ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**(ФИНАНСОВЫЙ УНИВЕРСИТЕТ)**

Департамент анализа данных

и машинного обучения

***Дисциплина: «Технологии анализа данных и машинного обучения»***

*Направление подготовки: «Прикладная математика и информатика»*

*Профиль: «Анализ данных и принятие решений в экономике и финансах»*

*Факультет информационных технологий и анализа больших данных*

*Форма обучения очная*

*Учебный 2022/2023 год, 7 семестр*

**Курсовая работа на тему:**

«Разработка и обучение виртуального агента для игры в “Города” с применением обучения с подкреплением»

*Выполнил(а):*

студент(ка) группы ПИ20-1

Шалимов К.А.

*Научный руководитель:*

ассистент Блохин Н.В.

**Москва 2023**

Содержание

[Введение 3](#_Toc132027331)

[Глава 1. Обучение с подкреплением](#_Toc132027332) 5

[1.1 Общая характеристика обучения с подкреплением 5](#_Toc132027333)

[1.2 Элементы обучения с подкреплением 5](#_Toc132027334)

[1.3 Выводы 6](#_Toc132027335)

[Глава 2. Используемая модель 7](#_Toc132027336)

[2.1 Агент 7](#_Toc132027337)

[2.2 Актор и критик 7](#_Toc132027338)

[Глава 3. Разработка среды и политика вознаграждения 9](#_Toc132027336)

[3.1 Входные данные и предобработка 9](#_Toc132027337)

[3.2 Реализация среды 10](#_Toc132027338)

[3.3 Политика начисления награды 12](#_Toc132027338)

[3.4 Процесс обучения и результаты 13](#_Toc132027339)

[Заключение 18](#_Toc132027340)

[Список использованных источников 19](#_Toc132027341)

[Приложение А. Программный код 21](#_Toc132027342)

Введение

Некоторые задачи могут иметь неочевидные оптимальные решения или же неочевидные проигрышные стратегии. Например, в шахматах на первый взгляд безопасный ход может привести к проигрышу всей партии. В таких случаях, если мы имеем единственный заранее составленный алгоритм действий, мы будем из разу в раз повторять одну и ту же ошибку. Если только мы не запомним, что это была за комбинация фигур и какие наши ходы привели к мату. В таком случае мы уже будем понимать, что этот ход является рискованным и скорее всего предпочтём другой вариант.

Так, в приведённом примере мы не можем заранее предугадать признаки подобной ситуации (особенно если мат должен случиться через один или два хода) и имеем большую вариативность дальнейшего изменения ситуации [1], но также имеем и строго заданные и заранее известные критерии победы, что не даёт нам достаточно надёжно оценивать ходы во время игры, но позволяет оценить результативность всей их совокупности по их результату. Для воплощения подобных задач применяется машинное обучение с подкреплением или же Reinforcement learning.

Одной из задач с неочевидными стратегиями достижения победы, является игра в города, за счет наличия большой вариативности выбора путей совершения хода, каждый из которых может повлечь за собой различные наборы городов, доступных для следующего хода. Так же для принятия решения нужно представлять ещё и возможности, открываемые для оппонента (ты можешь использовать все слова начинающиеся на определённую букву, только для того, чтобы предоставить оппоненту возможность использовать название города, начинающееся с этой буквы).

В связи с этим, в данной работе мы рассмотрим практическое применение механизмов обучения с подкреплением в контексте поиска стратегии действия для игры в «Города» и изучим реализацию машинного обучения с подкреплением, в том числе протестируем влияние политик начисление вознаграждений, реализуем среду и так далее.

Новизна работы состоит в том, что на данный момент мною не были обнаружены существующие алгоритмы на основе обучения с подкреплением, которые бы работали со схожими по механике играми.

# Обучение с подкреплением

## Общая характеристика обучения с подкреплением

Обучение с подкреплением – представляет из себя подраздел машинного обучения, предназначенный для определения оптимальной стратегии действий с целью максимизации вознаграждения, получаемого системой. Поиск стратегии осуществляется путем попыток взаимодействия со средой и изучения, полученного в результате эффекта. При этом не трудно провести аналогию с тем, как живое существо учится взаимодействовать с внешним миром, например малыш учится ходить (Для чего, при решении аналогичной задачи в робототехнике, часто также применяют обучение с подкреплением). Другие сферы приложения включают в себя: системы оптимизации, системы рекомендаций и статистику, теорию управления, теорию игр, теорию информации [2].

