МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

ЛЕКЦІЯ 7. Навчання з підкріпленням

Львів -- 2025

Лекція зі штучного інтелекту 2025-07

Вступ

На цьому занятті ми розглянемо навчання з підкріпленням — потужну парадигму машинного навчання, яка дозволяє агентам навчатися оптимальної поведінки через взаємодію з середовищем. На відміну від навчання з учителем та без учителя, при навчанні з підкріпленням агент отримує зворотний зв'язок у вигляді винагород і намагається максимізувати сумарну винагороду з часом. Ми розглянемо основні концепції, алгоритми та застосування цього підходу, а також його інтеграцію з глибоким навчанням — так зване глибоке навчання з підкріпленням (Deep Reinforcement Learning).

Теми, що розглядаються

1. Концепція навчання з підкріпленням та його історія
2. Основні компоненти та термінологія
3. Марковські процеси прийняття рішень (MDP)
4. Функції цінності та оптимальність
5. Динамічне програмування
6. Методи Монте-Карло
7. Часткова різниця (Temporal Difference) навчання
8. Q-навчання та SARSA
9. Функціональна апроксимація
10. Глибоке навчання з підкріпленням
11. Алгоритм DQN
12. Методи градієнта стратегії
13. Акторно-критичні методи
14. Сучасні алгоритми: PPO, SAC, TD3
15. Застосування навчання з підкріпленням

Концепція навчання з підкріпленням та його історія

Основна ідея навчання з підкріпленням

Навчання з підкріпленням (Reinforcement Learning, RL) — це область машинного навчання, в якій агент навчається приймати рішення, взаємодіючи з середовищем. Агент виконує дії, отримує зворотний зв'язок у вигляді винагороди та нового стану середовища, і адаптує свою поведінку, щоб максимізувати сумарну винагороду.

Ключові відмінності від інших парадигм машинного навчання:

- **Навчання з учителем**: навчання на основі маркованих прикладів
- **Навчання без учителя**: пошук структури в немаркованих даних

- **Навчання з підкріпленням**: навчання через досвід взаємодії та винагороду

Історія навчання з підкріпленням

1. **Ранні витоки (1950-1970-ті)**:
    - Дослідження в психології (Б.Ф. Скіннер) про навчання через позитивне та негативне підкріплення
    - Концепція динамічного програмування Річарда Беллмана (1957)
    - Марковські процеси прийняття рішень як математична основа
2. **Формування як окремої галузі (1980-1990-ті)**:
    - Робота Гаррі Кіл'юпіса про часткову різницю (1988)
    - Алгоритм Q-навчання, розроблений Крісом Воткінсом (1989)
    - Книга Річарда Саттона і Ендрю Барто "Reinforcement Learning: An Introduction" (1998)
3. **Інтеграція з нейронними мережами (2010-ті)**:
    - DQN (Deep Q-Network) від DeepMind (2013) — поєднання Q-навчання з глибокими нейронними мережами
    - AlphaGo (2016) — перемога над чемпіоном світу з го
    - Прорив у робототехніці, автономних транспортних засобах та ігрових агентах
4. **Сучасний стан (2020-ті)**:
    - Масштабні моделі та ефективні алгоритми (PPO, SAC)
    - Інтеграція з великими мовними моделями
    - Застосування в різноманітних галузях: від фінансів до медицини

Парадигма проб і помилок

Центральна концепція навчання з підкріпленням — навчання через проби і помилки:

- Агент експериментує з різними діями
- Отримує зворотний зв'язок про їх результативність
- Поступово виявляє дії, що призводять до найвищої винагороди
- Баланс між дослідженням (exploration) нових стратегій та експлуатацією (exploitation) відомих ефективних стратегій

Основні компоненти та термінологія

Ключові елементи системи навчання з підкріпленням

1. **Агент**: сутність, яка приймає рішення і навчається
2. **Середовище**: світ, з яким взаємодіє агент
3. **Стан** (s): представлення ситуації в середовищі
4. **Дія** (a): вибір, який агент може зробити

5. **Винагорода** (r): числове значення, що вказує на бажаність переходу з поточного стану в наступний
6. **Політика/стратегія** (π): стратегія прийняття рішень агентом
7. **Функція цінності** (V або Q): очікувана сумарна винагорода, починаючи з певного стану

Взаємодія агент-середовище

Процес взаємодії відбувається покроково:

