Depth First Learning: SVGD

Week 6: Stein's Method in Reinforcement Learning

June 20, 2019

One of the most exciting use cases of SVGD is in reinforcement learning, due to its connection to maximum entropy reinforcement learning. This week, we study two key techniques in reinforcement learning that use SVGD as the underlying mechanism. In reinforcement learning, the target distribution is not known, so we derive gradient updates to our parameters using policy gradients. As we derive the gradient estimators in the maximum-entropy framework of reinforcement learning, we will start to see what benefits SVGD-based methods have. In particular, we will focus on the explore-exploit tradeoff, as well as normalization constants for intractable distributions, and see how SVGD helps us get around complicated problems regarding both.

1 Table of Contents

2 Reinforcement Learning

In reinforcement learning, we often focus on learning a *representation* of the environment (an abstraction of states, a model of the world), or how to *operate* in it. We focus on the latter: the paradigm of *control*.

In reinforcement learning, the base setup consists of a Markov Decision Process: a tuple of S: the state space, A: the action space, R: the reward function, and T: the transition function. The state space could be images, or x-y coordinates of the agent's location in a map, while the action space might describe the different commands a robot's motors may take. Both the reward function and transition function take in a state $s \in S$ and action $a \in A$, and generate a numerical reward for that pair and the next state $s' \in S$ respectively. In the control paradigm, an agent aims to learn a policy π, which is a mapping from $S \to A$. The policy tells an agent what actions to take in which state, in order to maximize some long-term expectation of the cumulative rewards: the *return*.

While return can take many forms, we focus on the general notation. Below describes the general form of *return*:

$$G_t = \sum_{k}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

where $\gamma \in [0, 1)$ is the *discount factor*, which effectively tunes our horizon: from this state, how much should we *discount* future returns. A smaller gamma prioritizes early returns (myopic), where as $\gamma \to 1$, we give later rewards larger contributions to *this* state's expected return.

3 Policy Optimization via Policy Gradients

Note: We will abstract away the derivation of the policy gradient theorem. A fantastic explanation / derivation of this sits in Lilian Weng's blog on policy gradient algorithms [12].

Of course, in the beginning, our policy won't know what to do generally. Based on feedback it gets from the environment (in the form of rewards), we want to *teach* it what to do in each state. However, unlike supervised learning, we don't have a *ground truth* distribution for this (in the form of labels), and must learn how to do this via exploration (more on the explore-exploit tradeoffs later).

However, without labels, a bigger question arises: *how* do we actually use these to optimize our policy, which aims to maximizes the return G_t? Here, we use the policy gradient theorem, which uses frames the cost function of policy optimization as follows:

$$J(\theta) = \mathbf{E}_{\tau \sim \pi_\theta}[R(\tau)] \tag{2}$$

where τ is the *trajectory*: the states and actions taken by $\pi : s_0, a_0, s_1, a_1 ... \sim \pi$ in the environment.

Effectively, we are learning a policy π with parameters θ, looking to perform gradient ascent on the cost function above to maximize it:

$$\theta_{t+1} = \theta_t + \epsilon \nabla_\theta J(\theta)|_{\theta_t} \tag{3}$$

where ϵ is some small step size.

The policy gradient theorem explains how we can use the sampled actions and corresponding returns to actually calculate the above gradient, which we summarize below:

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a|s) R(\tau)] \tag{4}$$

This is what we call the REINFORCE policy gradient estimator [13] in the undiscounted finite horizon (where the horizon is given by H). REINFORCE comes with a whole host of its own issues, but the following sections don't depend as much on the details of these, so we abstract them away and provide resources for interested readers. Policy gradient algorithms have gone through

ups-and-downs in popularity over the years, but currently, they are a popular research topic with lots of interesting work coming out.

4 Stein Variational Policy Gradient

In this section, we focus on the first of three papers in our exploration into Applied SVGD in RL: Stein Variational Policy Gradient [5]. Before that, we set the scene for one of the toughest open problems in RL research today: the explore-exploit tradeoff.

