A GB-Q Stable Point

Dealing with an unlimited graph backup target, we can transform the problem into policy evaluation in an MDP that have been solved by conventional RL algorithms, say, value iteration or tabular Q learning. The data graph can be used to construct an MDP in which the optimal bellman equation is the same as the definition of graph backup target. Formally, this data graph MDP would be $(\bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{\mathcal{R}}, \bar{p})$:

- $\bar{\mathcal{S}} = \mathcal{S}_{\text{seen}} \cup \{S_{\text{absorb}}\}$ is the set of all seen states unioned with a special absorbing state, whose value is 0 for any policy.
- $\bar{\mathcal{A}} = \mathcal{A}$ is the same set of actions as original MDP.
- $\bar{\mathcal{R}} = \mathcal{R} \cup \{0\}$ is the of all seen rewards, unioned with 0.
- $\bar{p}(s', r|s, a) : \bar{\mathcal{S}} \times \bar{\mathcal{R}} \times \bar{\mathcal{S}} \times \bar{\mathcal{A}} \to [0, 1]$ specifies the transition probability.

The transition probability will be proportional to the frequency $f(s, a, r, s')$. Since we add a new state S_{absorb}, to expand the domain of f, we define

$$\forall (s, a, r, s') \in (\bar{\mathcal{S}} \times \bar{\mathcal{A}} \times \bar{\mathcal{R}} \times \bar{\mathcal{S}} - \mathcal{T}), f(s, a, r, s') = 0. \tag{5}$$

If (s, a) has not been experienced, the transition will point to the absorbing state S_{absorb}, written as:

$$\begin{aligned} \bar{p}(s', r|s, a) &= \frac{f(s, a, r, s')}{c(s, a)}, \quad \text{if } c(s, a) > 0 \\ \bar{p}(S_{\text{absorb}}, q_{\boldsymbol{\theta}'}(s, a)|s, a) &= 1, \qquad \text{otherwise} \end{aligned}. \tag{6}$$

, where again $c(s, a) = \sum_{(s,a,r,s') \in \mathcal{T}} f(s, a, r, s')$ is the normalizer. By expanding the optimal bellman equation, it is not difficult to validate the optimal Q function in this MDP is the same as the graph backup target. Let the $\bar{q}^*$ to be the optimal q function, its bellman equation is:

$$\bar{q}^*(s, a) = \sum_{r \in \bar{\mathcal{R}}, s' \in \bar{\mathcal{S}}} \bar{p}(s', r|s, a)[r + \gamma \max_{a'} \bar{q}^*(s', a')]. \tag{7}$$

Replacing $\hat{p}$ with definition in Eq. (6), we get:

$$\bar{q}^*(s, a) = \begin{cases} \sum_{r \in \bar{\mathcal{R}}, s' \in \bar{\mathcal{S}}} \frac{f(s,a,r,s')}{c(s,a)} \left(r + \gamma \max_{a'} \bar{q}^*(s', a')\right), & c(s, a) > 0 \\ q_{\boldsymbol{\theta}'}(s, a) + \gamma \max_{a'} \bar{q}^*(S_{\text{absorb}}, a') = q_{\boldsymbol{\theta}'}(s, a), & \text{otherwise} \end{cases}. \tag{8}$$

Therefore we get $\bar{q}^*(s, a) = G_s^a$ and computing the target value is transformed into a policy evaluation problem with known MDP dynamics. Without maintaining an explicit graph structure, the replay buffer can be treated as an any-state sampling model of this data graph MDP. We thus propose to run Q-learning within the replay buffer in order to compute the graph backup target. Since $\bar{q}^*(s, a) = q_{\boldsymbol{\theta}'}(s, a)$ for all the unseen state-action pairs, we can first initialise all the q values in the table to according to $q_{\boldsymbol{\theta}'}$. The q values of unseen state-action pairs will then kept correct as the update will not be applied to transitions that does not exist in the replay buffer.

