

Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика, искусственный интеллект и системы управления»
Кафедра «Системы обработки информации и управления»

Курс «Методы машинного обучения в АСОИУ»

Отчет по домашнему заданию
«Сравнение обучения с подкреплением и миварного подхода
для решения задач траекторного планирования»

Выполнил:

студент группы ИУ5-22М

Коценко А.А.

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Ю.Е.

Подпись и дата:

Москва, 2023 г.

Домашнее задание по дисциплине направлено на анализ современных методов машинного обучения и их применение для решения практических задач. Домашнее задание включает три основных этапа:

1. выбор задачи;
2. теоретический этап;
3. практический этап.

Этап выбора задачи предполагает анализ ресурса `paperswithcode` (<https://paperswithcode.com/sota>). Данный ресурс включает описание нескольких тысяч современных задач в области машинного обучения. Каждое описание задачи содержит ссылки на наиболее современные и актуальные научные статьи, предназначенные для решения задачи (список статей регулярно обновляется авторами ресурса).

Каждое описание статьи содержит ссылку на репозиторий с открытым исходным кодом, реализующим представленные в статье эксперименты. На этапе выбора задачи обучающийся выбирает одну из задач машинного обучения, описание которой содержит ссылки на статьи и репозитории с исходным кодом.

Теоретический этап включает проработку как минимум двух статей, относящихся к выбранной задаче. Результаты проработки обучающийся излагает в теоретической части отчета по домашнему заданию, включающей:

- описание общих подходов к решению задачи;
- конкретные топологии нейронных сетей, нейросетевых ансамблей или других моделей машинного обучения, предназначенных для решения задачи;
- математическое описание, алгоритмы функционирования, особенности обучения используемых для решения задачи нейронных сетей, нейросетевых ансамблей или других моделей машинного обучения;
- описание наборов данных, используемых для обучения моделей;
- оценка качества решения задачи, описание метрик качества и их значений;
- предложения обучающегося по улучшению качества решения задачи.

Практический этап включает повторение экспериментов авторов статей на основе представленных авторами репозитория с исходным кодом и возможное улучшение обучающимися полученных результатов. Результаты проработки обучающийся излагает в практической части отчета по домашнему заданию, которая может включать:

- исходные коды программ, представленные авторами статей, результаты документирования программ обучающимися с использованием диаграмм UML, путем визуализации топологий нейронных сетей и другими способами;
- результаты выполнения программ, вычисление значений для описанных в статьях метрик качества, выводы обучающегося о воспроизводимости экспериментов авторов статей и соответствии практических экспериментов теоретическим материалам статей;
- предложения обучающегося по возможным улучшениям решения задачи, результаты практических экспериментов (исходные коды, документация) по возможному улучшению решения задачи.

Отчет по домашнему заданию должен содержать:

1. титульный лист;
2. постановку выбранной задачи машинного обучения, соответствующую этапу выбора задачи;
3. теоретическую часть отчета;
4. практическую часть отчета;
5. выводы обучающегося по результатам выполнения теоретической и практической частей;
6. список использованных источников.

Этап выбора задачи

В процессе этапа выбора задачи были рассмотрены все разделы на сайте paperswithcode (<https://paperswithcode.com/sota>). Пройдемся по разделам:

- Robot Navigation
 - <https://paperswithcode.com/task/robot-navigation>
The fundamental objective of mobile Robot Navigation is to arrive at a goal position without collision. The mobile robot is supposed to be aware of obstacles and move freely in different working scenarios.
 - <https://paperswithcode.com/task/pointgoal-navigation>
 - <https://paperswithcode.com/task/social-navigation>
This task studies how to navigate robot(s) among humans in a safe and socially acceptable way.
- Content-Based Image Retrieval
 - <https://paperswithcode.com/task/drone-navigation>
(Satellite → Drone) Given one satellite-view image, the drone intends to find the most relevant place (drone-view images) that it has passed by. According to its flight history, the drone could be navigated back to the target place.
- Autonomous Vehicles
 - <https://paperswithcode.com/task/autonomous-navigation>
Autonomous navigation is the task of autonomously navigating a vehicle or robot to or around a location without human guidance.
- Logical Reasoning
 - <https://paperswithcode.com/task/navigate>

В рамках исследования решается задача сравнения обучения с подкреплением и миварного подхода для решения задач траекторного планирования. Для этих целей нам больше подходит раздел Robot Navigation с подразделами robot-navigation и social-navigation. В практическом этапе работы будут рассмотрены примеры реализации.

Теоретический этап

Обзор литературы. Учебник [1] – это основополагающий учебник по обучению с подкреплением, в нем рассмотрена история развития обучения с подкреплением и основные методы. В книге [2] подробно показаны теоретические основы глубокого обучения на большом количестве примеров. В работе [3] рассмотрено глубокое обучение с подкреплением на Python, в книге [4] – теория и практика глубокого обучения с подкреплением, а в учебном пособии [5] – мультиагентное обучение с подкреплением. В учебнике [6] рассмотрены алгоритмы обучения с подкреплением на Python. Сборник [7] рассказывает об обучении с подкреплением на PyTorch. Книга [8] посвящена моделированию нервных систем и созданию им подобных.

Основные концепции RL. Типичный алгоритм RL неформально [3]:

1. Агент взаимодействует со средой, выполняя действие.
2. Агент выполняет действие и переходит из одного состояния в другое.
3. Агент получает награду на основании выполненного действия.
4. В зависимости от награды агент понимает, было ли действие хорошим.
5. Если действие было хорошим, то есть если агент получил положительную награду, то он предпочитает выполнить это действие еще раз.
6. В противном случае агент пытается выполнить другое действие, приводящее к положительной награде. Таким образом, по сути, происходит процесс обучения методом проб и ошибок.

Агент (agent) – программный модуль, способный принимать осмысленные решения, играет роль обучаемого в RL. Агенты выполняют действия, контактируя со средой, и получают награды в зависимости от своих действий. Сумму наград, полученных агентом от среды, принято называть «возвратом». Политика (policy), называемая также «стратегией», определяет поведение агента в среде, направленное на достижение какой-либо цели. Способ выбора агентом действия, которое он будет выполнять, зависит от политики.

Допустим, агенту необходимо добраться из пункта А в пункт Б. Существует множество возможных вариантов маршрута; одни пути короткие, другие длинные. Эти пути называются «политиками», потому что они представляют собой способ выполнения действия для достижения цели. Политика часто обозначается символом и может быть реализована таблицей соответствия или сложным процессом поиска.

Модель (model) является представлением среды с точки зрения агента. Обучение делится на два типа: с моделью и без модели. В обучении с моделью агент эксплуатирует ранее усвоенную информацию для выполнения задачи, а при обучении без модели агент просто полагается на метод проб и ошибок для выполнения правильного действия.

Допустим, агенту необходимо добраться из пункта А в пункт Б. В обучении с моделью агент использует имеющуюся у него карту, а в обучении без модели агент методом проб и ошибок пробует разные пути и выбираете самый быстрый. Если в качестве модели используется глубокая нейронная сеть, то говорят о глубоком обучении с подкреплением.

Таксономия алгоритмов обучения с подкреплением (рисунок 1).

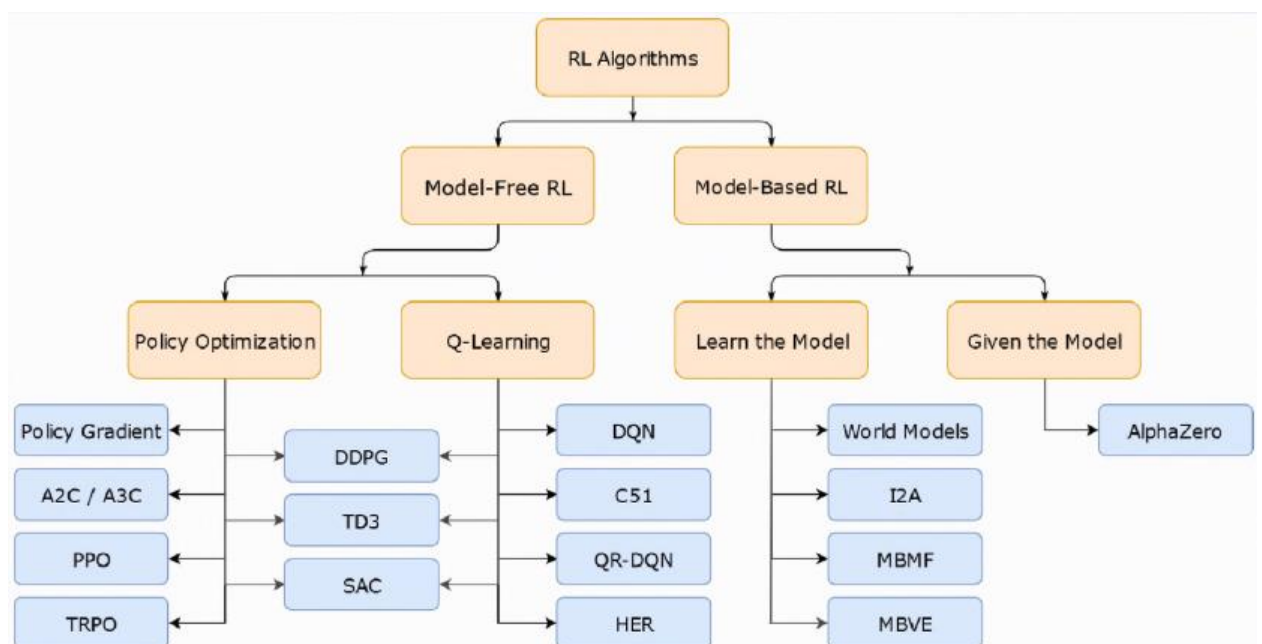


Рисунок 1 – Таксономия RL-алгоритмов

Model-Free RL – у агента нет модели среды. Model-Based RL – у агента есть модель среды, но создание исчерпывающей модели, как правило, невозможно. Model-Based RL предполагает использование техник автоматизированного планирования (рисунок 2). Given the Model означает, что модель полностью задана для агента. RL Algorithms Learn the Model означает, что агент изучает среду в процессе решения задачи.

Для чего используются глубокие методы в обучении с подкреплением? Представим, что у нас есть среда FrozenLake размером 31 600 x 31 600, что примерно соответствует миллиарду состояний. Чтобы рассчитать даже грубую аппроксимацию для каждого состояния этой среды, понадобятся сотни миллиардов переходов, равномерно распределенных по состояниям, что почти невозможно с вычислительной точки зрения.

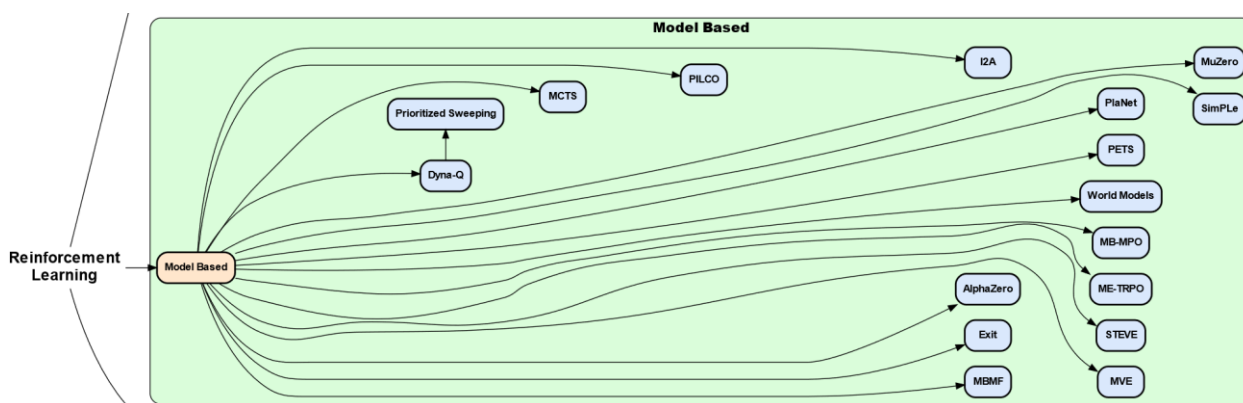


Рисунок 2 – Таксономия Model-Based RL

AlphaZero. Рассмотрим алгоритм AlphaZero, который является Model-Based RL алгоритмом и относится к подходу Given the Model, в котором модель полностью задается для среды. AlphaZero исторически был обобщением алгоритма AlphaGo для игры в Го, как общей процедуры обучения стратегии для произвольной игры с доступным симулятором. Для случая игры двух игроков вроде шахмат и Го, для которого она изначально и строилась, предполагается, что в симуляторе за противника играет просто недавняя версия текущей MCTS-стратегии; он также строился для детермированных сред с ненулевой наградой лишь в конце эпизода по типу «выиграл-проиграл».

Обучение с подкреплением на основе модели [6]

Алгоритмы обучения с подкреплением делятся на два больших класса: безмодельные и основанные на модели. Различаются они допущением относительно модели окружающей среды. Безмодельные алгоритмы обучаются стратегии просто на взаимодействиях с окружающей средой, о которой ничего не знают, тогда как основанные на модели алгоритмы обладают глубоким пониманием среды и используют эти знания для выбора следующего действия с учетом динамики модели.

Для начала вспомним, что такое модель. Модель состоит из динамики переходов и вознаграждений от среды. Динамика переходов – это отображение состояния s и действия a на следующее состояние s' . Располагая этой информацией, мы можем полностью заменить окружающую среду ее моделью. Агент, имеющий доступ к модели, может предсказать свое будущее.

Модель может быть известной или неизвестной. В первом случае модель используется сама по себе для изучения динамики среды, т.е. модель дает представление, используемое вместо среды. Во втором случае неизвестную модель можно обучить путем прямого взаимодействия со средой. Но поскольку в большинстве случаев мы получаем только приближение к окружающей среде, при использовании такой модели нужно учитывать дополнительные факторы.

Поняв, что такое модель, мы можем задуматься о том, как ее использовать и как она может помочь нам уменьшить количество взаимодействий со средой. Способ использования модели зависит от двух важных факторов: самой модели и способа выбора действий.

Действительно, как мы только что заметили, модель может быть известна или неизвестна, а действия можно планировать или выбирать с помощью обученной стратегии

Алгоритмы при этом существенно различны, поэтому сначала разберемся с подходами в случае, когда модель известна (т.е. мы уже знаем динамику переходов и вознаграждения от среды).

Известная модель [6]

Если модель известна, то ее можно использовать для имитации полных траекторий и вычисления дохода на каждой из них. Затем выбираются действия, приносящие наибольший доход. Этот подход называется планированием, без модели в нем никак не обойтись, потому что она дает информацию, необходимую для порождения следующего состояния (если известны текущее состояние и действие) и вознаграждения.

Алгоритмы планирования используются повсеместно, но те, что нас интересуют, зависят от типа пространства действий. Одни работают с дискретными действиями, другие – с непрерывными. Для дискретных действий обычно применяются алгоритмы поиска, которые строят решающее дерево на подобии показанного на рисунке 3.

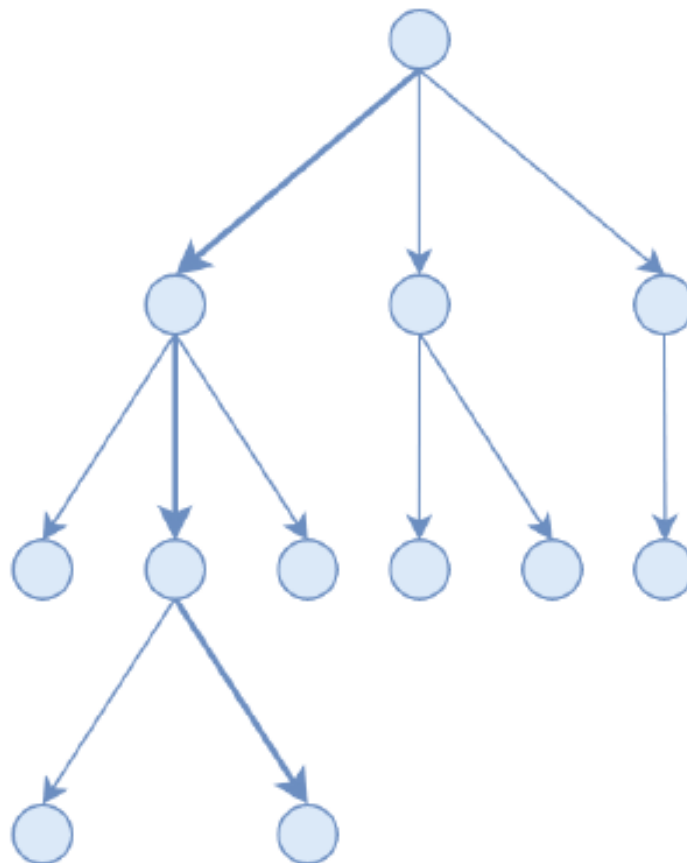


Рисунок 3 – Решающее дерево для дискретных действий

Текущее состояние является корнем, возможные действия представлены стрелками, а прочие узлы – это состояния, достижимые путем следования по стрелкам.

