DOI: 10.25140/2411-5363-2023-2(32)-175-184
УДК 004.421

Артем Волокита[1], Богдан Герега[2]

[1]кандидат технічних наук, доцент кафедри обчислювальної техніки
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського» (Київ, Україна)
E-mail: artem.volokita@kpi.ua. **ORCID:** http://orcid.org/0000-0001-9069-5544

[2] студент 6-го курсу факультету ІОТ
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського» (Київ, Україна)
E-mail: bogdangerega19@gmail.com

ЕВОЛЮЦІЯ АГЕНТІВ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ЗА ДОПОМОГОЮ ГЕНЕТИЧНОГО АЛГОРИТМУ

Це дослідження вивчає використання генетичних алгоритмів для покращення продуктивності агентів, що навчаються за допомогою підкріплення. Ми провели випробування, використовуючи різні параметри нейронної мережі, зокрема ваги, зсуви та функції активації, з метою знайти оптимальні значення, які змушують агента отримувати більше винагород. Наш підхід включає використання знань про предметну область для ініціалізації популяції генетичного алгоритму, а також для оцінки рішень. Це дозволяє нам спрямувати пошук до більш перспективних рішень. Особлива увага приділена впливу різних параметрів генетичного алгоритму на ефективність навчання. Потенційні застосування цього дослідження широкі – від робототехніки та автономних транспортних засобів до ігор та фінансів. Результати дослідження також можна використовувати для розробки нових алгоритмів та методів для покращення продуктивності агентів, що навчаються за допомогою підкріплення, що далі сприятиме розвитку машинного навчання.

Наше дослідження показало, що використання генетичного алгоритму може значно покращити ефективність навчання агентів. Результатом роботи є успішне проходження гри CartPole-v0 еволюціонований агентами. 98 % нашої популяції досягнуть максимуму, тобто успішно пройдуть гру.

Ключові слова: *навчання з підкріпленням; генетичний алгоритм; агент; безградієнтний підхід; нейронна мережа; CartPole, policy gradients.*

Рис.: 5. Бібл.: 10.

Актуальність теми дослідження. Навчання з підкріпленням є потужним методом навчання машин приймати рішення на основі зворотного зв'язку, але досягти високої продуктивності може бути складно через складну та динамічну природу багатьох реальних середовищ. Генетичні алгоритми забезпечують перспективний підхід до вирішення цієї проблеми, дозволяючи агентам еволюціонувати з часом, адаптуючись до мінливих умов і покращуючи свою продуктивність шляхом спроб і помилок. Потенційні застосування цих досліджень дуже широкі – від робототехніки та автономних транспортних засобів до ігор і фінансів. Наприклад, натреновані агенти можуть бути використані для оптимізації логістики ланцюгів поставок, управління складними фінансовими портфелями або для контролю поведінки інтелектуальних роботів у динамічних і непередбачуваних середовищах. Крім того, висновки, отримані в результаті цього дослідження, можуть бути використані при розробці нових алгоритмів і методів для поліпшення продуктивності агентів RL, що сприятиме подальшому прогресу в галузі машинного навчання. Загалом, актуальність цієї теми дослідження підкреслює важливість розробки більш ефективних методів навчання, які здатні адаптуватися до складних середовищ.

Постановка проблеми. Навчання з підкріпленням прогресує повільно через низьку якість градієнтів. Відсутність інформації про градієнти призводить до нестабільного ландшафту, подібного до нерівної поверхні [1]. Припустимо, що мережа нашого агента подібна до дна океану, де між глибокими водами та поверхнею існує безліч різних підвищень та ущелин. У цьому випадку нерівності поверхні відображають параметри мережі нашого агента, які змінюються залежно від винагороди, яку отримує агент. Кілька ущелин на поверхні відображають високу винагороду, що відповідає добре працюючим параметрам, тоді як плоска поверхня відображає параметри, які не є ефективними. Ініціалізація агента на плоскій поверхні може призвести до того, що агент буде рухатися досить випадково та неефективно протягом тривалого часу. Таким чином, з низькоякісними градієнтами потрібно багато часу, щоб навчити агента.

