Lecture 9: Policy Gradient II [1]

Emma Brunskill

CS234 Reinforcement Learning.

Winter 2019

- Additional reading: Sutton and Barto 2018 Chp. 13

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

Class Feedback

- Thanks to those that participated!
- Of 79 responses, 26.6% thought too fast, 65.8% just right, 7.6% too slow
- 49% wanted us to keep sessions on Thursday and Friday, 51% wanted to switch to office hours

Class Feedback

- Thanks to those that participated!
- Of 70 responses, 54% thought too fast, 43% just right
- Good: Derivations, homeworks

Class Feedback

- Thanks to those that participated!
- Of 70 responses, 54% thought too fast, 43% just right
- Good: Derivations, homeworks
- Do more of: big picture explaining, speaking loudly throughout, more examples

Class Structure

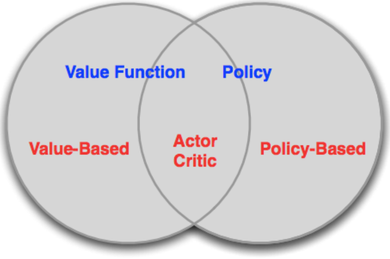
- Last time: Policy Search
- **This time: Policy Search**
- Next time: Midterm review

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 and POMDPs

Disadvantages:

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

Recall: Policy Gradient

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

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

ultimately, the value function depends on the policy and the policy depends on the parameters

- 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 parameter

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

In DQN, the performance is jagged and goes up and down a lot
Using policy gradients, you can have a monotonically improvement

V_pi1 < V_pi2 < ... meaning that we want the second iteration of the policy has higher value that the first

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

Monte Carlo returns. Run policy for entire trajectory and sum

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

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

- 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 $R(\tau^{(i)})$ as targets

the reward you get at time t is not influence by decisions you make at later timesteps

Policy Gradient: Introduce Baseline

Gt : the reward we get from this timestep

- 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

goal is to show that this is equal to zero

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

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

if the baseline were a function of state as well as ACTION, it would not be unbiased!

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

$$
\begin{aligned}
&\mathbb{E}_\tau[\nabla_\theta \log \pi(a_t|s_t;\theta) b(s_t)] \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta) b(s_t)]\right] \text{ (break up expectation)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta)]\right] \text{ (pull baseline term out)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[b(s_t)\mathbb{E}_{a_t}[\nabla_\theta \log \pi(a_t|s_t;\theta)]] \text{ (remove irrelevant variables)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\sum_a \pi_\theta(a_t|s_t)\frac{\nabla_\theta \pi(a_t|s_t;\theta)}{\pi_\theta(a_t|s_t)}\right] \text{ (likelihood ratio)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\sum_a \nabla_\theta \pi(a_t|s_t;\theta)\right] \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\nabla_\theta \sum_a \pi(a_t|s_t;\theta)\right] \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[b(s_t)\nabla_\theta 1] \\
&= \mathbb{E}_{s_0:t,a_0:(t-1)}[b(s_t)\cdot 0] = 0
\end{aligned}
$$

"Vanilla" Policy Gradient Algorithm

This is a baseline which is like a fixed function that can be parameterized! It could be a NN or a table lookup, or even something really simple. The baseline is an estimate of the average of the G's for that timestep t and trajectory i.

It's a function where you input the state at timestep t and trajectory i, and you get out a scalar

so we set the baseline params such that they minimize this

return for that timestep until the end

```
Initialize policy parameter $\theta$, baseline $b$
for iteration$=1, 2, \cdots$ do    $\pi^i, \pi^{i+1} \dots$
  Collect a set of trajectories by executing the current policy
  At each timestep $t$ in each trajectory $\tau^i$, 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
```

she forgot to put an 'i' indicating that it was for iteration i

this is an estimate for V_pi_i

this is just supervised learning. This is the loss!
So here we are fitting our baseline model to all of the trajectories in our current iteration i.

Just to clarify:
- We run multiple iterations, where each iteration has a particular policy pi. Within each iteration, we run multiple trajectories. For a given trajectory, we sum up the total reward, and also calculate our advantage for the entire trajectory
- Then we recalculate our baseline using the average rewards for all the trajectories in the current iteration

This is the basic template for all policy methods. We can chose different ways to calculate return and the advantage (reducing variance and getting better estimates of our gradient with less data), and step size (for monotimic improvement)

Practical Implementation with Autodiff

- 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

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

Other Choices for Baseline?

