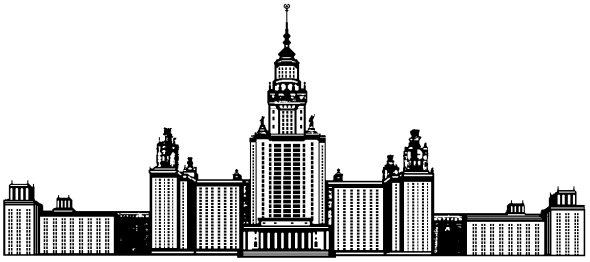

Московский государственный университет имени М. В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Математических Методов Прогнозирования

Коробов Павел Андреевич

Обучение с подкреплением в задаче рекомендаций

Reinforcement learning for recommender systems

КУРСОВАЯ РАБОТА

Научный руководитель:
н. с. Д. А. Кропотов

Москва, 2020

Содержание

1 Введение

1.1 Обучение с подкреплением

Обучение с подкреплением – это область машинного обучения, в которой решается задача максимизации некоторой награды в ходе взаимодействий с окружающей средой.

Окружающая среда в задаче обучения с подкреплением обычно задаётся марковским процессом принятия решений. Это четвёрка $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, где:

- $\mathcal{S}$ – пространство состояний
- $\mathcal{A}$ – пространство действий
- $\mathcal{P} : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \to \mathbb{R}^+$ – функция переходных вероятностей $p\left(s'|s, a\right)$
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ – функция награды

Система, отвечающая за выбор действий при взаимодействии со средой, называется агентом. На рисунке 1 схематично изображена схема взаимодействия агента со средой.

Сумма $R_t = \sum_{i=t}^{T} \gamma^{i-t} r\left(S_i, A_i\right)$ называется кумулятивной наградой. Цель агента – найти стратегию $\pi : \mathcal{S} \to \mathcal{A}$, максимизирующую ожидаемую кумулятивную награду $\mathbb{E}\left[R_1\right]$. Агенту неизвестны переходные вероятности и функция награды, он должен найти оптимальную стратегию методом проб и ошибок.

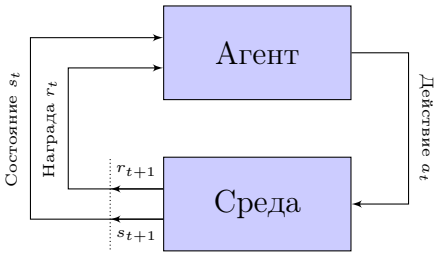

Рис. 1: Схема марковского процесса принятия решений

Q-функция состояний-действий $Q^\pi(s, a) = \mathbb{E}\left[R_1 | S_1 = s, A_1 = a, \pi\right]$ определяется как ожидаемая кумулятивная награда при условии начального состояния s, выбора действия a и следования политике π.

1.2 Задача рекомендаций

Пусть M и N – числа пользователей и объектов соответственно. Матрица $Y \in \mathbb{R}^{M \times N}$ хранит в себе оценки, выставленные объектам пользователями. Компонента y_{ij} соответствует оценке i-м пользователем j-го объекта. Наша цель – предсказать для пользователей рейтинги непросмотренных ими объектов, то есть пропущенные значения в данной матрице, и рекомендовать пользователям объекты с наибольшим предсказанным рейтингом.

Как правило, эта задача формулируется как задача обучения с учителем, и её решение никак не учитывает влияние рекомендаций на поведение пользователей в долговременной перспективе.

Существует известный эффект, присущий персонализированным сервисам, заключающийся в изоляции пользователей от разнообразия точек зрения. Этот эффект называется пузырём фильтров, и рекомендательные сервисы потенциально подвержены этому эффекту. В [?] авторы показали, что разнообразие рекомендаций для пользователей сервиса рекомендаций фильмов MovieLens стало статистически значимо меньше в конце истории использования сервиса пользователем, чем оно было в начале.

Независимо от того, вызывается этот эффект естественно (например, тем, что хороших фильмов в целом не так много, и с большей персонализацией рекомендаций это множество только сужается) или недостатками алгоритма рекомендаций, обучение с подкреплением кажется более подходящей парадигмой для задачи рекомендаций, чем обучение с учителем.

