УДК 004.89 http://doi.org/10.15407/pp2023.04.098

А.Є. Вітюк, А.Ю. Дорошенко

ПРОГРАМНИЙ ПАКЕТ ДЛЯ АДАПТИВНОГО НАВЧАННЯ КОНТРОЛЕРІВ РОБОРУКИ НА ОСНОВІ НЕЙРОМЕРЕЖ

В статті досліджено процес розробки програмних засобів із застосуванням алгоритмів нейроеволюції для навчання нейромережевих контролерів у процесі управління роботизованою кінцівкою. Описано основні аспекти нейроеволюційного підходу для навчання нейромережі у задачах, що не мають найкращого рішення та потребують навчання з підкріпленням. Задача позиціонування має такий ландшафт функцій пристосованості, що робить її схильною потрапляти у пастки локального оптимуму, звідки виникає потреба розробки програмного забезпечення для роботи з нейроеволюційними підходами.

Розробка оптимальних засобів управління із застосуванням нейроеволюційного підходу може забезпечити не лише високу точність позиціонування, а й мінімальну конфігурацію, що підвищує швидкодію системи. Майбутні дослідження з використанням представленого пакету для вирішення задач маніпулювання та позиціонування становлять особливий інтерес для покращення програмних контролерів управління роботизованими кінцівками в динамічних умовах реального світу.

Ключові слова: нейронна мережа, нейроеволюція, роботизована рука, навчання з підкріпленням, NEAT, функція пристосованості.

Вступ

Використання контролерів на основі нейронних мереж у роботизованих системах із робочою кінцівкою-маніпулятором вивела розробку програмного забезпечення для управління роботизованих систем на новий рівень. Використання структур адаптивної нейронної мережі в розробці контролерів для задач маніпулювання рукою робота відкриває нові можливості для дослідження і розробки таких систем. Основна роль адаптивних контролерів випливає з їхньої здатності вирішувати задачі, властиві широкому спектру сценаріїв маніпулювання цільовим об'єктом робочою кінцівкою із захватним пристроєм.

Задача позиціонування роборуки та маніпулювання цільовим об'єктом характеризується своєю складністю, передбачає точний контроль кількох ступенів свободи та взаємодії з динамічним середовищем невідомої конфігурації. Нейронні мережі, з їхньою здатністю навчатися та представляти складні залежності від вхідних даних, пропонують автоматизований шлях для вирішення цих проблем. Крім того, адаптивність структур нейронної мережі додатково надає таким системам здатність динамічно налаштовувати свою архітектуру відповідно до вимог різноманітних завдань, що можуть змінюватися.

Важливість автоматизованої розробки підкреслюється потребою в контролерах, які можуть легко адаптуватися до мінливого середовища, різних форм об'єктів маніпулювання і факторів, пов'язаних із середовищем. Навчальні можливості нейронних мереж, особливо з адаптивними структурами, роблять їх придатними для сценаріїв, де класичне програмування не може забезпечити знаходження мінімальної моделі, розмір якої є критично важливим для забезпечення заданого рівня ефективності.

Процес розробки контролерів на основі нейронних мереж для роборуки має ряд складнощів. Починаючи від отримання та анотації різноманітних наборів даних і закінчуючи складним балансом між складністю моделі та її налаштуваннями. Дослідження еволюційних підходів для навчання адаптивних контролерів дозволить пришвидшити розробку контролерів для широкого спектру можливих сценаріїв використання кінцівки роборуки.

Програмне середовище для тренування агента-маніпулятора з використанням нейроеволюційного підходу має на меті автоматизацію процесу розробки, що є особливо актуальним у сценаріях, де ручне проєктування та налаштування є непрактичним або забирає багато часу.

1. Існуючі програмні рішення

Активний інтерес до дослідження та розробки систем, що базуються на навчанні з підкріпленням, дав поштовх для створення програмних середовищ, які дозволяють легку інтеграцію засобів для навчання моделі та інструментів для роботи з різноманітними агентами та середовищами.

