Policy Gradients

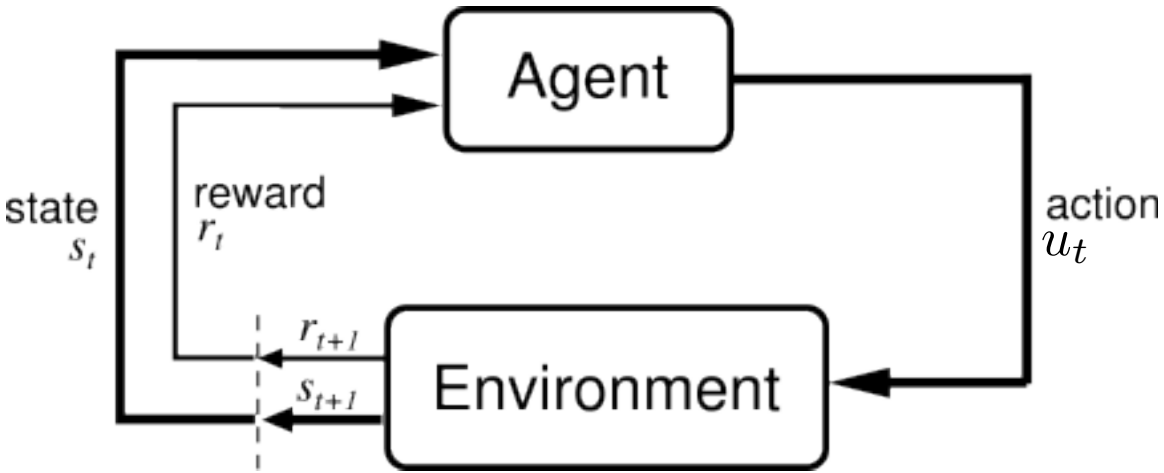
Pieter Abbeel – OpenAI / UC Berkeley / Gradescope

Slides authored with John Schulman and Xi (Peter) Chen

Schedule -- Saturday

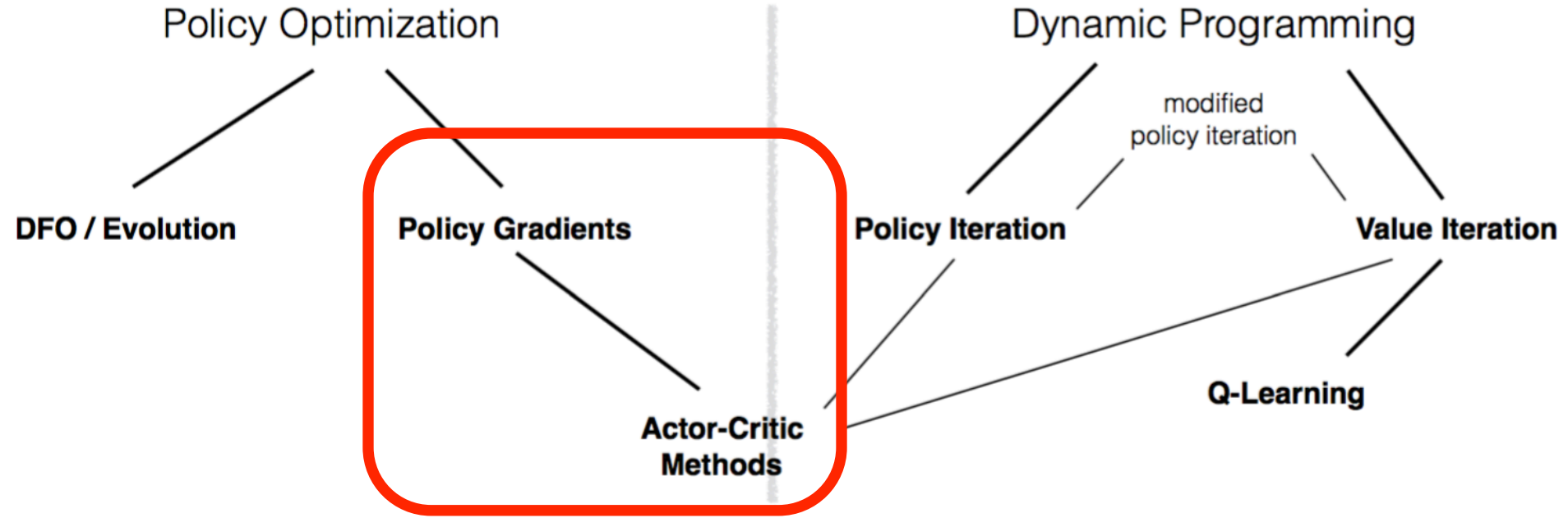
8:30: Breakfast / Check-in
9-10: Core Lecture 1 Intro to MDPs and Exact Solution Methods (Pieter Abbeel)
10-10:10 Brief Sponsor Intros
10:10-11:10 Core Lecture 2 Sample-based Approximations and Fitted Learning (Rocky Duan)
11:10-11:30: Coffee Break
11:30-12:30 Core Lecture 3 DQN + variants (Vlad Mnih)
12:30-1:30 Lunch [catered]
1:30-2:15 Core Lecture 4a Policy Gradients and Actor Critic (Pieter Abbeel)
2:15-3:00 Core Lecture 4b Pong from Pixels (Andrej Karpathy)
3:00-3:30 Core Lecture 5 Natural Policy Gradients, TRPO, and PPO (John Schulman)
3:30-3:50 Coffee Break
3:50-4:30 Core Lecture 6 Nuts and Bolts of Deep RL Experimentation (John Schulman)
4:30-7 Labs 1-2-3
7-8 Frontiers Lecture I: Recent Advances, Frontiers and Future of Deep RL (Vlad Mnih)

Reinforcement Learning



[Figure source: Sutton & Barto, 1998]

