Policy Gradients for Reinforcement Learning

Keith L. Downing

The Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
keithd@ntnu.no

January 5, 2021
Error Gradients -vs- Policy Gradients

- **Goal:** minimize error by changing weights
- **Goal:** maximize performance measure by changing weights
Why Bother?

- Sometimes it’s easier to train the actor directly than to train a value function. The policy may be a simpler function to approximate than a state-value or action-value function.
- MCTS (for training actors) is computationally demanding.
- Easily modified to handle continuous action spaces.
- User bias is often easier to achieve in a policy network than a value network...for domains where you want to bias behavior.
- Stochastic policies (when desired, e.g. for stochastic games) evolve more naturally, since NN learns a probability distribution over actions -vs- training NN to individually learn each $Q(s, a)$ value.

Design Problems to Solve

- What is the performance measure, $J(\theta)$? $\forall i : w_i \in \theta$
- How to compute the policy gradient $\equiv \nabla_\theta J(\theta)$
Policy Gradient Theorem

- Performance measure = value of start state:
  \[ J(\theta) \equiv V_\pi(s_0) \]
  = Total expected reward from start state to final state.

- The policy gradient theorem shows that:
  \[ \nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a Q_\pi(s, a) \nabla_\theta \Pi(a \mid s, \theta) \]

where:
- \( \mu(s) \) = frequency of state \( s \) visits under policy \( \Pi \)
- \( Q_\pi(s, a) \) = action-value function when following policy \( \Pi \)
- \( \nabla_\theta \Pi(a \mid s, \theta) \) = deriv of the policy function w.r.t. its parameters \( \theta \), evaluated at \( (s, a) \).
The effect of the function parameters (e.g. NN wgts) on performance is proportional to:

- their effect on the output action probabilities
- weighted by the values of those actions, and
- averaged over all state-action combinations, weighted by the probabilities of being in each state.

An influential parameter, $w \in \theta$, is one that, for input state $s$, creates high (magnitude) gradients for the output corresponding to action $a$ when $Q(s,a)$ also has a high magnitude and state $s$ occurs frequently.
Using the Policy Gradient Theorem (PGT)

Stochastic Gradient **Ascent** (goal: performance ↑)

\[ \forall w \in \theta : \triangle w = \alpha \nabla_w J(\theta) \]

where \( \alpha = \) learning rate.

- Need to insure that our samples (of states and actions) during RL search yield expected values for the gradients that are proportional to those given by PGT.
- As long as sampling is governed by the policy, \( \Pi \), then since PGT is also based on \( \Pi \), our gradient estimates (based on samples) should be true to the gradients of PGT.
- Thus, we should be able to define quantities that can be calculated on each timestep, based on the most recent \((s,a)\), that give gradients proportional to those of PGT \( \rightarrow \) we can gradually optimize performance, \( J(\theta) \).
Sources of Policy-Gradient Data

- $\Pi(a | s, \theta) =$ outputs of the actor network
- $\nabla_\theta \Pi(a | s, \theta)$ - gradients of outputs w.r.t. weights
- $Q(s,a)$ - action values approximated by another network and/or using the latest return, $G_t$
Goal: Design an RL algorithm so that changes to parameters ($\theta$) at each step are true to the gradients given by PGT.

- Assuming policy $\Pi$ is used for sampling during RL search, the distribution of state visits should match the $\mu(s)$ from PGT. Thus, we can rewrite PGT in terms of an expected value across our samples:

$$\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a Q_\pi(s, a) \nabla_\theta \Pi(a \mid s, \theta)$$

$$= E_\pi [\sum_a Q_\pi(S_t, a) \nabla_\theta \Pi(a \mid S_t, \theta)]$$

- where $s$, $\mu(s)$, and $\sum_s$ are replaced with $S_t$, a variable representing a sampled state. This simplification is legal as long as $S_t$ is drawn from the same state distribution that is expected under $\Pi$.

- We would like to do the same with $a$ (the action) but cannot do so directly: we can’t just replace $a$ with $A_t$ and remove $\sum_a$. 
However, we could stop here and use this as our parameter update rule:

$$\Delta \theta = \alpha \sum_a \hat{Q}(S_t, a) \nabla_{\theta} \Pi(a \mid S_t, \theta)$$

where $\hat{Q}$ = a learned approximation to $Q_\pi$.

But ideally, we’d like an update rule based purely on the most recent state ($S_t$) and action ($A_t$).

