МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт (Филиал) № 8 «Компьютерные науки и прикладная математика» **Кафедра** 806

Группа М8О-408Б-19 **Направление подготовки** 01.03.02 «Прикладная математика и информатика»

Профиль Информатика

Квалификация: бакалавр

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему: Разработка алгоритма машинного обучения с подкреплением на основе градиента стратегии в игровых средах на примере игр Atari

Автор ВКРБ:	Гаптулхаков Руслан Рамилевич	(_____)
Руководитель:	Стрижак Сергей Владимирович	(_____)
Консультант:	—	(_____)
Консультант:	—	(_____)
Рецензент:	—	(_____)

К защите допустить

Заведующий кафедрой № 806 Крылов Сергей Сергеевич (_____)

____ мая 2023 года

Москва 2023

РЕФЕРАТ

Выпускная квалификационная работа бакалавра состоит из 41 страницы, 11 рисунков, 4 таблиц, 7 использованных источников, 1 приложения.

ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ, ГРАДИЕНТ ПОЛИТИКИ, ATARI

Объектом исследования в данной работе является изучение алгоритмов обучение с подкреплением на основе градиента политики на игровой среде Atari.

Цель работы - найти и реализовать наиболее эффективный алгоритм машинного обучения с подкреплением на основе градиента стратегии для игр Atari.

Для достижения поставленной цели были поставлены следующие задачи:

- Провести сравнение алгоритмов на основе градиента стратегии и выбрать наиболее эффективный.
- Выбрать стек системного ПО и несколько игр из среды Atari на основе анализа существующего ПО для алгоритмов RL.
- Реализовать выбранный алгоритм машинного обучения с подкреплением на основе градиента стратегии.
- Оценить достигнутые результаты.

Результатами работы стали программное обеспечение, позволяющее обучить алгоритм на выбранной игре из среды Atari, а также графики, позволяющие оценить качество обучения и видеозаписи взаимодействия агента со средой.

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе бакалавра применяют следующие термины с соответствующими определениями:

Action (действие) — это решение, принимаемое агентом на основе текущего состояния среды. Действие может быть выбрано из некоторого набора допустимых действий

Agent (агент) — это сущность, которая принимает решения в среде. Агент может быть реализован в виде компьютерной программы или робота

Environment (среда) — это система, в котором действует агент. Среда может быть физической, как в случае с роботами, или виртуальной, как в случае с видеоиграми

Exploitation (эксплуатация) — это использование известной информации для максимизации вознаграждения

Exploration (исследование) — это изучение окружающей среды путём выполнения случайных действий с целью получения дополнительной информации о среде

Policy (политика/стратегия) — это стратегия, которую агент использует для выбора действий в каждом состоянии

Reward (награда) — это скаляр, которое получает агент от среды после выбора действия

State (состояние) — это описание текущего состояния среды, в которой находится агент

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе бакалавра применяют следующие сокращения и обозначения:

ИИ — искусственный интеллект

ПО — программное обеспечение

AC — actor-critic

API — application programming interface

A2C — advantage actor-critic

CNN — convolutional neural network, сверточная нейронная сеть

LSTM — long short-term memory

MLP — multilayer perceptron

PPO — proximal policy optimization

RL — reinforcement learning, обучение с подкреплением

TRPO — trust region policy optimization

ВВЕДЕНИЕ

Обучение с подкреплением (RL) — это подраздел машинного обучения, который фокусируется на обучении интеллектуальных агентов принятию решений в сложных динамических средах. Алгоритмы RL позволяют агентам учиться на своем опыте методом проб и ошибок, используя вознаграждения и наказания, чтобы корректировать свое поведение и улучшать процесс принятия решений с течением времени. Этот подход привел к прорыву в широком спектре приложений, от робототехники и игр до финансов и здравоохранения. Сегодня RL продолжает развиваться и находить новые приложения. Например, RL используется в создании автономных систем для управления автомобилями и беспилотниками, в проектировании энергоэффективных систем и в борьбе с многими другими сложными задачами.

Актуальность и новизна темы заключается в том, что существующие алгоритмы машинного обучения с подкреплением на основе градиента стратегии все еще не обладают достаточной стабильностью и эффективностью, особенно в условиях большого количества состояний. Разработка новых алгоритмов, способных преодолеть эти проблемы, может значительно улучшить качество обучения и расширить область применения машинного обучения с подкреплением.

Связь с другими научно-исследовательскими работами заключается в том, что данная работа является продолжением и дополнением уже существующих исследований в области машинного обучения с подкреплением, таких как работа «Playing Atari with Deep Reinforcement Learning» [1].

Применение алгоритмов обучения с подкреплением (RL) может быть достаточно разнообразным и включает в себя множество приложений в различных областях, включая робототехнику, игровую индустрию, управление энергетическими системами, медицину и другие.

В робототехнике RL может использоваться для обучения роботов выполнению задач, таких как манипуляции объектами, навигация в неизвестных средах. RL позволяет роботу учиться на основе опыта, что позволяет ему адаптироваться к изменяющейся среде и ситуациям.

В игровой индустрии RL может быть использован для создания более

интеллектуальных и адаптивных игровых ботов, которые могут изменять свое поведение на основе действий игрока и результатов игры.

В управлении энергетическими системами RL может использоваться для оптимизации потребления энергии и управления сетью энергопотребления.

В медицине RL может быть использован для оптимизации лечения и подбора лекарственных препаратов, а также для создания интеллектуальных систем поддержки принятия решений в медицинских учреждениях.

В целом, применение RL может быть очень широким и зависит от конкретной области применения и поставленных задач. RL позволяет создавать более интеллектуальные и адаптивные системы, которые могут обучаться на основе опыта и изменять свое поведение в соответствии с изменяющимися условиями.

1 ПОСТАНОВКА И ФОРМАЛИЗАЦИЯ ЗАДАЧИ

Необходимо исследовать возможности нейронных сетей взаимодействовать с играми из игровой среды Atari.

На вход программа должна принимать гиперпараметры модели, от которых будет зависить качество обучения, а также параметры среды, с которой будет взаимодействовать нейронная сеть. Пользователь программы сможет управлять ими.

В таблице 1 приведём список основных изменяемых параметров.

Таблица 1 – Основные изменяемые параметры ПО

Параметр	Значение
–gym-id	Название игры, на которой будет обучаться нейронная сеть
–learning-rate	Коэффициент регулирущий скорость обучения
–seed	Детерминизация случайных величин
–total-timesteps	Общее количество шагов в среде
–cuda	Использование GPU для вычислений
–capture-video	Запись видео, обученной игры
–num-envs	Количество идентичный сред, которые генерируют данные
–num-steps	Синхронизация current и prior policy
–anneal-lr	Уменьшение скорости обучения в процессе тренировки
–clip-coef	Коэффициент регулирующий степень изменения политики

В качестве результата программа должна собирать различную статистику в ходе обучения, и представлять её в виде удобных графиков, а также видеозаписи взаимодейсвия агента со средой.

2 ОПИСАНИЕ АЛГОРИТМОВ ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ ОСНОВАННЫХ НА ГРАДИЕНТЕ ПОЛИТИКИ

Пример цикла обучения представлен на рисунке 1. Картинка была взята [2].

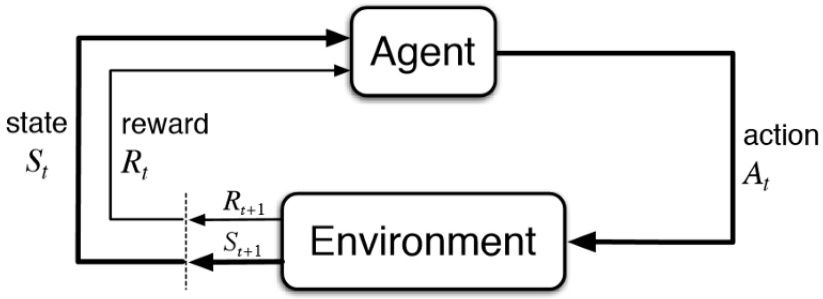


Рисунок 1 – Схема работы ПО

Цикл обучения начинается с первого шага - сбора данных. Агент взаимодействует с окружением, выполняя действия, и наблюдает состояния окружения и получает награды за свои действия. Наблюдения состояний и наград передаются в агента для дальнейшего обучения.

