Lecture 5: Q-Learning

Chenglei Si, Goh Yong Liang
sichenglei1125@gmail.com

1 Value Function Learning Theory

1.1 Tabular Value Function Learning

Recall Value-iteration Algorithm:

Loop:

1. Set $Q(s,a) \leftarrow r(s,a) + \gamma E[V(s')]$

2. Set $V(s) \leftarrow \max_a Q(s,a)$

To determine if it will converge, we first define the Bellman Backup Operator B as follows:

$BV = \max_a r_a + \gamma T_a V$,

where r_a is the stacked vector of rewards t all states for action a, T_a is the matrix of transitions for a such that $T_{a,i,j} = p(s' = i | s = j, a)$.

We always have an optimal policy $V*$ as a fixed point of B:

$V^*(s) = \max_a r(s,a) + \gamma E[V^*(s')]$, so $V^* = BV^*$.

We can prove that value iteration reaches V^* because B is a contraction.

Contraction: for any V and $\overline{V}$, we have $\left\|BV - B\overline{V}\right\|_\infty \leq \gamma \left\|V - \overline{V}\right\|_\infty$

If we choose V^* as $\overline{V}$, since $BV^* = V^*$, we have: $\|BV - V^*\|_\infty \leq \gamma \|V - V^*\|_\infty$. So it will converge to V^*.

Note that:

1. A contraction in a norm is not necessarily a contraction in another norm.

2. Composition of two contractions in different norms does not necessarily give a contraction.

3. A contraction will give you a fixed point.

1.2 Non-Tabular Value Function Learning

Recall fitted value iteration Algorithm:

Loop:

1. Set $y_i \leftarrow \max_{a_i}(r(s_i, a_i) + \gamma E[V_\phi(s_i')])$

2. Set $\phi \leftarrow argmin_\phi \frac{1}{2} \sum_i \left\|V'(s) - y_i\right\|^2$

We define a new operator Π: $\Pi V = argmin_{V' \in \Omega} \frac{1}{2} \sum \left\|V'(s) - V(s)\right\|^2$, Π can be interpreted as a projection onto Ω in terms of L2 norm.

Then we can express fitted value iteration as:

Exploration in Computer Science Research: Deep Reinforcement Learning (CS 6101, 2019), National University of Singapore.

loop: $V \leftarrow \Pi B V$

While B is a contraction w.r.t. $\infty - norm$,Π is a contraction w.r.t. L2-norm, ΠB is not a ontraction of any kind. Hence, fitted value iteration does not converge.

Similar proof can be done for fitted Q iteration algorithm as well:

Recall fitted Q iteration Algorithm:

Loop:

1. Set $y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s_i')]$

2. Set $\phi \leftarrow argmin_\phi \frac{1}{2} \sum_i \left\| Q_\phi(s_i, a_i) - y_i \right\|^2$

We define B and Π slightly differently here:

B: $BQ = r + \gamma T \max_a Q$

Π: $\Pi Q = argmin_{Q' \in \Omega} \frac{1}{2} \sum_i \left\| Q'(s, a) - Q(s, a) \right\|^2$

Then fitted Q-iteration is: loop: $Q \leftarrow \Pi B Q$. ΠB is not a contraction of any kind, so it does not converge. The same applies to online Q-learning as well.

2 How to make Q-learning work

2.1 Problem 1: Correlated samples in online Q-learning

There are 2 main problems with online Q-iteration: 1. Sequential states are strongly correlated (and we don't want samples to be correlated). 2. Target value is always changing.

Solution 1: synchronized/asynchronous parallel Q-learning

Solution 2: Replay Buffers

full Q-learning with replay buffer:

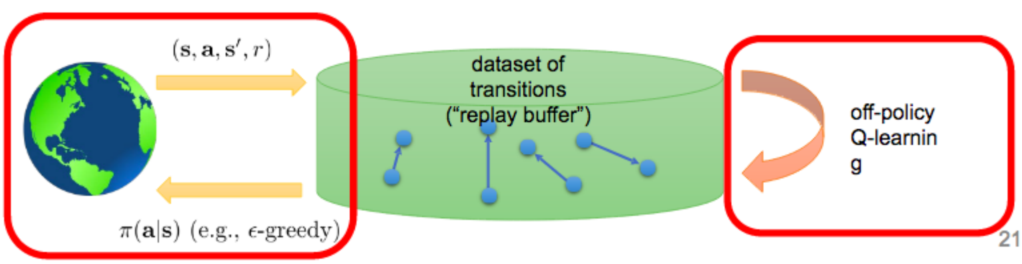
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)\}$ using some policy, add it to $\mathcal{B}$

