ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет Санкт-Петербургская школа физико-математических и компьютерных наук

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по направлению подготовки 01.04.02 Прикладная математика и информатика
образовательная программа «Программирование и анализ данных»

Применение методов исследования среды в model-based алгоритмах обучения с подкреплением

Выполнил:
Свидченко Олег Анатольевич

Научный руководитель:
кандидат ф-м. наук, доцент департамента информатики,
Подкопаев Антон Викторович

Консультант:
старший преподаватель департамента информатики,
Шпильман Алексей Александрович

Рецензент:
Стажер-исследователь международной лаборатории экспериментальной и поведенческой экономики НИУ ВШЭ,
Сусин Иван Сергеевич

Санкт-Петербург 2021

В последние годы глубокое обучение с подкреплением продемонстрировало свою способность решать сложные многоэтапные задачи итеративного взаимодействия с окружением с целью выполнения поставленных задач. Существенным препятствием на пути к применению алгоритмов глубокого обучения с подкреплением к широкому спектру реальных задач является то, что большинство современных алгоритмов глубокого обучения с подкреплением требуют большого количества опыта взаимодействия со средой для обучения агента. Поскольку в реальных задачах получение такого опыта является дорогим и долгим, данное ограничение является значительным.

На данный момент существует несколько подходов к уменьшению требуемого количества опыта. Так, model-based обучение с подкреплением использует обучаемую модель мира, которая учится генерировать синтетический опыт для обучения агента, а методы эффективного исследования среды позволяют получать более разнообразный и новый для агента опыт, что компенсирует его меньшее количество. Как правило, в существующих исследованиях данные подходы развиваются независимо друг от друга.

Целью данной работы является улучшение существующего алгоритма model-based обучения и разработка метода исследования среды, учитывающего особенности model-based подхода. Результатом данной работы является алгоритм Maximum Entropy Dreamer. За основу которого был взят model-based алгоритм Dreamer. Для данного алгоритма был разработан ряд модификаций с целью исправления недостатков оригинального алгоритма, а также повышения его стабильности и эффективности. Также был представлен общий подход к исследованию среды в алгоритмах model-based обучения с подкреплением, основанный на принципе максимизации энтропии распределения посещаемых агентом состояний во время обучения. Полученный алгоритм Maximum Entropy Dreamer был протестирован на окружениях, использующих физический симулятор PyBullet, и продемонстрировал значительно более высокую производительность во всех использованных для тестирования окружениях, чем оригинальный алгоритм Dreamer.

Ключевые слова: машинное обучение, обучение с подкреплением, исследование среды

ОГЛАВЛЕНИЕ

Введение 5
1. Обзор литературы 8
1.1. Глубокое обучение 8
1.1.1. Mixture Density Network 8
1.2. Обучение с подкреплением 9
1.2.1. Марковский процесс принятия решений 9
1.2.2. Политика агента 11
1.2.3. Value-функция и Value Iteration 11
1.2.4. Q-функция и Q-learning 12
1.3. Глубокое обучение с подкреплением 13
1.3.1. Deep Q-Network 13
1.3.2. Актор-критик алгоритмы 14
1.4. Исследование среды в обучении с подкреплением 16
1.4.1. Алгоритмы с внутренним вознаграждением 17
1.4.2. Maximum Entropy 18
1.5. Model-based обучение с подкреплением 19
1.5.1. Dreamer 19
2. Методы 23
2.1. Недостатки алгоритма Dreamer 23
2.1.1. Предположение о бесконечном эпизоде 23
2.1.2. Совершение первого действия без наблюдения 23
2.1.3. Декодирование наблюдения 24
2.1.4. Обучаемое априорное наблюдение 24
2.1.5. Использование случайной политики вместе со случайным шумом 25
2.2. Устранение недостатков алгоритма Dreamer 25
2.2.1. Предсказание окончания эпизода 25
2.2.2. Изменение порядка формирования скрытого состояния 28
2.2.3. Изменение модели кодирования и декодирования наблюдения 29
2.3. Адаптация maximum entropy exploration к model-based RL . . . 30
2.4. Maximum Entropy Dreamer 33

3. Эксперименты 35
3.1. Физический симулятор PyBullet 35
3.2. Гиперпараметры 36
3.3. Сравнение алгоритмов 36
Заключение 39
Библиографический список 41

ВВЕДЕНИЕ

В последние годы глубокое обучение с подкреплением продемонстрировало способность эффективно решать сложные задачи взаимодействия агента со средой, при этом зачастую демонстрируя успехи, превосходящие достижения людей. Так, в 2019 году компания OpenAI продемонстрировала алгоритм OpenAI Five [1], который оказался способен играть против лучших команд в мире и побеждать в игру Dota 2. В 2019 году также была опубликована статья от компании DeepMind, в которой разработанный данной компанией метод обучения с подкреплением AlphaStar [2] смог победить мировых чемпионов в сложной стратегической игре StarCraft 2. Эти и многие другие достижения продемонстрировали потенциал обучения с подкреплением в решении задач сложного итеративного взаимодействия с окружением, менеджмента ресурсов, планирования и других.

Однако, существенным ограничением данных работ является необходимость использования большого количества опыта взаимодействия со средой для обучения агента (например, OpenAI Five использовал примерно 45000 лет симулированного времени игры), что делает методы обучения с подкреплением неприменимыми для многих практических задач, т.к. зачастую получение нового опыта взаимодействия со средой является дорогим и долгим. На данный момент существует ряд подходов, позволяющих уменьшить количество опыта взаимодействия со средой, необходимого для обучения агентов. Так, алгоритмы model-based обучения с подкреплением используют обучаемую модель мира для того, чтобы генерировать синтетический опыт, с помощью которого обучается агент, при этом сама модель мира учится предсказывать динамику среды на реальном опыте. Кроме того, алгоритмы улучшенного исследования среды также позволяют уменьшить количество необходимого опыта, т.к. мотивируют агента повышать его разнообразие и новизну, что положительно сказывается на скорости обучения агента. На данный момент алгоритмы model-based обучения с подкреплением и методы исследования среды исследуются и развиваются отдельно друг от друга. Разработка метода исследования среды, который бы учитывал особенности model-based подхода, может значительно повысить эффективность model-based алгоритмов и стать ключом к получению эффективных алгоритмов, требующих значительно меньшего количества опыта взаимодействия со средой, чем существующие методы, что позволит в дальнейшем решать более широкий

спектр практических задач при помощи алгоритмов глубокого обучения с подкреплением.

Цели и задачи

Целью данной работы является повышение эффективности алгоритма Dreamer [3] путем устранения их недостатков и улучшения исследования среды обучаемым агентом.

Задачами данной работы являются:

— Разработать модификацию алгоритма Dreamer, позволяющую данному алгоритму учитывать вероятность завершения эпизода.
— Пересмотреть архитектуру модели мира алгоритма Dreamer с учетом вероятного возникновения проблем из-за неиспользования ей начального наблюдения среды.
— Разработать модификации алгоритма, повышающие стабильность и эффективность.
— Разработать способ исследования среды, учитывающий особенности model-based подхода
— Провести сравнение оригинального и модифицированного алгоритмов

В рамках данной работы не рассмотриваются все существующие методы исследования среды, разработанные в предположении model-free алгоритмов. Все модификации, улучшения и дополнения, разработанные в ходе работы, применены к алгоритму Dreamer, т.к. на данный момент он является наиболее современным и эффективным алгоритмом model-based обучения с подкреплением. Все эксперименты будут проведены на наборе окружений физического симулятора PyBullet.

Актуальность и значимость

В отличии от большинства алгоритмов глубокого обучения, которые используют заранее собранный и обработаный набор данных, методы обучения с подкреплением требуют возможности взаимодействия с окружением в процессе обучения, что накладывает серьезные ограничения на возможность применения данных алгоритмах при решении задач, в которых взаимодействие со средой является долгим или дорогим. К таким задачам относятся как и окружения в реальном мире, так и сложные высокоточные симуляции. Таким образом, большое количество используемого опыта взаимодействия со средой для обучения

агента с помощью алгоритмов глубокого обучения с подкреплением является важным препятствием на пути к применению методов обучения с подкреплением в задачах реального мира.

Model-based подход в обучении с подкреплением является одним из перспективных и активно развивающихся способов решения данной проблемы. В частности, недавно было продемонстрировано, что алгоритмы, основанные на данном подходе, способны эффективно решать задачи обучения с подкреплением при этом используя значительно меньшее количество взаимодействий со средой.

Кроме того, алгоритмы направленные на улучшение исследования среды агентом также позволяют значительно уменьшить количество опыта взаимодействия со средой необходимого для обучения алгоритма. Такие методы, однако, разрабатываются в предположении model-free алгоритмов обучения с подкреплением, и не учитывают специфику model-based подхода.

Интеграция методов эффективного исследования среды в model-based алгоритмы обучения с подкреплением, а также улучшение существующих model-based алгоритмов могут стать ключом к значительному снижению необходимого опыта взаимодействия с окружением, что позволит в дальнейшем использовать глубокое обучение с подкреплением для решения более широкого круга задач.

Содержание работы

В главе 1 вводятся используемые в работе термины, а также рассматриваются существующие алгоритмы. Далее, в главе 2 описываются недостатки существующих алгоримтов, способы их устранения, а также предлагаемый подход к интеграции методов исследования среды в алгоритм Dreamer. Глава **??** посвящена экспериментам на основе окружений физического симулятора PyBullet, которые позволяют сравнить оригинальный и полученный в ходе работы алгоритмы. Наконец, в главе 3.3 обсуждаются полученные в ходе исследования результаты и возможные пути развития данной работы.

ГЛАВА 1. ОБЗОР ЛИТЕРАТУРЫ

1.1. Глубокое обучение

Глубокое обучение - это область машинного обучения, исследующая алгоритмы, основанные на глубоких нейронных сетях. В последние годы алгоритмы глубокого обучения стали применяться при решении широкого спектра различных задач: классификация, регрессия, компьютерное зрение, анализ естественных языков и т.д., при этом демонстрируя результаты, зачастую превосходящие классические подходы.