Бібліотека Stable-Baselines3 [1] для навчання з підкріпленням мовою Python забезпечує набір високоякісних реалізацій різних алгоритмів навчання з підкріпленням. Вона створена на основі OpenAI Gym, що дозволяє легко використовувати її для тренувань агентів у різних середовищах, зокрема, для робототехнічних завдань. Бібліотека надає простий і зручний API для роботи з алгоритмами навчання з підкріпленням. Це робить її доступною як для початківців, так і для досвідчених користувачів. Крім того, вона пропонує різноманітні алгоритми навчання з підкріпленням, у тому числі такі популярні, як PPO (наближена оптимізація політики), DDPG (глибокі детерміновані градієнти політики) і TRPO (оптимізація політики довірчого регіону).

Stable-Baselines3 легко інтегрується з OpenAI Gym, набором інструментів для розробки та порівняння алгоритмів навчання з підкріпленням. Це дозволяє користувачам використовувати широкий спектр середовищ, зокрема, для задач маніпуляції з роботизованими робочими кінцівками. Бібліотека забезпечує добре оптимізовану реалізацію різноманітних алгоритмів, що робить її придатною як для досліджень, так і для практичних застосувань.

Проте, хоча Stable-Baselines3 підтримує низку алгоритмів, деякі з них більш придатні для дискретних просторів дій. Для просторів безперервної дії частіше використовуються такі алгоритми, як PPO та TRPO. Крім того, Stable-Baselines3 розроблено для навчання підкріплення з одним агентом. Якщо завдання передбачає мультиагентну взаємодію (наприклад, співпрацю кількох робототехнічних рук для маніпуляції одним об’єктом), інструментів бібліотеки буде недостатньо.

Важливою особливістю є підтримка Stable-Baselines3 фізичного симулятора MuJoCo, що широко використовується для проєктування робототехнічних агентів, зокрема, маніпуляторів та крокуючих роботів.

Альтернативою Stable-Baselines3 є бібліотека TF-Agents з відкритим кодом, побудована на TensorFlow та розроблена Google. Вона забезпечує модульну та розширювану структуру для навчання з підкріпленням. TF-Agents підтримує сценарії як з одним, так і з кількома агентами та має реалізацію різних алгоритмів.

Ray RLlib — ще одна бібліотека для навчання з підкріпленням, створена на основі розподіленої обчислювальної системи Ray. Вона підтримує широкий спектр алгоритмів і пропонує такі функції, як розподілене навчання та налаштування гіперпараметрів.

Для порівняння ефективності навчання контролерів роборуки на основі нейромереж доступні й інші бібліотеки, які мають схожий функціонал, але відрізняються набором алгоритмів навчання з підкріпленням. Серед них Garage, Tianshou, SLM Lab. Порівняння найбільш поширених бібліотек представлено в таблиці 1.

Таблиця 1. Програмні бібліотеки для дослідження методів машинного навчання роботизованих агентів

Бібліотека	Використана бібліотека глибокого навчання	Мультиагентна підтримка	Зручність використання API	Підтримка середовищ OpenAI Gym	Підтримка еволюційних алгоритмів	Переваги
Satble-Baselines3	TensorFlow	Обмежена	Висока	+	-	Великий вибір алгоритмів RL, інтеграція OpenAI Gym

Бібліотека	Використана бібліотека глибокого навчання	Мультиагентна підтримка	Зручність використання API	Підтримка середовищ OpenAI Gym	Підтримка еволюційних алгоритмів	Переваги
TF-Agents	TensorFlow	+	Середня	+	-	Модульність, підтримка сценаріїв з одним або кількома агентами
Ray RLib	TensorFlow, PyTorch	+	Середня	+	+ (Еволюційні стратегії)	Розподілене навчання, налаштування гіперпараметрів, різноманітне API
Garage	TensorFlow	+	Середня	+	+ (Генетичні алгоритми, CEM)	Розширена підтримка алгоритмів
Tianshou	PyTorch	+	Середня	+	+ (CMA-ES)	Модульність, широкий вибір алгоритмів
SLM Lab	TensorFlow	+	Середня	+	+ (CEM)	Модульність

Як бачимо, існуючі програмні рішення надають широкий вибір інструментів для дослідження алгоритмів навчання з підкріпленням для роботизованих агентів у визначеному середовищі. Проте більшість із них не надає підтримки нейроеволюційних підходів, які уможливлюють розробку адаптивних роботизованих систем, що є важливим для дослідження багатофункціональної системи, максимально наближеної до роботи із задачами реального світу.

