Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМА УПРАВЛЕНИЯ
КАФЕДРА Автоматизированные системы ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Обучение CNN с использованием глубокого алгоритма q-learning для игры «Connect4»

Студент ___ИУ5И-22М____ _____ ___**Ф. М. Шаххуд**_
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ __**Ю. Е. Гапанюк** _
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ___ИУ5___
(Индекс)
_____ _В. М. Черненький_
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение НИР по обработке и анализу данных

по теме _обучение CNN с использованием глубокого алгоритма q-learning для __игры «Connect 4»_____

_____

Студент группы ___ИУ5И-32М__
_____Шаххуд Фарах Муршед_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

_____исследовательская_____

Источник тематики (кафедра, предприятие, НИР) _____
График выполнения НИР: 25% к _3_ нед., 50% к _9_ нед., 75% к 12_ нед., 100% к _15_ нед.

Техническое задание _____ С помощью обучения с подкреплением задача состоит в том, чтобы обучить модель играть с человеком. Обучение проводится с использованием алгоритма q-learning. В результате получается обученный агент, способный нормально играть, как человек._____
Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на _____ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____
_____

Дата выдачи задания « _6_ » __юнья___ 2020_ г.

Руководитель НИР _____ ___ **Ю. Е. Гапанюк _**
(Подпись, дата) (И.О.Фамилия)

Студент _____ ___ __**Ф. М. Шаххуд__**
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Table of Contents

Цель работы

Обучение модели нейронной сети играть в четыре связанных игры в зависимости от алгоритма глубокого обучения Q-алгоритмов подкрепления обучения.

Введение в игру

Игра содержит доску с 7 колонками и 6 рядами. Каждый игрок в свою очередь выбирает один из столбцов и бросает один цветной диск сверху, который падает в самое низкое пустое место. Первый из четырех дисков одинакового цвета по горизонтали, вертикали или диагонали побеждает. Эта игра имеет более 10^{14} состояний.

Введение в алгоритм

Цель алгоритма состоит в том, чтобы изучить функцию Q, которая является функцией, которая отображает пары «состояние-действие» в значение, представляющее ожидаемое среднее значение будущих наград, которые каждый игрок получит после выполнения этого действия в определенном состоянии. После того, как мы изучим функцию Q, мы выберем играть действие, которое максимизирует это значение.
В нашем случае область функции Q содержит более $7 * 10^{14}$ (7 возможных действий для более чем 10^{14} состояний) и отображает их в число от -1 до 1 (диапазон вознаграждений). Для представления этой функции мы будем использовать CNN.

Модель нейронной сети

На входе CNN будет матрица, которая является игровой доской. И мы добавляем одну строку размером 7, представляющую действие (в числе от 1 до 7, которые представляют, в какой столбец будет добавлен диск). Итак, в итоге мы имеем матрицу 7 * 7. У нас есть три символа для представления игрового поля: 1, -1 и 0, которые представляют диск игрока, диск противника или пустое место соответственно. Для удобства отображения доски я изменил этот символ на «*», «-» или «0», но это ничего не меняет.

Чтобы иметь дело с входной матрицей, у нас будет сверточная нейронная сеть из семи слоев плюс два полностью связанных слоя в конце. Размер ядра свертки сначала выбирается равным 5, а затем он изменяется на 4. Что касается функции активации, то выбирается LeakyRelu, выход которого отличается от нуля в области <0. И это удобно для того, чтобы ни один нейтрон не научился иметь ноль в качестве выхода.