$K\times$

2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)$ from $\mathcal{B}$

3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')])$

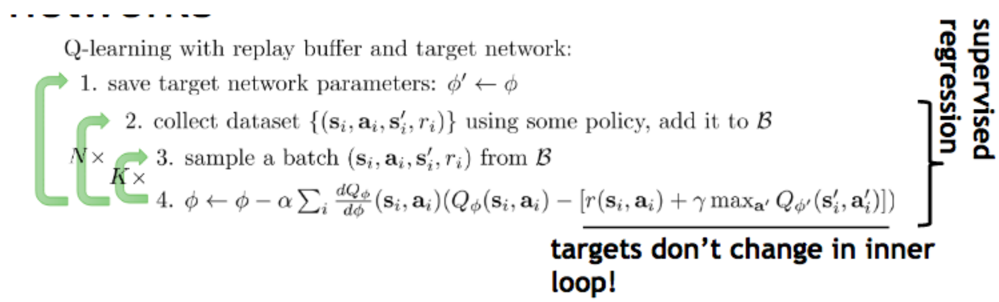
K = 1 is common, though larger K more efficient

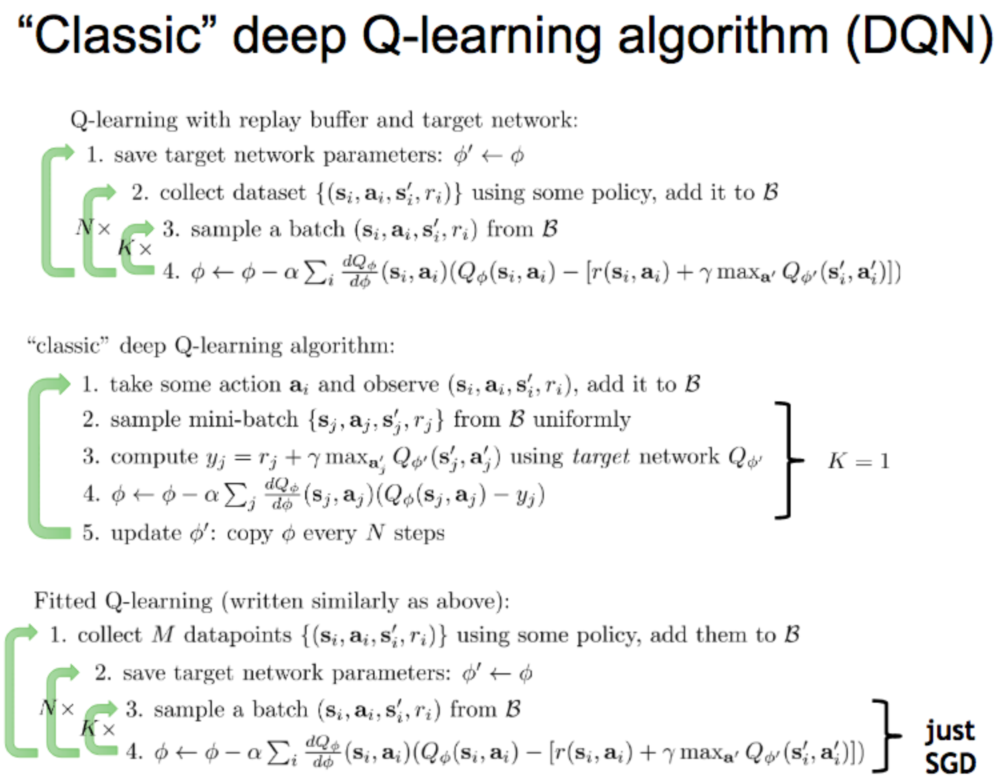


2.2 Problem 2: Moving Target in Regression

Q-learning is not exactly gradient descent because the target y depends on Q itself. In other words, the target keeps changing,

Solution: Q-learning with target networks





Other Solutions:

1. Prioritized Experience Replay:

Weight sampling from replay buffer by TD-error: $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, Correct for introduced bias via importance-sampling weights.