1. Агент спостерігає поточний стан s_t
2. На основі стану агент вибирає дію a_t згідно своєї стратегії π
3. Середовище змінюється до нового стану s_{t+1}
4. Агент отримує винагороду r_t
5. Процес повторюється до термінального стану або нескінченно

Типи задач навчання з підкріпленням

1. **За часовою структурою**:
   - **Епізодичні задачі**: мають чіткий початок і кінець (наприклад, партія в шахи)
   - **Неперервні задачі**: не мають природного завершення (наприклад, постійне управління процесом)
2. **За повнотою спостереження**:
   - **Повністю спостережувані**: агент має повну інформацію про стан середовища
   - **Частково спостережувані**: агент має лише часткову інформацію про стан
3. **За кількістю агентів**:
   - **Одноагентні**: один агент взаємодіє із середовищем
   - **Багатоагентні**: кілька агентів взаємодіють між собою та із середовищем

Ціль навчання з підкріпленням

Основна мета агента — максимізувати очікувану сумарну дисконтовану винагороду:

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

де:

- G_t — сумарна винагорода з моменту t
- γ — коефіцієнт дисконтування ($0 \leq \gamma \leq 1$)
- R_{t+k+1} — винагорода, отримана на кроці $t+k+1$

Коефіцієнт дисконтування γ визначає відносну важливість негайних та майбутніх винагород:

- $\gamma = 0$: агент враховує лише негайну винагороду
- γ близько до 1: агент враховує довгострокові винагороди майже так само, як негайні

Марковські процеси прийняття рішень (MDP)

Визначення MDP

Марковський процес прийняття рішень (Markov Decision Process, MDP) — це математична основа для формалізації задач навчання з підкріпленням. MDP визначається кортежем з п'яти елементів:

$$MDP = (S, A, P, R, \gamma)$$

де:

- S — скінченна множина станів
- A — скінченна множина дій
- P — функція ймовірності переходу: $P(s'|s,a)$ — ймовірність переходу до стану s' зі стану s при виконанні дії a
- R — функція винагороди: $R(s,a,s')$ — винагорода за перехід зі стану s до стану s' при виконанні дії a
- γ — коефіцієнт дисконтування: 0 ≤ γ ≤ 1

Властивість Маркова

Ключова властивість MDP — марковська властивість, яка стверджує, що майбутнє залежить від минулого тільки через поточний стан:

$$P(s_{t+1}, r_t | s_0, a_0, r_0, ..., s_t, a_t) = P(s_{t+1}, r_t | s_t, a_t)$$

Іншими словами, поточний стан містить всю інформацію, необхідну для прийняття оптимального рішення, і історія попередніх станів та дій не потрібна.

Типи MDP

1. **Скінченні MDP**: кількість станів і дій є скінченною
2. **Нескінченні MDP**: кількість станів і/або дій є нескінченною (зокрема, для неперервних просторів станів/дій)
3. **Детерміновані MDP**: перехід до наступного стану є детермінованим (ймовірність переходу дорівнює 1)
4. **Стохастичні MDP**: переходи є ймовірнісними

Політики в MDP

Політика (стратегія) π визначає поведінку агента. Вона може бути:

1. **Детермінованою**: $\pi(s) = a$ — для кожного стану s однозначно визначає дію a
2. **Стохастичною**: $\pi(a|s) = P(A_t = a | S_t = s)$ — визначає розподіл ймовірностей вибору дій у кожному стані

Оптимальна політика π^* — це політика, яка максимізує очікувану сумарну дисконтовану винагороду для всіх станів.

Функції цінності та оптимальність

Функції цінності

Функції цінності оцінюють "якість" станів або пар стан-дія з точки зору очікуваної майбутньої винагороди.

1. **Функція цінності стану** (V-функція):

   $$V^\pi(s) = \mathbb{E}*pi*left[*sum*{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$$

   Очікувана сумарна дисконтована винагорода, починаючи зі стану s і діючи згідно з політикою π.

2. **Функція цінності дії** (Q-функція):

   $$Q^\pi(s, a) = \mathbb{E}*pi*left[*sum*{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$$

   Очікувана сумарна дисконтована винагорода, починаючи зі стану s, виконуючи дію a, а потім діючи згідно з політикою π.