## Элементы обучения с подкреплением

В данной модели ответственный за принятие решений объект называется агентом, а предмет с которыми осуществляются его взаимодействия и который передаёт данные об изменившейся ситуации – окружение. В их взаимодействиях агент избирает один из доступных вариантов действия, а окружение реагирует на эти изменения, отправляет агенту изменившиеся данные о себе и награду в виде единственного числового значения. Задача агента состоит в максимизации значения награды. [3] Награда также может иметь отрицательное значение. Также имеется связанное понятие ценности, которое обозначает выгодность решения на длительном отрезке времени, таким образом агент может предпочесть решение, дающее меньшую сиюминутную награду, если оно имеет большую ценность то есть большую потенциальную награду. Кроме них также выделяют такой элемент как стратегия, то есть то какие решения принимаются агентом на входящем множестве состояний окружения. [2]

## Выводы

Таким образом, обучение с подкреплением представляет наиболее близкую имитацию обучения живого организма и не требует обязательных заранее заготовленных критериев кроме параметров состояния, которое требуется достичь, что открывает возможность нахождения неочевидных стратегий действий превосходящих по оптимальности используемые до этого. С другой стороны, данный метод также уязвим для неправильного создания механизма награждения, в результате чего он может формально добиться высокого значения награды без выполнения основной функции.

# Используемая модель

## Агент

Агент (Agent) использует для обучения метод Qleanring. Алгоритм основанный на сопоставлении каждого состояния, действия и разнице наград при принятии соответствующего действия. Данная награда вычисляется через уравнение Беллмана. [4]

Агент инициализирует в себе память модели (MemoryCls – предназначена для запоминания осуществлённых агентом ходов и возвращения данной информации, соответственно функции add и sample), пары Актеров (Actor) и Критиков (Critic) (пара необходима для реализации уравнение Беллмана) и оптимизаторы Adam для них[5,6,7,8].

Кроме init в агенте реализованы следующие функции:

* step – сохраняет ход в памяти агента, если принятый ход был последним в игре – запускает обучение
* act – принимает текущее состояние среды и передает его на соответствующего Актора.
* learn – осуществляет первые два шага уравнения Беллмана для критика и актера и вызывает их оптимизаторы.
* soft\_update – осуществляет третий шаг уравнения Беллмана и корректирует параметры модели в соответствии с результатами
* change\_bsize – предназначался для смены размера поля в выводе критика. Используется лишь для сценариев тестирования

## Актор и критик

Актор (Actor) - данный класс представляет непосредственно нейронную сеть управляющую процессом игры. Стартовые веса задаются не менее 0 в связи с особенностями представления информации о среде (использования отрицательных значений для обозначения закончившихся городов). По аналогичной же причине было решено отказаться от использования функции relu.

Критик (Critic) - данный класс предназначен для оценивания функции стоимости и задаёт Актору направление дальнейшей корректировки параметров. Сравнительно с органическими существами, критика обычно сравнивают с рациональным осмыслением ранее принятых решений. По аналогичным причинам стартовые веса принимают значение не менее 0.[9]

|  |  |  |
| --- | --- | --- |
| № | Структура Актора | Структура Критика |
| 1 | Линейное преобразование: 256 | Линейное преобразование: 256+число выходов |
| 2 | Регуляризация Дропаут | Линейное преобразование: 256 |
| 3 | Линейное преобразование: (количество выходных значений зависит от загруженной базы данных и равно количеству уникальных букв в начале названий городов умноженному на количество уникальных букв в конце названий городов, которые встречаются в списке первых букв) | Линейное преобразование: 128 |
| 4 |  | Линейное преобразование: 1 |

# Разработка среды и политика вознаграждения

## Входные данные и предобработка

От датасета для реализации агента требовалось удовлетворение следующих условий:

* Содержать большое количество уникальных названий городов
* Включать в себя названия городов разных стран и регионов планеты. Это обусловлено тем, что в связи с особенностями языков разных регионов, (например, арабские города содержат большое количество городов начинающихся на согласные) при анализе названий городов лишь из одного региона мы получим не корректное соотношение первых и последних букв по сравнению с миром.

Предобработка полученных данных включает в себя:

* Выбраковку названий городов, не заканчивающихся на символ русской буквы (независимо от регистра) и содержащих в себе аномальные символы (связано это в том числе с тем, что датасет на русском языке, удовлетворяющий первому и второму условию, содержал в себе ряд ячеек с не качественной информацией)
* Извлечение списка уникальных названий городов
* Выделение первого и последнего символа с проверкой вхождение последнего в набор первых (например, есть названия городов, заканчивающиеся на символ «ь», но отсутствуют города, начинающиеся с этого символа. В таком случае последний символ будет смещаться на шаг назад до тех пор, пока не будет найден символ, встречающийся среди первых символов в названиях городов).
* Очистка списка от городов начинающихся на запрещённые буквы (опционально).
* Добавление технической характеристики города "active" (использовалось ли название этого города в игре до этого или нет).

