Direct Policy Search

CSE599G: Deep Reinforcement Learning

Aravind Rajeswaran and Kendall Lowrey

University of Washington Seattle

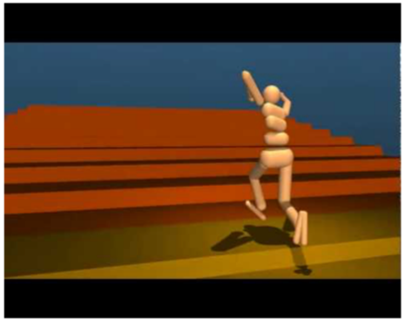
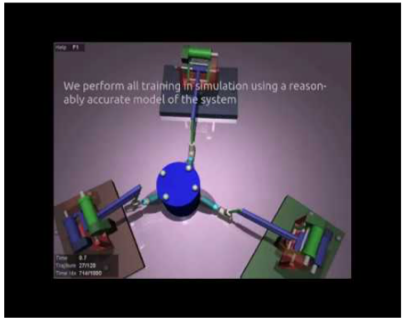
April 9, 2018

Announcements

- Use **slack** for all course related communication with instructors
- Homework 1 is posted on the website. Due: April 23
- Project proposals (2 page) due on April 20
- Teams: ideal size would be two (alone or teams of three are also fine, but no more than three members in a team)
- Use office hours to discuss about project and the homework. It’s very important to pick a problem and start working towards a good solution quickly!

Overview

- So far, we have studied how to solve small and known (tabular) MDPs with dynamic programming based methods like policy iteration.
- In this module: break assumptions and find general purpose methods
- State and action spaces can be very large, even infinite (e.g. continuous)
- Dynamics model of the environment is unknown

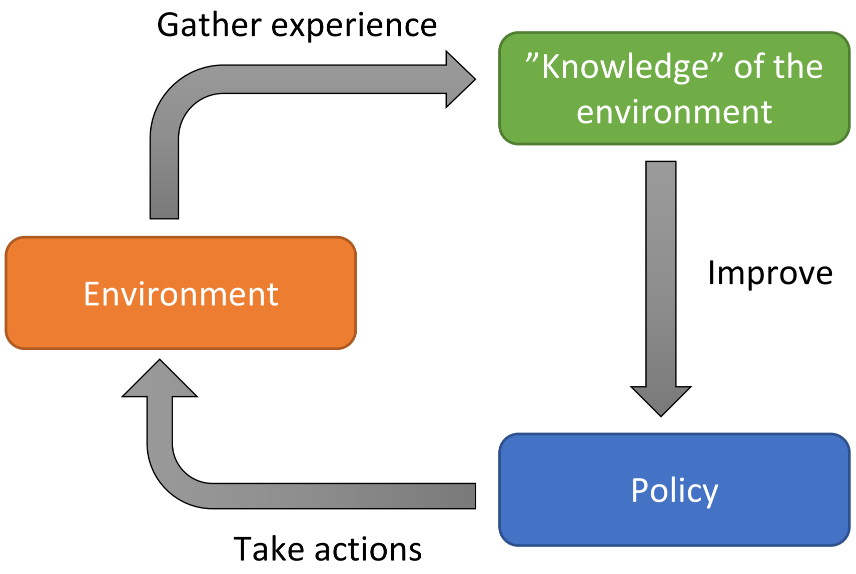


We perform all training in simulation using a reasonably accurate model of the system