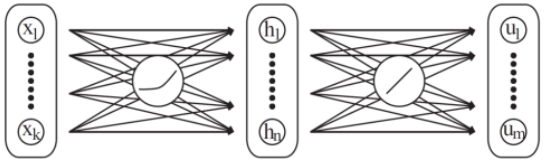
Policy Optimization in the RL Landscape

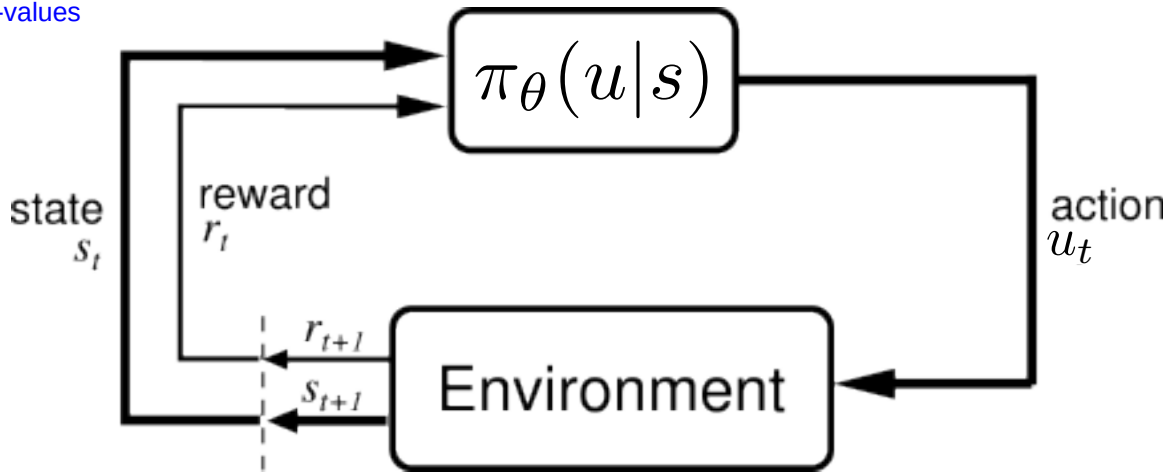


Policy Optimization

Goal: Find policy that maximizes sum of expected rewards. This policy is stochastic for the rest of this lecture. The policy gives us a distribution of actions given a state

Using Q-value, we find the policy indirectly by finding the Q-values



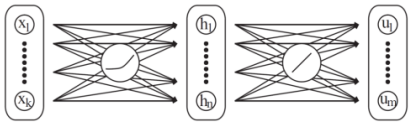


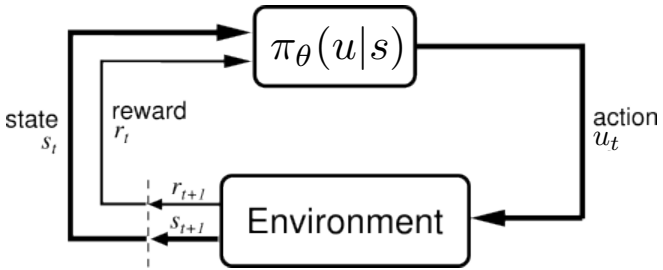
[Figure source: Sutton & Barto, 1998]

Policy Optimization

In q-learning, theta parameterizes a q-function. Now, it parameterizes a policy.

We still want to maximize expected sum of rewards, and for this lecture we wont discount future rewards.





- Consider control policy parameterized by parameter vector θ

$$\max_\theta \quad \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

- Stochastic policy class (smooths out the problem):

This policy class is the set of policies that we'll choose from, and we hope that within that set there will be a good policy that we'll find

$\pi_\theta(u|s)$: probability of action u in state s

Why use a stochastic policy class? Well a few reasons, but for one, it will smooth out the optimization problem. If we have a deterministic policy (and we can think about grid world as the env), changing the action within a given state will have a significant impact on the outcome...It's not very smooth to do this. But once we have a distribution of actions over every state, we can slightly shift the distribution, and that will only slightly shift the expected sum of rewards. By having a stochastic policy, we can use tools like gradient descent on this optimization problem to find the local optimum

[Figure source: Sutton & Barto, 1998]

Why Policy Optimization

- Often π can be simpler than Q or V
  - E.g., robotic grasp

If you wanted to learn a Q- or V- value, you'd have to learn the reward associated with everything in that process. If you just learn a policy, you don't have to learn those details.

- V: doesn't prescribe actions

Once you have V, you still need to figure out which action to take

  - Would need dynamics model (+ compute 1 Bellman back-up)
- Q: need to be able to efficiently solve $\arg\max_u Q_\theta(s, u)$
  - Challenge for continuous / high-dimensional action spaces*

if you have a continuous state, there are an infinite number of actions you can take. So you then need to take the action as an input. So the input would be action and state, and the outcome would be the value of that state-action pair. Then need to solve an optimization problem to see which pair maximizes value.

*some recent work (partially) addressing this:

NAF: Gu, Lillicrap, Sutskever, Levine ICML 2016

Input Convex NNs: Amos, Xu, Kolter arXiv 2016

Deep Energy Q: Haarnoja, Tang, Abbeel, Levine, ICML 2017

- Conceptually:
- Empirically:

Policy Optimization

You need to act with your current policy and improve it gradually.

Optimize what you care about

Policy optimization is generally easier to get to work. So if you had to try one thing, it might be Policy Optimization

More compatible with rich architectures (including recurrence)

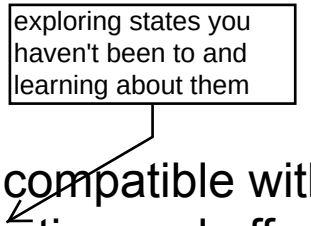
More versatile

More compatible with auxiliary objectives

Dynamic Programming

You can have a separate data-collection policy from the q-learning. The data-collection policy could be exploration oriented, which could be beneficial at times.

