Навчання нейронних мереж з підкріпленням

Матеріали лекційних занять

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

Огляд основ машинного навчання

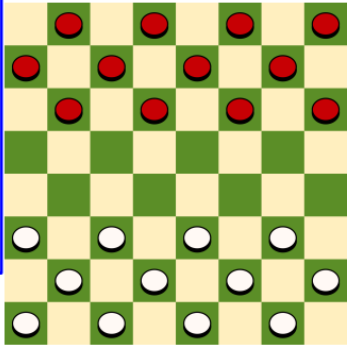
Що таке машинне навчання?

Визначення за Артур Семюель

Артур Семюель (1959): Машинне навчання - це область навчання, яка надає комп'ютеру можливість вчитися не будучи явно запрограмованим.

A. L. Samuel*

Some Studies in Machine Learning
Using the Game of Checkers. II—Recent Progress



Визначення за Том Мітчелл

Том Мітчелл (1998): Комп'ютерна програма, яка учиться з досвіду E по відношенню до деякого класу задач T та міри продуктивності P називається машинним навчанням, якщо її продуктивність у задачах з T, що вимірюється за допомогою P, покращується з досвідом E.

- Досвід (дані): ігри в які грає програма сама з собою
- Вимір продуктивності: коефіцієнт виграшу

Класичне програмуванн **vs** машинне навчання

Типи навчання

За характером навчальних даних (досвіду) машинне навчання поділяють на чотири типи: контрольоване (з учителем), напівконтрольоване, неконтрольоване (без учителя) та з підкріпленням.

Контрольоване навчання **Supervised learning**	**Напівконтрольоване навчання** **Semi-supervised learning**	**Неконтрольоване навчання** **Unsupervised learning**	**Навчання з підкріплення** **Reinforcement learning**
Дані: (x, y) x – приклад, y – мітка	**Дані:** (x, y) та $x, \lvert(x, y)\rvert < \lvert x\rvert$ x – приклад, y – мітка	**Дані:** x x – приклад, немає міток!	**Дані:** пари **стан-дія**
Мета – знайти функцію відображення $x \longrightarrow y$	**Мета** – знайти функцію відображення або категорію $x \longrightarrow y$	**Мета** – знайти правильну категорію.	**Мета** – максимізація загальної винагороди, отриманої агентом при взаємодії з навколишнім середовищем.
Приклад  Це є яблуко.	**Приклад**  Це є яблуко.	**Приклад**  Цей об'єкт схожий на інший.	**Приклад**  Їжте це, бо це зробить вас сильнішим.

Як вчиться людина?

- Ми та інші розумні істоти, вчимось завдяки взаємодії із своїм оточенням
- Взаємодії часто бувають послідовними - майбутні взаємодії можуть залежати від попередніх
- Ми направлені на результат
- Ми можемо вчитися не маючи прикладів оптимальної поведінки

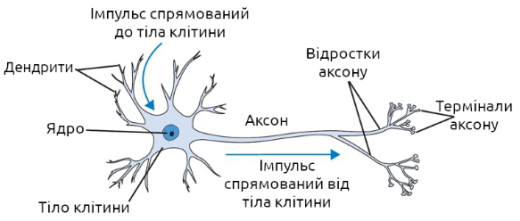
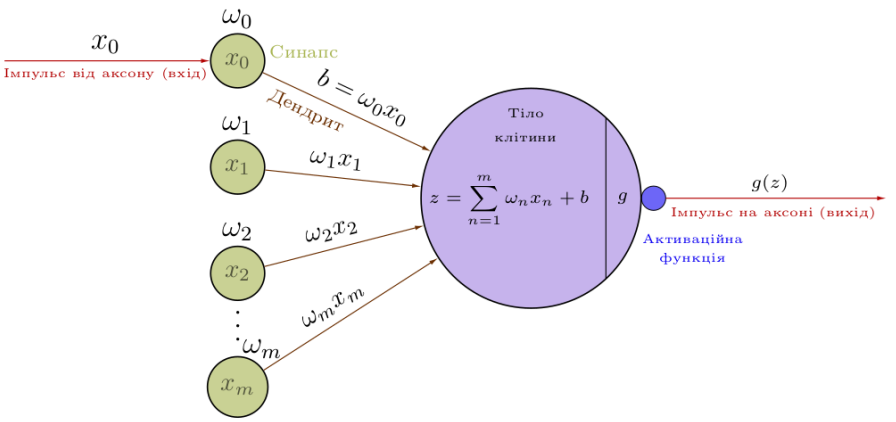
Мозок людини

Базовою обчислювальною одиницею мозку є нейрон. Мозок дорослої людини складається з 86 мільярдів нейронів, які з'єднані між собою приблизно 10^{14} – 10^{15} синапсами.

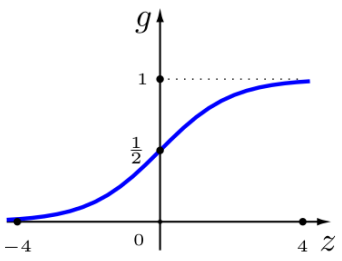
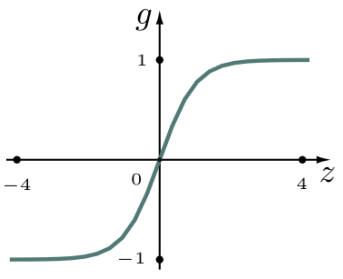
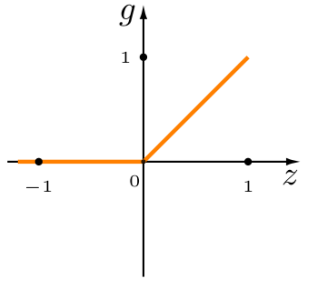
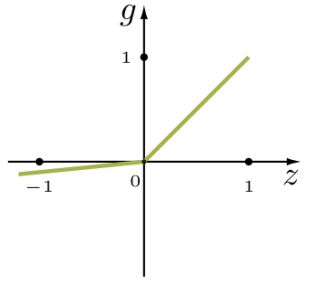
—

Джерело: F. A. Azevedo та ін., 2009.

Біологічний та штучний нейрон

Біологічний нейрон	Штучний нейрон
	

Деякі функції активації

Sigmoid	**Tanh**	**ReLU**	**Leaky ReLU**
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = max(0, z)$	$g(z) = max(\epsilon z, z)$ $\epsilon \ll 1$
			

Людина добре сприймати візуальну інформацію

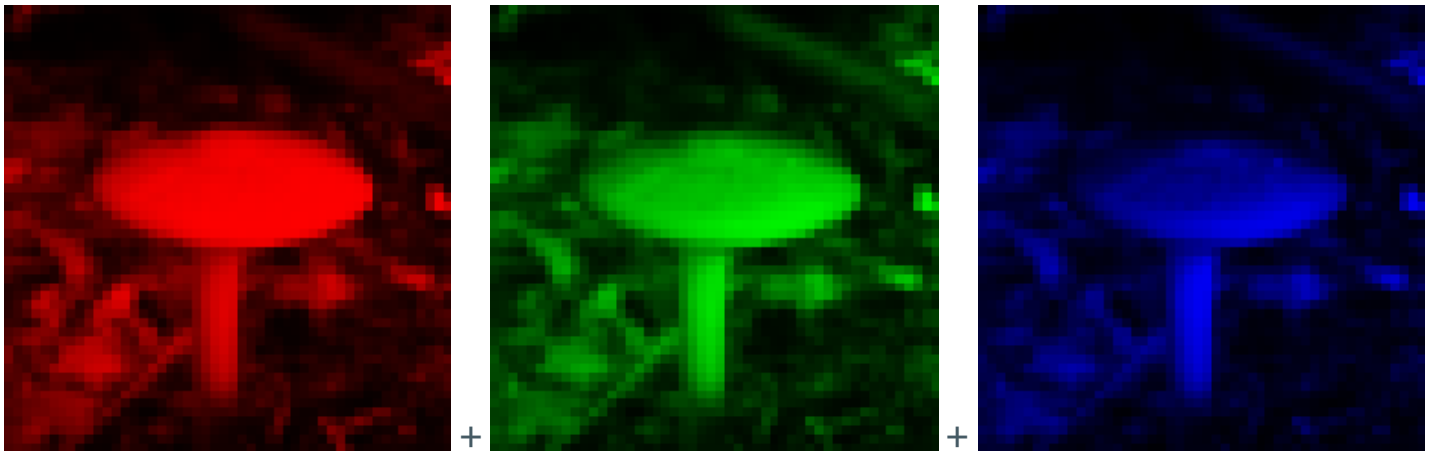
Що Ви бачите?

Собака-вівця чи швабра?

Людський мозок настільки добре інтерпретує візуальну інформацію, що розрив між зображенням та його семантичною інтерпретацією (пікселями) важко оцінити інтуїтивно:

Це мухомор.

Це мухомор.

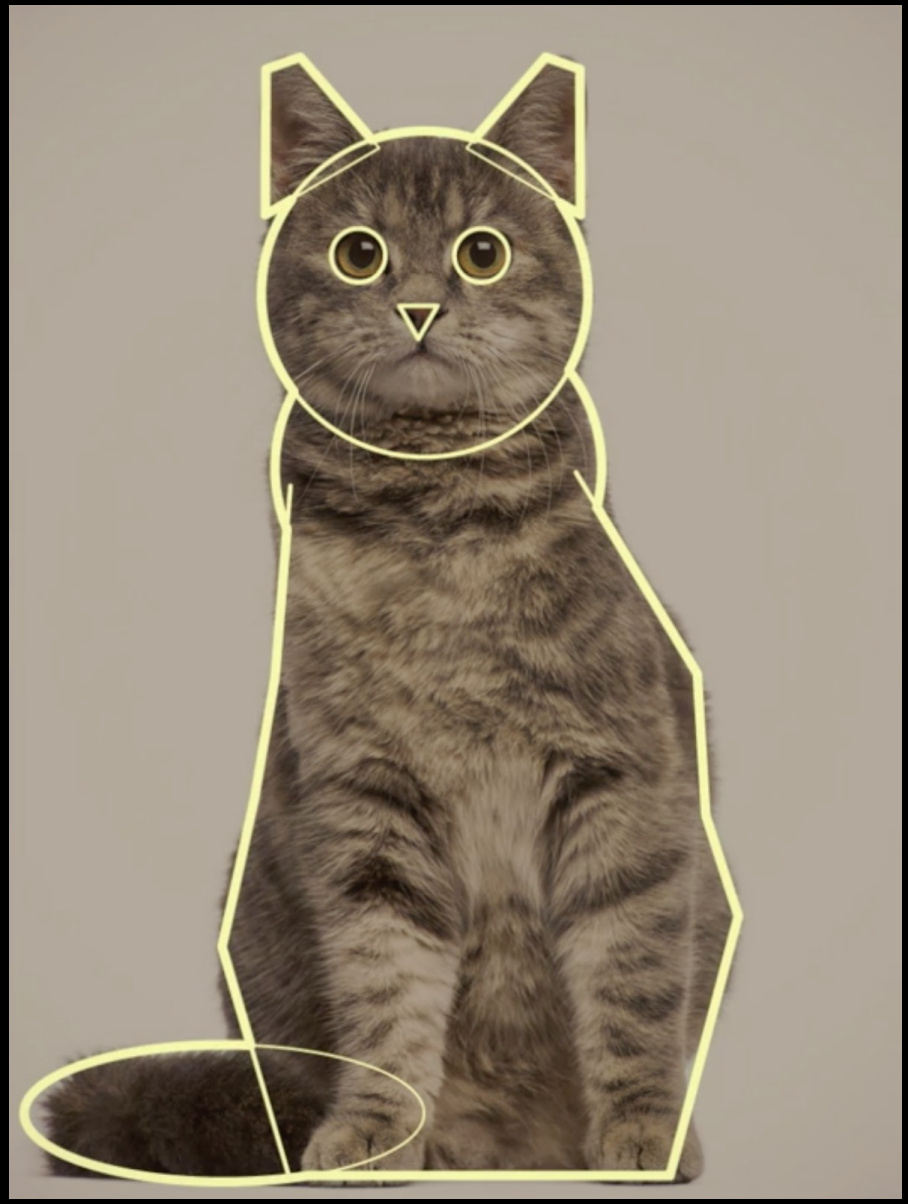


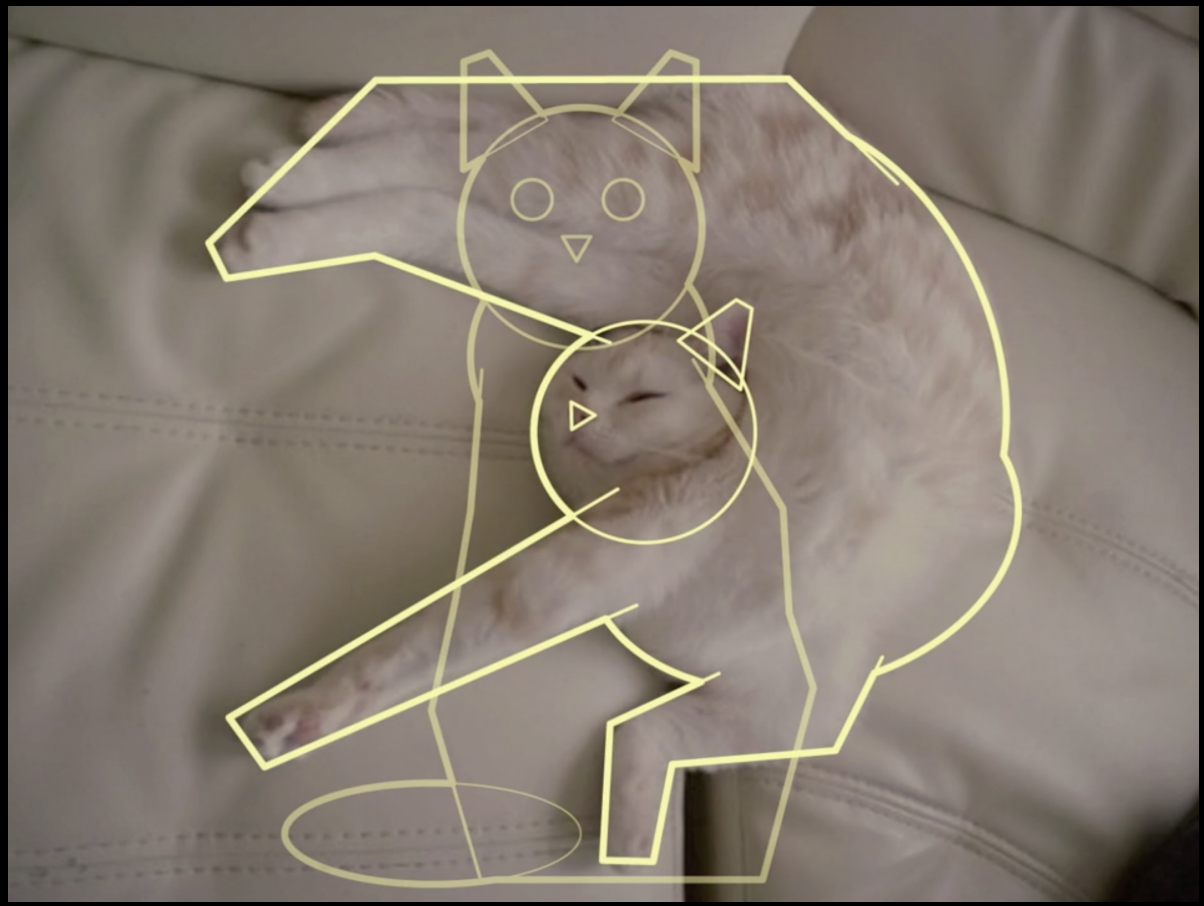
Це мухомор.

```
array([[[0.03921569, 0.03529412, 0.02352941, 1.        ],
        [0.2509804 , 0.1882353 , 0.20392157, 1.        ],
        [0.4117647 , 0.34117648, 0.37254903, 1.        ],
        ...,
        [0.20392157, 0.23529412, 0.17254902, 1.        ],
        [0.16470589, 0.18039216, 0.12156863, 1.        ],
        [0.18039216, 0.18039216, 0.14117648, 1.        ]],

       [[0.1254902 , 0.11372549, 0.09411765, 1.        ],
        [0.2901961 , 0.2509804 , 0.24705882, 1.        ],
        [0.21176471, 0.2       , 0.20392157, 1.        ],
        ...,
        [0.1764706 , 0.24705882, 0.12156863, 1.        ],
        [0.10980392, 0.15686275, 0.07843138, 1.        ],
        [0.16470589, 0.20784314, 0.11764706, 1.        ]],

       [[0.14117648, 0.12941177, 0.10980392, 1.        ],
        [0.21176471, 0.1882353 , 0.16862746, 1.        ],
        [0.14117648, 0.13725491, 0.12941177, 1.        ],
        ...,
        [0.10980392, 0.15686275, 0.08627451, 1.        ],
        [0.0627451 , 0.08235294, 0.05098039, 1.        ],
        [0.14117648, 0.2       , 0.09803922, 1.        ]],

       ...,
```

Це мухомор.

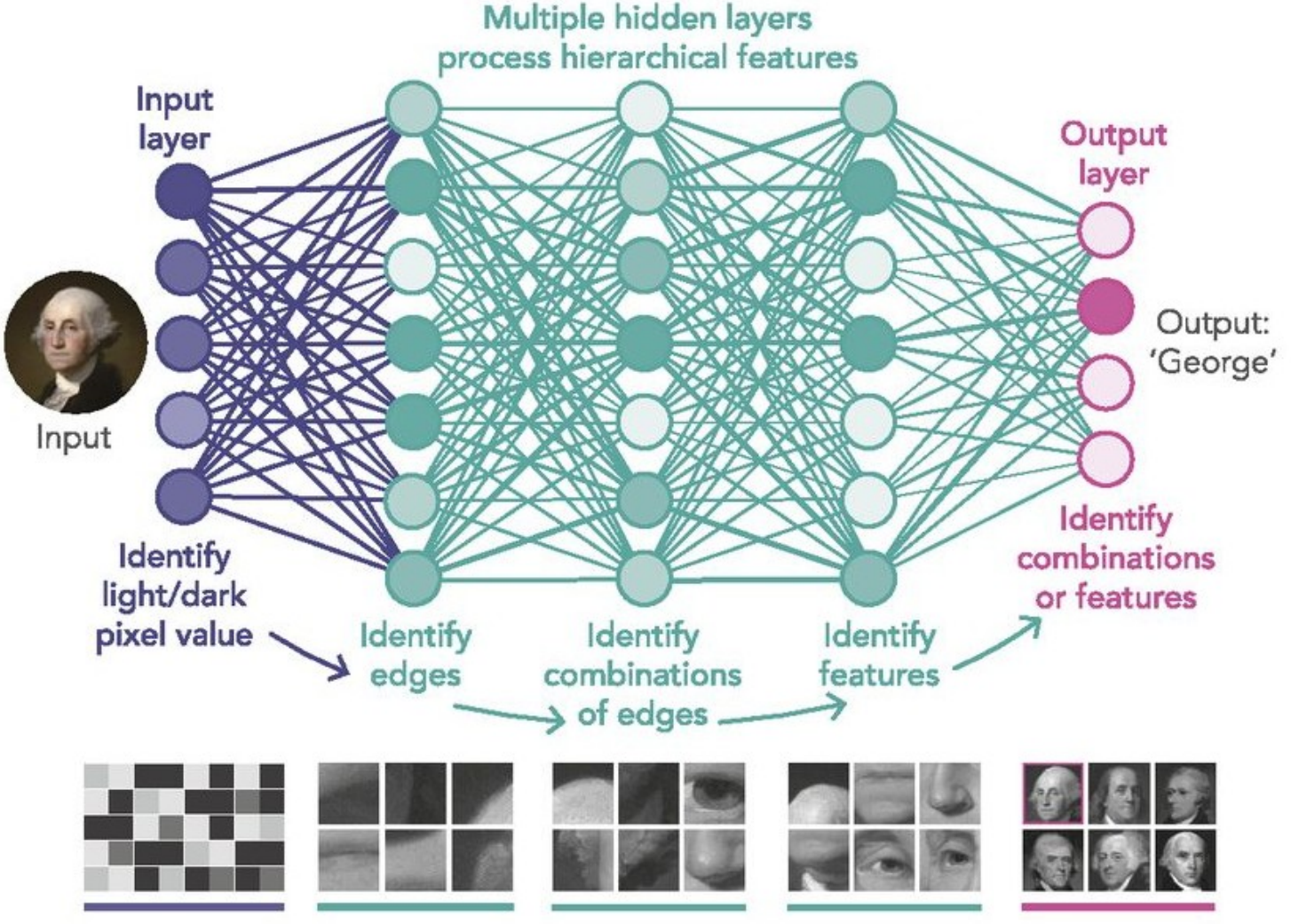
Як навчить машин бачити?





Для пошуку шаблону в даних (витягування семантичної інформації, ознак) потрібна побудова складних моделей, які б отримати вручну було б дуже складно.

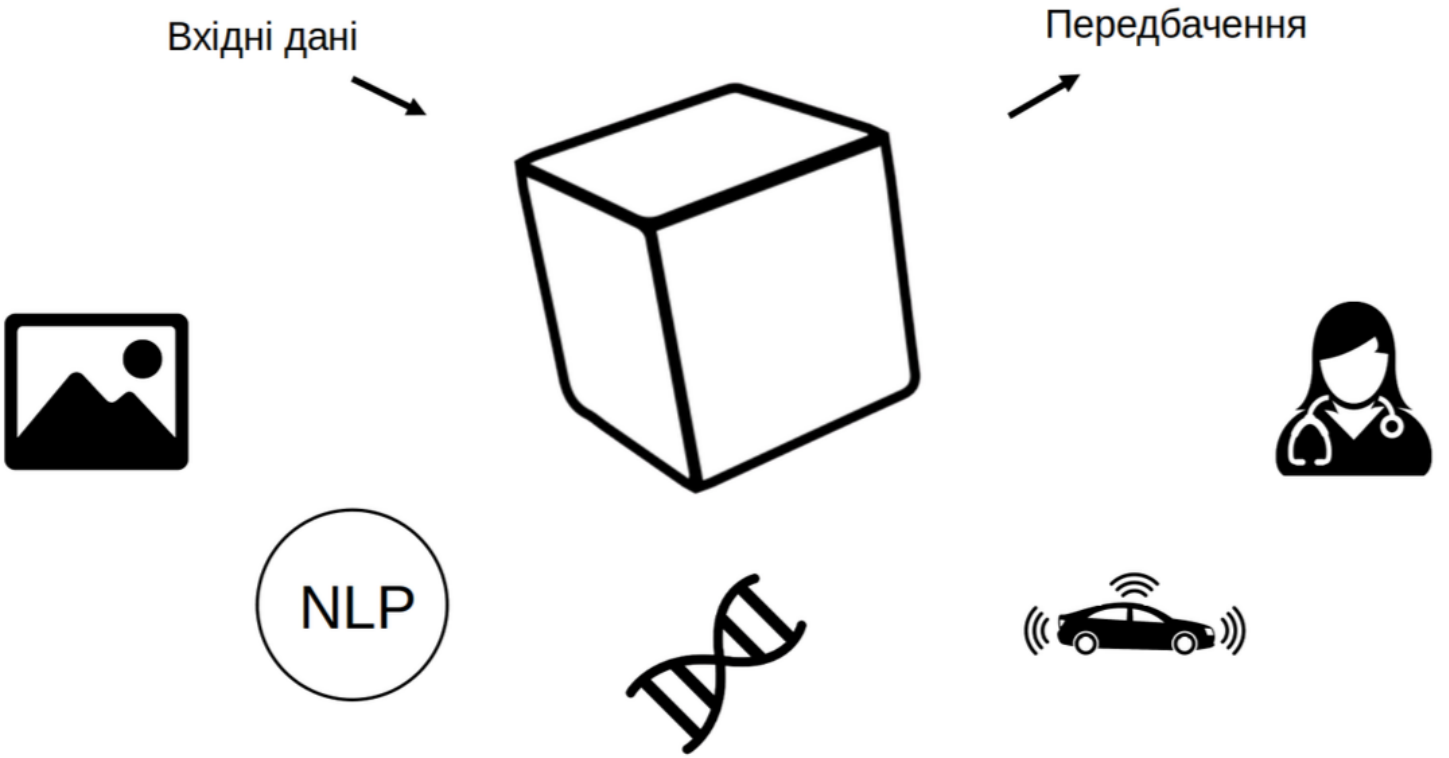
Однак, можна написати програму, яка буде вчитись знаходити шаблон в даних самостійно.


Multiple hidden layers
process hierarchical features
Input
layer
Output
layer
Output:
'George'
Input
Identify
light/dark
pixel value
Identify
edges
Identify
combinations
of edges
Identify
features
Identify
combinations
or features

Що входить до задачі машинного навчання?

- Постановка проблеми + дані
- Навчання моделі
- Визначення функції втрат
- Вибір алгоритму оптимізації

Які дані використовуються?



Ознаки у машинному навчанні

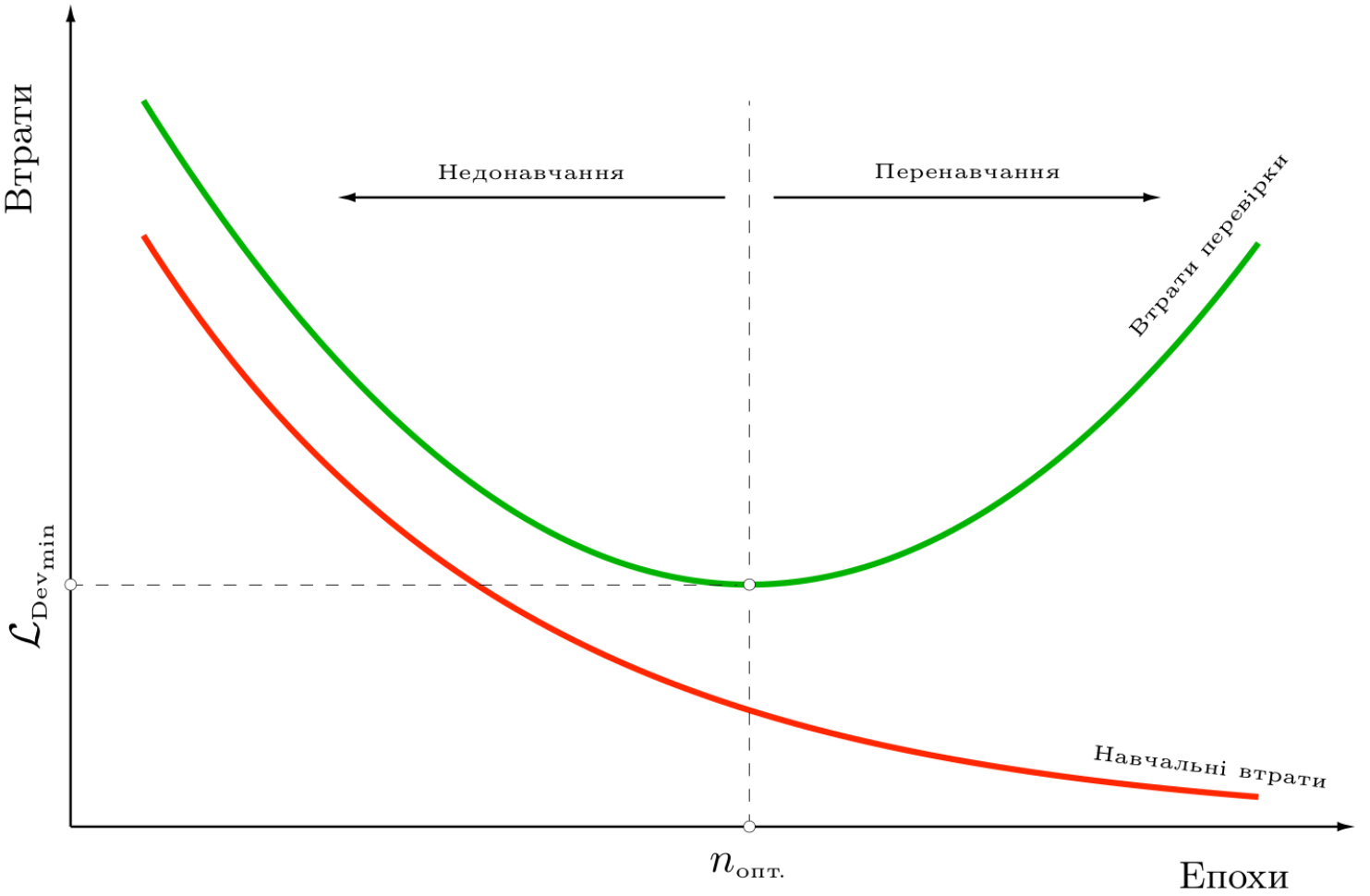
Ознаки - це спостереження, які використовуються для прийняття рішень моделлю.

- Для класифікації зображень кожен піксель є ознакою
- Для розпізнавання голосу, частота та гучність є ознаками
- Для безпілотних автомобілів дані з камер, радарів і GPS є ознаками

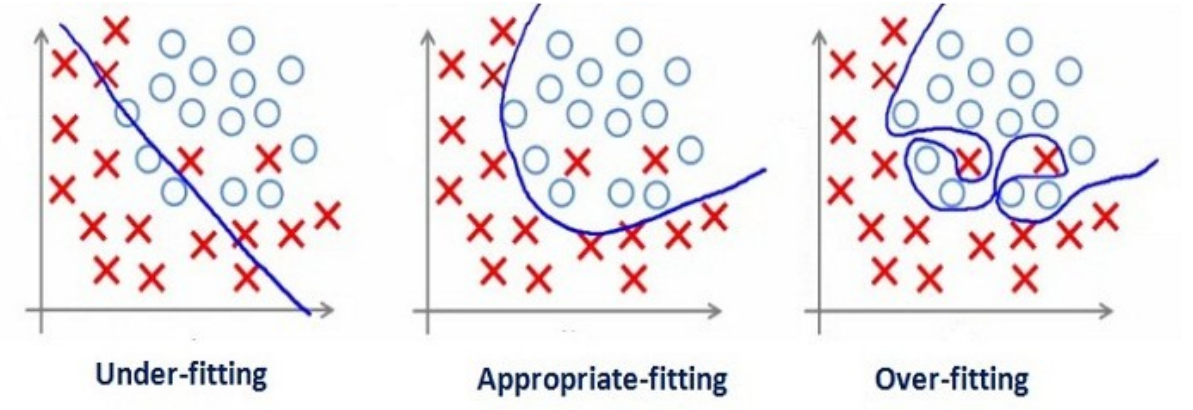
Типи ознак у робототехніці

- Пікселі (RGB дані)
- Глибина (сонар, лазерні далекоміри)
- Орієнтація або прискорення (гіроскоп, акселерометр, компас)

Недонавчання **vs** перенавчання

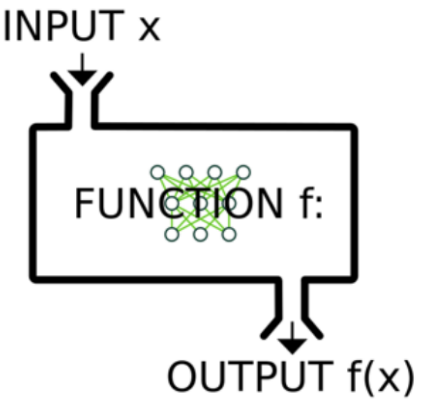


Недонавчання **vs** перенавчання



Що таке модель?