Задача рекомендаций хорошо вписывается в концепцию обучения с подкреплением: оно явным образом учитывает поведение пользователей (реакцию среды) и максимизирует долгосрочную награду. Более того, в случае, если пользователь начнёт получать однообразные рекомендации, и его пользовательский опыт от этого начнет ухудшаться, мы можем надеяться, что агент, заметив ухудшение оценок пользователя, сможет найти более удачную стратегию, чем слишком подстраиваться под интересы пользователя, и в дальнейшем использовать этот опыт.

Кроме того, есть основания полагать, что с помощью алгоритмов обучения с подкреплением получится явно поддерживать разнообразие рекомендаций с помощью методов исследования среды. Например, этого можно достичь с помощью Maximum Entropy RL алгоритмов [?, ?].

2 Постановка задачи

Для оценки алгоритма обучения с подкреплением нужна среда или некоторый симулятор среды, на котором можно было бы убедиться в успешной работе алгоритма. Мы будем использовать такой же симулятор среды для рекомендаций, как в оригинальной статье, представившей алгоритм Wolpertinger [**?**].

Предположим, что пользователи неразличимы и имеется некоторое множество объектов для рекомендаций, задающее пространство действий $\mathcal{A}$. Будем считать, что объекты из $\mathcal{A}$ некоторым образом пронумерованы. Пусть также задана матрица W, компоненты w_{ij} которой задают вероятность принятия рекомендации j-го объекта при условии, что пользователь просматривает i-ый объект. За состояния принимаются объекты, просматриваемые пользователем в данный момент. Таким образом, $\mathcal{S} = \mathcal{A}$.

Пользователь заканчивает эпизод с вероятностью 0.1, если рекомендация была принята, и с вероятностью 0.2 в противном случае. Если эпизод продолжается, в случае принятия рекомендации пользователь начинает просматривать рекомендованный объект, иначе равновероятно начинает просмотр любого из $\mathcal{A}$.

Несложно увидеть, что оптимальная политика является детерминированной и будет иметь вид

$$\pi(s) = \text{argmax}_j\{w_{ij} \mid i \text{ – номер текущего состояния } s\}.$$

По сути задача сводится к нахождению максимальных значений в строках матрицы W. К сожалению, авторы [**?**] не описали ни способа построения W в своем эксперименте, ни способа построения эмбеддингов для объектов.

За основу эксперимента в данной работе были взяты данные конкурса The Netflix Prize. В этом наборе данных содержится 24 053 764 рейтингов на 17 770 фильмов от 470 758 пользователей. Было решено строить матрицу W и эмбеддинги для элементов $\mathcal{A}$ по этим данным.

Опишем, как задаётся матрица W. Пусть P_{ij} – число положительных рейтингов фильма j, выставленных в течение l дат позднее просмотра фильма i, а C_{ij} – число всех рейтингов фильма j, выставленных в течение l дат позднее просмотра фильма i.

Для подсчета этих величин матрицы P и C инициализируются нулевыми матрицами, далее внутри истории каждого пользователя рейтинги разбиваются на даты.

Мы перебираем всех пользователей, и для текущего пользователя в цикле перебираем пары фильмов (i, j) из его истории, где i соответствует

фильму, просмотренному в некоторую обрабатываемую в данный момент дату, а j – фильму, просмотренному в одну из следующих l дат. При каждом нахождении такой пары (i, j) среди историй всех пользователей мы увеличиваем счетчики $C_{ij} := C_{ij} + 1$, $P_{ij} := P_{ij} + b(r_{ij})$, где

$$b(x) = \begin{cases} 1 & \text{если } x > 3 \\ \frac{1}{2} & \text{если } x = 3 \\ 0 & \text{иначе} \end{cases}.$$

Пусть также $p = \sum_r b(r)$, $c = \sum_r 1$ – суммы по всем рейтингам r в данных.

Наконец, будем определять компоненты $w_{ij} = \frac{P_{ij}+p}{C_{ij}+c}$. То есть это доли положительных рейтингов, поставленных пользователями фильму j в течение l дат в их истории после просмотра фильма i. Такое определение вполне соответствует определению матрицы W в эксперименте из [?], где смысл величины w_{ij} – вероятность принятия рекомендации объекта j при рассмотрении объекта i. В качестве априорных вероятностей берётся доля положительных рейтингов среди всех рейтингов $\frac{p}{c}$.

