|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | http://mai.ru/life/brand/mai.gif | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**  **(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)"** | |
| Институт №3 «Системы управления, информатика и электроэнергетика» |
| Кафедра 307 «Цифровые технологии и информационные системы» |

|  |
| --- |
| **Курсовая работа по курсу: «Современные методы построения математических моделей на основе искусственных нейронных сетей»** |
|  |
| «Reinforcement learning на примере шахмат» |
| Выполнили студенты группы М30-226М-21    Копылов Д.А.  Зайцев М.Д |

Москва**.**2022

Оглавление

[Введение. 2](#_Toc119943320)

[Датасет 4](#_Toc119943321)

[Шахматный движок 5](#_Toc119943322)

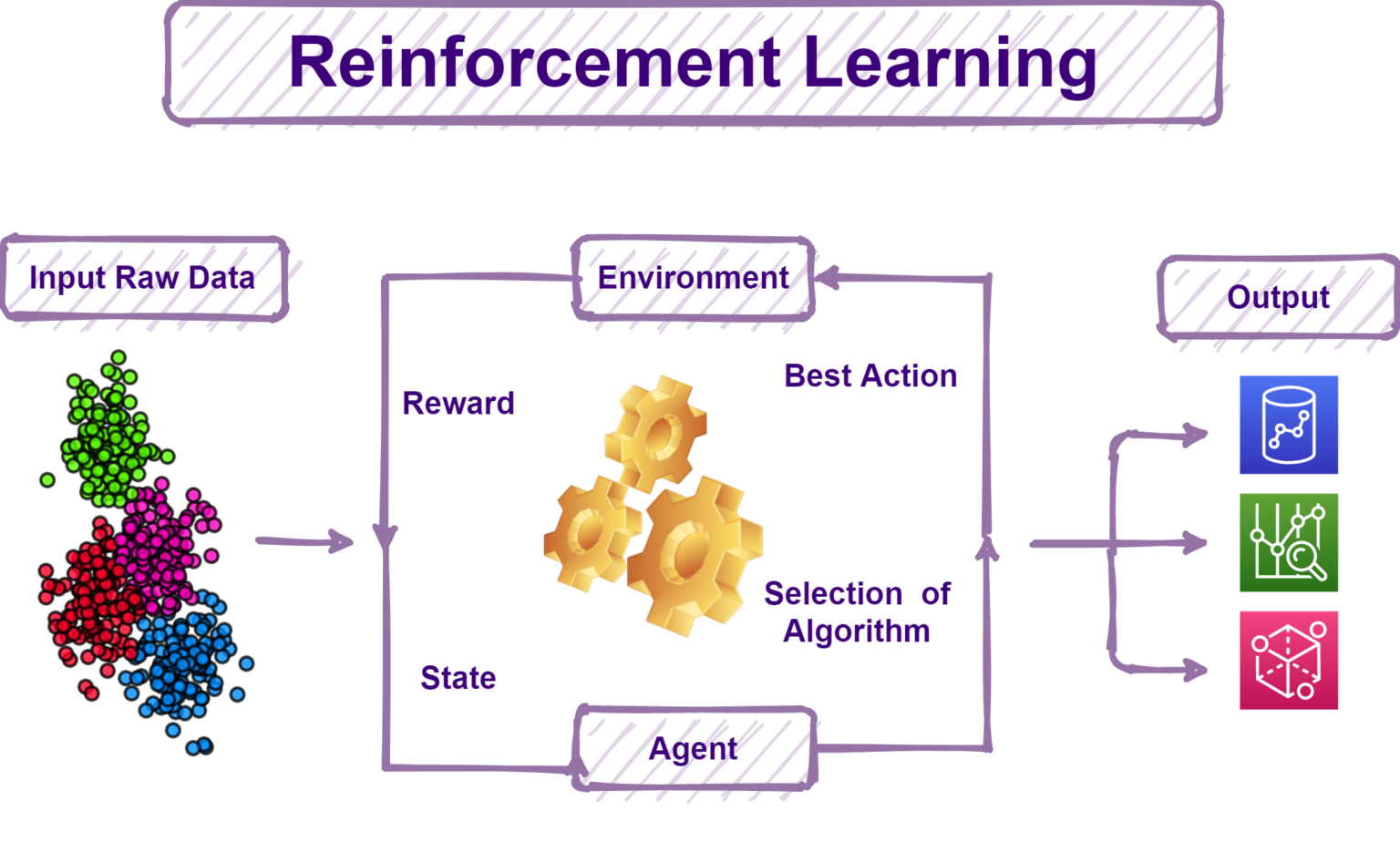
[Постановка задачи 5](#_Toc119943323)

[Обработка датасета 6](#_Toc119943324)

# Введение.

В настоящее время методы ИИ нашли своё применение почти во всех возможных отраслях, но большинство из сценариев применения являются автоматизации рутинных или давно известных процессов.