## Реализация среды

Среда отвечает за моделирование изучаемого процесса, предоставление информации о текущем состоянии этого процесса, его изменениях в зависимости от действий агента и выдачу агенту наград и штрафов[10]. В типовой реализации среды она возвращает своё следующее состояние как один из элементов вывода взаимодействия среды и агента, однако в нашем варианте предпочтение было отдано отдельной функции.

В целях ускорения работы было принято решение составить набор доступных агенту действий из выбора необходимой пары первой (в основном для первого хода игры) и последней буков. Таким образом стало возможным резко снизить размеры пула доступных вариантов максимум до 33\*33=1089 вариантов, при условии, что в мире существовали бы города начинающиеся и заканчивающиеся на любую букву алфавита, на практике же получилось меньшее число в 763 варианта, включая один город на букву «ы». К тому же стоит признать, что с точки зрения механики игры разница между городами с одинаковыми буквами в начале и конце отсутствует. Поэтому агент выбирает лишь буквы по краям слова и передаёт их функции with\_letter, которая по ним уже и выбирает название города для следующего хода.

Для реализации данного варианта политики в среду была добавлена переменная start\_last, содержащая в себе табличку с дами-переменными первой и последней буквы в названии города. Так же эта переменная содержит сами первую и последнюю букву для их последующего использования в составлении функции вывода текущего состояния среды. Так при выводе текущего состояния среды столбец с первой буквой отвечает за обозначение последней буквы предыдущего слова, а столбец с последней буквой – отражение того насколько большой выбор вариантов нового слова будет у оппонента на следующий ход. Схожая метрика применяется и для непосредственно рассматриваемых букв.

Другой технической модификацией датасета является добавление к списку городов технического значения (с 'active' всегда равным False) которое выдается, в случае если агент запрашивает вариант города в котором уже закончились все не использованные города.

При запросе текущего состояния среды предоставляется следующая информация, где каждая строчка обозначает информацию о городах начинающихся и заканчивающихся на определённые буквы:

Изображение выглядит как текст, снимок экрана, черно-белый, Шрифт

Автоматически созданное описание

Рисунок 1. Вывод текущего состояния среды

* first – булевое значение, отмечающие пары букв в которых первая буква совпадает с последней буквой предыдущего города (True). В случае если в данный момент осуществляется первый ход все строки считаются истинными. Учитывая важность данного параметра для ускорения обучения модели значению True соответствует сразу 100 балов. Значению False – 0.
* first\_{буква} – дамми переменные для первой буквы в комбинации (True - 1. False – 0)
* last\_{буква} – дамми переменные для последней буквы в комбинации (True - 1. False – 0)
* transf - определяет долю ещё не использованных городов от максимума (максимальное кол-во городов в данной категории) с данными первой и последней буквой. Измеряется от 0 до 1.
* last\_let\_av – определяет долю ещё не использованных городов начинающихся на последнюю букву этого слова (была попытка ввести схожий параметр для первой буквы, но испытания показали неэффективность данного решения). Измеряется от 0 до 1.
* no\_city - булевое значение, отмечающие пары букв под которые не подходит ни один ещё не использованный город (True). Учитывая важность данного параметра для ускорения обучения модели значению True соответствует сразу -100 балов. Значению False – 0.

На основании данной информации агент выбирает пару первая-последняя буква и через функцию with\_letter вводит соответственное ещё не использованное название города.

## Политика начисления награды

Награда начисляется по следующей политике.

* За простой ход игроку начисляется небольшой штраф для создания мотивации выигрыша на более раннем ходу (изначально рассматривался вариант в виде 0.01 и даже 0.1, но практика показала, что это скорее приводит к переобучению, чем к ускорению обучения, поэтому был выбран 0.005).
* За победный/проигрышный ход соответствующим агентам начисляется награда/штраф в 100 баллов.

Соответственно начисление происходит после хода оппонента (для 0.005) или конца игры (для 100)

Как альтернативы рассматривались возможности награждать проигравшего за большее количество шагов (с возвратом до 45% баллов при достаточно длинной игре), не штрафовать простые ходы (из соображения, что затягивание игры одним из агентов – возможность выиграть игру другому, что в итоге должно создавать альтернативу штрафу) и корректировка штрафов промежуточных шагов по результатам победы/поражения агента, но в результате эти идеи были отвергнуты как менее эффективные.