B Data Graph Definition

A MDP data graph is a bipartite multi graph $(\mathcal{S}_{\text{seen}}, \mathcal{N}, \mathcal{E}_{\text{out}}, \mathcal{E}_{\text{in}})$, where $\mathcal{S}_{\text{seen}} \subseteq \mathcal{S}$ is the set of seen state nodes, $\mathcal{N} \subseteq \mathcal{S}_{\text{seen}} \times \mathcal{A}$ is the set of (state conditioned) action nodes, $\mathcal{E}_{\text{out}}$ is a multiset of edges pointing from state nodes to action nodes and $\mathcal{E}_{\text{in}}$ is a multiset of reward weighted edges pointing from action nodes to state nodes. A state node can point to multiple action nodes because multiple actions might be tried, and the action nodes can point to multiple state nodes because of the stochastic dynamics. When a new transition (s, a, r, s') is observed, edge $(s, (s, a))$ will be added to $\mathcal{E}_{\text{in}}$ and $((s, a), r, s')$ will be added to $\mathcal{E}_{\text{out}}$. A visual example of this data graph can be seen in the graph backup diagram in Fig. 3 with tried (blue) action nodes only.

Relating this data graph to Eq. (4), we can see $c(s, a)$ is the number of $(s, (s, a))$ edges in $\mathcal{E}_{\text{in}}$ and $f((s, a, r, s'))$ is the number of $((s, a), r, s')$ edges $\mathcal{E}_{\text{out}}$.

C Discussion and Future Work

C.1 Computational Costs

When scaled up to more training steps, the computational overhead of graph backup mostly comes from target network inference. For GB-limited, we run the target network for all seen states after its update, which amortises the cost of repeated target network inference. However, this also means the computational cost of the whole learning algorithm will increase quadratically with the number of seen states. If we denote the size of the seen states be n, the breath of expansion to be b and the depth be d, the computational complexity of the DQN with graph GB-limited will be $O(n^2 + db)$, If the size of the replay buffer (or static data in offline RL), say, 100M of length, there might be no need to compute all the targets since we only use a subset of the data each target update cycle. In that case, we can run target network in real-time, when doing graph expansion. The computational complexity will then be linear to the size of the replay buffer but with a larger multiplier, namely, $O(dbn)$.

The situation for GB-Q is similar, as both the number of tabular Q updates needed and the cost of target network evaluation of the whole data set will increase with the size of the data set. This means its time complexity will be $O(n^2)$. However, the good news is these extra computations are highly parallelizable and the target network is not always updating. Further, the benefits of graph backup may also increase with the size of the data set because of more crossovers between trajectories. In fact, while using more basic backup operators is computationally less expensive, they may result in the need for more gradient descent steps and target updates, and hence on net be less efficient than graph backup.

In the setting where we have a perfect simulator, extra computation can be used to create more data for simulated environments, which may be a more efficient use of compute that graph backup. Hence, the higher computational cost means graph backup is more useful for the case where data efficiency is much more important than computational efficiency. For example in the offline (batch) RL setting where the size of the data set is pre-set to a limited amount, or problem settings where we do not have a simulator and online interactions are expensive. We're interested in future work which applies graph backup to offline RL problems.

C.2 High-dimensional or Continuous Observations

The power of graph backup comes from the graph-structured transition data. However, for some tasks with high-dimensional or continuous inputs, all the observations might be unique. In that case, the data graph will degenerate into a set of independent chains, and the graph backup will then become equivalent to tree backup. In order to get more benefits from backup in this case, we need to have a discrete representation of the states. In the RL setting, similar challenges are met for exploration mechanisms that require state counts. Go-explore [Ecoffet et al., 2021] and OPIQ [Rashid et al., 2020] simply downsampled the original image into a discrete greyscale image in Atari games, and Tang et al. [2017] shows the use of simhash or learned hash code can produce high-quality intrinsic rewards simply based on state counts in both Atari and continuous control tasks. These methods could be adapted to help graph backup scale to higher-dimensional or continuous inputs by better-detecting state crossovers between trajectories, which would be interesting future work.