

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

Кафедра дискретной математики и алгоритмики

**СРАВНЕНИЕ ПОДХОДОВ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ И
НЕЙРОННЫХ СЕТЕЙ ДЛЯ ПОСТРОЕНИЯ СТРАТЕГИЙ В ИГРЕ С
НЕПОЛНОЙ ИНФОРМАЦИЕЙ**

Курсовая работа

Кочева Ильи Александровича
обучающегося 4 курса
специальности «Информатика»

Научный руководитель:
Старший преподаватель
Буславский Александр Андреевич

Минск, 2025

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	1
ВВЕДЕНИЕ.....	2
ГЛАВА I ПОСТАНОВКА ЗАДАЧИ	3
Основные элементы теории игр.....	3
Методы решения проблем игр с неполной информацией и нулевой суммой ..	4
Метод Монте-Карло	5
Генетические алгоритмы.....	5
Алгоритмы обучения с подкреплением	5
Покер как игра с нулевой суммой и неполной информацией	6
Формализация задачи	6
Правила игры.....	7
ГЛАВА II НЕЙРОСЕТЕВЫЕ МЕТОДЫ И ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ В ИГРАХ С НЕПОЛНОЙ ИНФОРМАЦИЕЙ	9
Принципы работы искусственных нейронных сетей	9
Обучение с подкреплением (Reinforcement Learning).....	10
Подход на основе Q-обучения (DQN).....	12
Архитектура Actor-Critic в играх с неполной информацией	12
ГЛАВА III ПРОГРАММНАЯ РЕАЛИЗАЦИЯ.....	14
Архитектура и реализация модели игры	14
Формализация и реализация игрового агента	15
Механизм сохранения и восстановления состояния модели	16
Оценка силы руки	17
Описание генетического агента.....	18
Обучение нейросетевого агента	21
Анализ результатов нейросетевого подхода	22
Перспективы модификации и дальнейшего исследования.....	23
ЗАКЛЮЧЕНИЕ	24
Список использованных источников	25
Приложение А	26

ВВЕДЕНИЕ

Теория игр представляет собой раздел прикладной математики, посвящённый формализации и исследованию стратегических ситуаций, возникающих во взаимодействии рациональных агентов [1, с. 1]. Современная теория игр находит применение в самых различных областях человеческой деятельности, позволяя моделировать экономические процессы, социальные конфликты и взаимодействие человека с природой.

В средах, где взаимодействуют несколько агентов, результат действий одного игрока неизбежно зависит от решений, принятых другими. Что означает, что оптимальная стратегия участника должна учитывать вероятные действия оппонентов [2, с. 1]. Теория игр предоставляет нормативную базу для анализа таких ситуаций, предлагая концепции рационального поведения. Ключевым понятием здесь является равновесие Нэша – профиль стратегий, при котором ни один агент не имеет стимула отклоняться от выбранной тактики, если остальные участники придерживаются своих стратегий.

Однако для того, чтобы эта концепция работала на практике, необходимы эффективные алгоритмические методы поиска равновесия. Игры с полной информацией (например, шашки), где состояние игры полностью известно всем участникам, исторически поддавались решению легче, чем игры с неполной информацией (например, покер), где часть данных скрыта от одного или нескольких игроков. Основная причина заключается в том, что игры с полной информацией можно легко разбить на независимые подзадачи, решения которых объединяются в общую оптимальную стратегию. В играх с неполной информацией такой подход зачастую невозможен из-за неопределенности.

В связи с высокой вычислительной сложностью поиска равновесия в играх с неполной информацией, актуальным становится применение методов искусственного интеллекта. Основной задачей данной курсовой работы является сравнительный анализ двух таких подходов: генетических алгоритмов и нейронных сетей. В работе будет исследована их эффективность при поиске наилучших стратегий в условиях неопределенности на примере игры в покер.

ГЛАВА I ПОСТАНОВКА ЗАДАЧИ

Основные элементы теории игр

Перед тем как описывать модели игр, необходимо ввести некоторые базовые формальные определения, которые будут использоваться в ходе исследования.

Игра (объект исследования в теории игр) – упрощенная математическая модель конфликтной ситуации, отличающаяся от реального конфликта, тем, что ведется по определенным правилам, на основании которых игроки применяют совокупность целенаправленных действий, направленных на достижение собственного выигрыша (цели) в условиях конфликта.

Игрок – рациональный агент, принимающий решение в рамках игры для максимизации своей выгоды.

Состояние игры – это элемент множества возможных состояний, который включает в себя всю релевантную информацию, доступную игроку в данный момент времени.

Стратегия – полное описание плана действий игрока, определяющее выбор хода в каждой возможной ситуации, с которой может столкнуться игрок в игре.

Множество стратегий (S_i) – совокупность всех возможных стратегий, доступных i -му игроку.

Профиль стратегий – совокупность выбранных стратегий всеми игроками:

$$s = (s_1, s_2, \dots, s_n).$$

Функция выигрыша (u_i) – отображение, которое каждой комбинации стратегий всех игроков сопоставляет числовое значение выигрыша данного игрока.

$$u_i : S_1 \times S_2 \times \dots \times S_n \rightarrow \mathbb{R}.$$

Оптимальная стратегия – стратегия, обеспечивающая максимальный ожидаемый выигрыш для игрока, возможно с учётом неопределённости и вероятностных распределений.

С целью формализации игровых ситуаций и поиска оптимальных стратегий участников игры, выделяют, основываясь на том или ином признаке, множество классов игр. Например, в качестве признака можно выбрать: количество игроков (выделяют парные и n -игровые игры), степень информированности игроков (игры с полной и неполной информацией), количество игровых стратегий (конечное или бесконечное число), характер выигрыша (игры с нулевой или ненулевой суммой) [3, с. 15].

Игра с нулевой суммой – игра, в которой сумма выигрышей всех игроков для любого профиля стратегий равна нулю. Формально, данное понятие можно записать следующим образом:

$$\sum_{i=1}^n u_i(s_1, s_2, \dots, s_n) = 0, \quad (1.1)$$

где u_i – функция выигрыша i -го игрока.

Такие игры моделируют ситуации чистой конкуренции, где интересы игроков полностью противоположны.

В свою очередь игра с ненулевой суммой – игра, в которой равенство (1.1) не выполняется.

Игра с полной информацией – игра, в которой каждому игроку известны все параметры и состояния игры, а также все предыдущие ходы и возможные стратегии других игроков.

Игра с неполной информацией – игра, в которой хотя бы один игрок не обладает полной информацией о состоянии игры, возможных стратегиях или функциях выигрыша других игроков.