4.1 Explore or Exploit?

As we've seen throughout this curriculum, supervised learning learns a function $f : \mathcal{X} \to \mathcal{Y}$, whereas reinforcement learning learns a function $\pi : \mathcal{S} \to \mathcal{A}$, hoping to maximize return. On the surface, they seem quite similar. However, the reinforcement learning paradigm has to deal with a fundamental issue, one that supervised learning doesn't have: the issue of exploration.

In supervised learning, we are provided with targets (labels). This allows us, even in the beginning of training, to quickly gauge *how* wrong we are. We are even given the ground truth answer, allowing us to take advantage of this extra information. On the other end, there is also an easy way (given a set training set) to know when we are "done". While this doesn't address complex issues and intricacies such as overfitting, in general, it provides a good contrast to the RL paradigm.

In reinforcement learning, the agent doesn't have access to the ground truth. While some methods require the knowledge of transition matrices or reward functions, most RL algorithms don't like to incorporate these as givens into the algorithm. RL algorithms, therefore, have no notion of *ground truth* to optimize against. How good was this action in this state? Alone, it's unclear to say: only when we take *another* action that gives us 10x the original reward, do we have an understanding of how *good* something is. Good exploration is often tied with good performance in reinforcement learning, allowing our algorithms to sample various and diverse parts of the state space, leading to stronger and more robust policies.

(Un)fortunately, this leads to some very tough research questions: how much exploration should you do, before you try to *exploit* the reward sources you do know about? Should you ever re-explore? Should it be continuous, or in spurts? While the answers are unclear, one promising route as been maximum-entropy reinforcement learning, which we cover in the next section.

4.2 Maximum-Entropy Reinforcement Learning

Maximum-Entropy Reinforcement Learning [14] provides a framework for incorporating long-term exploration by augementing the per-timestep reward with the entropy, which we call the entropy-augmented return [10]:

$$J_{ME}(\theta) = \mathbf{E}_{\tau \sim \pi_\theta} \sum_t [r(s_t, a_t) + \alpha \mathcal{H}(\cdot|s_t)] \tag{5}$$

If you're familiar with reinforcement learning (particularly, implementationally), it is important to notice that this is different than an entropy *bonus*, which typically, is a single-step bonus given when calculating the policy gradient:

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau \sim \pi_\theta} [\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a|s) R(\tau)] - \alpha \mathcal{H}(\cdot|s_t) \tag{6}$$

where the entropy term sits outside the summation, controlled by a temperature term α.

While entropy can be thought of "randomness" (we calculate the entropy as a KL Divergence between the current policy and a uniform, random one, which maximizes the equation when the policy is random) at any given timestep, the entropy-augmented return incorporates this randomness into the objective function. From the introduction of Soft Q-Learning [3].

Maximum entropy RL does not just prioritize the single deterministic behavior that has the lowest cost, but the entire range of low-cost behaviors, explicitly maximizing the entropy of the corresponding policy. As we shall see in a later section, Maximum Entropy RL has other benefits, but for now, we attribute it to being able to learn diverse, exploratory policies.

4.3 SVGD for Diverse Policies

The pieces seem to be there already: Maximum Entropy RL looks for diverse policies, and the Stein gradient helps maximize some optimization variable (as we've seen it, the log probability of the samples) while diversifying the particles via the kernel term. Stein Variational Policy Gradient (SVPG) treats the the policy parameters as a random variable, and then search for a distribution $q(\theta)$ in a Bayesian Inference framework:

$$J_{ME}(\theta) = \max_q \{\mathbf{E}_{q(\theta)}[J(\theta)] + \alpha \mathcal{H}(q)\} \tag{7}$$

It is important here to note that the entropy is now considered in *parameter* space, rather than action space. We leave the discussion of that choice until the next section.

In addition, the entropy terms discussed in the above sections are all actually KL Divergences, comparing π or q with some base distribution. When we use a uniform prior, we now have the entropy (to a constant, which drops out in the optimization). But, *if* we did have a prior to use, maximum entropy allows for a principled way to incorporate it. We defer the discussion on priors until the last section.

