ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

УДК 519.2

Отчет об исследовательском проекте на тему:

Тензорные бандиты и их приложения

Выполнил студент:

группы БПМИ212, 3 курса Горбач Марина Павловна

Принял руководитель проекта:

Самсонов Сергей Владимирович
Научный сотрудник, аспирант
Международная лаборатория стохастических алгоритмов и анализа многомерных данных, ФКН НИУ ВШЭ

Москва 2024

Содержание

Аннотация

Алгоритмы многоруких бандитов являются мощным теоретическим аппаратом, который позволяет эффективно решать множество практических задач, таких как задачи рекомендательных систем и динамического ценообразования. При этом множество существующих алгоритмов страдает от так называемого “проклятья размерности” - оценки сложности алгоритма в лучшем случае зависят линейно от количества действий (ручек), которых может быть очень много. Для того, чтобы преодолеть эту теоретическую проблему, недавно была предложена парадигма низкоранговых тензорных бандитов, основанная на предположении о низком ранге у тензора ожидаемых наград. В рамках проекта ожидается обобщить этот подход на новые виды тензорных разложений и проверить работоспособность метода на различных данных.

Ключевые слова

Алгоритм многоруких бандитов, низкоранговые приближения, разложение Таккера, тензорынй поезд, тензорные вычисления

1 Введение

В современном мире принятие решений в условиях неопределенности становится всё более важной и трудоёмкой задачей, так как пространство выбора, как и объёмы данных, постоянно растёт. Например, как описано в [7], у сервисов, предоставляющих персонализированный контент, постоянно возникает необходимость подобрать рекламу, новые предложения и т. д. на основании данных контента и самого пользователя. Подобные задачи осложняются тем, что постоянно появляется новый контент, новые пользователи, а также данные непрерывно меняются. Соответственно, большинство ранее распространённых способов, таких как коллаборативная фильтрация, стали непригодными к использованию. Чтобы удовлетворять современным требованиям, алгоритм должен быстро обучаться на новых данных и учитывать обновления, а также быстро принимать решения.

Одним из способов решения таких задач являются алгоритмы многоруких бандитов, так как они позволяют быстро реагировать на изменение среды, а также учитывать контекст. Алгоритмы многоруких бандитов – алгоритмы, в которых агенту необходимо сделать выбор, где каждый вариант представлен ручкой. При выборе каждая из ручек возвращает некоторую награду, максимизировать которую – цель агента. В данной работе речь пойдёт о тензорном представлении алгоритмов такого типа.

В контексте задачи о рекомендательной системе [18] это будет устроено следующим образом. Например, пользователю нужно порекомендовать музыку, тогда каждый выбор в алгоритме будет характеризоваться набором из жанра, исполнителя, названия произведения и других характеристик. Такой набор далее будет представлен тензором, что поможет в последствии оценить ожидаемую награду и определить оптимальный вариант. После его выбора будет получен ответ от среды: пользователь прослушает трек, добавит его в плейлист или же пропустит, таким образом алгоритм будет общаться со средой и, следовательно, обучаться. Работа с тензорами в данном случае помогает эффективно моделировать контекст, который обладает множеством возможных значений.

На данный момент существует множество алгоритмов для решения задачи многорукого бандита. Один из наиболее популярных – алгоритм верхних доверительных границ, так как он показывает хорошие результаты на широком спектре задач [2]. Однако, применять его для тензорного варианта задачи затруднительно, так как операция разложения тензора не является выпуклой [18].

Для решения этой проблемы была создана серия алгоритмов, использующих низкоранговые приближения, так как это существенно снижает сложность алгоритма. Данная

работа подразумевает исследование существующих методов, их дальнейшее их развитие для иных тензорных разложений.

Исследование было начато с реализации алгоритмов, использующих разложение Таккера, которые были описаны в [18]. В качестве базовой модели был реализован известный алгоритм Tensor Elimininations, идея которого в том, чтобы постепенно уменьшать вероятность выбора не оптимальных ручек на основании данных предыдущих ходов. Были проведены эксперименты на искусственно сгенерированных данных. В качестве дополнительного эксперимента была рассмотрена возможность модификации данных во время работы алгоритма. Следующим шагом стала реализация других уже существующих алгоритмов, основанных на разложении Таккера, таких как Epoch Greedy и Ensemble Sampling, последний для контекстуального случая. Далее был разработан новый алгоритм, использующий разложение в формат тензорного поезда [8], проведено сравнение с остальными алгоритмами, в то числе с алгоритмом верхних доверительных границ. После чего для всех перечисленных алгоритмов были реализованы и протестированы контекстуальные версии. Итогом работы стал запуск контекстуальных алгоритмов на датасете Open Bandit [11].

2 Обзор литературы

Тензорные разложения В качестве самого первого источника для знакомства с особенностями тензорных разложений использовались материалы курса «Тензорные вычисления» [19]. В них содержалась основная информация об операциях с тензорами и низкоранговых разложениях, в том числе о разложении Таккера (1) [15].

$$
\begin{aligned}
&\mathbf{X} \approx \mathbf{G} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \cdots \times_n \mathbf{U}_n, \\
&\text{где } \mathbf{X} \in \mathbb{R}^{p_1 \times p_2 \times \cdots \times p_n}, \mathbf{G} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_n}, \mathbf{U}_i \in \mathbb{R}^{p_i \times r_i}, \\
&\mathbf{A} \times_1 \mathbf{B} = \left(\sum_{j_1=1}^{p_1} A_{j_1, i_2, i_3, \ldots, i_n} B_{i_i, j_1} \right)_{i_1 \in [1, r_1], i_2 \in [1, p_2], \ldots, i_n \in [1, p_n]} \\
&\text{где } \mathbf{A} \in \mathbb{R}^{p_1 \times p_2 \times \cdots \times p_n}, \mathbf{B} \in \mathbb{R}^{p_1 \times r_1}
\end{aligned}
\tag{1}
$$

Дальнейшее исследование предполагало создание алгоритма, который будет использовать другие тензорные разложения. До этого в литературе не встречалось использования тензорного поезда [8] таким образом, в то время как многие операции, например поиск максимального элемента в тензоре и сложение двух тензоров, могут быть реализованы эффективно для такой структуры [3], [4]. Поэтому выбор пал на данное разложение (2).

Тензор $\mathbf{B} \in \mathbb{R}^{p_1 \times p_2 \times \cdots \times p_n}$ приближается тензором $\mathbf{A} \in \mathbb{R}^{p_1 \times p_2 \times \cdots \times p_n}$, где

$$\begin{aligned}
&\mathbf{A}(i_1, i_2, \ldots, i_n) = \mathbf{G}_1(i_1)\mathbf{G}_2(i_2)\ldots\mathbf{G}_n(i_n), \\
&\mathbf{G}_j(i_j) \in \mathbb{R}^{r_{j-1} \times r_j}, \quad r_0 = r_n = 1.
\end{aligned} \tag{2}$$

Это можно также записать в виде:

$$\begin{aligned}
&\mathbf{A}(i_1, i_2, \ldots, i_n) = \sum_{j_0, j_1, \ldots, j_n} \mathbf{G}_1(j_0, i_1, j_1)\mathbf{G}_2(j_1, i_2, j_2)\ldots\mathbf{G}_n(j_{d-1}, i_n, j_d), \\
&\mathbf{G}_j \in \mathbb{R}^{r_{j-1} \times p_j \times r_j}, \quad r_0 = r_n = 1.
\end{aligned}$$

Многие алгоритмы, рассматриваемые в данной работе использует операцию дополнения тензоров (tensor completion). Для знакомства сданной операцией использовалась статья [5]. Алгоритм подразумевает восстановление неизвестных элементов тензора на основании тех, что даны. Обычно это требуется для построения оценки тензора, моделирующего распределение наград по ручкам, на основе данных, полученных во время сделанных ранее шагов. Один из вариантов реализации операции дополнения также описан в [9].