Эмбеддинги расчитывались с помощью алгоритма Skip-gram [?], реализованного классом Word2Vec библиотеки gensim. Для применения алгоритма к рекомендуемым объектам история пользователей трактуется как предложения, а объекты – как слова.

3 Методы

3.1 Алгоритм Wolpertinger

Применение алгоритмов, основанных на явной максимизации Q-функции, таких как DQN [?], может быть затруднительно при решении задачи рекомендаций. Перебор всех возможных действий во время поиска максимума Q-функции может быть слишком вычислительно затратен, так как нередко от рекомендательных систем требуется способность делать выбор среди миллионов объектов. К тому же, в случае DQN, нужно оценить Q-функцию на каждом действии из $\mathcal{A}$. То есть только для оценки $Q(s, \cdot)$ при одном фиксированном s алгоритму требуется как минимум $|\mathcal{A}|$ шагов. Хотелось бы, чтобы алгоритм имел чуть лучшие обобщающие свойства: если мы умеем оценивать Q-функцию для какого-то одного действия, то было бы хорошо иметь близкие её значения для похожих действий.

Таким свойством обладают алгоритмы типа Actor-Critic для непрерывного контроля благодаря непрерывности критика. Кроме того, алгоритмы этого класса избегают явной максимизации Q-функции.

Алгоритм Wolpertinger [?] переносит алгоритм DDPG [?] на дискретное пространство действий. Дискретные действия представляются эмбеддингами из $\mathbb{R}^d$. Wolpertinger использует актёра DDPG, которого будем обозначать как $f_{\theta^\pi}(\cdot)$, для выбора так называемых протодействий $\hat{a} = f_{\theta^\pi}(s)$ из $\mathbb{R}^d$. При этом $\hat{a}$ может вовсе не соответствовать ни одному действию из $\mathcal{A}$. Поэтому алгоритм ищет k ближайших соседей $\hat{a}$ из $\mathcal{A}$ и выбирает среди них действие наиболее высоко оцениваемое критиком. Таким образом формируется полная политика $\pi(\cdot)$ агента Wolpertinger, показанная в алгоритме 1.

Алгоритм 1: Политика Wolpertinger

Пусть s – состояние полученное из среды
$\hat{a} = f_{\theta^\pi}(s)$ // Выбрать протодействие
$\mathcal{A}_k = \text{kNN}_{\mathcal{A}}(\hat{a})$ // Найти k ближайших действий из $\mathcal{A}$ к $\hat{a}$
$a = \text{argmax}_{a_j \in \mathcal{A}_k} Q_{\theta^Q}(s, a_j)$ // Выбрать итоговое действие
Применить действие a к среде; получить r и $s\prime$

Алгоритм 2 полностью описывает устройство Wolpertinger. Критик обучается исключительно на полной политике $\pi(\cdot)$, в то время как актёр для дифференцируемости оптимизируется по частичной политике $f(\cdot)$.

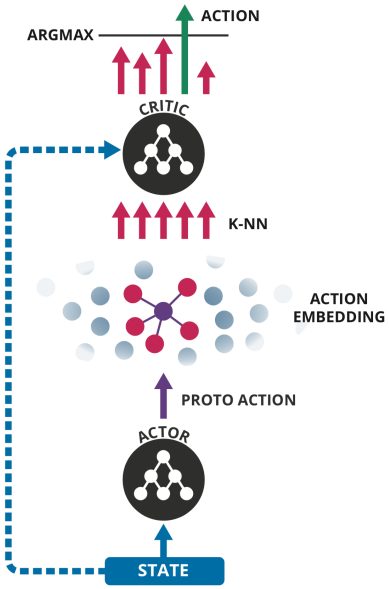

Рис. 2: Схема политики Wolpertinger

Алгоритм 2: Wolpertinger

Инициализировать нейронные сети критика Q_{θ^Q} и актёра f_{θ^π} весами θ^Q и θ^π

