Lecture 8: Policy Gradient I [1]

Emma Brunskill

CS234 Reinforcement Learning.

- Additional reading: Sutton and Barto 2018 Chp. 13

[1] With many slides from or derived from David Silver and John Schulman and Pieter Abbeel

Refresh Your Knowledge. Comparing Policy Performance

- This question asks you to think back to the performance difference lemma. Consider policy π_1 and π_2. (select all)
  1. We can define the performance difference lemma as $V^{\pi_1}(s_0) - V^{\pi_2}(s_0) = \sum_{t=0}^{H} \mathbb{E}_{s\sim\pi_1}[V^{\pi_1}(s) - Q^{\pi_1}(s, \pi_2(s))]$
  2. We can define the performance difference lemma as $V^{\pi_1}(s_0) - V^{\pi_2}(s_0) = \sum_{t=0}^{H} \mathbb{E}_{s\sim\pi_2}[V^{\pi_1}(s) - Q^{\pi_1}(s, \pi_2(s))]$
  3. We can define the performance difference lemma as $V^{\pi_1}(s_0) - V^{\pi_2}(s_0) = \sum_{t=0}^{H} \mathbb{E}_{s\sim\pi_2}[Q^{\pi_1}(s, \pi_1(s)) - Q^{\pi_1}(s, \pi_2(s))]$
  4. We require data from both policies to evaluate the performance difference lemma
  5. Not sure

Refresh Your Knowledge. Comparing Policy Performance

- This question asks you to think back to the performance difference lemma. Consider policy π_1 and π_2. (select all)
  1. We can define the performance difference lemma as $V^{\pi_1}(s_0) - V^{\pi_2}(s_0) = \sum_{t=0}^{H} \mathbb{E}_{s\sim\pi_1}[V^{\pi_1}(s) - Q^{\pi_1}(s, \pi_2(s))]$
  2. We can define the performance difference lemma as $V^{\pi_1}(s_0) - V^{\pi_2}(s_0) = \sum_{t=0}^{H} \mathbb{E}_{s\sim\pi_2}[V^{\pi_1}(s) - Q^{\pi_1}(s, \pi_2(s))]$
  3. We can define the performance difference lemma as $V^{\pi_1}(s_0) - V^{\pi_2}(s_0) = \sum_{t=0}^{H} \mathbb{E}_{s\sim\pi_2}[Q^{\pi_1}(s, \pi_1(s)) - Q^{\pi_1}(s, \pi_2(s))]$
  4. We require data from both policies to evaluate the performance difference lemma
  5. Not sure

Answer: 2 and 3 are correct. 4 is false because we only need data from one policy.

Last Time: We want RL Algorithms that Perform

- Optimization
- Delayed consequences
- Exploration
- Generalization
- And do it statistically and computationally efficiently

Last Time: Generalization and Efficiency

- Can use structure and additional knowledge to help constrain and speed reinforcement learning

Class Structure

- Last time: Deep RL
- **This time: Policy Search**
- Next time: Policy Search Cont.

Today

- Introduction to policy search methods
- Gradient-free methods
- Finite difference methods
- Score functions and policy gradient
- REINFORCE

Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters w,

$$V_w(s) \approx V^\pi(s)$$

$$Q_w(s,a) \approx Q^\pi(s,a)$$

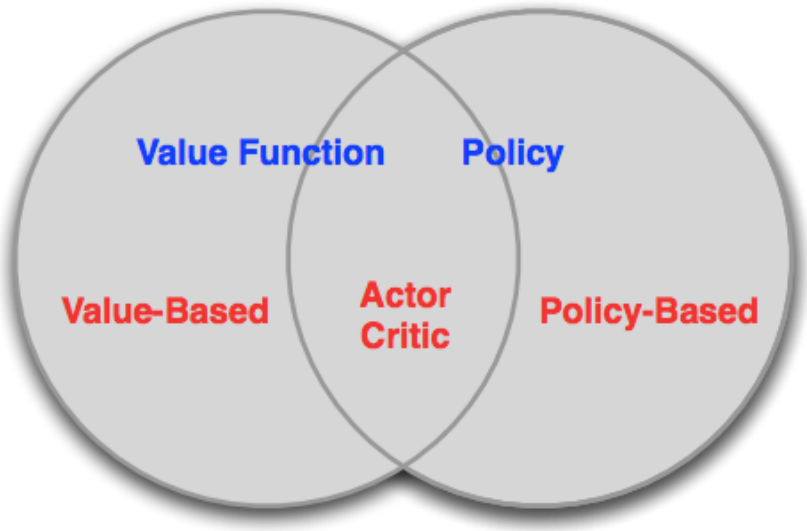
- A policy was generated directly from the value function
  - e.g. using ϵ-greedy
- In this lecture we will directly parametrize the policy, and will typically use θ to show parameterization:

$$\pi_\theta(s,a) = \mathbb{P}[a|s;\theta]$$

- Goal is to find a policy π with the highest value function V^π
- We will focus again on model-free reinforcement learning

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



Types of Policies to Search Over

- So far have focused on deterministic policies (why?)
- Now we are thinking about direct policy search in RL, will focus heavily on stochastic policies

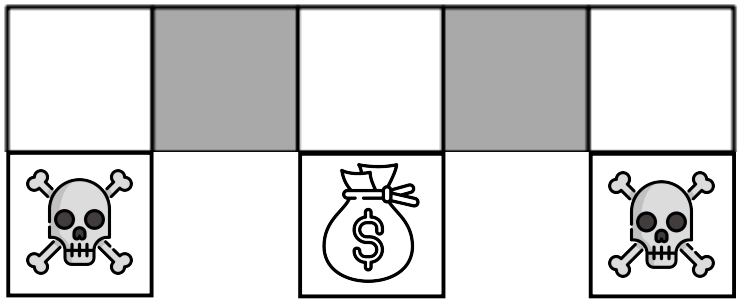
Example: Rock-Paper-Scissors

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Let state be history of prior actions (rock, paper and scissors) and if won or lost
- Is deterministic policy optimal? Why or why not?

Example: Rock-Paper-Scissors, Vote

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Let state be history of prior actions (rock, paper and scissors) and if won or lost

Example: Aliased Gridword (1)



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

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

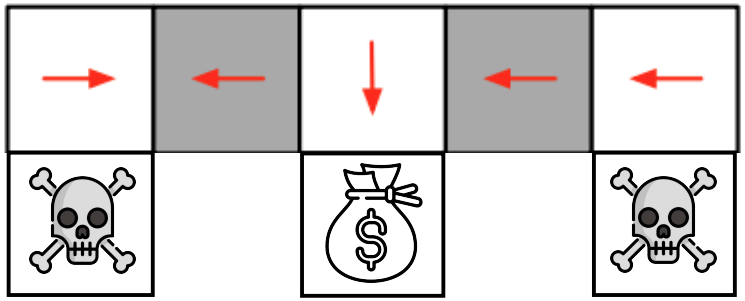
- Compare value-based RL, using an approximate value function

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

- To policy-based RL, using a parametrized policy

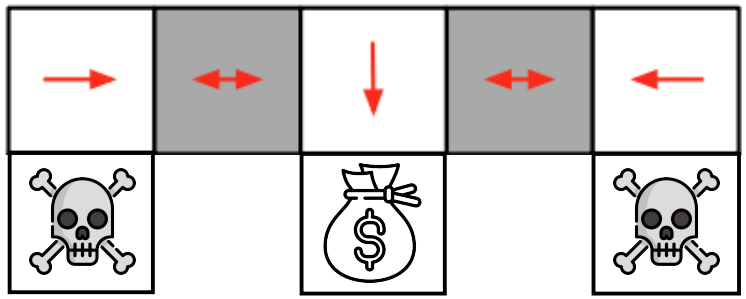
$$\pi_\theta(s, a) = g(\phi(s, a); \theta)$$

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 ϵ-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

Policy Objective Functions

- Goal: given a policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality for a policy π_θ?
- In episodic environments can use policy value at start state $V(s_0, \theta)$
- For simplicity, today will mostly discuss the episodic case, but can easily extend to the continuing / infinite horizon case

Policy optimization

- Policy based reinforcement learning is an optimization problem
- Find policy parameters θ that maximize $V(s_0, \theta)$

Today

- Introduction to policy search methods
- **Gradient-free methods**
- Finite difference methods
- Score functions and policy gradient
- REINFORCE

Policy optimization

- Policy based reinforcement learning is an optimization problem
- Find policy parameters θ that maximize $V(s_0, \theta)$
- Can use gradient free optimization
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
  - Cross-Entropy method (CEM)
  - Covariance Matrix Adaptation (CMA)