Многорукие бандиты В последнее время алгоритмы многоруких бандитов развивались достаточно активно. Для знакомства с существующими алгоритмами и изучения основных идей использовались [6], [13], где описаны различные алгоритмы многоруких бандитов, в том числе стохастические и контекстуальные. Один из наиболее популярных на данный момент алгоритмов в силу своих хороших результатов – алгоритм верхних доверительных границ [1]. Его идея заключается в том, что для каждой ручки строится доверительный интервал, в котором ожидается её награда. Это делается на основании предыдущих шагов. В этом алгоритме решения принимаются исходя из принципа Оптимизма перед лицом неопределенности [6]. Он утверждает, что выбор должен делаться исходя из предположения, что среда будет настолько выгодной агенту, насколько это возможно. Поэтому данный алгоритм сравнивает ручки по верхним границам доверительных интервалов. Стоит также заметить, что по мере обучения алгоритма, он получает всё больше данных, что позволяет сужать доверительный интервал и, следовательно, делать оценки более точными.

Так как зачастую считать вышеописанный доверительный интервал достаточно проблематично, ещё один подход набирает популярность – алгоритм Томпсоноского сэмплирования [14], [10]. Идея данного алгоритма строится на предположении, что награда каждой ручки имеет некоторое распределение, неизвестное агенту, но которое не меняется со временем. Действуя в таком предположении, агент инициализирует для каждой ручки некоторое априорное распределение, которое обычно выбирается из общей информации о задаче. Да-

лее после каждого шага, параметры распределения обновляются. Утверждается, что после некоторого числа итераций будет получено достаточно точное приближение реального распределения, на основании которого уже можно будет сделать выбор оптимальной ручки.

Обратим внимание также на метод, который является неотъемлемой частью большинства алгоритмов многоруких бандитов – метод рандомизации. Так как агенту никогда не известно распределение награды для каждой ручки, то нельзя исключать возможность того, что данные, которые были по ней получены содержат сильное смещение. Поэтому, с некоторой вероятностью, даже зная оптимальную ручку, алгоритм будет выбирать произвольную. Это уменьшит вероятность того, что наилучший выбор был отсеян в начале из-за недостаточного количества данных.

Контекстуальные многорукие бандиты Алгоритмы многоруких бандитов, которые описаны выше, обучаются исключительно на данных получаемых из общения со средой, которые представлены величинами наград. Однако, в реальности обычно существует некая дополнительная информация о среде, которую агент не может учесть таким образом, однако, она может быть полезна. Алгоритмы контекстуальных бандитов учитывают такую информацию [16], [6]. Чаще всего это нужно для рекомендательных систем, где у контекста есть несколько различных признаков, например, возраст и пол пользователя. Данная работа исследует алгоритм Ensemble Sampling, описанный в [18]. Он позволяет оптимизировать стратегию, используя разложение Таккера, что особенно эффективно для низкорангового случая. Также рассматриваются контекстуальные адаптации прочих алгоритмов.

Тензорные бандиты Одним из способов задать контекст в задаче многоруких бандитов является тензор. Каждая ручка задаётся в виде набора значений признаков, который позже представляется в тензорном виде.

Для работы с тензорными бандитами были представлены следующие алгоритмы [18, 12, 17]. В первой статье на эту тему [18] представлено несколько алгоритмов, использующих разложения Таккера. Наилучшей оценкой на потери среди представленных алгоритмов обладает tensor elimination: $\overline{O}(p^{\frac{d}{2}} + p^{\frac{d-1}{2}} n^{\frac{1}{2}})$, где n – временной горизонт, d – размерность тензора наград, $p = \max(p_1, \ldots, p_d)$ – наибольшая размерность. Алгоритм заключается в постепенном вытеснении не оптимальных ручек из множества выборов на основании их наград за предыдущие шаги. В [12] данный алгоритм был обобщён на более общее представление ручек, полученная оценка потерь: $\overline{O}(p^{\lfloor \frac{d}{2} \rfloor \lfloor \frac{r}{2} \rfloor} n^{\frac{1}{2}})$, где r – мультилинейный ранг тензора наград. Затем было получено улучшение: $\overline{O}(p^2 r^{n-2} \sqrt{n})$ – алгоритм TOFU.

В ноябре 2023 года вышла новая статья о тензорных бандитах [17]. Она существенно расширяет возможности для моделирования связи между признаками. В [18] для этого использовался метод one-hot encoding. Полученный алгоритм G-LowTESTR показывает результаты: $\overline{O}\left(\frac{p^2\sqrt{\ell \cdot n}}{\sqrt{a}(1-\gamma)}\right)$, где γ – мера между определенными сингулярными числами тензора, задающего действие, a и ℓ – коэффициенты линейного преобразования.

3 Алгоритм Tensor Elimination

Исследование алгоритмов, использующих низкоранговые разложения началось с алгоритма Tensor Elimination 1, описанного в [18]. Его идея несколько напоминает алгоритм верхних доверительных границ (UCB1) [1], поскольку на каждом шаге часть ручек удаляется из множества на основании того, что оценка из награды не попадает в заданный доверительный интервал. Разделим алгоритм на несколько частей и обсудим каждую из них.

Algorithm 1 Tensor Elimination

input: dimensions $\in \mathbb{N}^n$ - размерность тензора наград, ranks $\in \mathbb{N}^n$ - размерность ядра в разложении Таккера, λ_1, λ_2 - параметры регуляризации, T - общее число шагов, T_e - число шагов для исследования среды, α - длина доверительного интервала
for t in range(T_e) **do** ▷ Фаза исследования среды
    Выбор произвольной ручки
    Обновление тензора усреднённых наград
end for
Операция дополнения тензора усреднённых наград
Разложение тензора усредненных наград по Таккеру: $\hat{G}, \hat{U}_1, \hat{U}_2, \ldots, \hat{U}_n$
Смена пространства для ручек
for t in range(T_e, T) **do** ▷ Фаза отбора ручек
    Выбор оптимальной ручки:

$$\text{arm} = \arg \max_{\text{arm} \in \text{all-arms}} (\|\text{arm}\|_{V_t^{-1}}),$$

$$\text{где } V_t = \text{diag}\left(\underbrace{\lambda_1, \ldots, \lambda_1}_{\left(\prod_{i=1}^{n} p_i - \prod_{i=1}^{n}(p_i - r_i)\right)}, \underbrace{\lambda_1, \ldots, \lambda_2}_{\prod_{i=1}^{n}(p_i - r_i)}\right) + \sum_{t=\text{update-each}\cdot h}^{\text{update-each}\cdot(h+1)} \text{arm}_t(\text{arm}_t)^T$$

    Обновление оценок
    if t % update-each == 0 **then**
        Удаление ручек, оценка наград которых не принадлежит заданному интервалу: 2
        Операция дополнения тензора усреднённых наград
        Разложение тензора усредненных наград по Таккеру: $\hat{G}, \hat{U}_1, \hat{U}_2, \ldots, \hat{U}_n$
        Смена пространства для ручек
    end if
end for

В алгоритмах многоруких бандитов некоторое количество первых шагов делается для получения информации о среде. Поскольку нужна информация о наибольшем возможном числе ручек, но в то же время, выбор должен быть равномерным, обычно прибегают к стратегии выбора произвольной ручки. Так происходит и в данном случае. Во время фазы исследования собирается информация и считается средняя награда по каждой ручке, а далее выполняется операция дополнения тензора, так как некоторые ручки могли быть не использованы - это первая оценка тензора наград.

$$\mathbf{Y} = \mathbf{X} \times_1 \left[\hat{\mathbf{U}}_1 \hat{\mathbf{U}}_{1\perp}\right] \times_2 \left[\hat{\mathbf{U}}_2 \hat{\mathbf{U}}_{2\perp}\right] \times_3 \cdots \times_n \left[\hat{\mathbf{U}}_n \hat{\mathbf{U}}_{n\perp}\right] \tag{3}$$

Далее, алгоритм предлагает сменить пространство, это действие авторы статьи называют "вращением"(3), которое позволяет далее работать с векторизацией оценки тензора наград, в которой первые $\prod_{i=1}^{n} p_i - \prod_{i=1}^{n} (p_i - r_i)$ элементов – ненулевые, а оставшиеся очень близки к нулю. Аналогичная операция производится с тензорами, представляющими ручки. При этом в [18] доказано, что подобные действия не меняют задачу и, помимо этого, сохраняют процесс вычисления ожидаемой награды путём произведение тензора наград и тензора ручки.

