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$ define how we parameterize our policy
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t go backwards because we are using the future rewards
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

2. Classic paper: Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm

Reduce Variance Using a Baseline

1. The original update

this is high variance because randomness in enviornment and policy

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

   1. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory which might have high variance

2. We subtract a baseline b (that is a constant with respect to τ) from the policy gradient to reduce variance

mean of the gradient stays the same, but the variance is reduced
this is because we take the square distance from the mean and if the values are large, square distance would be larger

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} \big(G_t - b \big) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

3. A good baseline can be the expected return

can approx this with monte carlo

$$b = \mathbb{E}[r_t + r_{t+1} + \ldots + r_{T-1}]$$

Reduce Variance Using a Baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} (G_t - b) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

1. Interpretation: increase the logprob of action a_t proportional to how much returns G_t are better than the expected return
2. We can **prove** that including baseline $b(s)$ can reduce variance, without changing the expectation:

$$\mathbb{E}_\tau \Big[b \nabla_\theta \log \pi_\theta(a_t|s_t) \Big] = 0, \tag{1}$$

$$E_\tau \Big[(G_t - b) \nabla_\theta \log \pi_\theta(a_t|s_t) \Big] = E_\tau \Big[G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \Big] \tag{2}$$

$$Var_\tau \Big[(G_t - b) \nabla_\theta \log \pi_\theta(a_t|s_t) \Big] < Var_\tau \Big[G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \Big] \tag{3}$$

3. Thus subtracting a baseline is unbiased in expectation but reduces variance of the sample gradient

Vanilla Policy Gradient Algorithm with Baseline

procedure POLICY GRADIENT(α)

Initialize policy parameters θ and baseline values $b(s)$ for all s, e.g. to 0

for iteration $= 1, 2, \ldots$ **do**

Collect a set of m trajectories by executing the current policy π_θ

for each time step t of each trajectory $\tau^{(i)}$ **do**

Compute the *return* $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}$

Compute the *advantage estimate* $\hat{A}_t^{(i)} = G_t^{(i)} - b(s_t)$

Re-fit the baseline to the empirical returns by updating $\mathbf{w}$ to minimize

this is the MSE loss function

$$\sum_{i=1}^{m} \sum_{t=0}^{T-1} \| b(s_t) - G_t^{(i)} \|^2$$

Update policy parameters θ using the policy gradient estimate $\hat{g}$

$$\hat{g} = \sum_{i=1}^{m} \sum_{t=0}^{T-1} \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

ghat is the estimated gradient

if we're using a softmax policy over some weights, gradient would be \nabla_\theta = target action one hot encoded vector - probabilitiy vector

with an optimizer like SGD ($\theta \leftarrow \theta + \alpha \cdot \hat{g}$) or Adam

return θ and baseline values $b(s)$

1. Classic paper: Sutton, McAllester, Singh, Mansour (1999). Policy gradient methods for reinforcement learning with function approximation: actor-critic algorithms with value function approximation

Practical Implementation of the Algorithm

1. In practice we usually do not compute the gradients $\sum_t \hat{A}_t \nabla_\theta \log \pi_\theta(a_t|s_t)$ individually
2. Instead, we accumulate data from current batch as

we can treat this as a a "fake" objective function, so we get the policy gradient estimator when we differentiate

$$L(\theta) = \sum_t \hat{A}_t \log \pi_\theta(a_t|s_t;\theta)$$

3. Then the policy gradient estimator $\hat{g} = \nabla_\theta L(\theta)$
4. We also could have a joint loss with value function approximation as

$$L(\theta, \mathbf{w}) = \sum_t \Big(\hat{A}_t \log \pi_\theta(a_t|s_t;\theta) - ||b(s_t) - \hat{R}_t||^2 \Big)$$

maximize reward

minimize prediction error

Rhat is the trajectory that we've calculated alr (observed retu

w is used to approximate the second part of the loss

5. Then solve this using auto diff

auto diff allows you to calculate loss without for loop

Reduce Variance Using a Critic

1. The update is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

2. In practice, G_t is a sample from Monte Carlo policy gradient, which is the unbiased but noisy estimate of $Q^{\pi_\theta}(s_t, a_t)$ q function is the expected return
3. Instead we can use a critic to estimate the action-value function,