Human-in-the-Loop Exoskeleton Optimization (Zhang et al. Science 2017)

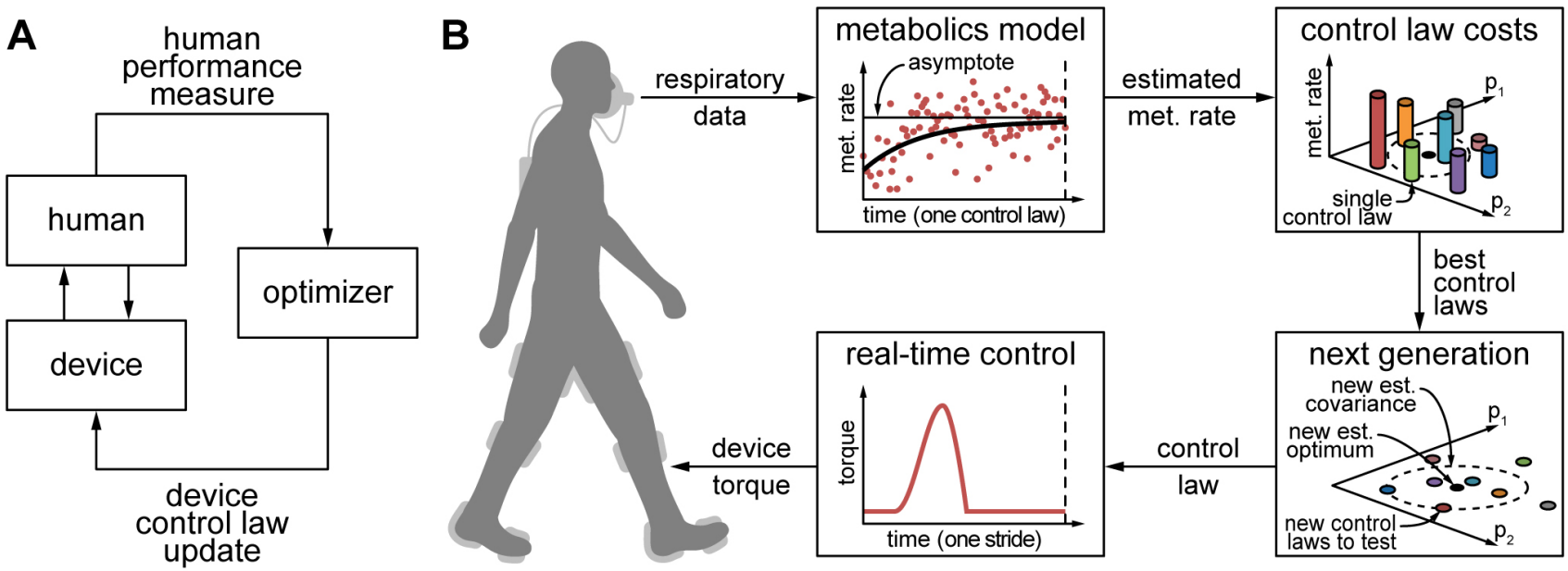


Figure: Zhang et al. Science 2017

- Optimization was done using CMA-ES, variation of covariance matrix evaluation

Gradient Free Policy Optimization

- Can often work embarrassingly well: "discovered that evolution strategies (ES), an optimization technique that's been known for decades, rivals the performance of standard reinforcement learning (RL) techniques on modern RL benchmarks (e.g. Atari/MuJoCo)" (https://blog.openai.com/evolution-strategies/)

Gradient Free Policy Optimization

- Often a great simple baseline to try
- Benefits
  - Can work with any policy parameterizations, including non-differentiable
  - Frequently very easy to parallelize
- Limitations
  - Typically not very sample efficient because it ignores temporal structure

Today

- Introduction to policy search methods
- Gradient-free methods
- **Finite difference methods**
- Score functions and policy gradient
- REINFORCE

Policy optimization

- Policy based reinforcement learning is an optimization problem
- Find policy parameters θ that maximize $V(s_0, \theta)$
- Can use gradient free optimization:
- 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

- Define $V(\theta) = V(s_0, \theta)$ to make explicit the dependence of the value on the policy parameters [but don't confuse with value function approximation, where parameterized value function]
- Assume episodic MDPs (easy to extend to related objectives, like average reward)

Policy Gradient

- Define $V^{\pi_\theta} = V(s_0, \theta)$ to make explicit the dependence of the value on the policy parameters
- Assume episodic MDPs
- Policy gradient algorithms search for a *local* maximum in $V(s_0, \theta)$ by ascending the gradient of the policy, w.r.t parameters θ

$$\Delta\theta = \alpha \nabla_\theta V(s_0, \theta)$$

- Where $\nabla_\theta V(s_0, \theta)$ is the policy gradient

$$\nabla_\theta V(s_0, \theta) = \begin{pmatrix} \frac{\partial V(s_0,\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial V(s_0,\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter

Simple Approach: Compute Gradients by Finite Differences

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

$$\frac{\partial V(s_0, \theta)}{\partial \theta_k} \approx \frac{V(s_0, \theta + \epsilon u_k) - V(s_0, \theta)}{\epsilon}$$

where u_k is a unit vector with 1 in kth component, 0 elsewhere.

Computing Gradients by Finite Differences

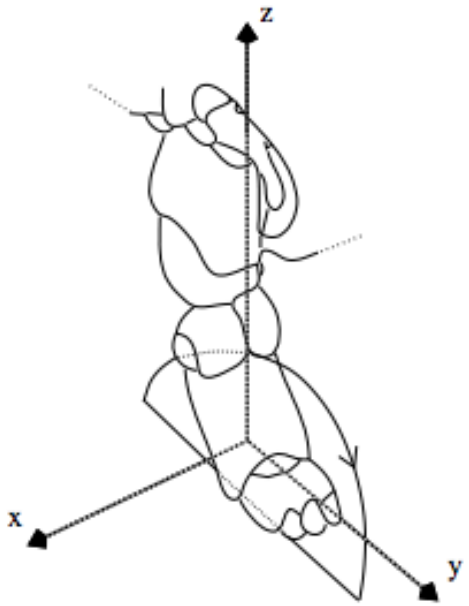
- To evaluate policy gradient of $\pi_\theta(s, a)$
- For each dimension $k \in [1, n]$
  - Estimate kth partial derivative of objective function w.r.t. θ
  - By perturbing θ by small amount ϵ in kth dimension

$$\frac{\partial V(s_0, \theta)}{\partial \theta_k} \approx \frac{V(s_0, \theta + \epsilon u_k) - V(s_0, \theta)}{\epsilon}$$

where u_k is a unit vector with 1 in kth 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[1]



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

[1] Kohl and Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. ICRA 2004. http://www.cs.utexas.edu/ ai-lab/pubs/icra04.pdf