We need to sneak $\Pi(a \mid S_t, \theta)$ inside the $\sum_a$. We can do that by multiplying and dividing by it:

$$= E_\pi \left[ \sum_a \Pi(a \mid S_t, \theta) Q_\pi(S_t, a) \frac{\nabla_{\theta} \Pi(a \mid S_t, \theta)}{\Pi(a \mid S_t, \theta)} \right]$$

Now, the expected value is based on the action-choice probabilities given by the policy, not on a flat average over all actions.

So we can substitute in a sampled action $A_t$, since that sampling will also be governed by $\Pi$, and thus our distribution of (s,a) visits will be governed by $\Pi$. 
Without altering the equality, we can now remove the $\Sigma_a \Pi(a \mid S_t, \theta)$, and replace $a$ with the sampled action, $A_t$:

$$= E_\pi \left[ Q_\pi(S_t, A_t) \frac{\nabla_\theta \Pi(A_t \mid S_t, \theta)}{\Pi(A_t \mid S_t, \theta)} \right]$$

Since, from calculus, $\frac{\nabla_\theta f(\theta)}{f(\theta)} = \nabla_\theta \ln(f(\theta))$, we simplify to:

$$= E_\pi \left[ Q_\pi(S_t, A_t) \times \nabla_\theta \ln(\Pi(A_t \mid S_t, \theta)) \right]$$

Finally, during sampling: $E[G_t \mid S_t, A_t] = Q_\pi(S_t, A_t)$, where $G_t$ = our normal return:

$$\nabla_\theta J(\theta) \propto E_\pi \left[ G_t \times \nabla_\theta \ln(\Pi(A_t \mid S_t, \theta)) \right]$$
The gradient of our sampling-based estimate of performance (true to PGT) is:

$$\nabla_\theta J(\theta) \propto E_\pi [G_t \nabla_\theta \ln(\Pi(A_t | S_t, \theta))]$$

And our parameter (e.g. weight) updates at each step of RL are then:

$$\Delta \theta = \alpha \nabla_\theta J(\theta) = \alpha G_t \nabla_\theta \ln(\Pi(A_t | S_t, \theta))$$

Intuitively...

$$\Delta \theta = \alpha G_t \frac{\nabla_\theta \Pi(A_t | S_t, \theta)}{\Pi(A_t | S_t, \theta)}$$

The updates of each parameter $w \in \theta$ are:

- directly proportional to the return, and
- directly proportional to the relative effect of $w$ upon the output probability corresponding to the action actually taken, $A_t$. This is relative, since it's scaled by the output probability itself.
The REINFORCE Algorithm

- A popular implementation of direct policy learning based on the Policy Gradient Theorem (PGT).
- A Monte-Carlo, Policy-Gradient, RL method.

Do one episode: $S_0, A_0, R_1, S_1, A_1, ..., R_T, S_T$ using $\Pi_\theta$.

For each step $t \in 0, 1, ..., T - 1$:

$G_t$ is the return from step $t$ (to $T$)

$\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \ln(\Pi(A_t | S_t, \theta))$

Use of discounting ($\gamma^t$) appears in some versions of the algorithm for theoretical completeness, but often $\gamma$ is assumed to be 1. See Sutton and Barto (2nd ed., pg. 328)
Policy Gradient Theorem (PGT) can be generalized to include an arbitrary baseline function of s, \( b(s) \), as long as \( b(s) \) is independent of the action \( a \).

This does not affect the average of \( \triangle \theta \) but normally reduces its variance → faster convergence.

\[
\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a (Q_\pi(s,a) - b(s)) \nabla_\theta \Pi(a \mid s, \theta)
\]

This has no effect on \( \nabla_\theta J(\theta) \), since:

\[
\sum_a b(s) \nabla_\theta \Pi(a \mid s, \theta) = b(s) \nabla_\theta \sum_a \Pi(a \mid s, \theta) = b(s) \nabla_\theta 1 = 0
\]

A natural choice for \( b(s) \) is \( V(s) \). Then \( Q(s,a) - V(s) \) indicates how much better (or worse) \( a \) is than the other possible actions from \( s \).
REINFORCE Algorithm with Baseline

- Episodic Monte-Carlo with policy $\Pi(a \mid s, \theta)$ and state-value function $V(s \mid \Phi)$. Both typically NNs.
- Parameters $\theta$ and $\Phi$ learned, using rates $\alpha_\theta$ and $\alpha_\Phi$, respectively. Only the $\theta$ updates involve discounting: $\gamma^t$.