$$Q_{\mathbf{w}}(s, a) \approx Q^{\pi_\theta}(s, a)$$

4. Then the update becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

Actor-Critic Policy Gradient

we used a learned prediction of the return

these use the weight w

This replaces G_t

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

1. It becomes Actor-Critic Policy Gradient — wants to maximize the value received from the critic
   1. Actor: the policy function used to generate the action
   2. Critic: the value function used to evaluate the reward of the actions
2. Actor-critic algorithms maintain two sets of parameters
   1. Actor: Updates policy parameters θ, in direction suggested by critic
   2. Critic: Updates action-value function parameters $\mathbf{w}$

low variance and high bias if the critic is dumb

this way we don't need to finish an entire trajectory, we update on the go

Estimate the Action-Value Function

1. Estimating the critic is solving a familiar problem: policy evaluation
    1. How good is policy π_θ under current parameter θ
2. Policy evaluation was explored in previous lectures, e.g.
    1. Monte-Carlo policy evaluation
    2. Temporal-Difference learning
    3. Least-squares policy evaluation

Action-Value Actor-Critic Algorithm

features of (s, a)

1. Using a linear value function approximation: $Q_{\mathbf{w}}(s,a) = \psi(s,a)^T \mathbf{w}$
   1. Critic: update $\mathbf{w}$ by a linear TD(0)
   2. Actor: update θ by policy gradient

linear aproximator

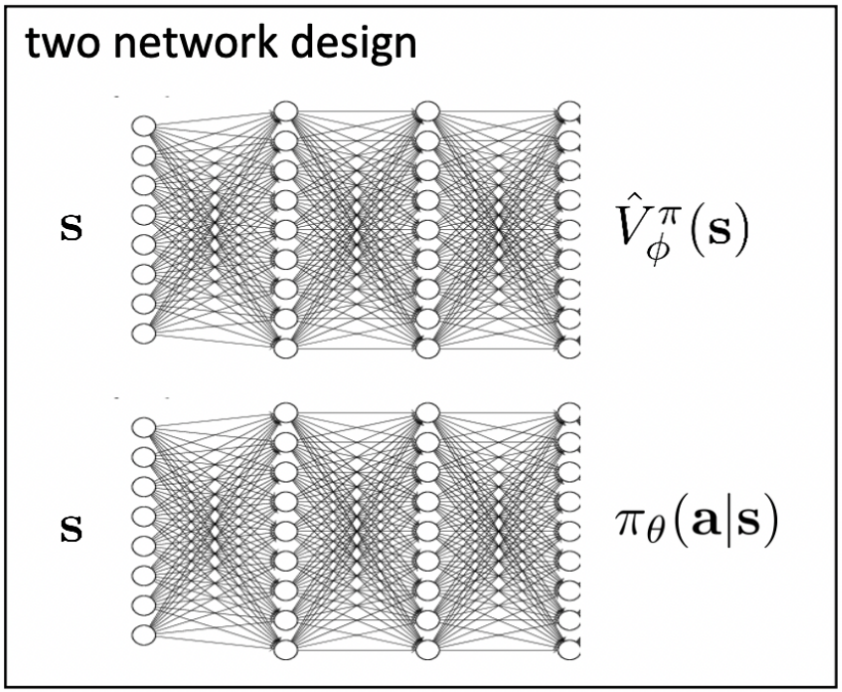
Algorithm 3 Simple QAC

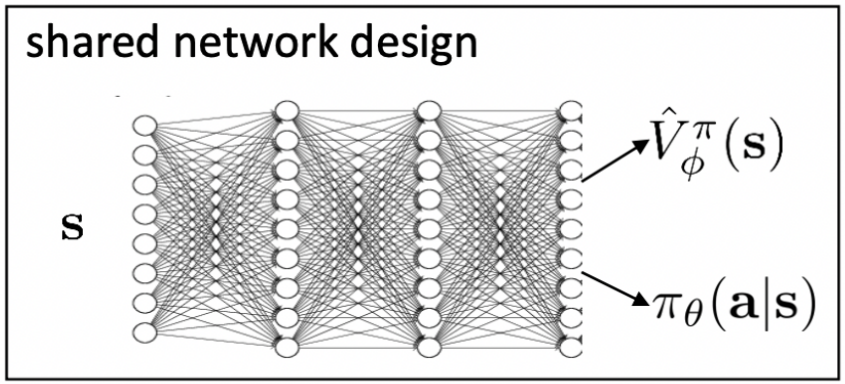
for each step **do**
  generate sample s, a, r, s', a' following π_θ
  $\delta = r + \gamma Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$ #TD error
  $\mathbf{w} \leftarrow \mathbf{w} - \beta \delta \psi(s, a)$ #Gradient descend
  $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) Q_{\mathbf{w}}(s, a)$ #Gradient ascend
end for

