МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ІВАНЬКОВ ДМИТРО ВАДИМОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
к. т. н., доцент
__________ О. В. Зелінська
«____»___________ 20__ р.

ДОСЛІДЖЕННЯ МЕТОДІВ СТАТИСТИЧНОГО НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ ПОШУКУ ШЛЯХУ У ВІРТУАЛЬНОМУ СВІТІ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Є. Є. Федоров, професор кафедри
інформаційних технологій,
д. т. н., професор
____

Оцінка: ______ / ______ / _________
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: ________________

Вінниця — 2024

АНОТАЦІЯ

Іваньков Д.В. Дослідження методів статистичного навчання з підкріпленням для пошуку шляху у віртуальному світі. Спеціальність 122 «Комп'ютерні науки», Освітня програма «Комп'ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, Вінниця, 2023.

У кваліфікаційній роботі досліджено: сучасний стан розвитку алгоритмів машинного навчання. Показано: практичне застосування алгоритмів машинного для навчання агента на знаходження найкоротшого шляху. Встановлено: найкращий алгоритм навчання з підкріпленням для пошуку шляху у віртуальному світі.

Ключові слова: агент, машинне навчання, Policy Iteration, DQN, SARSA, Q-навчання, Python.

86 ст., 1 табл., 18 рис., 40 джерел.

Ivankov D.V. Research of statistical reinforcement learning methods for wayfinding in a virtual world. Specialty 122 «Computer Science», Programme «Computer data processing technologies». Vasyl' Stus Donetsk National University, Vinnytsia, 2023.

The qualification work investigates: the current state of development of machine learning algorithms. Shown: the practical application of machine learning algorithms for training an agent to find the shortest path. Established: the best reinforcement learning algorithm for finding a path in the virtual world.

Keywords: agent, machine learning, Policy Iteration, DQN, SARSA, Q-learning, Python.

Page 86, 1 Tabl., 18 Fig., 40 Src.

Зміст

ВСТУП ... 3
РОЗДІЛ 1 СУЧАСНИЙ СТАН РОЗВИТКУ МАШИННОГО НАВЧАННЯ ... 6
1.1 Розвиток і впровадження машинного навчання ... 6
1.2 Методи машинного навчання ... 8
1.3 Нейронні мережі і глибоке навчання ... 10
1.4 Машинне навчання для бізнесу ... 11
1.5 Приклади реалізації ... 12
1.6 Неконтрольоване машинне навчання ... 14
1.7 Навчання під наглядом ... 20
1.8 Навчання без нагляду ... 22
1.9 Навчання з підкріпленням ... 24
1.10 Аналіз відмінностей між контрольованим і неконтрольованим навчанням ... 27
Висновки до розділу 1 ... 30
РОЗДІЛ 2 АЛГОРИТМИ НАВЧАННЯ І ПОШУКУ ШЛЯХУ ... 31
2.1 Класифікація методів навчання з підкріпленням ... 31
2.2 Приклади методів навчання з підкріпленням ... 34
DP (Policy Iteration) ... 34
DQN ... 37
SARSA ... 44
2.3 Алгоритми створення найкоротших шляхів ... 49
Висновки до розділу 2 ... 60
РОЗДІЛ 3 РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМИ ДЛЯ ПОШУКУ ШЛЯХУ У ВІРТУАЛЬНОМУ СВІТІ ... 61
3.1 Принцип використання машинного навчання для пошуку шляху ... 61
3.2 Використання алгоритму Q-навчання для матриці ... 64
3.3 Реалізація програми для пошуку найкоротшого шляху у віртуальному світі ... 69
3.4 Тестування програми пошуку шляху ... 79
Висновки до розділу 3 ... 81
ВИСНОВКИ ... 82
Список використаних джерел ... 83

ВСТУП

На сьогодні у сфері сучасного штучного інтелекту (ШІ) навчання з підкріпленням (RL) є однією з найактуальніших тем. Розробники, що виконують роботу в галузі штучного інтелекту та машинного навчання (ML), крім інших тем, використовують практики навчання з підкріпленням, щоб додати інтелектуальну складову в інструменти та програми, які вони створюють.

В основі кожного продукту де використаний штучний інтелект лежить машинне навчання і є його основним принципом. Розробники використовують різні методології машинного навчання для навчання своїх інтелектуальних додатків, ігор та інших застосувань. Машинне навчання (ML) є дуже різноманітною галуззю, і різні групи розробників пропонують нові методи для навчання машин.

Один із прибуткових підходів до ML - це глибоке навчання з підкріпленням. В цьому методі небажана поведінка машини карається, а бажані дії винагороджуються, що дозволяє розумним машинам навчатися на власних помилках. Експерти вважають, що цей метод машинного навчання дійсно допомагає штучній інтелекту вдосконалюватися за допомогою власного досвіду.

Актуальність теми.

Спільно з розвитком технологій зберігання та обробки даних зросла доступність великого обсягу даних. Машинне навчання дозволяє видобувати корисну інформацію із цих даних та приймати відповідні рішення.

Постійний розвиток алгоритмів машинного навчання дозволяє створювати більш точні моделі, які можуть вирішувати складні завдання.

Машинне навчання забезпечує можливість автоматизувати процеси та оптимізувати прийняття рішень у багатьох сферах, включаючи бізнес, медицину, фінанси, транспорт, науку тощо.

Завдяки розвитку NLP-моделей, які здатні розуміти та генерувати природну мову, ми можемо створювати продукти та послуги, які взаємодіють з користувачами більш природним способом.

Завдяки вдосконаленню алгоритмів комп'ютерного зору, ми можемо розпізнавати об'єкти, особливості та зображення в реальному часі, що розширює можливості віртуальної та доповненої реальності.

Машинне навчання допомагає у розробці систем діагностики та прогнозування захворювань, що може покращити якість медичної допомоги.

У сферах автомобільної техніки, дронів та робототехніки машинне навчання використовується для створення систем автономного управління.

Машинне навчання допомагає аналізувати фінансові ринки, передбачати тренди та ризики, що дозволяє приймати кращі фінансові рішення.

Машинне навчання використовується для аналізу даних у соціальних мережах, підбору рекомендацій та покращення взаємодії користувачів.

Машинне навчання допомагає виявляти підозрілу активність та ідентифікувати загрози в кіберпросторі.

Ці тенденції свідчать про те, що машинне навчання стає все більше важливим і широко використовується у багатьох галузях, що робить його дуже актуальною темою для вивчення та розвитку.

Мета і завдання дослідження

Метою дослідження є аналіз ефективності алгоритмів машинного навчання для пошуку шляху у віртуальному світі.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- Провести аналіз сучасного стану розвитку алгоритмів машинного навчання.
- Виявити переваги і недоліки різних алгоритмів.
- Використати найкращі алгоритми навчання з підкріпленням для пошуку шляху у віртуальному світі.

Об'єкт дослідження – процес взаємодії агента з середовищем для пошуку шляху у віртуальному світі.

Предмет дослідження – методи машинного навчання з підкріпленням для пошуку шляху у віртуальному світі.

РОЗДІЛ 1 СУЧАСНИЙ СТАН РОЗВИТКУ МАШИННОГО НАВЧАННЯ

1.1 Розвиток і впровадження машинного навчання

Застосуванням навчання з підкріпленням в програмах штучного інтелекту розробники надають можливість системі вчитися та приймати послідовні рішення на основі навчальних моделей. Це дозволяє програмам розвиватися та набувати досвіду у вирішенні завдань у складних та невизначених середовищах, схожих на ігрові сценарії.

Додатки зі штучним інтелектом використовують методи випробувань і помилок, щоб знаходити творчі рішення для поставлених завдань. Програмісти надають вказівки машині на основі правильних моделей машинного навчання, які вони вчать. Штучний інтелект отримує винагороду за правильне виконання завдань, але може стикатися з покаранням у вигляді втрати бонусних балів при неправильних виборах.

Головною метою для програми штучного інтелекту є накопичення якомога більшої кількості бонусних балів для досягнення перемоги в грі. Програмісти встановлюють правила гри та стратегії винагороди. На відміну від інших моделей машинного навчання, програма штучного інтелекту не отримує прямих підказок від програміста.

Штучний інтелект повинен самостійно з'ясовувати, як розв'язувати завдання гри для отримання максимальної винагороди. Для цього додаток може використовувати методи випробувань і помилок, випадкові експерименти, вміння працювати з потужними обчислювальними ресурсами та складні стратегії обробки інформації. Для досягнення цієї мети потрібно оснастити програму штучного інтелекту відповідною обчислювальною інфраструктурою та встановити зв'язок її системи мислення з різноманітними паралельними та історичними процесами гри. Це дозволяє штучному інтелекту проявляти креативну та високорівневу творчість, яку не можуть досягти люди.

Артур Самуель був пионером у розробці першої програми, яка використовувала алгоритми для самонавчання. Ця програма була створена в 1952 році і була спрямована на гру в шашки. Самуель вніс важливий внесок у сферу штучного інтелекту, і він також вперше визначив термін "машинне навчання" як "область досліджень, пов'язаних із створенням машин, які можуть навчатися без жорсткого програмування". Пізніше Т. М. Мітчелл надав більш точне визначення терміну "навчання", стверджуючи, що комп'ютерна програма вважається навченою, якщо вона поліпшує якість вирішення задач певного класу з плином часу, використовуючи набутий досвід.

Уже в 1957 році була представлена перша модель нейронної мережі, яка вперше реалізувала алгоритми машинного навчання, схожі на ті, що використовуються сьогодні. В даний час існують різні системи машинного навчання, які розробляються для майбутніх технологій, таких як Інтернет Речей, Промисловий Інтернет Речей, концепція "розумного" міста, безпілотний транспорт і багато інших.

Є ряд фактів, що свідчать про сучасні та майбутні перспективи машинного навчання.

- Компанія Google очікує переходу своїх продуктів від «результатів традиційного програмування» до таких, в основу яких покладено машинне навчання.
- Компанії Google, Facebook, Apple, Amazon, Microsoft і китайська Baidu активно набирають в штат талановитих фахівців у галузі штучного інтелекту;
- Марк Цукерберг, генеральний директор Facebook, особисто бере участь у рекрутингу найкращих випускників до його компанії.
- Майже в чотири рази збільшилася кількість відвідувачів на важливих академічних конференціях в галузі штучного інтелекту.
- Siri від Apple, M від Facebook, Echo від Amazon та інші продукти, створені з допомогою машинного навчання.

1.2 Методи машинного навчання

В загальному висновку, існують два основних типи машинного навчання: навчання по прецедентах, також відоме як індуктивне навчання, та дедуктивне навчання. Оскільки дедуктивне навчання в основному пов'язане з експертними системами, терміни "машинне навчання" і "навчання по прецедентах" можна вважати взаємозамінними. На сьогоднішній день метод навчання по прецедентах дуже популярний, в той час як експертні системи зазнають занепаду. Це пов'язано з тим, що бази знань, на яких ґрунтуються експертні системи, важко адаптувати до реляційної моделі даних, що призводить до неефективного використання промислових систем управління базами даних для наповнення баз знань експертних систем.

Навчання по прецедентах можна поділити на три основні типи: контрольоване навчання, також відоме як навчання з учителем, неконтрольоване навчання, або навчання без учителя, і навчання з підкріпленням.

Крім цих основних типів існують і інші методи навчання, такі як активне навчання, багатозадачне навчання, різноманітне навчання, трансферне навчання і інші. Особливо успішним розвитком останнім часом користується "глибоке навчання", яке дозволяє успішно поєднувати алгоритми навчання з учителем та без учителя.

Контрольоване навчання

Цей метод навчання застосовується у випадках, коли маємо справу з великими обсягами даних, наприклад, тисячами фотографій домашніх тварин, які мають маркери або ярлики для ідентифікації, наприклад, "це кішка", а "це собака". Нам необхідно створити алгоритм, який дозволить машині автоматично визначити, хто на фотографії, яку вона раніше не бачила: чи це кішка, чи собака. У цьому випадку людина виступає у ролі "вчителя", яка перед тим позначила маркерами об'єкти на фотографіях. Сама машина вивчає, які ознаки відрізняють кішок від собак і створює відповідний алгоритм. Тому отриманий алгоритм можна легко перенастроїти для вирішення інших завдань,

наприклад, розпізнавання курей і качок. В цьому випадку машина також автоматично визначить характеристики, за якими можна відрізнити цих птахів. Більше того, нейромережу, яку навчили розпізнавати кішок, можна швидко підготувати до обробки результатів комп'ютерної томографії.

Неконтрольоване навчання

Навіть при наявності значної кількості даних із позначками, безметові дані є численнішими. Сюди входять зображення без асоційованих підписів, аудіозаписи без експлікацій, та тексти без анотацій. Мета машини в умовах неконтрольованого навчання полягає в тому, щоб встановити взаємозв'язки між ізольованими даними, розпізнати закономірності, виявити шаблони, систематизувати дані або описати їх структуру, здійснити класифікацію даних. Зокрема, неконтрольоване навчання застосовується в рекомендаційних системах, де на підставі аналізу попередніх покупок покупцеві пропонуються товари з вищою ймовірністю зацікавленості, або у випадку відеоплатформ, як YouTube, де після перегляду відеокліпу надаються рекомендації на аналогічні ролики. Також Google може адаптувати результати пошуку для кожного користувача враховуючи його історію пошуків.

Навчання з підкріпленням

Цей вид навчання є варіацією контрольованого навчання, але вчителем у цьому випадку є "середовище". Машина (часто називається "агентом" у цьому контексті) не має попередньої інформації про середовище, але має можливість виконувати різні дії в ньому. Середовище реагує на ці дії, і таким чином, надає агенту дані, які дозволяють йому реагувати і навчатися. Фактично, агент та середовище утворюють систему з зворотнім зв'язком.

Навчання з підкріпленням використовується для розв'язання складніших завдань, ніж навчання з учителем чи без нього. Наприклад, цей метод застосовується в системах навігації для роботів, які навчаються уникати зіткнень з перешкодами через набуття досвіду та отримання зворотного зв'язку при кожному зіткненні. Навчання з підкріпленням також знаходить

застосування в логістиці, плануванні завдань і розкладах, а також у навчанні машин грати в логічні ігри, такі як покер, нарди, го і інші.

1.3 Нейронні мережі і глибоке навчання

Для машинного навчання застосовують різноманітні технології і алгоритми. Наприклад, можуть використовуватися методи, такі як дискримінантний аналіз, байєсівські класифікатори та інші математичні підходи. Проте в останній чверті XX століття виникла значна зацікавленість у напрямку штучних нейронних мереж (ANN). Цей інтерес посилився після значних досягнень у "Методі зворотного поширення помилки", що став важливим інструментом при навчанні нейронних мереж.

ANN представляють собою систему взаємопов'язаних штучних нейронів, які базуються на відносно простих процесорах. Кожен процесор у мережі ANN періодично отримує сигнали від інших процесорів або інших джерел, таких як сенсори, і періодично передає сигнали іншим процесорам. Загалом ці прості процесори, які з'єднані у мережу, здатні розв'язувати складні завдання.

Зазвичай нейрони розташовуються в мережі за рівнями, які також називають шарами. Нейрони першого рівня зазвичай є входами і отримують дані з зовнішнього середовища, наприклад, від сенсорів для системи розпізнавання облич. Після обробки ці дані передаються через синапси до нейронів на наступному рівні. Нейрони другого рівня, який називають прихованим, оскільки він не має прямих зв'язків з входами чи виходами ANN, обробляють ці дані і передають їх нейронам на вихідному рівні. Оскільки ми імітуємо нейрони, кожен процесор на вході пов'язаний з кількома процесорами на прихованому рівні, і кожен з них, в свою чергу, пов'язаний з кількома процесорами на виході. Ця архітектура представляє собою найпростішу ANN, яка здатна до навчання і може виявляти прості взаємозв'язки в даних.

Глибоке (глибинне) навчання застосовується переважно до більш складних ANN, які мають кілька прихованих рівнів. Кожен наступний рівень в мережі намагається знайти взаємозв'язки у попередньому. Така ANN здатна

виявляти не лише прості взаємозв'язки, але й взаємозв'язки між цими взаємозв'язками. Наприклад, перехід до ANN з глибоким навчанням дозволив Google значно покращити якість свого продукту "Перекладач", підвищивши якість перекладів між англійською та французькою мовами на 7 балів, що відповідає більше ніж 20% покращенню якості. Такий успіх не був досягнутий попередньою системою, яка використовувала статистичний метод машинного перекладу, існуючий з 2006 року.

1.4 Машинне навчання для бізнесу

Ринок машинного навчання швидко росте, перетнувши позначку в $1 млрд у 2016 році, і прогнозується, що до 2025 року ця галузь може збільшити свій обсяг до $39,98 млрд.

У кінці 2016 року MIT Technology Review та Google Cloud провели спільне дослідження з питань "Машинного навчання: новий спосіб отримати конкурентну перевагу". Були опитані 375 кваліфікованих респондентів із різних країн та галузей, таких як промисловість, послуги та фінанси. Результати дослідження показали, що 60% компаній вже використовують машинне навчання (ML), і третина з них перейшла від інноваційного використання до стадії зрілості. Крім того, 26% компаній вже здобувають конкурентну перевагу завдяки ML. Чверть компаній вкладає понад 15% від бюджету IT в розробку ML та отримує відмінні результати.

Машинне навчання, зокрема, нейронні мережі, стають актуальними при вирішенні бізнес-завдань в таких випадках:

- Існують великі обсяги різноманітних даних, для яких відсутні програми обробки та систематизації.

- Доступні дані спотворені, неповні або неструктуровані.

- Дані мають різний характер, і важко встановити між ними зв'язки і закономірності.

Бізнес-завдання, які можна вирішити за допомогою машинного навчання та нейронних мереж, включають:

- Прогнозування попиту, обсягу продажів, запасів на складі, завантаження обладнання та інших ресурсів, а також подальшого розвитку підприємства.

- Виявлення тенденцій, прихованих зв'язків, аномалій і повторюваних патернів у даних.

- Розпізнавання фото-, відео- та аудіоконтенту, а також виявлення спроб шахрайства, брехні, внутрішніх загроз та зовнішніх атак на систему безпеки.

- Автоматизація операцій, таких як робота операторів в онлайн-чатах та телефонних додатках.

- Класифікація та сегментація клієнтів, замовників та покупців за різними параметрами.

- Кластеризація даних для виявлення зв'язків, які спочатку не були відомі.

- Розробка чат-ботів та інших автоматизованих рішень.