Зв'язок між функціями цінності

V-функція і Q-функція пов'язані співвідношеннями:

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) Q^\pi(s,a)$$

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s')$$

Рівняння Беллмана

Рівняння Беллмана виражають рекурсивний характер функцій цінності:

1. **Рівняння Беллмана для V^π**:

   $$V^\pi(s) = \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s') \right)$$

2. **Рівняння Беллмана для Q^π**:

   $$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q^\pi(s',a')$$

Оптимальні функції цінності

Оптимальні функції цінності визначають максимально можливу очікувану винагороду:

1. **Оптимальна функція цінності стану**:

   $$V^*(s) = \max_\pi V^\pi(s) = \max_a Q^*(s,a)$$

2. **Оптимальна функція цінності дії**:

   $$Q^*(s,a) = \max_\pi Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s')$$

Рівняння оптимальності Беллмана

Рівняння оптимальності Беллмана визначають оптимальні функції цінності:

1. **Рівняння оптимальності Беллмана для V^***:

   $$V^(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^(s') \right)$$

2. **Рівняння оптимальності Беллмана для Q^***:

   $$Q^(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q^(s',a')$$

Оптимальна політика

Оптимальна політика π^* може бути безпосередньо виведена з оптимальної функції цінності:

$$\pi^(s) = \arg\max_a Q^(s,a)$$

Важливо зазначити, що:

- Може існувати більше однієї оптимальної політики
- Всі оптимальні політики мають однакові функції цінності $V^$ та $Q^$
- Хоча би одна з оптимальних політик є детермінованою

Динамічне програмування

Концепція динамічного програмування

Динамічне програмування (DP) — це набір алгоритмів для обчислення оптимальних політик, якщо відома повна модель MDP (функції переходів та винагород).

Ключові характеристики:

- Вимагає повної моделі середовища
- Обчислювально ефективне, але обмежене для великих просторів станів
- Є теоретичною основою для багатьох методів навчання з підкріпленням

Оцінка політики (Policy Evaluation)

Алгоритм для обчислення функції цінності V^π для заданої політики π:

1. Ініціалізувати $V(s) = 0$ для всіх $s \in S$
2. Повторювати до збіжності:

   $$V(s) \leftarrow \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right)$$

Покращення політики (Policy Improvement)

Техніка для покращення політики на основі поточної функції цінності:

$$\pi'(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s') \right)$$

Порівняння ітерації політики та ітерації цінності

Ітерація політики та ітерація цінності мають ключові відмінності:

1. **Структура алгоритму**:
    - Ітерація політики чітко розділяє етапи оцінки та покращення політики
    - Ітерація цінності поєднує ці етапи в одному кроці оновлення
2. **Обчислювальна ефективність**:
    - Ітерація політики вимагає повної оцінки політики на кожній ітерації
    - Ітерація цінності ефективніша, оскільки не чекає повної збіжності оцінки
3. **Проміжні результати**:
    - При ітерації політики проміжні політики є повністю визначеними та можуть використовуватися
    - При ітерації цінності проміжні значення функції цінності не відповідають жодній конкретній політиці
4. **Швидкість збіжності**:
    - Ітерація цінності зазвичай збігається швидше
    - Ітерація політики може потребувати менше ітерацій, але кожна ітерація обчислювально дорожча
5. **Застосування**:
    - Ітерація політики краще підходить, коли важлива інтерпретованість проміжних результатів
    - Ітерація цінності ефективніша для швидкого знаходження оптимального рішення

Порівняльна таблиця методів динамічного програмування

Характеристика	Ітерація цінності	Ітерація політики
Підхід	Ітеративно оновлює функцію цінності до збіжності.	Чергує етапи оцінки політики та покращення політики.
Збіжність	Збігається, коли функція цінності збігається.	Збігається, коли політика перестає змінюватися.
Обчислювальна вартість	Більш обчислювально затратний на кожній ітерації через повну оцінку всіх станів.	Потребує більше ітерацій, але може збігатися швидше з точки зору меншої кількості ітерацій.
Вихідна політика	Політика виводиться після збіжності функції цінності.	Політика оновлюється під час кожної ітерації.

Характеристика	Ітерація цінності	Ітерація політики
Швидкість збіжності	Може вимагати багато ітерацій для збіжності, особливо у великих просторах станів.	Зазвичай збігається швидше на практиці, особливо коли політика значно покращується на кожній ітерації.
Простір станів	Зазвичай підходить для менших просторів станів через обчислювальну складність.	Може ефективніше обробляти більші простори станів.