Легко видеть, что, опробовав все возможные последовательности действий, мы рано или поздно найдем оптимальную. К сожалению, в большинстве задач это неосуществимо, поскольку число действий растет экспоненциально. Алгоритмы планирования, применяемые в сложных задачах, подбирают стратегию, опираясь на ограниченное число траекторий.

Один из таких алгоритмов называется поиском по дереву методом Монте-Карло (ПДМК), он, кстати, применяется в программе AlphaGo. ПДМК итеративно строит решающее дерево, генерируя конечную последовательность имитированных игр, которой достаточно для исследования той части дерева, которая еще не посещалась. Когда имитированная траектория доходит до листового узла (т.е. игра заканчивается), результаты распространяются в обратном направлении через посещенные состояния, и при этом обновляется хранящаяся в узлах информация о выигрыше (проигрыше) или о вознаграждении. После этого выбирается действие с наибольшим отношением выигрыш/проигрыш или вознаграждением.

С другой стороны, алгоритмы планирования, работающие с непрерывными действиями, включают методы оптимизации траекторий. Они гораздо сложнее, чем алгоритмы с дискретными действиями, потому что должны решать задачу оптимизации в бесконечномерном пространстве. Кроме того, для многих из них нужен градиент модели. Примером может служить метод управления с прогнозирующими моделями (Model Predictive Control – MPC), который производит оптимизацию на конечном временном горизонте, но выполняет не всю найденную траекторию, а только первое действие на ней. Это позволяет MPC получать отклик быстрее, чем в других моделях с бесконечным горизонтом планирования.

Неизвестная модель [6]

Что делать, когда модель окружающей среды неизвестна? Обучить ее! Почти все, о чем мы говорили до сих пор, подразумевает обучение. Действительно ли это наилучшее решение? Если вы и вправду хотите

воспользоваться подходом на основе модели, то да, и вскоре мы увидим, как это сделать. Однако не всегда это самый предпочтительный путь. В обучении с подкреплением конечная цель – обучиться оптимальной стратегии решения данной задачи. Ранее мы сказали, что подход на основе модели применяется в основном, чтобы уменьшить количество взаимодействий с окружающей средой, но всегда ли это так? Единственный способ обучить модель (к сожалению) – взаимодействие со средой. От этого шага никуда не деться, потому что он позволяет собрать данные о среде.

Существует два основных способа обучить модель окружающей среды. В первом модель обучается один раз и остается неизменной, во втором модель обучается в начале, но переобучается всякий раз, как план или стратегия изменяются. Оба варианта показаны на Рисунке 4.

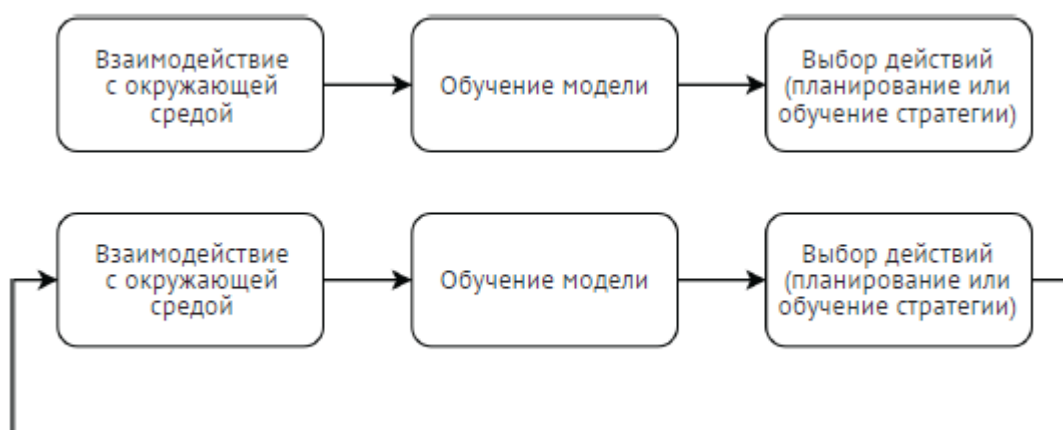


Рисунок 4 – Два способа обучить модель окружающей среды

В верхней части рис. 9.3 показан последовательный алгоритм на основе модели, когда агент взаимодействует с окружающей средой только до начала обучения модели. В нижней части мы видим циклический подход к обучению модели, когда модель уточняется с помощью дополнительных данных от другой стратегии.

На практике модель в большинстве случаев неизвестна и обучается с помощью агрегирования данных, чтобы адаптироваться к вновь выработанной стратегии. Однако обучить модель бывает трудно, на этом пути могут возникнуть следующие проблемы:

- переобучение модели: обученная модель слишком точно аппроксимирует локальную область, не улавливая глобальной структуры;
- неверная модель: если планировать действия или обучать стратегию на основе неверной модели, то может воспоследовать целый каскад ошибок с потенциально катастрофическими последствиями.

Хорошие алгоритмы на основе модели, которые обучают модель, должны справляться с этими проблемами. Например, можно использовать алгоритмы, оценивающие недостоверность, скажем байесовские нейронные сети. Можно также воспользоваться ансамблем моделей.

Достоинства и недостатки ОП на основе модели [6]

При разработке алгоритма обучения с подкреплением (любого вида) нужно принимать во внимание три свойства.

- Асимптотическое качество. Это наивысшее качество, которого может достичь алгоритм при наличии бесконечных ресурсов (времени и оборудования).
- Физическое время. Сколько времени алгоритм должен обучаться, чтобы достичь заданного качества при заданных вычислительных ресурсах.
- Выборочная эффективность. Количество взаимодействий с окружающей средой, необходимое для достижения заданного качества.

Мы уже изучали выборочную эффективность безмодельного и основанного на модели ОП и знаем, что последнее в этом смысле гораздо эффективнее. Но как насчет физического времени и качества? Обычно асимптотическое качество алгоритмов на основе модели ниже, а обучаются они медленнее, чем безмодельные алгоритмы. В общем случае повышение выборочной эффективности достигается ценой ухудшения качества и быстродействия.

Одна из причин низкого качества, основанного на модели обучения – неточность модели (если она была обучена), из-за чего в стратегии вносятся дополнительные ошибки. Большое физическое время может быть связано с медленным алгоритмом планирования или с большим количеством

взаимодействий, необходимым для обучения стратегии в неправильно смоделированной среде. Кроме того, у алгоритмов планирования на основе модели велико время логического вывода из-за высокой стоимости вычислений, которые тем не менее нужно производить на каждом шаге.

Итак, следует учитывать дополнительное время, требующееся для обучения алгоритма на основе модели, и смириться с низким асимптотическим качеством подобных подходов. Однако обучение на основе модели очень полезно, когда обучить модель проще, чем саму стратегию, и когда взаимодействие со средой медленное или дорогостоящее.

Оба подхода – безмодельное и основанное на модели обучение – имеют как явные преимущества, так и безусловные недостатки. Нельзя ли взять лучшее из обоих миров?

Сочетание безмодельного и основанного на модели обучения. Мы только что видели, что планирование может обойтись дорого с вычислительной точки зрения как на этапе обучения, так и на этапе выполнения и что в сложной окружающей среде алгоритмы планирования не в состоянии достичь хороших результатов. Но мы также намекнули на существование другого подхода – обучить стратегии. В этом случае логический вывод гораздо быстрее, потому что стратегии не нужно планировать на каждом шаге.

Простой, но эффективный способ обучения стратегии – объединить безмодельное и основанное на модели обучение. Учитывая недавние достижения в области безмодельных алгоритмов, эта комбинация завоевала популярность и на сегодня является самым распространенным подходом. Один из таких алгоритмов – ME-TRPO.

Выводы по ОП на основе модели в работе [6]

В этой главе автор исследовал алгоритмы, обучающиеся на модели окружающей среды. Мы рассмотрели основные причины смены парадигмы, побудившие разрабатывать такие алгоритмы. Мы выделили два случая работы с моделью: когда модель уже известна и когда ее предстоит сначала обучить.

Мы узнали, что модель можно использовать либо для планирования следующих действий, либо для обучения стратегии. Не существует четкого правила, диктующего, что выбрать, но, вообще говоря, это зависит от сложности пространств наблюдений и действий и от скорости логического вывода. Мы изучили достоинства и недостатки безмодельных алгоритмов и углубили понимание того, как обучить стратегию, сочетая безмодельные алгоритмы с обучением на основе модели. Это открыло новые возможности для применения моделей в пространствах наблюдений очень высокой размерности, например изображениях.

Наконец, чтобы лучше прочувствовать все сказанное о методах на основе модели, мы разработали алгоритм ME-TRPO. Он борется с недостоверностью модели, используя ансамбль моделей, а для обучения стратегии применяет оптимизацию стратегии в доверительной области. Все модели предсказывают следующие состояния, а затем участвуют в создании имитированных траекторий, на которых обучается стратегия. Таким образом, стратегия обучается только на обученной модели окружающей среды.

Алгоритмы глубокого обучения с подкреплением [4]

В RL агент настраивает функции, которые помогают ему действовать и максимизировать целевую функцию. Эта книга [4] посвящена глубокому обучению с подкреплением (глубокому RL), что означает, что в качестве аппроксимирующего семейства функций будут использоваться глубокие нейронные сети.

Есть три больших семейства алгоритмов глубокого обучения с подкреплением – методы, основанные на стратегии, методы, основанные на полезности, и методы, основанные на модели среды. Существуют также комбинированные методы, в которых агенты настраивают несколько этих функций, например функции стратегии и полезности или функцию полезности и модели среды. На рисунке 5 приведен обзор главных алгоритмов глубокого обучения с подкреплением в каждом из семейств и показаны взаимосвязи между ними.

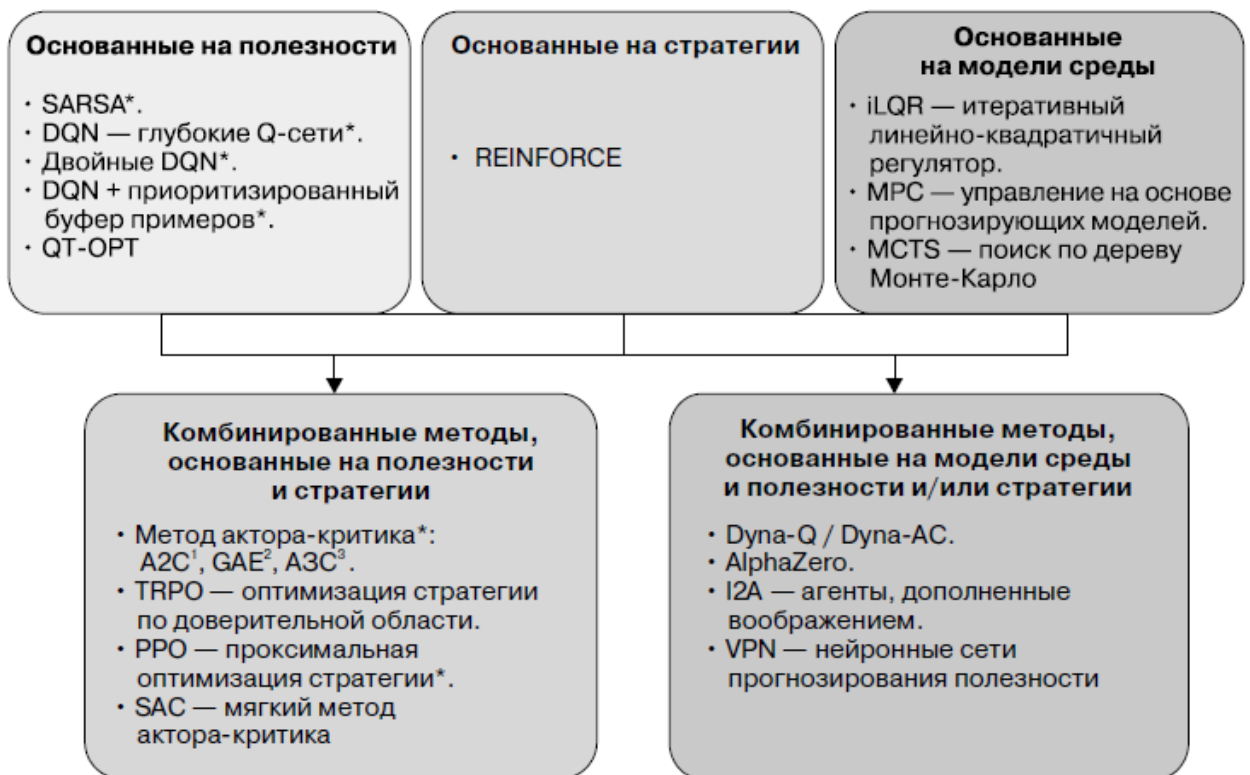


Рисунок 5 – Семейства алгоритмов глубокого обучения с подкреплением

В этой книге [4] упор сделан на методы, основанные на стратегии и полезности и их комбинациях. Приводится описание методов REINFORCE, SARSA, DQN и их расширений, актора-критика и PPO. Одна из глав посвящена методам параллелизации, которые могут применяться ко всем перечисленным алгоритмам.

Эта книга [4] призвана быть практическим руководством, поэтому в ней не затрагиваются алгоритмы, основанные на модели среды. Безмодельные методы значительно лучше исследованы и ввиду своей общности приложимы к более широкому классу задач. Один и тот же алгоритм (например, PPO) с минимальными изменениями может быть применен и к видеоиграм, таким как Dota 2, и к управлению робототехническим манипулятором. По сути, основанным на стратегии или полезности алгоритмам не нужна какая-то дополнительная информация. Их можно просто поместить в среду и позволить обучаться.

Основанным на модели среды методам, напротив, для работы обычно требуются дополнительные знания о среде, то есть модель динамики

переходов. Для таких задач, как шахматы или Го, модель – это просто правила игры, которые легко запрограммировать. И даже тогда заставить модель работать в связке с алгоритмами обучения с подкреплением – отнюдь не тривиальная задача, как видно по AlphaZero от DeepMind. Зачастую модель среды неизвестна и должна быть настроена, а это само по себе сложная задача.

Алгоритмы, основанные на модели среды [4]

Алгоритмы из этого семейства либо занимаются настройкой модели динамики переходов среды, либо задействуют известные динамические модели. Располагая моделью среды $P(s' | s, a)$, агент способен представить, что может случиться в будущем, прогнозируя траекторию на несколько шагов вперед. Пусть среда находится в состоянии s , тогда агент может оценить, насколько состояние изменится, если он предпримет последовательность действий $a_1, a_2 \dots a_n$, повторно применяя $P(s' | s, a)$. При этом он никак не изменяет среду, так что прогнозная траектория появляется у него в результате использования модели. Агент может спрогнозировать несколько разных траекторий с различными последовательностями действий i , оценив их, принять решение о наилучшем выборе действия a .

Исключительно модельный подход широко применяется к играм с известным целевым состоянием, таким как победа либо поражение в шахматах или навигационные задания с целевым состоянием s^* . Это связано с тем, что функции переходов в данном случае не моделируют никаких вознаграждений. И чтобы задействовать этот подход для планирования действий, потребуется закодировать информацию о целях агента в самих состояниях.

Поиск по дереву Монте-Карло (Monte Carlo Tree Search, MCTS) – широко распространенный метод, основанный на модели среды. Он применим к задачам с детерминированным дискретным пространством состояний с известными функциями переходов. Под эту категорию подпадают многие настольные игры, такие как шахматы и Го, и до недавнего времени многие компьютерные программы для игры в Го были основаны на MCTS. В нем не применяется машинное обучение. Для изучения игровых состояний и расчета

их полезностей производятся случайные выборки последовательностей действий – случайные симуляции. Этот алгоритм претерпел несколько улучшений, но основная идея осталась неизменной.

Другие методы, такие как итеративный линейно-квадратичный регулятор (iterative Linear Quadratic Regulators, iLQR) или управление на основе прогнозирующих моделей (Model Predictive Control, MPC), включают изучение динамики переходов зачастую с сильно ограничивающими допущениями¹. Для изучения динамики агенту нужно действовать в среде, чтобы собрать примеры действительных переходов (s, a, r, s').

Основанные на модели среды алгоритмы весьма интересны тем, что точная модель наделяет агента возможностью предвидения. Он может проигрывать сценарии и понимать последствия своих действий, не производя реальных действий в среде. Это может быть существенным преимуществом, когда накопление опыта из среды очень затратно с точки зрения ресурсов и времени, например, в робототехнике. Кроме того, по сравнению с методами, основанными на стратегии или полезности, этому алгоритму требуется намного меньше примеров данных для обучения оптимальной стратегии. Это обусловлено тем, что наличие модели дает агенту возможность дополнять его реальный опыт воображаемым.

