Tabular Reinforcement Learning

Federico Arangath

26 February 2024

Foundations of Reinforcement Learning

The Reinforcement Learning Problem

- Differently from what we have seen last week, in classical RL settings, we do not have access to the transition probabilities and the distribution of the rewards
- Hence, we cannot derive the optimal policy/Value function using Value Iteration
- We thus need to find algorithms that either learn the dynamics of the environment or find the optimal value function without estimating the transitions

Model Based vs Model Free Methods I

- **Model Based**: Learn the transition dynamics during training, optimize the objective function on the learned MDP
- **Model Free**: Optimize the objective function without estimating the transition dynamics

Model Based vs Model Free Methods II

- **Model Based**: need to store a model of the transition dynamics and the rewards (**Memory Inefficient**) but allows to do **planning** if the model is good enough (which translates into more **Sample Efficient**)
- **Model Free**: more Memory Efficient but cannot do planning (hence more Sample Inefficient)

In this lecture we are going to focus on Model Free approaches, whereas we are going to go in more detail in Model Based methods in Lecture 4

Estimation of the Returns

- **Recall**: We defined $V_\gamma^\infty(s) = \mathbb{E}_\pi\left[\sum_{t=1}^\infty \gamma^{(t-1)} R_t \,\middle|\, s\right]$
- The goal is still to maximize the Value function in each state
- **Problem**: dynamic is unknown, hence we cannot calculate the expectation or the value function in closed form
- **Idea**: Estimate the value function or Q-function in each state

Monte-Carlo Estimation: Idea

- **Observation**: We can interpret the Value Function as an expectation of cumulative rewards over all the possible infinite trajectories beginning at a given state
- **Idea**: As in MC estimation, we want to simulate a few trajectories and estimate the value function by averaging their cumulative reward
- **Note**: We cannot simulate infinite trajectories and hence we are introducing a second approximation by fixing the episode length

Monte-Carlo Estimation: Algorithm

Suppose we fix a policy π, we want to estimate $V_{\gamma}^{(\infty)}(\pi, s)$, which we denote by $V(s)$ for notational simplicity here

Monte Carlo Method

```
for each episode do
    Generate an episode $\tau = \{s_0, a_0, r_0, s_1, \ldots\}$ following $\pi$
    for each state $s_t$ do
        Compute return $G_t = r_t + \gamma r_{t+1} + \cdots$
        Update counter $n_{s_t} \leftarrow n_{s_t} + 1$
        Update $V(s_t) \leftarrow V(s_t) + \frac{1}{n_{s_t}}(G_t - V(s_t))$
    end for
end for
```

Monte-Carlo Estimation: Pros and Cons

- **Pros**:
  - Unbiased estimator
  - Easy to implement and interpret
- **Cons**:
  - Very high variance
  - High computational cost

TD-Learning: Idea

- To decrease the computational complexity of estimating the value function, we can take advantage of its recursive formulation

$$V_\gamma^{(\infty)}(\pi, s) = \sum_a \pi(a|s) \left\{ r_s^a + \gamma \sum_{s'} p_{s\to s'}^a V_\gamma^{(\infty)}(\pi, s') \right\}$$

- **Idea**: Approximating with $V_\gamma^{(\infty)}(\pi, s) \approx r_s^a + \gamma V_\gamma^{(\infty)}(\pi, s')$ by sampling an action $a \sim \pi(\cdot|s)$ and a state $s' \sim p_{s\to\cdot}^a$.

TD-Learning: Algorithm

Again, suppose we fix a policy π, we want to estimate $V_\gamma^{(\infty)}(\pi, s)$, which we denote by $V(s)$ for notational simplicity here. In particular, we denote $V(s_t)$ to be the current estimate of state s_t. Note that each time we visit a state, we are updating the estimate of the corresponding value function.