Нейронная сеть - это сложная функция, состоящая из блоков или слоев, каждый из которых имеет набор (возможно пустой) обучаемых параметров, который определяет функцию, задаваемую блоком или слоем. Простым примером является линейный слой $f(x) = W \cdot x + b$, где обучаемыми параметрами является матрица W и вектор смещения b. Все блоки и слои в нейронной сети являются дифференцируемым по значениям входа x и параметрам компонента, из чего следует, что выход нейронной сети также является дифференцируемым по параметрам всех входящих в нее компонент (правило сложной производной).

Как правило нейронная сеть обучается при помощи метода стохастического градиентного спуска, который заключается в том, что на каждом шаге значение параметров нейронной сети сдвигаются в противоположном от градиента некоторой дифференцируемой функции потерь направлении с некоторым шагом обучения (learning rate).

1.1.1. Mixture Density Network

Mixture Density Network [4] - это метод глубокого обучения, который приближает распределение $p(y|x)$ с помощью смеси нормальных распределений за счет минимизации дивергенции Кульбака-Лейблера $D_{\mathrm{KL}}(p(y|x)||\hat{p}(y|x))$ (где $\hat{p}(y|x)$) - приближение) между реальным распределением и реальным. Полная запись функции потерь для Mixture Density Network равна:

$$\begin{aligned} L =& \mathbb{E}_x D_{\mathrm{KL}}(p(y|x)||\hat{p}(y|x)) \\ =& \mathbb{E}_x \mathbb{E}_{y|x} \log \frac{p(y|x)}{\hat{p}(y|x)} \\ =& \mathbb{E}_x \mathbb{E}_{y|x} [-\log \hat{p}(y|x) + \log p(y|x)] \end{aligned}$$

Однако поскольку реальное распределение не зависит от параметров модели, а следовательно его градиент по параметрам модели равен 0, то второе слагаемое

можно исключить. В таком случае итоговая функция потерь равна:

$$L = \mathbb{E}_x \mathbb{E}_{y|x} - \log \hat{p}(y|x)$$

А поскольку математические ожидания также не зависят от параметров модели, то градиент по параметрам модели θ равен:

$$\nabla_\theta L = \mathbb{E}_x \mathbb{E}_{y|x} \nabla_\theta - \log \hat{p}(y|x)$$

Что позволяет приблизить его с помощью метода Монте Карло, используя для этого имеющийся набор данных.

1.2. Обучение с подкреплением

Обучение с подкреплением - это область машинного обучения, которая изучает способы взаимодействия агента со средой посредством итеративного совершения действий, что приводит к изменению внутреннего состояния среды и получению агентом награды. Такое взаимодействие, как правило, делится на эпизоды, и может быть ограничено временем или некоторыми условиями (например, в случае физического симулятора это может быть падение или поломка робота), при достижении которых эпизод завершается. Задачей, которую решает агент при взаимодействии со средой, является максимизация суммарной награды за весь эпизод. В простом случае формализовать данную задачу можно при помощи марковского процесса принятия решений (Markov decision process, MDP).

1.2.1. Марковский процесс принятия решений

Марковский процесс принятия решений определяется как набор (S, A, T, R, d), где:

- S - это множество возможных внтуренних состояний среды. Зачастую оно считается или конечным (дискретным), или континуальным.
- A - множество действий, которое может совершить агент. Как и множество состояний, может быть как конечным, так и континуальным.
- $T : S \times A \to S$ - функция перехода, определяющая, в какое состояние перейдет среда после совершения агентом определенного действия в заданном состоянии. Кроме того, эта функция может задавать не отображение

в множество состояний, а распредление или плотность веростнояти над множеством состояний, в которые может перейти среда.

— $R : S \times A \times S \to \mathbb{R}$ - функция награды, которая определяет, какую награду получит агентпри совершении определенного перехода. В некоторых формулировках может зависеть только от текущего состояния и действия, от следующего сосотояния или от текущего и следующего состояний.

— $d : S \to \{0, 1\}$ - функция, которая определяет, завершился эпизод взаимодействия со средой или нет. В данной формулировке завершение эпизода происходит только при достижении определенных состояний, однако на практике взаимодействие может завершаться по истечении опредленного времени.

Кроме того, в формулировке неявно присутствует распределение начальных состояний, в которых агент начинает взаимодействие со средой. Также стоит отметить то, что алгоритмами обучения с подкреплением зачастую решается задача не максимизации суммарной награды напрямую, а более широкая задача максимизация дисконтированной суммарной награды $\sum_{i=0}^{T} \gamma^i R(s_i, a_i, s_{i+1}$ при $0 < \gamma \leqslant 1$. Такая формулировка позволяет избежать ситуации, когда эпизод с максимальной наградой может быть сколь угодно длинным. Простой пример такой ситуации: среда, в которой есть циклический переход между состояниями s_0 и s_1, а также переход в терминальное состояние s_2 с наградой $+1$. Агент всегда получит награду $+1$, однако может перемещаться между сосотяниями s_0 и s_1 бесконечно долго.

Обобщением обычного Марковского процесса принятия решений является частично наблюдаемый марковский процесс принятия решений (partially observable Markov decision process, POMDP), который отличается от обычного MDP тем, что в каждый момент времени агент не имеет полной информации о текущем состоянии окружения и вынужден принимать решения основывась лишь на некотором наблюдении $o \in O$, которое зависит от текущего состояния s. В качестве примера такого окружения можно привести управление транспортным средством, где агент может наблюдать лишь ограниченную часть пространства вокруг себя, при этом не имея никакой информации о том, что происходит в ”слепых зонах”(скрыто за другими транспортными средствами, зданиями или просто не попадает в поле зрения камеры). Как правило, большая часть реальных или сложных окружений являются частично наблюдаемыми.

1.2.2. Политика агента

Политика $\pi : O \to A$ - это правило, которым руководствуется агент при взаимодействии со средой. Она может быть как детерминированно и отображать состояние в соответствующее действие, так и стохастической, задавая распредление над пространством действий A при условии текущего наблюдения из множества O. Поскольку обучение с подкреплением решает задачу взаимодействия агента со средой посредством задания политики агента, то оптимальной политикой можно назвать

$$\pi^*(a|o) = \arg\max_{\pi} \mathbb{E}_{\tau|\pi} \sum_{i=0}^{T} \gamma^i R(o_i, a_i, o_{i+1})$$

, где τ - это траектория, состоящая из переходов (s_i, a_i, s_{i+1}), в которых действие получено в соответствии с политикой агента, а наблюдение o_i соответствует состоянию s_i.

Стоит отметить, что обычно функции T, R и D считаются неизвестными, равно как и зависимость наблюдения от состояния. Это значит, что при решении задачи алгоритм обучения с подкреплением не может напрямую использовать эти функции, и вынужден использовать только опыт взаимодействия со средой.

1.2.3. Value-функция и Value Iteration

Одним из первых методов, решавших задачу обучения с подкреплением, был алгоритм Value Iteration [5], который основывается на так называемой value-функции:

$$V_\pi(s) = \mathbb{E}_{\tau|\pi, s_0=s} \sum_{i=0}^{T} \gamma^i R(o_i, a_i, o_{i+1})$$

Value-функцию можно записать в реккурентном виде:

$$V_\pi(s) = \mathbb{E}_{s' \sim T(s,a), a \sim \pi(.|s)} R(s, a, s') + \gamma \cdot (1 - D(s')) \cdot V_\pi(s')$$

Алгоритм Value Iteration использует эту реккурентную запись для того, чтобы итеративно приближать value-функцию, перебирая все возможные переходы в среде, и выбирая в процессе оптимизации действие, которое максимизирует матемаческое ожидание получаемой награды:

$$\hat{V}(s) \longleftarrow \max_a [\mathbb{E}_{s' \sim T(s,a)} R(s, a, s') + \gamma \hat{V}(s')]$$

После того, как процесс сошелся, или по прохождении максимального количества шагов оптимизации, политика определяется как

$$\pi^*(s) = \arg\max_a [\mathbb{E}_{s' \sim T(s,a)} R(s,a,s') + \gamma \cdot (1 - D(s')) \cdot \hat{V}(s')]$$

В случае конечного пространства состояний и действий, существуют теоретические гарантии того, что данный алгоритм сойдется к оптимальной политике. Недостатком данного алгоритма, однако, является то, что для его использования необходимо, чтобы пространство состояний и пространство действий были конечными, а также неободимо иметь возможность использовать $R(s,a,s')$ и $T(s,a)$, что не выполняется для большинства сред, т.к. обычно динамику среды принято считать "черным ящиком что означает, что $R(s,a,s')$ и $T(s,a)$ неизвестны.

1.2.4. Q-функция и Q-learning

В отличии от value-функции, Q-функция отражает зависимость получаемой суммарной награды от пары (s,a):

$$Q_\pi(s,a) = \mathbb{E}_{\tau|\pi, s_0=s, a_0=a} \sum_{i=0}^{T} \gamma^i R(o_i, a_i, o_{i+1})$$

Также, как и в случае value-функции, Q-функция имеет реккурентную запись:

$$Q_\pi(s,a) = \mathbb{E}_{s' \sim T(s,a)} R(s,a,s') + \gamma \mathbb{E}_{a' \sim \pi(.|s')} Q_\pi(s',a')$$

Важно отметить, что в отличии от value-функции для построения оптимальной политики при наличии Q-функции не нужно использовать функцию перехода или функцию награды: $\pi^*(s) = \arg\max_a Q_{\pi^*}(s,a)$. При этом оценку Q-функции можно построить итеративно используя уравнение Беллмана:

$$\hat{Q}(s,a) = r + \gamma \cdot (1-d) \cdot \max_{a'} \hat{Q}(s',a')$$

Где (s,a,s') - переход в среде, а $r = R(s,a,s')$ и $d = D(s')$ получены при взаимодействии со средой (т.е. без непосредственного применения данных функций).

В итоге алгоритм Q-learning [5] на протяжении T итераций обновляет существующую оценку Q-функции $\hat{Q}$, используя для этого опыт взаимодействия

со средой, т.е. для обновления оценки он совершает переход из текущего состояния s в новое состояние s' совершая при этом действие a и получая за это награду r и информацию о завершении эпизода d. Если эпизод завершился ($d = 1$), то на следующем шаге алгоритма будет начат новый эпизод (состояние s будет выбрано случайно из распределения начальных состояний), иначе эпизод будет продолжен ($s = s'$).