1.5 Приклади реалізації

Машинне навчання вже активно використовується на найбільших торгових площадках та компаніях, і це дозволяє їм покращити різні аспекти бізнесу. Великі компанії, такі як Alibaba, Target та Pinterest, вдало застосовують методи машинного навчання для вирішення різних завдань та поліпшення власного функціонування.

1. Alibaba:

- Alibaba використовує машинне навчання для індивідуалізації віртуальних вітрин для кожного покупця. Це дозволяє створювати персоналізовані пропозиції та полегшує процес покупки.

- Чат-бот Ali Xiaomi використовує машинне навчання для автономного вирішення більшості технічних питань клієнтів у техпідтримці.

- Розроблена Alibaba нейронна мережа показала вражаючі результати в тестах від Стенфордського університету, перевершуючи навіть результати людини у завданнях з читання та аналізу інформації.

2. Target:

- Target використовує машинне навчання для передбачення не тільки покупкового поведінки клієнтів, але й змін в їхньому житті, таких як вагітність. Алгоритми можуть визначати триместр вагітності жінки за її покупками.

3. Pinterest:

- Pinterest використовує машинне навчання для персоналізації вмісту, що показується користувачам, допомагаючи їм знайти найцікавіші фотографії.

4. Neuromation:

- Neuromation, стартап з українським корінням, розробляє платформу для створення штучних навчальних середовищ для глибокого навчання нейронних мереж.

Машинне навчання вже внесло суттєві зміни в ефективність та прибутковість багатьох компаній. Цей напрямок розвитку стає доступним навіть для менших компаній, що дозволяє їм ефективніше працювати і отримувати більше прибутку.

1.6 Неконтрольоване машинне навчання

Алгоритми машинного навчання без нагляду спеціалізуються на виявленні шаблонів у наборі даних, які не мають заздалегідь встановлених або позначених виходів. У контрасті до цього, алгоритми машинного навчання під керівництвом нагляду опираються на заздалегідь відомі відповіді.

Розуміння цієї відмінності допомагає усвідомити, чому неконтрольовані методи машинного навчання не застосовуються для завдань регресії або класифікації, оскільки у таких випадках вихідні дані не мають відомих значень. Без відомих відповідей, неможливо навчити алгоритм ефективно.

До того ж, неконтрольоване навчання може використовуватися для визначення фундаментальної структури даних. Ці алгоритми виявляють приховані шаблони або групи в даних без втручання людини.

Через їхню здатність визначати подібності та відмінності в інформації, неконтрольоване навчання виявляється корисним для дослідницького аналізу даних, методів перехресних продажів, сегментації споживачів та ідентифікації зображень.

Подумайте про такий сценарій: ви стоїте в продуктовому магазині і бачите невідомий фрукт, який раніше не зустрічали. Ви можете визначити цей невідомий плід серед інших фруктів, спираючись на свої спостереження щодо його форми, розміру або кольору.

Алгоритми Машинного Навчання Без Контролю

Кластеризація

Безумовно, кластеризація є найбільш поширеним підходом до неконтрольованого навчання. Цей метод впорядковує взаємопов'язані елементи даних у створені випадковим чином групи, які називаються кластерами.

Сама модель машинного навчання розпізнає будь-які закономірності, схожості та відмінності в структурі несортованих даних. Модель може виявити різні природні групи або класи у наборі даних.

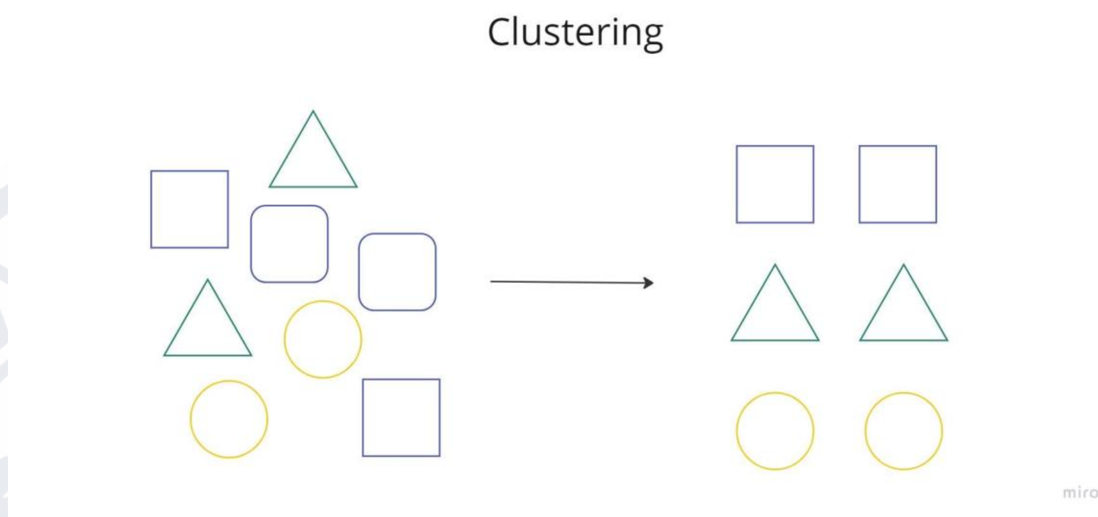

Рис. 1.1 – Приклад кластеризації

Типи

Існує кілька форм кластеризації, які можна використовувати. Давайте почнемо з розгляду найважливіших з них.

1. Ексклюзивна кластеризація, також іноді відома як "жорстка" кластеризація, - це тип групування, в якому кожен окремий об'єкт даних призначається лише одному кластеру.

2. Кластеризація з перекриттям, часто називається "м'якою" кластеризацією, допускає, що об'єкти даних можуть належати різною мірою до більш ніж одного кластеру. Цей підхід також може бути використаний для вирішення проблем "м'якої" кластеризації, оцінки щільності даних і визначення ймовірності того, що точка даних належить до певного кластеру.

3. Ієрархічна кластеризація створює ієрархію згрупованих об'єктів даних, як і вказує її назва. Елементи даних декомпозиціюються або об'єднуються на основі ієрархії для створення кластерів.

Користувачі:

1. **Виявлення аномалії:**

За допомогою кластеризації можна виявити будь-який вид аномалій в даних. Наприклад, компанії у сфері транспорту та логістики можуть використовувати алгоритми виявлення аномалій для виявлення логістичних перешкод або виявлення пошкоджених механічних частин (для проведення планового технічного обслуговування).

Фінансові установи можуть також скористатися цією технологією для виявлення шахрайських операцій і негайного реагування, що може в результаті призвести до значних економічних вигод. Дізнайтеся більше про виявлення аномалій і шахрайства, переглянувши наше відео.

2. **Сегментація клієнтів і ринків:**

Алгоритми кластеризації можуть сприяти групуванню осіб з подібними характеристиками, що дозволяє створити цільові споживацькі сегменти для більш ефективних маркетингових та цільових ініціатив.

К-середні

Метод K-середніх, також відомий як метод розділення або сегментації, є одним з алгоритмів кластеризації. Він розділяє точки даних на заздалегідь визначену кількість кластерів, відомих як K.

У методі K-середніх значення K визначається на вході, що означає, що ви повідомляєте комп'ютеру, скільки кластерів потрібно створити на основі ваших даних. Кожен елемент даних потім призначається найближчому центру кластера, який називається центроїдом (позначено чорними точками на рис. 1.2).

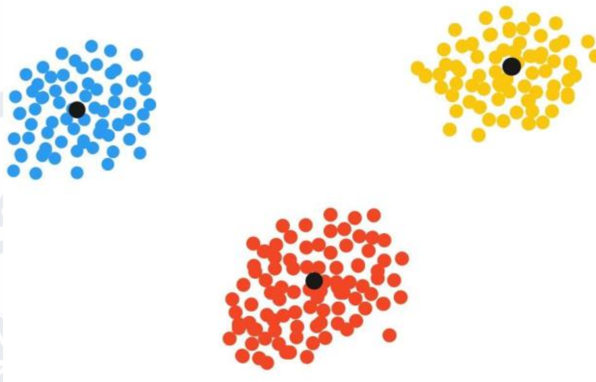

Рис. 1.2 – Визначення центроїдів

Місця для зберігання даних, такі як останні, використовуються для зберігання інформації. Техніку кластеризації можна застосовувати декілька разів, поки кластери не буде визначено чітко.

Нечіткі K-Середні

Нечеткий метод K-середніх є розширенням техніки K-середніх, яка використовується для кластеризації з можливістю перекриття. У відміну від методу K-середніх, нечеткі K-середні вказують на те, що точки даних можуть належати до багатьох кластерів з різним ступенем близькості до кожного. Близькість вимірюється за допомогою відстані між точками даних і центроїдом кластера, що призводить до можливості перекриття різних кластерів.

Моделі Гаусової Суміші

Моделі суміші Гауса (GMMs) - це підхід, що використовується у ймовірнісній кластеризації. У випадку, коли нам невідомі середнє значення та дисперсія, GMM припускає, що існує фіксована кількість розподілів Гауса, і кожен із них відображає окремий кластер. Щоб визначити, до якого саме кластера належить конкретна точка даних, використовується певний метод.

Ієрархічна Кластеризація

Стратегія ієрархічної кластеризації може розпочинати свій процес, призначаючи кожній точці даних свій власний, окремий кластер. Потім два

кластери, які розташовані найближче один до одного, поступово об'єднуються в один більший кластер. Цей ітеративний процес об'єднання триває до тих пір, поки не залишається лише один кластер на найвищому рівні ієрархії.

Цей метод відомий під назвою агломеративна кластеризація або висхідна кластеризація. Якщо, навпаки, ви починаєте з того, що всі елементи даних вважаються окремими кластерами і потім поділяєте їх на менші, до того часу, коли кожен елемент стає окремим кластером, то цей метод називається роздільною ієрархічною кластеризацією.

Апріорний Алгоритм

Аналіз ринкового кошика сприяв поширенню апріорних алгоритмів, що призвело до розробки різних механізмів рекомендацій для музичних платформ і онлайн-магазинів. Ці алгоритми використовуються в транзакційних наборах даних для виявлення часто спільних наборів товарів або груп товарів, а також для прогнозування ймовірності споживання одного продукту на основі споживання іншого.

Наприклад, якщо я розпочну відтворювати музику гурту OneRepublic на Spotify, починаючи з пісні «Counting Stars», однією з наступних пісень на цьому каналі, ймовірно, буде пісня гурту Imagine Dragons, наприклад, «Bad Liar». Це ґрунтується на моїх попередніх музичних вподобаннях, а також на моделях слухання інших користувачів. Апріорні методи визначають набори елементів, використовуючи хеш-дерево, проходячи через набір даних в широкому контексті.

Зменшення Розмірності

Зменшення розмірності - це вид неконтрольованого навчання, який використовує різні стратегії для зменшення кількості ознак або розмірів у наборі даних. Давайте розглянемо це детальніше.

Існує спокуса включити якнайбільше даних у свій набір для машинного навчання, і ця стратегія зазвичай призводить до більш точних результатів. Але, припустимо, що ваші дані існують в N-вимірному просторі, де кожна функція

відповідає окремому виміру. З великою кількістю даних може бути сотні таких вимірів.

Подумайте про електронні таблиці Excel, де стовпці представляють різні характеристики, а рядки - окремі дані. Коли кількість характеристик або вимірів стає надто великою, алгоритми машинного навчання можуть працювати неефективно, і візуалізація таких даних може стати важкою.

Таким чином, розумно обмежити кількість характеристик або вимірів та передавати лише необхідну інформацію. Зменшення розмірності надає можливість зберегти контрольовану кількість даних без втрати цілісності набору даних.

Аналіз Основних Компонентів (PCA)

Аналіз головних компонентів представляє собою прийом для зменшення розмірності даних, який використовується для скорочення кількості ознак у великих наборах даних, щоб зробити їх більш простими, не втрачаючи точність.

Скорочення розмірності набору даних відбувається за допомогою методу, відомого як вилучення ознак, що полягає в тому, що оригінальні ознаки об'єднуються в новий, менший набір. Ці нові ознаки називаються головними компонентами.

Звісно, існують інші алгоритми, які можна використовувати для неконтрольованого навчання. Перераховані вище методи є лише найпоширенішими і були розглянуті більш детально.

Застосування методів неконтрольованого навчання може бути наступним:

- Методи навчання без нагляду використовуються для завдань візуального сприйняття, таких як розпізнавання об'єктів.

- Машинне навчання без нагляду грає важливу роль у системах медичної візуалізації, таких як ідентифікація, класифікація та сегментація зображень, які використовуються у радіології та патології для швидкої та надійної діагностики пацієнтів.

- Неконтрольоване навчання може допомогти визначити тенденції даних, які можна використовувати для розробки ефективних стратегій перехресних продажів, використовуючи попередні дані про споживчу поведінку. Під час обробки замовлень це використовується онлайн-бізнесом, щоб пропонувати клієнтам необхідні додатки.

- Методи навчання без нагляду можуть аналізувати великі обсяги даних, щоб виявити аномалії, які можуть вказувати на несправність обладнання, помилки людини або порушення безпеки.

Проте навчання без нагляду має деякі недоліки:

- Оскільки вхідні дані не мають міток, які слугують ключами відповіді, результати моделей неконтрольованого навчання можуть бути менш точними.

- Неконтрольоване навчання зазвичай працює з масивними наборами даних, що може підвищити складність обчислень.

- Цей підхід потребує підтвердження вихідних даних людьми, як внутрішніми, так і зовнішніми експертами з предмету запиту.

- Алгоритми повинні досліджувати та обчислювати всі можливі сценарії протягом фази навчання, що може займати певний час.

1.7 Навчання під наглядом

Навчання під контролем відбувається в присутності керівника, подібно до навчання, коли маленька дитина навчається свого вчителя. Дитина вчиться розпізнавати фрукти, кольори та цифри під наглядом вчителя, тож цей метод є контрольованим.

У цьому методі кожний крок дитини перевіряється вчителем, і дитина набуває знання з результатів, які вона повинна досягти.

Принцип роботи контрольованого навчання

У керованому алгоритмі машинного навчання результати вже відомі. Існує відповідність між вхідними та вихідними даними. Тобто, для створення моделі машина отримує значну кількість тренувальних вхідних даних, де відомі вхідні параметри та відповідні результати.

Ці тренувальні дані допомагають досягти високого рівня точності для створеної моделі. Тепер ця модель готова до обробки нових вхідних даних та прогнозування результатів.

Побудова маркованого набору даних

Позначений набір даних - це набір даних, де для кожного вхідного запису відомі відповідні вихідні дані. Наприклад, якщо у нас є зображення фрукта разом із відомою назвою цього фрукта, то ми маємо позначений набір даних. У такому випадку, коли нам показують нове зображення фрукта, ми можемо використовувати цей набір даних для передбачення назви фрукта на основі зображення.

Навчання під наглядом - це ефективний метод навчання з високою точністю, особливо в задачах регресії та класифікації. У цьому підході модель навчається на підставі позначених даних, де для кожного вхідного прикладу відомий очікуваний вихід. Модель аналізує ці дані та навчається знаходити зв'язок між вхідними та вихідними даними, щоб потім здійснювати прогнози для нових вхідних даних.

Деякі з контрольованих алгоритмів навчання:

- Дерева прийняття рішень,
- K-Найближчий сусід,
- Лінійна регресія,
- Підтримка векторної машини та
- Нейронні мережі.

<u>Приклад навчання під наглядом</u>

- Спочатку ми вводимо навчальний набір даних в алгоритм машинного навчання.
- За допомогою цього навчального набору даних, машина налаштовується автоматично, адаптуючи свої параметри для створення логічної моделі.

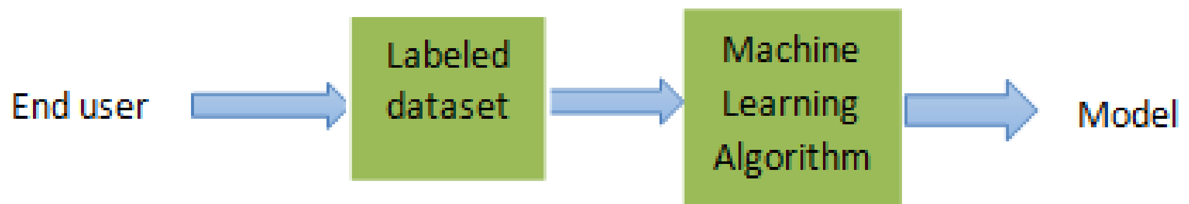

Рис. 1.3 – Процес створення логічної моделі

- Отриману модель потім можна використовувати для аналізу нового набору даних та здійснення прогнозу результату.

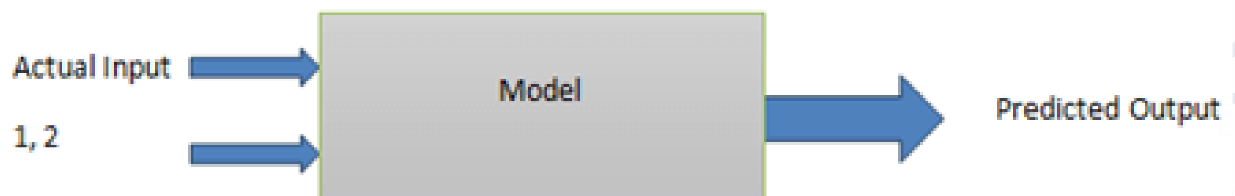

Рис. 1.4 – Процес прогнозування результату на основі нових даних

Типи керованих алгоритмів навчання:

1. Класифікація: У таких задачах передбачається відповідь у вигляді конкретних класів, таких як "так" чи "ні". Якщо присутні всього два класи, то це називається бінарною класифікацією. У випадку, коли існує більше ніж два класи, це називається багатокласовою класифікацією. Прогнозовані значення відповіді є дискретними. Наприклад, алгоритм класифікації може визначати, чи на зображенні зображено сонце чи місяць, розділяючи дані на різні класи.

2. Регресія: Задачі регресії передбачають відповідь у вигляді безперервних значень, таких як передбачення величини, яка може приймати будь-які значення від мінус нескінченності до плюс нескінченності. Це може охоплювати широкий діапазон значень. Наприклад, алгоритм лінійної регресії може передбачати вартість будинку на основі різних параметрів, таких як розташування, відстань до аеропорту, площа будинку тощо.

1.8 Навчання без нагляду

Навчання без нагляду - це процес навчання, що відбувається автономно, подібно до того, як риба самостійно вчиться плавати. У цій моделі відсутні вихідні дані, пов'язані з вхідними даними, і цільові значення не відомі або не позначені. Система самостійно вивчається з введених в неї даних і виявляє приховані закономірності.

