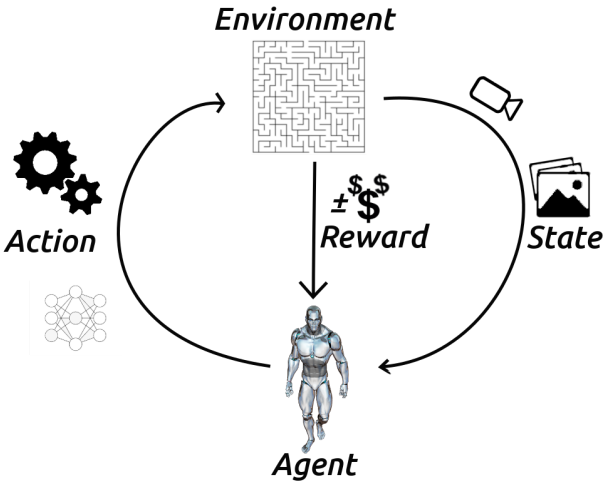
Навчання з підкріпленням

Лекція 4: Безмодельне передбачення

Кочура Юрій Петрович
iuriy.kochura@gmail.com
@y_kochura

Огляд



- Навчання з підкріпленням — це наука про те, як навчитися приймати правильні рішення
- Агент може вивчати стратегію, функцію цінності та/або модель
- Загальна проблема передбачає врахування часу та наслідків
- Прийняті рішення впливають на винагороду, стан агента та стан середовища

Сьогодні

- Вступ до безмодельного передбачення
- Методи Монте-Карло
- Методи часових різниць (temporal difference, TD)
- TD(λ): Покращена оцінка усіх відвіданих станів

- Минулого разу
  - Планування за допомогою динамічного програмування
  - Розв'язок відомого МППР
- Сьогодні
  - Безмодельне передбачення
  - Оцінка функції цінності невідомого МППР

Вступ до безмодельного передбачення

Агенти, які навчаються на зворотному зв’язку (методом проб і помилок) часто відносять до задач передбачення, тому що ми маємо оцінити функцію цінності, яка показує очікувану (середню) винагороду агента для певно стратегії. Функція цінності містить значення, які залежать від майбутнього, тому в певному сенсі ми вчимося передбачати майбутнє.

Винагорода R_t: скалярний сигнал, який отримує агент у якості зворотного зв'язку від середовища після виконання дії агеном. Відноситься до однокрокового сигналу: агент спостерігає за станом середовища, обирає дію та отримує сигнал винагороди. Миттєва винагорода є ключовим поняттм в RL, але це не те, що агент намагається максимізувати.

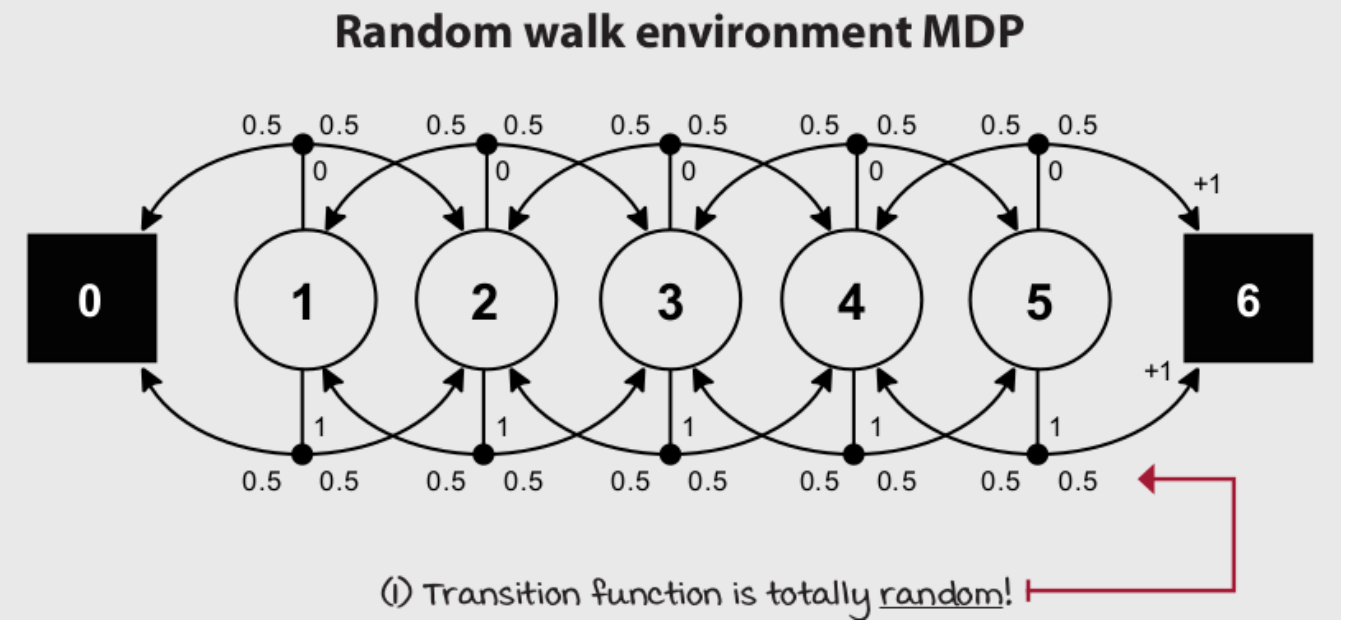
Загальна винагорода (return): сумарна винагорода отримана агентом з моменту часу t з урахування знецінювання γ. Розраховується від будь-якого стану агента і зазвичай триває до кінця епізоду. Тобто, коли досягається стан завершення (terminal state), обчислення припиняється.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Функція цінності: визначає усереднену загальну винагороду:

$$\begin{aligned} v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] = \\ &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] \end{aligned}$$

Середовище випадкового блукання



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Методи Монте-Карло (МК)

- Методи МК вчаться безпосередньо з епізодів (досвіду)
- МК методи безмодельні: відсутні знання про МППР
- Методи МК навчаються з повних епізодів: без бутстрапінга
- Ми називаємо пряму вибірку епізодів Монте-Карло
- Примітка: методи МК можна застосовувати лише до епізодичних МППР
  - Усі епізоди мають бути кінцевими
- Методи МК використовують просту ідею: цінність = середня загальна винагорода