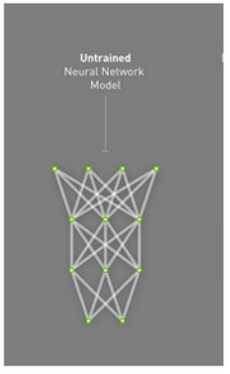
Хоча те, що знаходиться всередині глибинної нейронної мережі, може бути складним, за своєю суттю це просто функції. Вони беруть певні вхідні дані: INPUT x і генерують деякі вихідні дані: OUTPUT f(x)



З чого складається модель?

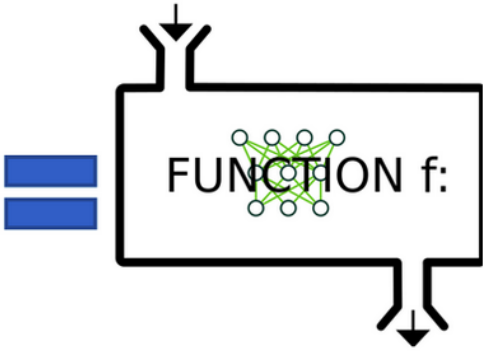
Компоненти моделі

Архітектура мережі = deploy.prototxt



Навчені ваги = ***.caffemodel

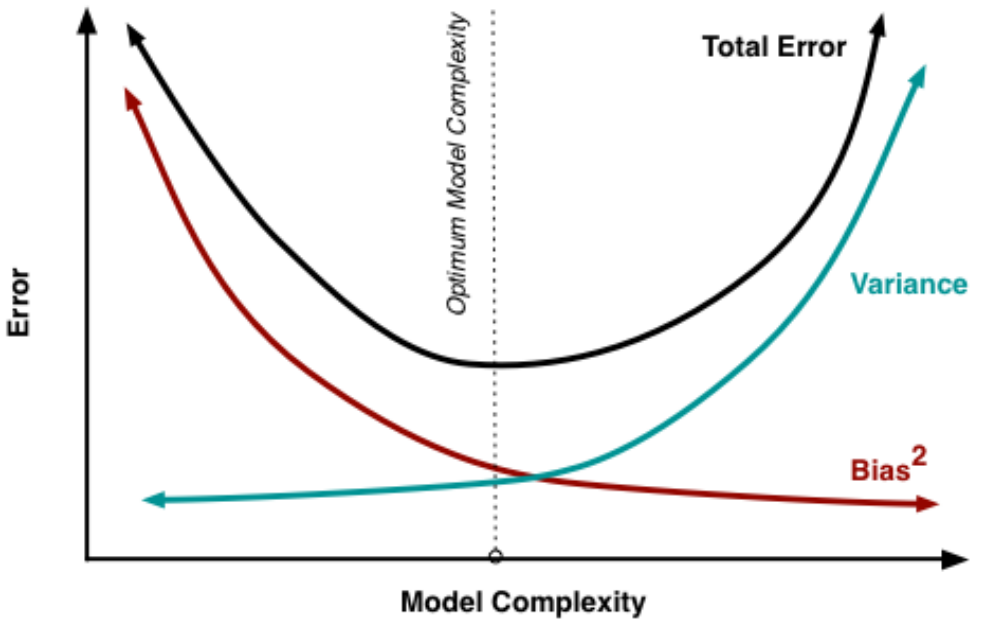
Модель



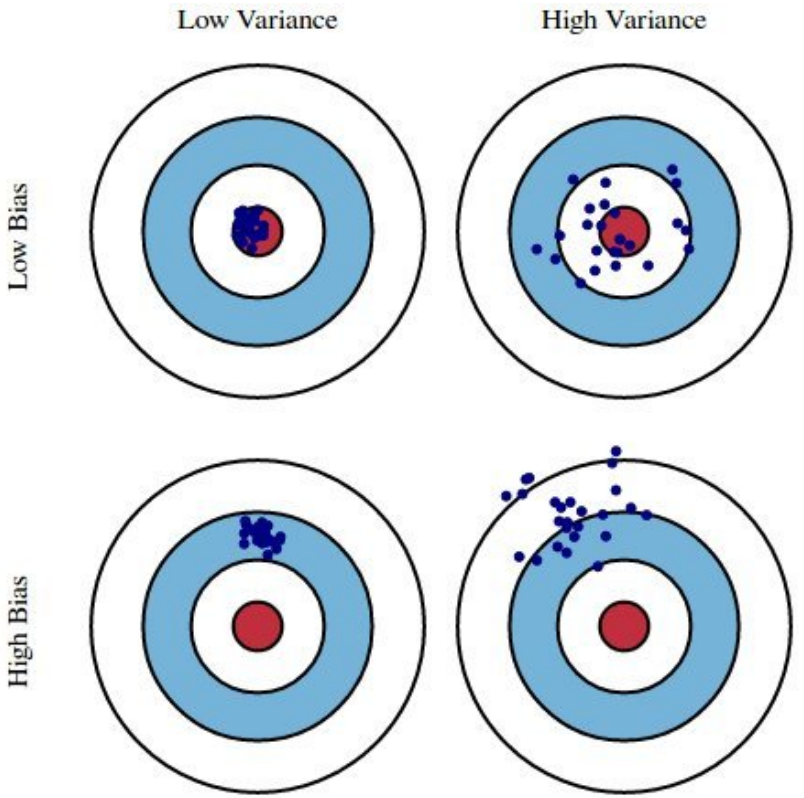
Джерела помилок моделі

- Зсув (Bias)
- Розкид (Variance)
- Шум (Irreducible error)

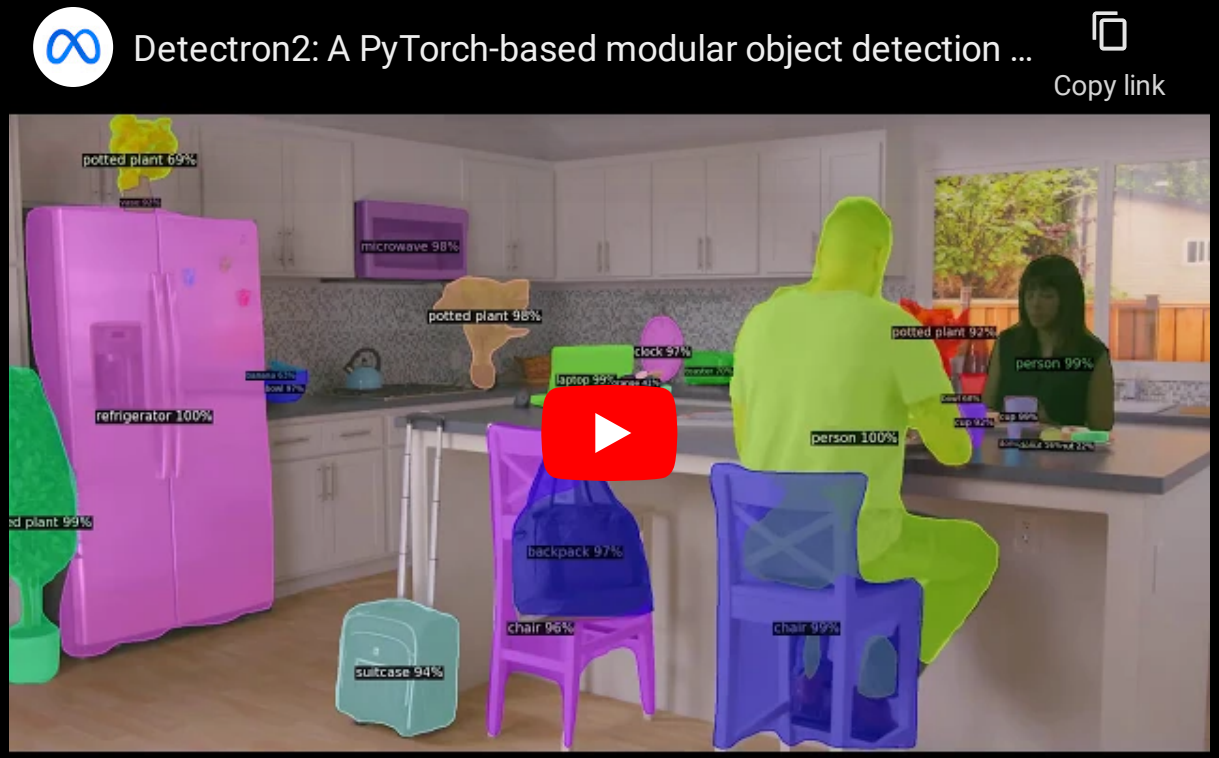
$$Err = Bias^2 + Variance + Irreducibleerror$$



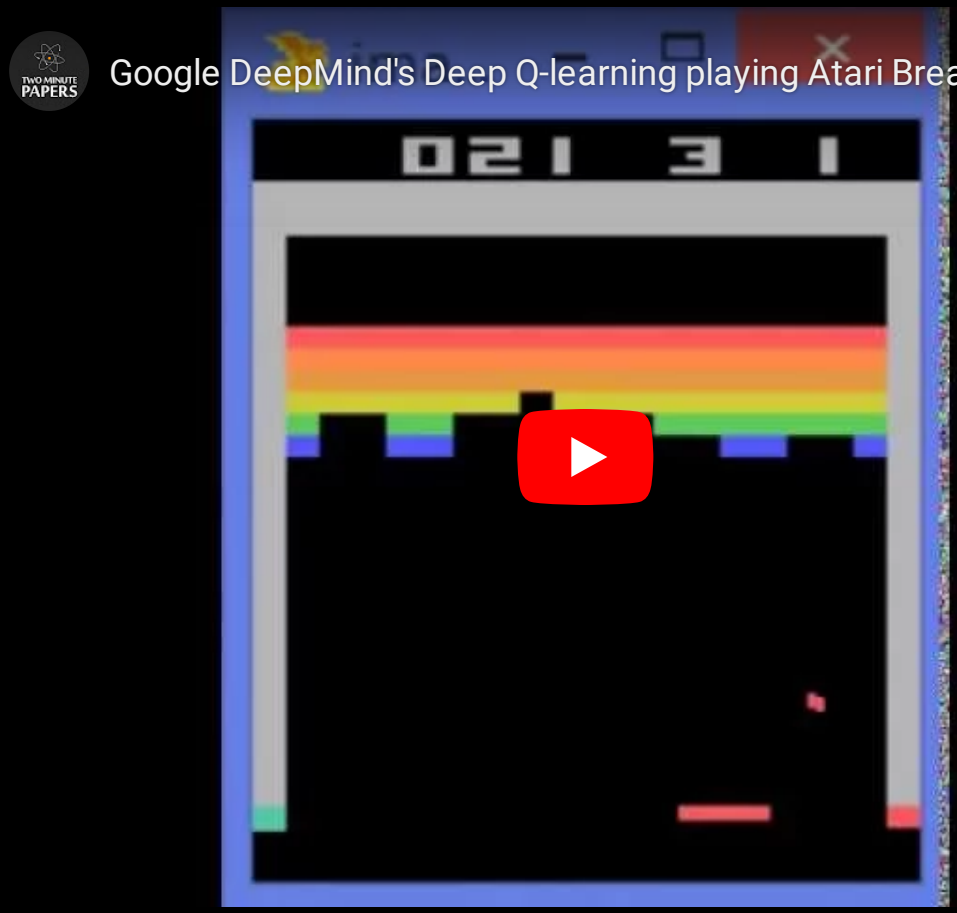
Інтуїція



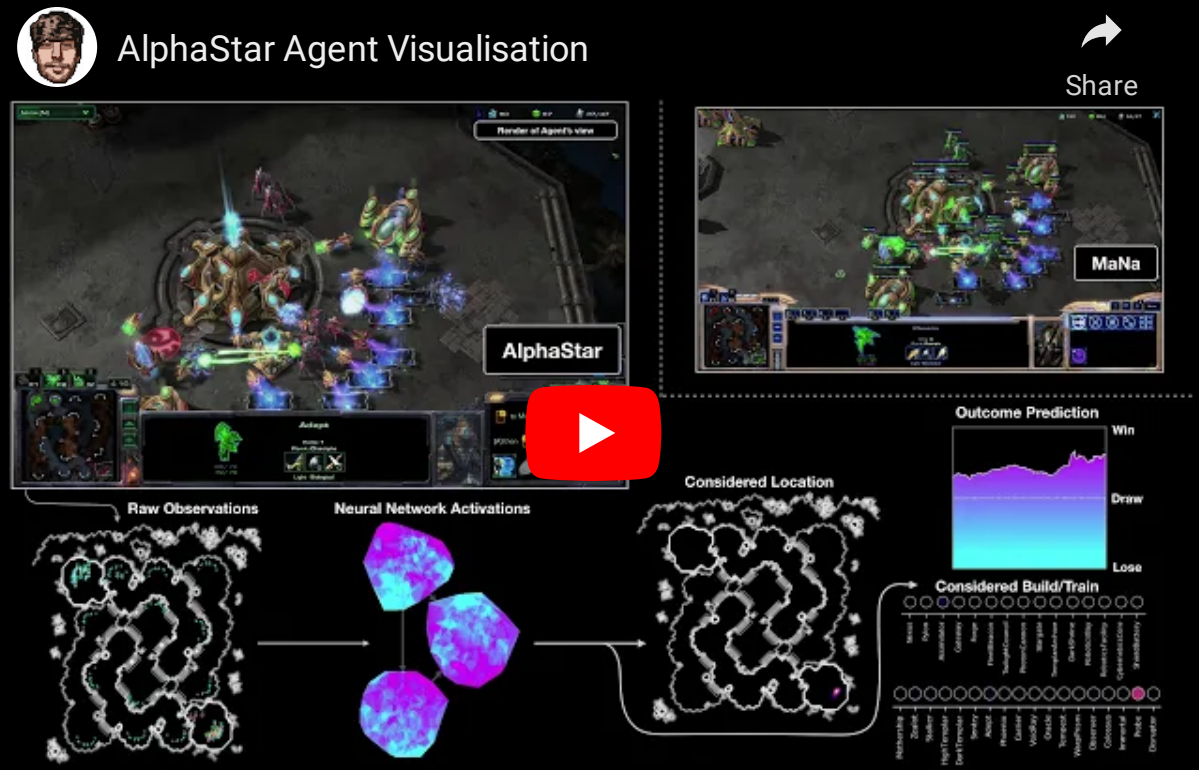
Applications and successes



Object detection, pose estimation, segmentation (2019)



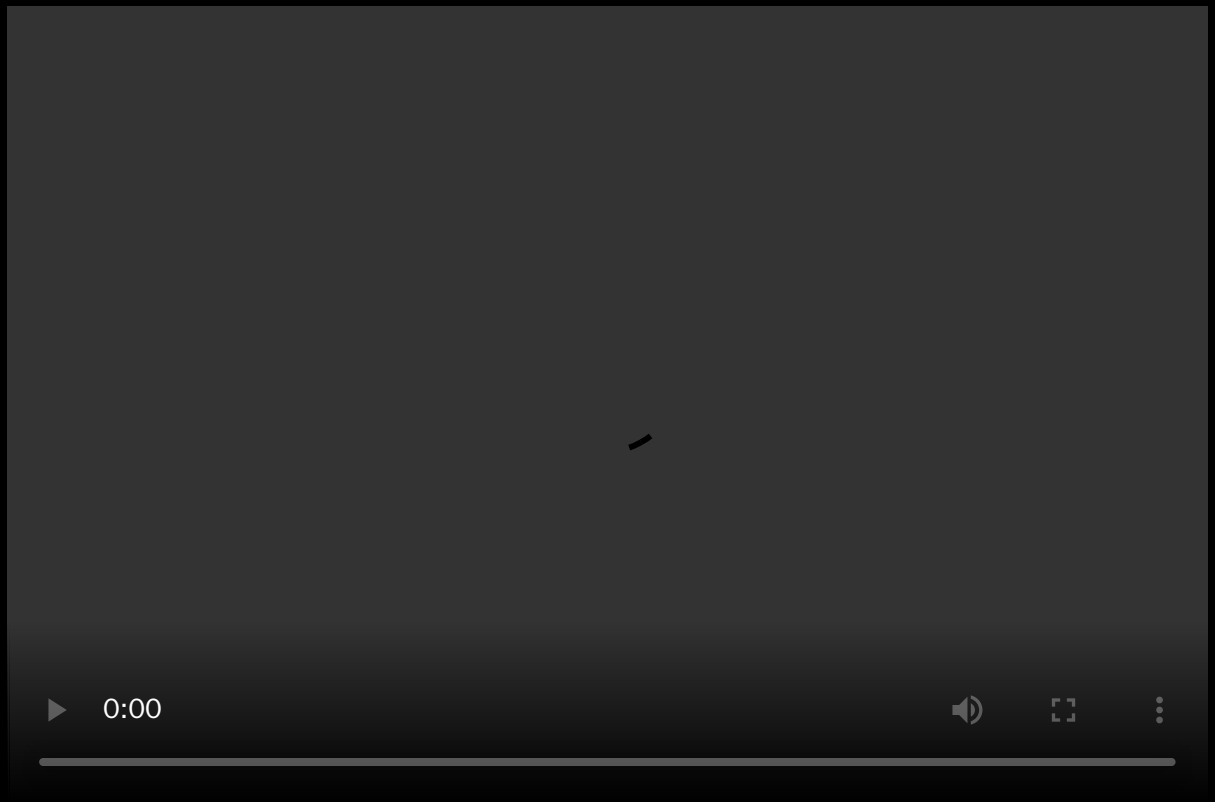
Reinforcement learning (Mnih et al, 2014)



Strategy games (Deepmind, 2016-2018)

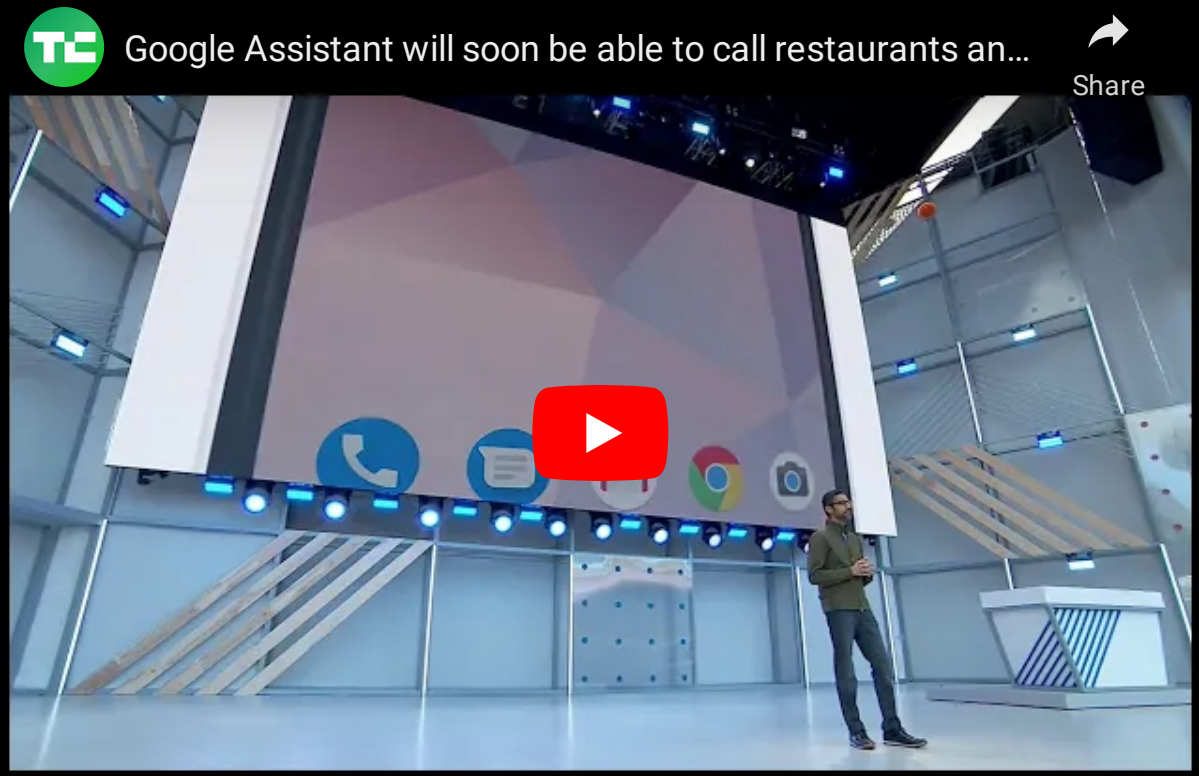
Autonomous cars (NVIDIA, 2016)

Autopilot (Tesla, 2019)

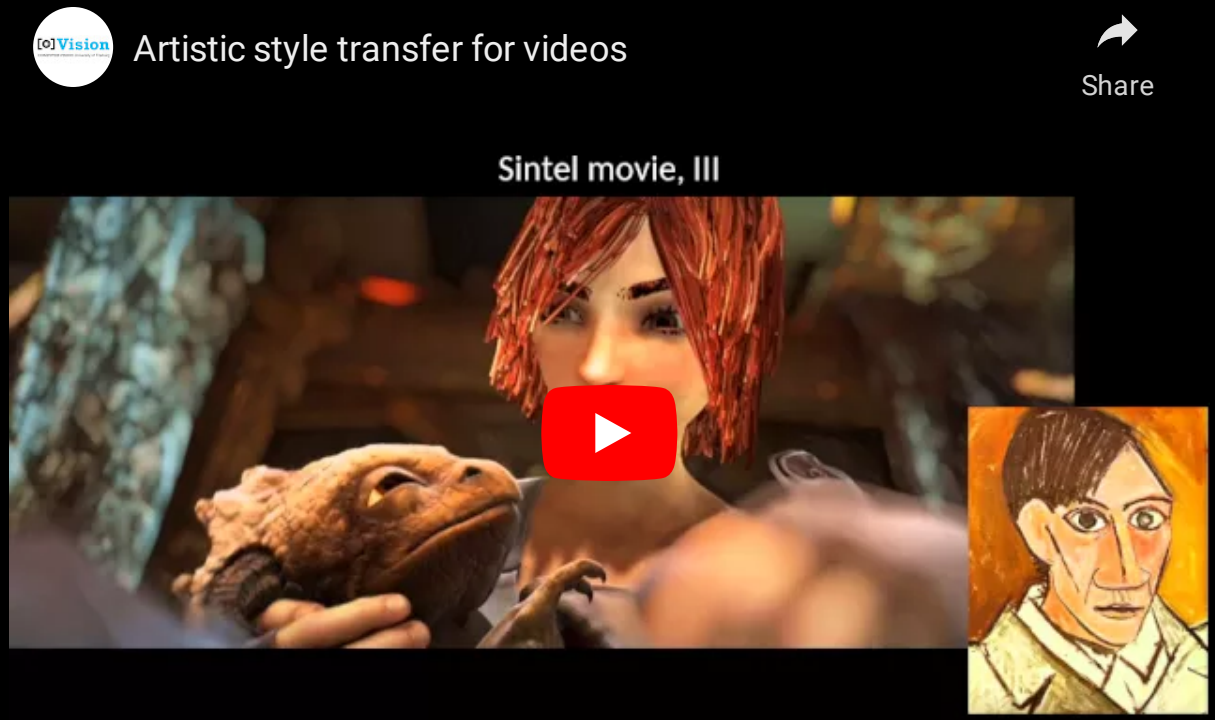


Physics simulation (Sanchez-Gonzalez et al, 2020)

AI for Science (Deepmind, AlphaFold, 2020)



Speech synthesis and question answering (Google, 2018)



Artistic style transfer (Ruder et al, 2016)

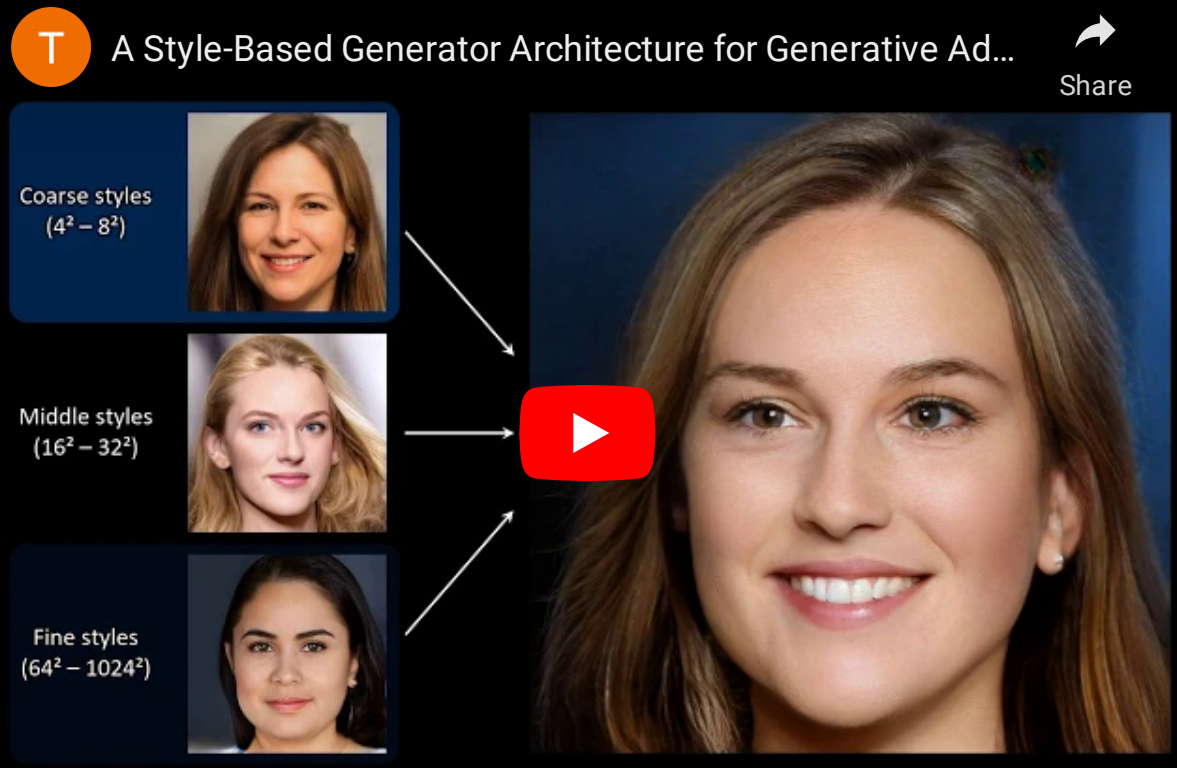
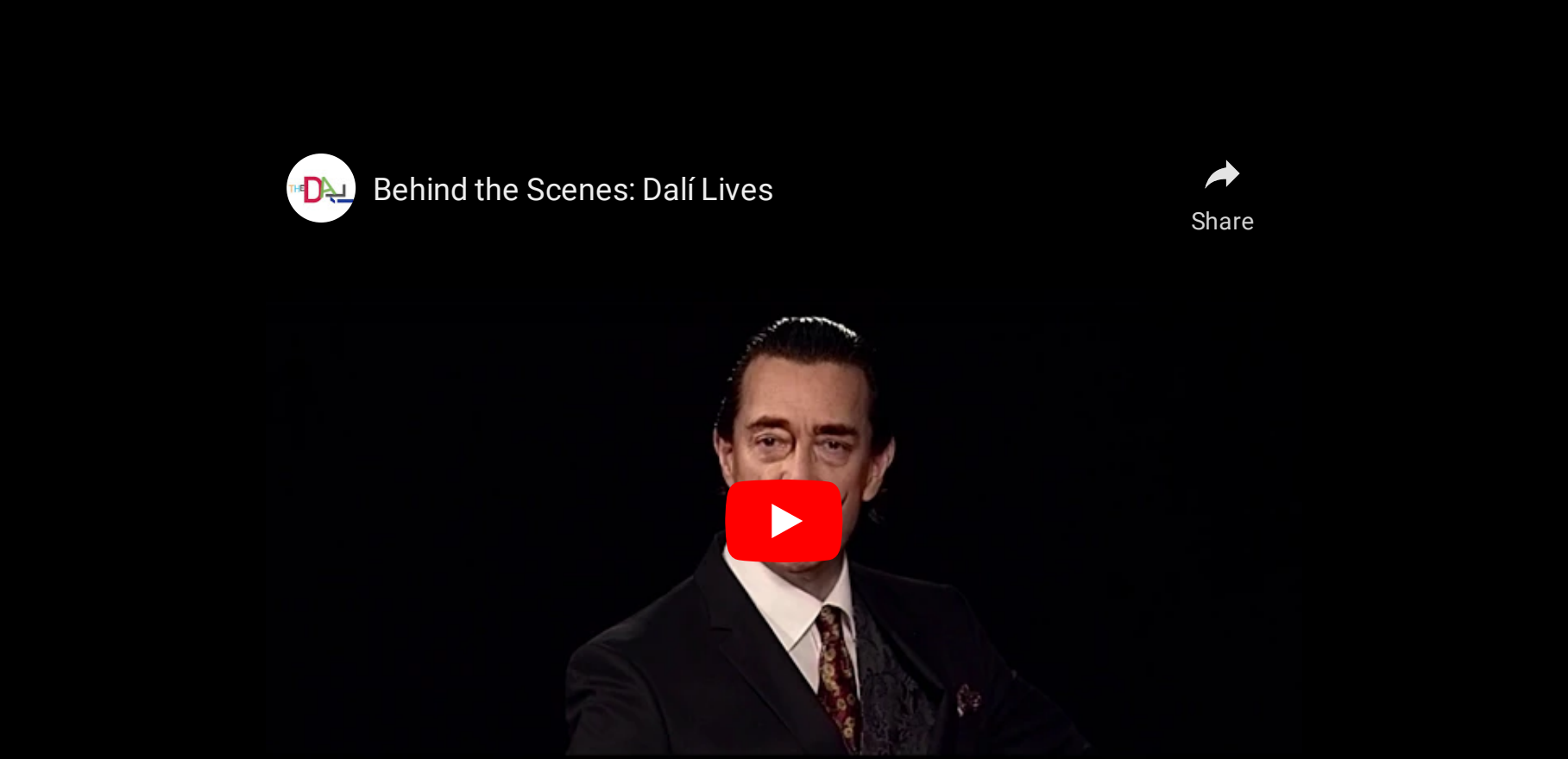


Image generation (Karras et al, 2018)

Music composition (NVIDIA, 2017)

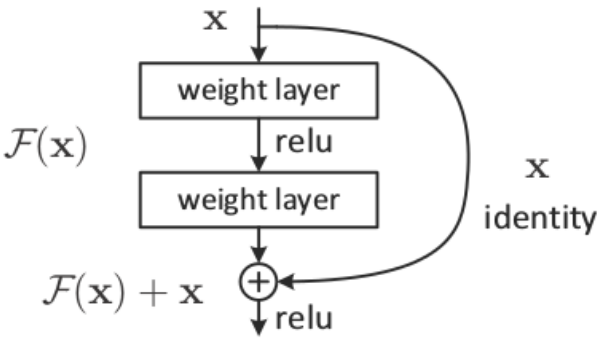


Dali Lives (2019)

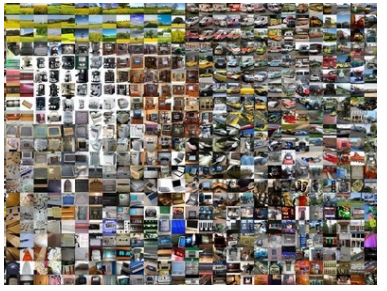
Асоціацією обчислювальної техніки (ACM) нагороджено в 2018 році премією Тюрінга таких науковців: ***Yann LeCun****,* ***Geoffrey Hinton****,* ***Yoshua Bengio*** *за концептуальні та інженерні прориви, які зробили в глибинних нейронних мережах.*

Чому **DL** працює?