Итак, наша модель выглядит следующим образом:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 7, 7, 20)	520
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 20)	0
conv2d_2 (Conv2D)	(None, 7, 7, 20)	6420
leaky_re_lu_2 (LeakyReLU)	(None, 7, 7, 20)	0
conv2d_3 (Conv2D)	(None, 7, 7, 20)	6420
leaky_re_lu_3 (LeakyReLU)	(None, 7, 7, 20)	0
conv2d_4 (Conv2D)	(None, 7, 7, 30)	9630
leaky_re_lu_4 (LeakyReLU)	(None, 7, 7, 30)	0
conv2d_5 (Conv2D)	(None, 7, 7, 30)	14430
leaky_re_lu_5 (LeakyReLU)	(None, 7, 7, 30)	0
conv2d_6 (Conv2D)	(None, 7, 7, 30)	14430
leaky_re_lu_6 (LeakyReLU)	(None, 7, 7, 30)	0
conv2d_7 (Conv2D)	(None, 7, 7, 30)	14430
leaky_re_lu_7 (LeakyReLU)	(None, 7, 7, 30)	0
flatten_1 (Flatten)	(None, 1470)	0
dense_1 (Dense)	(None, 49)	72079
leaky_re_lu_8 (LeakyReLU)	(None, 49)	0
dense_2 (Dense)	(None, 7)	350
leaky_re_lu_9 (LeakyReLU)	(None, 7)	0
dense_3 (Dense)	(None, 7)	56
leaky_re_lu_10 (LeakyReLU)	(None, 7)	0
dense_4 (Dense)	(None, 1)	8

У нас две из этой модели обучены представлять агентов, которые играют в первую очередь, и агентов, которые играют вторую.

Представление игры

Чтобы представить нашу игру, мы должны определить ее марковский процесс.

Марковские процессы являются естественными стохастическими аналогами детерминированных процессов, описываемых дифференциальными и разностными уравнениями.

Как мы уже говорили, у нас есть два агента (один представляет первого игрока, а другой - второго). Это потому, что каждая из этих двух моделей имеет пространство состояний, отличное от другого.

Состояние пространство

Пространство состояний - это все состояния, которые видит каждый игрок. Для первого плеера это все платы с четным количеством дисков, а для второго плеера это все платы с нечетным количеством дисков.

Пространство действия

Пространством действия будут номера 1–7 для каждой колонки, в которую может играть игрок.

Награда

Награда будет 1 за победу, -1 за проигрыш, 0.5 за ничью и 0 в противном случае.

Объясняя алгоритм Q-обучения

Q-learning - это алгоритм обучения, не поддерживающий политику, который стремится найти наилучшее действие, которое необходимо предпринять, учитывая текущее состояние. Это считается вне политики, потому что функция q-learning учится на действиях, которые находятся за пределами текущей политики. q-learning стремится изучить политику, которая максимизирует общее вознаграждение.

Когда выполняется q-learning, мы создаем так называемую q-таблицу или матрицу, которая соответствует форме [состояние, действие], и мы инициализируем наши значения равными нулю. Затем мы обновляем и сохраняем наши q-значения после эпизода. Эта q-таблица становится справочной таблицей для нашего агента, чтобы выбрать лучшее действие на основе q-значения.

Агент взаимодействует с окружающей средой одним из двух способов. Во-первых, использовать q-таблицу в качестве ссылки и просмотреть все возможные действия для данного состояния. Затем агент выбирает действие на основе максимального значения этих действий. Это известно как использование, так как мы используем информацию, которая нам доступна, для принятия решения.

Второй способ действовать - действовать случайным образом. Это называется исследованием. Вместо выбора действий, основанных на максимальной будущей награде, мы выбираем случайное действие.

Действовать случайным образом важно, потому что это позволяет агенту исследовать и открывать новые состояния, которые иначе не могут быть выбраны в процессе эксплуатации.

Балансирование исследования / эксплуатации выполняется с использованием epsilon (ε) и установки значения частоты, с которой мы хотим исследовать и использовать эксплойт.

Алгоритм объясняется с помощью трех основных шагов:

1. Агент запускается в состоянии (s1), выполняет действие (a1) и получает вознаграждение (r1).
2. Агент выбирает действие, ссылаясь на Q-таблицу с наибольшим значением (max) ИЛИ случайным образом (epsilon, ε)
3. Обновить q-значения

Правило обновления q-learning определяется следующим образом:
Q [состояние, действие] = Q [состояние, действие] + lr * (награда + гамма * np.max (Q [new_state,:]) - Q [состояние, действие])
где:

learning rate: lr или скорость обучения могут быть просто определены как то, насколько мы принимаем новое значение против старого значения. Выше мы берем разницу между новым и старым, а затем умножаем это значение на скорость обучения. Это значение затем добавляется к нашему предыдущему q-значению, что существенно смещает его в направлении нашего последнего обновления.