Оцінка стратегії Монте-Карло

- Мета: вивчити $v_\pi(s)$ з епізодів досвіду в рамках стратегії π

$$S_t, A_t, R_{t+1}, \cdots, S_T \sim \pi$$

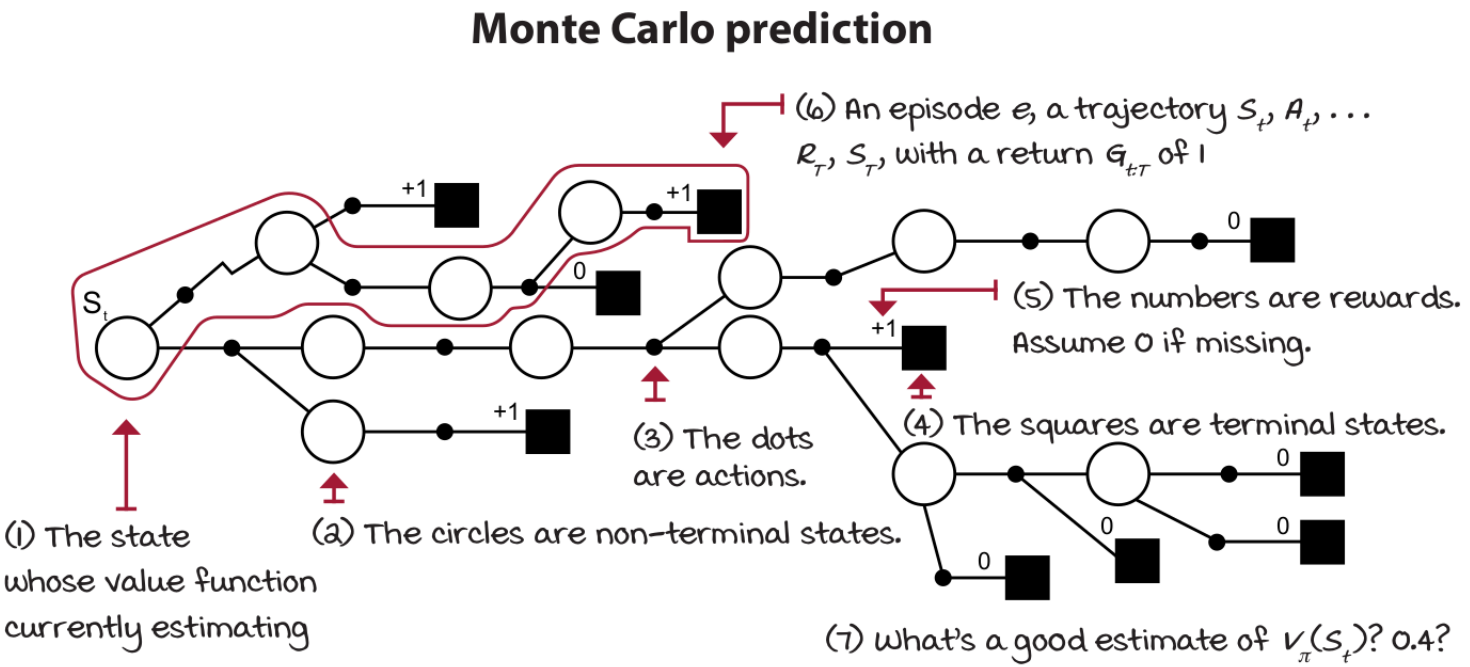
- Загальна винагорода:

$$G_{t:T} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

- Функція цінності:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Перше відвідування Монте-Карло **(First-visit Monte Carlo, FVMC):** покращення оцінок після кожного епізоду



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

FVMC

1. $v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$
2. $G_{t:T} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$
3. $S_t, A_t, R_{t+1}, \cdots, R_T, S_T \sim \pi$
4. $T_T(S_t) = T_T(S_t) + G_{t:T}$
5. $N_T(S_t) = N_T(S_t) + 1$
6. $V_T(S_t) = \frac{T_T(S_t)}{N_T(S_t)}$
7. Якщо $N(s) \to \infty$, тоді $V(s) \to v_\pi(s)$

$$V_T(S_t) = V_{T-1}(S_t) + \frac{1}{N_t(S_t)} \left[G_{t:T} - V_{T-1}(S_t) \right]$$

$$V_T(S_t) = V_{T-1}(S_t) + \alpha_t \Big[\overbrace{\underbrace{G_{t:T}}_{\text{MC target}} - V_{T-1}(S_t)}^{\text{MC error}} \Big]$$

Кожне відвідування Монте-Карло **(Every-visit Monte Carlo, EVMC):** інший спосіб обробки відвіданих станів

Ви, напевно, помітили, що на практиці можна реалізувати два різних способи алгоритму з усереднення загальної винагороди. Це викликано тим, що одна траєкторія може містити кілька відвідувань одного і того ж стану. У цьому випадку, чи варто розраховувати загальну винагороду після кожного з цих відвідувань незалежно, а потім включити всі ці значення до усереднення, чи ми повинні використовувати обраховану загальну винагороду лише від першого візиту до кожного стану?

Обидва підходи є робочими та мають схожі теоретичні властивості.

—
Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Перше **vs** Кожне відвідування Монте-Карло

Передбачення МК оцінює $v_\pi(s)$ як усереднене значення загальних винагород при дотриманні стратегії π. FVMC використовує лише одне значення загальної винагороди для одного стану протягом епізоду: загальна винагорода після першого відвідування стану. EVMC усереднює загальну винагороду для усіх відвідувань одного і того ж стану протягом епізоду.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Історія

Ви, напевно, чули раніше термін "симуляції Монте-Карло" або "імітаційне моделювання". Методи Монте-Карло, загалом відомі з 1940-х років і є широким класом алгоритмів, які використовують випадкову вибірку для оцінок. Проте, у 1996 році вперше методи першого та кожного відвідування МК були визначені у статті Сатіндера Сінгха (Satinder Singh) та Річарда Саттона "Reinforcement Learning with Replacing Eligibility Traces".

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

I Speak Python

Exponentially decaying schedule

```
def decay_schedule(init_value, min_value,
                   decay_ratio, max_steps,
                   log_start=-2, log_base=10):
    decay_steps = int(max_steps * decay_ratio)
    rem_steps = max_steps - decay_steps
```

(1) This function allows you to calculate all the values for alpha for the full training process.

(2) First, calculate the number of steps to decay the values using the decay_ratio variable.
(3) Then, calculate the actual values as an inverse log curve. Notice we then normalize between 0 and 1, and finally transform the points to lay between init_value and min_value.

```
    values = np.logspace(log_start, 0, decay_steps,
                         base=log_base, endpoint=True)[::-1]
    values = (values - values.min()) / \
                                (values.max() - values.min())
    values = (init_value - min_value) * values + min_value
    values = np.pad(values, (0, rem_steps), 'edge')
    return values
```

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

I Speak Python

Generate full trajectories

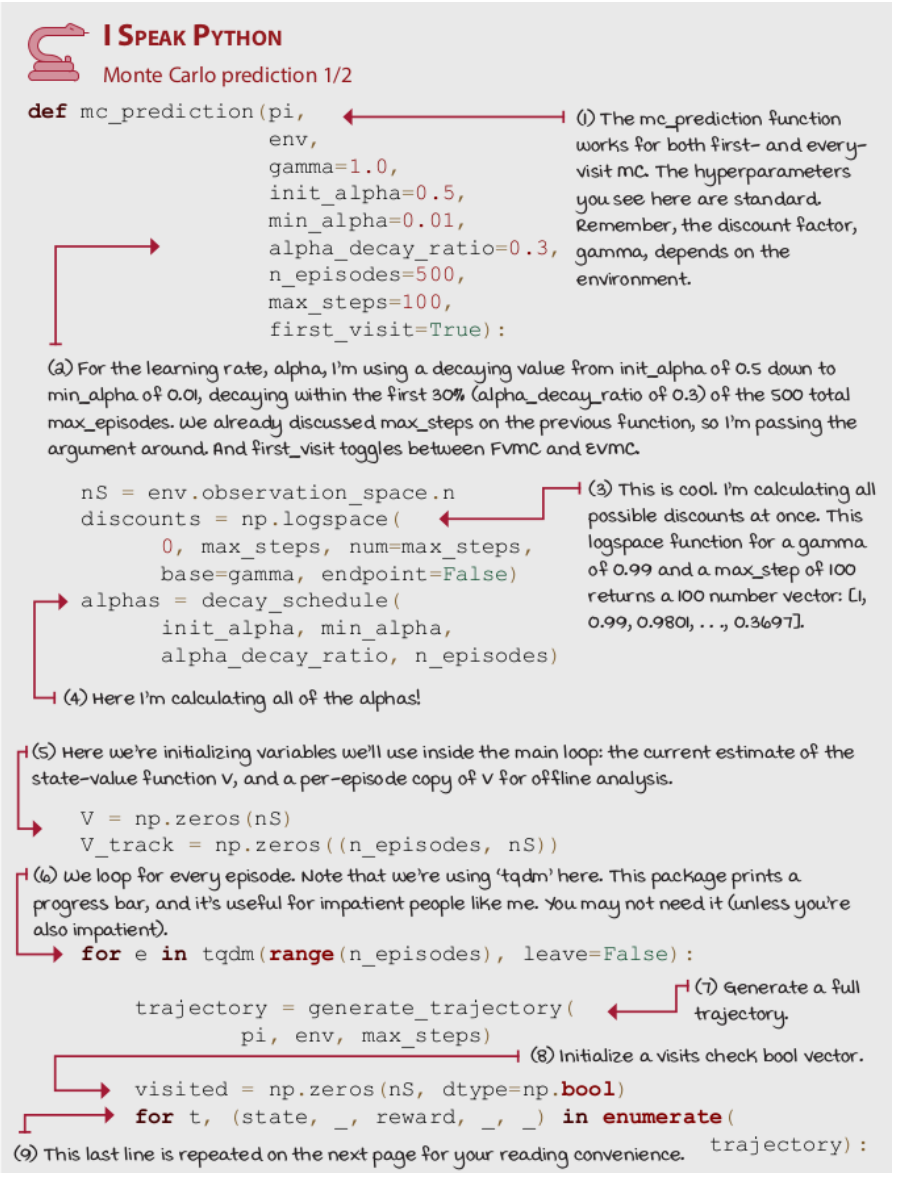
```
def generate_trajectory(pi, env, max_steps=20):
    done, trajectory = False, []
    while not done:
        state = env.reset()
        for t in count():
            action = pi(state)
            next_state, reward, done, _ = env.step(action)
            experience = (state, action, reward,
                          next_state, done)
            trajectory.append(experience)
            if done:
                break
            if t >= max_steps - 1:
                trajectory = []
                break
            state = next_state
    return np.array(trajectory, np.object)
```

(1) This is a straightforward function. It's running a policy and extracting the collection of experience tuples (the trajectories) for off-line processing.

