Московский государственный университет имени М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики

Магистерская программа
«Логические и комбинаторные методы анализа данных»

Магистерская диссертация

«Стратегии исследования окружений в обучении с подкреплением с непрерывными пространствами состояний»

Работу выполнил:
Гурьянов Алексей Константинович

Научный руководитель:
профессор, д.ф.-м.н.
Дьяконов Александр Геннадьевич

Москва
2017

Содержание

Аннотация

Для решения задач на стыке машинного обучения и оптимального управления была создана область обучения с подкреплением. В системах обучения с подкреплением выделяют важную проблему выбора между эксплуатацией текущих знаний для получения наибольшей награды и исследованием окружения с целью получения знаний о структуре окружения. В данной работе ставилась задача сравнения стратегий исследования окружений с непрерывными пространствами состояний. В качестве основы системы обучения с подкреплением была выбрана модель глубокого Q-обучения, показавшая в недавних исследованиях хорошие результаты на подобных задачах. Были предложены модификации существующих стратегий исследования и новая стратегий исследования окружений в обучении с подкреплением. Было обнаружено, что не существует стратегии исследования, показывающей наилучшее качество на всех окружениях. Предложенный метод показал наилучшее качество на одном из рассматриваемых окружений.

1 Введение

С развитием современных технологий человечество ставит себе на службу системы с все более и более возрастающей сложностью. При разработке таких систем возникает проблема оптимального их контроля, которую зачастую невозможно решить реализацией какого-то определенного алгоритма. Для решения подобных задач на стыке областей машинного обучения и оптимального управления была создана область обучения с подкреплением, фокусирующаяся на обучении поведения агента в определенных окружениях для достижения определенного результата. Задачи, в которых эти методы нашли применения, варьируются от задач управления роботами и контроля индустриальных процессов до управления портфелями ценных бумаг и участием в компьютерных играх. В последнее время популярными стали алгоритмы обучения с подкреплением, использующие нейронные сети над состояниями окружений.

Задача обучения с подкреплением предполагает наличие агента, получающего определенную информацию о состоянии окружения и способного совершать действия, оказывающие влияние на окружение. Цель обучения с подкреплением состоит в обучении агента способу получения максимального значения числовой награды, которая выдается агенту при совершении определенных действий.

В работе агентов выделяют важную проблему выбора между эксплуатацией текущих знаний агента для получения наибольшей награды и исследованием окружения с целью получения знаний о структуре окружения. Эти два варианта выбора действий подразумевают под собой одну и ту же итоговую цель, один вариант максимизирует полученную награду в ближайшем будущем, а другой – в долгосрочном периоде.

Пространства состояний окружения можно разделить на две большие группы: дискретные пространства состояний, в которых каждая переменная окружения является конечной дискретной величиной, и непрерывные пространства состояний. в которых хотя бы одна переменная окружения является непрерывной величиной. Мощность множества всех возможных состояний конечна для дискретных пространств состояний и бесконечна для непрерывных пространств состояний.

Существуют стратегии исследования окружений, гарантированно приводящие агента к оптимальной стратегии действий на пространствах с небольшой мощностью множества всех возможных состояний. На задачах с большой или бесконечной мощностью множества всех возможных состояний такие методы становятся неприменимы из-за больших вычислительных затрат. В современных системах обучения с подкреплением применяются простые стратегии исследования окружений, которые не всегда позволяют найти оптимальную стратегию или находят ее слишком долго. Поэтому возникает

необходимости в разработке более эффективных алгоритмов исследования окружений.

Одна из трудностей в изучении стратегий исследования окружений состоит в том, что не существует принятого определения для описания оптимального действия с точки зрения исследования оптимальной структуры окружения. Оптимальное с эксплуатационной точки зрения действие определяется как действие, максимизирующее ожидаемую от этого действия награду. Дополнительные трудности в формализацию понятия исследования окружений вносит тот факт, что многие современные системы обучения с подкреплением не хранят информацию о влиянии действий агента на окружение.

Данная работа состоит из следующих разделов: постановка задачи и введение основных терминов обучения с подкреплением; обзор существующих систем обучения с подкреплением; обзор стратегий исследования окружений, описание предложенного метода исследования окружений, описание программной реализации; сравнение реализованных методов на примере различных окружений.

2 Постановка задачи

Основной целью данной работы является нахождение алгоритма исследования окружений, показывающего наилучшее качество на рассматриваемых функционалах.

Подзадачами данной работы является:

- Изучение существующих методов исследования окружений.
- Выделение наилучших методов исследования окружений с точки зрения функционалов качества.
- Построение нового метода исследования окружений и экспериментальное сравнение с существующими методами.

3 Задача обучения с подкреплением

3.1 Постановка задачи

Под задачей обучения с подкреплением понимается задача обучения из взаимодействия для достижения какой-то цели. Обучающаяся и принимающая решения единица называется агентом. Объект, с которым агент взаимодействует, называется окружением. Агент и окружение постоянно взаимодействуют, агент выбирает действия и окружение отвечает на эти действия и предоставляет новое состояние агенту. Окружающая среда также порождает награду, специальное численное значение, которое агент пытается максимизировать за время своей работы.

Формально, агент и окружение взаимодействуют на каждом шаге последовательности дискретных временных шагов $t = 0, 1, 2, 3, \ldots$. На каждом шаге агент принимает состояние $S_t \in S$, где S является множеством всех возможных состояний. На основании состояния агент выбирает действие $A_t \in A(S_t)$, где $A(S_t)$ описывает множество действий, доступных агенту в состоянии S_t. На следующем временном шаге агент получает численную награду $R_{t+1} \in R$, а также новое состояние, S_{t+1}. На каждом временном шаге агент сопоставляет состоянию набор вероятностей выбора каждого доступного действия. Это сопоставление называется стратегией агента, и описывается π_t, где $\pi_t(a|s)$ описывает вероятность того, что действие $A_t = a$ будет выбрано в состоянии $S_t = s$. Методы обучения с подкреплением определяют способ, которым агент меняет свою стратегию в результате полученного опыта. Целью агента, грубо говоря, является максимизация награды, полученной агентом за все время его работы [1].

Использование награды для формализации понятия цели является одной из отличительных характеристик задач обучения с подкреплением. На практике подобная техника показал свою гибкость и широкую применимость. Например, для того, чтобы обучить робота ходить, исследователи предоставляли награду на каждом временном шаге пропорциональную достигнутой роботом скорости движения вперед. Для обучения робота побегу из лабиринта, награда часто задается равной нулю, до момента действия, приводящего к успешному побегу, после которого она задается равной единице.

Награды не должны включать в себя поощрения о том, как агенту следует достигать поставленной цели. В случае, например, шахмат это означает, что мы не должны включать в награду подцели, такие как удержание центра доски или взятие вражеских фигур. Агент может найти способ достижения подцелей без достижения главной цели, пытаясь взять как можно больше вражеских фигур даже ценой проигранной партии.

Пусть последовательность наград, получаемых после временного шага t, записываются как $R_{t+1}, R_{t+2} \ldots$. Тогда в общем случае в задаче обучения с подкреплением необ-

ходимо максимизировать ожидаемую награду, G_t, которая определена как некоторая функция от последовательности наград.

Самый простой способ заключается в сумме всех последующих наград:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{1}$$

где T задает последний временной шаг. Этот подход имеет смысл только при существовании в задаче понятия о последнем временном шаге, что приводит к разбиению взаимодействия агента и окружения на подпоследовательности - эпизоды. Примерами эпизода может служить одна партия игры, одно путешествие в лабиринте или одно какое-либо другое повторяющееся взаимодействие. Каждый эпизод заканчивается особым терминальным состоянием, за которым следует переход к стандартному начальному состоянию или к выбору начального состояния из распределения начальных состояний. Задачи с подобными эпизодами называются эпизодическими задачами.

Однако во многих случаях взаимодействие агента и окружения не разбивается на легко опознаваемые эпизоды, а продолжается постоянно без предела. Такие задачи называются продолжительными задачами. В подобных задачах применение формулы (1) невозможно, так как последний временной шаг не существует, а ожидаемая награда, которую мы стремимся максимизировать, может достигать бесконечности. Для решения задач с такими особенностями вводится обесценивание награды. В соответствии с этим подходом, агент пытается выбрать действия таким образом, чтобы сумма обесценивающихся получаемых наград была наибольшей. То есть выбирается такое действие A_t, которое максимизирует ожидаемую обесцененную награду:

$$G_t = R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \cdots = \textstyle\sum_{k=0}^{\infty} \lambda^k R_{t+k+1}, \tag{2}$$

где λ задает параметр обесценивания, $0 \leq \lambda < 1$.