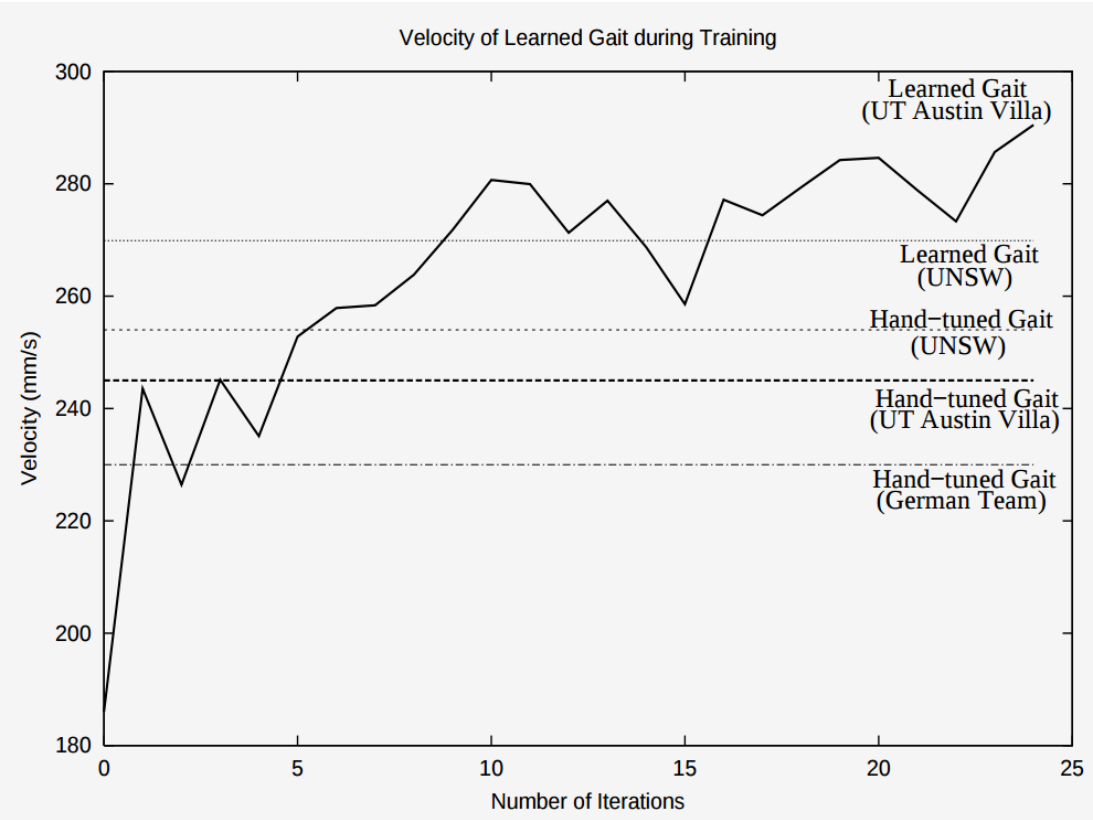
AIBO Policy Parameterization

- AIBO walk policy is open-loop policy
- No state, choosing set of action parameters that define an ellipse
- Specified by 12 continuous parameters (elliptical loci)
  - The front locus (3 parameters: height, x-pos., y-pos.)
  - The rear locus (3 parameters)
  - Locus length
  - Locus skew multiplier in the x-y plane (for turning)
  - The height of the front of the body
  - The height of the rear of the body
  - The time each foot takes to move through its locus
  - The fraction of time each foot spends on the ground
- New policies: for each parameter, randomly add (ϵ, 0, or $-\epsilon$)

AIBO Policy Experiments

- "All of the policy evaluations took place on actual robots... only human intervention required during an experiment involved replacing discharged batteries ... about once an hour."
- Ran on 3 Aibos at once
- Evaluated 15 policies per iteration.
- Each policy evaluated 3 times (to reduce noise) and averaged
- Each iteration took 7.5 minutes

Training AIBO to Walk by Finite Difference Policy Gradient Results



- Authors discuss that performance is likely impacted by: initial starting policy parameters, ϵ (how much policies are perturbed), learning rate for how much to change policy, as well as policy parameterization

Check Your Understanding

- Finite difference policy gradient (select all)
  1. Is guaranteed to converge to a local optima
  2. Is guaranteed to converge to a global optima
  3. Relies on the Markov assumption
  4. Uses a number of evaluations to estimate the gradient that scales linearly with the state dimensionality
  5. Not sure

Check Your Understanding

- Finite difference policy gradient (select all)
  1. Is guaranteed to converge to a local optima
  2. Is guaranteed to converge to a global optima
  3. Relies on the Markov assumption
  4. Uses a number of evaluations to estimate the gradient that scales linearly with the state dimensionality
  5. Not sure

Answer: Is guaranteed to converge to a local optima (not global), does not rely on the Markov assumption, uses a number of evaluations that scales linearly with the policy feature dimension (not state)

Summary of Benefits 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

Shortly will see some ideas to help with this last limitation

Today

- Introduction to policy search methods
- Gradient-free methods
- Finite difference methods
- **Score functions and policy gradient**
- REINFORCE

Computing the gradient analytically

- We now compute the policy gradient *analytically*
- Assume policy π_θ is differentiable whenever it is non-zero
- Assume we can calculate gradient $\nabla_\theta \pi_\theta(s, a)$ analytically
- What kinds of policy classes can we do this for?

Differentiable Policy Classes

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

Notation: Score Function

- A score function is the derivative of the log of a parameterized probability / likelihood
- Example: let $p(s;\theta)$ be the probability of state s under parameter θ
- Then the score function would be

$$\nabla_\theta \log p(s;\theta) \tag{1}$$

Softmax Policy

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

$$\pi_\theta(s,a) = e^{\phi(s,a)^T\theta}/(\sum_a e^{\phi(s,a)^T\theta})$$

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

Softmax Policy

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

$$\pi_\theta(s,a) = e^{\phi(s,a)^T\theta} / (\sum_a e^{\phi(s,a)^T\theta})$$

- The score function is

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

Gaussian Policy

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

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

Value of a Parameterized Policy

- Now assume policy π_θ is differentiable whenever it is non-zero and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Recall policy value is $V(s_0, \theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta, s_0\right]$ where the expectation is taken over the states & actions visited by π_θ
- We can re-express this in multiple ways
  - $V(s_0, \theta) = \sum_a \pi_\theta(a|s_0) Q(s_0, a, \theta)$

Value of a Parameterized Policy

- Now assume policy π_θ is differentiable whenever it is non-zero and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Recall policy value is $V(s_0, \theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta, s_0\right]$ where the expectation is taken over the states & actions visited by π_θ
- We can re-express this in multiple ways
  - $V(s_0, \theta) = \sum_a \pi_\theta(a|s_0) Q(s_0, a, \theta)$
  - $V(s_0, \theta) = \sum_\tau P(\tau; \theta) R(\tau)$
    - where $\tau = (s_0, a_0, r_0, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ is a state-action trajectory,
    - $P(\tau; \theta)$ is used to denote the probability over trajectories when executing policy $\pi(\theta)$ starting in state s_0, and
    - $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ the sum of rewards for a trajectory τ
- To start will focus on this latter definition. See Chp 13.1-13.3 of SB for a nice discussion starting with the other definition

Likelihood Ratio Policies

- Denote a state-action trajectory as $\tau = (s_0, a_0, r_0, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T)$
- Use $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ to be the sum of rewards for a trajectory τ
- Policy value is

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

- where $P(\tau; \theta)$ is used to denote the probability over trajectories when executing policy $\pi(\theta)$
- In this new notation, our goal is to find the policy parameters θ:

$$\arg\max_{\theta} V(\theta) = \arg\max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters θ:

$$\arg\max_{\theta} V(\theta) = \arg\max_{\theta} \sum_{\tau} P(\tau;\theta)R(\tau)$$

- Take the gradient with respect to θ:

$$\nabla_\theta V(\theta) \quad = \quad \nabla_\theta \sum_{\tau} P(\tau;\theta)R(\tau)$$

Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters θ:

$$\arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Take the gradient with respect to θ:

$$\begin{aligned}
\nabla_\theta V(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_\tau P(\tau;\theta)R(\tau) \underbrace{\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)}}_{\text{likelihood ratio}} \\
&= \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)
\end{aligned}$$

Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters θ:

$$\arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Take the gradient with respect to θ:

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

- Approximate with empirical estimate for m sample trajectories under policy π_θ:

$$\nabla_\theta V(\theta) \approx \hat{g} = (1/m)\sum_{i=1}^{m} R(\tau^{(i)})\nabla_\theta \log P(\tau^{(i)};\theta)$$

Decomposing the Trajectories Into States and Actions

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

$$\nabla_\theta V(\theta) \approx \hat{g} = (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log P(\tau^{(i)})$$

$\nabla_\theta \log P(\tau^{(i)}; \theta) =$

Decomposing the Trajectories Into States and Actions

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

$$\nabla_\theta V(\theta) \approx \hat{g} = (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log P(\tau^{(i)})$$

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \log \Big[\underbrace{\mu(s_0)}_{\text{Initial state distrib.}} \prod_{t=0}^{T-1} \underbrace{\pi_\theta(a_t|s_t)}_{\text{policy}} \underbrace{P(s_{t+1}|s_t, a_t)}_{\text{dynamics model}} \Big] \\
&= \nabla_\theta \Big[\log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t, a_t) \Big] \\
&= \sum_{t=0}^{T-1} \underbrace{\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\text{no dynamics model required!}}
\end{aligned}$$

Score Function

- Consider score function as $\nabla_\theta \log \pi_\theta(s, a)$

Likelihood Ratio / Score Function Policy Gradient

- Putting this together
- Goal is to find the policy parameters θ:

$$\arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Approximate with empirical estimate for m sample paths under policy π_θ using score function:

$$\begin{aligned}\nabla_\theta V(\theta) &\approx \hat{g} = (1/m)\sum_{i=1}^{m} R(\tau^{(i)})\nabla_\theta \log P(\tau^{(i)};\theta) \\ &= (1/m)\sum_{i=1}^{m} R(\tau^{(i)})\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\end{aligned}$$