(2) This allows you to pass a maximum number of steps so that you can truncate long trajectories if desired.

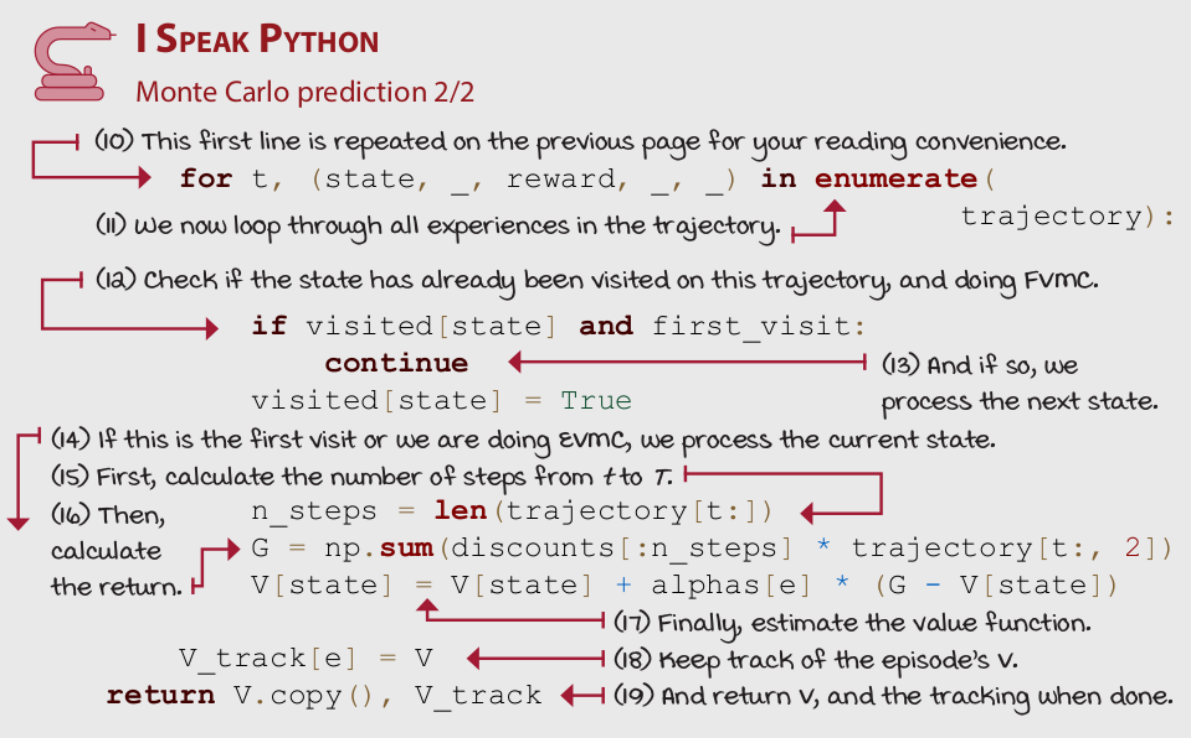
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

ŘŁ With an RL Accent

Incremental vs. sequential vs. trial-and-error

Incremental methods: Refers to the iterative improvement of the estimates. Dynamic programming is an incremental method: these algorithms iteratively compute the answers. They don't "interact" with an environment, but they reach the answers through successive iterations, incrementally. Bandits are also incremental: they reach good approximations through successive episodes or trials. Reinforcement learning is incremental, as well. Depending on the specific algorithm, estimates are improved on an either per-episode or per-time-step basis, incrementally.

Sequential methods: Refers to learning in an environment with more than one non-terminal (and reachable) state. Dynamic programming is a sequential method. Bandits are not sequential, they are one-state one-step MDPs. There's no long-term consequence for the agent's actions. Reinforcement learning is certainly sequential.

Trial-and-error methods: Refers to learning from interaction with the environment. Dynamic programming is not trial-and-error learning. Bandits are trial-and-error learning. Reinforcement learning is trial-and-error learning, too.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Методи часових різниць (temporal difference, TD)

ŘŁ With an RL Accent

True vs. actual vs. estimated

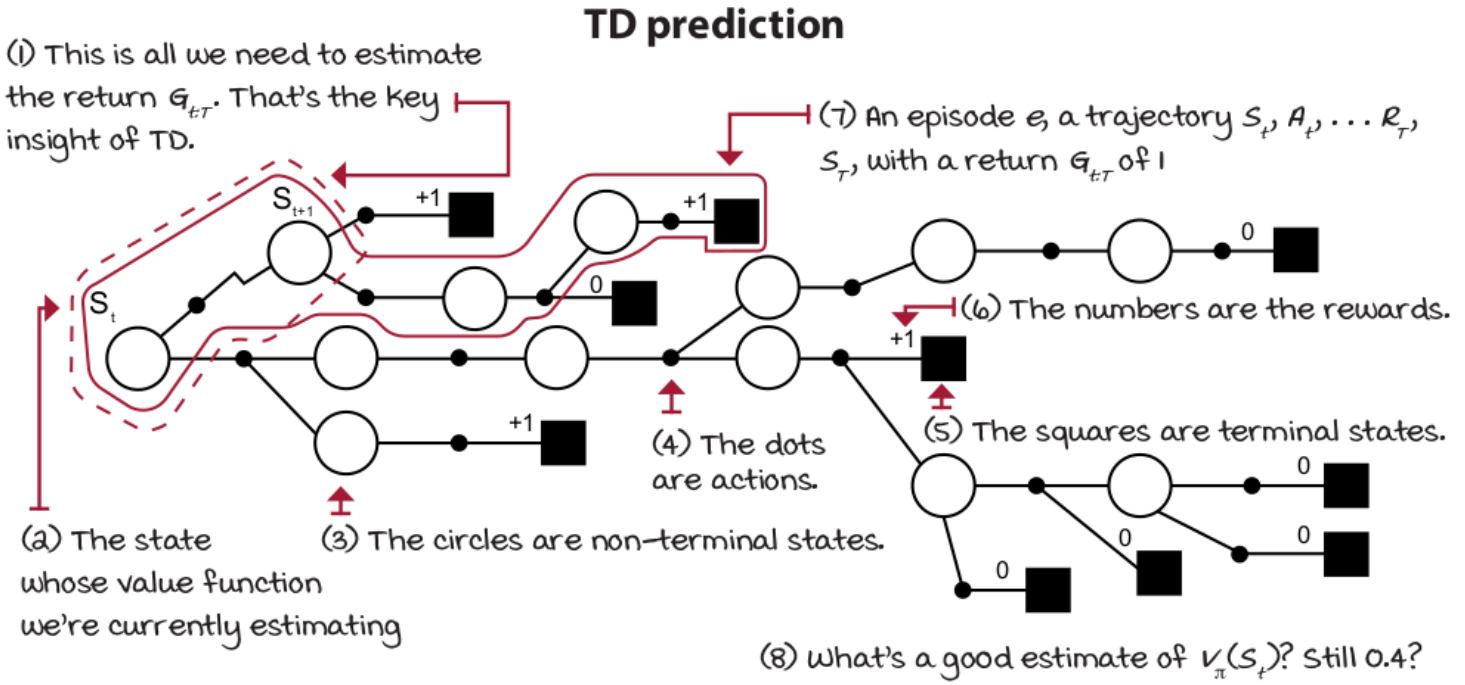
True value function: Refers to the exact and perfectly accurate value function, as if given by an oracle. The true value function is the value function agents estimate through samples. If we had the true value function, we could easily estimate returns.

