

Model-Free Reinforcement Learning

Learning Optimal Value Functions

Matthia Sabatelli

December 2, 2021

Today's Agenda

- ① Model-Free Reinforcement Learning
- ② Monte Carlo Methods
- ③ Temporal Difference Learning

Recap

Last lecture we have seen the **Dynamic Programming** family of algorithms:

Key Concepts

- We have seen what it means to **evaluate** a policy π
- Why this task is necessary to be able to **improve** π
- How to learn value functions **iteratively**
- The **important role** of the Bellman optimality equations

Model-Free Reinforcement Learning

We have therefore derived:

- How to **estimate** an optimal value function and **discover** an optimal policy π^*
- We did this under the assumption that the environment $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r \rangle$ was **known**

Plot Twist!

We will now consider situations where **no complete knowledge** is available:

- Set of possible states \mathcal{S} ✓
- Set of possible actions \mathcal{A} ✓
- Transition Function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ ✗
- Reward Function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ ✗

Model-Free Reinforcement Learning

When parts of the MDP \mathcal{M} are **unknown** we do not deal with Dynamic Programming methods anymore but with Reinforcement Learning algorithms!

- We need to overcome the **lack of information** of \mathcal{M}
- We can do this through **experience**
- Gathering experience means **sampling** states s , actions a and rewards r from the environment
- Recall the concept of **trajectory** $\tau \langle s_t, a_t, r_t, s_{t+1} \rangle$ seen in Lecture 1!

Model-Free Reinforcement Learning

What does it mean in practice?

- We do not know **the consequences** of our actions
- We do not know the **dynamics** of the environment we are interacting with
- We are no longer **computing** value functions but rather **learning** them

The transition function \mathcal{P} and the reward function \mathcal{R} are usually called the **model** of the environment

\mathcal{P} and \mathcal{R} can be **learned** \Rightarrow model-based Reinforcement Learning

Model-Free Reinforcement Learning

Why is model-free Reinforcement Learning so interesting?

- We learn without any **prior knowledge** of the environment
- Trajectories, and therefore **experience**, is sufficient for learning
- We "only" need a value function $Q(s, a)$, which is arguably **easier** to learn than the model

The effectiveness of model-free Reinforcement Learning algorithms **highly depends** from how much experience the agent is able to gather!

Monte Carlo (MC) Methods

Monte Carlo methods:

- Can be used for learning $V^\pi(s)$ as well as $Q^\pi(s, a)$
- Although in this lecture we only focus on learning $V^\pi(s)$
- The key idea is to learn through **sampling returns**

Assumption!

We assume that we are always dealing with **episodic tasks** i.e. episodes eventually **terminate**.

$$\langle (s_t, a_t, r_t, s_{t+1}) \rangle, t = 0, \dots, T - 1$$

Monte Carlo (MC) Methods

Let us consider the notion of expected discounted return introduced in Lecture 1:

$$\begin{aligned} G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \end{aligned}$$

- The **goal** is to learn the state-value function of a given policy $V^{\pi}(s)$: [Monte Carlo Prediction](#)
- We do this with respect to the G_t that is obtained by following π

Monte Carlo (MC) Methods

Learning $V^\pi(s)$ involves the following [steps](#):

- Before learning, each state has its own value $V(s_t)$
- We follow policy π until an episode terminates
- We compute the discounted return G_t that was obtained by π

Monte Carlo (MC) Update Rule

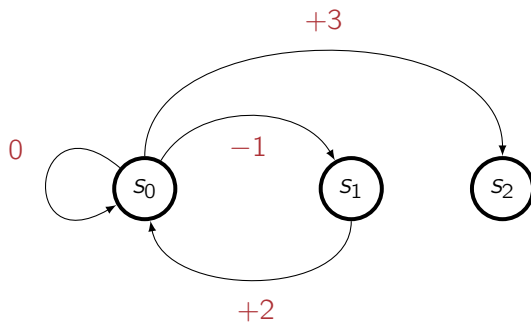
We change the value of each state $V(s_t)$ based on G_t

$$V(s_t) := V(s_t) + \alpha [G_t - V(s_t)].$$

where $\alpha \in [0, 1]$ is the learning rate parameter.

Monte Carlo (MC) Methods

Let us consider the following MDP:



- We have policy $\pi : s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_2$
- π results in the sequence of rewards $-1, +2, +3$
- The starting value of each state is 0, $\gamma = 0.99$ and $\alpha = 0.5$

Monte Carlo (MC) Methods

We know that G_t for starting in s_0 is

$$\begin{aligned} G_t &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \\ &= -1 + \gamma 2 + \gamma^2 3 \approx 3.92 \end{aligned}$$

Therefore

$$\begin{aligned} V(s_t) &:= V(s_t) + \alpha [G_t - V(s_t)] \\ V(0) &:= V(0) + \alpha [3.92 - V(0)] \approx 1.96 \end{aligned}$$

Monte Carlo (MC) Methods

More about Monte Carlo Methods:

- We have only considered the **prediction** case of learning $V^\pi(s)$
- Monte Carlo **control** algorithms learn an approximation of π^* through $Q^\pi(s, a)$
- They follow the ideas of **Dynamic Programming** seen in the previous lecture
- The key idea of **sampling real returns** remains!