TD Learning / TD(0)

```
for each step of an episode $\tau$ do
    Observe $(s_t, a_t, r_t, s_{t+1})$ following $\pi$
    Update $V(s_t) \leftarrow V(s_t) + \alpha_t(r_t + \gamma V(s_{t+1}) - V(s_t))$
end for
```

α_t is called the **learning rate** and is usually chosen to be a vanishing sequence

Bootstrapping

- **Note**: Instead of directly estimating $V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$ as in the original idea, we instead use $V(s_t) \leftarrow (1 - \alpha_t)V(s_t) + \alpha_t(r_t + \gamma V(s_{t+1}))$
- **Note**: The new estimate is a convex combination of the current estimate and the data observed in the current transition
- Whenever our estimate is constructed or depends on other/previous estimates of the quantity we want to approximate, we call it a **Bootstrapping Estimate**

TD estimates vs MC Estimation

- **MC estimates**:
  - **Unbiased**
  - **High Variance** since the estimate depends on samples from a long trajectory
  - The estimate depends exclusively on the collected rewards during the episodes
- **TD estimates**:
  - **Biased** since it is constructed via a convex combination
  - **Lower Variance** since the estimate depends on a sample from a single step
  - The estimate is constructed via bootstrapping, in particular using estimates of the value function and not the collected rewards exclusively

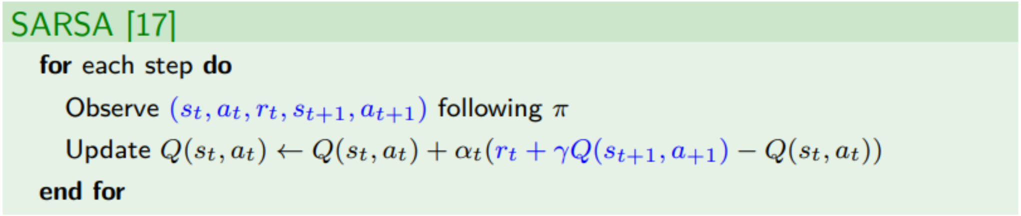
SARSA: Idea

- **Recall**: In Value Iteration or Policy Iteration, we often need the Q-function to then obtain the greedy policy or the optimal policy
- We want to use a similar approach to TD-learning but for Q-function
- Again, we leverage the recursive formulation of the Q-function

$$Q_\gamma^\infty(\pi, s, a) = r_s^a + \gamma \sum_{s'} p_{s\to s'}^a \sum_{a'} \pi(a'|s') Q_\gamma^\infty(\pi, s', a')$$

SARSA: Algorithm

- Similarly to TD Learning, we approximate the Q-function by sampling an action $a_t \sim \pi(\cdot|s_t)$ and a state $s_{t+1} \sim p^{a_t}_{s_t \to \cdot}$. However, note that here we also need to sample $a_{t+1} \sim \pi(\cdot|s_{t+1})$!!
- Again, we use bootstrapping to reduce variance



SARSA [17]

```
for each step do
    Observe $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ following $\pi$
    Update $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(r_t + \gamma Q(s_{t+1}, a_{+1}) - Q(s_t, a_t))$
end for
```

Towards Finding the Optimal Policy I

- **Recall**: Our end-goal is to find the optimal policy to deploy in the environment
- For now, we have only looked at how to approximate the value function or the Q-function of a fixed policy
- Can we instead estimate the optimal Q-function directly?

Towards Finding the Optimal Policy II

- **Recall**: Our end-goal is to find the optimal policy to deploy in the environment
- For now, we have only looked at how to approximate the value function or the Q-function of a fixed policy
- Can we instead estimate the optimal Q-function directly?
- Yes! This is the Key feature of Q-learning

Q-Learning I

- **Recall**: Bellman optimality equation

$$Q_\gamma^\infty(\pi^*, s, a) = r_s^a + \sum_{s'} p_{s \to s'}^a \max_{a' \in A} Q_\gamma^\infty(\pi^*, s', a')$$

- **Note**: This equation strictly resembles the recursive formulation that we used for TD-Learning and SARSA. Can we leverage the same techniques we used there in this setting?

Q-Learning II

- **Idea**: Approximate $Q_\gamma^\infty(\pi^*, s, a) \approx r_s^a + \max_{a' \in A} Q_\gamma^\infty(\pi^*, s', a')$ by sampling $s' \sim p_{s\rightarrow}^a$.
- **Note**: here, differently from SARSA, we do not need to sample an action from the policy π^* since we have a maximum operation and not an expectation. This "saves" us as we do not know π^* and hence we cannot sample from it
- We will again make use of bootstrapping to reduce variance