На втором шаге - обработки данных - агент анализирует полученные наблюдения и принимает решение о следующем действии, которое нужно выполнить. Для этого агент использует свою стратегию, которая определена в ходе предыдущего обучения. Стратегия может быть представлена в виде функции или алгоритма, которые оптимизируют выбор действий на основе оценки ожидаемой награды.

На третьем шаге - выполнении действия - агент выполняет выбранное действие в окружении и взаимодействует с ним. В результате окружение изменяет свое состояние, и агент получает новые наблюдения о состоянии окружения и получает новую награду.

На четвертом шаге - обновлении стратегии - агент использует полученные наблюдения и награды для обновления своей стратегии.

2.1 Классификация RL алгоритмов

Model-Based vs. Model-Free

Model-Based и Model-Free - это два подхода в обучении с подкреплением,

которые отличаются способом описания и использования среды.

Model-Based алгоритмы используют модель среды, чтобы предсказать будущие состояния и вознаграждения, которые ожидаются при выполнении разных действий. Они строят модель среды, которая описывает, как система будет вести себя в ответ на разные действия, и затем используют эту модель для выбора наилучшего действия. Примерами Model-Based алгоритмов являются алгоритмы Dynamic Programming, такие как Value Iteration и Policy Iteration.

С другой стороны, Model-Free алгоритмы напрямую изучают политику (policy) - отображение между состояниями и действиями - без построения модели среды. Они обучаются на основе опыта, собранного в процессе взаимодействия агента со средой, и оценивают полезность (value) каждого состояния. Примерами Model-Free алгоритмов являются алгоритмы Monte-Carlo и Temporal Difference, такие как Q-Learning и SARSA.

Например, если мы хотим научить робота играть в настольный футбол, мы можем использовать Model-Based подход для моделирования движения мяча и динамики стола, чтобы предсказать, как робот может лучше всего действовать. Но мы также можем использовать Model-Free подход для обучения робота на основе его опыта, собранного во время игры, и оценки эффективности его действий.

Value-Based vs. Policy-Based

Value-Based и Policy-Based - это два подхода в обучении с подкреплением, которые отличаются способом определения и использования оптимальной стратегии. Value-Based алгоритмы изучают Value-функцию, которая определяет, как хорошо мы можем ожидать, что агент достигнет определенного состояния или действия в будущем. Value-функция позволяет агенту выбирать действия, которые будут давать ему наилучшую вознаграждение в долгосрочной перспективе. Value-Based алгоритмы находят оптимальную Value-функцию и используют её для выбора оптимальной политики. Примерами Value-Based алгоритмов являются алгоритмы Q-Learning, SARSA и DQN.

Policy-Based алгоритмы изучают policy-функцию напрямую, которая определяет, какие действия должен выбрать агент в каждом состоянии, чтобы

максимизировать вознаграждение. Policy-Based алгоритмы находят оптимальную policy-функцию, которая минимизирует функцию потерь (loss function) и максимизирует вознаграждение. Примерами Policy-Based алгоритмов являются алгоритмы REINFORCE, A2C и PPO.

Рассмотрим пример использования Value-Based и Policy-Based подходов на примере игры в блэкджек. В этой игре агент должен выбирать между «взять еще карту» или «остаться» в зависимости от суммы очков в его руке и виде одной из карт дилера.

Value-Based алгоритмы будут оценивать оптимальную Value-функцию для каждого состояния в игре, что позволит агенту выбрать оптимальное действие в зависимости от текущего состояния.

Policy-Based алгоритмы будут находить оптимальную policy-функцию напрямую, определяя, какое действие должен выбрать агент в каждом состоянии, чтобы максимизировать вознаграждение. Например, если агент обучен Policy-Based алгоритмом, то в состоянии с 15 очками на руках и картой дилером с номиналом 6, алгоритм может рекомендовать «остаться», так как это действие дает больше шансов на победу. Policy-Based алгоритм будет настраивать параметры policy-функции таким образом, чтобы агент максимизировал вознаграждение.

В общем, Value-Based и Policy-Based подходы могут использоваться в различных задачах обучения с подкреплением, и выбор подхода зависит от поставленной задачи и доступных данных. Value-Based алгоритмы обычно используются для задач, где модель среды хорошо определена и может быть представлена в виде математического описания. Policy-Based алгоритмы, напротив, часто применяются для задач, где модель среды не может быть явно определена или где действия агента могут сильно влиять на состояние среды.

Примеры задач, для которых используются Value-Based и Policy-Based подходы, включают игры, управление роботами и финансовый анализ. Например, в задаче управления роботом, где агент должен выбирать действия, чтобы достичь определенной цели, Value-Based алгоритмы могут использоваться для определения оптимальных действий на основе заранее известной модели среды. Policy-Based алгоритмы могут использоваться для обучения роботов, которые должны учиться управлять своими действиями, и для задач, где нет явной модели среды.

Алгоритм AC использует Value-Based подход для обучения критика

(Critic), который оценивает ожидаемое вознаграждение в каждом состоянии. Затем, Policy-Based подход используется для обучения актера (Actor), который выбирает действия на основе вероятностей, полученных от критика. Таким образом, AC сочетает преимущества обоих подходов и может использоваться для решения широкого спектра задач обучения с подкреплением. Например, AC может быть использован для задачи игры в шахматы, где модель среды хорошо определена, но также в задачах управления роботами, где модель среды может быть менее явно определена и где действия агента могут сильно влиять на состояние среды.

On-Policy vs. Off-Policy

On-Policy и Off-Policy являются двумя основными подходами к обучению с подкреплением. Оба метода обучения основываются на теории Марковских процессов принятия решений (Markov Decision Processes), которые позволяют агенту принимать решения на основе текущего состояния среды.

On-Policy алгоритмы используют для обучения ту же политику, которая используется в процессе генерации данных для обучения. Это означает, что агент собирает данные и обучается на данных, которые были сгенерированы текущей политикой. Примером On-Policy алгоритма является алгоритм SARSA (State-Action-Reward-State-Action), который используется для обучения в задачах с одним агентом.

Off-Policy алгоритмы, наоборот, используют более старую политику для генерации данных и обучения на текущей политике. Это означает, что агент собирает данные, используя более старую политику, и обучается на этих данных с помощью текущей политики. Off-Policy методы часто используются в задачах с несколькими агентами, где они могут быть более эффективны, чем On-Policy методы. Примерами Off-Policy алгоритмов являются алгоритмы Q-Learning, TD-learning, и DQN (Deep Q-Network).

Monte-Carlo vs. Temporal-Difference

Monte-Carlo и Temporal-Difference являются двумя основными методами обучения с подкреплением, которые используются для оценки функций ценности и нахождения оптимальных стратегий в задачах принятия решений.

Monte-Carlo методы используют дерево Monte-Carlo для моделирования эпизода и оценки функции ценности на основе полученных результатов. Для этого агент играет в игру полностью до конца и затем использует все полученные вознаграждения для оценки функции ценности. Примером алгоритма Monte-Carlo является First-Visit Monte-Carlo, который оценивает функцию ценности на основе первого посещения состояния в эпизоде.

Temporal-Difference (TD) методы обновляют функцию ценности на основе разницы между предсказанным вознаграждением в данный момент времени и реальным вознаграждением. Этот подход позволяет агенту обновлять функцию ценности на каждом шаге, что делает его более эффективным в задачах с длинными эпизодами. Примерами TD алгоритмов являются SARSA (State-Action-Reward-State-Action), Q-Learning и TD(0), который обновляет функцию ценности на основе одного шага в будущее. Оба метода имеют свои преимущества и недостатки, и их эффективность зависит от конкретной задачи.