Role of function approximation

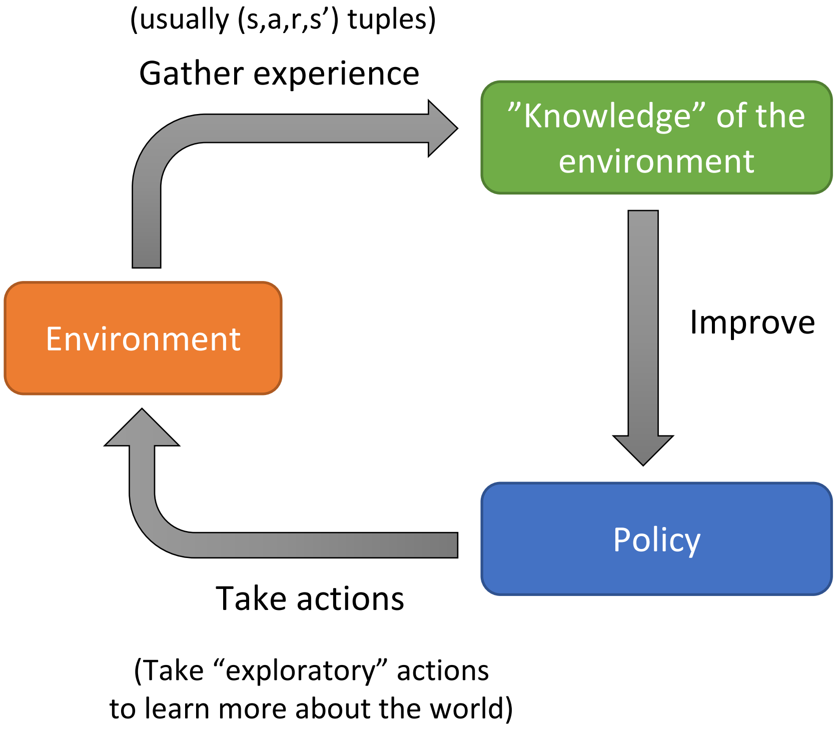
- Computation: we need some efficient way to work in very large or continuous (i.e. infinite) spaces. Not possible with tabular representation.
- Algorithmic: we have efficient methods such as gradient descent for searching in functional space. Very different solution paradigms for tabular settings.
- Generalization: tabular representation does not have any underlying metric space, so notions like “similarity” between states are hard to define

Functional approximation is the backbone of machine learning, and we want to use it for solving reinforcement learning problems as well !!

Anatomy of Reinforcement Learning



Anatomy of Reinforcement Learning


(usually (s,a,r,s') tuples)
Gather experience
"Knowledge" of the environment
Improve
Environment
Policy
Take actions
(Take "exploratory" actions to learn more about the world)

$\hat{Q} = \sum \gamma^t r_t$ (Monte-Carlo returns)
Fit $Q_\phi(s, a)$ (actor-critic, Q-learning)
Fit $p(s'|s, a; w)$ (model-based)

(Gradient-based,
dynamic programming,
continuation relaxation etc.)

$\theta \leftarrow \theta + \alpha \nabla_\theta \eta(\theta)$ (policy grad)
$\pi(s) = \arg\max_a Q_\phi(s, a)$ (Q-learning)
π = optimize using $p(s'|s, a; w)$
(model-based optimization)

Direct Policy Search

- One class of extremely successful general purpose RL methods
- We consider parameterized policies, denoted using $\pi(.|s;\theta)$ or $\pi_\theta(.|s)$, and optimize the parameters of the policy directly (without learning other quantities)
- Policy parameterization can be stochastic or deterministic (depends on algo)
- Gradient descent: $\theta \leftarrow \theta + \alpha \nabla_\theta \eta(\theta)$

- Very robust and works well uniformly across multiple types of RL problems like games, robotics, NLP etc.
- Generic but may not be as efficient as domain specific methods
- Very simple and general purpose solution: hard to beat baseline in general

Direct Policy Search

Computing the gradient does not require knowing the dynamics model !!

How to get the gradient? We have a few options:

- Stochastic finite differencing (DFO, evolution etc.)
- Score function estimator (REINFORCE)
- Score function estimator with variance reduction
- Parameterized Q-function (Actor-Critic methods)

Other considerations:

- Is gradient the best direction to follow? (Natural gradient, Gauss-Newton)
- How to pick the step size? (trust region, normalized gradients, adaptive methods)

Policy Optimization

$$\max_{\theta\in\mathbb{R}^d} \eta(\theta) := \mathbb{E}_{\boldsymbol{s_0\sim\rho_0}}\left[\sum_{t=0}^{T} \gamma^t\, r(s_t, \pi(.|s_t;\theta))\right]$$

- In the tabular case, we had an optimal policy that is optimal everywhere
- Being optimal everywhere is not possible with function approximation
- We need to specify where we wish to focus our approximation power
- We know that the MDP has a start state distribution of ρ_0 -- so we will focus our approximation power on those states that we tend to visit when started from this distribution

DFO and evolutionary methods

Cant compute analytical gradients, so use numerical approximation:

$$\nabla_\theta \eta(\theta)[i] = \frac{\eta(\theta + \epsilon e_i) - \eta(\theta)}{\epsilon}$$

