DDA4230 Reinforcement learning Trial and error

Lecture 14

Lecturer: Baoxiang Wang *Scribe: Baoxiang Wang*

1 Goal of this lecture

In this lecture, we investigate trial-and-error algorithms, which invokes the name *reinforcement learning* of this course.
Suggested reading: Chapter 5, 6 and 13 of *Reinforcement learning: An introduction*;

2 Model-free control

In the last lecture, we discussed how to evaluate a given policy assuming that we do not know how the world works (that is, by only interacting with the environment). This gives us our model-free policy evaluation methods, including Monte-Carlo policy evaluation and TD learning. In this lecture, we will discuss model-free control where we learn good policies under the same constraints (only interactions, no knowledge of reward structure or transition probabilities). This framework is important in two types of domains:

1. When the MDP model is unknown, but we can sample trajectories from the MDP;
2. When the MDP model is known but computing the value function via our model-based control methods is infeasible due to size of the domain, but we can sample trajectories from the MDP.

In this lecture, we will still restrict ourselves to the setting of discrete RL, where we can represent each state or state-action value as an element of a lookup table, but some elements of deep RL can be mentioned. In the next lecture, we will be re-examining the algorithms from LN13 and LN14 under the value function approximation, which involves attempts to fit a function to the state value function or the action value function.

2.1 Generalized policy iteration

Recall that we discussed the policy iteration algorithm with a known model before, which we rewrite in Algorithm 1.

In LN13, we introduced methods for model-free policy evaluation, so we can complete line 4 in a model-free way. However, in order to make this entire algorithm model-free, we must also find a way to do line 5, the policy improvement step, in a model-free way. By definition, we have $Q^\pi(s,a) = \mathbb{E}[\mathcal{R}(s,a)] + \gamma \sum_{s' \in S} \mathbb{P}(s'|s,a) V^\pi(s')$. Thus, we can get a model-free policy iteration algorithm (Algorithm 2) by substituting this value into the model-based policy iteration algorithm and using action values throughout.

There are a few caveats to this algorithm due to the substitution we made in line 5.

Algorithm 1: Policy iteration

Input: $\mathcal{M}, \epsilon$
$\pi \leftarrow$ Randomly choose a policy $\pi \in \Pi$
while *true* **do**
  $V^\pi \leftarrow$ POLICY EVALUATION $(\mathcal{M}, \pi, \epsilon)$
  $\pi^*(s) \leftarrow \arg\max_{a \in A} \mathbb{E}[R(s,a)] + \gamma \sum_{s' \in S} \mathbb{P}(s'|s,a) V^\pi(s')\,,\ \forall\ s \in S$
  if $\pi^*(s) = \pi(s)$ **then**
    break
  else
    $\pi \leftarrow \pi^*$
$V^* \leftarrow V^\pi$
return $V^*(s),\ \pi^*(s)$ for all $s \in S$

1. If policy π is deterministic or does not take every action a with some positive probability, then we cannot actually compute the argmax in line 5.

2. The policy evaluation algorithm gives us an estimate of Q^π, so it is not clear whether line 5 will monotonically improve the policy like in the model-based case.

Algorithm 2: Model-free generalized policy iteration

Input: ϵ
$\pi \leftarrow$ Randomly choose a policy $\pi \in \Pi$
while *true* **do**
  $Q^\pi \leftarrow$ MODEL-FREE POLICY EVALUATION (π, ϵ)
  $\pi^*(s) \leftarrow \arg\max_{a \in A} Q^\pi(s,a)\,,\ \forall\ s \in S$
  if $\pi^*(s) = \pi(s)$ **then**
    break
  else
    $\pi \leftarrow \pi^*$
$Q^* \leftarrow Q^\pi$
return $Q^*(s,a),\ \pi^*(s)$ for all $s \in S,\ a \in A$

2.2 Importance of exploration

2.2.1 Exploration

In the previous section, we saw that one caveat to the model-free policy iteration algorithm is that the policy π needs to take every action a with some positive probability, so the value of each state-action pair can be determined. In other words, the policy π needs to explore actions, even if they might be suboptimal with respect to our current Q-value estimates.