2. Застосування нейроеволюційного підходу

Генетичні алгоритми — це клас евристик пошуку, натхненних процесом природної еволюції. У природному відборі, ключовому механізмі біологічної еволюції, особини, які є більш придатними для умов, у яких вони живуть, «відбираються» для продовження роду, оскільки вони мають більшу ймовірність вижити до зрілості та знайти пару, ніж менш придатні особини. Генетичні алгоритми застосовують цю концепцію до простору пошуку, де «індивіди» є точками в просторі пошуку, що представляють можливі рішення.

Алгоритм починається з початкового покоління рішення, яке зазвичай генерується випадково. Функція пристосованості оцінює пристосованість кожного кандидата, а алгоритм обирає підмножину кандидатів на основі її значень. Ці кандидати поєднуються один з одним шляхом рекомбінації, щоб сформувати нове покоління рішень-кандидатів з вищою середньою пристосованістю. Залежно від конкретного алгоритму, вони також можуть бути змінені (їхні параметри можуть змінюватися випадковим чином), щоб розширити процес пошуку на нові горизонти простору пошуку. Процес повторюється, доки не буде досягнуто умови зупинки – залежно від алгоритму це може статися через певну кількість поколінь, або коли досягнуто певного рівня фізичної підготовки.

Двома основними елементами потенційного рішення є генотип і фенотип. Генотип — це набір параметрів, які визначають це конкретне рішення. Генотип можна закодувати будь-яким способом за умови, що кодування доступні для пошуку - одним із найпростіших і найпоширеніших кодувань є рядок бітів. Таке кодування називається фенотипом.

Генетичні алгоритми корисні в багатьох галузях, включно із економікою, проєктуванняи систем, криптоаналізом, відеоіграми та логістикою. Основна перевага генетичних алгоритмів перед іншими методами машинного навчання полягає в тому, що вони не вимагають попереднього аналізу предметної області, оскільки вони починаються з випадкового, неоптимального набору варіантів рішень і використовують еволюційні концепції для пошуку оптимального рішення. Це робить їх дуже корисними для вирішення задач із обмеженими знаннями області. Оскільки мутація гарантує, що їх пошук охоплює різні частини простору рішень, вони також добре справляються з проблемами, де функція помилок має локальні мінімуми.

Нейроеволюція — це еволюційний метод навчання, який поєднує генетичні алгоритми та нейронні мережі, в якому нейронні мережі є фенотипом генетичного алгоритму, а параметри нейронної мережі, такі як топологія або ваги з'єднань, є генотипом [9]. Алгоритм шукає нейронну мережу, оптимальну для певної задачі. По суті, алгоритм розвиває найкращу структуру для даної проблеми.

Алгоритм нейроеволюції може знаходити топологію мережі, або ваги з'єднань, або те й інше. Якщо він шукає обидва параметри, він може розвивати їх окремо один від одного або паралельно. Також може використовуватись пряме кодування, за якого кожен зв'язок і вузол фенотипу вказується в його генотипі, або непряме кодування, коли генотипи просто забезпечують правила для побудови фенотипів.

Нейроеволюційні алгоритми особливо корисні для класів задач, у яких правильна нейронна мережа була б корисною для вирішення проблеми, але важко або неможливо визначити правильно розмічені вхідні дані, необхідні для навчання з учителем. Зазвичай їх визначають як тип навчання з підкріпленням [4].

Поширені алгоритми NEAT і HyperNEAT розвивають як топологію мережі, так і ваги з'єднань. NEAT використовує пряме кодування, а HyperNEAT використовує схему непрямого кодування [2].

3. Алгоритм NEAT для управління агентом із роботизованою кінцівкою

NEAT (NeuroEvolution of Augmenting Topologies) — це алгоритм нейроеволюції, розроблений Остіні 2002 року [3]. Він кодує структуру мережі та ваги з'єднань за допомогою вузлів і використовує біологічні принципи підвищення складності та видоутворення.