- $\eta := \mathbb{E}[R]$ -- we actually don’t have access to η, but only noisy estimates
- Need $O(d)$ function calls to get the gradient: almost defeats the purpose of gradient descent, and random search could very well work better
- Even after getting the gradient, picking the step size can be a bit tricky

Cross Entropy Method

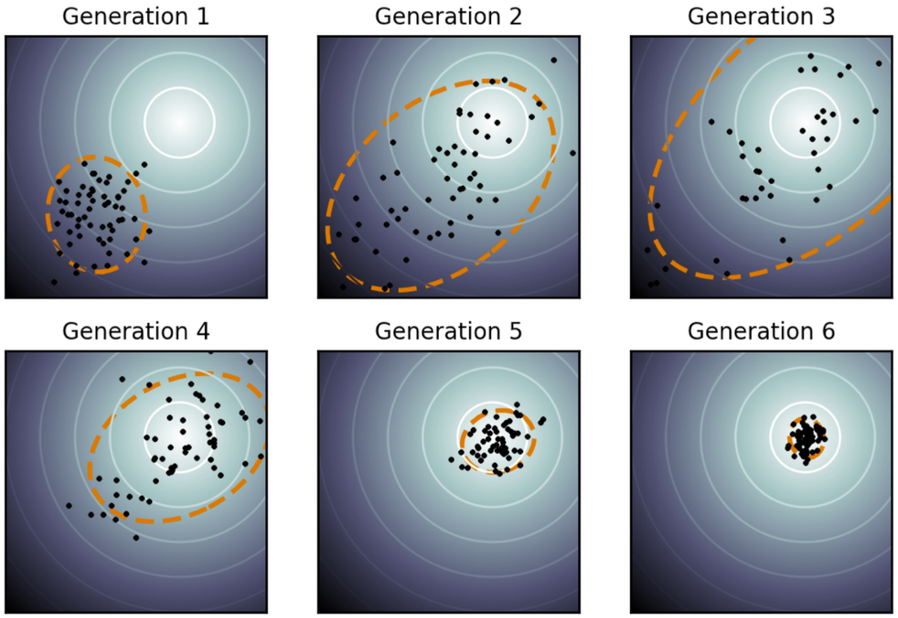
```
Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$
**for** iteration $= 1, 2, \ldots$ **do**
    Collect n samples of $\theta_i \sim N(\mu, \mathrm{diag}(\sigma))$
    Perform one episode with each $\theta_i$, obtaining reward $R_i$
    Select the top $p$% of $\theta$ samples (e.g. $p = 20$), the **elite set**
    Fit a Gaussian distribution, to the elite set, updating $\mu, \sigma$.
**end for**
Return the final $\mu$.
```

Sometimes works embarrassingly well

(slide: John Schulman)

CMA-ES

- A smarter version of the cross entropy method
- Adapts the covariance of the sampling distribution based on fitness



Score Function Estimator

- Let us pick some sampling distribution for the parameters: $\theta \sim P(.|\mu)$
- Here μ are parameters of sampling distribution (e.g. mean, variance of Gaussian)
- We want a sampling distribution that samples more often in the “good” parts
- Optimize over parameters of the sampling distribution

$$\max_{\mu \in \mathbb{R}^d} \eta(\mu) := \mathbb{E}_{\boldsymbol{s_0 \sim \rho_0}, \boldsymbol{\theta \sim P(.|\mu)}} \left[\sum_{t=0}^{T} \gamma^t \, r(s_t, \pi(.|s_t; \theta)) \right]$$

Note: the distribution for sample collection is a function of the decision variables.

Very different from stochastic optimization problems in supervised learning where the data comes from a fixed distribution.

Score Function Estimator

Simplify notation as:

$$\max_{\mu} \eta(\mu) \coloneqq \mathbb{E}_{x\sim P(.|\mu)}[f(x)]$$

$$\nabla_\mu \eta(\mu) = \nabla_\mu \int dx\, f(x) P(x|\mu) = \int dx\, f(x)\, \nabla_\mu P(x|\mu)$$

$$= \int dx\, f(x)\, \nabla_\mu P(x|\mu) \frac{P(x|\mu)}{P(x|\mu)} = \int dx\, f(x)\, \nabla_\mu \ln P(x|\mu)\, P(x|\mu)$$

$$= \mathbb{E}_{x\sim P(.|\mu)}\left[f(x) \nabla_\mu \ln P(x|\mu)\right]$$

Score Function Estimator

Final Algorithm

In a loop:

- Sample policy parameters $\theta^1, \theta^2, \theta^3, \dots, \theta^K$ using $P(.|\mu)$
- Compute respective evaluations or "fitness": $R(\theta^1), R(\theta^2), \dots, R(\theta^K)$
- Compute gradient: $\nabla_\mu \eta(\mu) = \frac{1}{K}\sum_{i=1}^{K} R\left(\theta^i\right)\nabla_\mu \ln P(\theta = \theta^i|\mu)$
- Gradient ascent: $\mu \leftarrow \mu + \alpha\nabla_\mu \eta(\mu)$

- Similar to weighted maximum likelihood estimation
- Theoretical analysis for the Gaussian sampling distribution by Nesterov
- We are optimizing for a ”smoothed” objective
- Has a terribly high variance (we will return to variance reduction later)

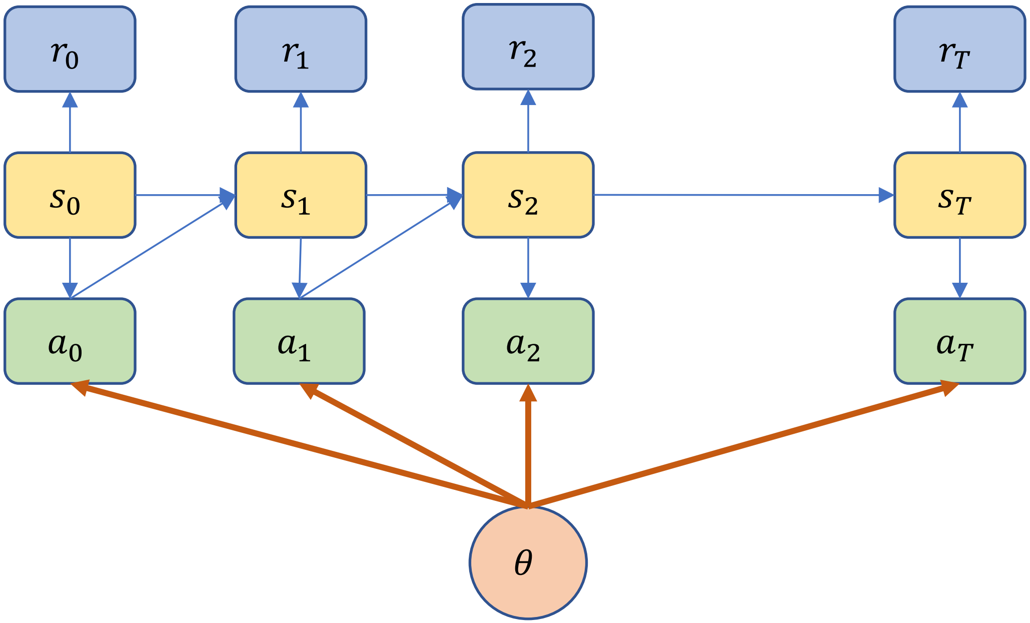
Flow graph of RL

Which parts are differentiable in the general case?

$$\begin{tikzcd}
r_0 & r_1 & r_2 & r_T \\
s_0 \arrow[u] \arrow[d] \arrow[r] & s_1 \arrow[u] \arrow[d] \arrow[r] & s_2 \arrow[u] \arrow[d] \arrow[r] & s_T \arrow[u] \arrow[d] \\
a_0 \arrow[ur] & a_1 \arrow[ur] & a_2 & a_T \\
 & & \theta \arrow[ull] \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