Навчання з підкріпленням може бути ефективним, коли агент розуміє, які дії призводять до винагороди. Оскільки у навчанні з підкріпленням агент не повідомляється про наступні дії, це призводить до низько ефективності [2]. Агент, не знаючи, що робити, починає виконувати випадкові рухи, і лише зрідка середовище видає винагороду. І тепер вже агент повинен з'ясувати, яка з тисяч дій, які він зробив, призвела до того, що середовище дало винагороду. Люди так не навчаються! Нам кажуть, що потрібно робити, ми розвиваємо навички, а винагороди відіграють порівняно незначну роль у нашому навчанні.

Аналіз останніх досліджень і публікацій. Останніми роками було продемонстровано, що еволюційні алгоритми є конкурентоспроможною альтернативою методам оптимізації на основі градієнта для навчання з підкріпленням. Наприклад, у роботі «Evolution algorithms as a Scalable Alternative to Reinforcement Learning» [3] Саліманс зі співавторами досліджували використання еволюційних алгоритмів, класу алгоритмів оптимізації чорної скриньки, як альтернативи популярним методам на основі градієнтів, таким як глибоке Q навчання та policy gradients. Вони проводили кілька експериментів у середовищах MuJoCo та Atari, щоб показати, що еволюційні алгоритми є життєздатним рішенням.

У «Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning» [4], Петроскі Суч та інші запропонувати подібний підхід до вирішення проблем навчання з підкріпленням за допомогою еволюційних алгоритмів, де потрібно розвивати попередні результати, щоб досягти кращого рівня, досягнутого за допомогою методів на основі градієнта. Це розширило попереднє уявлення про масштаб, у якому могли працювати генетичні алгоритми, і, використовуючи розпаралелювання, продемонструвало значні покращення в часі навчання, дозволяючи розв'язувати Atari приблизно за 4 години на сучасному обчислювальному пристрою.

Були ще розглянуті роботи щодо включення градієнтної інформації в еволюційні алгоритми для вирішення проблем глибокого навчання з підкріпленням. Наприклад, найбільш цікавою є робота «Evolutionary Reinforcement Learning» [5], у якій Хадка та Тумер пропонують гібридний алгоритм, який використовує популяцію генетичного алгоритму для надання різноманітних даних для навчання агента RL, і періодично повторно вводить агента в популяцію для введення градієнтної інформації.

Наша ж робота відрізняється тим, що нашим основним завданням є визначення оптимальних параметрів нейронної мережі, які змушують агента, що навчається за допомогою підкріплення, отримувати більше винагороди. Конкретно, ми досліджуємо вплив різних ваг, зсувів та функцій активації на продуктивність агента. Це означає, що генеруючи агентів, з часом ви знайдете агента, який працює добре. Таким чином ми перекриваємо фундаментальні недоліки, оскільки у RL є дуже мало інформації, яку ми можемо використовувати для навчання алгоритму. З цього випливає декілька переваг, які властиві нашій роботі, таких як незалежність від джерела ініціалізації, простота реалізації та висока ефективність.

Додатково, ми вивчаємо, як включення знань про предметну область у генетичний алгоритм може допомогти спрямувати пошук до більш перспективних рішень. Ми використовуємо ці знання для ініціалізації популяції алгоритму та оцінки рішень.

Виділення недосліджених частин загальної проблеми. Генетичні алгоритми покладаються на метод спроб і помилок для еволюції агентів RL, що може бути обчислювально дорогим і трудомістким. Включення знань про предметну область у генетичний алгоритм може допомогти спрямувати пошук до більш перспективних рішень. Однак досліджень про те, як ефективно інтегрувати знання предметної області в процес еволюції, поки що мало.

У цьому дослідженні знання про предметну область включаються в генетичний алгоритм через використання спеціальних процедур ініціалізації. Замість випадкової ініціалізації, яка є типовою для багатьох генетичних алгоритмів, ми використовуємо процедуру, яка враховує знання про предметну область. Ця процедура ініціалізації включає в себе створення популяційних рішень, які вже відомі як ефективні або перспективні. Це може включати, наприклад, використання відомих ефективних стратегій або використання інформації про структуру проблеми для створення початкових рішень.

Цей алгоритм дозволяє почати роботу з перспективніших місць в просторі пошуку, що може прискорити збіжність до оптимального рішення. Проте необхідно зберегти деякий рівень випадковості в процесі ініціалізації, щоб забезпечити достатню різноманітність у популяції та уникнути передчасної збіжності.