2.2.2 ϵ-greedy policies

In order to explore actions that are suboptimal with respect to our current Q-value estimates, we will need a systematic way to balance exploration of suboptimal actions with exploitation of the optimal, or greedy, action. One naive strategy is to take a random action with small probability and take the greedy action the rest of the time. This type of exploration strategy is called an ϵ-greedy policy. Mathematically, an ϵ-greedy policy with respect to the state-action value $Q^\pi(s,a)$ takes the form

$$\pi(a \mid s) = \begin{cases} \text{Uniform}(\mathcal{A}) & \text{with probability } \epsilon \\ \arg\max_a Q^\pi(s,a) & \text{with probability } 1-\epsilon\,. \end{cases}$$

2.2.3 Monotonic ϵ-greedy policy improvement

Recall the property of policy improvement that if we take the greedy action with respect to the current values and then follow a policy π thereafter, this policy is an improvement to the policy π. A natural question then is whether an ϵ-greedy policy within the family of ϵ-random policies is an improvement on policy π. This would help us address our second caveat of the generalized policy iteration algorithm, Algorithm 2. Fortunately, there is an analogue of the argument in policy improvement for the ϵ-greedy policy, which we state and derive below.

Lemma 1 (Monotonic ϵ-greedy policy improvement) *Let π_i be an ϵ-greedy policy. Then, the ϵ-greedy policy with respect to Q^{π_i}, denoted π_{i+1}, is a monotonic improvement on policy π. In other words, $V^{\pi_{i+1}} \geq V^{\pi_i}$.*

Proof: We first show that $\mathbb{E}_{a' \sim \pi_{i+1}(s)}[Q^{\pi_i}(s,a')] \geq V^{\pi_i}(s)$ for all states s. In fact,

$$\begin{aligned}
\mathbb{E}_{a' \sim \pi_{i+1}(s)}[Q^{\pi_i}(s,a')] &= \sum_{a \in \mathcal{A}} \pi_{i+1}(a \mid s) Q^{\pi_i}(s,a) \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} Q^{\pi_i}(s,a) + (1-\epsilon) \max_{a'} Q^{\pi_i}(s,a') \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} Q^{\pi_i}(s,a) + (1-\epsilon) \max_{a'} Q^{\pi_i}(s,a') \frac{1-\epsilon}{1-\epsilon} \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} Q^{\pi_i}(s,a) + (1-\epsilon) \max_{a'} Q^{\pi_i}(s,a') \sum_{a \in A} \frac{\pi_i(a \mid s) - \frac{\epsilon}{|A|}}{1-\epsilon} \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} Q^{\pi_i}(s,a) + (1-\epsilon) \sum_{a \in A} \frac{\pi_i(a \mid s) - \frac{\epsilon}{|A|}}{1-\epsilon} \max_{a'} Q^{\pi_i}(s,a') \\
&\geq \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} Q^{\pi_i}(s,a) + (1-\epsilon) \sum_{a \in \mathcal{A}} \frac{\pi_i(a \mid s) - \frac{\epsilon}{|\mathcal{A}|}}{1-\epsilon} Q^{\pi_i}(s,a) \\
&= \sum_{a \in \mathcal{A}} \pi_i(a \mid s) Q^{\pi_i}(s,a) \\
&= V^{\pi_i}(s)\,.
\end{aligned}$$

The first equality follows from the fact that the first action we take is with respect to policy π_{i+1}, then we follow policy π_i thereafter. The fourth equality follows because $1 - \epsilon = \sum_a \left[\pi_i(a \mid s) - \frac{\epsilon}{|\mathcal{A}|}\right]$.

Now from the lemma, we have that $\mathbb{E}_{a' \sim \pi_{i+1}(s)}[Q^{\pi_i}(s, a')] \geq V^{\pi_i}(s)$, which implies that $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$ for all $s \in \mathcal{S}$, as desired. □

Thus, the monotonic ϵ-greedy policy improvement shows us that our policy does in fact improve if we act ϵ-greedy on the current ϵ-greedy policy.

2.2.4 Greedy in the limit of exploration

We introduced ϵ-greedy strategies above as a naive way to balance exploration of new actions with exploitation of current knowledge. An easy argument find that this naive approach is insufficient to guarantee the convergence. We can further refine this balance by introducing a new class of exploration strategies that allow us to make convergence guarantees about our algorithms. This class of strategies is called greedy in the limit of infinite exploration (GLIE).