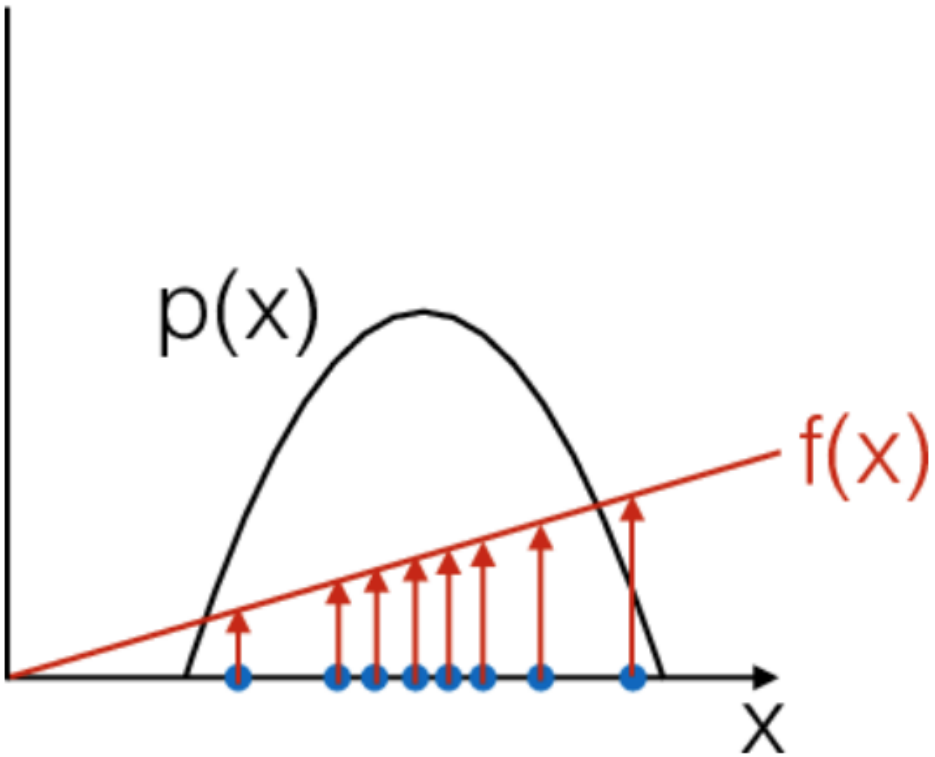
- Do not need to know dynamics model

Score Function Gradient Estimator: Intuition

- Consider generic form of $R(\tau^{(i)})\nabla_\theta \log P(\tau^{(i)};\theta)$:
  $\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$
- $f(x)$ measures how good the sample x is.
- Moving in the direction $\hat{g}_i$ pushes up the logprob of the sample, in proportion to how good it is
- *Valid even if $f(x)$ is discontinuous, and unknown, or sample space (containing x) is a discrete set*

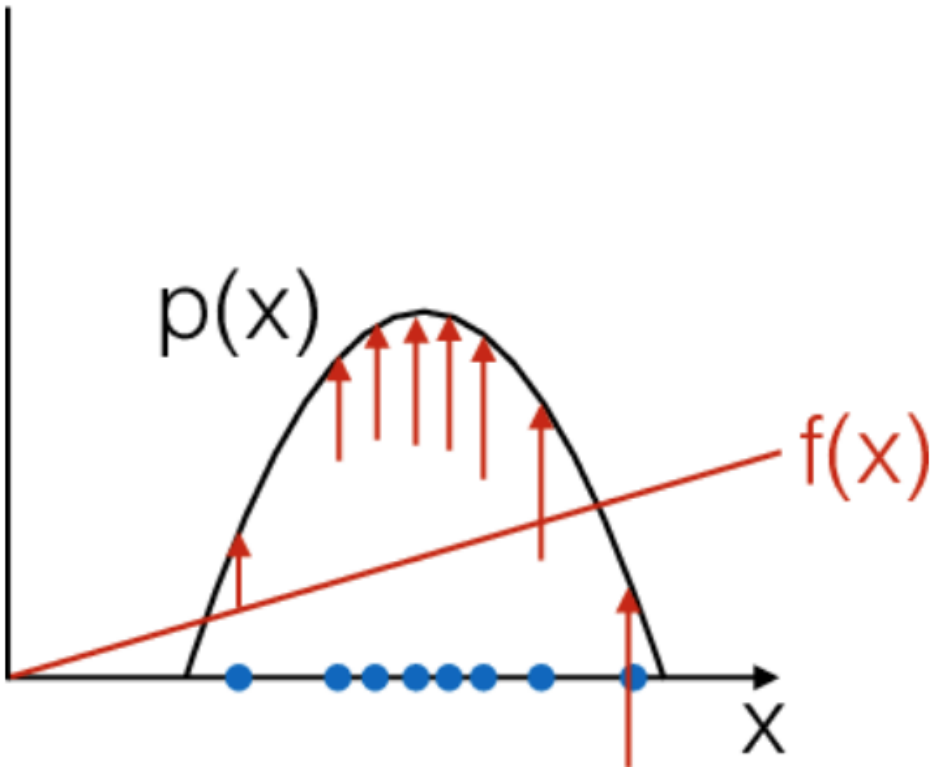
Score Function Gradient Estimator: Intuition

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$$



Score Function Gradient Estimator: Intuition

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$$



Policy Gradient Theorem

- The policy gradient theorem generalizes the likelihood ratio approach

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective function $J = J_1$, *(episodic reward),* J_{avR} *(average reward per time step), or* $\frac{1}{1-\gamma} J_{avV}$ *(average value),*
the policy gradient is

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

- Chapter 13.2 in SB has a nice derivation of the policy gradient theorem for episodic tasks and discrete states

Today

- Introduction to policy search methods
- Gradient-free methods
- Finite difference methods
- Score functions and policy gradient
- **REINFORCE**

Likelihood Ratio / Score Function Policy Gradient

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

- Unbiased but very noisy
- Fixes that can make it practical
  - Temporal structure
  - Baseline
- Next time will discuss some additional tricks

Policy Gradient: Use Temporal Structure

- Previously:

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

- We can repeat the same argument to derive the gradient estimator for a single reward term $r_{t'}$.

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

- Summing this formula over t, we obtain

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

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

Policy Gradient: Use Temporal Structure

- Recall for a particular trajectory $\tau^{(i)}$, $\sum_{t'=t}^{T-1} r_{t'}^{(i)}$ is the return $G_t^{(i)}$

$$\nabla_\theta \mathbb{E}[R] \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) G_t^{(i)}$$

Monte-Carlo Policy Gradient (REINFORCE)

- Leverages likelihood ratio / score function and temporal structure

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

REINFORCE:

Initialize policy parameters θ arbitrarily

for each episode $\{s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

  for $t = 1$ to $T - 1$ **do**

    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t$

  endfor

endfor

return θ

Likelihood Ratio / Score Function Policy Gradient

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

- Unbiased but very noisy
- Fixes that can make it practical
  - Temporal structure
  - **Baseline**
- Next time will discuss some additional tricks

Class Structure

- Last time: Imitation Learning in Large State Spaces
- **This time: Policy Search**
- Next time: Policy Search Cont.

Lecture 9: Policy Gradient II [1]

Emma Brunskill

CS234 Reinforcement Learning.

- Additional reading: Sutton and Barto 2018 Chp. 13

[1] With many slides from or derived from David Silver and John Schulman and Pieter Abbeel

Refresh Your Knowledge

- Select all that are true about policy gradients:
  1. $\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$
  2. θ is always increased in the direction of $\nabla_\theta \ln(\pi(S_t, A_t, \theta)$.
  3. State-action pairs with higher estimated Q values will increase in probability on average
  4. Are guaranteed to converge to the global optima of the policy class
  5. Not sure

Refresh Your Knowledge Solutions

- Select all that are true about policy gradients:
  1. $\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$
  2. θ is always increased in the direction of $\nabla_\theta \ln(\pi(S_t, A_t, \theta)$.
  3. State-action pairs with higher estimated Q values will increase in probability on average
  4. Are guaranteed to converge to the global optima of the policy class
  5. Not sure

  Answers: 1 and 3 are true. The direction of θ also depends on the Q-values /returns. We are only guaranteed to reach a local optima.