В теории игр основной целью является поиск оптимальных стратегий для игроков, позволяющих максимизировать свой выигрыш (или минимизировать потери) в условиях конкуренции. Для игр с нулевой суммой и неполной информацией данная задача сводится к определению оптимальных стратегий в условиях неопределённости, когда часть информации о состоянии игры и действиях других игроков скрыта.

В прикладном смысле решение такой задачи позволяет не только повысить эффективность систем принятия решений, но и использовать полученные модели для анализа реальных ситуаций.

Методы решения проблем игр с неполной информацией и нулевой суммой

Игры с нулевой суммой и неполной информацией представляют особый интерес из-за сложности поиска оптимальных стратегий. Решение таких игр требует не только логики, но и оценки вероятностей, что делает задачу значительно сложнее и ближе к реальным задачам искусственного интеллекта. Также, интерес к данной классификации игр вызван высоким уровнем реалистичности. Большинство реальных конфликтов и конкурирующих систем (например, экономические рынки, карточные игры, кибербезопасность) имеют скрытые параметры и конкурирующие интересы. Именно такие игры привели к развитию новых разделов теории игр, включая байесовские равновесия и вероятностные методы.

Из-за специфики данного класса игр выделяют следующие особенности:

- неполнота данных;

- наличие обмана;
- большая роль вероятности.

Исходя из особенностей игр с неполной информацией и нулевой суммой, можно выделить ряд методов для нахождения оптимальных стратегий.

Метод Монте-Карло

Метод Монте-Карло – численный метод решения различных задач при помощи моделирования случайных событий, основанный на получении большого числа реализаций случайных величин, которые формируются таким образом, чтобы их вероятностные характеристики совпадали с аналогичными величинами решаемой задачи.

Основная идея метода состоит в использовании выборки случайных чисел для получения искомых оценок. Вместо описания процесса с помощью аналитического аппарата производится «розыгрыш» случайного явления с помощью специально организованной процедуры, включающей в себя элементы случайности [4, с. 7].

В играх с неполной информацией и нулевой суммой данный метод помогает найти или приблизить оптимальную стратегию игрока, когда точный расчет слишком сложен или невозможен. Для нахождения оптимальной стратегии, случайным образом моделируется множество раскладов игры с учётом скрытых параметров, затем игрок применяет каждую из возможных стратегий, после чего производится анализ результатов и отбор стратегии, показавшей наилучшие результаты.

Генетические алгоритмы

Генетические алгоритмы — это класс эволюционных методов, вдохновлённых процессами естественного отбора и биологической эволюции. Они особенно полезны для поиска и оптимизации стратегий в сложных, высоко размерных и плохо формализуемых пространствах, где традиционные методы оказываются неэффективны. Генетические алгоритмы широко используются для разработки стратегий в настольных играх, робототехнике, экономическом моделировании и других областях, где пространство решений слишком велико для полного перебора.

Алгоритмы обучения с подкреплением

Обучение с подкреплением – это область машинного обучения, в которой игрок учится оптимальной стратегии через взаимодействие, получая обратную связь в виде наград или штрафов. В процессе обучения с подкреплением стратегия совершенствуется даже в условиях частичной наблюдаемости, что даёт возможность учитывать неопределённость и использовать блеф или скрытую информацию для повышения выигрыша. Однако, несмотря на все

достоинства, обучение с подкреплением имеет и определённые недостатки. Во-первых, такой подход зачастую требует значительных вычислительных ресурсов и большого числа симуляций, что делает процесс обучения достаточно длительным. Во-вторых, агенту может понадобиться тщательная настройка параметров и архитектуры, чтобы избежать переобучения или застревания на неэффективных стратегиях. Кроме того, в играх с особенно сложной и нестабильной динамикой, стратегия, полученная с помощью обучения с подкреплением, может оказаться недостаточно устойчивой к неожиданным тактикам соперника, особенно если в процессе обучения не учитывались редкие, но критичные сценарии.

Глубокое обучение

Данный метод особенно применим в играх с большим числом возможных состояний. Применение глубоких нейронных сетей даёт возможность анализировать и обрабатывать многомерные игровые пространства, недоступные для классических алгоритмов. Благодаря этому становятся возможными такие достижения, как победы искусственного интеллекта в игре го или превосходство над профессиональными игроками в покер, где традиционные методы анализа не могли обеспечить необходимой гибкости и глубины стратегии. Глубокое обучение также позволяет автоматически выявлять сложные паттерны поведения, что делает агенту доступными новые, ранее неочевидные пути достижения преимущества.

Покер как игра с неполной информацией

В качестве исследования и реализации одного из методов в данной курсовой работе выбрана игра, которая является одной из разновидностей игры покер – техасский холдем.

Покер — один из наиболее известных примеров игр с нулевой суммой и неполной информацией. В классических покерных играх игроки делают ставки, имея на руках скрытую от других игроков информацию (карты), а также наблюдая частично открытую информацию (общие карты, ставки других игроков).

Цель игры – нахождение оптимальной стратегии поведения игрока, которая максимизирует его ожидаемый выигрыш в течение серии игр в Техасский холдем против некоторого множества оппонентов. Данная задача относится к классу оптимизационных задач.

Формализация задачи

Введём следующие обозначения:

$N \in \mathbb{N}$ – количество игроков, участвующих в игре.

$s \in S$ – состояние игры, где S – множество возможных состояний игры.

$a \in A$ – действие в игре, где A – набор возможных действий игры.

$\pi : S \rightarrow \Delta(A)$ – стратегия игрока, где $\Delta(A)$ распределение вероятностей на множестве действий.

$U(\pi)$ – функция выигрыша.

Π – множество допустимых стратегий.

Тогда задача сводится к нахождению стратегии игрока π^* , удовлетворяющей следующему равенству:

$$\pi^* = \arg \max_{\pi \in \Pi} \mathbb{E}[U(\pi)].$$

Правила игры

Техасский Холдем – одна из самых популярных разновидностей покера, в которой игроки соревнуются, составляя наилучшую комбинацию из пяти карт, используя две свои закрытые карты и до пяти общих (открытых) карт на столе. Данная игра насчитывает обычно от 2 до 10 человек за столом. Каждому игроку в начале раздачи выдаются две закрытые карты – они известны только самому игроку. В течение игры на стол выкладываются до пяти общих карт, которые доступны всем игрокам. Игроки поочередно совершают действия: уравнивать, повысить, сбросить карты. Игра делится на четыре фазы: pre-flop, flop, turn, river. В каждую из фаз, за исключением стадии pre-flop, на стол добавляется одна, а в случае фазы flop три карты из колоды. После каждой фазы игроки могут принимать решения: уравнивать (*call*), повысить (*raise*) или сбросить карты (*fold*). Ставки совершаются по кругу, начиная с игрока, следующего за большим блайндом (на pre-flop), а затем – с малого блайнда. Чтобы стимулировать активную игру, в каждой раздаче два игрока обязаны делать слепые ставки: малый (small) и большой (big) блайнд. Эти ставки "входят в игру" до раздачи карт. Раздача заканчивается, когда:

- все игроки, кроме одного, сбросили карты — и этот игрок автоматически выигрывает банк;
- если два или более игроков дошли до конца раздачи, происходит вскрытие карт. Побеждает игрок с лучшей комбинацией из пяти карт.