Тем не менее для большинства задач модели среды получить трудно. Многие среды являются стохастическими, и их динамика переходов неизвестна. В таких случаях необходимо настраивать модель. Этот подход все еще находится на ранней стадии разработки, и его реализация сопряжена с рядом проблем. Во-первых, моделирование среды с обширными пространствами состояний и действий может быть очень трудным. Эта задача может даже оказаться неразрешимой, особенно если переходы чрезвычайно сложные. Во-вторых, от моделей есть польза только тогда, когда они могут точно прогнозировать переходы в среде на много шагов вперед. В зависимости от точности модели ошибки прогнозирования могут суммироваться на каждом шаге и быстро расти, что делает модель ненадежной.

На данный момент нехватка хороших моделей – главное ограничение для применения основанного на модели подхода. Однако основанные на модели методы могут быть очень эффективными. Если они работают, то эффективность выборок для них на 1–2 порядка выше, чем у безмодельных методов.

Различие между методами, основанными на модели среды, и безмодельными методами применяется и в классификации алгоритмов обучения с подкреплением. Алгоритм, основанный на модели, – это любой алгоритм, в котором используется динамика переходов среды, как настраиваемая, так и известная заранее. Безмодельные алгоритмы не задействуют динамику переходов среды в явном виде.

Обучение с подкреплением на основе модели среды [3]

Модель (model) является представлением среды с точки зрения агента. Обучение делится на два типа: с моделью и без модели. В обучении с моделью агент эксплуатирует ранее усвоенную информацию для выполнения задачи, а при обучении без модели агент просто полагается на метод проб и ошибок для выполнения правильного действия. Предположим, вы хотите быстрее добираться из дома на работу. В обучении с моделью вы просто используете уже имеющуюся информацию (карту), а в обучении без модели вы не пользуетесь имеющейся информацией, пробуете разные пути и выбираете самый быстрый.

Подводя итог, в обучении с моделью агент использует предыдущий опыт, тогда как при обучении без модели предыдущий опыт отсутствует.

Методы, основанные на моделях среды: воображение [2]

Данная глава книги [2] представляет собой основанный на моделях подход к RL с использованием результатов последних исследований о воображении в RL. В этой главе мы кратко рассмотрим методы RL, основанные на моделях. Методы, основанные на моделях, позволяют уменьшить объем взаимодействий со средой путем создания модели среды и ее использования во время обучения.

Термин «без использования модели» означает, что в данном методе не создается модель среды или вознаграждения; в нем используется прямая связь между наблюдениями и действиями (или ценностями, которые относятся к действиям). Другими словами, агент берет текущие наблюдения среды, выполняет некоторые вычисления и в результате получает действие, которое ему следует предпринять. В противоположность этому подходу методы, основанные на моделях, стремятся предсказать, какими будут последующие наблюдение и/или вознаграждение. Исходя из этого прогноза, агент пытается выбрать наилучшее возможное действие, очень часто делая такие предположения многократно, чтобы заглянуть в будущее на как можно большее количество шагов.

У обоих классов методов есть сильные и слабые стороны, однако методы, основанные на моделях, в чистом виде обычно применяются в детерминированных средах, таких как настольные игры с четкими правилами. С другой стороны, в случае с методами без использования модели обучение происходит проще, поскольку построить хорошую модель сложной среды с разнообразными входными данными большой размерности трудно. Все методы, которые описываются в данной книге, относятся к категории без использования модели, поскольку в течение нескольких последних лет именно они были объектом наиболее активных исследований. Только недавно исследователи начали комбинировать преимущества обоих подходов (в качестве примера можно привести статью DeepMind о воображении у агентов).

Сравнение безмодельных методов и основанных на моделях [2]

Исторически благодаря своей эффективности использования наблюдений методы, основанные на моделях, были гораздо более популярны в таких областях, как робототехника и другие сферы промышленной автоматизации. Это обусловлено стоимостью оборудования и физическими ограничениями примеров, которые можно получить от реального робота. Современные роботы с большим количеством степеней свободы обычно

довольно дорогое удовольствие, поэтому в исследовательской среде большей популярностью пользуются игры, которые гораздо лучше масштабируются и требуют почти нулевых вложений. Тем не менее область развивается очень быстро, и, как знать, возможно, методы, основанные на моделях, снова окажутся в центре внимания. А пока начнем с самого начала, чтобы разобраться, в чем разница.

В названиях обоих классов «модель» означает модель среды, которая может иметь самое разное представление, в наиболее очевидном случае возвращая новое наблюдение и вознаграждение по действию. Все методы, которые мы видели до сих пор, не прилагали усилий для прогнозирования, понимания или моделирования среды. Нас интересовало лишь правильное поведение (с точки зрения окончательного вознаграждения), которое представлялось непосредственно (стратегия) или косвенно (ценность) с учетом наблюдения. Источником наблюдений и вознаграждений была сама среда, которая в некоторых случаях могла быть очень медленной и неэффективной.

В подходе, основанном на моделях, мы пытаемся выучить модель среды, чтобы уменьшить зависимость от реальной среды. Если у нас есть точная модель среды, наш агент может создать любое нужное ему количество траекторий, просто используя эту модель вместо выполнения действий в реальном мире.

Есть две причины для использования подхода, основанного на моделях, по сравнению с безмодельным.

- Первая и самая важная – это эффективность использования данных, вызванная меньшей зависимостью от реальной среды. В идеале, имея точную модель, мы можем избежать взаимодействия с реальным миром и использовать только обученную модель. В реальных приложениях почти невозможно получить точную модель среды, но даже несовершенная модель может значительно сократить количество необходимых данных для обучения.

- Вторая причина применения подхода, основанного на моделях, заключается в переносимости модели среды между целями. Если у вас есть хорошая модель для робота-манипулятора, можете использовать ее для самых разных целей, не переучивая полностью.

Недостатки моделей [2]

У подхода, основанного на моделях, есть серьезная проблема: когда модель допускает ошибки или просто неточна в некоторых режимах среды, стратегия, выученная на основании этой модели, может быть абсолютно неверной в реальных ситуациях. Чтобы справиться с этим, есть несколько вариантов. Наиболее очевидный – сделать модель лучше. К сожалению, это может означать, что потребуется больше наблюдений со стороны среды, чего мы старались избежать. Чем более сложным и нелинейным является поведение среды, тем труднее выполнить ее корректное моделирование.

Было найдено несколько способов решения этой проблемы, например семейство методов локальных моделей, когда одна большая модель среды заменяется ансамблем простых линейных моделей, которые обучаются в своих режимах таким же образом, как в оптимизации стратегии по доверительной области (TRPO). Еще один интересный способ взглянуть на модели среды заключается в том, чтобы дополнить безмодельную стратегию путями, основанными на моделях. В этом случае мы не пытаемся создать наилучшую возможную модель среды, а просто даем агенту дополнительную информацию и позволяем ему самостоятельно решить, будет она полезна во время обучения или нет.

AlphaGo Zero [2]

В последней главе книги [2] продолжим обсуждать методы, основанные на моделях, и рассмотрим случаи, когда у нас есть модель среды, но эта среда используется двумя конкурирующими сторонами. Такая ситуация очень характерна для настольных игр, где правила фиксированы и каждому игроку доступна вся информация, но есть соперник, основная цель которого – помешать нам выиграть.

Недавно DeepMind предложил очень элегантный подход к таким проблемам, когда не требуется предварительного знания предметной области, а агент улучшает свою стратегию, только соревнуясь с самим собой. Этот метод называется AlphaGo Zero.

В целом метод состоит из трех компонентов:

- Мы постоянно перемещаемся по дереву игры, используя алгоритм поиска по дереву Монте-Карло (Monte-Carlo Tree Search, MCTS), основной смысл которого состоит в том, чтобы почти случайно выбирать игровые состояния, расширять их и собирать статистику о частоте ходов и основных результатах игры. Поскольку дерево игры почти всегда огромно как в глубину, так и в ширину, мы не пытаемся построить его полностью, а просто случайным образом выбираем в нем наиболее многообещающие пути.
- На каждом ходу у нас есть лучший игрок, который является моделью, используемой для генерации данных посредством самостоятельной игры. Первоначально эта модель имеет случайные веса, поэтому она делает ходы случайным образом, как четырехлетний ребенок, который только учится тому, как перемещаются шахматные фигуры. Однако со временем мы заменяем этого лучшего игрока его более совершенными версиями, которые будут генерировать все более значимые и сложные игровые сценарии. Самостоятельная игра означает, что на обеих сторонах доски применяется одна и та же лучшая модель.
- Третьим компонентом метода является процесс обучения ученической модели (apprenticeship model) на данных, собранных лучшей моделью на протяжении самостоятельной игры. Периодически мы проводим несколько матчей между обучаемой и лучшей действующей моделями. Если ученик сможет побить лучшую модель в большом количестве игр, обученная модель объявляется новой лучшей и процесс продолжается.

Несмотря на свою простоту и незамысловатость, AlphaGo Zero смог превзойти все предыдущие версии AlphaGo и стал лучшим игроком в Го в мире, не зная предварительно ничего, кроме правил игры.

Безмодельные системы и системы, основанные на модели [1]

Один из элементов некоторых систем обучения с подкреплением – модель окружающей среды. Это то, что имитирует поведение окружающей среды или, более общими словами, то, что позволяет делать выводы о том, как поведет себя среда. Например, зная состояние и действие, модель могла бы предсказать следующее состояние и следующее вознаграждение. Модели используются для планирования, под которым мы понимаем любой способ выбора порядка действий путем рассмотрения возможных будущих ситуаций, до того, как они фактически произошли.

Методы решения задач обучения с подкреплением, в которых используются модели и планирование, называются основанными на модели, в отличие от более простых безмодельных методов, в которых обучаемый явно действует методом проб и ошибок, считающимся чуть ли не полной противоположностью планированию. Далее автор книги [1] будет рассматривать системы обучения с подкреплением, которые одновременно обучаются методом проб и ошибок, а в результате обучения строят модель окружающей среды и используют эту модель для планирования. Современное обучение с подкреплением охватывает весь спектр систем – от низкоуровневого обучения методом проб и ошибок до высокоуровневого обоснованного планирования.

Безмодельные системы даже помыслить не могут о том, как изменится среда в ответ на одиночное действие. В этом смысле игрок в крестики-нолики является безмодельной системой по отношению к противнику: у него нет никакой модели противника. Поскольку модель полезна, только если достаточно верна, то безмодельные методы могут иметь преимущество над более сложными в случаях, когда трудность решения задачи связана именно со сложностью построения достаточно точной модели окружающей среды. Кроме того, безмодельные методы часто являются важными структурными элементами методов, основанных на модели.

Под моделью окружающей среды мы понимаем все, что агент может использовать для предсказания реакции среды на свои действия в терминах переходов состояний и вознаграждений, а под планированием имеется в виду любой процесс, который вычисляет стратегию по такой модели.

Модель окружающей среды состоит из двух частей: переходы состояний кодируют знания о влиянии действий на изменение состояний, а модель вознаграждения – знания о сигналах вознаграждения, ожидаемых для каждого состояния или каждой пары состояние-действие. Основанный на модели алгоритм выбирает действия, используя модель для предсказания последствий возможного образа действий в терминах будущих состояний и сигналов вознаграждения, ожидаемых в этих состояниях. Простейший вид планирования – сравнить предсказанные последствия групп различных «воображаемых» последовательностей решений.

Модель окружающей среды и планирование [1]

Под моделью окружающей среды мы понимаем всё, что агент может использовать для предсказания реакции среды на свои действия. Зная состояние и действие, модель порождает предсказание следующего состояния и вознаграждения. Если модель стохастическая, то существует несколько возможных следующих состояний и вознаграждений, с каждым из которых связана вероятность. Некоторые модели порождают описание всех возможностей и их вероятностей; они называются моделями распределения. Другие модели порождают только одну из возможностей, выбранную согласно вероятности; они называются моделями выборки.

Модели распределения более универсальны, чем модели выборки, поскольку их всегда можно использовать для порождения выборки. Но во многих приложениях гораздо проще получить модель выборки, чем модель распределения. Бросание нескольких костей как раз из числа таких приложений. Легко написать программу, которая моделирует бросок костей и возвращает выпавшую сумму, а вот перечислить все возможные суммы и их вероятности труднее и чревато ошибками.

Модели можно использовать для имитации опыта. Если известны начальное состояние и действие, то модель выборки порождает возможный переход, а модель распределения генерирует все возможные переходы, каждому из которых сопоставлен вес, определяемый его вероятностью. Если известны начальное состояние и стратегия, то модель выборки могла бы породить весь эпизод, а модель распределения – все возможные эпизоды и их вероятности. В любом случае мы говорим, что модель используется для имитации окружающей среды и порождения имитированного опыта.

Слово «планирование» имеет различный смысл в разных областях. В данной книге [1] мы понимаем под этим любой вычислительный процесс, который принимает на входе модель и порождает или улучшает стратегию взаимодействия с моделируемой средой (рисунок 6).

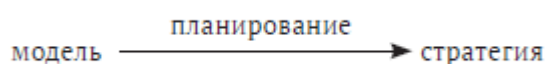


Рисунок 6 – Структура метода планирования

В области искусственного интеллекта есть два подхода к так определенному планированию. Планирование в пространстве состояний, включающее описанный в этой книге [1] подход, рассматривается в основном как поиск оптимальной стратегии или оптимального пути к цели в пространстве состояний. Действия вызывают переходы из одного состояния в другое, а функции ценности вычисляются для состояний.

Имеется также планирование в пространстве планов, когда поиск производится в пространстве планов. Операторы преобразуют один план в другой, а функции ценности, если таковые существуют, определяются в пространстве планов. Планирование в пространстве планов включает эволюционные методы и «планирование с частичным порядком» – часто встречающийся в искусственном интеллекте вид планирования, при котором порядок шагов не полностью определен на всех этапах планирования. Методы планирования в пространстве планов не подходят для эффективного применения к стохастическим задачам последовательного принятия решений.

Унифицированный подход, излагаемый в этой главе, заключается в том, что все методы планирования в пространстве состояний имеют одинаковую структуру, которая присутствует также в методах обучения, представленных в этой книге [1]. Основных идей всего две:

1. все методы планирования в пространстве состояний включают вычисление функций ценности в качестве ключевого промежуточного шага в направлении улучшения стратегии;
2. они вычисляют функции ценности с помощью операций обновления, или восстановления, применяемых к имитированному опыту.

Эта общая структура схематически представлена на Рисунке 7.

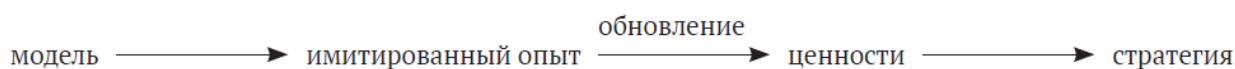


Рисунок 7 – Унифицированный подход к структуре метода планирования

Привычное и целеустремленное поведение [1]

Различие между алгоритмами обучения с подкреплением, основанными на модели и работающими без таковой, соответствует различию, которое психологи проводят между привычным и целеустремленным управлением, обученными паттернами поведения. Привычки – это паттерны поведения, запускаемые в ответ на подходящие стимулы и продолжающиеся более-менее автоматически. Целеустремленное поведение в терминологии психологов преднамеренно в том смысле, что управляется знаниями о ценности целей и связи между действиями и их последствиями.

Иногда говорят, что привычками управляют предшествующие стимулы, а целеустремленным поведением – его последствия. Целеустремленное управление обладает тем преимуществом, что может быстро изменить поведение животного, когда изменяется реакция окружающей среды на его действия. Хотя привычное поведение быстро реагирует на информацию, поступающую от окружающей среды, оно не способно быстро адаптироваться к изменениям среды. Выработка целеустремленного управления поведением, вероятно, стала главным достижением в эволюции интеллекта животных.

На Рисунке 8 иллюстрируется различие между основанными на модели и безмодельными стратегиями принятия решений:

- Сверху: крыса проходит лабиринт с двумя кормушками, с которыми ассоциировано вознаграждение указанной величины.
- Слева внизу: безмодельная стратегия опирается на сохраненные ценности действий для всех пар состояние-действие, полученные по результатам многих испытаний. Для принятия решения крыса просто выбирает в каждом состоянии действие с наибольшей ценностью.
- Справа внизу: в основанной на модели стратегии крыса обучается модели окружающей среды, состоящей из информации о переходах состояние действие-следующее состояние, и модели вознаграждения, состоящей из информации о вознаграждении, ассоциированном с каждой кормушкой. Крыса принимает решения, куда поворачивать в каждом состоянии, пользуясь моделью для имитации последовательного выбора действий с целью найти путь, приносящий максимальный доход.