Параметр обесценивания задает ценность наград в будущем. Если $\lambda < 1$, то бесконечная сумма имеет конечное значение при условии ограниченности последовательность наград R_k. Если $\lambda = 0$, то агент учитывает только награду текущего шага, и действие выбирается с целью максимизации R_{t+1}.

3.2 Марковские процессы

Процесс обладает свойством Маркова, если все последующие изменения в процессе возможно описать только по последнему этапу процесса. В контексте задач обучения с подкреплением процесс удовлетворяет свойству Маркова, если для любых возможных значений прошедших состояний, действий и наград: $S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t,$

вероятность получения награды r и перехода в состояние s' удовлетворяет формуле:

$$\Pr\{R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, \ldots . S_{t-1}, A_{t-1}, R_t, S_t, A_t\} = \Pr\{R_{t+1} = r, S_{t+1} = s' | S_t, A_t\}.$$

Если окружение обладает свойством Маркова, то история одного последнего шага может быть использована для расчета следующего состояния и ожидаемой награды для текущих состояния и действия.

Задача обучения с подкреплением, которая удовлетворяет свойству Маркова называется Марковским процессом принятия решения (в дальнейшем МППР). Если пространство состояний и действий конечны, то задача называется конечным Марковским процессом принятия решений (конечным МППР).

Каждый конечный МППР определен его наборами состояний и действий и одношаговыми правилами переходов. Для любого состояния s и действия a вероятность каждого возможного следующего состояния s' равна:

$$p(s'|s, a) = \Pr\{S_{t+1} = s' \mid S_t = s,\ A_t = a\}.$$

Для любого текущего состояния и действия s и a, вместе с каждым следующим состоянием s' ожидаемая функция награды r равна:

$$r(s, a, s') = \mathrm{E}[R_{t+1} \mid S_t = s,\ A_t = a,\ S_{t+1} = s'],$$

где E обозначает математическое ожидание случайной величины.

3.3 Функция ценности

Большинство алгоритмов обучения с подкреплением включает в себя понятие функции ценности. Функция ценности - это функция от состояния (или от пары состояния и действия), которая оценивает, насколько ценно для агента пребывать в заданном состоянии (или насколько ценно применить данное действие в данном состоянии). Понятие о том, насколько ценно какое-то действие или состояние выражается в понятии будущей ожидаемой награды. Награда, которую агент может рассчитывать получить в будущем, зависит от принимаемых в дальнейшем действий, поэтому функции ценности определены для каждой стратегии принятия решений.

Вспомним, что стратегия π - это сопоставление состояния $s \in S$ и действия $a \in A(s)$ вероятности $\pi(a|s)$ принятия действия a в состоянии s. Ценность состояния s при стратегии π, равная $V_\pi(s)$, вычисляется как ожидаемая награда при начале действий агента из состояния s и выполнения политики π в будущем. Для Марковского процесса принятия решений мы можем записать $V_\pi(s)$ как:

$$V_\pi(s) = \mathrm{E}_\pi[G_t|S_t = s] = \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \lambda^k r_{t+k+1} \middle| S_t = s\right],$$

где $E_\pi\Big[\ \cdot\ \Big]$ показывает ожидаемое значение награды при действии агента в соответствии со стратегией π на каждом временном шаге. Заметим, что значение ценности терминального состояния всегда равно нулю. Функция V_π называется функцией ценности состояния для стратегии π.

Ценность действия a в состоянии s при стратегии π, обозначаемая $Q_\pi(s, a)$, определяется как ожидаемое значение награды начиная с состояния s, принимая действие a и в дальнейшем придерживаясь стратегии π:

$$\begin{aligned} Q_\pi(s,a) &= \mathrm{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \lambda^k R_{t+k+1} \middle| S_t = s, A_t = a\right], \end{aligned}$$

где Q_π называется функцией ценности действия для стратегии π.

Функции ценности V_π, Q_π могут быть оценены из опыта. Например, если агент следует стратегии π и для каждого состояния поддерживает среднее значений наград, которые следовали за данных состоянием, то это среднее будет сходится к среднему этого состояния, по мере того как число пребываний в этом состоянии стремится к бесконечности [1].

Фундаментальное свойство функций ценности, которое используется в методах обучения с подкреплением, заключается в том, что оно для каждой стратегии π и для каждого состояния s удовлетворяет определенному рекурсивному отношению:

$$\begin{aligned} V_\pi(s) &= E_\pi[G_t|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \lambda V_\pi(s')\Big]. \end{aligned}$$

Это соотношение называется уравнением Беллмана для V_π. Оно выражает отношение между значениями ценности состояния и значением ценности последующих состояний. Функция ценности состояния V_π выражает единственное решение уравнения Беллмана.

Решение задачи обучения с подкреплением означает нахождение стратегии, под действием которой агент получит наибольшую награду в долгосрочном периоде. Чтобы определить оптимальную стратегию, введем частичный порядок на пространстве стратегий на основании функции ценности. Стратегия π является больше либо равной стратегии π' если ожидаемая награда для стратегии π больше, чем ожидаемая награда для

стратегии π' для всех состояний. В такой системе частичного порядка будет хотя бы одна стратегия, которая больше либо равна всех остальных стратегий. Она будет называться оптимальной стратегией. Все оптимальные стратегии обозначаются через π_*. У таких стратегий одинаковые функции ценности состояния, которые называются оптимальными функциями ценности состояния:

$$V_*(s) = \max_\pi V_\pi(s),\ \forall s \in S.$$

У всех оптимальных стратегий одна и та же оптимальная функция ценности действия:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a),\ \forall s \in S,\ a \in A(s).$$

Для каждой пары состояния и действия эта функция выдает ожидаемую награду при взятии действия a в состоянии s, а затем придерживаясь оптимальной стратегии. Таким образом возможно выразить оптимальную функцию ценности действия через оптимальную функцию действия состояния:

$$Q_*(s,a) = \mathrm{E}[R_{t+1} + \lambda V_*(S_{t+1}) \mid S_t = s,\ A_t = a].$$

3.4 Фундаментальные методы решения

3.4.1 Динамическое программирование

Основная идея алгоритма динамического программирования заключается в использовании функций награды для структурирования поиска хороших стратегий [1]. Такие алгоритмы используют уравнение Беллмана в качестве правила обновления для улучшения приближений функций награды:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \lambda V_\pi(s')\Big], \forall s \in S.$$

Если схема функционирования окружения известна, то уравнение Беллмана сводится к $|S|$ линейным уравнениям с $|S|$ неизвестными ($V_\pi(S), s \in S$). Прямое решение подобной системы приведет к ответу, но будет требовать больших затрат вычислительных ресурсов. Часто для нахождения функции ценности используют итерационные методы нахождения линейной системы уравнений. Исходное приближение v_0 выбирается произвольным образом, а каждое последующее приближение находится через уравнение Беллмана, интерпретируемого как правило обновления:

$$\begin{aligned} V_{k+1} &= \mathrm{E}_\pi[R_{t+1} + \lambda V_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \lambda V_l(s')\Big]. \end{aligned}$$

Такой алгоритм называется итерационным оцениванием стратегий. Алгоритм заканчивает свою работу после того, как изменение оценки для каждого состояния становится достаточно маленьким. После вычисления функции награды для каждого состояния при фиксированной стратегии, алгоритмы динамического программирования используют ее для улучшения текущей стратегии. Для обоснования выбора улучшенной стратегии используется теорема об улучшении стратегии, которая гласит, что если для двух стратегий π и π' для любого состояния $s \in S$ верно, что $Q_\pi(s, \pi'(s)) \geq V_\pi(s)$, то стратегия π' даст большую ожидаемую награду на всех состояниях $s \in S$: $V_\pi \geq V_\pi(s)$.

В соответствии с теоремой улучшения стратегии мы можем создать такую стратегию, которая выбирает наилучшее действие в каждом состоянии в соответствии с данной функцией награды:

$$\begin{aligned} \pi'(s) &= \text{argmax}_a Q_\pi(s, a) \\ &= \text{argmax}_a \text{E}[R_{t+1} + \lambda V_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \text{argmax}_a \sum_{s'} p(s'|s, a) \Big[r(s, a, s') + \lambda V_\pi(s') \Big]. \end{aligned}$$

Такая стратегия соответствует условию теоремы улучшения стратегии, поэтому она будет не хуже, чем исходная стратегия π. Причем если функция награды новой стратегии не улучшает исходную стратегию, то из правила обновления стратегии будет следовать, что:

$$\begin{aligned} V_{\pi'}(s) &= max_a E[R_{t+1} + \lambda V_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \\ &= max_a \sum_{s'} p(s'|s, a) \Big[r(s, a, s') + \lambda V_{\pi'}(s') \Big]. \end{aligned}$$

Эта формула является записью уравнения Беллмана, из чего будет следовать, что обе стратегии π и π' являются оптимальными. Таким образом процесс улучшения стратегии выдает строго лучшую стратегию, за исключением тех случаев, когда текущая стратегия оптимальная.