В разновидности техасский холдем существует 10 комбинаций, каждая имеет свой ранг. В случае равенства комбинаций, сравниваются старшие карты [5].

Холдем относится к играм с неполной информацией, поскольку:

- игроки не знают карты других игроков;
- игроки не знают, какие действия основаны на блефе, а какие – на реальной силе руки.

Это делает игру сложной для математического анализа и идеально подходящей для обучения агентов с использованием эвристических или эволюционных подходов – таких как генетические алгоритмы.

В практической части работы особое внимание будет уделено реализации нейросетевого подхода с целью его последующего сравнительного анализа с генетическими алгоритмами и оценки эффективности в условиях неопределенности.

ГЛАВА II НЕЙРОСЕТЕВЫЕ МЕТОДЫ И ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ В ИГРАХ С НЕПОЛНОЙ ИНФОРМАЦИЕЙ

Принципы работы искусственных нейронных сетей

Нейронные сети (НС) представляют собой математические модели, вдохновленные принципами работы биологических нервных систем. Основой таких сетей является перцептрон – модель, предложенная Фрэнком Розенблаттом, которая имитирует процесс обработки информации нейроном. Перцептрон принимает вектор входных данных, умножает его на весовые коэффициенты, суммирует и передает результат через функцию активации для получения выходного сигнала. Данную модель можно проиллюстрировать при помощи схемы (рис. 2.1) [6, с. 20].

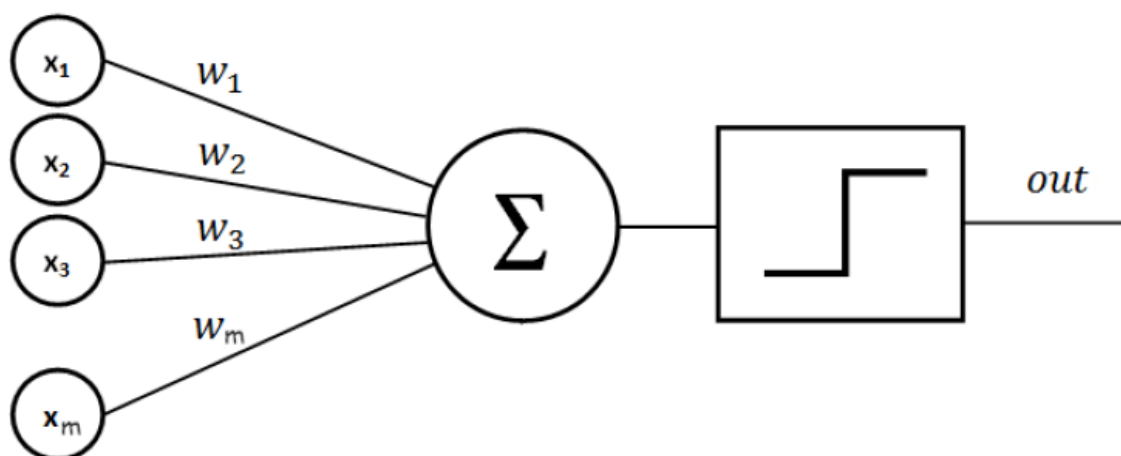


Рисунок 2.1 – Схематичное представление модели перцептрона.

Современные задачи, такие как игра в покер, требуют использования глубокого обучения (Deep Learning). Глубокие нейронные сети состоят из множества слоев нейронов, что позволяет им выявлять сложные закономерности в данных и аппроксимировать нелинейные функции. Важнейшим компонентом обучения таких сетей является алгоритм обратного распространения ошибки (Backpropagation).

Backpropagation – это алгоритм, основанный на градиентном спуске для коррекции весов. Сначала выбирается функция стоимости. Функция стоимости описывает, как вычислить ошибку между прогнозируемым и реальным выходным значением. Распространенной функцией стоимости является среднеквадратичная ошибка:

$$Cost = \frac{(y_{pred} - y_{real})^2}{2},$$

где y_{pred} — прогнозируемое значение величины, y_{real} — действительное значение величины.

Функция стоимости показывает, насколько предсказание отличается от желаемого выходного значения. Используя эту меру, алгоритм обратного распространения ошибки применяет градиентный спуск для вычисления того, насколько нужно изменить параметры сети w , чтобы минимизировать функцию стоимости в соответствии со скоростью обучения α :

$$w = w - \alpha \nabla_w C.$$

Когда ошибка становится приемлемой, обратное распространение ошибки останавливается, и нейронная сеть считается обученной [6, с. 22].

Для успешного обучения в скрытых слоях часто используются нелинейные функции активации, такие как ReLU (Rectified Linear Unit) (рис. 2.2), которая позволяет избежать проблемы затухания градиента и ускоряет процесс обучения, или гиперболический тангенс.

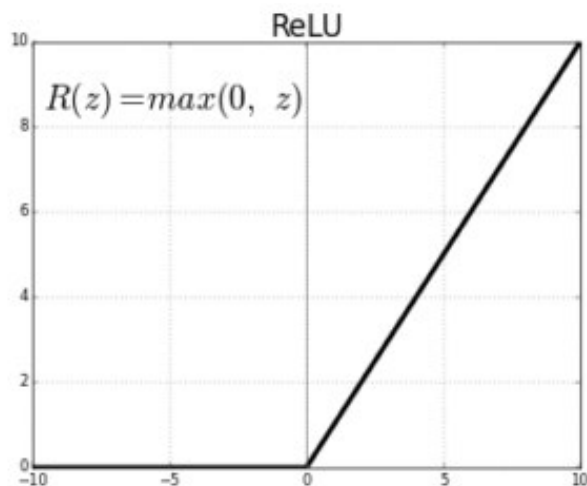


Рисунок 2.2 Функция активации ReLU.

Обучение с подкреплением (Reinforcement Learning)

Обучение с подкреплением (RL) — это раздел машинного обучения, где агент обучается, взаимодействуя с динамической средой. В отличие от обучения с учителем, где есть правильные ответы, в RL агент получает обратную связь в виде наград или штрафов за свои действия.

Процесс формализуется как Марковский процесс принятия решений (MDP), который описывается кортежем (S, A, P, R) , где S — множество состояний, A — множество действий, P — вероятность перехода между состояниями, а R —

функция награды. В покере целью агента является нахождение такой стратегии (политики) $\pi(S)$, которая максимизирует ожидаемую сумму дисконтированных наград за время игры.

