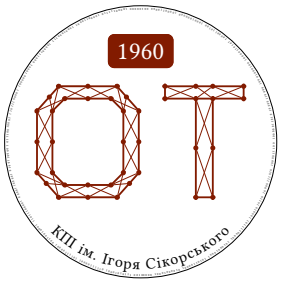
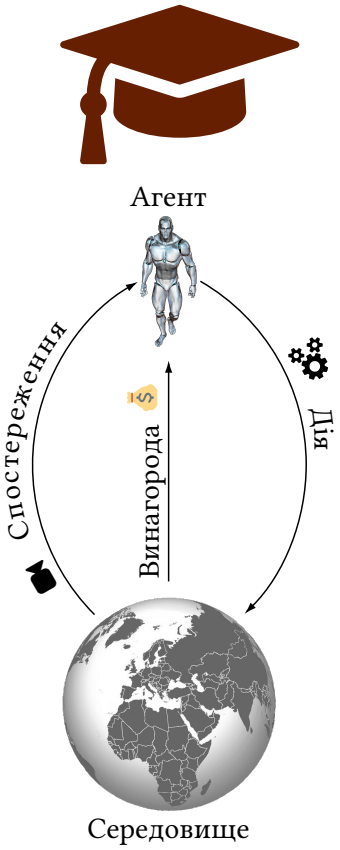
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”



Навчання з підкріпленням

Динамічне програмування vs Монте-Карло | Осінь, 2021



Інструктор

КОЧУРА Юрій Петрович

Кафедра ОТ, ФІОТ

@y_kochura

iuriy.kochura@gmail.com

Особливості

Для магістрів 2-го курсу, осінь

121 – “Інженерія програмного забезпечення”
123 – “Комп’ютерна інженерія”

Вибіркова дисципліна

Очна форма навчання

Українська, англійська

4 кредити ЄКТС

7 лекцій

3 практичні роботи + проєкт

Залік

Розміщення курсу

Опис

Навчання з підкріпленням (англ. reinforcement learning, RL) — це галузь машинного навчання, а також формалізм для автоматизованого прийняття рішень на основі взаємодій. За останні 5 років глибинне навчання з підкріпленням (deep RL) стало одним з найінтенсивніших напрямків досліджень у сфері штучного інтелекту. Сьогодні deep RL дозволяє досягати надлюдської продуктивності в ряді завдань: відео ігри, покер, а також у настільних іграх, включаючи ґо та шахи.

Цей курс познайомить Вас з сімейством статистичних алгоритмів, які вивчають оптимальну стратегію, метою якої є максимізація загальної винагороди, отриманої агентом при взаємодії з навколишнім середовищем.

Потрібні навички

Для проходження цього курсу потрібно володіти наступними навичками:

- Базові знання з лінійної алгебри та теорії ймовірностей.
- Досвід тренування глибинних мереж (ініціалізація, оптимізація, регуляризація, вибір методу та метрик для оцінки).

Система оцінювання

30%	Практичні завдання (10% кожне)
40%	Проєкт
30%	Залік

Важливо! Умова допуску до семестрового контролю (заліку):

$$\text{Практичні завдання} + \text{Проєкт} \geq 42\%$$

Шкала оцінок КПІ ім. Ігоря Сікорського:

A = 95–100	Відмінно
B = 85–94	Дуже добре
C = 75–84	Добре
D = 65–74	Задовільно
E = 60–64	Достатньо
F < 60	Незадовільно
Fx < 42	Недопущений
Порушення кодексу честі	Усунений

Кодекс честі

Ви можете обговорювати завдання практичних робіт у групах. Однак, кожен студент/студентка повинен/повинна підготувати розв’язки завдань самостійно.

Під час проходження цього курсу Ви зобов’язані дотримуватись Кодекс честі КПІ ім. Ігоря Сікорського та усі наступні правила:

1. Кожен з Вас повинен відправляти на перевірку власно виконану роботу. Використання чужих розв’язків або програмного коду і представлення їх за свої напрацювання є плагіатом та серйозним порушенням основних академічних стандартів.
2. Ви не повинні ділитися своїми розв’язками з іншими студентами, а також просити інших ділитися своїми розв’язками з Вами.
3. Якщо Ви отримували допомогу у вирішенні певного завдання, Ви повинні зазначити це у звіті, а саме: від кого та яку допомогу отримали.

Навчання з підкріпленням

"Small daily improvements over time create stunning results."
"Маленькі щоденні зусилля згодом дають приголомшливі результати. "

– Робін Шарма

Динамічне програмування vs Монте-Карло

Ми можемо навчити алгоритм виконувати якесь завдання високого рівня, призначаючи за виконання завдання високу винагороду (тобто позитивне підкріплення) і негативно підкріплюючи те, що ми не хочемо, щоб алгоритм виконував. Давайте розглянемо конкретний приклад, коли, скажімо, мета високого рівня полягає у тому, щоб навчити робота-пилососа переміщатися з деякої кімнати будинку до його місця підзарядки (док-станції), яка знаходиться на кухні. Робот-пилосос може виконувати чотири дії: рухатись ліворуч, рухатись праворуч, рухатись вперед та рухатись назад. У кожен момент часу робот повинен вирішити, яку з цих чотирьох дій йому слід виконати. Якщо він дістається док-станції, він отримує винагороду +100, а якщо він вдариться у що-небудь на своєму шляху, він отримує від'ємну винагороду −10. Скажімо, робот має повну 3D-карту будинку і знає точне розташування док-станції, але він все ще не знає, яку послідовність примітивних дій йому слід виконати, щоб дістатися до цього місця.

