Qraft Technolongies (AXE)

BCQ: Off-Policy Deep Reinforcement Learning without Exploration

RL Research

1 Abstract

Many of the most successful modern (i.e., "deep") off-policy algorithms use some variant of experience replay, but the authors claim that this only works when the data in the buffer is correlated with the data induced by the current agent's policy. This does not work if there is what the authors define as **extrapolation error**, which is when there is a mismatch between the two datasets, since with function approximation, unseen state-action pairs (s, a) might be more or less attractive than seen pairs and get terrible reward from environment.

Though experience replay is actually designed to break correlation among samples, the most recent information is put into the buffer, bumping older stuff out. By definition, that means some of the data in the experience replay is correlated with the agent's policy. To address such problems, the author introduces about restricting the actions so that we keep funneling the agent towards the high-quality states in the batch.

We introduce a novel class of off-policy algorithms, batch-constrained reinforcement learning, which restricts the action space in order to force the agent towards behaving close to on-policy with respect to a subset of the given data = We want the set of states the agent experiences to be similar to the set of stated from the batch.

2 Idea

Idea of Batch Constrained Deep Q-learning (BCQ) performs general Q-learning, but considers one of such partial methods in the phase of calculating maximum values for Q (usually $\max_{a'} Q\left(s', a'\right)$ by bellman equation).

1. A method to consider only a' of (s', a') that appeared "in real" in batch data, without obtaining maximum values for all possible actions.
2. A method to remove behaviors that are not well-selected by behavior policy π_b (the policy that generates static data in off-policy setting)

A summary of the paper's theory is that batch-constrained learning still converges to an optimal policy for deterministic MDPs. Much of the theory involves redefining or inducing a new MDP based on the batch, and then deferring to standard Q-learning theory.

3 Method

Their first foray is to try and redefine the Bellman operator in finite, discrete MDPs in the context of reducing extrapolation error so that the induced policy will visit the state-action pairs that more closely correspond with the distribution of state-action pairs from the batch.

A summary of the paper's theory is that batch-constrained learning still converges to an optimal policy for deterministic MDPs. Much of the theory involves redefining or inducing a new MDP based on the batch, and then deferring to standard Q-learning theory.

The paper claims that normal Q-learning on the batch of data will result in an optimal value function for an alternative MDP, $M_{\mathcal{B}}$, based on the batch $\mathcal{B}$. A related and important definition is the tabular extraploation error ϵ_{MDP}, defined as discrepancy between the value function computed with the batch versus the value function computed with the true MDP M :

$$\epsilon_{\text{MDP}}(s,a) = Q^{\pi}(s,a) - Q^{\pi}_{\mathcal{B}}(s,a)$$

For any policy π, the exact form of $\epsilon_{\text{MDP}}(s,a)$ can be computed through a Bellman-like equation:

$$\begin{aligned}\epsilon_{\text{MDP}}(s,a) = \sum_{s'} &\left(p_M\left(s' \mid s,a\right) - p_{\mathcal{B}}\left(s' \mid s,a\right)\right) \\ &\left(r\left(s,a,s'\right) + \gamma \sum_{a'} \pi\left(a' \mid s'\right) Q^{\pi}_{\mathcal{B}}\left(s',a'\right)\right) \\ &+ p_M\left(s' \mid s,a\right) \gamma \sum_{a'} \pi\left(a' \mid s'\right) \epsilon_{\text{MDP}}\left(s',a'\right)\end{aligned}$$

it's simpler to write as:

$$\epsilon^{\pi}_{\text{MDP}} = \sum_{s} \mu_{\pi}(s) \sum_{a} \pi(a \mid s) \left|\epsilon_{\text{MDP}}(s,a)\right|$$

By using the above, they are able to derive a new algorithm: Batch-Constrained Q-learning (BCQL) which restricts the possible actions to be in the batch:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma\left\{_{a' \text{ s.t. } (s',a') \in \mathcal{B}} Q\left(s',a'\right)\right\}\right)$$

4 Practical Algorithm for high-dimensional, continuous control

Batch-Constrained deep Q-learning (BCQ) utilizes four parameterized networks.

1. A Generative model $G_{\omega}(s)$ which, given the state as input, produces an action. Using a generative model this way assumes we pick actions using:

   $$\underset{a}{\text{argmax}} P^{G}_{\mathcal{B}}(a \mid s)$$

   or in other words, the most likely action given the state, with respect to the data in the batch. This is difficult to model in high dimensional continuous control environments, so they approximate it with a variational autoencoder. This is trained along with the policy parameters during each for loop iteration.

2. A Perturbation model $\xi_{\phi}(s,a,\Phi)$ which aims to "optimally perturb" the actions, so that they don't need to sample too much from $G_{\omega}(s)$. The perturbation applies noise in $[-\Phi,\Phi]$. It is updated via a deterministic policy gradient rule:

   $$\phi \leftarrow \underset{\phi}{\text{argmax}} \sum_{(s,a) \in \mathcal{B}} Q_{\theta}\left(s, a + \xi_{\phi}(s,a,\Phi)\right)$$

   The above is a maximization problem over a sum of Q-function terms. The Q-function is differentiable as we parameterize it with a deep neural network, and stochastic gradient descent methods will work with stochastic inputs.

3. Two Q-networks $Q_{\theta_1}(s,a)$ and $Q_{\theta_2}(s,a)$, to help push their policy to select actions that lead to "more certain data." This is based Twin-Delayed DDPG (TD3) algorithm.

All networks other than the generative model also consist of target networks, following standard DDPG practices.

All together, their algorithm uses this policy:

$$\pi(s) = \underset{a_i+\xi_\phi(s,a_i,\Phi)}{\operatorname{argmax}} Q_\theta\left(s, a_i + \xi_\phi\left(s, a_i, \Phi\right)\right) \quad \{a_i \sim G_\omega(s)\}_{i=1}^n$$

To be clear, they approximate this maximization by sampling n actions each time step, and picking the best one. The perturbation model increases the diversity of the sampled actions.

Algorithm 1 BCQ

Input: Batch $\mathcal{B}$, horizon T, target network update rate τ, mini-batch size N, max perturbation Φ, number of sampled actions n, minimum weighting λ.
Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$, perturbation network ξ_ϕ, and VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, with random parameters θ_1, θ_2, ϕ, ω, and target networks $Q_{\theta'_1}, Q_{\theta'_2}$, $\xi_{\phi'}$ with $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$.
for $t = 1$ **to** T **do**
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\mu, \sigma = E_{\omega_1}(s, a), \quad \tilde{a} = D_{\omega_2}(s, z), \quad z \sim \mathcal{N}(\mu, \sigma)$
  $\omega \leftarrow \operatorname{argmin}_\omega \sum (a - \tilde{a})^2 + D_{\mathrm{KL}}(\mathcal{N}(\mu, \sigma) || \mathcal{N}(0, 1))$
  Sample n actions: $\{a_i \sim G_\omega(s')\}_{i=1}^n$
  Perturb each action: $\{a_i = a_i + \xi_\phi(s', a_i, \Phi)\}_{i=1}^n$
  Set value target y (Eqn. 13)
  $\theta \leftarrow \operatorname{argmin}_\theta \sum (y - Q_\theta(s, a))^2$
  $\phi \leftarrow \operatorname{argmax}_\phi \sum Q_{\theta_1}(s, a + \xi_\phi(s, a, \Phi)), a \sim G_\omega(s)$
  Update target networks: $\theta'_i \leftarrow \tau\theta + (1 - \tau)\theta'_i$
  $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
end for

Figure 1: Alogorithm of BCQ