Например, Monte-Carlo методы обычно требуют больше вычислительных ресурсов, но более точны при оценке функции ценности в задачах с длинными эпизодами и когда невозможно выполнить обновление функции ценности на каждом шаге. В то время как TD методы более эффективны в задачах с короткими эпизодами и могут быстрее сходиться к оптимальной стратегии. Как утверждается [3], у TD-обучения по сравнению с Monte-Carlo меньше дисперсия, но больше смещение.

2.2 Описание алгоритмов

REINFORCE

Reinforce - это один из классических алгоритмов обучения с подкреплением, который использует политику агента напрямую для обновления параметров. Агент совершает действия в среде, получает награды за эти действия и обновляет свою политику на основе собранных наград. Формула обновления в алгоритме reinforce [4] выглядит следующим образом:

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta \log \pi_\theta(a|s) G, \tag{1}$$

где θ - обучаемые параметры политики агента,

α - скорость обучения (learning rate),

∇_θ - градиент по параметрам θ,

$\pi_\theta(a|s)$ - вероятность выбора действия a в состоянии s при политике θ,

G - обобщенная награда, которая считается как сумма наград на траектории, умноженных на коэффициенты дисконтирования γ.

Обобщенная награда считается с помощью метода Monte-Carlo, поэтому его можно отнести к алгоритмам Monte-Carlo.

Actor-Critic (AC)

Actor-Critic (AC) - это расширение алгоритма reinforce, в котором добавляется оценочная функция состояний (критик), помимо политики агента. Оценочная функция используется для оценки ценности состояний путём вычисления Q-функции. Это позволяет сократить дисперсию оценки и сделать обучение более стабильным. Формула обновления в алгоритме AC имеет две части.

Для обновления политики:

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta \log \pi_\theta(a|s) Q_w(s,a), \tag{2}$$

где θ - обучаемые параметры политики агента,

α - скорость обучения для политики,

∇_θ - градиент по параметрам θ,

$\pi_\theta(a|s)$ - вероятность выбора действия a в состоянии s по текущей политике θ,

$Q_w(s,a)$ - функция отражающая полезность выбора действия a из состояния s.

Для обновления оценочной функции:

$$w_{j+1} = w_i + \beta(r + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)) \nabla_w Q_w(s_t, a_t), \tag{3}$$

где w - обучаемые параметры критика,

β - скорость обучения для оценочной функции,

r - вознаграждение полученное от среды после перехода из s_t к s_{t+1},

γ - коэффициент дисконтирования,

$Q_w(s, a)$ - функция отражающая полезность выбора действия a из состояния s,

∇_w - градиент по параметрам w.

Advantage Actor-Critic (A2C)

Advantage Actor-Critic (A2C) - это улучшенный алгоритма AC, в котором используется оценка $A(s, a)$ (преимущество) вместо оценки $Q(s, a)$. Преимущество позволяет оценить полезность действия исключая ценность состояния.

По определению преимущество вычисляется по формуле:

$$A(s, a) = Q(s, a) - V(s), \tag{4}$$

где $A(s, a)$ - преимущество,

$Q(s, a)$ - функция отражающая полезность выбора действия a из состояния s,

$V(s)$ - ценность состояния s.

Для обновления политики:

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta \log \pi_\theta(a|s) A_w(s, a), \tag{5}$$

где θ - обучаемые параметры политики агента,

α - скорость обучения для политики,

∇_θ - градиент по параметрам θ,

$\pi_\theta(a|s)$ - вероятность выбора действия a в состоянии s по текущей политике θ,

$A_w(s, a)$ - функция отражающая среднюю по состоянию s полезность выбора действия a.

Для обновления оценочной функции:

$$w_{j+1} = w_i + \beta \nabla_w A_w^2(s_t, a_t), \tag{6}$$

где w - обучаемые параметры критика,

β - скорость обучения для оценочной функции,

$A_w(s, a)$ - функция отражающая среднюю по состоянию s полезность выбора действия a,

∇_w - градиент по параметрам w.

Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) - это алгоритм, который также использует оценочную функцию (критика) и политику агента (актера), но вводит ограничения на размер обновлений политики с помощью расстояния Кульбака-Лейблера, чтобы гарантировать, что обновления не слишком сильные и сохраняют близость к исходной политике. TRPO оптимизирует политику с учетом «доверительного интервала» для изменений, что делает обучение более стабильным и предсказуемым. Это делается с помощью метода «конуса ограничений», который гарантирует наименьшее изменение политики в пределах заданного доверительного интервала. TRPO можно считать алгоритмом второго порядка, так как использует производные второго порядка [5].

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) - это алгоритм, являющийся улучшенной версией TRPO, который решает проблему сложности оптимизации ограничений на обновления политики. PPO использует обновления политики, которые ограничены близостью к исходной политике, но в более простой и более стабильной форме, чем TRPO. Это делает PPO более эффективным в плане вычислительных ресурсов и времени обучения.

Формула обновления политики в алгоритме PPO выглядит следующим образом:

$$\theta_{i+1} = \theta_i + \alpha min(\nabla_\theta \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_w(s_t, a_t), \\ clap(\nabla_\theta \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_w(s_t, a_t), \epsilon)), \tag{7}$$

где θ - обучаемые параметры политики агента,

α - скорость обучения для политики,

$\pi_\theta(a|s)$ - вероятность выбора действия a в состоянии s по текущей политике θ,

$A_w(s, a)$ - функция отражающая среднюю по состоянию s полезность выбора действия a,

∇_θ - градиент по параметрам θ,

ϵ - коэффициент определяющий степень изменения политики.

Для обновления оценочной функции:

$$w_{j+1} = w_i + \beta \nabla_w A_w^2(s_t, a_t), \tag{8}$$

где w - обучаемые параметры критика,

β - скорость обучения для оценочной функции,

$A_w(s, a)$ - функция отражающая среднюю по состоянию s полезность выбора действия a,

∇_w - градиент по параметрам w.

2.3 Выбор алгоритма

Для обучения на Atari играх можно рассмотреть все алгоритмы: Reinforce, Advantage Actor-Critic (AC), Advantage Actor-Critic (A2C), Trust Region Policy Optimization (TRPO) и Proximal Policy Optimization (PPO). Каждый из этих алгоритмов имеет свои преимущества и недостатки, и выбор оптимального алгоритма зависит от требований задачи и ресурсов доступных для обучения.

Среди предложенных вариантов, PPO является одним из наиболее популярных алгоритмов для обучения на Atari играх, и его использование можно обосновать по нескольким причинам:

- Эффективность обучения: PPO предлагает оптимизацию политики с использованием проксимальных методов, что позволяет стабильно и эффективно обучаться на большом количестве эпизодов в средах с высокими размерностями состояний и действий, таких как Atari игры.
- Стабильность: PPO имеет встроенные механизмы для контроля скорости обучения, что помогает избежать резких изменений политики и улучшает стабильность обучения, особенно в случаях с большими изменениями награды и состояний окружающей среды.
- Использование параллелизации: PPO может быть легко распараллелен, что позволяет использовать множество процессоров или графических процессоров для ускорения процесса обучения и снижения времени, необходимого для получения хороших результатов.
- Гибкость: PPO позволяет настраивать различные гиперпараметры, такие как размер батчей, коэффициенты сноса, коэффициенты регуляризации и другие, что дает возможность тонко настраивать алгоритм под конкретную задачу и улучшать его производительность.
- Простота реализации: PPO имеет относительно простую структуру и реализацию, что упрощает его применение и адаптацию в различных средах и сценариях.

3 ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1 Среда и её особенности

Проблема окружающей среды для исследователей

В обучении с учителем базовый программный стек обычно состоит только из трех компонентов: набора данных, предварительной обработки набора данных и библиотеки глубокого обучения. В обучении с подкреплением программный стек намного сложнее. Он начинается с создания самой среды, обычно это часть программного обеспечения, такого как симуляция или видеоигра. Затем логика базовой среды оборачивается API-интерфейсом, к которому можно применить обучающий код.