Class Structure

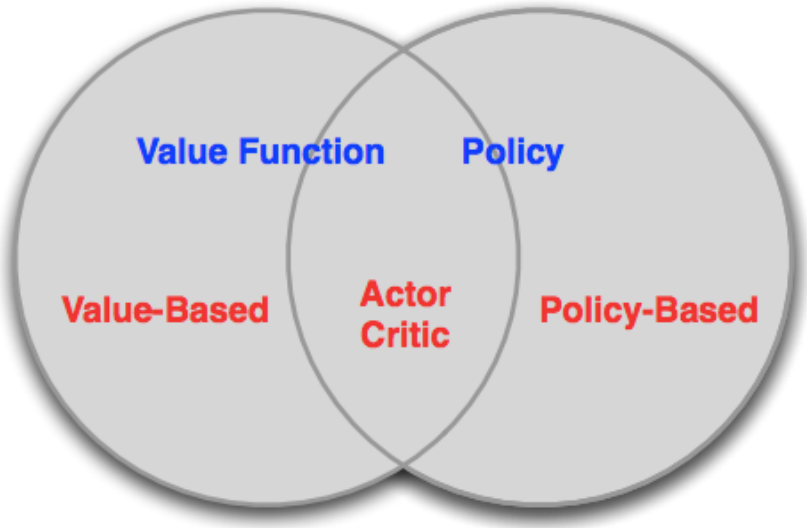
- Last time: Policy Search
- **This time: Policy Search**
- Next time: Exploration

Recall: Policy-Based RL

- Policy search: directly parametrize the policy

$$\pi_\theta(s, a) = \mathbb{P}[a|s; \theta]$$

- Goal is to find a policy π with the highest value function V^π
- (Pure) Policy based methods
  - No Value Function
  - Learned Policy
- Actor-Critic methods
  - Learned Value Function
  - Learned Policy



Recall: 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

Recall: Policy Gradient

- Defined $V(\theta) = V^{\pi_\theta}(s_0) = V(s_0, \theta)$ to make explicit the dependence of the value on the policy parameters
- Assumed episodic MDPs
- Policy gradient algorithms search for a *local* maximum of $V(\theta)$ by ascending the gradient of the policy, w.r.t parameters θ

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

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

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

- and α is a step-size hyperparameter

Desired Properties of a Policy Gradient RL Algorithm

- Goal: Converge as quickly as possible to a local optima
  - Incurring reward / cost as execute policy, so want to minimize number of iterations / time steps until reach a good policy

Desired Properties of a Policy Gradient RL Algorithm

- Goal: Converge as quickly as possible to a local optima
  - Incurring reward / cost as execute policy, so want to minimize number of iterations / time steps until reach a good policy
- During policy search alternating between evaluating policy and changing (improving) policy (just like in policy iteration)
- Would like each policy update to be a monotonic improvement
  - Only guaranteed to reach a local optima with gradient descent
  - Monotonic improvement will achieve this
  - And in the real world, monotonic improvement is often beneficial

Desired Properties of a Policy Gradient RL Algorithm

- Goal: Obtain large monotonic improvements to policy at each update
- Techniques to try to achieve this:
  - Last time and today: Get a better estimate of the gradient (intuition: should improve updating policy parameters)
  - Today: Change, how to update the policy parameters given the gradient

Table of Contents

Likelihood Ratio / Score Function Policy Gradient

- Recall last time (m is a set of trajectories):

$$\nabla_\theta V(s_0, \theta) \approx (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- Unbiased estimate of gradient but very noisy
- Fixes that can make it practical
  - Temporal structure (discussed last time)
  - Baseline
  - Alternatives to using Monte Carlo returns $R(\tau^{(i)})$ as targets

Policy Gradient: Introduce Baseline

- Reduce variance by introducing a *baseline* $b(s)$

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

- For any choice of b, gradient estimator is unbiased.
- Near optimal choice is the expected return,

$$b(s_t) \approx \mathbb{E}[r_t + r_{t+1} + \cdots + r_{T-1}]$$

- Interpretation: increase logprob of action a_t proportionally to how much returns $\sum_{t'=t}^{T-1} r_{t'}$ are better than expected

Baseline $b(s)$ Does Not Introduce Bias–Derivation

$$\mathbb{E}_\tau[\nabla_\theta \log \pi(a_t|s_t;\theta)b(s_t)]$$
$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta)b(s_t)]\right]$$

Baseline $b(s)$ Does Not Introduce Bias–Derivation

$$\mathbb{E}_\tau[\nabla_\theta \log \pi(a_t|s_t;\theta) b(s_t)]$$

$$= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T}, a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta) b(s_t)]\right] \text{ (break up expectation)}$$

$$= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{s_{(t+1):T}, a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta)]\right] \text{ (pull baseline term out)}$$

$$= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}[b(s_t)\mathbb{E}_{a_t}[\nabla_\theta \log \pi(a_t|s_t;\theta)]] \text{ (remove irrelevant variables)}$$

$$= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t)\sum_a \pi_\theta(a_t|s_t)\frac{\nabla_\theta \pi(a_t|s_t;\theta)}{\pi_\theta(a_t|s_t)}\right] \text{ (likelihood ratio)}$$

$$= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t)\sum_a \nabla_\theta \pi(a_t|s_t;\theta)\right]$$

$$= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t)\nabla_\theta \sum_a \pi(a_t|s_t;\theta)\right]$$

$$= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}[b(s_t)\nabla_\theta 1]$$

$$= \mathbb{E}_{s_0:t, a_0:(t-1)}[b(s_t)\cdot 0] = 0$$

"Vanilla" Policy Gradient Algorithm

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \cdots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep t in each trajectory τ^i, compute
    Return $G_t^i = \sum_{t'=t}^{T-1} r_{t'}^i$, and
    Advantage estimate $\hat{A}_t^i = G_t^i - b(s_t)$.
  Re-fit the baseline, by minimizing $\sum_i \sum_t ||b(s_t) - G_t^i||^2$,
  Update the policy, using a policy gradient estimate $\hat{g}$,
    Which is a sum of terms $\nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_t$.
    (Plug $\hat{g}$ into SGD or ADAM)
endfor

Other Choices for Baseline?

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \cdots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep t in each trajectory τ^i, compute
    Return $G_t^i = \sum_{t'=t}^{T-1} r_{t'}^i$, and
    Advantage estimate $\hat{A}_t^i = G_t^i - b(s_t)$.
  Re-fit the baseline, by minimizing $\sum_i \sum_t ||b(s_t) - G_t^i||^2$,
  Update the policy, using a policy gradient estimate $\hat{g}$,
    Which is a sum of terms $\nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_t$.
    (Plug $\hat{g}$ into SGD or ADAM)
endfor

Choosing the Baseline: Value Functions

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

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 \cdots | s_0 = s, a_0 = a\right]$$

- State-value function can serve as a great baseline

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 \cdots | s_0 = s\right] \\ &= \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a)] \end{aligned}$$

Table of Contents

Likelihood Ratio / Score Function Policy Gradient

- Recall last time:

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

- Unbiased estimate of gradient but very noisy
- Fixes that can make it practical
  - Temporal structure (discussed last time)
  - Baseline
  - **Alternatives to using Monte Carlo returns G_t^i as estimate of expected discounted sum of returns for the policy parameterized by θ?**

Choosing the Target

- G_t^i is an estimation of the value function at s_t from a single roll out
- Unbiased but high variance
- Reduce variance by introducing bias using bootstrapping and function approximation
  - Just like in we saw for TD vs MC, and value function approximation

Actor-critic Methods

- Estimate of V/Q is done by a **critic**
- **Actor-critic** methods maintain an explicit representation of policy and the value function, and update both
- A3C (Mnih et al. ICML 2016) is a very popular actor-critic method

Policy Gradient Formulas with Value Functions

- Recall:

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

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t;\theta) \left(Q(s_t;\boldsymbol{w}) - b(s_t) \right) \right]$$

- Letting the baseline be an estimate of the value V, we can represent the gradient in terms of the state-action advantage function

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t;\theta) \hat{A}^\pi(s_t,a_t) \right]$$

- where the advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

Choosing the Target: N-step estimators

$$\nabla_\theta V(\theta) \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} R_t^i \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- Note that critic can select any blend between TD and MC estimators for the target to substitute for the true state-action value function.