Отдельный интерес для оптимизации представляю так называемые игровые процесс. Игровые процессы представляют собой сценарий, при котором система должна взаимодействовать со средой, получая отклик и взаимодействуя с ним, а затем отправлять управляющее воздействие. В большинстве таких процессов общее количество возможных состояний среды не определено или стремится к бесконечности. Пример такой среды приведён ниже

Несмотря на то, что теория игры в шахматы не является предметом интерес науки, среда шахматной партии является хорошим тренажером для изучения алгоритмов обучения с подкреплением.

# Датасет

В целях базового исследования для данного проекта решено использовать датасет не из целых партий, а лишь из их концовок. Обучившись на таком наборе данных затем предполагается расширение его до начального положения и таким образом обучить алгоритм игре в шахматы.

На [просторах интернета](https://www.google.com/) найден датасет представляющий собой текстовые файлы с ~3000 партиями разделёнными на 4 варианта: 1 ход до победы, 2 хода до победы, 3 хода до победы, 4 хода до победы. Партии представлены в файлах в fen типе

# Шахматный движок

Для данного исследования был использован шахматный open-source движок Stockfish.

Stockfish – свободный шахматный движок с поддержкой UCI с открытым исходным кодом, доступный для различных настольных и мобильных платформ. Он разработан Марко Костальбой, Джоной Кийски, Гэри Линскоттом и Тордом Ромстадом, при большом вкладе сообщества разработчиков с открытым исходным кодом.

Stockfish занимает первые места большинства рейтинговых списков и соревнований среди компьютерных шахматных программ, и признаётся сильнейшей шахматной программой, не использующей GPU. Он выиграл неофициальный чемпионат мира по компьютерным шахматам в 6 сезоне (2014), 9 сезоне (2016), 11 сезоне (2018), 12 сезоне (2018), 13 сезоне (2018), 14 сезоне (2019) , 16 сезоне (2019),18 сезоне (2020) 19 сезоне (2020) , 20 сезоне (2021), 21 сезоне (2021) и в 22 сезоне (2022). Он финишировал вторым в 5 сезоне (2013), 7 сезоне (2014), 8 сезоне (2015), 15 (2019) и 17 (2020).

# Постановка задачи

В ходе исследовательской работы необходимо изучить алгоритмы Reinforcement learning в среде шахмат. Разработать и обучить систему способную решать шахматные задачи в один, два, три, четыре хода с использованием алгоритмов обучения с подкреплением. Для сравнения разработать игрока, выбирающего случайные ходы и проводить сравнение с ним. Дополнительно провести сравнение всех алгоритмов RE (Q-learning, DQN)

# Обработка датасета

Для последующего процесса обучения найденный датасет подавался на вход шахматному движку с целью получить не достающие шаги до победы. Недостающие шаги необходимы для последующей оценки ходов, предлагаемых разрабатываемой системы. На выходе сформирован файл формата json (распространенный формат хранения и передачи данных). Пример полученного файла ниже:

# Random agent

Рисунок . Пример JSON файла.

3,11 строки fen позиции. 5,13 следующие состояния до победы. 8,16 ходы для победы

Как уже упоминалось ранее Random agent необходим как базовый игрок, с котором возможно проводить сравнение разрабатываемых алгоритмов.

Алгоритм работы:

Игровая среда возвращает доступные позиции для возможных ходов

Агент выбирает случайный ход из списка

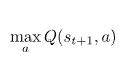
Возвращает среде, выбранный ход

# Q-learning agent

Метод q-learning основан на введении функции Q, отражающей ценность каждого возможного действия a агента для текущего состояния s, в котором сейчас находится симуляция. Или коротко:



Эта функция задает оценку агентом той награды, которую он может получить, совершив в определенный ход определенное действие. А также она включает в себя оценку того, какую награду агент может получить в будущем. Процесс обучения представляет из себя итерационное уточнение значения функции Q на каждом ходу агента. В первую очередь следует определить величину награды, которую агент получит в этот ход. Запишем ее переменной r\_t. Далее определим величину максимальной ожидаемой награды на последующих ходах:



Теперь следует решить, что для агента имеет большую ценность: сиюминутные награды или будущие. Данная проблема разрешается дополнительным коэффициентом при составляющей оценки последующих наград. В итоге предсказываемая агентом величина функции Q на данном шаге должна быть максимально приближена к значению:



Таким образом, ошибка предсказания агентом значения функции Q для текущего хода запишется следующим образом:



Введем коэффициент, который будет регулировать скорость обучения агента. Тогда формула для итерационного расчета функции Q имеет вид:



Общая формула итерационного расчета функции Q:



Алгоритм агента для исследуемого процесса:

Результаты применения и обучений Q-Agent.

Все эпохи пройдены?

Шах и мат?