Q-Learning III

Algorithm 11.12: Q-learning

```
initialize $Q^\star(x, a)$ arbitrarily (e.g., as $\mathbf{0}$)
for $t = 0$ to $\infty$ do
    observe the transition $(x, a, r, x')$
    $Q^\star(x, a) \leftarrow (1 - \alpha_t) Q^\star(x, a) + \alpha_t (r + \gamma \max_{a' \in A} Q^\star(x', a'))$
```

Q-Learning IV

non salviamo stime delle transition probabilities né dei rewards

- Q-Learning (similarly to TD-Learning and SARSA) is **Model Free**
- Note that we are free to choose how to sample the transitions and we are not obliged to follow a particular policy.
- When an algorithm is such that we are free to choose how to ”explore” the environment to collect data, we call it **Off Policy**

Off Policy vs On Policy I

- **On Policy**: The agent executes its current policy to sample new trajectories (and possibly learn the optimal policy). Examples: **TD-Learning, SARSA**
- **Off Policy**: The agent uses observational data (coming from an arbitrary policy) to learn the optimal policy. Examples: **Q-Learning**

Off Policy vs On Policy II

- **On Policy**: The agent has full control over its own actions and on how to explore the environment
- **Off Policy**: The agent might not be able to choose how to interact with the environment

Off Policy vs On Policy III

- **On Policy**: The agent cannot use data coming from other policies
- **Off Policy**: The agent can use data coming from an arbitrary policy. This is important as usually interactions with the environment are expensive and hence using offline data to train the algorithm is more convenient!

Exploration-Exploitation Dilemma I

i metodi off-policy disaccoppiano la policy ottimale da quella
che usiamo per esplorare l'ambiente

- We usually call the policy we use to sample the transition an **Exploratory Policy** (not to be confused here with the policy π^* we are optimizing for)
- The question arises: how do we choose the exploratory policy?

Exploration-Exploitation Dilemma II

- The question of how to optimally explore the environment and balance it with optimizing our objective is at the heart of RL
- Intuitively, before starting optimizing our objective, we would want to **explore the environment sufficiently**
- Indeed, if we do not explore enough we might for instance **miss states with high reward**!

Exploration-Exploitation Dilemma III

- Do we want to sample transitions **uniformly at random**? No, this will be very inefficient since we might visit low reward states repeatedly
- Do we want to always choose the **greedy policy**? No, this will likely lead to suboptimal policies as in the beginning our estimate of the objective is not good enough and hence we are probably not going to explore the environment enough
- We would like something in between...

ϵ-greedy Policies: Idea

- In the beginning, since we do not have any knowledge of the environment and of the rewards, we would like to explore more, i.e. we would like to choose actions at random
- As time passes, our estimates of the objective become better and better and hence we would like to act more greedily with respect to our current model (i.e. pick greedy policy)

ϵ-greedy Policies: Algorithm

Algorithm 11.2: ϵ-greedy

```
1 for $t = 0$ to $\infty$ do
2     sample $u \in \mathrm{Unif}([0,1])$
3     if $u \leq \epsilon_t$ then pick action uniformly at random among all actions
4     else pick best action under the current model
```

where ϵ_t is a vanishing sequence

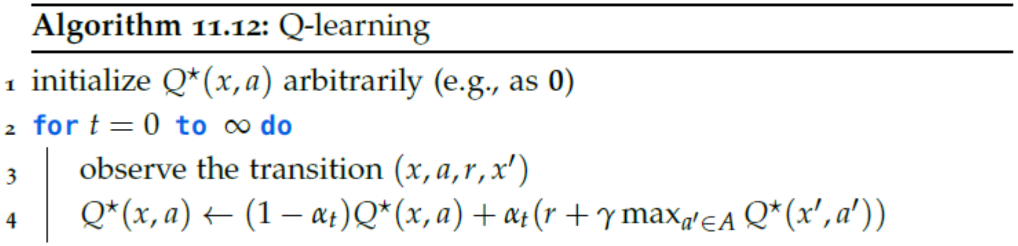
Issues with Q-Learning I

- In many applications, the state and action space can be huge (e.g. Chess, Go) or even infinite (e.g. Robotics)
- These settings pose challenges to Vanilla Q-Learning. Can you think why?