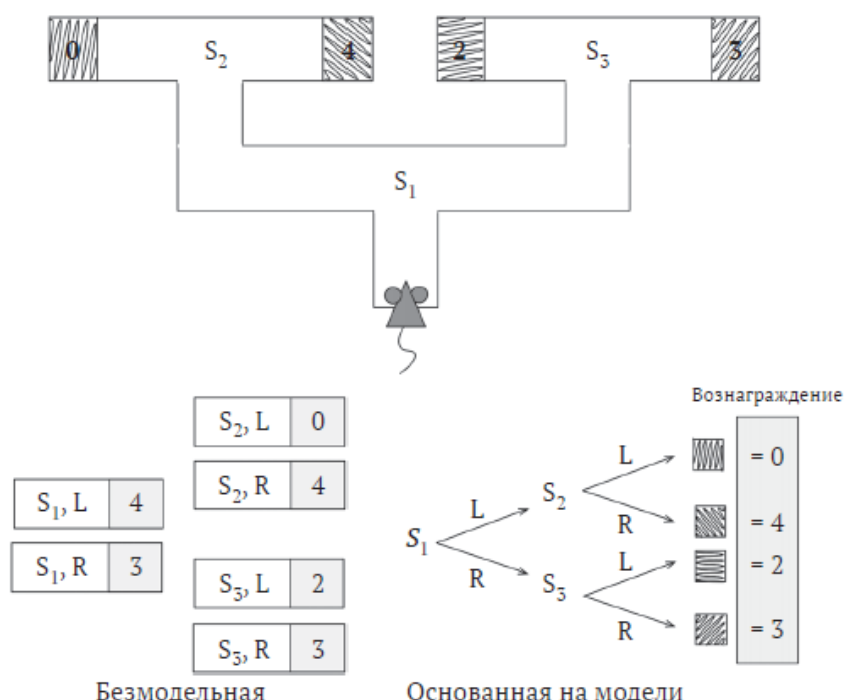


Рисунок 8 – Основанная на модели и безмодельная стратегии решения гипотетической задачи о последовательном выборе действий

Сравнение предсказанного дохода на имитированных путях – простая форма планирования, различные виды которого обсуждались ранее.

Ключевой момент заключается в том, что для того, чтобы безмодельный агент изменил действие, которое его стратегия предписывает в некотором состоянии, или изменил ценность действия, ассоциированного с состоянием, он должен перейти в это состояние, совершать действия из него, быть может много раз, и на опыте испытывать последствия своих действий.

Основанный на модели агент может адаптироваться к изменениям в окружающей среде, не прибегая к такого рода «личному опыту» исследования состояний и действий, на которые повлияло изменение. Изменение в его модели автоматически (посредством планирования) изменяет его стратегию.

На ранних этапах обучения более надежным считается процесс планирования, являющийся частью основанной на модели системы, поскольку он связывает воедино краткосрочные предсказания, которые могут оказаться точными при наличии меньшего опыта, чем в случае долгосрочных предсказаний безмодельного процесса. Но по мере продолжения обучения более надежным становится безмодельный процесс, потому что планирование может приводить к ошибкам из-за неточности модели и срезания углов, без которого планирование было бы неосуществимо, в частности из-за различных форм «обрезки ветвей»: удаления кажущихся бесперспективными ветвей дерева поиска. В соответствии с этой идеей следовало бы ожидать перехода от целеустремленного поведения к привычному по мере накопления опыта.

Различие между безмодельными и основанными на модели алгоритмами обучения с подкреплением соответствует различию между привычным и целеустремленным поведением в психологии. Безмодельные алгоритмы принимают решения с помощью доступа к информации, сохраненной в стратегии или функции ценности действий, тогда как основанные на модели методы выбирают действия в результате заблаговременного планирования с применением модели окружающей агента среды. Эксперименты по снижению ценности желаемого исхода дают информацию о том, является ли поведение животного привычным или целеустремленным. Теория обучения с подкреплением помогла прояснить представления об этих вопросах.

Основанные на модели методы в мозге [1]

Резюмируя ситуацию, Doll, Simon and Daw (2012) писали, что «основанные на модели влияния, похоже, присутствуют чуть ли не везде, где мозг обрабатывает информацию о вознаграждении». Сюда относятся сами дофаминовые сигналы, которые могут свидетельствовать о влиянии основанной на модели информации, помимо ошибок предсказания вознаграждения, которые, как полагают, лежат в основе безмодельных процессов. Вероятно также, что в формировании зависимости участвуют основанные на модели процессы.

Продолжающиеся нейронаучные исследования, в которых учитывается различие между безмодельным и основанным на модели обучением с подкреплением, потенциально способны улучшить наше понимание привычных и целеустремленных процессов в мозге. А это, в свою очередь, может привести к алгоритмам, объединяющим безмодельные и основанные на модели методы способами, еще неизвестными в вычислительном обучении с подкреплением.

Практический этап

Миварный подход. В настоящее время активно развиваются исследования в области управления роботами, например, для беспилотного транспорта. Системы технического зрения реализуются преимущественно с помощью нейросетевых технологий. Однако для принятия решений в сложных и нестандартных ситуациях следует применять логический искусственный интеллект. Для решения этой задачи целесообразно использовать миварные технологии логического искусственного интеллекта (ЛИИ), которые позволяют находить решение с линейной вычислительной сложностью для задач в форматах продукционных сетей «если – то» или описания бизнес-процессов в формате «вход; выход; действие».

В настоящее время, миварные технологии ЛИИ применяют в различных областях, включая: гетерогенные мультиагентные систем и группы роботов, для создания автоматизированных систем управления технологическими процессами, для тегирования изображений и интеллектуального распознавания образов, для поиска траекторий роботов и планирования их действий, для моделирования мышления водителя высокоавтоматизированных транспортных средств, для обучения людей в разных областях, управления образовательными программами в вузе, смысловой обработки потоков текстов, а также в медицине для диагностики сахарного диабета, автоматизации психодиагностики, определения безопасности применения компонентов крови и для многих других областей.

Низкие требования к вычислительным мощностям позволяют применять МЭС в автономной робототехнике. Ранее были предложены методика поиска нескольких траекторий движения робота в двумерном пространстве, а также автоматизированная методика последовательного удаления правил миварной сети и поиска новых траекторий движения робота для их дальнейшего сравнения по количеству активированных правил перехода на пути от изначального местоположения к целевому.

Эти методики могут применяться для решения задач оптимизации распределения ресурсов в машиностроительном искусственном интеллекте. Миварные технологии успешно применяют в робототехнике для создания систем принятия решений (СПР) автономных робототехнических комплексов (РТК).

Таким образом, на данный момент созданы все предпосылки для перехода к применению миварных экспертных систем для планирования трехмерных маршрутов роботов и робототехнических комплексов (РТК) с учетом различных видов препятствий. Итак, создание систем принятия решений (СПР), основанных на логике, является актуальной и важной задачей для робототехники.

Миварные системы принятия решений. В связи с низкими требованиями к вычислительным мощностям было принято решение создавать системы поддержки принятия решения (СППР) для роботов в автономном исполнении. Так как в таких роботах человек не принимает решения, было решено назвать эти системы не «поддержки принятия решения», а «системы принятия решения» (СПР). Последующие исследования подтвердили правильность выделения в явном виде на уровне управления новых систем, которые получили название «Системы принятия решений роботов» и были выполнены на основе миварных экспертных систем (МЭС).

За последние несколько лет на основе МЭС создано несколько рабочих прототипов и макетов, в которых вместе собраны: системы управления (СУ) движением различных роботов и их группировок; системы технического зрения (СТЗ); СПР для роботов, например, система «РобоРазум». Эта СПР создана на основе серверной версии «КЭСМИ (Разуматор)», к которой добавлены два блока для получения исходных данных от СТЗ и для передачи управляющих воздействий на СУ движением различных роботов и их группировок. В настоящее время решается проблема создания миварных моделей для СПР роботов.

Автоматическая генерация миварных баз знаний. В результате анализа было принято решение о создании программы для автоматической генерации XML файла модели в целях дальнейшего обеспечения трехмерного движения роботов. Реализовываться эта программа будет на языке программирования Python третьей версии. Для работы с XML будет использоваться библиотека «xml.etree.ElementTree». Библиотека «uuid» реализует создание универсальных уникальных идентификаторов. В начале создается заголовок XML файла с названием и описанием модели. Далее создаются отношения. В зависимости от веса отношения делятся на 3 типа. Первое описывает единичный переход, второе переход корень из двух, третье – корень из трех. Каждое отношение описывает входные вектора рассматриваемой вершины.

После этого создается класс, который включает в себя параметры, правила, ограничения и классы. Параметры – это вершины, между которыми двигается робот. Каждый параметр – это отдельное состояние в пространстве. Правила описывают переходы между параметрами, из какой точки в какую можно перейти, а также какое отношение относится к этому переходу. В зависимости отношения у перехода будет определенный вес. Ограничения и классы в данной работе не используются, поэтому в итоговом XML файле они остаются пустыми. Далее приведен фрагмент программного кода, демонстрирующий создание XML файла миварной модели (рисунок 9).

```
def ModelGen3D(NumX, NumY, NumZ):
    # Создание модели
    Model_XML = ET.Element('model', id='{'+str(uuid.uuid4())+'}',
                           shortName='Model',
                           formatXmlVersion='2.0',
                           description='Model')

    # Создание отношений
    Relations_XML, Rel_ID = Create_Relations_XML()
    Model_XML.append(Relations_XML)

    # Создание класса
    Class_XML = ET.Element('class', id='{'+str(uuid.uuid4())+'}',
                           shortName='Field')

    # Создание параметров
    Parameters_XML, ParamDict = Create_Parameters_XML(NumX, NumY, NumZ)
    Class_XML.append(Parameters_XML)

    # Создание правил
    Rules_XML, NumRules = Create_Rules_XML(NumX, NumY, NumZ, Rel_ID, ParamDict)
    Class_XML.append(Rules_XML)

    # Создание ограничений и классов
    Class_XML.append(ET.Element("constraints"))
    Class_XML.append(ET.Element("classes"))
    Model_XML.append(Class_XML)
```

Рисунок 9 – Функция автоматической генерации XML файла миварной модели на языке программирования Python

Ниже показан пример созданной XML модели, состоящей из отношений параметров и правил, для трехмерного логического пространства единичного размера (рисунок 10).

```
<?xml version='1.0' encoding='UTF-8'?>
<model id="{84f56a6d-1b0c-4c5d-a7f1-5de9ef83c62d}" shortName="Model" formatXmlVersion="2.0" description="Model">
  <relations>
    <relation id="8a799370-fc53-4afa-8c74-ea4e0fc3e6dd" shortName="Relation_1" inObj="x:double"
      relationType="simple" outObj="y:double">y=x+1</relation>
    <relation id="fd455649-1db8-493b-9089-c4af5f636b17" shortName="Relation_sqrt2" inObj="x:double"
      relationType="simple" outObj="y:double">y=x+1.4142</relation>
    <relation id="3e76cc44-75a4-44cc-8a37-c071e31939cf" shortName="Relation_sqrt3" inObj="x:double"
      relationType="simple" outObj="y:double">y=x+1.7321</relation>
  </relations>
  <class id="{4ad49495-65b5-411d-8334-55516e3a1e43}" shortName="Field">
    <parameters>
      <parameter id="4cc83ec2-99ae-49d6-bf0f-d9b5702a3013" shortName="(0,0,0)" type="double" />
      <parameter id="5dc0d363-0498-4208-a38b-675b17628119" shortName="(0,0,1)" type="double" />
      <parameter id="e1511e94-9412-41b3-b117-a781762db833" shortName="(0,1,0)" type="double" />
      <parameter id="e4386e8b-bb97-428b-9be5-fe691192e527" shortName="(0,1,1)" type="double" />
      <parameter id="651bbac5-8b0f-487f-9493-130f7de1d12e" shortName="(1,0,0)" type="double" />
      <parameter id="ba699a1a-b0c5-44c5-bcf8-0650acaba7c9" shortName="(1,0,1)" type="double" />
      <parameter id="2c136f8e-c107-43ae-9efc-291822781dca" shortName="(1,1,0)" type="double" />
      <parameter id="ad50de78-ff2a-48cf-bb47-ff47dad806b3" shortName="(1,1,1)" type="double" />
    </parameters>
    <rules>
      <rule id="c864c96a-f072-4ad7-acc9-845c7836b6ab" shortName="(0,0,0)→(1,0,0)"
        relation="8a799370-fc53-4afa-8c74-ea4e0fc3e6dd"
        initId="x:4cc83ec2-99ae-49d6-bf0f-d9b5702a3013"
        resultId="y:651bbac5-8b0f-487f-9493-130f7de1d12e" />
      <rule id="03f7216b-2da7-4a6a-97d6-ee2488ce4bb6" shortName="(0,0,0)→(0,1,0)"
        relation="8a799370-fc53-4afa-8c74-ea4e0fc3e6dd"
        initId="x:4cc83ec2-99ae-49d6-bf0f-d9b5702a3013"
        resultId="y:e1511e94-9412-41b3-b117-a781762db833" />
      <rule id="d50cc035-b880-4656-9497-a40a2cc234bd" shortName="(0,0,0)→(0,0,1)"
        relation="8a799370-fc53-4afa-8c74-ea4e0fc3e6dd"
        initId="x:4cc83ec2-99ae-49d6-bf0f-d9b5702a3013"
        resultId="y:5dc0d363-0498-4208-a38b-675b17628119" />
      ...
      <rule id="852a96e0-eeef-465e-9c05-ec91b37383e0" shortName="(1,1,1)→(1,0,0)"
        relation="fd455649-1db8-493b-9089-c4af5f636b17"
        initId="x:ad50de78-ff2a-48cf-bb47-ff47dad806b3"
        resultId="y:651bbac5-8b0f-487f-9493-130f7de1d12e" />
      <rule id="a645dd2e-add3-4667-b2b1-54b63fcc0b8d" shortName="(1,1,1)→(0,0,0)"
        relation="3e76cc44-75a4-44cc-8a37-c071e31939cf"
        initId="x:ad50de78-ff2a-48cf-bb47-ff47dad806b3"
        resultId="y:4cc83ec2-99ae-49d6-bf0f-d9b5702a3013" />
    </rules>
    <constraints />
    <classes />
  </class>
</model>
```

Рисунок 10 – Пример сгенерированного XML файла миварной модели

Проверка работы XML модели. Когда миварная модель создана, ее можно проверить в программном продукте для создания экспертных систем «КЭСМИ». КЭСМИ – это инструмент для создания моделей знаний с неограниченным количеством связей, параметров и отношений, обладающий логическим выводом. КЭСМИ может эффективно использоваться, как для

создания программных роботов (RPA) или виртуальных специалистов, так и в целях построения сложных экспертных систем, систем управления знаниями или логически решающих систем (Logical Reasoning Systems). В текущей задаче требуется задать начальную и конечную точки для дальнейшего построения графа решения в КЭСМИ для всех одиночных переходов в трехмерном логическом пространстве (рисунки 11-12).

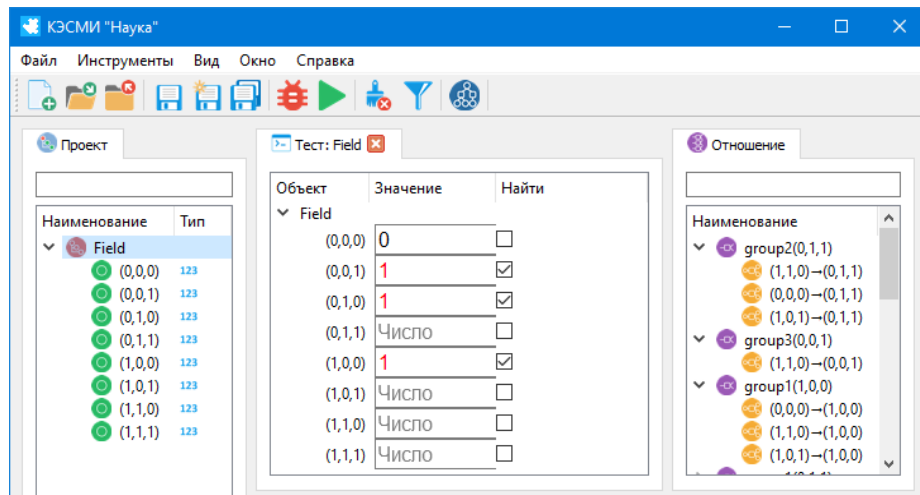


Рисунок 11 – Начальная и конечные точки в КЭСМИ для одиночных переходов

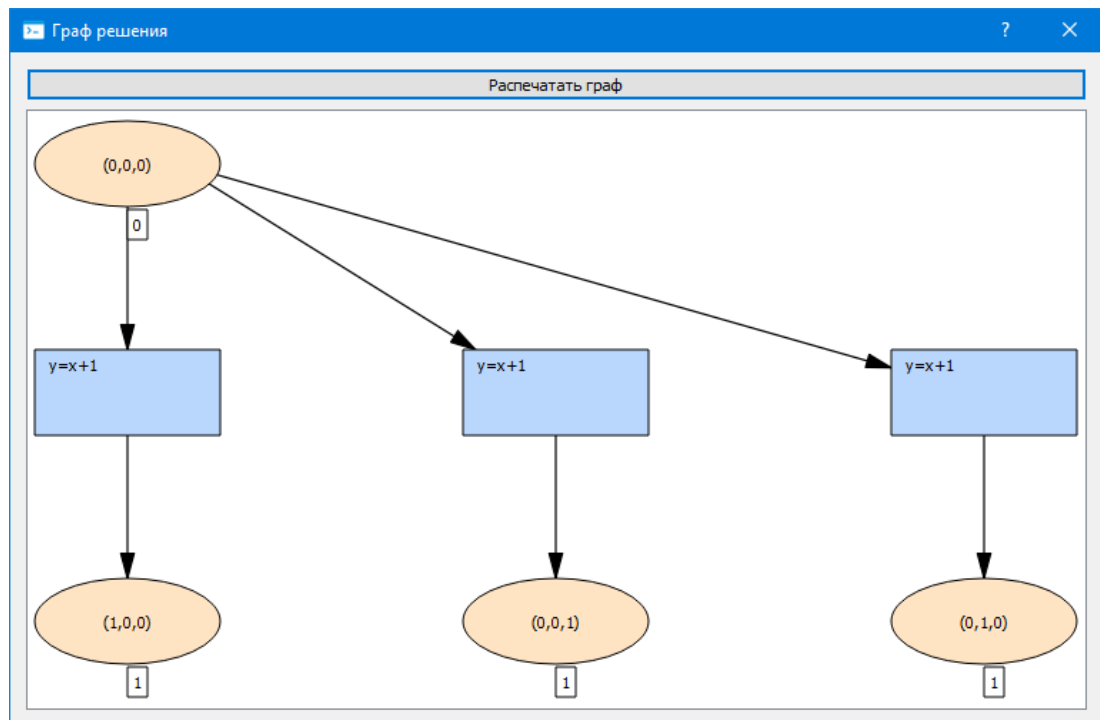


Рисунок 12 – Граф решения для одиночных переходов в логическом пространстве

После, для демонстрации переходов с разными весами в трехмерном логическом пространстве, строится граф для всех переходов с весами корень из 2 и корень из 3 (рисунки 13-14).