Actual return: Refers to the experienced return, as opposed to an estimated return. Agents can only experience actual returns, but they can use estimated value functions to estimate returns. *Actual return* refers to the full experienced return.

Estimated value function or estimated return: Refers to the rough calculation of the true value function or actual return. "Estimated" means an approximation, a guess. True value functions let you estimate returns, and estimated value functions add bias to those estimates.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Методи часових різниць **(TD)** та бутстрапінг

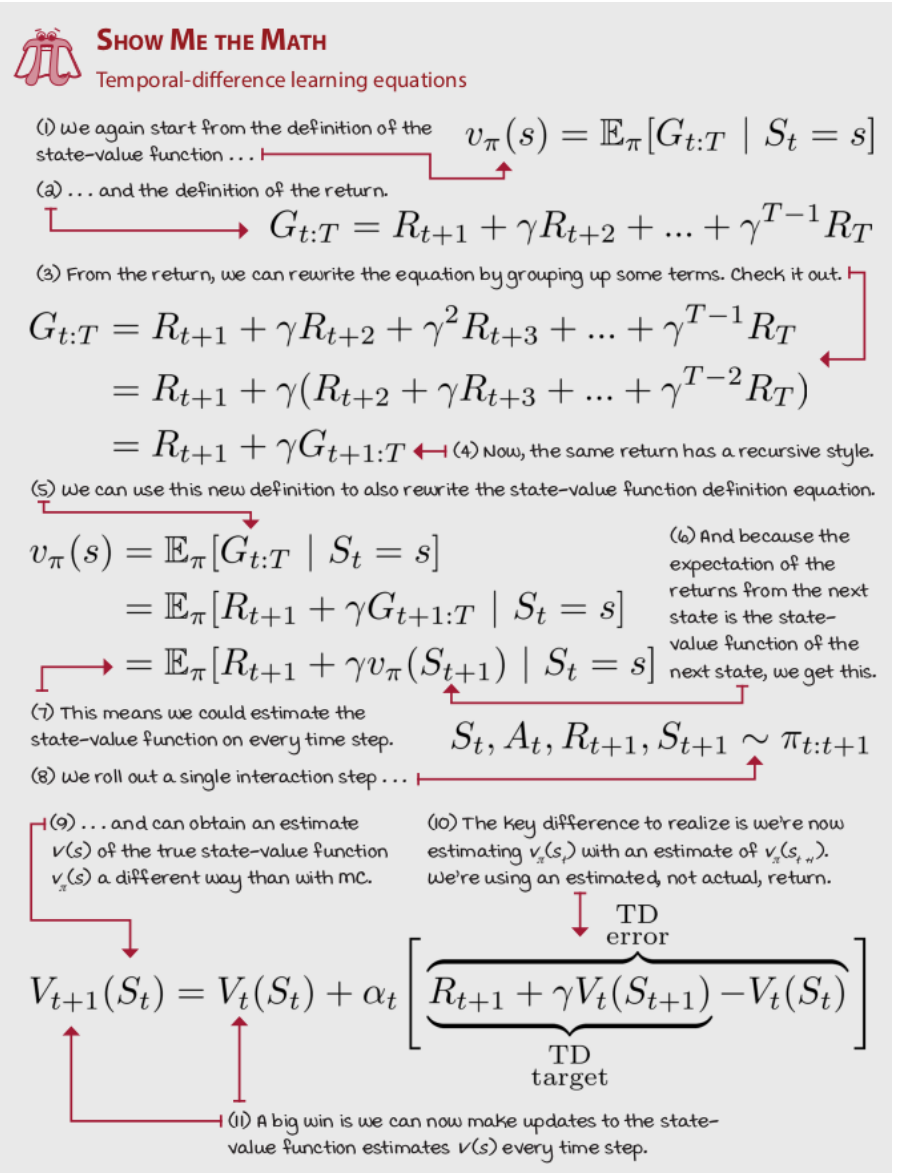
TD методи оцінюють $v_\pi(s)$ з використанням оцінки $v_\pi(s)$. Це підхід відомий як бутстрапінг, робить здогадку з здогадки; він використовує оціночну загальну винагороду замість фактичної. Формально цей метод використовує:

$$R_{t+1} + \gamma V_t(S_{t+1})$$

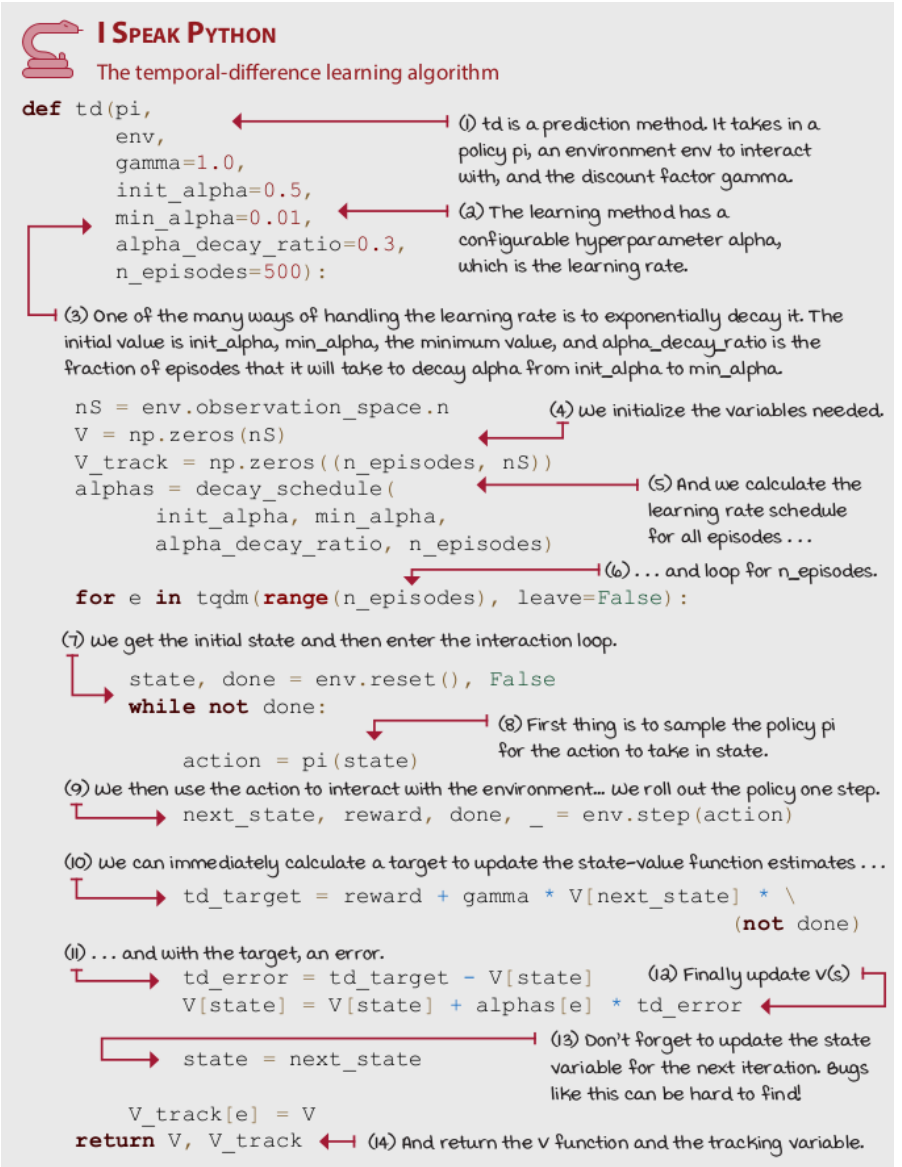
для розрахунку та оцінки $V_{t+1}(S_t)$.