Indirect, exploit the problem structure, self-consistency



More compatible with exploration and off-policy learning

More sample-efficient when they work

Likelihood Ratio Policy Gradient

so this is the entire sequence (aka trajectory), not just one transition

H denotes "horizon"

We let τ denote a state-action sequence $s_0, u_0, \ldots, s_H, u_H$. We overload notation: $R(\tau) = \sum_{t=0}^{H} R(s_t, u_t)$.

Given policy pi_theta, its utility is the expected sum of rewards over all trajectories

sum over all possible trajectories, the probability of that trajectory given the policy, times the reward of that trajectory

$$U(\theta) = \mathrm{E}[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

In our new notation, our goal is to find θ:

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

we want to find the parameters of the policy that maximize utility! optimization problem.

Likelihood Ratio Policy Gradient

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

Taking the gradient w.r.t. θ gives

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

[Aleksandrov, Sysoyev, & Shemeneva, 1968]
[Rubinstein, 1969]
[Glynn, 1986]
[Reinforce, Williams 1992]
[GPOMDP, Baxter & Bartlett, 2001]

Likelihood Ratio Policy Gradient

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

Taking the gradient w.r.t. θ gives

$$\begin{aligned} \nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) \\ &= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) \end{aligned}$$

[Aleksandrov, Sysoyev, & Shemeneva, 1968]
[Rubinstein, 1969]
[Glynn, 1986]
[Reinforce, Williams 1992]
[GPOMDP, Baxter & Bartlett, 2001]

Likelihood Ratio Policy Gradient

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

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau)
\end{aligned}$$

we use this trick of multiplying and dividing by the same term

[Aleksandrov, Sysoyev, & Shemeneva, 1968]
[Rubinstein, 1969]
[Glynn, 1986]
[Reinforce, Williams 1992]
[GPOMDP, Baxter & Bartlett, 2001]

Likelihood Ratio Policy Gradient

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

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau)
\end{aligned}$$

[Aleksandrov, Sysoyev, & Shemeneva, 1968]
[Rubinstein, 1969]
[Glynn, 1986]
[Reinforce, Williams 1992]
[GPOMDP, Baxter & Bartlett, 2001]

Likelihood Ratio Policy Gradient

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

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

some math rule allows us to do this.

So why did we do these mathematical changes?

We did this because now we've formulated an expectation. Once we've got an expectation, we can use a sample-based method to approximate that expectation. This is what we did in q-learning too!

[Aleksandrov, Sysoyev, & Shemeneva, 1968]
[Rubinstein, 1969]
[Glynn, 1986]
[Reinforce, Williams 1992]
[GPOMDP, Baxter & Bartlett, 2001]

Likelihood Ratio Policy Gradient

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

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

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

so by taking samples, we can get an estimate for the gradient

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

so we take m sample paths under this policy and find the average of the value on the right of the summation sign

[Aleksandrov, Sysoyev, & Shemeneva, 1968]
[Rubinstein, 1969]
[Glynn, 1986]
[Reinforce, Williams 1992]
[GPOMDP, Baxter & Bartlett, 2001]

Likelihood Ratio Gradient: Validity

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

we don't need to know the reward function. We just need to know what value of R we got from the sample

- Valid even when
  - R is discontinuous and/or unknown
  - Sample space (of paths) is a discrete set

Likelihood Ratio Gradient: Intuition

for the good path, the grad-log probability of that path gets increased
for an okay path where you got a small positive R, the probability still gets increased, but by less than if you got a large positive R
for the really bad path where you got negative R, the grad-log probability of that path gets decreased

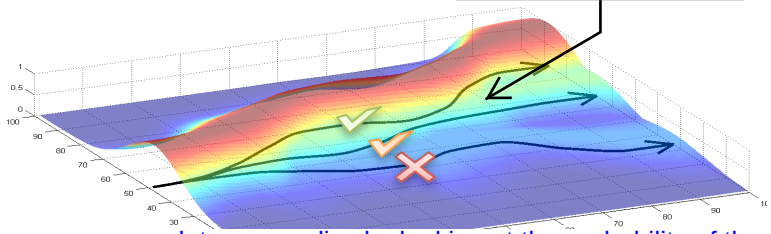
$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

you could update your policy even if you take one sample (m=1). It won't be a very good gradient estimate, but you could do it

- Gradient tries to:
  - Increase probability of paths with positive R
  - Decrease probability of paths with negative R

so if all your paths had a positive R, their probabilities would increase. This is a bit of a problem and makes it tough to learn the best policy. To solve this, we'll introduce a baseline!

so m=3.
we get a good reward, a medium reward, and a bad reward



so you update your policy by looking at the probability of the paths under that policy. Good paths get their probabilities increased and bad paths get their probabilities decreased

! Likelihood ratio changes probabilities of experienced paths, does not try to change the paths (<-> Path Derivative)

Let's Decompose Path into States and Actions

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right]$$

Let's Decompose Path into States and Actions

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right]$$

Let's Decompose Path into States and Actions

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)},u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)},u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right]$$

$$= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$$

Let's Decompose Path into States and Actions

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)},u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)},u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right]$$

$$= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$$

$$= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Likelihood Ratio Gradient Estimate

The following expression provides us with an unbiased estimate of the gradient, and we can compute it without access to a dynamics model:

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

Here:

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Unbiased means:

$$\mathrm{E}[\hat{g}] = \nabla_\theta U(\theta)$$

Derivation from Importance Sampling

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

[Tang&Abbeel, NIPS 2011]

Derivation from Importance Sampling

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

[Tang&Abbeel, NIPS 2011]

Derivation from Importance Sampling

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta \, U(\theta)|_{\theta=\theta_{\text{old}}} = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta \, P(\tau|\theta)|_{\theta_{\text{old}}}}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

[Tang&Abbeel, NIPS 2011]

Derivation from Importance Sampling

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta \, U(\theta)|_{\theta=\theta_{\text{old}}} = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta \, P(\tau|\theta)|_{\theta_{\text{old}}}}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$= \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\nabla_\theta \, \log P(\tau|\theta)|_{\theta_{\text{old}}} \, R(\tau) \right]$$