Back to SVPG, when we treat θ as a random variable, we can calculated ∇_q of the above optimization, leading us to:

$$\nabla_q J_{ME}(\theta) = \int_\theta (J(\theta) - \alpha \log q(\theta) - \alpha - \alpha \log q_0(\theta)) \tag{8}$$

where q_0 is the prior that we calculate the KL Divergence against. Without a proper prior, the authors choose to set $\log q_0(\theta)$ to 1, leading to the last two terms cancelling out. This leads to an optimal distribution of:

$$\log q^*(\theta) = \frac{1}{\alpha} J(\theta) \tag{9}$$

and when we exponentiate both sides (and keep the original prior term inside the equation), we get the proportionality statement seen in Equation 7 of the original paper.

The key insight is plugging in the SVPG posterior into the SVGD update rule, leading to the following update rule:

$$\hat{\phi}(\theta_i) = \frac{1}{n} \sum_{j=1}^{n} \nabla_{\theta_j} \frac{1}{\alpha} J(\theta) k(\theta_i, \theta_j) + \nabla_{\theta_j} k(\theta_i, \theta_j) \tag{10}$$

We can therefore calculate $\nabla_\theta J(\theta)$ using any of our original policy gradient estimators (the Tensorflow implementations at the bottom used REINFORCE, whereas the Pytorch one uses Advantage Actor Critic (A2C) [7], and plug them into the update rule above.

In practice, this means that we have to maintain n policies (often small neural networks, parameterized by θ_i), roll them out separately (but, in parallel), and calculate their Stein policy gradient updates in the above, interactive fashion. The authors show strong exploratory behavior on a few continuous control tasks, but maintaining multiple, ensembled policies cuts a divide in the RL community. However, SVPG is a good fit for methods or areas where parallelization is already a part of the framework, such as domain randomization [6].

4.4 Self Imitation Learning: Towards Diversity in Policy Space

Diversity is good, but makes an underlying assumption: diversity according to *what*? SVPG makes the assumption that similarly-*parameterized* policies generate similar behavior, but when dealing with deep networks, calculating kernel similarity scores between their parameters may not work as well. Instead, a more practical comparison would compare the behavioral similarity between particles. Learning Self-Imitating Diverse Policies [2] improves on SVPG in two ways: they move away from the RBF kernel and instead utilize a JS-kernel, based on the Jensen-Shannon Divergence, in order to better model similarities between policies. They build upon the self-imitation learning framework [8], which itself builds upon the equivalence between soft Q-learning [10] (discussed below) and policy gradient methods! Almost full circle!

5 Soft Q-Learning

There are lots of benefits to using more complicated distributions, especially in the context of exploration and stability in reinforcement learning. However, a major issue arises: how do we sample from them? In this section, we cover Amortized SVGD [1] and show how Soft Q-Learning [3] uses this to learn exploratory and compositional policies.

5.1 Amortized SVGD: Learning To Sample

As we've seen in previous weeks, there exist intractable distributions that we want to model with variational distributions: distributions that are restricted to a simpler form that approximate the target density $p(x)$. Variational inference traditionally performs well when we can sample $p(x)$ (but maybe, cannot calculate normalization constant Z), but the assumption is that we have the full ability to calculate the density $q(x)$, as well as its gradients (in order to minimize the KL Divergence, $D(q||p)$).

However, sometimes, we cannot calculate the density q; Deemed wild variational inference [9], the problem is set up as follows:

> Given some density $p(x)$ (i.e real images) defined on $\mathcal{X}$ and a simulator $\hat{x} = f(\psi; \eta)$, parameterized by η and takes random seed ψ drawn from unknown density q_0 to output some value on $\mathcal{X}$, find parameters η s.t $f(\psi \sim \mathrm{q}_0; \eta)$ closely approximates density p.

