|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 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»** |
|  |
| « Применение алгоритмов обучения с подкреплением для решения шахматных задач» |
| Выполнили студенты группы М30-226М-21    Копылов Д.А.  Зайцев М.Д |

Москва**.**2022

**Оглавление**

[Введение 3](#_Toc123226452)

[Постановка задачи 4](#_Toc123226453)

[Выбор алгоритмов обучения 5](#_Toc123226454)

[Ограничение, накладываемое на задачу 7](#_Toc123226455)

[Разработка и реализации среды 9](#_Toc123226456)

[Разработка и реализация агентов 10](#_Toc123226457)

[Random agent 10](#_Toc123226458)

[Q-learning agent 10](#_Toc123226459)

[Алгоритм работы 11](#_Toc123226460)

[Результаты применения и обучений Q-Agent 13](#_Toc123226461)

[Deep Policy Agent 14](#_Toc123226462)

[Предобработка 16](#_Toc123226463)

[Токенизация 16](#_Toc123226464)

[Вектор фитч 16](#_Toc123226465)

[Тестирование и отладка архитектуры (Тюнинг модели) 17](#_Toc123226466)

[Прямой ход модели 20](#_Toc123226467)

[Игра агентов между собой 22](#_Toc123226468)

[Выводы 23](#_Toc123226469)

# Введение

Обучение с подкреплением — это тип машинного обучения, который фокусируется на обучении агентов максимизировать сигнал вознаграждения, выполняя ряд действий в окружающей среде. Это тип обучения, основанный на концепции проб и ошибок, когда агент получает обратную связь о своих действиях и учится корректировать свое поведение, чтобы максимизировать вознаграждение.

В контексте обучения компьютера игре в шахматы обучение с подкреплением включает в себя обучение агента принятию решений на шахматной доске на основе текущего состояния игры и потенциальных вознаграждений за каждый возможный ход. Это предполагает использование алгоритмов, которые могут оценивать положение на доске и выбирать наилучшее действие, основанное на текущем состоянии игры и потенциальных вознаграждениях за каждый возможный ход.

Одно из ключевых преимуществ использования обучения с подкреплением при обучении компьютера игре в шахматы заключается в том, что оно позволяет агенту учиться на собственном опыте, а не быть явно запрограммированным на правила игры. Это означает, что агент может научиться адаптироваться к новым ситуациям и стратегиям, что делает его более гибким и способным обрабатывать более широкий спектр сценариев.

В целом, обучение с подкреплением предлагает мощный подход к обучению компьютера игре в шахматы, позволяющий агенту учиться на собственном опыте и адаптироваться к новым ситуациям. Используя принципы проб и ошибок и обучения на основе вознаграждения, алгоритмы обучения с подкреплением могут быть использованы для обучения агентов принимать разумные решения и преуспевать в решении сложных задач, таких как игра в шахматы.

В настоящей работе описана реализация на языке программирования Python несколько алгоритмов обучения с подкреплением в игре шахматы.

# Постановка задачи

Задачей курсовой работы является разработка и реализация нескольких алгоритмов обучения с подкреплением.

Это задача подразумевает:

* разработку и реализацию среды (environment), которая должна сохранять текущее состояние доски, принимать ход от игрока и возвращать ответный ход (новое состояние доски) вместе с оценкой сделанного агентом (роботом) хода;
* разработку и реализацию агентов (agents), которые реализуют алгоритмы обучения с подкреплением;
* реализация процесса повторяемых шахматных партий, в ходе которых агент обучается в соответствии с конкретным алгоритмом;
* сравнение эффективности обучения агентов.