Особенность покера заключается в том, что это игра с неполной информацией. Агент не видит карт оппонентов, поэтому состояние среды S является частично наблюдаемым. В таких условиях агент строит свою стратегию на основе наблюдений O , которые являются функцией от глобального состояния мира W [7, с. 4].

Алгоритмы обучения с подкреплением можно классифицировать по тому, как агент строит свою стратегию и использует ли он модель среды для планирования.

Классификация моделей обучения с подкреплением:

1. По способу построения стратегии:

- методы, основанные на ценности (Value-based): Цель – оценить функцию ценности $V(s)$ (ожидаемую награду в состоянии s) или функцию ценности действия $Q(s, a)$ (ожидаемую награду за совершение действия a в состоянии s). Стратегия π затем выводится из этой функции ценности (например, выбирается действие с наибольшим Q). Примером является Q-обучение и DQN;
- методы, основанные на политике (Policy-based): Агент напрямую обучается оптимальной стратегии $\pi(a|s)$, которая представляет собой распределение вероятностей действий в данном состоянии. Эти методы часто лучше сходятся в больших и непрерывных пространствах действий. Примерами являются REINFORCE и PPO;
- методы Actor-Critic: комбинируют оба подхода, используя две сети: Актера (Policy-based) для выбора действия и Критика (Value-based) для оценки этого действия.

2. По наличию модели среды:

- модели-свободные алгоритмы (Model-Free): Агент обучается, непосредственно взаимодействуя со средой, без построения явной модели функции перехода P и функции награды R . Большинство современных алгоритмов, таких как DQN и Actor-Critic, являются модели-свободными;
- модели-зависимые алгоритмы (Model-Based): Агент строит внутреннюю модель динамики среды, которую затем использует для планирования и прогнозирования будущих состояний и

наград. Это позволяет агенту проигрывать сценарии и принимать решения без непосредственного взаимодействия со средой.

Подход на основе Q-обучения (DQN)

Одним из популярных методов RL является Q-обучение, где агент стремится выучить функцию ценности действия $Q(s, a)$, оценивающую полезность совершения действия a в состоянии s . В задачах с большим пространством состояний, таких как покер, невозможно хранить все значения Q в таблице, поэтому для аппроксимации этой функции используются нейронные сети. Такой подход получил название Deep Q-Network (DQN).

Однако методы, основанные только на функции ценности, имеют недостатки при работе в пространствах высокой размерности и могут быть нестабильны в стохастических играх с неполной информацией. Это приводит к необходимости использования более сложных архитектур, сочетающих преимущества различных подходов.

Архитектура Actor-Critic в играх с неполной информацией

Для эффективного построения стратегий в многопользовательском покере наиболее перспективным является подход Actor-Critic. Этот метод объединяет преимущества методов, основанных на политике (Policy-based), и методов, основанных на ценности (Value-based).

Архитектура состоит из двух нейронных сетей:

- сеть Актера (Actor): отвечает за принятие решений. Она принимает на вход наблюдение агента (неполную информацию, доступную игроку) и выдает распределение вероятностей действий;
- сеть Критика (Critic): оценивает действия Актера. Главная особенность применения этого метода в покере заключается в том, что Критик может обучаться на основе полной информации о состоянии игры, включая карты оппонентов, которая доступна на этапе тренировки, но недоступна во время реальной игры.

Такой подход позволяет реализовать парадигму «централизованное обучение, децентрализованное исполнение». Во время обучения Критик использует глобальное состояние W для вычисления точной оценки $Q(W, a)$, направляя обучение Актера. После завершения обучения Актер способен действовать самостоятельно, опираясь только на свои локальные наблюдения O .

Одной из особенностей реализации данного подхода в игре покер является обработка истории действий, которая имеет неопределенную длину. В архитектуре сетей для решения поставленной задачи эффективно использовать слои LSTM (Long Short-Term Memory). LSTM позволяет

извлекать признаки из последовательности действий оппонентов, что критически важно для понимания контекста раздачи. Статическая информация (карты на столе, размеры стеков) обрабатывается полносвязными слоями. Представление данного метода изображено на схеме (рис. 2.3) [7, с.5].

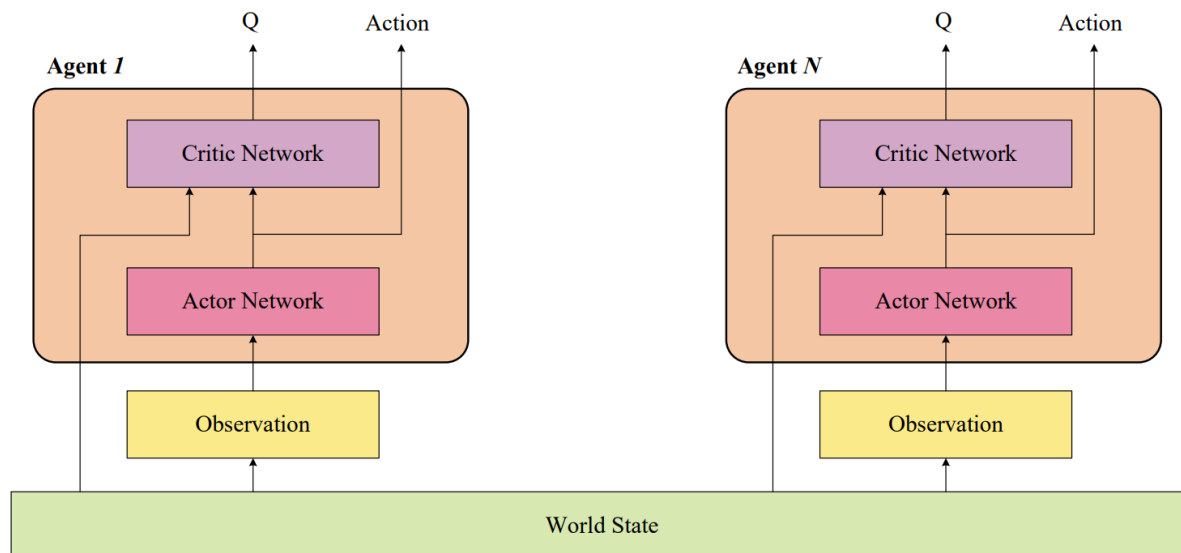


Рисунок 2.3 Схематичное представление подхода Actor-Critic.

ГЛАВА III ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Для успешной реализации и применения методов обучения нейронной сети была разработана программная модель игры в техасский холдем, адаптированная для тестирования.

