15-889e
Policy Search: Gradient Methods
Emma Brunskill

All slides from David Silver (with EB adding minor modifications), unless otherwise noted
Outline

1. Introduction

2. Finite Difference Policy Gradient

3. Monte-Carlo Policy Gradient

4. Actor-Critic Policy Gradient
Introduction

In past lectures we approximated the value or action-value function using parameters $\theta$,

\[
V_\theta(s) \approx V^\pi(s) \\
Q_\theta(s, a) \approx Q^\pi(s, a)
\]

A policy was generated directly from the value function
- e.g. using epsilon-greedy
- In this lecture we will directly parametrise the policy

\[
\pi_\theta(s, a) = P[a \mid s, \theta]
\]
Value-Based and Policy-Based RL

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

- **Policy Based**
  - No Value Function
  - Learn Policy

- **Actor-Critic**
  - Learn Value Function
  - Learn Policy
Advantages of Policy-Based RL

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Value/Q function may be much more complicated to represent than optimal policy
  - \( Q(s, \text{up}) = 0.9872, Q(s, \text{down}) = 0.5894 \). action: go up!
- Can learn stochastic policies
  - When is this important? When is this not important?

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance
Example: Rock-Paper-Scissors

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Consider policies for iterated rock-paper-scissors
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (i.e. Nash equilibrium)
**Example: Aliased Gridworld (1)**

- The agent cannot differentiate the grey states.
- Consider features of the following form (for all N, E, S, W)
  \[
  \phi(s, a) = 1(\text{wall to N, a = move E})
  \]
- Compare value-based RL, using an approximate value function
  \[
  Q_\theta(s, a) = f(\phi(s, a), \theta)
  \]
- To policy-based RL, using a parametrised policy
  \[
  \pi_\theta(s, a) = g(\phi(s, a), \theta)
  \]
Example: Aliased Gridworld (2)

- Under aliasing, an optimal **deterministic** policy will either
  - move W in both grey states (shown by red arrows)
  - move E in both grey states
- Either way, it can get stuck and never reach the money
- Value-based RL learns a near-deterministic policy
  - e.g. greedy or $\varepsilon$-greedy
- So it will traverse the corridor for a long time
An optimal **stochastic** policy will randomly move E or W in grey states

\[
\pi_0 (\text{wall to N and S, move E}) = 0.5 \\
\pi_0 (\text{wall to N and S, move W}) = 0.5
\]

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy
Goal: given policy $\pi_\theta(s, a)$ with parameters $\theta$, find best $\theta$

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

In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R_s^a$$

where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for $\pi_\theta$
Policy Optimisation

- Policy based reinforcement learning is an optimization problem
- Find $\theta$ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure
Table 1: Optimization methods in robotics: Properties of various optimization methods commonly used for optimization in robotics. As discussed in Section 2.1, the ideal optimizer for robotic applications should be global, zero-order, and assuming stochasticity.

(*) Extensions exist for the stochastic case, but they increase the number of experiments required.

<table>
<thead>
<tr>
<th>Method</th>
<th>Order optimizer</th>
<th>Stochasticity assumption</th>
<th>Global optimizer</th>
<th>Re-usability evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Search</td>
<td>Zero-order</td>
<td>No*</td>
<td>Global</td>
<td>Limited</td>
</tr>
<tr>
<td>Pure Random Search</td>
<td>Zero-order</td>
<td>No*</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td>Gradient-descent family</td>
<td>First-order</td>
<td>No*</td>
<td>Local</td>
<td>No</td>
</tr>
<tr>
<td>Bayesian Optimization</td>
<td>Zero-order</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td>Evolutionary Algorithms</td>
<td>Zero-order</td>
<td>No*</td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>Particle Swarm</td>
<td>Zero-order</td>
<td>No*</td>
<td>Global</td>
<td>No</td>
</tr>
</tbody>
</table>
Policy Gradient

- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters $\theta$

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

- Where $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \left( \frac{\partial J(\theta)}{\partial \theta_1} \atop \vdots \atop \frac{\partial J(\theta)}{\partial \theta_n} \right)$$

- and $\alpha$ is a step-size parameter
To evaluate policy gradient of $\pi_\theta(s, a)$

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

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

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

where $u_k$ is unit vector with 1 in $k$th component, 0 elsewhere

- Uses $n$ evaluations to compute policy gradient in $n$ dimensions
  - Scales linearly with number of parameters in policy!
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable
Training AIBO to Walk by Finite Difference Policy Gradient