Однако если при обучении агент будет все время следовать жадной политике ($\pi(s) = \arg\max_a \hat{Q}(s,a)$), то алгоритм может сойтись к субоптимальной политике, т.к. будет большое количество возможных траекторий в среде, которые агент никогда не посетит из-за того, что будет считать субоптимальные траектории лучше, чем непосещенные траектории. Чтобы этого избежать, агент может совершать случайные действия с заданной вероятностью ε, тем самым исследуя ранее неизвестные части пространства переходов. Данный подход теоретически гарантирует, что алгоритм сойдется к оптимальной политике в случае конечного пространства состояний и конечного пространства действий.

1.3. Глубокое обучение с подкреплением

Существенным ограничением рассмотренных в разделе 1.2 алгоритмов Q-learning и Value iteration является условие на конечность пространства состояний и пространства действий, что в значительной степени сужает круг сред, решаемых данными алгоритмами. Как правило, сложные среды имеют континуальное пространство состояний высокой размерности (например, изображения) и континуальное пространство действий (например, вектор ускорения различных частей механизма). Для устранения данного ограничения современные алгоритмы обучения с подкреплением используют нейронные сети в связи с их способностью точно приближать сложные функции.

1.3.1. Deep Q-Network

Одним из первых алгоритмов глубокого обучения с подкреплением, с которого началось активное развитие данной области, является алгоритм Deep Q-Network [6] (DQN). Данный метод использует нейронную сеть для приближения значения Q-function для каждого возможного действия, т.е. принимает на вход $s \in S$ (может быть вектором или тензором) и возвращает вектор размера $|A|$, где компонента с индексом i соответствует приближению значения Q-функции для действия с индексом i. Использование данной архитектуры поз-

воляет определить оптимальную с учетом имеющегося приближения политику как $\pi(s) = \arg\max_a \hat{Q}(s,a)$.

Как правило для обучения нейронных сетей при подсчете градиента используются минибатчи для того, чтобы приблизить математическое ожидание градиента по распределению данных. В случае DQN аналогом набора данных, из которого случайным образом выбирается минибатч данных, является буфер опыта. Данный буфер заполняется опытом, получаемым во время взаимодействия агента со средой. Затем, на каждом шаге алгоритма, из него собирается новый минибатч данных, который затем используется для обновления приближения Q-функции. Мотивацией к использованию буфера опыта вместо непосредственного получения данных из среды является то, что данные, полученные во время взаимодействия со средой, являются сильно скореллированными, что приводит к некорректной оценки математического ожидания градиента.

Кроме того, алгоритм использует дополнительную целевую нейронную сеть для того, чтобы стабилизировать рекуррентную оценку Q-функции $Q(s,a) = \mathbb{E}_{r,s',d} r + \gamma \cdot (1-d) \cdot \max_{a'} Q(s',a')$, т.к. при использовании обновляемой нейронной сети в качестве оценки Q-функции для следующего состояния значение целевой функции для одного и того же набора (s,a,r,s',d) будет постоянно меняться, что приведет к нестабильному обучению агента. В итоге, нейронная сеть, приближающая Q-функцию, обучается с помощью функции потерь MSE, используя рекуррентное приближение Q-функции в качестве значений целевой функции:

$$L = \mathbb{E}_{(s,a,s',r,d)\sim \mathrm{B}} (Q_\theta(s,a) - (r + \gamma \cdot (1-d) \cdot \max_{a'} Q_{\theta_T}(s',a')))^2$$

Где $Q_\theta(s,a)$ - приближение Q-функции обновляемой нейронной сетью, $Q_{\theta_T}(s',a'))$ - приближение Q-функции целевой нейронной сетью, а B - буфер опыта. Параметры целевой нейронной сети приравниваются параметрам обновляемой нейронной сети раз в некоторое количество (задается в качестве гиперпараметра) обновлений обновляемой сети.

1.3.2. Актор-критик алгоритмы

Ограничением Deep Q-Network является применимость только в случае дискретного пространства действий, т.к. используемая для приближения Q-функции нейронная возвращает значение Q-функции для каждого возможного

действия. Для того, чтобы избавиться от данного ограничения, был изобретен подход, в котором обучаются две модели: актор и критик. Общая идея заключается в том, что критик приближает Q-функцию или Value-функцию, а актор с помощью данного приближения выучивает политику, которая максимизирует математическое ожидание суммарной дисконтированной награды, и в зависимости от алгоритма может быть детерминированным или стохастическим.

Одним из алгоритмов, использующих данный подход, является алгоритм Advantage Actor-Critic [**A2C**] (A2C). Данный метод использует нейронную сеть актора для получения распределения дествий политики и нейронную сеть критика для приближения value-функции, который принимает на вход пару из состояния и действия, и возвращает одно число, соответствующее оценке value-функции. Актор для обучения использует функцию преимущества (advantage function)

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

Данная функция отражает, насколько больше награды получит агент, если совершит действие a в состоянии s, чем если бы он следовал текущей политике π. A2C приближает значение функции преимущества как

$$\hat{A}_\pi(s,a) = (r + \gamma \cdot \hat{V}_\pi(s')) - \hat{V}_\pi(s)$$

Поскольку данное выражение функции преимущество не дифференцируемо по действию (а следовательно и по параметрам нейронной сети актора), для подсчета градиента функции потерь актора в данном алгоритме используется так называемый log-derivative trick, который позволяет выразить градиент функции потерь актора как

$$\nabla_\theta L = \mathbb{E}_{(s,a,s',r,d)\in \mathrm{B}} \hat{A}_{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s)$$

Данный алгоритм использует функцию потерь MSE для обучения критика, как и DQN.

Другим представителем данного класса алгоритмов является алгоритм Deep Deterministic Policy Gradient [7] (DDPG). В отличии от A2C, данный метод использует критика для приближения Q-функции и, поскольку такое приближение Q-функции дифференцируемо по действию, обучает актора выдавать действие, которое минимизирует приближенное значение Q-функции. Данный

алгоритм также, как и DQN, использует целевые сети для актора и критика, однако обновляет их с помощью плавных обновлений после каждого шага градиентного спуска:

$$\theta_{\text{T}} = \tau\theta + (1-\tau)\theta_{\text{T}}$$

Также существует множество других алгоритмов, использующих архитектуру актор-критик (PPO [8], TRPO [9], TD3 [10], SAC [11]), некоторые из которых используют стохастическую политику, а некоторые - детерминированную. Важность этого различия заключается в том, что при использовании стохастической политики как правило необходимо вводить дополнительную регуляризацию энтропии распределения действий, иначе политика стремится выродиться в жадную детерминированную политику, которая скорее всего будет субоптимальной. При этом использование дополнительного случайного шума или случайных действий в ряде алгоритмов может привести к расхождению процесса градиентного спуска в следствии слишком большого по модулю значения логарифма плотности вероятности совершить определенное действие.

1.4. Исследование среды в обучении с подкреплением

Проблема исследования среды алгоритмами обучения с подкреплением появилась вместе с появлением алгоритмов, обучающихся при помощи взаимодействия со средой, и заключается в максимизации релевантности получаемого опыта от данного взаимодействия. Это важно, поскольку имея какую-то политику, мы не можем заведомо знать, существует ли политика, которая решала бы задачу эффективней, чем уже имеющаяся. Особенно остро данная проблема проявляется в средах, в которых для достижения положительного результата нужно совершить сложную последовательность строго определенных действий.

Существует несколько классических подходов к исследованию среды, которые помогают агенту получать новый опыт в простых случаях. Так, в алгоритмах Q-learning и Deep Q-Network используется подход ε-жадного исследования среды, который заключается в совершении случайного действия с вероятностью ε и совершением жадного действия с вероятностью $1-\varepsilon$. Такой подход гарантирует сходимость алгоритма в случае конечного пространства состояний, однако является не самым эффективным из возможных. В случае континуального пространства действий широко распространено использование случайного шума при использовании детерминированных политик и регуляризация энтропии распределения совершаемых политикой действий в случае использования

стохастической политики. Так, алгоритм DDPG добавляет несмещенный случайный нормальный шум со стандартным отклонением ε, а Soft Actor-Critic добавляет $-\log \pi(a|s)$ с некоторым коэффициентом к награде за совершение действия.

1.4.1. Алгоритмы с внутренним вознаграждением

Несмотря на то, что стандартные методы исследования среды подходят для решения задач широкого спектра сред, их часто не достаточно в случае, когда для достижения оптимального результата требуется совершение длинной последовательности строго определенных действий, или же когда количество допустимых взаимодействий со средой ограничено, и их необходимо использовать как можно эффективнее.

Одним из подходов к улучшению исследования среды агентом является дополнительное вознаграждение агента за нахождение новых, ранее не наблюдавшихся состояний или переходов в среде. Одним из методов определения новизны, который предлагает считать количество посещений состояния или пары состояние-дейсвия, и чем реже было посещено состояние ранее, тем большую награду давать агенту за его посещение. В наивном варианте такой подход применим только в случае конечного пространства состояний, однако существуют попытки [12, 13] обобщить его на континуальное пространство состояний или на пространство состояний большой размерности.

Другим методом определения новизны полученной информации является использование ошибки предсказания модели в качестве внутреннего вознаграждения. Так, алгоритм Random Network Distillation [14] (RND) обучает малую нейронную сеть, принимающую на вход состояние, повторять выходы большой случайно инициализированной нейронной сети, а ошибку предсказания выходов большой нейронной сети использует для задания внутренней награды. Данный метод предполагает, что чем чаще посещается определенное состояние, тем меньше будет ошибка предсказания, и наоборот, чем оно посещается реже, тем больше будет ошибка. Алгоритм Curiosity Driven Exploration [15], в отличии от RND, использует модель, которая предсказывает динамику среды:

$$L = (MSE)(g_\theta(\phi(s), a), \phi(s'))$$

Где ϕ - это некоторая функция, которая кодирует состояние, и может быть задана разными способами. Существует множество других алгоритмов исследования среды [16, 17], использующих ошибку предсказания в качестве внутренней награды для повышения эффективности исследования среды, однако большинство из них так или иначе идейно схоже с Curiosity Driven Exploration.

Стоит отметить, что алгоритмы исследования среды, использующие внутреннее вознаграждение, разработаны без каких-либо дополнительных предположений об алгоритме обучения с подкреплением, с помощью которого решается задача, в следствии чего данные методы не учитывают особенности алгоритмов обучения с подкреплением, вместе с которыми используются.