Issues with Q-Learning II

- **Memory issues**: cannot store $\widehat{Q}_\gamma^\infty(s, a, \pi^*)$ for each pair (s, a)
- **Solution**: Use Function Approximation (e.g. Neural Networks) to estimate $\widehat{Q}_\gamma^\infty(s, a, \pi^*)$. This will result in the DQN algorithm (see next week)

Issues with Q-Learning III



Algorithm 11.12: Q-learning

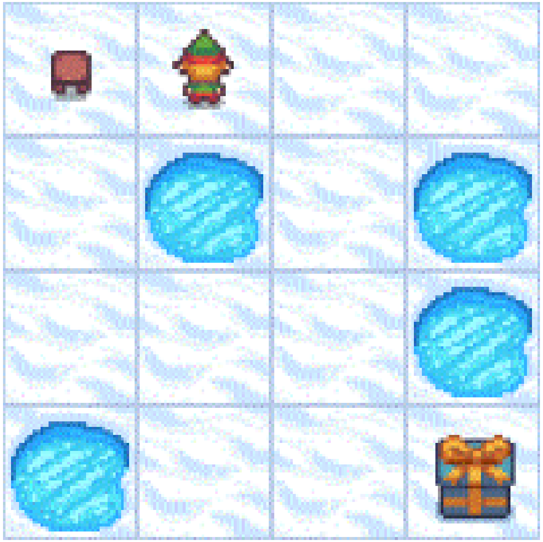
```
1 initialize $Q^\star(x,a)$ arbitrarily (e.g., as $\mathbf{0}$)
2 for $t = 0$ to $\infty$ do
3     observe the transition $(x,a,r,x')$
4     $Q^\star(x,a) \leftarrow (1-\alpha_t)Q^\star(x,a) + \alpha_t(r + \gamma \max_{a' \in A} Q^\star(x',a'))$
```

- If $|\mathcal{A}|$ is too large, we cannot efficiently compute the maximum in the update of the Q-function
- **Solution**: Policy Gradient Methods and Actor-Critics (See Lecture 5)

Quick Recap before Demo

- The RL problem and the Exploration-Exploitation Dilemma: unknown rewards and dynamic force us to **explore the environment before starting optimizing our objective**
- **Q-Learning**: Estimate Q-function using recursive formulation of Bellman Equation and **Bootstrapping**
- **Model Based** vs **Model Free** and **Off-Policy** vs **On-Policy** distinction

Demo: Solving Frozen Lake with Q-Learning



References

- Reinforcement Learning: An Introduction, Sutton and Barto
- Theory and Methods for Reinforcement Learning, EE-618, EPFL, Prof. Volkan Cevher
- Script from Probabilistic Artificial Intelligence course, ETH, Andreas Krause and Jonas Hubotter
- Frozen Lake Environment: https://www.gymlibrary.dev/environments/toy_text/frozen_lake/
- Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. Machine learning, 38(3):287–308, 2000
- Tommi Jaakkola, Michael Jordan, and Satinder Singh. Convergence of stochastic iterative dynamic programming algorithms. Advances in neural information processing systems, 6, 1993

Appendix: Robbins-Monro (RM) Conditions

- **Definition**: A sequence $\{\alpha_t\}_{t\in\mathbb{N}}$ is said to satisfy the RM conditions if $\sum_{t=1}^{\infty}\alpha_t = \infty$ and $\sum_{t=1}^{\infty}\alpha_t^2 < \infty$
- Example: take $\alpha_t = 1/t$, the harmonic sum
- The RM conditions are often used when giving convergence guarantees of various optimization methods, not only in the RL setting

Appendix: Convergence of TD Learning, Sarsa and Q-Learning

- **Theorem** (Jaakkola and al., 1993): If $\{\alpha_t\}_{t\in\mathbb{N}}$ satisfies the RM-conditions and all state-action pairs are chosen infinitely often, then the estimates of the value function in TD-Learning converge to V_γ^∞ with probability 1.
- **Theorem** (Singh et al., 2000): Similarly, if $\{\alpha_t\}_{t\in\mathbb{N}}$ satisfies the RM-conditions and all state-action pairs are chosen infinitely often, then the estimates of the Q-function in Sarsa converge to Q_γ^∞ with probability 1.
- **Theorem** (Jaakkola and al., 1993): If $\{\alpha_t\}_{t\in\mathbb{N}}$ satisfies the RM-conditions and all state-action pairs are chosen infinitely often, then the estimates of the Q^* (the optimal Q-function) in Q-Learning converge to the optimal Q-function with probability 1.