После построения улучшенной стратегии π' по исходной стратегии π и ее функции награды V_π, новую стратегию π' можно таким же образом оценить и улучшить. Таким образом последовательное применение оценивания и улучшения стратегий выдает последовательность монотонно улучшающихся стратегий и функций награды. Такой способ нахождения оптимальной стратегии называется итерированием стратегий.

Большой недостаток алгоритмов динамического программирования состоит в том, что их применение включает в себя операции, использующие все множество возможных

состояний. Если пространство этих состояний очень велико, то даже один проход по всему этому пространству может быть слишком вычислительно дорог.

3.4.2 Методы Монте-Карло

Методы Монте-Карло (МК) не предполагают полного знания об окружающей среде, в отличие от динамического программирования. Методы Монте-Карло требуют наличие опытных данных, под которым понимается набор последовательностей состояний, действий и наград, полученных в результате онлайн или смоделированного взаимодействия с окружающей средой. Хотя для смоделированных опытных данных и требуется модель, данной модели требуется только генерировать выборки перемещений, а не полное распределение всех возможных перемещений, как того требует динамическое программирование [1].

Методы Монте-Карло решают задачу обучения с подкреплением, основываясь на средней ценности выборки. Чтобы удостовериться, что ценность будет определенной, методы Монте-Карло определяются только для эпизодических задач. Это предполагает, что опытные данные разделены на эпизоды, и что все эпизоды в итоге заканчиваются, вне зависимости от того, какие действия выбраны. Только после завершения эпизода происходит оценивание значений и стратегии изменяются.

МК-метод всех посещений вычисляет $V_\pi(s)$ как среднее значение ценностей, которые были получены в результате посещения всех состояний s в некотором наборе эпизодов. МК-метод первого посещения усредняет только те значения, которые были получены в результате первых посещений состояния s.

Как метод первых посещений, так и метод всех посещений сходятся к $V_\pi(s)$ при стремлении числа посещений (или первых посещений) s стремящихся к бесконечности.

Важный факт о МК-методах состоит в том, что оценки для каждого состояния являются независимыми, в отличии от методов динамического программирования. В особенности, следует заметить, что вычисление оценки отдельного состояния не зависит от числа состояний. Это делает МК-методы особенно привлекательными в тех случаях, когда требуется оценить только некоторое подмножество состояний.

Если модель недоступна, то тогда полезно получить оценки действий, а не состояний. При наличии модели вполне достаточно иметь оценки состояний, чтобы определить стратегию. Для этого необходимо просмотреть на один шаг вперед и выбрать, какое действие приводит к наилучшей комбинации вознаграждения и следующего состояния. При отсутствии модели, однако, одних оценок состояния недостаточно. Необходимо в явном виде оценить значение каждого действия для того, чтобы в дальнейшем полученные оценки использовать для построения стратегии.

Задача оценки стратегии состоит в вычислении функции $Q_\pi(s, a)$ – ожидаемой награды при начале в состоянии s, выполнении действия a, и следуя стратегии π. Данная задача является основной задачей обучения с подкреплением.

Общая идея формирования управления состоит в действиях по схеме, которая опирается на понятие обобщенной итерации по стратегиям (ОИС). В концепции ОИС используется как приближенная стратегия, так и приближенная функция ценности. Коррекция функции ценности происходит таким образом, чтобы наилучшим образом отвечать текущей стратегии, а стратегия постоянно уточняется в зависимости от текущей функции ценности.

МК-версия классической итерации по стратегиям состоит в попеременном выполнении полных шагов оценки стратегии и улучшении стратегии, начиная с произвольной стратегии π_0, и заканчивая оптимальной стратегией и оптимальной функцией ценности.

Улучшение стратегии происходит за счет создания жадной стратегии по отношению к функции ценности. Так как имеется функция ценности действия, то не требуется модели для того, чтобы построить жадную стратегию. Для любой функции оценки действия Q, соответствующая ей жадная стратегия – это такая, что для каждого состояния $s \in S$, можно определенным образом выбрать действие с максимальной ценностью действия:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a).$$

Улучшение стратегии может быть осуществлено за счет построения каждой стратегии π_{k+1} как жадной стратегии по отношению к Q_{π_k}.

Как правило в методах Монте-Карло чередуются операции оценивания и улучшения от эпизода к эпизоду. После завершения эпизода происходит оценивание стратегии, используя награду, которая была получена в эпизоде, после чего стратегия улучшается. Алгоритм, соответствующий данной схеме получил название ИС-алгоритм Монте-Карло с изучающими стартами.

Обобщая вышесказанное, следует сказать, что методы Монте-Карло имеют 3 вида преимуществ по сравнению с методами динамического программирования. Во-первых, оптимальное поведение можно построить в результате непосредственного взаимодействия с окружающей средой, при этом не требуется знания модели динамики окружающей среды. Во-вторых, данные методы можно использовать в сочетании с методами моделирования, позволяющими получить смоделированные совокупности эпизодов. В-третьих, методы Монте-Карло могут легко и эффективно фокусироваться на небольшом подмножестве состояний.

3.4.3 Методы временных различий, Q−learning

Методы временных различий (TD-методы − Temporal Difference) совмещают в себе идеи методов Монте-Карло и динамического программирования. Как и методы Монте-Карло, TD-методы могут обучаться без модели динамики окружения, а непосредственно из опыта. Как и методы динамического программирования, TD-методы обновляют расчетные оценки, частично основываясь на других оценках, при этом не ожидая финального результата [2].

И TD-методы, и методы Монте-Карло используют опыт, чтобы решить задачу предсказания. Из некоторого данного опыта следования стратегии π, оба метода обновляют их оценки V функции V_π для состояний S_t, которые не являются финальными и присутствуют в данном опыте. В случае посещения состояния S_t в момент времени t оба метода корректируют свои оценки основываясь на том, что случилось после посещения данного состояния. Метод всех посещений Монте-Карло, который подходит для всех нестационарных состояний:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)],$$

где G_t − фактическая выгода, полученная после момента времени t, α − значение коэффициента обучения. Данный метод называется МК-метод с постоянной α. Однако, в то время как методы Монте-Карло должны ждать окончания эпизода для определения приращения к $V(s_t)$, в случае с TD-методами необходимо дождаться следующего временного шага.

Простейший TD-метод, известный как TD(0), имеет следующий вид:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1} - V(S_t)].$$

Так как при коррекции TD-метод частично основывается на существующих оценках, то будем называть этот метод самонастраивающимся, как и метод ДП. TD-метод сочетает в себе выборки метода МК с самонастройкой методов ДП.

Одним из преимуществ TD-методов над методами ДП является то, что первые не требуют знания модели окружающей среды. Преимуществу TD-методов над методами МК является то, что первые являются интерактивными, полностью инкрементными методами. Если в МК-методах, необходимо дождаться завершения эпизода, чтобы узнать награду, то в TD-методах необходимо дождаться лишь следующего временного шага. Кроме того, МК-методы должны игнорировать эпизоды или снижать значимость эпизодов, в которых предпринимаются экспериментальные действия, что может сильно замедлить обучение. TD-методы не являются чувствительными к проблемам такого рода,

так как обучаются на основе каждого перехода, вне зависимости от осуществляемых в дальнейшем действий. На практике было показано, что TD-методы сходятся быстрее методов Монте-Карло с постоянной α [1].

TD-метод управления с интегрированной оценкой ценности стратегий состоит в оценке функции $Q_\pi(s, a)$ для текущей стратегии π и для всех состояний s и действий a:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)].$$

Данная коррекция происходит после каждого перехода из не являющегося конечным состояния S_t. Если состояние S_{t+1} является конечным, то значение $Q(S_{t+1}, A_{t+1})$ полагается равным нулю. Это правило использует каждый элемент из пятерки событий $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, из которых состоит переход от одной пары состояние-действие к другой. Данный алгоритм получил название SARSA. Метод оценки ценности стратегии управления состоит в оценке функции Q_π для стратегии π, в то время как стратегия π делается более жадной по отношению к Q_π.

Другой метод управления с разделенной оценкой ценности стратегий известен как Q-обучение. Один из его подвидов, одношаговое Q-обучение, определяется следующим образом:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)].$$

В этом случае искомая функция ценности действия Q, непосредственно приближает Q^*, оптимальную функцию ценности действия, независимо от применяющейся стратегии. Стратегия по-прежнему определяет какие пары состояние-действие корректируются и посещаются, однако для обеспечения сходимости необходимо лишь, чтобы все пары продолжали корректироваться.