Структура генотипу NEAT містить набір генів нейронів і набір генів ланок. Гени нейрона присвоюють нейрону ідентифікаційний номер і вказують, чи є він вхідним, вихідним або прихованим вузлом. Гени зв'язку містять інформацію про два нейрони, до яких вони під'єднані, очікування з'єднання цього зв'язку та прапорці, які вказують, чи є зв'язок активним. Вони також мають номер оновлення, щоб вказати порядок, у якому вони були додані. Щоразу, коли додається з'єднання, алгоритм перевіряє, чи існувало воно раніше (чи воно там було, але видалене, на відміну від того, що воно було повністю новим). Якщо так, йому присвоюється номер попередньої інновації, інакше йому присвоюється наступний доступний.

NEAT розвиває дедалі складніші нейронні мережі відповідно до складності задачі. NEAT розвиває як вагу з'єднання, так і топологію, що приводить до значного підвищення продуктивності. Було показано, що NEAT є ефективним для багатьох задач, таких як балансування маятника, керування роботами, управління транспортними засобами, настільні й відеоігри.

NEAT може формувати різноманітну популяцію мереж шляхом додавання нових генів і розумного схрещування. Однією з проблем цього підходу є те, що менші мережі оптимізуються швидше, ніж великі, а додавання нових зв'язків і генів зазвичай спочатку знижує придатність мережі, тому новостворені структури мають менші шанси вижити. Цю проблему вирішує видоутворення, яке поділяє популяцію на окремі підпопуляції. Структура кожної особини динамічно порівнюється з іншими, а ті, що мають подібну структуру, групуються. Особи в межах виду поділяють загальну пристосованість виду та конкурують переважно всередині цього виду. Специфікація дозволяє оптимізувати нові інновації, не стикаючись з конкуренцією з боку особин із різними структурами.

На рисунку 1 зображено мережу, розроблену для завдання позиціонування кінцівки. Мережа є складною та має багато прихованих шарів для ефективної апроксимації нелінійних зворотних кінематичних рівнянь, необхідних для переміщення кінцівки.

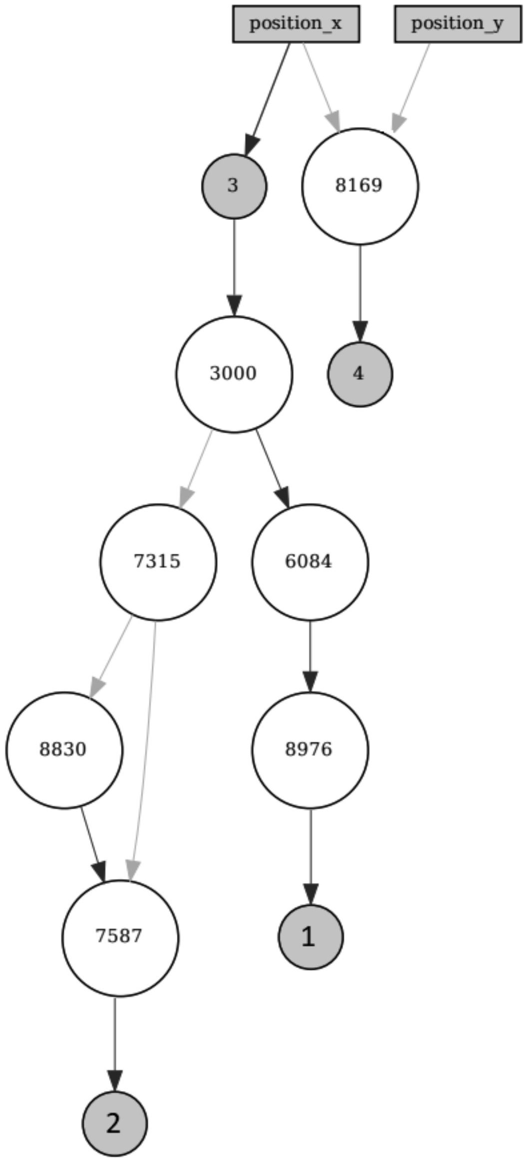

Рис. 1. Мережа для позиціонування роботизованої кінцівки

4. Застосування нейро-еволюційного підходу для вхідних даних у форматі зображень

Алгоритм NEAT є ефективним для задачі управління роботизованою рукою, де вхідні дані представлені набором даних з кількох датчиків, наприклад, двома координатами кожного з'єднання. Якщо ж вхідна інформація про стан агента надходить з камери, розміщеної на робочій кінцівці або на основі робота, задача стає на порядок складнішою. Для такої системи доцільним є використання алгоритму HyperNEAT [3].