Инициализировать таргет-сети критика $Q_{\theta^{Q'}}$ и актёра $f_{\theta^{\pi'}}$ теми же весами $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\pi'} \leftarrow \theta^\pi$

Инициализировать буффер памяти B

для эпизода $e = 1, M$

  Получить начальное состояние s_1

  для $t = 1, T$

    Выбрать действие $a_t = \pi_\theta(s_t)$ согласно текущей политике и методу исследования среды

    Отдать в среду a_t, пронаблюдать r_t и s_{t+1}

    Сохранить кортеж (s_t, a_t, r_t, s_{t+1}) в буфер B

    Семплировать из B минибатч из N кортежей (s_i, a_i, r_i, s_{i+1})

    Присвоить $y_i = r_i + \gamma \cdot Q_{\theta^{Q'}}\left(s_{i+1}, \pi_{\theta'}(s_{i+1})\right)$

    Обновить веса критика, минимизируя невязку $L\left(\theta^Q\right) = \frac{1}{N}\sum_i \left[y_i - Q_{\theta^Q}\left(s_i, a_i\right)\right]^2$

    Обновить веса актёра, сделав шаг по градиенту политики $g_\pi \approx \frac{1}{N}\sum_i \nabla_a Q_{\theta^Q}(s,a)\big|_{a=f_{\theta^\pi}(s_i)} \nabla_{\theta^\pi} f_{\theta^\pi}(s)\Big|_{s=s_i}$

    Обновить таргет-сети:

    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$

    $\theta^{\pi\prime} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi\prime}$

Авторы предлагают использовать приближенный поиск ближайших соседей, но чтобы убедиться в корректной работе алгоритма, мы будем искать соседей точно.

3.2 Модификация Wolpertinger

Непрерывность функции критика $Q_{\theta^Q}(s, \cdot)$ по действиям и её оценивание лишь в точках, заданных эмбеддингами действий из $\mathcal{A}$, может привести к тому, что протодействия, удаленные от множества $\mathcal{A}$ могут ошибочно максимизировать $Q_{\theta^Q}(s, \cdot)$. Это может спровоцировать нежелательное поведение актёра выбирать протодействия, находящиеся в областях, где критик элементарно не обучен ни на какие значения. Непрерывность критика может помочь близко оценивать похожие протодействия лишь в некоторых окрестностях точек $a \in \mathcal{A}$, чего может быть недостаточно в случае неплотного размещения эмбеддингов в $\mathbb{R}^d$.

Мы покажем, что l_2-регуляризация сетей вместе с использованием описанного в уравнении 1 добавочного слагаемого в функции потерь может существенно улучшить результаты работы алгоритма. Мы попробу-

ем регуляризовать функцию потерь критика следующим образом:

$$L\left(\theta^{Q}\right) := \frac{1}{N}\sum_{i}\left[y_i - Q_{\theta^Q}\left(s_i, a_i\right)\right]^2 + \frac{1}{N}\sum_{i}\left[Q_{\theta^Q}\left(s_i, \hat{a}_i\right) - Q_{\theta^Q}\left(s_i, \mathrm{1NN}\left(\hat{a}_i\right)\right)\right]^2 \tag{1}$$

Второе слагаемое несёт следующий смысл: даже если рассматриваемая точка $\hat{a}$ находится далеко от множества $\mathcal{A}$, мы хотим, чтобы значение в этой точке не превышало значения на ближайшей точке из этого множества. Таким образом, при оптимизации функции потерь актёра $J(\theta^\pi) = Q(s, \pi_{\theta^\pi}(s))$, должна снизиться вероятность того, что актёр обучится выбирать слишком далёкие точки в качестве протодействий.

4 Эксперименты

Взглянем на средние значения кумулятивных наград. Будем проводить эксперимент в течение 500 000 шагов взаимодействия со средой. На графиках будем изображать среднюю награду за каждые 500 шагов. Для каждого агента будем проводить по три запуска эксперимента и усреднять получаемые графики, отображая средние по запускам значения и коридор стандартного отклонения.

Первые 50000 шагов алгоритм накапливает опыт, выбирая случайные действия, далее исследование среды проводится ε-жадно с $\varepsilon = 0.05$, $\gamma = 0.99$, $\tau = 0.001$, размер буффера памяти равен 500000, так что он хранит весь опыт в течение эксперимента.