## Процесс обучения и итоги

Модель обучается, играя сама с собой (при этом в каждую эпоху первый и последний игрок меняются местами). Эпоха заканчивается после первого же неправильного хода. Награда начисляется после хода оппонента или же после окончания текущего раунда игры. После этого выводится информация о причинах завершения игры, количестве шагов до конца игры, количестве городов которые могли бы быть названы для продолжения игры и использованные во время игры города в хронологическом порядке. После завершения последней эпохи функция возвращает следующие списки: количества ходов, ходы игроков, причины завершения игры, последняя буква последнего использованного города.

В результате обучения модели на данных, содержащих и не содержащих букву «ы» (единственный город в датасете) были получены следующие результаты:

|  |  |  |
| --- | --- | --- |
|  | С «ы» | Без «ы» |
| Количество ходов в игре | Первые 11 эпох почти всегда завершаются в течение менее чем 10 ходов, после чего Агент адаптируется к условиям игры и количество ходов возрастает | Стабильное количество ходов в первые 9 эпох (Агенты корректируют свои тактики, но не настолько серьёзно, чтобы это сказалось на длительности партий. После этого до 12 эпохи длительность игр снижается вплоть до 10 раз и после 12 эпохи вновь начинает расти |
| На какой букве наступает проигрыш | Ы (один город) | Й (881 город) и Е (1885) |

Итоги работы модели с буквой «ы» и без

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

Рисунок 2. График изменения количества ходов с буквой «ы»

Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание

Рисунок 3. Г рафик изменения количества ходов без городов на букву «ы»

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 4. Список ходов в ходе проведения обучения

Изображение выглядит как круг, диаграмма, Графика

Автоматически созданное описание

Рисунок 5. Процентное распределение последних букв в играх без городов на букву «ы»

Изображение выглядит как текст, снимок экрана, Шрифт, черно-белый

Автоматически созданное описание

Рисунок 6. Вывод состояния среды после 15 эпохи

Во всех случаях агенты смогли не совершать ошибочных ходов при наличии альтернативного решения.

Заключение

В результате выполнения работы мы получили функционального виртуального агента, способного развивать свою тактику в результате обучения, выявлять наиболее оптимальные ходы и редко встречающиеся буквы, а также создавать для оппонента условия гарантированного проигрыша.

Были освоены механизмы обучения с подкреплением, получены практические знания по созданию политики среды, применению обучения с подкреплением с критиком.

Список использованных источников

1. Люгер Д. Ф. Искусственный интеллект: стратегии и методы решения сложных проблем : пер. с англ. 4-е изд. М. : Вильямс, 2003. 864 с
2. Обучение с подкреплением / Саттон Р.С., Барто Эн. Дж.; главный редактор Д. А. Мовчан. — Москва: Издательство ДМК ПРЕСС, 2020. — 553 с.
3. Стратегии исследования окружений в обучении с подкреплением с непрерывными пространствами состояний / Гурьянов Ал. К.; Научный руководитель Дьяконов Ал. Г.. — Московский государственный университет имени М.В. Ломоносова Факультет Вычислительной Математики и Кибернетики, 2017. — 38 с. — (Магистерская диссертация).
4. Сравнительный анализ систем глубокого обучения с подкреплением и систем обучения с учителем / Жиленков А. А., Силкин А. А., Серебряков М. Юр., Колесова С. В. // Известия ТулГУ. Технические науки. 2022. №10. URL: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-sistem-glubokogo-obucheniya-s-podkrepleniem-i-sistem-obucheniya-s-uchitelem (дата обращения: 07.05.2023)
5. DEEP LEARNING НА ПАЛЬЦАХ / Бабуров Юр. [и др.] ; — (интернет источник). — Текст: электронный // ДЛ курс [сайт]. — URL: https://dlcourse.ai/ (дата обращения: 01.05.2023).
6. Deep Reinforcement Learning Tutorials - All Videos; [Электронный ресурс]. – URL:https://www.youtube.com/playlist?list=PL-9x0\_FO\_lgmP3TtVCD4X1U9oSalSuI1o (дата обращения: 27.04.2023).
7. Practical RL / Machine Learning with Phil; [Электронный ресурс]. – URL: <https://github.com/yandexdataschool/Practical_RL> (дата обращения: 20.04.2023).
8. Ростовцев В.С. Искусственные нейронные сети: учебник для вузов / В.С. Ростовцев. 2-е изд., стер. Санкт-Петербург: Лань, 2021. 216 с.
9. Хливненко Л.В. Практика нейросетевого моделирования: учебное пособие для вузов / Л.В. Хливненко, Ф.А. Пятакович. 2-е изд., стер. Санкт-Петербург: Лань, 2021. 200 с.
10. Training a Goal-Oriented Chatbot with Deep Reinforcement Learning [Электронный ресурс]. – URL: https://towardsdatascience.com/training-a-goal-oriented-chatbot-with-deep-reinforcement-learning-part-i-introduction-and-dce3af21d383 (дата обращения: 12.04.2021)