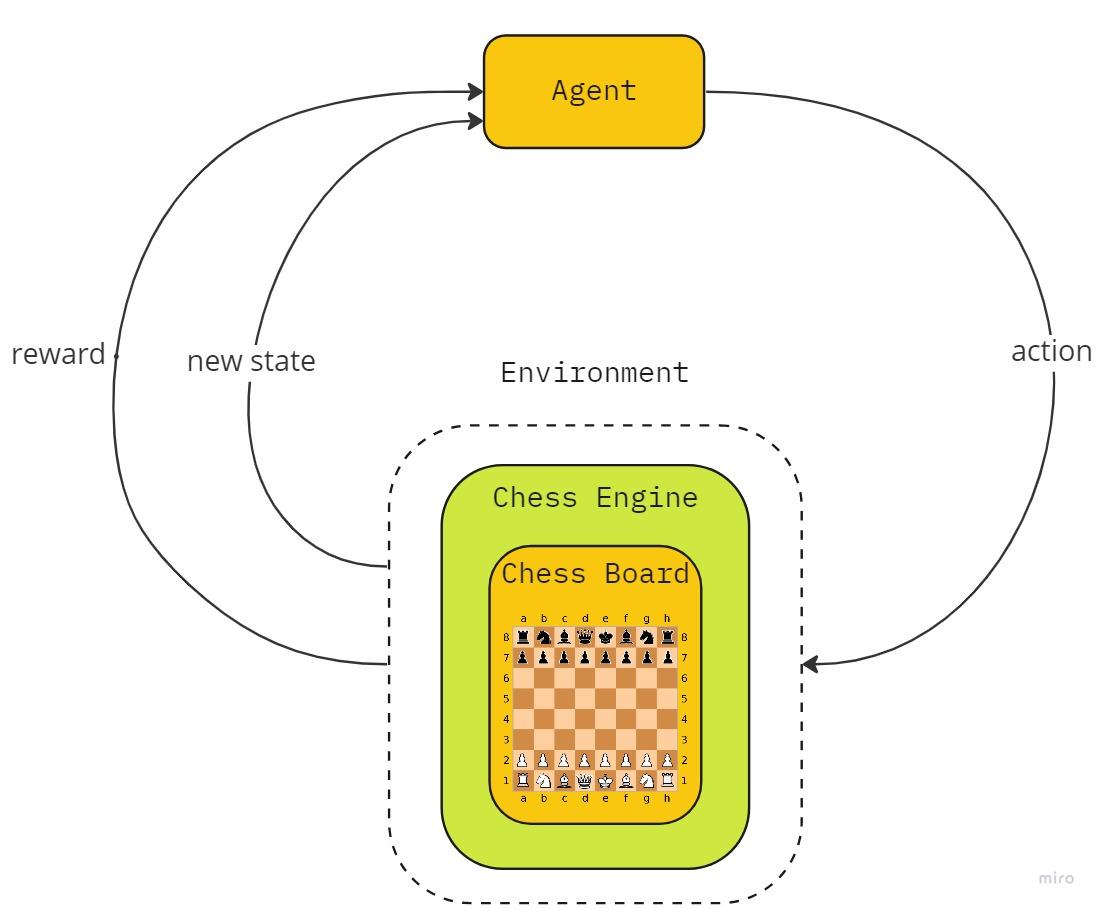
Принципиальная схема разрабатываемой программы выглядит следующим образом (Рисунок 1):