Ітерація політики (Policy Iteration)

Алгоритм, що поєднує оцінку та покращення політики:

1. Ініціалізувати π довільним чином
2. **Оцінка політики**: Обчислити V^π для поточної політики
3. **Покращення політики**: $\pi'(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s') \right)$
4. Якщо $\pi' = \pi$, то зупинитись; інакше $\pi \leftarrow \pi'$ і повернутись до кроку 2

Ітерація цінності (Value Iteration)

Ефективніший алгоритм, який поєднує оцінку та покращення в одному кроці:

1. Ініціалізувати $V(s) = 0$ для всіх $s \in S$
2. Повторювати до збіжності:

   $$V(s) \leftarrow \max_a \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right)$$

3. Визначити оптимальну політику: $\pi(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right)$

Узагальнене ітераційне програмування

Загальна парадигма методів навчання з підкріпленням:

- Ітеративне наближення до оптимальних функцій цінності та політики
- Чергування оцінки та покращення (явно чи неявно)
- Асимптотична збіжність до оптимального рішення

Методи Монте-Карло

Навчання без моделі

Методи Монте-Карло (MC) належать до класу методів, які не вимагають знання моделі середовища (P і R). Замість цього вони:

- Навчаються безпосередньо з досвіду (епізодів взаємодії)

- Оцінюють функції цінності на основі вибірок повних траєкторій
- Здатні навчатися з фактичних чи симульованих епізодів

Оцінка функції цінності методом Монте-Карло

Метод MC оцінює функцію цінності стану $V(s)$ як середню повну винагороду, отриману після відвідування стану s:

1. Генерувати епізоди, дотримуючись політики π
2. Для кожного стану s в епізоді:
    - Визначити повну винагороду G_t з цього стану до кінця епізоду
    - Оновити оцінку $V(s)$ як середнє всіх попередніх винагород

$$V(s) \leftarrow V(s) + \alpha [G_t - V(s)]$$

де α — швидкість навчання.

Типи методів Монте-Карло

1. **Перший візит MC**: оновлювати $V(s)$ тільки при першому відвідуванні стану s в епізоді
2. **Кожен візит MC**: оновлювати $V(s)$ при кожному відвідуванні стану s в епізоді

Монте-Карло для функцій цінності дій

Для оцінки $Q(s,a)$ потрібно відвідувати пари стан-дія:

$$Q(s,a) \leftarrow Q(s,a) + \alpha [G_t - Q(s,a)]$$

Проблема дослідження

Для навчання оптимальної політики, агент повинен випробувати всі можливі пари стан-дія, але політика π може ніколи не вибрати деякі дії.

Рішення:

1. **Дослідження з початковим бонусом**: почати з оптимістичної оцінки цінності
2. **Неперервне дослідження**: забезпечити ненульову ймовірність вибору будь-якої дії (ε-жадібна політика)

Монте-Карло без модельного контролю

1. **MC з дослідженням**: Використання ε-жадібної політики для забезпечення дослідження
2. **Поза політикою MC**: Навчання оптимальної політики, слідуючи іншій (дослідницькій) політиці

Загальний алгоритм:

1. Ініціалізувати $Q(s,a)$ довільним чином
2. Повторювати:
    - Генерувати епізод, використовуючи ε-жадібну політику на основі Q
    - Для кожної пари (s,a) в епізоді:

        - $G \leftarrow$ повна винагорода після (s,a)
        - $Q(s,a) \leftarrow Q(s,a) + \alpha [G - Q(s,a)]$
    - Покращувати політику: $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Переваги та недоліки методів Монте-Карло

Переваги:

- Не вимагають знання моделі середовища
- Можуть навчатися з фактичного досвіду
- Не страждають від систематичної помилки через неточну модель
- Ефективні для епізодичних задач

Недоліки:

- Вимагають повних епізодів (не підходять для неперервних задач)
- Висока дисперсія оцінок через використання повних траєкторій
- Повільна збіжність порівняно з методами, що використовують бутстрепінг

Часткова різниця (Temporal Difference) навчання

Концепція часткової різниці

Часткова різниця (TD) навчання поєднує ідеї методів Монте-Карло та динамічного програмування:

- Навчається з досвіду як методи MC (не вимагає моделі)
- Використовує бутстрепінг як DP (оновлює оцінки на основі інших оцінок)
- Може навчатися крок за кроком, без очікування кінця епізоду

TD(0) для оцінки стану

Алгоритм TD(0) оновлює функцію цінності стану на основі часткової різниці:

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

де $r_{t+1} + \gamma V(s_{t+1})$ — TD-ціль, а $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ — TD-помилка.