В зависимости от того, как алгоритм обучения с подкреплением взаимодействует с окружающей средой, затем применяются оболочки предварительной обработки (например, для создания изображений в оттенках серого). Только после того, как все это сделано, можно применять алгоритм обучения с подкреплением, который обычно реализуется с помощью инструментов глубокого обучения (например, PyTorch, TensorFlow, Jax). Сравнение программных стеков показано в таблице 2.

Таблица 2 – Сравнения стека обучения с учителем и обучения с подкреплением [6]

Стек обучения с учителем	Стек обучения с подкреплением
Набор данных	Сред
Предварительная обработка	Оболочка среды
–	Обертки предварительной обработки
–	RL-алгоритм
Библиотека глубокого обучения	Библиотека глубокого обучения

Зависимость обучения с подкреплением от стандартных программных сред, а не от наборов данных, создает множество уникальных проблем в этой области. Имея фрагмент кода классификации изображений, вы можете передать ему любой достаточно большой помеченный набор данных изображения, и он в основном будет работать. В отличие от наборов данных, когда вы имеете дело с программными средами, такими как игры, вы должны

убедиться, что среда работает на вашей операционной системе и архитектуре ЦП, не содержит ошибок, работает с современными версиями базовых инструментов (например, с текущей версией Python) и имеет API, совместимый с вашим учебным кодом. Вы также хотели бы, чтобы он имел документацию, был доступен для установки через диспетчер пакетов и имел специальные функции воспроизводимости.

Ранее проекты среды RL поддерживались и принадлежали на GitHub отдельным группам или отдельным лицам. Это означает, что если кто-то внезапно увольняется, сгорает или попадает под автобус , или компания разоряется, техническое обслуживание полностью прекращается. Даже стандартный подход с открытым исходным кодом, когда кто-то разветвляет среду, здесь нежелателен, потому что всем нужно использовать одну и ту же версию среды для воспроизводимости и согласованности в исследовательской области.

Помимо проблем качества, воспроизводимости и производительности, которые это создает для исследователей, реальность такова, что, пока существует область обучения с подкреплением, необходимы стандартные поддерживаемые среды, которые могут позволить сравнивать производительность.

Для решения этих проблем была создана некоммерческая организация Farama Foundation, которая занимается продвижением области обучения с подкреплением через продвижение лучшей стандартизации и инструментов с открытым исходным кодом как для исследователей, так и для промышленности. Они поддерживают три широко используемые библиотеки ядра, которые предлагают стандартные API между обучением с подкреплением различных видов сред.

Проблема стандартного API и происхождение фонда Farama Foundation

Чтобы иметь стандартизированные среды и модульный код RL в целом, необходим хорошо продуманный и простой в использовании стандартный API для доступа к средам обучения с подкреплением. В большинстве случаев это уже существует в библиотеке Python под названием Gym. Gym изначально был создан OpenAI 6 лет назад и включает в себя стандартный API,

инструменты для приведения сред в соответствие с этим API, а также набор различных эталонных сред, которые стали очень широко использоваться в качестве эталонных тестов. Он был установлен более 43 миллионов раз через pip, более 4500 раз процитирован в Google Scholar и используется более чем в 32 000 проектов на GitHub. Это делает ее самой используемой библиотекой RL в мире.

Farama Foundation фактически начал с разработки PettingZoo , который по сути представляет собой ПО для мультиагентных сред. Среди прочего, его разработка включала стандартизацию или создание около 60 сред, включая первое добавление поддержки ALE для многопользовательских игр Atari.

Gym со многими вещами работал очень хорошо, но OpenAI не выделял на него существенных ресурсов после его первоначального выпуска. Обслуживание Gym постепенно уменьшалось, пока в конце 2020 года Gym полностью не обслуживался. В начале 2021 года OpenAI предоставил контроль над репозиторием Gym разработчикам из Farama Foundation.

С тех пор произошло гораздо больше развития, чем за предыдущие 5 лет после его выпуска. Обновления для Gym включали изменение основного API для решения давних проблем с дизайном , создание полноценного веб-сайта документации впервые, добавление средства проверки соответствия для API, исправление массовые ошибки на всех поверхностях кодовой базы, удаление нескольких зависимостей от давно устаревшего программного обеспечения и многое другое.

Gymnasium

Gymnasium – это новая версия Gym, которая будет поддерживаться в будущем. Его можно легко добавить в любую существующую кодовую базу, заменив import gym на import gymnasium as gym, а Gymnasium 0.26.2 в остальном такой же, как Gym 0.26.2.

Даже для самых крупных проектов обновление тривиально, если они обновлены до последней версии Gym. Это делается это для того, чтобы API, от которого зависит целая область, можно было поддерживать в нейтральном состоянии в течение длительного времени, и чтобы предоставить нам доступ к дополнительным разрешениям, чтобы мы могли иметь более продуктивный и устойчивый рабочий процесс разработки и выпуска. OpenAI не планирует

развивать Gym в будущем, поэтому это не создаст ситуации, когда сообщество разделится на две конкурирующие библиотеки.

Прямо сейчас Gymnasium доступен, и его можно установить его с помощью обычного файла pip install gymnasium. Многие крупные проекты уже согласились перейти на Gymnasium в ближайшем будущем, например, CleanRL и StableBaselines3 .

Разработчики в основном сосредоточились на обновлении векторизованных сред и повторной реализации всех встроенных сред в Gymnasium, чтобы они могли работать на аппаратных ускорителях, таких как графические процессоры, изменения, которые представляют собой потенциальное ускорение среды в 10 и 1000 соответственно. Эти изменения сделают обучение с подкреплением более доступным в образовании и позволят исследователям гораздо быстрее экспериментировать с новыми идеями.

Пример цикла обучения в gymnasium

На рисунке 2 представлен псевдокод цикла обучения с использованием gymnasium.

```
import gymnasium as gym

# Создание среды
env = gym.make('CartPole-v1')

# Инициализация переменных
num_episodes = 1000 # Количество эпизодов
max_steps = 500 # Максимальное количество шагов в эпизоде

# Цикл обучения
for episode in range(num_episodes):
    state = env.reset() # Сброс состояния среды в начальное состояние
    total_reward = 0 # Инициализация суммарной награды для текущего эпизода
    for step in range(max_steps):
        env.render() # Визуализация текущего состояния среды
        action = env.action_space.sample() # Выбор случайного действия
        next_state, reward, done, _ = env.step(action) # Выполнение действия и получение следующего состояния, награды и флага завершения эпизода
        total_reward += reward # Добавление награды к суммарной награде текущего эпизода
        if done:
            print(episode+1, num_episodes, step+1, total_reward)
            break
        state = next_state # Обновление текущего состояния
env.close() # Закрытие среды

```

Рисунок 2 – Цикл обучения

3.2 Особенности реализации PPO

Сначала мы представим 10 основных деталей реализации PPO, которые обычно используемых независимо от задач.

