Reinforcement Learning as Variational Inference: Two Recent Approaches

Rohith Kuditipudi

Duke University

11 August 2017
Outline

1. Background
2. Stein Variational Policy Gradient
3. Soft Q-Learning
4. Closing Thoughts/Takeaways
Background: Reinforcement Learning

- A Markov Decision Process (MDP) is a tuple \((S, A, p, r, \gamma)\), where:
  - \(S\) is the state space
  - \(A\) is the action space
  - \(p(s_{t+1}|s_t, a_t)\) is the probability of the next state \(s_{t+1} \in S\) given the current state \(s_t \in S\) and action \(a_t \in A\) taken by the agent
  - \(r : S \times A \rightarrow [r_{\text{min}}, r_{\text{max}}]\) is the reward function
  - \(\gamma \in [0, 1]\) is the discount factor

- A policy \(\pi(\cdot|s_t)\) is a distribution over \(A\) conditioned on the current state \(s_t\)

- The goal of reinforcement learning is to learn a policy \(\pi\) that maximizes the total expected (discounted) reward \(J(\pi)\), where:
  \[
  J(\pi) = \mathbb{E}_{(s_t, a_t) \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]
  \]
SVGD\(^1\) is a variational inference algorithm that iteratively transports a set of particles \(\{x_i\}_{i=1}^{n}\) to match a target distribution

- at each step, particles are updated as follows:

\[
x_i \leftarrow x_i + \epsilon \phi(x_i)
\]

where \(\epsilon\) is the step size and \(\phi\) is a perturbation direction chosen to greedily minimize KL divergence with target distribution

- By restricting \(\phi\) to lie in the unit ball of an RKHS with corresponding kernel \(k\), we can obtain a closed form solution for the optimal perturbation direction...

\(^1\)Q. Liu and D. Wang. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm.
Background: Stein Variational Gradient Descent (SVGD)

**Input**: A target distribution with density function $p(x)$ and a set of initial particles $\{x_i^0\}_{i=1}^n$

**Output**: A set of particles $\{x_i\}_{i=1}^n$ that approximates the target distribution

```
for iteration $\ell$ do
    $x_i^{\ell+1} \leftarrow x_i^\ell + \epsilon_\ell \hat{\phi}^*(x_i^\ell)$ where
    $\hat{\phi}^*(x_i^\ell) = \frac{1}{n} \sum_{j=1}^n \left[ k(x_j^\ell, x) \nabla_{x_j^\ell} \log p(x_j^\ell) + \nabla_{x_j^\ell} k(x_j^\ell, x) \right]$

and $\epsilon_\ell$ is the step size at the $\ell$-th iteration
end
```

**Algorithm 1**: Stein Variational Gradient Descent (SVGD)
Outline

1. Background

2. Stein Variational Policy Gradient

3. Soft Q-Learning

4. Closing Thoughts/Takeaways
Stein Variational Policy Gradient\textsuperscript{2}: Preliminaries

- **Policy iteration**: parametrize policy as $\pi(a_t|s_t; \theta)$ and iteratively update $\theta$ to maximize $J(\pi(a_t|s_t; \theta))$ (a.k.a. $J(\theta)$)

- **MaxEnt Policy Optimization**: instead of searching for a single policy $\theta$, optimize a distribution $q$ on $\theta$ as follows:

  $$\max_q \left\{ \mathbb{E}_{q(\theta)}[J(\theta)] - \alpha D_{KL}(q||q_0) \right\}$$

  where $q_0$ is a prior on $\theta$.

  - If $q_0 = \text{constant}$, then the objective simplifies to
    $$\max_q \left\{ \mathbb{E}_{q(\theta)}[J(\theta)] + \alpha H(q) \right\}$$

    where $H(q)$ is the entropy of $q$

  - Optimal distribution is
    $$q(\theta) \propto \exp \left( \frac{1}{\alpha} J(\theta) q_0(\theta) \right)$$

\textsuperscript{2}Liu et al. Stein Variational Policy Gradient.
**Stein Variational Policy Gradient: Algorithm**