Викладення основного матеріалу дослідження. Для того щоб моделювати середовище, ми використовуємо інструмент, розроблений за допомогою OpenAI, під назвою OpenAI Gym. Він пропонує декілька заздалегідь створених середовищ для тренування та тестування агентів навчання з підкріпленням, включаючи класичні завдання для фізичного контролю, відеоігри Atari, а також симуляції роботів [6].

Ми отримали доступ до середовища, під'єднавши пакет Gym, де Gym — це бібліотека Python з відкритим вихідним кодом для розробки та порівняння алгоритмів навчання з підкріпленням, яка забезпечує стандартний API для зв'язку між алгоритмами навчання та середовищами, а також стандартний набір середовищ, сумісних із цим API.

API Gym моделює середовища як прості env класи Python. Створювати екземпляри середовища та взаємодіяти з ними дуже просто. Тому розглянемо важливі функції для взаємодії із середовищем [7]:

- Reset: ця функція скидає середовище до початкового стану та повертає спостереження середовища, що відповідає початковому стану.
- Step: ця функція приймає дію як вхідні дані та застосовує її до середовища, що призводить до переходу середовища в новий стан. Функція скидання повертає чотири параметри:
  - Observation: спостереження за станом навколишнього середовища.
  - Reward: винагорода, яку ви можете отримати від середовища після виконання дії, наданої як вхідні дані для step функції.
  - Done: чи було перервано серію. Якщо істина, вам може знадобитися завершити симуляцію або скинути середовище, щоб перезапустити епізод.
  - Info: це надає додаткову інформацію залежно від середовища, таку як кількість спроб, що залишилися, або загальну інформацію, яка може бути сприятливою для налагодження.
- Render: якщо ви хочете побачити, як середовище виглядає в поточному стані, ви можете використовувати цей метод. Якщо ви хочете бачити скриншот гри як зображення, а не як вигулькове вікно, вам слід встановити mode аргумент render функції на rgb_array.
- Close: закриття вигулькового вікна.

У Cartpole стовп прикроплений за допомогою шарніру до візка, що рухається вздовж шляху без тертя. На початку стовп вертикальний, і метою є запобігання його падіння. Система контролюється шляхом прикладання сили зі значенням +1 чи -1 до візка. Винагорода +1 надається за кожен момент часу, коли стовп залишається вертикальним. Епізод закінчується коли стовп відхиляється більше ніж на 15 градусів від вертикалі, або візок від'їжджає більше ніж на 2,4 одиниці від центра шляху. Візуальна репрезентація візка зображена на рисунку 1.

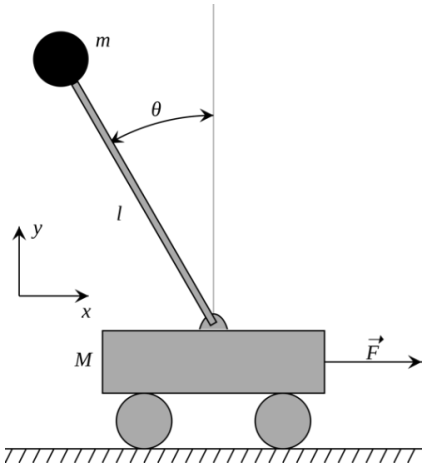

Рис. 1. Візуальна репрезентація CartPole

У цьому середовищі нашими спостереженнями будуть: позиція візка, швидкість візка, кут стовпа та швидкість повороту стовпа. На кожному кроці, агент може пересуватися двома способами: вліво або вправо.

У роботі використовується безградієнтний підхід до навчання з підкріпленням. Це означає що ми можемо навчати нейронні мережі без жодних обчислень градієнта. Отже, основна частина буде приділена генетичному алгоритму, оскільки з його допомогою ми удосконалюватимемо нашого агента. Тому розпочнемо з короткого його огляду.

Генетичні алгоритми – це типи алгоритмів, які засновані на концепції біологічної еволюції. Процес природного відбору починається з відбору найбільш пристосованих особин із популяції. Вони дають потомство, яке успадковує характеристики батьків і потомство буде додане до наступного покоління. Якщо батьки мають кращу фізичну форму, їхні нащадки будуть кращими за батьків і матимуть більше шансів вижити. Цей процес повторюється, і в кінці буде знайдено покоління з найбільш пристосованими особами [8]. Це поняття можна застосувати до проблеми пошуку. Ми розглядаємо набір рішень проблеми і вибираємо з них найкращі.