Векторизованная архитектура

PPO использует эффективную парадигму, известную как

векторизованная архитектура, которая включает одного обучающегося, который собирает образцы и учится в нескольких средах. На рисунке 3 приведен псевдокод использования векторизованной архитектуры среды.

```
envs = VecEnv(num_envs=N)
agent = Agent()
next_obs = envs.reset()
next_done = [0] * N # of length N
for update in range(1, total_timesteps // (N*M)):
    data = []
    # ROLLOUT PHASE
    for step in range(0, M):
        obs = next_obs
        done = next_done
        action, other_stuff = agent.get_action(obs)
        next_obs, reward, next_done, info = envs.step(
            action
        ) # step in N environments
        data.append([obs, action, reward, done, other_stuff]) # store data

    # LEARNING PHASE
    agent.learn(data, next_obs, next_done) # `len(data) = N*M`

```

Рисунок 3 – Векторизованная архитектура

В этой архитектуре PPO сначала инициализирует векторизованную среду $envs$, которая запускает N, как правило независимых сред, либо последовательно, либо параллельно, используя несколько процессов. $envs$ представляет синхронный интерфейс, который всегда выводит пакет N наблюдения из N сред и принимает пакет из N действий, чтобы сделать шаг в средах.При вызове $next\_obs = envs.reset()$, $next\_obs$ возвращает батч с N начальными наблюдениями. PPO также инициализирует переменную-маркер завершения среды $next\_done$, которая представляет массив из N элементов. Каждый элемент массива может принимать два значения: 1, если игра окончена, иначе 0. Затем векторизованная архитектура зацикливает две фазы: rollout phase (фаза развёртывания) и learning phase (фаза обучения):

– Rollout phase: агент отбирает действия для N сред и продолжает

выполнять их в течение фиксированного числа M шагов. Во время этих M шагов агент продолжает добавлять соответствующие данные в пустой список $data$. Если i-я подсреда выполнена (завершена или усечена) после выполнения i-го действия $action[i]$, $envs[i]$ возвращает $next\_done[i]$ равной 1 и перезапускает i-ую подсреду;

– Learning phase: агент учится на собранных данных на этапе развертывания. В частности, PPO может оценить значение для следующего наблюдения $next\_obs$ зависящий от $next\_done$ и рассчитать преимущество(advantages) и returns, оба из которых также имеют длину $N * M$. Затем PPO учится на подготовленных данных.

Важно понимать роль $next\_obs$ и $next\_done$, чтобы помочь переходу между фазами: В конце j-й фазы развертывания $next\_obs$ можно использовать для оценки значения конечного состояния на этапе обучения и в начале (j + 1)-й этап развертывания $next\_obs$ становится начальным наблюдением в $data$. $next\_done$ сообщает, $next\_obs$ это первое наблюдение нового эпизода. Этот сложный дизайн позволяет PPO продолжать шагать подсредах, а поскольку агент всегда учится на сегментах траектории фиксированной длины после шаги, PPO может обучать агента, даже если подсреды никогда не прерываются и не усекаются. В принципе, поэтому PPO может обучаться в играх с длинным горизонтом, которые длятся 100 000 шагов (ограничение усечения по умолчанию для игр Atari в gym) в одном эпизоде.

Распространенной неправильной реализацией является обучение PPO на основе эпизодов и установка максимального горизонта эпизодов. У этого подхода есть несколько недостатков. Во-первых, это может быть неэффективно, поскольку агенту приходится выполнять один прямой проход на каждый шаг среды. Во-вторых, он не подходит для игр с более широкими горизонтами, таких как StarCraft II (SC2). Один эпизод SC2 может длиться 100 000 шагов, что увеличивает требования к памяти в этой реализации.

Векторизованная архитектура обрабатывает эти 100 000 шагов, обучаясь на сегментах траектории фиксированной длины . Если мы установим $N = 2$ и $M = 100$, агент будет учиться на первых 100 шагах из 2 независимых сред. Затем обратите внимание, что $next\_obs$ является 101-м наблюдением из этих двух сред, и агент может продолжать развертывание и учиться на шагах от

101 до 200 из 2 сред. По сути, агент изучает частичные траектории эпизода, M шагов за раз.

N — это количество сред, а $M * N$ — это размер итерации, которые предполагают, что увеличение N (например, $N = 256$) повышает скорость обучения, но ухудшает качество. Ухудшение качества произошло из-за «укороченных фрагментов опыта» (M становится меньше из-за увеличения N в их настройке) и «более ранней загрузки стоимости». Хотя мы согласны с тем, что увеличение N может снизить эффективность выборки, мы утверждаем, что оценка должна основываться на эффективности настенных часов. То есть, если алгоритм завершается намного раньше при большем N по сравнению с другими конфигурациями, почему бы не запустить алгоритм дольше?

Векторизованные среды также поддерживают среды многоагентного обучения с подкреплением (MARL). Например, если есть игра для двух игроков, мы можем создать векторизованную среду, которая порождает две подсреды. Затем векторизованная среда создает пакет из двух наблюдений, где первое наблюдение — от игрока 1, а второе наблюдение — от игрока 2. Затем векторизованная среда выполняет пакет из двух действий и сообщает игровому движку, чтобы позволить игроку 1 выполнить первое действие, а игрок 2 выполняет второе действие. Следовательно, PPO учится управлять как игроком 1, так и игроком 2 в этой векторизованной среде.

Ортогональная инициализация весов и постоянная инициализация смещений

Как правило, веса скрытых слоев используют ортогональную инициализацию весов с масштабированием $np.sqrt(2)$, а смещения устанавливаются равными 0. Однако веса выходного слоя политики инициализируются со шкалой 0.01, а веса выходного слоя значений инициализируются со шкалой 1. Однако стоит отметить, что это детализация очень низкого уровня, которая не должна влиять на производительность.

Эпсилон параметр Adam оптимизатора

PPO устанавливает для параметра $\epsilon = 10^{-5}$, которое отличается от

значения $\epsilon = 10^{-8}$ по умолчанию в PyTorch и $\epsilon = 10^{-7}$ TensorFlow. Мы перечисляем эту деталь реализации, потому что параметр epsilon не упоминается ни в документе, ни в качестве настраиваемого параметра в реализации PPO. Хотя эта деталь реализации может показаться слишком специфической, важно, чтобы мы соответствовали ей для воспроизведения с высокой точностью.

Затухание скорости обучения

Скорость обучения оптимизатора Adam может быть либо постоянной, либо затухающей. По умолчанию гиперпараметры для обучающих агентов, играющих в игры Atari, задают линейное уменьшение скорости обучения от $2.5 * 10^{-4}$ до 0 по мере увеличения количества временных шагов. Затухание скорости обучения Адама помогает агентам получать более высокую эпизодическую отдачу.

Generalized Advantage Estimation(GAE)

Две важные детали:

- Value bootstrap: если подсреда не завершена и не усечена, PPO оценивает значение следующего состояния в этой подсреде как целевое значение.
- $TD(\lambda)$ return estimation: PPO реализует цель возврата как $returns = advantages + values$, что соответствует $TD(\lambda)$ для оценки стоимости.

Хотя документ PPO использует абстракцию оценки преимущества в задаче PPO, реализация PPO использует обобщенную оценку преимущества.

Обновление минибатчами

Некоторые распространенные неправильные реализации включают в себя:

- Постоянное использование всего пакета для обновления.
- Реализацию минибатча путем случайной выборки из обучающих данных (что не гарантирует получение всех точек обучающих данных).

На этапе обучения векторизованной архитектуры реализация PPO перемешивает индексы обучающих данных размера $N *$ и разбивает его на мини-пакеты для вычисления градиента и обновления политики.

Нормализация преимуществ

После расчета преимуществ на основе GAE, PPO нормализует преимущества путем вычитания их среднего значения и деления на стандартное отклонение. В частности, эта нормализация происходит на уровне мини-пакета, а не на уровне всего пакета.

Отсечение потери функции значения

PPO соответствует сети ценности, сводя к минимуму следующие потери:

$$L^V = max((V_{\theta_t} - V_{targ})^2, (clip(V_{\theta_t}, V_{\theta_{t-1}} + \epsilon, V_{\theta_{t-1}} - \epsilon) - V_{targ})^2), \qquad (9)$$

где L^V — value loss,
V_{θ_t} — предсказанный value с новыми весами,
$V_{\theta_{t-1}}$ — предсказанный value со старыми весами,
V_{targ} — посчитанный value,
ϵ — коэффициент обрезания.

Общий Loss

Общий Loss рассчитывается как

$$loss = policy\_loss - c1 * entropy + c2 * value\_loss, \qquad (10)$$

где $c1$ — коэффициет энтропии,
$c2$ — коэффициет потери значения.
Параметры политики и параметры значений используют один и тот же оптимизатор.