Рисунок . Схема алгоритма разрабатанного входе исследования

# Выбор алгоритмов обучения

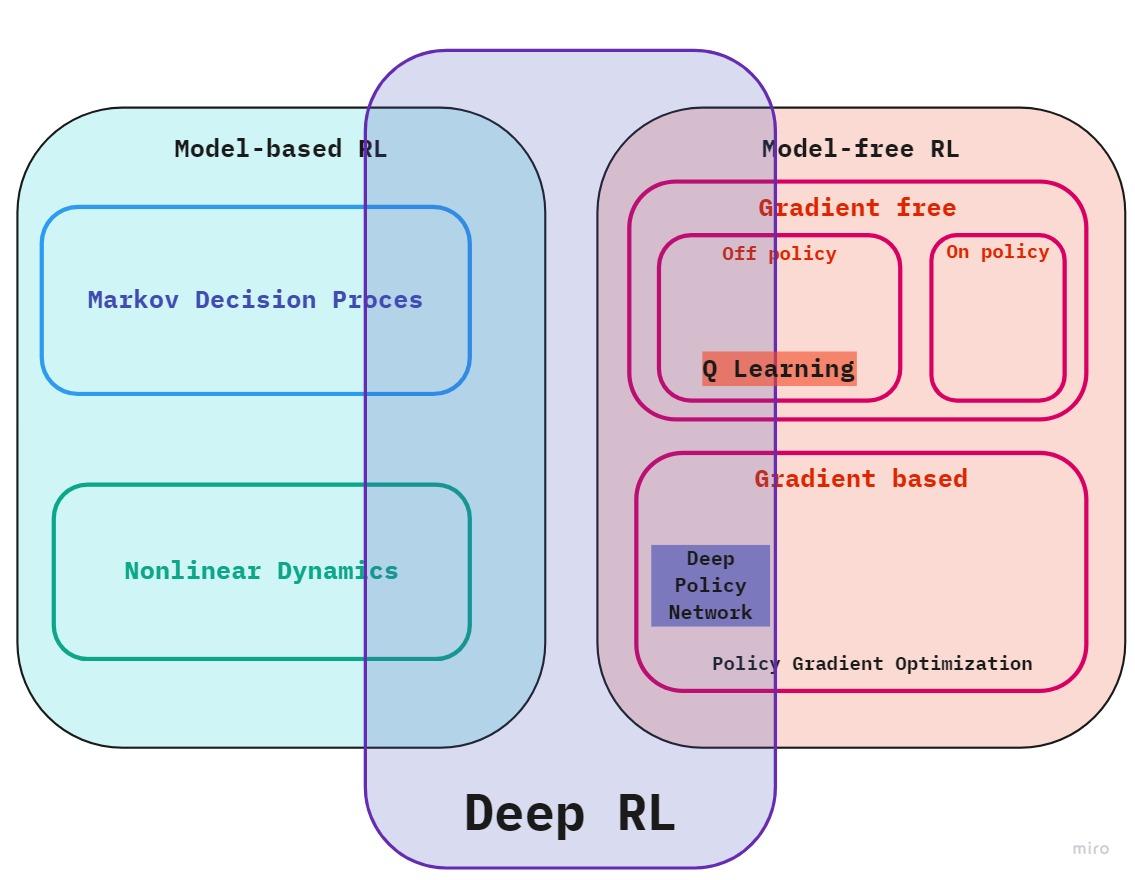


Рисунок . Схема существующих подходов к реализации RL агоритмов

Алгоритмы RL можно разделить на две основные категории: model-based и model-free (Рисунок 2). Алгоритмы без моделей не требуют явной модели динамики среды и учатся непосредственно на опыте. С другой стороны, алгоритмы на основе моделей изучают модель среды и используют эту модель для планирования и принятия решений.

В настоящей работе были выбраны два алгоритмы группы model-free: Q-learning и Deep Policy Network. В пользу этого выбора можно назвать несколько аргументов:

* Для шахмат крайне затруднительно описать модель среды и использовать её для обучения, поэтому выбор падает на model-free алгоритмы, к которым относится Q-обучение и Deep Policy Net;
* Q Learning и Deep Policy Net являются примерами алгоритмов RL, которые можно использовать для решения различных типов проблем и задач. Q-обучение — это алгоритм, основанный на ценности, который оценивает ценность выполнения действия в заданном состоянии, а Deep Policy Net — это алгоритм, основанный на политике, который оценивает градиент ожидаемого вознаграждения по отношению к параметрам политики. Это может сделать их интересными и поучительными для сравнения и противопоставления;
* Оба алгоритма широко используются и хорошо известны в области RL и применялись к множеству задач, включая шахматы. Это означает, что по этим алгоритмам доступно множество информации и ресурсов, которые могут упростить их изучение и понимание.

# Ограничение, накладываемое на задачу

Шахматы — это сложная игра с огромным количеством возможных ходов и конфигураций. Это затрудняет для обучающегося алгоритма изучение оптимальных действий, которые следует предпринять в каждой ситуации. Более того, алгоритму недостаточно просто выбрать следующий ход, который представляется наиболее перспективным в текущей ситуации. Вместо этого алгоритм должен быть способен рассматривать несколько ходов вперед и предвидеть потенциальные последствия каждого хода, чтобы принять наилучшее долгосрочное решение.

Отсюда следует, что для в ходе обучения алгоритма обучения с подкреплением может потребоваться значительное количество времени и вычислительных ресурсов, что затрудняет быстрое тестирование и отладку реализованных алгоритмов.

В целях сокращения сложности предполагаемого обучения, в настоящей работе вводится существенное ограничение: алгоритмы будут учиться разыгрывать не всю партию, а только её окончание: эндшпили. Обученный таким образом теоретически можно постепенно дообучать на более ранних этапах игры.

Для этого на [просторах интернета](https://www.google.com/) был 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 строках.

Fen строка – запись состояния шахматной в нотации Форсайта-Эдвардса.

Запись FEN описывает позицию на шахматной доске в виде строки ASCII символов. Эта запись состоит из нескольких полей:

* Положение фигур со стороны белых. Позиция описывается цифрами и буквами по горизонталям сверху вниз начиная с восьмой горизонтали и заканчивая первой. Расположение фигур на горизонтали записывается слева направо, данные каждой горизонтали разделяются косой чертой /. Белые фигуры обозначаются заглавными буквами. K, Q, R, B, N, P — соответственно белые король, ферзь, ладья, слон, конь, пешка. k, q, r, b, n, p — соответственно чёрные король, ферзь, ладья, слон, конь, пешка. Обозначения фигур взяты из англоязычного варианта алгебраической нотации. Цифра задаёт количество пустых полей на горизонтали, счёт ведётся либо от левого края доски, либо после фигуры (8 означает пустую горизонталь).
* Активная сторона: w — следующий ход принадлежит белым, b — следующий ход чёрных.
* Возможность рокировки. k — в сторону королевского фланга (короткая), q — в сторону ферзевого фланга (длинная). Заглавными указываются белые. Невозможность рокировки обозначается «-». Возможность взятия пешки на проходе. Указывается проходимое поле, иначе «-».
* Счётчик полуходов. Число полуходов, прошедших с последнего хода пешки или взятия фигуры. Используется для определения применения правила 50 ходов.
* Номер хода. Любой позиции может быть присвоено любое неотрицательное значение (по умолчанию 1), счётчик увеличивается на 1 после каждого хода чёрных.