Приложение А. Программный код

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import re

import torch.nn as nn

import random

import torch

import torch.optim as optim

from collections import deque,namedtuple

import warnings

from collections import OrderedDict

warnings.filterwarnings('ignore')#выключаем системные предупреждения

sity\_num=650#городов в эпохе - переменная использовалась при тестировании

ep=15#кол-во эпох

#настройки сети

dis\_fuc = 0.99

TAU = 1e-4

SN\_WEIGHT = 1e-4

LR\_CR = 5e-4

LR\_AC = 1e-4

MEM\_SIZE = int(1e6)

forbidden\_letters = ['ы']

data=pd.read\_csv('cities.csv', encoding='cp1251')

cites=pd.DataFrame(data['Город'].unique(),columns=['name'])

cites

cites=cites[cites.name.apply(lambda x: True if (re.fullmatch(r'[а-яА-ЯёЁ][а-яА-ЯёЁ\s\'-]+[а-яА-ЯёЁ]', str(x))) else False)]

cites['first']=cites['name'].str[0].str.lower()

cites=cites[~cites['first'].isin(forbidden\_letters)]

f\_unique=cites['first'].unique()

print(f\_unique)

cites['last']='0'#cites['name'].str[-1].str.lower()

cites['active']=True

#Ищем букву входящую в число букв с которых начинаются названия городов

def f(row, f\_unique):

i=-1

lett=row['last']

while True:

if lett in f\_unique:

break

else:

lett=row['name'][i].lower()

i-=1

return lett

cites['last']=cites.apply(f, axis=1,f\_unique=f\_unique)

stat=cites['name'].str[-1].str.lower()!=cites['last']

print(sum(stat))

print(cites[stat])

cites['first'].value\_counts()

cites['transf']=cites['first']+cites['last']

stat=pd.merge(cites['last'].value\_counts(), cites['first'].value\_counts(), left\_index=True, right\_index=True).sort\_index()

stat['last/first']=stat['last']/stat['first']

fig = plt.figure(figsize =(15, 12))

stat[['first','last']].plot.bar();

plt.title('Количество букв в начале и в конце')

plt.figure(figsize =(12, 9))

stat['first'].plot.pie(autopct='%1.1f%%',pctdistance=1,explode=[0.1 for i in range (stat.shape[0])], textprops={'fontsize': 8});

plt.title('Процентное разбиение первых букв')

plt.figure(figsize =(12, 9))

pd.Series(last\_let).value\_counts().plot.pie(autopct='%1.1f%%',pctdistance=0.8, textprops={'fontsize': 8});

plt.title('Последняя буква в игре')

plt.figure(figsize =(12, 9))

stat['last'].plot.pie(autopct='%1.1f%%',pctdistance=1,explode=[0.1 for i in range (stat.shape[0])], textprops={'fontsize': 8});

plt.title('Процентное разбиение последних букв')

stat=pd.DataFrame((cites['transf']).value\_counts().sort\_index())

letters=stat.index

start\_last=pd.DataFrame(stat.index,index=stat.index)#,columns=['transf'])

start\_last['first']=start\_last[0].str[0]

#'''

start\_last['last']=start\_last[0].str[1]

start\_last=start\_last.drop(0, axis=1)

#'''

for i in (['first','last']):

k=start\_last[i].unique()

start\_last=start\_last.join(pd.get\_dummies(start\_last[i]))#.drop(i, axis=1)

for ii in (k):

start\_last=start\_last.rename(columns={ii: i+"\_"+ii})

#'''

#start\_last=start\_last.drop('last', axis=1)

start\_last#'''

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

class Environment():

def \_\_init\_\_(self, cites, start\_last, apendd=False):

#Последняя буква последнего названия города. False - любая (для начала игры)

self.prev\_letter=False

#Техническо-костыльное значение.