- Goal: learn a fast AIBO walk (useful for Robocup)
- AIBO walk policy is controlled by 12 numbers (elliptical loci)
- Adapt these parameters by finite difference policy gradient
- Evaluate performance of policy by field traversal time

For more details, see paper: Kohl and Stone, ICRA 2004
AIBO Walk Policies

- Before training
- During training
- After training

Videos at: http://www.cs.utexas.edu/users/AustinVilla/?p=research/learned_walk
We now compute the policy gradient directly. Assume policy $\pi_\theta$ is differentiable whenever it is non-zero and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$. Likelihood ratios exploit the following identity:

$$\nabla_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} = \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)$$

The score function is $\nabla_\theta \log \pi_\theta(s, a)$.
We will use a softmax policy as a running example
Weight actions using linear combination of features $\phi(s, a)^T \theta$
Probability of action is proportional to exponentiated weight

$$\pi_\theta(s, a) \propto e^{\phi(s,a)^T \theta}$$

The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \phi(s, a) - E_{\pi_\theta}[\phi(s, \cdot)]$$
In continuous action spaces, a Gaussian policy is natural.
Mean is a linear combination of state features $\mu(s) = \phi(s) \theta$
Variance may be fixed $\sigma^2$, or can also parametrised
Policy is Gaussian, $a \sim N(\mu(s), \sigma^2)$
The score function is
\[
\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}
\]
Consider a simple class of one-step MDPs
- Starting in state $s \sim d(s)$
- Terminating after one time-step with reward $r = R_{s,a}$

Use likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r]$$
$$= \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \in S} \sum_{a \in A} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) R_{s,a}$$
$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a)r]$$
The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs.

- Replaces instantaneous reward $r$ with long-term value $Q^\pi(s, a)$.

- Policy gradient theorem applies to start state objective, average reward and average value objective.

**Theorem**

For any differentiable policy $\pi_\theta(s, a)$, for any of the policy objective functions $J = J_1, J_{avR},$ or $\frac{1}{1-\gamma}J_{avV}$, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) \ Q^{\pi_\theta}(s, a)]$$
See board derivation.
Benefit of Likelihood Ratio Approach:

- Number of samples need to approximate no longer depends on the # of policy parameters
- Gradient calculation is independent of the underlying system dynamics (ratio cancels): don’t need to know dynamics!
Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return $v_t$ as an unbiased sample of $Q^\pi_\theta(s_t, a_t)$

$$\Delta \theta_t = \alpha \nabla_{\theta} \log \pi_\theta(s_t, a_t)v_t$$

function REINFORCE
   Initialise $\theta$ arbitrarily
   for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
      for $t = 1$ to $T - 1$ do
         $\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_\theta(s_t, a_t)v_t$
      end for
   end for
   return $\theta$
end function
Puck World Example

- Continuous actions exert small force on puck
- Puck is rewarded for getting close to target
- Target location is reset every 30 seconds
- Policy is trained using variant of Monte-Carlo policy gradient
Monte-Carlo policy gradient still has high variance
We use a critic to estimate the action-value function,

\[ Q_w(s, a) \approx Q_{\pi_\theta}(s, a) \]

Actor-critic algorithms maintain two sets of parameters

- Critic Updates action-value function parameters \( w \)
- Actor Updates policy parameters \( \theta \), in direction suggested by critic

Actor-critic algorithms follow an approximate policy gradient

\[ \nabla_\theta J(\theta) \approx E_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) \, Q_w(s, a)] \]
\[ \Delta \theta = \alpha \nabla_\theta \log \pi_\theta(s, a) \, Q_w(s, a) \]
The critic is solving a familiar problem: policy evaluation
How good is policy $\pi_0$ for current parameters $\theta$?
Policy evaluation problem. See earlier lectures.
- Monte-Carlo policy evaluation
- Temporal-Difference learning

