Policy Search

Ron Parr
CSCI 5951-F
Brown University

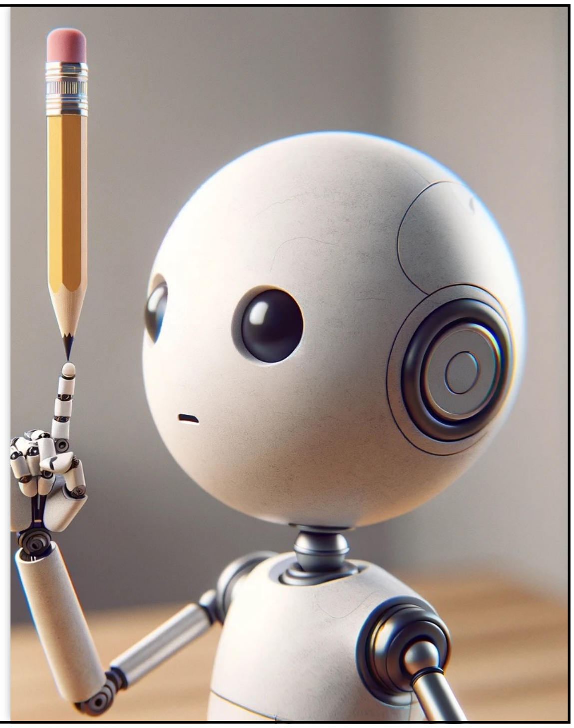
Some portions adapted from Sutton & Barto ch. 13

Find good policies w/o using Q/Value functions?

- Why bother?
- Approximate value function methods can be unstable
  - Values can diverge
  - Hard to provide meaningful performance guarantees
- In some problems finding a good approximation for a policy function may be easier than finding a good approximation for a value function

Example: Inverted pendulum

- Observation from homework: Obtaining a good functional form to represent the Value function/Q-functions isn't trivial
- A (near) optimal policy has a very simple form:
  - When angle and angular velocity have same sign, push in opposite direction
  - When angle and angular velocity have different sign, do nothing



Managing policy space

- Just like value functions, policies defined explicitly over huge state spaces are unwieldy
- Possible policy representations:
  - Lookup tables
  - Implicit in Q-functions
  - Decision trees
  - Neural networks mapping states to distributions over actions
  - Arbitrary programs
  - Etc. - almost anything goes

Searching policy space

- Natural representation choice for value functions: differentiable functions
- Natural optimization method for value functions: gradient descent

- Many choices for policy functions
- Many optimization methods

- Brief review of black(ish) box optimization methods...

Evaluating policies

- An in homework, we can evaluate a policy at a particular state by:
  - Simulating/running the policy
  - Recording discounted sum of rewards
  - Repeating and average until variance is reduced
- Evaluate policy overall by:
  - Sampling a start state from a distribution over start states
  - Repeat, average, etc. to get an expected policy value as a single number

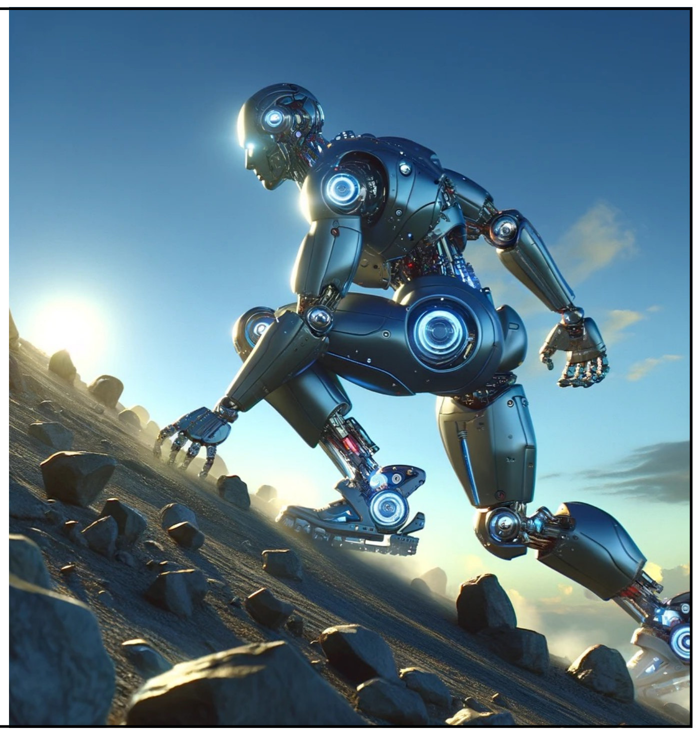
Improving policies

- Can view as a generic optimization problem
- Black box tells us f(x)
- Figure out how to adjust x to maximize f(x)

