Nightmares of Policy Optimization

Sanjiban Choudhury
WHAT MAKES IT HARDER THAN REINFORCEMENT LEARNING THAN SUPERVISED LEARNING?
Bellman is Beautiful …

\[ V^*(s) \rightarrow \text{max}_{a} r(s, a) + V^*(s') \]
But errors in Bellman compound!!!

\[ V_\theta(s) \]

\[ \max_a r(s, a) + V_\theta(s') \]

\[ V_\theta(s) \]

\[ \max_a r(s, a) + V_\theta(s') \]
The problem of distribution shift

Upper half of state is BAD

Lower half of state is GOOD

---

Approximated Q

True Q

---

Figure 8.2: Value function overestimation in value iteration. Bellman backups proceed. Figure 8.2.3 shows an illustration of this effect. Because the upper half of the state space (which is bad) is overestimated by the function approximator, policies switch to direct probability mass towards that state by choosing actions that make arriving at these states more likely. Error in overestimation of the value function has a cascading effect as we iterate backwards in time. We further noted that the pure policy evaluation variant of dynamic programming is much more stable– without the max to drive behavior towards states with high value estimates we are less subject to the amplification of errors. However, on the surface it seems that we've merely pushed the problem into the policy improvement step. That is, while the estimation of the action-value function for a current policy becomes stable, the improvement step would instead drive probability mass towards states-actions that tend to be over-estimates of quality, leading to instability between iterations of any approximate policy iteration procedure.

This objection is, in fact, well-founded and approximate policy iteration algorithms aren't noted to be more stable or effective than approximate value iteration counterparts. However, the maintenance of an explicit policy opens up a new possibility: the ability to manage or mitigate the distribution shift that occurs when we update the policy.

Conservativity and Trust Regions

A broad class of algorithms, initiated by the seminal development of Conservative Policy Iteration (CPI) S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In Proceedings of the 19th International Conference on Machine Learning (ICML), 2002 constrain modification to the current policy S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In Proceedings of the 19th International Conference on Machine Learning (ICML), 2002 to prevent the state-action distribution from changing too radically between iterations and thus ensure errors don't explode. The result is algorithms that are stable and effective, although they can be slower than raw policy iteration. CPI modifies the policy update step to stochastically mix between policies p_{new} = \alpha p_{new \ greedy} + (1 - \alpha) p_{old}, where the mixing weight \alpha is interpreted as the probability of choosing that component. Careful analysis in S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In Proceedings of the 19th International Conference on Machine Learning (ICML), 2002 ensures a strategy for choosing \alpha that ensures improvement, while in practice a simple line-search strategy can be employed to ensure monotonic improvement.

This is a somewhat impractical algorithm as it can take many steps and requires maintenance of a mixture of a number of policies equal to the number of update steps. Later approaches, including No-Regret Policy Iteration S. Ross and J. A. Bagnell. Reinforcement and imitation learning via interactive no-regret learning. arXiv preprint arXiv:1406.5979, 2014 and the Natural or Covariant Policy Search Approach S. Kakade, A. Y. Ng, S. Kakade, and J. Schneider. Policy search by dynamic programming. In Advances in Neural Information Processing Systems, 2003; and J. A. Bagnell, A. Y. Ng, S. Kakade, and J. Schneider. Policy search by dynamic programming. In Advances in Neural Information Processing Systems, 2003 (and later implementations of these like “Trust Region Policy Optimization”21) manage to keep one policy, albeit a typically stochastic one, but keep the same intuition of a controlled policy change through the stability of no-regret learning, line
The problem of distribution shift

Upper half of state is BAD

Lower half of state is GOOD

---

Approximated Q
True Q
The problem of distribution shift

Upper half of state is BAD

Lower half of state is GOOD

---

Approximated Q

True Q
Compounding Errors in Mountain Car

Car-on-the-Hill

\[ V^*(x, y) \]
What happens when we run value iteration with a 2 Layer MLP?

Iteration 11

![Graph showing a 3D plot with labeled axes and values]
What happens when we run value iteration with a 2 Layer MLP?