Порівняння з Монте-Карло

- **MC**: $V(s_t) \leftarrow V(s_t) + \alpha [G_t - V(s_t)]$ (фактична повна винагорода)
- **TD**: $V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$ (бутстрепінг)

Відмінності:

- TD не чекає кінця епізоду
- TD має меншу дисперсію, але схильне до зміщення
- TD ефективніше використовує досвід

n-крокові методи TD

Узагальнення TD(0) з використанням n-крокових повернень:

$$V(s_t) \leftarrow V(s_t) + \alpha [G_{t:t+n} - V(s_t)]$$

де $G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$

Це створює спектр між TD(0) (n=1) і MC (n=∞).

TD(λ) та сліди придатності

TD(λ) поєднує повернення різної довжини з експоненційним зваженням:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$$

Реалізується через сліди придатності (eligibility traces) — механізм, який відстежує, які стани нещодавно відвідувалися:

$$E_0(s) = 0$$ $$E_t(s) = \gamma \lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$ $$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

де $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ — TD-помилка.

Q-навчання та SARSA

Q-навчання (Q-Learning)

Q-навчання — це позаполітичний (off-policy) TD алгоритм, який безпосередньо оцінює оптимальну функцію цінності дій Q^*:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Основні характеристики:

- Навчається оптимальної стратегії, слідуючи дослідницькій стратегії
- Збігається до оптимальної функції цінності дій незалежно від політики, якій слідує
- Тенденція до переоцінки через операцію max

SARSA

SARSA (State-Action-Reward-State-Action) — це внутрішньополітичний (on-policy) TD алгоритм:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Основні характеристики:

- Навчається цінності поточної політики (зазвичай ε-жадібної)
- Збігається до функції цінності політики, якій слідує
- Більш консервативний підхід порівняно з Q-навчанням

Порівняння Q-навчання та SARSA

- **Q-навчання** (off-policy):
  - Оцінює оптимальну політику, слідуючи іншій
  - Ціль: $r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a)$
  - Більш ризикований, може призводити до оптимальніших стратегій
  - Може бути нестабільним під час навчання
- **SARSA** (on-policy):
  - Навчається цінності політики, якій фактично слідує
  - Ціль: $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$
  - Більш безпечний, враховує дослідницьку поведінку
  - Стабільніший під час навчання

n-крокове SARSA та Q(λ)

Розширення базових алгоритмів з використанням n-кроків та слідів придатності:

- **n-крокове SARSA**: $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [G_{t:t+n} - Q(s_t, a_t)]$$

  де $G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n Q(s_{t+n}, a_{t+n})$
- **SARSA(λ)** і **Q(λ)**: використовують сліди придатності для зваженого усереднення повернень різної довжини

Double Q-навчання

Вирішує проблему переоцінки в стандартному Q-навчанні, використовуючи дві функції цінності:

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha [r_{t+1} + \gamma Q_2(s_{t+1}, \arg\max_a Q_1(s_{t+1}, a)) - Q_1(s_t, a_t)]$$

Аналогічно оновлюється Q_2 з використанням Q_1.

Функціональна апроксимація

Обмеження табличних методів

До цього моменту ми розглядали табличні методи, які зберігають окремі значення для кожного стану або пари стан-дія. Це має серйозні обмеження:

- Непрактично для великих просторів станів/дій
- Неможливо для неперервних просторів
- Не узагальнюється на нові стани

Функціональна апроксимація в RL

Функціональна апроксимація замінює таблицю параметризованою функцією:

- $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$ або $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$
- $\mathbf{w}$ — вектор параметрів
- Узагальнення досвіду між схожими станами

Типи функціональних апроксиматорів

1. **Лінійні моделі**: $$\hat{v}(s, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s)$$ де $\mathbf{x}(s)$ — вектор ознак стану s
2. **Нейронні мережі**: $$\hat{v}(s, \mathbf{w}) = f(\mathbf{w}, s)$$ де f — нелінійна функція (нейронна мережа)
3. **Інші апроксиматори**:
    - Дерева рішень
    - Радіальні базисні функції
    - Kernel-методи

Навчання з градієнтним спуском

Оновлення параметрів для мінімізації помилки:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

де $\mathcal{L}(\mathbf{w})$ — функція втрат, наприклад, середньоквадратична помилка:

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}*pi[(v*pi(s) - \hat{v}(s, \mathbf{w}))^2]$$

Напівградієнтні методи TD

Для TD-навчання з функціональною апроксимацією:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [r_{t+1} + \gamma \hat{v}(s_{t+1}, \mathbf{w}) - \hat{v}(s_t, \mathbf{w})] \nabla_{\mathbf{w}} \hat{v}(s_t, \mathbf{w})$$

Це називається напівградієнтним методом, оскільки $\hat{v}(s_{t+1}, \mathbf{w})$ також залежить від $\mathbf{w}$, але ця залежність ігнорується.