Следующий ход

Инициализация класса RandomAgent

Инициализация класса Q-learningAgent

Цикл обучения на n эпох

Получение очередной партии из JSON

Цикл игры до стадии «Шах и Мат»

Игра/Обучение

Входе проведения экспериментов по обучению Q-agentа выявлено большое количество недостатков данного метода в применение к шахматам. Большинство шахматных ситуаций имеют большое количество возможных вариантов, из-за чего возникает огромное количество различных состояний.

Даже при обучении за одну сторону, например белых, возникают различные ходы оппонента и таким образом различные FEN строки, в которых позиция белых одна и та же, а оппонент меняет свои позиции.

Для решения возникающей проблемы необходимо длительное обучение с изучением большого количества состояний или разработка алгоритма «хэширования» FEN строки, второй вариант выходит за рамки данной работы. Первый вариант был испробован, но достичь взвешенной игры не вышло, полученные результаты были ближе к случайному агенту или хуже. На проверке в соперничестве со случайным агентом, часто возникала ошибка не изученного состояния

Таким образом, можно сделать следующий вывод по Q-learning алгоритму:

Алгоритм слабо или вовсе не применим для процессов с несчетным или огромным количеством возможных состояний, так как исследование занимает большое количество времени и ресурсов, а Q-table разрастается до Гб занимаемого времени, рекомендуется применять либо другие виды обучения, либо вводить дополнительные ограничения

# DQN Agent

DQN – deep q network алгоритм обучения с подкреплением на основе нейросетей, в основе лежит следующий подход:

На вход неройсети необходимо подать текущее состояние сети, иногда может требоваться дополнительная обработка входного состояния среды. Затем после прямого прохождения по сети на выходе получаем действие.

Сеть

Предобработка

Состояние

Действие

После действие передается среде, которая возвращает награду за это действие, рассчитывается разница между максимальной наградой на этом шаге и полученной, затем корректируются веса сети

Функция ошибки была выбрана следующая:

,

Где,

Rewards.T – транспонированный вектор наград за выбранный промежуток;

Actions – вектор действий за выбранный промежуток;

Sum – оператор суммирования;

\* - оператор матричного умножения

Таким образом такая функция позволяет свести разницу между максимальной наградой, которая равна 1 (победа) и текущей к минимуму

# Предобработка

В связи с использованием FEN строки, как основой форма передачи состояний от среды, решено использовать несколько различных сценариев предобработки, трансформирующих FEN строку в вектор для передачи в нейросеть.

Входе исследования и экспериментов прибегли к следующим методам:

* Токенезация
* Извлечения фитч из FEN строки

**Токенизация**

Токенизация подразумевает под собой обработку FEN строки как символьную строку, таким образом FEN строкам ставится в соответствие не который вектор заданной размерности, характеризующий конкретную строку.

Входе применения данного метода столкнулись с теми же проблемами, что и при применении Q-learning, одни и те же состояния одной из сторон получали различные токены и словарь токенов бесконечно множился и обучения не происходило

**Вектор фитч**

После неудачных экспериментов по внедрению токенизации решено использовать другой подход, основанный на трансформации FEN строки в вектор 1\*64 + дополнительные признаки из FEN строки. Данный метод показал хорошие результаты в совокупности с классическим алгоритмами машинного обучения [[1](https://www.kaggle.com/datasets/petert1/chess-position-evaluation)].

В сравнении с [1](https://www.kaggle.com/datasets/petert1/chess-position-evaluation) вектор значительно уменьшен, так как множество признаков связанных с «взятием на проходе» для решаемой задачи не имеют значения или просто невозможно установить, так как отсутствует такая информация, а также заменены признаки отвечающие за положение фигур, в качестве обозначения позиции для каждой клетке установлены 12 признаков обозначающие наличие той или иной фигуры.

# Архитектура нейросети для DQN

Исходя из того, что для DQN используются небольшие нейросети с небольшим количеством слоев и нейронов собрана сеть со следующей архитектурой:

Input (X)

Linear (2\*X, X)

Linear (X, X \\ 2)

Output (X\\2)

Linear (X, 2\*X)

Использовано 3 слоя с характеристиками, представленными на изображении выше, на выходе каждого слоя в качестве функции активации используется ReLu.

Для ускорения обучения и эксперимента добавлен ещё один вход, на которой из среды подается N наилучших ходов, такой подход должен улучшить качество разрабатываемой модели

Input (N)

Linear (N, 2\*N)

Output (2\*N)

Итоговая архитектура нейросети представляет собой объединение выходов двух ранее представленных структур, прохождение через ещё один линейный слой, а выходной слой представлен SoftMax, таким образом применив затем функцию поиска максимума, будет получен № хода из N который нужно совершить. Общая структура модели представлена ниже

Input1 (X)

Input2 (N)

Main Net (X, X\\2)

Add Net (N, 2\*N)

add (Add Net, Main Net)

Linear ((2\*N, X\\2), N)

ReLu

SoftMax (X\\2, 5)

Max (1)

# Обучение разработанной архитектуры

# Выводы