self.cites=cites.append(pd.DataFrame({'name': ['Уже использован'], 'first': ['%'],'active':[False]}), ignore\_index = True)

self.apendd=apendd

self.start\_last=start\_last

def reset(self):

self.cites['active']=self.cites["name"]!="Уже использован"

self.prev\_letter=False

def cites\_updet(self,cites):

self.cites=cites.append(pd.DataFrame({'name': ['Уже использован'], 'first': ['%'],'active':[False]}), ignore\_index = True)

def step(self,city):

done=False

reward=-0.005#0.05

inf=''

if self.prev\_letter==False or city[0].lower()==self.prev\_letter:

cit=self.cites[self.cites["name"] == city]

if cit.shape[0]>0:

if cit["active"].iloc[0]:

self.prev\_letter=cit['last'].iloc[0]

self.cites["active"][self.cites["name"] == city]=False

else:

done=True

reward=-100

inf='Повторное использование города'

print()

elif self.apendd:

self.cites.append(f(pd.DataFrame({'name': [city], 'first': [city[0].lower()], 'last': [city[-1]],'active':[False]}), f\_unique), ignore\_index = True)

self.prev\_letter=self.cites[self.cites["name"] == city]['last']

else:

done=True

reward=-100

inf='Введённое значение не является городом'

print(inf)

else:

done=True

reward=-100

inf='Не правильная первая буква.'

print(inf,'Город:',city,'Искомая буква',self.prev\_letter)

return(reward,done,inf)

def stat(self):

zam=100

cit=pd.concat([self.start\_last,self.cites[self.cites["active"]]['transf'].value\_counts()], axis=1, join='outer')

t=cit.groupby('first').sum()["transf"]

t=pd.DataFrame({'lastt':t.index,'last\_let\_av':t.values})

cit=cit.merge(t,left\_on='last',right\_on='lastt',how='outer')

cit=cit.sort\_values(by=['first','last'])

cit=cit.drop('last', axis=1).drop('lastt', axis=1)

if self.prev\_letter:

cit['first']=(cit['first']==self.prev\_letter)\*zam

else:

cit['first']=zam

cit['transf']=(1-cit['transf']/cit['transf'].max())#\*1.5#\*10#1-

#cit['transf']=cit['transf'].fillna(-zam)

cit['no\_city']=cit['transf'].isnull()\*-zam

cit['transf']=cit['transf'].fillna(0.5)#-zam)

cit['last\_let\_av']=cit['last\_let\_av'].fillna(0)

cit['last\_let\_av']=(1-cit['last\_let\_av']/cit['last\_let\_av'].max())/2#\*1.5

return cit

def with\_letter(self, letters):

cit=self.cites[(self.cites["active"])&(self.cites["first"]==letters[0])&(self.cites["last"]==letters[1])]

if cit.shape[0]>0:

cit=cit.sample(n=1).name.values[0]

else:

cit="Уже использован"

return cit

env=Environment(cites, start\_last)

env.stat()

class Actor(nn.Module):

def \_\_init\_\_(self, state\_size, action\_size, seed, hid\_units=256):

super(Actor, self).\_\_init\_\_()

self.seed = torch.manual\_seed(seed)

self.fc1 = nn.Linear(state\_size, hid\_units)

#self.relu = nn.ReLU()# Ч

self.drout = nn.Dropout(p=0.5, inplace=False)

self.fc2 = nn.Linear(hid\_units, action\_size)

self.reset\_parameters()

def reset\_parameters(self):

self.fc1.weight.data.uniform\_(\*hidden\_init(self.fc1))

self.fc2.weight.data.uniform\_(0, 1e-4)

def forward(self, state):

#state = self.drout(self.relu(self.fc1(state)))

state = self.drout((self.fc1(self.drout(state))))

return nn.functional.tanh(self.fc2(state))

class MemoryCls:

def \_\_init\_\_(self, action\_size, mem\_size, seed):

self.mem\_size=mem\_size

self.action\_size = action\_size

self.memory = deque(maxlen=mem\_size)

self.experience = namedtuple("Experience", field\_names=["states", "action", "reward", "next\_states", "done"])

self.seed = random.seed(seed)

def add(self, state, action, reward, next\_state, done):

e = self.experience(state, action, reward, next\_state, done)

self.memory.append(e)

def sample(self):

states = torch.from\_numpy(np.vstack([e.states for e in self.memory if e is not None])).float().to(device)