Апроксимація Q-функції

Для Q-навчання з функціональною апроксимацією:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [r_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w})] \nabla_{\mathbf{w}} \hat{q}(s_t, a_t, \mathbf{w})$$

Для SARSA:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{q}(s_t, a_t, \mathbf{w})] \nabla_{\mathbf{w}} \hat{q}(s_t, a_t, \mathbf{w})$$

Проблеми збіжності

Функціональна апроксимація може призводити до нестабільності:

- Нелінійні апроксиматори (нейронні мережі) особливо схильні до дивергенції
- Корелація у даних та бутстрепінг ускладнюють навчання
- Навіть лінійні апроксиматори не гарантують збіжність для off-policy методів

Глибоке навчання з підкріпленням

Інтеграція глибокого навчання та RL

Глибоке навчання з підкріпленням (Deep Reinforcement Learning, DRL) поєднує:

- Репрезентативну силу глибоких нейронних мереж
- Можливості навчання з підкріплення

Це дозволяє:

- Працювати з високорозмірними просторами станів (зображення, звук, тощо)
- Автоматично вивчати релевантні ознаки
- Масштабувати RL до складних задач реального світу

Проблеми DRL

Основні виклики:

1. **Нестабільність навчання**: через нестаціонарність даних та корельовані вибірки
2. **Висока дисперсія**: сигнал винагороди може бути зашумленим
3. **Неефективність вибірок**: RL вимагає багато взаємодій із середовищем
4. **Баланс дослідження-експлуатації**: особливо критичний для нейромережевих апроксиматорів

Алгоритм DQN

Deep Q-Network (DQN)

DQN — проривний алгоритм, розроблений DeepMind, який успішно застосував глибоке навчання для Q-навчання:

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Основні інновації:

1. **Буфер повторів досвіду** (Experience Replay): зберігання переходів $(s_t, a_t, r_{t+1}, s_{t+1})$ та навчання на випадкових міні-пакетах
2. **Цільова мережа** (Target Network): окрема мережа з замороженими параметрами θ^- для обчислення цілей
3. **Згорткові нейронні мережі** для обробки візуальних вхідних даних

Алгоритм DQN

1. Ініціалізувати буфер повторів $\mathcal{D}$
2. Ініціалізувати Q-мережу з випадковими вагами θ
3. Ініціалізувати цільову мережу з $\theta^- = \theta$
4. Для кожного епізоду: a. Ініціалізувати початковий стан s_1 b. Для $t = 1, T$:
   - Вибрати a_t з політики ϵ-жадібності на основі $Q(s_t, a; \theta)$
   - Виконати дію a_t, отримати r_{t+1} та s_{t+1}
   - Зберегти перехід $(s_t, a_t, r_{t+1}, s_{t+1})$ в $\mathcal{D}$
   - Вибрати випадковий міні-пакет переходів $(s_j, a_j, r_{j+1}, s_{j+1})$ з $\mathcal{D}$
   - Встановити $y_j = r_{j+1} + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^-)$
   - Оновити θ градієнтним спуском для мінімізації $(y_j - Q(s_j, a_j; \theta))^2$
   - Кожні C кроків оновити $\theta^- = \theta$

Розширення DQN

1. **Double DQN**: вирішує проблему переоцінки, використовуючи основну мережу для вибору дії та цільову для оцінки:

   $$y_j = r_{j+1} + \gamma Q(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a'; \theta); \theta^-)$$

2. **Dueling DQN**: розділяє Q-функцію на функцію цінності стану та перевагу дії:

   $$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) - \frac{1}{|A|}\sum_{a'} A(s, a'; \theta, \alpha)$$

3. **Prioritized Experience Replay**: вибір переходів з більшою TD-помилкою з вищою ймовірністю

4. **Noisy DQN**: додавання шумових шарів для покращення дослідження

Rainbow DQN

Rainbow об'єднує шість розширень DQN:

- Double Q-навчання
- Prioritized Experience Replay
- Dueling архітектура
- Multi-step навчання
- Distributional RL (моделювання розподілу винагороди)
- Noisy Networks

Комбінований підхід показує значне покращення порівняно з базовим DQN.