Could also use e.g. least-squares policy evaluation
Action-Value Actor-Critic

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s, a) = \phi(s, a)^T w$
  
  **Critic** Updates $w$ by linear TD(0)
  
  **Actor** Updates $\theta$ by policy gradient

```plaintext
function QAC
  Initialise $s, \theta$
  Sample $a \sim \pi_\theta$
  for each step do
    Sample reward $r = R_a s$; sample transition $s^0 \sim P_{a,s}$
    Sample action $a^0 \sim \pi_\theta(s^0, a^0)$
    $\delta = r + \gamma Q_w(s^0, a^0) - Q_w(s, a)$
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a^0, s \leftarrow s^0$
  end for
end function
```
Approximating the policy gradient introduces bias
A biased policy gradient may not find the right solution
  e.g. if $Q_w(s, a)$ uses aliased features, can we solve gridworld example?
Luckily, if we choose value function approximation carefully
Then we can avoid introducing any bias
i.e. We can still follow the exact policy gradient
Compatible Function Approximation

Theorem (Compatible Function Approximation Theorem)

If the following two conditions are satisfied:

1. Value function approximator is compatible to the policy

\[ \nabla_w Q_w(s, a) = \nabla_\theta \log \pi_\theta(s, a) \]

2. Value function parameters \( w \) minimise the mean-squared error

\[ \varepsilon = \mathbb{E}_{\pi_\theta} \left[ (Q_{\pi_\theta}(s, a) - Q_w(s, a))^2 \right] \]

Then the policy gradient is exact,

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)] \]
Proof of Compatible Function Approximation Theorem

If $w$ is chosen to minimise mean-squared error, gradient of $\varepsilon$ w.r.t. $w$ must be zero,

$$\nabla_w \varepsilon = 0$$

$$\mathbb{E}_{\pi_\theta} [(Q^\theta(s, a) - Q_w(s, a))\nabla_w Q_w(s, a)] = 0$$

$$\mathbb{E}_{\pi_\theta} [(Q^\theta(s, a) - Q_w(s, a))\nabla_\theta \log \pi_\theta(s, a)] = 0$$

$$\mathbb{E}_{\pi_\theta} [Q^\theta(s, a)\nabla_\theta \log \pi_\theta(s, a)] = \mathbb{E}_{\pi_\theta} [Q_w(s, a)\nabla_\theta \log \pi_\theta(s, a)]$$

So $Q_w(s, a)$ can be substituted directly into the policy gradient,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a)Q_w(s, a)]$$
Reducing Variance Using a Baseline

- We subtract a baseline function $B(s)$ from the policy gradient.
- This can reduce variance, without changing expectation:

$$
\mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) B(s)] = \sum_{s \in S} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) B(s)
$$

$$
= \sum_{s \in S} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a \in A} \pi_\theta(s, a)
$$

$$
= 0
$$

- A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$.
- So we can rewrite the policy gradient using the advantage function $A^{\pi_\theta}(s, a)$:

$$
A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)
$$

$$
\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]
$$
The advantage function can significantly reduce variance of policy gradient

So the critic should really estimate the advantage function

For example, by estimating both $V^{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s, a)$

Using two function approximators and two parameter vectors,

$$V_v(s) \approx V^{\pi_\theta}(s)$$
$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

$$A(s, a) = Q_w(s, a) - V_v(s)$$

And updating both value functions by e.g. TD learning
For the true value function $V^{\pi_\theta}(s)$, the TD error $\delta^{\pi_\theta}$
\[
\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s^0) - V^{\pi_\theta}(s)
\]
is an unbiased estimate of the advantage function
\[
E^{\pi_\theta}[\delta^{\pi_\theta} | s, a] = E^{\pi_\theta} \left[ r + \gamma V^{\pi_\theta}(s) | S^0, a \right] - V^{\pi_\theta}(s) \\
= Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \\
= A^{\pi_\theta}(s, a)
\]
So we can use the TD error to compute the policy gradient
\[
\nabla_\theta J(\theta) = E^{\pi_\theta} \left[ \nabla_\theta \log \pi_\theta(s, a) \delta^{\pi_\theta} \right]
\]
In practice we can use an approximate TD error
\[
\delta_v = r + \gamma V_v(s^0) - V_v(s)
\]
This approach only requires one set of critic parameters $v$
Alternative Policy Gradient Directions

- Gradient ascent algorithms can follow any ascent direction
- A good ascent direction can significantly speed convergence
- Also, a policy can often be reparametrised without changing action probabilities
- For example, increasing score of all actions in a softmax policy
- The vanilla gradient is sensitive to these reparametrisations
The natural policy gradient is parameterisation independent. It finds ascent direction that is closest to vanilla gradient, when changing policy by a small, fixed amount:

$$\nabla_{\theta}^{nat} \pi_{\theta}(s, a) = G_{\theta}^{-1} \nabla_{\theta} \pi_{\theta}(s, a)$$

where $G_{\theta}$ is the Fisher information matrix:

$$G_{\theta} = \mathbb{E}_{\pi_{\theta}} \left[ \nabla_{\theta} \log \pi_{\theta}(s, a) \nabla_{\theta} \log \pi_{\theta}(s, a)^T \right]$$
Natural Actor-Critic