Гамма: гамма или γ - коэффициент дисконтирования. Он используется для балансирования немедленной и будущей награды. Из приведенного выше правила обновления мы можем видеть, что мы применяем скидку к будущему вознаграждению. Обычно это значение может находиться в диапазоне от 0,8 до 0,99.

Награда: награда - это ценность, полученная после выполнения определенного действия в данном состоянии.

Max: np.max () использует библиотеку numpy и берет максимум будущей награды и применяет ее к награде за текущее состояние. Это влияет на текущее действие возможной будущей наградой.

Реализация программы

Класс Model

Этот класс определяет модель для игры. Класс модели Connected-4 унаследован от него.

```
class Model:

    def __init__(self,tag):
```

```
        self.tag = tag
        self.epsilon = 0.1
        self.alpha = 0.5
        self.gamma = 1
        self.model = self.load_model()
```

Функция Load_model

Для загрузки агента из файла. Если файл не существует, он создает новую модель нейронной сети, вызывая create_model

```
def load_model(self):
        if self.tag == 1:
           tag = '_first'
        else:
            tag = '_second'''
        s = 'model_values' + tag + '.h5'
        model_file = Path(s)

        if model_file.is_file():
            print('load model:',s)
            model = Km.load_model(s)
        else:
            model = self.create_model()
        return model
```

Функция Calc_value

Он извлекает значение перемещения из модели, прогнозируя его.

```
def calc_value(self, state, move):
        tensor = self.state_to_tensor(state, move)
        value = self.model.predict(tensor)
        # K.backend.clear_session()
        return value
```

Функция Calc_target

После получения значения прогноза от модели для предыдущего хода. Мы видим прогноз для каждого из доступных ходов. Предсказанное значение модели на самом деле представляет собой оценку вознаграждения в сети в зависимости от этого конкретного шага. Итак, теперь мы используем уравнение функции q-learning для вычисления цели состояния, в котором мы находимся:

```
target = np.array(v_s + self.alpha * (reward + v_s_tag - v_s))
```

Где: `v_x_tag is v_s_tag = self.gamma * np.max(v)`

```
def calc_target(self, prev_state, prev_move, state, ava_moves, reward):
```

```
        v_s = self.calc_value(prev_state, prev_move)

        v = []
        for move in ava_moves:
            v.append(self.calc_value(state, move))

        if reward == 0:
            if len(v) > 0:
                v_s_tag = self.gamma * np.max(v)
            else:
                v_s_tag = 0
            target = np.array(v_s + self.alpha * (reward + v_s_tag -
v_s))
        else:
            target = reward

        return target
```

Функция train_model

Вычисляет цель (вызывая Calc_target) состояния с использованием model.fit

```
def train_model(self, prev_state, prev_move, target, epochs):

        tensor = self.state_to_tensor(prev_state, prev_move)

        if target is not None:

            self.model.fit(tensor, target, epochs=epochs, verbose=0)
            K.backend.clear_session()
```

Функция Save_model

Сохраняет модель после обучения с помощью функции Keras в виде файла h5.

```
def save_model(self):
        if self.tag == 1:
            tag = '_first'
        else:
            tag = '_second'
        s = 'model_values' + tag + '.h5'

        try:
            os.remove(s)
        except:
            pass
        print('model save to: ',s)
        self.model.save(s)
```

Функция Learn_batch

Это процесс обучения модели с использованием значений, которые хранятся в списке, называемом памятью. Он соответствует целевому значению для штатов, обучает и оценивает модель. Затем он рассчитывает потери. Затем он сохраняет модель в файл.

```
def learn_batch(self, memory):
        print('start learning player', self.tag)
        print('data length:', len(memory))
        # build x_train
        ind = 0
        x_train = np.zeros((len(memory), 7, 7, 1))
        # x_train = np.zeros((len(memory), 2, 9))
        for v in memory:
            [prev_state, prev_move, _, _, _] = v
            sample = self.state_to_tensor(prev_state, prev_move)
            x_train[ind, :, :, :] = sample
            ind += 1

        # train with planning
        # for i in range(5):
        loss = 20
        count = 0
        while loss > 0.02 and count<20:
            # tic()
            y_train = self.create_targets(memory)
            # toc()
            self.model.fit(x_train, y_train, epochs=5, batch_size=256,
verbose=1)
            loss = self.model.evaluate(x_train, y_train, batch_size=256,
verbose=0)[0]
            count += 1
            #print('planning number:', count, 'loss', loss)

        loss = self.model.evaluate(x_train, y_train, batch_size=256,
verbose=0)
        print('player:', self.tag,'loss: ', loss, 'loops: ', count)

        self.save_model()
```