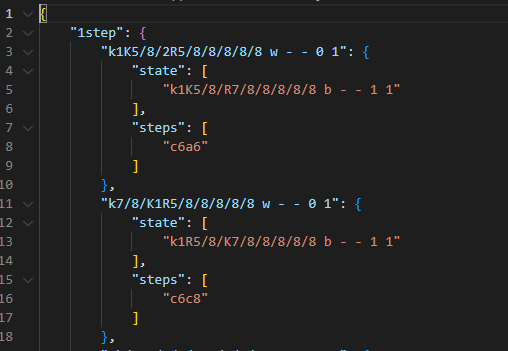
Для последующего процесса обучения найденный датасет подавался на вход шахматному движку с целью получить не достающие шаги до победы. Недостающие шаги необходимы для последующей оценки ходов, предлагаемых разрабатываемой системы. На выходе сформирован файл формата json (распространенный формат хранения и передачи данных). Пример полученного файла ниже (Рисунок 3):

Рисунок . Пример датасета в json формате

# Разработка и реализации среды

В качестве среды для разрабатываемой системы может послужить любой шахматный движок, позволяющий разыгрывать шахматную партию через специальный интерфейс. Такие движки могут хранить и обновлять состояние доски, оценивать сделанный игроком ход, а также отвечать на него собственным. 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).

# Разработка и реализация агентов

*Рисунок SEQ Рисунок \\* ARABIC 1. Пример JSON файла.*

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

В ходе настоящей курсовой работы была разработана и реализована иерархия классов, реализующих интерфейс взаимодействия с шахматным движком, который заключается в выдаче следующего хода в соответствии запрограммированного алгоритма обучения с подкреплением.

Ниже описан каждый из реализованных агентов.

## Random agent

Random agent – базовый игрок, который выбирает следующий ход случайным образом. Он играет роль baseline’а, с котором будут сравниваться разработанные алгоритмы обучения с подкреплением

Алгоритм работы:

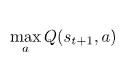
1. Игровая среда возвращает доступные позиции для возможных ходов
2. Агент выбирает случайный ход из списка
3. Возвращает среде выбранный ход

## 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 работы

Принципиальная схема работы алгоритма обучения представлена ниже (Рисунок 4)