Везде используются полносвязные нейронные сети с тремя скрытыми слоями и 256 скрытыми нейронами. На выходах критика и актёра находятся линейная активация и гиперболический тангенс соответственно. После каждого внутреннего слоя используется ReLU-активация. Размерность эмбеддингов действий равна 20, learning_rate = 0.0003 для обеих сетей, batch_size = 128.

Мы возьмём небольшое пространство действий: $|\mathcal{A}| = 100$, чтобы можно было легче оценивать работу алгоритма. Будем рассматривать алгоритм Wolpertinger, использующий в политике 10 ближайших действий (10NN Wolpertinger), так как согласно результатам [**?**] алгоритм, использующий 10% ближайших действий, наиболее сопоставим с использованием 100%.

В первую очередь, хочется отметить, что регуляризация положительно влияет на обучение, даже если актёр не влияет на работу алгоритма: это полезно для критика. Если мы будем рассматривать Wolpertinger,

использующий все $|\mathcal{A}| = 100$ ближайших действий, мы по сути перейдем к варианту Q-learning, так как выбор протодействий актёром не будет влиять на конечный выбор действия. Стоит отметить, что даже в таком случае у Wolpertinger может остаться преимущество обобщения значений критика на похожих действиях. Однако, это может и наоборот мешать корректно оценивать Q-функцию, если действия приводящие к совершенно разной награде, имеют близкие эмбеддинги. Поэтому выбор эмбеддингов действий важен. И для критика, и для актёра будем использовать коэффициент регуляризации, равный 0.001.

Как видим на рисунке 3, алгоритм не смог найти оптимальную политику, хотя и способен получать награду существенно более высокую, чем случайный алгоритм. Регуляризация помогает раньше получить более высокую награду, но со временем получаемые награды стали примерно равны.

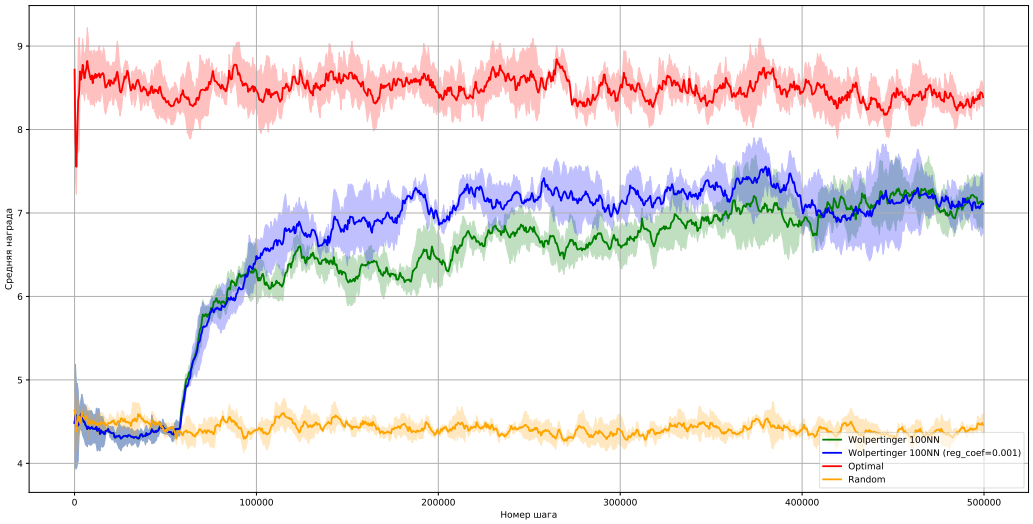

Рис. 3: Wolpertinger, использующий в политике $|\mathcal{A}| = 100$ ближайших действий

Гланый интерес представляет случай, когда актёр включается в работу алгоритма. Мы видим на рисунке 4, что алгоритм, не использующий l_2-регуляризацию или регуляризацию с помощью ближайшего соседа даёт результаты сильно хуже, чем 100NN Wolpertinger. Но в случае, если мы добавим l_2-регуляризацию, мы получим очень близкие графики. Если дополнительно добавить к функции потерь критика слагаемое из уравнения 1, мы получим синюю кривую, лежащую на графике выше всех остальных. Таким образом, получен алгоритм с достигающий награды даже большей, чем 100NN Wolpertinger.