После этого запускается основной алгоритм. Каждые k (параметр алгоритма) шагов происходит обновление полученного разложения Таккера и всех вытекающих. В том числе оценка тензора наград происходит с помощью решения задачи линейной регрессии с регуляризацией. После чего на её основе вычисляется ожидаемая награда, которая влияет на удаление или же сохранение ручки в множестве активных. Между этими обновлениями, каждый раз выбирается ручка, оценка дисперсии у которой максимальна, таким образом алгоритм позволяет уточнять имеющиеся данные о среде.

Авторы алгоритма приводят в [18] оценку регрета данного алгоритма: $\widetilde{O}(p^{n/2} + p^{(n-1)/2} T^{1/2})$, где $p = \max(p_1, p_2, \ldots, p_n)$. Это делает его одним из наименее зависимых от роста размерности алгоритмов, как и одним из наиболее эффективных.

Авторы статьи представили свою реализацию данного алгоритма для размерности тензора наград равной 3. В данной работе алгоритм был реализован для произвольной размерности, протестирован несколькими способами (на искусственно сгенерированных данных, имеющих и не имеющих низкоранговой структуры), было проведено сравнение с другими аналогичными алгоритмами. Об этом речь пойдёт в следующих главах.

Algorithm 2 Tensor Elimination: update arms set

1: Строим оценку вектора наград, основываясь на полученных ранее наградах и низкоранговой структуре: оценка делается в векторизованном виде. Пусть $m = \prod_{i=1}^{n} p_i - \prod_{i=1}^{n} (p_i - r_i)$
▷ Решаем задачу регрессии с регуляризацией

$$\text{rewards-tensor} = \arg \min_{rt \in \mathbb{R}^{p_1 \times \cdots \times p_n}} \sum_{t=1}^{\text{current-step}} \langle \text{arm}_t, rt \rangle + \lambda_1 \|rt_{:m}\|^2 + \lambda_2 \|rt_{m:}\|^2$$

2: С помощью полученной оценки вычисляем ожидаемую награду для каждой ручки:

$$\text{reward}_{\text{arm}} = \langle \text{arm}, \text{rewards-tensor} \rangle + \|\text{arm}\|_{V_t^{-1}} \alpha$$

3: Находим наибольшую из нижних границ:

$$\text{bound} = \max_{\text{arm} \in \text{all-arms}} \langle \text{arm}, \text{rewards-tensor} \rangle - \|\text{arm}\|_{V_t^{-1}} \alpha$$

4: Оставляем лишь ручки, для которых верно: $\text{reward}_{\text{arm}} \geq \text{bound}$

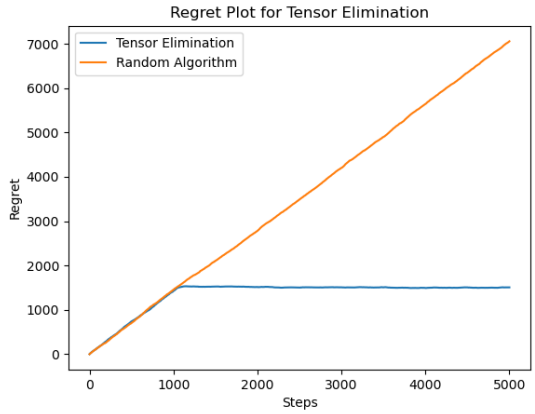
4 Алгоритм Epoch Greedy

Следующим алгоритмом стал Epoch Greedy, который представляет собой жадный алгоритм, действующий на основе низкоранговой оценки тензора наград с помощью разложения Таккера.

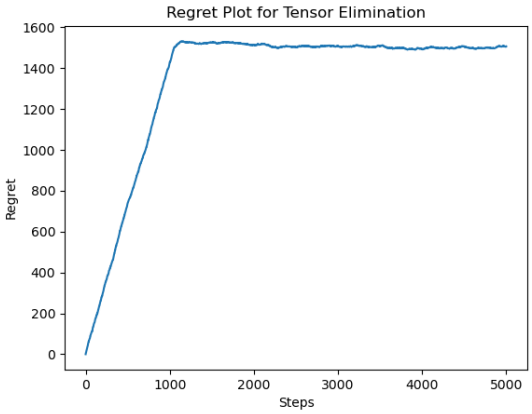
Как и Tensor Elimination, данный алгоритм 3 начинается с фазы исследования среды, во время которой формируется тензор из средних наград, которую сменяет жадный алгоритм. На каждом шаге строится низкоранговая оценка тензора наград с помощью разложения Таккера, максимальное значение в которой позволяет выбрать оптимальную ручку для текущего хода.

Algorithm 3 Epoch Greedy

1: **input:** dimensions $\in \mathbb{N}^n$ - размерность тензора наград, ranks $\in \mathbb{N}^n$ - размерность ядра в разложении Таккера, T - общее число шагов, T_e - число шагов для исследования среды
2: **for** t in range(T_e) **do** ▷ Фаза исследования среды
3: Выбор произвольной ручки
4: Обновление тензора усреднённых наград
5: **end for**
6: Операция дополнения тензора усреднённых наград
7: Разложение тензора усредненных наград по Таккеру: $\hat{G}, \hat{U}_1, \hat{U}_2, \ldots, \hat{U}_n$
8: Получение низкоранговой оценки тензора наград с помощью Таккера
9: **for** t in range(T_e, T) **do** ▷ Фаза эксплуатации
10: Выбор оптимальной ручки: индекс максимального элемента в оценке тензора наград
11: Операция дополнения тензора усреднённых наград
12: Обновление разложения Таккера и оценки тензора наград
13: **end for**



Tensor Elimination и Random



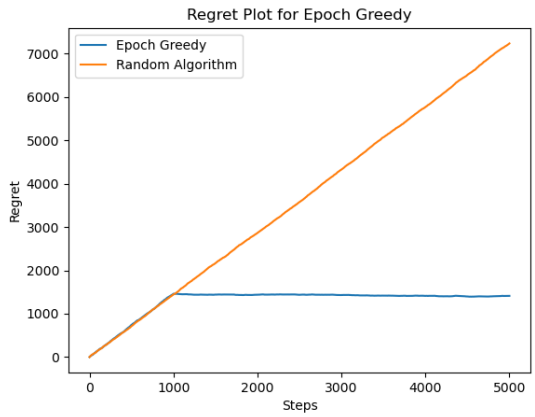
Регрет Tensor Elimination

Рис. 3.1: Эксперименты проводились на одном и том же тензоре наград размерности $3 \times 3 \times 3$, длина фазы исследований 1000, всего было сделано 5000 шагов

Интересной особенностью данного алгоритма является то, что авторы предлагают не разделять фазы исследования и эксплуатации, а время от времени делать шаг произвольной ручкой. Это уменьшает вероятность пропустить наиболее оптимальный вариант, выбрав до этого ручку, которая близка к наилучшей, но всё же приносит меньше награды.

Согласно [18] регрет данного алгоритма можно оценить как $\widetilde{O}(p^{n/2} + p^{(n+1)/3}T^{2/3})$, где $p = \max(p_1, p_2, \ldots, p_n)$. Исходя из этого можно заключить, что его результаты обычно несколько хуже, чем у Tensor Elimination. Как далее будет видно в экспериментах, это действительно так.

Алгоритм тестировался аналогично Tensor Elimination, результаты можно увидеть на графике 4.1. Далее в работе будет произведено сравнение алгоритмов.