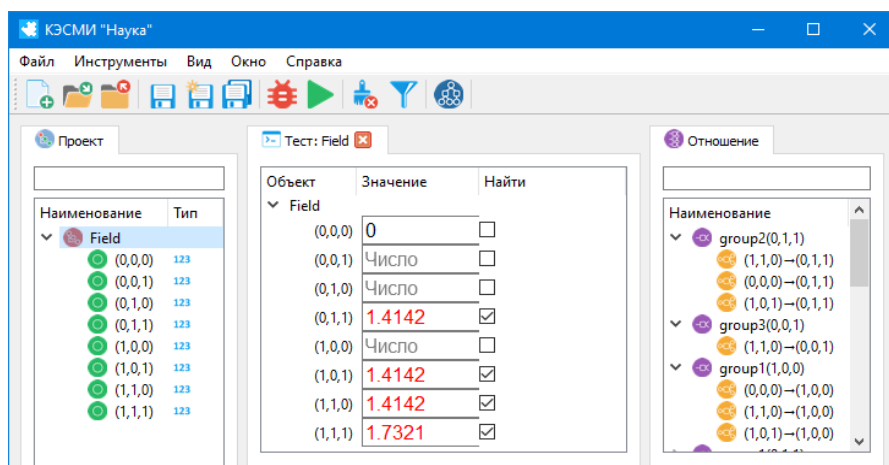


Рисунок 13 – Начальная и конечные точки в КЭСМИ для переходов с весом

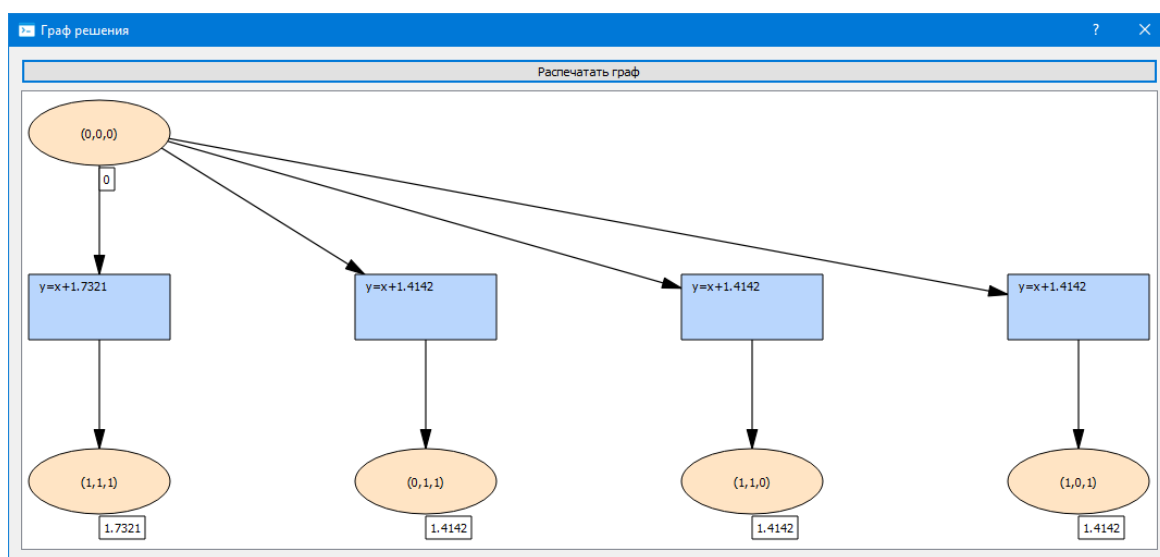


Рисунок 14 – Граф решения для переходов в пространстве с весом корень из 2 и корень из 3

Применение МЭС для планирования трехмерных маршрутов.

После генерации XML модели, включающую в себя отношения, параметры и правила, требуется задать препятствия и удалить их из XML файла модели. Препятствия могут быть трех видов. Первый вид – это препятствия, по форме схожие с прямоугольным параллелепипедом. Для их задания требуется обозначить два противоположных угла параллелепипеда. Остальные точки

фигуры будут найдены автоматически и удалены из XML файла. Второй вид препятствия – это прямые линии. У них есть точка начала и конца. Для большей функциональности при визуализации эти прямые отображаются как вектор из начальной точки в конечную. Третий вид препятствия – это отдельные вершины графа, в которые по каким-либо причинам нельзя попасть. Далее препятствия второго и третьего вида также удаляются из XML файла модели. При удалении вершин из модели удаляются параметр и отношения, относящиеся к этой вершине, а также все правила, для которых эта точка является входом или выходом.

После этого полученная XML модель с помощью POST запроса загружается в серверную версию «КЭСМИ Wi!Mi Разуматор» (рисунок 15). Если загрузка прошла успешно, то далее можно переходить к нахождению маршрута. Для этого требуется с помощью POST запроса передать на сервер ID загруженной модели, начальную точку маршрута и конечную точку маршрута. Далее полученный от сервера ответ требуется преобразовать в итоговый маршрут робота. Если нужно найти маршрут для нескольких роботов, то требуется выполнить несколько POST запросов с начальной и конечной точками. Если окружающая обстановка изменилась, то модель требуется сгенерировать заново, после загрузить на сервер, а далее построить маршрут с учетом измененной обстановки.

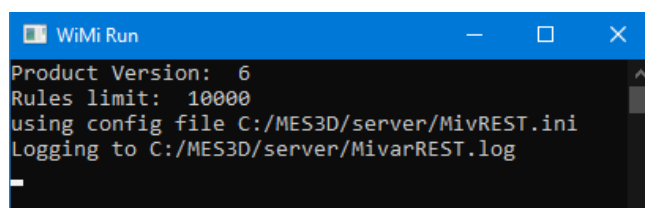


Рисунок 15 – Интерфейс серверной версии «КЭСМИ Wi!Mi Разуматор»

После получения маршрутов роботов, можно визуализировать полученные результаты. Для этого используются данные о всех трех типах препятствий, а также о всех построенных маршрутах и их длинах. Визуализация реализуется на языке программирования Python с помощью библиотеки «Matplotlib» (рисунок 16).

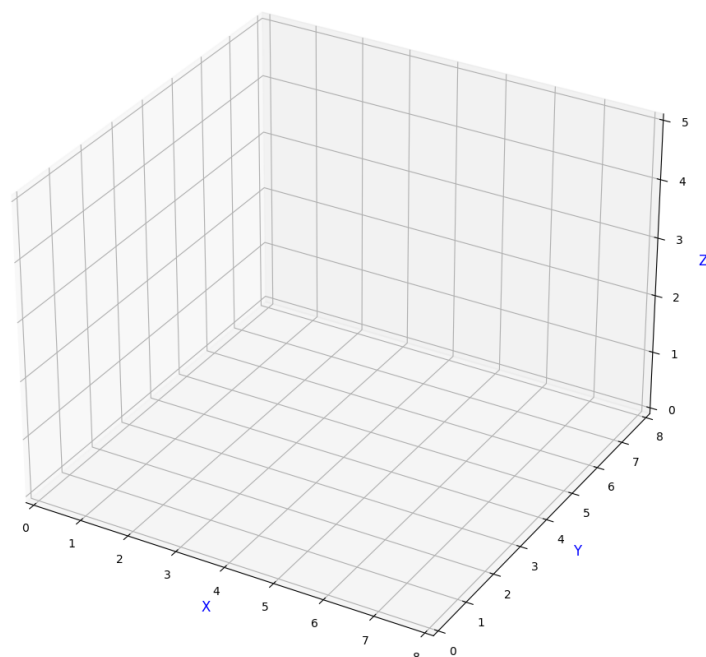


Рисунок 16 – Пример визуализации пространства с использованием «Matplotlib»

Пример визуализации маршрута одного робота без учета препятствий продемонстрирован на Рисунке 17.

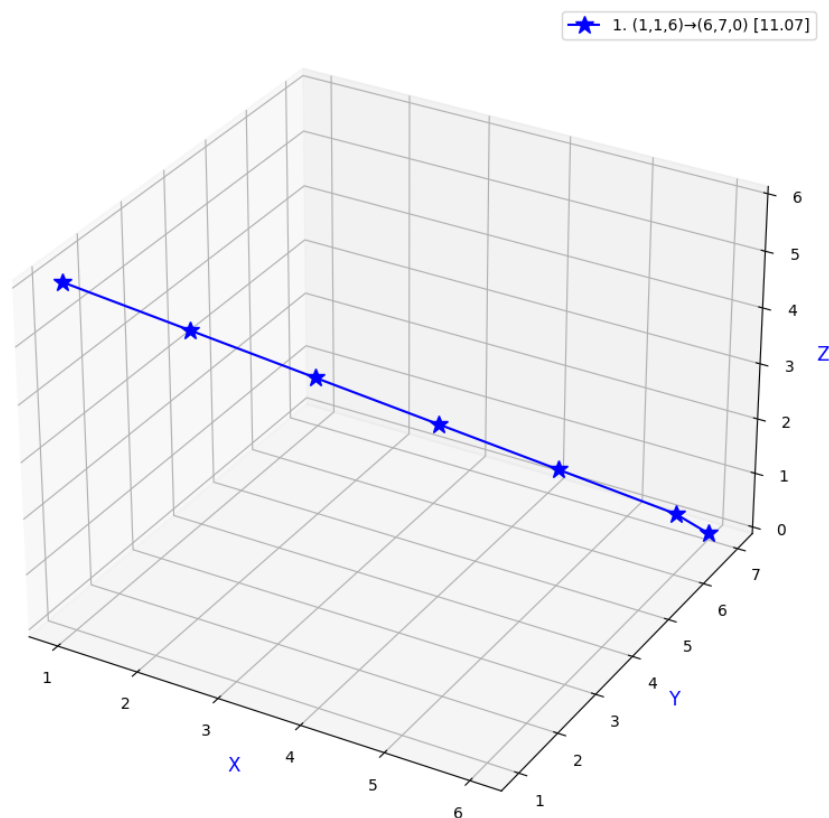


Рисунок 17 – Пример визуализации маршрута одного робота без учета препятствий

Далее будут продемонстрированы примеры визуализации маршрутов одного робота (рисунок 18) и трех роботов (рисунок 19) с учетом описанных выше трех типов препятствий.

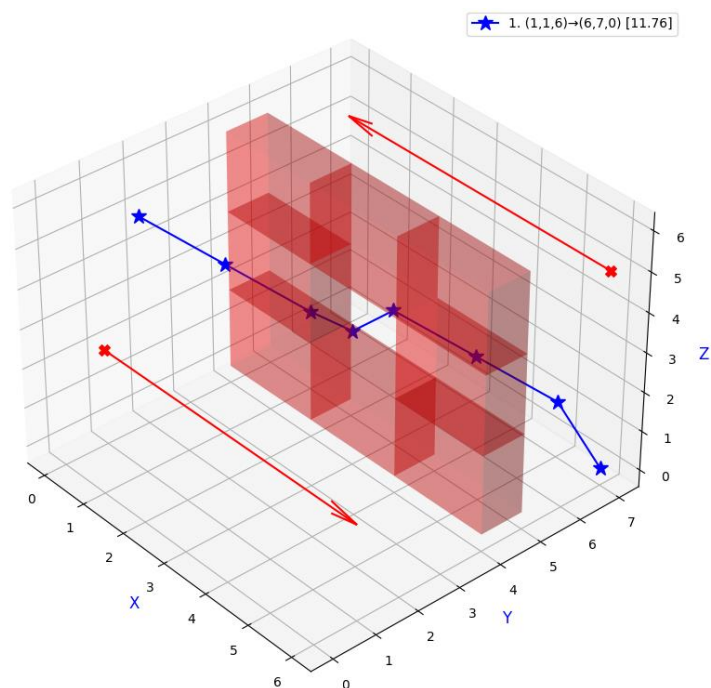


Рисунок 18 – Пример визуализации маршрута одного робота с учетом препятствий

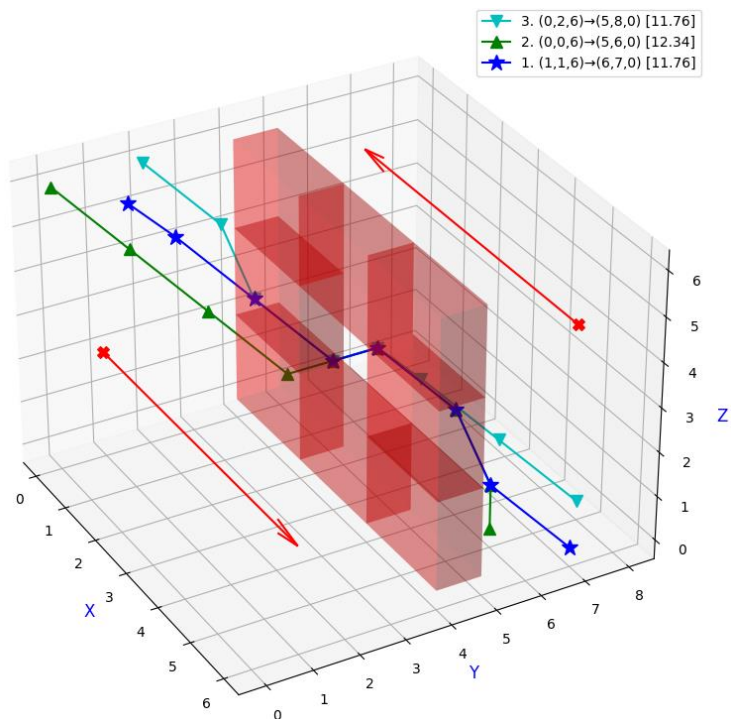


Рисунок 19 – Пример визуализации маршрутов трех роботов с учетом препятствий

Реализация AlphaGo Zero для Connect4 [1]

Игра рассчитана на двух участников, действующих на поле 6×7 . У игроков есть фишки двух цветов, которые они по очереди ставят в любой из семи столбцов. Фишки опускаются на дно, выстраиваясь в вертикальные столбики. Цель игры состоит в том, чтобы первым сформировать горизонтальную, вертикальную или диагональную группу из четырех фишек одного цвета. На рисунке 20 показаны две игровые ситуации. В первой светло-серые только что выиграли, а во второй темно-серые находятся в одном шаге от победы.