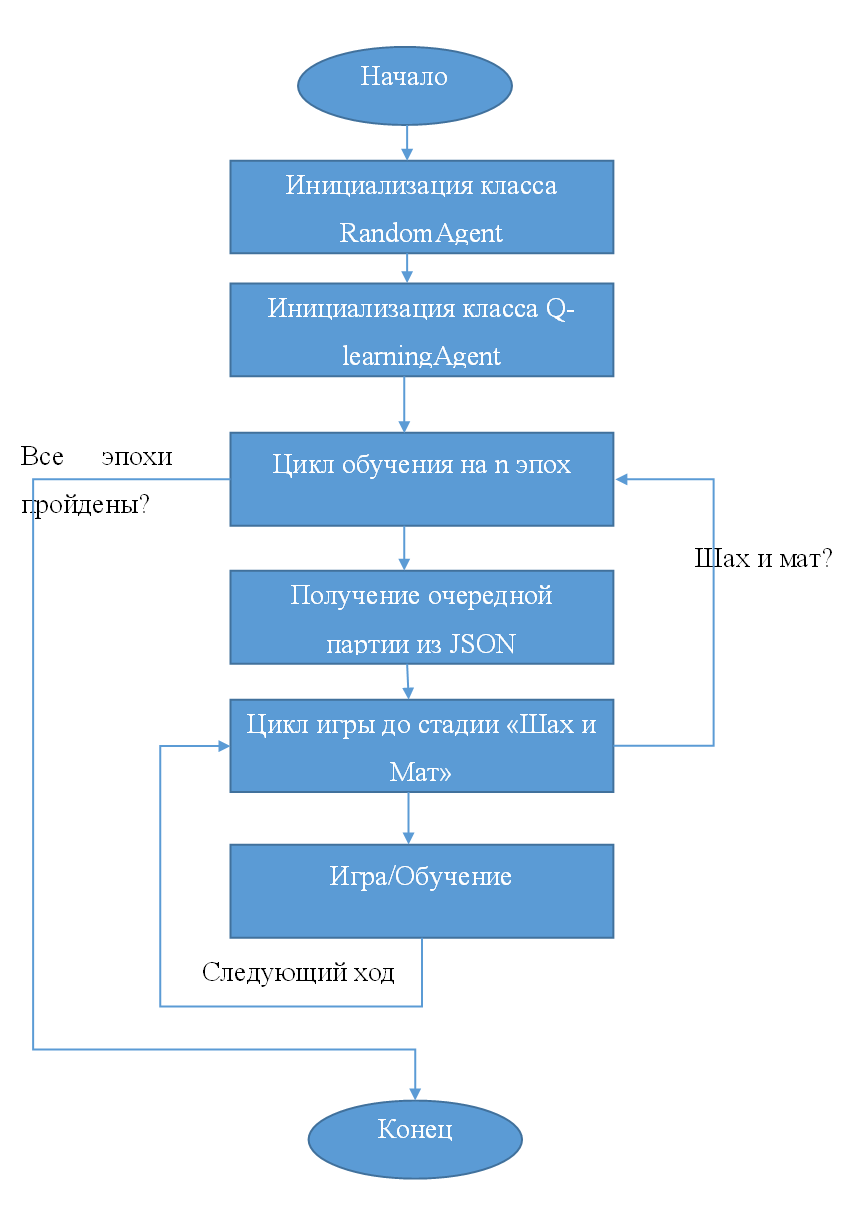


Рисунок . Принципиальная схема работы алгоритма Q-learning

### Результаты применения и обучений Q-Agent

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: алгоритм Q-learning 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 занимаемой памяти, рекомендуется применять либо другие виды обучения, либо вводить дополнительные ограничения

## Deep Policy Agent

Deep Policy Net – алгоритм, который оценивает градиент ожидаемого вознаграждения по отношению к параметрам политики и использует этот градиент для обновления политики в направлении, которое увеличивает ожидаемое вознаграждение. Это простой и эффективный метод градиента политик, который можно применять к широкому кругу проблем RL.

В качестве политики обычно используются нейронную сеть.

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.

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 –вектор наград за выбранный промежуток;
* Actions – вектор действий за выбранный промежуток;
* Sum – оператор суммирования;
* \* - оператор матричного умножения

Для реализации Deep policy net предложена следующая архитектура (Рисунок 5). Более подробное описание прямого прохождения, тестирования и отладки представлено в главе Тестирование и отладка архитектуры (Тюнинг модели)