HyperNEAT (Hypercube-based Neuro-Evolution of Augmenting Topologies) є розширенням NEAT, яке використовує форму непрямого кодування під назвою композиційні шаблони створення мереж (CPPN). Воно було створено як спроба скоротити розрив між результатами, отриманими алгоритмами нейроеволюції, та масштабом природного мозку.

У CPPN вхідні дані є декартовими координатами в n-вимірному просторі, де n є кількістю входів, і передаються у функцію f. Значення f, оцінене за кожною координатою, забезпечує наявність, відсутність або інтенсивність вираження точки в n-вимірному просторі, і тому фенотип, отриманий оцінкою f (який можна розглядати як генотип), є шаблоном. Різниця між звичайними штучними нейронними мережами і CPPN полягає в тому, що штучні нейронні мережі зазвичай містять лише сигмоїдальні функції, тоді як CPPN можуть містити багато різних типів функцій.

Хоча CPPN можна розробити за допомогою генетичних алгоритмів так само, як і штучні нейронні мережі, у HyperNEAT CPPN не розробляються як кінцевий продукт, а використовуються для кодування нейромережі. Оскільки CPPN створює просторовий шаблон, а нейромережа є шаблоном зв'язків, алгоритм потребує відображення між просторовим шаблоном і шаблоном зв'язку. Вхідними даними в CPPN є дві точки, а не одна, і дві точки представляють два вузли в нейромережі, а вихід функції CPPN є вагою зв'язку між двома вузлами. Таким чином, модель, створена CPPN, є топологією нейронної мережі. Цей CPPN, що створює топологію, називається з'єднувальним CPPN, на відміну від просторових CPPN, які створюють просторові моделі. З'єднувальний CPPN бере сітку вузлів, яка називається субстратом, і запитує кожне потенційне з'єднання, для визначення, чи дійсно існує з'єднання та яка його вага, беручи положення двох вузлів, і знаходить вагу з'єднання між ними. Це створює шаблон з'єднань між вузлами в просторі, який є похідним від конфігурації субстрату.

HyperNEAT розвиває CPPN за допомогою стратегії NEAT. CPPN є представленнями шаблонів у гіперпросторі, де кожна точка обмежена чотиривимірним гіперкубом. Отже, шаблон у чотиривимірному гіперкубі можна відобразити на двовимірній нейронній мережі. Першим кроком HyperNEAT є вибір субстрату та зв'язок між входами та виходами. Алгоритм ініціалізує перше покоління мінімальних CPPN з випадковими вагами, так само, як NEAT ініціалізує перше покоління мінімальних штучних нейронних мереж з випадковими вагами. Потім він запитує кожну CPPN, і створює шаблон зв'язку, подібний до нейромережі. Алгоритм визначає придатність цієї нейромережі так само, як це робить NEAT зі своїми мережами, і розвиває друге покоління CPPN відповідно до стратегії NEAT. Це повторюється, як і будь-який алгоритм нейроеволюції, доки не буде знайдено рішення.

HyperNEAT особливо добре відповідає задачам, де важлива інформація про геометричні представлення. За допомогою субстрату він «бачить» координати нейронів. Він може «бачити», який піксель поруч із яким на зображенні, або який квадрат біля якого на шахівниці, замість того, щоб вивчати їх на прикладах, як це робила б традиційна нейронна мережа.

5. Програмна реалізація пакету для адаптивного навчання контролерів робочої кінцівки

Пакет для адаптивного навчання нейромережі, що буде використовуватись для окремої задачі маніпулювання має забезпечувати підтримку як середовища для простору дій агента-роборуки, так і можливості використання необхідного нейроеволюційного алгоритму зі зручним налаштуванням параметрів та інструментами для аналізу результатів.

Для цього розроблений пакет містить набір модулів:

- *Config*: для роботи з параметрами нейроеволюційного алгоритму та його налаштування
- *Population*: для роботи з відповідним типом популяції
- *Reporter*: для забезпечення зручного представлення результатів користувачеві.

