DDA4230 Reinforcement learning Discrete Q-learning

Lecture 11

Lecturer: Guiliang Liu *Scribe: Baoxiang Wang*

1 Goal of this lecture

In the last lecture of discrete MDPs we give an example of how algorithm could lift the requirement of estimating a world model. This is the first time we analyze model-free algorithms.

Suggested reading: *Is Q-learning provably efficient?* by Jin, Allen-Zhu, Bubeck, and Jordan.

2 Model-based v.s. model-free algorithms

A model-based algorithm is an algorithm that maintains an estimate of the model (the transition probability function and the reward function) and uses the model when interacting with the environment. This estimation could be in forms of point estimation or distribution estimation like posterior sampling. A model-free algorithm is an algorithm that does not estimate the world model. Bandit algorithms (ε-greedy, ETC, UCB, TS) and discrete RL algorithms (UCVI, PSRL) that are introduced in previous lectures are model-based algorithms.

If we have a good way (for example, if the generative model assumption is realistic) to estimate the model or we even have an access to it, then model-based and model-free methods will not make much a difference. But the problem is that we do not have a good way and even more so in the regime of large state and action spaces and continuous settings. We call the error induced by a wrongly estimated model as the *model bias*. In fact, in the 1980s reinforcement learning algorithms are referring to trial-and-error approaches that do not involve a model.

A possible source of confusion here is that the programmer or developer of the algorithm might know exactly how rewards and state transitions are designed or they even designed so themselves. But in model-free algorithms the agent will not be told any information about the world model and is forced to learn the policy and value alone. When designing algorithms the developer should also be aware of this, and avoid taking any part of the model into consideration, if they desire the algorithm to work in a wide range of tasks. In fact, many RL algorithm lack the capacity to generate the policy to new environments especially when the state space is large. This is one of the three major difficulties which we discussed in the first lecture.

In this lecture, we discuss the first model-free algorithm, Q-learning. The algorithm is a extension of the value iteration algorithm but it lifts the requirement of a model. We also give an introduction on how ε-greedy and UCB ways of exploration are incorporated into Q-learning. The latter obtains a $O(\sqrt{K})$ regret upper bound if we choose the parameters carefully.

3 Q-learning

We start with the value iteration algorithm and discuss how the model could be lifted.

Algorithm 1: Value iteration

Input: ϵ
For all states $s \in S$, $V'(s) \leftarrow 0$, $V(s) \leftarrow \infty$
while $\|V - V'\|_\infty > \epsilon$ **do**
  $V \leftarrow V'$
  For all states $s \in S$, $V'(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V(s')\right]$
$V^* \leftarrow V$ for all $s \in S$
$\pi^* \leftarrow \arg\max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^*(s')\right] \quad , \forall\, s \in S$
return $V^*(s),\ \pi^*(s)$ for all $s \in S$

The model appears in two places. One for the computing of the action value $R(s,a) + \sum_{s' \in S} P(s' \mid s,a) V(s')$, and one for the computing of the optimal policy through computing every $R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^*(s')$. Both places could remove the dependency on P by representing the action value directly (instead of relying on the state value). The terms $\sum_{s' \in S} P(s' \mid s,a) V(s')$ and $\sum_{s' \in S} P(s' \mid s,a) V^*(s')$ could be replaced by $Q(s,a)$ and $Q^*(s,a)$, respectively. Thus, maintaining the action value representation would be sufficient. The dependency on $R(s_t, a_t)$ could be removed by the sample r_t at step t, which forms a approximate contraction.

Another generalization of the algorithm is the introduction of the step size. Instead of updating the action value Q to be the target action value described by the Bellman optimality equation, the update only takes at α portion of the action value while the $1 - \alpha$ portion of the action value remain the same. An alternative formulation of this update is written as $Q'(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a \in \mathcal{A}} Q(s', a') - Q(s,a) \right)$, where the term $\delta_t = r + \gamma \max_{a \in \mathcal{A}} Q(s', a') - Q(s,a)$ is known as the temporal difference (TD) error.

Algorithm 2: Q-learning

Input: ϵ, α
For all $(s,a) \in \mathcal{S} \times \mathcal{A}$, $Q'(s,a) \leftarrow 0$, $Q(s,a) \leftarrow \infty$
while $\|Q - Q'\|_\infty > \epsilon$ **do**
  $Q \leftarrow Q'$
  Sample a trajectory τ from the policy $\pi(a \mid s) = \arg\max_{a \in A} Q(s,a)$
  For all state-action-reward-state tuple $(s, a, r, s') \in \tau$,
    $Q'(s,a) \leftarrow (1 - \alpha) Q(s,a) + \alpha \max_{a' \in \mathcal{A}} [r + \gamma Q(s', a')]$
$Q^* \leftarrow Q$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$
$\pi^* \leftarrow \arg\max_{a \in \mathcal{A}} Q(s,a)$
return $Q^*(s,a),\ \pi^*(s)$ for all s, a

With Q^*, the agent does not even have to do a one-step-ahead search and for any state s, it can simply find any action that maximizes $q^*(s, a)$. In this way, the action-value function effectively caches the results of all one-step-ahead searches. This is though at the cost of representing a function of state-action pairs, instead of just of states, the optimal value function. This makes the storage requirement increasing from n real numbers to nm real numbers.