4 Базовый алгоритм обучения с подкреплением

4.1 Q-сеть для задач обучения с подкреплением

Хорошие результаты на играх Atari показал алгоритм глубокого Q-обучения [3]. Разработанная система использовала сверточную нейронную сеть над набором последних кадров для аппроксимации функции награды и для проведения шагов обновления алгоритма Q-обучения.

Дополнительным нововведением в этом подходе является техника повтора опыта, согласно которой совершенные агентом переходы, задающиеся четверкой:

$$E_t = (S_t, A_t, R_{t+1}, S_{t+1}),$$

сохранялись в базу данных. Для обучения нейронной сети из полученной базы опытов выделяется подвыборка, которая затем используется в шаге обновления Q-обучения. Алгоритм сохраняет фиксированное число последних переходов. Подобный подход к обучению увеличивает скорость сходимости алгоритма и стабильность обучения.

4.2 Двойная Q-сеть

Одним из недостатков стандартного алгоритма Q-обучения является задокументированая и теоретически обоснованная тенденция к значительному завышению оценок функции награды [4]. Причиной подобного поведения является тот факт, что Q-обучение, при наличии нескольких альтернатив действий, выбирает действие с наибольшей оценкой его функции награды. Это приводит, при наличие неизбежного из-за стохастической природы алгоритма шума, к тому, что алгоритм будет стремиться выбирать действия с наибольшим размером завышения оценки. Подобное систематическое завышение приводит к изменению оптимальной стратегии поведения агента, к которой в асимптотике сходится алгоритм.

Для решения данной проблемы было предложено использовать две нейронные сети [5]. Одна из них обучается обычным алгоритмом Q-обучения на выбранном обучающем наборе, вторая же используется для оценки функции награды от выбранного действия, и обновляется значением первой нейронной сети каждые несколько десятков шагов. Такое решение значительно уменьшило размер переоценки функции награды и привело к улучшению достигнутого результата на тестовом наборе окружений.

4.3 Дуэлирующая двойная Q-сеть

Основная идея, лежащая в основе дуэлирующей двойной Q-сети [6], заключается в том, что для большинства состояний в большинстве исследуемых окружений функцию награды действия в состоянии $Q(s, a)$ можно разделить на функцию выгоды действия и функцию награды состояния:

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

$$E_{a \in \pi} A(s, a) = 0.$$

Каждая из этих функций обладает своим интерпретируемым смыслом и изменяется по-разному в зависимости от обрабатываемых опытов. В связи с этим разделение вычисления функции награды на две вышеописанные части приводит к более точному изменению оцениваемой функции награды при использовании тех же обучающих примеров.

Другим важным усовершенствованием, использованным в архитектуре с дуэлирующей двойной Q-сетью, является приоритезированный повтор опыта [7], который вместо полностью случайного выбора обучающей выборки из накопленного банка опыта пытается выбрать те моменты, которые наиболее важны для обучения агента. Центральным понятием такого подхода является критерий, оценивающий важность для обучения каждого из совершенных агентом переходов. Одна из самых простых эвристик, позволяющая оценить подобную характеристику − это ошибка в оценке функции награды итогового состояния до и после совершения действия. Вероятность выбора каждого перехода в набор обучения равна:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha},$$

где $p_i > 0$ - оценка важности перехода i. Параметр α определяет, насколько сильно используется приоритизация, и при α равном нулю система выбирает различные переходы с равной вероятностью. p_i возможно вычислять пропорциональной приоритизацией, где $p_i = |\delta_i| + \epsilon$ или приоритизацией, основанной на ранге, где $p_i = \frac{1}{rank(i)}$ - обратное число к индексу опыта под номером i после сортировки этих опытов в соответствии с ошибкой δ_i.

Введение приоритезированного использования опыта изменяет математическое ожидание оцениваемых величин функции награды из-за изменения вероятностей использования различных данных. Для исправления этой ошибки оценивания необходимо внести дополнительные штрафующие веса на обновляющее правило в Q-обучении:

$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)}\right)^\beta,$$

где параметр β показывает насколько сильно необходима поправка на веса в данный момент обучения, а параметр N показывает количество объектов в обучающей выборке. Так как точность схождения важна в большей степени в финальных этапах обучения, уменьшение параметра β на первых этапах обучения позволяет алгоритму сильнее обучаться на начальном опыте.

5 Стратегии исследования окружений

Базовые стратегии исследования окружений включают в себя ϵ-жадную стратегию и стратегию исследования по Больцману [8].

5.1 $\mathcal{E}$-жадная стратегия исследования

$\mathcal{E}$-жадная стратегия исследования совершает на каждом шаге алгоритма случайное действие с вероятностью ϵ, и совершает оптимальное действие согласно с текущим значением функции награды с вероятностью $1 - \epsilon$. Типичная стратегия обучения состоит в инициализации значения параметра ϵ большим значением и постепенным понижением этого параметра до маленького значения по мере обучения для уменьшения доли случайного поведения. Формула, описывающая эту стратегию, выражается таким образом:

$$\pi_\epsilon(s) = \begin{cases} \text{random}, & \text{с вероятностью } p = \epsilon \\ \text{argmax}_a Q(s,a), & \text{с вероятностью } p = 1 - \epsilon \end{cases}.$$

5.2 Стратегия исследования по Больцману

Стратегия исследования по Больцману выбирает случайное действие a^* с весом, пропорциональным текущему значению функции награды для этого действия:

$$P_b(\pi_\epsilon(s) = a^*) = \frac{e^{\frac{Q(s,a^*)}{T}}}{\sum_a e^{\frac{Q(s,a)}{T}}},$$

где T – это параметр температуры, который задает силу сглаживания распределения вероятностей между действиями. Такая стратегия позволяет приоритизировать действия, которые потенциально приведут к большей награде в будущем. При стремлении параметра температуры к бесконечности алгоритм начинает выбирать случайные действия, при движении параметра температуры к нулю алгоритм начинает выбирать оптимальное действие согласно текущей функции награды:

$$\begin{cases} P_b(\pi_\epsilon(s) = a^*) = \frac{1}{|A|}, & \text{при } T \to \infty \\ \pi_\epsilon(s) = \text{argmax}_a Q(s,a), & \text{при } T \to 0 \end{cases}.$$

5.3 Приоритезированное исследование на основе модели, предсказывающей динамику окружения

Одним из подходов исследования окружений является изменение полученной агентом награды для учета пользы, которая получена из-за проведенного исследования

окружения. Один из таких подходов использует кодированное представление состояния s, которые обозначается $\sigma(s)$ [9]. На основе этого кодированного представления можно создать предсказывающую модель M, которая по кодированному представлению старого состояния $\sigma(s_t)$ и совершенному действию a_t выдает кодированное представления нового состояния $\sigma(s_{t+1})$:

$$\mathrm{M} \ : \ \sigma(S) \times A \to \sigma(S),$$

$$e(s_t, a_t) = ||\sigma(s_{t+1}) - M(\sigma(s_t), a_t)||,$$

$$\tilde{e_t} = \frac{e_t}{\max_{i<t} e_i},$$

где $\tilde{e_t}$ задает нормализованную ошибку предиктивной модели на шаге t. Эту ошибку можно использовать для изменения награды, полученной после совершения действия, подавая агенту дополнительный стимул совершать те действия, результат которых модель M предсказывает плохо:

$$R_(S_t, A_t) \leftarrow R(S_t, A_t) + R_{bonus\ pred}(S_t, A_t),$$

$$R_{bonus\ pred}(S_t, A_t) = \beta * \left(\frac{\tilde{e_t}}{t^C}\right)$$

где параметр β задает размер дополнительной награды, а параметр C задает скорость угасания дополнительной награды со временем. В качестве кодирования состояний применяются автокодировщики − нейронные сети, уменьшающие размерность входа и оптимизирующие ошибку при возврате исходного значение по сжатому представлению.

В данной работе предложена модификация данного алгоритма, заключающаяся в том, что автокодировщик делит свою первую половину слоев нейронной сети с Q-сетью. Подобный подход позволит использовать первые слои Q-сети для извлечения закодированного представления состояния. Такая архитектура предоставляет несколько преимуществ: увеличенная скорость работы системы, из-за отсутствия необходимости извлекать закодированное представление состояния дважды за один проход алгоритма, а так же учет закодированного представления состояний в расчете значений Q-функции. Модель предсказывания реализована как нейронная сеть от конкатенированных значений центрального слоя автокодировщика и совершенного действия в one-hot кодировке.