[Tang&Abbeel, NIPS 2011]

Derivation from Importance Sampling

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta \, U(\theta)|_{\theta=\theta_{\text{old}}} = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta \, P(\tau|\theta)|_{\theta_{\text{old}}}}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$= \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\nabla_\theta \, \log P(\tau|\theta)|_{\theta_{\text{old}}} \, R(\tau) \right]$$

Suggests we can also look at more than just gradient!
E.g., can use importance sampled objective as “surrogate loss” (locally)

[Tang&Abbeel, NIPS 2011]

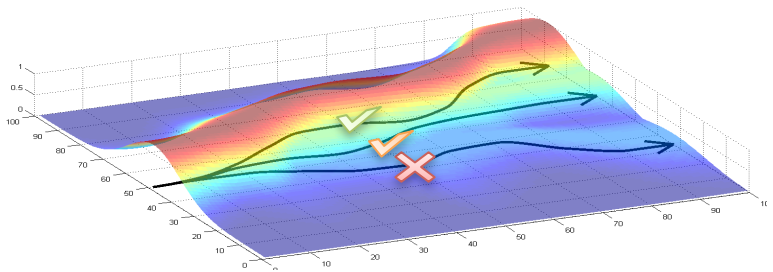
Likelihood Ratio Gradient Estimate

- As formulated thus far: unbiased but very noisy
- Fixes that lead to real-world practicality
  - Baseline
  - Temporal structure
  - Also: KL-divergence trust region / natural gradient (= general trick, equally applicable to perturbation analysis and finite differences)

Likelihood Ratio Gradient: Intuition

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

- Gradient tries to:
  - Increase probability of paths with positive R
  - Decrease probability of paths with negative R



! Likelihood ratio changes probabilities of experienced paths, does not try to change the paths (<-> Path Derivative)

Likelihood Ratio Gradient: Baseline

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

baseline

→ Consider baseline b: $\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta)(R(\tau^{(i)}) - b)$

intuitively, we want to increase probability of paths that are better than average, and decrease probability of paths that are worse than average. We if can measure average by looking at the rollouts and seeing what the average is, we can call this our baseline and plug it into the equation

still unbiased!
[Williams 1992]

Math that proves that *b* is unbiased. Not really important.

$$\mathbb{E}\left[\nabla_\theta \log P(\tau;\theta) b\right]$$
$$= \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) b$$
$$= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} b$$
$$= \sum_\tau \nabla_\theta P(\tau;\theta) b$$
$$= \nabla_\theta \left(\sum_\tau P(\tau) b \right) = b \nabla_\theta \left(\sum_\tau P(\tau) \right) = b \times 0$$

OK as long as baseline doesn't depend on action in logprob(action)

it doesn't, though so we don't have to worry about this

Likelihood Ratio and Temporal Structure

- Current estimate:

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

this is our policy gradient estimate now

so far we've largely ignored the time component. But we know we can decompose this equation to be a sum over the time steps

$$= \frac{1}{m}\sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=0}^{H-1} R(s_t^{(i)}, u_t^{(i)}) - b\right)$$

$$= \frac{1}{m}\sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \left[\left(\sum_{k=0}^{t-1} R(s_k^{(i)}, u_k^{(i)})\right) + \left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)})\right) - b\right]\right)$$

(assuming R is positive) we increase the grad-log probability for an action we took, given the state we were in, by the amount the rewards experienced from that time onward from time t to H-1 (the end) is better than what you'd get on average from that time onwards

we remove this term

Doesn't depend on $u_t^{(i)}$

Ok to depend on $s_t^{(i)}$

- Removing terms that don't depend on current action can lower variance:

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - b(s_t^{(i)})\right)$$

so now we have the standard policy gradient equation that we'll see a lot

[Policy Gradient Theorem: Sutton et al, NIPS 1999; GPOMDP: Bartlett & Baxter, JAIR 2001; Survey: Peters & Schaal, IROS 2006]

Baseline Choices

- Good choice for b?
  - Constant baseline: $b = \mathbb{E}\left[R(\tau)\right] \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)})$
  - Optimal Constant baseline: $b = \frac{\sum_i \left(\nabla_\theta \log P(\tau^{(i)};\theta)\right)^2 R(\tau^{(i)})}{\sum_i \left(\nabla_\theta \log P(\tau^{(i)};\theta)\right)^2}$

    the optimal baseline uses a weight average of the grad-log probability of the rollouts...this is an improvement, but no one really uses this because it's not a huge improvement.
  - Time-dependent baseline: $b_t = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)})$

    you can also take the average of some time period. So you could take the average over the remaining time you have after making an action from a state.
  - State-dependent expected return:

    value tells us how much reward we will get from this state onward by following policy pi

$$b(s_t) = \mathbb{E}\left[r_t + r_{t+1} + r_{t+2} + \ldots + r_{H-1}\right] = V^{\pi}(s_t)$$

→ Increase logprob of action proportionally to how much its returns are better than the expected return under the current policy

State-dependent return is what we'll do. We can say, I've seen rewards from state s_t, so we can create a value function V(s_t)
Remember V(s_t) is the amount of value you'll get from that state onwards. This is what we really want because this Value function tells us how much reward on average we would get based on our current policy. So if our action got us a better value, then we increase the logprob of that action. If it got us a worse value, we decrease the logprob of that action.

[See: Greensmith, Bartlett, Baxter, JMLR 2004 for variance reduction techniques.]

This is called the Monte Carlo Estimate of Value Function estimate of V_pi

Estimation of V^π

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1}R(s_k^{(i)},u_k^{(i)}) - V^\pi(s_k^{(i)})\right)$$

How to estimate?

We could use a Transition Model. But what if we don't have the Transition model? We can estimate it using sampling!