Щоб краще зрозуміти, як біологічну ідею можна перекласти в її математичне представлення найкраще спочатку розглянути кожен компонент окремо:

- Gene: ген або «індивід» – це просто кодування рішення, яке є довільним для кожної проблеми. Для нашої проблеми кодування генів є параметри (weights and biases) для нашої мережі.
- Population: популяція – це список генів, які наш алгоритм намагатиметься еволюціонувати, використовуючи обчислені рівні придатності кожного гена та еволюційні оператори.
- Fitness: фітнес оцінюється для кожної особи (гена) і також залежить від поточної проблеми. У нашому випадку придатність - це винагорода, яку агенти отримують від середовища.
- Evolutionary operators: існує безліч методів і операторів, які ми можемо використовувати в нашій еволюційній стратегії. Найпоширенішими є мутація та кросинговер.

– Мутація — це випадкова зміна частин гена (якщо ген — це рядок бітів, мутація може бути переворотом одного з бітів).

– Кросовер — це поєднання двох генів для утворення нового.

Розглянемо псевдокод генетичного алгоритму:

- Уявіть агента як організм.
- Параметрами будуть його гени, які визначають його поведінку (стратегію).
- Винагороди вказуватимуть на придатність організму (тобто чим вище винагорода, тим вище ймовірність виживання).
- У першій ітерації ви починаєте з X агентів із випадково ініціалізованими параметрами.

- Деякі з агентів випадково будуть працювати краще, ніж інші.
- За аналогією природної еволюції в реальному світі, ви реалізуєте виживання найбільш пристосованих: просто берете 10 % найпристосованіших агентів і відтворюєте їх для наступної ітерації, доки у вас знову не буде *X* агентів для наступної ітерації. Найслабших 90 % позбуваємось.
- Під час реплікації найкращих найпридатніших агентів, які становлять 10%, потрібно додати невеликий випадковий гаусівський шум до його параметрів, щоб у наступній ітерації ви могли досліджувати околиці навколо параметрів найкращих агентів.
- Зберігайте найефективнішого агента без додавання шуму, це підстрахування на випадок зниження продуктивності, яка може бути спричинена шумом Гаусса [9].

Схема роботи генетичного алгоритму представлена на рисунку 2.

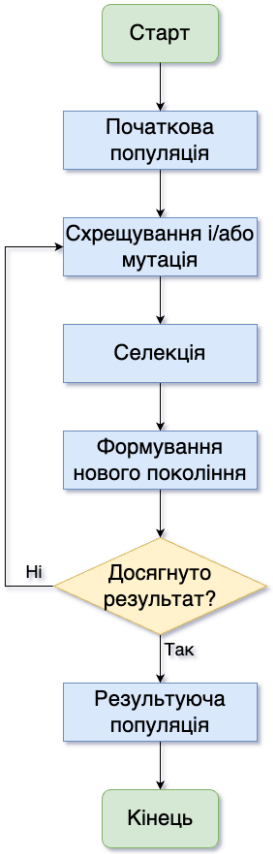

Рис. 2. Схема роботи генетичного алгоритму

Після огляду алгоритму роботи варто розглянути теоретичні переваги такого підходу:

- По-перше, еволюційні алгоритми гарантовано знайдуть глобальний мінімум поверхні втрат за достатньої кількості ітерацій, тоді як градієнтні методи можуть застрягти в локальних мінімумах.
- По-друге, такий алгоритм легко реалізувати та масштабувати для будь-якої мережі.

Натомість основним недоліком є те, що генетичні алгоритми не ідеальні. Наприклад, немає вказівок щодо вибору мультиплікативного коефіцієнта під час додавання гауссівського шуму. Не вірно підібравши його ваш алгоритм може повністю вийти з ладу.

Також варто описати нашу нейронну мережу. Використовуючи PyTorch [10], ми параметризуємо агента через двошарову нейронну мережу. Тут використовується послідовна модель (Sequential) і перший шар є лінійним (Linear), як і інший другий. У першому шарі є 128 нейронів та вхідні розміри дорівнюють розміру стану, тому чотири стани середовища є входами в нашу модель. Активація для нашого шару встановлена, як Relu. Наступний шар має 128 нейрони, а активацією є функція Softmax. Це неглибока нейронна

мережа. Наш вихідний рівень містить стільки нейронів, скільки у нас можливих дій. Тобто дві можливі дії – праворуч або ліворуч є вихідними нейронами. Графічне зображення мережі продемонстровано на рисунку 3.

