Choosing Which Queries to Make

David Krueger, John Salvatier

August 2016

We're proposing a few methods for how to choose the number of queries per state-action in model-based active RL. We consider the tabular MDP case, with state and actions sets $\mathcal{S}, \mathcal{A}$, and transition and reward functions $\mathcal{T}, \mathcal{R}$. We use r to denote instantaneous rewards.

We suppose each query has a fixed cost of c, and let $\mathcal{Q} := \mathbb{N}^{|\mathcal{S}\times\mathcal{A}|}$ represent the set of all possible sets of queries. Then let $q_t \in \mathcal{Q}$ be the set of queries which have been performed at time t. We can make our decision to query (or not) by specifying some *desired query set*, $q_t^{max} \in \mathcal{Q}$, representing to set of all queries we'd like to make, and querying whenever we have not yet made all of those queries, i.e. if $q_t < q_t^{max}$.

We measure performance as returns minus total query cost, i.e. c times the number of queries performed. Our baseline heuristic considers a fixed desired query set, $q_t^{max} = (n, ..., n)$, with n as a hyperparameter. It seems clear that the optimal $n \to \infty$ as $c \to 0$, or $\gamma \to 1$, and $n \to 0$ as $c \to \infty$ or $\gamma \to 0$. But it's not clear what specific value of n to choose. Since we care about our cost during learning, we'd like a way to choose n before learning (or adjusting it during learning).

We use the $PSRL$ algorithm with a slight modification: we fix the agent's prior over rewards for a given state and action to be its expectation when no more queries of that state-action pair will be performed. We use $PSRL_n$ to denote this modification of $PSRL$ using with $q_t^{max} = (n, ..., n)$.

1 A Proposal for Tuning n

We propose tuning n via Simulated Query Rollouts (SQR).

The basic idea is to sample k different reward functions (or, more generally, environments) $\tilde{\mathcal{R}}_1, ..., \tilde{\mathcal{R}}_k$ from the agent's prior on $\mathcal{R}$, and evaluate the agent's performance on each of these simulated worlds for different query strategies, e.g. different values of n. We then use the best performing query strategy to act in the real environment. This strategy may be sensitive to differences between the prior and the real environment.

Note that

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} perf_i(\underset{n}{PSRL}) = \mathbb{E}\, PSRL\, perf(\underset{n}{PSRL}).$$

Algorithm 1 Simulated Query Rollouts

1: **for** $i \in \{1, ..., k\}$ **do**
2: Sample $\tilde{\mathcal{R}}_i \sim P(\mathcal{R})$
3: Sample $\{\tilde{r}\}_i \sim \tilde{\mathcal{R}}_i$
4: **for** $n \in \{0, ..., N\}$ **do**
5: Run $PSRL_n$, using pre-sampled rewards $\{\tilde{r}\}_i$.
6: Record the resulting performance: $perf_i(PSRL_n)$
return $n = \text{argmax} \frac{1}{k} \sum_{i=1}^{k} perf_i(PSRL_n)$

Also, note that we propose using the same instantaneous sampled rewards $\{\tilde{r}\}_i$ for each value of n in each environment $\tilde{\mathcal{R}}_i$, in order to reduce stochasticity.

The running time to compute n here is roughly $N * k * PSRL(S, A, T)$

2 A Cheaper Approximation

We now present a procedure for Approximate Simulated Query Rollout (ASQR). To remove the need to run $PSRL$ in line 5 of SQR (Algorithm 1), we can instead:

1. Update the agent's posterior over reward functions using all the queries in its desired query set.

2. Do planning and compute expected performance under this updated posterior (subtracting the cost of all the simulated queries).

This only involves one iteration of planning, unlike $PSRL$, which performs planning numerous times, e.g. once per episode. This approach may underestimate performance because the actual $PSRL$ agent might not end up making all of the desired queries (and hence pay less query cost). Or it may over-estimate performance, since the $PSRL$ agent does not have the benefit of observing the queries up-front, and must make decisions with less information during learning. These differences would be especially large when many states or important states are not very reachable (and so would be queried more slowly or less often by the $PSRL$ agent).

3 Further Extensions

We can run the above algorithms at any time during learning, in order to dynamically adjust our desired query set based on new information we've acquired. At a high level, this is analogous to the way Thompson Sampling resamples throughout learning.

Since querying each state equally often is likely highly suboptimal, we'd like to consider a broader class of desired query sets, but the number of such sets grows exponentially in the total number of queries. We believe techniques for linear bandits might be used to specify the priority of each possible query at a

given moment, thus specifying a strict total order over desired query sets (with the smallest element being the queries already performed). Then we could use the above techniques to instead tune which index in this order to select as q_t^{max} at any given t.

The connection with linear bandits comes from viewing each state, action pair, (s, a), as one dimension of arm-space. Note that given transition operator $\mathcal{T}$, each policy, π induces a distribution over (s, a), and hence a point on the L_1 sphere in arm-space. Unlike a standard linear bandit, however, our queries are restricted to standard basis vectors, while only the arms corresponding to valid policies can ultimately be exploited. Furthermore, an agent must visit a state, action in order to query it (although notably, this is *not* the case for ASQR.