Маркований набір даних, в якому відсутні вихідні значення для всіх вхідних даних, називається немаркованим набором даних.

Принцип роботи неконтрольованого навчання полягає в тому, що, оскільки відомі вихідні значення, які можна було б використовувати для побудови логічної моделі між входом і виходом, відсутні, застосовуються методи видобутку правил даних, шаблонів і груп подібних даних. Ці групи допомагають кінцевим користувачам краще зрозуміти дані і виявити значущі зв'язки.

Подані вхідні дані можуть бути неструктурованими і містити викиди, шум та інші неточності. Під час навчання моделі вхідні дані організовуються для формування кластерів.

Алгоритми навчання без нагляду включають алгоритми кластеризації та асоціації, такі як "Апріорі", "K-середні" та інші алгоритми видобутку правил асоціацій.

Коли нові дані подаються в модель, вона прогнозує результат як мітку класу, до якого відноситься вхід. Якщо мітки класу не існує, модель може створити новий клас.

Під час процесу виявлення закономірностей у даних модель самостійно коригує свої параметри, тому її також називають самоорганізуючою. Кластери формуються на основі подібності між вхідними даними.

Наприклад, під час покупок продуктів в Інтернеті, якщо кладуть вершкове масло в кошик, модель, що не контролюється, може автоматично пропонувати купити хліб, сир і інші атрибути, пов'язані з цим товаром.

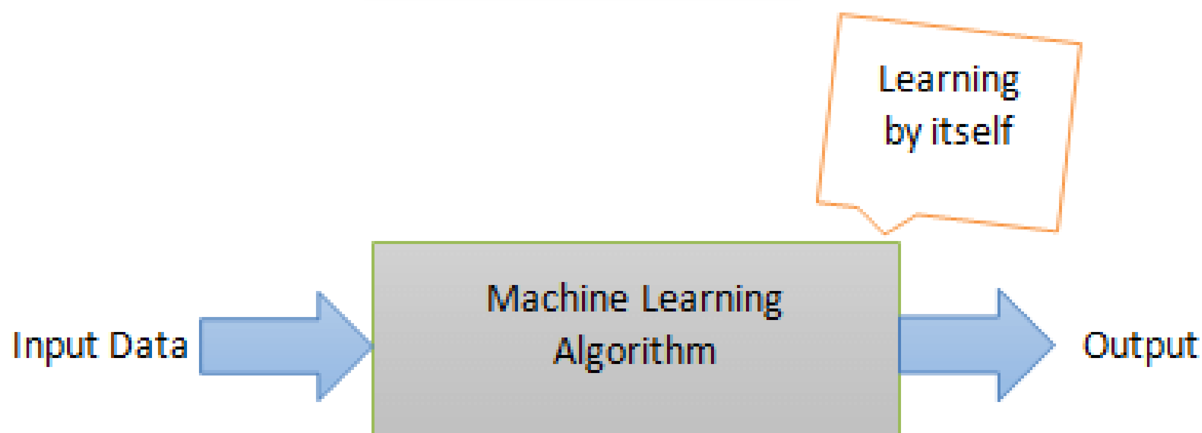

Рис. 1.5 – Схема навчання без нагляду

Типи неконтрольованих алгоритмів

• Алгоритм кластеризації: Цей метод використовується для пошуку схожих або однотипних елементів даних, таких як елементи з аналогічною формою, розміром, кольором, ціною і т. д. Після цього вони групуються разом у кластери, щоб створити логічні групи даних у процесі, що називається кластерним аналізом.

• Виявлення сторонніх джерел: Цей метод використовується для пошуку будь-яких аномальних даних або незвичайних відхилень в наборі даних. Наприклад, система виявлення шахрайства може автоматично впізнавати високовартісні транзакції на кредитних картках, які можуть бути підозрілими.

• Асоціація та видобуток правил: Цей тип видобутку даних виявляє найчастіше зустрічаються набори предметів або асоціації між елементами даних. Наприклад, цей метод може виявити асоціації між товарами, які часто купують разом, що корисно для стратегій маркетингу та розміщення товарів.

• Автокодери: Вхідні дані стискаються до закодованої форми, а потім реконструюються для видалення зайвого шуму або для покращення якості зображення та відео. Цей метод застосовується, наприклад, для зменшення розміру зображень або для покращення їх якості.

1.9 Навчання з підкріпленням

Під час цього типу навчання алгоритм вчиться за допомогою механізму зворотного зв'язку та власного досвіду. Кожен крок в алгоритмі завжди орієнтований на досягнення мети.

Таким чином, кожен наступний крок здійснюється на основі зворотного зв'язку від попереднього кроку, враховуючи накопичений досвід, щоб передбачити, який може бути наступний оптимальний крок. Цей процес також називається процесом "спроб і помилок" для досягнення мети.

Підкріплене навчання - це довготривалий ітераційний процес, де більше навчальних випробувань допомагає системі стати точнішою. Цей вид навчання також відомий як "Марковське прийняття рішень".

Прикладом підкріпленого навчання є відеоігри, де гравці проходять рівні гри та отримують бали винагороди. Гра надає зворотний зв'язок гравцю за допомогою бонусів та покарань для покращення їх продуктивності.

Підкріплене навчання також застосовується в навчанні роботів, автономних автомобілів, систем управління запасами тощо.

Деякі популярні алгоритми підкріпленого навчання включають:

- Q-навчання,
- Глибокі змагальні мережі,
- Часову різницю.

Малюнок нижче ілюструє механізм зворотного зв'язку у підкріпленому навчанні:

1. Спостереження вхідних даних агентом, який є частиною системи штучного інтелекту.

2. Цей агент діє на навколишнє середовище на основі свого прийнятого рішення.

3. Реакція середовища відправляється агентові у вигляді винагороди відповідно до результатів його дій.

4. Паралельно з цим агент зберігає інформацію про стан та дії, які він здійснив у навколишньому середовищі.

Рис. 1.6 – Механізм зворотного зв'язку у підкріпленому навчанні

1.10 Аналіз відмінностей між контрольованим і неконтрольованим навчанням

Навчання машин – це розгалужена галузь, яка включає в себе різні підходи і методи для навчання комп'ютерів і штучних інтелектів виконувати завдання. Два основних підходи до навчання машин – це контрольоване навчання і неконтрольоване навчання. Ці два підходи мають суттєві відмінності, які визначають їх використання та області застосування.

1. Визначення:

- Контрольоване навчання: Це метод навчання, в якому модель отримує набір вхідних даних, який має відомі вихідні значення. Модель навчається з використанням цих даних з метою встановлення зв'язку між вхідними та вихідними даними.

- Неконтрольоване навчання: В цьому методі модель отримує набір вхідних даних без відомих вихідних значень. Модель самостійно намагається виявити закономірності, структуру або групи в цих даних.

2. Цільові завдання:

- Контрольоване навчання: Зазвичай використовується для задач, де маємо відомі вихідні дані і хочемо навчити модель виконувати точні прогнози або класифікацію.

- Неконтрольоване навчання: Використовується для завдань, де немає відомих вихідних даних і метою є виявлення прихованих структур або групувань в даних.

3. Приклади задач:

- Контрольоване навчання: Класифікація об'єктів на основі їх характеристик (наприклад, розпізнавання образів), прогнозування цільових змінних (наприклад, ціна акцій).

- Неконтрольоване навчання: Кластеризація даних для виявлення груп схожих об'єктів, зменшення розмірності даних (наприклад, метод головних компонентів).

4. Тип вихідних даних:

- Контрольоване навчання: Результати цього виду навчання - це точні прогнози або класифікація, які можна перевірити за допомогою відомих вихідних даних.

- Неконтрольоване навчання: Вихідні дані можуть бути менш очевидні, і їх інтерпретація може вимагати додаткового аналізу.

5.Використання в реальному світі:

- Контрольоване навчання: Застосовується в задачах, де маємо точно визначені цілі та потребуємо високої точності (наприклад, медична діагностика).

- Неконтрольоване навчання: Застосовується в областях, де потрібно виявити нові закономірності або розуміти структуру даних (наприклад, аналіз соціальних мереж).

6. Прогностична здатність:

- Контрольоване навчання: Зазвичай має високу прогностичну здатність, оскільки модель навчається на основі відомих вихідних даних.

- Неконтрольоване навчання: Прогностична здатність може бути менш точною, оскільки модель формується на основі прихованих структур.

7. Кількість даних:

- Контрольоване навчання: Зазвичай вимагає більше даних, оскільки модель навчається точно відповідати вихідним даним.

- Неконтрольоване навчання: Може використовувати менше даних, оскільки модель шукає загальні закономірності.

8. Питання етики і конфіденційності:

- Контрольоване навчання: Може виникнути питання щодо конфіденційності даних, оскільки модель має доступ до відомих вихідних даних.

- Неконтрольоване навчання: Менш вразливе до питань конфіденційності, оскільки вихідні дані не обов'язкові.

9. Підходи до валідації:

- Контрольоване навчання: Використовує перевірку з використанням відомих тестових даних для оцінки точності моделі.

- Неконтрольоване навчання: Вимагає більш складних методів валідації, таких як перехресна перевірка або внутрішні показники якості.

Таблиця 1. Відмінності між контрольованим і неконтрольованим навчанням

Контрольоване навчання	**Неконтрольоване навчання**
У керованих алгоритмах навчання відомий результат для заданих вхідних даних.	При використанні навчання без нагляду результат для вхідних даних невідомий.
Алгоритми навчаються з використанням маркованого набору даних. Які допомагають оцінити коректність навчання.	Алгоритм не використовує маркованих даних, але він намагається знайти закономірності та асоціації між елементами даних.
Це техніка, яка на основі прогнозного моделювання, передбачає майбутні результати.	Основою цієї техніки є описове моделювання, яке відображає взаємозв'язок між елементами.
Він використовує алгоритми класифікації та регресії.	Він застосовує алгоритми навчання кластерів та правил асоціації.
Приклади алгоритмів контрольованого навчання: Лінійна регресія, Наїв Байєс, Нейронні мережі.	Приклади алгоритмів навчання без нагляду: k-кластеризація, Apriori тощо.
Відносно складніший тип навчання, оскільки потребує маркованих даних.	Це менш складний алгоритм, у якого відсутня необхідність розуміння та розмічування даних.
Цей процес аналізу даних не вимагає взаємодії з людиною.	Цей аналіз даних відбувається в режимі реального часу.

У підсумку, контрольоване і неконтрольоване навчання є двома важливими підходами в галузі машинного навчання, кожен з яких має свої особливості і застосування. Вибір між ними залежить від конкретної задачі, доступних даних і мети навчання. Навчання машин стає все більш важливою галуззю, і розуміння цих відмінностей допомагає вибрати належний підхід для досягнення бажаних результатів.

Висновки до розділу 1

У розділі розглянуто історичні аспекти розвитку машинного навчання. Проаналізовано основні сучасні методи забезпечення машинного навчання. Наведено приклади використання автоматизованих систем здатних до самонавчання.

Виявлено, що системи самонавчання мають історію розвитку, яка почалася разом зі створенням обчислювальних систем. За цей час створено різні підходи до самонавчання машин, в розділі визначені їх основні особливості, переваги і недоліки. Аналіз використання систем самонавчання показав постійний розвиток як алгоритмів самонавчання так і галузей, в яких вони використовуються.

РОЗДІЛ 2 АЛГОРИТМИ НАВЧАННЯ І ПОШУКУ ШЛЯХУ

2.1 Класифікація методів навчання з підкріпленням

По-перше, методи навчання з підкріпленням діляться на:

- методи на основі моделі (model-based);

- методи без моделі (model-free).

Під *моделлю* розумітимемо модель взаємодії агента з навколишнім середовищем.

У методах на основі моделі існує модель переходів (є апріорні знання про ймовірності переходів між станами), яка використовується для отримання оптимальних дій. Таким чином, використовується оптимальна політика.

У методах без моделі оптимальні дії отримують методом проб і помилок взаємодії з навколишнім середовищем.

По-друге, методи навчання з підкріпленням поділяються на:

- методи *on-policy*, які використовують однакову політику для вибору дії та обчислення функції вартості (наприклад, метод управління Монте-Карло on-policy, метод SARSA, будь-який метод актора-критика використовує безжальну політику);

- методи *off-policy*, які використовують різні політики для вибору дії та обчислення функції вартості (наприклад, метод управління Монте-Карло off-policy, методи на основі Q-навчання). Одна політика жадібна (нежадібна), інша нежадібна (жадібна).

По-третє, методи навчання з підкріпленням діляться на:

- методи на основі функції вартості;

- методи на основі політики;

- методи актора-критика (actor-critic).

Методи на основі функції вартості поділяють на чотири класи:

- динамічне програмування;

- адаптивне динамічне програмування;

- Монте-Карло;

- навчання з часовою різницею (Temporal-Difference Learning).

Методи динамічного програмування (DP) використовують модель і оновлюють значення функції вартості на основі попередніх значень функції вартості, не чекаючи отримання всієї траєкторії.

Методи адаптивного динамічного програмування (ADP) навчають модель, використовуючи досвіди (experience) у середовищі; подібно до методів динамічного програмування використовують модель і оновлюють значення функції вартості на основі попередніх значень функції вартості, не чекаючи отримання всієї траєкторії.

Методи Монте-Карло (MC) не використовують модель, а використовують досліди (experience) у середовищі, причому оновлюють значення функції вартості тільки після отримання всієї траєкторії.

Методи навчання з часовою різницею (TD) подібно до методів Монте-Карло не використовують модель, а використовують досвіди (experience) у середовищі; подібно до методів динамічного програмування оновлюють значення функції вартості на основі попередніх значень функції вартості, не чекаючи отримання всієї траєкторії. Назва "навчання з часовою різницею" пов'язана з тим, що для оновлення значення функції вартості використовують значення функції вартості на попередніх послідовних часових кроках.

Методи на основі політики не використовують модель, а використовують досвіди (experience) у середовищі та напряму оптимізують політику.

Методи актора-критика є комбінація методів на основі функції вартості та методів на основі політики.

З методами навчання з підкріпленням пов'язана проблема недонавчання-перенавчання, яка називається *дилемою зміщення-дисперсії* (dilemma bias-variance).

Під *недонавчанням* (underfitting) розуміють те, що методи можуть слабо враховувати взаємозв'язки між вхідними і вихідними навчальними даними (наприклад, модель методу недостатньо складна або навчена на малому наборі даних), тобто висока ймовірність помилки на навчальному наборі даних.

Під *перенавчанням* (overfitting) розуміють те, що методи можуть погано працювати на зашумлених тестових даних через слабку узагальнювальну здатність (наприклад, модель методу надто складна або навчена тільки на конкретному наборі даних), тобто висока ймовірність помилки на тестовому наборі даних.

По-четверте, методи навчання з підкріпленням діляться на:

- методи з дискретним простором дій (наприклад, методи навчання з часовою різницею SARSA, Q-навчання і DQN);

- методи з безперервним простором дій (наприклад, деякі методи на основі функції вартості, деякі методи актора-критика);

- методи з дискретним або безперервним простором дій (наприклад, методи на основі політики, методи актора-критика A2C, A3C і PPO).

По-п'яте, методи навчання з підкріпленням поділяються на:

- активні (з нефіксованою політикою);

- пасивні (з фіксованою політикою, наприклад, методи прямого оцінювання, пасивного адаптивного динамічного програмування, TD-навчання).

По-шосте, методи навчання з підкріпленням поділяються на:

- одноагентні (single-agent) методи;

- мультиагентні (multiagent) методи.

Мультиагентні методи можуть бути з кооперативними (cooperative), конкурентними (змагальними) (competitive), змішаними (кооперативно-конкурентними) агентами.

Якщо кооперативні агенти мають однакову функцію винагороди, то вони називаються *гомогенними*. Якщо кооперативні агенти мають різну функцію винагороди, то вони називаються *гетерогенними*. У разі гетерогенності обчислюється середня винагорода за всіма агентами.

Мультиагентні методи з кооперативними агентами бувають:

- централізованими (присутній центральний контролер; мультиагентна система пов'язана, оскільки кожен агент обмінюється інформацією з усіма

агентами для уникнення нестаціонарності; такі методи погано масштабуються і менше орієнтовані на реальний світ) (наприклад, Q-RTS);

- децентралізованими (відсутній центральний контролер; мультиагентна система слабко пов'язана, оскільки кожен агент обмінюється інформацією тільки з агентами зі своєї околиці; такі методи добре масштабуються і більше орієнтовані на реальний світ) (наприклад, QD-навчання).

Сума значень функцій винагороди всіх агентів-конкурентів для будь-якої пари (s, a) дорівнює 0. Найчастіше використовують двох агентів-конкурентів.

2.2 Приклади методів навчання з підкріпленням

DP (Policy Iteration)

Цей метод у разі використання функції вартості стану представлений у такому вигляді:

1 Оцінюють політику за допомогою обчислення значення функції вартості стану відповідно до фіксованої політики на основі рівняння Беллмана

$$V(s) = \sum_{a \in A(s)} \pi(a \mid s) \sum_{s' \in S} P(s' \mid s, a)\big(R(s, a, s') + \gamma V(s')\big), \quad s \in S,$$

де $\pi(a \mid s)$ - політика (імовірність того, що дія *a* відбувається за умови перебування агента в стані *s*),

$P(s' \mid s, a)$ - функція переходу (імовірність переходу зі стану *s* у стан s' у результаті дії *a*),

$R(s, a, s')$ - винагорода (нагорода за перехід зі стану *s* у стан s' внаслідок дії *a*),

$V(s)$ - функція вартості стану (прибуток у разі стану), s

γ - коефіцієнт дисконтування (важливість винагороди),

$A(s)$ - безліч дій, доступних у стані *s*,

S - множина станів.

Крок 1 повторюється задану кількість разів.

Для цього методу заздалегідь відомі $,\pi(a \mid s)\ R(s,a,s')$ і $P(s' \mid s,a)$.

Для цього методу ініціалізується $V(s)=0$ або $,V(s)=U(0,1)\ s \in S$.

Цей метод у разі використання функції вартості стану-дії представлений у такому вигляді:

1. оцінювання політики за допомогою обчислення значення функції вартості стану-дії відповідно до фіксованої політики

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

$a \in A(s)\ s \in S$,

де $Q(s,a)$ - функція вартості стану-дії (прибуток у разі стану *s* і дії *a*, яка можлива в цьому стані),

Крок 1 повторюється задану кількість разів.