Алгоритми (старі та нові)



Зростає кількість даних



Програмне забезпечення

Caffe2 · Microsoft CNTK · TensorFlow · theano · Chainer · PYTORCH · dmlc mxnet · K

Більш швидкі обчислювальні машини

"For the last forty years we have programmed computers; for the next forty years we will train them."

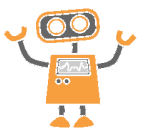
Chris Bishop, 2020.

Вступ до навчання з підкріпленням

Основним викликом штучного інтелекту та машинного навчання є прийняття правильних рішень в умовах невизначеності

Визначення **RL**

Навчання з підкріпленням (reinforcement learning, RL) - сімейство алгоритмів, які вивчають оптимальну стратегію, метою якої є максимізація загальної винагороди, отриманої агентом при взаємодії з навколишнім середовищем.



- Наприклад, кінцевою винагородою більшості ігор є перемога. Система навчання з підкріплення може стати експертом у складних іграх, шляхом оцінювання послідовності попередніх ігрових ходів, які в підсумку призвели до перемоги або програшу.

Визначення **RL**

RL - наука про те, як приймати рішення на основі взаємодій

- Це вимагає від нас задуматися над:
  - часом
  - (довгостроковими) наслідками спричинені діями
  - збором досвіду
  - передбаченням майбутнього
  - боротьбою з невизначеністю

Застосування **RL**

- Ігри (Atari, AlphaGo)
- Робототехніка (End-to-End Training)
- Фінанси
- Взаємодія людини з комп’ютером
- ...

Причини використання **RL**

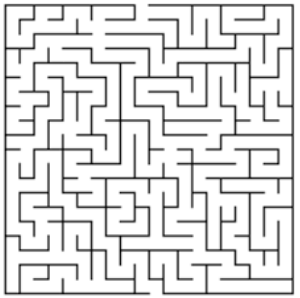
1. Пошук раніше невідомих рішень
    - Приклад, програма, яка може грати в Go краще, ніж будь-яка людина, будь-коли
2. Пошук рішень в режимі реального часу за непередбачених обставин
    - Приклад, робот, який може орієнтуватися на місцевості, яка значно відрізняється від будь-якої очікуваної місцевості

- Алгоритми навчання з підкріпленням намагаються задовільнити обидва випадки
- Зауважте, що другий пункт стосується не (просто) узагальнення - це більшою мірою про ефективне навчання в режимі реального часу під час взаємодії з середовищем

Агент **(agent)**

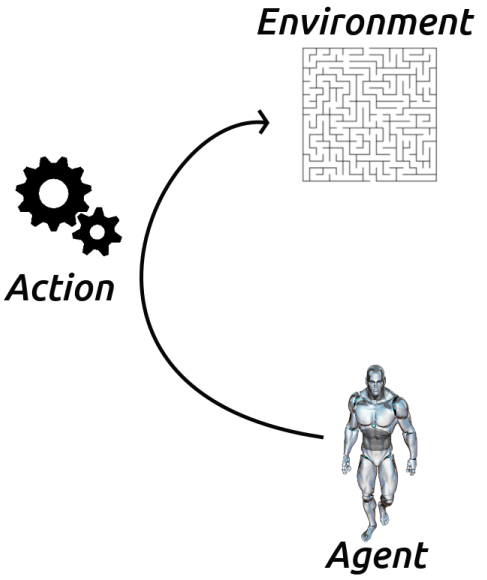
Агент (agent) - це те, що існує окремо від інших речей та використовує певну стратегію (policy) для максимізації очікуваної винагороди (reward), отриманої від переходу між станами середовища (environment).

Середовище **(environment)**



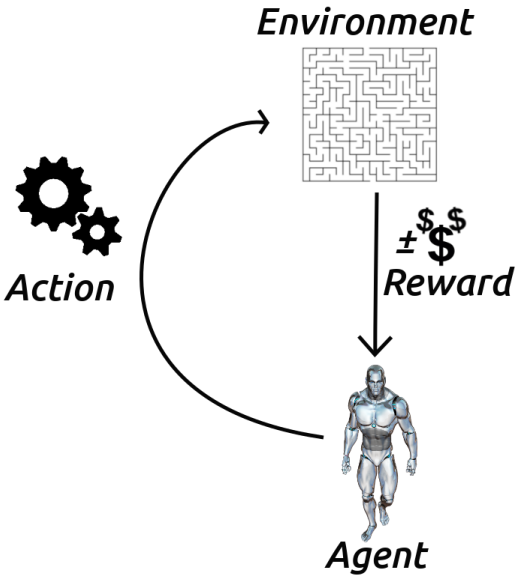
Середовище - це стохастичний та невизначений світ, в якому агент існує та діє.

Дія **(action)**



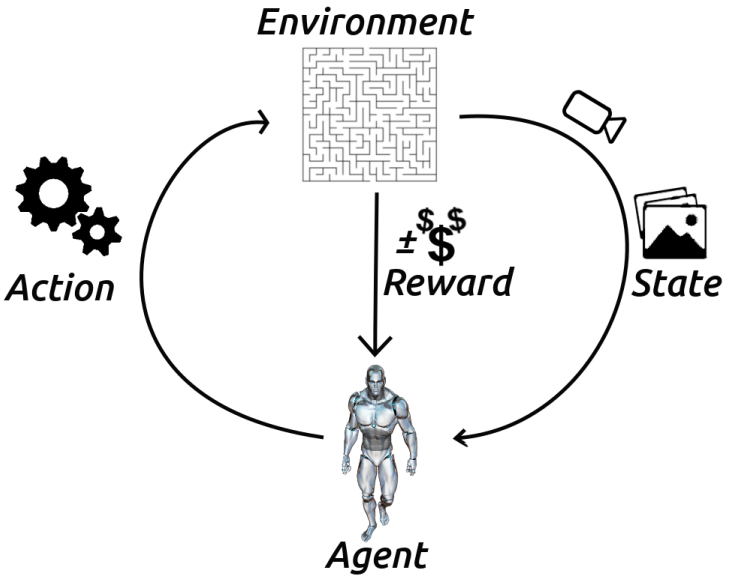
Дія - механізм, за допомогою якого агент переходить між дозволеними середовищем станами. Агент обирає дію, використовуючи стратегію.

Винагорода **(reward)**



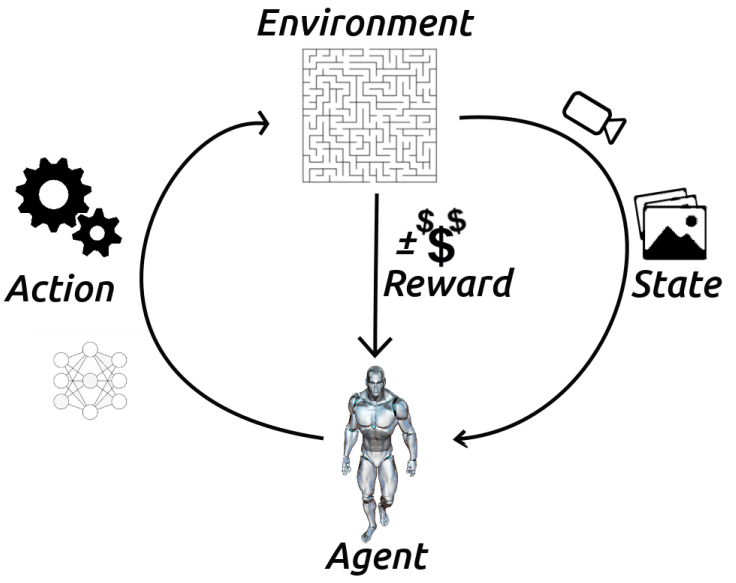
Винагорода - числовий результат, отриманий агентом у наслідок переходу між стананами, які визначені середовищем (дії).

Цикл взаємодії



Стан - значення параметрів, що описують поточну конфігурацію середовища. Агент використовує ці параметри для вибору дії.

Глибинне **RL (Deep RL)**



При глибинному навчанні з підкріпленням агент зазвичай обробляє 2D-зображення із використанням згорткових нейронних мереж (CNN) - це дає йому можливість навчатись "із побаченого" завдяки наскрізній мережі, яка перетворює набір пікселів у дії.

Характеристика **RL**

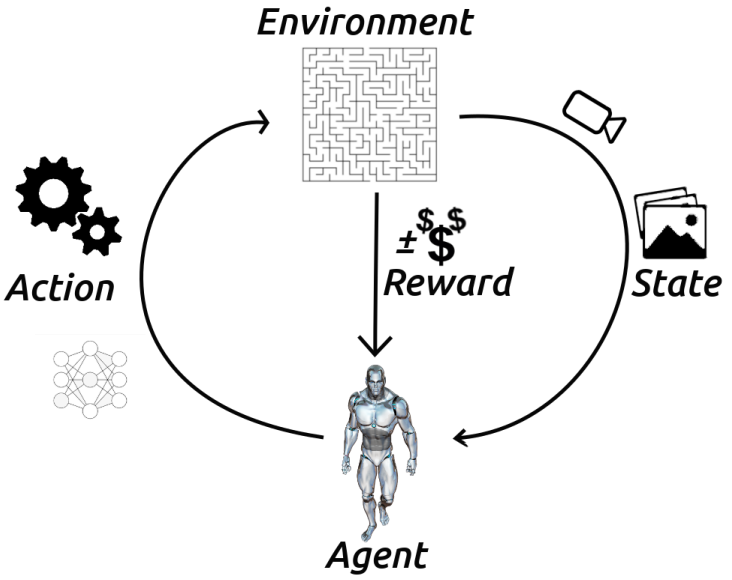
Чим навчання з підкріплення відрізняється від інших парадигм машинного навчання?

- Ніякого контролю, лише сигнал про винагороду
- Зворотній зв'язок може затримуватися, а не миттєво передаватися
- Час має значення
- Більш ранні рішення агента впливають на його наступні дії

Основні поняття **RL**

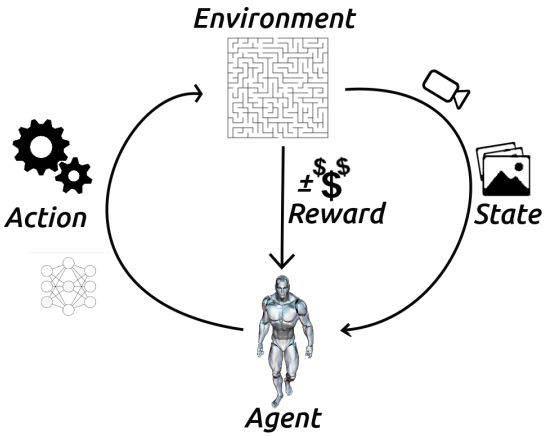
- Середовище (environment)
- Винагорода (reward)
- Агент (agent), який включає:
  - Стан агента (agent state)
  - Стратегію (policy)
  - Q-функцію, яка відома як функція значення стан-дія (state-action value function)
  - Модель (за бажанням)

Цикл взаємодії



Мета — оптимізувати загальну винагороду, отриману агентом при взаємодії з навколишнім середовищем.

Агент та середовище



На кожному кроці в момент часу t агент:

- Отримує спостереження O_t та винагороду R_t
- Виконує дію A_t

Середовище:

- Отримує дію A_t
- Продукує спостереження O_{t+1} та винагороду R_{t+1}

Винагорода

Винагорода R_t — це скалярний сигнал, який отримує агент у якості зворотного зв'язку від середовища.

- Показує, наскільки добре працює агент у момент часу t відповідно до поставленої мети.
- Завдання агента — максимізувати кумулятивну винагороду:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots$$

- G_t називається загальною винагородою (return) — сума всіх винагород, які агент розраховує отримати при дотриманні стратегії від певного стану до кінця епізоду.
    - Епізод — кожна спроба агента вивчити середовище.

Гіпотеза винагороди

Навчання з підкріпленням базується на гіпотезі винагороди:

> "Будь-яка мета може бути формалізована як результат максимізації сукупної винагороди."

Цінність

Очікувана сукупна винагорода від стану s називається цінністю (value):

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

- Цінність залежить від дій агента
- Метою є максимізація цінності $v(s)$ шляхом вибору агентом правильних дій
- Винагороди та цінності визначають користь станів та дій (немає контрольованого зворотного зв'язку)
- Зверніть увагу, що загальна винагорода та цінність можуть бути визначені рекурсивно:

$$\begin{aligned} G_t &= R_{t+1} + G_{t+1} \\ v(s) &= \mathbb{E}\left[R_{t+1} + v(S_{t+1}) \mid S_t = s\right] \end{aligned}$$

Цінність дій — Q-функція

- 'Q' означає якість (quality)

Q-функція дозволяє оцінити цінність (якість) дій :

$$q(s,a) = \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right] = \\ = \mathbb{E}\left[R_{t+1} + R_{t+2} + R_{t+3} + \cdots \mid S_t = s, A_t = a\right]$$

Q-функція — функція якості, яка передбачає очікувану загальну винагороду (return) від виконання дій у певному стані та дотриманні заданої стратегії.

- Значення стану та дії буде детальніше розглянуто пізніше

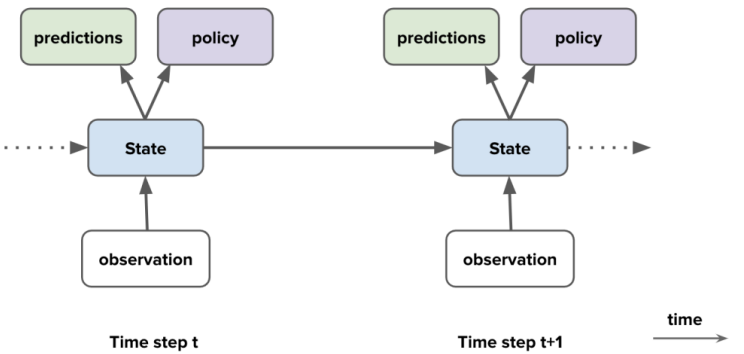
Ключові поняття

Формалізм навчання з підкріплення включає у себе такі поняття:

- Середовище (динаміка задачі)
- Винагорода (визначає мету)
- Агент, який включає:
  - Стан агента
  - Стратегію (policy)
  - Q-функцію, відома також як функція цінності стан-дія (state-action value function)
  - Модель (за бажанням)

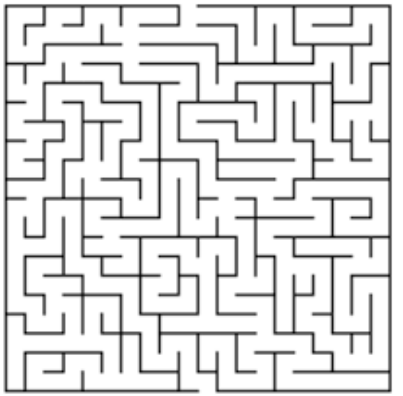
Компоненти агента

- Стан агента (agent state)
- Стратегія
- Q-функція
- Модель



Стан середовища

- Стан середовища — це внутрішній стан середовища
- Зазвичай цей стан невидимий агенту
- Навіть якщо стан середовища видимий агенту він може містити багато зайвої інформації

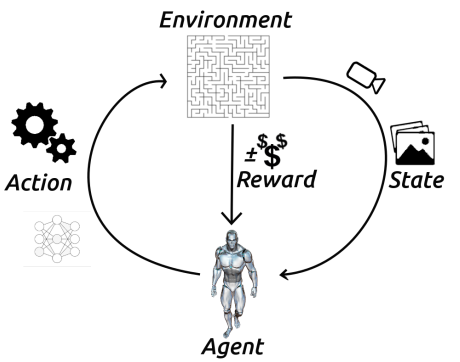


Стан агента

- Історія — це послідовність з спостережень O, дій A та винагород R:

$$H_t = O_0, A_0, R_1, O_1, \cdots, O_{t-1}, A_{t-1}, R_t, O_t$$

- Історія використовується для побудови стану агента S_t



Повністю оглядове середовище

Припустимо, що агент бачить повністю стан середовища. Тоді:

- спостереження = стан середовища
- Стан агента є просто спостереженням:

$$S_t = O_t = \text{стан середовища}$$

У цьому випадку агент бере участь у процесі прийняття рішень Маркова (Markov decision process - MDP). Цей процес названий на честь Андрія Маркова. MDP слугує математичною основою для того, щоб змоделювати прийняття рішення в ситуаціях, де результати є частково випадкові та частково під контролем агента, який приймає рішення.

—

Додатково можна почитати про MDP у розділі 3 Reinforcement Learning: An Introduction або у розділі 2 Algorithms for Reinforcement Learning

Марковські процеси прийняття рішень **(MDPs)**

MDPs надають корисний математичний апарат

Визначення. Процес прийняття рішень є Марковським, якщо

$$\boxed{p(r,\ s \mid S_t,\ A_t) = p(r,\ s \mid H_t,\ A_t)}$$

- Це означає, що стан містить все, що нам потрібно знати з історії
- Це не означає, що стан містить усе, але просто додавання історії не допомагає
- $\Longrightarrow$ Як тільки стан стане відомим, історію можна буде відкинути
  - Усе середовище + стан агента — Марковські
  - Повна історія H_t є Марковською
- Як правило, стан агента S_t є деяким стисненням H_t
- Примітка: S_t — стан агента, а не середовища

Частково оглядове середовище

- Часткова оглядовість: агент отримує неповну інформацію про стан середовища
  - Камера зору не повідомляє роботу його абсолютне місце розташування
  - Агент, що грає в покер, бачить лише відкриті карти
- Тепер спостереження не є Марковським процесом
- Формально — це частково оглядовий процес прийняття рішень Маркова (partially observable Markov decision process, POMDP)
- Стан середовища все ще може бути Марковським, але агент цього не знає
- Ми все ще можемо побудувати стан агента, який буде Марковським

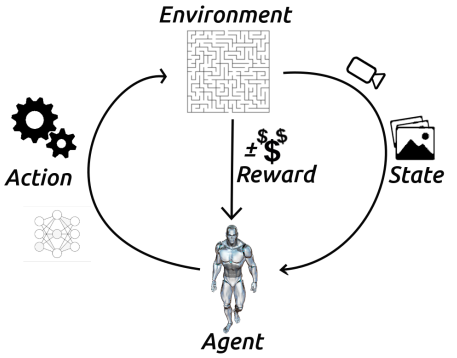
Стан агента

- Дії агента залежать від його стану
- Стан агента є функцією історії
- Для конкретного стану: $S_t = O_t$
- Більш загально:

$$\boxed{S_{t+1} = u(S_t, A_t, R_{t+1}, O_{t+1})}$$

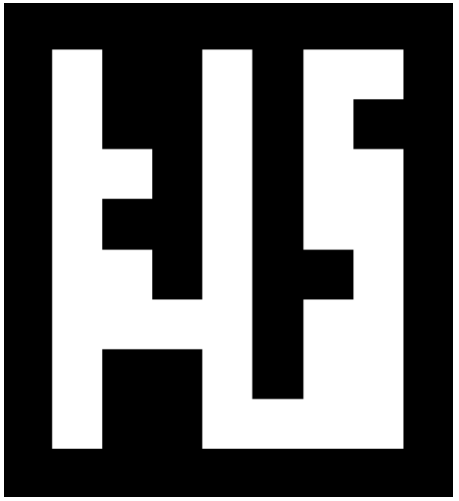
де u — функція оновлення стану

- Стан агента, як правило, набагато менший, ніж стан середовища



Стан агента

Повний стан середовища-лабіринту



Стан агента

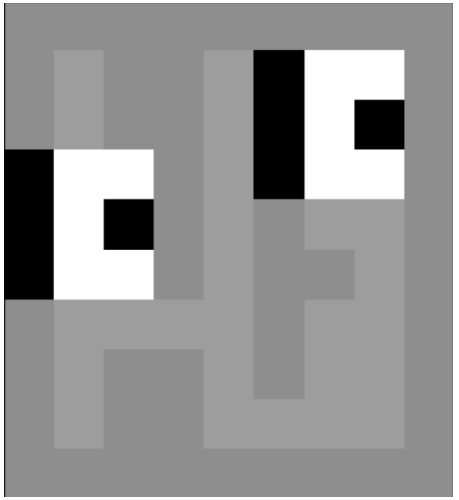
Потенційна дальність спостережень агента

Стан агента

Спостереження в іншому місці

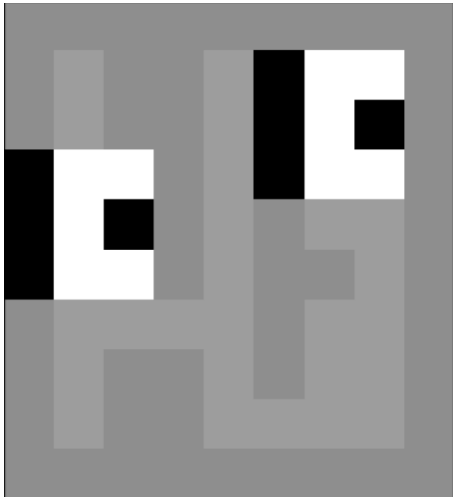
Стан агента

Два спостереження неможливо відрізнити



Стан агента

Ці два стани не є Марковськими



Частково оглядове середовище

- Маючи справу з частково оглядовим середовищем, агент може побудувати правильне представлення стану
- Приклади станів агента:
  - Останнє спостереження: $S_t = O_t$ (може бути недостатньо)
  - Уся історія: $S_t = H_t$ (може бути занадто великим)
  - Загальне оновлення: $S_t = u(S_{t-1}, A_{t-1}, R_t, O_t)$ (але як обрати/вивчити u?)
- Побудувати повнісю Марковський стан агента часто є неможливим

Компоненти агента

- Стан агента
- Стратегія (Policy)
- Q-функція
- Модель

Стратегія

- Стратегія визначає поведінку агента
- Стратегія — це план переходу між станом агента до дії
- Детерімінована стратегія: $A = \pi(S)$
- Стохастична стратегія: $\pi(A|S) = p(A|S)$

Компоненти агента

- Стан агента
- Стратегія
- Q-функція, функція цінності
- Модель

Функція цінності

- Фактична функція цінності — це очікувана загальна винагорода:

$$v_\pi(s) = \mathbb{E}\left[G_t \mid S_t = s, \pi\right] = \\ = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s, \pi\right]$$

- Тут введено фактор зменшення $\gamma \in [0, 1]$. Чим він менший, тим менше агент замислюється над вигодою від майбутніх своїх дій.
  - Вимірює важливість найближчих vs довгострокових винагород
- Цінність $v_\pi(s)$ залежить від стратегії
- Може використовуватися для оцінки бажаних станів
- Може використовуватися для вибору між діями

Функції цінності

- Загальна винагорода має рекурсивну форму: $G_t = R_{t+1} + \gamma G_{t+1}$
- Тому функція цінності може бути записана так:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s,\ A_t \sim \pi(s)\right] = \\ &= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t \sim \pi(s)\right] \end{aligned}$$

Тут $A_t = a \sim \pi(s)$ означає, що a вибрано стратегією π у стані s (π є детермінованою)

- Це рівняння відоме як рівняння Беллмана (Bellman 1957)
- Подібне рівняння можна отримати для оптимальної (= максимально можливої) цінності:
  - Не залежить від стратегії

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s,\ A_t = a]$$

Апроксимація функції цінності

- Агент постійно апроксимує значення функції цінності
- Для виконання апроксимації існують спеціальні алгоритми
- Завдяки правильній функції цінності агент може поводитися оптимально
- При правильних наближеннях агент може добре поводитися навіть у надзвичайно великих середовищах

Компоненти агента

- Стан агента
- Стратегія
- Q-функція, функція цінності
- Модель

Модель

- Модель передбачає поведінку середовища
- Передбачає наступний стан агента $\mathcal{P}$:

$$\boxed{\mathcal{P}(s, a, s') \approx p(S_{t+1} = s' \mid S_t = s,\ A_t = a)}$$

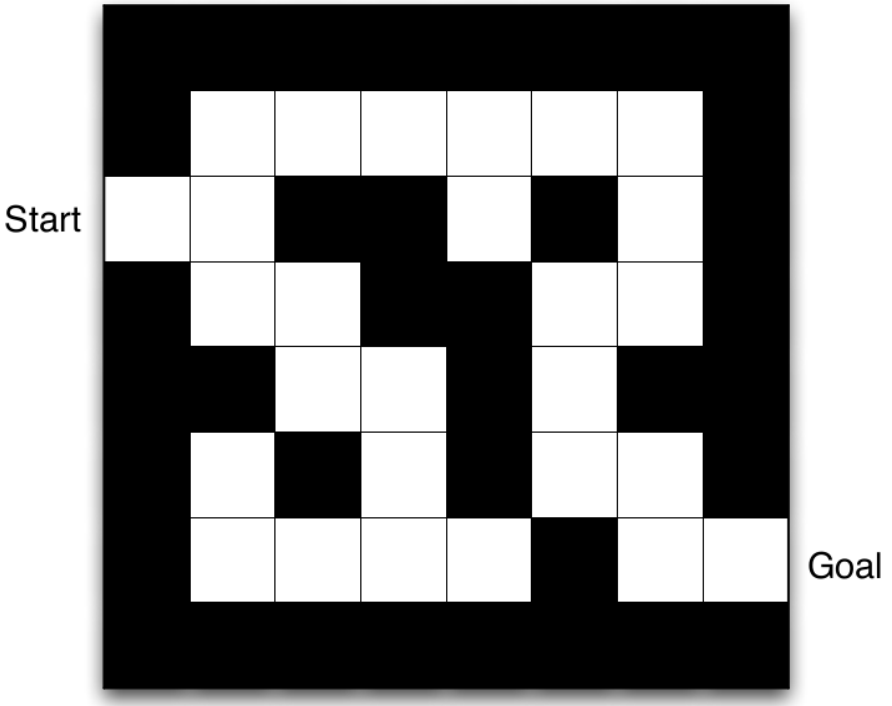
- Або передбачиає наступну (миттєву) винагороду $\mathcal{R}$:

$$\boxed{\mathcal{R}(s, a) \approx \mathbb{E}[R_{t+1} \mid S_t = s,\ A_t = a)]}$$

- Модель не відразу дає нам хорошу стратегію, тому приходиться агенту планувати свої дії
- Можуть також розглядатись стохастичні (генеративні) моделі

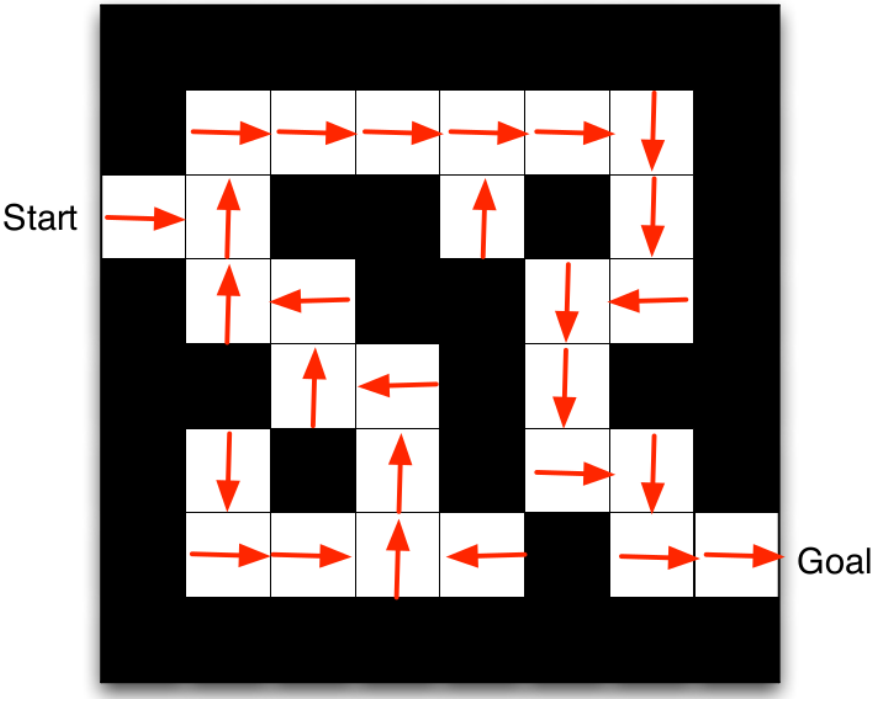
Приклад

Приклад з лабіринтом



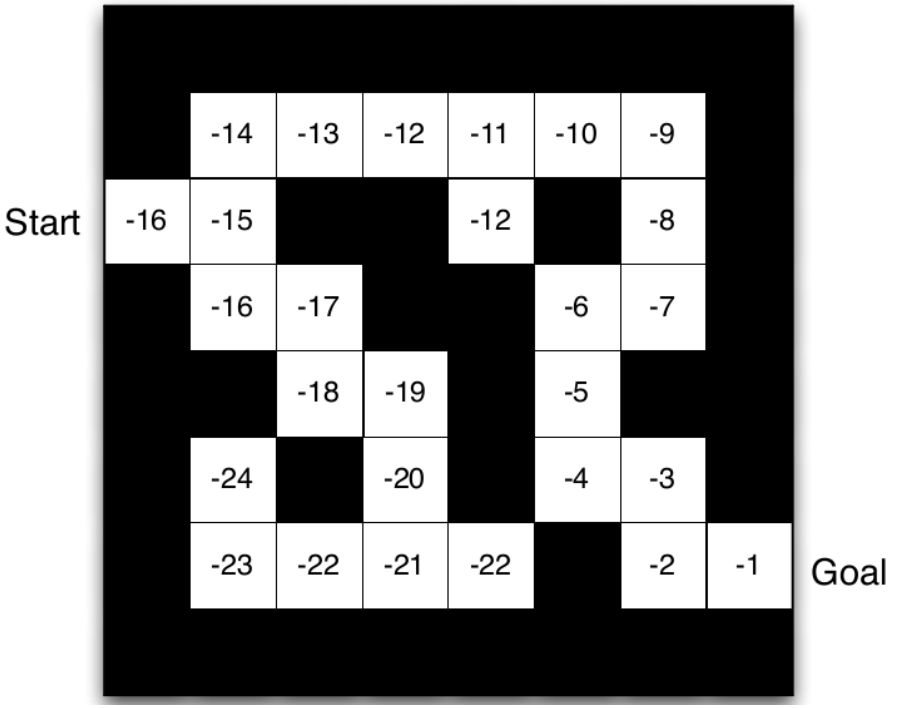
- Винагорода: -1 або 1 за крок
- Дії: N, E, S, W
- Стани: місцезнаходження агента