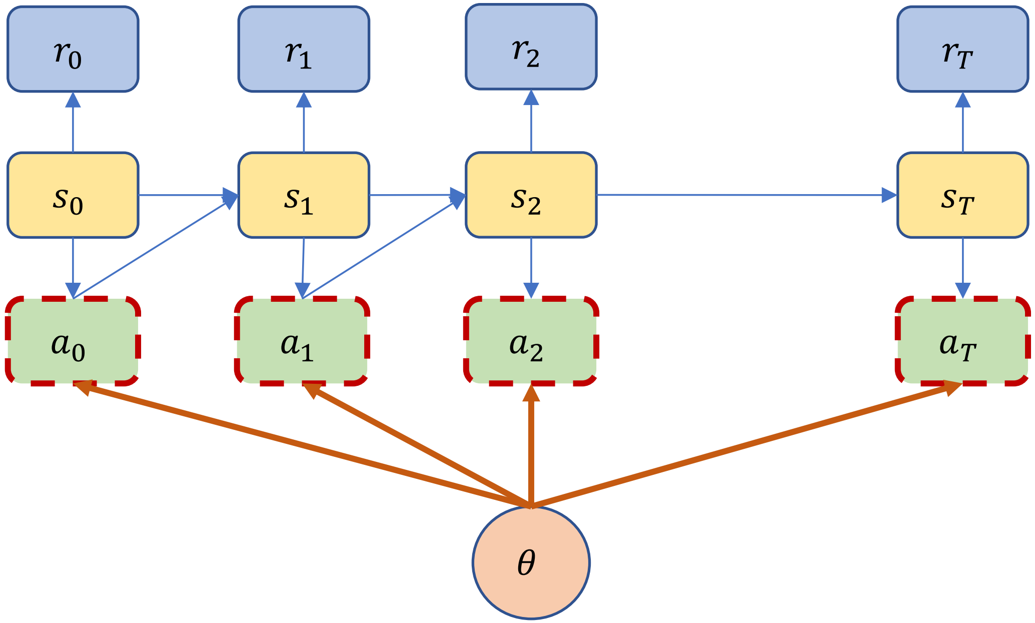
Flow graph of RL

Policy parameterization is known to us: we can certainly differentiate through it!


$$\begin{tikzcd}
r_0 & r_1 & r_2 & & r_T \\
s_0 \arrow[u] \arrow[r] \arrow[d] & s_1 \arrow[u] \arrow[r] \arrow[d] & s_2 \arrow[u] \arrow[rr] \arrow[d] & & s_T \arrow[u] \arrow[d] \\
a_0 \arrow[ur] & a_1 \arrow[ur] & a_2 & & a_T \\
& & \theta \arrow[ull] \arrow[ul] \arrow[u] \arrow[urr] & &
\end{tikzcd}$$

Flow graph of RL

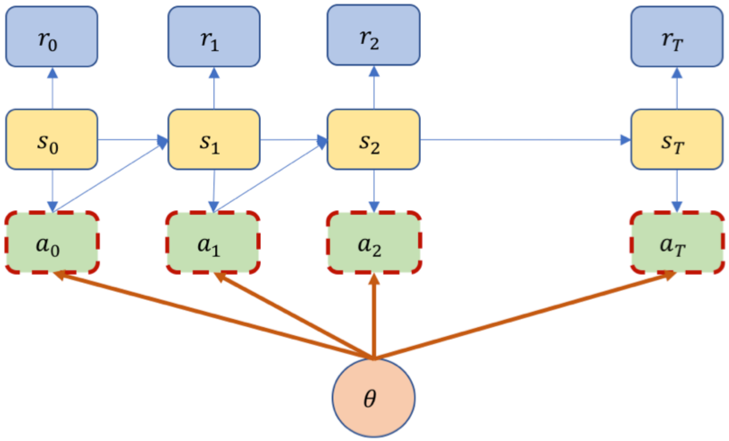
Instead of sampling the policy parameters, sample the actions!



REINFORCE

Sample in the action space and use the score function estimator. In this case, we will pick a stochastic policy, which defines the sampling distribution.

$$\nabla_\theta \eta(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \ln \pi(a_t|s_t) \left(\sum_{t'=t}^{T} \gamma^{t'} r_t\right)\right]$$



Based on causality (time) we see that actions only affect future rewards and not the past ones!

Monte Carlo REINFORCE

In a loop:

- Sample K trajectories: $\tau^1, \tau^2, \tau^3, \ldots, \tau^K$ using π_θ

  where $\tau^i = (s_0^i, a_0^i, r_0^i, s_1^i . a_2^i, r_2^i, \ldots, s_T^i, a_T^i, r_T^i)$

- For every trajectory and time-step, compute future performance:

  $R_t^i = \sum_{t'=t}^{T} \gamma^{t'} r_t^i$ which serves as the fitness or score.

- Compute gradient: $\nabla_\theta \eta(\theta) = \frac{1}{K}\frac{1}{T}\sum_{i=1}^{K}\sum_{t=0}^{T} R_t^i \, \nabla_\theta \ln \pi(a_t^i | s_t^i; \theta)$

- Gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta \eta(\theta)$

Next Steps

- Variance reduction for policy gradient methods
- Importance sampling view
- Natural policy gradient and related ideas
- Bias-variance trade-offs with TD(λ) and GAE