Оскільки TD використовує один крок фактичного значення загальної винагороди R_{t+1}, він усе ще працюватиме добре. Цей сигнал винагороди R_{t+1} поступово «вносить реальність» в оцінки.

—
Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



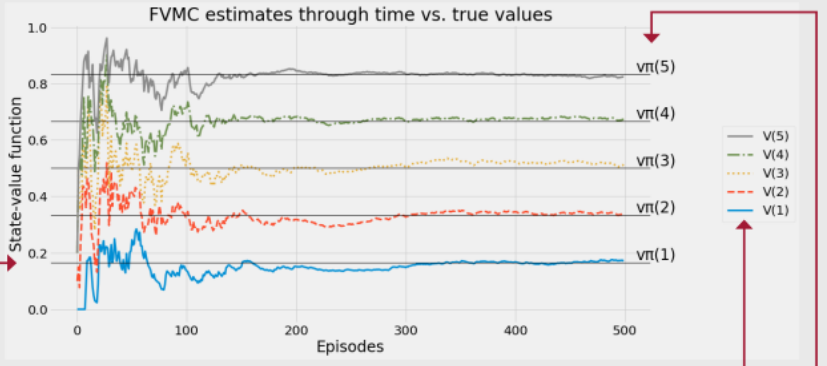
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Tally it Up

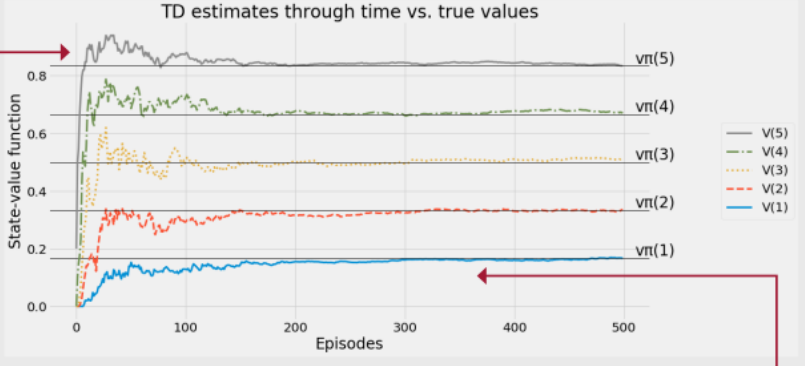
MC and TD both nearly converge to the true state-value function

(1) Here I'll show only first-visit Monte Carlo prediction (FVMC) and temporal-difference learning (TD). If you head to the Notebook for this chapter, you'll also see the results for every-visit Monte Carlo prediction, and several additional plots that may be of interest to you!



(2) Take a close look at these plots. These are the running state-value function estimates $v(s)$ of an all-left policy in the random-walk environment. As you can see in these plots, both algorithms show near-convergence to the true values.

(3) Now, see the difference trends of these algorithms. FVMC running estimates are very noisy; they jump back and forth around the true values.



(4) TD running estimates don't jump as much, but they are off-center for most of the episodes. For instance $v(5)$ is usually higher than $v_\pi(5)$, while $v(1)$ is usually lower than $v_\pi(1)$. But if you compare those values with FVMC estimates, you notice a different trend.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

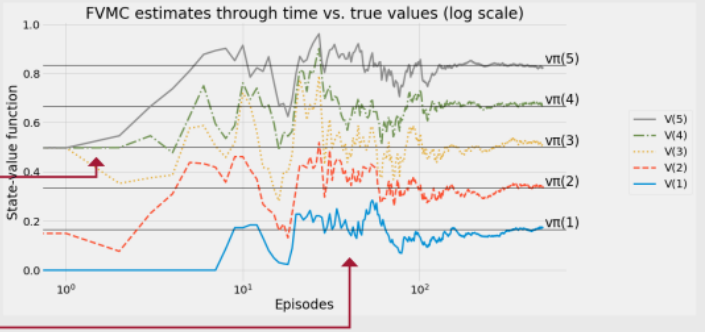
Tally it Up

MC estimates are noisy; TD estimates are off-target

$$V_T(S_t) = V_{T-1}(S_t) + \alpha_t \left[\overbrace{\underbrace{G_{t:T}}_{\text{MC target}} - V_{T-1}(S_t)}^{\text{MC error}} \right]$$

(1) If we get a close-up (log-scale plot) of these trends, you'll see what's happening. MC estimates jump around the true values. This is because of the high variance of the MC targets.

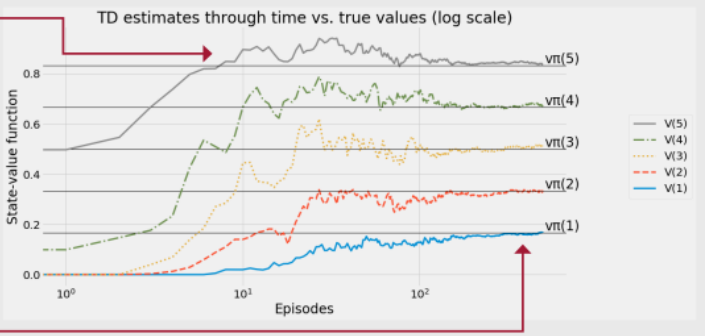
(2) A couple of pros though; first, you can see all estimates get close to their true values *very early* on. Also, the estimates jump around the *true* values.



$$V_{t+1}(S_t) = V_t(S_t) + \alpha_t \left[\overbrace{\underbrace{G_{t:t+1}}_{\text{TD target}} - V_t(S_t)}^{\text{TD error}} \right]$$

(3) TD estimates are off-target most of the time, but they're less jumpy. This is because TD targets are low variance, though biased. They use an estimated return for target.

(4) The bias shows, too. In the end, TD targets give up accuracy in order to become more precise. Also, they take a bit long before estimates ramp up, at least in this environment.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

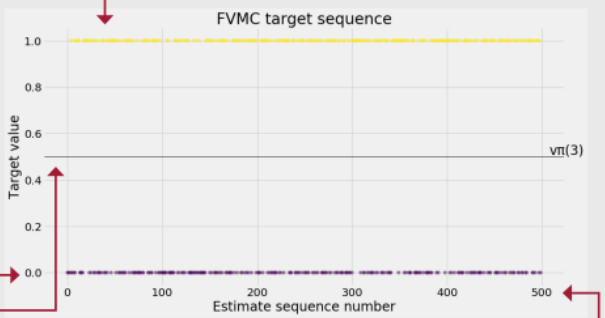
Tally it Up

MC targets high variance; TD targets bias

$$G_{t:T} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

(1) Here we can see the bias/variance trade-off between MC and TD targets. Remember, the MC target is the return, which accumulates a lot of random noise. That means high variance *targets*.

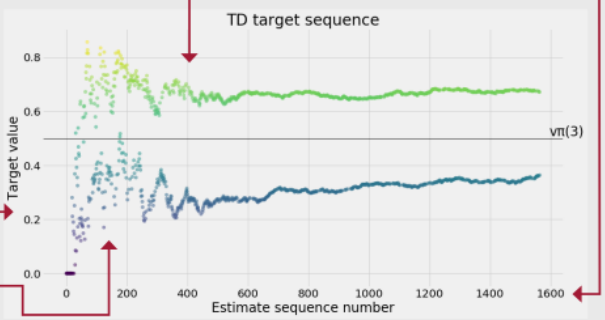
(2) These plots are showing the targets for the initial state in the RW environment. MC targets, the returns, are either 0 or 1 because the episode terminates either on the left, with a 0 return or on the right, with a 1 return, while the optimal value is 0.5!



$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$$

(3) TD targets are calculated using an estimated return. We use the value function to predict how much value we'll get from the next state onward. This helps us truncate the calculations and get more estimates per episode (as you can see on the *x*-axis, we have ~1600 estimates in 500 episodes), but because we use $V_t(S_{t+1})$, which is an estimate and therefore likely wrong, TD targets are biased.