**Input:** Learning rate $\epsilon$, kernel $k(x, x')$, temperature $\alpha$, initial particles $\{\theta_i\}$

for $t = 0, 1, \ldots, T$ do

    for $i = 0, 1, \ldots n$ do
        Compute $\nabla_{\theta_i} J(\theta_i)$ (using RL method of choice)
    end

    for $i = 0, 1, \ldots n$ do
        $\Delta \theta_i = \frac{1}{n} \sum_{j=1}^{n} \left[ \nabla_{\theta_j} \left( \frac{1}{\alpha} J(\theta_j) + \log q_0(\theta_j) \right) k(\theta_j, \theta_i) + \nabla_{\theta_j} k(\theta_j, \theta_i) \right]$ 
        $\theta_i \leftarrow \theta_i + \epsilon \Delta \theta_i$
    end

end

**Algorithm 2: Stein Variational Policy Gradient (SVPG)**
Stein Variational Policy Gradient: Results

Figure 1: Learning curves by SVPG and two baseline versions. The x-axis denotes the training iteration while the y-axis denotes the average return achieved by the policy. Since all three algorithms use the same number of samples per iteration, the x-axis is also proportional to the total number of samples used in training.
# Stein Variational Policy Gradient: Results

## Table 1: Best test return and the number of episodes required to reach within 5% of the maximum return.

<table>
<thead>
<tr>
<th>Method</th>
<th>Task</th>
<th>Joint Return</th>
<th>Joint Episodes</th>
<th>Independent Return</th>
<th>Independent Episodes</th>
<th>SVPG Return</th>
<th>SVPG Episodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2C</td>
<td>Cartpole Swing Up</td>
<td>308.71</td>
<td>189</td>
<td>419.62</td>
<td>474</td>
<td>436.84</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>Double Pendulum</td>
<td>-938.73</td>
<td>46</td>
<td>-256.64</td>
<td>638</td>
<td>-244.85</td>
<td>199</td>
</tr>
<tr>
<td>REINFORCE</td>
<td>Cartpole Swing Up</td>
<td>232.96</td>
<td>253</td>
<td>391.46</td>
<td>594</td>
<td>426.69</td>
<td>238</td>
</tr>
<tr>
<td></td>
<td>Double Pendulum</td>
<td>-892.31</td>
<td>446</td>
<td>-797.45</td>
<td>443</td>
<td>-319.66</td>
<td>327</td>
</tr>
</tbody>
</table>
Figure 2: **State visitation density by REINFORCE-SVPG/Independent algorithms.** The state visitation landscapes of the best four policies learned by SVPG (first row) and Independent agents (second row). The value in the parenthesis is the average return of a policy. All states are projected from a 4D space into a 2D space by t-SNE (Maaten and Hinton, 2008).
Stein Variational Policy Gradient: Results

Figure 3: Average return comparison for all agents learned by the SVPG and Independent algorithms.
Figure 4: The influence of the temperature hyperparameter in REINFORCE/A2C-SVPG.
Soft Q-Learning\(^3\): Preliminaries

- Recall the standard RL objective (assuming \(\gamma = 1\)):
  \[
  \pi_{\text{std}}^* = \arg\max_\pi \sum_t \mathbb{E}_{(s_t,a_t) \sim \pi}[r(s_t,a_t)]
  \]

- In MaxEnt RL, we augment the reward with an entropy term that encourages exploration:
  \[
  \pi_{\text{MaxEnt}}^* = \arg\max_\pi \sum_t \mathbb{E}_{(s_t,a_t) \sim \pi}[r(s_t,a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))]
  \]

Note: when \(\gamma \neq 1\), the MaxEnt RL objective becomes:

\[
\arg\max_\pi \sum_t \mathbb{E}_{(s_t,a_t) \sim \pi}\left[\sum_{l=t}^{\infty} \gamma^{l-t} \mathbb{E}_{(s_l,a_l)}[r(s_l,a_l) + \alpha \mathcal{H}(\pi(\cdot | s_l))| s_t, a_t]\right]
\]

\(^3\)Haarnoja et al. Reinforcement Learning with Deep Energy-Based Policies.
Theorem

Let the soft Q-function be defined by

\[
Q^{\ast}_{\text{soft}}(s_t, a_t) = r_t + \mathbb{E}_{\pi^{\ast}_{\text{MaxEnt}}} \left[ \sum_{l=1}^{\infty} \gamma^l (r_{t+l} + \alpha \mathcal{H}(\pi^{\ast}_{\text{MaxEnt}}(\cdot | s_{t+l})) \right]
\]

and soft value function by

\[
V^{\ast}_{\text{soft}}(s_t) = \alpha \log \int_A \exp \left( Q^{\ast}_{\text{soft}}(s_t, a') \right) da'
\]

then

\[
\pi^{\ast}_{\text{MaxEnt}}(s_t, a_t) = \exp \left( \frac{1}{\alpha} \left( Q^{\ast}_{\text{soft}}(s_t, a_t) - V^{\ast}_{\text{soft}}(s_t) \right) \right)
\]
Soft Q-Learning: Preliminaries