Отсечение градиента

Для каждой итерации обновления в эпоху PPO перемасштабирует градиенты политики и сети значений так, чтобы «глобальная норма l2» не превышала 0.5.

3.3 Особенности реализации PPO для игр Atari

Далее разберём 9 деталей реализации, специфичных для Atari.

Использование NoopResetEnv

Эта оболочка выбирает начальные состояния, беря случайное число (от 1 до 30) отсутствия операций при сбросе.

Использование MaxAndSkipEnv

Эта оболочка по умолчанию пропускает 4 кадра, повторяет последнее действие агента над пропущенными кадрами и суммирует вознаграждения в пропущенных кадрах. Такой метод пропуска кадров может значительно ускорить алгоритм, потому что шаг окружения в вычислительном отношении дешевле, чем прямой проход агента. Эта оболочка также возвращает максимальные значения пикселей за последние два кадра, чтобы помочь справиться с некоторыми особенностями игры Atari.

Использование EpisodicLifeEnv

В играх, где есть счетчик жизней, таких как breakout, эта обертка отмечает конец жизни как конец эпизода.

Использование FireResetEnv

Эта оболочка выполняет $FIRE$ действие при сбросе для сред, которые зафиксированы до запуска.

Использование WarpFrame

Эта обертка изменяет размер изображения. У нас по умолчанию 210x160 пикселей меняется на 84x84.

Использование ClipRewardEnv

Эта обертка заменяет вознаграждения на $+1, 0, -1$.

Использование FrameStack

Эта оболочка складывает несколько последних кадров, чтобы агент мог сделать вывод о скорости и направлении движущихся объектов.

Масштабирование изображений

Входные данные имеют диапазон [0,255], но они делятся на 255, чтобы быть в диапазоне [0,1]. Без него первое обновление политики приводит к сильному расхождению Кульбака-Лейблера, вероятно, из-за того, как инициализируются слои.

3.4 Особенности реализации PPO с LSTM слоем

Инициализация слоя для слоев LSTM

Веса слоев LSTM инициализируются с помощью $std = 1$, а смещения инициализируются с помощью 0.

Сбросить состояния LSTM в конце эпизода

Во время развертывания или обучения агенту передается флаг конца эпизода, чтобы он мог сбросить состояния LSTM на нули.

Подготовьте последовательные развертывания в мини-пакетах

В настройках, отличных от LSTM, мини-пакеты извлекают случайно проиндексированные обучающие данные, поскольку порядок обучающих данных не имеет значения. Однако порядок обучающих данных имеет значение в настройке LSTM. В результате мини-пакеты извлекают последовательные обучающие данные из подсред.

3.5 Архитектуры нейронных сетей

В таблице 3 показаны различия в архитектурах нейронных сетей алгоритма PPO и PPO с LSTM.

Таблица 3 – Архитектуры нейронных сетей

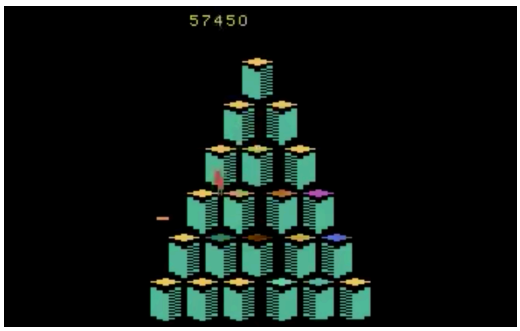
Общая часть	
Conv2d(4, 32, 8, stride=4) Conv2d(32, 64, 4, stride=2) Conv2d(64, 64, 3, stride=1) Linear(64 * 7 * 7, 512)	
PPO	PPO с LSTM
–	LSTM(512, 128)
Linear(512, action_dim)	Linear(128, action_dim)
Linear(512, 1)	Linear(128, 1)

Добавление LSTM слоя позволяет снизить $stack\_frame$ до 1, так как он в своём представлении запоминает предыдущие состяния, что позволёт сетям отслеживать динамику среды без конкатенации нескольких состояний.

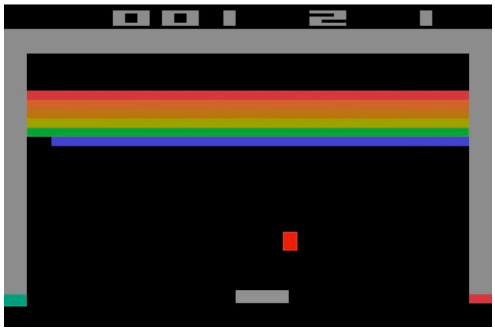
4 РЕЗУЛЬТАТЫ РАБОТЫ

4.1 Тестируемые игры

На рисунках 4 и 5 изображены интерфейсы трёх тестируемых игр.



а) breakout



б) qbert

Рисунок 4 – Тестируемые игры

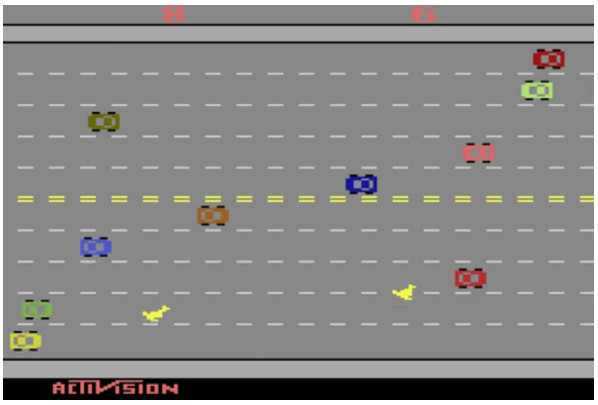


Рисунок 5 – freeway

Опишем каждую их них:

— Breakout: цель игры заключается в том, чтобы разбить все блоки, расположенные в верхней части экрана, с помощью мячика и платформы. Игрок управляет платформой, которая находится внизу экрана, и отбивает мячик, который летит вверх и сталкивается со стенами и блоками. Когда мячик сталкивается с блоком, блок исчезает, а мячик отскакивает. Игрок должен управлять платформой так, чтобы мячик не упал за пределы экрана. Если мячик упадет, то игрок потеряет одну жизнь из трёх. Как

начисляются баллы: красный блок- 7, оранжевый блок - 7, желтый блок - 4, зеленый - 4, блок цвета морской волны - 1, синий блок - 1;

— Qbert: аркадная игра, выпущенная компанией Gottlieb в 1982 году, и позже портированная на различные платформы, включая Atari 2600. Игра представляет собой платформер с головоломками, где игрок управляет существом по имени Qbert и пытается изменить цвета пирамид, избегая врагов. Основная цель игры Qbert - изменить цвета всех блоков на пирамиде, перепрыгивая по ним и избегая врагов. Игровое поле представляет собой трехмерную пирамиду, состоящую из блоков разных цветов. Каждый блок имеет определенное состояние, и задача игрока - изменить цвет всех блоков. Игрок управляет Qbertom, созданием, похожим на оранжевый сферический существа с носом. Qbert может перепрыгивать с одного блока на другой, меняя их цвет. Перепрыгивая на блоки, Qbert меняет их цвет, превращая их в целевой цвет. Например, если блок начинает синим, Qbert должен перепрыгнуть на него, чтобы он стал желтым. По пути Qbert-а есть различные враги, такие как Coily - змееподобное создание, и другие, которые мешают прогрессу игрока. Если Qbert касается врага или падает с пирамиды, он теряет жизнь. Игра состоит из нескольких уровней, и каждый уровень представляет собой новую конфигурацию пирамиды и больше врагов. Цель состоит в том, чтобы пройти все уровни и набрать максимальное количество очков. На игровом поле могут присутствовать различные объекты, такие как пружины, которые помогают Qberty перемещаться на разные блоки или отталкивающие мячи, которые могут оттолкнуть врагов. Игра продолжается до тех пор, пока у игрока есть жизни. Если все жизни исчерпаны, игра завершается;

— Freeway: аркадная игра для Atari 2600, выпущенная компанией Activision в 1981 году. Игра представляет собой симуляцию перехода дороги, где игрок управляет курицей, пытающейся пересечь проезжую часть безопасно. Основная цель игры Freeway - помочь курице пересечь дорогу, избегая автомобилей, движущихся по проезжей части. Экран игры разделен на несколько горизонтальных полос, которые представляют дорожные полосы.