1.4.2. Maximum Entropy

В 2019 году был представлен метод [18] исследования среды, основанный на максимизации энтропии распределения посещаемых агентом состояний. Более того, было показано, что при использовании данного метода в случае конечного пространства действий и конечного пространства состояний для получения политики π такой, что $R(p_\pi(s)) \geqslant \max_\pi R(p_\pi(s)) - \varepsilon$, требуется не более, чем $O(|S|^2 \cdot |A| \cdot \text{poly}(\beta, \varepsilon^{-1}))$ переходов в среде в предположении, что у нас есть оракул, который приближает распределение посещаемых политикой состояний, и оракул, который для любой функции награды R и $\varepsilon > 0$ способен построить такую политику π, что $V_\pi \geqslant \max_\pi V_\pi - \varepsilon$. Очевидно, что условие на наличие данных оракулов, а так же наличие гарантий только в случае дискретных сред ограничивает применимость данного алгоритма. Тем не менее, в данной статье показано, что максимизация энтропии распределения посещаемых агентом состояний позволяет в значительной степени повысить эффективность исследования среды агентом обучения с подкреплением. Также важным является тот факт, что в работе распределение посещаемых агентом состояний вводится как дисконтированная смесь распределений состояний, посещаемых агентом на каждом конкретном шаге:

$$p_\pi^\gamma(s_{t+}|s_t) = (1-\gamma) \cdot \sum_{i=0}^{\infty} \gamma^i p_\pi(s_{t+i+1}|s_t) \tag{1.4.1}$$

Кроме того, стоит отметить очевидную связь с count-based подходами, описанными в предыдущем подразделе. В случае конечного пространства со-

стояний оба эти подхода фактически имеют общую точку оптимума, в котором распределение посещаемых агентом состояний является равномерным.

1.5. Model-based обучение с подкреплением

Алгоритмы, описанные в разделе 1.3, учат агента взаимодействовать со средой основываясь только на полученном ранее агентом опыте без использования какой-либо дополнительной информации о том, как устроена среда. В противовес такому подходу, алгоритмы model-based обучения с подкреплением предполагают наличие данной информации о динамике среды, что значительно облегчает задачу обучения агента. Для большинства окружений, однако, модель мира не известна, поэтому современные методы model-based обучения с подкреплением приближают ее с помощью нейронных сетей.

1.5.1. Dreamer

Одним из последних разработанных алгоритмов model-based обучения с подкреплением является алгоритм Dreamer [3], взявший за основу архитектуру модели мира из алгоритма PlaNet [19]. Данный метод состоит из двух основных частей: модели мира и агента. Агент также, как и в других алгоритмах обучения с подкреплением, учится максимизировать математическое ожидание суммарной дисконтированной награды, однако использует для этого не реальный опыт из окружения, а синтезированный с помощью модели мира, а также использует получаемые моделью мира скрытые представления состояния среды во время взаимодействия с настоящим окружением.

В Dreamer модель мира (рисунок 1.5.1) также состоит из нескольких частей. В процессе обучения она учится строить скрытое представление состояния, состоящее из детерминированной части h_i и стохастической части z_i. Детерминированная часть состояния строится рекуррентно при помощи рекуррентных нейронных сетей (в частности архитектуры GRU), которая принимает на вход детерминированную часть предыдущего состояния и преобразованное с помощью полносвязной нейронной сети вектор, являющийся конкатенацией предыдущего состояния и совершенного агентом действия. Стохастическая часть может быть получена из приближаемого моделью мира априорного распределения $p(z_i|h_i)$ или апостериорного распределения $p(z_i|h_i, o_i)$. Первое распределение применяется в случае, когда модель используется для генерации синтетических траекторий для обучения агента, второе распределение используется во время обучения модели мира или в процессе взаимодействия агента

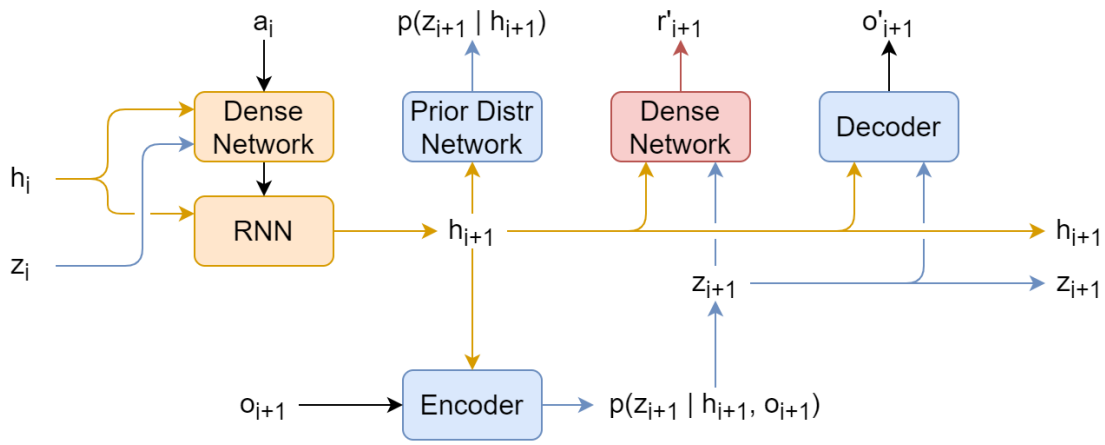

Рисунок 1.5.1 – Схема модели мира алгоритма Dreamer. Оранжевый цвет на схеме соответствует части модели, отвечающей за детерминированную составляющую h скрытого состояния. Синий цвет на схеме соответствует части модели, отвечающей за стохастическую составляющую z скрытого состояния. Красный цвет соответствует части модели, отвечающей за предсказание награды, полученной за совершение перехода в среде.

с реальным окружением. Модель мира обучается таким образом, чтобы минимизировать адаптированный ELBO-loss и тем самым приблизить распределения $p(o_i|z_i, h_i)$ (реконструкция наблюдения) и $p(r_i|z_i, h_i)$ (распределение получаемых наград за переход в состояния (z_i, h_i)). Для подсчета функции потерь и ее градиента используется случайные отрезки траекторий, полученных во время взаимодействия с настоящей средой и сохраненные в буфере опыта.

Для обучения агента используются данные, синтезированные при помощи модели мира. В качестве начальных скрытых состояний для построения синтетических траекторий используются скрытые состояния, полученные из отрезков реальных траекторий, использовавшихся для обновления модели мира. Поскольку модель мира дифференцируема, т.к. представлена нейронной сетью, есть возможность оптимизировать суммарную награду напрямую, посчитав градиент по совершенным агентом действиям. Однако для оценки суммарной награды за весь эпизод пришлось бы синтезировать очень длинные траектории, что было бы слишком затратно и, кроме того, привело бы к значительным ошибкам в оценке, поскольку с увеличением длины синтезируемой траектории ошибка предсказания накапливается. Чтобы этого избежать, авторы предлагают использовать λ-returns - метод оценки суммарной награды на основе последовательности переходов:

$$TD_\lambda = (1-\lambda)\sum_{k=1}^{n}\lambda^k[\gamma^k \cdot V_\pi(s_k) + \sum_{i=0}^{k-1}\gamma^i r_i] + \lambda^n[\gamma^n \cdot V_\pi(s_n) + \sum_{i=0}^{n-1}\gamma^i r_i] \quad (1.5.1)$$

При этом также, как и любые другие алгоритмы обучения с подкреплением, использующие актор-критик архитектуру, агент использует две нейронные сети: актора, отвечающего за политику, и критика, приближающего value-функцию. Однако в отличии от алгоритмов A2C и PPO, которым из-за недифференцируемого по действиям способа оценки advantage-функции приходится использовать log-derevative trick для того, чтобы обучать актора, агент в Dreamer берет действия из распределения, генерируемого актором, и, используя reparametrization trick, совершает градиентный спуск напрямую по оценке суммарной награды. Это является возможным, поскольку все награды и состояния, используемые при подсчете оценки, являются дифференцируемыми по совершенным агентом действиям. Стоит отметить, что при обновлении актора оценки суммарной награды для всех состояний в синтетических траекториях усредняются. При обучении критика полученные в ходе обновления актора оценки суммарных наград используются в качестве значений целевой функции для подсчета функции ошибки.

В итоге алгоритм на каждой своей итерации совершает следующую последовательность шагов:

а) Получить новую траекторию из окружения, используя текущую политику агента с дополнительным случайным зашумлением действий для исследования среды
б) Добавить полученную последовательность в буфер опыта
в) Выполнить n обновлений следующим образом:
    1) Получить минибатч отрезков траекторий из буфера опыта
    2) При помощи минибатча и модели мира посчитать скрытые состояния (z_i, h_i)
    3) Посчитать функцию потерь ELBO для модели мира используя (z_i, h_i) и обновить модель мира при помощи градиентного спуска.
    4) Сгенерировать синтетические траектории, используя (z_i, h_i) в качестве начальных состояний.

5) Использовать синтетические траектории для оценки суммарной награды, используя метод λ-returns.
6) Обновить актора и критика при помощи посчитанных оценок суммарной награды.

Для исследования среды алгоритмом Dreamer используется случайный шум, аддитивно добавляемый к совершаемым агентом действиям. При этом важно отметить, что несмотря на то, что политика актора является стохастичной, ее энтропия не регуляризируется так, как это обычно принято делать в алгоритмах глубокого обучения с подкреплением с использованием обучаемой стохастической политики (A2C, PPO, TRPO, SAC).

Недавно было показано, что данный алгоритм может быть не менее эффективен, чем model-free алгоритмы обучения с подкреплением, при этом требуя значительно меньшего количества взаимодействий со средой для обучения. Кроме того, существуют также вариации алгоритма с альтернативными способами построения представления наблюдений, что также может влиять на эффективность решения определенных окружений с помощью алгоритма Dreamer.

ГЛАВА 2. МЕТОДЫ

2.1. Недостатки алгоритма Dreamer

Несмотря на то, что алгоритм Dreamer уже продемонстрировал свою эффективность при решении различных задач обучения с подкреплением, при его разработке авторами был допущен ряд изъянов, которые потенциально могут привести к ошибкам и меньшей производительности. В данном разделе разобраны те недочеты, которые были выявлены в ходе анализа архитектуры алгоритма Dreamer.