Для навчання політики у формі нейромережі для управління робочою кінцівкою в середовищі була обрана бібліотека NEAT-Python як реалізація алгоритму NEAT. На її основі працює модуль *Population.* Бібліотека NEAT-Python забезпечує реалізацію стандартних методів NEAT для моделювання генетичної еволюції геномів організмів у популяції. Вона містить утиліти для перетворення генотипу організму в його фенотип (штучну нейронну мережу) і надає зручні методи для завантаження та збереження конфігурацій геному разом із параметрами NEAT. Крім того, вона надає корисні процедури, які допомагають збирати статистичні дані про хід еволюційного процесу та зберігати/завантажувати проміжні контрольні точки. Контрольні точки дозволяють нам періодично зберігати стан еволюційного процесу і пізніше відновлювати виконання процесу [1].

Набір середовищ може бути використаний для дослідження завдань роботизованої руки за допомогою наступних наявних інтерфейсів.

Двовимірний симулятор з дискретним простором дій. Робот складається з двох з'єднань, кожне довжиною 100 пікселів, і мета — досягти червоної крапки, яка випадковим чином генерується в кожному епізоді. Простір дій складається з керування кутами з'єднання маніпулятора робота (7 можливих управляючих подій). Простір спостережень дає інформацію про поточний стан навколишнього середовища, він є безперервним і містить відповідну інформацію про цільове та поточне положення з'єднань (4 можливі стани) [7].

Двовимірний симулятор з неперервним простором дій. Подібно до попереднього агента, робот складається з двох з'єднань, кожне довжиною 100 пікселів, і мета — досягти червоної крапки, яка випадково генерується в кожному епізоді. Але простір дій складається з неперервного сигналу керування кутами з'єднання маніпулятора робота (два значення для з'єднань у проміжку -1..1). Простір спостережень є безперервним і містить відповідну інформацію про цільове та потоне положення з'єднань (4 можливі стани).

Тривимірний симулятор PandaReach. Симулятор роборуки Franka Emika Panda, який має на меті позиціонування робочої кінцівки у цільову точку тривимірного середовища. Значення винагороди повертається додатне, якщо цільова позиція досягнута. Управляюча дія відповідає переміщенню кінцевого елемента роборуки.

Тривимірний симулятор KukaCamBulletEnv. Середовище, в якому поточний стан представлений зображенням з камери, що розміщена на платформі основі робота. Керуючими діями є повороти відповідних кутів з'єднання.

Вибір відповідної модифікації алгоритму NEAT може бути здійснений шляхом заповнення конфігураційного файлу, обробку якого здійснює модуль *Config*:

```
if is_hyper:
    config = GymHyperConfig(args)
if is_eshyper:
    config = GymEsHyperConfig(args)
else:
    config = GymNeatConfig(args)
```

Для прискорення процесу навчання було використано підхід паралелізації обчислень функції пристосованості між процесорами за допомогою класу ParallelEvaluator:
neat.ParallelEvaluator(mp.cpu_count(), config.eval_genome, timeout=20)

Розглянемо основний функціонал пакету на прикладі роботи із середовищем двовимірної роборуки з неперервним простором подій.

Параметри обраного середовища та максимальну кількість повторень у підході встановлюються у конфігураційному файлі, який обробляється модулем Config:

```
environment = gym_robot_arm:robot-arm-v1
episode_reps = 10
```

Оптимальний набір параметрів відповідної модифікації алгоритму NEAT також встановлений за допомогою конфігураційного файлу:

```
[NEAT]
compatibility_disjoint_coefficient = 1.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultGenome]
conn_add_prob           = 0.3
conn_delete_prob        = 0.2

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism      = 1

[DefaultReproduction]
survival_threshold = 0.2
min_species_size   = 2
```

Після цього обрана функція оцінки може бути встановлена, та запущений процес навчання на кількості поколінь, заданій користувачем:

winner = population.run (parallel_evaluator.evaluate, params.ngen, params.maxtime)

Структура початкової мережі, структура мережі на найкращому геномі та графік видоутворення будуть збережені для подальшого аналізу. Структура мережі-переможця для тестового випадку представлена на рисунку 2:

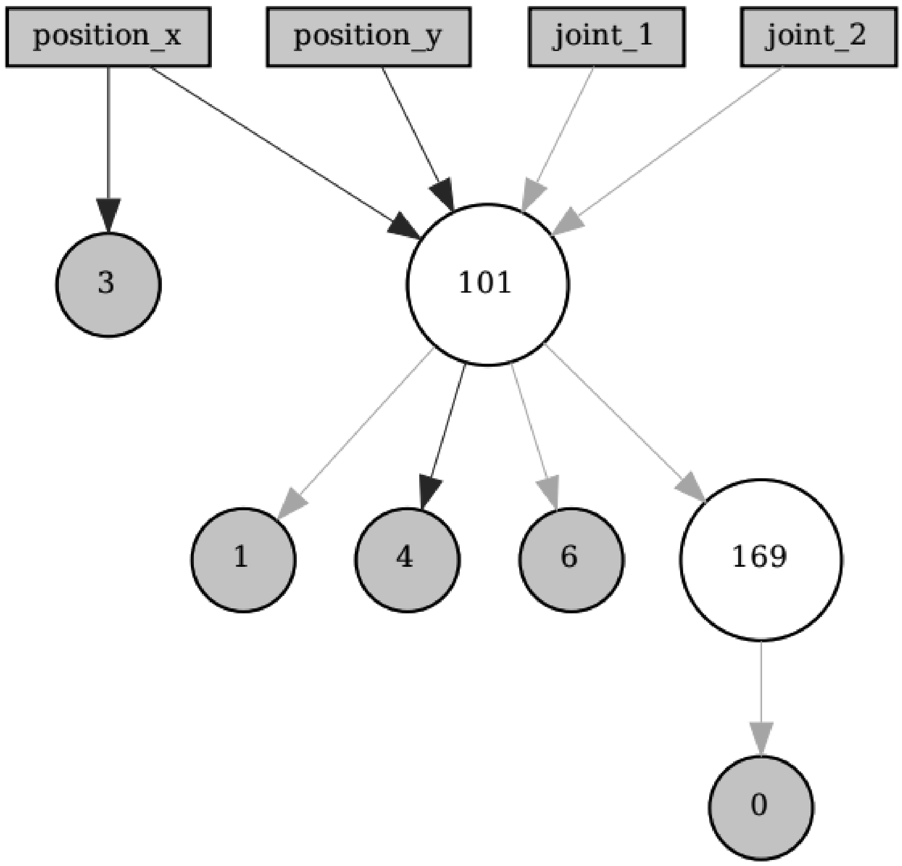

Рис. 2. Структура нейромережі-переможця

Висновки

Розробка програмних засобів для адаптивного навчання управляючих моделей роборуки на основі нейромереж вимагає комплексного підходу, що має враховувати варіативність умов середовища, максимально наближених до реального світу. Позиціонування робочої кінцівки має значний простір пошуку, що може бути також ускладнено різним представленням вхідних даних: як набору значень сенсорів, так і вхідним зображенням. Для умов довільного середовища мають бути використані контролери, що використовують знання, накопичені під час взаємодії з цим середовищем. Інтеграція таких методів штучного інтелекту, як машинне навчання, та навчання з підкріпленням, дала поштовх розвитку програмних засобів для моделювання задач позиціонування рук і маніпулювання об’єктами.

В статті розглянуто основний інструментарій, який має бути представлено у програмному пакеті для ефективного дослідження нейроеволюційного підходу до навчання контролерів роборуки. А також представлено розроблений пакет, який забезпечує доступний інтерфейс для роботи з набором середовищ для виконання задач, які становлять основний інтерес у роботі кінцівки робота.

Залежно від типу вхідних даних та характеристик агента, пакет надає набір інструментів для налаштування як самого агента та його взаємодії з середовищем, так і пареметрів досліджуваного нейроеволюційного алгоритму.

Дослідження конфігурацій NEAT та HyperNEAT покращують здатність алгоритму знаходити інноваційні та ефективні рішення, сприяючи розвитку більш універсальних і потужних архітектур нейронних мереж.