- Starting point:
  - Policy is an arbitrary function from states to actions
  - We have bounded set of possible changes we can try

Hill Climbing

- Evaluate current policy
- Evaluate set of candidate changes
- Picking a change:
  - Steepest ascent (largest improvement)
  - Stochastic – randomly pick one of the good ones
  - First choice
- This is a *greedy* procedure
- Inefficient



Genetic Algorithms (over discrete spaces)

- GAs run hot and cold (cold now, hotish in 90’s)
- Biological metaphors to motivate search
- Organism is a word from a finite alphabet (organisms = policies)
- Fitness measures performance on task (fitness = policy evaluation)
- Uses multiple organisms (parallel search)
- Uses mutation (random steps) - asexual reproduction
- Uses crossover (combining elements of “fit” solutions) – sexual reproduction

Is this a good idea?

- Has worked well in some examples
- Can be very brittle
  - Representations must be carefully engineered
  - Sensitive to mutation rate
  - Sensitive to details of crossover mechanism
- For the same amount of engineering & computation, other approaches might do better

Stochastic policies

- But policies are discrete, so how do we do something like gradient descent in policy space???
- So far, we have assumed a deterministic policy, i.e., we always take the same action in every state
- Why not $\pi(s,a) = p(a|s)$?
- Nothing wrong with doing this, i.e., Bellman equation still works, but…

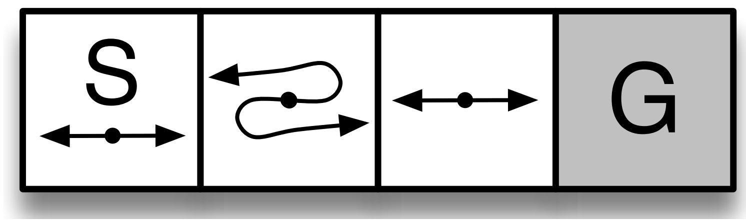
Deterministic policies are optimal

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

- Could use a stochastic policy to break ties for max, but why?
- There always exists an optimal deterministic policy

Why use stochastic policies then?

- Can't tune a non-differentiable max by gradient descent
- Deterministic policies are optimal for true MDPs, but if Markov property is violated (or our features have the same effect), then stochastic policies may be better
- Example from Sutton & Barto (assumes all states appear identical)



An example policy function: Softmax

$$\pi(a|s,\boldsymbol{\theta}) = \frac{\exp(h(s,a,\boldsymbol{\theta}))}{\sum_b \exp(h(s,b,\boldsymbol{\theta})}$$

- Actions with higher h values selected more often
- Optionally add a temperature parameter τ that we multiply all h values by
- Low values of τ approach uniform random distribution
- High values of τ approach a max

From Sutton & Barto ch. 13

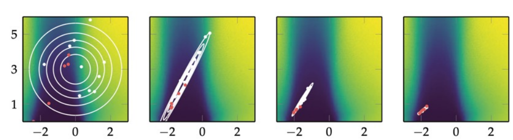
But what is h?

$$\pi(a|s,\boldsymbol{\theta}) = \frac{\exp(h(s,a,\boldsymbol{\theta}))}{\sum_b \exp(h(s,b,\boldsymbol{\theta}))}$$

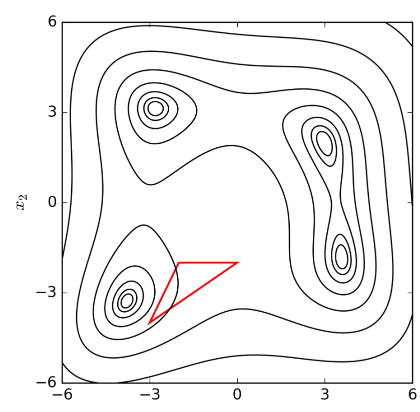
- Due to normalization, this will always be a distribution
- h can be an arbitrary (family of) function(s)
- One natural choice is to have have one linear function per action
- Other choices could be neural networks