Приклад з лабіринтом: стратегія



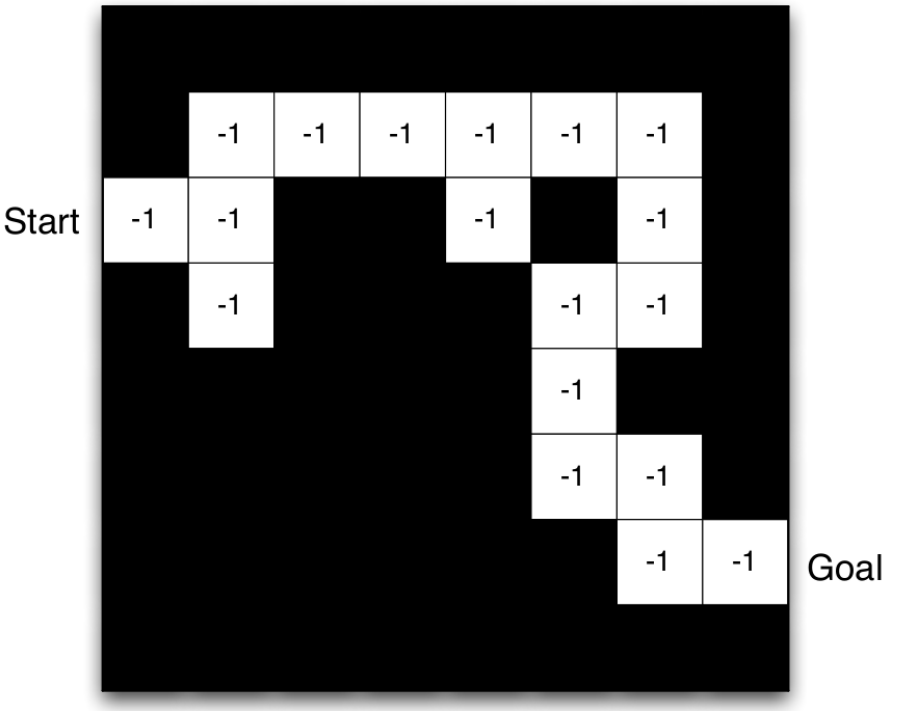
- Стрілки представляють стратегію агента $\pi(s)$ для кожного стану s

Приклад з лабіринтом: функція цінності



- Числа представляють значення $v_\pi(s)$ для кожного стану s

Приклад з лабіринтом: модель



- Даний шаблон являє собою модель часткового переходу $\mathcal{P}^{a}_{ss'}$
- Цифри позначають миттєву винагороду $\mathcal{R}^{a}_{ss'}$ (у цьому випадку однакова для усіх a та s')

Класифікація агентів

Класифікація агентів

- На основі цінності (Value Based)
  - Відсутня стратегія (неявна)
  - Функція цінності
- На основі стратегії (Policy Based)
  - Стратегія
    - Відсутня функція цінності
- Actor Critic
  - Стратегія
  - Функція цінності

Класифікація агентів

- Без моделі (Model Free)
  - Стратегія і/або функція цінності
    - Немає моделі
- На основі моделі (Model Based)
  - Стратегія і/або функція цінності (за бажанням)
  - Модель

Підзадачі RL

Передбачення та контроль

- Передбачення: оцінити майбутнє (для певної стратегії)
- Контроль: оптимізувати майбутнє (знайти найкращу стратегію)
- Передбачення та контроль можуть бути сильно пов'язані між собою:

$$\pi_*(s) = \operatorname*{argmax}_a v_\pi(s)$$

Навчання та планування

Два фундаментальні завдання навчання з підкріплення

- Навчання:
  - Середовище спочатку невідоме агенту
  - Агент взаємодіє з середовищем
- Планування:
  - Дається (або вивчається) модель середовища
  - Плани агента в цій моделі (без зовнішньої взаємодії)
  - У літературі використовується такі терміни: reasoning, pondering, thought, search, planning

Навчальні компоненти агента

- Усі компоненти є функціями:
  - Стратегія: $\pi : \mathcal{S} \to \mathcal{A}$
  - Функція цінності: $v : \mathcal{S} \to \mathbb{R}$
  - Модель: $m : \mathcal{S} \to \mathcal{S}$ та/або $r : \mathcal{S} \to \mathbb{R}$
  - Оновлення стану: $u : \mathcal{S} \times \mathcal{O} \to \mathcal{S}$
- Наприклад, ми можемо використовувати нейронні мережі та використовувати методи глибинного навчання для вивчення
- Глибинне навчання — важливий інструмент
- Глибинне навчання з підкріпленням — це багата та активна галузь досліджень

Приклад: Пересування

DeepMind - Emergence of Locomotion Behaviours in Rich Environments

Вступ до МППР

- Марківські процеси прийняття рішень формально описують середовище для навчання з підкріпленням
- Там, де середовище є повністю оглядовим
- Поточний стан агента повністю характеризує процес
- Майже всі задачі RL можна формалізувати як МППР
  - Оптимальне управління насамперед стосується безперервних МППР
  - Задачі в частково оглядовому середовищі можуть бути зведені до МППР

Властивість Маркова

Майбутнє процесу не залежить від минулого, а залежить лише від поточного стану

Стан S_t є Марківським тоді і тільки тоді

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \cdots, S_t]$$

- Це означає, що поточний стан агента містить все, що нам потрібно знати з його історії
- Як тільки стан стане відомим, історію можна буде відкинути
- Тобто, стан — це достатня статистика для майбутнього

Властивість Маркова

Щоб перевірити своє розуміння властивості Маркова, розглянемо декілька задач управління або задач прийняття рішень і подивимось, які з них мають властивість Маркова:

- Водіння автомобіля
- Рішення інвестувати в акції чи ні
- Вибір лікування пацієнта
- Діагностика хвороби пацієнта
- Передбачити, яка команда виграє у футбольному матчі
- Пошук найкоротшого маршруту (найкоротшого) до певного пункту призначення
- Наведення прицілу гармати на постріл у далеку мішень

Матриця зміни стану **(state transition matrix)**

Ймовірність переходу між Марківськими станами $s \rightarrow s'$, визначається так:

$$\boxed{\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]}$$

Матриця зміни стану $\mathcal{P}$ визначає ймовірності переходу між усіма станами s у всі можливі стани s':

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix},$$

де кожен рядок матриці у сумі дорівнює 1.

Марківський процес

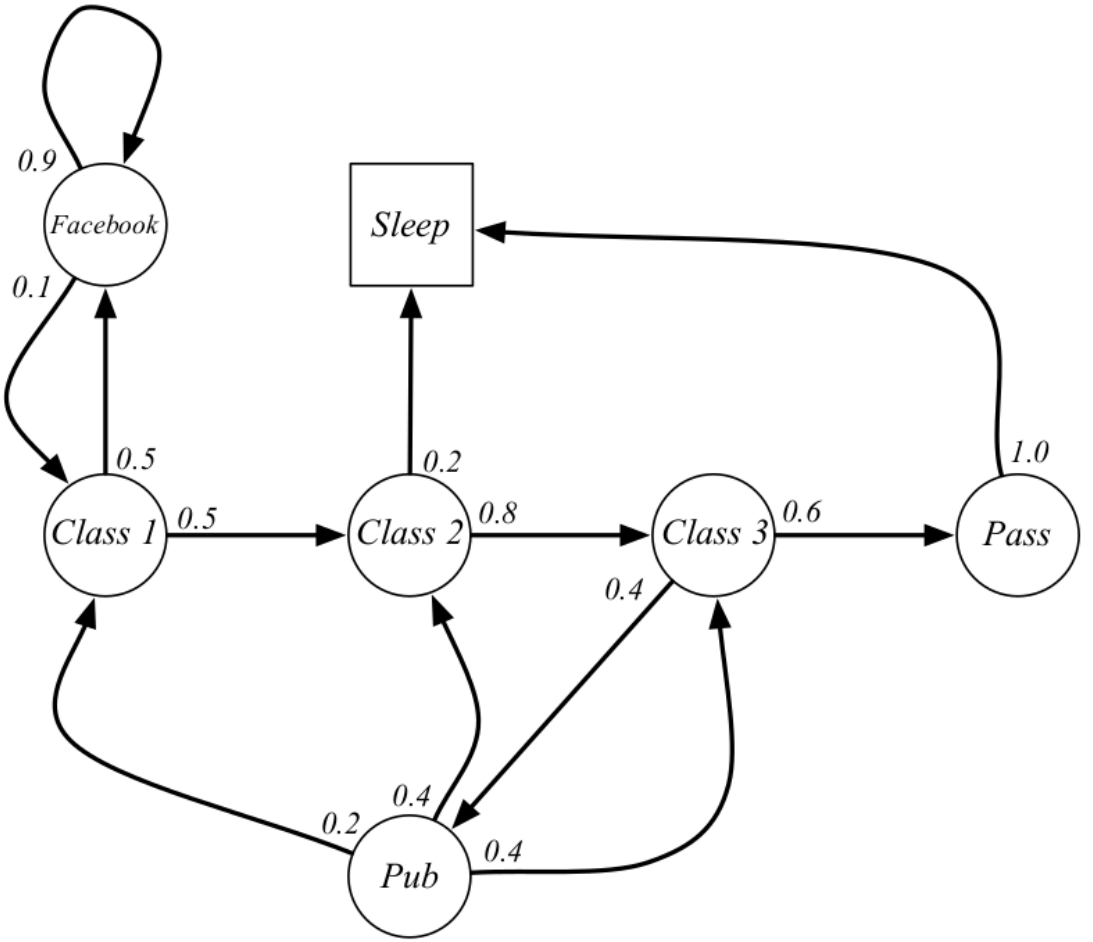
Марківський процес — це випадковий процес у якого відсутня пам'ять, тобто послідовність випадкових станів $S_1, S_2, \cdots$, які володіють властивістю Маркова.

Марківський процес (або ланцюг Маркова) — це кортеж $\langle \mathcal{S}, \mathcal{P} \rangle$:

- $\mathcal{S}$ — скінченна множина станів
- $\mathcal{P}$ — матриця зміни стану: $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$

Приклад

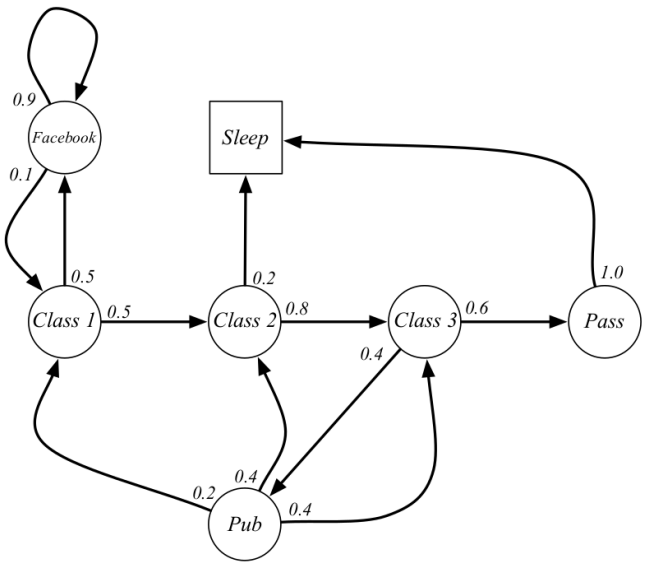
Студентський ланцюг Маркова



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Студентський ланцюг Маркова



Початковий епізод починається з $S_1 = C_1$

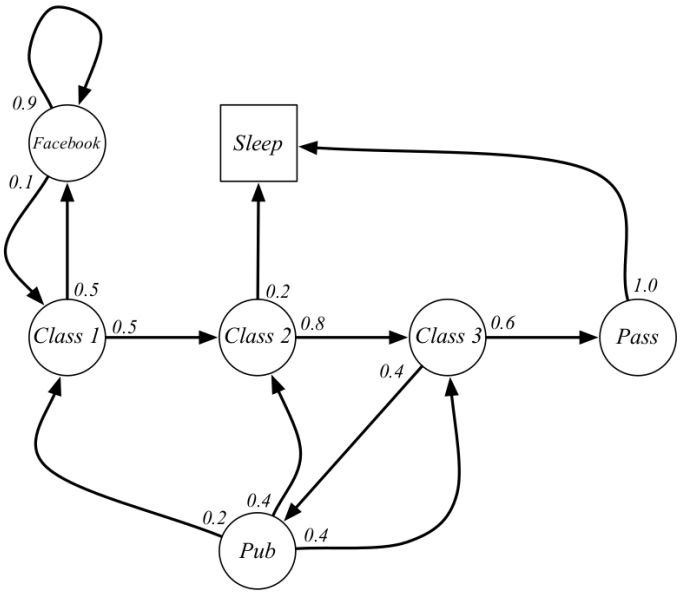
$$S_1, S_2, \cdots, S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Студентський ланцюг Маркова: матриця зміни стану



$$\mathcal{P} = \begin{array}{c|ccccccc} & C1 & C2 & C3 & Pass & Pub & FB & Sleep \\ C1 & & 0.5 & & & & 0.5 & \\ C2 & & & 0.8 & & & & 0.2 \\ C3 & & & & 0.6 & & 0.4 & \\ Pass & & & & & & & 1.0 \\ Pub & 0.2 & 0.4 & 0.4 & & & & \\ FB & 0.1 & & & & & 0.9 & \\ Sleep & & & & & & & 1 \end{array}$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

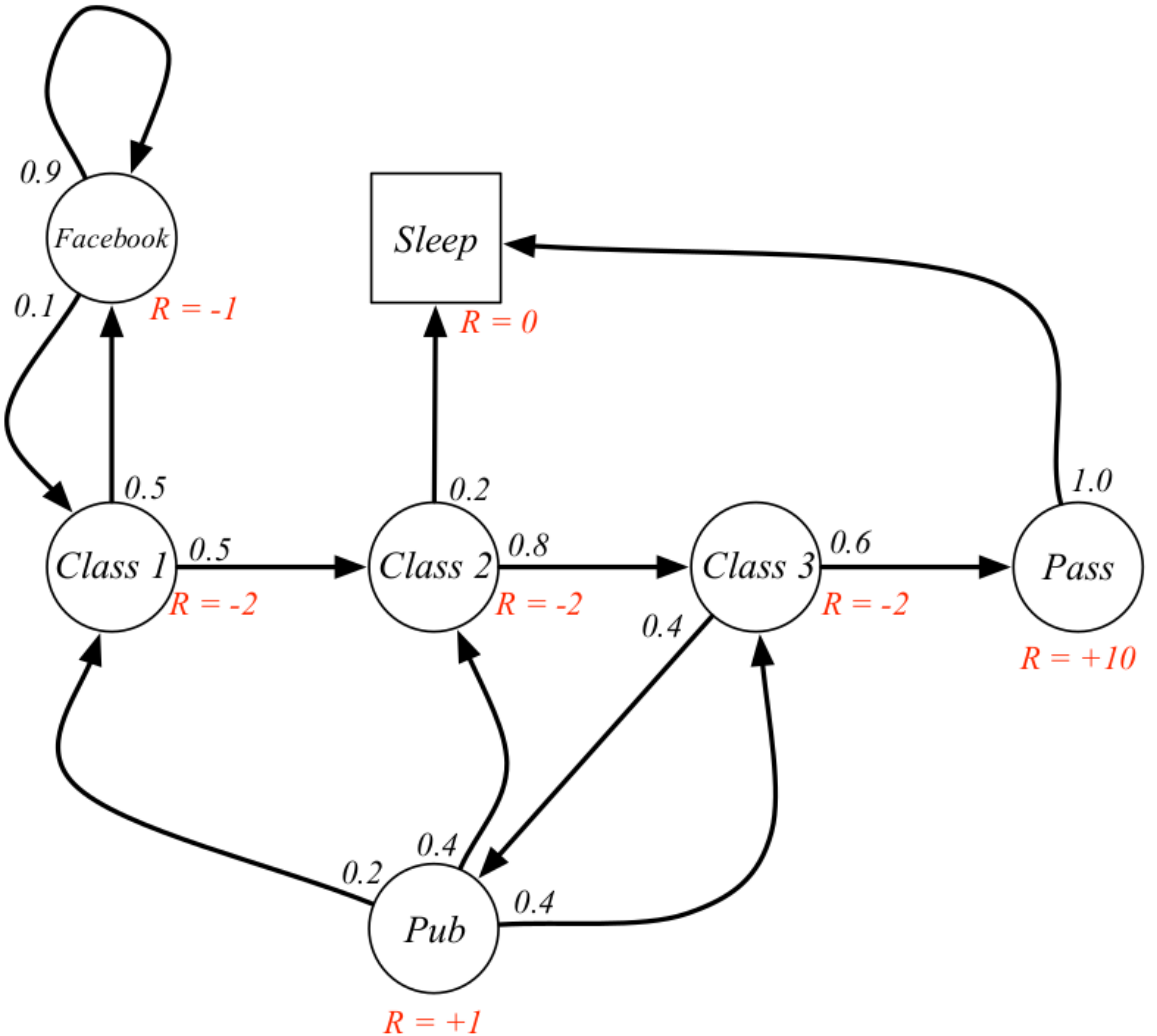
Марківські процеси винагороди

Марківський процес винагороди — ланцюг Маркова з винагородою.

Марківський процес винагороди — це кортеж $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{S}$ — скінченна множина станів
- $\mathcal{P}$ — матриця зміни стану: $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$
- $\mathcal{R}$ — функція винагороди: $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- γ — коефіцієнт зменшення (знецінювання), $\gamma \in [0, 1]$

Приклад: МПВ


0.9
Facebook
0.1
R = -1
Sleep
R = 0
0.5
0.2
1.0
Class 1
0.5
Class 2
0.8
Class 3
0.6
Pass
R = -2
R = -2
R = -2
0.4
R = +10
0.4
0.2
0.4
Pub
R = +1

—

Загальна винагорода

Загальна винагорода — сумарна винагорода отримана агентом з моменту часу t з урахування знецінювання:

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

- Коефіцієнт знецінювання $\gamma \in [0, 1]$ показує на цінність майбутніх винагород
- Значення винагороди R, отримане після $k + 1$ кроків: $\gamma^k R$
- Чим менший коефіцієнт знецінювання, тим менше агент замислюється над вигодою від майбутніх своїх дій.

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Яка роль знецінювання?

- Дозволяє уникнути нескінченної загальної винагороди в циклічних марківських процесах
- Невизначеність щодо майбутнього може бути представлена не повністю
- Якщо винагорода є фінансовою, негайні винагороди можуть бути більш цікавими, ніж відстрочені винагороди
- Поведінка тварин/людини демонструє перевагу миттєвій винагороді
- Іноді можна використовувати марківський процес винагороди без знецінювання(тобто $\gamma = 1$), наприклад якщо всі послідовності закінчуються.

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Функція цінності

Функція цінності $v(s)$ показує довгострокову цінність перебування агента у стані s

Функція цінності $v(s)$ марківського процесу винагороди — середнє значення загальної винагороди починаючи від стану s

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

Приклад: МПВ загальна винагорода

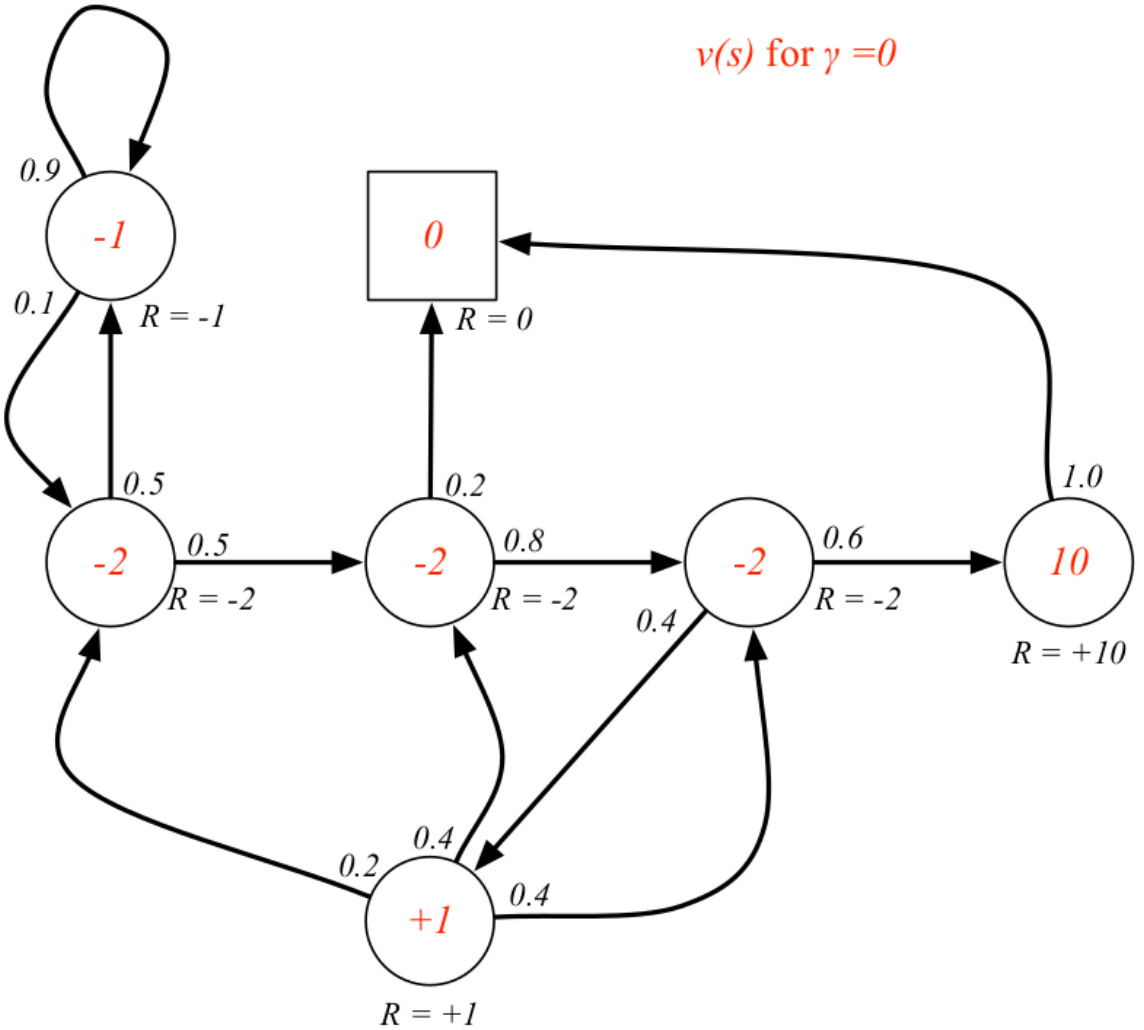
Приклади загальної винагороди для раніше розглянутого прикладу.
Покачок з $S_1 = C_1$ з $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + \cdots + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	$=$	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	$=$	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	$=$	-3.41
C1 FB FB C1 C2 C3 Pub C1 … FB FB FB C1 C2 C3 Pub C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	$=$	-3.20

—

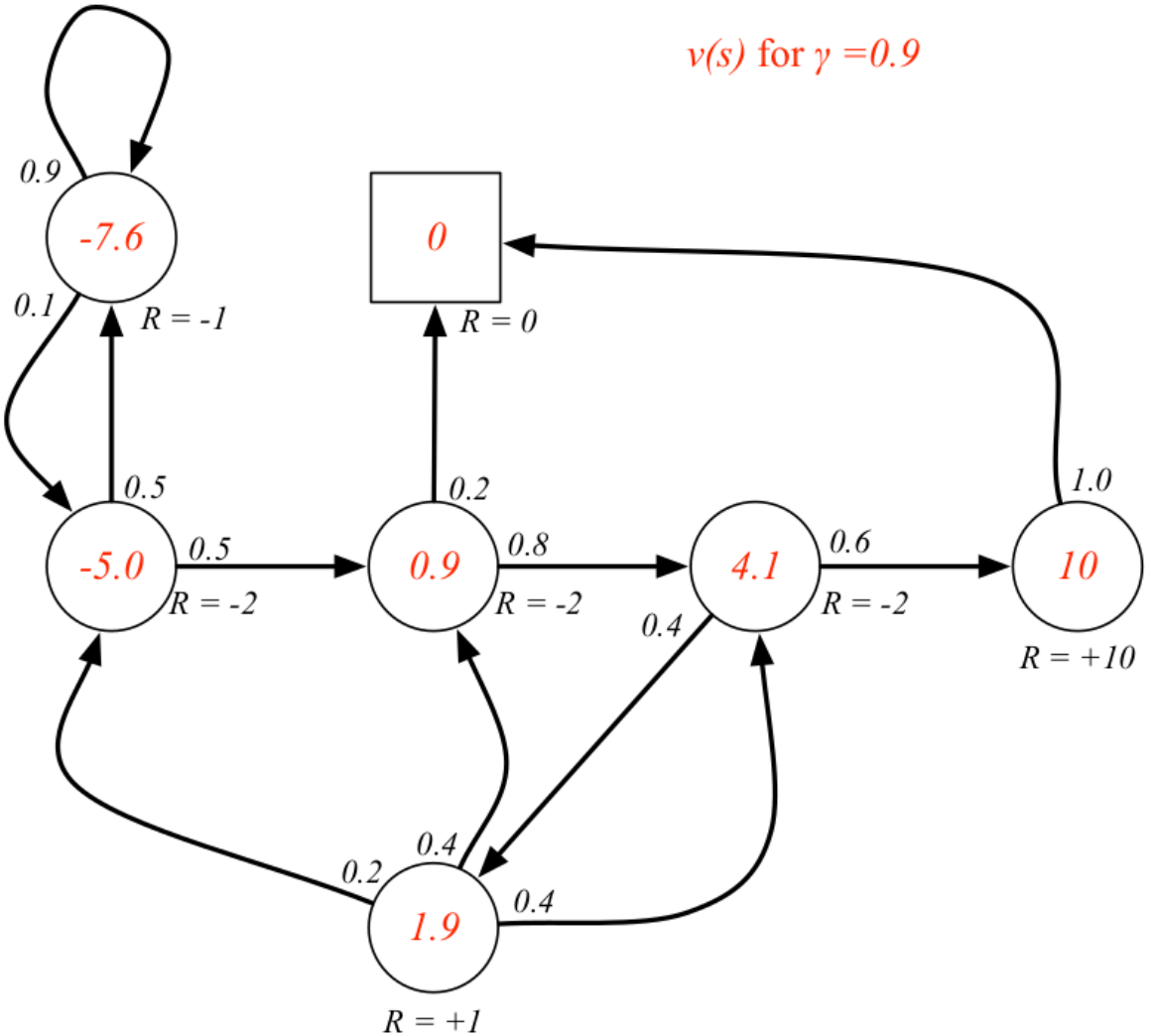
Приклад: Функція цінності



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

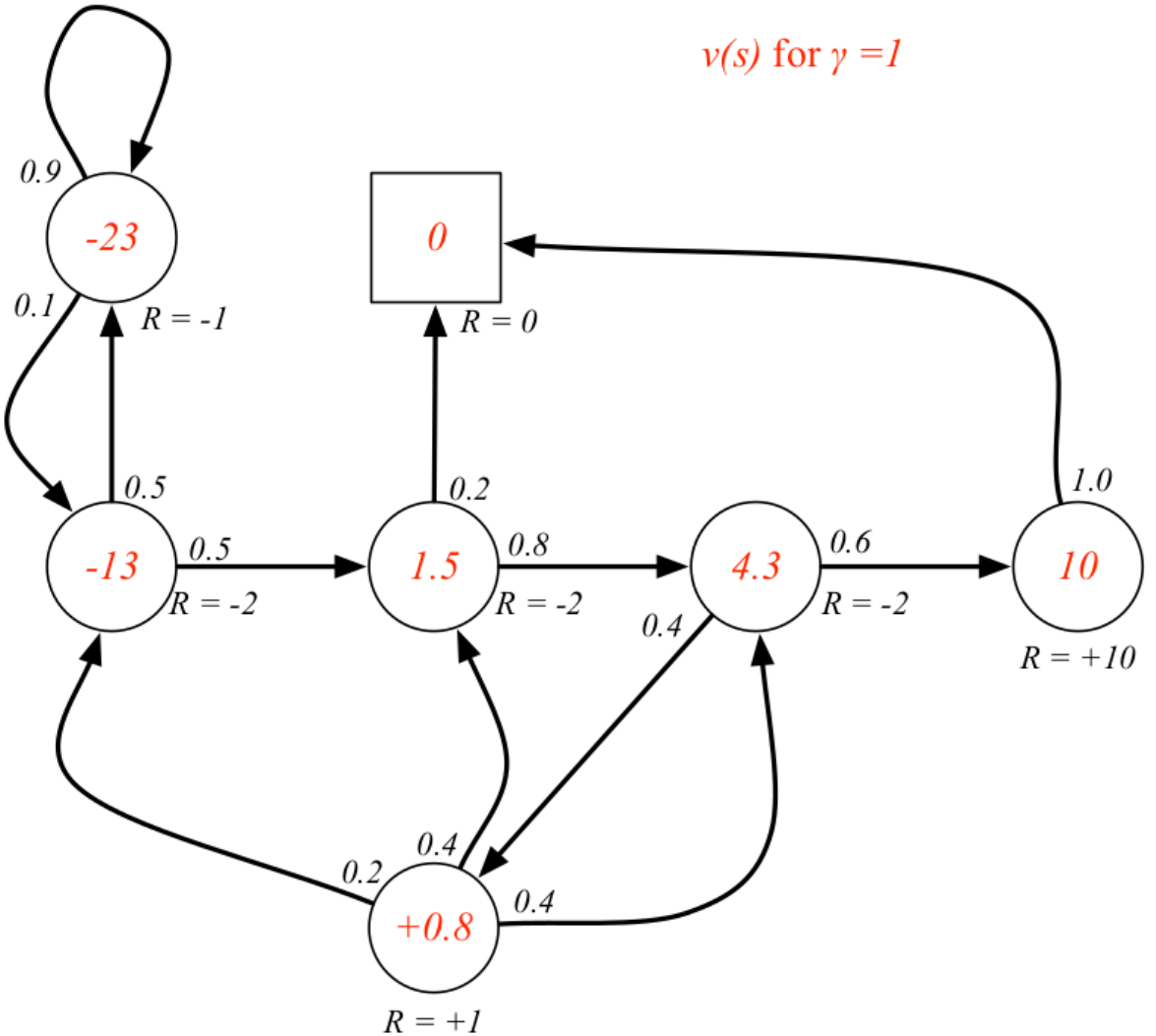
Приклад: Функція цінності



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Приклад: Функція цінності



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Рівняння Беллмана для МПВ

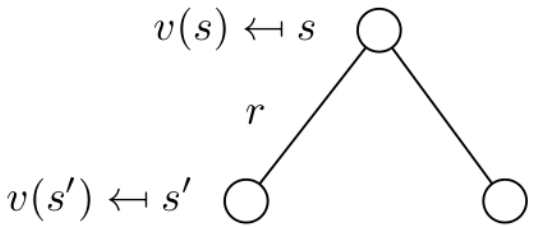
$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] = \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] = \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \cdots\right) \mid S_t = s\right] = \\
&= \mathbb{E}\left[R_{t+1}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

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Рівняння Беллмана: усереднення

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

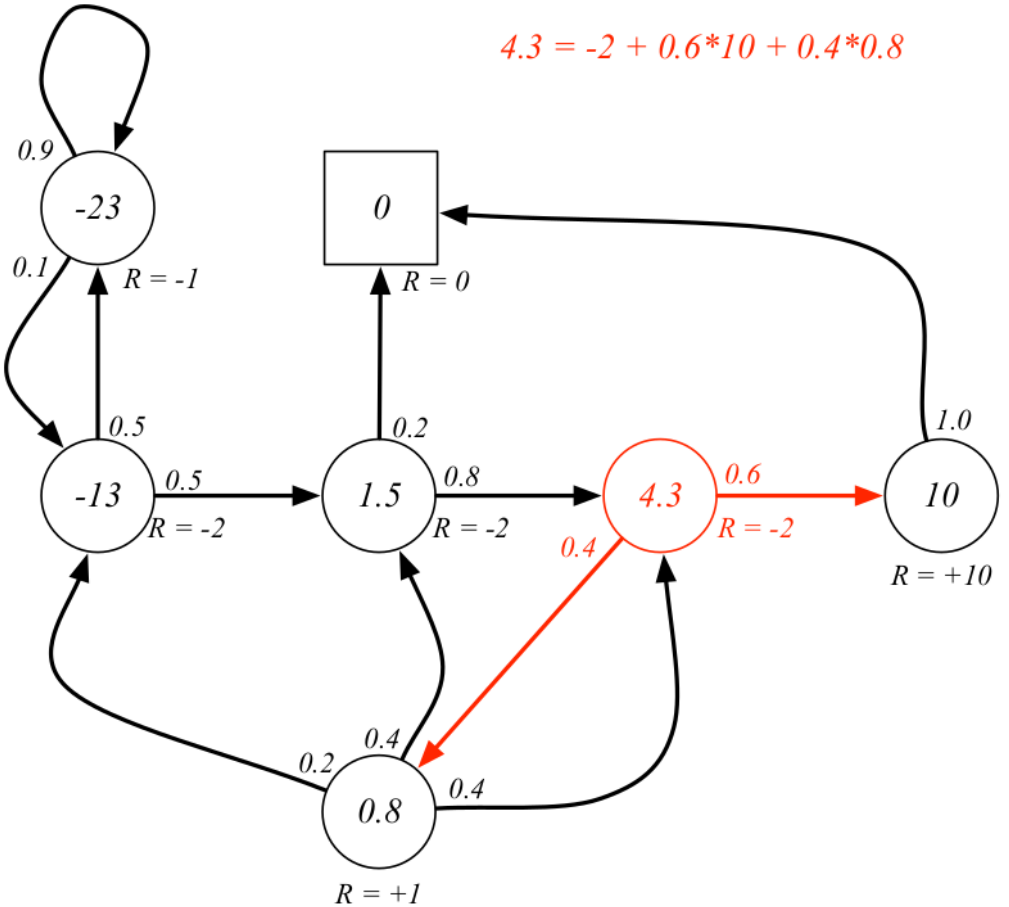


$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Приклад усереднення рівняння Беллмана



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Матрична форма рівняння Беллмана

Рівняння Беллмана можна виразити у матричній формі:

$$v = \mathcal{R} + \gamma \mathcal{P} v,$$

де v — вектор-стовпець з одним записом для кожного стану.

