Реализация контроллера управления реальным физическим объектом с применением методов нейроэволюционного алгоритма NEAT

А.А. Абузяров, А.А. Макаров

Российский государственный университет имени А.Н. Косыгина (Технологии. Дизайн. Искусство)

Аннотация: В данном эксперименте реализуется решатель (NEAT) и симулятор (объект тележки с обратным маятником), где решатель будет оказывать влияние на объект с целью удержать его в стабильном состоянии, т.е. не дать маятнику упасть. Основной задачей эксперимента является исследование возможности реализации симулятора реального физического объекта и использование его для определения целевой функции нейроэволюционного алгоритма NEAT. Решение данной задачи позволит реализовывать контроллеры на основе алгоритма NEAT, способные управлять реальными физическими объектами.

Ключевые слова: машинное обучение, нереволюционные алгоритм, генетические алгоритмы, нейронные сети.

Генетический алгоритм — это класс эволюционных алгоритмов поиска. Идея генетических алгоритмов основана на эволюционной теории Чарльза Дарвина. Этот алгоритм симулирует процесс естественного отбора, когда более сильные особи из популяции переживают более слабых и производят следующее поколение особей [1]. Данный подход можно использовать и для поиска наилучшей структуры нейросети – нейроэволюция. Основная идея нейроэволюции состоит в том, чтобы создавать нейросеть с помощью стохастических методов поиска, основанных на популяции. Используя эволюционный подход, можно разработать оптимальные архитектуры нейронных сетей, которые точно решают конкретные задачи. В результате могут быть созданы компактные и энергоэффективные сети с умеренными требованиями к вычислительной мощности.

Решатель - Неэйроэволюционный алгоритм (NEAT)

NEAT - Neuroevolution of augmenting topologies (Алгоритм нейроэволюции нарастающей топологии) предназначен для уменьшения размерности пространства поиска параметров посредством постепенного

развития структуры нейросети в процессе эволюции [2]. Основное преимущество метода - построенная сеть не нуждается в обучении обратным распространением ошибки, что уменьшает вычислительные затраты. Кроме того, эволюционно развиваемая топология сети позволяет построить структуру с очень высокой ёмкостью информации.

Симулятор – тележка с обратным маятником

Обратный маятник представляет собой неустойчивую механическую систему, центр масс которой находится выше точки вращения. Его можно стабилизировать, прикладывая внешние силы под управлением специализированной системы, которая контролирует угол маятника и перемещает точку вращения горизонтально назад и вперед под центром масс, когда маятник начинает падать [3].

Эксперимент предусматривает моделирование обратного маятника, реализованного в виде тележки, которая может перемещаться горизонтально, с установленными на ней сверху шарниром и маятником в виде вертикального стержня.

Для обучения контроллера балансировки маятника алгоритм обучения NEAT должен получать из среды минимальное количество знаний о объекте [4]. Знания должны максимально отражать близость контроллера к желаемой цели. В каждый данный момент времени решатель получает масштабированные значения:

- горизонтальное положения тележки - x1;
- скорость тележки - x2;
- угол наклона стержня – x3;
- скорость стержня – x4.

Эти значения служат входными данными для нейросети.

Взаимодействие решателя и симулятора

Задача состоит в том, чтобы стабилизировать неустойчивую систему и удерживать контроль над ней как можно дольше [5]. Контроллер балансировки маятника принимает масштабированные входные значения, описанные выше, и выдает выходной сигнал, являющийся двоичным значением, определяющим действие, которое должно быть применено. Ниже приведена начальная структура нейронной сети контроллера балансировщика (решателя).

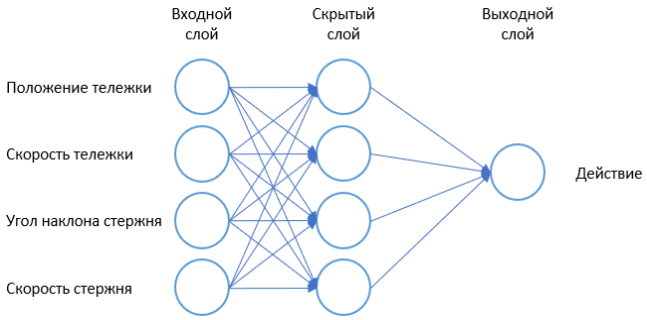

Рис. 1. - Архитектура нейронной сети

Скрытый слой является необязательным и при необходимости может быть исключен. Выходной слой является битовым узлом, выдающим значения 0 или 1.