Один з підходів, який дозволяє вирішити цю задачу називається `динамічним програмуванням` (ДП). ДП вперше було сформульовано Річардом Беллманом[1] у 1957 році. Динамічне програмування можна було б краще назвати `декомпозицією мети`, оскільки воно вирішує складні задачі (високого рівня), розкладаючи їх на все менші й менші підзадачі, поки не дійде до простої підзадачі, яку можна вирішити без додаткової інформації.

Замість того, щоб робот намагався придумати довгу послідовність примітивних дій, які приведуть його до док-станції, він може спочатку розбити проблему на окремі підзадачі: "залишитися в цій кімнаті" та "вийти з цієї кімнати". Оскільки у нього є повна карта будинку, він знає, що йому потрібно вийти з кімнати, оскільки док-станція знаходиться на кухні. Але він все ще не знає, яка послідовність дій дозволить йому вийти з кімнати, тому продовжує розкладати проблему на підзадачі: "рухатися до дверей" або "рухатись від дверей". Оскільки двері розташовані ближче до док-станції, а від дверей до док-станції можна прокласти далі маршрут, робот знає, що йому потрібно рухатися до дверей, але знову ж таки він не знає, яка послідовність примітивних дій приведе його до дверей. Нарешті, йому потрібно вирішити у якому напрямку слід рухатись: вліво, вправо, вперед чи назад. Бачачи перед собою двері, робот рухається вперед. Він продовжує цей процес до тих пір, поки не вийде з кімнати. Далі, коли робот покине кімнату, йому доведеться розбивати наступну задачу на окремі підзадачі, поки не дістанеться док-станції.

У цьому суть динамічного програмування. Це загальний підхід до вирішення певних типів задач, які можна розбити на підзадачі, у свою чергу ці підзадачі можна розбити ще на окремі підзадачі і повторювати цей процес до тих пір, поки не дійдемо до простої підзадачі, яку можна буде вирішити без додаткової інформації. Динамічне програмування застосовується у багатьох областях, включаючи біоінформатику, економіку та комп'ютерні науки.

[1] Річард Ернест Беллман – американський математик, який вперше сформулював та ввів динамічне програмування у 1957 році у своїй книзі "Dynamic Programming".

Щоб застосувати динамічне програмування Беллмана, ми повинні вміти розбивати нашу задачу на підзадачі, які ми знаємо як вирішити. Але навіть це, здавалося б, логічне припущення важко реалізувати в реальному світі. Як розбити завдання високого рівня для самокерованого автомобіля на окремі підзадачі, щоб він міг доїхати з точки А до точки Б не спричинивши аварію? Чи навчиться дитина ходити, перш ніж вона вирішить легші задачі з ходьбою? У RL, де ми часто маємо нюанси, які можуть включати елемент випадковості, ми не можемо застосувати динамічне програмування саме так, як це виклав Беллман. Фактично, ДП можна вважати одним з методів вирішення проблем такого роду. Іншим підходом буде випадковий метод проб і помилок.

В одних ситуаціях ми маємо максимальні знання про середовище, а в інших ми володіємо мінімальними знаннями про середовище, тому нам потрібно використовувати різні стратегії в кожному випадку. Якщо вам потрібно скористатися ванною у вашому власному будинку, ви точно знаєте (ну, принаймні несвідомо), яка послідовність м'язових рухів приведе вас до ванної кімнати з будь-якої вихідної позиції (тобто, маємо динамічне програмування). Це тому, що ви дуже добре знаєте свій будинок – у вас на думці є більш-менш ідеальна модель вашого будинку. Якщо ви йдете на вечірку в будинок у якому ви ніколи раніше не були, вам, можливо, доведеться озирнутися, поки не знайдете ванну самостійно (тобто методом проб і помилок), оскільки у вас немає хорошої моделі цього будинку.

Стратегія проб і помилок зазвичай підпадає під егіду методів Монте-Карло. Метод Монте-Карло, по суті, являє собою випадкову вибірку середовища. У багатьох реальних задачах ми маємо лише певні знання про те, як працює середовище, тому ми в кінцевому підсумку використовуємо змішану стратегію, яка включає в себе певну кількість проб і помилок і деяку кількість використання того, що ми вже знаємо про середовище для безпосереднього вирішення проблеми (легкі підзадачі) [1].

Бібліографія

[1] A. Zai and B. Brown, *Deep Reinforcement Learning in Action*. Manning Publications Co., 2020.