Monte Carlo (MC) Methods

Pros & Cons of Monte Carlo Methods:

- Yield **unbiased** updates thanks to G_t ✓
- **Scale well** to function approximators ✓
- Learning can be very **slow** as one has to wait until the very end of an episode ✗
- There can be **large variance** in the value updates ✗

Temporal Difference (TD) Learning

"The simplest and most elegant idea of Reinforcement Learning ..."

Temporal Difference (TD) Learning

With TD-Learning methods we **do not** have to wait until the end of an episode before updating a value estimate

- We only need to wait until the **next step**
- At $t + 1$ we immediately create a **target** for learning called the TD-target
- We do this by using the observed reward r_t and a future value estimate e.g. $V(s_{t+1})$

Temporal Difference (TD) Learning

Let us again consider the problem of estimating $V^\pi(s)$:

TD Prediction

We change the value of each state $V(s_t)$ with respect to $t + 1$ only:

$$V(s_t) := V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)].$$

- It is clear that we only learn by *looking ahead* in the future one single step
- This is called TD(0) or *one-step TD*

Temporal Difference (TD) Learning

The key idea of TD-Learning is to learn through **bootstrapping**:

- We update the value of a state with respect to the value of its **successor** state only
- Ideally we would like to use $V^\pi(s_{t+1})$ for learning but it is unfortunately **unknown**

$$V(s_t) := V(s_t) + \alpha [r_t + \gamma V^\pi(s_{t+1}) - V(s_t)].$$

Therefore we **replace** it with a guess instead:

$$V(s_t) := V(s_t) + \alpha [r_t + \gamma V(s_{t+1}) - V(s_t)].$$

Temporal Difference (TD) Learning

The key idea of TD-Learning is to learn through **bootstrapping**:

- We update the value of a state with respect to the value of its **successor** state only
- Ideally we would like to use $V^\pi(s_{t+1})$ for learning but it is unfortunately **unknown**

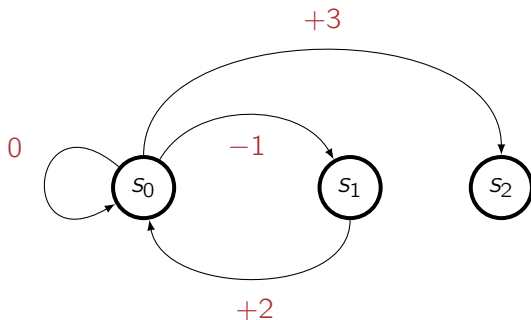
$$V(s_t) := V(s_t) + \alpha [r_t + \gamma V^\pi(s_{t+1}) - V(s_t)].$$

Therefore we **replace** it with a guess instead:

$$V(s_t) := V(s_t) + \alpha [r_t + \gamma V(s_{t+1}) - V(s_t)].$$

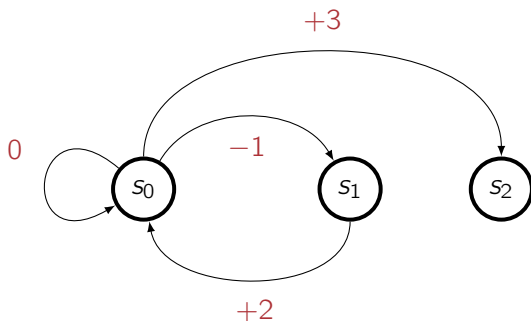
Temporal Difference (TD) Learning

Let us go back to the previous MDP:



- We again have policy $\pi : s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_2$
- π results in the sequence of rewards $-1, +2, +3$
- The starting value of each state is 0, $\gamma = 0.99$ and $\alpha = 0.5$

Temporal Difference (TD) Learning



Our first state transition is $s_0 \rightarrow s_1$

$$V(s_t) := V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

$$V(s_0) := V(s_0) + \alpha[r_t + \gamma V(s_1) - V(s_0)]$$

$$V(s_0) := -0.5$$

Temporal Difference (TD) Learning

What is so cool about the TD Learning update rule?

$$V(s_t) := V(s_t) + \alpha [r_t + \gamma V(s_{t+1}) - V(s_t)]$$

We are learning by guessing!

- We want to learn $V(s_t)$ with respect to $V(s_{t+1})$
- But $V(s_{t+1})$ is as unknown as $V(s_t)$
- We use the information provided by the environment r_t immediately to construct the **TD-error** δ_t

$$V(s_t) := V(s_t) + \alpha \underbrace{[r_t + \gamma V(s_{t+1}) - V(s_t)]}_{\delta_t}$$

Temporal Difference (TD) Learning

Why are the TD-errors δ_t so important?