\pi_\theta is like a neurla network where \theta is the weights and biases, THIS MAKES SO MJUCH MORE SENSE

Actor-Critic Function Approximators

1. We can have two separate functions to approximate value function and policy function, or use a shared network design (feature extraction is shared but output two heads), as below:





Compatible Function Approximation

Theorem (Compatible Function Approximation Theorem)

If the following two conditions are satisfied:

1. *Value function approximator is compatible to the policy*

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

2. *Value function parameters w minimize the mean-squared error*

$$\epsilon(w) = E_{\pi_\theta}[(Q^{\pi_\theta}(s,a) - Q_w(s,a))^2]$$

then the policy gradient is exact

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

RS Sutton, et al. Policy gradient methods for reinforcement learning with function approximation. NIPS'01

Compatible Function Approximation

1. Given a parameterized policy, compatible function approximation theorem can be used to derive an appropriate form of the value-function parameterization.
   1. For example, consider a softmax policy as Gibbs distribution where ϕ_{sa} is the feature vector for state-action pair s,a

$$\pi_\theta(a|s) = \frac{e^{\theta^T \phi_{sa}}}{\sum_b e^{\theta^T \phi_{sb}}} \tag{4}$$

   2. Then meeting the compatibility condition requires

$$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(a|s) = \phi_{sa} - \sum_b \pi_\theta(s,b)\phi_{sb}$$

   3. so the compatible parameterization of $Q_w(s,a)$ is

$$Q_w(s,a) = w^T \left(\phi_{sa} - \sum_b \pi_\theta(s,b)\phi_{sb}\right) = w^T \nabla_\theta \log \pi_\theta(a|s)$$

Reducing the Variance of Actor-Critic by a Baseline

1. Recall Q-function / state-action-value function:

(discounted future rewards)

$$Q^{\pi,\gamma}(s,a) = \mathbb{E}_{\pi}[r_1 + \gamma r_2 + ...|s_1 = s, a_1 = a]$$

2. State value function can serve as a great baseline

averaged over all the actions at a state (weighted by the probability that we choose an action)

$$\begin{aligned} V^{\pi,\gamma}(s) =& \mathbb{E}_{\pi}[r_1 + \gamma r_2 + ...|s_1 = s] \\ =& \mathbb{E}_{a\sim\pi}[Q^{\pi,\gamma}(s,a)] \end{aligned}$$

3. Advantage function: combining Q with baseline V

$$A^{\pi,\gamma}(s,a) = Q^{\pi,\gamma}(s,a) - V^{\pi,\gamma}(s)$$

4. Then the policy gradient becomes:

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

Advantage Actor-Critic

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

1. Critic should estimate the advantage function
2. Use two function approximators and two sets of parameter vectors,

$$V_{\mathbf{v}}(s) \approx V^{\pi}(s)$$
$$Q_{\mathbf{w}}(s,a) \approx Q^{\pi}(s,a)$$

3. Update both parameters by TD learning or MC

Advantage Actor-Critic

1. For the true value function $V^{\pi_\theta}(s)$, the TD error δ^{π_θ}

   discounted value of the next state

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

   baseline

   is an unbiased estimate of the advantage function

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

2. So we can use the TD error to compute the policy gradient

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

3. It requires only one set of critic parameter κ that is estimated by TD

   since we don't have a q value function, we just need to have one set of weights for the value function