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Розв'язок рівняння Беллмана

- Рівняння Беллмана є лінійним рівнянням
- Його можна розв'язати точних методів (алгебраїчним способом):

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
v(1 - \gamma \mathcal{P}) &= \mathcal{R} \\
v &= (1 - \gamma \mathcal{P})^{-1} \mathcal{R}
\end{aligned}
$$

- Обчислювальна складність становить $O(n^3)$ для n станів
- Алгебраїчний спосіб розв'язку можливий лише для малих МПВ ($n \sim 10^4$)
- Існує багато ітераційних методів для великих МПВ ($n \sim 10^7$)
  - Динамічне програмування
  - Оцінка Монте-Карло
  - Навчання часових різниць

Марківські процеси прийняття рішень (МППР)

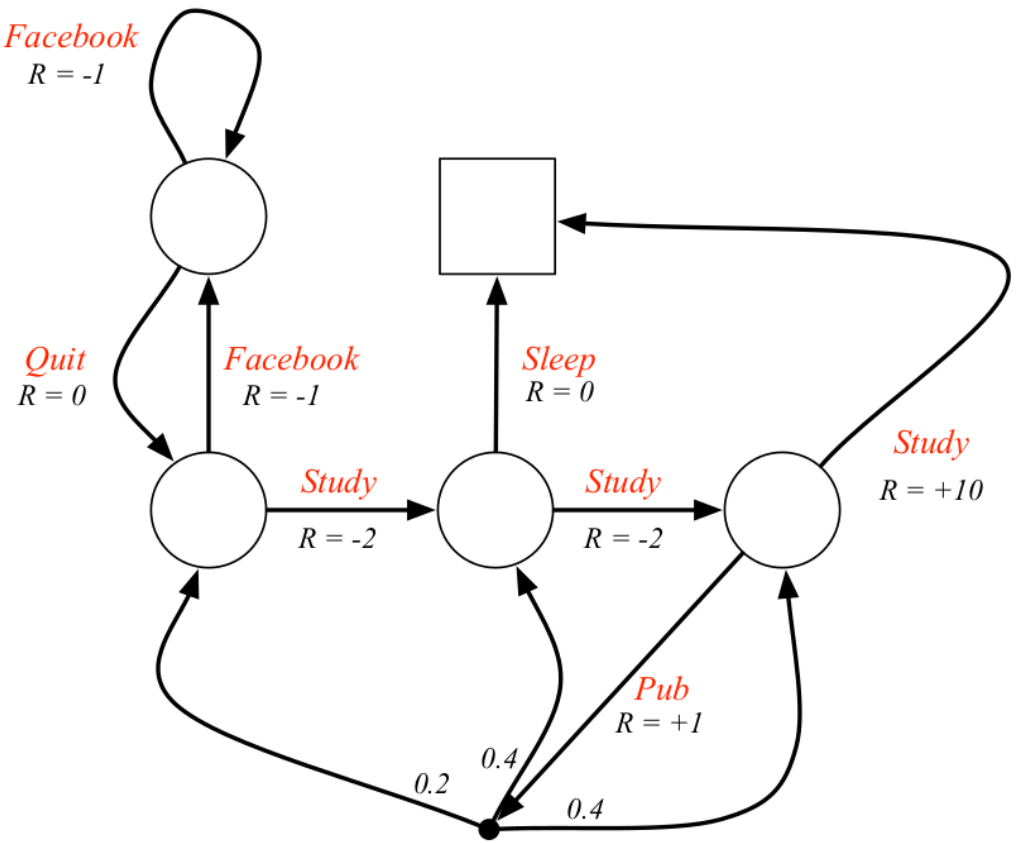
МППР

Марківський процес прийняття рішень (МППР) — марківський процес винагороди з рішеннями (прийнятими діями). Це середовище, у якому всі стани є марківськими.

МППР — це кортеж $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{S}$ — скінченна множина станів
- $\mathcal{A}$ — скінченна множина дій
- $\mathcal{P}$ — матриця зміни стану: $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ — функція винагороди: $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ — коефіцієнт зменшення (знецінювання), $\gamma \in [0, 1]$

Приклад: МППР



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Стратегія

Стратегія

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

- Стратегія повністю визначає поведінку агента
- Стратегія у МППР залежить від поточного стану, а не від історії
- Тобто, стратегія є стаціонарною (не залежить від часу):
  $A_t \sim \pi(\cdot|S_t), \forall t > 0$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

- Для заданого МППР $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ та стратегії π
- Послідовність станів $S_1, S_2, \cdots$ &dmash; марківський процес $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- Послідовність зі станів та винагород $S_1, R_2, S_2, \cdots$ &dmash; марківський процес винагород $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$

$$\mathcal{P}^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{ss'}$$

$$\mathcal{R}^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^a_s$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Функція цінності

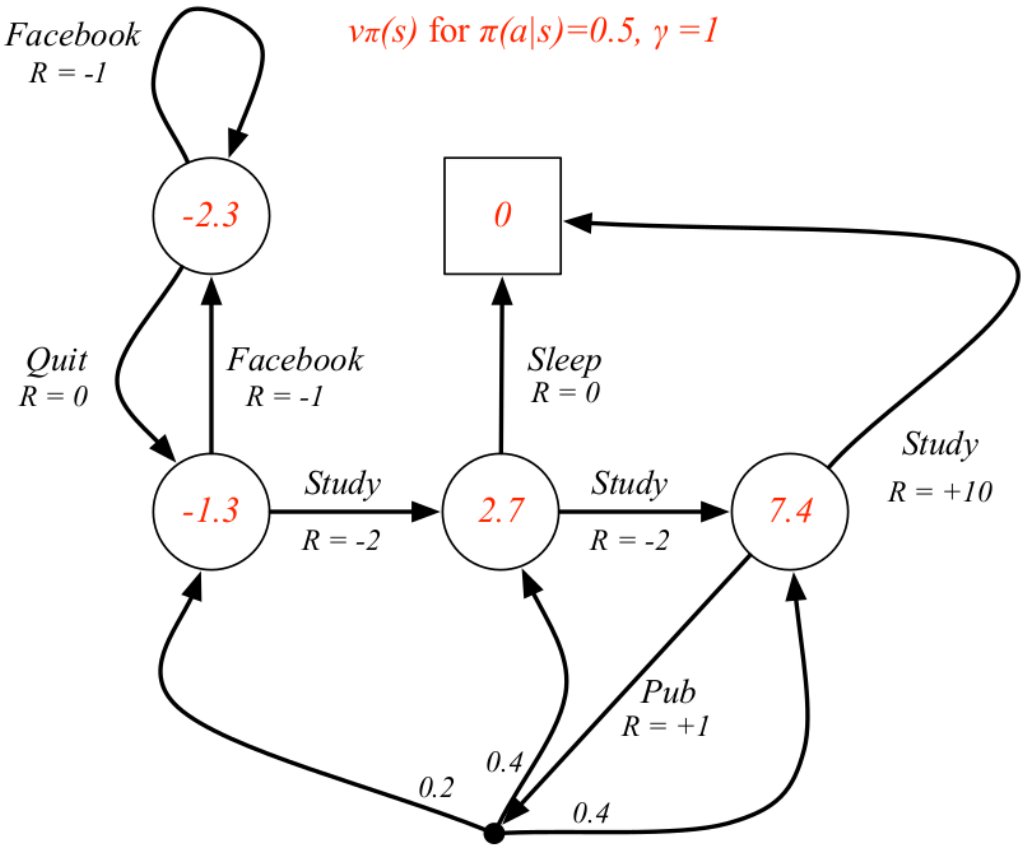
Функція цінності $v_\pi(s)$ МППР — середнє значення загальної винагороди починаючи від стану s при дотриманні заданої стратегії π

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

Q-функція:

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

Приклад функції цінності



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Рівняння Беллмана для МППР

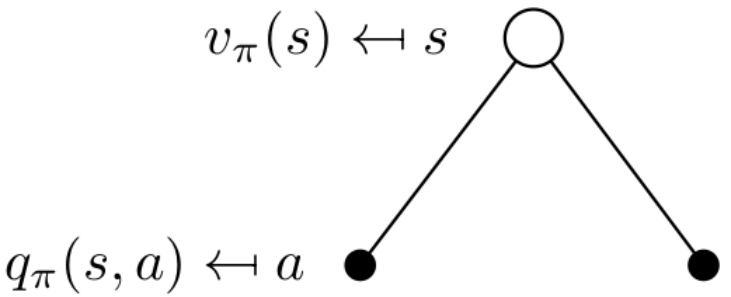
$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}\left[G_t \mid S_t = s, \pi\right] = \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s, \pi\right] = \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \cdots\right) \mid S_t = s, \pi\right] = \\
&= \mathbb{E}\left[R_{t+1}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

$$
\begin{aligned}
q_\pi(s, a) &= \mathbb{E}\left[G_t \mid S_t = s, A_t = a, \pi\right] = \\
&= \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a, \pi\right]
\end{aligned}
$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Рівняння Беллмана v_π

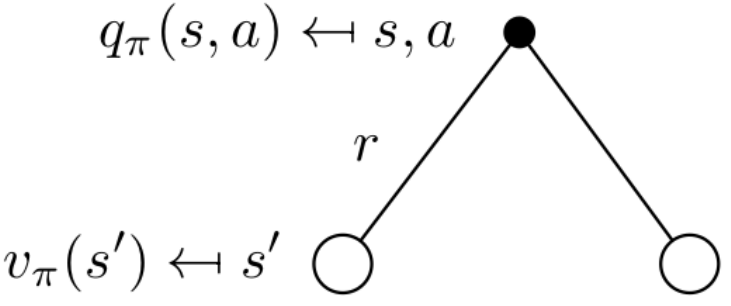


$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Рівняння Беллмана q_π

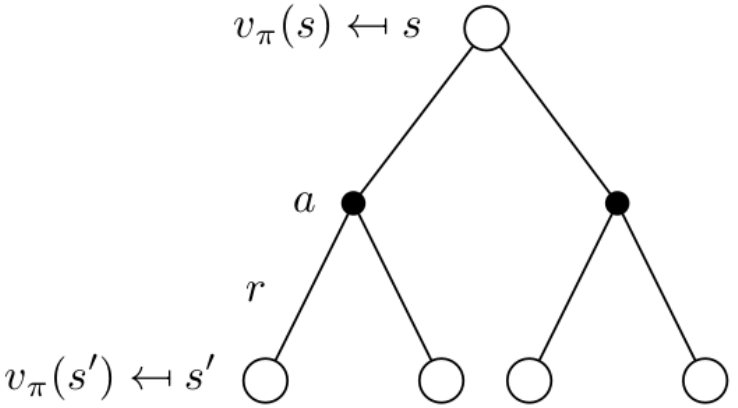


$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Рівняння Беллмана — 2 v_π

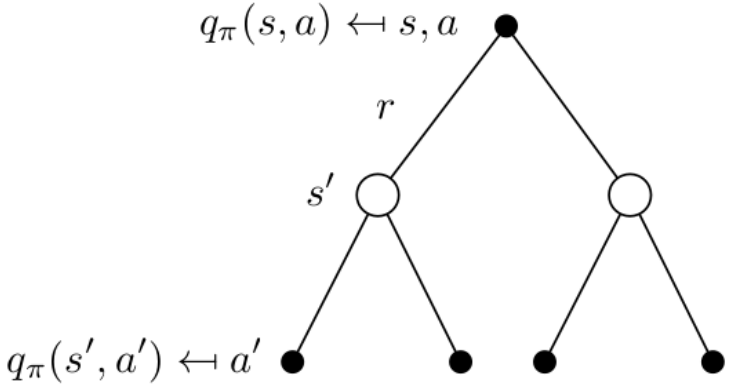


$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Рівняння Беллмана — 2 q_π

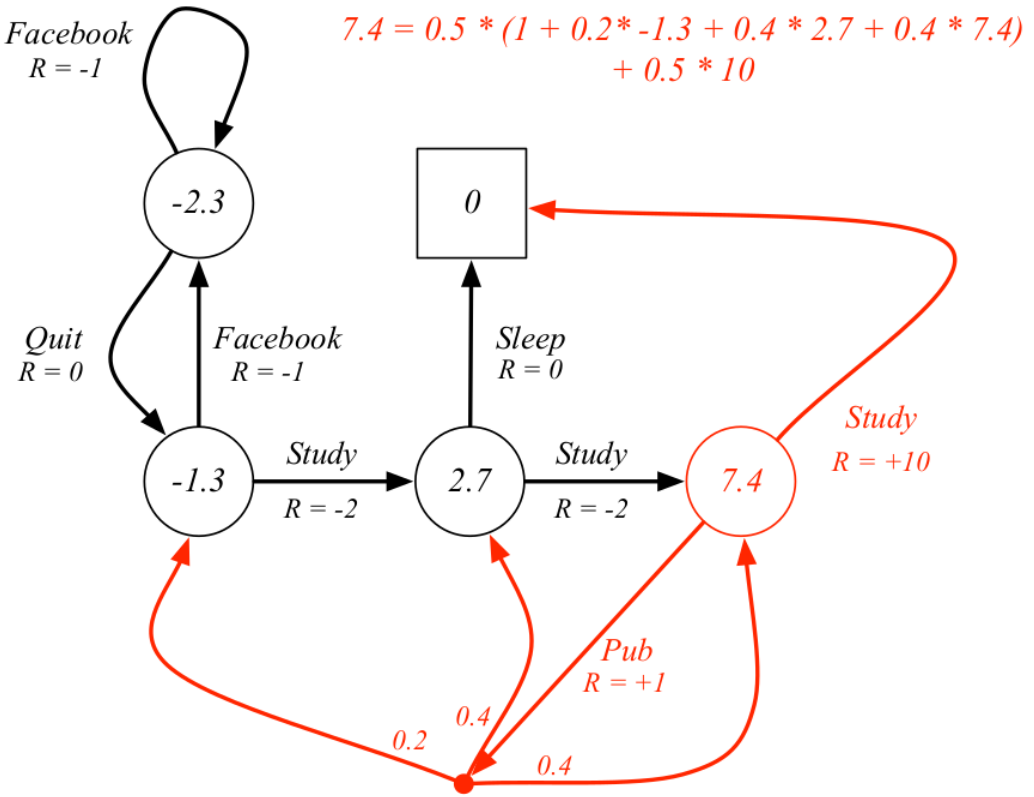


$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Приклад рівняння Беллмана для МППР



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Матрична форма рівняння Беллмана для МППР

Рівняння Беллмана можна виразити у матричній формі:

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi,$$

де v_π — вектор-стовпець з одним записом для кожного стану.

$$\begin{bmatrix} v_\pi(1) \\ \vdots \\ v_\pi(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1^\pi \\ \vdots \\ \mathcal{R}_n^\pi \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11}^\pi & \cdots & \mathcal{P}_{1n}^\pi \\ \vdots & & \\ \mathcal{P}_{n1}^\pi & \cdots & \mathcal{P}_{nn}^\pi \end{bmatrix} \begin{bmatrix} v_\pi(1) \\ \vdots \\ v_\pi(n) \end{bmatrix}$$

Точний розв'язок:

$$v_\pi = (1 - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Оптимальна функція цінності

Оптимальна функція цінності $v_*(s)$ — це максимальне значення функції серед усіх стратегій:

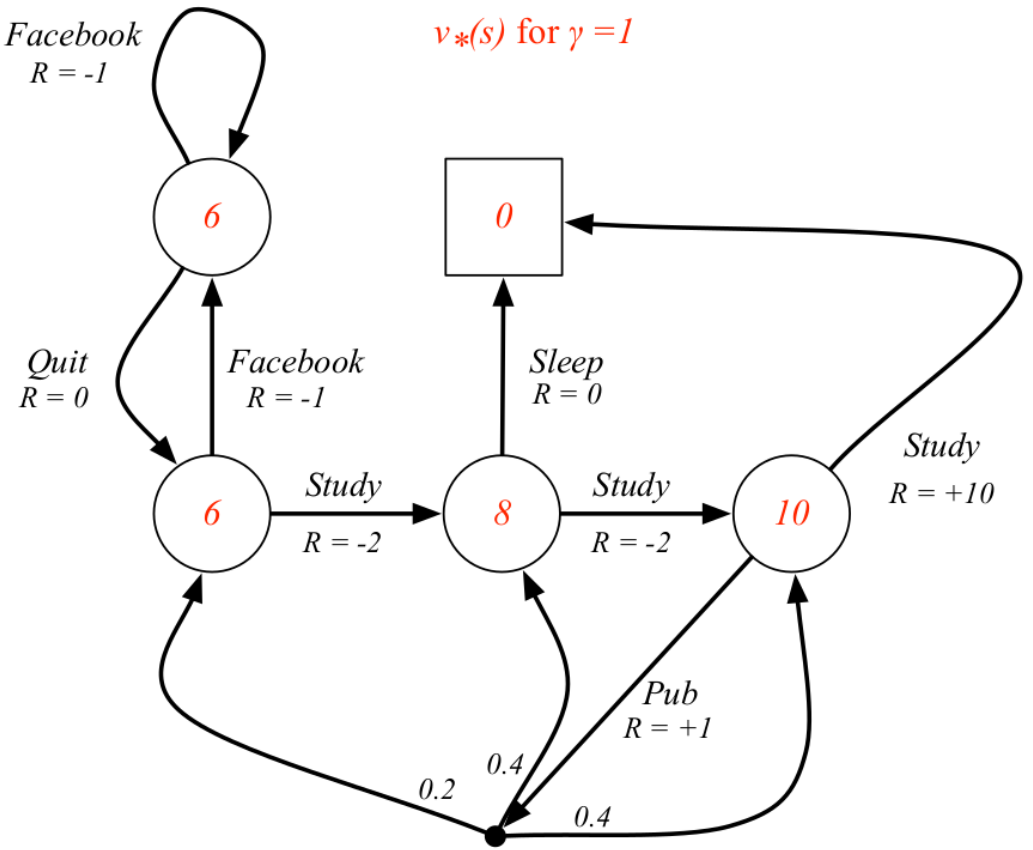
$$v_*(s) = \max_\pi v_\pi(s)$$

Оптимальна Q-функція $q_*(s, a)$ — це максимальне значення функції серед усіх стратегій:

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

- Оптимальна функція цінності вказує на найкращу з можливих продуктивностей у МППР.
- МППР є "вирішиним", коли ми знаємо оптимальне значення функції цінності.

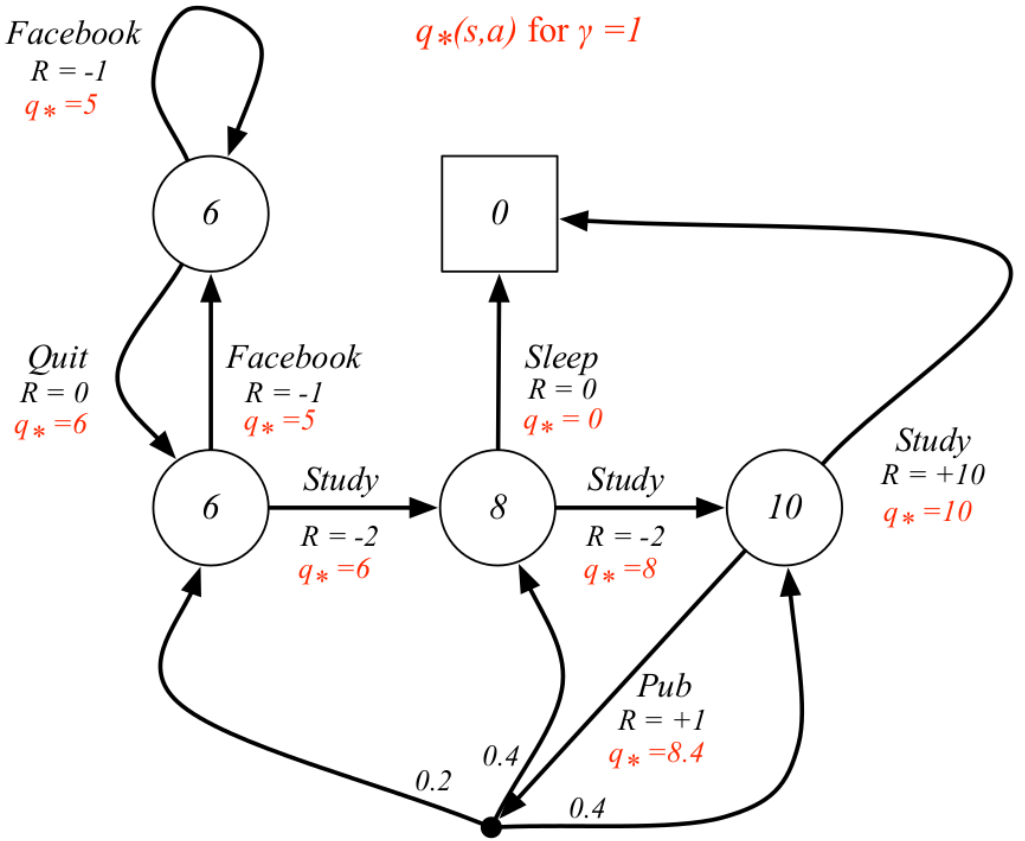
Приклад: оптимум $v_*(s)$



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Приклад: оптимум $q_*(s, a)$



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

Оптимальна стратегія

Упорядкування стратегій:

$\pi > \pi'$ якщо $v_\pi(s) > v'_\pi(s), \forall s$

Теорема. Для будь-якого МППР

- існує оптимальна стратегія π_*, яка краща або не гірша за інші стратегії: $\pi_* > \pi, \forall \pi$
- усі оптимальні стратегії досягають оптимальної функції цінності: $v_{\pi_*}(s) = v_*(s)$
- усі оптимальні стратегії досягають оптимального значення Q-функції: $q_{\pi_*}(s, a) = q_*(s, a)$

—

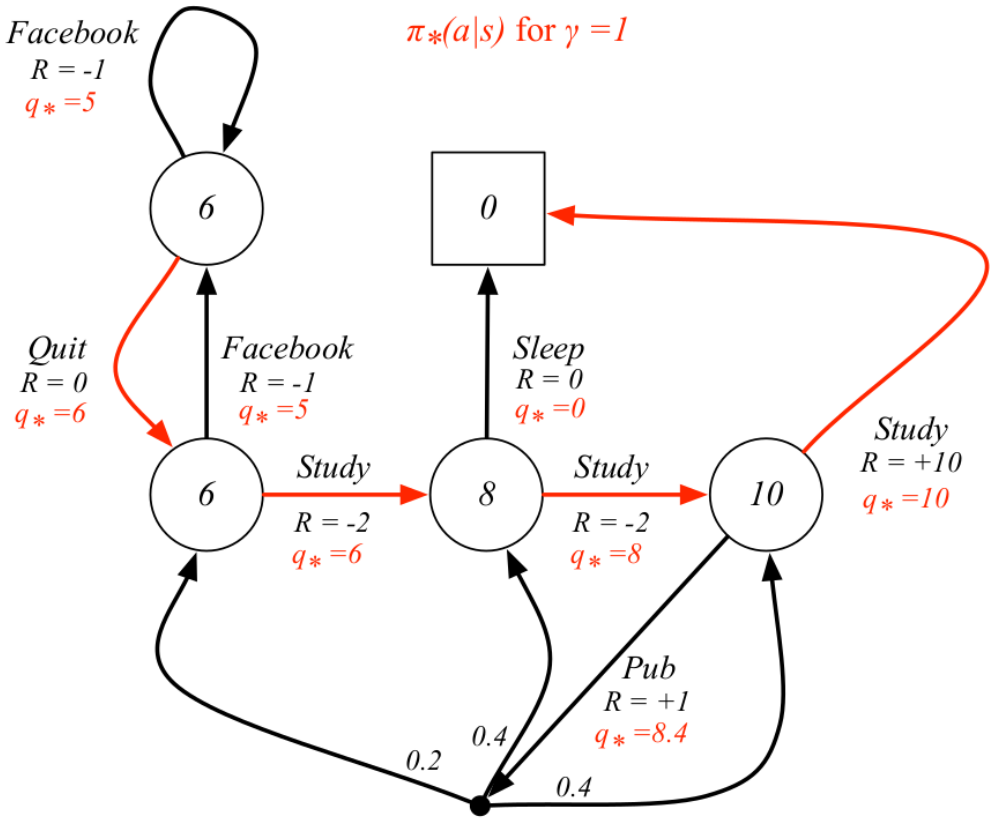
Пошук оптимальної стратегії

Оптимальна стратегія може бути знайдена, шляхом знаходження максимуму $q_*(s, a)$

$$\pi_*(a|s) = \begin{cases} 1, \text{ if } a = arg\max_{a \in \mathcal{A}} q_*(s, a) \\ 0, \text{ else} \end{cases}$$

- Для будь-якого МППР завжди існує детермінована оптимальна стратегія
- Якщо відомо $q_*(s, a)$, ми одразу маємо оптимальну стратегію

Приклад: оптимальна стратегія для МППР



—

Джерело слайду: Lecture 2: Markov Decision Processes [video], David Silver.

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.

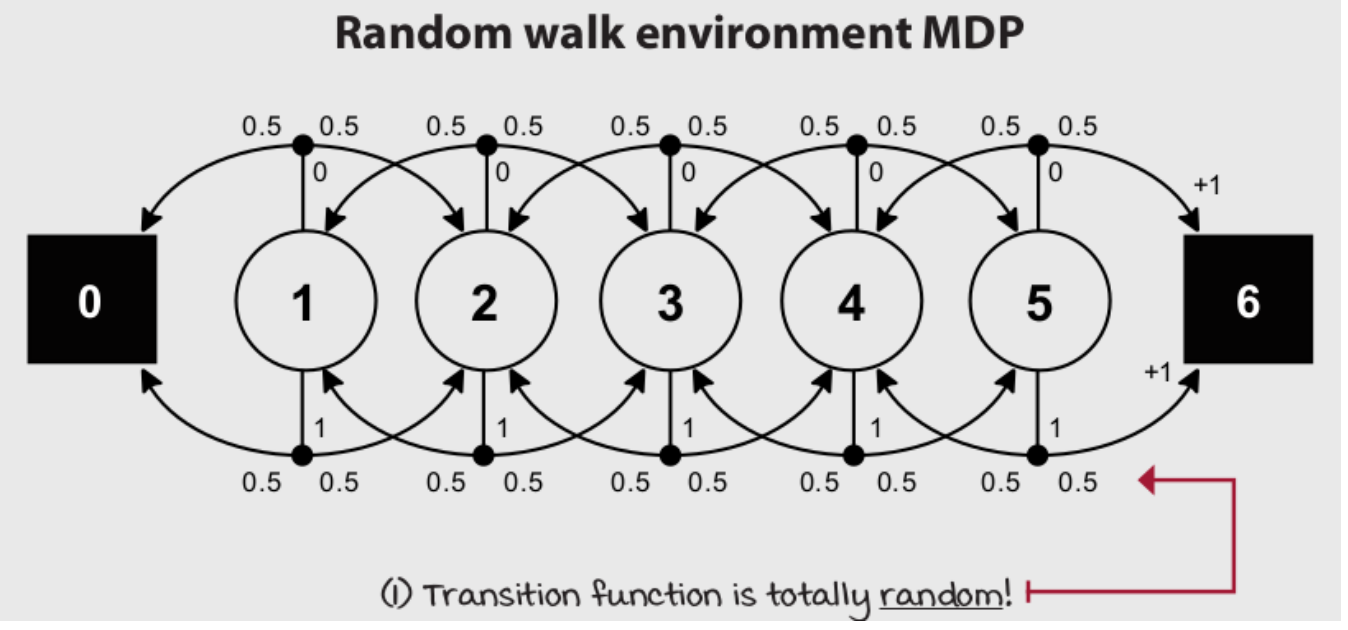
Загальна винагорода (return): сумарна винагорода отримана агентом з моменту часу t з урахування знецінювання γ. Розраховується від будь-якого стану агента і зазвичай триває до кінця епізоду. Тобто, коли досягається стан завершення (terminal state), обчислення припиняється.