Definition 1 (Greedy in the limit of infinite exploration) *A policy π is greedy in the limit of infinite exploration if it satisfies the following two properties:*

1. *All state-action pairs are visited for infinitely many times, i.e., for all $s \in \mathcal{S}, a \in \mathcal{A}$,*

$$\lim_{k \to \infty} N_k(s, a) \to \infty \text{ with probability } 1\,,$$

   where $N_k(s, a)$ is the number of times action a is taken at state s up to and including episode k.

2. *The behavior policy converges to the policy that is greedy with respect to the learned Q-function, i.e., for all $s \in \mathcal{S}, a \in \mathcal{A}$,*

$$\lim_{k \to \infty} \pi_k(a \mid s) = \arg\max_a Q(s, a) \text{ with probability } 1\,.$$

One example of a GLIE strategy is an ϵ-greedy policy where ϵ is decayed to zero with $\epsilon_k = O(1/k)$, where k is the episode number. We can see that since $\sum_{k=1}^{K} \epsilon_k = O(\log K)$, we will explore each action for infinitely many times, hence satisfying the first GLIE condition (we leave the rigorous proof to the reader). Since $\epsilon_k \to 0$ as $k \to \infty$, we also have that the policy is greedy in the limit, hence satisfying the second GLIE condition.

2.3 Monte-Carlo control

Now, we will incorporate the exploration strategies described above with our model-free policy evaluation algorithms to give us some model-free control methods. The first algorithm that we discuss is the Monte-Carlo online control algorithm. In Algorithm 3, we give the formulation for first-visit online Monte-Carlo control. An equivalent formulation for every-visit control is given by not checking for the first visit condition.

Now, as stated before, GLIE strategies can help us arrive at convergence guarantees for our model-free control methods. In particular, we have the following statement.

Algorithm 3: Online Monte-Carlo control

Initialize $Q(s,a) = 0$, $Returns(s,a) = 0$ for all $s \in S, a \in A$
Set $k \leftarrow 1$
while *true* **do**
  Sample k-th episode $\{s_{t,k}, a_{t,k}, r_{t,k}\}_{t \in [H]}$ under policy π
  for $t = 1, \ldots, H$ **do**
    if *First visit to* (s,a) *in episode* k **then**
      Append $\sum_{t'=t}^{H} r_{t',k}$ to $Returns(s_t, a_t)$
      $Q(s_t, a_t) \leftarrow \text{average}(Returns(s_t, a_t))$
  $k \leftarrow k + 1$, $\epsilon = \frac{1}{k}$
  $\pi_k = \epsilon$-greedy with respect to Q
Return Q, π_k

Lemma 2 *GLIE Monte-Carlo control converges to the optimal state-action value function. That is* $Q(s,a) \rightarrow Q^*(s,a)$.

In other words, if the ϵ-greedy strategy used in Algorithm 3 is GLIE, then the Q value derived from the algorithm will converge to the optimal Q function. We leave the proof to the reader.

2.4 Temporal-difference methods for control

Recall that we also introduced TD(0) as a method for model-free policy evaluation in LN13. Now, we can build on TD(0) with our ideas of exploration to get TD-style model-free control. There are two methods of doing this, on-policy and off-policy. We first introduce the on-policy method in Algorithm 4, infamously known as the state action reward state action (SARSA) algorithm.

Algorithm 4: SARSA

Input: ϵ, α_t
Initialize $Q(s,a)$ for all $s \in S, a \in A$ arbitrarily except $Q(terminal, \cdot) = 0$
$\pi \leftarrow \epsilon$-greedy policy with respect to Q
for *each episode* **do**
  $t \leftarrow 1$
  Set s_1 as the starting state
  Choose action a_1 from policy $\pi(s_1)$
  while *until episode terminates* **do**
    Take action a_t and observe reward r_t and next state s_{t+1}
    Choose action a_{t+1} from policy $\pi(s_{t+1})$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
    $\pi \leftarrow \epsilon$-greedy with respect to Q
    $t \leftarrow t + 1$
Return Q, π

We can see the policy evaluation update takes place in line 11, while the policy improvement is at line 12. SARSA gets its name from the parts of the trajectory used in the update equation. We can see that to update the Q-value at state-action pair (s, a), we need the reward, next state and next action, thereby using the values (s, a, r, s', a'). SARSA is an on-policy method because the actions a and a' used in the update equation are both derived from the policy that is being followed at the time of the update.

Just like in Monte Carlo, we can arrive at the convergence of SARSA given one extra condition on the step-sizes denoted in the following lemma.

