UNIVERSITY OF TORONTO

CSC415: Introduction to Reinforcement Learning

Lecture 5: Policy Gradient

Dr. Amey Pore

Winter 2026

February 04, 2026

Additional Reading: Sutton and Barto: Chp 13. Structure adapted from David Silver's and Emma Brunskill's course on Introduction to RL.

L5N1 Refresh Your Knowledge. Comparing Policy Performance

Consider doing experience replay over a finite, but extremely large, set of (s,a,r,s') tuples). Q-learning is initialized to 0 everywhere and all rewards are positive. Select all that are true

1. Assume all tuples were gathered from a fixed, deterministic policy π. Then in the tabular setting, if each tuple is sampled at random and used to do a Q-learning update, and this is repeated an infinite number of times, then there exists a learning rate schedule so that the resulting estimate will converge to the true Q^{π}.
2. In situation (1) (the first option above) the resulting Q estimate will be identical to if one computed an estimated dynamics model and reward model using maximum likelihood evaluation from the tuples, and performed policy evaluation using the estimated dynamics and reward models.
3. If one uses DQN to populate the experience replay set of tuples, then doing experience replay with DQN is always guaranteed to converge to the optimal Q function.

Class Structure

- Last time: Learning to Control using value function parametrization
- **This time: Direct Policy parameterization**
- Next time: Advanced Policy parametrization

Outline

Policy-Based Reinforcement Learning

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

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

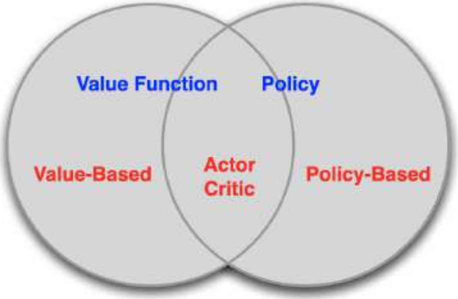
- A policy was generated directly from the value function
    - e.g. using ϵ-greedy
- In this lecture we will directly parametrize the policy, and will typically use θ to show parameterization:

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

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



Advantages of Policy-Based RL

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

End-to-End Training of Deep Visuomotor Policies: `https://www.youtube.com/watch?v=Q4bMcUk6pcw`

Types of Policies to Search Over

- So far have focused on deterministic policies or ϵ-greedy policies
- 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
- Is deterministic policy optimal? Why or why not?
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (i.e. Nash equilibrium)

Example: Aliased Gridworld (1)

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

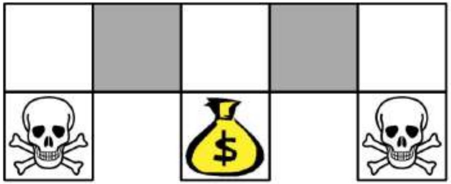
$$\phi(s, a) = \mathbf{1}(\text{wall to N}, a = \text{move E})$$

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

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

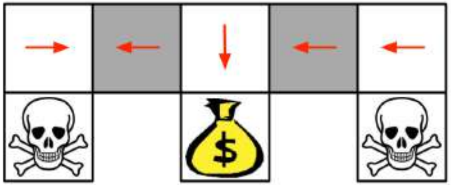
- To policy-based RL, using a parametrized policy

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



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

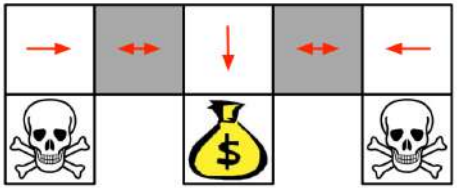


Example: Aliased Gridworld (3)

- 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

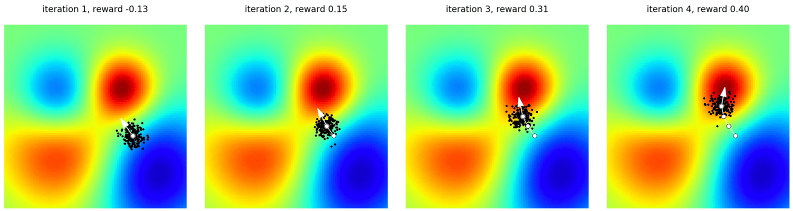
Policy optimization

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

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)