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

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

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

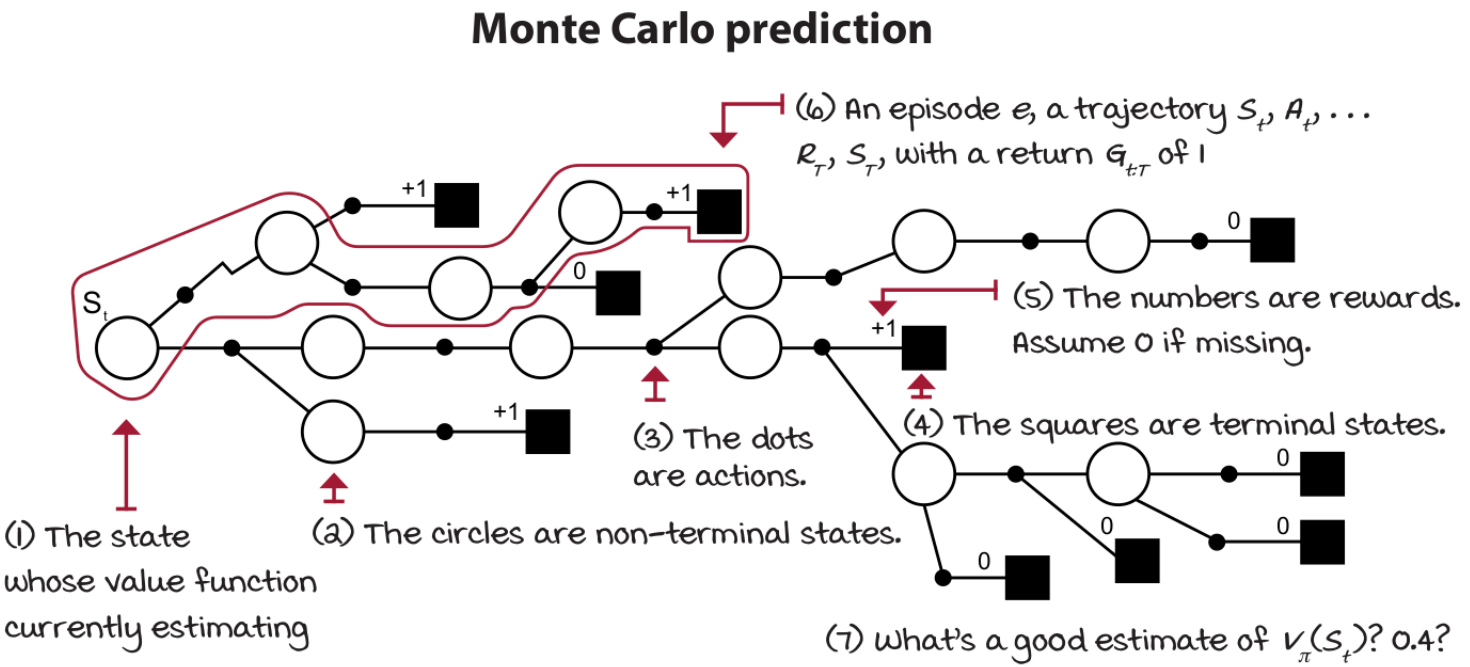
- Загальна винагорода:

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

- 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
    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
```

(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.

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.

I Speak Python

Monte Carlo prediction 1/2

```
def mc_prediction(pi,
                  env,
                  gamma=1.0,
                  init_alpha=0.5,
                  min_alpha=0.01,
                  alpha_decay_ratio=0.3,
                  n_episodes=500,
                  max_steps=100,
                  first_visit=True):
```

(1) The mc_prediction function works for both first- and every-visit MC. The hyperparameters you see here are standard. Remember, the discount factor, gamma, depends on the environment.

(2) For the learning rate, alpha, I'm using a decaying value from init_alpha of 0.5 down to min_alpha of 0.01, decaying within the first 30% (alpha_decay_ratio of 0.3) of the 500 total max_episodes. We already discussed max_steps on the previous function, so I'm passing the argument around. And first_visit toggles between FVMC and EVMC.

```
    nS = env.observation_space.n
    discounts = np.logspace(
            0, max_steps, num=max_steps,
            base=gamma, endpoint=False)
    alphas = decay_schedule(
            init_alpha, min_alpha,
            alpha_decay_ratio, n_episodes)
```

(3) This is cool. I'm calculating all possible discounts at once. This logspace function for a gamma of 0.99 and a max_step of 100 returns a 100 number vector: [1, 0.99, 0.9801, . . ., 0.3697].

(4) Here I'm calculating all of the alphas!

(5) Here we're initializing variables we'll use inside the main loop: the current estimate of the state-value function V, and a per-episode copy of V for offline analysis.

```
    V = np.zeros(nS)
    V_track = np.zeros((n_episodes, nS))
```

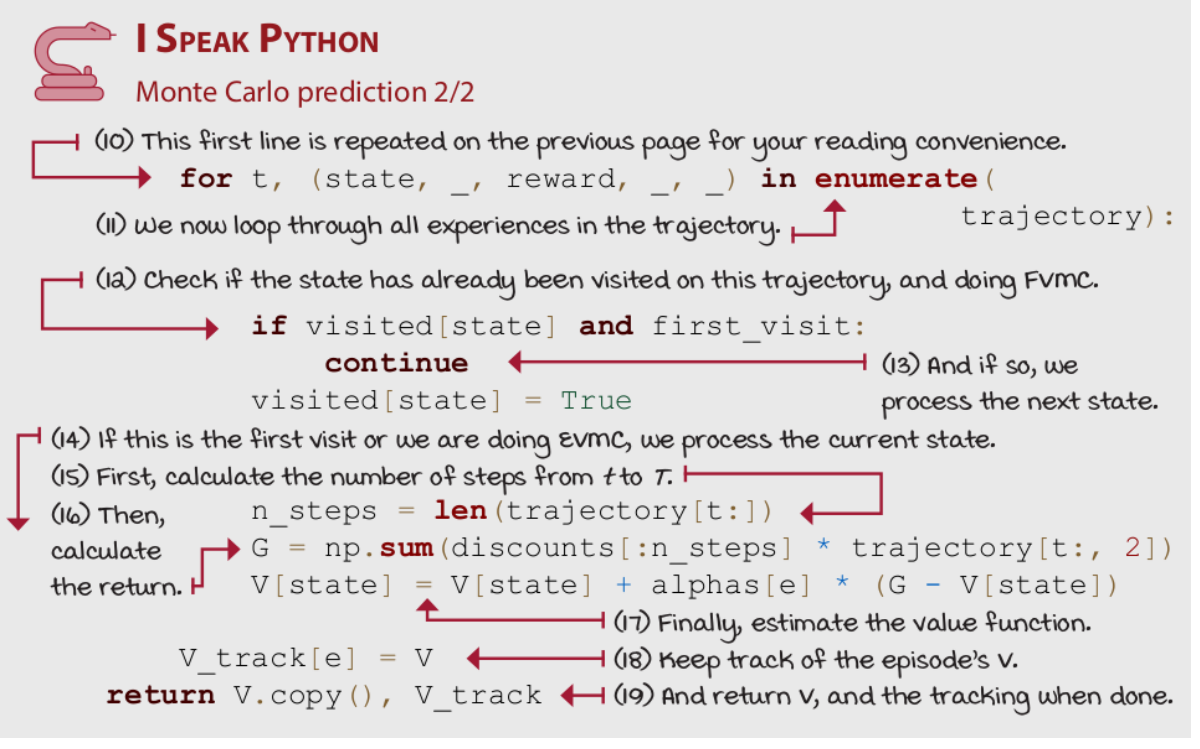
(6) We loop for every episode. Note that we're using 'tqdm' here. This package prints a progress bar, and it's useful for impatient people like me. You may not need it (unless you're also impatient).