Choosing the Target: N-step estimators

$$\nabla_\theta V(\theta) \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} R_t^i \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- Note that critic can select any blend between TD and MC estimators for the target to substitute for the true state-action value function.

$$\hat{R}_t^{(1)} = r_t + \gamma V(s_{t+1})$$
$$\hat{R}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \quad \cdots$$
$$\hat{R}_t^{(\text{inf})} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \cdots$$

- If subtract baselines from the above, get advantage estimators

$$\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t^{(\text{inf})} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \cdots - V(s_t)$$

Check Your Understanding: Blended Advantage Estimators

$$\nabla_\theta V(\theta) \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} R_t^i \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- If subtract baselines from the above, get advantage estimators

$$\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t^{(\text{inf})} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \cdots - V(s_t)$$

- Select all that are true
- $\hat{A}_t^{(1)}$ has low variance & low bias.
- $\hat{A}_t^{(1)}$ has high variance & low bias.
- $\hat{A}_t^{(\infty)}$ low variance and high bias.
- $\hat{A}_t^{(\infty)}$ high variance and low bias.
- Not sure

Check Your Understanding: Blended Advantage Estimators Answers

$$\nabla_\theta V(\theta) \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} R_t^i \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- If subtract baselines from the above, get advantage estimators

$$\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t^{(\text{inf})} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \cdots - V(s_t)$$

- Solution: $\hat{A}_t^{(1)}$ has low variance & high bias. $\hat{A}_t^{(\infty)}$ high variance but low bias.

"Vanilla" Policy Gradient Algorithm

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \cdots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep t in each trajectory τ^i, compute
    Advantage estimate $\hat{A}_t^i$
  Update the policy, using a policy gradient estimate $\hat{g}$,
    Which is a sum of terms $\nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_t$.
    (**Plug $\hat{g}$ into SGD or ADAM**)
endfor

Table of Contents

Policy Gradient and Step Sizes

- Goal: Each step of policy gradient yields an updated policy π' whose value is greater than or equal to the prior policy π: $V^{\pi'} \geq V^{\pi}$
- Gradient descent approaches update the weights a small step in direction of gradient
- **First order** / linear approximation of the value function's dependence on the policy parameterization
- Locally a good approximation, further away less good

Why are step sizes a big deal in RL?

- Step size is important in any problem involving finding the optima of a function
- Supervised learning: Step too far $\rightarrow$ next updates will fix it
- Reinforcement learning
  - Step too far $\rightarrow$ bad policy
  - Next batch: collected under bad policy
  - **Policy is determining data collection!** Essentially controlling exploration and exploitation trade off due to particular policy parameters and the stochasticity of the policy
  - May not be able to recover from a bad choice, collapse in performance!

Simple Step Size

- Simple step-sizing: Line search in direction of gradient
  - Simple but expensive (perform evaluations along the line)
  - Naive: ignores where the first order approximation is good or bad

Policy Gradient Methods with Auto-Step-Size Selection

- Can we automatically ensure the updated policy π' has value greater than or equal to the prior policy π: $V^{\pi'} \geq V^{\pi}$?
- Consider this for the policy gradient setting, and hope to address this by modifying step size

Objective Function

- Goal: find policy parameters that maximize value function[1]

$$V(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t); \pi_\theta\right]$$

- where $s_0 \sim P(s_0)$, $a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
- Have access to samples from the current policy π_θ (param. by θ)
- Want to predict the value of a different policy (off policy learning!)

[1] For today we will primarily consider discounted value functions

Objective Function

- Goal: find policy parameters that maximize value function[1]

$$V(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t); \pi_\theta\right]$$

- where $s_0 \sim P(s_0)$, $a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
- Express value of $\tilde{\pi}$ in terms of advantage over π

$$\begin{aligned} V(\tilde{\theta}) &= V(\theta) + \mathbb{E}_{\pi_{\tilde{\theta}}}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right] && (1) \\ &= V(\theta) + \sum_s \mu_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a) && (2) \\ \mu_{\tilde{\pi}}(s) &= E_{\tilde{\pi}} \sum_{t=0}^{\infty} \gamma^t I(s_t = s) && (3) \end{aligned}$$

- $\mu_{\tilde{\pi}}(s)$ is the discounted weighted frequency of state s under policy $\tilde{\pi}$

[1] For today we will primarily consider discounted value functions

Objective Function

- Goal: find policy parameters that maximize value function[1]

$$V(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t); \pi_\theta\right]$$

- where $s_0 \sim \mu(s_0)$, $a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
- Express expected return of another policy in terms of the advantage over the original policy

$$V(\tilde{\theta}) = V(\theta) + \mathbb{E}_{\pi_{\tilde{\theta}}}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right] = V(\theta) + \sum_s \mu_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$$

- where $\mu_{\tilde{\pi}}(s)$ is defined as the discounted weighted frequency of state s under policy $\tilde{\pi}$
- We know the advantage A_π and $\tilde{\pi}$
- But we can't compute the above because we don't know $\mu_{\tilde{\pi}}$, the state distribution under the new proposed policy

[1] For today we will primarily consider discounted value functions

Table of Contents

Local approximation

- Can we remove the dependency on the discounted visitation frequencies under the new policy?
- Substitute in the discounted visitation frequencies under the current policy to define a new objective function:

$$L_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$$

- Note that $L_{\pi_{\theta_0}}(\pi_{\theta_0}) = V(\theta_0)$
- Gradient of L is identical to gradient of value function at policy parameterized evaluated at θ_0: $\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)|_{\theta=\theta_0} = \nabla_\theta V(\theta)|_{\theta=\theta_0}$

Conservative Policy Iteration

- Is there a bound on the performance of a new policy obtained by optimizing the surrogate objective?
- Consider mixture policies that blend between an old policy and a different policy

$$\pi_{new}(a|s) = (1-\beta)\pi_{old}(a|s) + \beta\pi'(a|s)$$

- In this case can guarantee a lower bound on value of the new π_{new}:

$$V^{\pi_{new}} \geq L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\beta^2$$

- where $\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}\left[A_\pi(s,a)\right]\right|$

Check Your Understanding: Conservative Policy Iteration

- Is there a bound on the performance of a new policy obtained by optimizing the surrogate objective?
- Consider mixture policies that blend between an old policy and a different policy

$$\pi_{new}(a|s) = (1-\beta)\pi_{old}(a|s) + \beta\pi'(a|s)$$

- In this case can guarantee a lower bound on value of the new π_{new}:

$$V^{\pi_{new}} \geq L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\beta^2$$

- where $\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}\left[A_\pi(s,a)\right]\right|$

What can we say about this lower bound? (Select all)

1. It is tight if $\pi_{new} = \pi_{old}$
2. It is most loose if $\beta = 1$
3. It is most tight if $\beta = 1$
4. It is most tight if $\beta = 0$
5. Not sure

Conservative Policy Iteration

- Is there a bound on the performance of a new policy obtained by optimizing the surrogate objective?
- Consider mixture policies that blend between an old policy and a different policy

$$\pi_{new}(a|s) = (1-\beta)\pi_{old}(a|s) + \beta\pi'(a|s)$$

- In this case can guarantee a lower bound on value of the new π_{new}:

$$V^{\pi_{new}} \geq L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\beta^2$$

- where $\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}\left[A_\pi(s,a)\right]\right|$

Find the Lower-Bound in General Stochastic Policies

- Would like to similarly obtain a lower bound on the potential performance for general stochastic policies (not just mixture policies)
- Recall $L_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$

Theorem

Let $D_{TV}^{\max}(\pi_1, \pi_2) = \max_s D_{TV}(\pi_1(\cdot|s), \pi_2(\cdot|s))$. *Then*

$$V^{\pi_{new}} \geq L_{\pi_{old}}(\pi_{new}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}(D_{TV}^{\max}(\pi_{old}, \pi_{new}))^2$$

where $\epsilon = \max_{s,a} |A_\pi(s,a)|$.

Find the Lower-Bound in General Stochastic Policies

- Would like to similarly obtain a lower bound on the potential performance for general stochastic policies (not just mixture policies)
- Recall $L_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$

Theorem

Let $D_{TV}^{\max}(\pi_1, \pi_2) = \max_s D_{TV}(\pi_1(\cdot|s), \pi_2(\cdot|s))$. *Then*

$$V^{\pi_{new}} \geq L_{\pi_{old}}(\pi_{new}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}(D_{TV}^{\max}(\pi_{old}, \pi_{new}))^2$$

where $\epsilon = \max_{s,a} |A_\pi(s,a)|$.