2.1.1. Предположение о бесконечном эпизоде

Оригинальный алгоритм Dreamer, разработанный авторами, действует в предположении бесконечно продолжительного взаимодействия со средой, т.е. не рассматривает вероятность завершения эпизода до истечения времени эпизода. Однако на практике зачастую такое предположение оказывается не верным, поскольку во многих средах обучения с подкреплением существуют определенные условия, при достижении которых эпизод прерывается досрочно. Так, при решении задач управления различными роботами в физических симуляциях таким условием является достижение роботом положения, в котором он будет не способен выполнить поставленную задачу, которое привело бы к возможности повреждению робота в реальном мире или в котором задача, поставленная перед роботом, считается успешно выполненной.

2.1.2. Совершение первого действия без наблюдения

Модель мира (рис. 1.5.1), предлагаемая авторами оригинального алгоритма Dreamer, использует информацию о наблюдении для обновления текущего состояния только после того, как агент совершит какое-либо действие, а изначально скрытое состояние инициализируется вектором, состоящим из 0. У такого решения есть два существенных недостатка.

Первый недостаток заключается в том, что действие в начале эпизода всегда совершается ”вслепую т.е. без какой-либо информации о текущем состоянии среды. Однако существует ряд сред, в которых от каждого действия зависит очень много, и совершение даже одного неправильного действия может существенно повлиять на исход эпизода.

Второй недостаток заключается в том, что при обучении начальное состояние в отрезке траектории, для которого строится предсказание, также инициализируется вектором, состоящим из 0, что приводит к, во-первых, менее точным

предсказаниям в начале отрезка траектории, а во-вторых, к тому, что модель мира не сопоставляет скрытое состояние, состоящие из 0, с началом эпизода, что еще больше увеличивает вероятность возникновения первой проблемы.

2.1.3. Декодирование наблюдения

При кодировании наблюдения с целью получения распределения $p(z_t|o_t, h_t)$ и декодировании текущего скрытого состояния модели мира с целью восстановления оригинального наблюдения $p(\hat{o}_t|z_t, h_t)$ используется не только стохастическая часть скрытого состояния, которая регуляризуется с помощью дивергенции Кульбака-Лейблера в функции потерь ELBO, но также и детерминированная часть скрытого состояния, на которую ELBO не накладывает никаких дополнительных ограничений. На практике, в следствии отсутствия данных ограничений, это может привести к излишне информативной (с точки зрения восстановления наблюдения) детерминированной части скрытого состояния при недостаточно информативной стохастической части состояния, т.е. большая часть информации, содержащейся в детерминированной части скрытого состояния будет использоваться для более точного восстановления наблюдения и при этом может не содержать достаточной информации, необходимой для приближения функции награды.

2.1.4. Обучаемое априорное наблюдение

Алгоритм Dreamer при обучении модели мира приближает распределение $p(z_t|o_t, h_t)$, что позволяет кодировать с учетом детерминированной части скрытого состояния полученное из среды наблюдение в случайный вектор, являющийся стохастической частью скрытого состояния. В то же время, модель мира приближает распределение $p(z_t|h_t)$, которое является априорным распределением в функции потерь ELBO и используется для генерации стохастической части скрытого состояния при использовании модели мира для генерации синтетических траекторий. Проблема обучаемого априорного распределения заключается в том, что совместная оптимизация априорного и апостериорного распределений может привести к нестабильному обучению из-за возможного появления большого (по модулю) смещения априорного распределения, а также слишком большого или слишком маленького значения стандартного отклонения.

2.1.5. Использование случайной политики вместе со случайным шумом

Классические model-free алгоритмы глубокого обучения с подкреплением, основанные на актор-критик архитектуре, как правило используют или стохастическую политику, или случайный шум для исследования среды. В случае использования стохастической политики, ее использование может быть мотивировано особенностями алгоритма (A2C, PPO, TRPO) или использованием ее для лучшего исследования среды (SAC), однако каждый из данных алгоритмов применяют регуляризацию энтропии распределения действий, выдаваемого политикой, чтобы предотвратить вырождение политики в детерминированную. Однако в алгоритме Dreamer такой регуляризации нет, что приводит к вырождению политики в детерминированную, что, однако, не мешает агенту исследовать среду за счет добавляемого случайного шума.

2.2. Устранение недостатков алгоритма Dreamer

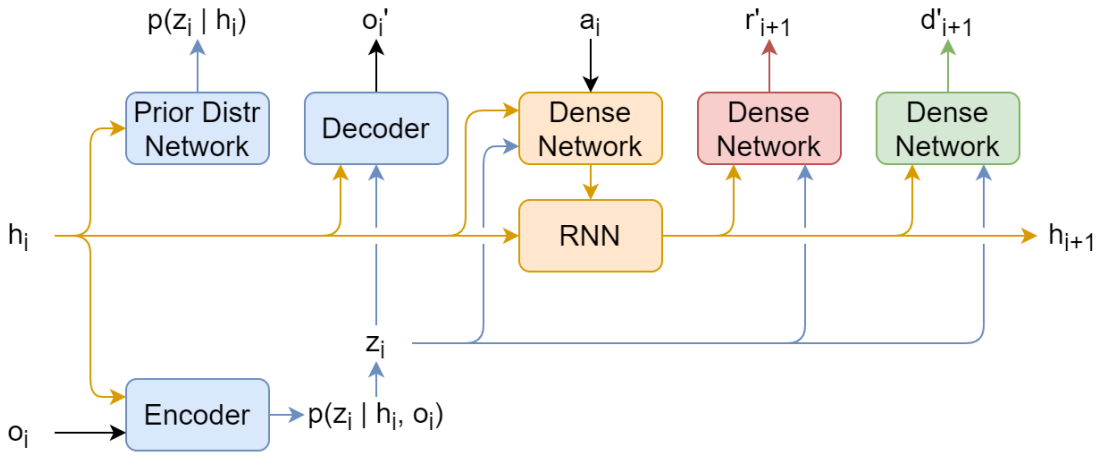

Рисунок 2.2.1 – Схема измененной модели мира алгоритма Dreamer. Оранжевый цвет на схеме соответствует части модели, отвечающей за детерминированную составляющую *h* скрытого состояния. Синий цвет на схеме соответствует части модели, отвечающей за стохастическую составляющую *z* скрытого состояния. Красный цвет соответствует части модели, отвечающей за предсказание награды, полученной за совершение перехода в среде. Зеленый цвет соответствует части модели, отвечающей за предсказание вероятности завершения эпизода взаимодействия со средой.

2.2.1. Предсказание окончания эпизода

Как было сказано в разделе 2.1, то, что модель мира не берет в расчет вероятность завершения эпизода является значительным недостатком алгоритма,

т.к. в ряде сред может привести к формированию субоптимальной политики у агента. Для того, чтобы исправить данный изъян оригинальной архитектуры модели мира, в архитектуру был добавлен блок (на рис. 2.2.1 отображен зеленым цветом), отвечающий за определение вероятности того, что текущее состояние является терминальным, т.е. эпизод завершится после перехода в данное состояние. Этот дополнительный блок представляет из себя полносвязную нейронную сеть с промежуточной функцией активации ReLU после скрытых слоев и функцией активации Sigmoid после выходного слоя. Данный бок принимает на вход скрытое представление состояния (z_i, h_i), которое строит модель мира, и возвращает приближение вероятности $p(d_i|z_i, h_i)$ того, что состояние является терминальным. Для обучение данного блока используется перекрестная энтропия в качестве функции потерь, которая аддитивно добавляется к функции потерь модели мира, что позволяет обучать данный блок совместно с остальными частями модели мира.

Поскольку ошибки из-за неучтенной вероятности завершения эпизода возникают именно при обучении агента, т.к. во время обучения он получает некорректную оценку ожидаемой суммарной награды, необходимо модифицировать процесс его обучения с учетом данной вероятности. В частности, для этого нужно исправить оценку ожидаемой суммарной награды, которую получит агент в ходе синтетической траектории, сгенерированной при помощи модели мира.

Формула ожидаемой суммарной награды (value функции) для состояния s_i в среде с учетом вероятности завершения эпизода выглядит следующим образом:

$$V_\pi(s_i) = \begin{cases} \mathbb{E}_{a_i \sim \pi(.|s_i), s_{i+1} \sim T(s_i, a_i), r_i \sim R(s_{i+1})}(r_i + \gamma V_\pi(s_{i+1})) & \text{если } d_i = 0 \\ 0 & \text{иначе} \end{cases}$$

Или, если записать данную формулу как математическое ожидание по d_i:

$$V_\pi(s_i) = (1 - p(d_i|s_i)) \cdot \mathbb{E}_{a_i \sim \pi(.|s_i), s_{i+1} \sim T(s_i, a_i), r_i \sim R(s_{i+1})}(r_i + \gamma V_\pi(s_{i+1}))$$

Данную формулу можно рекуррентно развернуть на n шагов вперед, что позволит получить математическое ожидание суммарной дисконтированной награды по n последовательным переходам в среде, начавшимся состояния s_i, где

каждый следующий переход с вероятностью $p(d_{i+c}|s_{i+c}$ приводит к завершению эпизода:

$$V_\pi(s_i) = \mathbb{E}_\tau \sum_{k=0}^{n-1} (\gamma^k \cdot r_{i+k} \cdot \prod_{j=0}^{k} (1 - p(d_{i+j}|s_{i+j}))) + \gamma^n \cdot V_\pi(s_{i+n}) \prod_{j=0}^{n-1} (1 - p(d_{i+j}|s_{i+j})) \tag{2.2.1}$$

Где τ - это траектория, начавшаяся в состоянии s_i и полученная в соответствии с политикой π. Важно отметить, что такое математическое ожидание можно приблизить с помощью метода Монте Карло (в частности при помощи синтетических траекторий, сгенерированных моделью мира), при этом посчитав математическое ожидание по $d_{i:i+n}$ полностью, что позволит встроить вероятности завершения эпизода в оценку математического ожидания суммарной дисконтированной награды с сохранением возможности приближения с ее помощью градиента по совершаемым агентом действиям. Это является важным свойством, поскольку оригинальный алгоритм Dreamer использует reparametrization trick для того, чтобы получить возможность выразить $\nabla_{a_{i:i+n}} \mathbb{E}_\tau \mathrm{TD}_\lambda(s_{i:i+n})$ как $\mathbb{E}_\tau \nabla_{a_{i:i+n}} \mathrm{TD}_\lambda(s_{i:i+n})$ при подсчете градиента для последующего обновления актора, однако в случае невозможности разделения τ и $d_{i:i+n}$ данный переход был бы невозможен, т.к. reparametrization trick использует свойства нормального распределения, а $d_{i:i+n}$ распределено категориально. Благодаря формуле 2.2.1 совршение данного перехода при подсчете градиента является возможным, и для того, чтобы учесть вероятность завершения эпизода, достаточно модифицировать оценку суммарной дисконтированной награды $\mathrm{TD}_\lambda(s_{i:i+n})$ с учетом этой вероятности.