Задача состоит в том, чтобы реализовать контроллер балансировки обратного маятника, который сможет поддерживать систему в стабильном состоянии в рамках определенных ограничений как можно дольше, или в течение ожидаемого количества временных шагов, указанных в конфигурации эксперимента.

Моделирование симулятора

Моделирование симулятора производится на языке программирования Python. Также используются следующие библиотеки:

- neat-python – реализация алгоритма NEAT на Python [6].

- PyTorch - это оптимизированная тензорная библиотека для глубокого обучения с использованием графических и центральных процессоров [7].
- NumPy - фундаментальный пакет для научных вычислений на Python [8].
- Matplotlib – это комплексная библиотека для создания статических, анимированных и интерактивных визуализаций на Python [9].

Тележка с обратным маятником:

- Для моделирования симулятора используются статичные значения – константы, указанные ниже:
    1. GRAVITY = 9.8 # ускорение свободного падения m/s^2
    2. MASSCART = 1.0 # масса тележки kg
    3. MASSPOLE = 0.5 # масса маятника kg
    4. TOTAL_MASS = (MASSPOLE + MASSCART) # общая масса
    5. LENGTH = 0.5 # Расстояние от центра масс маятника до шарнира m
    6. POLEMASS_LENGTH = (MASSPOLE * LENGTH) # kg * m
    7. FORCE_MAG = 10.0 # усилие N
    8. FOURTHIRDS = 4.0/3.0 # константа уравнения балансирования
    9. TAU = 0.02 # Количество секунд между обновлениями состояния sec
- Фрагмент кода уравнения движения симулятора:
    1. force = -FORCE_MAG if action <= 0 else FORCE_MAG
    2. cos_theta = math.cos(theta)
    3. sin_theta = math.sin(theta)

4. temp = (force + POLEMASS_LENGTH * theta_dot * theta_dot * sin_theta) / TOTAL_MASS
5. theta_acc = (GRAVITY * sin_theta - cos_theta * temp) / (LENGTH * (FOURTHIRDS - MASSPOLE * cos_theta * cos_theta / TOTAL_MASS)) # Угловое ускорение маятника.
6. x_acc = temp - POLEMASS_LENGTH * theta_acc * cos_theta / TOTAL_MASS # # Линейное ускорение тележки.
7. # Обновление четырех переменных состояния по методу Эйлера.
8. x_ret = x + TAU * x_dot
9. x_dot_ret = x_dot + TAU * x_acc
10. theta_ret = theta + TAU * theta_dot
11. theta_dot_ret = theta_dot + TAU * theta_acc

Пример кода, представленный выше в качестве входных данных, использует текущее состояние системы (x, x_dot, theta, theta_dot).

После описания констант и уравнения движения можно реализовать полный цикл моделирования симулятора и активации нейронной сети решателя, который состоит из следующих шагов:

1. Инициализация переменных начального состояния (x, x_dot, theta, theta_dot) около нулевыми значениями;
2. Запуск цикла моделирования на N – количество шагов (задается при выполнении эксперименте);
3. Масштабирования входных значений цикла для передачи в качестве входных данных нейросети (в диапазоне [0,1]);

   input[0] = (x + 2.4) / 4.8

   input[1] = (x_dot + 1.5) / 3

   input[2] = (theta + 0.21) / .42

   input[3] = (theta_dot + 2.0) / 4.0

4. Активация нейронной сети и масштабирование выходного значения сети

   # Активация сети

   output = net.activate(input)

   # Определение действие на основе выходного значения

   action = 0 if output[0] < 0.5 else 1

5. После шага моделирования новые переменные состояния проверяются на соответствие ограничениям, находится ли система в допустимых границах.

   Если система вышла за ограничения, возвращается пройденное количество шагов моделирования, если система смогла продержать систему в течение заданного количества шагов (N) моделирования возвращается максимальное количество шагов.

Моделирование решателя

В работе «Применение и сравнение эволюционных алгоритмов в рамках задачи обучения с подкреплением для неустойчивых систем» мы сравнивали два эволюционных алгоритма для управления неустойчивой средой CartPole и пришли к выводу, что алгоритм NEAT более гибок и масштабируем, что позволяет управлять разными объектами [10].