4 Exploration and ε-greedy Q-learning

The problem with the above version of Q-learning is that the trajectory is sampled is subject to the current policy and thereof the current value estimation. It is possible that the algorithm stuck at a suboptimal action value estimate and does not update itself. It is also possible that some state are never explored with some initialization of the policy and value functions.

A simple way of involving exploration is to force the algorithm to select a random action with probability ε, which is called ε-greedy Q-learning. This ε could delay over the iterations, as is in the ε-greedy algorithm for multi-armed bandits. In practice, the algorithm sees some use in simple RL tasks that otherwise could not be solved without exploration.

Algorithm 3: Q-learning with ε-greedy exploration

Input: ϵ, α

For all $(s, a) \in \mathcal{S} \times \mathcal{A}$, $Q'(s, a) \leftarrow 0$, $Q(s, a) \leftarrow \infty$

while $\|Q - Q'\|_\infty > \epsilon$ **do**

$Q \leftarrow Q'$

Sample a trajectory τ from the policy

$$\pi(a \mid s) = \begin{cases} \arg\max_{a \in A} Q(s, a) & \text{with probability } 1 - \varepsilon \\ \text{random action} & \text{with probability } \varepsilon \end{cases}$$

For all state-action-reward-state tuple $(s, a, r, s') \in \tau$,

$Q'(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \max_{a' \in \mathcal{A}} [r + \gamma Q(s', a')]$

$Q^* \leftarrow Q$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$

$\pi^* \leftarrow \arg\max_{a \in \mathcal{A}} Q(s, a)$

return $Q^*(s, a)$, $\pi^*(s)$ for all s, a

5 Q-learning with UCB

We present another variant of Q-learning with UCB exploration. This algorithm is the first Q-learning variant that is with a rigorous regret guarantee of $\sqrt{K}$. To present the theoretical results, we again describe the algorithm in the episodic MDP formulation. While each of the "sample a trajectory" step is an episode in the setting. We again use $Q_h(s, a)$ as the

time-dependent action value function, which is necessary when the horizon of each episode is constant.

Despite that the algorithm is still time-dependent, the inner loop of h starts with $H-1$ as like dynamic programming.

Algorithm 4: Q-learning with UCB exploration

Input: α: adaptive step size; δ: confidence level
Initialize $Q_h(s,a) \leftarrow 0$, $N_h(s,a) \leftarrow 0$ for all $h \in [H]$, $k \leftarrow 0$
while $k \leq K-1$ **do**
  Start an episode with s_0
  for $h \leq H-1, \ldots, 0$ **do**
    Take action $a_h^k = \arg\max_a Q_h(s_h^k, a)$ and observe s_{h+1}^k
    $N_h(s_h^k, a_h^k) \leftarrow N_h(s_h^k, a_h^k) + 1$
    Update the action value as

$$Q_h(s_h^k, a_h^k) \leftarrow (1-\alpha) Q_h(s_h^k, a_h^k) + \alpha \left[r_h(s_h^k, a_h^k) + V_{h+1}(s_{h+1}^k) + c\sqrt{\frac{H^3 \log(nmHK/\delta)}{N_h(s_h^k, a_h^k)}} \right]$$

    Update the state value as

$$V_h(s_h^k) = \min\left\{ \max_a Q_h(s_h^k, a), H \right\}$$

  $k \leftarrow k+1$
$Q_h^* \leftarrow Q_h$
$\pi_h^* \leftarrow \arg\max_a Q_h(s,a)$
return Q_h^*, π_h^* for all $h \in [H]$

Theorem 1 *By choosing $\alpha = \frac{H+1}{H+N}$ with the visitation count $N = N_h(s_h^k, a_h^k)$, there exists an absolute constant c such that with probability at least $1-p$ the regret of Q-learning with UCB exploration is at most $O(\sqrt{nmH^5 K \log(nmHK/\delta)})$.*

The proof relies on the cast of the variables into a filtration and therefore the use of the Azuma-Hoeffding inequality (introduced in LN3). For those students that are interested in the proof we could host you with a presentation of it.

Acknowledgement

This lecture notes partially use material from *Reinforcement learning: An introduction*, *Reinforcement learning: Theory and algorithms*, the paper referred in suggested reading, and *CS234: Reinforcement learning* from Stanford.

The majority of the midterm exam of DDA4230 covers up to this lecture (LN12).