Функция Create_targets

Он начинается с заданных нулевых значений цели для всех состояний, которые хранятся в памяти. Затем он вычисляет их значение на основе функции q-learning и сохраняет их в памяти.

```
def create_targets(self, memory):
```

```
        y_train_ = np.zeros((len(memory), 1))
        count_ = 0
        for v_ in memory:
            [prev_state_, prev_move_, state_, ava_moves_, reward_] = v_
            target = self.calc_target(prev_state_, prev_move_, state_,
ava_moves_, reward_)
            y_train_[count_, :] = target
            count_ += 1
        return y_train_
```

Класс Agent

Этот класс определяет, как агент игры видит среду и как она взаимодействует с ней. Переменная tag имеет значение 1 или 2, относящееся к первому агенту игрока и второму агенту игрока. Исследовательский фактор - это фактор, который определяет вероятность выполнения случайного движения или выбора лучшего хода на основе расчета модели (для изучения условий для лучших ходов). Переменные состояния начинаются как матрица размером 6 на 7 np.zeros. И переменные памяти инициализируются как пустой список.

```
class Agent:

    def __init__(self, tag,exploration_factor=1):
        self.tag = tag
        self.exp_factor = exploration_factor
        self.prev_state = np.zeros((6, 7))
        self.prev_move = -1
        self.state = None
        self.move = None
        self.print_value = False
        self.model = Model(self.tag)
        self.memory = []
        self.count_memory = 0
        self.winner_flag = False
```

Функция Choose_move

Сначала он загружает в память значения предыдущего состояния, предыдущего хода, текущего состояния, доступных ходов для следующего состояния и вознаграждение за победителя, если оно есть. Если в игре есть победитель, он сбрасывает состояние и начинает обучение, вызывая learn_batch, если размер памяти достиг определенного числа.

Если победителя нет, он генерирует случайное значение между 0 и 1, а если оно ниже, чем коэффициент исследования, он выбирает лучший доступный ход. Если нет, он просто выбирает случайный ход из доступных.

```
def choose_move(self, state, winner, learn):

        self.load_to_memory(self.prev_state, self.prev_move, state,
self.ava_moves(state), self.reward(winner))

        if winner is not None:

            self.count_memory += 1

            self.prev_state = np.zeros((6, 7))
            self.prev_move = -1

            if learn is True and self.count_memory == 1000:
                self.count_memory = 0
                # Offline training
                self.model.learn_batch(self.memory)
                self.memory = []
                # Online training
                #self.learn(self.prev_state, self.prev_move, state,
self.ava_moves(state),  -1, self.reward(winner))
            return None

        p = random.uniform(0, 1)

        if p < self.exp_factor:
            idx = self.choose_optimal_move(state)
        else:
            ava_moves = self.ava_moves(state)
            idx = random.choice(ava_moves)

        self.prev_state = state
        self.prev_move = idx

        return idx
```

Функция Choose_optimal_move

Сначала он генерирует все доступные ходы состояния. Затем он вызывает функцию calc_value для получения значения ходов из модели. Затем он выбирает ход с лучшим значением.

```
def choose_optimal_move(self, state):
```

```
        ava_moves = self.ava_moves(state)
        v = -float('Inf')
        v_list = []
        idx = []
        for move in ava_moves:
            value = self.model.calc_value(state, move)
            v_list.append(round(float(value), 5))

            if value > v:
                v = value
                idx = [move]
            elif v == value:
                idx.append(move)

        idx = random.choice(idx)
        return idx
```