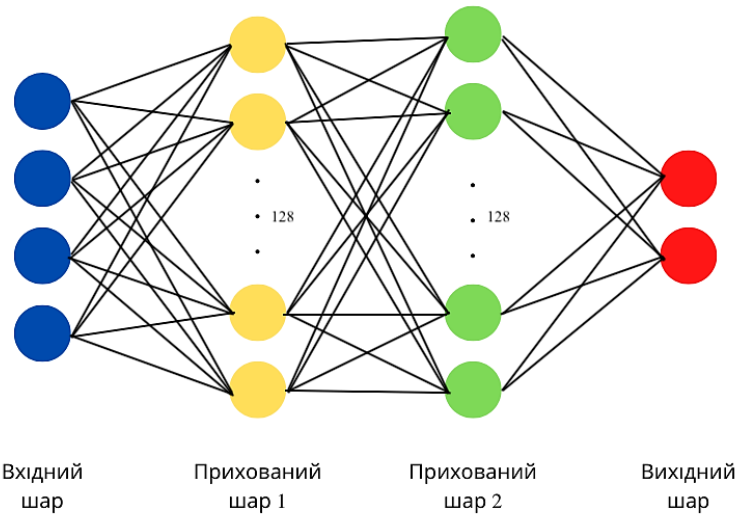

Рис. 3. Глибока нейронна мережа з двома прихованими шарами

У генетичному алгоритмі важливою частиною є вірний підбір параметрів, тому розглянемо їх на конкретному застосуванні в алгоритмі:

- Популяція становить 500 (numAgents) агентів, і ми генеруємо агентів випадковим чином під час першої ітерації. Розмір популяції має бути не надто великим, але й водночас не надто малим. Для вирішення нашої задачі оптимальною кількістю є саме 500, їх достатньо, щоб досліджувати простір.
- З 500-ста ми вибираємо лише 20 найкращих, як батьків (topLimit). Оптимальне значення батьків знаходиться як 1-10% від загальної кількості.
- Максимальна кількість ітерацій, яку ми хочемо запустити в циклі, становить 200 поколінь (generations). Хоча зазвичай для CartPole досить ефективний агент виявляється протягом кількох ітерацій.
- У кожному поколінні ми спочатку запускаємо всіх випадково згенерованих агентів і отримуємо їхню середню продуктивність за 3 прогони (все-таки одного разу може пощастити, тому ми хочемо усереднити).
- Ми сортуємо агентів у порядку спадання їхніх винагород (продуктивностей).
- Потім ми беремо 20 найкращих агентів і вибираємо випадковим чином серед них, щоб створити дітей для наступної ітерації. Також додаємо невеликий гауссівський шум до всіх параметрів під час копіювання агента, але зберігаємо одного найкращого з найкращих елітних агентів (без додавання шуму). Гауссівський шум – це мультиплікативний коефіцієнт, який є гіперпараметром і приблизно нагадує швидкість навчання в градієнтному спуску. Методом підбору було знайдено оптимальне значення, яке становить 0,02.
- Під час реплікації найкращих найпридатніших агентів, які становлять 10%, потрібно додати невеликий випадковий гауссівський шум до його параметрів, щоб у наступній ітерації ви могли досліджувати околиці навколо параметрів найкращих агентів.
- З дочірніми агентами тепер як батьками ми повторюємо та запускаємо весь цикл знову, доки не буде виконано всі 200 поколінь або ми не знайдемо агента з хорошою продуктивністю.

На рисунку 4 зображено графік середніх винагород, які отримали агенти на кожному з поколінь, тобто ми можемо відстежувати еволюцію агентів:

- generation 1: ми випадковим чином зініціювали агентів та отримали середню винагороду (meanRewards) для усіх агентів − 22, для найкращих п’яти (meanT opRewards) − 46.
- generation 26: meanRewards = 50, a meanT opRewards = 110.
- generation 52: середня винагорода (meanRewards) для усіх агентів становить 100, для найкращих п’яти (meanTopRewards) − 183. Це є дуже хорошим показником, адже наші агенти навчились проходити вже 100 кроків гри.
- generation 55: ми отримали першого агента, який отримав винагороду 200, тобто один агент вже зумів пройти гру.
- generation 63: meanRewards = 135, a meanTopRewards = 200. Тобто у нас вже є 5 агентів, які пройшли гру до кінця.
- generation 68: meanRewards = 150, що свідчить про те, що значна кількість агентів практично досягають успіху.
- generation 71: 20 агентів отримали максимальну винагороду 200.
- generation 97: meanRewards = 181, що є показником, що еволюцію агентів можна було б припинити, але ми продовжуємо, щоб отримати ще кращі результати.
- generation 139: починаючи з цього покоління середня винагорода всіх агентів meanRewards становить 195 і вище.
- generation 200: meanRewards = 197, можемо з впевненістю стверджувати, що практично всі наші агенти вміють досягати максимуму.

Графік 4 свідчить про результативність обраного підходу та швидку еволюцію агентів. Тепер залишилось переконатись у ефективності еволюціонованих агентів.

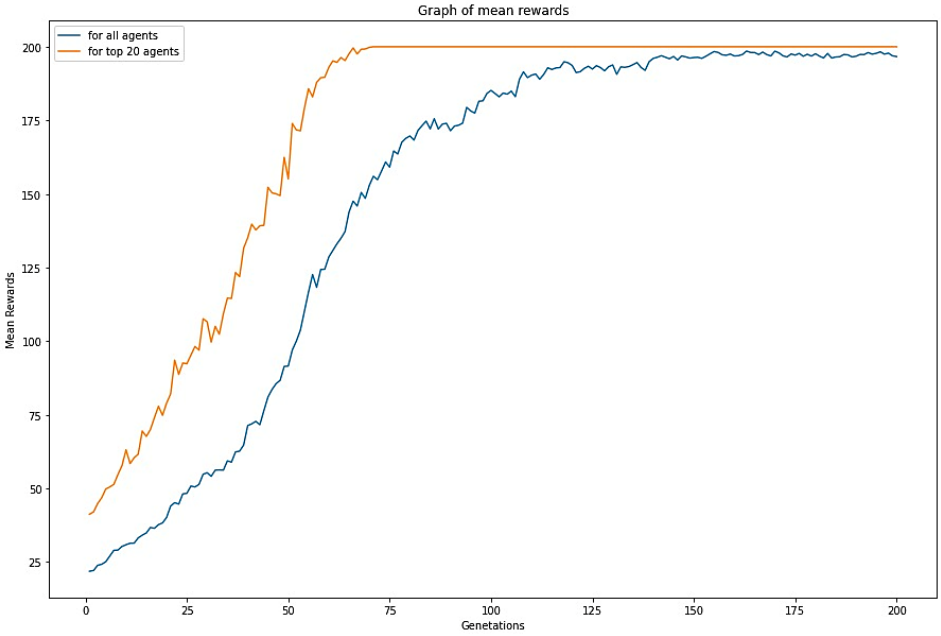

Рис. 4. Графік середніх винагород агентів на кожному поколонні

На рисунку 5 продемонстровано графік залежності винагороди для випадкових ста агентів з нашої популяції агентів, яка становить 500. Ми можемо побачити, що зі ста агентів, тільки двоє не досягли максимальної винагороди, вони отримали 199 та 189, що є також хорошим результатом.

Підбиваючи підсумки, можна заявити, що 2 % агентів можуть не виграти гру, натомість 98 % агентів успішно пройдуть гру CartPole-v0. Це є хорошим показником того, що ми еволюціонували агентів, та навчили їх досягати максимальної винагороди.