Lemma 3 *SARSA for finite-state and finite-action MDPs converges to the optimal action-value, i.e., $Q(s, a) \to Q^*(s, a)$, if the following two conditions hold:*

1. *The sequence of policies π from is GLIE*
2. *The step-sizes α_t satisfy the Robbins-Munro sequence such that*

$$\sum_{t=1}^{\infty} \alpha_t = \infty\,, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty\,.$$

We leave the proof to the reader. We also leave the following questions to the reader. What is the benefit to performing the policy improvement step after each update in line 11 of Algorithm 4? What would be the benefit to performing the policy improvement step less frequently? Recall that regret bounds of the ε-greedy algorithm has been derived. What is the connection between regret and convergence?

2.5 Importance sampling for off-policy TD

Before diving into off-policy TD-style control, let's first take a step back and look at one way to do off-policy TD policy evaluation. Recall that our TD update took the form

$$V(s) \to V(s) + \alpha(r + \gamma V(s') - V(s))\,.$$

Suppose that like in off-policy Monte-Carlo policy evaluation, we have data from a policy π_b, and we want to estimate the value of policy π_e. Then much like in the Monte-Carlo policy evaluation case, we can weight the target by the ratio of the probability of seeing the sample in the behavior policy and the evaluated policy via importance sampling. This new update then takes the form

$$V^{\pi_e}(s) \to V^{\pi_e}(s) + \alpha\left(\frac{\pi_e(a \mid s)}{\pi_b(a \mid s)}(r + \gamma V^{\pi_e}(s') - V^{\pi_e}(s))\right).$$

Notice that because we only use one trajectory sample instead of sampling the entire trajectory like in Monte Carlo, we only incorporate the likelihood ratio from one step. For the same reason, this method also has significantly lower variance than Monte Carlo.

Additionally, π_b does not need to be the same at each step, but we do need to know the probability for every step. As is in Monte Carlo, we need the two policies to have the same support. That is, if $\pi_e(a \mid s) \cdot V^{\pi_e}(s') > 0$, then $\pi_b(a \mid s) > 0$.

2.6 Q-learning

Now, we return to finding an off-policy method for TD-style control. In the above formulation, we again leverage importance sampling, but in the control case, we do not need to rely on this. Instead, we can maintain the Q estimates and bootstrap the value of the best future action. Our SARSA update took the form

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right) ,$$

but we can instead bootstrap the Q value at the next state to get the following update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right) .$$

This gives rise to Q-learning, which is detailed in Algorithm 5. Now that we take a maximum over the actions at the next state, this action is not necessarily the same as the one we would derive from the current policy. Therefore, Q-learning is considered an off-policy algorithm.

Algorithm 5: Q-learning with ϵ-greedy exploration

Input: ϵ, α, γ
Initialize $Q(s, a)$ for all $s \in S, a \in A$ arbitrarily except $Q(terminal, \cdot) = 0$
$\pi \leftarrow \epsilon$-greedy policy with respect to Q
for *each episode* **do**
  $t \leftarrow 1$
  Set s_1 as the starting state
  while *until episode terminates* **do**
    Sample action a_t from policy $\pi(s_t)$
    Take action a_t and observe reward r_t and next state s_{t+1}
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$
    $\pi \leftarrow \epsilon$-greedy policy with respect to Q
    $t \leftarrow t + 1$
return Q, π

3 Maximization bias

Finally, we are going to discuss a phenomenon known as the maximization bias. We will first examine maximization bias in a small example.

3.1 Example: Game of coins

Suppose there are two identical fair coins, but we do not know that they are fair or identical. If a coin lands on heads, we get one dollar and if a coin lands on tails, we lose a dollar. We ask the following two questions.

1. Which coin will yield more money for future flips?
2. How much can we expect to win/lose per flip using the coin from question 1?

In an effort to answer this question, we flip each coin once. We then pick the coin that yields more money as the answer to question 1. We answer question 2 with however much that coin gave us. For example, if coin 1 landed on heads and coin 2 landed on tails, we would answer question 1 with coin 1, and question 2 with one dollar.

We examine the possible scenarios for the outcome of this procedure. If at least one of the coins is heads, then our answer to question 2 is one dollar. If both coins are tails, then our answer is negative one dollar. Thus, the expected value of our answer to question 2 is $\frac{3}{4} \times (1) + \frac{1}{4} \times (-1) = 0.5$. This gives us a higher estimate of the expected value of flipping the better coin than the true expected value of flipping that coin. In other words, we are systematically going to think the coins are better than they actually are.