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: Genetic Algorithms

- Often a great simple baseline to try
- Benefits
  - Can work with any policy parameterizations, including non-differentiable
  - Frequently very easy to parallelize
- Limitations
  - Often less sample efficient because it ignores temporal structure

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

Outline

Policy Gradient

- Define $V^{\pi_\theta} = V(s_0, \theta)$ 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

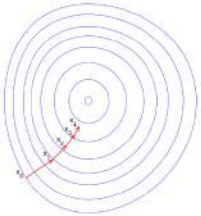
- Define $V^{\pi_\theta} = V(s_0, \theta)$ to make explicit the dependence of the value on the policy parameters
- Assume episodic MDPs
- Policy gradient algorithms search for a local maximum in $V(s_0, \theta)$ by ascending the gradient of the policy, w.r.t parameters θ

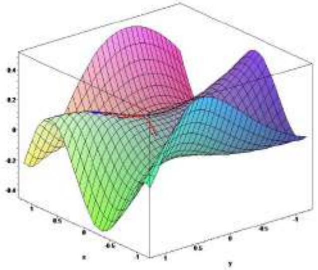
$$\Delta\theta = \alpha \nabla_\theta V(s_0, \theta)$$

- 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





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 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

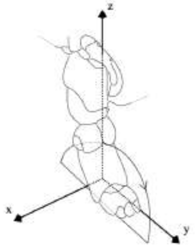


Figure: Early example of policy gradient methods: training a AIBO to have a faster walk. Paper: Kohl and Stone, ICRA 2004

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

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 $V(s_0, \theta) = E_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta, s_0\right]$ 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

- Assume policy π_θ is differentiable whenever it is non-zero and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$
- Recall policy value is $V(s_0, \theta) = E_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta, s_0\right]$ 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 $\tau = (s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ 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

Likelihood Ratio Policies

- Denote a state-action trajectory as $\tau = (s_0, a_0, r_0, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$
- 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

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

  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 using 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 using 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 using m sample paths under policy π_θ:

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

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

Decomposing the Trajectories Into States and Actions

- Approximate using m sample paths under policy π_θ:

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

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \log \Bigg[\underbrace{\mu(s_0)}_{\text{Initial state distrib.}} \prod_{t=0}^{T-1} \underbrace{\pi_\theta(a_t|s_t)}_{\text{policy}} \underbrace{P(s_{t+1}|s_t, a_t)}_{\text{dynamics model}} \Bigg] \\
&= \nabla_\theta \left[\log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t, a_t) \right] \\
&= \sum_{t=0}^{T-1} \underbrace{\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\text{score function}}
\end{aligned}$$

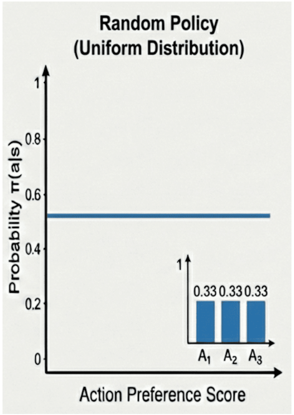
Score Function

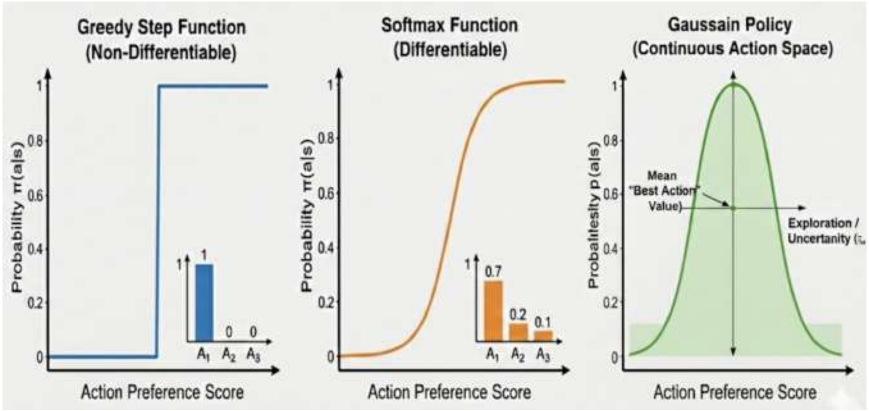
- A score function is the derivative of the log of a parameterized probability / likelihood
- Example: let $\pi(s;\theta)$ be the probability of state s under parameter θ
- Then the score function would be