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

N-step estimators

1. We used the Monte-Carlo estimates of the reward
2. We can also us TD methods for the policy gradient update, or any intermediate blend between TD and MC methods:
3. Consider the following n-step returns for $n = 1, 2, \infty$

$$\begin{aligned} n = 1 (TD) \quad & G_t^{(1)} = r_{t+1} + \gamma v(s_{t+1}) \\ n = 2 \quad & G_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) \\ n = \infty (MC) \quad & G_t^{(\infty)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-t-1} r_T \end{aligned}$$

4. Then the advantage estimators become

$$\begin{aligned} \hat{A}_t^{(1)} &= r_{t+1} + \gamma v(s_{t+1}) - v(s_t) \\ \hat{A}_t^{(2)} &= r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) - v(s_t) \\ \hat{A}_t^{(\infty)} &= r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-t-1} r_T - v(s_t) \end{aligned}$$

5. $\hat{A}^{(1)}$ has low variance and high bias. $\hat{A}^{(\infty)}$ has high variance but low bias

Critic at Different Time-Scales

1. Critic can estimate linear value function $V_\kappa(s) = \psi(s)^T \kappa$ from many targets as follows
2. For MC, the update is

$$\Delta\kappa = \alpha(G_t - V_\kappa(s))\psi(s)$$

3. For TD(0), the update is

$$\Delta\kappa = \alpha(r + \gamma V_\kappa(s') - V_\kappa(s))\psi(s)$$

4. For k-step return, the update is

$$\Delta\kappa = \alpha(\sum_{i=0}^{k} \gamma^i r_{t+i} + \gamma^k V_\kappa(s_{t+k}) - V_\kappa(s_t))\psi(s)$$

Actors at Different Time-Scales

1. The policy gradient can also be estimated at many time-scales

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

2. Monte-Carlo Actor-Critic policy gradient uses error from complete return

$$\Delta\theta = \alpha(G_t - V_\kappa(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

3. TD Difference Actor-Critic policy gradient uses TD error

$$\Delta\theta = \alpha(r + \gamma V_\kappa(s_{t+1}) - V_\kappa(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

4. k-step return Actor-Critic policy gradient uses the k-step return error

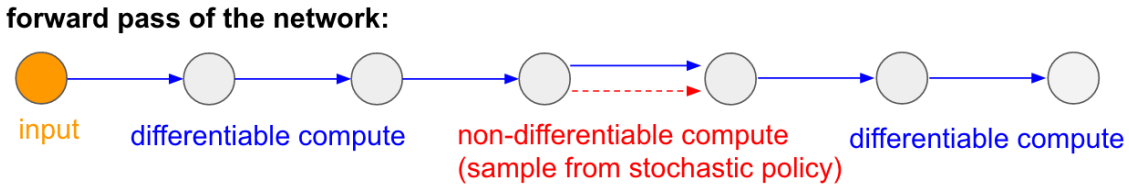
$$\Delta\theta = \alpha(\sum_{i=0}^{k} \gamma^i r_{t+i} + \gamma^k V_\kappa(s_{t+k}) - V_\kappa(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

Property of PG: Policy Gradient is On-Policy RL

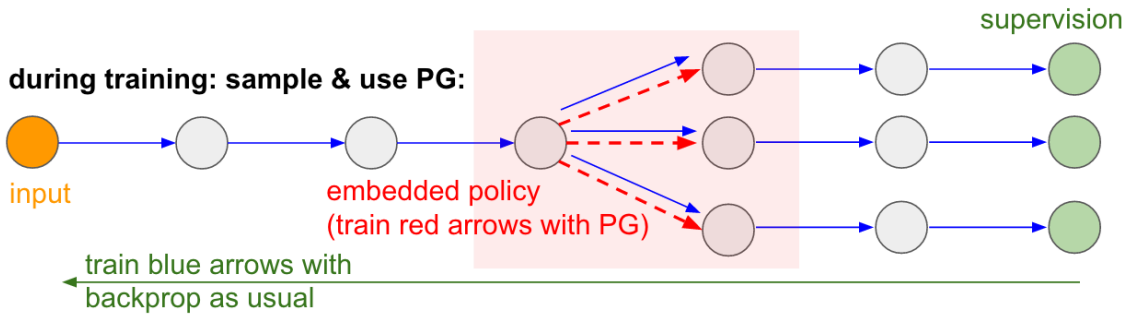
1. $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log P_\theta(\tau) r(\tau)]$
2. In REINFORCE algorithm: there is the on-policy sampling (sample $\{\tau\}$ from π_θ)
3. On-policy learning is stable but might be inefficient