actions = torch.from\_numpy(np.vstack([e.action for e in self.memory if e is not None])).float().to(device)

rewards = torch.from\_numpy(np.vstack([e.reward for e in self.memory if e is not None])).float().to(device)

next\_states = torch.from\_numpy(np.vstack([e.next\_states for e in self.memory if e is not None])).float().to(device)

dones = torch.from\_numpy(np.vstack([e.done for e in self.memory if e is not None]).astype(np.uint8)).float().to(device)

self.memory = deque(maxlen=self.mem\_size)

return (states, actions, rewards, next\_states, dones)

def \_\_len\_\_(self):

return len(self.memory)

class Agent():

def \_\_init\_\_(self, state\_size, action\_size, random\_seed, bsize,MEM\_SIZE,LR\_AC,SN\_WEIGHT,dis\_fuc):

self.state\_size = state\_size

self.action\_size = action\_size

self.seed = random.seed(random\_seed)

#Память

self.memory = MemoryCls(action\_size, MEM\_SIZE, random\_seed)

self.dis\_fuc=dis\_fuc

#Актор

self.actish = Actor(state\_size, action\_size, random\_seed).to(device)

self.actor\_fi = Actor(state\_size, action\_size, random\_seed).to(device)

self.actor\_optimizer = optim.Adam(self.actish.parameters(), lr=LR\_AC)

#Критик

self.critic\_local = Critic(state\_size, action\_size, random\_seed, bsize=bsize).to(device)

self.critic\_fi = Critic(state\_size, action\_size, random\_seed, bsize=bsize).to(device)

self.critic\_optimizer = optim.Adam(self.critic\_local.parameters(), lr=LR\_CR, weight\_decay=SN\_WEIGHT)

def change\_bsize(self, bsize):#смена размера поля. В текущей версии использовалось лишь для текстов

self.critic\_local.bsize=bsize

self.critic\_fi.bsize=bsize

def step(self, state, action, reward, next\_state, done):

self.memory.add(state, action, reward, next\_state, done)

#В образце работали с передвижением объекта, поэтому там обучение шло по прошествию определённого интервала в ходах.

#Здесь же лучше себя показала тактика с разбиением на отдельные игры и обучением после конца каждой игры

if done:

self.learn(self.memory.sample())

def act(self, state):#Непосредственно получение реакции на ситуацию (state)

state = torch.from\_numpy(state.to\_numpy().astype(np.float32)).float().to(device)

self.actish.eval()

with torch.no\_grad():

action = self.actish(state).cpu().data.numpy()

self.actish.train()

return action

def learn(self,mem\_l):#Запоминание результатов

states, actions, rewards, next\_states, dones = mem\_l

actions\_next = self.actor\_fi(next\_states)

Q\_fis\_next = self.critic\_fi(next\_states, actions\_next)

Q\_fis = rewards + (self.dis\_fuc \* Q\_fis\_next \* (1 - dones))

Q\_expected = self.critic\_local(states, actions)

critic\_loss = nn.functional.mse\_loss(Q\_expected, Q\_fis)

self.critic\_optimizer.zero\_grad()

critic\_loss.requires\_grad = True

critic\_loss.backward()

self.critic\_optimizer.step()

actions\_pred = self.actish(states)

actor\_loss = -self.critic\_local(states, actions\_pred).type(torch.FloatTensor).mean()

self.actor\_optimizer.zero\_grad()

actor\_loss.requires\_grad = True

actor\_loss.backward()

self.actor\_optimizer.step()

self.soft\_update(self.critic\_local, self.critic\_fi, TAU,['fcs1.weight', 'fcs1.bias', 'fc2.weight', 'fc2.bias', 'fc3.weight', 'fc3.bias', 'fc4.weight','fc4.bias'])

self.soft\_update(self.actish, self.actor\_fi, TAU,['fc1.weight', 'fc1.bias', 'fc2.weight', 'fc2.bias'])

def soft\_update(self, act\_model, cel\_model, tau,names):#Обновление весов

ordered\_list = []#Для костыльного метода обновления, так как тот же что и у остальных не работает

i=0

with torch.no\_grad():

for cel\_param, act\_param in zip(cel\_model.parameters(), act\_model.parameters()):

ordered\_list.append((names[i],act\_param.data.copy\_(act\_param.data + (1.0-tau)\*cel\_param.data)))

i+=1

act\_model.load\_state\_dict(OrderedDict(ordered\_list))

class Critic(nn.Module):

def \_\_init\_\_(self, state\_size, action\_size, seed, bsize,fcs1\_units=256, fc2\_units=256, fc3\_units=128):