Для цього методу заздалегідь відомі $,\pi(a \mid s)\ R(s,a,s')$ і $P(s' \mid s,a)$.

Для цього методу ініціалізується $Q(s,a)=0$ або , , $Q(s,a)=U(0,1)\ a \in A(s)$ $s \in S$.

Зауваження. По завершенню методу оцінювання ітеративної політики вибираються дії

$$b=\arg\max_{a \in A(s)} Q(s,a),\ s \in S$$

```
### DP (Policy Iteration)
import numpy

gamma = 0.9
EPISODES = 200 # кількість епізодів
ROAD = 0 # вільно
HOLE = 1 # пастка
GOAL = 2 # ціль
# карта світу плиток
MYMAP = [[ROAD,HOLE,ROAD,ROAD,ROAD],
      [ROAD,HOLE,ROAD,ROAD,ROAD],
      [ROAD,HOLE,ROAD,ROAD,ROAD],
      [ROAD,HOLE,ROAD,ROAD,ROAD],
      [ROAD,ROAD,ROAD,ROAD,GOAL]]
```

```
Y_MYMAP = 5 # висота карти
X_MYMAP = 5 # ширина карти
ActionName = ["UP", "DOWN", "LEFT", "RIGHT"]

## Обчислення винагороди в залежності від стану наступної клітинки
def get_reward_of_next_state(next_state):
    if MYMAP[next_state // X_MYMAP][next_state % X_MYMAP] == HOLE:
        reward = -1
    elif MYMAP[next_state//X_MYMAP][next_state%X_MYMAP] == ROAD:
        reward = 0
    elif MYMAP[next_state//X_MYMAP][next_state%X_MYMAP] == GOAL:
        reward = 1
    return reward

## обчислення вірогідності переходу
def calcilate_probability(state, action, p):
    # якщо UP і не в першому рядку карти
    if (action == 0) and (state // X_MYMAP > 0):
        p[state][action][state - X_MYMAP] = 1.0
    # якщо DOWN і не в останньому рядку карти
    elif (action == 1) and (state // X_MYMAP < Y_MYMAP-1):
        p[state][action][state + X_MYMAP] = 1.0
    # якщо LEFT і не в першому стовбчику карти
    elif (action == 2) and (state % X_MYMAP > 0):
        p[state][action][state - 1] = 1.0
    # якщо RIGHT і не в останньому стовбчику карти
    elif (action == 3) and (state % X_MYMAP < X_MYMAP-1):
        p[state][action][state + 1] = 1.0
    return p
state_size = X_MYMAP * Y_MYMAP # кількість клітинок
action_size = len(ActionName) # кількість дій
# ініціалізація вірогідності
p = numpy.zeros((state_size,action_size,state_size))
for state in range(state_size):
    for action in range(action_size):
        p = calcilate_probability(state, action, p)
# ініціалізація винагороди
r = numpy.zeros((state_size,action_size,state_size))
for state in range(state_size):
    for action in range(action_size):
        for next_state in range(state_size):
            r[state][action][next_state] = get_reward_of_next_state(next_state)
# ініціалізація політики
policy = numpy.zeros(state_size)
for s in range(state_size):
    policy[s] = numpy.random.choice(action_size)

V = numpy.random.rand(state_size)
while True:
    # оцінювання політики
    for e in range(EPISODES):
        for s in range(state_size):
            a = int(policy[s])
            for next_state in range(state_size):
                V[s] += p[s][a][next_state]*(r[s][a][next_state] + gamma * V[next_state])
    # покращення політики
    V_temp = numpy.zeros((state_size, action_size))
    for s in range(state_size):
        for a in range(action_size):
```

```
            for next_state in range(state_size):
                V_temp[s][a] += p[s][a][next_state]*(r[s][a][next_state] + gamma * V[next_state])
    policy_old = policy
    policy = numpy.argmax(V_temp, axis=1)
    z=False # флаг покращення політики
    for s in range(state_size):
        if (policy_old[s]!=policy[s]):
            z=True
    if (z==False):
        break

## друк результаів політики
MYMAP_named = numpy.empty(shape=(state_size,), dtype=str)
for s in range(state_size):
    MYMAP_named[s] = ActionName[policy[s]]
for y in range(Y_MYMAP):
    print(MYMAP_named[y*X_MYMAP:(y+1)*X_MYMAP-1])
```

```
PS C:\py\marathon>  & 'C:\Program Files\Python311\python.exe' 'c:\Users\stafme\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '5
5200' '--' 'C:\py\DP(pollicy iteration).py'
['D' 'D' 'D' 'D']
['D' 'D' 'D' 'D']
['D' 'D' 'D' 'D']
['D' 'D' 'D' 'D']
['R' 'R' 'R' 'R']
PS C:\py\marathon>  c:; cd 'c:\py\marathon'; & 'C:\Program Files\Python311\python.exe' 'c:\Users\stafme\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/.
./..\debugpy\launcher' '55212' '--' 'C:\py\DP(pollicy iteration).py'
['D' 'D' 'D' 'D']
['D' 'D' 'D' 'D']
['D' 'D' 'D' 'D']
['D' 'D' 'D' 'D']
['R' 'R' 'R' 'R']
PS C:\py\marathon>  c:; cd 'c:\py\marathon'; & 'C:\Program Files\Python311\python.exe' 'c:\Users\stafme\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/.
./..\debugpy\launcher' '55223' '--' 'C:\py\DP(pollicy iteration).py'
C:\py\DP(pollicy iteration).py:69: RuntimeWarning: overflow encountered in scalar add
  V[s] += p[s][a][next_state]*(r[s][a][next_state] + gamma * V[next_state])
C:\py\DP(pollicy iteration).py:69: RuntimeWarning: invalid value encountered in scalar multiply
  V[s] += p[s][a][next_state]*(r[s][a][next_state] + gamma * V[next_state])
['U' 'U' 'U' 'U']
['U' 'U' 'U' 'U']
['U' 'U' 'U' 'U']
['U' 'U' 'U' 'U']
['U' 'U' 'U' 'U']
PS C:\py\marathon>  c:; cd 'c:\py\marathon'; & 'C:\Program Files\Python311\python.exe' 'c:\Users\stafme\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/.
./..\debugpy\launcher' '55234' '--' 'C:\py\DP(pollicy iteration).py'
C:\py\DP(pollicy iteration).py:69: RuntimeWarning: overflow encountered in scalar add
  V[s] += p[s][a][next_state]*(r[s][a][next_state] + gamma * V[next_state])
C:\py\DP(pollicy iteration).py:69: RuntimeWarning: invalid value encountered in scalar multiply
  V[s] += p[s][a][next_state]*(r[s][a][next_state] + gamma * V[next_state])
['U' 'U' 'U' 'U']
['U' 'U' 'U' 'U']
['U' 'U' 'U' 'U']
['U' 'U' 'U' 'U']
['U' 'U' 'U' 'U']
PS C:\py\marathon>
```

Рис. 2.1 – Результат роботи програми для машинного навчання за методом DP (Policy Iteration)

DQN

Нехай як модель функції вартості стану-дії обрано багатошаровий персептрон вигляду

$$z_j^{(0)} = s_j,\ j \in \overline{1, S^{(0)}}\ ,$$

$$z_k^{(l)} = f^{(l)}\left(b_k^{(l)} + \sum_{j=1}^{S^{(l-1)}} w_{jk}^{(l)} z_j^{(k-1)} \right), k \in \overline{1, S^{(l)}}, l \in \overline{1, L-1}\ ,$$

$$y_k = z_k^{(L)} = b_k^{(L)} + \sum_{j=1}^{S^{(L-1)}} w_{jk}^{(L)} z_j^{(L-1)}, k \in \overline{1, S^{(L)}},$$

де $S^{(l)}$ - кількість нейронів у l -му шарі,

L - кількість шарів,

$b_k^{(l)}$ - зміщення (пороги),

$w_{kj}^{(l)}$ - ваги,

$f^{(l)}$ - функція активації.

Зазвичай обмежуються двома прихованими шарами, причому зазвичай кількість нейронів у прихованому шарі збігається з кількістю нейронів у вхідному шарі.

Для методу DQN нейромережу можна представити як

$$y_k = Q_\theta(\mathbf{s}, k), k \in \overline{1, S^{(L)}},$$

де $\theta = (b_1^{(1)}, ..., b_{S^{(L)}}^{(L)}, w_{11}^{(1)}, ..., w_{S^{(L-1)}S^{(L)}}^{(L)})$ - вектор параметрів.

Світ плиток складається з таких клітин - плитка, пастка (дірка), цільова (наприклад, нижня права клітина).

Для цього методу кожен стан *s* відповідає клітині світу плиток, у якій перебуває агент.

Стосовно світу плиток:

$S^{(0)}$ - кількість клітин світу плиток, , $S^{(0)} = |S| |S| = height * width$,

$S^{(L)}$ - кількість можливих дій агента, $S^{(L)} = |A|$,

$\mathbf{s}$ - бінарний вектор стану з однією 1 (1 відповідає *s*),

$\mathbf{y}$ - вектор значень функції вартості стану-дії,

$height$ - висота світу плиток у клітинах,

$width$ - ширина світу плиток у клітинах,

$Q_\theta(\mathbf{s}, k)$ - функція вартості стану-дії.

Метод складається з таких етапів:

1. ініціалізація

1.1 Ініціалізація вектора параметрів нейромережі θ за допомогою рівномірного розподілу на інтервалі (0,1) або [-0.5, 0.5].

1.2 Задається дискретна множина станів (клітин) S

1.3 Задано дискретну множину дій A

1.4 Задається параметр ε для ε-жадібної політики, $0 < \varepsilon < 1$, коефіцієнт дисконтування γ, $0 < \gamma < 1$.

1.5 Ініціалізується винагорода $R(s, a)$.

Наприклад, використовуються такі правила:

- якщо агент перейшов у клітку-пастку (дірку), то $R(s, a) = -1$;

- якщо агент перейшов у клітку-плитку, то $R(s, a) = 0$;

- якщо агент перейшов у цільову клітину, то $R(s, a) = 1$.

2. Номер ітерації $n = 1$.

3. Номер моменту часу $t = 1$.

4. Спостерігається початковий стан (клітина) s_t , який перетворюється на вектор початкового стану $\mathbf{s}_t$

5. Обирається дія a_t для переходу з вектора стану $\mathbf{s}_t$, використовуючи ε-жадібну політику (якщо $U(0,1) < \varepsilon$, то вибрати дію a_t випадковим чином із множини дій A, інакше $a_t = \arg\max_{c \in A} Q_\theta(\mathbf{s}_t, c)$).

6. Спостерігається винагорода $R(s_t, a_t)$ і новий стан (клітина) s_{t+1} , який перетворюється на вектор нового стану $\mathbf{s}_{t+1}$

Наприклад, у разі чотирьох дій агента (рухи вгору, вниз, вліво, вправо) використовуються такі правила:

- якщо дія a_t відповідає руху вгору і стан s_t не відповідає клітині в першому рядку карти, то агент переходить у стан s_{t+1} , який відповідає клітині зверху s_t ;

- якщо дія a_t відповідає руху вниз і стан s_t не відповідає клітині в останньому рядку карти, то агент переходить у стан s_{t+1} , який відповідає клітині знизу s_t ;

- якщо дія a_t відповідає руху ліворуч і стан s_t не відповідає клітині в першому стовпчику карти, то агент переходить у стан s_{t+1} , що відповідає клітині ліворуч s_t ;

- якщо дія a_t відповідає руху праворуч і стан s_t не відповідає клітині в останньому стовпчику карти, то агент переходить у стан s_{t+1} , який відповідає клітині праворуч s_t .

7. Обчислюється Q цільове значення:

- якщо агент перейшов у клітку-плитку, то, використовуючи жадібну політику

$$G_\theta(s_t, a_t) = R(s_t, a_t) + \gamma \max_{c \in A} Q_\theta(\mathbf{s}_{t+1}, c)$$

- якщо агент перейшов у цільову клітину або клітину-пастку (дірку), то

$$G_\theta(s_t, a_t) = R(s_t, a_t)$$

8. Обчислюється вектор значень функції вартості стану-дії

$$d_{tk} = \begin{cases} G_\theta(s_t, a_t), & k = a_t \\ Q_\theta(\mathbf{s}_t, k), & k \neq a_t \end{cases}, \ k \in A$$

9. Навчається нейромережа (визначається вектор параметрів θ) на одній епосі навчання (наприклад, на основі методу градієнтного спуску), причому навчальними вхідними даними є вектор $\mathbf{s}_t$, а навчальними вихідними даними є вектор $\mathbf{d}_t$

$$\theta = \theta + \eta \nabla_{\theta} \left(\mathbf{d}_t - Q_{\theta}(s_t, a_t) \right)^2$$

10. Якщо клітина, у яку перейшов агент, є плиткою, то $t = t + 1$, перехід на крок 5.

11. якщо поточна ітерація не є останньою, тобто $n < N$, то збільшити номер ітерації, тобто $n = n + 1$, перехід на крок 3, інакше зупинка.

Зауваження. $U(0,1)$ - функція, що повертає рівномірно розподілене випадкове число в діапазоні $[0,1]$.

Зауваження. У традиційному методі DQN навчання відбувалося на міні-пакеті, обраного випадковим чином з усієї множини попередніх пар {(, $\mathbf{s}_t$ $\mathbf{d}_t$)} *m-ї* ітерації, а не на поточній парі (,). $\mathbf{s}_t$ $\mathbf{d}_t$

```
import numpy, tensorflow

EPISODES = 50 # кількість епізодів
T = 200 # тривалість епізоду
ROAD = 0 # вільно
HOLE = 1 # пастка
GOAL = 2 # ціль
# карта світу плиток
MYMAP = [[ROAD,HOLE,ROAD,ROAD,ROAD],
         [ROAD,HOLE,ROAD,HOLE,ROAD],
         [ROAD,ROAD,ROAD,HOLE,ROAD],
         [ROAD,ROAD,ROAD,HOLE,ROAD],
         [ROAD,ROAD,ROAD,HOLE,GOAL]]
Y_MYMAP = 5 # висота карти
X_MYMAP = 5 # ширина карти
ActionName = ["UP", "DOWN", "LEFT", "RIGHT"]

## Обчислення винагороди в залежності від стану наступної клітинки
def get_reward_of_state(state):
    if MYMAP[state // X_MYMAP][state % X_MYMAP] == HOLE:
        reward = -1
    elif MYMAP[state // X_MYMAP][state % X_MYMAP] == ROAD:
        reward = 0
    elif MYMAP[state // X_MYMAP][state % X_MYMAP] == GOAL:
        reward = 1
    return reward

## Переміщення по карті в залежності від дії і стану поточної клітинки
def env_step(action, state):
    done = False # флаг завершення
    # якщо UP і не в першому рядку карти
    if (action == 0) and (state // X_MYMAP > 0):
        state = state - X_MYMAP
    # якщо DOWN і не в останньому рядку карти
```

```
        elif (action == 1) and (state // X_MYMAP < Y_MYMAP-1):
            state = state + X_MYMAP
        # якщо LEFT і не в першому стовбчику карти
        elif (action == 2) and (state % X_MYMAP > 0):
            state = state - 1
        # якщо RIGHT і не в останньому стовбчику карти
        elif (action == 3) and (state % X_MYMAP < X_MYMAP-1):
            state = state + 1
        # обчислення винагороди
        reward = get_reward_of_state(state)
        # якщо досягнуто цілі або потраплено у пастку
        if MYMAP[state // X_MYMAP][state % X_MYMAP] == GOAL or MYMAP[state //
X_MYMAP][state % X_MYMAP] == HOLE:
            done = True
        return state, reward, done

    class DQNAgent:
        def __init__(self, state_size, action_size):
            self.state_size =  state_size # кількість клітинок
            self.action_size = action_size # кількість дій
            self.gamma = 0.1
            self.epsilon = 0.9
            self.min = 0.1
            self.decay = 0.995
            self.model = self._build_model()

        ## створення моделі нейромережі (MLP з двома прихованими слоями)
        def _build_model(self):
            inputs = tensorflow.keras.Input(shape=(self.state_size,))
            hidden1 = tensorflow.keras.layers.Dense(units=self.state_size, activation="relu")(inputs)
            hidden2 = tensorflow.keras.layers.Dense(units=self.state_size,
activation="relu")(hidden1)
            outputs = tensorflow.keras.layers.Dense(units=self.action_size,
activation="linear")(hidden2)
            mlp = tensorflow.keras.Model(inputs=inputs, outputs=outputs)
            mlp.compile(optimizer="Adam", loss="mse")
            return mlp

        ## epsilon-жадібний підхід
        def act(self, flat_states):
            if numpy.random.rand() < self.epsilon:
                # випадковий вибір дії
                return numpy.random.choice(self.action_size)
            else:
                # обчислення дії за моделлю нейромережі
                act_values = self.model.predict(flat_states)
                return numpy.argmax(act_values[0])

        ## Обчислення DQN моделі нейромережі, оновлення параметрів
        def replay(self, flat_states, action, reward, next_flat_states, done):
            # обчислення Q
            target_f = self.model.predict(flat_states)
            if done:
                target = reward
            else:
                target = reward + self.gamma * numpy.max(self.model.predict(next_flat_states)[0])
            target_f[0][action] = target
            # навчання моделі нейромережі
```

```
            self.model.fit(x=flat_states, y=target_f, batch_size=1, epochs=1, verbose=0,
shuffle=True, initial_epoch=0, steps_per_epoch=None)
            # оновлення параметрів
            if self.epsilon > self.min:
                self.epsilon *= self.decay
            self.gamma = 1.0-self.epsilon

    state_size = X_MYMAP * Y_MYMAP # кількість клітинок
    action_size = len(ActionName) # кількість дій
    done = False # флаг завершення
    agent = DQNAgent(state_size, action_size)
    for e in range(EPISODES):
        state = 0  # крайня верхня ліва клітинка
        # перетворення стану в бінарний вектор з формою (1,25)
        flat_states = numpy.zeros((1, state_size))
        flat_states[0][state] = 1
        for t in range(T):
            # epsilon-жадібний підхід для вибору дії
            action = agent.act(flat_states)
            # Переміщення по карті в залежності від дії і стану поточної клітинки
            next_state, reward, done = env_step(action, state)
            # перетворення стану в бінарний вектор з формою (1,25)
            next_flat_states = numpy.zeros((1, state_size))
            next_flat_states[0][next_state] = 1
            # обчислення Q, навчання моделі нейромережі, оновлення параметрів
            agent.replay(flat_states, action, reward, next_flat_states, done)
            if ~done:
                state = next_state # оновлення позиції агента
                flat_states = next_flat_states # оновлення карти
            if done:
                print(reward, e, t)
                break
```

```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 16ms/step
-1 48 42
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 14ms/step
```

Рис. 2.2 – Результат роботи програми для машинного навчання за методом DQN

SARSA

Світ плиток складається з таких клітин - плитка, пастка (дірка), цільова (наприклад, нижня права клітина).