```
    for e in tqdm(range(n_episodes), leave=False):
        trajectory = generate_trajectory(
                pi, env, max_steps)
        visited = np.zeros(nS, dtype=np.bool)
        for t, (state, _, reward, _, _) in enumerate(
                trajectory):
```

(7) Generate a full trajectory.

(8) Initialize a visits check bool vector.

(9) This last line is repeated on the next page for your reading convenience.



—

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

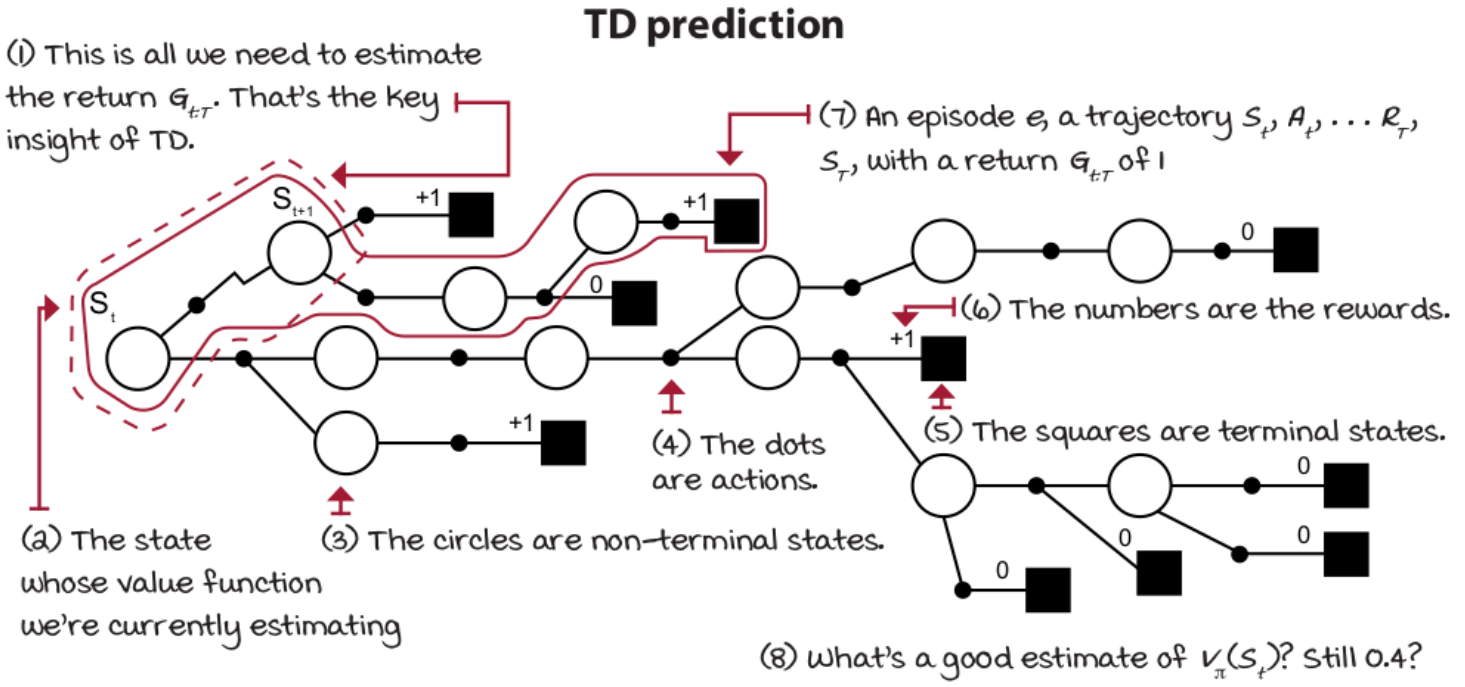
True value function: Refers to the exact and perfectly accurate value function, as if given by an oracle. The true value function is the value function agents estimate through samples. If we had the true value function, we could easily estimate returns.

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:

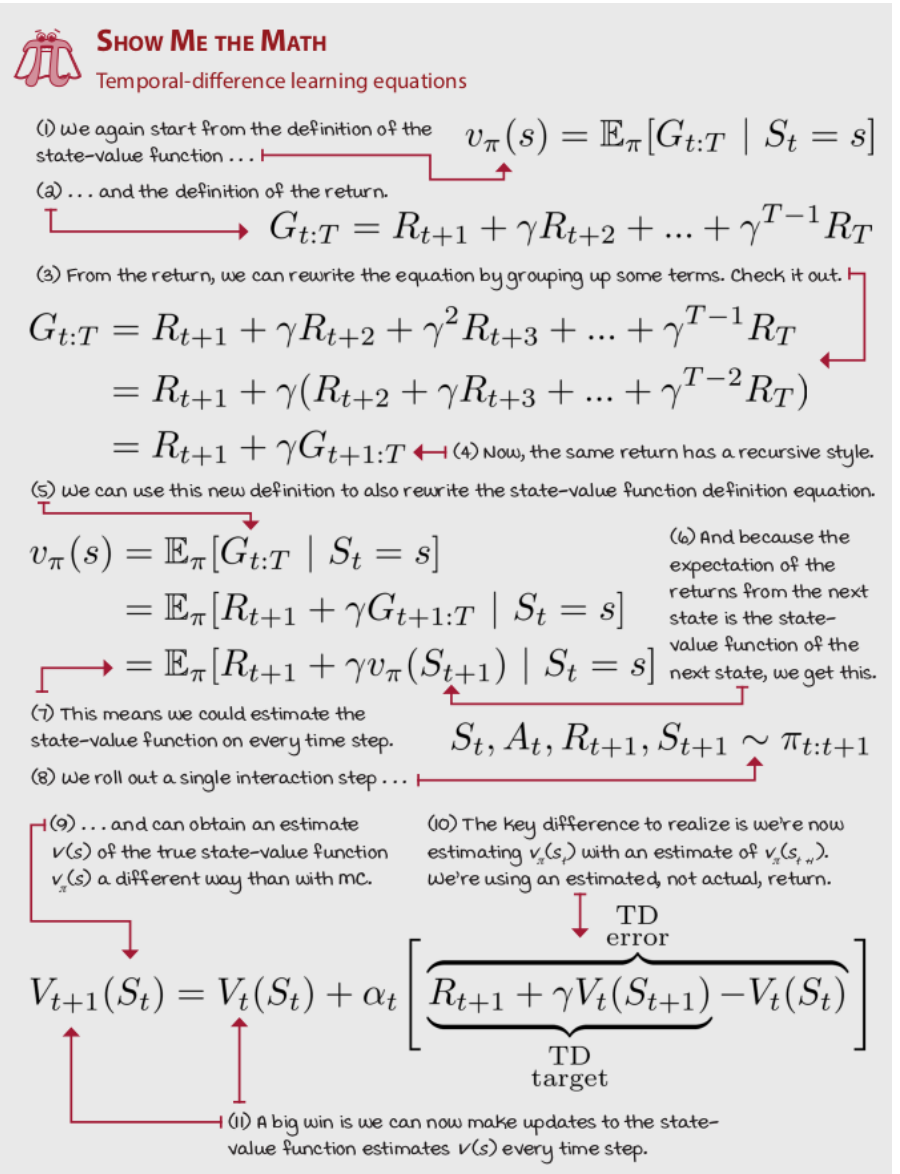
$$R_{t+1} + \gamma V_t(S_{t+1})$$

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

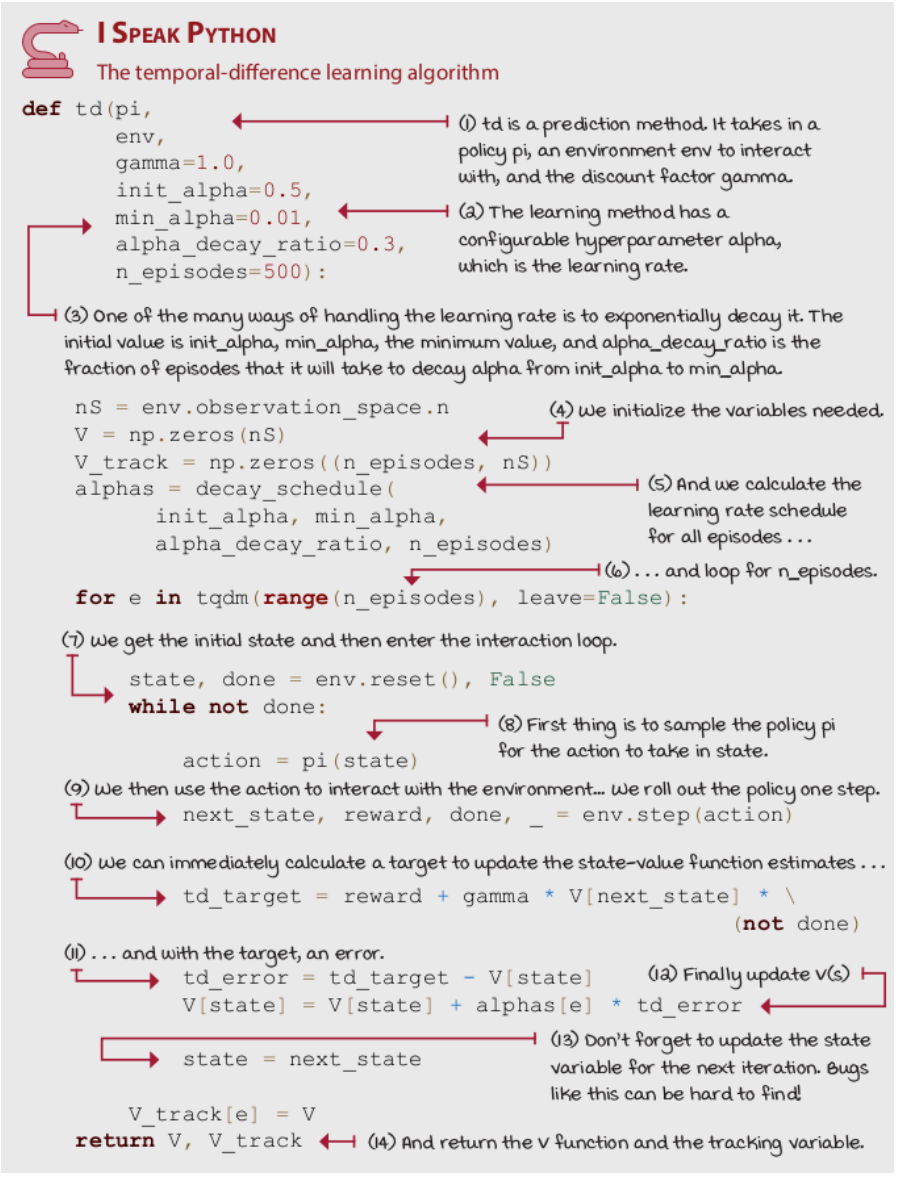
Оскільки TD використовує один крок фактичного значення загальної винагороди R_{t+1}, він усе ще працюватиме добре. Цей сигнал винагороди R_{t+1} поступово «вносить реальність» в оцінки.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

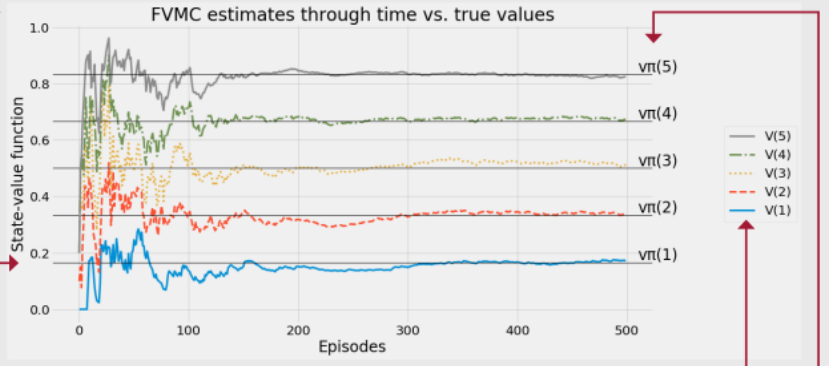

I Speak Python
The temporal-difference learning algorithm
def td(pi,
env,
gamma=1.0,
init_alpha=0.5,
min_alpha=0.01,
alpha_decay_ratio=0.3,
n_episodes=500):
(1) td is a prediction method. It takes in a policy pi, an environment env to interact with, and the discount factor gamma.
(2) The learning method has a configurable hyperparameter alpha, which is the learning rate.
(3) One of the many ways of handling the learning rate is to exponentially decay it. The initial value is init_alpha, min_alpha, the minimum value, and alpha_decay_ratio is the fraction of episodes that it will take to decay alpha from init_alpha to min_alpha.
nS = env.observation_space.n
V = np.zeros(nS)
V_track = np.zeros((n_episodes, nS))
alphas = decay_schedule(
init_alpha, min_alpha,
alpha_decay_ratio, n_episodes)
(4) We initialize the variables needed.
(5) And we calculate the learning rate schedule for all episodes . . .
for e in tqdm(range(n_episodes), leave=False):
(6) . . . and loop for n_episodes.
(7) We get the initial state and then enter the interaction loop.
state, done = env.reset(), False
while not done:
action = pi(state)
(8) First thing is to sample the policy pi for the action to take in state.
(9) We then use the action to interact with the environment... We roll out the policy one step.
next_state, reward, done, _ = env.step(action)
(10) We can immediately calculate a target to update the state-value function estimates . . .
td_target = reward + gamma * V[next_state] * \
(not done)
(11) . . . and with the target, an error.
td_error = td_target - V[state]
V[state] = V[state] + alphas[e] * td_error
(12) Finally update V(s)
state = next_state
(13) Don't forget to update the state variable for the next iteration. Bugs like this can be hard to find!
V_track[e] = V
return V, V_track
(14) And return the V function and the tracking variable.

—

Tally it Up

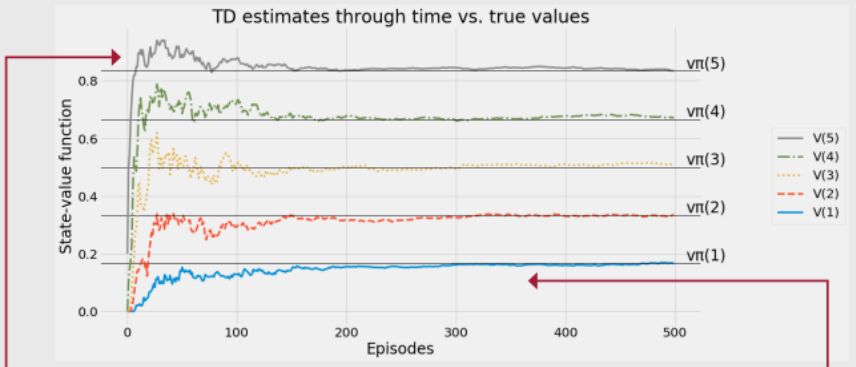
MC and TD both nearly converge to the true state-value function

(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.

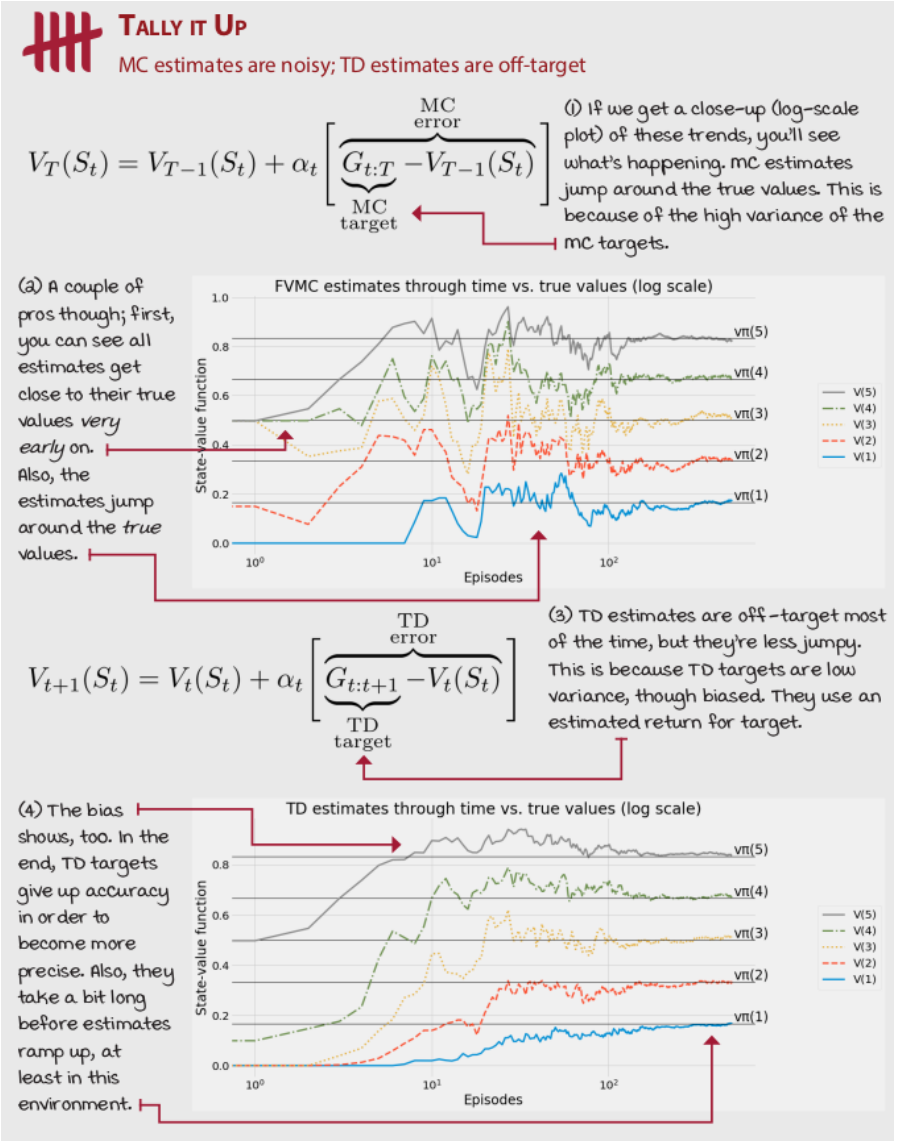
(3) Now, see the difference trends of these algorithms. FVMC running estimates are very noisy; they jump back and forth around the true values.



(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.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

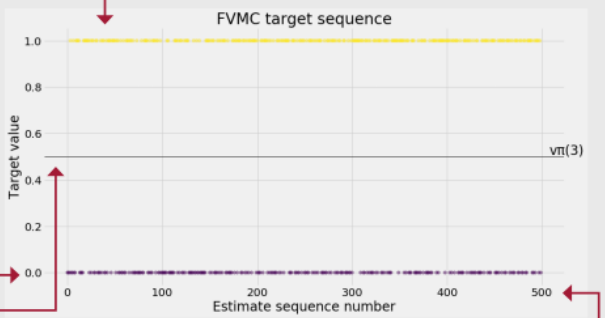
Tally it Up

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*.

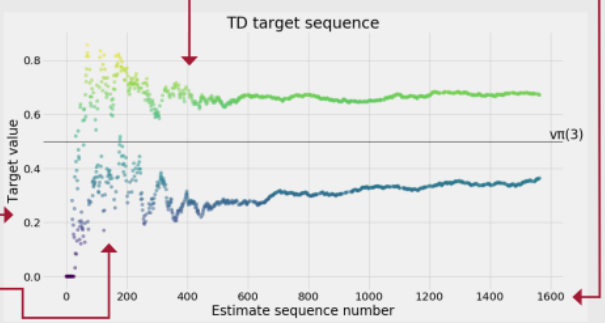
(2) These plots are showing the targets for the initial state in the RW environment. MC targets, the returns, are either 0 or 1 because the episode terminates either on the left, with a 0 return or on the right, with a 1 return, while the optimal value is 0.5!



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

(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.

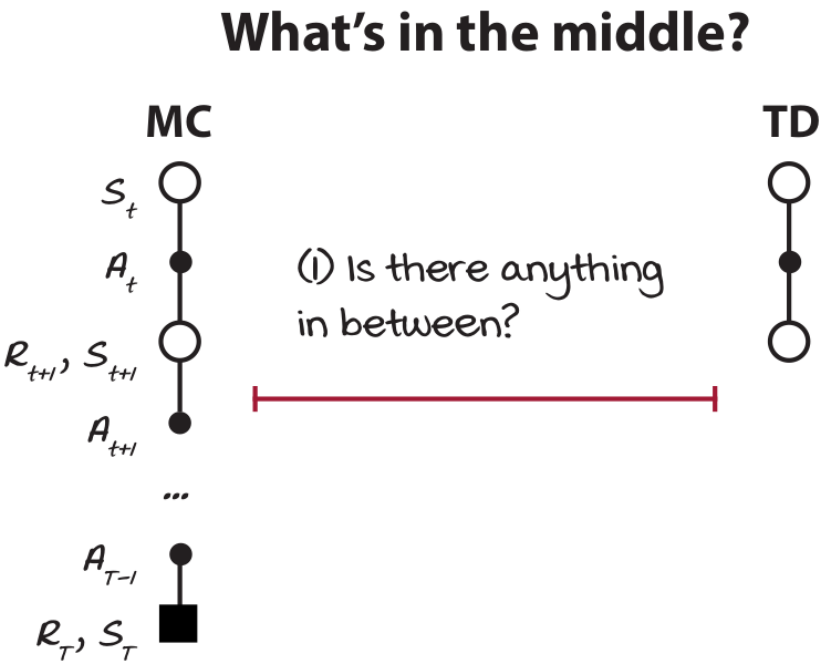
(4) Here you can see the range of the TD targets is much lower, MC alternates *exactly* between 1 and 0, and TD jumps between *approximately* 0.7 and ~0.3, depending on which "next state" is sampled. But as the $V_t(S_{t+1})$ is an estimate, $G_{t:t+1}$ is biased, off-target, and inaccurate.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Оцінювання функції цінності з кількох кроків



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

N-крокoвий TD

Show Me the Math

N-step temporal-difference equations

$$S_t, A_t, R_{t+1}, S_{t+1}, \ldots, R_{t+n}, S_{t+n} \sim \pi_{t:t+n}$$

(1) Notice how in *n*-step TD we must wait *n* steps before we can update *V*(*s*).
(2) Now, *n* doesn't have to be ∞ like in MC, or 1 like in TD. Here you get to pick. In reality *n* will be *n* or less if your agent reaches a terminal state. It could be less than *n*, but never more.

$$G_{t:t+n} = R_{t+1} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

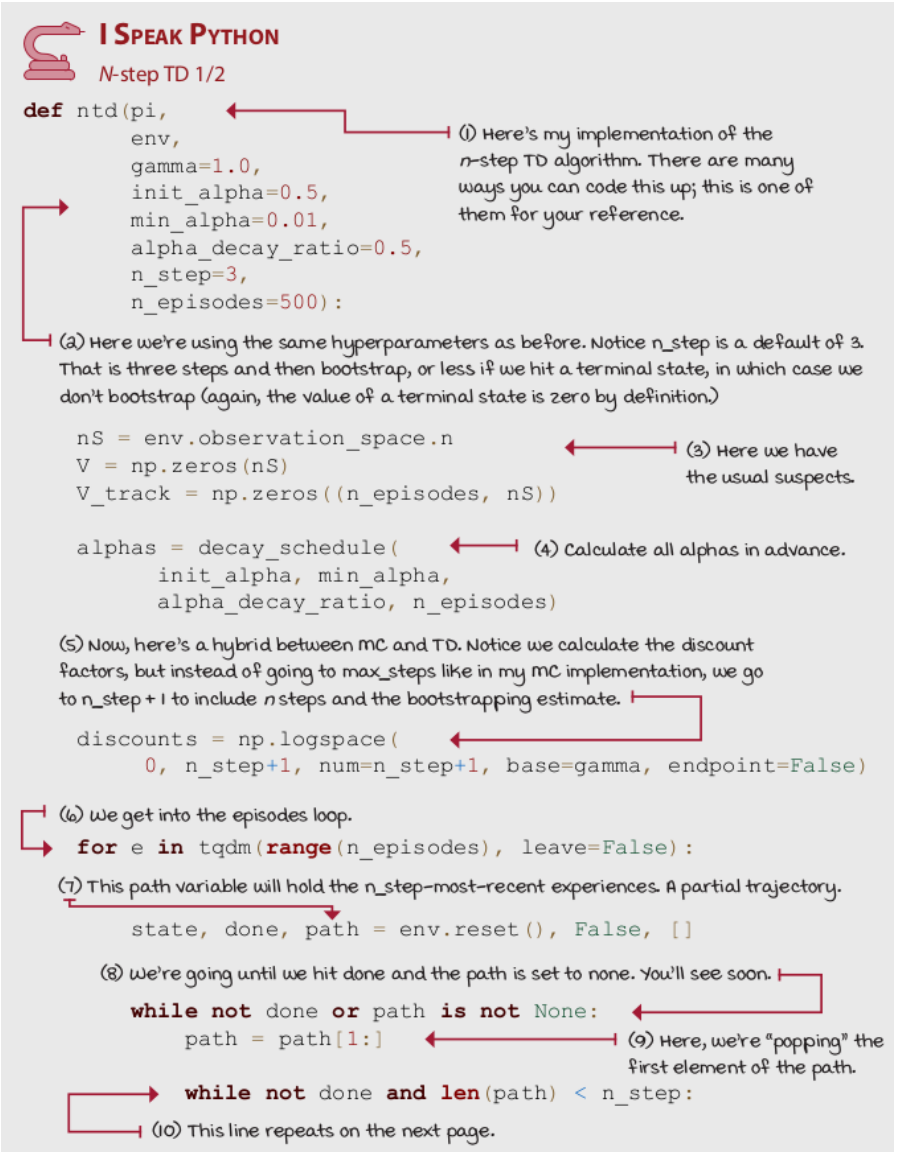
(3) Here you see how the value-function estimate gets updated approximately every *n* steps.

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

(4) But after that, you can plug in that target as usual.

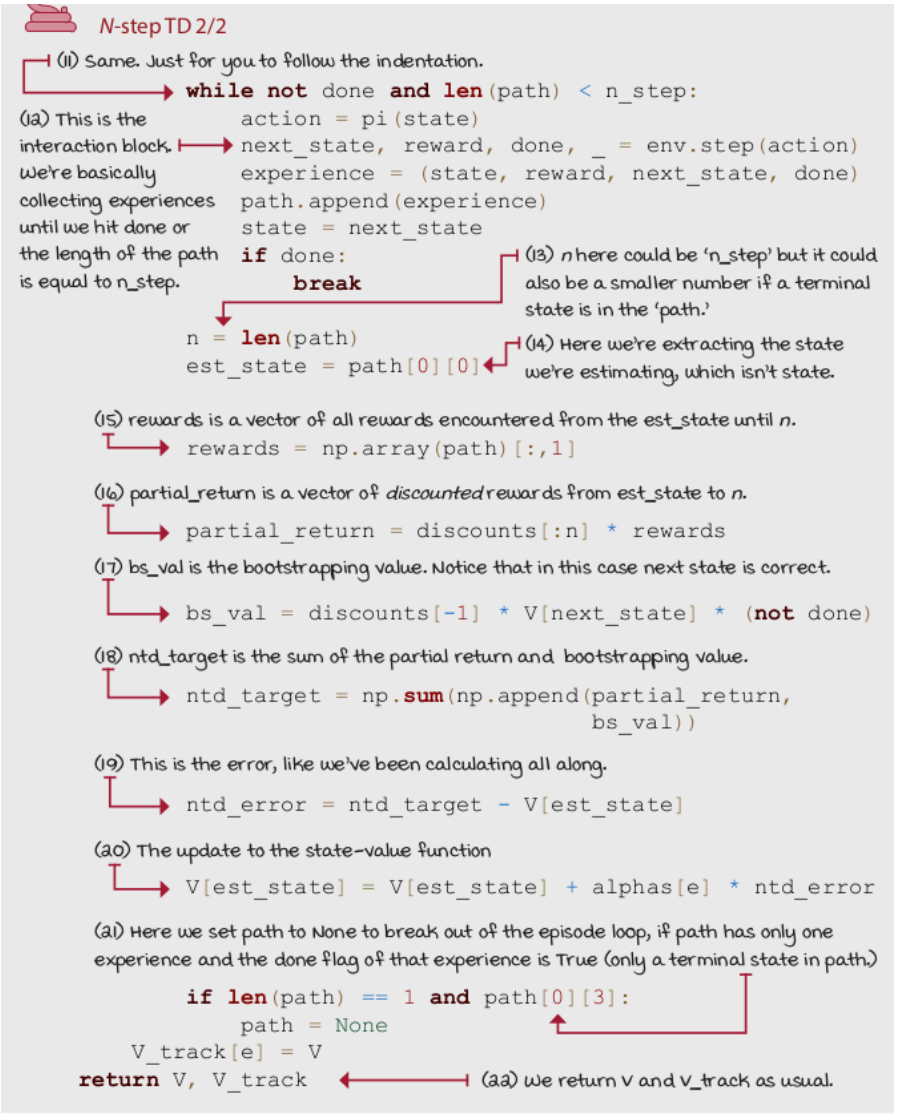
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



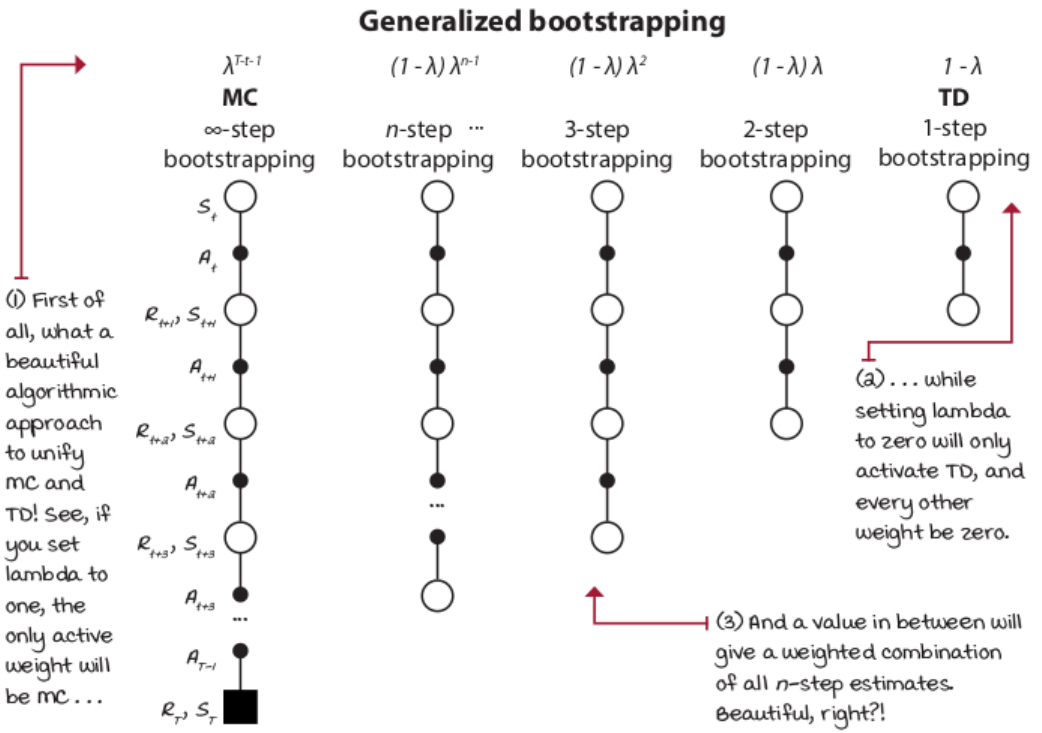
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

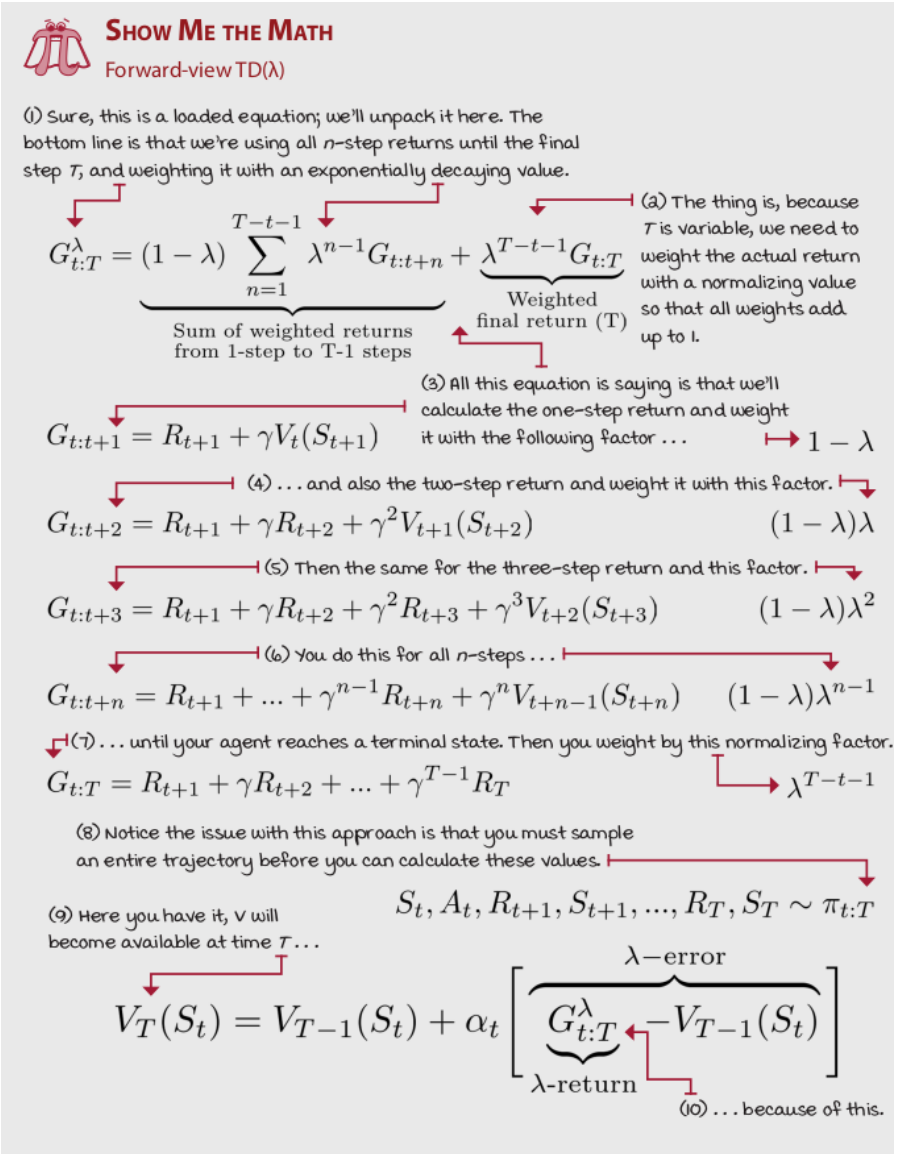

N-step TD 2/2
(11) Same. Just for you to follow the indentation.
while not done and len(path) < n_step:
(12) This is the interaction block. We're basically collecting experiences until we hit done or the length of the path is equal to n_step.
action = pi(state)
next_state, reward, done, _ = env.step(action)
experience = (state, reward, next_state, done)
path.append(experience)
state = next_state
if done:
break
(13) n here could be 'n_step' but it could also be a smaller number if a terminal state is in the 'path.'
n = len(path)
est_state = path[0][0]
(14) Here we're extracting the state we're estimating, which isn't state.
(15) rewards is a vector of all rewards encountered from the est_state until n.
rewards = np.array(path)[:,1]
(16) partial_return is a vector of discounted rewards from est_state to n.
partial_return = discounts[:n] * rewards
(17) bs_val is the bootstrapping value. Notice that in this case next state is correct.
bs_val = discounts[-1] * V[next_state] * (not done)
(18) ntd_target is the sum of the partial return and bootstrapping value.
ntd_target = np.sum(np.append(partial_return, bs_val))
(19) This is the error, like we've been calculating all along.
ntd_error = ntd_target - V[est_state]
(20) The update to the state-value function
V[est_state] = V[est_state] + alphas[e] * ntd_error
(21) Here we set path to None to break out of the episode loop, if path has only one experience and the done flag of that experience is True (only a terminal state in path.)
if len(path) == 1 and path[0][3]:
path = None
V_track[e] = V
return V, V_track
(22) We return V and V_track as usual.

—

TD(λ): Покращена оцінка усіх відвіданих станів

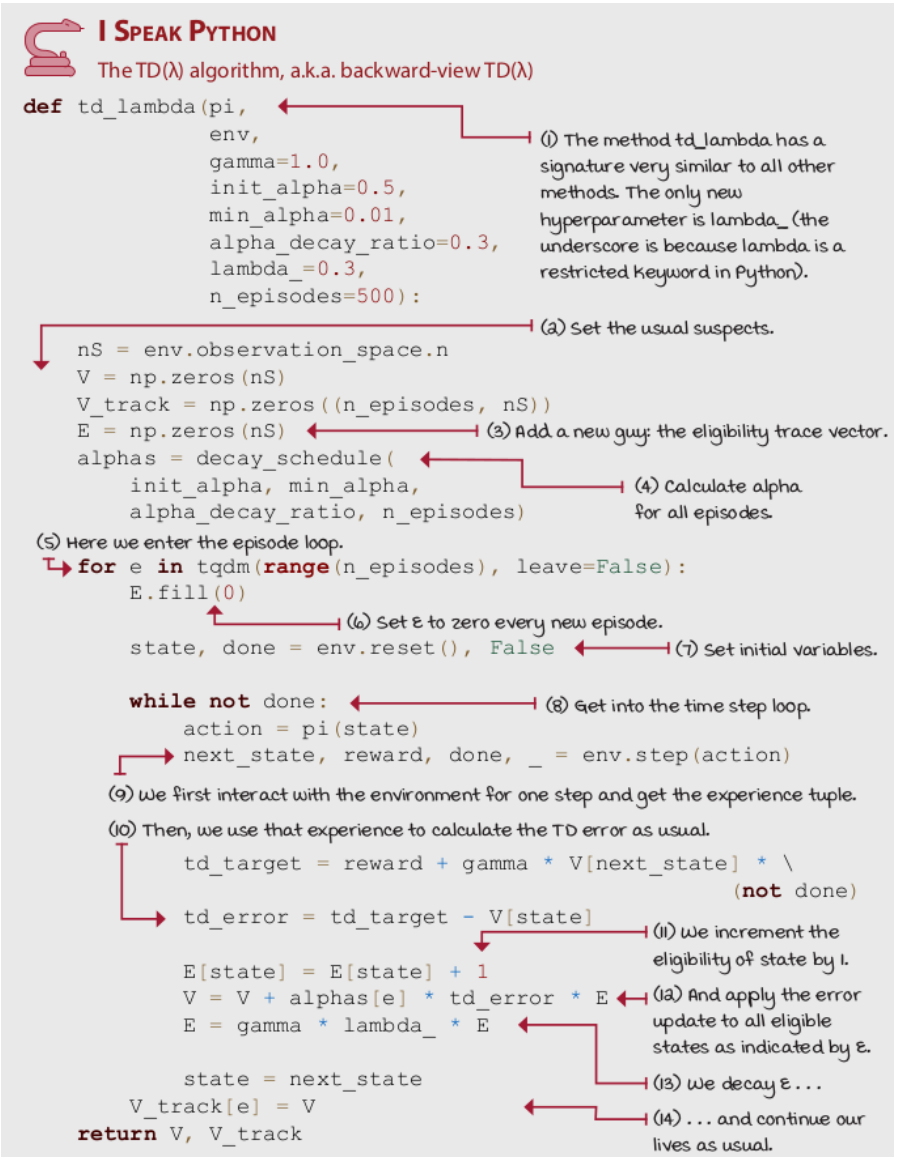


Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Tally it Up

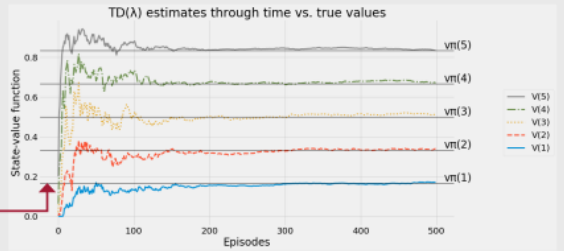
Running estimates that *n*-step TD and TD(λ) produce in the RW environment

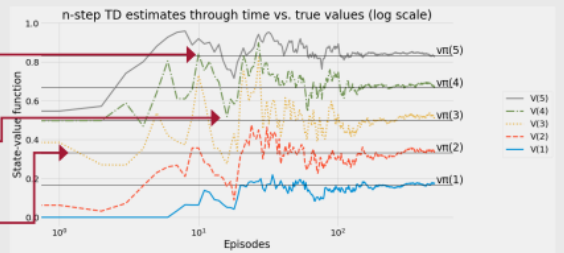
(1) I think the most interesting part of the differences and similarities of MC, TD, *n*-step TD, and TD(lambda) can be visualized side by side. For this, I highly recommend you head to the book repository and check out the corresponding Notebook for this chapter. You'll find much more than what I've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

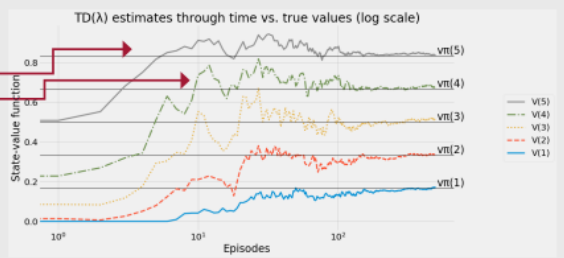
(2) But for now I can highlight that *n*-step TD curves are a bit more like MC: noisy and centered, while TD(lambda) is a bit more like TD: smooth and off-target.

(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!







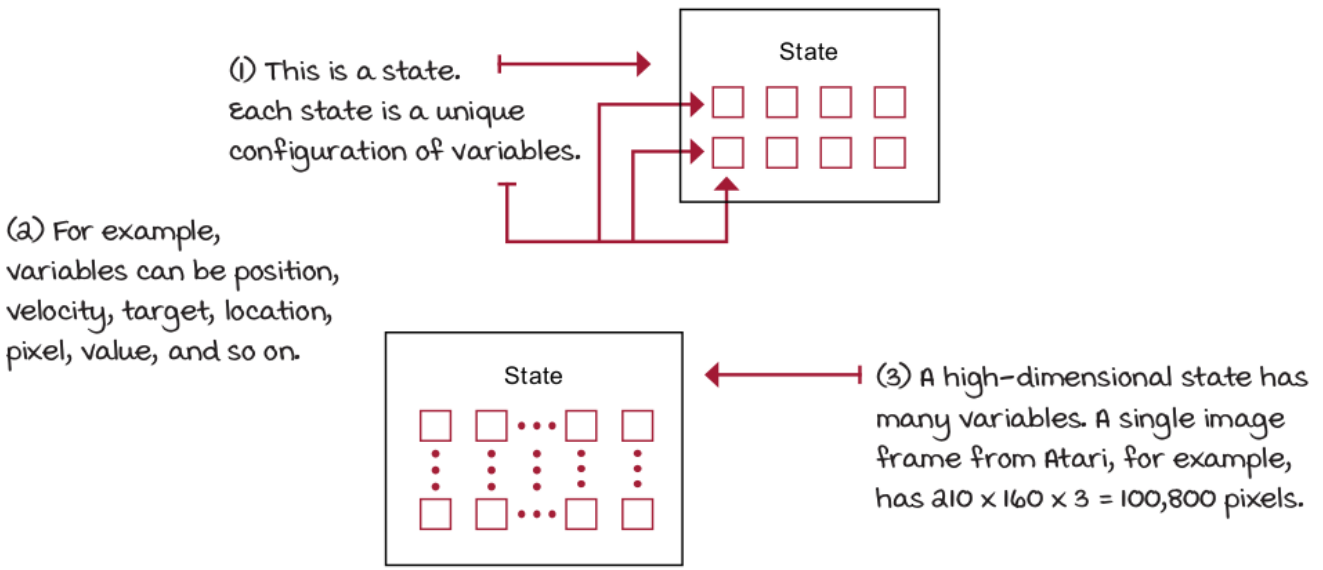