- Note that $D_{TV}(p,q)^2 \leq D_{KL}(p,q)$ for prob. distrib p and q.
- Then the above theorem immediately implies that

$$V^{\pi_{new}} \geq L_{\pi_{old}}(\pi_{new}) - \frac{4\epsilon\gamma}{(1-\gamma)^2} D_{KL}^{\max}(\pi_{old}, \pi_{new})$$

- where $D_{KL}^{\max}(\pi_1, \pi_2) = \max_s D_{KL}(\pi_1(\cdot|s), \pi_2(\cdot|s))$

Guaranteed Improvement[1]

- Goal is to compute a policy that maximizes the objective function defining the lower bound:

[1] $L_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$

Guaranteed Improvement[1]

- Goal is to compute a policy that maximizes the objective function defining the lower bound:

$$
\begin{aligned}
M_i(\pi) &= L_{\pi_i}(\pi) - \frac{4\epsilon\gamma}{(1-\gamma)^2} D_{KL}^{\max}(\pi_i, \pi) \\
V^{\pi_{i+1}} &\geq L_{\pi_i}(\pi_{i+1}) - \frac{4\epsilon\gamma}{(1-\gamma)^2} D_{KL}^{\max}(\pi_i, \pi_{i+1}) = M_i(\pi_{i+1}) \\
V^{\pi_i} &= M_i(\pi_i) = L_{\pi_i}(\pi_i) \\
V^{\pi_{i+1}} - V^{\pi_i} &\geq M_i(\pi_{i+1}) - M_i(\pi_i)
\end{aligned}
$$

- So as long as the new policy π_{i+1} is equal or an improvement compared to the old policy π_i with respect to the lower bound, we are guaranteed to to monotonically improve!
- The above is a type of Minorization-Maximization (MM) algorithm

[1] $L_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$

Guaranteed Improvement[1]

$$V^{\pi_{new}} \geq L_{\pi_{old}}(\pi_{new}) - \frac{4\epsilon\gamma}{(1-\gamma)^2} D_{KL}^{\max}(\pi_{old}, \pi_{new})$$

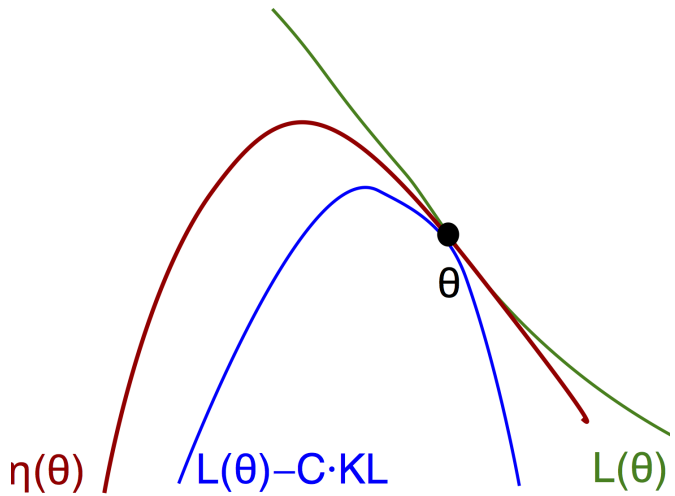


Figure: Source: John Schulman, Deep Reinforcement Learning, 2014

[1] $L_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$

Table of Contents

Optimization of Parameterized Policies[1]

- Goal is to optimize

$$\max_{\theta} L_{\theta_{old}}(\theta_{new}) - \frac{4\epsilon\gamma}{(1-\gamma)^2} D_{KL}^{\max}(\theta_{old}, \theta_{new}) = L_{\theta_{old}}(\theta_{new}) - C D_{KL}^{\max}(\theta_{old}, \theta_{new})$$

- where C is the penalty coefficient
- In practice, if we used the penalty coefficient recommended by the theory above $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$, the step sizes would be very small
- New idea: Use a trust region constraint on step sizes. Do this by imposing a constraint on the KL divergence between the new and old policy.

$$\max_{\theta} L_{\theta_{old}}(\theta)$$
$$\text{subject to } D_{KL}^{s\sim\mu_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta$$

- This uses the average KL instead of the max (the max requires the KL is bounded at all states and yields an impractical number of constraints)

[1] $L_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$

From Theory to Practice

- Prior objective:

$$\max_{\theta} L_{\theta_{old}}(\theta)$$

$$\text{subject to } D_{KL}^{s\sim\mu_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta$$

where $L_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$

- Don't know the visitation weights nor true advantage function
- Instead do the following substitutions:

$$\sum_s \mu_\pi(s) \rightarrow \frac{1}{1-\gamma} \mathbb{E}_{s\sim\mu_{\theta_{old}}}[\ldots],$$

From Theory to Practice

- Next substitution:

$$\sum_a \pi_\theta(a|s_n) A_{\theta_{old}}(s_n, a) \rightarrow \mathbb{E}_{a\sim q}\left[\frac{\pi_\theta(a|s_n)}{q(a|s_n)} A_{\theta_{old}}(s_n, a)\right]$$

- where q is some sampling distribution over the actions and s_n is a particular sampled state.
- This second substitution is to use importance sampling to estimate the desired sum, enabling the use of an alternate sampling distribution q (other than the new policy π_θ).
- Third substitution:

$$A_{\theta_{old}} \rightarrow Q_{\theta_{old}}$$

- Note that the above substitutions do not change solution to the above optimization problem

Selecting the Sampling Policy

- Optimize

$$\max_{\theta} \mathbb{E}_{s\sim\mu_{\theta_{old}},a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)}Q_{\theta_{old}}(s,a)\right]$$
$$\text{subject to } \mathbb{E}_{s\sim\mu_{\theta_{old}}} D_{KL}(\pi_{\theta_{old}}(\cdot|s),\pi_\theta(\cdot|s)) \leq \delta$$

- Standard approach: sampling distribution is $q(a|s)$ is simply $\pi_{old}(a|s)$
- For the vine procedure see the paper

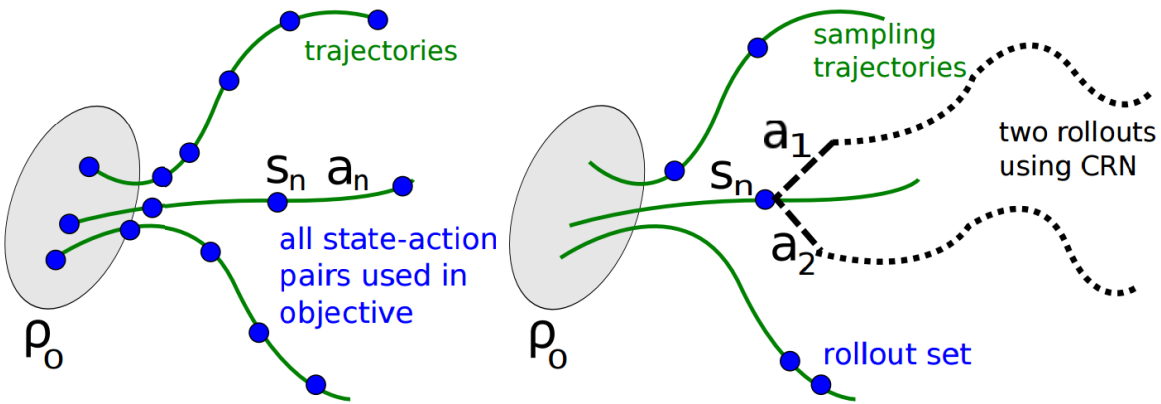


Figure: Trust Region Policy Optimization, Schulman et al, 2015

Searching for the Next Parameter

- Use a linear approximation to the objective function and a quadratic approximation to the constraint
- Constrained optimization problem
- Use conjugate gradient descent

Table of Contents

Practical Algorithm: TRPO

1: **for** iteration=$1, 2, \ldots$ **do**
2: Run policy for T timesteps or N trajectories
3: Estimate advantage function at all timesteps
4: Compute policy gradient g
5: Use CG (with Hessian-vector products) to compute $F^{-1}g$ where F is the Fisher information matrix
6: Do line search on surrogate loss and KL constraint
7: **end for**