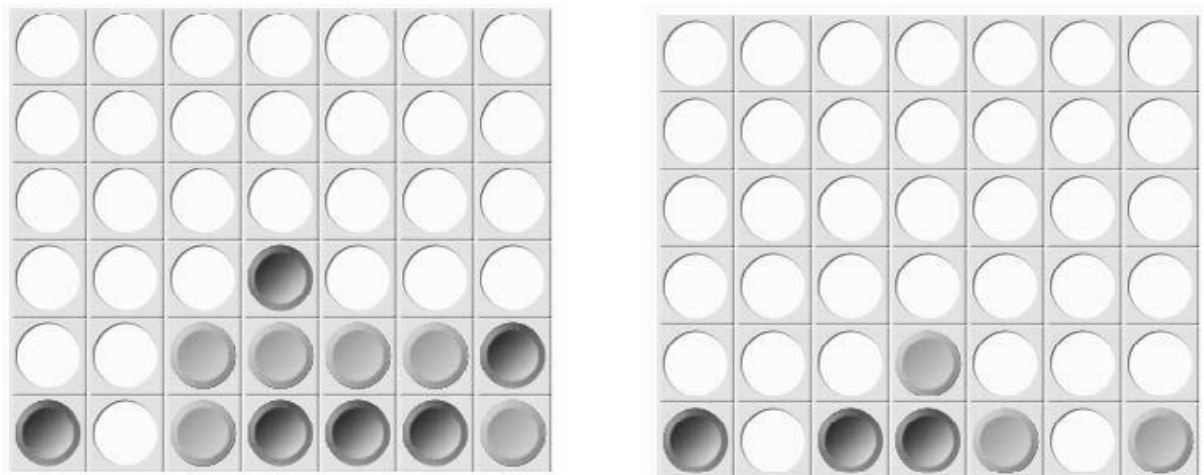


Рисунок 20 – Две игровые позиции в Connect4

Несмотря на простоту, в этой игре есть $4,5 \times 10^{12}$ различных состояний, которые обычным компьютерам довольно сложно проанализировать с помощью полного перебора. Этот пример состоит из нескольких инструментов и библиотечных модулей.

- В файле `Chapter18/lib/game.py` – базовое представление игры, которое содержит функции, позволяющие делать ходы, кодировать и декодировать игровое состояние, а также другие связанные с игрой утилиты.
- `Chapter18/lib/mcts.py` – реализация MCTS, которая допускает GPU-расширение листьев и обратное распространение узла. Центральный класс отвечает также за ведение статистики игровых узлов, которая повторно используется между поисками.

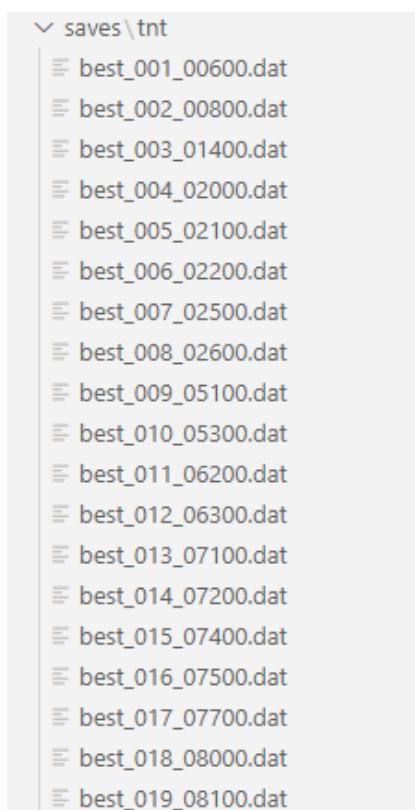
- Chapter18/lib/model.py – HC и другие связанные с моделью функции, такие как преобразование состояния игры для нейронной сети, а также функция для выполнения одной игры между двумя моделями.
- Chapter18/train.py – основная программа обучения, которая соединяет все компоненты воедино и создает контрольные точки модели для новых (рисунки 21) лучших сетей (рисунок 22).

```

Step 8275, steps 18, leaves 321, steps/s 12.86, leaves/s 229.29, best_idx 19, replay 5000
Step 8276, steps 12, leaves 217, steps/s 10.43, leaves/s 188.70, best_idx 19, replay 5000
Step 8277, steps 8, leaves 176, steps/s 7.49, leaves/s 164.75, best_idx 19, replay 5000
Step 8278, steps 6, leaves 98, steps/s 8.96, leaves/s 146.27, best_idx 19, replay 5000
Step 8279, steps 26, leaves 524, steps/s 11.70, leaves/s 235.83, best_idx 19, replay 5000
Step 8280, steps 8, leaves 157, steps/s 10.00, leaves/s 196.26, best_idx 19, replay 5000
Step 8281, steps 14, leaves 199, steps/s 12.35, leaves/s 175.48, best_idx 19, replay 5000
Step 8282, steps 14, leaves 364, steps/s 11.11, leaves/s 288.89, best_idx 19, replay 5000
Step 8284, steps 15, leaves 248, steps/s 11.51, leaves/s 190.33, best_idx 19, replay 5000
Step 8285, steps 8, leaves 163, steps/s 7.58, leaves/s 154.47, best_idx 19, replay 5000
Step 8287, steps 13, leaves 244, steps/s 12.08, leaves/s 226.77, best_idx 19, replay 5000
Step 8288, steps 9, leaves 160, steps/s 10.54, leaves/s 187.35, best_idx 19, replay 5000
Step 8290, steps 14, leaves 237, steps/s 8.92, leaves/s 151.04, best_idx 19, replay 5000
Step 8291, steps 12, leaves 234, steps/s 5.71, leaves/s 111.34, best_idx 19, replay 5000
Step 8292, steps 17, leaves 345, steps/s 5.14, leaves/s 104.29, best_idx 19, replay 5000
Step 8293, steps 8, leaves 192, steps/s 5.87, leaves/s 140.96, best_idx 19, replay 5000
Step 8295, steps 19, leaves 325, steps/s 7.90, leaves/s 135.06, best_idx 19, replay 5000
Step 8296, steps 23, leaves 350, steps/s 7.87, leaves/s 119.79, best_idx 19, replay 5000
Step 8298, steps 20, leaves 449, steps/s 9.85, leaves/s 221.05, best_idx 19, replay 5000
Step 8299, steps 11, leaves 236, steps/s 8.38, leaves/s 179.81, best_idx 19, replay 5000
Step 8300, steps 26, leaves 447, steps/s 10.34, leaves/s 177.83, best_idx 19, replay 5000
Step 8301, steps 10, leaves 256, steps/s 8.89, leaves/s 227.56, best_idx 19, replay 5000

```

Рисунок 21 – Часть вывода при обучении нейронных сетей



```

saves\tnt
├── best_001_00600.dat
├── best_002_00800.dat
├── best_003_01400.dat
├── best_004_02000.dat
├── best_005_02100.dat
├── best_006_02200.dat
├── best_007_02500.dat
├── best_008_02600.dat
├── best_009_05100.dat
├── best_010_05300.dat
├── best_011_06200.dat
├── best_012_06300.dat
├── best_013_07100.dat
├── best_014_07200.dat
├── best_015_07400.dat
├── best_016_07500.dat
├── best_017_07700.dat
├── best_018_08000.dat
└── best_019_08100.dat

```

Рисунок 22 – Полученные версии лучших нейронных сетей

- Chapter18/play.py – инструмент для организации автоматизированного турнира между состояниями модели. Он принимает несколько файлов моделей и играет определенное количество игр между ними (рисунок 23), чтобы сформировать таблицу лидеров.

```
Speed 1.05 games/s
best_019_08100.dat vs best_006_02200.dat -> w=1, l=1, d=0
Speed 1.34 games/s
best_019_08100.dat vs best_007_02500.dat -> w=2, l=0, d=0
Speed 1.31 games/s
best_019_08100.dat vs best_008_02600.dat -> w=1, l=1, d=0
Speed 0.97 games/s
best_019_08100.dat vs best_009_05100.dat -> w=1, l=1, d=0
Speed 2.01 games/s
best_019_08100.dat vs best_010_05300.dat -> w=0, l=2, d=0
Speed 1.10 games/s
best_019_08100.dat vs best_011_06200.dat -> w=1, l=1, d=0
Speed 1.17 games/s
best_019_08100.dat vs best_012_06300.dat -> w=0, l=2, d=0
Speed 0.93 games/s
best_019_08100.dat vs best_013_07100.dat -> w=2, l=0, d=0
Speed 0.74 games/s
best_019_08100.dat vs best_014_07200.dat -> w=1, l=1, d=0
Speed 1.19 games/s
best_019_08100.dat vs best_015_07400.dat -> w=1, l=1, d=0
Speed 0.97 games/s
best_019_08100.dat vs best_016_07500.dat -> w=0, l=2, d=0
Speed 0.72 games/s
best_019_08100.dat vs best_017_07700.dat -> w=0, l=2, d=0
Speed 0.81 games/s
best_019_08100.dat vs best_018_08000.dat -> w=0, l=2, d=0
```

Рисунок 23 – Часть вывода при автоматизированном турнире

```
Leaderboard:
best_009_05100.dat:      w=48, l=24, d=0
best_015_07400.dat:      w=43, l=29, d=0
best_005_02100.dat:      w=41, l=31, d=0
best_013_07100.dat:      w=41, l=31, d=0
best_006_02200.dat:      w=40, l=32, d=0
best_017_07700.dat:      w=40, l=32, d=0
best_003_01400.dat:      w=39, l=33, d=0
best_004_02000.dat:      w=37, l=35, d=0
best_019_08100.dat:      w=36, l=36, d=0
best_016_07500.dat:      w=35, l=37, d=0
best_008_02600.dat:      w=34, l=38, d=0
best_010_05300.dat:      w=34, l=38, d=0
best_012_06300.dat:      w=34, l=38, d=0
best_018_08000.dat:      w=33, l=39, d=0
best_002_00800.dat:      w=32, l=40, d=0
best_014_07200.dat:      w=32, l=40, d=0
best_007_02500.dat:      w=30, l=42, d=0
best_011_06200.dat:      w=29, l=43, d=0
best_001_00600.dat:      w=26, l=46, d=0
```

Рисунок 24 – Полученная турнирная таблица

- Chapter18/telegram-bot.py – бот для чата Telegram, который позволяет пользователю играть против любого файла модели (рисунок 25), сохраняя статистику. Этот бот был использован для проверки обученных моделей на живых людях (рисунок 26).

```
The list of available models with their IDs
0: best_001_00600.dat
1: best_002_00800.dat
2: best_003_01400.dat
3: best_004_02000.dat
4: best_005_02100.dat
5: best_006_02200.dat
6: best_007_02500.dat
7: best_008_02600.dat
8: best_009_05100.dat
9: best_010_05300.dat
10: best_011_06200.dat
11: best_012_06300.dat
12: best_013_07100.dat
13: best_014_07200.dat
14: best_015_07400.dat
15: best_016_07500.dat
16: best_017_07700.dat
17: best_018_08000.dat
18: best_019_08100.dat
```

15:07

Рисунок 25 – Файлы модели, доступные для игры

The first move is mine (I'm playing with X), moving..

Position evaluation: 0.67

0123456

X

0123456

15:07

Position evaluation: 0.84

0123456

X

O X

0123456

15:07

Position evaluation: 0.50

0123456

O

X

O X X

0123456

15:07

Position evaluation: 1.00

0123456

O

X

OO XXX

0123456

15:07

Position evaluation: 0.98

0123456

O

XO X

OO XXX

0123456

15:07

Position evaluation: 1.00

0123456

O O

XO X

OOXXXX

0123456

15:07

I won! Wheeee!

Рисунок 26 – Пример тестовой игры с моделью best_010_05300.dat

Листинг кода представлен в Приложении 1.

В итоге мы реализовали метод AlphaGo Zero, созданный DeepMind, для решения настольных игр с полной информацией. Основная задача этого метода состоит в том, чтобы позволить агентам повысить свой уровень мастерства с помощью самостоятельной игры, не имея предварительных знаний, основанных на пройденных людьми играх или полученных из других источников.

Список использованных источников

1. Саттон Р. С., Барто Э. Дж. Обучение с подкреплением: Введение. 2-е изд. / пер. с англ. А. А. Слинкина. М.: ДМК Пресс, 2020, 552 с.
2. Лапань Максим. Глубокое обучение с подкреплением. AlphaGo и другие технологии. СПб.: Питер, 2020, 496 с.
3. Равичандиран Судхарсан. Глубокое обучение с подкреплением на Python. OpenAI Gym и TensorFlow для профи. СПб.: Питер, 2020, 320 с.
4. Грессер Лаура, Кенг Ван Лун. Глубокое обучение с подкреплением: теория и практика на языке Python. СПб.: Питер, 2022, 416 с.
5. Алфимцев А.Н. Мультиагентное обучение с подкреплением: учебное пособие. Москва: Изд-во МГТУ им. Н.Э. Баумана, 2021, 222 с.
6. Лонца Андреа. Алгоритмы обучения с подкреплением на Python / пер. с англ. А. А. Слинкина. М.: ДМК Пресс, 2020, 286 с.
7. Лю Юси. Обучение с подкреплением на PyTorch: сборник рецептов / пер. с англ. А. А. Слинкина. М.: ДМК Пресс, 2020, 282 с.
8. Жданов А.А. Автономный искусственный интеллект. 5-е изд., М.: Лаборатория знаний, 2020. 362 с.

Приложение 1

Листинг кода реализации AlphaGo Zero для Connect4 [1].

game.py

```
"""
4-in-a-row game-related functions.

Field is 6*7 with pieces falling from the top to the bottom. There are two kinds of
pieces: black and white,
which are encoded by 1 (black) and 0 (white).

There are two representation of the game:
1. List of 7 lists with elements ordered from the bottom. For example, this field

0      1
0      1
10     1
10 0 1
10 1 1
101 111

Will be encoded as [
    [1, 1, 1, 1, 0, 0],
    [0, 0, 0, 0],
    [1],
    [],
    [1, 1, 0],
    [1],
    [1, 1, 1, 1, 1, 1]
]

2. integer number consists from:
    a. 7*6 bits (column-wise) encoding the field. Unoccupied bits are zero
    b. 7*3 bits, each 3-bit number encodes amount of free entries on the top.
In this representation, the field above will be equal to those bits:
[
    111100,
    000000,
    100000,
    000000,
    110000,
    100000,
    111111,
    000,
    010,
    101,
    110,
    011,
    101,
    000
]

All the code is generic, so, in theory you can try to adjust the field size.
But tests could become broken.
"""

GAME_ROWS = 6
GAME_COLS = 7
BITS_IN_LEN = 3
PLAYER_BLACK = 1
PLAYER_WHITE = 0
```

```

COUNT_TO_WIN = 4

# declared after encode_lists
# INITIAL_STATE = encode_lists([[]] * GAME_COLS)

def bits_to_int(bits):
    res = 0
    for b in bits:
        res *= 2
        res += b
    return res

def int_to_bits(num, bits):
    res = []
    for _ in range(bits):
        res.append(num % 2)
        num //= 2
    return res[::-1]

def encode_lists(field_lists):
    """
    Encode lists representation into the binary numbers
    :param field_lists: list of GAME_COLS lists with 0s and 1s
    :return: integer number with encoded game state
    """
    assert isinstance(field_lists, list)
    assert len(field_lists) == GAME_COLS

    bits = []
    len_bits = []
    for col in field_lists:
        bits.extend(col)
        free_len = GAME_ROWS - len(col)
        bits.extend([0] * free_len)
        len_bits.extend(int_to_bits(free_len, bits=BITS_IN_LEN))
    bits.extend(len_bits)
    return bits_to_int(bits)

INITIAL_STATE = encode_lists([[]] * GAME_COLS)

def decode_binary(state_int):
    """
    Decode binary representation into the list view
    :param state_int: integer representing the field
    :return: list of GAME_COLS lists
    """
    assert isinstance(state_int, int)
    bits = int_to_bits(state_int, bits=GAME_COLS*GAME_ROWS + GAME_COLS*BITS_IN_LEN)
    res = []
    len_bits = bits[GAME_COLS*GAME_ROWS:]
    for col in range(GAME_COLS):
        vals = bits[col*GAME_ROWS:(col+1)*GAME_ROWS]
        lens = bits_to_int(len_bits[col*BITS_IN_LEN:(col+1)*BITS_IN_LEN])
        if lens > 0:
            vals = vals[:lens]
        res.append(vals)
    return res

def possible_moves(state_int):
    """
    This function could be calculated directly from bits, but I'm too lazy

```

```

:param state_int: field representation
:return: the list of columns which we can make a move
"""

assert isinstance(state_int, int)
field = decode_binary(state_int)
return [idx for idx, col in enumerate(field) if len(col) < GAME_ROWS]

def _check_won(field, col, delta_row):
    """
    Check for horisontal/diagonal win condition for the last player moved in the
    column
    :param field: list of lists
    :param col: column index
    :param delta_row: if 0, checks for horisontal won, 1 for rising diagonal, -1 for
    falling
    :return: True if won, False if not
    """
    player = field[col][-1]
    coord = len(field[col])-1
    total = 1
    # negative dir
    cur_coord = coord - delta_row
    for c in range(col-1, -1, -1):
        if len(field[c]) <= cur_coord or cur_coord < 0 or cur_coord >= GAME_ROWS:
            break
        if field[c][cur_coord] != player:
            break
        total += 1
        if total == COUNT_TO_WIN:
            return True
        cur_coord -= delta_row
    # positive dir
    cur_coord = coord + delta_row
    for c in range(col+1, GAME_COLS):
        if len(field[c]) <= cur_coord or cur_coord < 0 or cur_coord >= GAME_ROWS:
            break
        if field[c][cur_coord] != player:
            break
        total += 1
        if total == COUNT_TO_WIN:
            return True
        cur_coord += delta_row
    return False

def move(state_int, col, player):
    """
    Perform move into given column. Assume the move could be performed, otherwise,
    assertion will be raised
    :param state_int: current state
    :param col: column to make a move
    :param player: player index (PLAYER_WHITE or PLAYER_BLACK)
    :return: tuple of (state_new, won). Value won is bool, True if this move lead
    to victory or False otherwise (but it could be a draw)
    """
    assert isinstance(state_int, int)
    assert isinstance(col, int)
    assert 0 <= col < GAME_COLS
    assert player == PLAYER_BLACK or player == PLAYER_WHITE
    field = decode_binary(state_int)
    assert len(field[col]) < GAME_ROWS
    field[col].append(player)
    # check for victory: the simplest vertical case