Epoch Greedy и Random

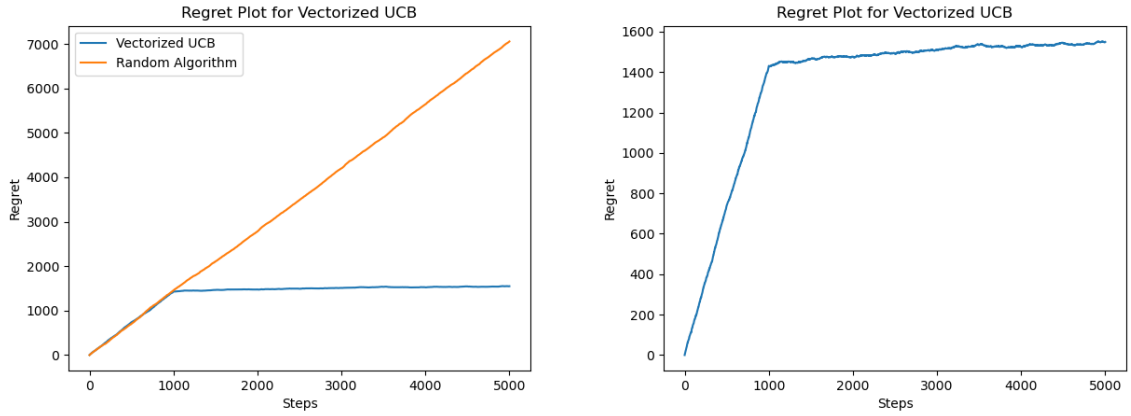
Регрет Epoch Greedy

Рис. 4.1: Эксперименты проводились на одном и том же тензоре наград размерности $3 \times 3 \times 3$, длина фазы исследований 1000, всего было сделано 5000 шагов

5 Алгоритм Vectorized UCB

Далее речь пойдёт о алгоритме, который является одним из наиболее популярных для решения задачи многорукого бандита в одномерном случае – об алгоритме верхних доверительных границ [1]. Как уже упоминалось ранее, использовать его эффективно для задачи тензорного бандита пока не удавалось, однако, существует подход, позволяющий его применить. Для этого нужно привести тензорную задачу к одномерной, представив тензор $X \in \mathbb{R}^{p_1 \times \cdots \times p_n}$ в виде вектора $Y \in \mathbb{R}^{p_1 \cdots p_n}$. После применения этой операции к изначальным данным классический алгоритм верхних доверительных границ может быть запущен.

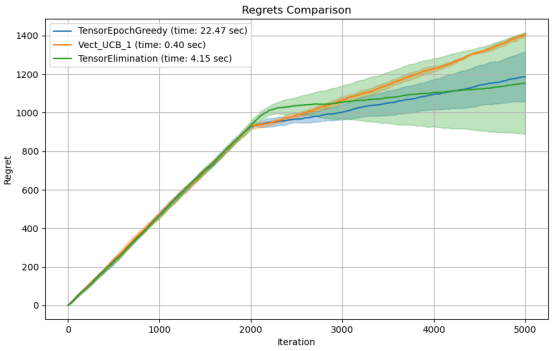
Алгоритм заключается в том, что на каждом шаге для всех ручек вычисляется величина $X(i) + \sqrt{\frac{2 \ln t}{t_i}}$, где X - вектор текущих значений средних наград ручек, t - текущий шаг, а t_i - количество шагов, которые сделала ручка номер i до сих пор. Когда все значения найдены, выбирается ручка с максимальным, что вытекает из принципа Оптимизма перед лицом неопределенности.



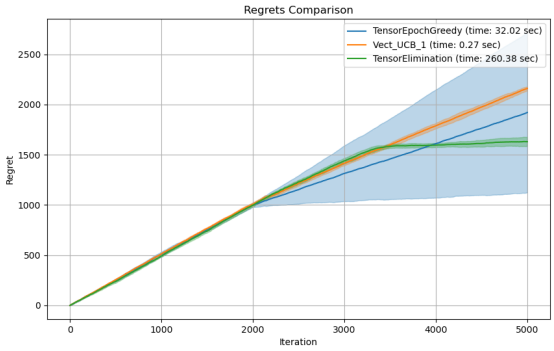
Vectorized UCB и Random Регрет Vectorized UCB

Рис. 5.1: Эксперименты проводились на одном и том же тензоре наград размерности $3 \times 3 \times 3$, длина фазы исследований 1000, всего было сделано 5000 шагов

Тестирование алгоритма проводилось так же, как и в предыдущих главах, результаты проиллюстрированы на графике 5.1. Согласно [18] регрет данного алгоритма оценивается как $\widetilde{O}(p^n + p^{n/2} T^{1/2})$, где $p = \max(p_1, p_2, \ldots, p_n)$. То есть, данный алгоритм действительно демонстрирует результаты хуже, чем тензорные алгоритмы, однако, на небольших размерностях это не слишком заметно, поэтому сравнение было проведено для больших размерностей (5.2).



Размерность тензора наград $5 \times 5 \times 5$



Размерность тензора наград $10 \times 10 \times 10$

Рис. 5.2: Эксперименты проводились следующим образом: 5 раз генерировался произвольный тензор наград заданной размерности, на котором запускался каждый из алгоритмов, после чего было проведено усреднение

6 Обновление данных во время работы алгоритма

После реализации первых трёх алгоритмов возникла идея провести эксперимент с модификацией данных (например, обновлением значений настоящего вектора наград, или же удалением ручек). Это может быть полезно при использовании алгоритмов в реальности, поскольку часто возникает необходимость удалить недоступные более варианты, а также среда периодически меняется (для алгоритма это значит изменение тензора наград). Поэтому, для лучшего из имеющихся алгоритмов Tensor Elimination и для алгоритма, который часто применяют на практике Vectorized UCB были реализованы операции по обновлению наград и удалению ручек, проведены эксперименты, в том числе сравнение с алгоритмом, который придерживается стратегии по выбору произвольной ручки на каждом шаге. Рассмотрим каждую из реализаций.

Начнём с алгоритма Tensor Elimination. В данном алгоритме несложно сделать удаление ручек, поскольку на каждом шаге в нем поддерживается множество доступных вариантов. Достаточно лишь удалить нужную ручку из данного множества, если она ещё в нём. Что касается обновления, ручка просто возвращается в это множество, если её там уже нет. В качестве улучшения данной операции можно рассмотреть следующий вариант. Сразу после обновления ручки, можно сыграть её несколько раз, чтобы обновить устаревшую информацию.

В алгоритме Vectorized UCB идея ещё проще. При удалении требовалось лишь установить наименьшее возможное значение для этой ручки в вектор средних наград, это позволяло убедится, что её больше не будут играть. А при обновлении, средняя награда устанавливалась равной 0, а также индикатор информации устанавливался в позицию «нет информации».

Это заставляло алгоритм исследовать данную опцию заново.

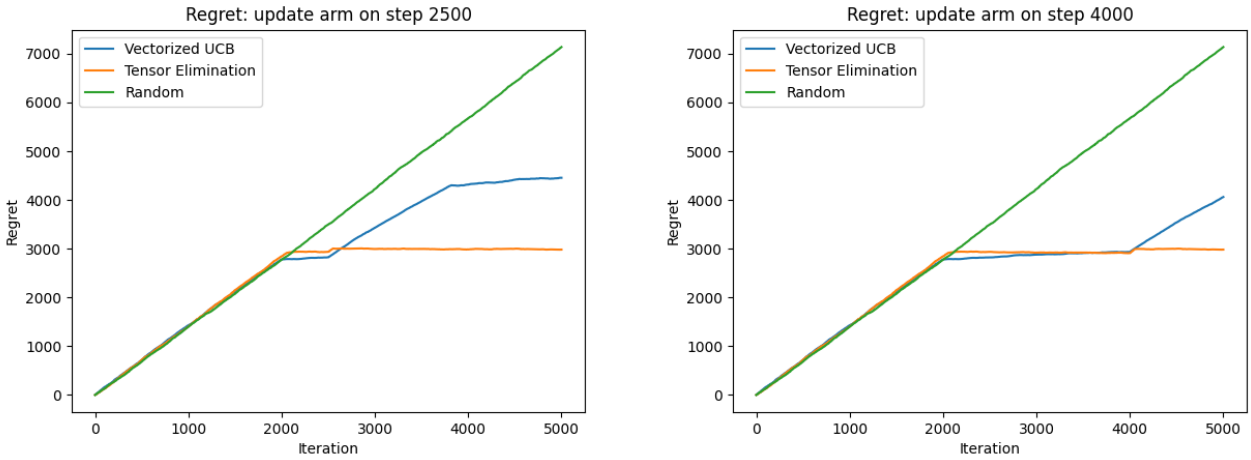


Рис. 6.1: На шагах номер 2500 и 4000 значение наград одной из ручек в среде было изменено. Все алгоритмы запускались на одинаковом тензоре наград размерности $3 \times 3 \times 3$, также обновление ручки было одинаковым для всех алгоритмов.