How should we choose the baseline?
- We could use V_pi_i : we'd have to compute this/estimate it using Monte Carlo or TD methods
- We can choose between doing Monte Carlo or TD methods for calculating 1) Return, and 2) Baseline

```
Initialize policy parameter $\theta$, 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 $\tau^i$, 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

Note: Pretty much, by selecting how we calculate our target and baseline, we are changing how we calculate gradient, and thus how we change our policy

Recall, before we had this function. This was very noisy. Now we'll talk about other things we can substitute in for R(tao) as our target. We could use TD or MC methods. If we try to use a Value function, we call this a Critic

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

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

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

- State-value function can serve as a great baseline

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

TD(0) higher bias, lower variance

MC no bias, high variance

...so there are tradeoffs

You don't have to use either TD or MC, you could use a combination.
TD(0) -> R = r_t + gamma*V(s)_t+1
TD(inf) = MC -> R = r_t + gamma* r_t+1 + gamma^2 * r_t+2 + ... + gamma^T * r_T
We could instead do something in between, like TD(2) or TD(50).

- Advantage function: Combining Q with baseline V

How often do we update the Critic? Well it depends on the application, but there's no reason it needs to be updated on the same schedule as the Policy. Doing it asynchronously often makes sense

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

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

Before our advantage function was a function of the Monte Carlo returns for that episode. Now we have a different advantage function using a critic.

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 targets**

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
- 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

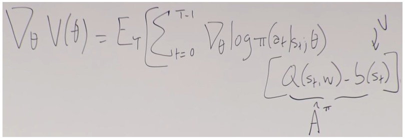
- Recall:

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

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

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

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



$$\nabla_\theta V(\theta) = E_\tau\Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t;\theta)\, \underbrace{[Q(s_t,w) - \overset{V}{b}(s_t)]}_{\hat{A}^\pi}$$

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.

boostrap

more bias,
less variance

$$\hat{R}_t^{(1)} = r_t + \gamma V(s_{t+1}) \quad \text{TD(0)}$$

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

low bias, more
variance

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

If subtract baselines from the above, get advantage estimators

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

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

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

"Vanilla" Policy Gradient Algorithm

```
Initialize policy parameter $\theta$, 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 $\tau^i$, compute
    *Target* $\hat{R}_t^i$
    *Advantage estimate* $\hat{A}_t^i = G_t^i - b(s_t)$.
  Re-fit the baseline, by minimizing $\sum_i \sum_t ||b(s_t) - \hat{R}_t^i||^2$,
  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**
```

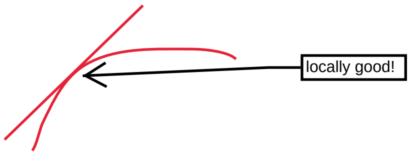
Table of Contents

Once we've gotten our gradient (regardless of how we decided to calculate it), we need to figure out how far to update our policy in the gradient's direction. In other words, how big should our step size be.

It's very important in RL to think about step size because it determines pi, and therefore the data we collect! So if we take too big of a step size, we might change our policy for the worse. This means that we won't get useful data. This ends up spiraling an bit into a bad local optima. So it matters even more than in supervised learning.

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-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

We don't know anything about the value of our new Policy once we take that step. We would like to figure out what the value of the new policy will be BEFORE we execute it. We do this by trying to relate it to the value of our previous policy plus some sort of "distance" between the new policy and old policy, ideally using something we can evaluate using our current samples

- 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 $s_0 \sim \mu(s_0)$, $a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
- 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!)

we have access to trajectories from pi_theta_i,
we try to figure out what pi_theta_i+1 should be
we want to predict the Value V_pi_theta_i+1

but this is fundamentally off-policy because we have data from our old policy and we want to figure out what our new policy should be

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

Objective Function

this allows us to compare Qpi(s, pi-tilde(s)) - Qpi(s, pi(s))

So we can compare the Advantage (i.e. relative performance) of taking actions using the new policy, compared to taking actions using the old policy

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

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

Value of NEW parameterized policy

Value of OLD parameterized policy

- 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

we went from time averaging (in the equation above) to state averaging

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

- where $\mu_{\tilde{\pi}}(s)$ is defined as the discounted weighted frequency of state s under policy $\tilde{\pi}$ (similar to in Imitation Learning lecture)

the distribution over the state and actions we get if we run our NEW policy - we don't know this

the sum of the Advantage of taking the NEW policy over the OLD policy

all of the actions we might take given our target policy, weighted by the relative advantage of each of those

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

Objective Function

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

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

if we had this we could compare the value of the new policy to the value old policy

- 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 $\tilde{\pi}$ (similar to in Imitation Learning lecture)
- 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

because we don't have samples from pi_tilde. we only have samples from pi

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

Table of Contents

Local approximation

so we're going to make up a new objective function since we can't compute mu_pi-tilde(s)!

stationary distribution of states from our OLD policy

- 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:

we can evaluate the advantage b/c it's represented only using our current policy pi

If you passed in pi instead, you just get the value of your old policy L_pi(pi) = V(theta)

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

if we're given this policy, we could compute this

- 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: $\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)|_{\theta=\theta_0} = \nabla_\theta V(\theta)|_{\theta=\theta_0}$

we could average the values from the all the trajectories for our current episode

we can estimate our stationary distribution using our current data of trajectories. To do this, you could just count how many times you're in each state, then divide by the total number of steps. In high dimensions we will want to use a parameterized function to do this instead of just counting

Conservative Policy Iteration

we can use the equation on the previous slide to do 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

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

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

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

this is L_pi(pi-tilde) from the previous slide

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

This lower bounds ensures that our policy is monotonically improving

Check Your Understanding: Is this Bound Tight if $\pi_{new} = \pi_{old}$?

- 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

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

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

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

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

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

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

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

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

- where $\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}\left[A_\pi(s,a)\right]\right|$
- Can we remove the dependency on the discounted visitation frequencies under the new policy?

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:

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

- 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] $\mathrm{L}_\pi(\tilde{\pi}) = V(\theta) + \sum_s \mu_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$

Guaranteed Improvement[1]

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

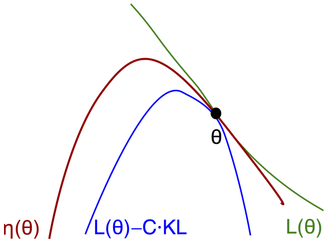


Figure: Source: John Schulman, Deep Reinforcement Learning, 2014

[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

$$\max_\theta \mathbb{E}_{s\sim\mu_{\theta_{old}}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{old}}(s,a)\right]$$

$$\text{subject to } \mathbb{E}_{s\sim\mu_{\theta_{old}}} D_{KL(}\pi_{\theta_{old}}(\cdot|s), \pi_\theta(\cdot|s)) \le \delta$$

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

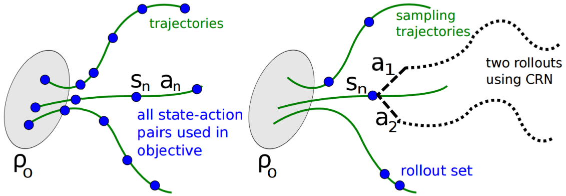


Figure: Trust Region Policy Optimization, Schulman et al, 2015

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

schulman2015trust

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
```