- Using compatible function approximation,

\[ \nabla_w A_w(s, a) = \nabla_\theta \log \pi_\theta(s, a) \]

- So the natural policy gradient simplifies,

\[
\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)] \\
= \mathbb{E}_{\pi_\theta} \left[ \nabla_\theta \log \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)^T w \right] \\
= G_\theta w \\
\n\nabla^{nat}_\theta J(\theta) = w
\]

- i.e. update actor parameters in direction of critic parameters
Natural Actor Critic in Snake Domain

(a) Crank course

(b) Sensor setting
Natural Actor Critic in Snake Domain (2)

Figure 3: Behaviors of snake-like robot
Natural Actor Critic in Snake Domain (3)
The policy gradient has many equivalent forms

\[
\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) \ n_t] \\
= \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q^w(s, a)] \\
= \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^w(s, a)] \\
= \mathbb{E}_{\pi_\alpha} [\nabla_\theta \log \pi_\theta(s, a) \ \delta]
\]

\[G_\delta^{-1} \nabla_\theta J(\theta) = w\]

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate \( Q^\pi(s, a) \), \( A^\pi(s, a) \) or \( V^\pi(s) \)

Summary of Policy Gradient Algorithms
EB: Key Ideas

- Gradient approaches only guaranteed to find a local optima
- Finite-difference methods scale with # of parameters needed to represent the policy, but don’t require differentiable policy
- Likelihood ratio gradient approaches
  - Cost independent of # params
  - Don’t need to know dynamics model
  - Benefit from using a baseline to reduce variance
  - Natural gradient approaches are more robust
  - Be able to implement at least 1 gradient method which leverages info (from a critic / baseline )
Critics at Different Time-Scales

- Critic can estimate value function $V_\theta(s)$ from many targets at different time-scales. From last lecture...
  - For MC, the target is the return $v_t$
    $$\Delta \theta = \alpha (v_t - V_\theta(s)) \phi(s)$$
  - For TD(0), the target is the TD target $r + \gamma V(s^0)$
    $$\Delta \theta = \alpha (r + \gamma V(s^0) - V_\theta(s)) \phi(s)$$
  - For forward-view TD(\(\lambda\)), the target is the $\lambda$-return $v^\lambda_t$
    $$\Delta \theta = \alpha (v^\lambda_t - V_\theta(s)) \phi(s)$$
  - For backward-view TD(\(\lambda\)), we use eligibility traces
    $$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$
    $$e_t = \gamma \lambda e_{t-1} + \phi(s_t)$$
    $$\Delta \theta = \alpha \delta_t e_t$$
The policy gradient can also be estimated at many time-scales

\[ \nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)] \]

Monte-Carlo policy gradient uses error from complete return

\[ \Delta \theta = \alpha(v_t - V_v(s_t)) \nabla_\theta \log \pi_\theta(s_t, a_t) \]

Actor-critic policy gradient uses the one-step TD error

\[ \Delta \theta = \alpha(r + \gamma V_v(s_{t+1}) - V_v(s_t)) \nabla_\theta \log \pi_\theta(s_t, a_t) \]
Policy Gradient with Eligibility Traces

- Just like forward-view TD(\(\lambda\)), we can mix over time-scales
  \[
  \Delta \theta = \alpha (v^\lambda_t - V_v(s_t)) \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)
  \]
- where \(v^\lambda_t - V_v(s_t)\) is a biased estimate of advantage fn
- Like backward-view TD(\(\lambda\)), we can also use eligibility traces
  - By equivalence with TD(\(\lambda\)), substituting \(\phi(s) = \nabla_{\theta} \log \pi_{\theta}(s, a)\)
    \[
    \delta = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)
    \]
    \[
    e_{t+1} = \lambda e_t + \nabla_{\theta} \log \pi_{\theta}(s, a)
    \]
    \[
    \Delta \theta = \alpha \delta e_t
    \]
- This update can be applied online, to incomplete sequences