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Демо

Методи апроксимації функції цінності

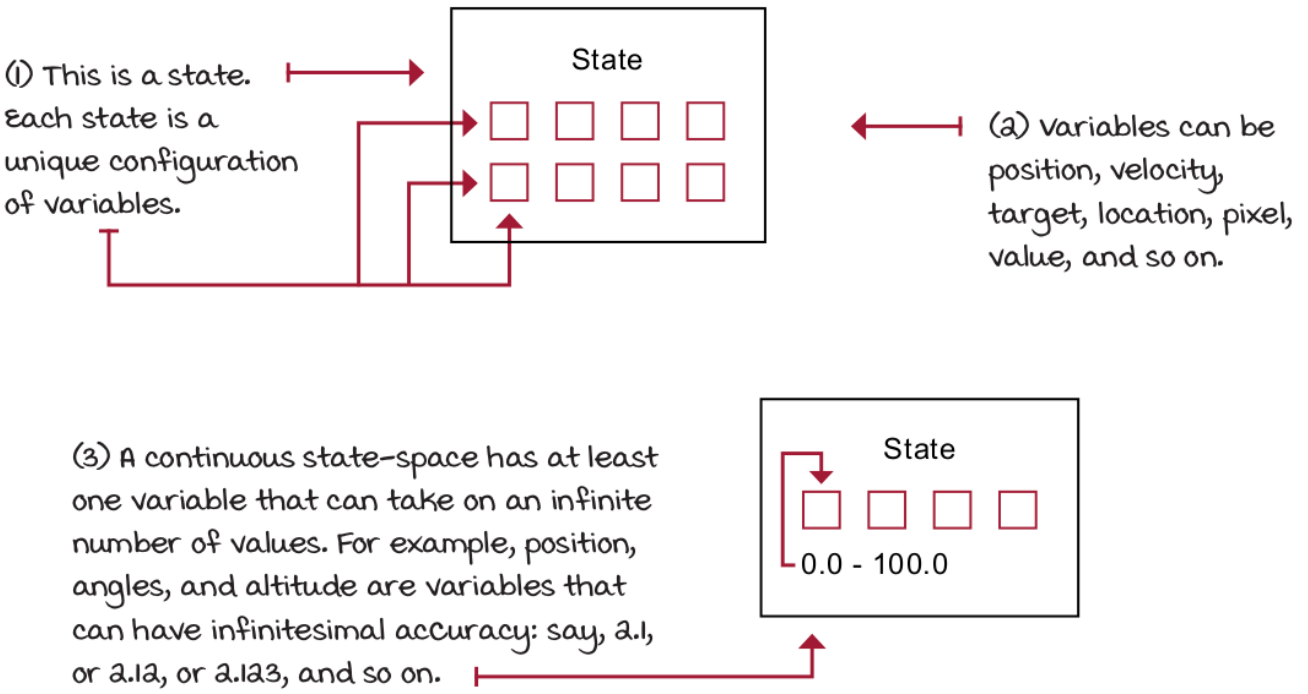
High-dimensional state spaces



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

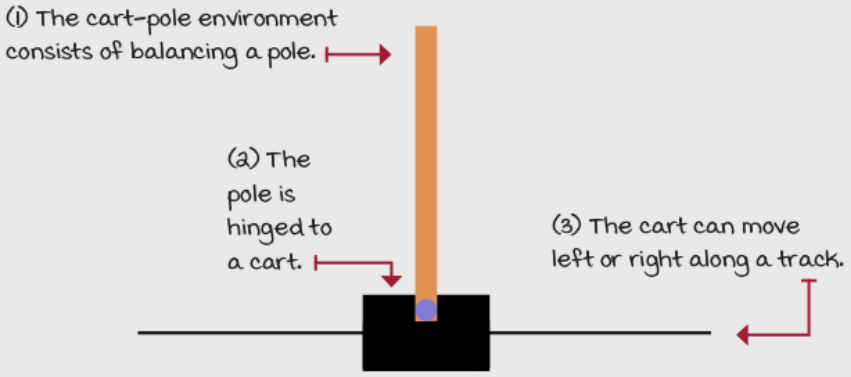
Continuous state spaces



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

This is the cart-pole environment



Its state space is comprised of four variables:

- The cart position on the track (x-axis) with a range from –2.4 to 2.4
- The cart velocity along the track (x-axis) with a range from –inf to inf
- The pole angle with a range of ~–40 degrees to ~ 40 degrees
- The pole velocity at the tip with a range of –inf to inf

There are two available actions in every state:

- Action 0 applies a –1 force to the cart (push it left)
- Action 1 applies a +1 force to the cart (push it right)

You reach a terminal state if

- The pole angle is more than 12 degrees away from the vertical position
- The cart center is more than 2.4 units from the center of the track
- The episode count reaches 500 time steps (more on this later)

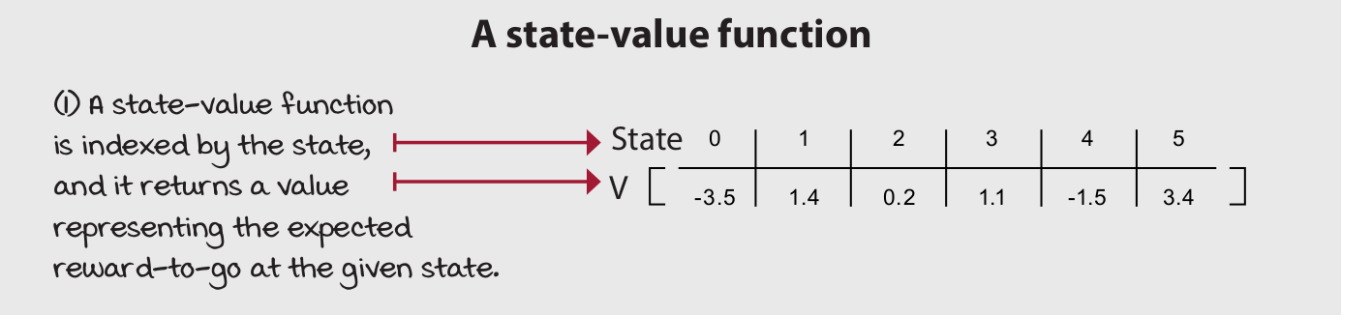
The reward function is

- +1 for every time step

—

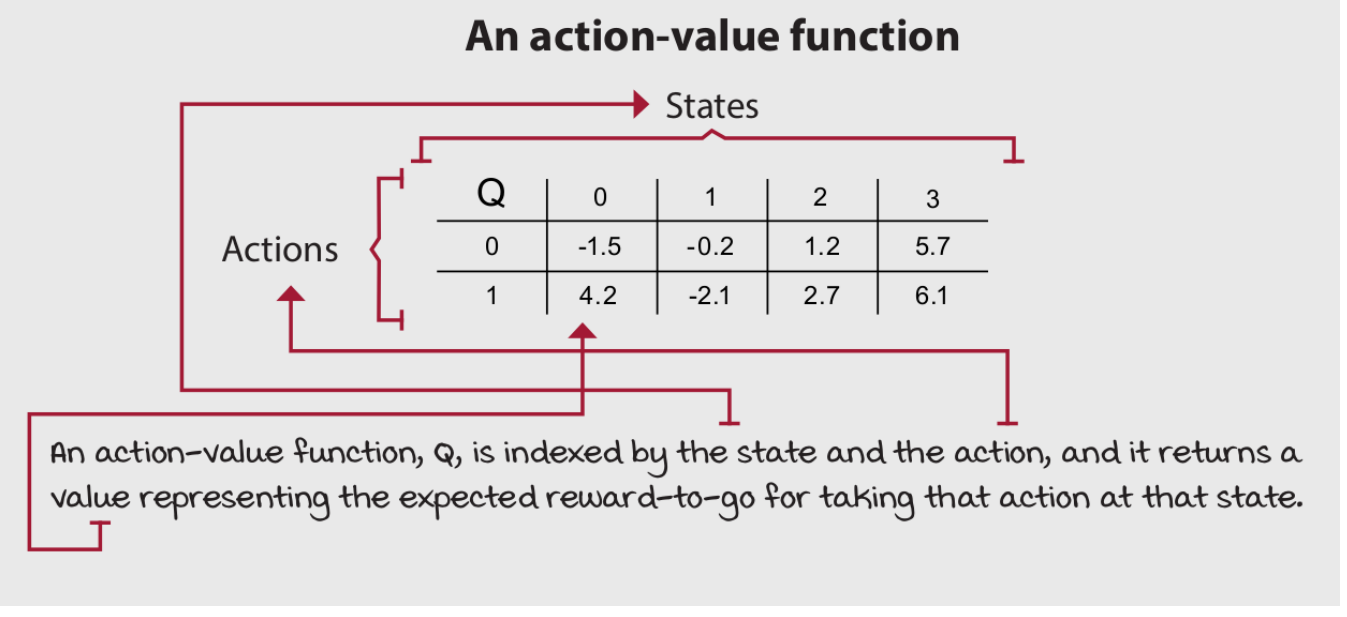
Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Апроксимація функцій має переваги



Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Апроксимація функцій має переваги



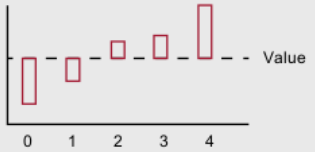
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

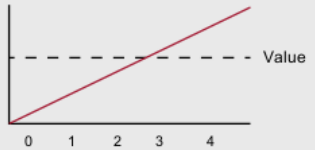
A state-value function with and without function approximation

(1) Imagine this state-value function. $V = [-2.5, -1.1, 0.7, 3.2, 7.6]$

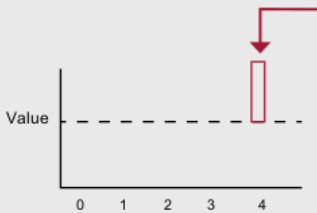
(2) Without function approximation, each value is independent.



(3) With function approximation, the underlying relationship of the states can be learned and exploited.

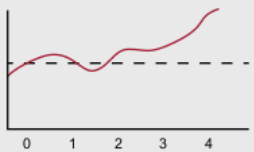


(4) The benefit of using function approximation is particularly obvious if you imagine these plots after even a single update.



(5) Without function approximation, the update only changes one state.

(6) With function approximation, the updates change multiple states.



(7) Of course, this is a simplified example, but it helps illustrate what's happening. What would be different in "real" examples?

First, if we approximate an action-value function, Q, we'd have to add another dimension.

Also, with a non-linear function approximator, such as a neural network, more complex relationships can be discovered.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Апроксимація функцій має переваги

Boil it Down

Reasons for using function approximation

Our motivation for using function approximation isn't only to solve problems that aren't solvable otherwise, but also to solve problems more efficiently.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

First decision point: Selecting a value function to approximate

- The state-value function $v(s)$
- The action-value function $q(s, a)$
- The action-advantage function $a(s, a)$

Which Function Approximator?

There are many function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

—

Джерело: David Silver.

Which Function Approximator?

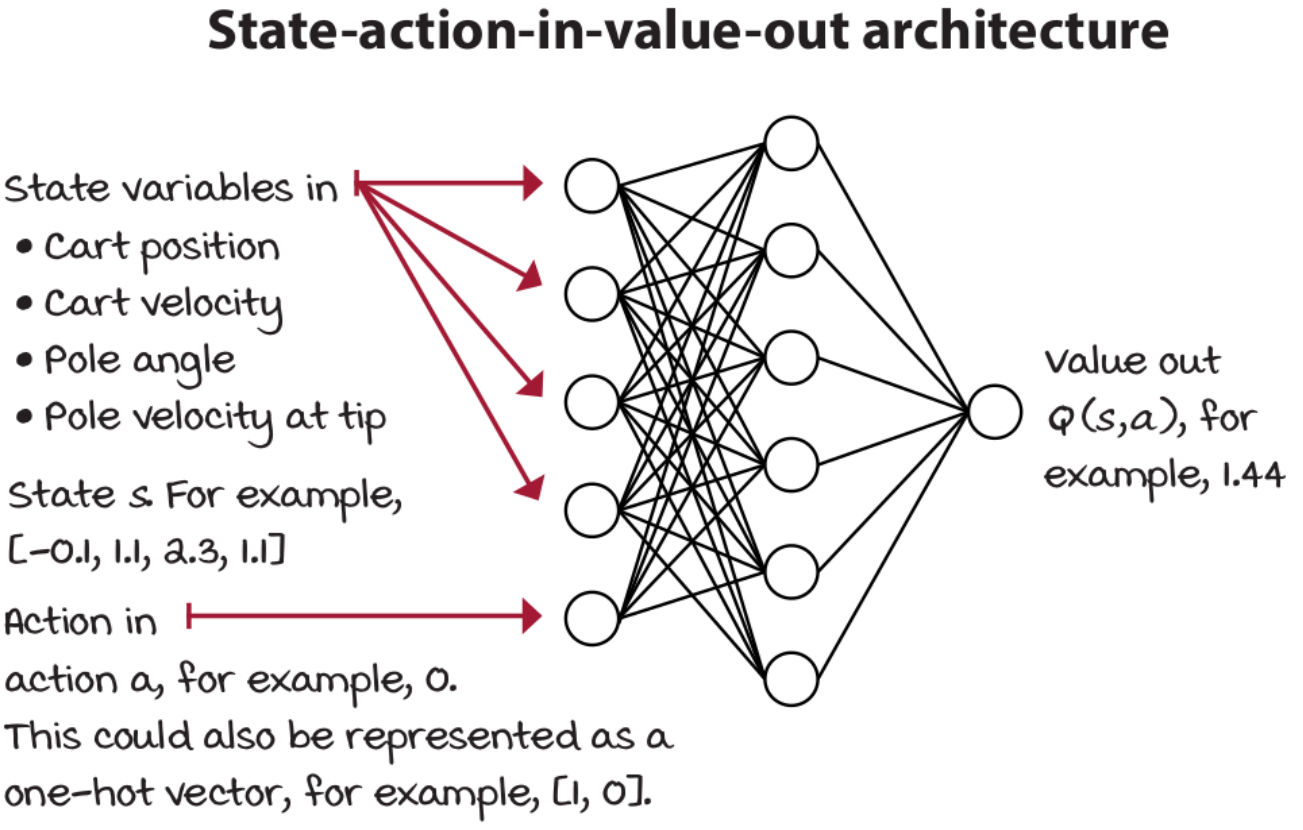
We consider differentiable function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

—

Джерело: David Silver.

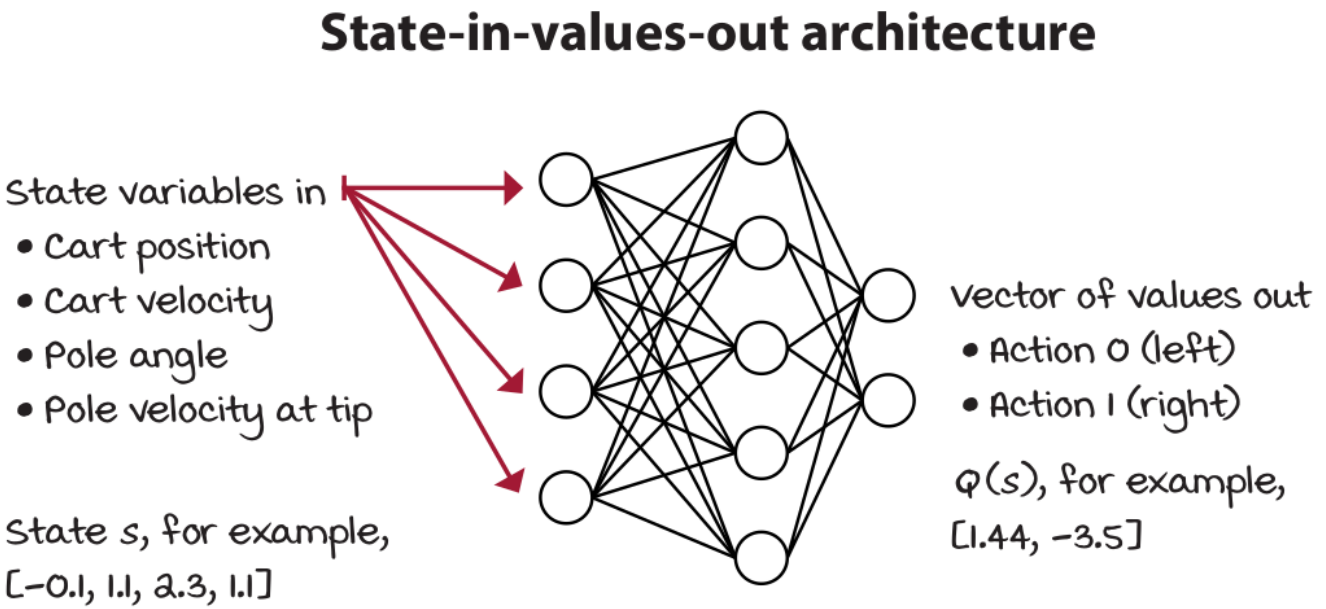
Second decision point: Selecting a neural network architecture



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Second decision point: Selecting a neural network architecture



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Third decision point: Selecting what to optimize

Show Me the Math

Ideal objective

(1) An ideal objective in value-based deep reinforcement learning would be to minimize the loss with respect to the optimal action-value function q_*.

(2) We want to have an estimate of q_*, Q, that tracks exactly that optimal function.

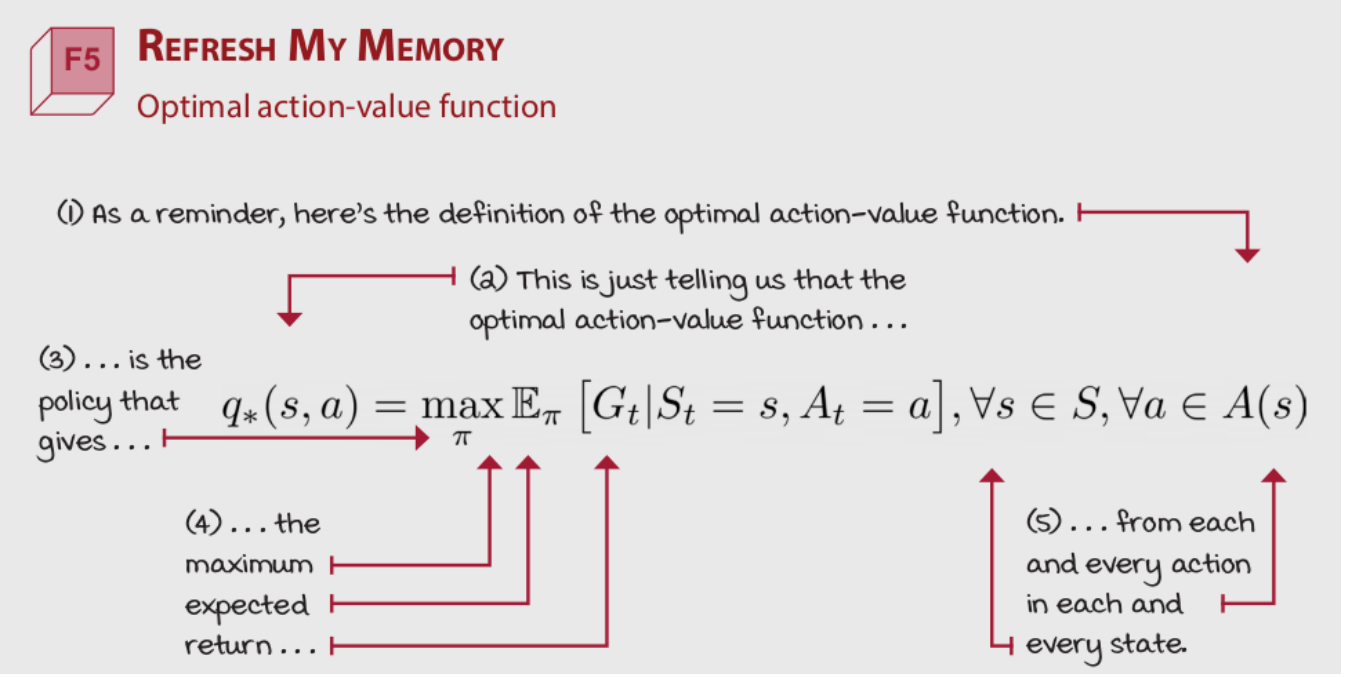
$$L_i(\theta_i) = \mathbb{E}_{s,a}\left[\left(q_*(s,a) - Q(s,a;\theta_i)\right)^2\right]$$

(3) If we had a solid estimate of q_*, we then could use a greedy action with respect to these estimates to get near-optimal behavior—only if we had that q_*.

(4) Obviously, I'm not talking about having access to q_* so that we can use it; otherwise, there's no need for learning. I'm talking about access to sampling the q_* some way: regression-style ML.

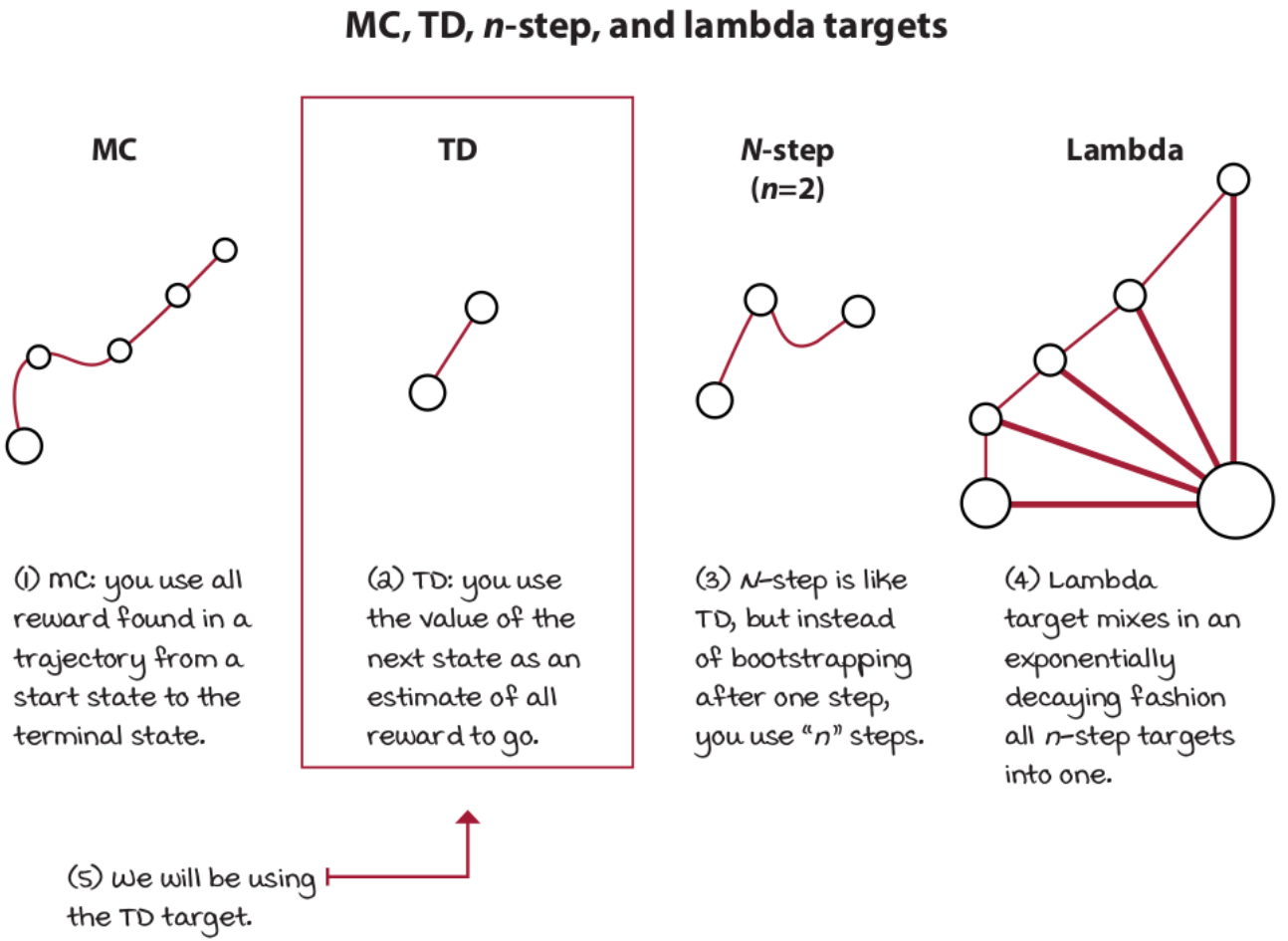
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Fourth decision point: Selecting the targets for policy evaluation



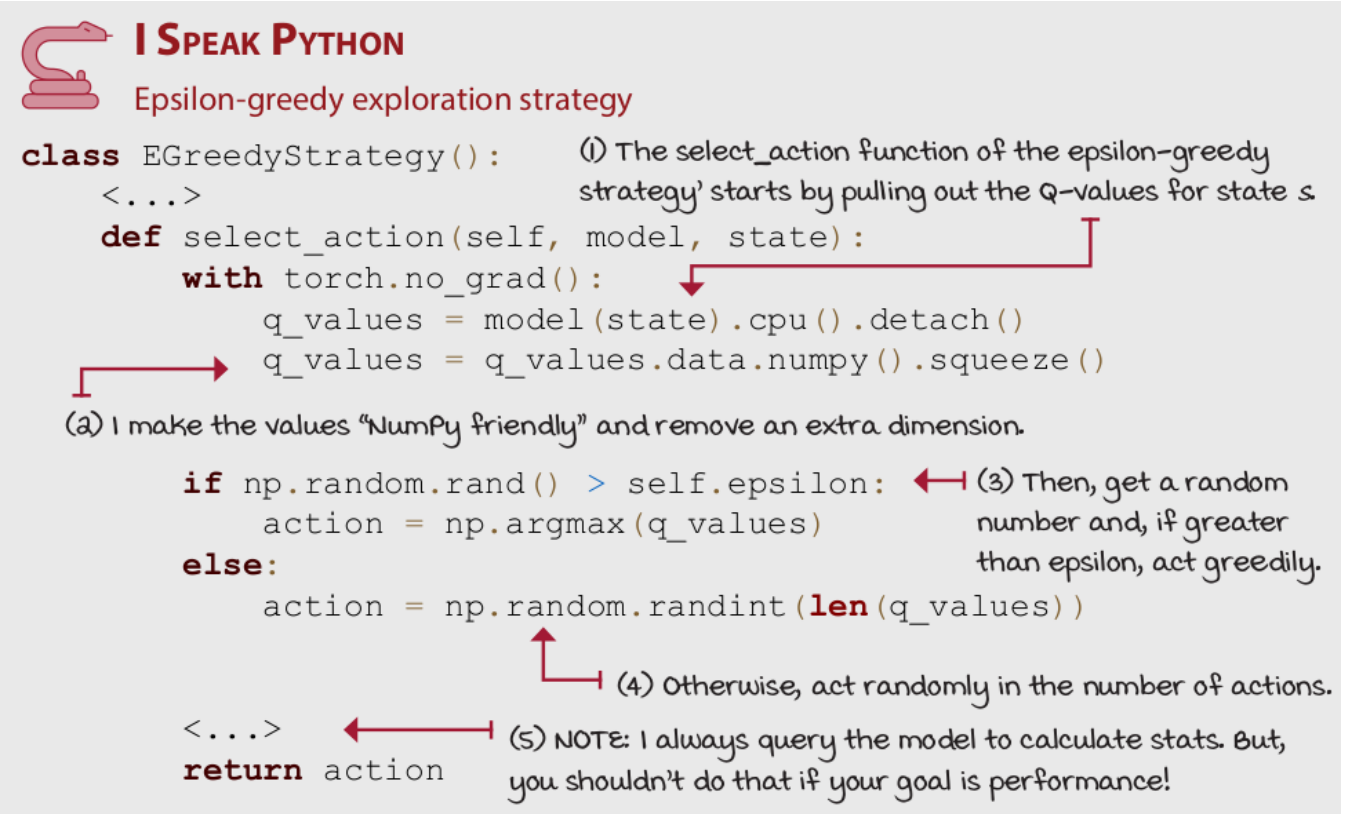
—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Fifth decision point: Selecting an exploration strategy

Another thing we need to decide is which policy improvement step to use for our generalized policy iteration needs.

—



Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Sixth decision point: Selecting a loss function

- L1
- L2, MSE
- ...

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Seventh decision point: Selecting an optimization method

- Batch gradient descent
- Mini-batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent vs. momentum

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Batch gradient descent

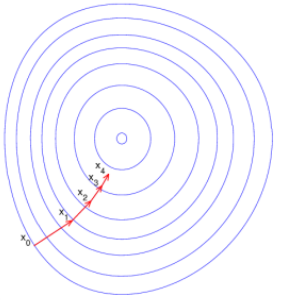
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be

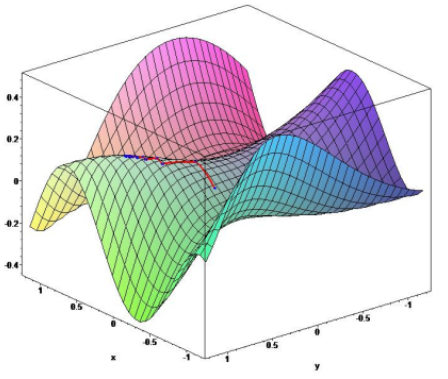
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$
- Adjust $\mathbf{w}$ in direction of -ve gradient

$$\Delta \mathbf{w} = -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

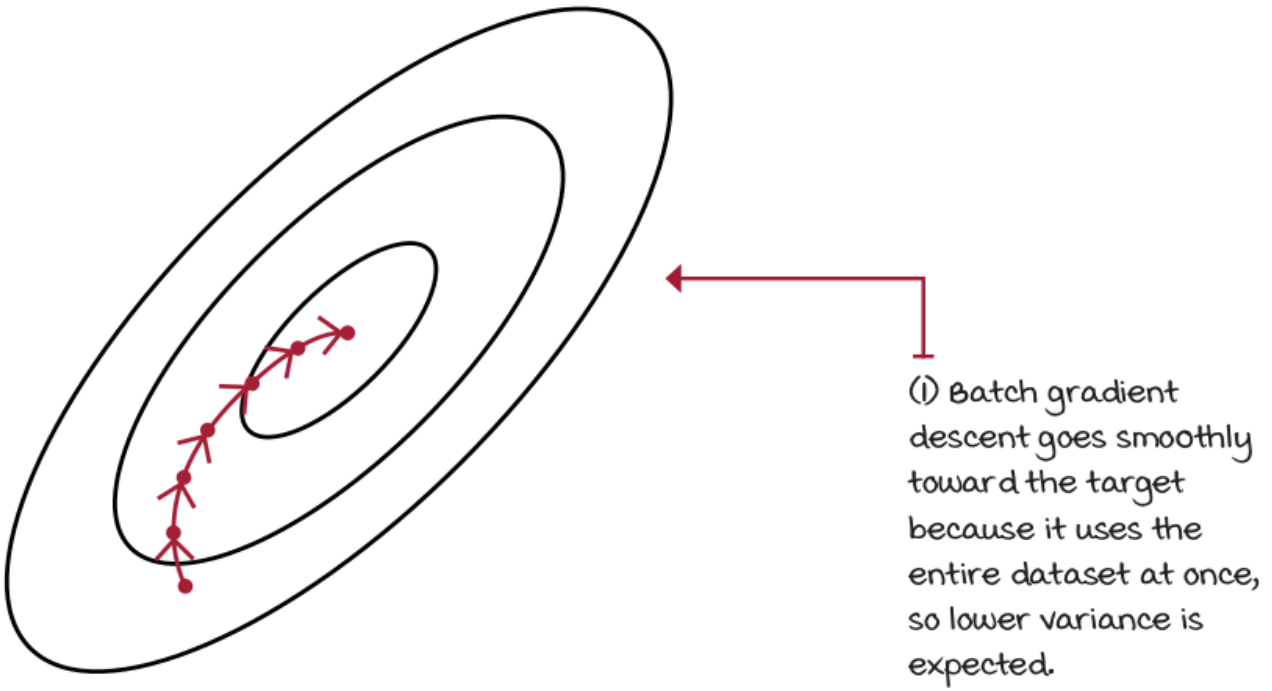
where α is a step-size parameter





Джерело: David Silver.

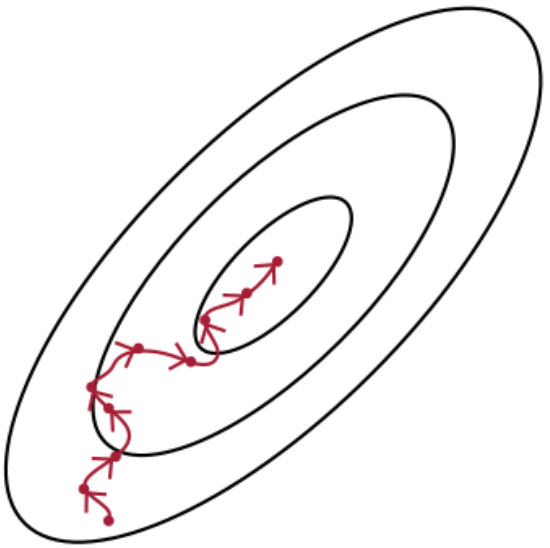
Batch gradient descent



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Mini-batch gradient descent

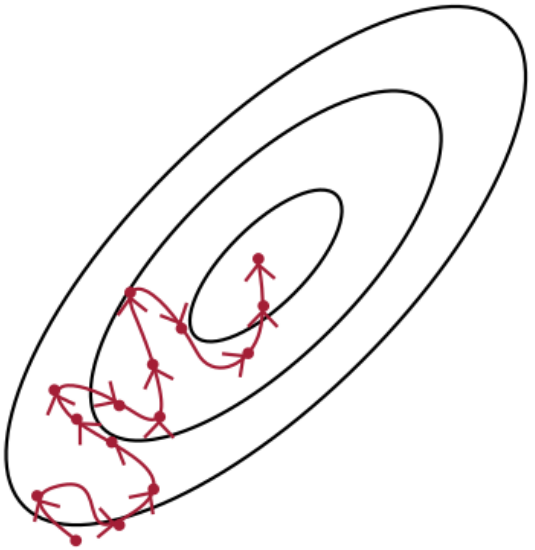


(1) In mini-batch gradient descent we use a uniformly sampled mini-batch. This results in noisier updates, but also faster processing of the data.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

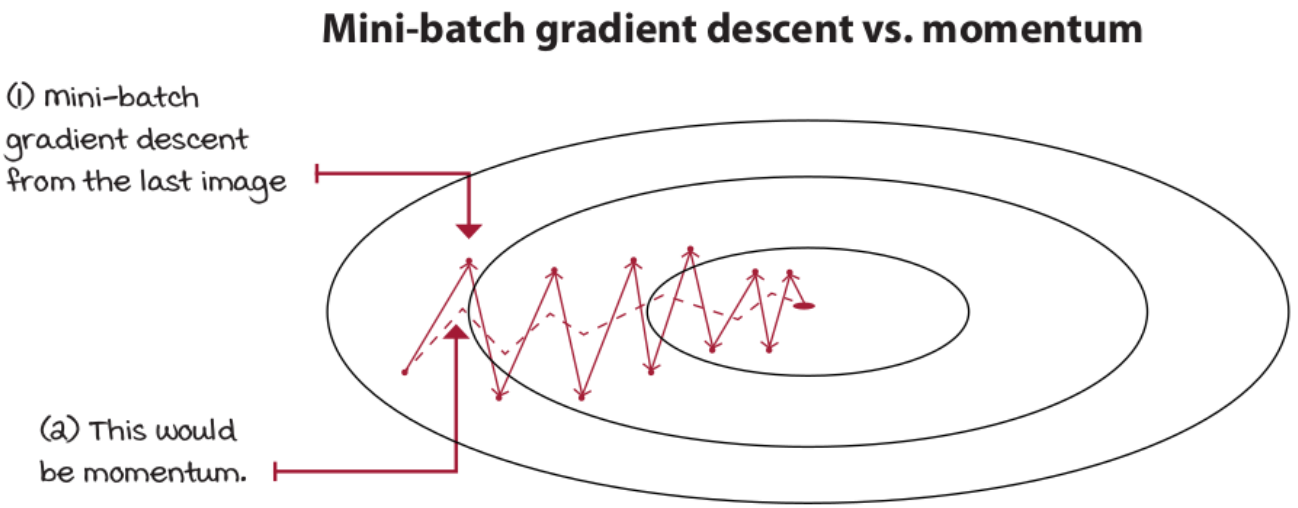
Stochastic gradient descent



(!) With stochastic gradient descent, in every iteration we step through only one sample. This makes it a noisy algorithm. It wouldn't be surprising to see several steps taking us further away from the target, and later back toward the target.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

It's in the Details

The full neural fitted Q-iteration (NFQ) algorithm

Currently, we've made the following selections:

- Approximate the action-value function $Q(s,a;\theta)$.
- Use a state-in-values-out architecture (nodes: 4, 512,128, 2).
- Optimize the action-value function to approximate the optimal action-value function $q^*(s,a)$.
- Use off-policy TD targets ($r + \gamma^* max\_a' Q(s',a';\theta)$) to evaluate policies.
- Use an epsilon-greedy strategy (epsilon set to 0.5) to improve policies.
- Use mean squared error (MSE) for our loss function.
- Use RMSprop as our optimizer with a learning rate of 0.0005.

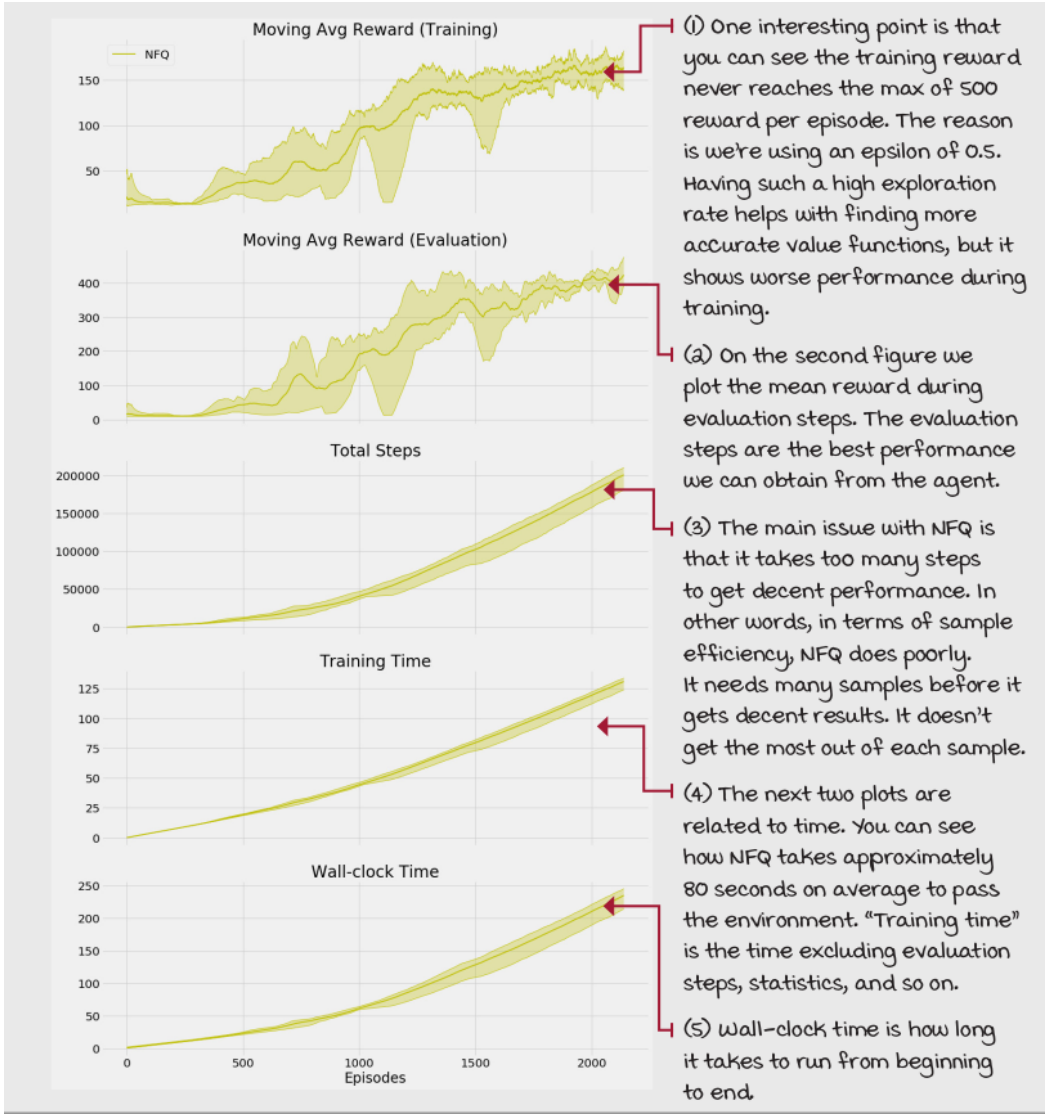
NFQ has three main steps:

1. Collect E experiences: (s, a, r, s', d) tuples. We use 1024 samples.
2. Calculate the off-policy TD targets: $r + \gamma^* max\_a' Q(s',a';\theta)$.
3. Fit the action-value function $Q(s,a;\theta)$ using MSE and RMSprop.

This algorithm repeats steps 2 and 3 K number of times before going back to step 1. That's what makes it fitted: the nested loop. We'll use 40 fitting steps K.

—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.



—

Джерело: Grokking Deep Reinforcement Learning, Miguel Morales.

Демо

Кінець

Література

- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.
    - Richard Sutton and Samuel Barto, Reinforcement Learning: an introduction, second edition
    - Richard Sutton Learning to predict by the methods of temporal differences
    - Marco Wiering and Martijn van Otterlo, Reinforcement Learning
    - Watkins Christopher and Peter Dayan, Q-Learning
    - David Silver, Lecture 2: Markov Decision Processes