Example: car on hill

Figure 8.1.4 shows the car-on-hill example.

Car-on-the-Hill $J^*(\text{pos}, \text{vel})$

Figure 8.1.5 shows that a two layer MLP can also diverge to underestimate the costs.

Iteration 101

Training with neural network.
What happens when we run value iteration with a 2 Layer MLP?

**Iteration 201**

![Diagram](image)
To hell with Value Estimates!

Trust ONLY actual Returns
Bye Bye Bellman ...

“not to be blinded by the beauty of the Bellman equation”

- Andrew Moore
What if we focused on finding good policies …?
Sometimes a policy is waaaaaay simpler than the value
Can we just focus on finding a good policy?

\[ \pi_\theta : S_t \rightarrow A_t \]

Learn a mapping from states to actions

Roll-out policies in the real-world to estimate value
The Game of Tetris
What’s a good policy representation for Tetris?

- States: Board configuration (each of $k$ cells can be filled/not filled), current piece (there are 7 pieces total). In this implementation, there are therefore approximately $2^k \times 7$ states. Note: not all configurations are valid, for example, there cannot be a piece floating in the air. This resulting in a smaller number of total valid states.
- Actions: A policy can select any of the columns and from up to 4 possible orientations for a total of about 40 actions (some orientation and column combinations are not valid for every piece).
- Transition Matrix: A deterministic update of the board plus the selection of a random piece for the next time-step.
- Cost Function: There are several options to choose from, including: reward $= +1$ for each line removed, $0$ otherwise; # of free rows at the top; $+1$ for not losing that round; etc.

Deterministic and Non-Deterministic MDP Algorithms

For Deterministic MDPs the transition model is deterministic or, equivalently, we know with certainty what the next state $x_{t+1}$ will be given the current state $x_t$ and the action $a_t$. Solving deterministic MDPs.

$$ \pi_\theta : s_t \rightarrow a_t $$

(4 rotations) * (10 slots) - (6 impossible poses) = 34

State ($s_t$) Action ($a_t$)
Activity!
Think-Pair-Share

Think (30 sec): Ideas for how to represent policy for tetris?

Pair: Find a partner

Share (45 sec): Partners exchange ideas
Some inspiration for Tetris policy

Until 2008, the best artificial Tetris player was handcrafted, as reported by Fahey (2003). Pierre Dellacherie, a self declared average Tetris player, identified six simple features and tuned the weights by trial and error.
Dellacherie Features

Landing Heights
Eroded Cells
Row Transitions
Column Transitions
Holes
Cumulative Wells

The contribution of the last piece to the cleared lines time the number of cleared lines.

The number of filled cells adjacent to the empty cells summed over all rows.

A well is a succession of empty cells and the cells to the left and right are occupied.
A *magic* formula?!?

$-4 \times \text{holes} - \text{cumulative wells} - \text{row transitions} - \text{column transitions} - \text{landing height} + \text{eroded cells}$
A magic formula ?!? 

- $4 \times$ holes – cumulative wells
- row transitions – column transitions
- landing height + eroded cells

This linear evaluation function cleared an average of 660,000 lines on the full grid ...

... In the simplified implementation used by the approaches discussed earlier, the games would have continued further, until every placement would overflow the grid. Therefore, this report underrates this simple linear rule compared to other algorithms.
Tetris Policy

\[
\pi_\theta(a|s) = \frac{\exp \left( \theta^\top f(s, a) \right)}{\sum_{a'} \exp \left( \theta^\top f(s, a') \right)}
\]

\[
f_1(s, a) = \# \text{ number of holes}
\]

\[
f_2(s, a) = \# \text{ max height}
\]
The Goal of Policy Optimization

\[ \pi_\theta(a|s) = \frac{\exp \left( \theta^T f(s, a) \right)}{\sum_{a'} \exp \left( \theta^T f(s, a') \right)} \]

Think of \( f(s, a) \) being dellacherie features

\[ \min_{\theta} J(\theta) = \sum_{t=0}^{T-1} \mathbb{E}_{\pi_\theta} c(s_t, a_t) \]

Think of \( c(s, a) \) as -num_rows_cleared
Can we do gradient descent if we don’t know the dynamics??
The Likelihood Ratio Trick!
Algorithm 20: The REINFORCE algorithm.