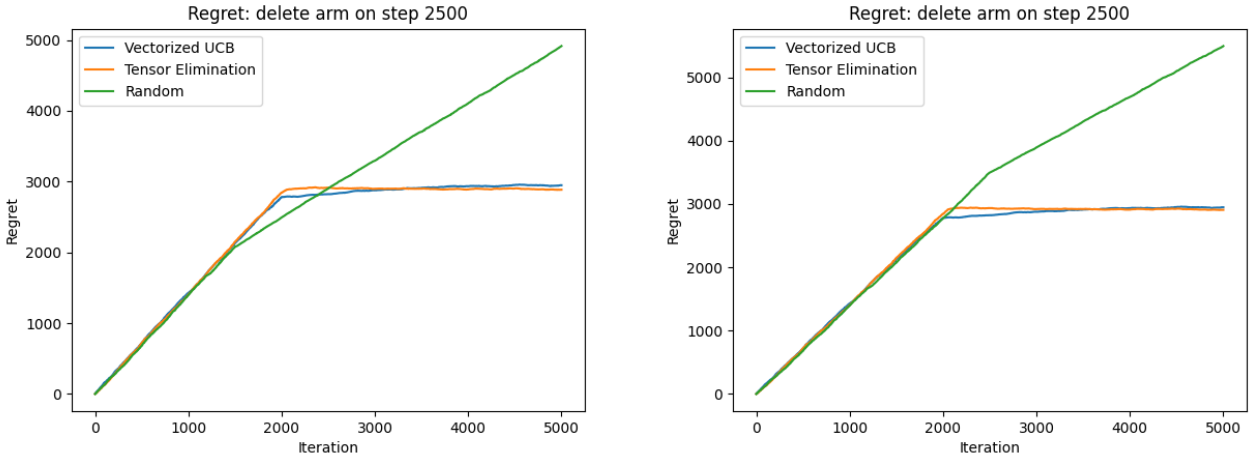


Рис. 6.2: На шагах номер 1500 и 2500 одна из ручек (одинаковая для всех алгоритмов) была удалена. Все алгоритмы запускались на одинаковом тензоре наград размерности $3 \times 3 \times 3$

На графике 6.1 можно увидеть, как алгоритмы реагируют на обновление данных о ручке. Можно заметить, что Tensor Elimination справляется с изменением с гораздо меньшими потерями, чем Vectorized UCB. График 6.2 отражает изменения при удалении ручки. Конечно, подобные результаты зависят от того, какая именно ручка удаляется и в какой момент. Например, если алгоритм уже успел определить множество наиболее оптимальных ручек, то изменение одной из не оптимальных никак не отразится на графике регретов, и в целом, практически не затронет работу алгоритма. В то время как изменение одной из оптимальных ручек может существенно увеличить регрет следующих шагов.

Цель этого эксперимента была в том, чтобы показать, что алгоритмы умеют справ-

ляться с подобными изменениями с не слишком большой потерей качества, так как это важно для реальных задач, например, рекомендательных систем. Исходя из полученных результатов, можно заключить, что Tensor Elimination отлично справляется с такими условиями.

Также был рассмотрен вопрос добавления новой ручки, однако, это достаточно затруднительно реализовать, поскольку это требовало бы изменения размерности тензора. Чтобы этого избежать, можно заранее создать тензор большей размерности, чем требуется, но не играть ручки, которые являются фиктивными, и позже их добавить.

7 Алгоритм TensorTrain

Следующим шагом исследования стала разработка своего алгоритма. Как и Tensor Elimination он использует низкоранговое разложение, только в этот раз это тензорный поезд [8], также используется операция дополнения тензора [5]. Мотивацией использования именно этого тензорного разложение стало то, что подобная структура позволяет находить максимальный элемент (с хорошей точностью), не перебирая все варианты [3]. Важную роль также сыграло то, что операция обновления одного элемента тензора также может осуществляться путём обновления разложения без восстановления исходного тензора.

Algorithm 4 TensorTrain Algorithm

input: dimensions $\in \mathbb{N}^n$ - размерность тензора наград, ranks $\in \mathbb{N}^n$ - размерность для разложения в тензорный поезд, T - общее число шагов, T_e - число шагов для исследования среды
for t in range(T_e) **do** ▷ Фаза исследования среды
    Выбор произвольной ручки
    Обновление тензора усреднённых наград
end for
Операция дополнения тензора усреднённых наград
Разложение тензора усредненных наград в тензорный поезд
for t in range(T_e, T) **do** ▷ Фаза эксплуатации
    Выбор оптимальной ручки путём нахождения максимума текущей оценки вектора наград с помощью алгоритма `optima_tt_max`, описанного в [3]
    Обновление оценки тензора наград
    if t % update-each == 0 **then**
        Восстановление тензора наград из разложения
        Операция дополнения тензора усреднённых наград
        Разложение тензора усредненных наград в тензорный поезд
    end if
end for

Рассмотрим подробнее реализацию 4. Как и обычно, алгоритм начинается с фазы сбора информации о среде, во время которой формирует тензор усредненных наград, по-

сле чего выполняет для него операцию дополнения тензора [5]. Далее, полученный тензор с информацией о среде раскладывается в тензорный поезд. Здесь начинается основная фаза алгоритма. На каждом шаге встаёт вопрос поиска оптимальной ручки, что делается с помощью алгоритма `optima_tt_max` [3]. Здесь важно отметить, что алгоритм может работать некорректно при наличии отрицательных наград в задаче, поскольку рассматриваемый алгоритм поиска максимума, на самом деле, ищет максимальный по модулю элемент. В целом, это можно исправить, передав предварительно награды в отображение, которое сделает значения положительными, не нарушив их порядок. Идея же самого алгоритма в том, чтобы представить элементы тензора, как элементы некоторого распределения. Тогда на каждом шаге, алгоритм будет выбирать k наиболее вероятных элементов, и таким образом, в конце получится наилучший вариант. Асимптотика подобной операции оценивается как $O(nkR^2P)$, где $P = \max(p_1, p_2, \ldots, p_n)$, $R = \max(r_1, r_2, \ldots, r_n)$, k - параметр алгоритма, отвечающий за то, сколько наиболее вероятных элементов выбирается каждый шаг, в TensorTrain используется не более 5 (в зависимости от размерности тензора наград). Однако, у этого алгоритма есть особенность, он требует, чтобы размерности ядер по первой и последней оси в тензорном поезде были меньше изначальных размерностей. Далее станет понятно, что при обновлении очередного элемента тензора эти размерности растут, так что в какой-то момент это условие перестанет выполняться. Поэтому, перед данной операцией периодически будет происходить дополнительное перестраивание тензорного поезда. Важно заметить, что чем больше размерность тензора наград, тем реже это будет происходить.

Далее, после того, как была сыграна ручка, наступает момент обновления разложения. Для этого есть формула модификации ядер при сложении двух тензорных поездов (тензор награды для конкретной ручки также раскладывается в тензорный поезд).

Пусть два тензора, разложенных в тензорный поезд $A, B \in \mathbb{R}^{p_1 \times \cdots \times p_n}$, их ядра обозначим так $C_{A,i} \in \mathbb{R}^{r_{A,(i-1)} \times p_i \times r_{A,i}}$ и $C_{B,i} \in \mathbb{R}^{r_{B,(i-1)} \times p_i \times r_{B,i}}$, тогда ядра $C_{D,0} \in \mathbb{R}^{1 \times p_1 \times (r_{A,1} + r_{B,1})}$, $C_{D,i} \in \mathbb{R}^{(r_{A,(i-1)} + r_{B,(i-1)}) \times p_i \times (r_{A,i} + r_{B,i})}$, $C_{D,n} \in \mathbb{R}^{(r_{A,(n-1)} + r_{B,(n-1)}) \times p_n \times 1}$ для разложения суммы $D = A + B$ можно определить следующим образом.

$$
\begin{aligned}
C_{D,0} &= [A_{A,0} \quad C_{B,0}], \\
C_{D,i} &= \begin{bmatrix} C_{A,i} & 0 \\ 0 & C_{B,i} \end{bmatrix} \quad i = 1, \ldots, n-1, \\
C_{D,n} &= [C_{A,n} \quad C_{B,n}].
\end{aligned}
$$

Сложность такой операции выражается как $\mathcal{O}(nR^2P)$, где $P = \max(p_1, p_2, \ldots, p_n)$,

$R = \max(r_{A1}, \ldots, r_{An}, r_{B1}, \ldots, r_{Bn})$.