Do one episode: $S_0, A_0, R_1, S_1, A_1 \ldots R_T, S_T$ using $\Pi_\theta$.

For each step $t \in 0, 1, \ldots, T - 1$:

- $G_t$ is the return from step $t$ (to $T$)
- $\delta = G_t - V(S_t \mid \Phi)$
- $\Phi \leftarrow \Phi + \alpha_\Phi \delta \nabla_\Phi V(S_t, \Phi)$
- $\theta \leftarrow \theta + \alpha_\theta \gamma^t \delta \nabla_\theta \ln(\Pi(A_t \mid S_t, \theta))$

- Note that $V$ is not used for bootstrapping, just as a baseline.
- MC (episodic) nature makes this slower to converge.
- Use TD actor-critic model to speed it up, and open up for continuous (non-episodic) tasks.
Now we update after every move in an episode.

Replace $G_t$ with $G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1} \mid \Phi)$

$\delta$ is now our TD error, and since we already showed that the baseline $V(s)$ does not violate PGT, we are still in good shape w.r.t. the underlying theory.

Repeat Forever:

- $S \leftarrow$ episode start state
- $t \leftarrow 0$
- While $S$ is not a final state:
  - Choose $A$ based on policy $\Pi(a \mid S, \theta)$
  - Do action $A$ from state $S \mapsto$ state $S'$ and reward $R$.
  - $\delta \leftarrow R + \gamma V(S' \mid \Phi) - V(S \mid \Phi)$
  - $\Phi \leftarrow \Phi + \alpha_\Phi \delta \nabla_\Phi V(S \mid \Phi)$
  - $\theta \leftarrow \theta + \alpha_\theta \gamma^t \delta \nabla_\theta \ln(\Pi(A \mid S, \theta))$
  - $S \leftarrow S'$; $t \leftarrow t + 1$
Now we can generalize to TD(\(\lambda\)), thus handling multi-step backups. Use a different backup factor for \(\theta\) and \(\Phi\): \(\lambda_\theta\) and \(\lambda_\Phi\).

Use eligibility traces \(e\) for each parameter set: \(e_\theta\) and \(e_\Phi\).

Repeat Forever:

\[ S \leftarrow \text{episode start state} \]
\[ t \leftarrow 0 ; e_\theta \leftarrow 0 ; e_\Phi \leftarrow 0 \]

While \(S\) is not a final state:

Choose A based on policy \(\Pi(a \mid S, \theta)\)
Do action A in \(S \mapsto R, S'\)
\[ \delta \leftarrow R + \gamma V(S' \mid \Phi) - V(S \mid \Phi) \]
\[ e_\theta \leftarrow \gamma \lambda_\theta e_\theta + \gamma^t \nabla_\theta \ln(\Pi(A \mid S, \theta)) \]
\[ e_\Phi \leftarrow \gamma \lambda_\Phi e_\Phi + \nabla_\Phi V(S \mid \Phi) \]
\[ \theta \leftarrow \theta + \alpha_\theta \delta e_\theta \]
\[ \Phi \leftarrow \Phi + \alpha_\Phi \delta e_\Phi \]
\[ S \leftarrow S' ; t \leftarrow t + 1 \]
The Policy Gradient Theorem can also be proven for continuing cases, so we can extend REINFORCE as well.

- S ← a global start state.
- $e_\theta \leftarrow 0$; $e_\Phi \leftarrow 0$; $R^* \leftarrow 0$.
- Repeat Forever:
  - Choose A based on policy $\Pi(a | S, \theta)$
  - Do action A in $S \mapsto R, S'$
  - $R^* \leftarrow f(R^*, R)$
  - $\delta \leftarrow (R - R^*) + \gamma V(S' | \Phi) - V(S | \Phi)$
  - $e_\theta \leftarrow \gamma \lambda_\theta e_\theta + \nabla_\theta \ln(\Pi(A | S, \theta))$
  - $e_\Phi \leftarrow \gamma \lambda_\Phi e_\Phi + \nabla_\Phi V(S | \Phi)$
  - $\theta \leftarrow \theta + \alpha_\theta \delta e_\theta$
  - $\Phi \leftarrow \Phi + \alpha_\Phi \delta e_\Phi$
  - $S \leftarrow S'$

$R^* \propto$ running average of R. Note: No exponential discounting: $\gamma^t$. 