В данной работе мы использовали ранее реализованные классы на языке программирования Python для проведения эксперимента в данной работе. Для более глубокого понимая в текущей работе приведены основные классы и переменные среды:

- fitness_criterion (max) - функция, вычисляющая критерий завершения из набора значений приспособленности всех геномов в популяции. Значения параметров – это имена стандартных агрегатных функций, таких как min, max и mean. Значения min и max используются для завершения процесса эволюции, если минимальная или максимальная

приспособленность популяции превышает заданный порог fitness_threshold. Когда значение задано как mean, в качестве критерия завершения используется средняя приспособленность популяции. В данном случае требуется максимизация оценки - max;

- fitness_threshold (1.0) - пороговое значение, которое сравнивается с приспособленностью, вычисленной с помощью функции fitness_criterion для проверки необходимости завершения эволюции. Для решения задачи критерий завершения равен 1.0;
- pop_size (150) – количество организмов в поколении;
- reset_on_extinction (False) – флаг определяет необходимость создания нового поколения, если все виды в текущем вымерли;
- num_hidden, num_inputs, num_outputs – количество скрытых, входных и выходных узлов в геномах исходной популяции. Num_hidden = 1, num_inputs = 4, num_outputs = 1;
- activation_options (sigmoid) – функция активации.

На основе библиотеки neat-python реализован класс NeatBase, который реализует следующие методы:

- _run – метод предназначен для инициализации и запуска алгоритма. На данном этапе инициализируется алгоритм с основными параметрами из файла config-neat, инициализирует виртуальное окружения для взаимодействия с симулятором тележки и вызывается метод _run_neat;
- _run_neat – метод предназначен для создания первичной популяции и запуска сопутствующих методов для сохранения статистики _stat_reports, создания нейронной сети (эволюционирующей топологией и весами сети), запуск метода _test_genome тестирования сети на объекте CartPole;
- _visualize – метод для визуализации лучшего решения;
- _stat_reports – метод сбора статистики работы алгоритма;

• _test_genome – метод оценки работы генома. Метод взаимодействует с симулятором тележки, совершая действие и получая вознаграждение, на основе которого строится приспособленность.

Проведение эксперимента

При достижении максимальной пороговой приспособленности в 1.0, т.е. выполнения эксперимента в течение 500 000 временных, или выход за пороговые значения балансирования маятника эксперимент прекращается.

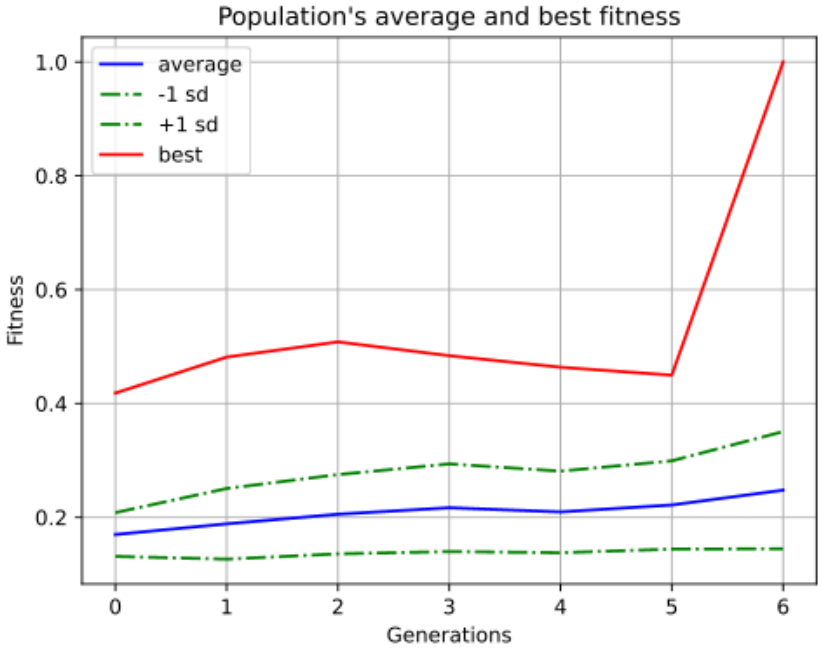

Рис. 2. – Приспособленность популяции.

По рисунку 2 видно, что алгоритм справляется с управлением симулятором. Видно, что на 5 поколении (красная линия) произошла мутация, что позволило найти наилучшее решение.

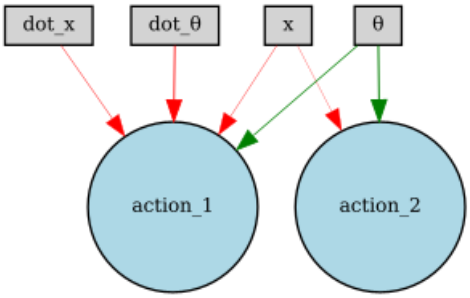

Рис. 3. – Стратегия управления объектом.