5.4 Приоритезированное исследование на основе количества посещений состояния

Другим способом поощрения исследования является способ, добавляющий награду на основе того, насколько часто агент бывал в текущем состоянии [10]. Для того, чтобы поддерживать счет количества посещений в непрерывном пространстве состояний, проделываются следующие действия:

- Создается автокодировщик над состояниями окружения.
- К центральному слою автокодировщика добавляется функция активации сигмоида: $f_s(x) = \frac{1}{1+e^{-x}}$.
- К оптимизируемой автокодировщиком функции ошибки добавляется регуляризационный член: $\text{Loss}(h) = \text{Mean}(0.25 - (0.5 - h)^2)$, где h - это вектор значений центрального слоя автокодировщика.
- После центрального слоя автокодировщика добавляется слой гауссового шума с фиксированной дисперсией.

Подобные изменения приводят к тому, что центральный слой автокодировщика выдает значения, близкие либо к 0, либо к 1, и при этом функция ошибки нейронной сети не теряет свойство дифференцируемости. Это позволяет получить для каждого состояния его дискретное представление $\phi(s)$ через округление вектора значений центрального слоя автокодировщика до ближайшего целого. На каждом этапе обучения алгоритм поддерживает количество посещений каждого значения закодированного представления состояний $n(\phi(s))$. На основе этой статистики вычисляется дополнительная награда:

$$R_(S_t, A_t) \leftarrow R(S_t, A_t) + R_{bonus\ count}(S_t, A_t),$$

$$R_{bonus\ count}(S_t, A_t) = \beta * \Big(\frac{1}{(n(\phi(s)))^C}\Big),$$

где параметр β задает размер дополнительной награды, а параметр C задает скорость угасания дополнительной награды со временем.

В данной работе предложена модификация данного алгоритма, заключающаяся в том, что автокодировщик делит свою первую половину слоев нейронной сети с Q-сетью. Мотивация данного изменения аналогична мотивации, предложенной в пункте 5.3.

5.5 Байесовское Q-обучение

Один из возможных подходов к Q-обучению заключается в выражении ожидаемой награды $Q(s, a)$ в состоянии s и действии a через нормальное распределение с средним значением распределения $\nu_{s,a}$ и точностью (обратное значения к дисперсии) распределения $\tau_{s,a}$, [11]. Эти гиперпараметры выражаются через априорное гамма-нормальное распределение − сопряженное распределение к нормальному распределению:

$$p(\mu, \tau) \sim NG(\mu_0, \lambda, \alpha, \beta),$$

$$p(\mu, \tau) \propto \tau^{\frac{1}{2}} e^{-\frac{1}{2}\lambda\tau(\mu-\nu_0)^2} \tau^{\alpha-1} e^{\beta\tau}.$$

Из таких предположений будет следовать, что апостериорное распределение гиперпараметров нормального распределения после учета данных о награде будет так же принадлежать гамма-нормальному распределению. Байесовское Q-обучение заключается в приближенном подсчете гиперпараметров этого распределения в каждой паре состояния-действия вместо приближенного расчета значения функции награды. При получении нового элемента выборки M правила обновления гиперпараметров задаются следующим образом:

$$\mu_0 \leftarrow \mu_0 + \frac{1}{\lambda + 1} * (M - \mu_0),$$

$$\beta \leftarrow \beta + \frac{\lambda}{2(\lambda + 1)} * (M - \mu_0)^2,$$

$$\lambda \leftarrow \lambda + 1,$$

$$\alpha \leftarrow \alpha + 0.5.$$

Математическое ожидание и дисперсия $Q(s, a)$ могут быть вычислены на основе гиперпараметров следующим образом:

$$\mathrm{E}[Q(s, a)] = \mu_0$$

$$\mathrm{Var}[Q(s, a)] = \frac{b}{\lambda(\alpha - 1)}$$

Исследование окружений, которое подразумевает под собой выбор действия на основе текущего состояния, при таком подходе можно проводить несколькими способами.

- Жадный выбор

При жадном выборе алгоритм выбирает такое действие, которое максимизирует значения математического ожидания распределения. Такой агент не будет исследовать окружения и будет пытаться выбрать оптимальное действие на каждом шаге.

- $\mathcal{E}$-жадное исследование на основе вычисленной дисперсии

При $\mathcal{E}$-жадном исследовании алгоритм с вероятностью ϵ принимает действие, максимизирующее функцию награды, а с вероятностью $1-\epsilon$ принимает то действие, расчитанная дисперсия которого $\mathrm{Var}[Q(s, a)]$ максимальна.

- Сэмплинг Q-функции

При использовании сэмплинга Q-функции действие a в состоянии s выбирается с вероятностью, равной вероятности того, что случайная величина функции награды $Q(s, a)$ будет максимальной среди всех доступных действий:

$$P(\pi(s) = a) = P(a = \underset{a^*}{\operatorname{argmax}}\, \mu_{s,a^*}) = P(\forall\ a^* \neq a, \mu_{s,a} > \mu_{s,a^*}) =$$

$$= \int_{-\infty}^{+\infty} P(\mu_{s,a} = q_a) \prod_{a^* \neq a} P(\mu_{s,a*} < q_a) dq_a \tag{3}$$

На практике точное вычисление этих вероятностей по вышеприведенным формулам можно заменить семплированием каждой случайной величины с распределением для каждого действия и выбором того действия, значение случайной величины для которого оказалось максимальным. Такая процедура выдает действия с такой же вероятностью, которую выдал бы прямолинейный подсчет вероятности по формуле (3).

В данной работе предлагается модификация метода, позволяющая избежать вычислений гиперпараметров гамма-нормального распределения λ и α с помощью нейронной сети. Это необходимо в связи с тем, что нейронная сеть не гарантирует того, что в процессе обучения эти параметры будут для всех состояний окружений иметь адекватные значения, а также из-за того, что постоянное увеличение параметров λ и α приведет к преждевременной сходимости процесса обучения. λ и α считаются гиперпараметрами алгоритма исследования и фиксируются на все время обучения алгоритма. В таких условиях нейронная сеть должна поддерживать только гиперпараметры μ_0 и β, на основании которых необходимо было бы вычислять математическое ожидание и дисперсию ожидаемой награды. В данной работе предлагается вычислять математическое ожидание и дисперсию ожидаемой награды напрямую, используя следующие правила обновления:

$$\mathrm{E}(Q(s,a)) \leftarrow \mathrm{E}(Q(s,a)) + \frac{1}{\lambda + 1} * (M - \mathrm{E}(Q(s,a)))$$

$$\mathrm{Var}(Q(s,a)) \leftarrow \frac{\lambda * (\alpha - 1)}{(\lambda + 1) * (\alpha - 0.5)} * \left(\mathrm{Var}(Q(s,a)) + \frac{(M - \mathrm{E}(Q(s,a)))^2}{2(\lambda + 1)(\alpha - 1)}\right)$$

Для выбора действия используется метод $\mathcal{E}$-жадного исследования на основе вычисленной дисперсии. Во всех рассматриваемых окружениях в качестве параметров λ и α взяты значения 4 и 2 соответственно.

5.6 Комбинированный метод на основе разделения оценки награды за эксплуатацию и исследование

Большинство рассмотренных методов исследования окружений использует изменение получаемых агентом наград как способ поощрить за посещение ценных с точки зрения исследования состояний. Побочным эффектом такого подхода является тот факт,

что вычисляемая алгоритмом Q-learning функция ожидаемой награды в процессе обучения перестает представлять приближение оптимальной функции награды. Такое изменение процесса обучения может негативно сказаться на скорости процесса обучения. Для преодоления этой проблемы предлагается реализовать вычисление $Q(s, a)$ как функции ожидаемой награды и функции $Exp(s, a)$ как функции ожидаемой награды за исследование при совершении в состоянии s действия a. В такой формулировке стратегия исследования окружения будет задаваться способом вычисления $Exp(s, a)$ и функцией C, которая на основе параметра $\delta \in (0, 1)$ и значений функций $Q(s, a)$ и $Exp(s, a)$ во всех доступных действиях для состояния s вычислит действие, которое необходимо предпринять в данном состоянии.

Все рассмотренные в данной работе способы исследования окружений представляются в такой форме:

- $\mathcal{E}$-жадная стратегия исследования.

$$Exp(s, a) = 1,$$

$$C(Q, Exp, \delta) \sim \begin{cases} \text{random}, & \text{с вероятностью } p = \delta \\ \text{argmax}_a Q_a, & \text{с вероятностью } p = 1 - \delta \end{cases}.$$

- Стратегия исследования по Больцману.

$$Exp(s, a) = 1,$$

$$C(Q, Exp, \delta) \sim P(C(Q, Exp, \delta) = a) = \frac{e^{\frac{Q_a}{\ln(\frac{1}{\delta})}}}{\sum_a e^{\frac{Q_a}{\ln(\frac{1}{\delta})}}},$$

где Q_a показывает оцениваемое значение функции награды $Q(s, a)$ при совершении действия a.

- Приоритезированное исследование на основе модели, предсказывающей динамику окружения.