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

- For many policy classes, it is not hard to compute the score function

Policies


Random Policy
(Uniform Distribution)
Probability π(a|s)
1
0.8
0.6
0.4
0.2
0
1
0.33 0.33 0.33
A_1 A_2 A_3
Action Preference Score


Greedy Step Function
(Non-Differentiable)
Probability π(a|s)
1
0.8
0.6
0.4
0.2
0
1
1
0 0
A_1 A_2 A_3
Action Preference Score
Softmax Function
(Differentiable)
Probability π(a|s)
1
0.8
0.6
0.4
0.2
0
1
0.7
0.2
0.1
A_1 A_2 A_3
Action Preference Score
Gaussain Policy
(Continuous Action Space)
Probabiltesity p(a|s)
1
0.8
0.6
0.4
0.2
0
Mean
"Best Action"
Value)
Exploration /
Uncertanity
Action Preference Score

Softmax Policy

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

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

- 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: $\pi_\theta(s,a) = e^{\phi(s,a)^T\theta} / \left(\sum_{a'} e^{\phi(s,a')^T\theta}\right)$
- The score function is:

$$\nabla_\theta \log \pi_\theta(s,a) = \nabla_\theta \log \left[\frac{e^{\phi(s,a)^T\theta}}{\sum_{a'} e^{\phi(s,a')^T\theta}}\right] = \nabla_\theta \left[\phi(s,a)^T\theta - \log \sum_{a'} e^{\phi(s,a')^T\theta}\right]$$

$$= \phi(s,a) - \frac{\sum_{a'} \phi(s,a') e^{\phi(s,a')^T\theta}}{\sum_{a'} e^{\phi(s,a')^T\theta}} = \phi(s,a) - E_{\pi_\theta}[\phi(s,\cdot)]$$

Softmax Policy

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

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

- The score function is

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

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
$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$
- Deep neural networks (and other models where can compute the gradient) can also be used to represent the policy

Likelihood Ratio / Score Function Policy Gradient

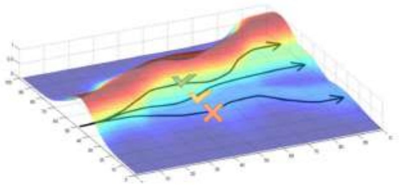
- Putting this together
- Goal is to find the policy parameters θ:

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

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

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

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



- Do not need to know dynamics model

Think Pair-wise: Score functions

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

The likelihood ratio / score function policy gradient (select one):

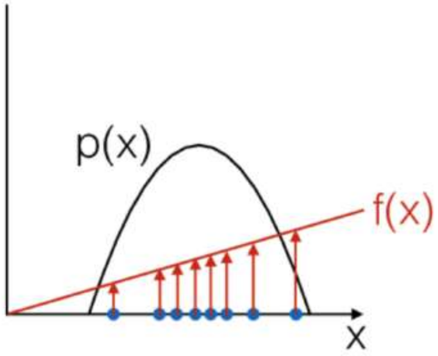
- (a) requires reward functions that are differentiable
- (b) can only be used with Markov decision processes
- (c) Is useful mostly for infinite horizon tasks
- (a) and (b)
- a,b and c
- None of the above
- Not sure

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

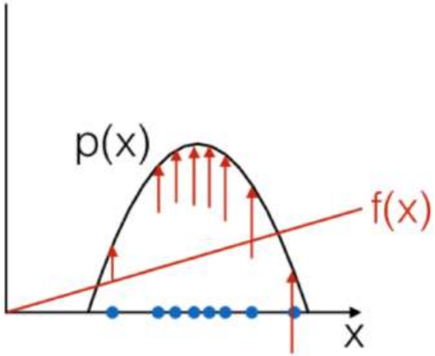
Score Function Gradient Estimator: Intuition

