Lecture 8: Policy Gradient I

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2020

- Additional reading: Sutton and Barto 2018 Chp. 13

---

1With many slides from or derived from David Silver and John Schulman and Pieter Abbeel
Behavior cloning (select all)

1. Involves using supervised learning to predict actions given states using expert demonstrations
2. If the expert demonstrates an action in all states in a tabular domain, behavior cloning will find an optimal expert policy
3. If the expert demonstrates an action in all states visited under the expert’s policy, behavior cloning will find an optimal expert policy
4. DAGGER improves behavior cloning and only requires the expert to demonstrate successful trajectories
5. Not sure
Last Time: We want RL Algorithms that Perform

- Optimization
- Delayed consequences
- Exploration
- Generalization
- And do it statistically and computationally efficiently
Can use structure and additional knowledge to help constrain and speed reinforcement learning
Class Structure

- Last time: Imitation Learning in Large State Spaces
- **This time:** Policy Search
- Next time: Policy Search Cont.
<table>
<thead>
<tr>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

1. Introduction

2. Policy Gradient

3. Score Function and Policy Gradient Theorem

4. Policy Gradient Algorithms and Reducing Variance
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 \( \epsilon \)-greedy

In this lecture we will directly parametrize the policy, and will typically use \( \theta \) to show parameterization:

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

Goal is to find a policy \( \pi \) with the highest value function \( V^\pi \)

We will focus again on model-free reinforcement learning
Value-Based and Policy-Based RL

- **Value Based**
  - Learnt Value Function
  - Implicit policy (e.g. $\epsilon$-greedy)

- **Policy Based**
  - No Value Function
  - Learnt Policy

- **Actor-Critic**
  - Learnt Value Function
  - Learnt 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
  \[
  \pi_\theta(s, a) = g(\phi(s, a); \theta)
  \]
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 $\epsilon$-greedy

So it will traverse the corridor for a long time
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
Goal: given a policy $\pi_\theta(s, a)$ with parameters $\theta$, find best $\theta$

But how do we measure the quality for a policy $\pi_\theta$?

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 based reinforcement learning is an optimization problem

Find policy parameters $\theta$ that maximize $V(s_0, \theta)$
Policy optimization

- Policy based reinforcement learning is an **optimization** problem.
- Find policy parameters $\theta$ 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)

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

**Figure:** Zhang et al. Science 2017
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
Policy based reinforcement learning is an optimization problem

Find policy parameters $\theta$ 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
| 1 | Introduction               |
| 2 | Policy Gradient            |
| 3 | Score Function and Policy Gradient Theorem |
| 4 | Policy Gradient Algorithms and Reducing Variance |
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)
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 $\theta$

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

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

$$\nabla_{\theta} V(s_0, \theta) = \left( \frac{\partial V(s_0, \theta)}{\partial \theta_1} \right)$$

$$\vdots$$

$$\left( \frac{\partial V(s_0, \theta)}{\partial \theta_n} \right)$$

and $\alpha$ 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 $k$th partial derivative of objective function w.r.t. $\theta$
  - By perturbing $\theta$ by small amount $\epsilon$ in $k$th 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 $k$th component, 0 elsewhere.
To evaluate policy gradient of $\pi_\theta(s, a)$

For each dimension $k \in [1, n]$

- Estimate $k$th partial derivative of objective function w.r.t. $\theta$
- By perturbing $\theta$ by small amount $\epsilon$ in $k$th 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 $k$th component, 0 elsewhere.

Uses $n$ evaluations to compute policy gradient in $n$ dimensions

Simple, noisy, inefficient - but sometimes effective

Works for arbitrary policies, even if policy is not differentiable
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

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 \((\epsilon, 0, \text{ or } -\epsilon)\)
"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
- Used $\eta = 2$ (learning rate for their finite difference approach)
Authors discuss that performance is likely impacted by: initial starting policy parameters, $\epsilon$ (how much policies are perturbed), $\eta$ (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
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
Computing the gradient analytically

- We now compute the policy gradient *analytically*
- Assume policy $\pi_\theta$ is differentiable whenever it is non-zero
- and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Focusing for now on $V(s_0, \theta) = \sum_\tau P(\tau; \theta)R(\tau)$
Many choices of differentiable policy classes including:
- Softmax
- Gaussian
- Neural networks
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

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

Connection to Q function?
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 $\sigma^2$, 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 $\pi_\theta$ 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 and actions visited by $\pi_\theta$
- 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
  - $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ to be the sum of rewards for a trajectory $\tau$
- 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 
  \( \tau = (s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T) \)
- Use \( R(\tau) = \sum_{t=0}^{T} R(s_t, a_t) \) to be the sum of rewards for a trajectory \( \tau \)
- 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 \( \theta \):
  \[
  \arg \max_{\theta} V(\theta) = \arg \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)
  \]
Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters $\theta$:
  \[
  \arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau; \theta) R(\tau)
  \]

- Take the gradient with respect to $\theta$:
  \[
  \nabla_\theta V(\theta) = \nabla_\theta \sum_\tau P(\tau; \theta) R(\tau)
  \]
Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters $\theta$:
  \[
  \arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau; \theta) R(\tau)
  \]

- Take the gradient with respect to $\theta$:
  \[
  \nabla_\theta V(\theta) = \nabla_\theta \sum_\tau P(\tau; \theta) R(\tau)
  \]
  \[
  = \sum_\tau \nabla_\theta P(\tau; \theta) R(\tau)
  \]
  \[
  = \sum_\tau \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_\theta P(\tau; \theta) R(\tau)
  \]
  \[
  = \sum_\tau P(\tau; \theta) R(\tau) \left( \nabla_\theta \log P(\tau; \theta) \frac{\nabla_\theta P(\tau; \theta)}{P(\tau; \theta)} \right)
  \]
  \[
  = \sum_\tau P(\tau; \theta) R(\tau) \nabla_\theta \log P(\tau; \theta)
  \]
Goal is to find the policy parameters $\theta$:

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

Take the gradient with respect to $\theta$:

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

Approximate with empirical estimate for $m$ sample trajectories under policy $\pi_{\theta}$:

$$\nabla_{\theta} V(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)})\nabla_{\theta} \log P(\tau^{(i)}; \theta)$$
Decomposing the Trajectories Into States and Actions

- Approximate with empirical estimate for $m$ sample paths under policy $\pi_\theta$:

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

$$\nabla_\theta \log P(\tau^{(i)}; \theta) =$$
Decomposing the Trajectories Into States and Actions

- Approximate with empirical estimate for \( m \) sample paths under policy \( \pi_\theta \):

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

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

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

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

no dynamics model required!
Define score function as $\nabla_\theta \log \pi_\theta(s, a)$
Putting this together

Goal is to find the policy parameters $\theta$:

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

Approximate with empirical estimate for $m$ sample paths under policy $\pi_\theta$ using score function:

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

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

Do not need to know dynamics model
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**
\[ \hat{g}_i = f(x_i) \nabla_\theta \log p(x_i | \theta) \]
\[ \hat{g}_i = f(x_i) \nabla_\theta \log p(x_i | \theta) \]
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
\[ \nabla_{\theta} V(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_{t}^{(i)}|s_{t}^{(i)}) \]

- 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
\[
V(\theta) = \nabla_\theta \mathbb{E}[R] = \mathbb{E} \left[ \sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]
\]
\[
= \mathbb{E} \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) \sum_{t'=t}^{T-1} r_{t'} \right]
\]
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 \( \theta \) 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 \( \theta \)
\[ \nabla_{\theta} V(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a^{(i)}_{t} | s^{(i)}_{t}) \]

- 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.