На рисунке 3 показана стратегия управления для лучшего поколения, влияние входных параметров на действие (action_1 = 0, action_2 = 1).

Из эксперимента видно, что алгоритм Neat способен реализовать контроллер, который в состоянии управлять реальными физическими объектами, алгоритм может выбирать различные стратегии управления и подстраиваться под объект.

Литература

1. Luo Xiong, Qian Qian, Fa Fu Yun. Improved Genetic Algorithm for Solving Flexible Job Shop Scheduling Problem, Knowledge Based Systems URL: sciencedirect.com/science/article/pii/S1877050920301836.
2. Lang Sebastian, Reggelin Tobias, Schmidt Johann. NeuroEvolution of augmenting topologies for solving a two-stage hybrid flow shop scheduling problem: A comparison of different solution strategies, Knowledge Based Systems URL: sciencedirect.com/science/article/pii/S095741742100107X.
3. Boriero Fabrizio, Sansonetto Nicola, Marigonda Antonio. Optimal Solution of Kinodynamic Motion Planning for the Cart-Pole System, Knowledge Based Systems URL: sciencedirect.com/science/article/pii/S2405896317313617.
4. Kapoor Arpit, Nukala Eshwar, Chandra Rohitash. Bayesian neuroevolution using distributed swarm optimization and tempered MCMC, Knowledge Based Systems URL: sciencedirect.com/science/article/abs/pii/S1568494622006056.
5. Brockman Greg, Terry Jordan. GYM Documentation URL: gymlibrary.dev/environments/classic_control/cart_pole/
6. Smith Allen W., Korinsky Kirill A. NEAT-Python Documentation URL: neat-python.readthedocs.io/en/latest/

7. Smith Allen W., Korinsky Kirill A. NEAT-Python Documentation URL: neat-python.readthedocs.io/en/latest/
8. Picus Matti, Berg Sebastian. NumPy Documentation URL: numpy.org/doc/stable/
9. Hunter John D. Matplotlib Documentation URL: matplotlib.org/stable/users/index
10. Абузяров А.А., Макаров А.А. Применение и сравнение эволюционных алгоритмов в рамках задачи обучения с подкреплением для неустойчивых систем // Инженерный вестник Дона. 2023. №6. URL: ivdon.ru/ru/magazine/archive/n6y2023/8477.

References

1. Luo Xiong, Qian Qian, Fa Fu Yun. Improved Genetic Algorithm for Solving Flexible Job Shop Scheduling Problem, Knowledge Based Systems. URL: sciencedirect.com/science/article/pii/S1877050920301836.
2. Lang Sebastian, Reggelin Tobias, Schmidt Johann. NeuroEvolution of augmenting topologies for solving a two-stage hybrid flow shop scheduling problem: A comparison of different solution strategies, Knowledge Based Systems. URL: sciencedirect.com/science/article/pii/S095741742100107X.
3. Boriero Fabrizio, Sansonetto Nicola, Marigonda Antonio. Optimal Solution of Kinodynamic Motion Planning for the Cart-Pole System, Knowledge Based Systems. URL: sciencedirect.com/science/article/pii/S2405896317313617.
4. Kapoor Arpit, Nukala Eshwar, Chandra Rohitash. Bayesian neuroevolution using distributed swarm optimization and tempered

MCMC, Knowledge Based Systems. URL: sciencedirect.com/science/article/abs/pii/S1568494622006056.

5. Brockman Greg, Terry Jordan. GYM Documentation. URL: gymlibrary.dev/environments/classic_control/cart_pole/
6. Smith Allen W., Korinsky Kirill A. NEAT-Python Documentation. URL: neat-python.readthedocs.io/en/latest/
7. Smith Allen W., Korinsky Kirill A. NEAT-Python Documentation. URL: neat-python.readthedocs.io/en/latest/
8. Picus Matti, Berg Sebastian. NumPy Documentation. URL: numpy.org/doc/stable/
9. Hunter John D. Matplotlib Documentation. URL: matplotlib.org/stable/users/index
10. Abuzyarov A.A., Makarov A.A. Inzhenernyj vestnik Dona. 2023. №6. URL: ivdon.ru/ru/magazine/archive/n6y2023/8477.