This problem comes from the fact that we are using our estimate to both choose the better coin and estimate its value. We can alleviate this by separating these two steps. One method for doing this would be to change the procedure as follows: After choosing the better coin, flip the better coin again and use this value as your answer for question 2. The expected value of this answer is now 0, which is the same as the true expected value of flipping either coin.

3.2 Maximization bias in reinforcement learning

We now formalize what we saw above in the context of a one-state MDP with two actions. Suppose we are in state s and have two actions a_1 and a_2, both with mean reward 0. Then, we have that the true values of the state as well as the state-action pairs are zero. That is, $Q(s, a_1) = Q(s, a_2) = V(s) = 0$. Suppose that we have sampled some rewards for taking each action so that we have some estimates for the state-action values, $\hat{Q}(s, a_1), \hat{Q}(s, a_2)$. Suppose further that these samples are unbiased because they were generated using a Monte-Carlo method. In other words, $\hat{Q}(s, a_i) = \frac{1}{N(s,a_i)} \sum_{j=1}^{N(s,a_i)} r_j(s, a_i)$, where $N(s, a_i)$ is the number of samples for the state-action pair (s, a_i). Let $\hat{\pi} = \arg\max_a \hat{Q}(s, a)$ be the greedy policy with respect to our Q value estimates. Then, we have that

$$
\begin{aligned}
\hat{V}(s) &= \mathbb{E}[\max(\hat{Q}(s, a_1), \hat{Q}(s, a_2)] \\
&\geq \max(\mathbb{E}[\hat{Q}(s, a_1)], \mathbb{E}[\hat{Q}(s, a_2)]) && \text{by Jensen's inequality} \\
&= \max(0, 0) && \text{by unbiased estimates} \\
&= 0 = V^*(s)\,.
\end{aligned}
$$

Thus, our state value estimate is at least as large as the true value of state s, so we are systematically overestimating the value of the state in the presence of finite samples.

3.3 Double Q-learning

As we saw in the previous subsection, the state values can suffer from maximization bias as well when we have finitely many samples. As we discussed in the coin example, decoupling taking the max and estimating the value of the max can get rid of this bias. In Q-learning, we can maintain two independent unbiased estimates, Q_1 and Q_2 and when we use one to select the maximum, we can use the other to get an estimate of the value of this maximum. This gives rise to double Q-learning which is detailed in algorithm 6. When we refer to the

ϵ-greedy policy with respect to $Q_1 + Q_2$ we mean the ϵ-greedy policy where the optimal action at state s is equal to $\arg\max_a Q_1(s,a) + Q_2(s,a)$.

Algorithm 6: Double Q-learning

Input: ϵ, α, γ
Initialize $Q_1(s,a), Q_2(s,a)$ for all $s \in S, a \in A$ arbitrarily
$t \leftarrow 0$
$\pi \leftarrow \epsilon$-greedy policy with respect to $Q_1 + Q_2$
while *true* **do**
  Sample action a_t from policy π at state s_t
  Take action a_t and observe reward r_t and next state s_{t+1}
  if *with 0.5 probability* **then**
    $Q_1(s_t,a_t) \leftarrow Q_1(s_t,a_t) + \alpha(r_t + \gamma Q_2(s_{t+1}, \arg\max_{a'} Q_1(s_{t+1},a')) - Q_1(s_t,a_t))$
  else
    $Q_2(s_t,a_t) \leftarrow Q_2(s_t,a_t) + \alpha(r_t + \gamma Q_1(s_{t+1}, \arg\max_{a'} Q_2(s_{t+1},a')) - Q_2(s_t,a_t))$
  $\pi \leftarrow \epsilon$-greedy policy with respect to $Q_1 + Q_2$
  $t \leftarrow t + 1$
return $\pi, Q_1 + Q_2$

On some tasks, double Q-learning can significantly speed up training time by eliminating suboptimal actions more quickly than normal Q-learning. *Reinforcement learning: An introduction* has a nice example of this in a toy MDP in section 6.7.

Acknowledgement

This lecture notes partially use material from *Reinforcement learning: An introduction* and *CS234: Reinforcement learning* from Stanford.