$Exp(s, a)$ вычисляется с помощью правила обновления Беллмана над дополнительной наградой метода $R_{bonus\ pred}(s, a)$, после чего проводится либо полностью жадная стратегия поведения агента, либо одна из базовых стратегий над модифицированной функцией награды $Q^*(s, a)$, которая вычисляется на основе функций $Q(s, a)$ и $Exp(s, a)$:

$$Q^*(s, a) = Q(s, a) + Exp(s, a).$$

- Приоритезированное исследование на основе количества посещений состояния.

$Exp(s, a)$ вычисляется с помощью правила обновления Беллмана над дополнительной наградой метода $R_{bonus\ count}(s, a)$, после чего проводится либо полностью жадная стратегия поведений агента, либо одна из базовых стратегий над модифицированной функцией награды $Q^*(s, a)$, которая вычисляется на основе функций $Q(s, a)$ и $Exp(s, a)$:

$$Q^*(s, a) = Q(s, a) + Exp(s, a).$$

- Байесовское Q-обучение.

$Exp(s, a)$ вычисляется как значение дисперсии случайной величины награды $\mathrm{Var}[Q(s, a)]$, после чего используется одна из стратегий исследования, описанных в пункте 5.5.

В системе обучения с подкреплением реализован метод, комбинирующий Байесовское Q-обучение и приоритизацию исследования на основе подсчета количества посещений. $Exp(s, a)$ вычисляется с помощью правила обновления Беллмана над дополнительной наградой по следующей формуле: $Exp(s, a) = \mathrm{Var}[Q(s, a)] + R_{bonus\ count}(s, a)$. Стратегия исследования с вероятностью ϵ принимает действие, максимизирующее функцию награды $Q(s, a)$, а с вероятностью $1-\epsilon$ принимает действие, максимизирующее функцию награды за исследование $Exp(s, a)$.

6 Эксперимантальное исследование

6.1 Программная структура системы обучения с подкреплением

В рамках проделанной работы была реализована система обучения с подкреплением[1] над окружениями из фреймворка OpenAI Gym [12] на языке Python 2.7 с использованием библиотек Theano [13] и Lasagne [14]. Данная система реализует метод дуэлирующей двойной Q-сети с стандартным повтором опыта и позволяет интегрировать любое окружение из фреймворка OpenAI Gym.

6.2 Рассматриваемые окружения

- Cart Pole

В окружении CartPole (Рис. 1) цель игры состоит в поддержании вертикального положения шеста на тележке. Пространство состояний состоит из 4 вещественных переменных, описывающих положение тележки и столба. Окружение предоставляет два действия, позволяющие толкнуть тележку направо или налево. Каждый кадр игры дает положительную награду. Эпизод заканчивается либо когда шест на тележке отклоняется от вертикального положения больше, чем на 15 градусов, в случае чего агент получает большую отрицательную награду, либо после прошествия 500 кадров.

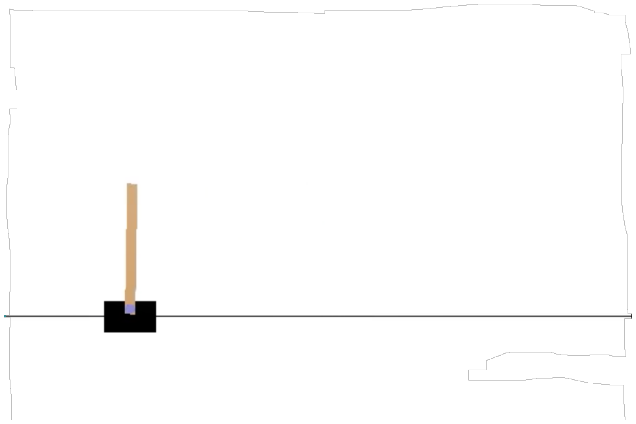

Рис. 1: Кадр из игры CartPole.

- Lunar Landing

В окружении LunarLanding (Рис. 2) цель игры состоит в посадке лунного модуля на выделенную площадку с минимальными повреждениями и затратами топлива. Пространство состояний состоит из 8 вещественных переменных, задающих физическую

[1] `https://github.com/Goorman/Reinforcement_Learning_Dissertation`

конфигурацию текущего состояния системы. Доступны четыре действия, позволяющие включать двигатели в одном из трех направлений, либо не делать ничего. Работа двигателей тратит топливо, что дает отрицательную награду. Каждый кадр игры дает награду, обратно пропорциональную расстоянию до посадочной площадки по горизонтали. Если при посадке на поверхность модуль не имел достаточно маленькой скорости, то агент получает большую отрицательную награду. Игра заканчивается либо при аварийной посадке модуля (приземление с достаточно большой скоростью), либо при удачной посадке модуля (приземление с достаточно маленькой скоростью), либо после прошествия 200 кадров.

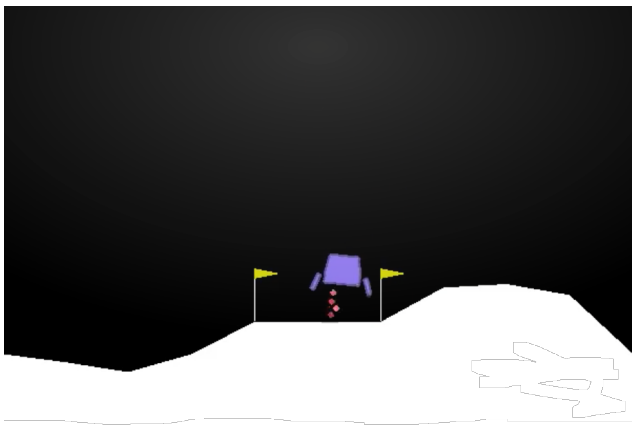

Рис. 2: Кадр из игры LunarLanding.

6.3 Методика проведения исследований

- Построение кривой обучения алгоритмов.

Для исследования алгоритмов каждый из реализованных алгоритмов применяется на каждом из выбранного набора окружений, обучение прекращается после достижения I_{maxep} эпизодов. Каждому эпизоду обучения под номером $i \in [1, I_{maxep}]$ сопоставляется полученная в этом эпизоде награда $R(i)$. Функция $R(i)$, отображенная на графике, задает кривую обучения алгоритма на заданном окружении. Эта кривая будет использоваться для анализа построенных алгоритмов.

- Усреднение результатов по нескольким экспериментам и сглаживание кривых обучения.

Каждая конкретная кривая обучения подвержена влиянию случайных эффектов. Эти случайные эффекты включают в себя выбор начальных весов нейронной сети, случайные действия алгоритмов исследования окружений, выбор уровня в каждом конкретном эпизоде окружения. Наличие этих случайных факторов означает, что одна

кривая обучения алгоритма на окружении не даст полезной информации о свойствах алгоритма. В этой работе каждый алгоритм исследования запускается $N_{exp} = 10$ раз, порождая функцию $R_k(i)$ для запуска номер $k \in [1, N_{exp}]$. После этого от всех полученных кривых обучения берется среднее:

$$R_{mean}(i) = \frac{\sum_{k=1}^{N_{exp}} R_k(i)}{N_{exp}}.$$

Кривая обучения сглаживается методом скользящего среднего с размером окна windowsize:

$$R_{alg}(i) = \frac{\sum_{k=1}^{\text{windowsize}} R_{mean}(i+1-k)}{\text{windowsize}},\ i \in [\text{windowsize}, I_{maxep}].$$

Полученная функция $R_{alg}(i)$ используется для анализа алгоритмов исследования.

- Функционалы качества.

В данной работе для каждого алгоритма на каждом окружении вычисляются два функционала качества: MaxR(1000), где

$$\text{MaxR}(K) = \max_{i \le K} R_{alg}(i),$$

которая показывает максимально достигнутую алгоритмом награду в окружении до заданного количества эпизодов в обучении, и MeanR(1000), где

$$\text{MeanR}(K) = \text{mean}_{i \le K} R_{alg}(i),$$

которая является средним значением награды по заданному количеству эпизодов в обучении. Этот функционал показывает нормированную площадь под кривой обучения и является аналогом скорости обучения алгоритма.

6.4 Результаты исследований.

Обозначим реализованные методы следующим образом:

- greedy – $\mathcal{E}$-жадная стратегия исследования.
- boltzman – Стратегия исследования по Больцману.
- count – Приоритезированное исследование на основе модели, предсказывающей динамику окружения.
- inc – Приоритезированное исследование на основе количества посещений состояния.

- bayes – Байесовское Q-обучение.
- comb – Комбинированный метод на основе разделения оценки награды за эксплуатацию и исследование.