Поясним, почему 10NN Wolpertinger с NN-регуляризацией может

иметь наилучшие результаты. Если взглянуть на значения растояний от протодействий до оптимальных действий

$$\delta_s = \|f_{\theta^\pi}(s) - a_{optimal}(s)\|_2,$$

получим следующие статистики:

	Без регуляризации	l_2	l_2 + NN
$\frac{1}{\mathcal{S}} \sum_{s \in \mathcal{S}} \delta_s$	1.962	1.927	1.164
$\min_{s \in \mathcal{S}} \delta_s$	1.598	1.675	0.484
$\max_{s \in \mathcal{S}} \delta_s$	2.146	2.278	1.788

Судя по третьему столбцу таблицы, вариант одновременной l_2 и NN-регуляризации помогает актёру выбирать протодействия существенно более близкие к оптимальным действиям. Возможно, что адекватный выбор протодействий вместе с максимизацией по меньшему числу соседей в политике позволяет в меньшей мере следовать неверным оценкам критика, которые имели место в случае 100NN Wolpertinger.

Минимальное и максимальное расстояния между парами используемых ембеддингов составляют 1.658 и 0.213 соответственно, откуда можно заключить, что первые два алгоритма из таблицы выдают крайне далёкие от множества $\mathcal{A}$ протодействия.

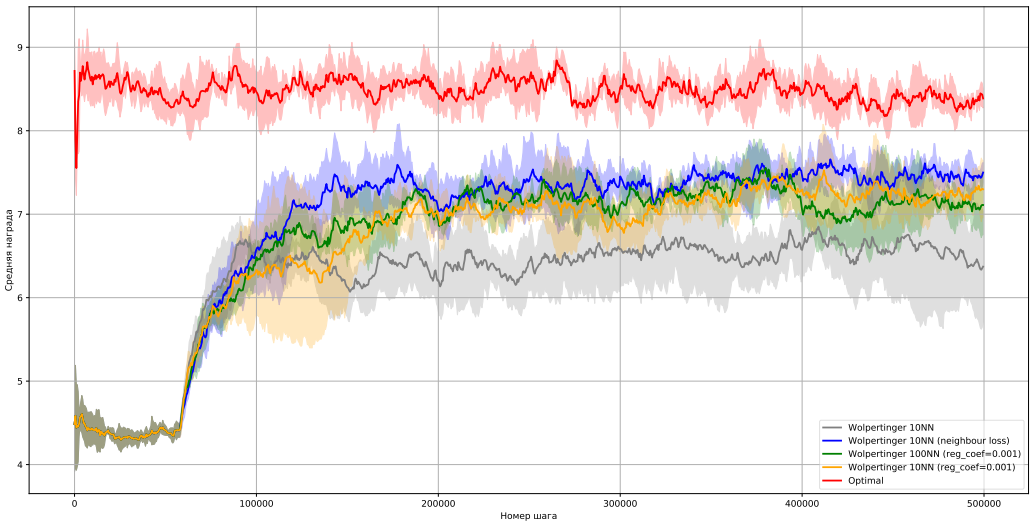

Рис. 4: Wolpertinger, использующий в политике 10 ближайших действий

Взглянем, как расположены протодействия, выбранные для некоторых случайных состояний. Рассмотрим проекции действий и протодействий на первые пять осей. Как видно по рисунку 5, версии алгоритма без NN-регуляризации склонны выбирать протодействия в сильно удаленных от $\mathcal{A}$ точках. Критик $Q_{\theta^Q}(s, \cdot)$ может иметь практически любые значения в точках, расположенных на большом расстоянии от $\mathcal{A}$, так как ни на что явно не обучается рядом с ними. Таким образом, актёр становится практически бесполезным и может в итоге просто всегда выбирать одну точку, далее выбор действия лишь как-то корректируется критиком, что не даёт выбирать совсем провальные действия.

В случае NN-регуляризации также видна тенденция к скучиванию точек и их удаленности, но в гораздо меньшей мере. Стоит при этом учитывать, что этот положительный эффект наблюдается по каждому измерению пространства эмбеддингов, из-за чего и сильно уменьшается среднее расстояние до оптимальных действий.