Функция Game_winner

Он создает квадрат размером 4 на 4 из игровой доскии вызывает функцию square_winner для каждого из них. Затем он печатает, кто является победителем или тот факт, что нет победителя.

```
def game_winner(self, state):
        winner = None
        for i in range(len(state[:,0])-3):
            for j in range(len(state[0, :])-3):
                winner = self.square_winner(state[i:i+4, j:j+4])
                if winner is not None:
                    # print('winner is:', self.winner)
                    break
            if winner is not None:
                # print('winner is:', self.winner)
                break

        if np.min(np.abs(state[0, :])) != 0:
            winner = 0
            # print('no winner')

        return winner
```

Функция Square_winner

Он суммирует значения квадратного массива в соответствии с его осью и его диагоналями, и если он находит значение 4 или -4, то он определяет, что есть победитель.

```
def square_winner(square):
        s = np.append([np.sum(square, axis=0), np.sum(square, axis=1).T],
                      [np.trace(square), np.flip(square,axis=1).trace()])
        if np.max(s) == 4:
            winner = 1
        elif np.min(s) == -4:
            winner = 2
        else:
            winner = None
        return winner
```

Функция Make_state_from_move

Он получает индекс хода, а затем находит самый низкий индекс столбца, который свободен, и помещает в него тег игрока.

```
def make_state_from_move(state, move, player):
        if move is None:
            return state

        state = np.array(state)
        if player == 1:
            tag = 1
        else:
            tag = -1

        if len(np.where(state[:, move] == 0)[0]) == 0:
            print(state)
        idy = np.where(state[:, move] == 0)[0][-1]
        state = np.array(state)
        state[idy, move] = tag

        return state
```

Функция Reward

Он присваивает значение вознаграждения в соответствии с агентом (либо 1, если он выигрывает, -1, если проигрывает, 0.5, если он ничейный, и 0, если победитель все еще не определен).

```
def reward(self, winner):

        if winner is self.tag:
            reward = 1
        elif winner is None:
            reward = 0
        elif winner == 0:
            reward = 0.5
        else:
```

```
            reward = -1
        return reward
```

Функция Learn

Он обучает модель не на основе значений памяти, но сразу же вычисляет цель и обучает модель на ней.

Загрузка в память добавляет значения в пустой список памяти, чтобы сформировать пакеты для обучения.

```
def learn(self, prev_state, prev_move, state, ava_moves, move, reward):

        if prev_move != -1:

            target = self.model.calc_target(prev_state, prev_move, state,
ava_moves, reward)
            # print(target)
            self.model.train_model(prev_state, prev_move, target, 1)
```

Функция Save_memory

Это функция, которая сохраняет значения в списке памяти в файл на случай, если они понадобятся позже.

```
def save_memory(self):
        is_file_ = True
        count = 1
        s = ''
        while is_file_:
            s = 'value_list_' + str(self.tag) + '_' + str(count) + '.pkl'
            if Path(s).is_file():
                is_file_ = True
                count = count + 1
            else:
                is_file_ = False

        with open(s, 'wb') as output:
            pickle.dump(self.memory, output)
```

Класс Game

Этот класс определяет игру в целом и то, как обмениваются ходы игроков. Сначала в качестве инициализации он берет агента и присваивает ему тег (персонаж игрока) и значение исследования.

```
class Game:

    def __init__(self, player1, player2, exp1=1, exp2=1, tag1=1, tag2=2):
```

```
        self.players = {1: player1(tag1, exploration_factor=exp1),
                        2: player2(tag2, exploration_factor=exp2)}

        self.state, self.winner, self.turn = self.init_game()
        self.memory = {}
```

Функция Play_game

Если победителя по-прежнему нет, вызывается функция play_move. Затем вызывает make_state from_move, чтобы изменить состояние игры. Затем вызовите game_winner, чтобы узнать, сделал ли этот ход любого игрока победителем. Затем вызывает next_player для обмена ходом.

```
def play_game(self, learn=False):

        move_count = 0

        while self.winner is None:
            move = self.play_move(learn)

            self.state = self.make_state_from_move(move)
            self.game_winner()
            if self.winner is not None:
              break
            self.next_player()
            move_count += 1

        self.play_move(learn)
        self.next_player()
        self.play_move(learn)
        self.next_player()

        return self.winner, move_count
```