Theorem

The soft Q-function satisfies the soft Bellman equation

\[ Q^*_{\text{soft}}(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1} \sim p}[V^*_{\text{soft}}(s_{t+1})] \]

Theorem (Soft Q-iteration)

Let \( Q_{\text{soft}}(\cdot, \cdot) \) and \( V_{\text{soft}}(\cdot) \) be bounded and assume that

\[ \int_{\mathcal{A}} \exp\left(\frac{1}{\alpha} Q_{\text{soft}}(\cdot, a')\right) da' < \infty \text{ and that } Q^*_{\text{soft}} < \infty \text{ exists. Then the fixed point iteration} \]

\[ Q_{\text{soft}}(s_t, a_t) \leftarrow r_t + \gamma \mathbb{E}_{s_{t+1} \sim p}[V_{\text{soft}}(s_{t+1})], \quad \forall s_t, a_t \]

\[ V_{\text{soft}}(s_t) \leftarrow \alpha \log \int_{\mathcal{A}} \exp\left(\frac{1}{\alpha} Q_{\text{soft}}(s_t, a')\right) da', \quad \forall s_t \]

converges to \( Q^*_{\text{soft}} \) and \( V^*_{\text{soft}} \) respectively.
Soft Q-Learning: Soft Q-iteration in Practice

- In practice, we model the Soft Q function using a function approximator with parameters $\theta$, denoted as $Q^{\theta}_{\text{soft}}$.
- To convert Soft Q Iteration into a stochastic optimization problem, we can re-express $V_{\text{soft}}(s_t)$ as an expectation via importance sampling (instead of an integral) as follows:

$$V^{\theta}_{\text{soft}}(s_t) = \alpha \log \mathbb{E}_q \left[ \frac{\exp\left(\frac{1}{\alpha} Q^{\theta}_{\text{soft}}(s_t, a')\right)}{q(a')} \right]$$

where $q$ is the sampling distribution (e.g. current policy).
- Furthermore, we can update $Q_{\text{soft}}$ to minimize

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim q} \left[ \frac{1}{2} \left( \hat{Q}^{\bar{\theta}}_{\text{soft}}(s_t, a_t) - Q^{\theta}_{\text{soft}}(s_t, a_t) \right)^2 \right]$$

where $\hat{Q}^{\bar{\theta}}_{\text{soft}}(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[ V^{\bar{\theta}}_{\text{soft}}(s_{t+1}) \right]$
Soft Q-Learning: Sampling from Soft Q-function

- **Goal:** learn a state-conditioned stochastic neural network 
  \[ a_t = f^\phi(\xi; s_t), \] 
  with parameters \( \phi \), that maps Gaussian noise \( \xi \) to unbiased action samples from EBM specified by \( Q^\theta_{\text{soft}} \)
  
  - Specifically, we want to minimize the following loss:

  \[
  J_\pi(\phi; s_t) = D_{KL} \left( \pi^\phi(\cdot|s_t) \parallel \exp \left( \frac{1}{\alpha} (Q^\theta_{\text{soft}}(s_t, \cdot) - V^\theta_{\text{soft}}(s_t)) \right) \right)
  \]

  where \( \pi^\phi(\cdot|s_t) \) is the action distribution induced by \( \phi \)

- **Strategy:** Sample actions \( a^{(i)}_t = f^\phi(\xi^{(i)}; s_t) \) and use SVGD to compute optimal greedy perturbations \( \Delta f^\phi(\xi^{(i)}; s_t) \) to minimize \( J_\pi(\phi; s_t) \), where:

  \[
  \Delta f^\phi(\xi^{(i)}; s_t) = \mathbb{E}_{a_t \sim \pi^\phi} \left[ \kappa(a_t, f^\phi(\xi^{(i)}; s_t)) \nabla_{a'} Q^\theta_{\text{soft}}(s_t, a') |_{a' = a_t} \right]
  \]

  \[
  + \alpha \nabla_{a'} \kappa(a', f^\phi(\xi^{(i)}; s_t)) |_{a' = a_t}
  \]
Stein variational gradient can be backpropagated into sampling network using:

\[
\frac{\partial J_\pi(\phi; s_t)}{\partial \phi} \propto \mathbb{E}_\xi \left[ \Delta f^\phi(\xi; s_t) \frac{\partial f^\phi(\xi; s_t)}{\partial \phi} \right]
\]