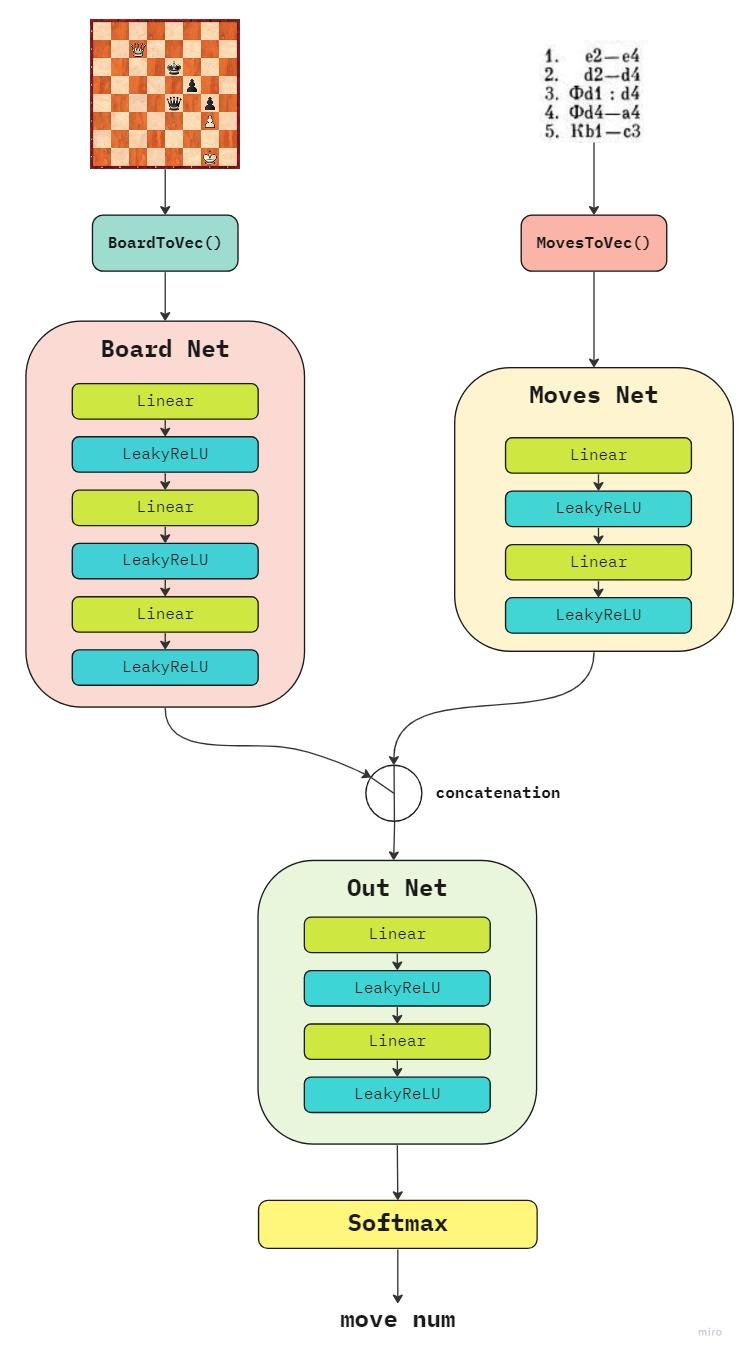


Рисунок . Архитектура разработанной нейросети

# Предобработка

В связи с использованием FEN строки, как основной 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 некоторый 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.

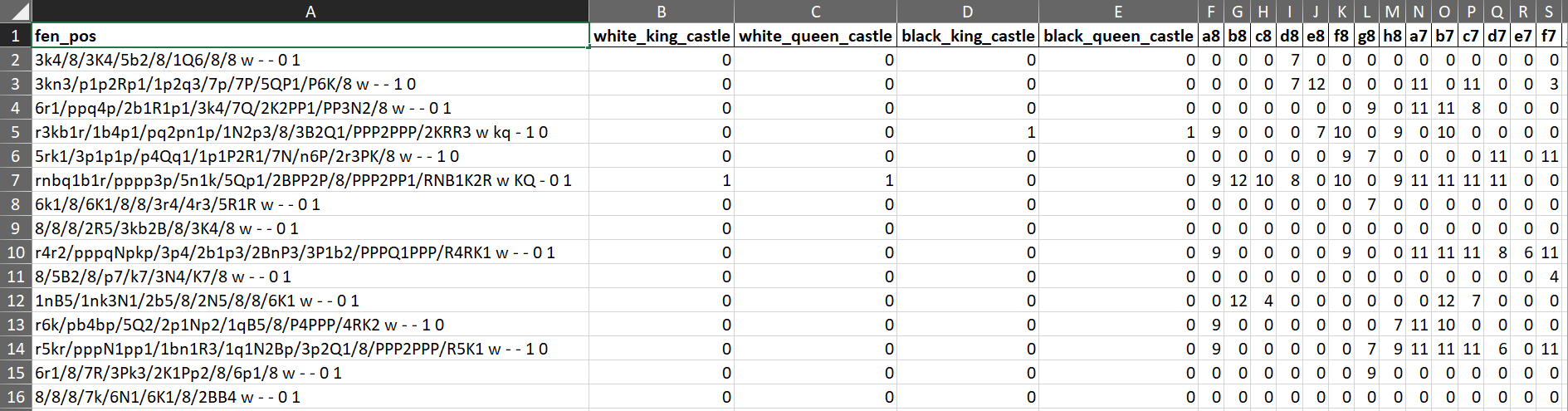
Таким образом из описания выше получен вектор следующего вида (Рисунок 6. Sample векторов полученных из FEN строк):

