Lecture 8: Policy Gradient I [1]

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2021

- 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,

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

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

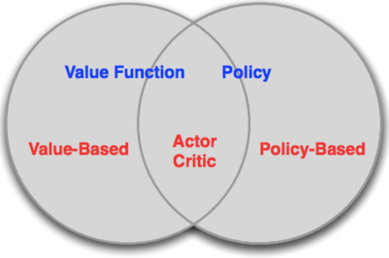
- A policy was generated directly from the value function
  - e.g. using ϵ-greedy
- In this lecture we will directly parametrize the policy, and will typically use θ to show parameterization:

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

- 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

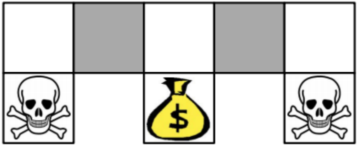
Example: Rock-Paper-Scissors

- 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)

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

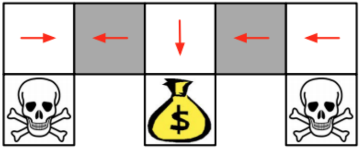
- Compare value-based RL, using an approximate value function

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

- To policy-based RL, using a parametrized policy

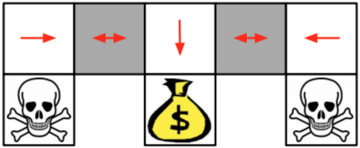
$$\pi_\theta(s, a) = g(\phi(s, a); \theta)$$

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)

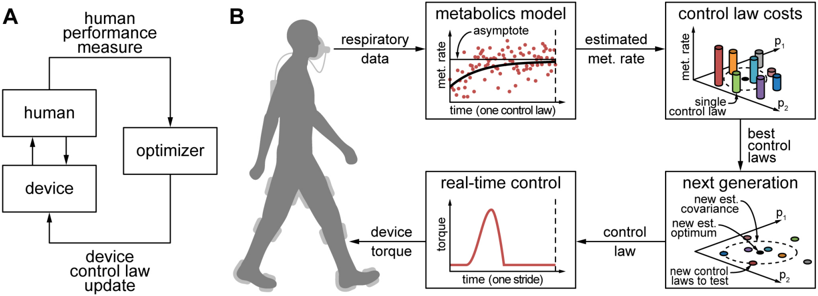


Figure: Zhang et al. Science 2017

- 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

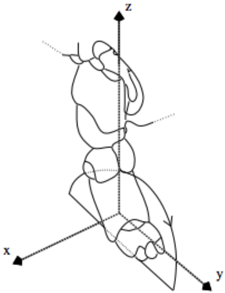
- To evaluate policy gradient of $\pi_\theta(s, a)$
- For each dimension $k \in [1, n]$
  - Estimate kth partial derivative of objective function w.r.t. θ
  - By perturbing θ by small amount ϵ in kth dimension

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

  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

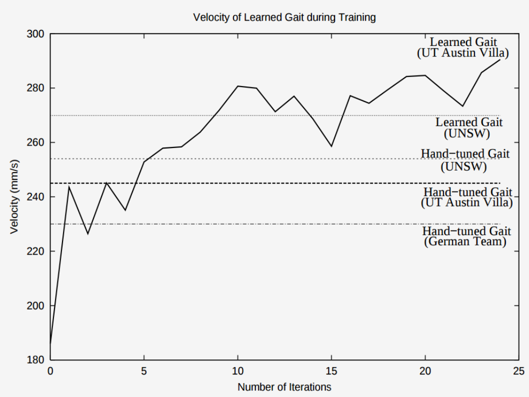
AIBO Policy Parameterization

- 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:

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

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:

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

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

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:

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

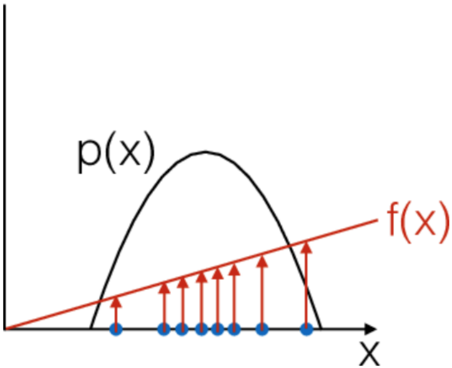
- Do not need to know dynamics model

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*

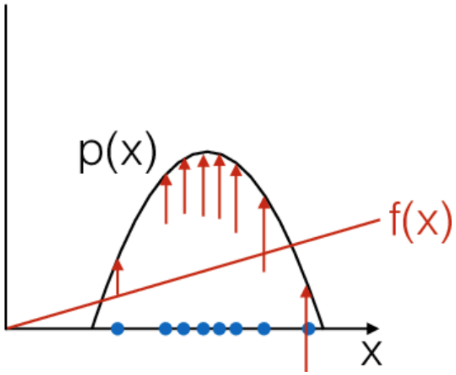
Score Function Gradient Estimator: Intuition

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



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

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

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.