Start with an arbitrary initial policy $\pi_{\theta}$

while not converged do

Run simulator with $\pi_{\theta}$ to collect $\{\xi^{(i)}\}_{i=1}^{N}$

Compute estimated gradient

$$\tilde{\nabla}_{\theta} J = \frac{1}{N} \sum_{i=1}^{N} \left[ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta} \left( a^{(i)}_t | s^{(i)}_t \right) R(\xi^{(i)}) \right]$$

Update parameters $\theta \leftarrow \theta + \alpha \tilde{\nabla}_{\theta} J$

return $\pi_{\theta}$
Chugging through the gradient ..

\[ \nabla_\theta \log \pi_\theta(a|s) = \nabla_\theta \left[ \theta^\top f(s,a) - \log \sum_{a'} \exp \left( \theta^\top f(s,a') \right) \right] \]

\[ = f(s,a) - \frac{\sum_{a'} f(s,a') \exp \left( \theta^\top f(s,a') \right)}{\sum_{a'} \exp \left( \theta^\top f(s,a') \right)} \]

\[ = f(s,a) - \sum_{a'} f(s,a') \pi_\theta(a'|s) \]

\[ = f(s,a) - E_{\pi_\theta(a'|s)} \left[ f(s,a') \right] \]
Understanding the REINFORCE update

Let \( f_1(s, a) = \# \text{holes} \).

\[
\begin{align*}
\text{Let } f_1(s, a) &= \# \text{holes}. \\
R = +1 &\Rightarrow \sum_{a' \in \mathcal{A}} f_1(s, a) - \frac{\mathbb{E}_{a' \sim \pi_0} f_1(s, a')}{\mathbb{E}_{a' \sim \pi_0}} = -5 \\
R = -1 &\Rightarrow \sum_{a' \in \mathcal{A}} f_1(s, a) - \frac{\mathbb{E}_{a' \sim \pi_0} f_1(s, a')}{{\mathbb{E}_{a' \sim \pi_0}}} = +3
\end{align*}
\]

\[
\begin{align*}
\Theta_1 &= \Theta_0 + \sum_{s \in \mathcal{S}} \left( \nabla_\theta \log \pi_0(a|s) \right) R(s) = \Theta_0 + \left[\begin{array}{c}
-5 \times (+1) \\
+3 \times (-1)
\end{array}\right] \\
&= \Theta_0 - \alpha 8 \quad \text{(Bump down this feature)}
\end{align*}
\]
Causality: Can actions affect the past?
The Policy Gradient Theorem

\[ \nabla_{\theta} J = E_p(\xi|\theta) \left[ \sum_{t=0}^{T-1} \left( \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \left( \sum_{t'=0}^{t-1} r(s_{t'},a_{t'}) + \sum_{t'=t}^{T-1} r(s_{t'},a_{t'}) \right) \right) \right] \]

\[ = E_p(\xi|\theta) \left[ \sum_{t=0}^{T-1} \left( \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \sum_{t'=t}^{T-1} r(s_{t'},a_{t'}) \right) \right] \]

\[ \nabla_{\theta} J = E_p(\xi|\theta) \left[ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) Q^{\pi_{\theta}}(s_t,a_t) \right] \]
Life is good!

This solves everything ...
The Three Nightmares of Policy Optimization
Nightmare 1:

Local Optima
The Ring of Fire

+1

+100

-10
The Ring of Fire
The Ring of Fire

Get's sucked into a local optima!!
Idea: What if we had a “good reset distribution?”

Nominal reset distribution
Idea: What if we had a “good reset distribution?”

Augmented reset distribution
Idea: What if we had a “good reset distribution?”

Run REINFORCE from different start states
Idea: What if we had a “good reset distribution?”

Run REINFORCE from different start states
Idea: What if we had a “good reset distribution?”