self.bsize=bsize

super(Critic, self).\_\_init\_\_()

self.seed = torch.manual\_seed(seed)

self.fcs1 = nn.Linear(state\_size, fcs1\_units)

self.fc2 = nn.Linear(fcs1\_units+action\_size, fc2\_units)

self.fc3 = nn.Linear(fc2\_units, fc3\_units)

self.fc4 = nn.Linear(fc3\_units, 1)

self.reset\_parameters()

def reset\_parameters(self):

self.fcs1.weight.data.uniform\_(\*hidden\_init(self.fcs1))

self.fc2.weight.data.uniform\_(\*hidden\_init(self.fc2))

self.fc3.weight.data.uniform\_(\*hidden\_init(self.fc3))

self.fc4.weight.data.uniform\_(0, 3e-3)

def forward(self, state, action):

"""Build a critic (value) network that maps (state, action) pairs -> Q-values."""

x = self.fcs1(state)

x = torch.cat((x, action), dim=1)

x = self.fc2(x)

x = self.fc3(x)

x = self.fc4(x)

x = x.split(self.bsize)

x = torch.stack(list(x), dim=0)

return torch.argmax(x, dim=1)

def hidden\_init(layer):

fan\_in = layer.weight.data.size()[0]

lim = 1. / np.sqrt(fan\_in)

return (0, lim)

def ddpg(n\_episodes=1):

steps\_ar=[]

cites\_ar=[]

inf\_ar=[]

last\_let=[]

for i\_episode in range(n\_episodes):

env.reset()

steps=0

city\_list=[]

while True:

player=steps%2

steps+=1

state = env.stat()

action = agent[player].act(state)

city\_name=env.with\_letter(letters[np.argmax(np.sum(action,axis=1))])

city\_list.append(city\_name)

reward, done, inf = env.step(city\_name)

#Присвоение награды откладывается на один ход, так как если оппонент на этом ходу проиграет, то стандартный штраф

#предназначенный для стимуляции сокращения кол-ва ходов в партии, заменяется на награду за победу

if done:

agent[(player+1)%2].step(prev\_state,prev\_action, -reward, env.stat(), done)

#agent[player].step(state,action, (reward+min(steps,49)), env.stat(), done)

agent[player].step(state,action, reward, env.stat(), done)

break

elif steps>1:

agent[(player+1)%2].step(prev\_state, prev\_action, reward, env.stat(), done)

prev\_action=action

prev\_state = state

#agent[0].actish.state\_dict()

#agent[1].actish.state\_dict()

inf\_ar.append(inf)

steps\_ar.append(steps)

cites\_ar.append(city\_list)

last\_let.append(env.prev\_letter)

print('\n\nЭпизод',i\_episode,' Шаг завершения игры:',steps,'Количество городов начинающихся на искомую букву (в идеале = 0):',sum(env.stat()['first']==1),'\nГорода: ','->'.join(city\_list))

agent[0], agent[1] = agent[1], agent[0]

return(steps\_ar,cites\_ar,inf\_ar,agent,last\_let)

env=Environment(cites, start\_last)

agent = [Agent(state\_size=start\_last.shape[1]+1,action\_size=start\_last.shape[0], random\_seed=10,bsize=env.stat().shape[0],MEM\_SIZE=MEM\_SIZE,LR\_AC=LR\_AC,SN\_WEIGHT=SN\_WEIGHT,dis\_fuc=dis\_fuc),#start\_last.shape[0]),

Agent(state\_size=start\_last.shape[1]+1,action\_size=start\_last.shape[0], random\_seed=10,bsize=env.stat().shape[0],MEM\_SIZE=MEM\_SIZE,LR\_AC=LR\_AC,SN\_WEIGHT=SN\_WEIGHT,dis\_fuc=dis\_fuc)]#start\_last.shape[0])]

steps\_ar,cites\_ar,inf\_ar,agent,last\_let =ddpg(ep)

fig = plt.figure()

plt.plot(np.arange(ep), steps\_ar)

plt.ylabel('Шагов')

plt.xlabel('Эпизод #')

plt.show()

plt.figure(figsize =(12, 9))

pd.Series(inf\_ar).value\_counts().plot.pie(autopct='%1.1f%%',pctdistance=0.8, textprops={'fontsize': 8});

plt.title('Процентное разбиение причин завершения игры')

env.stat()[env.stat()['first']!=0]

Ссылка на гитхаб: https://github.com/VladKsenobayt/Goroda\_kursov