Не очень понятно, почему одна лишь l_2-регуляризация помогла, пусть и в меньшей степени, достичь большей награды. Как мы видим, на актёра это практически не подействовало. Видимо, это помогло лучшим образом обучить критика и выбирать лучшее из 10 ближайших действий.

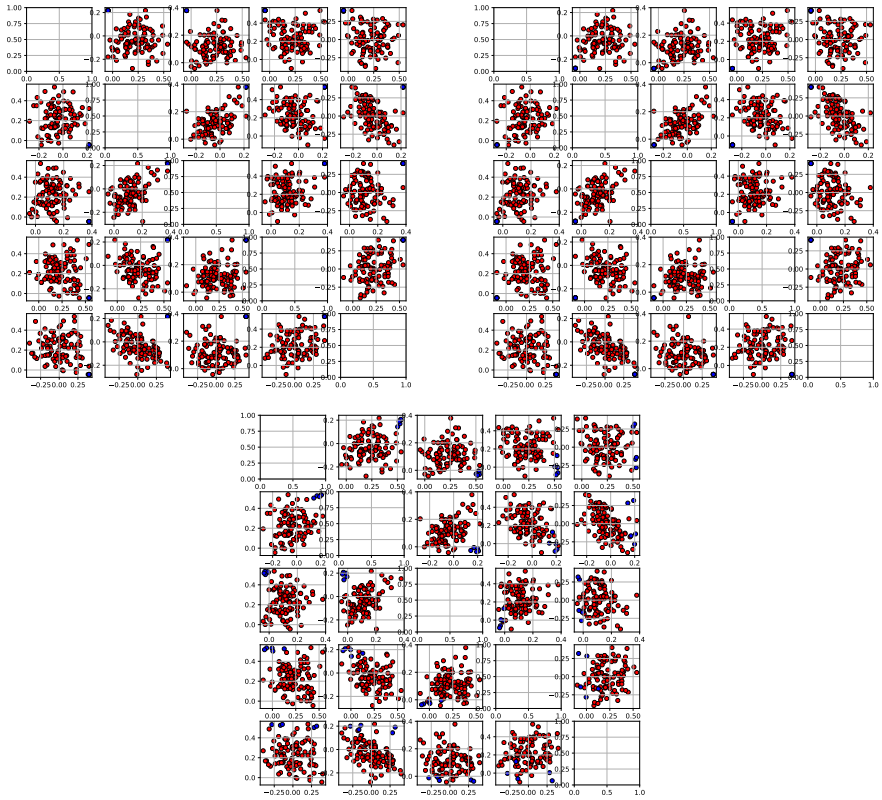

Рис. 5: Красным изображены проекции эмбеддингов на первые пять осей, синим – протодействия в 5 одинаковых для каждого алгоритма случайных состояниях

5 Заключение

В данной работе был воспроизведён эксперимент из [?]. Была построена среда, подобная среде из оригинального эксперимента. Было показано, что во время обучения может возникнуть проблема зацикливания актёра на нерелевантных протодействиях из-за оптимизации критика только на пространстве действий. Была предложена модификация функции потерь критика, которая способна исправить эту проблему.

Список литературы

[1] Tien T. Nguyen et al. Exploring the Filter Bubble: The Effect of Using Recommender Systems on Content Diversity. *WWW 2014 Proceedings*, pages 677–686, 2014.

[2] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement Learning with Deep Energy-Based Policies. *arXiv e-prints*, page arXiv:1702.08165, 2017.

[3] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *arXiv e-prints*, page arXiv:1812.05905, 2018.

[4] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep Reinforcement Learning in Large Discrete Action Spaces. *arXiv e-prints*, page arXiv:1512.07679, 2015.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints*, page arXiv:1301.3781, 2013.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1312.5602, 2013.

[7] Timothy P. Lillicrap, Jonathan J. Hunt, Alexand er Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv e-prints*, page arXiv:1509.02971, 2015.

[8] Barto Andrew and Sutton Richard. *Introduction to Reinforcement Learning*. The MIT Press, 2020.