Курица находится на нижней части экрана, а автомобили движутся по верхней части. Игрок управляет курицей, позволяя ей перемещаться только вверх или вниз, чтобы пересечь все полосы и достигнуть противоположного конца экрана. На дорогах движется несколько автомобилей, движущихся в разных направлениях и с разной скоростью. Игрок должен умело маневрировать, чтобы избежать столкновения с автомобилями. Если курица сталкивается с автомобилем, она получает ”тряску”и перемещается на начальную позицию. Игрок может продолжать попытки до тех пор, пока не достигнет противоположного конца экрана. Каждое успешное пересечение дороги награждается очками. Очки могут быть присуждены в зависимости от количества пересеченных полос и сложности прохождения уровня. С каждым новым уровнем сложность игры увеличивается. Автомобили становятся быстрее и движутся в большем количестве, что делает прохождение более сложным. Игра завершается после определённого количества времени. Максимальное количество очков, которое можно заработать – 34;

— Pong: классическая игра в видеопроекции, разработанная компанией Atari в 1972 году. Игра представляет собой виртуальный теннис, в котором два игрока управляют платформами на противоположных сторонах экрана, отражая мяч и пытаясь пропустить его в сторону противника. Цель каждого игрока - отбить мяч таким образом, чтобы противник не смог его поймать и отразить. Если мяч пролетает за платформу одного из игроков, то противник получает очко. Платформы могут двигаться вертикально вверх и вниз по полю, чтобы перехватывать мяч. Обычно управление осуществляется с помощью клавиш или джойстика. Игра продолжается до достижения заранее заданного количества очков, которое обычно равно 21;
— Beamrider: аркадная игра, выпущенная в 1983 году компанией Activision. В игре игрок управляет космическим кораблем и сражается с вражескими силами, находясь в космической атмосфере. Основная цель игры Beamrider - уничтожить вражеские объекты и заработать максимальное количество очков. Космический

корабль игрока находится на нижней части экрана и может двигаться горизонтально. Вражеские объекты появляются на верхней части экрана и медленно движутся вниз. Они могут быть различных форм и размеров, включая корабли, звездолеты и метеориты. Игрок должен уничтожить вражеские объекты, выпуская лазерные лучи в их сторону. Космический корабль может стрелять только вверх и горизонтально. При попадании лазерного луча в вражеский объект игрок получает очки. Очки могут быть назначены в зависимости от типа объекта и сложности его уничтожения. В процессе игры появляются препятствия, такие как метеоритные поля или энергетические барьеры, которые игрок должен избегать или уничтожать. С каждым новым уровнем сложность игры увеличивается. Вражеские объекты становятся быстрее и появляются в большем количестве. Также могут появляться боссы, с которыми игрок должен сражаться. Игра продолжается, пока у игрока есть жизни. Жизни могут быть потеряны при столкновении с врагами или их выстрелами. Если игрок исчерпывает все жизни, игра завершается.

4.2 Вычислительные ресурсы

Для обучения было использовано 8 процессоров и 1 видеокарты от Nvidia. В таблицы 4 сравнивается время необходимое для обучения алгоритм ов для трёх игр.

Таблица 4 – Время обучения

PPO	PPO с LSTM
breakout	
9 ч. 45 мин.	13 ч. 10 мин.
qbert	
2 ч. 25 мин.	4 ч. 30 мин.
freeway	
3 ч. 45 мин.	4 ч. 15 мин.

4.3 Графики

На рисунках 6, 7, 8, 9, 10 приведены графики суммы вознаграждений. Для сравнения я использовал модифицированный PPO алгоритм с добавлением LSTM слоя.

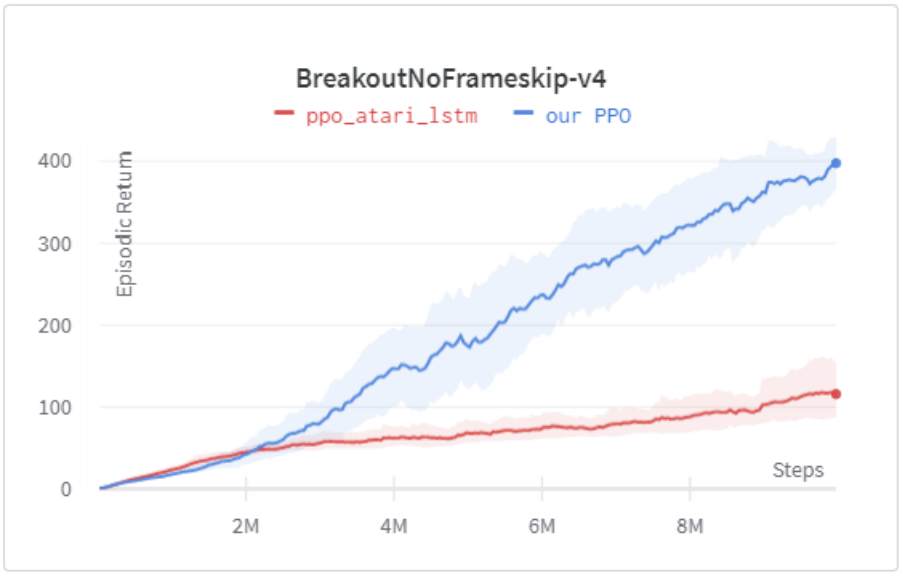


Рисунок 6 – Breakout

Из рисунка 6 видно, что классический алгоритм PPO лучше обучилось взаимодействовать, чем PPO с LSTM. Также мы смогли достигнуть результатов представленных в статье [7].

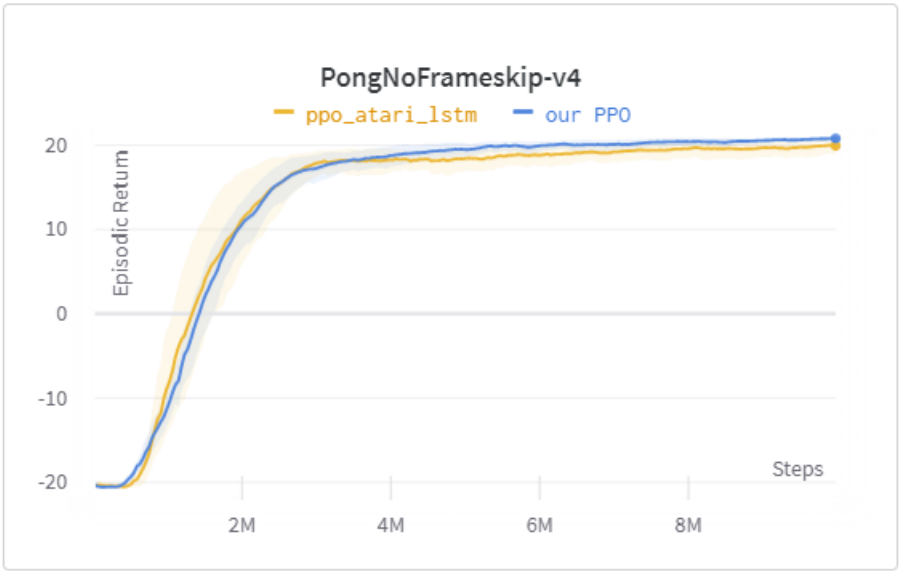


Рисунок 7 – Pong

Из рисунка 7 видно, что классический алгоритм PPO и PPO с LSTM

смогли достигнуть примерно одинаковых результатов. Также мы смогли достигнуть результатов представленных в статье [7].

