Policy Gradient Methods: Pathwise Derivative Methods and Wrap-up

March 15, 2017
Pathwise Derivative Policy Gradient Methods
Policy Gradient Estimators: Review
Want to compute $\nabla_\theta \mathbb{E}[R_T]$. We’ll use $\nabla_\theta \log \pi(a_t | s_t; \theta)$

Reparameterize: $a_t = \pi(s_t, z_t; \theta)$. $z_t$ is noise from fixed distribution.

Only works if $P(s_2 | s_1, a_1)$ is known.
Deriving the Policy Gradient, Reparameterized

- Episodic MDP:

![Diagram of an episodic MDP with states and actions connected by edges, ending in a terminal node labeled \( R_T \).]

Want to compute \( \nabla_\theta \mathbb{E}[R_T] \). We’ll use \( \nabla_\theta \log \pi(a_t \mid s_t; \theta) \).

- Reparameterize: \( a_t = \pi(s_t, z_t; \theta) \). \( z_t \) is noise from fixed distribution.

![Diagram showing reparameterization with \( z_t \) added as a new node, connected to \( s_t \).]

- Only works if \( P(s_2 \mid s_1, a_1) \) is known ⊹
Deriving the Policy Gradient, Reparameterized

- Episodic MDP:

Want to compute $\nabla_{\theta} \mathbb{E}[R_T]$. We’ll use $\nabla_{\theta} \log \pi(a_t | s_t; \theta)$

- Reparameterize: $a_t = \pi(s_t, z_t; \theta)$. $z_t$ is noise from fixed distribution.

- Only works if $P(s_2 | s_1, a_1)$ is known
Using a $Q$-function

\[ \frac{d}{d\theta} \mathbb{E} [R_T] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{da_t} \mathbb{E} [R_T | a_t] \frac{da_t}{d\theta} \right] \]

\[ = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{dQ(s_t, a_t)}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{d\theta} Q(s_t, \pi(s_t, z_t; \theta)) \right] \]
SVG(0) Algorithm

- Learn $Q_\phi$ to approximate $Q^{\pi,\gamma}$, and use it to compute gradient estimates.

SVG(0) Algorithm

- Learn $Q_\phi$ to approximate $Q^{\pi, \gamma}$, and use it to compute gradient estimates.
- Pseudocode:
  
  ```plaintext
  for iteration=1, 2, \ldots \ do
    Execute policy $\pi_\theta$ to collect $T$ timesteps of data
    Update $\pi_\theta$ using $g \propto \nabla_\theta \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$
    Update $Q_\phi$ using $g \propto \nabla_\phi \sum_{t=1}^{T} (Q_\phi(s_t, a_t) - \hat{Q}_t)^2$, e.g. with TD($\lambda$)
  end for
  ```

SVG(1) Algorithm

- Instead of learning $Q$, we learn
  - State-value function $V \approx V^\pi, \gamma$
  - Dynamics model $f$, approximating $s_{t+1} = f(s_t, a_t) + \zeta_t$

- Given transition $(s_t, a_t, s_{t+1})$, infer $\zeta_t = s_{t+1} - f(s_t, a_t)$

- $Q(s_t, a_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})] = \mathbb{E}[r_t + \gamma V(f(s_t, a_t) + \zeta_t)]$, and $a_t = \pi(s_t, \theta, \zeta_t)$
SVG(∞) Algorithm

- Just learn dynamics model $f$
- Given whole trajectory, infer all noise variables
- Freeze all policy and dynamics noise, differentiate through entire deterministic computation graph
SVG Results

- Applied to 2D robotics tasks

- Overall: different gradient estimators behave similarly

---

Deterministic Policy Gradient

- For Gaussian actions, variance of score function policy gradient estimator goes to infinity as variance goes to zero
  - Intuition: finite difference gradient estimators
- But SVG(0) gradient is fine when $\sigma \to 0$
  \[ \nabla_\theta \sum_t Q(s_t, \pi(s_t, \theta, \zeta_t)) \]
- Problem: there’s no exploration.
- Solution: add noise to the policy, but estimate $Q$ with TD(0), so it’s valid off-policy
- Policy gradient is a little biased (even with $Q = Q^\pi$), but only because state distribution is off—it gets the right gradient at every state

Deep Deterministic Policy Gradient

- Incorporate replay buffer and target network ideas from DQN for increased stability
- Use lagged (Polyak-averaging) version of $Q_\phi$ and $\pi_\theta$ for fitting $Q_\phi$ (towards $Q^{\pi,\gamma}$) with TD(0)

$$\hat{Q}_t = r_t + \gamma Q_\phi'(s_{t+1}, \pi(s_{t+1}; \theta'))$$

- Pseudocode:

```python
for iteration=1, 2, ... do
    Act for several timesteps, add data to replay buffer
    Sample minibatch
    Update $\pi_\theta$ using $g \propto \nabla_\theta \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$
    Update $Q_\phi$ using $g \propto \nabla_\phi \sum_{t=1}^{T} (Q_\phi(s_t, a_t) - \hat{Q}_t)^2$
end for
```

DDPG Results

Applied to 2D and 3D robotics tasks and driving with pixel input

Policy Gradient Methods: Comparison

- Two kinds of policy gradient estimator
  - REINFORCE / score function estimator: $\nabla \log \pi(a \mid s) \hat{A}$.
    - Learn $Q$ or $V$ for variance reduction, to estimate $\hat{A}$
  - Pathwise derivative estimators (differentiate wrt action)
    - SVG(0) / DPG: $\frac{d}{da} Q(s, a)$ (learn $Q$)
    - SVG(1): $\frac{d}{da} (r + \gamma V(s'))$ (learn $f, V$)
    - SVG($\infty$): $\frac{d}{da_t} (r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots)$ (learn $f$)

- Pathwise derivative methods more sample-efficient when they work (maybe), but work less generally due to high bias
Policy Gradient Methods vs Q-Function Regression Methods

- Q-function regression methods are more sample-efficient when they work, but don’t work as generally
- Policy gradients are easier to debug and understand
  - Don’t have to deal with “burn-in” period
  - When it’s working, performance should be monotonically increasing
  - Diagnostics like KL, entropy, baseline’s explained variance
- Q-function regression methods are more compatible with exploration and off-policy learning
- Policy-gradient methods are more compatible with recurrent policies
- Q-function regression methods CAN be used with continuous action spaces (e.g., S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. “Continuous deep Q-learning with model-based acceleration”. (2016)) but final performance is worse (so far)
Recent Papers on Connecting Policy Gradients and Q-function Regression