Архитектура и реализация модели игры

Программная реализация представляет собой проект, созданный на языке программирования python. Реализованная модель представляет собой вариацию игры, известную как Heads-up Limited Texas Hold'em (HULHE). Выбор именно этой модификации обусловлен необходимостью снижения размерности пространства действий для эффективного обучения агентов в рамках исследования. Специфика реализации заключается в следующем:

1. Формат Heads-up. В игре принимают участие ровно два агента (игрок и оппонент). Это позволяет сосредоточиться на моделировании стратегий без учета мультиагентной динамики, характерной для полных столов.
2. Лимитированная структура ставок. В отличие от безлимитного холдема, размер повышения ставки строго фиксирован. Это дискретизирует пространство действий, сводя его к трем основным опциям: сброс (*fold*), уравнивание (*call*) и повышение (*raise*) на фиксированную величину. В программной реализации шаг повышения задается константой, что значительно упрощает сходимость нейросетевых алгоритмов.

Для выдачи карт реализован специальный класс *Card*, который имеет два параметра: *suite* и *value*. Также реализован класс, соответствующий колоде карт *Deck*. Полем данного класса является массив, состоящий из 52 карт. Также реализован процесс перемешивания карт в колоде при помощи метода *Deck.shuffle()*. Основные правила и механизм фаз в каждом раунде реализованы при помощи класса *GameManager*, который хранит в качестве поля класса данные об игре, объект класса *Game*. В данной классе реализована такая логика как:

- корректный опрос игроков в соответствии с правилами игры, в зависимости от реализации игрока;
- корректная реализация повышения или понижения ставок;
- запоминание действий каждого игрока.

В качестве класса для тестирования был создан класс *Player*, способный играть в игровую модель. Данный класс имеет функцию *ask_player()* с помощью которой пользователь может выбрать одно из трёх действий: *fold*, *call*, *raise*. Помимо этого, в качестве тестовых игроков были реализованы классы *RandomPlayer* – модель игрока, случайно выбирающего действия,

CallingPlayer – модель игрока, всегда выбирающего действие уравнивать (*call*), *SimpleGeneticBot* – модель адаптивного игрока, где геном кодирует уникальный стиль игры, что позволяет эволюционировать стратегию через генетические алгоритмы, отбирая наиболее успешные комбинации поведенческих коэффициентов. Все модели наследованы от класса *Player* и способные выполнять необходимые для игры действия в каждом раунде.

Формализация и реализация игрового агента

Для реализации агента, обучающегося с подкреплением, была выбрана архитектура Actor-Critic, реализованная с использованием библиотеки *PyTorch*. Данный подход позволяет одновременно аппроксимировать стратегию поведения (Actor) и функцию ценности состояния (Critic).

В классе *ActorCriticNet* реализованы две независимые полносвязные нейронные сети, разделяющие ответственность за выбор действия и оценку ситуации:

1. Сеть Актера (Actor Network):

- вход: вектор локального состояния S_{actor} размерностью 7 признаков (сила руки, нормированная текущая ставка агента, нормализованный текущий стек агента, нормализованный размер банка, стадия игры, история решений игрока в текущей раздаче, история решений всех игроков в текущей раздаче);
- скрытые слои: два слоя (128 и 64 нейрона) с функцией активации ReLU (рис. 3.1);
- выход: вектор логитов размерностью 3, соответствующий действиям: fold, raise, call. Для выбора действия используется категориальное распределение.

2. Сеть Критика (Critic Network):

- вход: расширенный вектор состояния S_{critic} размерностью 8 признаков;
- особенность обучения: в вектор S_{critic} добавляется параметр *avg_opp_strength* (средняя сила руки оппонента), который вычисляется на основе полной информации о раздаче. Это реализует принцип «централизованного обучения»: Критик располагает информацией о картах соперника во время тренировки, что позволяет точнее оценивать функцию ценности $V(s)$, в то время как Актер принимает решения, опираясь только на публичную информацию;
- выход: скалярное значение – оценка текущего состояния.

Обучение агента происходит эпизодически. В классе *NeuralACAgentManager* реализован буфер опыта, накапливающий данные о состояниях, действиях и

наградах. Обновление весов происходит после завершения эпизода с использованием дисконтированной награды. Функция потерь (*Loss*) комбинирует три компонента:

$$Loss = L_{actor} + 0.5 \cdot L_{critic} - 0.01 \cdot H(\pi),$$

где:

- $L_{actor} = - \sum \log(\pi(a|s)) \cdot A(s, a)$ – потеря Актера, направленная на максимизацию вероятности действий с положительным преимуществом.
- $L_{critic} = MSE(V_{pred}, R)$ – среднеквадратичная ошибка предсказания награды Критиком.
- $H(\pi)$ – энтропия распределения вероятностей, добавленная с коэффициентом 0.01 для стимулирования исследования среды и предотвращения преждевременной сходимости к локальным минимумам. В качестве оптимизатора используется алгоритм *Adam* со скоростью обучения 10^{-3} .

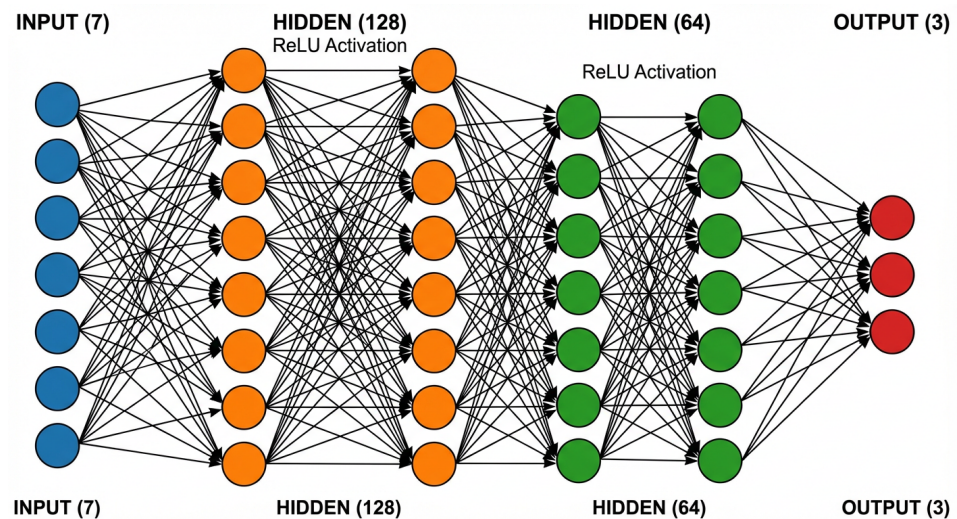


Рисунок 3.1 Схематичное представление архитектуры нейронной сети Актера.

Особое внимание уделено предобработке данных. Сила руки вычисляется с помощью метода Монте-Карло, описанного далее. Входные параметры, такие как размеры стеков и ставок, подвергаются нормировке для обеспечения стабильной работы градиентных методов.