Рисунок . Sample векторов полученных из FEN строк

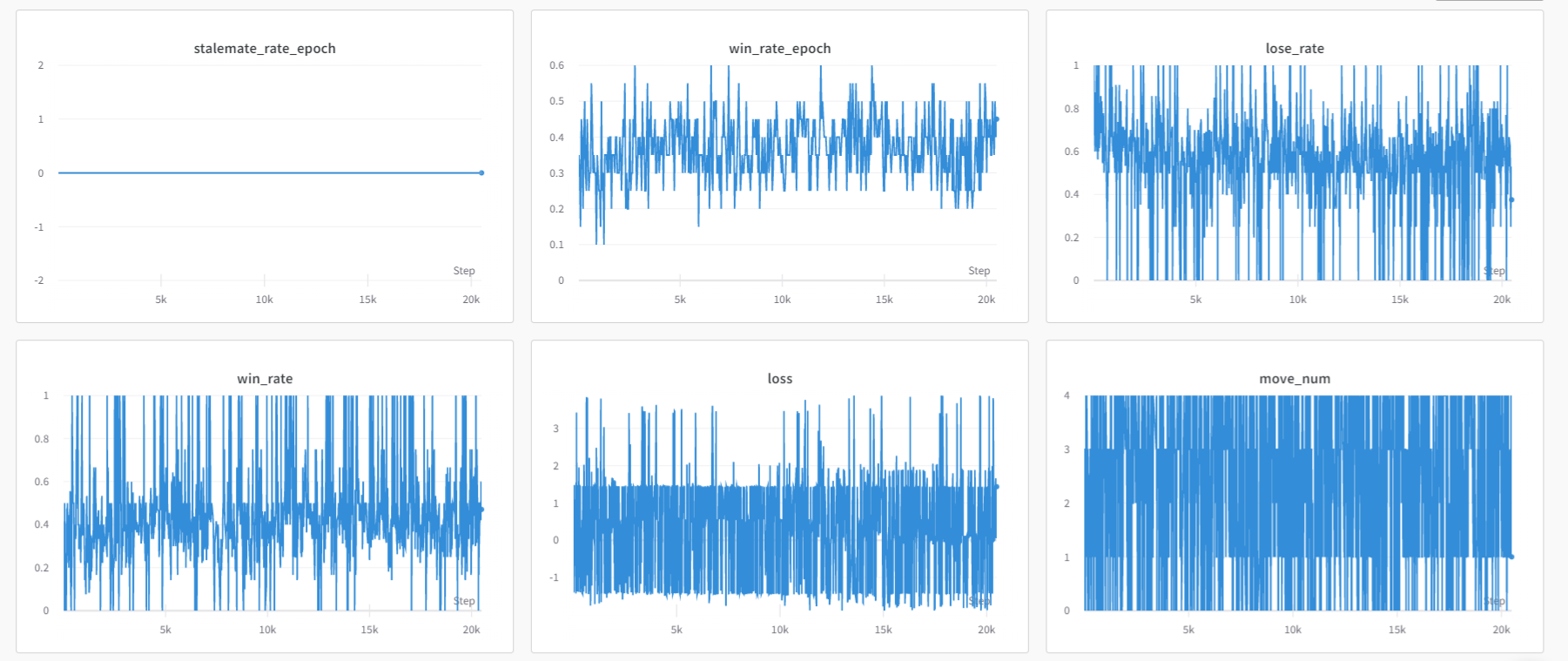
# Тестирование и отладка архитектуры (Тюнинг модели)

Так как данная работа носит исследовательский характер, и основная тема предмета связана с математическими моделями и нейросетями, основной акцент практической части связан на выборе архитектуры нейросети и подборе гиперпараметров.

Для этого при помощи интеграции сервиса wandb в процесс обучения построен dashboard и для всего процесса обучения настроено логирование параметров. Примеры dashboardов можно увидеть ниже ()



Рисунок . Dashboard в свернутом виде

Основные параметры располагаются во вкладке Charts

Основой код инициализации отслеживания

Рисунок 9. Инициализация отслеживания обучения

Рисунок 8. Основные фиксируемые результат

Входе исследования совершено более 150 запусков этапа обучения для различных гиперпараметров.

*Эксперименты со структурой нейросети.*

Входе различных запусков проводились различные манипуляции со структурой нейросети, изменялось количество слоёв, как в большую сторону, так и меньше. Значительных изменений в качестве работы модели получено не было в следствии чего структура выбранной модели представлена на Рисунок 5. Архитектура разработанной нейросети

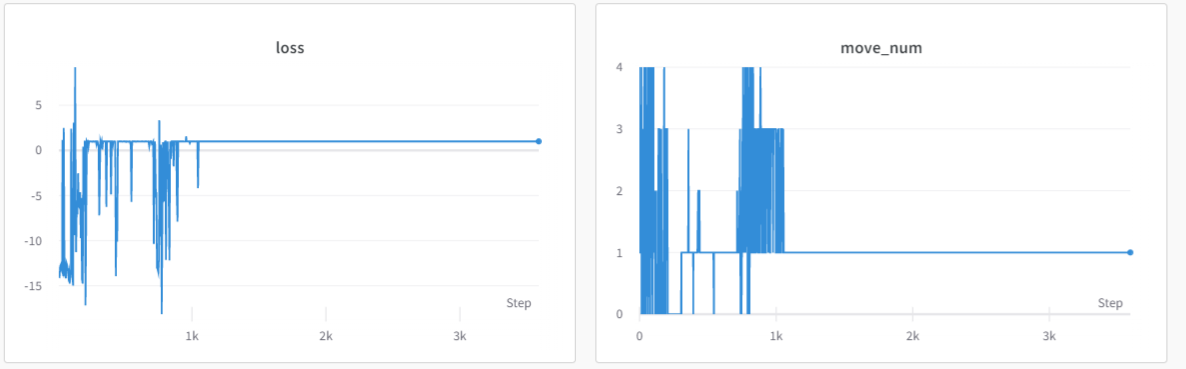
*Функции активации и Dropout.* На ранних этапах тестовых обучений нейросети обучение скатывалось к выбору одного и того же значения в независимости от состояния, но от запуска к запуску. Во избежание подобной ситуации применялись различные функции активации на внутренних слоях, а также внедрялся слой Dropout. Хотя проблема решилась частично подобным подходом за время испытаний получено значительное количество информации о работе модели. Пример зацикливания на одном значение представлено ниже(Рисунок 8)

Рисунок . Ошибочное выучивание одной позиции

*Гиперпараметры.* Основное влияние на качество обучения оказали запуски с различными гиперпараметры. Описанная выше проблема имела решение путем уменьшения величины learning rate, таким образом, получается, что был найден локальный минимум и потому выбиралось одно и тоже значение.