Для начала перепишем формулу λ-return в виде одной суммы по переходам в отрезке траектории:

$$\mathrm{TD}_\lambda(s_{i:i+n}) = \lambda^n \gamma^n V_\pi(s_n) + \sum_{t=0}^{n-1} \gamma^t \lambda^t \cdot (r_t + \gamma \cdot (1 - \lambda) V_\pi(s_{t+1}))$$

При использовании такой записи, данную формулу можно достаточно просто модифицировать с учетом вероятности завершения эпизода:

$$\mathrm{TD}_\lambda(s_{i:i+n}) = w_n V_\pi(s_n) + \sum_{t=0}^{n-1} w_t \cdot (r_t + \gamma \cdot (1 - p(d_{t+1}|s_{t+1})) \cdot (1-\lambda) V_\pi(s_{t+1}))$$
$$\text{где } w_t = \lambda^t \gamma^t \prod_{k=0}^{t} (1 - p(d_k|s_k)) \tag{2.2.2}$$

Поскольку сгенерированные моделью мира состояния $s_i = (z_i, h_i)$, приближенные вероятности $p(d_i|s_i)$ и награды r_i, а также оценки value-функции $V_\pi(s_i)$ полученная при помощи критика являются дифференцируемым по совершаемым агентом действиями, формула **??** также является дифференцируемой по действиям и, следовательно, позволяет использовать алгоритм градиентного спуска для обучения актора.

Наконец, для обучения актора и критика будут использоваться только оценки суммарной награды, полученные только для начальных состояний, поскольку только начальные состояния являются гарантированно достижимыми в реальной среде (поскольку получены моделью из реального опыта), в то время, как синтезированные состояния могут быть не всегда гарантированно достижимы: например, вероятность достижения данного состояния до того, как будет завершен эпизод, может быть ничтожно малой, что на практике соответствует выполнению условия завершения эпизода до достижения данного состояния.

2.2.2. Изменение порядка формирования скрытого состояния

В оригинальной архитектуре модели мира, предложенной авторами алгоритма Dreamer, наблюдение учитывается только после совершения агентом хотя бы одного действия, что может привести к различным проблемам как на этапе обучения модели, так и на этапе применения модели во время взаимодействия агента с реальным окружением. Данные проблемы были подробно описаны в разделе 2.1. С целью устранения данного недостатка, архитектура модели мира была модифицирована таким образом, чтобы состояние учитывало наблюдение до того, как совершит действие (рис. 2.2.1).

Таким образом, на шаге i траектории при совершении действия агент руководствуется детерминированной частью состояния h_i, полученной в результате совершения предыдущего действия на предыдущем шаге траектории

(или инициализированного 0 если эпизод только начался), а также стохастической частью состояния z_i, полученной при помощи апостериорного распределения $p(z_i|h_i, o_i)$ или априорного распределения $p(z_i|h_i)$. После совершения агентом действия, строится детерминированная часть следующего состояния среды h_{i+1}, которая теперь включает в себя информацию о стохастической части z_i текущего состояния и совершенном агентом действии a_i. В отличии от оригинальной архитектуры, для приближения полученной агентом награды, используется не полное следующее состояние (z_{i+1}, h_{i+1}), а детерминированная часть следующего состояния h_{i+1} и стохастическая часть предыдущего состояния z_i, т.е., информация о новом полученном наблюдении никак не учитывается. Это же верно для модели, предсказывающий завершение эпизода, и для критика агента.

2.2.3. Изменение модели кодирования и декодирования наблюдения

Оригинальная модель мира при обучении использует обучаемое априорное распределение, а также учитывает градиенты декодера наблюдения по детерминированной части состояния, что может привести к различным проблемам, подробно описанным в разделе 2.1.

Для того, чтобы исправить возможные проблемы, вызванные обучаемым априорным распределением, было решено использовать фиксированное стандартное нормальное распределение в качестве априорного распределения при обучении стохастической части скрытого состояния. При этом, поскольку фактическое распределение $p(z_i|h_i)$ может в значительной степени отличаться от априорного, было решено оставить блок, отвечающий за приближение данного распределения, и использовать его приближение во время генерации синтетических данных при помощи модели мира. При этом была введена дополнительная регуляризация между распределениями $p(z_i|h_i)$ и $p(z_i|h_i, o_i)$ с помощью дивергенции Джеффри $D_J(p||q) = 0.5(D_{\mathrm{KL}}(p||q) + D_{\mathrm{KL}}(q||p))$, которую было решено использовать из-за наличия у нее свойства симметричности.

С целью уменьшения вероятности концентрации информации о наблюдении в детерминированной части скрытого состояния было принято решение не использовать градиент функции потерь восстановления наблюдения по детерминированной части скрытого состояния. При этом h_i все еще используется при декодировании, что позволяет декодеру извлечь дополнительную информацию, что помогает сделать z_i более информативным, т.к. при идеальном кодировании

в нем не будет информации, закодированной в h_i. В каком-то смысле данный подход можно рассматривать как адаптацию идеи условного вариационного автокодировщика к архитектуре модели мира.

2.3. Адаптация maximum entropy exploration к model-based RL

Основным преимуществом алгоритмов model-based обучения с подкреплением является то, что для обучения агента они требуют значительно меньшего количества опыта взаимодействия со средой, чем классические model-free алгоритмы. Тем не менее, на данный момент современные model-based методы не используют каких-либо сложных техник исследования среды, ограничиваясь лишь простыми подходами (раздел 1.4). Это приводит к тому, что получаемый алгоритмом опыт, используемый в последствии для обучения модели мира, является менее информативным, чем мог бы быть в случае, если бы алгоритм использовал улучшенные методы исследования среды.

Все современные методы исследования среды, рассмотренные в разделе 1.4, стремятся повысить новизну получаемого во время взаимодействия со средой опыта различными способами, причем многие из этих методов могут быть применены к любому model-free алгоритму. Среди всех рассмотренных методов, однако, выделяется метод, основанный на идее максимизации энтропии распределения посещаемых агентом состояний. Авторы данного метода привели ряд теоретических гарантий того, что данный метод является гарантированно эффективным для тех задач, для которых он применим, однако имеет ряд ограничений собственной применимости, особенно в задачах с континуальными пространствами действий и состояний, вызванных в том числе тем, что в model-free постановке задачи нет возможности эффективно приблизить распределение посещаемых агентом состояний и использовать его в процессе обучения агента. В отличии от model-free алгоритмов, которые используют только опыт взаимодействия с реальной средой, model-based алгоритмы позволяют получить большое количество синтетического опыта, сгенерированного моделью мира, что позволяет эффективно использовать идею максимизации энтропии посещаемых агентом состояний с целью направленного исследования среды агентом.

Также, как и в оригинальном алгоритме maximum entropy exploration, будем использовать дисконтированное распределение (уравнение 1.4.1) в качестве распределения состояний. Поскольку синтетические состояния, сгенерирован-

ные моделью мира, могут быть не достижимы в реальном окружении, данное распределение необходимо модифицировать с учетом вероятности завершения эпизода. Кроме того, поскольку получать синтетические траектории бесконечной длины не представляется возможным, распределение необходимо представить в рекуррентном виде.

Для начала выведем рекуррентную запись дисконтированного распределения посещаемых агентом состояний:

$$\begin{aligned} p_\pi^\gamma(s_{t+}|s_t) &= (1-\gamma)\sum_{i=0}^{\infty}\gamma^i p_\pi(s_{t+i+1}|s_t) \\ &= (1-\gamma)\cdot p_\pi(s_{t+1}|s_t) + \gamma\cdot(1-\gamma)\cdot\sum_{i=0}^{\infty}\gamma^i p_\pi(s_{t+i+2}|s_t) \\ &= (1-\gamma)\cdot p_\pi(s_{t+1}|s_t) + \gamma\cdot p_\pi^\gamma(s_{(t+1)+|s_t} \end{aligned} \tag{2.3.1}$$

Заметим, что благодаря свойству марковости функции перехода в марковском процессе принятия решений можно записать, что:

$$p_\pi^\gamma(s_{(t+i)+}|s_t) = \mathbb{E}_{p_\pi(s_{t+i}|s_t)} p_\pi^\gamma(s_{(t+i)+}|s_{t+i}) \tag{2.3.2}$$

Обозначим $p_\pi^\gamma(s_{t+}|s_t)$ как $\rho_\pi^\gamma(s_t)$. Тогда, используя формулу 2.3.1 и формулу 2.3.2, можно записать:

$$\rho_\pi^\gamma(s_t) = (1-\gamma)p_\pi(s_{t+1}|s_t) + \gamma\mathbb{E}_{p_\pi(s_{t+1}|s_t)}\rho_\pi^\gamma(s_{t+1}) \tag{2.3.3}$$

Как уже было сказано ранее, в данную формулу необходимо интегрировать вероятность завершения эпизода. Для начала определим, как завершение эпизода влияет на $\rho_\pi^\gamma(s_t)$:

$$\rho_\pi^\gamma(s_t) = \begin{cases} (1-\gamma)p_\pi(s_{t+1}|s_t) + \gamma\mathbb{E}_{p_\pi(s_{t+1}|s_t)}\rho_\pi^\gamma(s_{t+1}) & \text{если } d_{t+1}=0 \\ p_\pi(s_{t+1}|s_t) & \text{иначе} \end{cases}$$