- They allow us to start learning **immediately**
- They separate RL algorithms into two **families** of techniques: *off-policy* and *on-policy* techniques
- Each family comes with its own **convergence properties**

To know more about these families let us consider the problem of learning the state-action value function $Q^\pi(s, a)$

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a, \pi \right].$$

Temporal Difference (TD) Learning

The arguably most popular algorithm for learning $Q^\pi(s, a)$ is **Q-Learning** (Watkins & Dayan, 1992)

- Is an *off-policy* learning algorithm
- Able of converging to the optimal state-action value function $Q^*(s, a)$ with probability 1
- Works by keeping track of an estimate of the state-action value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

Q-Learning

The update rule of each visited state-action pair used by Q-Learning is:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \right].$$

Temporal Difference (TD) Learning

How does Q-Learning work?

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[\underbrace{r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)}_{\delta_t} \right]$$

- We create the TD-error δ_t by using the $\max_{a \in \mathcal{A}}$ operator

Temporal Difference (TD) Learning

How does Q-Learning work?

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[\underbrace{r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)}_{\delta_t} \right]$$

- We create the TD-error δ_t by using the $\max_{a \in \mathcal{A}}$ operator
- We **always** update $Q(s_t, a_t)$ with respect to a greedy policy

Temporal Difference (TD) Learning

How does Q-Learning work?

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[\underbrace{r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)}_{\delta_t} \right]$$

- We create the TD-error δ_t by using the $\max_{a \in \mathcal{A}}$ operator
- We **always** update $Q(s_t, a_t)$ with respect to a greedy policy
- Even if the agent is exploring the environment, learning is done **greedily** \rightarrow *off-policy* learning

Temporal Difference (TD) Learning

We can also learn the $Q^\pi(s, a)$ function in a way which is more similar to how we learned $V^\pi(s)$ beforehand: [SARSA](#) (Rummery & Niranjan, 1994)

- Is an *on-policy* learning algorithm
- Also works by keeping track of $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
- Has different convergence properties

SARSA

The update rule of each visited state-action pair used by SARSA is:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right].$$

Temporal Difference (TD) Learning

If we take a look at SARSA's TD-error ...

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \underbrace{\left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]}_{\delta_t}$$

- We **do not use** the max operator anymore

Temporal Difference (TD) Learning

If we take a look at SARSA's TD-error ...

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \underbrace{\left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]}_{\delta_t}$$

- We **do not use** the max operator anymore
- We learn with respect to the next state that is visited by the agent s_{t+1} , which might or **might not** (important!) correspond to a greedy policy

Temporal Difference (TD) Learning

If we take a look at SARSA's TD-error ...

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \underbrace{\left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]}_{\delta_t}$$

- We **do not use** the max operator anymore
- We learn with respect to the next state that is visited by the agent s_{t+1} , which might or **might not** (important!) correspond to a greedy policy
- As we are always taking into account the actions chosen by the agent $Q(s_{t+1}, a_{t+1})$ we learn \rightarrow *on-policy*

Temporal Difference (TD) Learning

If we take a look at SARSA's TD-error ...

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \underbrace{\left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]}_{\delta_t}$$

- We **do not use** the max operator anymore
- We learn with respect to the next state visited by the agent s_{t+1} which might or **might not** (important!) correspond to a greedy policy
- As we are always taking into account the actions chosen by the agent $Q(s_{t+1}, a_{t+1})$ we learn \rightarrow *on-policy*

Temporal Difference (TD) Learning

Quiz Time!

Imagine an algorithm which learns the state-value function $V^\pi(s)$ as follows

$$V(s) := V(s) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

and the state-action value function $Q^\pi(s, a)$ as follows:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha[r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)].$$

- What is the TD-target of this algorithm?
- Is this an *off-policy* or *on-policy* learning algorithm?

Temporal Difference (TD) Learning

Quiz Time!

Now imagine an algorithm which learns the state-value function $V^\pi(s)$ as follows

$$V(s) := V(s) + \alpha [r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - V(s_t)]$$

and the state-action value function $Q^\pi(s, a)$ as follows:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha [r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)].$$

- What is the TD-target of this algorithm?
- Is this an *off-policy* or an *on-policy* learning algorithm?

Temporal Difference (TD) Learning

Pros & Cons of TD-Learning methods:

- They **do not** yield unbiased estimates **✗**
- But there is **small variance** in the updates **✓**
- Learning starts **faster** **✓**
- It can be **complicated** to combine them with non-linear function approximators **✗**

Both approaches can be combined resulting in TD(λ) algorithms!

⇒ TD-Learning methods empirically work better than MC algorithms but a formal proof about why this is the case is missing ...

Final Slide!

Lecture Takeaway

- What happens when parts of the Markov Decision Process are unknown
- Dynamic Programming → Reinforcement Learning
- Two families of model-free Reinforcement Learning algorithms
- We have seen the difference between *on-policy* and *off-policy* learning