- Init $V^\pi_{\phi_0}$
  - Collect trajectories $\tau_1, \ldots, \tau_m$ collect a bunch of trajectories under the current policy. This will serve as our training data
  - Regress against empirical return:

we can use a neural network to approximate the Value function. This NN will be parameterized by vector phi. The input would be the state s at time t, and the output would be the Value of s at time t. In other words the NN will predict for the state you experienced the total rewards after being in that state.

So now we'd have one NN for approximating the value function V(s) and one NN for approximating the Policy Function

$$\phi_{i+1} \leftarrow \arg\min_\phi \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\left(V^\pi_\phi(s_t^{(i)}) - \left(\sum_{k=t}^{H-1}R(s_k^{(i)},u_k^{(i)})\right)\right)^2$$

then we can use this Value Function in our Policy gradient update

So this is another way to estimate V_pi!

This is called the TD Estimate of Value Function estimate of V_pi

Estimation of V^π

There are pros and cons to both approaches for estimating our Value Function.

On the previous slides, it's simple, but it's not sample efficient.

On this slide, it is sample efficient. But it can be harder to stabilize and you might need to tweak it to be stable.

- Bellman Equation for V^π

$$V^\pi(s) = \sum_u \pi(u|s) \sum_{s'} P(s'|s,u)[R(s,u,s') + \gamma V^\pi(s')]$$

we are doing sample based evaluation because we don't have P

This is saying that we want our V(s) equal our immediate reward + the value of V(s') based on the current estimate we have of our Value function.

- Init $V^\pi_{\phi_0}$
  - Collect data {s, u, s', r}

so as we collect samples, we only care about these transitions. In the previous slide we looked at entire trajectories.

  - Fitted V iteration:

regularization term so you change your Value function estimate too much based on the most recent sample

phi_i is our current estimate of the value function

$$\phi_{i+1} \leftarrow \min_\phi \sum_{(s,u,s',r)} \|r + V^\pi_{\phi_i}(s') - V_\phi(s)\|_2^2 + \lambda\|\phi - \phi_i\|_2^2$$

We want our current V(s) to be equal to our immediate reward + V(s') per our current estimate of the Value function. This is the ideal case.

so this is calculating the difference between our immediate reward + our current estimate of V(s'), and the actual aggregate reward we ended up getting from state s V(s)

Vanilla Policy Gradient

randomly initialize your NN that predicts the policy

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b

we saw many ways to do this but estimating the Value Function is probably the best way

for iteration=$1, 2, \ldots$ **do**

after every iteration we update the policy. It reality this is a stochastic approach but in general it should continue to get better and better

we execute the current policy several times to get a lot of data

Collect a set of trajectories by executing the current policy

then for each timestep in a trajectory, we compute the total (discounted) reward from that time onwards

At each timestep in each trajectory, compute the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

even if we don't want to discount it, discounting can help reduce variance which is really good. So we might discount it anyways, even if we don't want future rewards to necessarily be less important that recent rewards

We subtract the total discounted reward for a given timestep t by the baseline at that state and timestep. This is what we did on the previous slide.

the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.

recompute Value Function based on the trajectories we've seen so far. In this alg, we have the actual rewards so we're using the Monte Carlo method. We could have instead used the TD method

Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$, summed over all trajectories and timesteps.

summed over all times and all states and actions that we encountered

Update the policy, using a policy gradient estimate $\hat{g}$, which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$

end for

~ [Williams, 1992]

Recall Our Likelihood Ratio PG Estimator

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)},u_k^{(i)}) - V^\pi(s_k^{(i)})\right)$$

Recall Our Likelihood Ratio PG Estimator

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)},u_k^{(i)}) - V^\pi(s_k^{(i)})\right)$$

Recall Our Likelihood Ratio PG Estimator

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)},u_k^{(i)}) - V^\pi(s_k^{(i)})\right)$$

what we really want is the Q-value for the state and action, instead of just an estimate from one rollout.

If we just use the rollout we'll have high variance

- Estimation of Q from *single* roll-out

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

Recall Our Likelihood Ratio PG Estimator

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)},u_k^{(i)}) - V^\pi(s_k^{(i)})\right)$$

- Estimation of Q from *single* roll-out

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

- = high variance per sample based / no generalization

Further Refinements

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)},u_k^{(i)}) - V^\pi(s_k^{(i)})\right)$$

- Estimation of Q from *single* roll-out

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

- = high variance per sample based / no generalization
  - Reduce variance by discounting

Recall Our Likelihood Ratio PG Estimator

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)},u_k^{(i)}) - V^\pi(s_k^{(i)})\right)$$

- Estimation of Q from *single* roll-out

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

- = high variance per sample based / no generalization
  - Reduce variance by discounting
  - Reduce variance by function approximation (=critic)

Variance Reduction by Discounting

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

→ introduce discount factor as a hyperparameter to improve estimate of Q:

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

Reducing Variance by Function Approximation

$$Q^{\pi,\gamma}(s,u) = \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, u_0 = u]$$

Reducing Variance by Function Approximation

$$
\begin{aligned}
Q^{\pi,\gamma}(s,u) &= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, u_0 = u] \\
&= \mathbb{E}[r_0 + \gamma V^{\pi}(s_1) \mid s_0 = s, u_0 = u]
\end{aligned}
$$

Reducing Variance by Function Approximation

$$
\begin{aligned}
Q^{\pi,\gamma}(s,u) &= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, u_0 = u] \\
&= \mathbb{E}[r_0 + \gamma V^{\pi}(s_1) \mid s_0 = s, u_0 = u] \\
&= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 V^{\pi}(s_2) \mid s_0 = s, u_0 = u]
\end{aligned}
$$