Механизм сохранения и восстановления состояния модели

Одной из фундаментальных проблем применения алгоритмов обучения с подкреплением (RL) в играх с неполной информацией является низкая скорость сходимости. Из-за высокой дисперсии результатов в покере и стохастической природы раздач агенту необходимо провести сотни тысяч

эпизодов для выявления устойчивых закономерностей и формирования оптимальной стратегии. В связи с этим, критически важным компонентом программной реализации является система сохранения промежуточных состояний.

В классе *NeuralACAgentManager* реализованы методы сериализации и десериализации агента, позволяющие прерывать и возобновлять процесс обучения.

Метод *save_ac_agent* использует инструменты библиотеки *PyTorch* для сохранения полного контекста обучения в бинарный файл. Сохраняемый словарь включает в себя: веса нейронных сетей, состояние оптимизатора, буфер воспроизведения опыта, гиперпараметры, необходимые для корректного восстановления архитектуры.

Метод *load_ac_agent* реализует загрузку сохраненной модели. При инициализации происходит проверка соответствия архитектуры сохраненным весам. Реализация предусматривает автоматическое перераспределение вычислений на доступное устройство.

Оценка силы руки

Определим силу руки как вероятность выигрыша текущей руки против одного или нескольких противников, учитывая все возможные варианты оставшихся карт, которых мы не видим. Тогда нахождение данной величины является стохастической комбинаторной задачей.

Пусть:

C – множество всех карт стандартной колоды, $|C| = 52$.

$H = \{c_1, c_2\} \subset C$ – карманные карты игрока (известны).

$B = \{c_3, \dots, c_k\} \subset C \setminus H$ – карты на столе (известные общие карты), $k \in \{0, 3, 4, 5\}$.

$U = C \setminus (H \cup B)$ – неизвестные карты.

$H' = \{c_7, c_8\} \subset U$ – рука оппонента.

$R = \{c_{k+1}, \dots, c_5\} \subset U \setminus H'$ – недостающие общие карты (если $k < 5$).

Обозначим:

$V(H, B, R)$ – сила руки игрока после полной раздачи.

$V(H', B, R)$ – сила руки оппонента после полной раздачи.

Функция выигрыша:

$$W(H, H', B, R) = \begin{cases} 1, & V(H) > V(H'), \\ 0.5, & V(H) = V(H'), \\ 0, & V(H) < V(H'). \end{cases}$$

Задача вычислить:

$$\mathbb{E}[W(H, H', B, R)] = \mathbb{P}(\text{игрок выигрывает}) + 0.5 \cdot \mathbb{P}(\text{ничья}),$$

где \mathbb{P} – вероятность по равномерному распределению при всех допустимых H', R, U .

Тогда сила руки:

$$S(H, B) = \mathbb{E}_{H', R} [W(H, H', B, R)].$$

Сложность задачи:

$$H': C_{|U|}^2,$$
$$R: C_{|U|-2}^{5-k}.$$

Тогда комбинированное число случаев:

$$O\left((C_{|U|}^2 * C_{|U|-2}^{5-k})\right) \gg 10^6.$$

Точная оценка становится экспоненциально дорогой при увеличении числа игроков или недостающих карт.

Для решения данной задачи было принято решение воспользоваться методом Монте-Карло для нахождения силы руки.

Шаги реализованного метода:

1. Фиксация известных карты.
2. Генерация случайным образом:
Оставшихся карт (одна или две для финиша стола).
Руки оппонента (2 карты).
3. Оценка расклада.
4. Повторение процесса N раз.
5. Вычисление относительной частоты побед (это и будет оценкой вероятности выигрыша руки).

$$WinRate = \frac{\text{победы игрока}}{\text{всего симуляций}}.$$

Данная функция в проекте реализована в файле с названием `ranking_cards.py`. В качестве оптимизации и экономии вычислительных ресурсов использовалась библиотека `eval7` для оценки силы руки игрока.

Описание генетического агента

Модель генетического агента представлена в классе *SimpleGeneticBot*.

Перед принятием решения бот рассчитывает следующие параметры:

- сила руки (*hand_strength*) – числовая оценка текущей комбинации игрока, вычисляемая функцией *evaluate_hand_strength()*. Эта функция определяет, насколько сильна рука игрока по сравнению с возможными комбинациями;
- случайность блефа (*bluff_rand*) – значение, равномерно случайное от 0 до 1. Вводится с целью моделирования элемента случайного блефа, позволяющего в редких случаях играть агрессивно даже со слабыми руками;

- отношение ставки к стеку ($self.game_bet / (self.game_bet + self.stack)$) – параметр, отражающий насколько сильно бот склонен рисковать: чем больше ставка по отношению к оставшемуся стеку, тем более осторожно бот должен себя вести.

После вычисления данных параметров высчитывается скалярное произведение двух векторов, соответствующих параметрам состояния игры и геному бота. Программная реализация вычисления оценки:

```
score = self.genome[0] * hand_strength + self.genome[1] * bluff_rand - self.genome[2] * self.game_bet / (self.game_bet + self.stack).
```

Таким образом, разные геномы задают разные стили игры: осторожный игрок будет иметь большой коэффициент у последнего веса, агрессивный – высокий вес блефа и силы руки.

После вычисления итогового значения *score* бот выбирает одно из трёх действий (*fold*, *call*, *raise*) по следующему принципу, приведённому в таблице 3.1.

Таблица 3.1 Принятие решения по итоговому значению.

Условие для <i>score</i>	Действие
$score > 0.8$	raise
$0.4 < score < 0.8$	call
$score < 0.4$	fold

В рамках эксперимента были выделены несколько типов стратегий (табл. 3.2) для игры ботов друг с другом.

Таблица 3.2 Классификация стратегий и интерпретация стратегий в качестве генома бота.

Стратегия	aggression	bluffing	tightness	Пояснение
Агрессор (Aggressor)	0.8	0.1	0.1	Часто повышает, редко блефует
Боязливый (Tight)	0.15	0.05	0.8	Играет мало рук, ждёт сильные комбинации
Блефующий (Bluff)	0.2	0.6	0.2	Много блефует, может играть с плохими руками
Сбалансированный (Balanced)	0.33	0.33	0.33	Играет сбалансировано

Стратегия	aggression	bluffing	tightness	Пояснение
Маньяк (Maniac)	0.45	0.45	0.1	Часто повышает, даже с плохими руками

Для определения оптимального соперника для обучения нейросетевого агента была проведена серия экспериментов с использованием эвристических стратегий, описанных в таблице 3.2. Целью эксперимента являлось выявление наиболее эффективной стратегии, способной служить «эталоном» для оценки качества обучения.