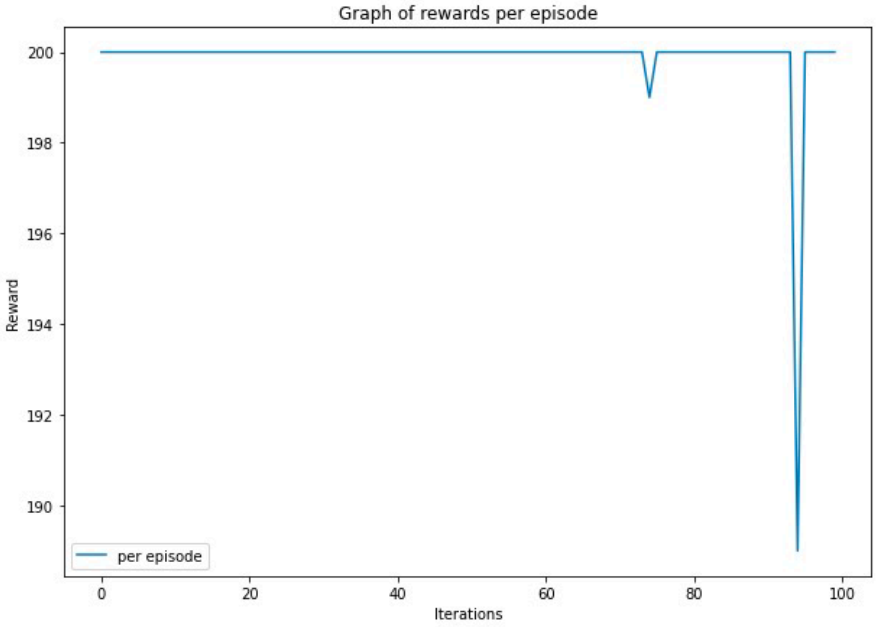

Рис. 5. Графік середніх винагород випадкових ста агентів

Висновки. У цій роботі було проведено дослідження використання генетичного алгоритму для еволюції агентів навчання з підкріпленням. Було виявлено, що генетичний алгоритм може бути ефективним інструментом для покращення процесу навчання агентів.

У цій роботі ми провели детальне дослідження використання генетичного алгоритму для еволюції агентів навчання з підкріпленням. Наші результати показали, що генетичний алгоритм може значно покращити процес навчання агентів, зокрема в контексті складних задач, де традиційні методи навчання можуть бути недостатніми.

Ми виявили, що використання генетичного алгоритму дозволяє агентам більш ефективно адаптуватися до нових ситуацій, що підтверджує його потенціал як інструмента для покращення процесу навчання.

Однак, ми також виявили, що вибір оптимальних параметрів генетичного алгоритму є важливим фактором, який впливає на його ефективність. Тому ми рекомендуємо провести додаткові дослідження для визначення оптимальних параметрів генетичного алгоритму в різних контекстах.

Також, ми бачимо потенціал для подальшого дослідження в області комбінації генетичного алгоритму з іншими методами машинного навчання. Це може відкрити нові можливості для покращення ефективності навчання агентів.

Список використаних джерел

1. The marginal value of adaptive gradient methods in machine learning [Electronic resource] / A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, B. Recht // Advances in neural information processing systems. – Access mode: https://arxiv.org/abs/1705.08292.

2. Arzate C. A survey on interactive reinforcement learning: Design principles and open challenges / C. Arzate, T. Igarashi / Proceedings of the 2020 ACM designing interactive systems conference (2020, July). – Pp. 1195-1209.

3. Evolution strategies as a scalable alternative to reinforcement learning [Electronic resource] / T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever. – 2017. – Access mode: arXivpreprintarXiv:1703.03864.

4. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning [Electronic resource] / F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, J. Clune. – Access mode: https://arxiv.org/abs/1712.06567.

5. Khadka S. Evolution-guided policy gradient in reinforcement learning [Electronic resource] / S. Khadka, K. Tumer // Advances in Neural Information Processing Systems. – 2018. – № 31. – Access mode: https://arxiv.org/abs/1805.07917.

6. Openai gym [Electronic resource] / G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba. – Access mode: https://arxiv.org/abs/1606.01540.

7. Nandy A. OpenAI basics [Electronic resource] / A. Nandy, M. Biswas // Reinforcement Learning. – 2018. – Pp. 71-87. – Access mode: https://link.springer.com/chapter/10.1007/978-1-4842-3285-9_3.

8. Lambora A. Genetic algorithm-A literature review / A. Lambora, K. Gupta, K. Chopra // 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon) (2019, February). – Pp. 380-384.

9. Genetic algorithm: Review and application / M. Kumar, D. Husain, N. Upreti, D. Gupta // International Journal of Information Technology and Knowledge Management. – 2010. – Vol. 2, No. 2, July-December. – Pp. 451-454.

10.Mishra P. Introduction to neural networks using PyTorch / P. Mishra // PyTorch Recipes: A Problem-Solution Approach to Build, Train and Deploy Neural Network Model. – Apress, Berkeley, CA, 2023. – Pp. 117-133.

References

1. Wilson, A.C., Roelofs, R., Stern, M., Srebro, N., & Recht B. (2017). The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*. https://arxiv.org/abs/1705.08292.