Reducing Variance by Function Approximation

$$
\begin{aligned}
Q^{\pi,\gamma}(s,u) &= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, u_0 = u] \\
&= \mathbb{E}[r_0 + \gamma V^{\pi}(s_1) \mid s_0 = s, u_0 = u] \\
&= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 V^{\pi}(s_2) \mid s_0 = s, u_0 = u] \\
&= \mathbb{E}[r_0 + \gamma r_1 + +\gamma^2 r_2 + \gamma^3 V^{\pi}(s_3) \mid s_0 = s, u_0 = u] \\
&= \cdots
\end{aligned}
$$

- ***Async Advantage Actor Critic (A3C)*** *[Mnih et al, 2016]*
  - $\hat{Q}$ one of the above choices (e.g. k=5 step lookahead)

so instead of using the sum of all future rewards, you use the sum of k rewards plus your value function estimate

Reducing Variance by Function Approximation

$$
\begin{aligned}
Q^{\pi,\gamma}(s,u) &= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, u_0 = u] && (1-\lambda) \\
&= \mathbb{E}[r_0 + \gamma V^{\pi}(s_1) \mid s_0 = s, u_0 = u] && (1-\lambda)\lambda \\
&= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 V^{\pi}(s_2) \mid s_0 = s, u_0 = u] && (1-\lambda)\lambda^2 \\
&= \mathbb{E}[r_0 + \gamma r_1 + +\gamma^2 r_2 + \gamma^3 V^{\pi}(s_3) \mid s_0 = s, u_0 = u] \\
&= \cdots && (1-\lambda)\lambda^3
\end{aligned}
$$

- ***Generalized Advantage Estimation (GAE)*** *[Schulman et al, ICLR 2016]*
  - $\hat{Q}$ = lambda exponentially weighted average of all the above

Reducing Variance by Function Approximation

$$
\begin{aligned}
Q^{\pi,\gamma}(s,u) &= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, u_0 = u] && (1-\lambda) \\
&= \mathbb{E}[r_0 + \gamma V^{\pi}(s_1) \mid s_0 = s, u_0 = u] && (1-\lambda)\lambda \\
&= \mathbb{E}[r_0 + \gamma r_1 + \gamma^2 V^{\pi}(s_2) \mid s_0 = s, u_0 = u] && (1-\lambda)\lambda^2 \\
&= \mathbb{E}[r_0 + \gamma r_1 + +\gamma^2 r_2 + \gamma^3 V^{\pi}(s_3) \mid s_0 = s, u_0 = u] \\
&= \cdots && (1-\lambda)\lambda^3
\end{aligned}
$$

- ***Generalized Advantage Estimation (GAE)*** *[Schulman et al, ICLR 2016]*
  - $\hat{Q}$ = lambda exponentially weighted average of all the above
- ~ TD(lambda) / eligibility traces [Sutton and Barto, 1990]

Actor-Critic with A3C or GAE

- Policy Gradient + Generalized Advantage Estimation:
  - Init π_{θ_0} $V^{\pi}_{\phi_0}$
  - Collect roll-outs {s, u, s', r} and $\hat{Q}_i(s,u)$
  - Update:

update value function

$$\phi_{i+1} \leftarrow \min_{\phi} \sum_{(s,u,s',r)} \|\hat{Q}_i(s,u) - V^{\pi}_{\phi}(s)\|_2^2 + \kappa\|\phi - \phi_i\|_2^2$$

update policy gradient

$$\theta_{i+1} \leftarrow \theta_i + \alpha \frac{1}{m} \sum_{k=1}^{m} \sum_{t=0}^{H-1} \nabla_{\theta} \log \pi_{\theta_i}(u_t^{(k)}|s_t^{(k)}) \left(\hat{Q}_i(s_t^{(k)}, u_t^{(k)}) - V^{\pi}_{\phi_i}(s_t^{(k)}) \right)$$

Note: many variations, e.g. could instead use 1-step for V, full roll-out for pi:

What does Actor-Critic mean?
- It means you are running a Policy Gradient method where you estimate a Value Function for your Baseline
- Once you have a value function for your Baseline it's no longer called a policy gradient

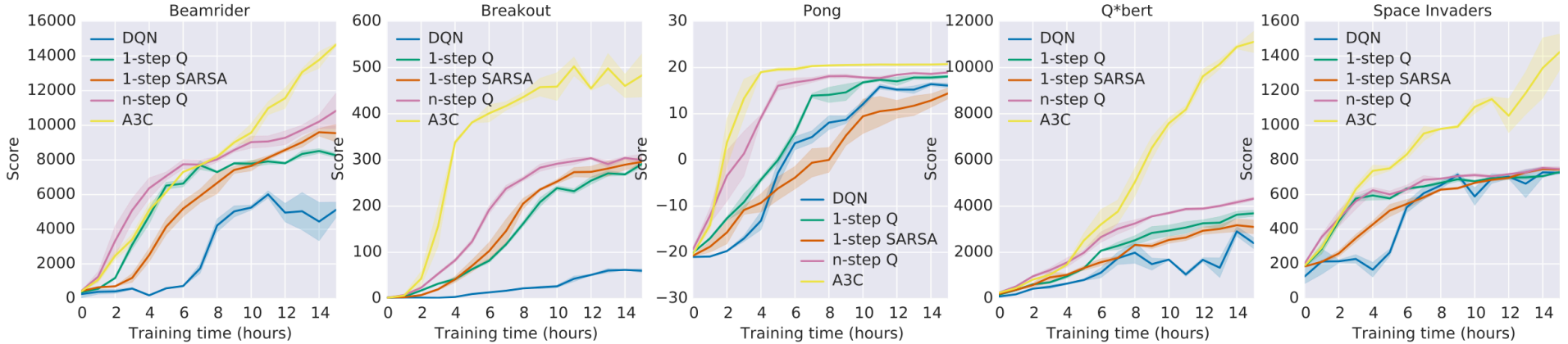
$$\phi_{i+1} \leftarrow \min_{\phi} \sum_{(s,u,s',r)} \|r + V^{\pi}_{\phi_i}(s') - V_{\phi}(s)\|_2^2 + \lambda\|\phi - \phi_i\|_2^2$$

$$\theta_{i+1} \leftarrow \theta_i + \alpha \frac{1}{m} \sum_{k=1}^{m} \sum_{t=0}^{H-1} \nabla_{\theta} \log \pi_{\theta_i}(u_t^{(k)}|s_t^{(k)}) \left(\sum_{t'=t}^{H-1} r_{t'}^{(k)} - V^{\pi}_{\phi_i}(s_{t'}^{(k)}) \right)$$

lots of variations, could use Monte Carlo vs A3C vs. GAE

Async Advantage Actor Critic (A3C)