Был проведен турнир, состоящий из 1000 игровых симуляций по 30 раундов, в котором стратегии соревновались друг с другом. Критерием эффективности выступало итоговое количество побед. Результаты эксперимента представлены в виде диаграммы (рис. 3.1).

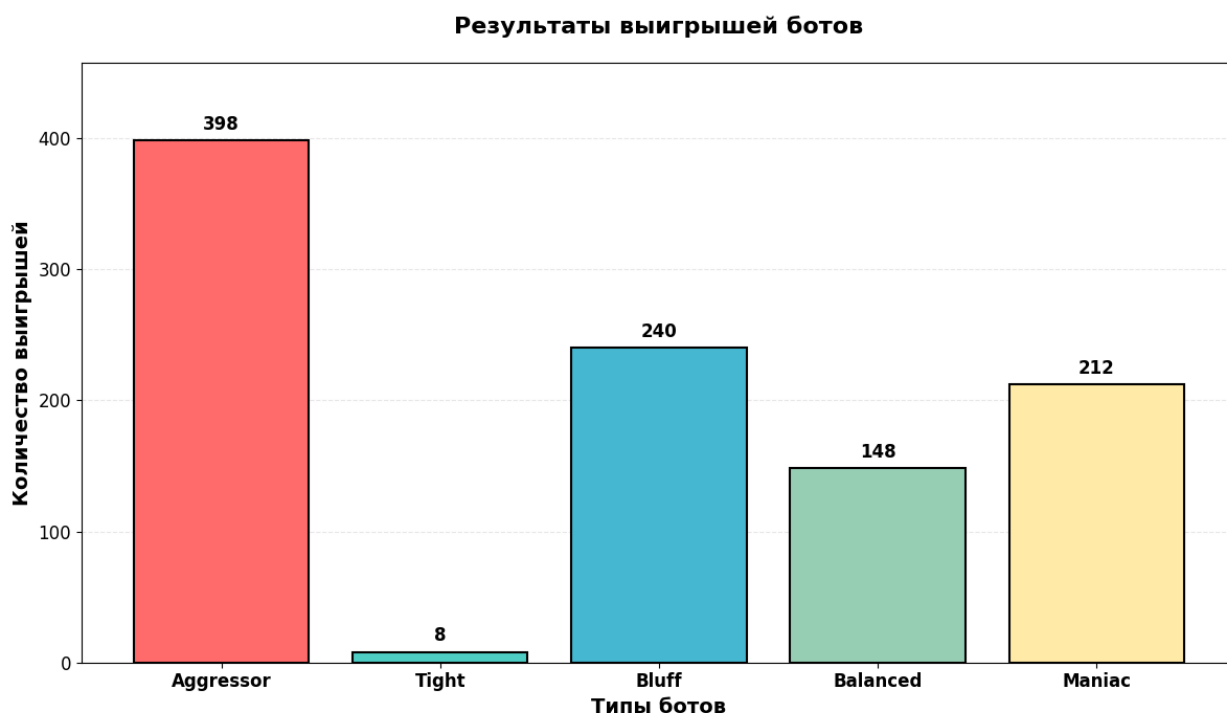


Рисунок 3.1 Результаты стратегий ботов по числу выигрышей.

Анализ результатов показал, что стратегия «Агрессор» демонстрирует наилучшие показатели, значительно опережая стратегии «Боязливый» и «Сбалансированный». Это объясняется спецификой Heads-up покера, где агрессия позволяет часто забирать банк без вскрытия карт. На основании этого стратегия «Агрессор» была выбрана в качестве основного оппонента для обучения и тестирования нейросетевого агента.

Обучение нейросетевого агента

Процесс обучения агента, основанного на методе Actor-Critic, проходил в несколько этапов, от простых соперников к сложным. Такой подход позволяет нейросети сначала усвоить базовые правила игры, а затем переходить к поиску оптимальных стратегий.

Этап 1. Обучение против случайного агента (*RandomPlayer*). На начальном этапе нейросеть обучалась против агента, выбирающего действия (*fold*, *call*, *raise*) равновероятно. Было проведено 1000 игр. Целью этапа являлась проверка корректности работы алгоритма градиентного спуска и способности агента выучить простейшую зависимость: сильная рука должна приводить к ставке, слабая – к сбросу. Агент сразу же достиг показателя выигрышей в 100%. Это говорит о том, что случайный выбор действия – самая неэффективная стратегия в данной игре.

Этап 2. Обучение против пассивного агента. Далее оппонентом выступал *CallingPlayer* – агент, который всегда уравнивает ставки, но никогда не повышает первым. Было проведено 30 000 игр. Данный тип противника требует от нейросети умения эксплуатировать пассивность: ставить много с сильными руками и не блефовать. Как можно заметить на графике (рис. 3.2) нейросетевой агент по истечению 5000 игр стал стабильно играть лучше агента *CallingPlayer* и к концу эксперимента достиг показателя побед равного 65%, продолжая рост.

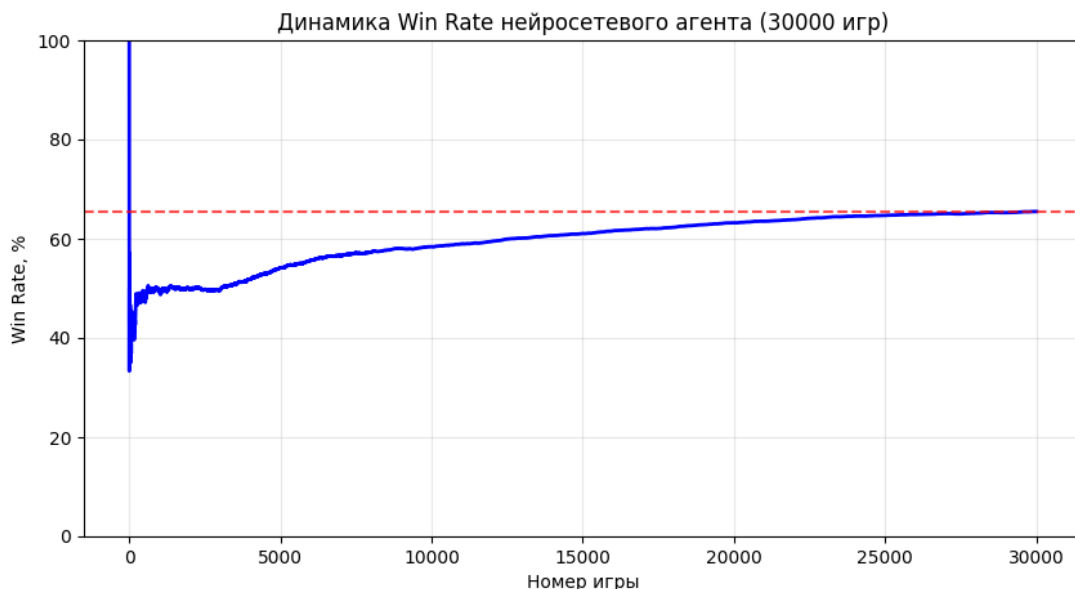


Рисунок 3.2 – График процента побед нейросетевого агента против *CallingPlayer*.