In our case, the simulator will be a neural network, which takes in some random seed, and at convergence, generates samples $\hat{x}$ that closely sample those that make up the ground-truth density $p(x)$. ψ is usually noise sampled from some distribution (i.e $\mathcal{N}(0, 1)$) which then gets passed through the network. Due to the complexity of the network's underlying function, We can only have $\hat{x}$ and the derivative for optimization.

Amortized SVGD can be succinctly described as follows, from the original paper:

> At convergence, the samples drawn from q_η reach the equilibrium state of SVGD, and hence form a good approximation of p. We can view this method as amortizing or distilling the SVGD dynamics using parametric family q_η or its inference network $f(\psi; \eta)$ and call it amortized SVGD.

Knowing the SVGD will provide arbitrarily good estimates of some target distribution, Amortized SVGD trains its network outputs to match those given by SVGD, and at convergence, uses the amortized ones for rollouts. They use various optimization techniques (MSE between the network output and Stein samples), but interestingly, Equation (15) of their paper, reproduced below, is effectively a "backprop of the SVG into f":

$$\eta \leftarrow \eta + \epsilon \sum_n \partial_\eta f(\psi_i; \eta)\phi^*(x_i) \tag{11}$$

where ϕ^* is the Stein gradient calculated when given sample x_i.

5.2 Soft Q-Learning

As we've seen, maximum entropy RL is great for learning diverse policies: in general, we find this works best when we have *multimodal* distributions: i.e there is often more than one group, or mode, of optimal actions, and ideally, we'd like to explore both. The problem for RL in continuous control is that a choice of policy parameterization has to be made - traditionally, we'd use normal distributions and use noise to explore, but this doesn't cover the multi-modal case described above. The authors in Soft Q-Learning choose to use energy-based models, where the optimal policy takes the form of:

$$\pi(a|s) \propto \exp Q(s, a) \tag{12}$$

where the Q function is represented by a neural network. However, energy-based models are notoriously hard to sample from, and traditionally, MCMC methods have been used. As we've seen above, amortized SVGD learns a network that, at convergence, can learn to generate samples from arbitrary distributions, so the authors use it to generate actions from the energy-based model.

They use the same ideas above and backpropogate the Stein gradient in order to transform their action network samples - which take noise samples ξ and maps them to actions via $f^\phi(\xi, s_t)$ - into samples from the target energy-based model specified by Q_{soft} (the soft Q-network, hence the name of the paper).

Amortized inference for samples from this distribution allows for other benefits of energy-based models (namely good exploration and compositionality; you can find really amazing videos of this on their blog post [11] to meet the expressivity of deep learning. Soft Q-Learning paved the way for Soft Actor-Critic [4], an incredibly popular actor-critic algorithm in wide use in the deep RL community today.

6 Implementations

Below you can find some open-source implementations of the various algorithms discussed above:

- largelymfs's Tensorflow Implementation of SVPG
- jsikyoon's Tensorflow Implementation of SVPG
- Bhairav's Pytorch Implementation of SVPG
- Tuomas Haarnoja's Implementation of Soft Q-Learning

References

[1] Y. Feng, D. Wang, and Q. Liu. Learning to draw samples with amortized stein variational gradient descent, 2017.

[2] T. Gangwani, Q. Liu, and J. Peng. Learning self-imitating diverse policies, 2018.

[3] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies, 2017.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[5] Y. Liu, P. Ramachandran, Q. Liu, and J. Peng. Stein variational policy gradient, 2017.

[6] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull. Active domain randomization, 2019.

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[8] J. Oh, Y. Guo, S. Singh, and H. Lee. Self-imitation learning, 2018.

[9] R. Ranganath, J. Altosaar, D. Tran, and D. M. Blei. Operator variational inference, 2016.

[10] J. Schulman, X. Chen, and P. Abbeel. Equivalence between policy gradients and soft q-learning, 2017.

[11] H. Tang and T. Haarnoja. Soft q learning. *https://bair.berkeley.edu/blog*, 2017.

[12] L. Weng. Policy gradient algorithms. *lilianweng.github.io/lil-log*, 2018.

[13] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(34):229256, May 1992.

[14] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI08, page 14331438. AAAI Press, 2008.