2. Arzate, C., & Igarashi, T. (2020, July). A survey on interactive reinforcement learning: Design principles and open challenges. *Proceedings of the 2020 ACM designing interactive systems conference* (pp. 1195-1209).

3. Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). *Evolution strategies as a scalable alternative to reinforcement learning*. arXivpreprintarXiv:1703.03864.

4. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017). *Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning*. https://arxiv.org/abs/1712.06567.

5. Khadka, S., & Tumer, K. (2018). Evolution-guided policy gradient in reinforcement learning. *Advances in Neural Information Processing Systems*. https://arxiv.org/abs/1805.07917.

6. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *Openai gym*. https://arxiv.org/abs/1606.01540.

7. Nandy, A., & Biswas, M. (2018). OpenAI basics. *Reinforcement Learning* (pp. 71-87). https://link.springer.com/chapter/10.1007/978-1-4842-3285-9_3.

8. Lambora, A., Gupta, K., & Chopra, K. (2019, February). Genetic algorithm-A literature review / A. Lambora // *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)* (pp. 380-384).

9. Kumar, M., Husain, D., Upreti, N., & Gupta, D. (2010). Genetic algorithm: Review and application. *International Journal of Information Technology and Knowledge Management, 2*(2), 451-454.

10. Mishra, P. (2023). Introduction to neural networks using PyTorch. *PyTorch Recipes: A Problem-Solution Approach to Build, Train and Deploy Neural Network Model* (pp. 117-133). Apress, Berkeley, CA.

Отримано 10.05.23

УДК 004.421

Artem Volokyta[1], Bohdan Hereha[2]

[1]PhD in Technical Sciences, Associate Professor of Department of Computer Engineering
National Technical University of Ukraine "Ihor Sikorskyi Kyiv Polytechnic Institute" (Kyiv, Ukraine)
E-mail: artem.volokita@kpi.ua. **ORCID:** http://orcid.org/0000-0001-9069-5544

[2]Student of the 6th year of the Faculty of IOT
National Technical University of Ukraine "Kyiv Polytechnic Institute named after Igor Sikorsky" (Kyiv, Ukraine)
E-mail: bogdangerega19@gmail.com

EVOLUTION OF REINFORCEMENT LEARNING AGENTS USING THE GENETIC ALGORITHM

Reinforcement learning (RL) allows agents to make decisions based on a reward function. However, in the process of learning, the choice of the values of the parameters of the learning algorithm can significantly affect the overall learning process. Agents using the policy gradient algorithm can be trained for a long time, but even then, they may not behave perfectly.

Thinking more about it, we realized that the reason for the long training is that gradients are almost absent, and therefore not very useful. Gradients help in supervised learning tasks, such as image classification, by providing useful information on how to change the parameters (weights or offsets) of the network for better accuracy. In image classification, after each mini-series of training, backpropagation provides a clear gradient (direction) for each parameter in the network. In reinforcement learning, however, the gradient information is only provided occasionally when the environment provides a reward or punishment. In most cases, our agent performs actions without knowing whether they are useful or not. Therefore, in this paper, we will improve the agents by using a genetic algorithm, i.e., we evolve the agents.

This research explores the use of genetic algorithms to improve the performance of reinforcement learning agents. We conducted a series of trials using various neural network parameters, including weights, biases, and activation functions, in order to find the optimal values that cause the agent to receive more rewards. Our approach includes the use of domain knowledge to initialize the population of the genetic algorithm as well as to evaluate solutions. This allows us to direct the search towards more promising solutions. Special attention is paid to the impact of various genetic algorithm parameters on learning efficiency. The potential applications of this research are broad, ranging from robotics and autonomous vehicles to gaming and finance. The results of the study can also be used to develop new algorithms and methods to improve the performance of reinforcement learning agents, which further contributes to the development of machine learning.

Our research has shown that the use of a genetic algorithm can significantly improve the efficiency of agent learning. The result is the successful completion of the CartPole-v0 game by evolved agents. 98 % of our population will reach the maximum, i.e. successfully complete the game.

Keywords: *reinforcement learning; genetic algorithm; agent; gradient-free approach; neural network; CartPole; policy gradients.*

Fig.: 5. References: 10.

Волокита А., Герега Б. Еволюція агентів навчання з підкріпленням за допомогою генетичного алгоритму. *Технічні науки та технології.* 2023. № 2(32). С. 175-184.