Этап 3. Обучение против стратегии «Агрессор». Финальным и наиболее сложным этапом стало обучение против лучшей эвристической стратегии – «Агрессор». Было проведено 10 000 симуляций. Агрессивный бот оказывает постоянное давление на нейросеть, часто повышая ставки, что затрудняет обучение «Критика» из-за высокой дисперсии результатов.

По итогам 10 000 эпизодов нейросетевой агент вышел с показателем побед около 50% (рис. 3.3).



Рисунок 3.3 – Динамика результативности нейросетевого агента против стратегии «Агрессор».

Анализ результатов нейросетевого подхода

Достижение показателя в 50% побед против сильнейшей жестко заданной стратегии является значимым результатом, учитывая ограниченность ресурсов обучения. Тот факт, что агент не проигрывает «всухую», а удерживает паритет, говорит о том, что нейросеть научилась базовым принципам защиты и выбора рук. Однако отсутствие доминирования над эвристическим алгоритмом обусловлено следующими факторами:

- недостаточный объем выборки. Алгоритмы обучения с подкреплением, такие как Actor-Critic, требуют миллионов эпизодов для сходимости в стохастических средах. Выборка в 10 000 игр является минимальной для задачи покера;
- ограниченность представления состояния. Текущий вектор состояния S_{actor} подает на вход сети только мгновенные параметры (текущая ставка, стадия). Он не учитывает временную динамику, что критически важно для распознавания блефа. Отсутствие рекуррентных слоев

(LSTM/GRU) в архитектуре не позволяет агенту эффективно запоминать контекст агрессии оппонента;

- статичность гиперпараметров. Обучение проводилось с фиксированным шагом обучения и коэффициентом энтропии, что могло привести к застреванию в локальном оптимуме.

Несмотря на эти ограничения, эксперимент подтвердил гипотезу о возможности применения методов RL для создания конкурентоспособных агентов в HULHE без использования экспертных знаний, исключительно на основе взаимодействия со средой.

Перспективы модификации и дальнейшего исследования

Для преодоления барьера в 50% и достижения высокого уровня игры возможно внедрение следующих улучшений:

- внедрение рекуррентных нейронных сетей (RNN/LSTM). Замена полносвязных слоев на LSTM-блоки позволит агенту сохранять скрытое состояние между ходами. Это даст возможность агенту «помнить» историю ставок в текущей раздаче и адаптироваться к частоте агрессии оппонента;
- метод игры с самим собой. Обучение против фиксированных стратегий приводит к переобучению под конкретного противника. Переход к методу Self-Play, когда агент играет против своей копии из прошлого, позволит находить более робастные стратегии, приближающиеся к Равновесию Нэша, а не просто эксплуатирующие слабости конкретного бота;
- использование PPO (Proximal Policy Optimization). Переход от классического Actor-Critic к PPO повысит стабильность обучения за счет ограничения шага обновления политики, что критически важно в дисперсионных играх.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была решена задача построения и сравнительного анализа интеллектуальных агентов для игры в Heads-up Limited Texas Hold'em с использованием эволюционных и нейросетевых подходов.

Были систематизированы методы решения игр с неполной информацией. Показано, что покер является сложной средой из-за необходимости вероятностной оценки скрытых параметров и противодействия обману.

Была разработана гибкая среда моделирования на языке Python, в которой были реализованы:

- генетический алгоритм позволяющий оптимизировать параметры эвристического бота. Эксперименты показали, что эволюционный подход быстро сходится к локально-оптимальным стратегиям, превосходящим базовые скриптовые алгоритмы;
- нейросетевой агент. Реализована архитектура с разделением на сети Актера и Критика. Агент успешно обучился эксплуатировать слабых оппонентов (100% доля побед против *RandomPlayer*, около 65% против *CallingPlayer*).

Эксперимент показал, что генетические алгоритмы демонстрируют более быстрый и стабильный результат при наличии качественной эвристической базы (формулы оценки ситуации). Они эффективны для настройки коэффициентов агрессии и осторожности. Нейросетевой подход (RL) обладает значительно большим потенциалом, так как не ограничен рамками заданной программистом формулы, а способен находить сложные зависимости от входных параметров. Однако, в ходе экспериментов агент достиг паритета (50% побед) против сильного эвристического бота, но не смог его уверенно обыграть. Анализ выявил, что для полноценного превосходства нейросети необходима архитектурная доработка: внедрение блоков памяти (LSTM) для учета истории торгов и значительное увеличение объема обучающей выборки (более 100 000 эпизодов).

Таким образом, для быстрого создания компетентного бота в условиях ограниченных ресурсов эффективнее использовать генетические алгоритмы. Для создания агента высокого уровня, способного к самостоятельному поиску неочевидных стратегий, необходимы методы глубокого обучения с подкреплением с использованием рекуррентных архитектур и методов Self-Play.

Список использованных источников

1. Osborne, M. J. An Introduction to Game Theory. Oxford University Press, 2004. – pp. 1-9
2. Gilpin, A., Sandholm, T. Lossless abstraction of imperfect information games // Journal of the ACM. — 2007. — Vol. 54, no. 5. — pp. 1–30.
3. Кремлев, А. Г. Основные понятия теории игр / А. Г. Кремлев ; ред. О. В. Климова ; корр. Е. Е. Афанасьева ; комп. набор А. Г. Кремлева ; верстка О. П. Игнатьевой. — Екатеринбург : Изд-во Урал. ун-та, 2016. — 11-25 с.
4. Раменская, А. В. Метод Монте-Карло и инструментальные средства его реализации : методические указания / А.В. Раменская, К.В. Пивоварова; Оренбургский гос. ун-т. – Оренбург: ОГУ, 2018. – 7-13 с.
5. Rules of Poker [Электронный ресурс]. – URL: <https://www.pagat.com/poker/rules/>
6. Costa, A. M. A Study on Neural Networks for Poker Playing Agents: Dissertation presented for the degree of Mestre em Informática / A. M. Costa. — Rio de Janeiro: PUC-Rio, 2019. — 59 p.
7. Shi, D., Guo, X., Liu, Y., Fan, W. Optimal Policy of Multiplayer Poker via Actor-Critic Reinforcement Learning // Entropy. — 2022. — Vol. 24, no. 6. — pp. 1–17.

Приложение А

Ссылка на репозиторий с практической частью курсовой работы:

<https://github.com/KochevIlya/PokerGameCourseWork7Sem>