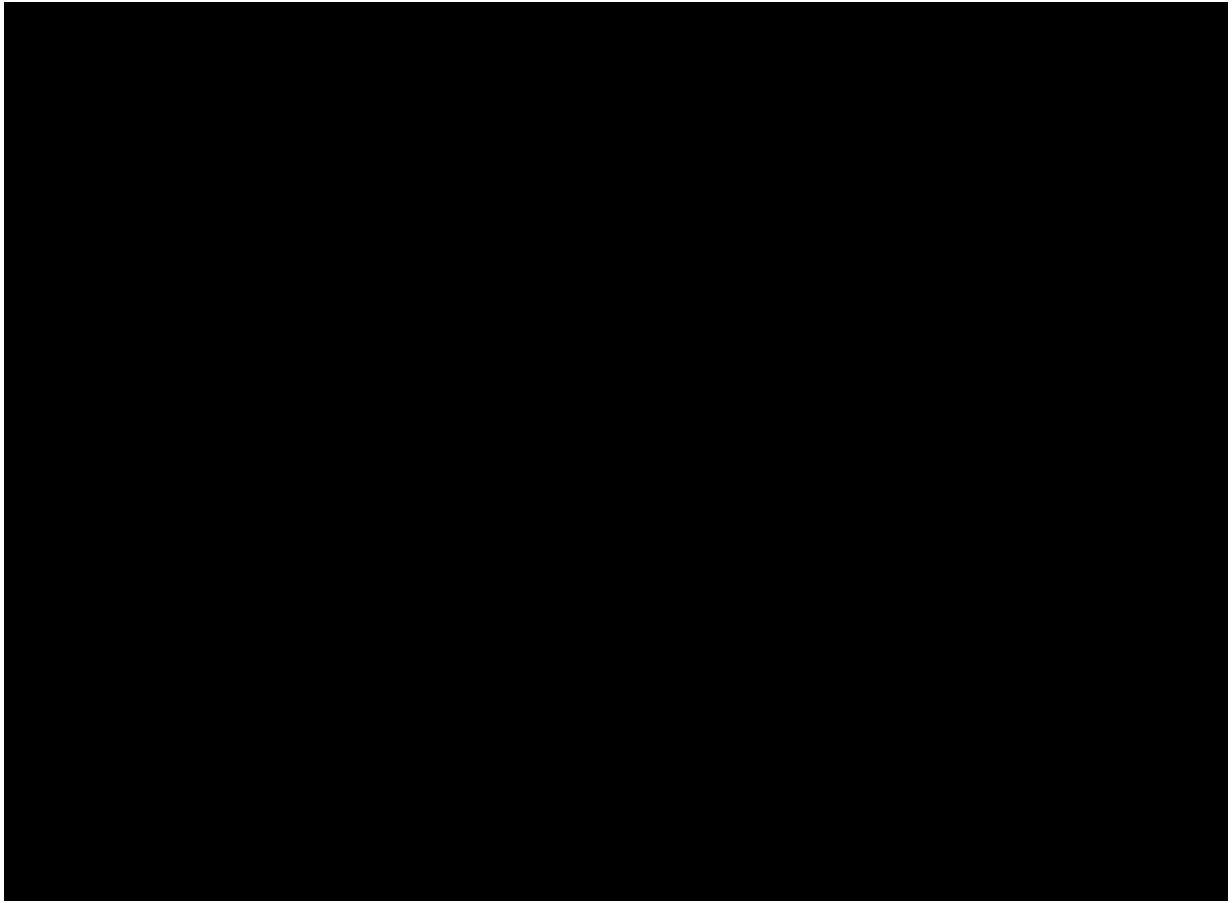
- [Mnih et al, ICML 2016]
  - Likelihood Ratio Policy Gradient
  - n-step Advantage Estimation



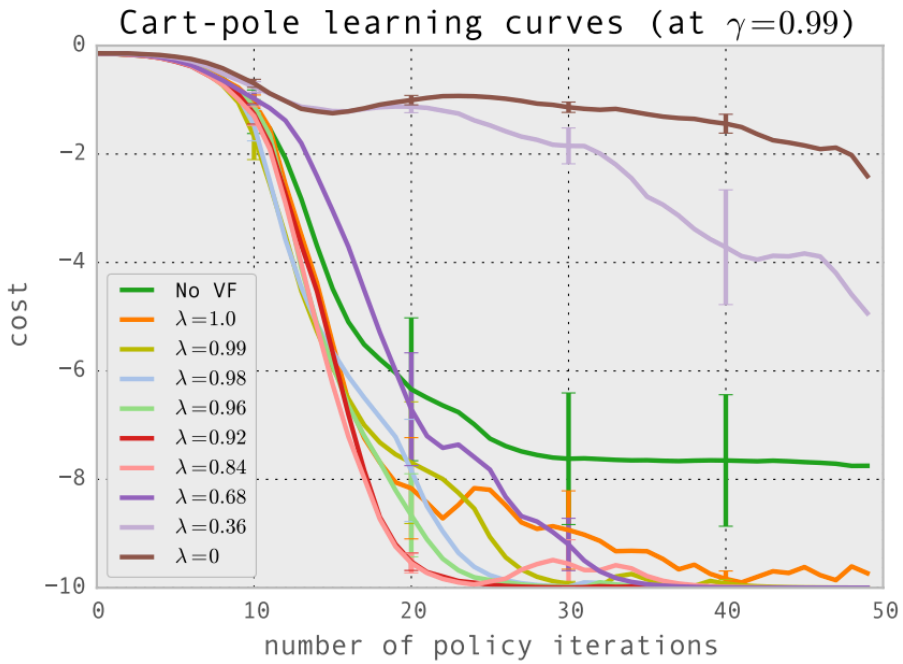
uses RNN so it can remember where in the labyrinth it's been