2. Hindsight Experience Replay:

Consider bit-flipping environment:

State space: S in $\{0, 1\}^n$ Action space: A = {0, 1, ..., n-1 }

Each episode is a uniformly sampled initial state, and target state, and the reward is -1 if not in the target state.

2.3 Problem 3: Overestimation in Q-learning

For random variables X_1, X_2: $E[max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$.

In target value $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$, $\max_{a'_j} Q_{\phi'}(s'_j, a'_j)$ overestimates the next value. Note that $\max_{a'_j} Q_{\phi'}(s'_j, a'_j) = Q_{\phi'}(s', argmax_{a'} Q_{\phi'}(s', a'))$ here both the value and action come from $Q_{\phi'}$. We want the noise to be decorrelated.

Solution: Double Q-Learning

Idea: Don't use the same network to to choose the action and evaluate value. In practice, just use the current and target network:

$y \leftarrow r + \gamma Q_{\phi'}(s', argmax_{a'} Q_{\phi}(s', a'))$

3 Q-Learning with continuous actions

In target value $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$, how do we perform the max?

3.1 Option 1: Optimization

Gradient based optimization (e.g., SGD) a bit slow in the inner loop. Simple Solution:

$max_a Q(s, a) \approx \max\{Q(s, a_1), ..., Q(s, a_N)\}$

More accurate solutions: Cross-Entropy Method, Covariance Matrix Adaptation Evolution Strategy

3.2 Option 2: Easily maximizable Q-functions

Use function class that is easy to optimize:

Normalized Advantage Functions. NAF represents the critic by a quadratic function with a negative curvature:

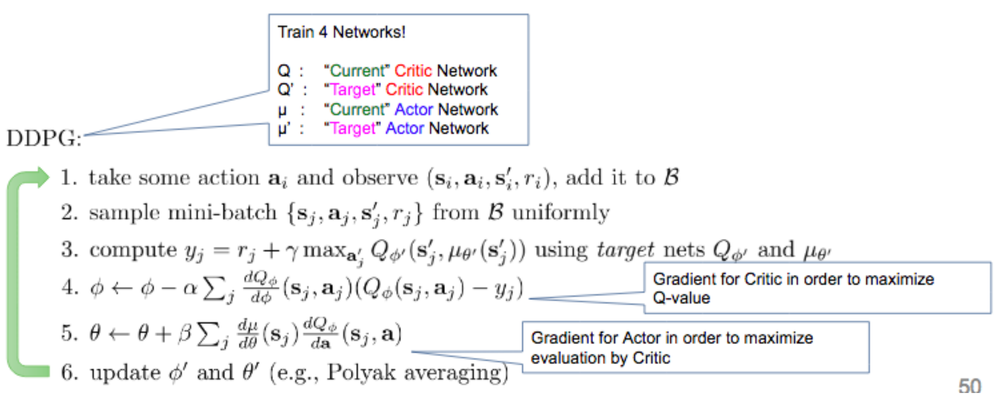
$Q_{\phi}(s, a) = -\frac{1}{2}(a - \mu_{\phi}(s))^T P_{\phi}(s)(a - \mu_{\phi}(s)) + V_{\phi}(s)$

$\text{argmax}_a Q_{\phi}(s, a) = \mu_{\phi}(s)$

$\max_a Q_{\phi}(s, a) = V_{\phi}(s)$

3.3 Option 3: learn an approximate maximizer

Basic idea: Train Neural Networks that approximates max function given state and possible actions.



3.4 Practical Tips

1. Q-learning takes some care to stabilize Test on easy, reliable tasks first, make sure your implementation is correct.

2. Large replay buffers help improve stability

3. It takes time, be patient – might be no better than random for a while.

4. Start with high exploration (epsilon) and gradually reduce.

5. Bellman error gradients can be exploded; clip gradients, or use Huber loss Huber loss is robust against outlier compared to squared loss

6. Double Q-learning helps a lot in practice, simple and no downsides.
7. N-step returns also help a lot, but have some downsides.
8. Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too.
9. Run multiple random seeds, it's very inconsistent between runs.

References