And so we have:

\[
\hat{\nabla}_\phi J_\pi(\phi; s_t) = \frac{1}{KM} \sum_{j=1}^{K} \sum_{i=1}^{M} \left( \kappa(a_t^{(i)}, \tilde{a}_t^{(j)}) \nabla_{a'} Q^\theta_{\text{soft}}(s_t, a')|_{a'=a_t^{(i)}} + \alpha \nabla_{a'} \kappa(a', \tilde{a}_t^{(j)})|_{a'=a_t^{(i)}} \right) \nabla_\phi f^\phi(\tilde{\xi}^{(j)}; s_t)
\]

the ultimate update direction \( \hat{\nabla}_\phi J_\pi(\phi) \) is the average of \( \hat{\nabla}_\phi J_\pi(\phi; s_t) \) over a mini-batch sampled from replay memory.
Soft Q-Learning: Algorithm

for each epoch do
  for each t do
    Collect Experience
    \( a_t \leftarrow f^\phi(\xi, s_t) \) where \( \xi \sim \mathcal{N}(0, I) \)
    \( s_{t+1} \sim p(s_{t+1}|a_t, s_t) \)
    \( \mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, a_t, r(s_t, a_t), s_{t+1}\} \)
    Sample minibatch from replay memory
    \( \{s_t^{(i)}, a_t^{(i)}, r_t^{(i)}, s_{t+1}^{(i)}\}_{i=0}^N \sim \mathcal{D} \)
    Update soft Q-function parameters
    Sample \( \{a^{(i,j)}\}_{j=0}^M \sim q \) for each \( s_{t+1}^{(i)} \)
    Compute \( \hat{V}_{\text{soft}}(s_{t+1}^{(i)}) \) and \( \hat{\nabla}_\theta J_Q \), update \( \theta \) using ADAM
    Update policy
    Sample \( \{\xi^{(i,j)}\}_{j=0}^M \sim \mathcal{N}(0, I) \) for each \( s_t^{(i)} \)
    Compute actions \( a_t^{(i,j)} = f^\phi(\xi^{(i,j)}, s_t^{(i)}) \)
    Compute \( \Delta f^\phi \) and \( \hat{\nabla}_\phi J_\pi \), update \( \phi \) using ADAM
  end
  if epoch mod update interval = 0 then
    Update target parameters: \( \bar{\theta} \leftarrow \theta, \bar{\phi} \leftarrow \phi \)
  end
end

Algorithm 3: Soft Q-Learning
Soft Q-Learning: Results

**Figure 1.** Illustration of the 2D multi-goal environment. Left: trajectories from a policy learned with our method (solid blue lines). The $x$ and $y$ axes correspond to 2D positions (states). The agent is initialized at the origin. The goals are depicted as red dots, and the level curves show the reward. Right: Q-values at three selected states, depicted by level curves (red: high values, blue: low values). The $x$ and $y$ axes correspond to 2D velocity (actions) bounded between -1 and 1. Actions sampled from the policy are shown as blue stars. Note that, in regions (e.g. (2.5, 2.5)) between the goals, the method chooses multimodal actions.

**Figure 2.** Simulated robots used in our experiments.

(a) Swimming snake 
(b) Quadrupedal robot

**Figure 3.** Comparison of soft Q-learning and DDPG on the swimming snake task and the quadrupedal robot maze task. (a) Shows the maximum traveled forward distance since the beginning of training for several runs of each algorithm; there is a large reward after crossing the finish line. (b) Shows our method was able to reach a low distance to the goal faster and more consistently. The different lines show the minimum distance to the goal since the beginning of training. For both domains, all runs of our method cross the threshold line, acquiring the more optimal strategy, while some runs of DDPG do not.
Soft Q-Learning: Results

Figure 4. Quadrupedal robot (a) was trained to walk in random directions in an empty pretraining environment (details in Figure 7, see Appendix D.3), and then finetuned on a variety of tasks, including a wide (b), narrow (c), and U-shaped hallway (d).

Figure 5. Performance in the downstream task with fine-tuning (MaxEnt) or training from scratch (DDPG). The x-axis shows the training iterations. The y-axis shows the average discounted return. Solid lines are average values over 10 random seeds. Shaded regions correspond to one standard deviation.
Soft Q-Learning: Results

Figure 7. The plot shows trajectories of the quadrupedal robot during maximum entropy pretraining. The robot has diverse behavior and explores multiple directions. The four columns correspond to entropy coefficients $\alpha = 10, 1, 0.1, 0.01$ respectively. Different rows correspond to policies trained with different random seeds. The $x$ and $y$ axes show the $x$ and $y$ coordinates of the center-of-mass. As $\alpha$ decreases, the training process focuses more on high rewards, therefore exploring the training ground more extensively. However, low $\alpha$ also tends to produce less diverse behavior. Therefore the trajectories are more concentrated in the fourth column.