Результаты

По результатам экспериментов над окружением CartPole, показанных на Рис. 3, Рис. 4, Таблице 1, можно увидеть, что максимальной награды за 1000 эпизодов добился комбинированный алгоритм, а наилучшей скорости обучения, представляемой средней наградой за 1000 эпизодов, добился алгоритм Байеса. Все алгоритмы показали в целом хорошие результаты в решении поставленной задачи обучения с подкреплением.

По результатам экспериментов над окружением LunarLanding, показанных на Рис. 5, Рис. 6, Таблице 2, можно увидеть, что максимальной награды за 1000 эпизодов добился алгоритм, основанный на приоритизации исследования на основе количества посещений состояний, а наилучшей скорости обучения добился алгоритм, основанный на приоритизации исследования с помощью предсказывающей модели. Остальные алгоритмы показали плохие результаты на этом окружении. Причина этой неудачи скорее всего заключается в структуре наград в окружении LunarLanding. В этом окружении присутствует награда, поощряющая близость лунного модуля к центру посадочной площадки по горизонтали, выдающаяся агенту на каждом шаге. Это может создавать локальный оптимум в пространстве стратегий действия агента, считающий оптимальным действия, которые удерживают модуль как можно ближе к посадочной вертикали не стремясь посадить лунный модуль без аварии. Алгоритмы «count» и «inc» основаны на изменении исходных наград, поступающих от окружения агенту, и эта особенность алгоритма позволяет избежать локального оптимума в пространстве стратегий.

На основе проведенных экспериментов можно сделать вывод, что невозможно выделить алгоритм исследования окружений, проявляющий себя лучше всех остальных на всех окружениях. В связи с этим возникает потребность исследования более широкого спектра окружений с целью выделения подмножеств окружений, для каждого из которых будет определен алгоритм исследования, проявляющий себя наилучшим образом. В данной работе такое исследование было невозможно произвести из-за нехватки вычислительных ресурсов.

Стоит отметить, что новые методы, которые были предложены в данной работе, показывают себя как наилучшие на одном из окружений, из чего следует сделать вывод, что существует ряд окружений на которых эти методы являются наилучшими. В будущем следует выделить характеристики таких окружений.

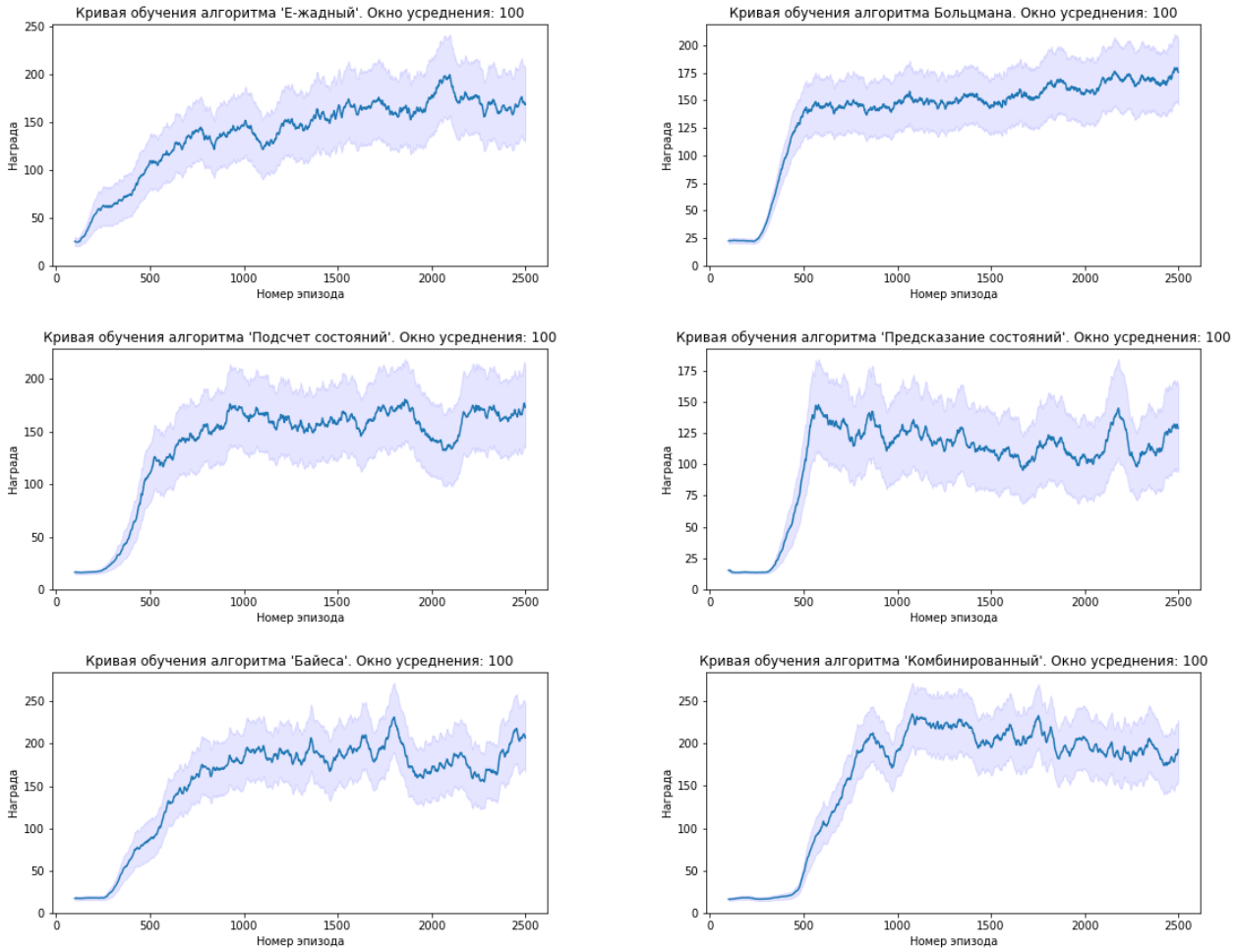

Рис. 3: Усредненные и сглаженные кривые обучения алгоритмов на окружении CartPole. Размытием показано стандартное отклонение награды по запускам с коэффициентом 0.2.

Алгоритм исследования	MaxR(1000)	MeanR(1000)
greedy	151.81	112.00
boltzman	158.05	120.36
count	176.10	114.97
inc	147.56	96.40
bayes	196.45	**121.15**
comb	**234.59**	113.66

Таблица 1: Результаты экспериментальных исследований на окружении CartPole

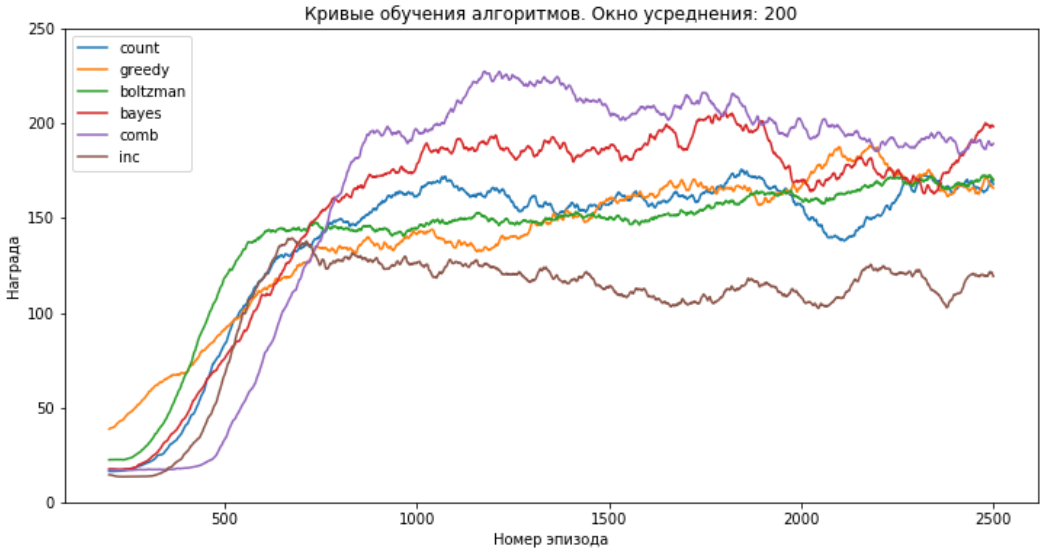

Рис. 4: Усредненные и сглаженные кривые обучения алгоритмов на окружении CartPole.