Поскольку d_i - это случайная величина, то запишем это в виде математического ожидания:

$$\begin{aligned} \rho_\pi^\gamma(s_t) = &(p(d_{t+1}|s_{t+1}) + (1-p(d_{t+1}|s_{t+1}))(1-\gamma))p_\pi(s_{t+1}|s_t) \\ &+\gamma\cdot(1-p(d_{t+1}|s_{t+1}))\mathbb{E}_{p_\pi(s_{t+1}|s_t)}\rho_\pi^\gamma(s_{t+1}) \end{aligned} \tag{2.3.4}$$

Теперь осталось научиться научиться приближать данное распределение таким образом, чтобы можно было взять градиент логарифма плотности данного распределения по состоянию, т.е. вычислить $\nabla_s \log \rho_\pi^\gamma(s|s_t)$. Для этого, например, подходит алгоритм глубокого обучения, который называется Mixture Density Network. Его задачей является приближение некоторого условного распределения $p(y|x)$ с помощью смеси нормальных распределений, что позволяет вычислять необходимый нам градиент логарифма плотности данного распределения в некоторой заданной точке y при условии x. Оригинальный алгоритм MDN определяет функцию потерь как $\mathbb{E}_{x,y\sim \mathrm{D}} - \log \hat{p}(y|x)$, где D - набор данных, $\hat{p}(y|x)$ - плотность вероятности, приближенная моделью. Для того, чтобы применить данный алгоритм для приближения распределения $\rho_\pi^\gamma(s|s_t)$, модифицируем функцию потерь в соответствии с разложением, представленным в формуле 2.3.4:

$$L = -\mathbb{E}_{p_\pi(s_{t+1}|s_t)}[(p(d_{t+1}|s_{t+1}) + (1 - p(d_{t+1}|s_{t+1}))(1-\gamma)) \log \hat{\rho}_\pi^\gamma(s_{t+1}|s_t) \\ + \gamma \cdot (1 - p(d_{t+1}|s_{t+1}))\mathbb{E}_{s\sim\rho_\pi^\gamma(s_{t+1})} \log \hat{\rho}_\pi^\gamma(s|s_t)]$$

Где s_{t+1} распределено в соответствии с моделью мира и текущей политикой агента. Эту формулу можно также представить как математическое ожидание по отрезку траектории длины n просто развернув формулу 2.3.4 на n шагов. Также стоит отметить, что в формуле используется математическое ожидание по реальному распределению $\rho_\pi^\gamma(s_{t+1})$. При использовании метода Монте Карло для оценки градиента приближения $\hat{\rho}_\pi^\gamma(s|s_t)$ семплирование из реального распределение можно заменить на семплирование из существующего приближения данного распределения $\hat{\rho}_\pi^\gamma(s_{t+1})$. При этом важно отметить, что поскольку оператор математического ожидания будет вынесен из-под знака градиента, семплирование из приближенного распределения не повлечет за собой изменения градиентов по параметрам модели.

Теперь модифицируем процесс обучения агента таким образом, чтобы он максимизировал энтропию распределения $\rho_\pi^\gamma(s_t)$, для чего достаточно вычислить $\nabla_\theta H(\rho_\pi^\gamma(s_t))$ (где θ - параметры нейронной сети актора), чтобы затем при-

менить метод градиентного спуска:

$$\begin{aligned}\nabla_\theta H(\hat{\rho}^\gamma_\pi(s_t)) =& \nabla_\theta - \mathbb{E}_{s\sim\rho^\gamma_\pi(s_t)} \log \rho^\gamma_\pi(s|s_t) \\ =& \nabla_\theta - \mathbb{E}_{p_\pi(s_{t+1}|s_t)}[w \cdot \log \rho^\gamma_\pi(s_{t+1}|s_t) \\ &+ (1-w)\cdot \mathbb{E}_{s\sim\rho^\gamma_\pi(s_{t+1})} \log \rho^\gamma_\pi(s|s_t)]\end{aligned}$$

Где $w = (p(d_{t+1}|s_{t+1}) + (1 - p(d_{t+1}|s_{t+1}))(1-\gamma))$. Важно отметить, что оператор $\nabla_{a_{t+}}$ можно внести под $\mathbb{E}_{p_\pi(s_{t+1}|s_t)}$ используя reparametrization trick, и затем приблизить градиент с помощью метода Монте Карло также, как и в случае с оценкой суммарной награды при помощи λ-returns. Такое преобразование можно итеративно повторять, чтобы получить формулу на n шагов вперед. Осталось только научиться считать градиент для последнего слагаемого $\nabla_\theta \mathbb{E}_{s\sim\rho^\gamma_\pi(s_{t+n})} \log \rho^\gamma_\pi(s|s_t)$. Заметим, что s_{t+n} является дифференцируемым по параметрам политики θ, т.к. получено при помощи reparametrization trick. Если заменить $\rho^\gamma_\pi(s_{t+n})$ на приближение при помощи MDN, а затем разложить математическое ожидание на две составляющие: математическое ожидание по категориальному распределению, определяющему нормальное распределение, и по выбранному нормальному распределению, а затем расписать математическое ожидание по категариальному распределнию как сумму и применить reparamtrization trick к математическому ожиданию по нормальному распределению, то можно также приблизить градиент при помощи метода Монте Карло. Однако на практике для более стабильной работы алгоритма предлагается предположить независимость распределения $\rho^\gamma_\pi(s_{t+n})$ от текущих параметров политики и приблизить математическое ожидание при помощи метода Монте Карло без проведения дополнительных преобразований.

2.4. Maximum Entropy Dreamer

Финальный алгоритм Maximum Entropy Dreamer, предлагаемый в этой работе, основан на алгоритме Dreamer (раздел 1.5), включает в себя модификации, подробно описанные в разделе 2.2, с целью исправления недостатков оригинального алгоритма (раздел 2.1). Кроме того, предлагаемый алгоритм использует разработанный в ходе этой работы метод исследования среды, использующий свойства подхода model-based обучения с подкреплением для максимизации энтропии посещаемых агентом состояний в процессе обучения (раздел 2.3).

В алгоритме Maximum Entropy Dreamer каждая итерация обучения состоит из следующих шагов:

а) Получить новую траекторию из окружения, используя текущую политику агента с дополнительным случайным зашумлением действий для исследования среды

б) Добавить полученную последовательность в буфер опыта

в) Выполнить n обновлений следующим образом:

1) Получить минибатч отрезков траекторий из буфера опыта
2) При помощи минибатча и модели мира посчитать скрытые состояния (z_i, h_i)
3) Посчитать функцию потерь ELBO для модели мира используя (z_i, h_i) и обновить модель мира при помощи градиентного спуска.
4) Сгенерировать синтетические траектории, используя (z_i, h_i) в качестве начальных состояний.
5) Использовать синтетические траектории для оценки суммарной награды, используя модифицированный метод λ-returns, учитывающий вероятность завершения состояния.
6) Использовать синтетические траектории и Mixture Density Network для оценки энтропии распределения посещаемых агентом состояний с помощью метода, описанного в разделе 2.3.
7) Обновить актора и критика при помощи посчитанных оценок суммарной награды и оценки энтропии распределения посещаемых агентом состояний.
8) С помощью синтетических траекторий обновить модель, приближающую распределение посещаемых агентом состояний с помощью метода, описанного в разделе 2.3.

ГЛАВА 3. ЭКСПЕРИМЕНТЫ

3.1. Физический симулятор PyBullet

Физический симулятор PyBullet предоставляет большой набор окружений, которые реализуют задачи управления различными видами роботов в сложной физической среде. Такие задачи часто используются для тестирования алгоритмов обучения с подкреплением. В частности, авторы оригинального алгоритма Dreamer использовали данные задачи, имплементированные при помощи физического симулятора MuJoCo, который, в отличии от PyBullet, не является открытым программным обеспечением.

Для тестирования разработанных в ходе работы модификаций и улучшений будут использованы следующие задачи:

— Окружение Ant. В данном окружении агенту предлагается управлять четырехногим пауком, на каждая нога которого прикреплена к шарообразному туловищу и имеет дополнительный коленный сустав. В данном окружении эпизод завершается в случае, если робот перевернулся на спину (это требует совершения довольно сложной последовательности действий), или по истечении времени эпизода.

— Окружение Hopper. Агенту предлагается управлять одноногим существом, находящимся в двухмерном пространстве. Эпизод завершается при падении агента или в случае, когда истекло время эпизода.

— Окружение Walker. Данная задача аналогична задаче Hopper, только теперь существо имеет две ноги. Условия завершения эпизода также аналогичные.

— Окружение HalfCheetah. В данном окружении агент управляет двуногим аналогом четырехногого животного, находящимся в двухмерном мире. Эпизод завершается в случае падения робота или в случае, когда истекло время эпизода.

Во всех окружениях в качестве наблюдения используется изображение с камеры, которая следит за роботом и перемещается вместе с ним. Время эпизода ограничено 1000 шагов окружения. Награда основана на том, как быстро робот двигается вперед, и варьируется в зависимости от окружения.

3.2. Гиперпараметры

Для оригинального алгоритма Dreamer были использованы гиперпараметры, указанные авторами алгоритма. Для модифицированного алгоритма Dreamer были изменены следующие гиперпараметры:

— Размер стохастической части состояния увеличен до 128. Это сделано для того, чтобы компенсировать недостаток информации при декодировании наблюдения, которую оригинальный алгоритм получал из детерминированной части состояния.

— Коэффициент D_{KL} в функции потерь ELBO уменьшен до 0.1. Также, как и увеличение размера стохастической части состояния, это позволяет компенсировать недостаток информации при декодировании наблюдения и сделать стохастическую часть наблюдения более информативной.

— Функции потерь для модели, приближающей награду, и для критика были заменены на сглаженное L1-расстояние, что позволяет модели стабильней обучаться при большой разнице между предсказанным и реальным значениями целевой функции.

Для модифицированного Dreamer'а с Maximum Entropy Exploration были введены дополнительные гиперпараметры, которые связаны с исследованием среды агентом. В частности, был добавлен параметр дисконтирования для распределения посещаемых агентом состояний, коэффициент, с которым энтропия распределения последующих состояний суммируется с оригинальной функцией потерь актора, а также количество голов (нормальных распределений в получаемой смеси) нейронной сети Mixture Density Network, приближающей распределение посещаемых агентом состояний. Фактор дисконтирования во всех экспериментах был равен 0.9, количество голов было равно 8, а коэффициент аддитивной добавки к функции потерь выбирался отдельно для каждого окружения, чтобы учитывать различный масштаб получаемых агентом наград.