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



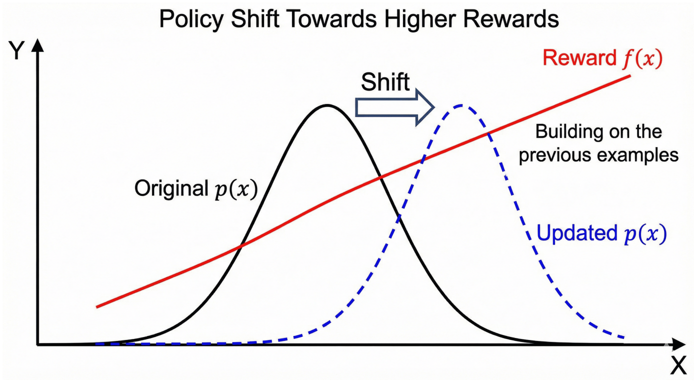
Score Function Gradient Estimator: Intuition

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



Score Function Gradient Estimator: Intuition

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


Policy Shift Towards Higher Rewards
Y
X
Reward $f(x)$
Shift
Building on the previous examples
Original $p(x)$
Updated $p(x)$

Policy Gradient Theorem

- The policy gradient theorem generalizes the likelihood ratio approach
- Replaces reward over the trajectory $R(\tau^{(i)})$ with long-term value $Q^{\pi_\theta}(s, a)$

Theorem

For any differentiable policy $\pi_\theta(a|s)$, the policy gradient is

$$\nabla_\theta V(s_0, \theta) = E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)\right]$$

Outline

Course Logistics

- Time to form teams. We can accomodate at max 15 teams.
- Grades for lab 1 are out on MarkUs.
- Grades for lab 2 will be out this week.
- A1 Recommendations.
    - Start implementing as early as possible, as you will encounter lots of errors.
    - Expectation is that while you read these papers, you understand the problem, and use it with your team in project proposal.

Additional textbooks and resources

1) RL Theory

- Algorithms for Reinforcement Learning (Intermediate)
- Reinforcement Learning: Foundations (Advanced)
- Mathematical Foundations of Reinforcement Learning

2) Deep RL

- Grokking Deep Reinforcement Learning
- Deep Reinforcement Learning Hands-On
- Foundations of Deep Reinforcement Learning: Theory and Practice in Python

Break

Break - 5 minutes

Policy Gradient Theorem

- The policy gradient theorem generalizes the likelihood ratio approach
- Replaces reward over the trajectory $R(\tau^{(i)})$ with long-term value $Q^{\pi_\theta}(s, a)$

Theorem

For any differentiable policy $\pi_\theta(a|s)$, the policy gradient is

$$\nabla_\theta V(s_0, \theta) = E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)\right]$$

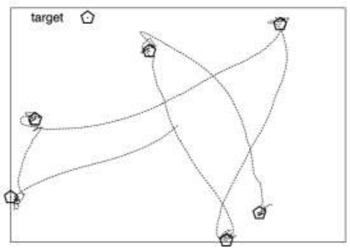
Monte-Carlo Policy Gradient (REINFORCE)

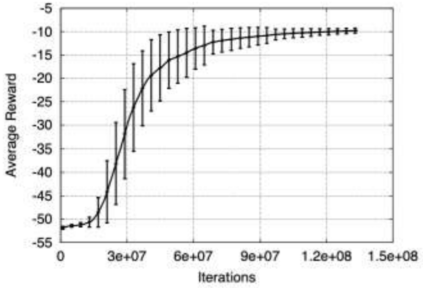
- Using policy gradient theorem
- Using return G_t as an unbiased estimate of $Q^{\pi_\theta}(s_t, a_t)$
- Stochastic gradient ascent update:

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

```
1: Initialize policy parameters $\theta$ arbitrarily
2: for each episode $\{s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
3:     for $t = 1$ to $T - 1$ do
4:         $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t|a_t) G_t$
5:     end for
6: end for
7: return $\theta$
```

Puck World Example





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

Outline

Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function,

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

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

$$\nabla_\theta V(s_0,\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_w(s,a)]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(a|s) Q_w(s,a)$$

Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- How good is policy π_θ for current parameters θ?
- This problem was explored in previous two lectures, e.g.
    - Monte-Carlo policy evaluation
    - Temporal-Difference learning
- Could also use e.g. least-squares policy evaluation

Action-Value Actor-Critic

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

```
1: function Q-Actor Critic
2: Initialize $s, \theta$
3: Sample action $a \sim \pi_\theta(\cdot|s)$
4: for each step do
5:     Sample reward $r = R(s,a)$; sample transition $s' \sim P(\cdot|s,a)$
6:     Sample action $a' \sim \pi_\theta(\cdot|s')$
7:     $\delta \leftarrow r + \gamma Q_w(s',a') - Q_w(s,a)$
8:     $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) Q_w(s,a)$
9:     $w \leftarrow w + \beta \delta \phi(s,a)$
10:    $a \leftarrow a', s \leftarrow s'$
11: end for
12: end function
```