Получается, что подобная операция постоянно увеличивает размерности ядер, которые влияют на сложность нахождения максимального элемента, поэтому периодически нужно обновлять разложение, возвращая его к исходным рангам. В том числе, как уже говорилось ранее, будут происходить дополнительные обновления, когда это необходимо.

Теперь оценим асимптотику полученного алгоритма. Для упрощения расчетов будем считать, что обновление разложения происходит каждый шаг, так как это не повлияет на асимптотику (но в реальности, обновление происходит реже, что уменьшает константу). Само разложение вычисляется за $\mathcal{O}(nPR + (n-3)R^3)$, где $P = \max(p_1, p_2, \ldots, p_n)$, $R = \max(r_1, \ldots, r_n)$. Есть 2 варианта развития событий при обновлении разложения. Так как тензор средних наград в любом случае будет создан для фазы инициализации, можно поддерживать его и далее, тогда при обновлении, достаточно разложить его в тензорный поезд. Если же не поддерживать его, то нужно также учесть операцию восстановления тензора. Посчитаем сложность более для первого случая. Получается, что сложностью каждого шага будет равна: $O(nR^2P)$. Тогда сложность всего алгоритма $O(nTR^2P)$.

Посмотрим на результаты тестирования при различных тензорах наград на графиках 7.1 и 7.2. Первое, что можно заметить, что время работы TensorTrain алгоритма гораздо меньше, чем у Tensor Elimination, и что в увеличением размерности, разница растёт. А вот по качеству TensorTrain всё же уступает Tensor Elimination в большинстве случаев, однако не слишком сильно. Более того, в большинстве случаев он работает лучше остальных алгоритмов.

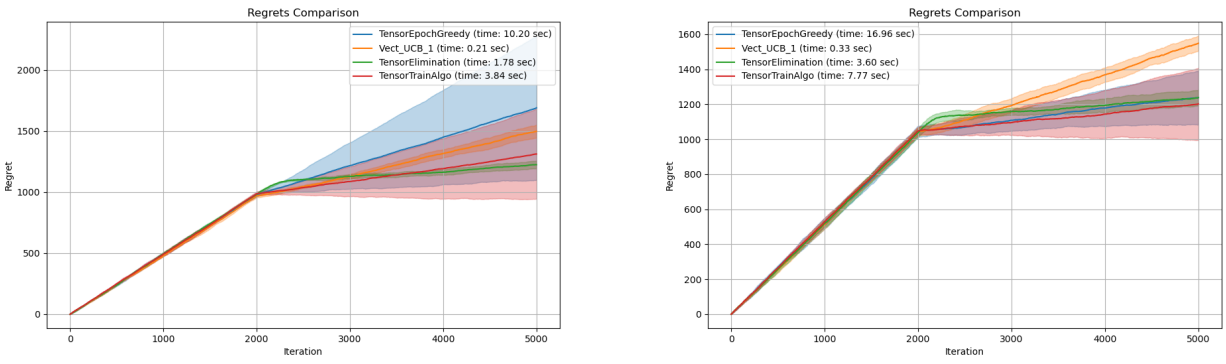


Рис. 7.1: Производилось 5 запусков на сгенерированных из нормального распределения тензорах наград размера $5 \times 5 \times 5$, после чего происходило усреднение регрета на каждом шаге и времени работы алгоритма

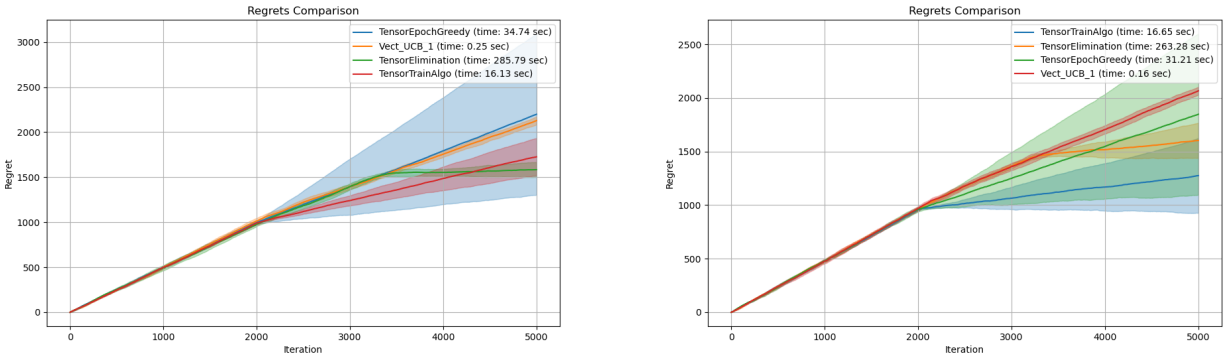


Рис. 7.2: Производилось 5 запусков на сгенерированных из нормального распределения тензорах наград размера $10 \times 10 \times 10$, после чего происходило усреднение регрета на каждом шаге и времени работы алгоритма

8 Контекстуальные бандиты

Ранее было предложено множество вариантов решения задачи тензорных бандитов, однако при построении рекомендательной системы часто есть дополнительные данные, которые касаются каждого варианта, так называемый контекст, который подобные алгоритмы никак не учитывают. Для таких случаев используются контекстуальные бандиты. Такую задачу можно представить и как тензорную, для этого нужно добавить в тензор наград дополнительные размерности, которые будут кодировать контекст. В общем виде алгоритм будет выглядеть следующим образом. Сначала алгоритм получает контекст, выбирает часть тензора наград, соответствующую ему, и выбирает оптимальную ручку с помощью некоторого тензорного алгоритма.

Первым контекстуальным алгоритмом в данном исследовании стал Ensemble Sampling, предложенный в [18]. Свою идею он наследует от алгоритма Томпсоновского сэмплирования, а также использует разложение Таккера, как и прочие алгоритмы из [18]. Рассмотрим его подробнее 5. Изначально алгоритму требуется априорное распределение для так называемых моделей, каждая из которых является некоторой оценкой разложения Таккера тензора наград, поэтому первым шагом происходит сэмплирование $U_{m,0}, U_{m,1}, \ldots, U_{m,n}$, где m - номер модели. Ядро Таккера задаётся тензором из всех единиц. После этого следует фаза исследования среды, в течение которой сохраняется история ручек и соответствующих им наград для каждой из моделей.

Далее начинается основная фаза алгоритма. Чтобы сделать шаг, сначала сэмплируется номер модели. Затем происходит обновление параметров модели с помощью собранной на предыдущих шагах информации, что позволяет далее вычислить наиболее актуальную оценку тензора наград для этой модели. После того, как алгоритм получает контекст (для

тестирования он сэмплировался случайным образом), происходит поиск оптимальной ручки. Для этого находится максимальный элемент в той части тензора наград, которая соответствует заданному контексту. Среда отвечает наградой на выбор ручки, но к ней ещё прибавляется некоторый шум из нормального распределения, что помогает моделям быть более разнообразными.