Функция Play_move

Он определяет игрока, у которого есть ход, а затем вызывает Choose_move и возвращает ход, который был выбран.

```
def play_move(self, learn):
        player = self.players[self.turn]
        move=1
        if self.winner is None:
          move = player.choose_move(self.state, self.winner, learn)
        return move
```

Функция Play_multiple games

Это функция, которая позволяет играть в эпизоды игр и возвращать статистику сыгранных игр, показывая, сколько игр выиграл каждый агент.

```
def play_multiple_games(self, episodes, learn):
        statistics = {1: 0, 2: 0, 0: 0, 'move_count': 0}
        move_count_total = []
        for i in range(episodes):
            winner, move_count = self.play_game(learn)
            move_count_total.append(move_count)
            statistics[winner] = statistics[winner] + 1

            self.state, self.winner, self.turn = self.init_game()

        statistics['move_count'] = np.mean(move_count_total)
        return statistics
```

Класс ConnectFourModel

Модель, которая определила функцию создания модели, когда агент создается без сохраненного для него файла.

```
class ConnectFourModel(Model):

    def __init__(self, tag):
        super().__init__(tag)
        pass
```

Функция Create_model

Он определяет последовательность нейронных слоев с использованием последовательного класса. Он имеет 7 слоев с LeakyRelu в качестве функции активации, за которыми следуют два полностью связанных слоя. Использование Адама в качестве оптимизатора и функция потерь рассчитывается как среднеквадратическая ошибка.

```
def create_model(self):
        print('new model')

        model = Km.Sequential()
        model.add(Kl.Conv2D(20, (5, 5), padding='same', input_shape=(7,
7, 1)))
        model.add(Kl.LeakyReLU(alpha=0.3))
        model.add(Kl.Conv2D(20, (4, 4), padding='same'))
        model.add(Kl.LeakyReLU(alpha=0.3))
```

```
        model.add(Kl.Conv2D(20, (4, 4), padding='same'))
        model.add(Kl.LeakyReLU(alpha=0.3))
        model.add(Kl.Conv2D(30, (4, 4), padding='same'))
        model.add(Kl.LeakyReLU(alpha=0.3))
        model.add(Kl.Conv2D(30, (4, 4), padding='same'))
        model.add(Kl.LeakyReLU(alpha=0.3))
        model.add(Kl.Conv2D(30, (4, 4), padding='same'))
        model.add(Kl.LeakyReLU(alpha=0.3))
        model.add(Kl.Conv2D(30, (4, 4), padding='same'))
        model.add(Kl.LeakyReLU(alpha=0.3))

        model.add(Kl.Flatten(input_shape=(7, 7, 1)))
        model.add(Kl.Dense(49))
        model.add(Kl.LeakyReLU(alpha=0.3))
        model.add(Kl.Dense(7))
        model.add(Kl.LeakyReLU(alpha=0.3))
        model.add(Kl.Dense(7))
        model.add(Kl.LeakyReLU(alpha=0.3))

        model.add(Kl.Dense(1, activation='linear'))
        opt = optimizers.adam()
        model.compile(optimizer=opt, loss='mean_squared_error',
metrics=['accuracy'])

        model.summary()

        return model
```

Функция State_to_tensor

Эта функция добавляет выбранный ход (после определения его как 1 в массиве из 7 нулевых значений) к состоянию, чтобы сформировать вход нейронной сети (7 на 7) с 1 каналом и без глубины.

```
def state_to_tensor(self, state, move):

        vec = np.zeros((1, 7))
        if move != -1:
            vec[0, move] = 1
        tensor = np.append(vec, state, axis=0)
        tensor = tensor.reshape((1, 7, 7, 1))

        return tensor
```

Класс ConnectFour

Это класс, производный от игрового класса, который переопределяет некоторые его функции.

```
class ConnectFour(Game):

    def __init__(self, player1, player2, exp1=1, exp2=1, tag1=1, tag2=2):
        super().__init__(player1, player2, exp1, exp2, tag1, tag2)
```

Функция Init_game

Инициализирует игру, возвращая массив нулей размера (6 на 7).

```
def init_game(self):
        return np.zeros((6, 7)), None, 1
```

Функция Next_player

Изменяет переменную хода игры между 1 и 2.

```
def next_player(self):
        if self.turn == 1:
            self.turn = 2
        else:
            self.turn = 1
```