Practical Algorithm: TRPO

Applied to

- Locomotion controllers in 2D

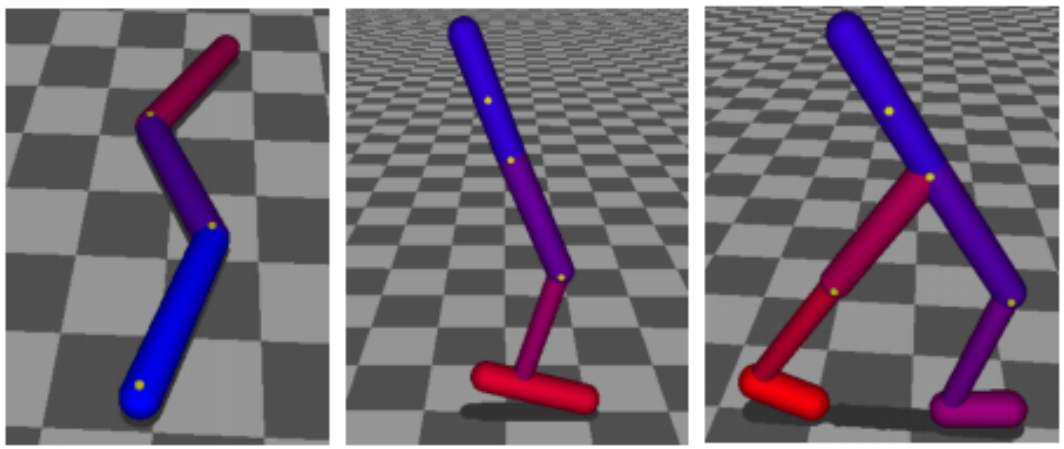


Figure: Trust Region Policy Optimization, Schulman et al, 2015

- Atari games with pixel input

TRPO Results

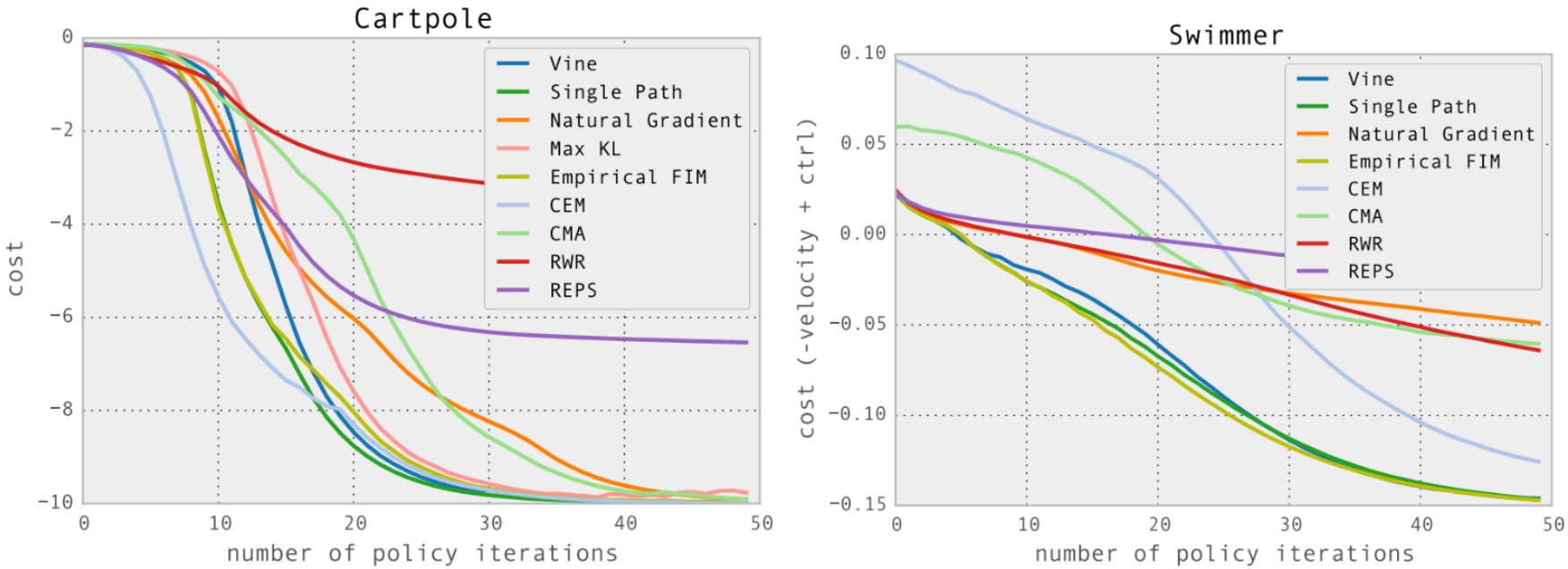


Figure: Trust Region Policy Optimization, Schulman et al, 2015

TRPO Results

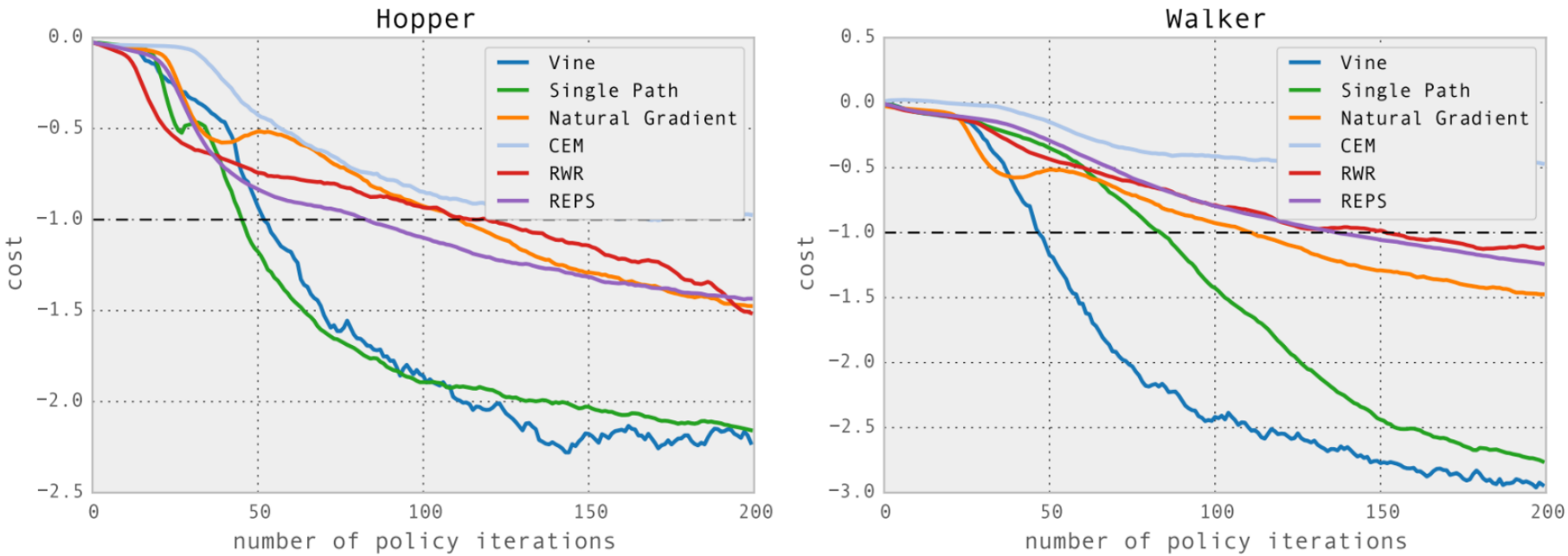


Figure: Trust Region Policy Optimization, Schulman et al, 2015

TRPO Summary

- Policy gradient approach
- Uses surrogate optimization function
- Automatically constrains the weight update to a trusted region, to approximate where the first order approximation is valid
- Empirically consistently does well
- Very influential: +350 citations since introduced a few years ago

Common Template of Policy Gradient Algorithms

1: **for** iteration=$1, 2, \ldots$ **do**
2: Run policy for T timesteps or N trajectories
3: At each timestep in each trajectory, compute target $Q^{\pi}(s_t, a_t)$, and baseline $b(s_t)$
4: Compute estimated policy gradient $\hat{g}$
5: Update the policy using $\hat{g}$, potentially constrained to a local region
6: **end for**

Policy Gradient Summary

- Extremely popular and useful set of approaches
- Can incorporate prior knowledge by choosing the policy parameterization
- You should be very familiar with REINFORCE and the policy gradient template on the prior slide
- Understand where different estimators can be slotted in (and implications for bias/variance)
- Don't have to be able to derive or remember the specific formulas in TRPO for approximating the objectives and constraints
- Will have the opportunity to practice with these ideas in homework 3

Class Structure

- Last time: Policy Search
- This time: Policy Search
- **Next time: Exploration**

Practical Implementation with Auto differentiation

- Usual formula $\sum_t \nabla_\theta \log \pi(a_t|s_t;\theta)\hat{A}_t$ is inefficient–want to batch data
- Define "surrogate" function using data from current batch

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

- Then policy gradient estimator $\hat{g} = \nabla_\theta L(\theta)$
- Can also include value function fit error

$$L(\theta) = \sum_t \left(\log \pi(a_t|s_t;\theta)\hat{A}_t - ||V(s_t) - \hat{G}_t||^2\right)$$