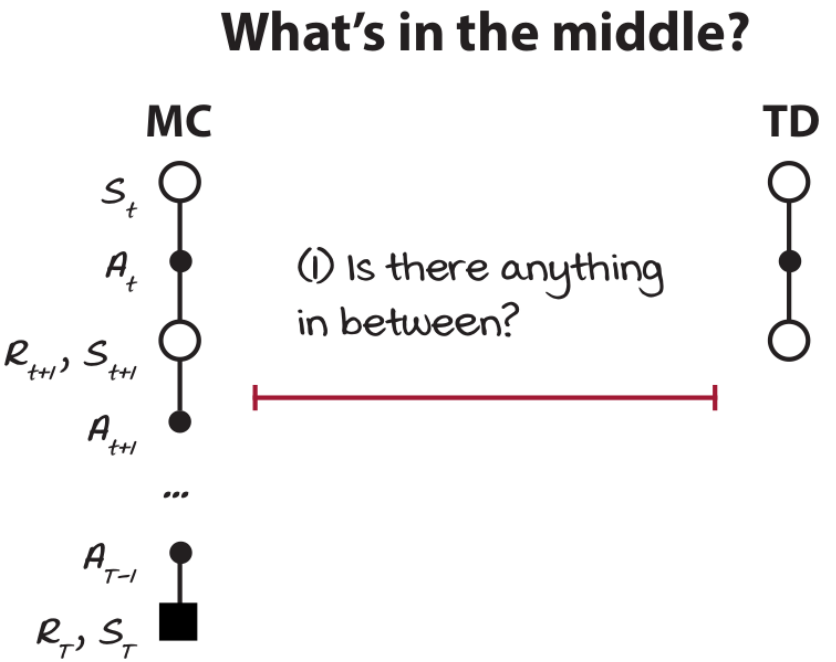
(4) Here you can see the range of the TD targets is much lower, MC alternates *exactly* between 1 and 0, and TD jumps between *approximately* 0.7 and ~0.3, depending on which "next state" is sampled. But as the $V_t(S_{t+1})$ is an estimate, $G_{t:t+1}$ is biased, off-target, and inaccurate.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

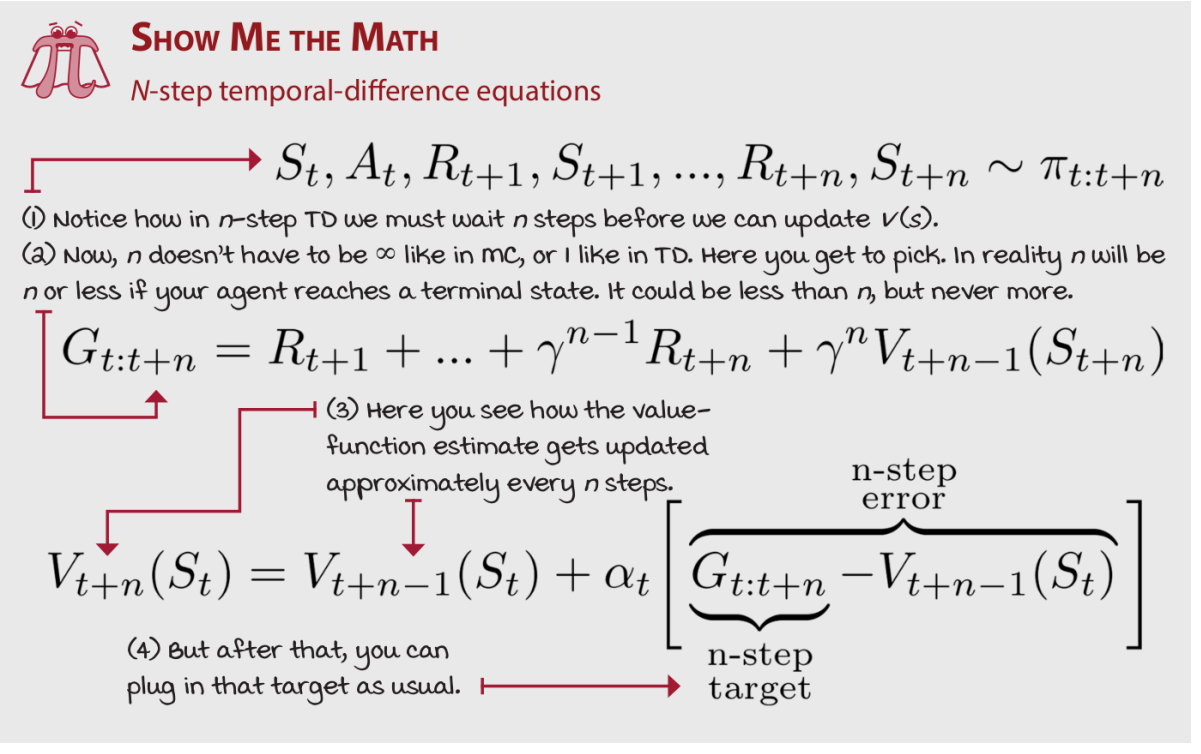
Оцінювання функції цінності з кількох кроків



—

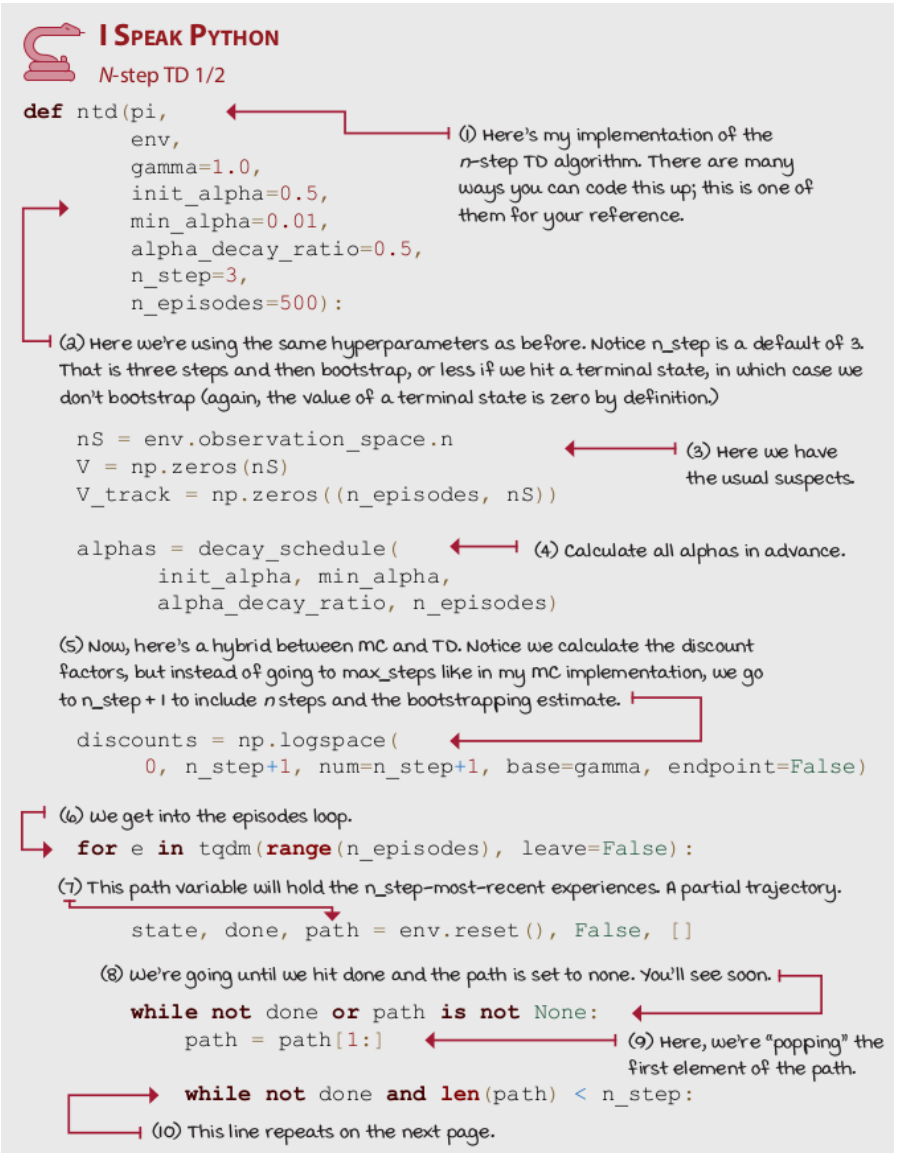
Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