Методи градієнта стратегії

Прямий пошук стратегії

На відміну від методів на основі цінності, які навчають функцію цінності та виводять стратегію, методи градієнта стратегії:

- Безпосередньо параметризують стратегію $\pi(a|s; \theta)$
- Оптимізують очікувану винагороду шляхом градієнтного підйому
- Особливо ефективні для неперервних просторів дій

Теорема про градієнт стратегії

Градієнт очікуваної винагороди щодо параметрів стратегії:

$$\nabla_\theta J(\theta) = \mathbb{E}{\pi\theta} \left[\nabla_\theta \log \pi(a|s; \theta) \cdot Q^{\pi_\theta}(s, a) \right]$$

Це дозволяє оновлювати параметри для максимізації очікуваної винагороди:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

REINFORCE

Базовий алгоритм градієнта стратегії:

1. Генерувати епізод $s_0, a_0, r_1, s_1, ..., s_{T-1}, a_{T-1}, r_T$ за $\pi(\cdot|\cdot; \theta)$
2. Для кожного кроку t в епізоді:
   - Обчислити повну винагороду: $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_{k+1}$
   - Оновити параметри: $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \log \pi(a_t|s_t; \theta)$

REINFORCE з базовою лінією

Для зменшення дисперсії градієнта, можна відняти базову лінію:

$$\nabla_\theta J(\theta) = \mathbb{E}{\pi\theta} \left[\nabla_\theta \log \pi(a|s; \theta) \cdot (Q^{\pi_\theta}(s, a) - b(s)) \right]$$

Часто базовою лінією є функція цінності стану $V^{\pi_\theta}(s)$:

$$\theta \leftarrow \theta + \alpha \gamma^t (G_t - V(s_t)) \nabla_\theta \log \pi(a_t|s_t; \theta)$$

Ключові переваги методів градієнта стратегії

1. Можуть навчати стохастичні політики
2. Ефективні для високорозмірних та неперервних просторів дій
3. Часто стабільніші за методи, базовані на цінності
4. Можуть оптимізувати політику безпосередньо без обчислення функції цінності дій

Акторно-критичні методи

Концепція актора та критика

Акторно-критичні методи (Actor-Critic) поєднують підходи градієнта стратегії та методи на основі цінності:

- **Актор**: нейронна мережа, що параметризує стратегію $\pi(a|s; \theta)$
- **Критик**: нейронна мережа, що оцінює функцію цінності $V(s; w)$ або $Q(s, a; w)$

Актор вибирає дії, а критик оцінює їх якість та забезпечує низькодисперсійний сигнал для навчання актора.

Алгоритм Advantage Actor-Critic (A2C)

A2C використовує перевагу (advantage) для навчання актора:

$$A(s, a) = Q(s, a) - V(s)$$

Алгоритм:

1. Стан s_t, нейронні мережі параметризовані θ та w
2. Актор вибирає дію $a_t \sim \pi(a|s_t; \theta)$
3. Отримати винагороду r_{t+1} та новий стан s_{t+1}
4. Обчислити TD-помилку: $\delta_t = r_{t+1} + \gamma V(s_{t+1}; w) - V(s_t; w)$
5. Оновити параметри критика: $w \leftarrow w + \alpha_w \delta_t \nabla_w V(s_t; w)$
6. Оновити параметри актора: $\theta \leftarrow \theta + \alpha_\theta \delta_t \nabla_\theta \log \pi(a_t|s_t; \theta)$

Asynchronous Advantage Actor-Critic (A3C)

A3C розширює A2C для асинхронного навчання з використанням кількох паралельних агентів:

- Кожен агент взаємодіє з власною копією середовища
- Періодично агенти синхронізують параметри з глобальною моделлю
- Прискорює та стабілізує навчання через декореляцію досвіду

Actor-Critic з досвідом

Інші варіації включають:

- **ACER** (Actor-Critic with Experience Replay): використовує буфер повторів для ефективнішого використання даних
- **ACKTR** (Actor-Critic using Kronecker-Factored Trust Region): використовує природний градієнтний спуск для ефективнішої оптимізації

Сучасні алгоритми: PPO, SAC, TD3

Proximal Policy Optimization (PPO)

PPO — один з найпопулярніших сучасних алгоритмів RL, розроблений OpenAI:

1. **Обмежена оптимізація**: запобігає надто великим оновленням політики

2. **Проста реалізація**: легший у налаштуванні порівняно з методами довірчої області (TRPO)
3. **Хороша продуктивність**: хороший баланс між простотою, стабільністю та ефективністю вибірок

Цільова функція PPO з обрізанням:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t) \right]$$

де $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ — відношення ймовірностей.