Search in continuous spaces

- Assume function is continuous/differentiable
- Find local optimum
- Black box approaches (no analytic gradient)
  - Nelder-Mead
  - Hooke-Jeeves
  - Bayesian optimziation
  - Vaious particle/swarm/genetic approaches
  - All use multiple points to explicitly or implicitly represent the shape of the optimization surface



Cross entropy figure from text



Nelder Mead figu by

Limitations of black box search methods

- Ignore that we are solving an MDP (generic optimization)
- Can be particularly inefficient
  - When function evaluation (return on policy) is noisy
  - in high dimensional spaces

Policy gradient

- A family of methods for searching continuous policy space
- Smarter/more specific than black box methods
- Takes advantage of fact that we are solving an MDP

Policy gradient: What are we optimizing?

$$U(\theta) = \int p_\theta(\tau) R(\tau)\, d\tau$$

- Note: S&B takes a different (also interesting and useful to read) approach to deriving policy gradient
- **I am using the textbook's notation and derivation for the slides**
- U – utility
- θ – policy parameters
- τ – trajectory
- R(τ) – (discounted) sum of rewards accrued on τ

Finding the gradient

$$\begin{aligned}
\nabla U(\theta) &= \nabla_\theta \int p_\theta(\tau) R(\tau)\, d\tau \\
&= \int \nabla_\theta p_\theta(\tau) R(\tau)\, d\tau \\
&= \int p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau)\, d\tau \\
&= \mathbb{E}_\tau \left[\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau) \right]
\end{aligned}$$

The log trick: $\nabla \log x = \frac{\nabla x}{x}$

$$\mathbb{E}_\tau\left[\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau)\right]$$

$$\mathbb{E}_\tau[\nabla_\theta \log p_\theta(\tau) R(\tau)]$$

Probability of a trajectory

- Probability of a trajectory is product of probability of action choices
- Product decomposes into sum inside log
- For trajectory of length t, d=t

$$\nabla_\theta \log p_\theta(\tau) = \sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})$$

Basic Policy Gradient Algorithm (trajectory based REINFORCE)

- Sample a trajectory, compute discounted sum of returns
- Multiply by $\nabla_{\theta} \log p_{\theta}(\tau) = \sum_{k=1}^{d} \nabla_{\theta} \log \pi_{\theta}(a^{(k)} \mid s^{(k)})$
- Take gradient step in policy space
- Repeat

How well does this work?

- Digression…
- ~20 years ago, when very little worked, people had strong opinions about what would eventually work
- Value function proponents:
  - Value functions use the Bellman equation to enforce consistency
  - Should be more efficient than ignoring the Bellman equation
- Policy search proponents:
  - Value function approximation is unstable
  - Policy gradient directly optimizes the thing we care about
  - “Guaranteed” to find a local optimum in policy space

What actually happened

- Basic policy gradient has a huge problem with variance
- Probability of a trajectory of length n is a product of O(n) random events (both policy randomness and environment randomness)
- Variance grows with n
- Gradient signal for PG was very noisy

- Getting PG to work at all required:
  - Averaging over many trajectories and/or
  - Taking very small step sizes

A comment about gradients

- A gradient gives us 2 types of information
  - A direction
  - A magnitude
- For gradient descent, we tend to care most about getting direction right
- (GD algorithms use their own step size tricks)

- Noise corrupts the gradient direction information

Variance reduction

$$\nabla U(\boldsymbol{\theta}) = \mathbb{E}_\tau \left[\left(\sum_{k=1}^{d} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a^{(k)} \mid s^{(k)}) \right) \left(\sum_{k=1}^{d} r^{(k)} \gamma^{k-1} \right) \right]$$

(Notation)

$$\nabla U(\boldsymbol{\theta}) = \mathbb{E}_\tau \left[\left(\sum_{k=1}^{d} f^{(k)} \right) \left(\sum_{k=1}^{d} r^{(k)} \gamma^{k-1} \right) \right]$$