Algorithm 5 Ensemble Sampling

input: dimensions $\in \mathbb{N}^n$ - размерность тензора наград, ranks $\in \mathbb{N}^n$ - размерность ядра в разложении Таккера, M - количество моделей, μ_i, σ_i - априорные параметры для каждой из моделей. T - общее число шагов, T_e - число шагов для исследования среды, α - длина доверительного интервала
Инициализируем каждую модель из заданного априорного распределения
for t in range(T_e) **do** ▷ Фаза исследования среды
    Выбор произвольной ручки
    Сохранение шага в историю
end for
for t in range(T_e, T) **do** ▷ Фаза отбора ручек
    Сэмплируем номер модели m из равномерное распределения
    Обновление параметров модели: решаем оптимизационную задачу (формулы обновления предложены в [18])

$$\min_{S,U_{m,1},\dots,U_{m,n}} \frac{1}{\sigma^2} \sum_{s=1}^{t} (\text{noisy reward} - \langle S_m \times_1 U_{m,1} \times_2 \cdots \times_n U_{m,n}, arm \rangle)^2 +$$

$$+ \sum_{k=1}^{n} \frac{1}{\sigma^2} \sum_{i=1}^{p_k} \| [U_k]_i - [U_{m,k}]_i^{\text{start}} \|$$

    Получаем контекст для данного шага: context $= (i_1, i_2, \dots, i_c)$
    Выбор оптимальной ручки:

$$\text{arm} = \arg \max_{\text{arm} \in \text{all arms with fixed context}} (\hat{R}_m(\text{context}, \text{arm}), \text{где}$$

$$\hat{R}_m = S_m \times_1 U_{m,1} \times_2 \cdots \times_n U_{m,n}$$

    Добавление шума к полученной награде
    Обновление истории наград
end for

Изначально алгоритм тестировался на небольших размерностях, результаты можно увидеть на графике 8.1. Можно заметить, что он достаточно хорошо справляется с задачей. Алгоритм не всегда справляется с поиском глобального максимума для каждого из контекстов, однако, в среднем выбирает достаточно хорошие опции.

После тестирования данного алгоритма, были реализованы контекстуальные версии каждого из ранее рассмотренных тензорных алгоритмов, кроме Vectorized UCB, согласно идее описанной в начале главы. Тут стоит выделить подход для поиска оптимальной ручки

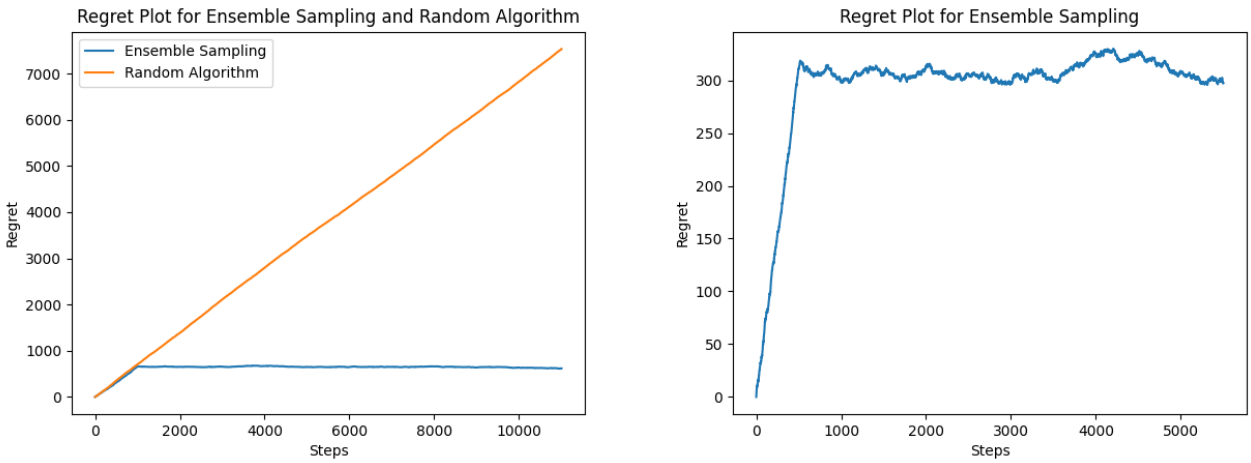


Рис. 8.1: Тестирование алгоритма Ensemble Sampling на тензоре наград размера $3 \times 3 \times 3$, где первая размерность соответствует контексту

для заданного контекста в алгоритме TensorTrain. Поскольку восстанавливать исходный тензор наград неэффективно, нужно получить тензорный поезд для той части, что отвечает за заданный контекст. Было решено делать это с помощью поэлементного произведения, которое можно сделать в формате тензорного поезда [4]. Тензор наград поэлементно умножается на нулевой тензор, где на позициях, соответствующих контексту, стоят 1. После чего оптимальная ручка, как и раньше, находится с помощью алгоритма `optima_tt_max`, описанного в [3]. Так как все неподходящие элементы были обнулены, а случай отрицательных наград не рассматривается, то максимум будет обязательно соответствовать контексту.

На графике 8.2 отображено сравнение алгоритмов на достаточно большой размерности. Видно, что Ensemble Sampling работает лучше остальных с достаточно сильным отрывом, однако, время его работы также существенно больше, чем у остальных алгоритмов. На втором месте по регрету находится Tensor Train, но остальные алгоритмы также очень близки к нему.

После тестирования алгоритмов на искусственно сгенерированных данных, возникла идея протестировать их на известном датасете Open Bandit [11]. Данная библиотека предоставляет несколько вариантов симуляторов, которые позволяют обучать и тестировать контекстуальные алгоритмы. На графике 8.3 приведены результаты запусков на симуляторе `Synthetic Bandit Dataset`.

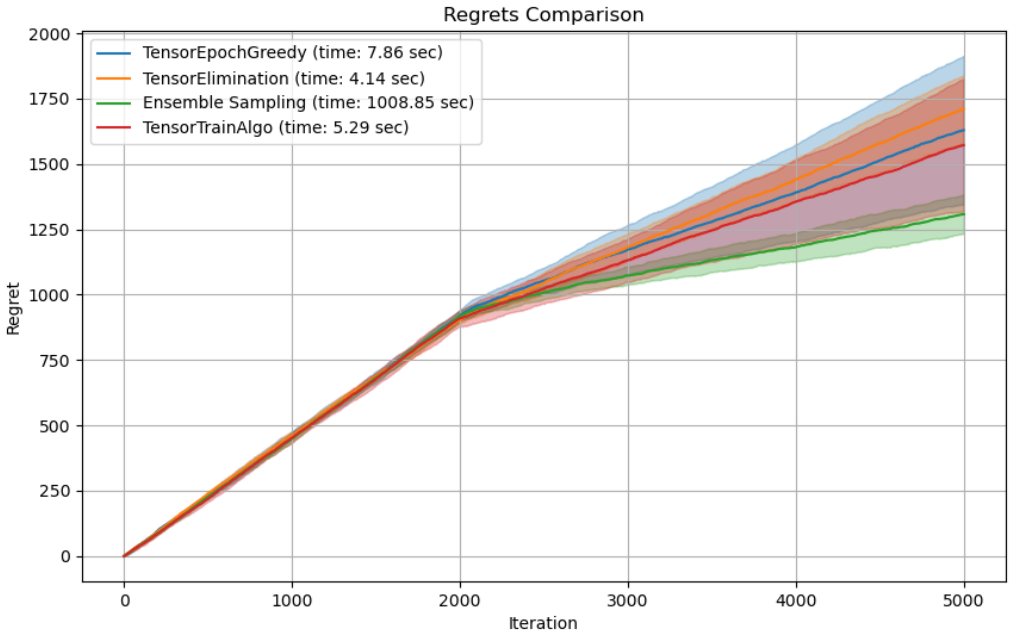


Рис. 8.2: Алгоритмы запускались 5 раз на тензорах наград $10 \times 10 \times 10$, где первая размерность соответствовала контексту, после чего происходило усреднение регретов

9 Заключение

В данной работы были исследованы, реализованы и протестированы различные ранее известные алгоритмы для решения задачи тензорного многорукого бандита. Полученная реализация алгоритмов поддерживает произвольные размерности тензоров. Следующим шагом было создание нового алгоритма TensorTrain для решения подобной задачи с помощью низкорангового разложения в тензорный поезд. По результатам экспериментов, этот алгоритм показал себя как один из наиболее быстрых среди существующих, в то время как по качеству он уступает некоторым, в том числе алгоритму Tensor Elimination.

Далее было проведено исследование по изменению данных во время работы алгоритма. Это полезно для реальных задач, поскольку опции в рекомендательных системах часто обновляются. Было показано, что алгоритмы достаточно быстро справляются с изменениями в среде, и регрет выходит на плато.