Soft Actor-Critic (SAC)

SAC — off-policy алгоритм актор-критик, який максимізує і очікувану винагороду, і ентропію:

1. **Максимізація ентропії**: заохочує дослідження та робастність
2. **Off-policy навчання**: ефективно використовує дані за домогою буфера повторів
3. **Стохастична політика**: підтримує розподіл над діями, а не точкові оцінки

Цільова функція SAC:

$$J(\theta) = \mathbb{E}_{s_t \sim D, a_t \sim \pi_\theta} \left[Q_\phi(s_t, a_t) - \alpha \log \pi_\theta(a_t|s_t) \right]$$

де α — температурний параметр, що контролює відносну важливість ентропії.

Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 — вдосконалення DDPG для неперервних просторів дій:

1. **Подвійне Q-навчання**: використовує два критики для зменшення переоцінки
2. **Затримані оновлення політики**: оновлює політику рідше, ніж критиків
3. **Шум цілей**: додає шум до цільових дій для згладжування функції цінності

$$y = r + \gamma \min_{i=1,2} Q_{\phi'_i}(s', \pi_{\theta'}(s') + \epsilon)$$

де ϵ — шум дії для згладжування.

Порівняння алгоритмів

- **PPO**: On-policy, простий, надійний, але менш ефективний за використанням вибірок
- **SAC**: Off-policy, ефективний за використанням вибірок, хороший для дослідження, але складніший у налаштуванні
- **TD3**: Off-policy, ефективний для неперервних просторів дій, але можливі проблеми зі стохастичними середовищами

Застосування навчання з підкріпленням

Ігри та розваги

- **Настільні ігри**: AlphaGo, AlphaZero (го, шахи, сьогі)

- **Відеоігри**: DQN для Atari, AlphaStar для StarCraft II
- **Карткові ігри**: Pluribus для покеру з кількома гравцями

Робототехніка та управління

- **Маніпуляція об'єктами**: захоплення, сортування, збірка
- **Локомоція**: ходьба, біг, стрибки для роботів-гуманоїдів та чотириногих роботів
- **Автономне керування**: навігація, планування руху, уникнення перешкод

Комп'ютерні системи

- **Оптимізація ресурсів**: управління пам'яттю, розподіл обчислювальних ресурсів
- **Мережева маршрутизація**: адаптивна маршрутизація пакетів
- **Енергоефективність**: оптимізація споживання енергії в центрах обробки даних

Фінанси

- **Торгові стратегії**: автоматизована торгівля на фінансових ринках
- **Управління портфелем**: динамічний розподіл активів
- **Ризик-менеджмент**: оптимізація стратегій хеджування

Медицина та охорона здоров'я

- **Персоналізована медицина**: адаптивні схеми лікування
- **Оптимізація дозування**: індивідуалізація дозування ліків
- **Медична візуалізація**: навігація інструментів і виявлення аномалій

Природні науки

- **Молекулярний дизайн**: оптимізація хімічних структур
- **Експериментальний дизайн**: оптимізація параметрів експериментів
- **Квантова фізика**: контроль квантових систем

Висновки

Навчання з підкріпленням є потужною парадигмою, яка дозволяє створювати агенти, здатні навчатися та адаптуватися через взаємодію з середовищем. Від класичних алгоритмів до сучасних методів глибокого навчання з підкріпленням, ця галузь пропонує широкий спектр підходів до вирішення складних проблем прийняття рішень.

Ключові переваги:

- Автономне навчання через досвід
- Можливість вирішення задач з відкладеною винагородою
- Адаптивність до динамічних середовищ
- Здатність знаходити неінтуїтивні стратегії

Виклики та напрямки розвитку:

- Покращення ефективності використання вибірок
- Масштабування до складніших середовищ
- Інтеграція з іншими підходами ШІ, зокрема з мовними моделями
- Пояснюваність та безпека систем RL

Навчання з підкріпленням продовжує бути одним із найдинамічніших напрямків штучного інтелекту з багатообіцяючими перспективами для вирішення широкого спектру практичних задач.

Література

1. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT Press.
2. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.
3. Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.
4. Schulman, J., et al. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
5. Haarnoja, T., et al. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290.
6. Fujimoto, S., et al. (2018). Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477.