Run REINFORCE from different start states.
Solution: Use a good “restart” distribution

Choose a restart distribution $\mu(s)$ instead of start state distribution $\mu(s)$

Try your best to “cover” states the expert will visit

Suffer at most a penalty of $\| \frac{d_{\pi^*}}{\mu} \|_\infty$
Nightmare 2:

Distribution Shift
Is gradient descent the best direction?

$$\nabla_\theta J = E_{p(\xi|\theta)} \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t) \right]$$

Note all the terms in the above equation that depend on theta. If we change theta by a small amount, how do these terms change?
What would gradient descent do here?

What assumption does it make that is breaking?
How can we make it choose a better direction?
Gradient Descent as Steepest Descent

Gradient Descent is simply Steepest Descent with L2 norm

$$\min_{\Delta \theta} J(\theta + \Delta \theta) \quad \text{s.t.} \quad ||\Delta \theta|| \leq \epsilon \quad \rightarrow \quad \Delta \theta = -\nabla_{\theta} J(\theta)$$
Steepest Descent with a different norm

A different norm $G$ means a different notion of “small step”

$$
\min_{\Delta \theta} J(\theta + \Delta \theta) \quad \text{s.t.} \quad \Delta \theta^T G \Delta \theta \leq \epsilon \quad \rightarrow \quad \Delta \theta = - G^{-1} \nabla_\theta J(\theta)
$$
What is the best norm for policy gradient?

\[
\nabla_\theta J = E_{p(\xi|\theta)} \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t,a_t) \right]
\]

Don’t make small changes in \( \theta \), make small changes in the “distribution \( \pi_\theta(a|s) \)”

\[
\min_{\Delta \theta} J(\theta + \Delta \theta) \quad \text{s.t.} \quad KL(\pi_{(\theta+\Delta \theta)} \| \pi_\theta) \leq \epsilon
\]
"Natural" Gradient Descent

Start with an arbitrary initial policy \( \pi_\theta \)

**while not converged do**

Run simulator with \( \pi_\theta \) to collect \{\( \xi^{(i)} \)\}_{i=1}^N

Compute estimated gradient

\[
\tilde{\nabla}_\theta J = \frac{1}{N} \sum_{i=1}^N \left[ \left( \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \left( a_i^{(i)} | s_i^{(i)} \right) \right) R(\xi^{(i)}) \right]
\]

\[
\tilde{G}(\theta) = \frac{1}{N} \sum_{i=1}^N \left[ \nabla_\theta \log \pi_\theta(a_i | s_i) \nabla_\theta \log \pi_\theta(a_i | s_i)^T \right]
\]

Update parameters \( \theta \leftarrow \theta + \alpha \tilde{G}^{-1}(\theta) \tilde{\nabla}_\theta J \).

return \( \pi_\theta \)

Modern variants are TRPO, PPO, etc
But does this work on *real robots*?
Initially, we teach a rudimentary stroke by supervised learning as can be seen in Figure 3 (b); however, it fails to reproduce the behavior as shown in (c); subsequently, we improve the performance using the episodic Natural Actor-Critic which yields the performance shown in (a) and the behavior in (d). After approximately 200-300 trials, the ball can be hit properly by the robot.
Nightmare 3: High Variance
The Policy Gradient Theorem

\[
\nabla_{\theta} J = E_{p(\xi|\theta)} \left[ \sum_{t=0}^{T-1} \left( \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \left( \sum_{r=0}^{t-1} r(s_r, a_r) + \sum_{r=t}^{T-1} r(s_r, a_r) \right) \right) \right]
\]

\[
= E_{p(\xi|\theta)} \left[ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \sum_{r=t}^{T-1} r(s_r, a_r) \right],
\]

\[
\nabla_{\theta} J = E_{p(\xi|\theta)} \left[ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) Q^{\pi_{\theta}}(s_t, a_t) \right]
\]

1. Local Optima: Use Exploration Distribution
2. Distribution Shift: *Natural* Gradient Descent
3. High Variance: Subtract baseline