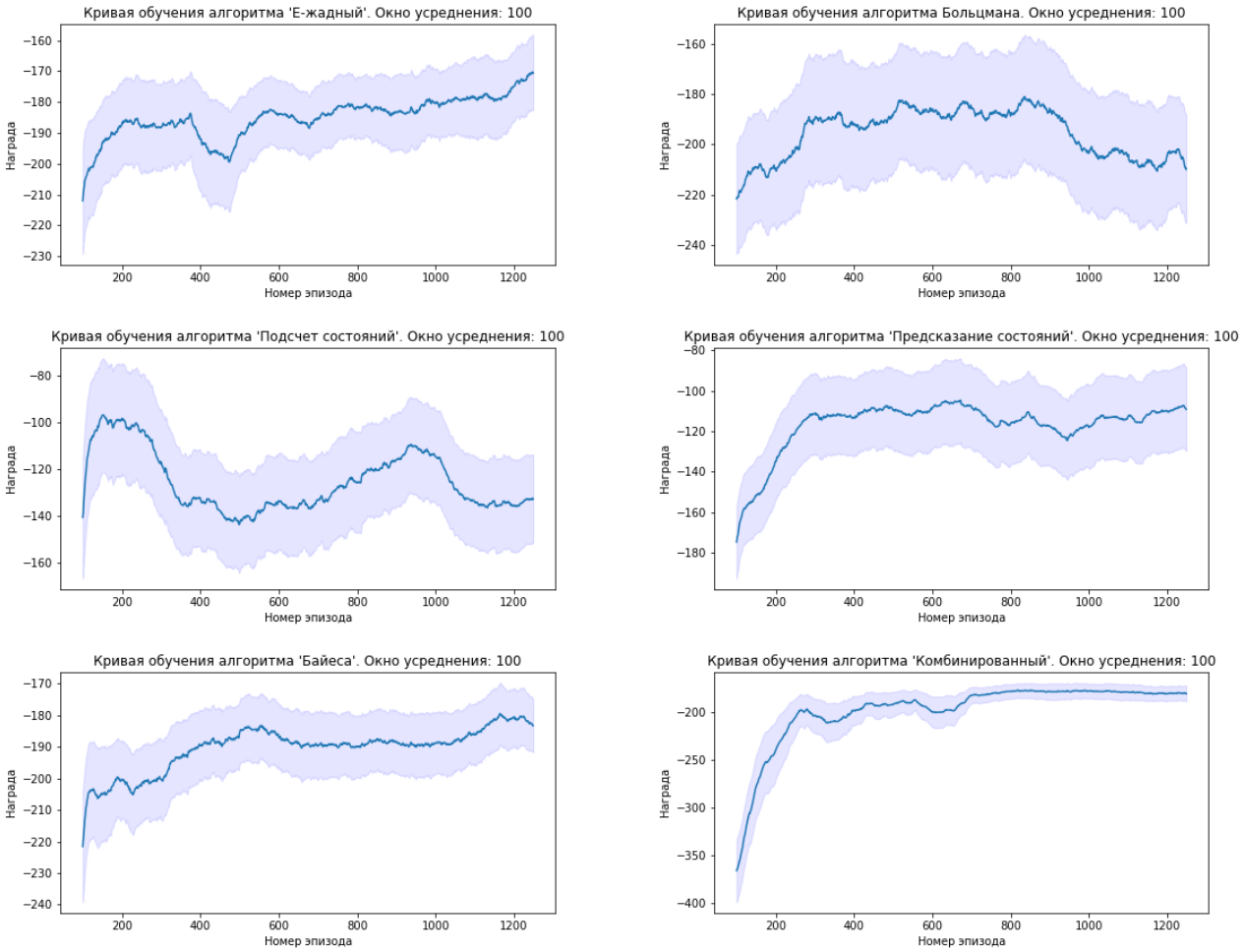

Рис. 5: Усредненные и сглаженные кривые обучения алгоритмов на окружении LunarLanding. Размытием показано стандартное отклонение награды по запускам с коэффициентом 0.2.

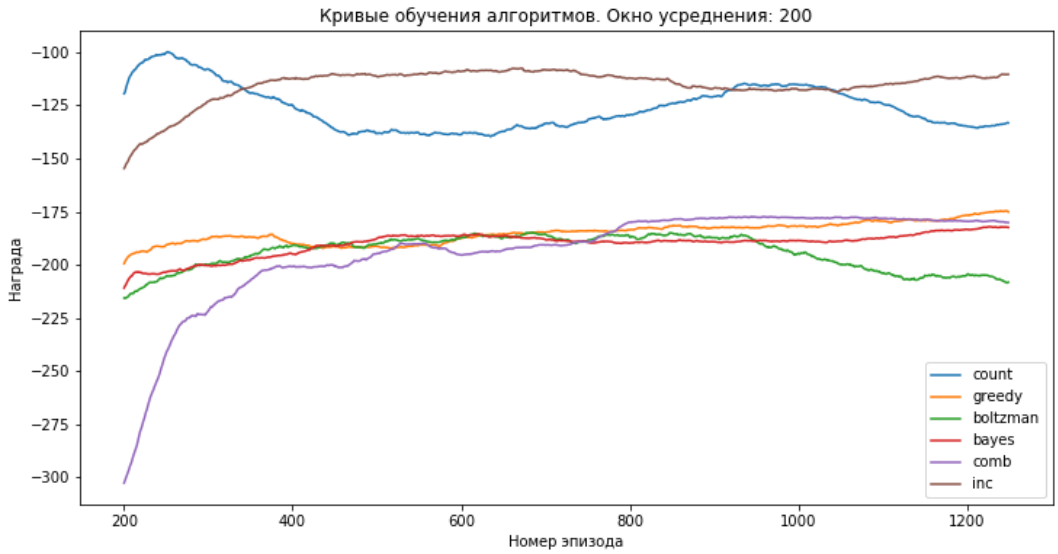

Рис. 6: Усредненные и сглаженные кривые обучения алгоритмов на окружении LunarLanding.

Алгоритм исследования	MaxR(1000)	MeanR(1000)
greedy	-178.06	-185.32
boltzman	-181.08	-191.77
count	**-98.18**	-126.07
inc	-104.72	**-113.66**
bayes	-183.17	-190.27
comb	-176.69	-189.99

Таблица 2: Результаты экспериментальных исследований на окружении LunarLanding.

7 Заключение

В ходе данной работы было проделано следующее:

- Реализована система обучения с подкреплением на основе фреймворка OpenAI Gym.
- Предложены модификации к существующим методам(«count», «inc.», «bayes») и новый метод исследования окружений («comb»)
- Реализованы 6 методов исследования окружений в составе реализованной системы обучения с подкреплением.
- Проведено экспериментальное исследование реализованных методов на двух окружениях OpenAI Gym.

На основе результатов проделанной работы были сделаны следующие выводы:

- Среди реализованных алгоритмов исследования окружений нельзя выделить алгоритм, наилучшим образом проявляющий себя на всех окружениях.
- Предложенный метод позволяет получить наилучшее качество по одной из метрик на одном из рассматриваемых окружений.
- По другой метрике реализованная модификация одного из существующих методов получила наилучшее качество.
- Следует провести анализ характеристик окружений, на которых предложенный алгоритм исследования и реализованные модификации к существующим методам показывают наилучшее качество по введенным метрикам.

Список литературы

[1] Sutton R. S., Barto A. G. Reinforcement learning: An introduction. – Cambridge : MIT press, 1998. – Т. 1. – №. 1.

[2] Wiering M., Van Otterlo M. Reinforcement learning //Adaptation, Learning, and Optimization. – 2012. – Т. 12.

[3] Mnih V. et al. Playing atari with deep reinforcement learning //arXiv preprint arXiv:1312.5602. – 2013.

[4] Hasselt H. V. Double Q-learning //Advances in Neural Information Processing Systems. – 2010. – С. 2613-2621.

[5] Van Hasselt H., Guez A., Silver D. Deep reinforcement learning with double Q-learning //CoRR, abs/1509.06461. – 2015.

[6] Wang Z., de Freitas N., Lanctot M. Dueling network architectures for deep reinforcement learning //arXiv preprint arXiv:1511.06581. – 2015.

[7] Schaul T. et al. Prioritized experience replay //arXiv preprint arXiv:1511.05952. – 2015.

[8] McFarlane R. A Survey of Exploration Strategies in Reinforcement Learning.

[9] Stadie B. C., Levine S., Abbeel P. Incentivizing exploration in reinforcement learning with deep predictive models //arXiv preprint arXiv:1507.00814. – 2015.

[10] Tang H. et al. # Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning //arXiv preprint arXiv:1611.04717. – 2016.

[11] Dearden R., Friedman N., Russell S. Bayesian Q-learning //AAAI/IAAI. – 1998. – С. 761-768.

[12] Brockman G. et al. OpenAI gym //arXiv preprint arXiv:1606.01540. – 2016.

[13] Team T. T. D. et al. Theano: A Python framework for fast computation of mathematical expressions //arXiv preprint arXiv:1605.02688. – 2016.

[14] Dieleman S. et al. Lasagne: First release //Zenodo: Geneva, Switzerland. – 2015.