Property of PG: Overcoming Non-differentiable Computation

1. Another interesting strength of Policy Gradient is that it allows us to overcome the non-differentiable computation



2. During training we will produce several samples (indicated by the branches below), and then we'll encourage samples that eventually led to good outcomes (in this case for example measured by the loss at the end)



Summary of Policy Gradient

1. Softmax policy: weight actions using linear combination of features $\phi(s,a)^T\theta$

$$\pi_\theta(a|s) = \frac{\exp^{\phi(s,a)^T\theta}}{\sum_{a'} \exp^{\phi(s,a')^T\theta}}$$

2. Gaussian policy:
   1. Mean is a linear combination of state features $\mu(s) = \phi(s)^T\theta$
   2. Variance may be fixed σ^2 or can also be parameterized
   3. Policy is Gaussian, the continuous $a \sim \mathcal{N}(\mu(s), \sigma^2)$
3. Neural network policy: $\pi_\theta(a|s) = F(s)$ where $F(\cdot)$ is a neural network

Summary of Policy Gradient

1. In policy optimization, for the policy function π_θ the objective is to maximize

this is an on-policy loss function

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

   1. $R(\tau)$ could be any reasonable reward function on τ
2. So the gradient is

$$\begin{aligned}\nabla_\theta J(\theta) =& \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\ =& \mathbb{E}_\tau[R(\tau) \nabla_\theta \log P(\tau;\theta)]\end{aligned}$$

   1. **log likelihood trick**: $\nabla_\theta f_\theta(x) = f_\theta(x) \nabla_\theta \log f_\theta(x)$
   2. Trajectory distribution: $P(\tau;\theta) = \mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$
   3. **canceling trick**: $\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Summary of Policy Gradient

1. After decomposing τ we derived that

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\Big(\sum_{t=0}^{T-1} r_t \Big) \Big(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \Big) \Big]$$

2. After applying the causality, we have

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

   1. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory at step t

Summary of Policy Gradient

1. The policy gradient has many forms

$$\begin{aligned}\nabla_\theta J(\theta) =& \mathbb{E}_\tau[G_t \nabla_\theta \log \pi_\theta(a|s)] \text{ — REINFORCE} \\ =& \mathbb{E}_\tau[Q^\pi(s,a) \nabla_\theta \log \pi_\theta(a|s)] \text{ — Q Actor-Critic} \\ =& \mathbb{E}_\tau[A^\pi(s,a) \nabla_\theta \log \pi_\theta(a|s)] \text{— Advantage Actor-Critic} \\ =& \mathbb{E}_\tau[\delta \nabla_\theta \log \pi_\theta(a|s)] \text{— TD Actor-Critic}\end{aligned}$$

2. We can use policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$, or $V^\pi(s,a)$

Extension of Policy Gradient

1. State-of-the-art RL methods are almost all policy-based
    1. **A2C and A3C**: Asynchronous Methods for Deep Reinforcement Learning, ICML'16. Representative high-performance actor-critic algorithm: `https://openai.com/blog/baselines-acktr-a2c/`
    2. **TRPO**: Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
    3. **PPO**: Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient

Resource on policy gradient

1. A very nice summary of policy gradient algorithms: `https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html`
2. Educational blog by Karpathy: `http://karpathy.github.io/2016/05/31/rl/`

Code example

1. REINFORCE code on CartPole: https://github.com/ucla-rlcourse/RLexample/blob/master/policygradient/reinforce.py
2. Policy Gradient on Pong : https://github.com/ucla-rlcourse/RLexample/blob/master/policygradient/pg-pong-pytorch.py
3. Policy Gradient with Baseline on Pong: https://github.com/ucla-rlcourse/RLexample/blob/master/policygradient/pgb-pong-pytorch.py
4. Actor Critic on Pong: https://github.com/ucla-rlcourse/RLexample/blob/master/policygradient/ac-pong-pytorch.py