Bias in Actor-Critic Algorithms

- Approximating the policy gradient introduces bias
- A biased policy gradient may not find the right solution
    - e.g. if $Q_w(s, a)$ uses aliased features, can we solve gridworld example?
- Luckily, if we choose value function approximation carefully
- Then we can avoid introducing any bias
- i.e. We can still follow the *exact* policy gradient

Reducing Variance Using a Baseline

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

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

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

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

- 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

$$
\begin{aligned}
&E_\tau[\nabla_\theta \log \pi(a_t|s_t;\theta) b(s_t)] \\
&= E_{s_{0:t},a_{0:(t-1)}}\left[E_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta) b(s_t)]\right] && \text{(break up expectation)} \\
&= E_{s_{0:t},a_{0:(t-1)}}[b(s_t) E_{a_t}[\nabla_\theta \log \pi(a_t|s_t;\theta)]] && \text{(pull out baseline)} \\
&= E_{s_{0:t},a_{0:(t-1)}}\left[b(s_t) \sum_a \pi_\theta(a_t|s_t) \frac{\nabla_\theta \pi(a_t|s_t;\theta)}{\pi_\theta(a_t|s_t)}\right] && \text{(likelihood ratio)} \\
&= E_{s_{0:t},a_{0:(t-1)}}\left[b(s_t) \sum_a \nabla_\theta \pi(a_t|s_t;\theta)\right] = E_{s_{0:t},a_{0:(t-1)}}[b(s_t)\nabla_\theta 1] \\
&= E_{s_{0:t},a_{0:(t-1)}}[b(s_t)\cdot 0] = 0
\end{aligned}
$$

“Vanilla” Policy Gradient Algorithm

```
1: Initialize policy parameter $\theta$, baseline $b$
2: **for** iteration $= 1, 2, \cdots$ **do**
3:     Collect a set of trajectories by executing the current policy
4:     At each timestep $t$ in each trajectory $\tau^i$, compute:
5:         Return $G_t^i = \sum_{t'=t}^{T-1} r_{t'}^i$
6:         Advantage estimate $\hat{A}_t^i = G_t^i - b(s_t)$
7:     Re-fit the baseline, by minimizing $\sum_i \sum_t \|b(s_t) - G_t^i\|^2$
8:     Update the policy, using a policy gradient estimate $\hat{g}$:
9:         $\hat{g} = \sum_i \sum_t \nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_t^i$
10:        (Plug $\hat{g}$ into SGD or ADAM)
11: **end for**
```

- Other choices for Baseline?

Choosing the Baseline: Value Functions

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

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

- State-value function can serve as a great baseline

$$V^{\pi}(s) = E_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 \cdots | s_0 = s\right]$$

$$= E_{a \sim \pi}[Q^{\pi}(s,a)]$$

Policy Gradient Formulas with Value Functions

- Recall:

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

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

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

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

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

Choosing the Target

- 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 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

Estimating the Advantage Function (1)

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

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

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

Estimating the Advantage Function (2)

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

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

- is an unbiased estimate of the advantage function

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

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

$$\nabla_\theta V(s_0,\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)\delta^{\pi_\theta}]$$

- In practice we can use an approximate TD error

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

- This approach only requires one set of critic parameters v

Choosing the Target: N-step estimators

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

- Critic can select any blend between TD and MC estimators:

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

- If subtract baselines from the above, get advantage estimators:

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

Think Pair wise: 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

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

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

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$
\begin{aligned}
\nabla_\theta V(s_0, \theta) &= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) G_t] && \text{REINFORCE} \\
&= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^w(s,a)] && \text{Q Actor-Critic} \\
&= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A^w(s,a)] && \text{Advantage Actor-Critic} \\
&= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \delta] && \text{TD Actor-Critic}
\end{aligned}
$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$ or $V^\pi(s)$

Class Structure

- Last time: Deep Model-free Value Based RL
- This time: Policy Gradients
- Next time: Policy Gradients Cont.