3.3. Сравнение алгоритмов

Как было описано в разделе 3.1, эксперименты проводились на наборе окружений, предоставленных физическим симулятором PyBullet, а именно на средах Ant, Walker2D, HalfCheetah и Hopper. Предложенный в данной работе алгоритм Maximum Entropy Dreamer был сравнен с оригинальным алгоритмом Dreamer, а также с модифицированным алгоритмом Dreamer, использующим

классический подход к исследованию среды (случайный шум). Результаты эксперимента представлены на рисунке 3.3.1.

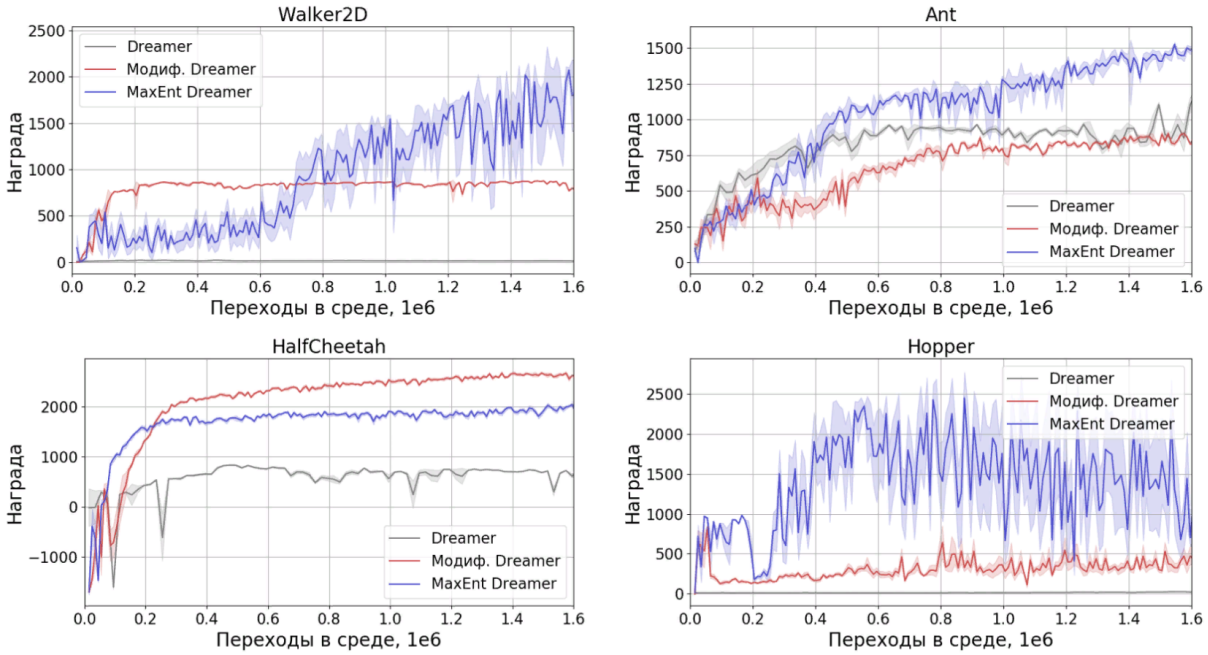

Рисунок 3.3.1 – Графики, отражающие получаемую агентом награду во врем обучения в средах Ant, HalfCheetah, Walker2D и Hopper. Серым цветом изображен оригинальный алгоритм Dreamer. Красным цветом изображен алгоритм Dreamer с модификациями, описанными в разделе 2.2. Синим цветом изображен алгоритм Maximum Entropy Dreamer, описанный в разделе 2.4.

Как видно из графиков, оригинальный алгоритм Dreamer успешно решает задачу окружения Ant, менее успешно решает задачу окружения HalfCheetah и не способен решить задачи окружений Walker2D и Hopper, что в значительной степени коррелирует с возможностью завершить эпизод досрочно: в среде Ant досрочное завершение эпизода возможно только в случае совершения сложной последовательности действий, в среде HalfCheetah достичь терминального состояния проще, чем в среде Ant, а в средах Walker2D и Hopper терминальное состояние достигается при совершении большинства возможных последовательностей действий, т.к. робот является неустойчивым. В то же время, модифицированный алгоритм Dreamer куда лучше справляется с решением задач HalfCheetah, Walker2D и Hopper, показывая незначительно отличающиеся в худшую сторону результаты на среде Ant. Алгоритм Maximum Entropy Dreamer показывает более высокие результаты во всех средах по сравнению с оригинальным алгоритмом Dreamer и проигрывает модифицированному алгоритму

Dreamer с оригинальным способом исследования среды только при решении задачи окружения HalfCheetah, превосходя его во всех остальных алгоритмах.

Стоит также отметить куда больший разброс получаемых наград алгоритмом Maximum Entropy Dreamer по сравнению с другими вариантами алгоритма, что вызвано большой энтропией политики, выученной агентом.

ЗАКЛЮЧЕНИЕ

В результате данной работы были разработаны все необходимые для исправления недостатков, повышения эффективности и стабильности модификации алгоритма Dreamer - одного из популярных современных алгоритмов model-based обучения с подкреплением. Это позволило исправить недостатки алгоритма, которые приводили к потери эффективности и субоптимальной производительности обучаемого агента. В частности, была разработана модификация, которая позволяет алгоритму учитывать вероятность завершения эпизода как на стадии обучения модели мира, так и на стадии обучения агента при помощи синтетических данных, генерируемых данной моделью. Также была пересмотрена архитектура модели мира, благодаря чему теперь агент получает информацию о наблюдении среды до того, как совершит первое действие в среде, что позволяет избежать ряда проблем, вызванных совершением субоптимального первого действия в начале эпизода. Кроме того, был разработан ряд модификаций, повышающих эффективность и стабильность работы алгоритма. Данные модификации позволили значительно повысить эффективность алгоритма в трех из четырех окружений, на которых проводилось тестирование.

Помимо модификации уже существующего алгоритма Dreamer, в ходе работы был предложен новый подход исследования среды, учитывающий особенности model-based алгоритмов обучения с подкреплением и использующий идею максимизации энтропии распределения посещаемых агентом состояний. Затем, данный метод был применен к модифицированному алгоритму Dreamer, что также позволило повысить эффективность алгоритма в трех из четырех сред по сравнению с модифицированной версией алгоритма Dreamer, использующей случайный шум для исследования среды. Финальный алгоритм Maximum Entropy Dreamer смог значительно превзойти оригинальный алгоритм Dreamer во всех четырех окружениях.

Стоит отметить, что способ исследования среды в алгоритмах model-based обучения с подкреплением при помощи максимизации энтропии распределения состояний, предложенный в этой работе, может быть применен, помимо алгоритма Dreamer, и к другим алгоритмам model-based обучения с подкреплением, использующим синтетические данные для обучения агента, что делает его достаточно общим подходом.

Помимо использования model-based подхода в алгоритмах обучения с подкреплением и методов, повышающих эффективность исследования среды, существует также большое количество других способов уменьшения количества необходимого для обучения агента опыта, которые не были рассмотрены в рамках данной работы, и которые могут потенциально быть адаптированы к model-based алгоритмам с целью повышения их эффективности.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Dota 2 with large scale deep reinforcement learning / C. Berner [и др.] // arXiv preprint arXiv:1912.06680. — 2019.

2 *Arulkumaran K.*, *Cully A.*, *Togelius J.* Alphastar: An evolutionary computation perspective // Proceedings of the Genetic and Evolutionary Computation Conference Companion. — 2019. — С. 314–315.

3 Dream to control: Learning behaviors by latent imagination / D. Hafner [и др.] // arXiv preprint arXiv:1912.01603. — 2019.

4 *Bishop C. M.* Mixture density networks. — 1994.

5 *Montague P. R.* Reinforcement learning: an introduction, by Sutton, RS and Barto, AG // Trends in cognitive sciences. — 1999. — Т. 3, № 9. — С. 360.

6 Human-level control through deep reinforcement learning / V. Mnih [и др.] // Nature. — 2015. — Т. 518, № 7540. — С. 529.

7 Continuous control with deep reinforcement learning / T. P. Lillicrap [и др.] // arXiv preprint arXiv:1509.02971. — 2015.

8 Proximal policy optimization algorithms / J. Schulman [и др.] // arXiv preprint arXiv:1707.06347. — 2017.

9 Trust region policy optimization / J. Schulman [и др.] // International Conference on Machine Learning. — 2015. — С. 1889–1897.

10 *Fujimoto S.*, *Hoof H.*, *Meger D.* Addressing function approximation error in actor-critic methods // International Conference on Machine Learning. — PMLR. 2018. — С. 1587–1596.

11 Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor / T. Haarnoja [и др.] // International Conference on Machine Learning. — PMLR. 2018. — С. 1861–1870.

12 Unifying count-based exploration and intrinsic motivation / M. G. Bellemare [и др.] // arXiv preprint arXiv:1606.01868. — 2016.

13 Count-based exploration with neural density models / G. Ostrovski [и др.] // International conference on machine learning. — PMLR. 2017. — С. 2721–2730.

14 Exploration by random network distillation / Y. Burda [и др.] // arXiv preprint arXiv:1810.12894. — 2018.

15 Large-scale study of curiosity-driven learning / Y. Burda [и др.] // arXiv preprint arXiv:1808.04355. — 2018.

16 *Tao R. Y.*, *François-Lavet V.*, *Pineau J.* Novelty Search in representational space for sample efficient exploration // arXiv preprint arXiv:2009.13579. — 2020.

17 *Ermolov A.*, *Sebe N.* Latent World Models For Intrinsically Motivated Exploration // arXiv preprint arXiv:2010.02302. — 2020.

18 Provably efficient maximum entropy exploration / E. Hazan [и др.] // International Conference on Machine Learning. — PMLR. 2019. — С. 2681–2691.

19 Learning latent dynamics for planning from pixels / D. Hafner [и др.] // International Conference on Machine Learning. — PMLR. 2019. — С. 2555–2565.