schulman2015trust

Practical Algorithm: TRPO

Applied to

- Locomotion controllers in 2D

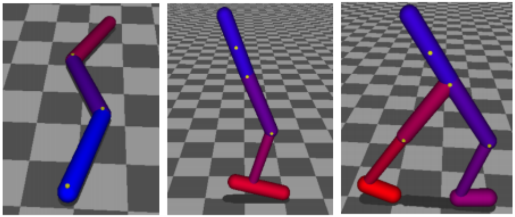
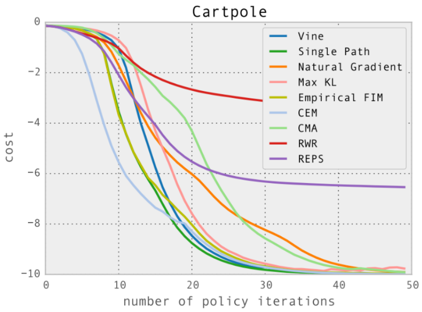


Figure: Trust Region Policy Optimization, Schulman et al, 2015

- Atari games with pixel input

schulman2015trust

TRPO Results



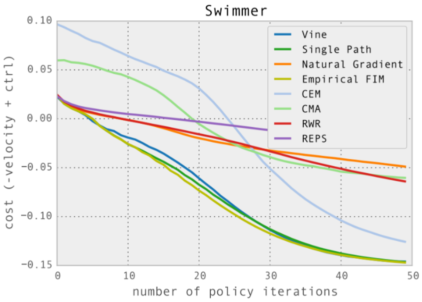


Figure: Trust Region Policy Optimization, Schulman et al, 2015

TRPO Results

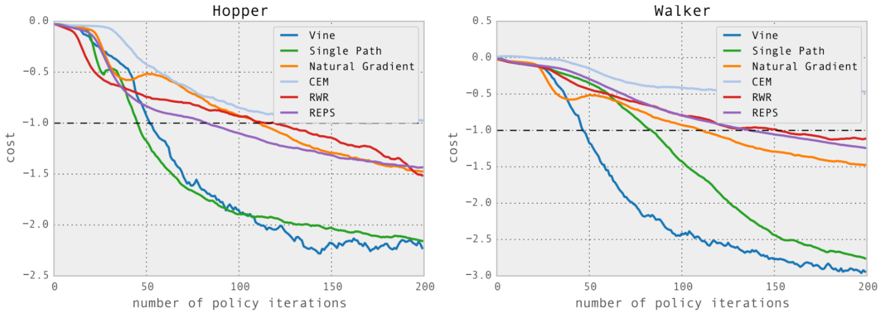


Figure: Trust Region Policy Optimization, Schulman et al, 2015

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=$1, 2, \dots$ 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 input prior knowledge in the form of specifying 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: Midterm review**