Variance reduction

$$\begin{aligned}
\nabla U(\boldsymbol{\theta}) &= \mathbb{E}_\tau \left[\left(\sum_{k=1}^{d} f^{(k)} \right) \left(\sum_{k=1}^{d} r^{(k)} \gamma^{k-1} \right) \right] \\
&= \mathbb{E}_\tau \left[\left(f^{(1)} + f^{(2)} + f^{(3)} + \cdots + f^{(d)} \right) \left(r^{(1)} + r^{(2)}\gamma + r^{(3)}\gamma^2 + \cdots + r^{(d)}\gamma^{d-1} \right) \right] \\
&= \mathbb{E}_\tau \begin{bmatrix} f^{(1)}r^{(1)} + f^{(1)}r^{(2)}\gamma + f^{(1)}r^{(3)}\gamma^2 + \cdots + f^{(1)}r^{(d)}\gamma^{d-1} \\ + f^{(2)}r^{(1)} + f^{(2)}r^{(2)}\gamma + f^{(2)}r^{(3)}\gamma^2 + \cdots + f^{(2)}r^{(d)}\gamma^{d-1} \\ + f^{(3)}r^{(1)} + f^{(3)}r^{(2)}\gamma + f^{(3)}r^{(3)}\gamma^2 + \cdots + f^{(3)}r^{(d)}\gamma^{d-1} \\ \vdots \\ + f^{(d)}r^{(1)} + f^{(d)}r^{(2)}\gamma + f^{(d)}r^{(3)}\gamma^2 + \cdots + f^{(d)}r^{(d)}\gamma^{d-1} \end{bmatrix}
\end{aligned}$$

Terms that multiply rewards by later actions contribute to magnitude by not direction

The REINFORCE Algorithm (Step based)

- Sample a trajectory
- Compute discounted sum of returns from each step as $r^{(k)}_{\text{to-go}}$ for each k
- Compute gradient for trajectory as

$$\sum_{k=1}^{d} \nabla_{\theta} \log \pi_{\theta}(a^{(k)} \mid s^{(k)})\gamma^{k-1} r^{(k)}_{\text{to-go}}$$

- Take one gradient step in policy space for each state in trajectory
- Repeat

How does step based reinforce work?

- Very, very slowly
- Still can be tricky to get to work in practice
- Variance can still be large
- Small step sizes needed for robust performance

Further variance reduction

$$\nabla U(\theta) = \mathbb{E}_\tau \begin{bmatrix} f^{(1)}r^{(1)} + f^{(1)}r^{(2)}\gamma + f^{(1)}r^{(3)}\gamma^2 + \cdots + f^{(1)}r^{(d)}\gamma^{d-1} \\ + f^{(2)}r^{(2)}\gamma + f^{(2)}r^{(3)}\gamma^2 + \cdots + f^{(2)}r^{(d)}\gamma^{d-1} \\ + f^{(3)}r^{(3)}\gamma^2 + \cdots + f^{(3)}r^{(d)}\gamma^{d-1} \\ \vdots \\ + f^{(d)}r^{(d)}\gamma^{d-1} \end{bmatrix}$$

$$= \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \left(\sum_{\ell=k}^{d} r^{(\ell)} \gamma^{\ell-1} \right) \right]$$

$$= \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \left(\gamma^{k-1} \sum_{\ell=k}^{d} r^{(\ell)} \gamma^{\ell-k} \right) \right]$$

$$= \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \gamma^{k-1} r_{\text{to-go}}^{(k)} \right]$$

$$\nabla U(\theta) = \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \gamma^{k-1} Q_\theta\left(s^{(k)}, a^{(k)}\right) \right]$$

Baselines

$$\nabla U(\theta) = \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \gamma^{k-1} \left(r_{\text{to-go}}^{(k)} - r_{\text{base}}(s^{(k)}) \right) \right]$$

Does not depend on action

- Why would this help?
- Further reduces variance
- Does not bias gradient since it does not depend upon actions

Baselines in practice

$$\nabla U(\theta) = \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\gamma^{k-1} A_\theta\left(s^{(k)}, a^{(k)}\right)\right]$$

Advantage function

$$A(s,a) = Q(s,a) - U(s)$$

Baseline

Comments

- This combines REINFORCE with Q-function approximation
- Good news: We no longer need to feud about which is better
- Bad news: Some motivation for policy gradient is lost

Conclusions

- Policy search originally viewed as alternative to value function methods
- Increasingly, we see these methods combined
- Compare with modified policy iteration
- Next: More advanced ways of combining these concepts