N-крековий TD



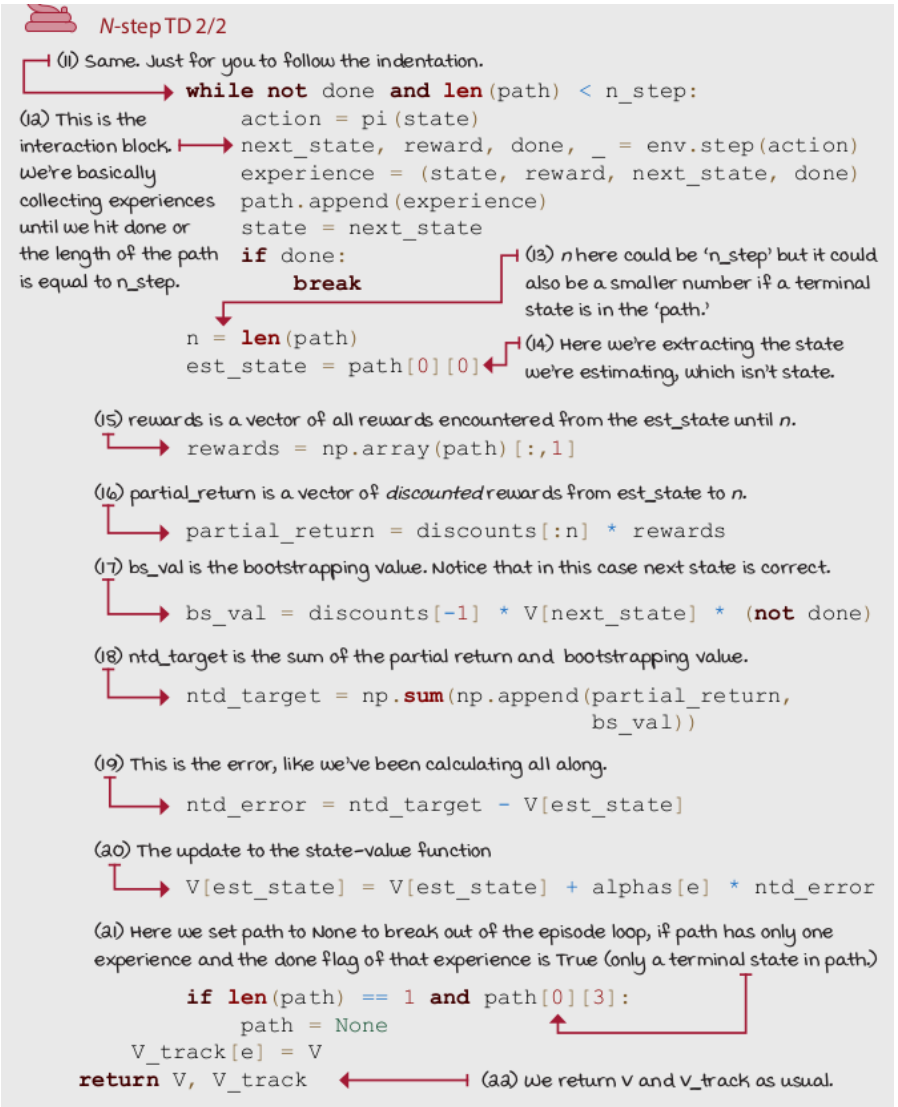
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

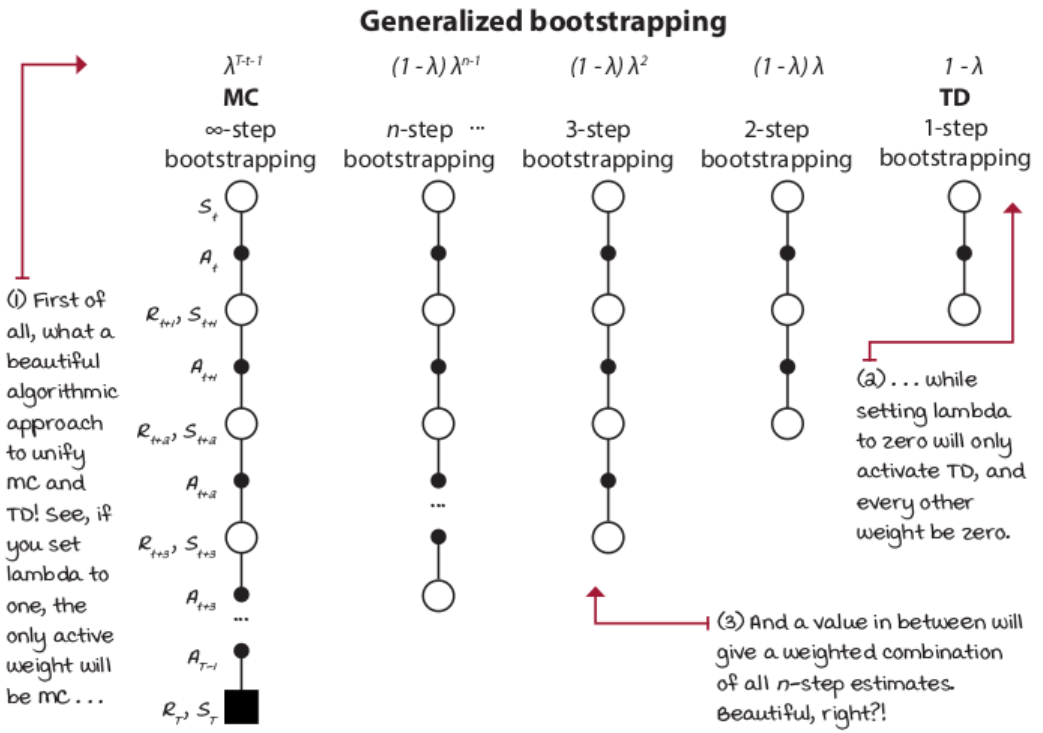
Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



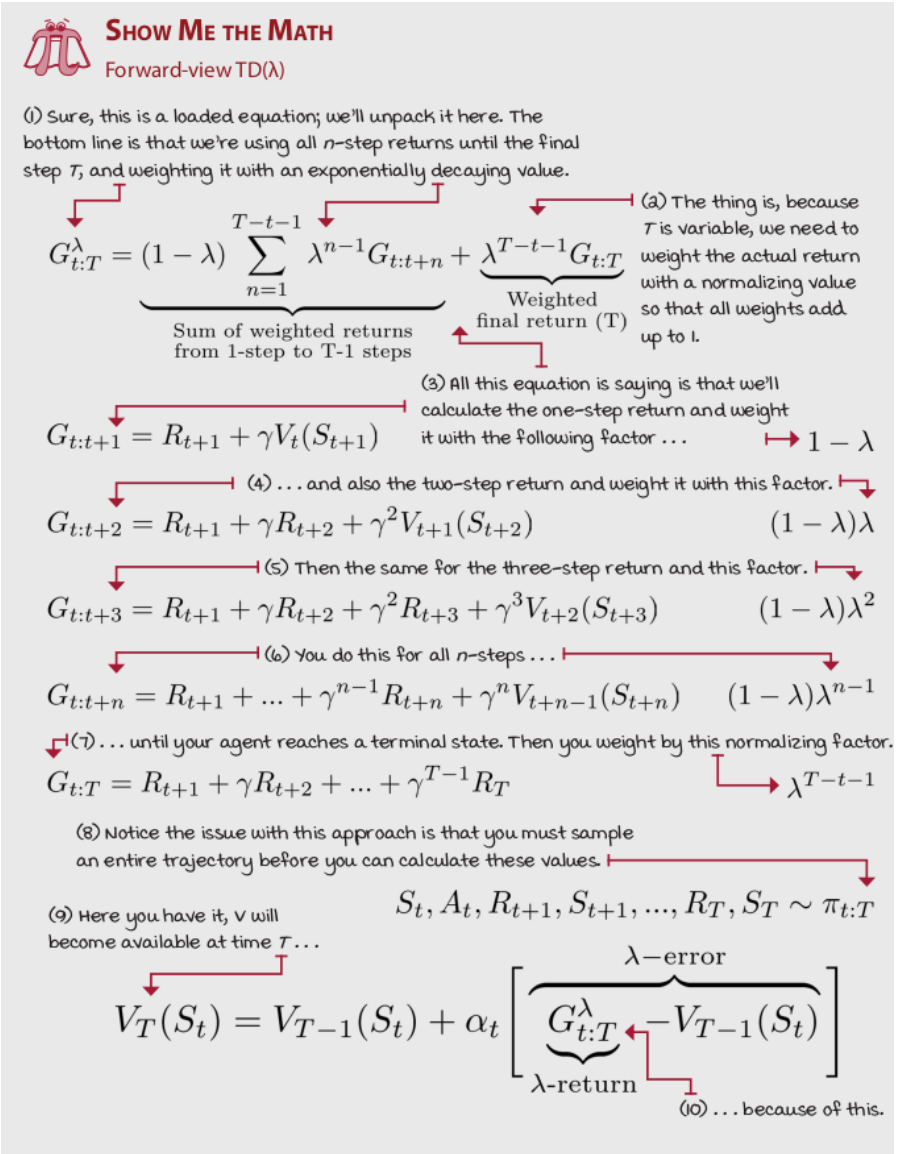
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