На данный момент наиболее перспективным направлением, где используются подобные алгоритмы, являются рекомендательные системы, в которых обычно доступны дополнительные данные о каждой из ручек. Поэтому следующим шагом работы было исследование контекстуальных бандитов. Были реализованы уже известные алгоритмы для произвольной размерности, а также произведено обобщение алгоритма TensotTrain для контекстуального случая. Были проведены эксперименты на различных данных, в том числе на сгенерирован-

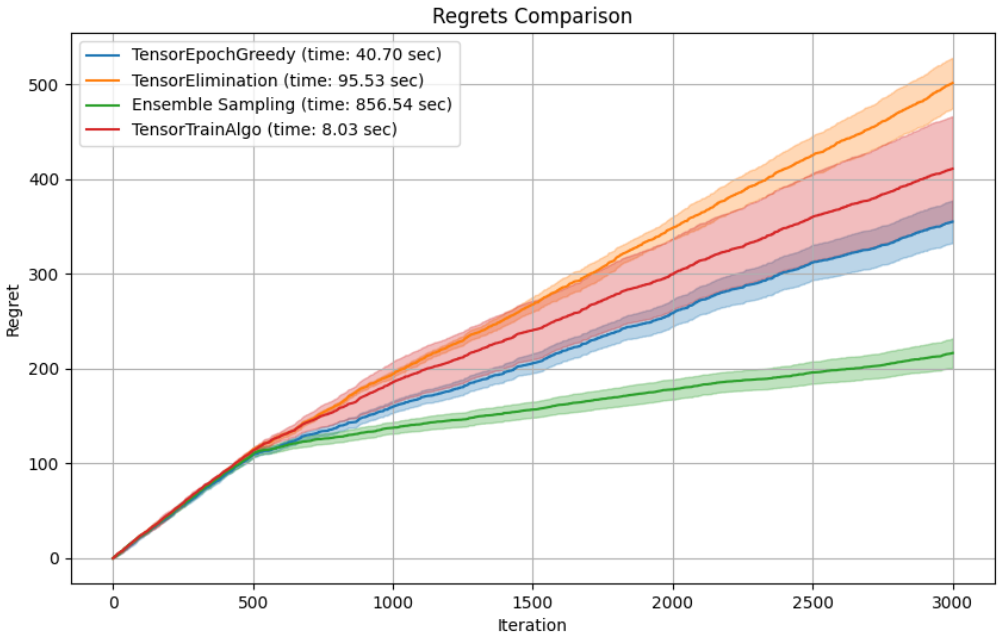


Рис. 8.3: Алгоритмы запускались 5 раз на на симуляторе `SyntheticBanditDataset`, где размерность контекста равна 3, а количество ручек 5, после чего происходило усреднение регретов

ных искусственно, а также на симуляторе из библиотеки Open Bandit [11].

По результатам тестирования, алгоритмы показали, что использование контекста улучшает показатели регрета, что говорит о более качественной рекомендации. Наилучший результат по регрету показал алгоритм Ensemble Sampling, однако время его работы существенно превышает показатели остальных алгоритмов. Обобщенный TensorTrain показал себя на уровне с Tensor Elimination и Epoch Greedy.

В заключение, отметим, что данная сфера не так давно начала активно развиваться, что было связано с новыми исследованиями в сфере тензорных вычислений. Вероятно, в ближайшее время будут появляется новые алгоритмы для контекстуальных бандитов, поскольку построение качественных и эффективных рекомендательных систем в современном мире крайне востребовано и находит применение во многих сферах.

10 План дальнейшей работы

Дальнейшее исследование можно разделить на два направления. Первое – развитие нового алгоритма и его оптимизация, второе – построение полноценной рекомендательной системы и её тестирование на реальных данных.

Обсудим подробнее каждое направление. На данный момент основная проблема алго-

ритма TensorTrain заключается в частом обновлении разложения, поскольку алгоритм поиска максимума имеет ограничения по размерности, в то время как обновление тензора увеличивает размерность на каждом шаге. Сейчас для обновления тензора используется механизм сложения тензоров в формате тензорного поезда, но обновлять на каждом шаге нужно всего один элемент. Вероятно, такое обновление можно сделать, без изменения размерности, или же с меньшим её увеличением. Поиск подобного решения является важной частью улучшения алгоритма, это помогло бы сделать его более эффективным.

Для построения настоящей рекомендательной системы нужно будет выбрать наилучший из имеющихся алгоритмов, исходя из информации о сфере применения. Основной проблемой для алгоритмов многоруких бандитов зачастую является отсутствие релевантных данных для обучения алгоритмов, поэтому на данном этапе будет важно найти подходящий датасет, или же сформировать его. После этого нужно будет представить контекст в категориальном виде, так как этого требует формат входных данных алгоритмов контекстуальных бандитов, обучить и протестировать алгоритм.

Список литературы

[1] Peter Auer, Nicolo Cesa-Bianchi и Paul Fischer. “Finite-time analysis of the multiarmed bandit problem”. В: *Machine learning* 47 (2002), с. 235—256.

[2] Sébastien Bubeck, Nicolo Cesa-Bianchi и др. “Regret analysis of stochastic and nonstochastic multi-armed bandit problems”. В: *Foundations and Trends® in Machine Learning* (2012), с. 1—122.

[3] Andrei Chertkov, Gleb Ryzhakov, Georgii Novikov и Ivan Oseledets. “Optimization of functions given in the tensor train format”. В: *arXiv preprint arXiv:2209.14808* (2022).

[4] Peter Georg. “Tensor Train Decomposition for solving high-dimensional Mutual Hazard Networks”. Дис. ... док. 2022.

[5] Olga Klopp. “Noisy low-rank matrix completion with general sampling distribution”. В: (2014).

[6] Tor Lattimore и Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[7] Lihong Li, Wei Chu, John Langford и Robert E Schapire. “A contextual-bandit approach to personalized news article recommendation”. В: *Proceedings of the 19th International Conference on the World Wide Web*. 2010, с. 661—670.

[8] Ivan V Oseledets. “Tensor-train decomposition”. В: *SIAM Journal on Scientific Computing* 33.5 (2011), с. 2295—2317.

[9] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen и др. “A tutorial on thompson sampling”. В: *Foundations and Trends® in Machine Learning* (2018).

[10] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen и др. “A tutorial on thompson sampling”. В: *Foundations and Trends® in Machine Learning* 11.1 (2018), с. 1—96.

[11] Yuta Saito, Shunsuke Aihara, Megumi Matsutani и Yusuke Narita. “Open bandit dataset and pipeline: Towards realistic and reproducible off-policy evaluation”. В: *arXiv preprint arXiv:2008.07146* (2020).

[12] Chengshuai Shi, Cong Shen и Nicholas D Sidiropoulos. “On High-dimensional and Low-rank Tensor Bandits”. В: *arXiv preprint arXiv:2305.03884* (2023).

[13] Aleksandrs Slivkins и др. “Introduction to multi-armed bandits”. В: *Foundations and Trends® in Machine Learning* 12.1-2 (2019), с. 1—286.

[14] W. R. Thompson. “On the theory of apportionment”. В: *American Journal of Mathematics.* (1935).

[15] L. R. Tucker. “Implications of factor analysis of three-way matrices for measurement of change”. В: *Problems in measuring change.* Под ред. C. W. Harris. Madison WI: University of Wisconsin Press, 1963, с. 122—137.

[16] Chih-Chun Wang, Sanjeev R Kulkarni и H Vincent Poor. “Bandit problems with side observations”. В: *IEEE Transactions on Automatic Control* 50.3 (2005), с. 338—355.

[17] Qianxin Yi, Yiyang Yang, Yao Wang и Shaojie Tang. “Efficient Generalized Low-Rank Tensor Contextual Bandits”. В: *arXiv preprint arXiv:2311.01771* (2023).

[18] Jie Zhou, Botao Hao, Zheng Wen, Jingfei Zhang и Will Wei Sun. “Stochastic Low-rank Tensor Bandits for Multi-dimensional Online Decision Making”. В: *arXiv preprint arXiv:2007.1578* (2020).

[19] Рахуба Максим. *Основы тензорных вычислений.* ФКН НИУ ВШЭ. 2023. URL: http://wiki.cs.hse.ru/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B_%D1%82%D0%B5%D0%BD%D0%B7%D0%BE%D1%80%D0%BD%D1%8B%D1%85_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B9_(2023/24) (дата обр. 01.09.2023).