Для цього методу кожен стан *s* відповідає клітині світу плиток, у якій перебуває агент.

Стосовно світу плиток:

$|A|$ - кількість можливих дій агента,

$|S|$ - кількість клітин світу плиток, $|S| = height * width$,

$height$ - висота світу плиток у клітинах,

$width$ - ширина світу плиток у клітинах,

Метод складається з таких етапів:

1. ініціалізація

1.1 Задається параметр ρ (керує швидкістю навчання), $0 < \rho < 1$, параметр ε для ε-жадібної політики, $0 < \varepsilon < 1$, коефіцієнт дисконтування γ, $0 < \gamma < 1$.

1.2 Задається дискретна множина станів (клітин) *S*

1.3 Задано дискретну множину дій *A*

1.4 Ініціалізуються значення функції вартості стану-дії , , $Q = [Q(s,a)]$ $Q(s,a) = U(0,1)$ $s \in S$ $a \in A$,.

1.5 Ініціалізується винагорода $R(s,a)$.

Наприклад, використовуються такі правила:

- якщо агент перейшов у клітку-пастку (дірку), то $R(s,a) = -1$;

- якщо агент перейшов у клітку-плитку, то $R(s,a) = 0$;

- якщо агент перейшов у цільову клітину, то $R(s,a) = 1$.

2. Номер ітерації n=1

3. спостерігається початковий стан (клітина) s

4. Обирається дія *a*, за якою потрібно переміститися з клітини *s*, використовуючи ε-жадібну політику (якщо $U(0,1) < \varepsilon$, то вибрати *a* випадковим чином із множини дій *A*, інакше вибрати дію *a* як із множини дій *A*, тобто $a = \arg\max_b Q(s,b)$,). $b \in A$

5. Спостерігається винагорода $R(s,a)$ і новий стан (клітка) s'

Наприклад, у разі чотирьох дій агента (рухи вгору, вниз, вліво, вправо) використовуються такі правила:

- якщо дія *a* відповідає руху вгору і стан s не відповідає клітині в першому рядку карти, то агент переходить у стан s', який відповідає клітині зверху s ;

- якщо дія *a* відповідає руху вниз і стан s не відповідає клітині в останньому рядку карти, то агент переходить у стан s', що відповідає клітині знизу s ;

- якщо дія *a* відповідає руху ліворуч і стан s не відповідає клітині в першому стовпчику карти, то агент переходить у стан s', що відповідає клітині ліворуч s ;

- якщо дія *a* відповідає руху праворуч і стан s не відповідає клітині в останньому стовпчику карти, то агент переходить у стан s', що відповідає клітині праворуч s .

6. Вибирається дія a', за якою потрібно переміститися з клітини s', використовуючи ε-жадібну політику (якщо $U(0,1) < \varepsilon$, то вибрати a' випадковим чином із множини дій *A*, інакше вибрати дію a' як із множини дій *A*, тобто $a' = \arg\max_b Q(s',b)$,). $b \in A$

7. Обчислюється Q цільове значення:

- якщо агент перейшов у клітку-плитку, то

$$G(s,a) = R(s,a) + \gamma Q(s',a')$$

- якщо агент перейшов у цільову клітину або клітину-пастку (дірку), то

$$G(s,a) = R(s,a)$$

8. Обчислюється значення функції вартості стану-дії $Q(s,a)$ у вигляді

$$Q(s,a) = (1-\rho)Q(s,a) + \rho G(s,a)$$

9. Встановлюється поточний стан (клітина) $s = s'$, поточна дія $a = a'$.

10. Якщо поточна клітина s є плиткою, то перехід на крок 5.

11. якщо поточна ітерація не є останньою, тобто $n < N$, то збільшити номер ітерації, тобто $n = n+1$, перехід на крок 3, інакше зупинка

Зауваження. $U(0,1)$ - функція, що повертає рівномірно розподілене випадкове число в діапазоні $[0,1]$.

```
import numpy

gamma = 0.9
lr = 0.1
epsilon = 0.1
EPISODES = 40 # кількість епізодів
T = 150 # тривалість епізоду
ROAD = 0 # вільно
HOLE = 1 # пастка
GOAL = 2 # ціль
# карта світу плиток
MYMAP = [[ROAD,HOLE,ROAD,ROAD,ROAD],
            [ROAD,HOLE,ROAD,ROAD,ROAD],
            [ROAD,HOLE,ROAD,HOLE,ROAD],
            [ROAD,HOLE,ROAD,HOLE,ROAD],
            [ROAD,ROAD,ROAD,HOLE,GOAL]]
Y_MYMAP = 5 # висота карти
X_MYMAP = 5 # ширина карти
ActionName = ["UP", "DOWN", "LEFT", "RIGHT"]

## Обчислення винагороди в залежності від стану наступної клітинки
def get_reward_of_state(gState):
  if MYMAP[gState // X_MYMAP][gState % X_MYMAP] == HOLE:
    reward = -1
  elif MYMAP[gState // X_MYMAP][gState % X_MYMAP] == ROAD:
    reward = 0
  elif MYMAP[gState // X_MYMAP][gState % X_MYMAP] == GOAL:
    reward = 1
  return reward

## Переміщення по карті в залежності від дії і стану поточної клітинки
```

```
def env_step(action, state):
    done = False # флаг завершення
    # якщо UP і не в першому рядку карти
    if (action == 0) and (state // X_MYMAP > 0):
        state = state - X_MYMAP
    # якщо DOWN і не в останньому рядку карти
    elif (action == 1) and (state // X_MYMAP < Y_MYMAP-1):
        state = state + X_MYMAP
    # якщо LEFT і не в першому стовбчику карти
    elif (action == 2) and (state % X_MYMAP > 0):
        state = state - 1
    # якщо RIGHT і не в останньому стовбчику карти
    elif (action == 3) and (state % X_MYMAP < X_MYMAP-1):
        state = state + 1
    # обчислення винагороди
    reward = get_reward_of_state(state)
    # якщо досягнуто цілі або потраплено у пастку
    if MYMAP[state // X_MYMAP][state % X_MYMAP] == GOAL or MYMAP[state // X_MYMAP][state % X_MYMAP] == HOLE:
        done = True
    return state, reward, done

## epsilon-жадібний підхід
def act(Q_table, state):
    if numpy.random.rand() < epsilon:
        # випадковий вибір дії
        return numpy.random.choice(action_size)
    else:
        # жадібний вибір дії
        return numpy.argmax(Q_table[int(state), :])

state_size = X_MYMAP * Y_MYMAP # кількість клітинок
action_size = len(ActionName) # кількість дій
done = False # флаг завершення
Q_table = numpy.random.rand(state_size, action_size)
for e in range(EPISODES):
    state = 0  # крайня верхня ліва клітинка
    # epsilon- жадібний підхід для вибору дії
    action = act(Q_table, state)
    # генерація епізоду
    for t in range(T):
        # Переміщення по карті в залежності від дії і стану поточної клітинки
        next_state, reward, done = env_step(action, state)
        # epsilon- жадібний підхід для вибору дії
        next_action = act(Q_table, next_state)
        # оновлення Q
        Q_table[int(state), int(action)] = (1-lr)*Q_table[int(state), int(action)] + lr * (reward + gamma * Q_table[int(next_state), int(next_action)])
        if ~done:
            state = next_state # оновлення позиції агента
            action = next_action # оновлення деї агента
        if done:
            print(reward, e, t)
            break
```

```
-1 0 55
-1 1 9
-1 2 27
-1 3 27
-1 4 19
-1 5 6
-1 6 7
-1 7 35
-1 8 0
-1 9 8
-1 10 56
-1 11 49
-1 12 11
-1 13 23
-1 14 91
-1 15 4
-1 16 51
-1 17 0
-1 18 19
-1 19 15
-1 20 9
-1 21 20
-1 22 21
-1 23 13
-1 24 67
-1 25 13
-1 26 130
-1 27 38
-1 28 72
-1 29 12
-1 31 3
-1 32 79
-1 33 82
-1 34 5
-1 35 20
-1 36 85
-1 37 8
-1 38 9
-1 39 20
```

Рис. 2.3 – Результат роботи програми для машинного навчання за методом SARSA

2.3 Алгоритми створення найкоротших шляхів

Алгоритм пошуку в ширину

Для ініціалізації алгоритму спочатку створюється черга, в яку послідовно додаються вже відвідані вершини. Крім того, для визначення шляху назад у випадку пройдення всіх вершин графа з одного напрямку формується масив попередників, де зберігаються шляхи до поточної вершини. Основна концепція алгоритму полягає в послідовному обході вершин: для кожної вершини шукається її нащадок, якщо він існує, відбувається перехід до цього нащадка; у випадку відсутності нащадка та невиконаної задачі пошуку, алгоритм повертається на попередню вершину. [28]. Математично алгоритм можна записати наступним чином:

1) У графі S присутні вершини, що утворюють конкретну множину. Позначимо множину вершин як V та множину ребер як T. В графі також існує особлива вершина M, яка є найвищою серед усіх. Для позначення шляху між вершиною M та іншою вершиною X використовується позначення P, тобто шлях буде позначений як P(M,X). Щоб вказати кількість ребер між цими вершинами, введено позначення D, де довжина шляху позначається як D(M,X).

2) На початку роботи алгоритму відстань від початкової вершини дорівнює $D = 0$, а до всіх інших вершин $D = \infty$.

3) Будується множина вершин і до відстаней додається кількість пройдених вершин.

4) Якщо множина вершин, які складають граф, буде дорівнювати 0, то алгоритм закінчує свою роботу.

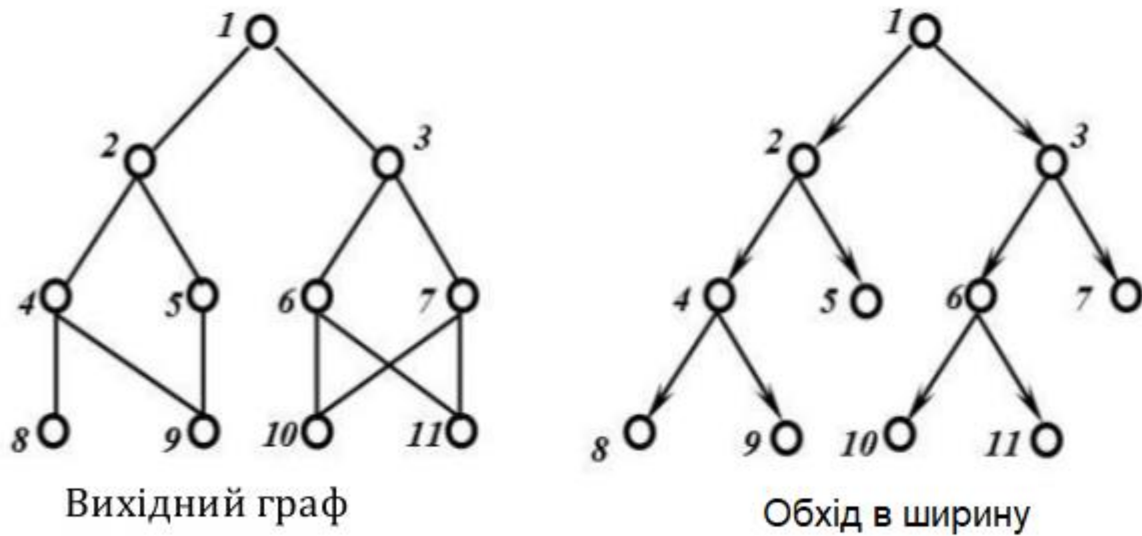

Рис. 2.4 - Алгоритм обходу в ширину

Алгоритм пошуку в глибину

Пошук в глибину подібний до пошуку в ширину, але основна відмінність полягає в тому, що в пошуку в глибину алгоритм спочатку просувається вглиб графа від початкової вершини до досягнення максимальної глибини, а потім, коли він доходить до кінцевої вершини, перевіряє, чи є ще невідомі вершини. Якщо такі вершини існують, алгоритм повертається назад до останньої розглянутої вершини і продовжує пошук нових шляхів вниз. [29]. Математично алгоритм можна записати наступним чином:

1) Нехай заданий граф $G = (V, E)$, в якому початкова найвища вершина позначена як M, V – множина вершин, E – множина ребер. Шлях між двома будь-якими вершинами записується як $P(M, L)$, а довжина шляху - $D(M, L)$. Довжина шляху показує кількість ребер між вершинами M, L.

2) Вводиться поняття підмножини вершин X, таке що $X \in V$. Відбувається перехід по всім вершинам підмножини X.

3) Якщо залишились непройдені вершини, алгоритм підіймається вгору, та обходить вершини іншої підмножини.

4) Алгоритм закінчує роботу, коли множина всіх вершин буде дорівнювати 0.

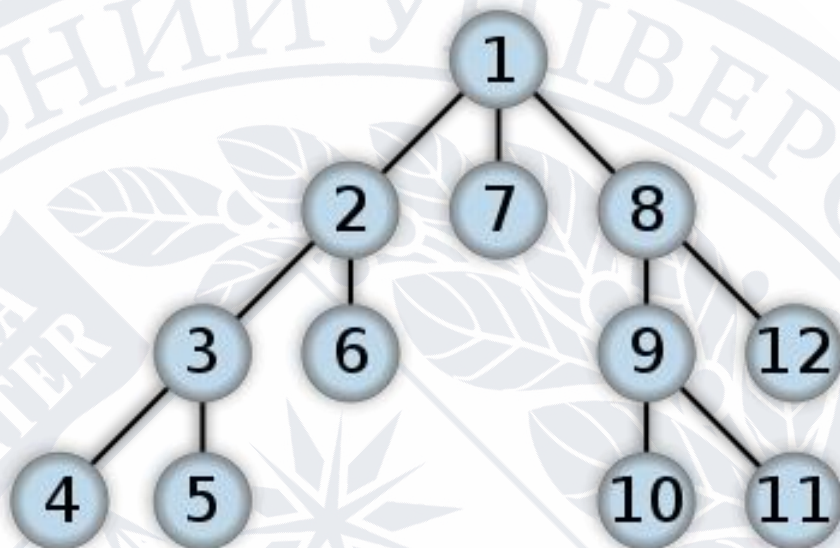

Рис. 2.5 - Алгоритм обходу в глибину

Алгоритм Дейкстри

Даний алгоритм дозволяє знаходити найбільш оптимальні маршрути в графі, де спочатку визначається початкова вершина, а шляхи до всіх інших вершин є невідомими. Однак його основним обмеженням є можливість використання лише в графах, де всі ребра мають додатні ваги. У реальному житті існують випадки, коли певні маршрути для компанії можуть бути неоптимальними з фінансової точки зору (і, звісно, такі маршрути варто відкидати). Однак, якщо це є шляхи важливі для бізнесу, і вибір інших маршрутів не є варіантом, алгоритм Дейкстри не може бути застосованим у таких ситуаціях. Алгоритм Дейкстри ідеально підходить для вирішення задач прокладання маршрутів, оскільки його використання передбачає використання зважених графів (де для кожного ребра встановлюється вага, наприклад, платні автомагістралі) та орієнтованих графів (де кожне ребро має напрямок). [30]. В процесі роботи алгоритму необхідно використовувати три множини: V1 – множина вершин, до яких відстань вже була підрахована, V2 –

множина вершин , до яких відстань ще рахується, V3 – множина вершин, відстань до яких ще не рахувалась. Алгоритм працює наступним чином:

1) Вибирається вершина з мінімальною вагою серед множини вершин.

2) Порівнюється ваги сусідніх вершин, сусідня вершина з мінімальною вагою вибирається для переходу. Вершина, з якої був виконаний перехід запам'ятовується.

3) Для вершини, в яку був зроблений перехід, виконується крок 2.

4) Алгоритм закінчує свою роботу, коли сусідніх вершин більше немає, тобто множина вершин = 0.

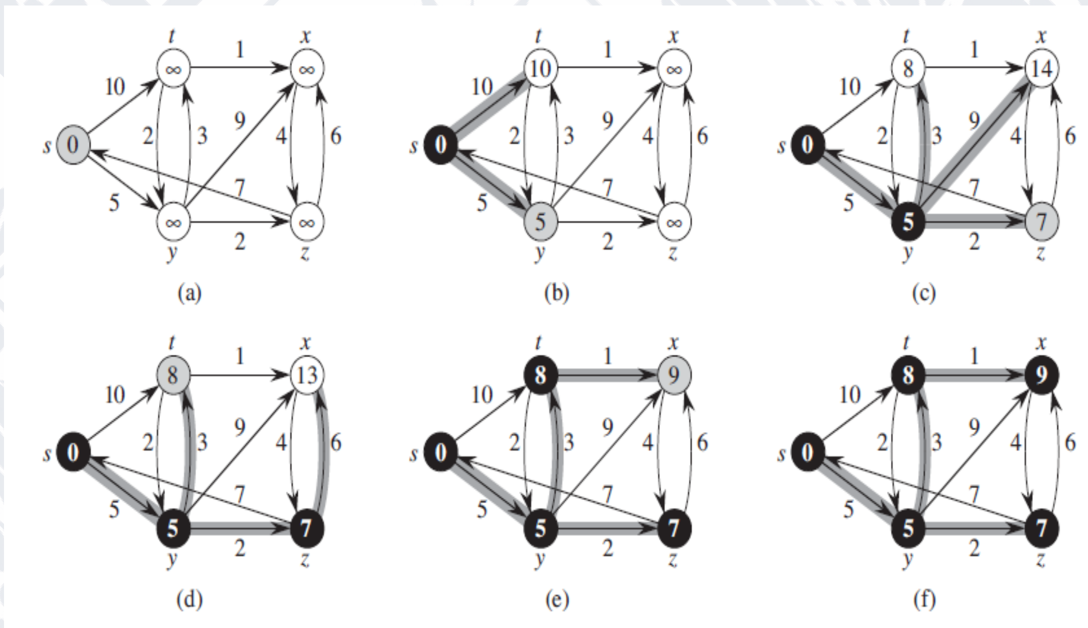

Рис. 2.6 - Алгоритм Дейкстри

Алгоритм Беллмана-Форда

Алгоритм Белмана-Форда подібний до алгоритму Дейкстри, але його основною перевагою є можливість обробки ребер з від'ємними вагами. У своїй роботі алгоритм Белмана-Форда використовує метод динамічного програмування, що означає розбиття основної задачі на підзадачі. Для кожної підзадачі шукається рішення, і результати підзадач потім об'єднуються для вирішення основної задачі. Однією з основних переваг динамічного

програмування є його зрозумілість і звичність для людини, оскільки ми звикли розбивати великі задачі на менші частини. Опис алгоритму:

1) Нумерація всіх вершин графа для того, щоб показати, з якої вершини буде починатися пошук.