Функция Print_game

Он преобразует множество нулей и единиц и минусов в красивую форму, более удобную для пользователя.

```
def print_game(self):

      print('  --------------')
      charar = np.chararray(state.shape,unicode=True)
      for i in range(state.shape[0]):
        for j in range(state.shape[1]):
          if state[i][j]==1: charar[i][j]='*'
          elif state[i][j]==-1: charar[i][j]='-'
          else: charar[i][j]='0'
      print(charar)
      print('  --------------')
```

Класс HumanPlayer

Этот класс определяет человека как агента игр, поэтому, когда игрок хочет играть в игры, он должен иметь возможность принять его участие и нормально играть в игру. Это происходит от класса агента.
Он инициализирует агента, давая ему тег и устанавливая фактор исследования (который здесь не имеет значения).

```
class HumanPlayer:

    def __init__(self, tag, exploration_factor=1):
        self.print_value = False
        self.exp_factor = exploration_factor
        self.tag = tag
```

Функция Choose_move

Эта функция готова принять ввод от пользователя, имеющего индекс, где он хочет сыграть свой следующий ход. И это печатает игры, чтобы человек мог легко выбирать.

```
def choose_move(state, winner, learn):
        if winner is not None:
          return 1
        print('  --------------')
        charar = np.chararray(state.shape,unicode=True)
        for i in range(state.shape[0]):
          for j in range(state.shape[1]):
            if state[i][j]==1: charar[i][j]='*'
            elif state[i][j]==-1: charar[i][j]='-'
            else: charar[i][j]='0'
        print(charar)
        print('  --------------')
        idx = int(input('Choose move number: ')) - 1
        return idx
```

Класс ConnectFourAgent

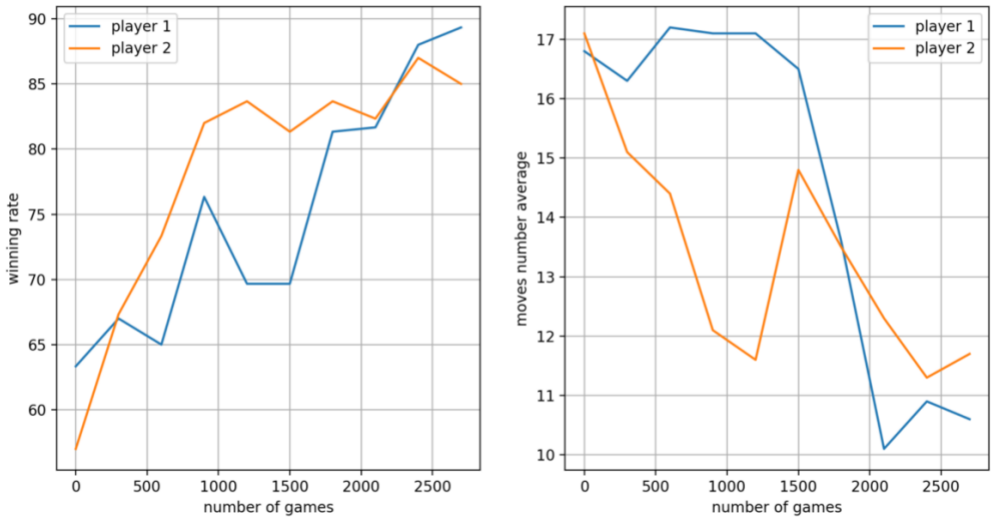
Он также является производным от класса агента и только инициализирует все значения.

```
class ConnectFourAgent(Agent):

    def __init__(self, tag, exploration_factor=1):

        super().__init__(tag, exploration_factor)
        self.model = ConnectFourModel(tag)
        self.prev_state = np.zeros((6, 7))
```

Модельное обучение

Определяя игру как ConnectFour, мы создаем объект класса ConnectFour и определяем игрока, передавая его атрибуты. Для обучения модели мы собираемся создать два экземпляра ConnectFourAgent с коэффициентом