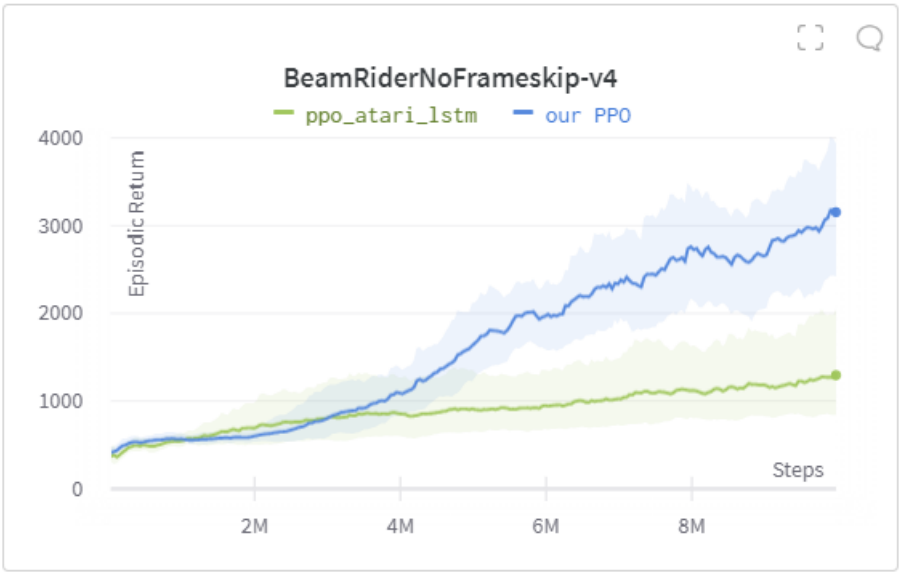


Рисунок 8 – Beamrider

Из рисунка 8 видно, что классический алгоритм PPO сильно лучше справился, чем PPO с LSTM смогли достигнуть примерно одинаковых результатов. Также мы смогли достигнуть значительно лучшие результаты, чем резульаты представленные в статье [7].

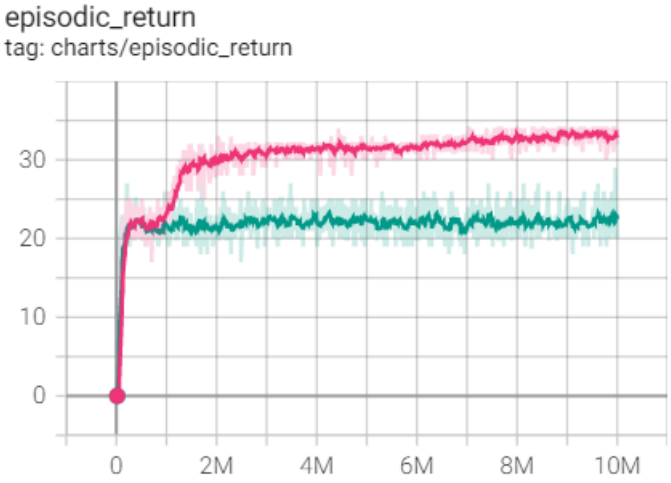


Рисунок 9 – Freeway

Из рисунка 9 видно, что классический алгоритм PPO смог достигнуть максимального результата по игре freeway. PPO с LSTM упёрся в порог 20-22. Обычно такой результат появляется, когда агент выучил движение «только вперёд». При такой политике он может получать суммарное вознаграждение

около 20. Также мы смогли достигнуть результаты представленные в статье [7].

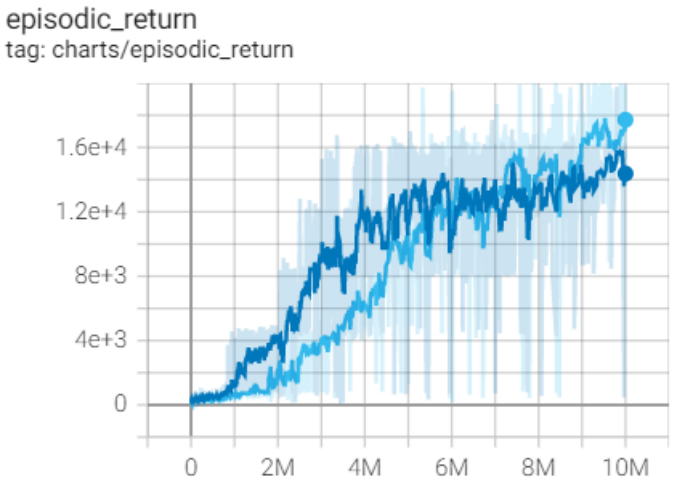


Рисунок 10 – Qbert

Из рисунка 10 видно, что на игре qbert уже PPO с LSTM смог достигнуть более высокие результаты, чем классический PPO. Оба алгоритма смогли превзойти результаты представленные в статье [7].

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе бакалавра были исследованы различные алгоритмы обучения с подкреплением на основе градиента стратегии. Основной целью работы было выбрать наиболее эффективный алгоритм и применить его к играм Atari.

В ходе исследования был выбран алгоритм PPO (Proximal Policy Optimization), который считается одним из наиболее прогрессивных методов обучения с подкреплением. Данный алгоритм был реализован и применён к набору игр Atari. Достижение результатов, описанных в соответствующей научной статье, являлось одной из основных целей работы.

Кроме того, был исследован модифицированный вариант PPO, в котором использовался LSTM (Long Short-Term Memory) слой в нейронных сетях. LSTM представляет собой рекуррентный слой, способный моделировать долгосрочные зависимости в последовательностях данных. Модифицированный алгоритм был также применен к играм Atari, и полученные результаты были сравнены с базовой реализацией PPO.

Анализ полученных графиков и данных привел к выводу, что модификация с использованием LSTM слоя привела к улучшению результатов только для игры Qbert. В остальных играх применение LSTM слоя не дало значительного преимущества по сравнению с базовым алгоритмом PPO.

Одним из важных аспектов было то, что в работе было уделено внимание деталям реализации алгоритмов. Это было сделано с целью обеспечить максимально точное воспроизведение результатов, описанных в научной статье, и предотвратить возможные ошибки, которые могут возникнуть у других исследователей при реализации алгоритма.

Кроме того, в работе были обозначены проблемы, связанные с разработкой программного обеспечения и повышением эффективности алгоритма PPO. Особое внимание уделялось использованию ускоренных векторизованных сред, которые позволяют обрабатывать несколько параллельных сред одновременно. Это позволяет значительно повысить эффективность обучения и сократить время обучения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DeepMind. Playing Atari with Deep Reinforcement Learning. — 2013. — DOI: https://doi.org/10.48550/arXiv.1312.5602.

2. MachineLearningMastery. Машинное обучение для начинающих. — 2018. — URL: https://machinelearningmastery.ru/machine-learning-for-beginners-d247a9420dab/ (дата обращения 25.04.2023).

3. Лонца А. Алгоритмы обучения с подкреплением на Python. — Packt Publishing 2019, 2020.

4. IFMO. Policy gradient. — 2022. — URL: https://vk.cc/cnZkR0 (дата обращения 25.04.2023).

5. OpenAI. Trust Region Policy Optimization. — 2015. — DOI: https://doi.org/10.48550/arXiv.1502.05477.

6. Foundation F. Announcing The Farama Foundation. — 2022. — URL: https://farama.org/Announcing-The-Farama-Foundation (дата обращения 25.04.2023).

7. OpenAI. Proximal Policy Optimization Algorithms. — 2017. — DOI: https://doi.org/10.48550/arXiv.1707.06347.

ПРИЛОЖЕНИЕ А

Исходный код

Перейдя по QR-коду на рисунке А.1, можно скачать исходный код программы.

Рисунок А.1 – QR-code с исходными файлами программы