2) Формується двовимірний масив, який буде зберігати інформацію про найкоротші шляхи. Масив є двовимірним, тому що він зберігає інформацію про вагу ребра до вершини та номер даної вершини.

3) На початку роботи алгоритму всі елементи двовимірного масиву, крім першої строки, дорівнюють нескінченості. Перша строчка, в свою чергу, складається з нулів, тому що відстань від вибраної, початкової першини до самої себе дорівнює нулю.

4) На кожному кроці алгоритм виконує релаксацію ребер, тобто спроба покращити значення вершини, за допомогою ребра з іншою вершиною

5) Для того, щоб підрахувати мінімальні ваги всіх ребер, вважається, що алгоритму необхідно n – 1 фаз.

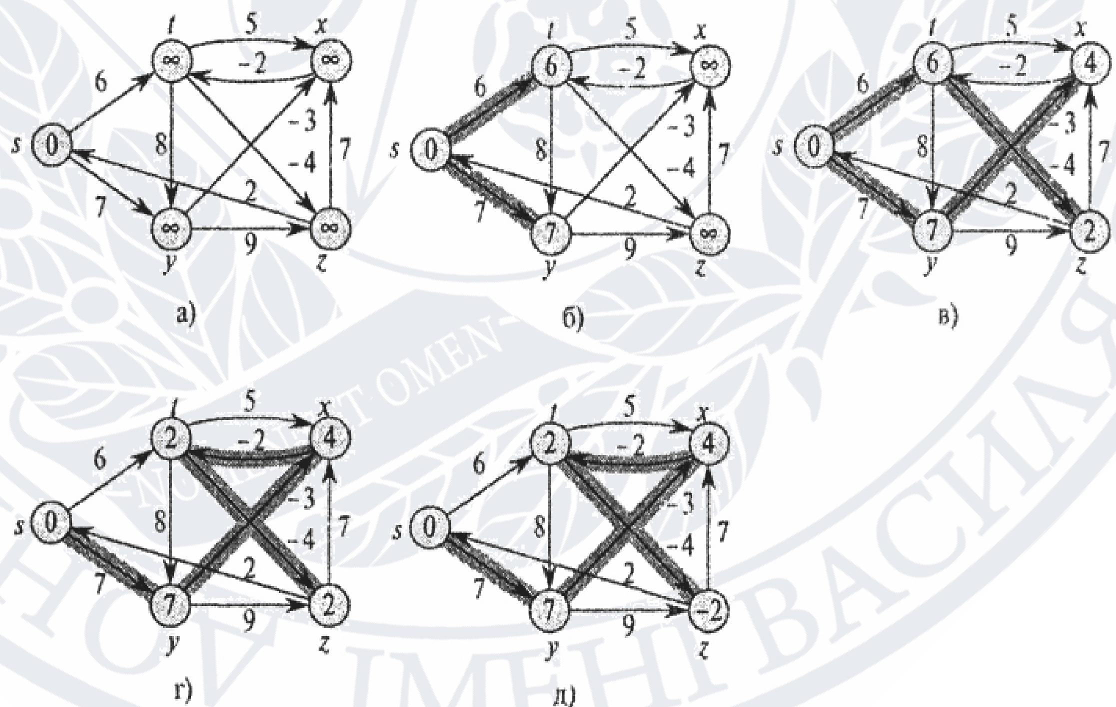

Рис. 2.7 - Алгоритм Беллмана-Форда

Алгоритм Флойда-Уоршела

Алгоритм, який використовує динамічне програмування, призначений для пошуку найбільш оптимальних шляхів між усіма вершинами в зваженому орієнтованому графі. Він працює коректно лише у випадку, якщо в графі відсутні цикли з від'ємними вагами. Якщо такий цикл присутній, алгоритм все одно може знайти хоча б один від'ємний цикл. Час роботи алгоритму $O\ (n^3)$, використання пам'яті $O(n^2)$. Алгоритм Флойда-Уоршела виявляється більш загальним у порівнянні з алгоритмом Дейкстри, оскільки він спроможний працювати з графами, що мають від'ємні цикли, і може виявляти ці цикли під час своєї роботи. Математична модель алгоритму:

1) Маємо граф $G = (V, E)$, в якому кожна вершина є пронумерованою від 1 до кількості вершин. Формуємо матрицю суміжності D, розмір такої матриці це квадрат розмірності початкового графа G, кожному елементу матриці суміжності проставлена вага ребра, суміжних вершин.

2) Під час роботи алгоритму, оптимальна вага даного ребра постійно змінюється за допомогою релаксації елементів матриці суміжності: якщо $D_{ik} + D_{kj} < D_{ij}$, то вага ребра D_{ij} змінюється на суму тимчасових ваг ребер $D_{ik} + D_{jk}$.

3) Алгоритм закінчує свою роботу, коли ваги всіх ребер є мінімальним і для кожного ребра не виконується умова: $D_{ik} + D_{kj} < D_{ij}$.

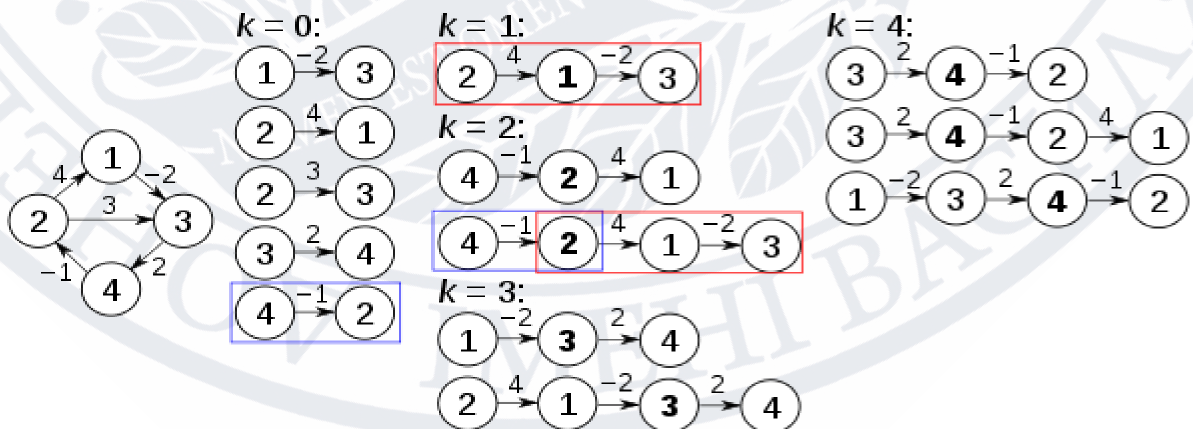

Рис. 2.8 - Алгоритм Флойда-Уоршела

Алгоритм А*

Алгоритм А* (зірка) базується на моделі пошуку за першим найкращим збігом. Це означає, що алгоритм розглядає граф та розширює найбільш перспективні вузли, вибрані за певними правилами.

Основний процес алгоритму включає визначення, який зі шляхів розширювати на кожному кроці. Це визначається за допомогою відомої вартості поточного шляху та оцінки вартості розширення шляху до цільової точки. Однією з головних вимог алгоритму А* є допустимість евристичної функції, яка повинна правильно оцінювати відстань до цільової точки, не переоцінюючи її.

Опис кроків алгоритму:

1. Покроково розглядаються всі можливі шляхи від початкової вершини до цільової, при цьому використовується евристична функція для вибору найбільш перспективних маршрутів.

2. Обираються суміжні вузли з найменшим значенням функції F(n), і дана вершина відзначається як відвідана.

3. На кожному етапі алгоритм працює з вершинами, які ще не були відвідані і знаходяться у черзі з пріоритетом.

4. Алгоритм продовжує свою роботу, доки значення функції F(n) для цільової вершини не буде мінімальним серед усіх значень у черзі або доки алгоритм не пройде весь граф. Результатом є рішення з найменшою вартістю серед усіх розглянутих шляхів.

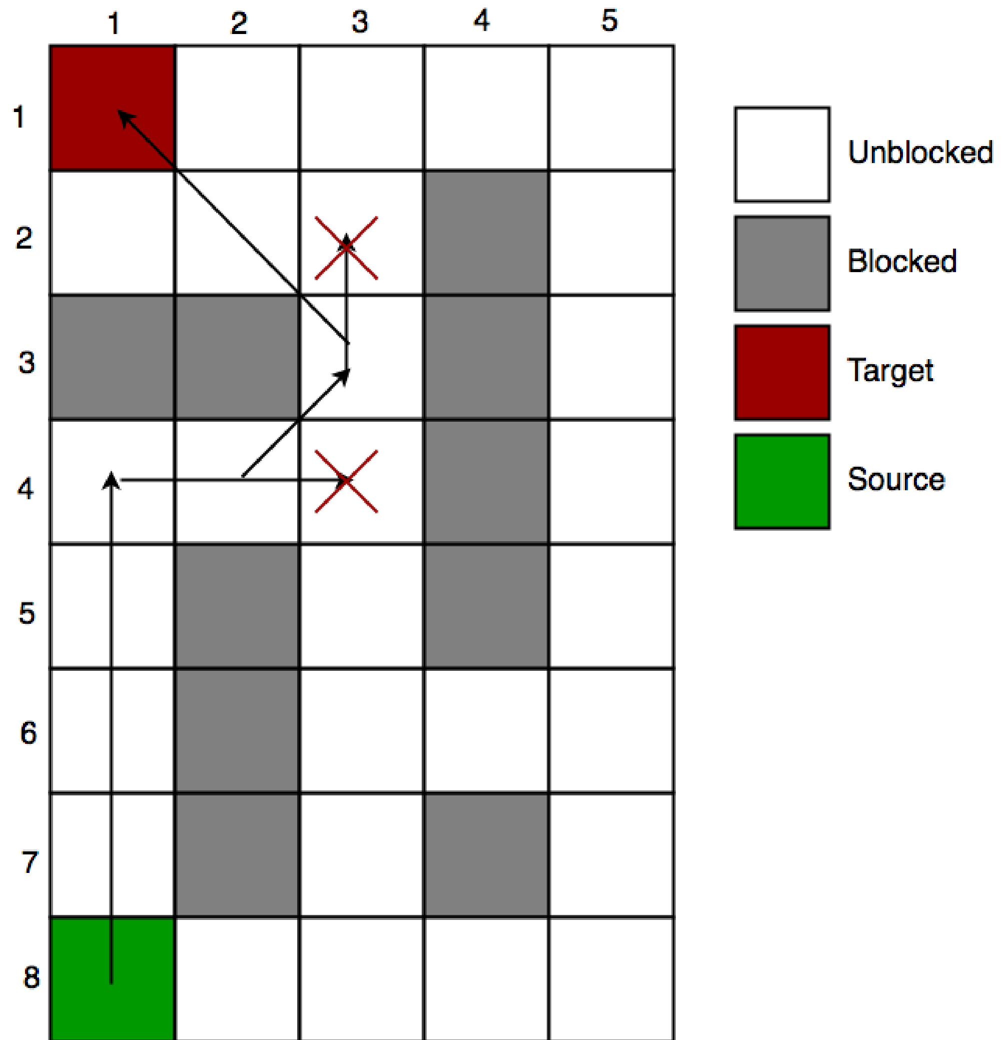

Рис. 2.9 - Алгоритм A*

IDA*

Алгоритм Iterative Deepening A* (IDA*) представляє собою модифікацію пошуку в глибину з ітераційним заглибленням (IDDFS). На кожній ітерації IDA* визначає максимальний поріг для функції оцінки $f(v)$. Потім алгоритм шукає шлях до того часу, поки функція оцінки для даного шляху не перевищить встановлений поріг. Коли поріг досягнутий, шлях "збрасується", повертаючись до попередньої вершини, яка має непройденого сусіда. Алгоритм продовжує розглядати шляхи від цього сусіда, завершуючи ітерацію, коли всі можливі шляхи досягли встановленого порогу.

Після завершення ітерації IDA* встановлює новий поріг для наступної ітерації, який є мінімумом серед усіх знайдених значень, що перевищували поточний поріг. Цей процес повторюється до знаходження цільового вузла.

Хоча виглядає, що IDA* здійснює багато зайвих обчислень, розширюючи вузли кілька разів, фактичні накладні витрати залишаються невеликими. Це пояснюється тим, що зовнішній шар розширених вузлів на останній ітерації містить значно більше вузлів. Факт того, що інші шари розширюються кілька разів, не має значущого впливу. IDA* забезпечує оптимальне рішення з обмеженими ресурсами пам'яті, використовуючи ті ж обмеження евристики, що і A*. [5].

SMA*

Недолік у методі IDA* полягає у видаленні значної кількості інформації після кожної ітерації. Один з методів, спрямованих на максимальне збереження інформації, відомий як Memory Bounded A* або MA* [6]. Впроваджений Расселом [7] метод SMA* вдосконалює MA*, дозволяючи зберігати ще менше інформації і спрощуючи реалізацію алгоритму.

SMA* функціонує приблизно так само, як A*, за винятком обмеження кількості вузлів, які можуть зберігатися в пам'яті. Якщо це обмеження досягнуте, алгоритм видаляє листовий вузол з пам'яті лише тоді, коли простір необхідний для кращого вузла. При видаленні вузла видаляється той, що має найменше значення $f(v)$ серед всіх вузлів.

Щоб уникнути непотрібної втрати інформації, дані про нащадків зберігаються в попередніх вузлах. Після розширення всіх нащадків вузла v, значення $f(v)$ оновлюється найменшим значенням серед всіх нащадків, і ця інформація оновлюється для всіх предків v.

Теорема 4 в [7] стверджує, що, крім здатності генерувати окремі послідовності, SMA* веде себе так само, як A*, коли кількість вузлів, що генеруються A*, не перевищує заданого ліміту. SMA* забезпечує оптимальний шлях, якщо є достатньо пам'яті для зберігання найменшого оптимального рішення.

ALT алгоритм

Гольдберг і Харрелсон [10] описують метод покращення ефективності пошуку найкоротших шляхів, який вони назвали алгоритмом ALT (A* search, Landmarks and Triangle inequality), оскільки він ґрунтується на A*, використовує визначення орієнтирів і нерівності трикутника для найкоротших відстаней (не евклідових). Мета методу - знайти кращу нижню межу для відстані $h(v)$, яку використовує алгоритм A*. Більш низькі межі призводять до меншої кількості вузлів, які необхідно обробити під час пошуку.

Алгоритм ALT вибирає низку орієнтирів під час передпошуку. Для кожного вузла обчислюється відстань до цих орієнтирів. Нехай $d(u, v)$ - це відстань від будь-якого вузла u до будь-якого вузла v. Тоді за нерівністю трикутника, $d(u, L) - d(v, L) \leq d(v, u)$. Також, $d(L, v) - d(L, u) \leq d(v, u)$.

Оскільки $d(w, L)$ і $d(L, w)$ обчислені під час передпошуку, цей факт можна використовувати для обчислення нижньої межі відстані між двома вузлами. Для досягнення найнижчої нижньої межі обирається максимум по всіх орієнтирів. Як тільки нижня межа знайдена, вона використовується як евристика для $\hat{h}(v)$ в алгоритмі A*.

Вибір найкращих орієнтирів є ключовим аспектом для загальної ефективності алгоритмів ALT. Гольдберг та ін. описують три методи вибору орієнтирів. Перший, описаний метод, просто обирає k вузлів випадковим чином. Однак є кращі методи.

Другий метод - вибір найвіддаленішого орієнтира. Алгоритм починає роботу, обираючи один випадковий вузол і вузол, що знаходиться якнайдалі від нього, як перший орієнтир. Потім для кожного нового орієнтиру знаходиться найвіддаленіший вузол від поточного набору орієнтирів.

У випадку дорожніх карт, наявність орієнтира, який знаходиться за пунктом призначення, допомагає отримати непогані межі. У цьому випадку метод, відомий як планарний вибір орієнтирів, є хорошим варіантом. Згідно цього методу, потрібно знайти вузол v поблизу центру графа і поділити граф на секції k сегментів, для яких v буде центром, так щоб кожен сектор містив у

собі приблизно однакову кількість вершин. Потім для кожного сектора обирається найвіддаленіший вузол від c, і він стає орієнтиром.

Щоб уникнути ситуацій, коли два орієнтири розташовані занадто близько один до одного, якщо орієнтир у секторі A обраний близько межі наступного сектора B, необхідно пропустити вузли B, які знаходяться поблизу межі A.

Багаторівневий підхід

При плануванні маршруту від одного міста до іншого, часто виявляється, що більша частина найкоротшого маршруту не залежить від конкретних початкової та кінцевої точок. Деякі частини шляху можуть входити до множини найкоротших шляхів. У багаторівневому підході використовується цей факт для оптимізації процесу.

Сандерс і Шульц [11] представляють варіант цього методу, який вони називають ієрархіями магістралей. Їхній метод побудови маршрутів створює кілька рівнів для початкового графа під час попередньої обробки, де кожен вищий рівень є абстракцією нижнього. Оригінальний граф є найнижчим рівнем. При створенні нового рівня деякі вузли і ребра можуть бути видалені, а деякі ребра можуть бути додані.

Багаторівневий підхід передбачає знаходження так званих магістралей на графі та підняття їх на вищий рівень. Спочатку визначається околиця навколо кожного вузла. Магістралі складаються з ребер, які знаходяться на найкоротшому шляху від деякого вузла s до деякого кінцевого вузла t і мають початок та кінець за межами обох околиць s і t, які або покидають околицю s або входять в околицю t, проте не в обидві. Після цього виконуються кроки очищення нового рівня, включаючи видалення шляхів, що складаються лише з вузлів з двома сусідами, та видалення ізольованих вузлів.

Коли необхідна кількість рівнів створена, всі вузли на кожному рівні пов'язані з відповідним рівнем нижче, утворюючи результуючий граф. Побудова маршруту виконується з використанням модифікованої версії

алгоритму A*. Дослідження показують, що багаторівневий підхід на реальних дорожніх мережах працює у 500 – 2000 разів швидше, ніж алгоритм Дейкстри.