```

```

    suff = field[col][-COUNT_TO_WIN:]
    won = suff == [player] * COUNT_TO_WIN
    if not won:
        won = _check_won(field, col, 0) or _check_won(field, col, 1) or
        _check_won(field, col, -1)
    state_new = encode_lists(field)
    return state_new, won

def render(state_int):
    state_list = decode_binary(state_int)
    data = [[' ']* GAME_COLS for _ in range(GAME_ROWS)]
    for col_idx, col in enumerate(state_list):
        for rev_row_idx, cell in enumerate(col):
            row_idx = GAME_ROWS - rev_row_idx - 1
            data[row_idx][col_idx] = str(cell)
    return [''.join(row) for row in data]

def update_counts(counts_dict, key, counts):
    v = counts_dict.get(key, (0, 0, 0))
    res = (v[0] + counts[0], v[1] + counts[1], v[2] + counts[2])
    counts_dict[key] = res

```

mcts.py

```

"""
Monte-Carlo Tree Search
"""
import math as m
import numpy as np

from lib import game, model

import torch.nn.functional as F

class MCTS:
    """
    Class keeps statistics for every state encountered during the search
    """
    def __init__(self, c_puct=1.0):
        self.c_puct = c_puct
        # count of visits, state_int -> [N(s, a)]
        self.visit_count = {}
        # total value of the state's action, state_int -> [W(s, a)]
        self.value = {}
        # average value of actions, state_int -> [Q(s, a)]
        self.value_avg = {}
        # prior probability of actions, state_int -> [P(s,a)]
        self.probs = {}

    def clear(self):
        self.visit_count.clear()
        self.value.clear()
        self.value_avg.clear()
        self.probs.clear()

    def __len__(self):
        return len(self.value)

    def find_leaf(self, state_int, player):
        """
        Traverse the tree until the end of game or leaf node

```

```

        :param state_int: root node state
        :param player: player to move
        :return: tuple of (value, leaf_state, player, states, actions)
        1. value: None if leaf node, otherwise equals to the game outcome for the
player at leaf
        2. leaf_state: state_int of the last state
        3. player: player at the leaf node
        4. states: list of states traversed
        5. list of actions taken
        """
        states = []
        actions = []
        cur_state = state_int
        cur_player = player
        value = None

        while not self.is_leaf(cur_state):
            states.append(cur_state)

            counts = self.visit_count[cur_state]
            total_sqrt = m.sqrt(sum(counts))
            probs = self.probs[cur_state]
            values_avg = self.value_avg[cur_state]

            # choose action to take, in the root node add the Dirichlet noise to the
probs
            if cur_state == state_int:
                noises = np.random.dirichlet([0.03] * game.GAME_COLS)
                probs = [0.75 * prob + 0.25 * noise for prob, noise in zip(probs,
noises)]

            score = [value + self.c_puct * prob * total_sqrt / (1 + count)
                    for value, prob, count in zip(values_avg, probs, counts)]
            invalid_actions = set(range(game.GAME_COLS)) -
set(game.possible_moves(cur_state))
            for invalid in invalid_actions:
                score[invalid] = -np.inf
            action = int(np.argmax(score))
            actions.append(action)
            cur_state, won = game.move(cur_state, action, cur_player)
            if won:
                # if somebody won the game, the value of the final state is -1 (as it
is on opponent's turn)
                value = -1.0
                cur_player = 1-cur_player
                # check for the draw
                if value is None and len(game.possible_moves(cur_state)) == 0:
                    value = 0.0

            return value, cur_state, cur_player, states, actions

    def is_leaf(self, state_int):
        return state_int not in self.probs

    def search_batch(self, count, batch_size, state_int, player, net, device="cpu"):
        for _ in range(count):
            self.search_minibatch(batch_size, state_int, player, net, device)

    def search_minibatch(self, count, state_int, player, net, device="cpu"):
        """
        Perform several MCTS searches.
        """
        backup_queue = []

```



```

        expand_states = []
        expand_players = []
        expand_queue = []
        planned = set()
        for _ in range(count):
            value, leaf_state, leaf_player, states, actions =
self.find_leaf(state_int, player)
            if value is not None:
                backup_queue.append((value, states, actions))
            else:
                if leaf_state not in planned:
                    planned.add(leaf_state)
                    leaf_state_lists = game.decode_binary(leaf_state)
                    expand_states.append(leaf_state_lists)
                    expand_players.append(leaf_player)
                    expand_queue.append((leaf_state, states, actions))

# do expansion of nodes
if expand_queue:
    batch_v = model.state_lists_to_batch(expand_states, expand_players,
device)

    logits_v, values_v = net(batch_v)
    probs_v = F.softmax(logits_v, dim=1)
    values = values_v.data.cpu().numpy()[0]
    probs = probs_v.data.cpu().numpy()
    # create the nodes
    for (leaf_state, states, actions), value, prob in zip(expand_queue,
values, probs):
        self.visit_count[leaf_state] = [0] * game.GAME_COLS
        self.value[leaf_state] = [0.0] * game.GAME_COLS
        self.value_avg[leaf_state] = [0.0] * game.GAME_COLS
        self.probs[leaf_state] = prob
        backup_queue.append((value, states, actions))

# perform backup of the searches
for value, states, actions in backup_queue:
    # leaf state is not stored in states and actions, so the value of the
leaf will be the value of the opponent
    cur_value = -value
    for state_int, action in zip(states[::-1], actions[::-1]):
        self.visit_count[state_int][action] += 1
        self.value[state_int][action] += cur_value
        self.value_avg[state_int][action] = self.value[state_int][action] /
self.visit_count[state_int][action]
        cur_value = -cur_value

def get_policy_value(self, state_int, tau=1):
    """
    Extract policy and action-values by the state
    :param state_int: state of the board
    :return: (probs, values)
    """
    counts = self.visit_count[state_int]
    if tau == 0:
        probs = [0.0] * game.GAME_COLS
        probs[np.argmax(counts)] = 1.0
    else:
        counts = [count ** (1.0 / tau) for count in counts]
        total = sum(counts)
        probs = [count / total for count in counts]
    values = self.value_avg[state_int]
    return probs, values

```

model.py

```
import collections
import numpy as np

import torch
import torch.nn as nn

from lib import game, mcts

OBS_SHAPE = (2, game.GAME_ROWS, game.GAME_COLS)
NUM_FILTERS = 64

class Net(nn.Module):
    def __init__(self, input_shape, actions_n):
        super(Net, self).__init__()

        self.conv_in = nn.Sequential(
            nn.Conv2d(input_shape[0], NUM_FILTERS, kernel_size=3, padding=1),
            nn.BatchNorm2d(NUM_FILTERS),
            nn.LeakyReLU()
        )

        # layers with residual
        self.conv_1 = nn.Sequential(
            nn.Conv2d(NUM_FILTERS, NUM_FILTERS, kernel_size=3, padding=1),
            nn.BatchNorm2d(NUM_FILTERS),
            nn.LeakyReLU()
        )
        self.conv_2 = nn.Sequential(
            nn.Conv2d(NUM_FILTERS, NUM_FILTERS, kernel_size=3, padding=1),
            nn.BatchNorm2d(NUM_FILTERS),
            nn.LeakyReLU()
        )
        self.conv_3 = nn.Sequential(
            nn.Conv2d(NUM_FILTERS, NUM_FILTERS, kernel_size=3, padding=1),
            nn.BatchNorm2d(NUM_FILTERS),
            nn.LeakyReLU()
        )
        self.conv_4 = nn.Sequential(
            nn.Conv2d(NUM_FILTERS, NUM_FILTERS, kernel_size=3, padding=1),
            nn.BatchNorm2d(NUM_FILTERS),
            nn.LeakyReLU()
        )
        self.conv_5 = nn.Sequential(
            nn.Conv2d(NUM_FILTERS, NUM_FILTERS, kernel_size=3, padding=1),
            nn.BatchNorm2d(NUM_FILTERS),
            nn.LeakyReLU()
        )

        body_out_shape = (NUM_FILTERS, ) + input_shape[1:]

        # value head
        self.conv_val = nn.Sequential(
            nn.Conv2d(NUM_FILTERS, 1, kernel_size=1),
            nn.BatchNorm2d(1),
            nn.LeakyReLU()
        )

        conv_val_size = self._get_conv_val_size(body_out_shape)
        self.value = nn.Sequential(
            nn.Linear(conv_val_size, 20),
            nn.LeakyReLU(),
```

```

        nn.Linear(20, 1),
        nn.Tanh()
    )

    # policy head
    self.conv_policy = nn.Sequential(
        nn.Conv2d(NUM_FILTERS, 2, kernel_size=1),
        nn.BatchNorm2d(2),
        nn.LeakyReLU()
    )
    conv_policy_size = self._get_conv_policy_size(body_out_shape)
    self.policy = nn.Sequential(
        nn.Linear(conv_policy_size, actions_n)
    )

def _get_conv_val_size(self, shape):
    o = self.conv_val(torch.zeros(1, *shape))
    return int(np.prod(o.size()))

def _get_conv_policy_size(self, shape):
    o = self.conv_policy(torch.zeros(1, *shape))
    return int(np.prod(o.size()))

def forward(self, x):
    batch_size = x.size()[0]
    v = self.conv_in(x)
    v = v + self.conv_1(v)
    v = v + self.conv_2(v)
    v = v + self.conv_3(v)
    v = v + self.conv_4(v)
    v = v + self.conv_5(v)
    val = self.conv_val(v)
    val = self.value(val.view(batch_size, -1))
    pol = self.conv_policy(v)
    pol = self.policy(pol.view(batch_size, -1))
    return pol, val

def _encode_list_state(dest_np, state_list, who_move):
    """
    In-place encodes list state into the zero numpy array
    :param dest_np: dest array, expected to be zero
    :param state_list: state of the game in the list form
    :param who_move: player index (game.PLAYER_WHITE or game.PLAYER_BLACK) who to
move
    """
    assert dest_np.shape == OBS_SHAPE

    for col_idx, col in enumerate(state_list):
        for rev_row_idx, cell in enumerate(col):
            row_idx = game.GAME_ROWS - rev_row_idx - 1
            if cell == who_move:
                dest_np[0, row_idx, col_idx] = 1.0
            else:
                dest_np[1, row_idx, col_idx] = 1.0

def state_lists_to_batch(state_lists, who_moves_lists, device="cpu"):
    """
    Convert list of list states to batch for network
    :param state_lists: list of 'list states'
    :param who_moves_lists: list of player index who moves
    :return Variable with observations
    """

```

```

    assert isinstance(state_lists, list)
    batch_size = len(state_lists)
    batch = np.zeros((batch_size,) + OBS_SHAPE, dtype=np.float32)
    for idx, (state, who_move) in enumerate(zip(state_lists, who_moves_lists)):
        _encode_list_state(batch[idx], state, who_move)
    return torch.tensor(batch).to(device)

# def play_game(net1, net2, cuda=False):
#     cur_player = 0
#     state = game.INITIAL_STATE
#     nets = [net1, net2]
#
#     while True:
#         state_list = game.decode_binary(state)
#         batch_v = state_lists_to_batch([state_list], [cur_player], cuda)
#         logits_v, _ = nets[cur_player](batch_v)
#         probs_v = F.softmax(logits_v, dim=1)
#         probs = probs_v[0].data.cpu().numpy()
#         while True:
#             action = np.random.choice(game.GAME_COLS, p=probs)
#             if action in game.possible_moves(state):
#                 break
#         state, won = game.move(state, action, cur_player)
#         if won:
#             return 1.0 if cur_player == 0 else -1.0
#         # check for the draw state
#         if len(game.possible_moves(state)) == 0:
#             return 0.0
#         cur_player = 1 - cur_player
#
def play_game(mcts_stores, replay_buffer, net1, net2, steps_before_tau_0,
mcts_searches, mcts_batch_size,
              net1_plays_first=None, device="cpu"):
    """
    Play one single game, memorizing transitions into the replay buffer
    :param mcts_stores: could be None or single MCTS or two MCTSes for individual net
    :param replay_buffer: queue with (state, probs, values), if None, nothing is
    stored
    :param net1: player1
    :param net2: player2
    :return: value for the game in respect to player1 (+1 if p1 won, -1 if lost, 0 if
    draw)
    """
    assert isinstance(replay_buffer, (collections.deque, type(None)))
    assert isinstance(mcts_stores, (mcts.MCTS, type(None), list))
    assert isinstance(net1, Net)
    assert isinstance(net2, Net)
    assert isinstance(steps_before_tau_0, int) and steps_before_tau_0 >= 0
    assert isinstance(mcts_searches, int) and mcts_searches > 0
    assert isinstance(mcts_batch_size, int) and mcts_batch_size > 0

    if mcts_stores is None:
        mcts_stores = [mcts.MCTS(), mcts.MCTS()]
    elif isinstance(mcts_stores, mcts.MCTS):
        mcts_stores = [mcts_stores, mcts_stores]

    state = game.INITIAL_STATE
    nets = [net1, net2]
    if net1_plays_first is None:
        cur_player = np.random.choice(2)
    else:

```

```

        cur_player = 0 if net1_plays_first else 1
    step = 0
    tau = 1 if steps_before_tau_0 > 0 else 0
    game_history = []

    result = None
    net1_result = None

    while result is None:
        mcts_stores[cur_player].search_batch(mcts_searches, mcts_batch_size, state,
                                             cur_player, nets[cur_player],
device=device)
        probs, _ = mcts_stores[cur_player].get_policy_value(state, tau=tau)
        game_history.append((state, cur_player, probs))
        action = np.random.choice(game.GAME_COLS, p=probs)
        if action not in game.possible_moves(state):
            print("Impossible action selected")
        state, won = game.move(state, action, cur_player)
        if won:
            result = 1
            net1_result = 1 if cur_player == 0 else -1
            break
        cur_player = 1-cur_player
        # check the draw case
        if len(game.possible_moves(state)) == 0:
            result = 0
            net1_result = 0
            break
        step += 1
        if step >= steps_before_tau_0:
            tau = 0

    if replay_buffer is not None:
        for state, cur_player, probs in reversed(game_history):
            replay_buffer.append((state, cur_player, probs, result))
            result = -result

    return net1_result, step

```

train.py

```

#!/usr/bin/env python3
import os
import time
import ptan
import random
import argparse
import collections

from lib import game, model, mcts

from tensorboardX import SummaryWriter

import torch
import torch.optim as optim
import torch.nn.functional as F

PLAY_EPISODES = 1 #25
MCTS_SEARCHES = 10
MCTS_BATCH_SIZE = 8
REPLAY_BUFFER = 5000 # 30000

```

```

LEARNING_RATE = 0.1
BATCH_SIZE = 256
TRAIN_ROUNDS = 10
MIN_REPLAY_TO_TRAIN = 2000 #10000

BEST_NET_WIN_RATIO = 0.60

EVALUATE_EVERY_STEP = 100
EVALUATION_ROUNDS = 20
STEPS_BEFORE_TAU_0 = 10

def evaluate(net1, net2, rounds, device="cpu"):
    n1_win, n2_win = 0, 0
    mcts_stores = [mcts.MCTS(), mcts.MCTS()]

    for r_idx in range(rounds):
        r, _ = model.play_game(mcts_stores=mcts_stores, replay_buffer=None,
net1=net1, net2=net2,
                                steps_before_tau_0=0, mcts_searches=20,
mcts_batch_size=16,
                                device=device)

        if r < -0.5:
            n2_win += 1
        elif r > 0.5:
            n1_win += 1
    return n1_win / (n1_win + n2_win)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    # parser.add_argument("-n", "--name", required=True, help="Name of the run")
    parser.add_argument("--cuda", default=False, action="store_true", help="Enable
CUDA")
    args = parser.parse_args()
    device = torch.device("cuda" if args.cuda else "cpu")

    args.name = "tnt"

    saves_path = os.path.join("saves", args.name)
    os.makedirs(saves_path, exist_ok=True)
    writer = SummaryWriter(comment="-" + args.name)

    net = model.Net(input_shape=model.OBS_SHAPE, actions_n=game.GAME_COLS).to(device)
    best_net = ptan.agent.TargetNet(net)
    print(net)

    optimizer = optim.SGD(net.parameters(), lr=LEARNING_RATE, momentum=0.9)

    replay_buffer = collections.deque(maxlen=REPLAY_BUFFER)
    mcts_store = mcts.MCTS()
    step_idx = 0
    best_idx = 0

    with ptan.common.utils.TBMeanTracker(writer, batch_size=10) as tb_tracker:
        while True:
            t = time.time()
            prev_nodes = len(mcts_store)
            game_steps = 0
            for _ in range(PLAY_EPISODES):
                _, steps = model.play_game(mcts_store, replay_buffer,
best_net.target_model, best_net.target_model,
                                        steps_before_tau_0=STEPS_BEFORE_TAU_0,
mcts_searches=MCTS_SEARCHES,