TD(λ): Покращена оцінка усіх відвіданих станів

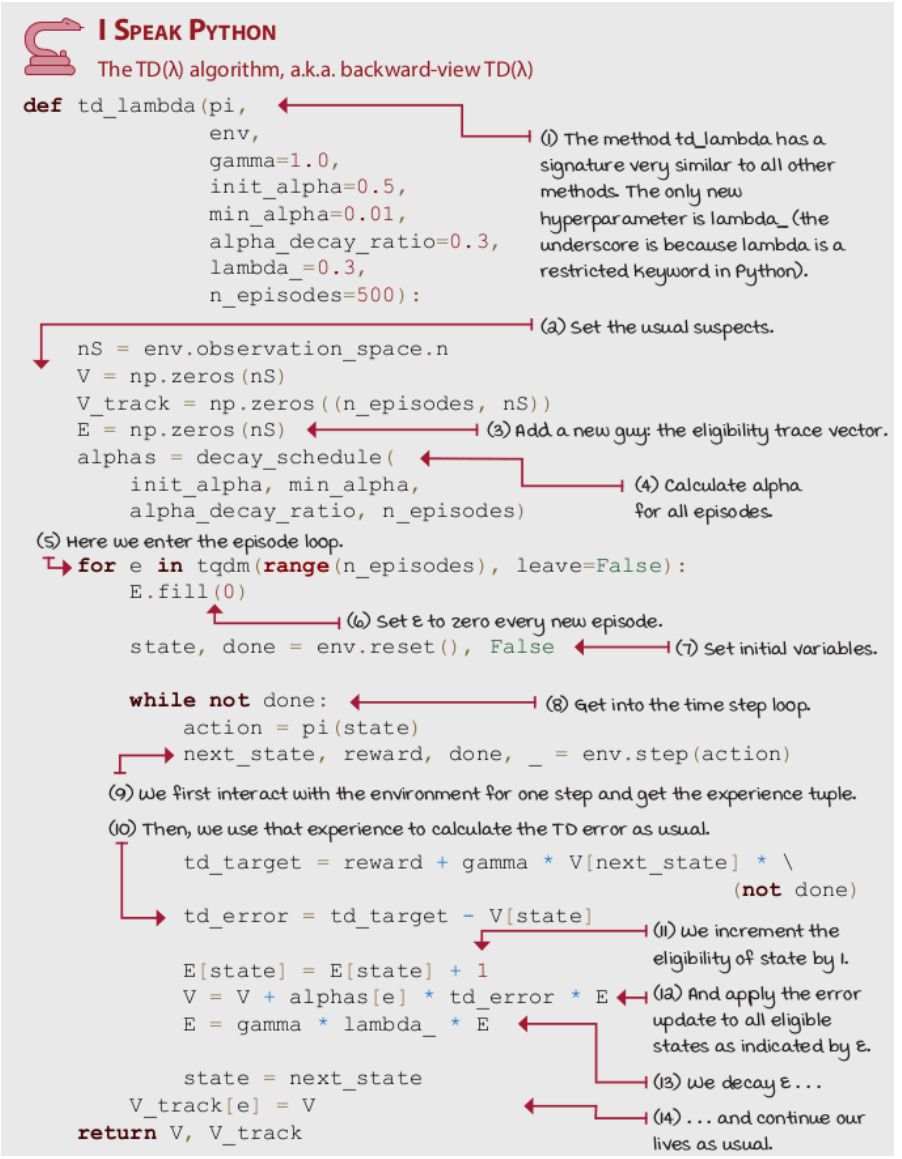


Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Tally it Up

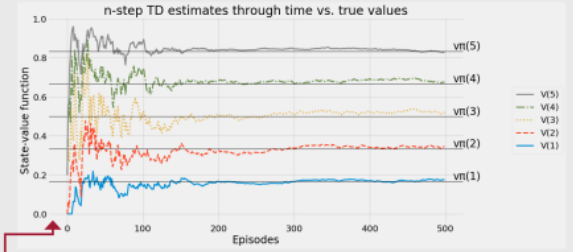
Running estimates that *n*-step TD and TD(λ) produce in the RW environment

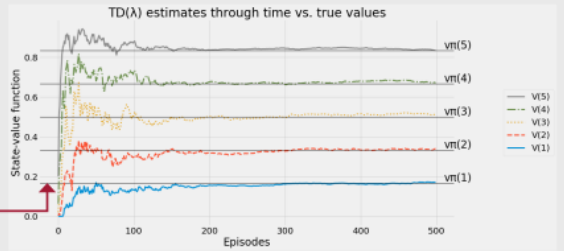
(1) I think the most interesting part of the differences and similarities of MC, TD, *n*-step TD, and TD(lambda) can be visualized side by side. For this, I highly recommend you head to the book repository and check out the corresponding Notebook for this chapter. You'll find much more than what I've shown you in the text.

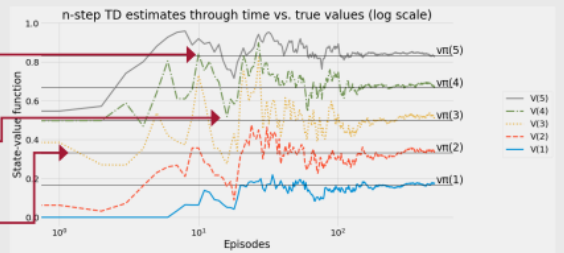
(2) But for now I can highlight that *n*-step TD curves are a bit more like MC: noisy and centered, while TD(lambda) is a bit more like TD: smooth and off-target.

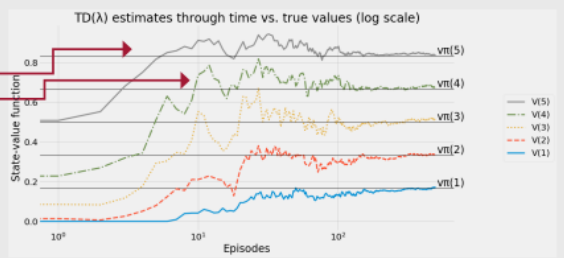
(3) When we look at the log-scale plots, we can see how the high variance estimates of *n*-step TD (at least higher than TD(lambda) in this experiment), and how the running estimates move above and below the true values, though they're centered.

(4) TD(lambda) values aren't centered, but are also much smoother than MC. These two are interesting properties. Go compare them with the rest of the methods you've learned about so far!









—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Демо

Кінець

Література

- David Silver, Lecture 4: Model-Free Prediction