Субоптимальні алгоритми

Деякі автори, такі як Ботеа, Мюллер і Шеффер, введенням субоптимальних алгоритмів значно знижують час обчислення. Втім, за даними Сандерса і Шульца [12], алгоритми найкоротшого шляху в останні роки досить значно поліпшили свою продуктивність, що призвело до того, що накладні витрати, такі як відображення маршрутів та їх передача по мережі, тепер стали основним обмеженням швидкості цих алгоритмів.

Висновки до розділу 2

У розділі 2 розглянуто основні алгоритми навчання з підкріпленням, які можуть бути застосовані для пошуку шляху у віртуальному світі. Наведені основні алгоритми пошуку найкоротшого шляху.

Виявлено, що використання алгоритмів самонавчання для пошуку шляху не гарантує виявлення найкоротшого шляху, а використання алгоритмів пошуку найкоротшого шляху передбачає, що всі можливі переходи відомі до початку пошуку.

РОЗДІЛ 3 РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМИ ДЛЯ ПОШУКУ ШЛЯХУ У ВІРТУАЛЬНОМУ СВІТІ

3.1 Принцип використання машинного навчання для пошуку шляху

Нейромережі фіксують прийнятні і неприйнятні стани під час навчання завдяки процесу, що називається зворотнє поширення помилки (backpropagation). Основна ідея полягає в тому, щоб нейромережа коригувала свої ваги, оптимізуючи їх так, щоб вихід був ближчий до бажаного.

Ось кроки, які відбуваються під час цього процесу:

1. **Проходження вперед (forward pass):** Вхідні дані подаються в мережу, і вони проходять через кожен шар нейронів до вихідного шару. Кожен нейрон обчислює аговану суму вхідних сигналів і передає його через активаційну функцію.

2. **Обчислення помилки:** Результати порівнюються з бажаними вихідними значеннями, і обчислюється помилка. Це може бути різниця між передбаченим значенням і фактичним (помилка передбачення).

3. **Зворотнє поширення помилки:** Помилка потім "повертається" назад через мережу. Кожен шар отримує інформацію про те, як його внутрішні параметри впливають на помилку. Це відбувається за допомогою часткового диференціювання функції втрати відносно ваг нейронів у кожному шарі.

4. **Оновлення ваг:** Ваги в кожному шарі оновлюються, щоб зменшити помилку. Цей процес використовує градієнти, що були розраховані під час зворотного поширення помилки, і використовує їх для корекції ваг в кожному нейроні.

5. **Повторення:** Ці кроки повторюються протягом багатьох епох навчання (повторень навчання на всьому наборі даних), поки мережа не досягне прийнятного рівня точності або поки навчання не буде зупинено.

Таким чином, нейромережі "вчаться" шляхом адаптації їхніх внутрішніх параметрів (ваг) залежно від того, наскільки вони сприймають помилку між передбаченим і фактичним результатами.

Щоб проілюструвати фіксацію результатів при пошуку шляху в Python, давайте розглянемо простий приклад з використанням бібліотеки **networkx** для роботи з графами і пошуку шляху:

```
import networkx as nx
import matplotlib.pyplot as plt

# Створення напрямленого графа
G = nx.DiGraph()

# Додавання ребер і ваг
G.add_edge("A", "B", weight=4)
G.add_edge("A", "C", weight=2)
G.add_edge("B", "D", weight=5)
G.add_edge("C", "D", weight=1)
G.add_edge("D", "E", weight=7)

# Візуалізація графа
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_size=700, node_color="skyblue", font_size=10, font_color="black", font_weight="bold", arrowsize=20)
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

# Знаходження шляху
path = nx.shortest_path(G, source="A", target="E", weight="weight")
print("Шлях:", path)

# Відображення шляху червоним
highlighted_edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]
```

edge_colors = ["red" if edge in highlighted_edges else "gray" for edge in G.edges()]

nx.draw(G, pos, edge_color=edge_colors, with_labels=True, node_size=700, font_size=10, font_color="black", font_weight="bold", arrowsize=20)

Показ графа

plt.show()

У цьому прикладі використовується алгоритм пошуку найкоротшого шляху для знаходження шляху між вузлами "A" та "E" у напрямленому графі. Ребра шляху виділяються червоним кольором. За допомогою цього шаблону ви можете досліджувати і фіксувати результати пошуку шляху власних графів.

Алгоритм навчання з підкріпленням (reinforcement learning) для пошуку шляху в такій матриці може бути реалізований за допомогою методу Q-навчання. Основна ідея полягає в тому, щоб навчити агента приймати дії, які приведуть його до мети, отримуючи позитивні винагороди (реворди), і уникати ділянок, які непридатні для переходу, отримуючи негативні винагороди.

Ось загальний алгоритм:

1. **Визначте Стан**: Представте кожен можливий стан агента. Ваша матриця 5x5 може бути використана як область, де кожна клітина представляє стан. Наприклад, ліва верхня клітина буде (0, 0), а права нижня - (4, 4).

2. **Визначте Дії**: Визначте можливі дії для агента. У вашому випадку це може бути рух вгору, вниз, вліво, вправо.

3. **Ініціалізація Q-функції**: Створіть Q-функцію для оцінки якості (винагороди) для кожної пари стан-дія.

4. **Навчання агента**:

a. **Обрати дію**: Виберіть дію на підставі вже вивчених знань або за допомогою стратегії випадкового вибору для дослідження нових дій.

b. **Виконати дію**: Перейдіть до нового стану відповідно до вибраної дії.

c. **Отримати винагороду**: Перевірте, чи новий стан є метою або призводить до позитивної чи негативної винагороди.

d. **Оновити Q-функцію**: Використовуючи формулу Q-навчання, оновіть значення Q для поточної пари стан-дія на підставі отриманої винагороди та майбутніх очікуваних винагород.

e. **Повторити**: Повторюйте процес для кількох епох.

3.2 Використання алгоритму Q-навчання для матриці

Для реалізації алгоритму Q-навчання слід врахувати велику кількість деталей, і вибір конкретного методу може залежати від вимог. Нижче наведено приклад базової реалізації для простої матриці 5x5:

```
import numpy as np

# Розмірність матриці
ROWS = 5
COLS = 5

# Початковий стан
start_state = (0, 0)

# Мета
goal_state = (4, 4)

# Навчання
def q_learning(matrix, start, goal, epochs, alpha, gamma):
    q_values = np.zeros((ROWS, COLS, 4))  # 4 - кількість можливих дій (вгору, вниз, вліво, вправо)

    for epoch in range(epochs):
```

```
        current_state = start
        while current_state != goal:
            # Вибір дії
            action = np.argmax(q_values[current_state])

            # Виконання дії та отримання винагороди
            new_state = take_action(current_state, action)
            reward = get_reward(new_state, goal)

            # Оновлення Q-функції
            q_values[current_state][action] += alpha * (reward + gamma * np.max(q_values[new_state]) - q_values[current_state][action])

            current_state = new_state

    return q_values

# Визначення можливих дій
def take_action(state, action):
    if action == 0:  # вгору
        return max(state[0] - 1, 0), state[1]
    elif action == 1:  # вниз
        return min(state[0] + 1, ROWS - 1), state[1]
    elif action == 2:  # вліво
        return state[0], max(state[1] - 1, 0)
    elif action == 3:  # вправо
        return state[0], min(state[1] + 1, COLS - 1)

# Отримання винагороди
def get_reward(state, goal):
```

```
    return 1 if state == goal else 0

# Тестування
q_values = q_learning(np.zeros((ROWS, COLS)), start_state, goal_state, epochs=1000, alpha=0.1, gamma=0.9)

# Виведення отриманих Q-значень
print("Q-Values:")
print(q_values)
```

Наступна програма дозволяє користувачеві ввести розміри матриці, а потім навчає алгоритм Q-навчання. Результати навчання зберігаються у файлі **q_values.npy**. Кожен раз, коли запускається програма, вона буде завантажувати попередні результати навчання, якщо такі існують.

```
import numpy as np
import os

def save_q_values(q_values, file_path):
    np.save(file_path, q_values)

def load_q_values(file_path):
    if os.path.exists(file_path):
        return np.load(file_path)
    else:
        return np.zeros((ROWS, COLS, 4))

def train_q_learning(q_values, matrix, learning_rate=0.1, discount_factor=0.9, epochs=1000):
    for epoch in range(epochs):
```

```
            current_state = (0, 0)  # Assuming the start state is (0, 0)

            while current_state != (ROWS-1, COLS-1):  # Continue until reaching the goal state
                action = np.argmax(q_values[current_state])
                next_state = get_next_state(current_state, action)
                reward = matrix[next_state]

                # Q-value update
                q_values[current_state][action] += learning_rate * (reward + discount_factor * np.max(q_values[next_state]) - q_values[current_state][action])

                current_state = next_state

    def get_next_state(state, action):
        row, col = state

        if action == 0:  # Up
            row = max(row - 1, 0)
        elif action == 1:  # Down
            row = min(row + 1, ROWS - 1)
        elif action == 2:  # Left
            col = max(col - 1, 0)
        elif action == 3:  # Right
            col = min(col + 1, COLS - 1)

        return (row, col)

    # Define the size of the matrix
    ROWS = int(input("Enter the number of rows: "))
```

```
COLS = int(input("Enter the number of columns: "))

# Initialize Q-values
q_values_file = "q_values.npy"
q_values = load_q_values(q_values_file)

# Initialize the matrix
matrix = np.random.choice([0, 1], size=(ROWS, COLS))

# Train Q-learning
train_q_learning(q_values, matrix)

# Save Q-values
save_q_values(q_values, q_values_file)
```

3.3 Реалізація програми для пошуку найкоротшого шляху у віртуальному світі

Розвитком попередніх програм є наступна програма написана на мові Python, що використовує алгоритм Q-навчання для навчання агента знаходження найкоротшого шляху в матриці 5x5.

Спочатку необхідно підключити відповідні бібліотеки, що додають в Python підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами, можливість генерувати випадкові послідовності, та можливість обробляти структури json.

```
import numpy as np
import random
import json
```

Після цього необхідно ініціалізувати клас `QLearningAgent` для створення агента Q-навчання

```
class QLearningAgent:
    def __init__(self, matrix, states, actions, alpha=0.1, gamma=0.9, epsilon=0.1):
        self.q_values = {state: {action: 0.0 for action in actions} for state in states}
        self.alpha = float(alpha)
        self.gamma = gamma
        self.epsilon = epsilon
        self.actions = actions
        self.matrix = matrix
```

Далі необхідно реалізувати методи для роботи агента.

Функція `choose_action` в класі `QLearningAgent` відповідає за вибір дії агента в певному стані. Ця функція реалізує стратегію вибору дій для агента, яка може бути заснована на деяких правилах чи ймовірностях.

Основна ідея за таким вибором дії полягає в тому, що агент може випадковим чином обирати нову дію з ймовірністю, відомої як "ймовірність дослідження". Це дозволяє агентові досліджувати різні дії та відкривати нові можливості.

Якщо випадкове число менше, то агент вибирає випадкову дію зі свого списку доступних дій. Інакше агент обирає дію, яка має найвище значення в Q-функції для поточного стану. Такий підхід називається "ймовірністю винятку".

Отже, `choose_action` забезпечує баланс між дослідженням нових можливостей і використанням вже вивчених дій з використанням Q-функції.

```
def choose_action(self, state):
    if random.uniform(0, 1) < self.epsilon:
        return random.choice(self.actions)
    else:
        if state in self.q_values:
            available_actions = [action for action in self.actions if
self.q_values[state].get(action) != 0.0]
            if available_actions:
                return random.choice(available_actions)
        return random.choice(self.actions)
```

Функція `update_q_value` в класі `QLearningAgent` використовується для оновлення значень Q-функції (Q-values) в процесі навчання агента. Q-функція визначає цінність кожної можливої дії в кожному стані.

Отже, основна мета `update_q_value` - це адаптація Q-функції на основі отриманих даних під час взаємодії агента з оточенням. Оновлення здійснюється з використанням формули оновлення для методу Q-навчання, який може бути виражений як:

Отже, функція `update_q_value` оновлює Q-функцію, враховуючи отриману нагороду та очікуване майбутнє значення Q-функції.

```
    def update_q_value(self, state, action, reward, next_state,
best_next_action):
        if state not in self.q_values:
            self.q_values[state] = {action: 0.0 for action in self.actions}
        if next_state in self.q_values:
            current_q = self.q_values[state][action]
            learned_value = reward + self.gamma *
self.q_values[next_state][best_next_action]
            new_q = current_q + self.alpha * (learned_value - current_q)
            self.q_values[state][action] = new_q
```

Функція `get_reward` в класі `QLearningAgent` визначає винагороду, яку агент отримує за перехід в новий стан. У контексті задачі навчання з підкріпленням, де агент вивчається через взаємодію з середовищем, винагорода вказує на те, наскільки добре агент виконав певну дію в певному стані.

У конкретному випадку, функція `get_reward` повертає винагороду за перехід з поточного стану в новий стан. Якщо новий стан є цільовим станом (goal state), то винагорода може бути позначена значенням 1.0, щоб

підкреслити успішний результат. У інших випадках, коли цільовий стан не досягнутий, винагорода може бути 0.0 або іншим визначеним значенням.

Ця функція визначає те, як середовище (представлене станом) визнає результат агента і яким чином це оцінюється винагородою.

```
def get_reward(self, state, goal_state):
    if state == goal_state:
        return 1.0
    else:
        return 0.0
```

Функція `train` в класі `QLearningAgent` використовується для тренування агента за допомогою алгоритму Q-навчання. Основна ідея полягає в тому, щоб агент вивчав оптимальні стратегії дій у різних станах середовища, максимізуючи очікувані сумарні винагороди.

Основні кроки функції `train`:

1. **Ініціалізація параметрів:** Початковий стан встановлюється на `start_state`. Обираються параметри навчання, такі як кількість епох (`num_episodes`).

2. **Цикл навчання**: Протягом кожної епохи агент взаємодіє з середовищем, обираючи дії, отримуючи винагороди та оновлюючи значення Q-функції.

- *Внутрішній цикл взаємодії з середовищем*: Поки не досягнуто цільового стану, агент обирає дію, отримує винагороду, визначає наступний стан та оновлює значення Q-функції за допомогою алгоритму Q-навчання.

- *Друк інформації*: У кінці кожної епохи може бути виведена інформація про навчання, така як загальна винагорода для епохи.

Ця функція є ключовою для навчання агента, визначає його взаємодію з середовищем та процес вивчення оптимальних стратегій в різних станах.

```
def train(self, start_state, goal_state, num_episodes=100):
    for episode in range(num_episodes):
        current_state = start_state
        total_reward = 0
        while current_state != goal_state:
            action = self.choose_action(current_state)
            next_state = self.get_next_state(current_state, action)
            reward = self.get_reward(next_state, goal_state)
            total_reward += reward
            if next_state not in self.q_values:
                self.q_values[next_state] = {action: 0.0 for action in self.actions}
            best_next_action = max(self.q_values[next_state], key=self.q_values[next_state].get)
            self.update_q_value(current_state, action, reward, next_state, best_next_action)
            current_state = next_state
        print(f"Episode {episode + 1}, Total Reward: {total_reward}")
```

Метод `find_shortest_path` в класі `QLearningAgent` використовує навчені значення Q-функції, щоб знайти найкоротший шлях від визначеного початкового стану до цільового стану. Основна ідея полягає в тому, щоб агент обирати дії, спираючись на значення Q-функції, щоб максимізувати очікувані винагороди на кожному кроці.

Основні кроки методу:

1. **Ініціалізація змінних**: Починається з початкового стану, додає його до шляху.

2. **Цикл пошуку шляху**: Поки поточний стан не дорівнює цільовому стану, агент обирає дію, спираючись на значення Q-функції. Вибрана дія додає наступний стан до шляху.

3. **Завершення та повернення результату**: Повертається знайдений шлях.

Цей метод допомагає агентові використовувати навчені стратегії для досягнення певних цілей в середовищі, і його результат може використовуватися для візуалізації або подальших аналізів.

```
def find_shortest_path(self, start_state, end_state, matrix):
    current_state = start_state
    path = [current_state]
    while current_state != end_state:
        str_state = str(current_state)  # Перетворення кортежа на рядок
        available_actions = [action for action in self.actions if 
self.is_action_valid(current_state, action, matrix)]
        if available_actions:
            # Вибрати дію, яка має максимальне значення в q_values
            action = max(available_actions, key=lambda a: 
self.q_values[str_state].get(a, 0.0))
            next_state = self.get_next_state(current_state, action)
            path.append(next_state)
            current_state = next_state
        else:
            break  # Зупинити цикл, якщо всі дії недоступні
    return path

def is_action_valid(self, current_state, action, matrix):
    next_state = self.get_next_state(current_state, action)
    return next_state != current_state and 
matrix[next_state[0]][next_state[1]] != 0
```

Метод `get_next_state` в класі `QLearningAgent` визначає, який буде наступний стан в середовищі, якщо агент обере певну дію. Цей метод використовується для оновлення стану під час тренування агента.

Основні кроки методу:

1. **Отримання поточного стану**: Розпаковує координати поточного стану.

2. **Перевірка обраної дії**: Дивиться на обрану дію і визначає, як буде змінено стан відповідно до цієї дії.

3. **Повернення наступного стану**: Повертає нові координати як наступний стан після обраної дії.

Цей метод є ключовим для оновлення станів під час навчання агента, дозволяючи йому взаємодіяти з середовищем та вдосконалювати свої стратегії на основі отриманих винагород і навчених значень Q-функції.

```
def get_next_state(self, current_state, action):
    i, j = current_state
    if action == "UP" and i > 0 and self.matrix[i - 1, j] == 1:
        return i - 1, j
    elif action == "DOWN" and i < self.matrix.shape[0] - 1 and
self.matrix[i + 1, j] == 1:
        return i + 1, j
    elif action == "LEFT" and j > 0 and self.matrix[i, j - 1] == 1:
        return i, j - 1
    elif action == "RIGHT" and j < self.matrix.shape[1] - 1 and
self.matrix[i, j + 1] == 1:
        return i, j + 1
    else:
        return i, j
```

Функція ` display_matrix ` призначена для відображення матриці

```
def display_matrix(matrix):
    for row in matrix:
        print(row)
```

Функція `save_results` в класі `QLearningAgent` призначена для збереження результатів навчання агента у файлі JSON.