```

```

device=device)
        mcts_batch_size=MCTS_BATCH_SIZE,

        game_steps += steps
        game_nodes = len(mcts_store) - prev_nodes
        dt = time.time() - t
        speed_steps = game_steps / dt
        speed_nodes = game_nodes / dt
        tb_tracker.track("speed_steps", speed_steps, step_idx)
        tb_tracker.track("speed_nodes", speed_nodes, step_idx)
        print("Step %d, steps %3d, leaves %4d, steps/s %5.2f, leaves/s %6.2f,
best_idx %d, replay %d" % (
            step_idx, game_steps, game_nodes, speed_steps, speed_nodes, best_idx,
len(replay_buffer)))
        step_idx += 1

        if len(replay_buffer) < MIN_REPLAY_TO_TRAIN:
            continue

        # train
        sum_loss = 0.0
        sum_value_loss = 0.0
        sum_policy_loss = 0.0

        for _ in range(TRAIN_ROUNDS):
            batch = random.sample(replay_buffer, BATCH_SIZE)
            batch_states, batch_who_moves, batch_probs, batch_values =
zip(*batch)
            batch_states_lists = [game.decode_binary(state) for state in
batch_states]
            states_v = model.state_lists_to_batch(batch_states_lists,
batch_who_moves, device)

            optimizer.zero_grad()
            probs_v = torch.FloatTensor(batch_probs).to(device)
            values_v = torch.FloatTensor(batch_values).to(device)
            out_logits_v, out_values_v = net(states_v)

            loss_value_v = F.mse_loss(out_values_v.squeeze(-1), values_v)
            loss_policy_v = -F.log_softmax(out_logits_v, dim=1) * probs_v
            loss_policy_v = loss_policy_v.sum(dim=1).mean()

            loss_v = loss_policy_v + loss_value_v
            loss_v.backward()
            optimizer.step()
            sum_loss += loss_v.item()
            sum_value_loss += loss_value_v.item()
            sum_policy_loss += loss_policy_v.item()

            tb_tracker.track("loss_total", sum_loss / TRAIN_ROUNDS, step_idx)
            tb_tracker.track("loss_value", sum_value_loss / TRAIN_ROUNDS, step_idx)
            tb_tracker.track("loss_policy", sum_policy_loss / TRAIN_ROUNDS, step_idx)

        # evaluate net
        if step_idx % EVALUATE_EVERY_STEP == 0:
            win_ratio = evaluate(net, best_net.target_model,
rounds=EVALUATION_ROUNDS, device=device)
            print("Net evaluated, win ratio = %.2f" % win_ratio)
            writer.add_scalar("eval_win_ratio", win_ratio, step_idx)
            if win_ratio > BEST_NET_WIN_RATIO:
                print("Net is better than cur best, sync")
                best_net.sync()
                best_idx += 1

```

```

        file_name = os.path.join(saves_path, "best_%03d_%05d.dat" %
(best_idx, step_idx))
        torch.save(net.state_dict(), file_name)
        mcts_store.clear()

```

play.py

```

#!/usr/bin/env python3
import sys
import time
import argparse

from lib import game, model

import torch

MCTS_SEARCHES = 10
MCTS_BATCH_SIZE = 8

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("models", nargs='+', help="The list of models (at least 2) to
play against each other")
    parser.add_argument("-r", "--rounds", type=int, default=2, help="Count of rounds
to perform for every pair")
    parser.add_argument("--cuda", default=False, action="store_true", help="Enable
CUDA")
    args = parser.parse_args()
    device = torch.device("cuda" if args.cuda else "cpu")

    nets = []
    for fname in args.models:
        net = model.Net(model.OBS_SHAPE, game.GAME_COLS)
        net.load_state_dict(torch.load(fname, map_location=lambda storage, loc:
storage))
        net = net.to(device)
        nets.append((fname, net))

    total_agent = {}
    total_pairs = {}

    for idx1, n1 in enumerate(nets):
        for idx2, n2 in enumerate(nets):
            if idx1 == idx2:
                continue
            wins, losses, draws = 0, 0, 0
            ts = time.time()
            for _ in range(args.rounds):
                r, _ = model.play_game(mcts_stores=None, replay_buffer=None,
net1=n1[1], net2=n2[1], steps_before_tau_0=0,
mcts_searches=MCTS_SEARCHES,
mcts_batch_size=MCTS_BATCH_SIZE, device=device)
                if r > 0.5:
                    wins += 1
                elif r < -0.5:
                    losses += 1
                else:
                    draws += 1
            speed_games = args.rounds / (time.time() - ts)
            name_1, name_2 = n1[0], n2[0]

```



```

        print("%s vs %s -> w=%d, l=%d, d=%d" % (name_1, name_2, wins, losses,
draws))
        sys.stderr.write("Speed %.2f games/s\n" % speed_games)
        sys.stdout.flush()
        game.update_counts(total_agent, name_1, (wins, losses, draws))
        game.update_counts(total_agent, name_2, (losses, wins, draws))
        game.update_counts(total_pairs, (name_1, name_2), (wins, losses, draws))

# leaderboard by total wins
total_leaders = list(total_agent.items())
total_leaders.sort(reverse=True, key=lambda p: p[1][0])

print("Leaderboard:")
for name, (wins, losses, draws) in total_leaders:
    print("%s: \t w=%d, l=%d, d=%d" % (name, wins, losses, draws))

```

telegram-bot.py

```

#!/usr/bin/env python3
# This module requires python-telegram-bot
import os
import sys
import glob
import json
import time
import datetime
import random
import logging
import numpy as np
import configparser
import argparse

from lib import game, model, mcts

MCTS_SEARCHES = 20
MCTS_BATCH_SIZE = 4

try:
    import telegram.ext
    from telegram.error import TimedOut
except ImportError:
    print("You need python-telegram-bot package installed to start the bot")
    sys.exit()

import torch

# Configuration file with the following contents
# [telegram]
# api=API_KEY
CONFIG_DEFAULT = "~/config/r1_ch18_bot.ini"

log = logging.getLogger("telegram")

class Session:
    BOT_PLAYER = game.PLAYER_BLACK
    USER_PLAYER = game.PLAYER_WHITE

    def __init__(self, model_file, player_moves_first, player_id):
        self.model_file = model_file
        self.model = model.Net(input_shape=model.OBS_SHAPE, actions_n=game.GAME_COLS)

```

```

        self.model.load_state_dict(torch.load(model_file, map_location=lambda
storage, loc: storage))
        self.state = game.INITIAL_STATE
        self.value = None
        self.player_moves_first = player_moves_first
        self.player_id = player_id
        self.moves = []
        self.mcts_store = mcts.MCTS()

    def move_player(self, col):
        self.moves.append(col)
        self.state, won = game.move(self.state, col, self.USER_PLAYER)
        return won

    def move_bot(self):
        self.mcts_store.search_batch(MCTS_SEARCHES, MCTS_BATCH_SIZE, self.state,
self.BOT_PLAYER, self.model)
        probs, values = self.mcts_store.get_policy_value(self.state, tau=0)
        action = np.random.choice(game.GAME_COLS, p=probs)
        self.value = values[action]
        self.moves.append(action)
        self.state, won = game.move(self.state, action, self.BOT_PLAYER)
        return won

    def is_valid_move(self, move_col):
        return move_col in game.possible_moves(self.state)

    def is_draw(self):
        return len(game.possible_moves(self.state)) == 0

    def render(self):
        l = game.render(self.state)
        l = "\n".join(l)
        l = l.replace("0", 'O').replace("1", "X")
        board = "0123456\n-----\n" + l + "\n-----\n0123456"
        extra = ""
        if self.value is not None:
            extra = "Position evaluation: %.2f\n" % float(self.value)
        return extra + "<pre>%s</pre>" % board

class PlayerBot:
    def __init__(self, models_dir, log_file):
        self.sessions = {}
        self.models_dir = models_dir
        self.models = self._read_models(models_dir)
        self.log_file = log_file
        self.leaderboard = {}
        self._read_leaderboard(log_file)

    def _read_models(self, models_dir):
        result = {}
        for idx, name in enumerate(sorted(glob.glob(os.path.join(models_dir,
"*.*dat")))):
            result[idx] = name
        return result

    def _read_leaderboard(self, log_file):
        if not os.path.exists(log_file):
            return
        with open(log_file, 'rt', encoding='utf-8') as fd:
            for l in fd:
                data = json.loads(l)

```

```

        bot_name = os.path.basename(data['model_file'])
        user_name = data['player_id'].split(':')[0]
        bot_score = data['bot_score']
        self._update_leaderboard(bot_score, bot_name, user_name)

def _update_leaderboard(self, bot_score, bot_name, user_name):
    if bot_score > 0.5:
        game.update_counts(self.leaderboard, bot_name, (1, 0, 0))
        game.update_counts(self.leaderboard, user_name, (0, 1, 0))
    elif bot_score < -0.5:
        game.update_counts(self.leaderboard, bot_name, (0, 1, 0))
        game.update_counts(self.leaderboard, user_name, (1, 0, 0))
    else:
        game.update_counts(self.leaderboard, bot_name, (0, 0, 1))
        game.update_counts(self.leaderboard, user_name, (0, 0, 1))

def _save_log(self, session, bot_score):
    self._update_leaderboard(bot_score, os.path.basename(session.model_file),
                             session.player_id.split(':')[0])
    data = {
        "ts": time.time(),
        "time": datetime.datetime.utcnow().isoformat(),
        "bot_score": bot_score,
        "model_file": session.model_file,
        "player_id": session.player_id,
        "player_moves_first": session.player_moves_first,
        "moves": session.moves,
        "state": session.state
    }
    with open(self.log_file, "a+t", encoding='utf-8') as f:
        f.write(json.dumps(data, sort_keys=True) + '\n')

def command_help(self, bot, update):
    bot.send_message(chat_id=update.message.chat_id, parse_mode="HTML",
                     disable_web_page_preview=True,
                     text="""
This a <a href="https://en.wikipedia.org/wiki/Connect_Four">4-in-a-row</a> game bot
trained with AlphaGo Zero method for the <a href="https://www.packtpub.com/big-data-
and-business-intelligence/practical-deep-reinforcement-learning">Practical Deep
Reinforcement Learning</a> book.

<b>Welcome!</b>

This bot understands the following commands:
<b>/list</b> to list available pre-trained models (the higher the ID, the stronger
the play)
<b>/play MODEL_ID</b> to start the new game against the specified model
<b>/top</b> show leaderboard

During the game, your moves are numbers of columns to drop the disk.
""")

def command_list(self, bot, update):
    if len(self.models) == 0:
        reply = ["There are no models currently available, sorry!"]
    else:
        reply = ["The list of available models with their IDs"]
        for idx, name in sorted(self.models.items()):
            reply.append("<b>%d</b>: %s" % (idx, os.path.basename(name)))

    bot.send_message(chat_id=update.message.chat_id, text="\n".join(reply),
                     parse_mode="HTML")

```

```

def command_play(self, bot, update, args):
    chat_id = update.message.chat_id
    player_id = "%s:%s" % (update.message.from_user.username,
update.message.from_user.id)
    try:
        model_id = int(args[0])
    except ValueError:
        bot.send_message(chat_id=chat_id, text="Wrong argumants! Use '/play
<MODEL_ID>, to start the game")
        return

    if model_id not in self.models:
        bot.send_message(chat_id=chat_id, text="There is no such model, use /list
command to get list of IDs")
        return

    if chat_id in self.sessions:
        bot.send_message(chat_id=chat_id, text="You already have the game in
progress, it will be discarded")
        del self.sessions[chat_id]

    player_moves = random.choice([False, True])
    session = Session(self.models[model_id], player_moves, player_id)
    self.sessions[chat_id] = session
    if player_moves:
        bot.send_message(chat_id=chat_id, text="Your move is first (you're
playing with O), please give the column to put your checker - single number from 0 to
6")
    else:
        bot.send_message(chat_id=chat_id, text="The first move is mine (I'm
playing with X), moving...")
        session.move_bot()
        bot.send_message(chat_id=chat_id, text=session.render(), parse_mode="HTML")

def text(self, bot, update):
    chat_id = update.message.chat_id

    if chat_id not in self.sessions:
        bot.send_message(chat_id=chat_id, text="You have no game in progress.
Start it with <b>/play MODEL_ID</b> "
                                                                    "(or use <b>/help</b> to see the
list of commands)",
                                                                    parse_mode='HTML')
        return
    session = self.sessions[chat_id]

    try:
        move_col = int(update.message.text)
    except ValueError:
        bot.send_message(chat_id=chat_id, text="I don't understand. In play mode
you can give a number "
                                                                    "from 0 to 6 to specify your
move.")
        return

    if move_col < 0 or move_col > game.GAME_COLS:
        bot.send_message(chat_id=chat_id, text="Wrong column specified! It must
be in range 0-6")
        return

    if not session.is_valid_move(move_col):

```

```

        bot.send_message(chat_id=chat_id, text="Move %d is invalid!" % move_col)
        return

    won = session.move_player(move_col)
    if won:
        bot.send_message(chat_id=chat_id, text="You won! Congratulations!")
        self._save_log(session, bot_score=-1)
        del self.sessions[chat_id]
        return

    won = session.move_bot()
    bot.send_message(chat_id=chat_id, text=session.render(), parse_mode="HTML")

    if won:
        bot.send_message(chat_id=chat_id, text="I won! Wheeee!")
        self._save_log(session, bot_score=1)
        del self.sessions[chat_id]
    # checking for a draw
    if session.is_draw():
        bot.send_message(chat_id=chat_id, text="Draw position. That's unlikely,
but possible. 1:1, see ya!")
        self._save_log(session, bot_score=0)
        del self.sessions[chat_id]

    def error(self, bot, update, error):
        try:
            raise error
        except Timeout:
            log.info("Timed out error")

    def command_top(self, bot, update):
        res = ["Leader board"]
        items = list(self.leaderboard.items())
        items.sort(reverse=True, key=lambda p: p[1][0])
        for user, (wins, losses, draws) in items:
            res.append("%20s: won=%d, lost=%d, draw=%d" % (user[:20], wins, losses,
draws))
        l = "\n".join(res)
        bot.send_message(chat_id=update.message.chat_id, text="<pre>" + l + "</pre>",
parse_mode="HTML")

    def command_refresh(self, bot, update):
        self.models = self._read_models(self.models_dir)
        bot.send_message(chat_id=update.message.chat_id, text="Models reloaded, %d
files have found" % len(self.models))

if __name__ == "__main__":
    logging.basicConfig(format="%(asctime)-15s %(levelname)s %(message)s",
level=logging.INFO)
    parser = argparse.ArgumentParser()
    parser.add_argument("--config", default=CONFIG_DEFAULT,
                        help="Configuration file for the bot, default=" +
CONFIG_DEFAULT)
    parser.add_argument("-m", "--models", required=True, help="Directory name with
models to serve")
    parser.add_argument("-l", "--log", required=True, help="Log name to keep the
games and leaderboard")
    prog_args = parser.parse_args()

    conf = configparser.ConfigParser()
    if not conf.read(os.path.expanduser(prog_args.config)):
        log.error("Configuration file %s not found", prog_args.config)

```

```

    sys.exit()

    player_bot = PlayerBot(prog_args.models, prog_args.log)

    updater = telegram.ext.Updater(conf['telegram']['api'])
    updater.dispatcher.add_handler(telegram.ext.CommandHandler('help',
player_bot.command_help))
    updater.dispatcher.add_handler(telegram.ext.CommandHandler('list',
player_bot.command_list))
    updater.dispatcher.add_handler(telegram.ext.CommandHandler('top',
player_bot.command_top))
    updater.dispatcher.add_handler(telegram.ext.CommandHandler('play',
player_bot.command_play, pass_args=True))
    updater.dispatcher.add_handler(telegram.ext.CommandHandler('refresh',
player_bot.command_refresh))
    updater.dispatcher.add_handler(telegram.ext.MessageHandler(telegram.ext.Filters.t
ext, player_bot.text))
    updater.dispatcher.add_error_handler(player_bot.error)

    log.info("Bot initialized, started serving")
    updater.start_polling()
    updater.idle()

    pass

```