исследования (0,7, 0,7), которые позволяют им немного изучить за пределами определенной матрицы. Создание агентов вызовет create_model из ConnectFourModel.
Затем мы присвоили статистику возвращаемой функции play_multiple_games. Это запустит количество игр (которое здесь определено как 6000 игр), и каждый раз оно будет хранить значения победителей состояний и перемещаться в список памяти. Когда память достигнет определенного размера, если переменная обучения установлена в значение true, будет вызвана функция learn_batch. Это обучит модель с использованием значений памяти и вычислит цель, как было описано. После окончания каждой партии модель будет сохранена в файл.
Когда мы снова вызываем функцию play_game и он видит, что существует файл с заданным именем, он не будет создавать модель с нуля. Он загрузит модель из файла.
Таким образом, вызывая play_multiple_games и устанавливая переменную обучения в значение false. Мы действительно можем играть с обученной моделью.
На приведенном ниже графике показан коэффициент выигрыша агента Q по отношению к случайному игроку в начале процесса обучения. Он также показывает среднее количество ходов, которое каждая игра прошла в среднем за 300 игр.



Результаты

1. Два агента обучены играть друг против друга. Мы видим, что первый агент узнал, что игра в середине - лучший ход. Для этого второй агент обучен всегда играть против этого конкретного хода. Вот почему фактор разведки

так важен, потому что он дает модели возможность узнать больше о тех состояний, которые она не посещала ранее.

2. Человек-агент, если играть оптимально, всегда может победить модель агента. Если у нас есть файлы записанных игр между оптимальными игроками, будет очень полезно обучить агентов оптимально играть против неожиданных ходов.

Пример играть с агентом

```
load model: model_values_first.h5
  --------------
[['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '*' '0' '0' '0']]
  --------------
Choose move number: 3
  --------------
[['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '*' '0' '0' '0']
 ['0' '0' '-' '*' '0' '0' '0']]
  --------------
Choose move number: 2
  --------------
[['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '*' '0' '0' '0']
 ['0' '0' '0' '*' '0' '0' '0']
 ['0' '-' '-' '*' '0' '0' '0']]
  --------------
Choose move number: 4
  --------------
[['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '-' '0' '0' '0']
 ['0' '0' '0' '*' '0' '0' '0']
 ['0' '0' '0' '*' '0' '0' '0']
 ['0' '-' '-' '*' '*' '0' '0']]
  --------------
Choose move number: 3
  --------------
[['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '-' '0' '0' '0']
 ['0' '0' '0' '*' '0' '0' '0']
 ['0' '0' '-' '*' '0' '0' '0']
 ['0' '-' '-' '*' '*' '*' '0']]
  --------------
Choose move number: 7
  --------------
[['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '-' '0' '0' '0']
 ['0' '0' '0' '*' '0' '0' '0']
 ['0' '0' '-' '*' '0' '*' '0']
 ['0' '-' '-' '*' '*' '*' '-']]
  --------------
Choose move number: 3
  --------------
[['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '*' '-' '0' '0' '0']
 ['0' '0' '-' '*' '0' '0' '0']
 ['0' '0' '-' '*' '0' '*' '0']
 ['0' '-' '-' '*' '*' '*' '-']]
  --------------
Choose move number: 2
  --------------
[['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '*' '-' '0' '0' '0']
 ['0' '*' '-' '*' '0' '0' '0']
 ['0' '-' '-' '*' '0' '*' '0']
 ['0' '-' '-' '*' '*' '*' '-']]
  --------------
Choose move number: 1
winner is: 2
```

Ссылка для Google Colab Notebook

https://github.com/farahshahhoud/Qlearningconnect4/blob/main/4InRow.ipynb

Список литературы

1- Simple Reinforcement Learning: Q-learning, Andre Violante, Mar 18, 2019, https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56
2- Deep Reinforcement Learning and Monte Carlo Tree Search With Connect, Gilad Wisney, Apr 25, 2019, https://towardsdatascience.com/deep-reinforcement-learning-and-monte-carlo-tree-search-with-connect-4-ba22a4713e7a
3- Reinforcement Learning in Tic-Tac-Toe Game and Its Similar Variations, Peng Ding and Tao Mao, Thayer School of Engineering at Dartmouth College, https://www.cs.dartmouth.edu/~lorenzo/teaching/cs134/Archive/Spring2009/final/PengTao/final_report.pdf
4- Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, The MIT Press, second edition, 2018-2020.