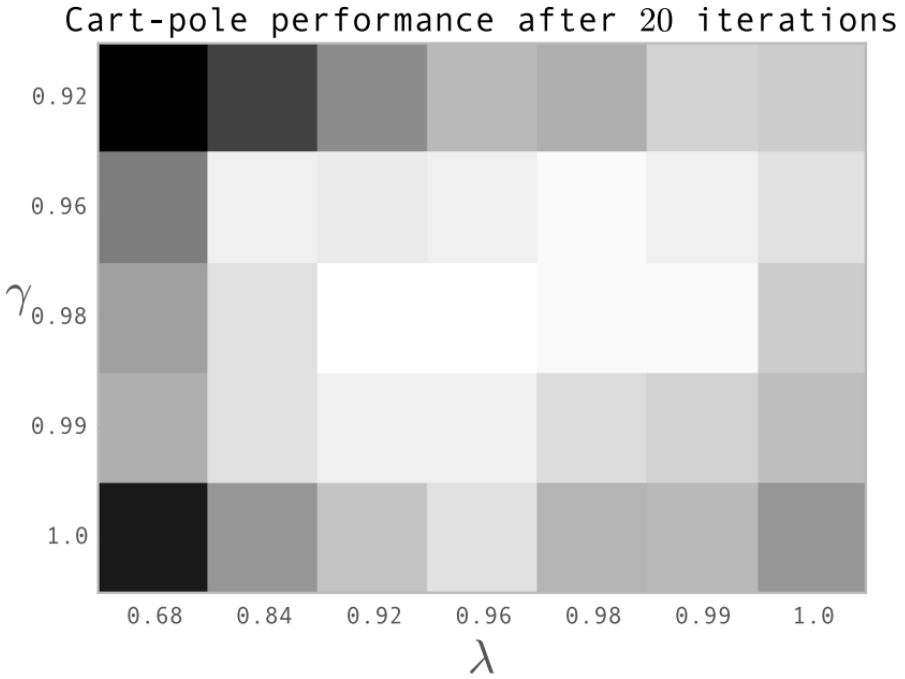
RNN is good for remembering history

A3C -- labyrinth



GAE: Effect of gamma and lambda

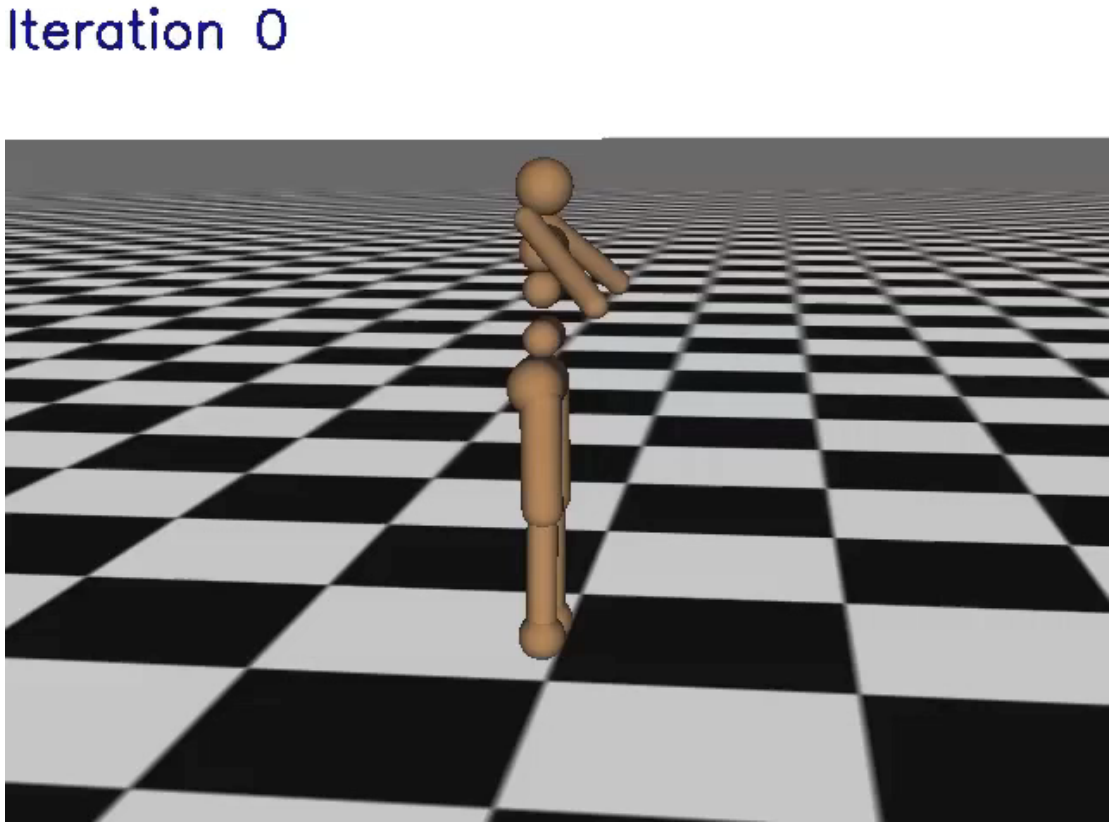




lambda: exponential decay of horizons
gamma: discount factor

[Schulman et al, 2016 -- GAE]

Learning Locomotion (TRPO + GAE)



[Schulman, Moritz, Levine, Jordan, Abbeel, 2016]

In Contrast: Darpa Robotics Challenge