Розробка оптимальних засобів управління із застосуванням нейроеволюційного підходу може забезпечити не лише високу точність позиціонування, а і мінімальну конфігурацію, що підвищує швидкодію системи. Майбутні дослідження з використанням представленого пакету для вирішення задач маніпулювання та позиціонування становлять особливий інтерес для покращення програмних контролерів управління роботизованими кінцівками в динамічних умовах реального світу.

Література

1. Antonin Raffin, et al. Stable-Baselines3: Reliable Reinforcement Learning Implementations, 2021
https://jmlr.org/papers/volume22/20-1364/20-1364.pdf
2. I. Omelianenko, Hands-On Neuroevolution with Python: Build high-performing artificial neural network architectures using neuroevolution-based algorithms, Packt Publishing, 2019
3. Kenneth O. Stanley and Risto Miikkulainen, Evolving Neural Networks Through Augmenting Topologies, Evolutionary Computation 10 (2): 99-127, 2002.
4. Застосування засобів нейроеволюції в технічних системах автоматизації керування / А.Ю. Дорошенко, І.З. Ашур // Проблеми програмування. — 2021. — № 1. — С. 16-25
5. Альона Вітюк, Анатолій Дорошенко. Програмний пакет для оцінки похибка калібрування стереокамери в системі комп'ютерного зору. Проблеми програмування, 2022. № 3-4. С. 469–477. https://doi.org/10.15407/pp2022.03-04.469
6. Neuroevolution of a Robot Arm Controller, 2004. URL: https://www.cs.utexas.edu/ftp/AI-Lab/tech-reports/UT-AI-TR-05-322.pdf
7. OpenAI Gym 2D Robot Arm Environment, URL: https://github.com/ekorudiawan/gym-robot-arm
8. Verbancsics, P., Stanley, K.O., 2010. Evolving static representations for task transfer. Journal of Machine Learning Research. Vol. 11. pp 1737-1769

References

1. Antonin Raffin, et al. Stable-Baselines3: Reliable Reinforcement Learning Implementations, 2021
https://jmlr.org/papers/volume22/20-1364/20-1364.pdf
2. I. Omelianenko, Hands-On Neuroevolution with Python: Build high-performing artificial neural network architectures using neuroevolution-based algorithms, Packt Publishing, 2019
3. Kenneth O. Stanley and Risto Miikkulainen, Evolving Neural Networks Through Augmenting Topologies, Evolutionary Computation 10 (2): 99-127, 2002.
4. Application of neuroevolution tools in technical control automation systems / A.Y. Doroshenko, I.Z. Ashur // Problems of programming. — 2021. — No. 1. — P. 16-25 (in Ukrainian)
5. A. Vitiuk, A. Doroshenko. A software package for estimating stereo camera calibration error in a computer vision system. Problems of programming, 2022. No. 3-4. P. 469–477. https://doi.org/10.15407/pp2022.03-04.469 (in Ukrainian)
6. Neuroevolution of a Robot Arm Controller, 2004. URL: https://www.cs.utexas.edu/ftp/AI-Lab/tech-reports/UT-AI-TR-05-322.pdf
7. OpenAI Gym 2D Robot Arm Environment, URL: https://github.com/ekorudiawan/gym-robot-arm
8. Verbancsics, P., Stanley, K.O., 2010. Evolving static representations for task transfer. Journal of Machine Learning Research. Vol. 11. pp 1737-1769

Одержано: 28.11.2023

Про авторів:

Вітюк Альона Євгеніївна,
аспірантка НТУУ "КПІ імені Ігоря Сікорського".
Кількість наукових публікацій в українських виданнях – 4.

https://orcid.org/0000-0002-1445-9598

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук, професор, завідувач відділу теорії комп'ютерних обчислень Інституту програмних систем НАН Укрїни,
професор кафедри автоматики і управління в технічних системах НТУУ "КПІ імені Ігоря Сікорського".
Кількість наукових публікацій в українських виданнях – понад 200.
Кількість наукових публікацій в іноземних виданнях – понад 80.
Індекс Гірша – 7.
http://orcid.org/0000-0002-8435-1451,

Місце роботи авторів:
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»,
проспект Перемоги 37.
Інститут програмних систем НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
e-mail: alyonavityuk@gmail.com,
doroshenkoanatoliy2@gmail.com