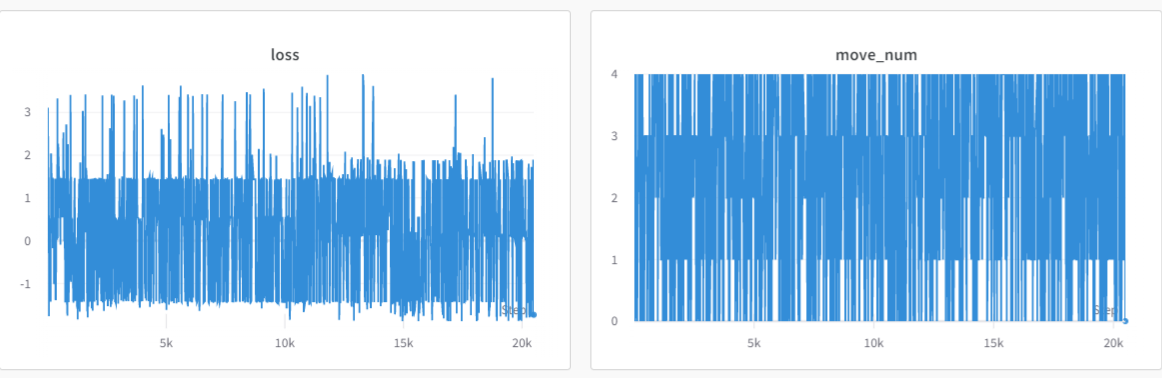
Правильное обучение представлено на следующем рисунке (Рисунок 9)

Рисунок 9. Нормальный процесс обучения

# Прямой ход модели

Рисунок 10. Схематическое изображение прямого хода разработанной модели

Прямой ход разработанной архитектуры Deep Policy Net можно описать следующим образом:

* + - 1. Fen-строка, описывающая состояние доски, векторизируется с помощью алгоритма BoardTovec();
      2. Список из пяти строковых представлений возможных ходов для данной позиции векторизируется с помощью MovesToVec();
      3. Оба полученных вектора подаются на вход соответствующих компонентам модели Deep Policy Net:
         1. векторное представление fen-строки на BoardNet;
         2. векторное представления списка возможных ходов на MovesNet.
      4. Названные компоненты модели обрабатывает векторы таким образом, чтобы найти более компактную, деколлелированную форму, после выход двух компонентов конкатенируются;
      5. Сконкатенированный вектор подается на вход последнего компонента модели, OutputNet, с выходным softmax слоем. Выход этого слоя можно интерпретировать как распределение вероятностей того, что наилучшего ход с конкретным номером является наилучшим.
      6. Модель выбирает номер хода, который соответствует максимальной вероятности.

# Игра агентов между собой

Для оценки обученной модели разработан простой алгоритм, в котором агент, обученный нами, играет с агентом выбирающем ходы случайно. Результаты 5 замеров на 25 партиях приведены в таблице ниже:

|  |  |  |
| --- | --- | --- |
| Номер партии | Количество побед DQN | Количество побед Random |
| 1 | 23 | 3 |
| 2 | 24 | 2 |
| 3 | 23 | 4 |
| 4 | 22 | 2 |
| 5 | 25 | 1 |

Таким образом видно, что обученный алгоритм решает шахматные задачи многократно лучше случайного агента.

# 

# Выводы

В ходе работы изучены алгоритмы обучения с подкреплением: DQN и Q-learning. Реализована система, обучающая эти алгоритмы, а также их последующую интеграцию в среду.

Разаработа среда для игры в шахматы на основе шахматного движка Stockfish, среда подготовлена для интеграции в алгоритмы RE.

Q-learning показал себя слабо применимым для работы с процессами в которых присутствует значительное количество различных состояний в силу особенностей алгоритма.

Изучены различные подходы к представлению шахматной партии: токенайз, кодирование в вектор признаков, преобразование в матрицу шахматной доски

Исследованы различные вариации тюнинга модели. Опробованы различные подходы к изменению гиперпараметров, методы нормализации данных, исследованы различные функции потерь и архитектуры сетей

Проведено сравнение обученной нейросети со случайным агентом