Ця функція є важливою для збереження вивчених значень Q-функції агента для подальшого використання або аналізу.

```
def save_results(q_agent, path):
    # Перетворення ключів словників на рядки
    q_values_str_keys = {str(key): value for key, value in
q_agent.q_values.items()}
    with open(path, "w") as file:
        json.dump(q_values_str_keys, file)
```

Функція ` load_results ` призначена для завантаження результатів з файлу JSON.

```
def load_results(path):
    with open(path, "r") as file:
        return json.load(file)
```

Програма використовує введення матриці з клавіатури за допомогою консолі. Основні особливості цього введення:

1. **Цикл введення**: Існує цикл, який очікує введення користувача рядка для кожного рядка матриці. Цикл продовжується до того моменту, коли користувач натискає клавішу Enter, вказуючи завершення введення.

2. **Обробка рядків**: Кожний введений рядок розбивається на числа за допомогою методу `split()`. Ці числа інтерпретуються як елементи рядка матриці, де 1 відповідає доступній клітинці, а 0 - недоступній.

3. **Створення матриці**: Введені дані перетворюються в двовимірний масив за допомогою бібліотеки NumPy.

4. **Створення списку станів**: Стани для Q-навчання обчислюються як позиції у матриці, де значення 1 вказує на доступні клітинки, які агент може відвідати.

Це введення забезпечує можливість створення середовища для навчання Q-агента на основі введених даних про доступність клітинок у матриці.

```
matrix_rows = []
while True:
    row_input = input("Enter matrix row (1 for accessible, 0 for inaccessible, press Enter to finish): ")
    if not row_input:
        break
    matrix_rows.append(list(map(int, row_input.split())))
matrix = np.array(matrix_rows)
states = [(i, j) for i in range(matrix.shape[0]) for j in range(matrix.shape[1]) if matrix[i][j] == 1]
```

Створення списку можливих дій агента.

```
actions = ["UP", "DOWN", "LEFT", "RIGHT"]
```

Створення та тренування агента полягає у створенні об'єкту класу `QLearningAgent` та виклику для нього методу `train`, параметрами якого є

координати початкового і кінцевого місця в матриці, між якими потрібно знайти шлях, а також кількість епізодів для повторення.

```
agent = QLearningAgent(matrix, states, actions)
agent.train((0, 0), (4, 4), num_episodes=1000)
```

Збереження результатів у файл

```
save_results(agent, "q_values.json")
```

Завантаження результатів з файлу

```
loaded_agent = QLearningAgent(matrix, states, actions)
loaded_agent.q_values = load_results("q_values.json")
```

Знаходження та виведення найкоротшого шляху

```
start = (0, 0)
end = (4, 4)
shortest_path = loaded_agent.find_shortest_path(start, end, matrix)
print(f"Shortest Path from {start} to {end}: {shortest_path}")
```

Відображення матриці

```
display_matrix(matrix)
```

Цей код використовує алгоритм Q-навчання для тренування агента, який знаходить найкоротший шлях в матриці. Результати тренування зберігаються у файл, і потім можна завантажити ці результати, щоб знайти найкоротший шлях.

3.4 Тестування програми пошуку шляху

Програмування і тестування здійснювалося в редакторі Visual Studio Code

Version: 1.75.1 (user setup)

Date: 2023-02-08T21:32:34.589Z

Electron: 19.1.9

Chromium: 102.0.5005.194

Node.js: 16.14.2

V8: 10.2.154.23-electron.0

OS: Windows_NT x64 10.0.18363

Sandboxed: No

Встановлено версію Python 3.11.3 і відповідне розширення для Visual Studio Code

Апаратна складова: процесор Intel Core i7-4510u + 8GB оперативної пам'яті.

Під час проведення тестування протестовано введення даних і виконання програми.

Введення здійснювалося з клавіатури, при введенні помилок не виникло.

```
PS G:\Мій диск\2023\ml\prog>  & 'C:\Program Files\Python311\python.exe' 'c:\Users\stafme\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '54987' '--' 'g:\М?й диск\2023\ml\prog\graph2.py'
Enter matrix row (1 for accessible, 0 for inaccessible, press Enter to finish): 1 1 1 0 0
Enter matrix row (1 for accessible, 0 for inaccessible, press Enter to finish): 1 0 1 1 0
Enter matrix row (1 for accessible, 0 for inaccessible, press Enter to finish): 1 1 0 1 1
Enter matrix row (1 for accessible, 0 for inaccessible, press Enter to finish): 1 1 1 0 1
Enter matrix row (1 for accessible, 0 for inaccessible, press Enter to finish): 1 0 1 1 1
Enter matrix row (1 for accessible, 0 for inaccessible, press Enter to finish):
```

Рис. 3.1 – Введення матриці з клавіатури

Процес виконання обчислень триває близько 2 секунд. Результати навчання зберігаються у файл ‘q_values.json’, де можна проаналізувати отримані результати навчання (рис. 3.2).


```
{
  "(0, 0)": {
    "UP": 0.43046720999999744,
    "DOWN": 0.47829689999999736,
    "LEFT": 0.43046720999999744,
    "RIGHT": 0.47829689999999736
  },
  "(0, 1)": {
    "UP": 0.47829689999999736,
    "DOWN": 0.47829689999999736,
    "LEFT": 0.43046720999999744,
    "RIGHT": 0.5314409999999974
  },
  "(0, 2)": {
    "UP": 0.5314409999999974,
    "DOWN": 0.5904899999999977,
    "LEFT": 0.47829689999999736,
    "RIGHT": 0.5314409999999974
  },
  "(1, 0)": {
    "UP": 0.43046720999999744,
    "DOWN": 0.5314409999999974,
    "LEFT": 0.47829689999999736,
    "RIGHT": 0.47829689999999736
  },
  "(1, 2)": {
    "UP": 0.5314409999999974,
    "DOWN": 0.5904899999999977,
    "LEFT": 0.5904899999999977,
    "RIGHT": 0.6560999999999979
  },
  "(1, 3)": {
    "UP": 0.6560999999999979,
    "DOWN": 0.7289999999999983,
    "LEFT": 0.5904899999999977,
    "RIGHT": 0.6560999999999979
  },
  "(2, 0)": {
    "UP": 0.47829689999999736,
    "DOWN": 0.5904899999999977,
    "LEFT": 0.5314409999999974,
    "RIGHT": 0.5904899999999977
  },
  "(2, 1)": {
    "UP": 0.5904899999999977,
    "DOWN": 0.6560999999999979,
    "LEFT": 0.5314409999999974,
    "RIGHT": 0.5904899999999977
  },
  "(2, 3)": {
    "UP": 0.6560999999999979,
    "DOWN": 0.7289999999999983,
    "LEFT": 0.7289999999999983,
    "RIGHT": 0.8099999999999987
  },
  "(2, 4)": {
    "UP": 0.8099999999999987,
    "DOWN": 0.899999999999999,
    "LEFT": 0.7289999999999983,
    "RIGHT": 0.8099999999999987
  },
  "(3, 0)": {
    "UP": 0.5314409999999974,
    "DOWN": 0.5314409999999974,
    "LEFT": 0.5904899999999977,
    "RIGHT": 0.6560999999999979
  },
  "(3, 1)": {
    "UP": 0.5904899999999977,
    "DOWN": 0.6560999999999979,
    "LEFT": 0.5904899999999977,
    "RIGHT": 0.7289999999999983
  },
  "(3, 2)": {
    "UP": 0.7289999999999983,
    "DOWN": 0.8099999999999987,
    "LEFT": 0.6560999999999979,
    "RIGHT": 0.7289999999999983
  },
  "(3, 4)": {
    "UP": 0.8099999999999987,
    "DOWN": 0.9999999999999996,
    "LEFT": 0.899999999999999,
    "RIGHT": 0.899999999999999
  },
  "(4, 0)": {
    "UP": 0.5904899999999977,
    "DOWN": 0.5314409999999974,
    "LEFT": 0.5314409999999974,
    "RIGHT": 0.5314409999999974
  },
  "(4, 2)": {
    "UP": 0.7289999999999983,
    "DOWN": 0.8099999999999987,
    "LEFT": 0.8099999999999987,
    "RIGHT": 0.899999999999999
  },
  "(4, 3)": {
    "UP": 0.899999999999999,
    "DOWN": 0.899999999999999,
    "LEFT": 0.8099999999999987,
    "RIGHT": 0.9999999999999996
  },
  "(4, 4)": {
    "UP": 0.0,
    "DOWN": 0.0,
    "LEFT": 0.0,
    "RIGHT": 0.0
  }
}
```


Рис. 3.2 – Результати навчання з файлу ‘q_values.json’

В процесі виконання розрахунків програма виводить номер кожного епізоду з отриманою винагородою. Після завершення розрахунків програма виводить список координат матриці, послідовний перехід, між якими, є найкоротшим шляхом між початковою і кінцевою точками (рис. 3.3), а також матрицю, для якої виконане навчання.

```
Shortest Path from (0, 0) to (4, 4): [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (4, 2), (4, 3), (4, 4)]
[1 1 1 0 0]
[1 0 1 1 0]
[1 1 0 1 1]
[1 1 1 0 1]
[1 0 1 1 1]
PS G:\Мій диск\2023\ml\prog>
```

Рис. 3.3 – Результати роботи програми

Програма працює без помилок, введення виведення результатів відбуваються коректно, результати навчання зберігаються у файл.

Висновки до розділу 3

У розділі 3 реалізовано алгорим Q-навчання для навчання агента на знаходження найкоротшого шляху в матриці 5х5. Для цього створено програму пошуку найкоротшого шляху у графі. Реалізовано приклад базової реалізації пошуку шляху для простої матриці 5х5. Після чого написана програма на мові Python, що використовує алгоритм Q-навчання для навчання агента для знаходження найкоротшого шляху в матриці 5х5.

ВИСНОВКИ

У розділі 1 розглянуто історичні аспекти розвитку машинного навчання. Проаналізовано основні сучасні методи забезпечення машинного навчання. Наведено приклади використання автоматизованих систем здатних до самонавчання.

Виявлено, що системи самонавчання мають історію розвитку, яка почалася разом зі створенням обчислювальних систем. За цей час створено різні підходи до самонавчання машин, в розділі визначені їх основні особливості, переваги і недоліки. Аналіз використання систем самонавчання показав постійний розвиток як алгоритмів самонавчання так і галузей, в яких вони використовуються.

У розділі 2 розглянуто основні алгоритми навчання з підкріпленням, які можуть бути застосовані для пошуку шляху у віртуальному світі. Наведені основні алгоритми пошуку найкоротшого шляху.

Виявлено, що використання алгоритмів самонавчання для пошуку шляху не гарантує виявлення найкоротшого шляху, а використання алгоритмів пошуку найкоротшого шляху передбачає, що всі можливі переходи відомі до початку пошуку.

У розділі 3 реалізовано алгорим Q-навчання для навчання агента на знаходження найкоротшого шляху в матриці 5x5. Для цього створено програму пошуку найкоротшого шляху у графі. Реалізовано приклад базової реалізації пошуку шляху для простої матриці 5x5. Після чого написана програма на мові Python, що використовує алгоритм Q-навчання для навчання агента для знаходження найкоротшого шляху в матриці 5x5.

Таким чином мету дослідження за темою «Дослідження методів статистичного навчання з підкріпленням для пошуку шляху у віртуальному світі», яка полягає в аналізі сучасного стану розвитку алгоритмів машинного навчання, виявленні переваг і недоліків різних алгоритмів, а також використанні найкращих алгоритмів навчання з підкріпленням для пошуку шляху у віртуальному світі, досягнуто.

Список використаних джерел

1. Jonker, Roy; Volgenant, Ton (). "Transforming asymmetric into symmetric traveling salesman problems". Operations Research Letters. 2 (161–163): 1983.
2. Goemans, M.; Bertsimas, D. (1991). "Probabilistic analysis of the Held and Karp lower bound for the Euclidean traveling salesman problem". Mathematics of Operations Research. 16 (1): 72–89.
3. Marco Dorigo. "Ant Colonies for the Traveling Salesman Problem. IRIDIA, Université Libre de Bruxelles. IEEE Transactions on Evolutionary Computation, 1(1):53–66. 1997.
4. Lawler, E. L. (2013). The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization (Repr. with corrections. ed.). John Wiley & sons. ISBN 978-0471904137.
5. Костевич Л. С. Математическое программирование: Информ. технологии оптимальных решений: Учеб. пособие / Л. С. Костевич. — Мн.: Новое знание, 2003. ил., стр. 150, ISBN 985-6516-83-8
6. Haider A Abdulkarim, Ibrahim F Alshammari (). "Comparison of Algorithms for Solving Traveling Salesman Problem". International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-4 Issue-6, August 2015
7. Wang Hui (). "Comparison of several intelligent algorithms for solving TSP problem in industrial engineering". Systems Engineering Procedia 4 (2012) 226 – 235
8. Marek Antosiewicz, Grzegorz Koloch, Bogumił Kamiński. "Choice of best possible metaheuristic algorithm for the travelling salesman problem with limited computational time: quality, uncertainty and speed". Journal of Theoretical and Applied Computer Science Vol. 7, No. 1, 2013, pp. 46-55 ISSN 2299-2634
9. Dorigo, Marco. Ant colony optimization / Marco Dorigo, Thomas

Stu¨tzle. p. cm. ‘‘A Bradford book.’’ Includes bibliographical references (p.). ISBN 0-262-04219-3 (alk. paper) 1. Mathematical optimization. 2. Ants–Behavior–Mathematical models. I. Stu¨tzle, Thomas. II. Title. QA402.5.D64 2004 519.6—dc22 2003066629

10. Angel E., Zissimopoulos V. On the classification of NP-complete problems in terms of their correlation coefficient // Discrete Appl. Math. 2. V. 9. P. 261–277.
11. Korte B, Vugen J. Combinatorial optimization. Berlin: Springer, 2007.
12. Yannakakis M. Computational complexity. In: Aarts E., Lenstra J. (Eds.) Local search in combinatorial optimization. NY:Wiley, 1997.
13. Immerman, Neil (1982). "Relational Queries Computable in Polynomial Time". STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing. pp. 147–152. doi:10.1145/800070.802187. Revised version in Information and Control, 68 (1986), 86–104.
14. Pocklington, H. C. (1910–1912). "The determination of the exponent to which a number belongs, the practical solution of certain congruences, and the law of quadratic reciprocity". Proc. Camb. Phil. Soc. 16: 1–5.
15. Gautschi, Walter (1994). Mathematics of computation, 1943–1993: a half-century of computational mathematics: Mathematics of Computation 50th Anniversary Symposium, August 9–13, 1993, Vancouver, British Columbia. Providence, RI: American Mathematical Society. pp. 503–504. ISBN 978-0-8218-0291-5.
16. Hopcroft, John E.; Rajeev Motwani; Jeffrey D. Ullman (2001). Introduction to automata theory, languages, and computation (2. ed.). Boston: Addison-Wesley. pp. 425–426. ISBN 978-0201441246.
17. Immerman, Neil (1999). Descriptive Complexity. New York: Springer-Verlag. p. 66. ISBN 978-0-387-98600-5.
18. Vardi, Moshe Y. (1982). "The Complexity of Relational Query Languages". STOC '82: Proceedings of the fourteenth annual ACM

symposium on Theory of computing. pp. 137–146. doi:10.1145/800070.802186.

19. Kleinberg, Jon; Tardos, Éva (2006). Algorithm Design (2nd ed.). Addison-Wesley. p. 464. ISBN 0-321-37291-3.

20. Ladner, R. E. (1975). "On the structure of polynomial time reducibility". J. ACM. 22: 151–171. doi:10.1145/321864.321877. Corollary 1.1.

21. Kleinberg, Jon; Tardos, Éva (2006). Algorithm Design (2nd ed.). p. 496. ISBN 0-321-37291-3.

22. Impagliazzo, R.; Paturi, R. (2001). "On the complexity of k-SAT". Journal of Computer and System Sciences. Elsevier. 62 (2): 367–375. doi:10.1006/jcss.2000.1727. ISSN 1090-2724.

23. Miltersen, P.B. (2001). "DERANDOMIZING COMPLEXITY CLASSES". Handbook of Randomized Computing. Kluwer Academic Pub: 843.

24. Kumar, Ravi; Rubinfeld, Ronitt (2003). "Sublinear time algorithms". SIGACT News. 34 (4): 57–67. doi:10.1145/954092.954103.

25. Terence (2010). "1.11 The AKS primality test". An epsilon of room, II: Pages from year three of a mathematical blog. Graduate Studies in Mathematics. 117. Providence, RI: American Mathematical Society. pp. 82–86. doi:10.1090/gsm/117. ISBN 978-0-8218-5280-4. MR 2780010.

26. Sipser, Michael (2006). Introduction to the Theory of Computation. Course Technology Inc. ISBN 0-619-21764-2.

27. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, AA, Veness, J., Bellemare, MG, ... & Petersen, S. (2015). Контроль на рівні людини через глибоке навчання з підкріпленням. Природа, 518 (7540), 529-533.

28. Саттон, Р. С., Барто, А. Р. (2018). Навчання з підкріпленням: Вступ (2-е вид.). MIT Press.

29. Лілікрап, Т.П., Хант, Дж.Дж., Прітцель, А., Гесс, Н., Ерез, Т., Тасса, Ю., ... і Вірстра, Д. (2019). Постійний контроль із глибоким навчанням з підкріпленням. препринт arXiv arXiv:1509.02971.
30. Шульман, Дж., Вольскі, Ф., Дхарівал, П., Редфорд, А., і Клімов, О. (2017). Алгоритми оптимізації проксимальної політики. Препринт arXiv arXiv:1707.06347.
31. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). "Introduction to Algorithms." MIT Press.
32. LeCun, Y., Bengio, Y., & Hinton, G. (2015). "Deep learning." Nature, 521(7553), 436-444.
33. Bishop, C. M. (2006). "Pattern Recognition and Machine Learning." springer.
34. Ng, A. (2011). "Machine Learning: A Coursera Online Course." Stanford University.
35. TensorFlow Documentation
36. Scikit-Learn Documentation
37. Murphy, K. P. (2012). "Machine Learning: A Probabilistic Perspective." MIT Press.
38. Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). "Deep Learning." MIT press Cambridge.
39. Aggarwal, C. C. (2018). "Neural Networks and Deep Learning: A Textbook." Springer.
40. Типи машинного навчання: Контрольоване проти ненавчаного навчання [Електронний ресурс] Режим доступу - https://uk.myservername.com/types-machine-learning

Додаток 2 до наказу
від «31» березня 2023 року
№119/05

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, ____
____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

____________________ (дата)

____________________ (підпис)