RL-агенты с памятью в обучении с подкреплением

Введение

В современном мире наиболее остро встаёт вопрос работы с **POMDP** процессами, в которых агент наблюдает лишь какую-то **часть** от всего состояния среды. Это связано с тем, что в реальных задачах почти наверное не удается снять показания со всей среды. Особенное применение, конечно, здесь стимулирует развивающаяся область робототехники, в которой обычно мы ограничены базовыми датчиками вроде камеры глубины, лидара и т.д.

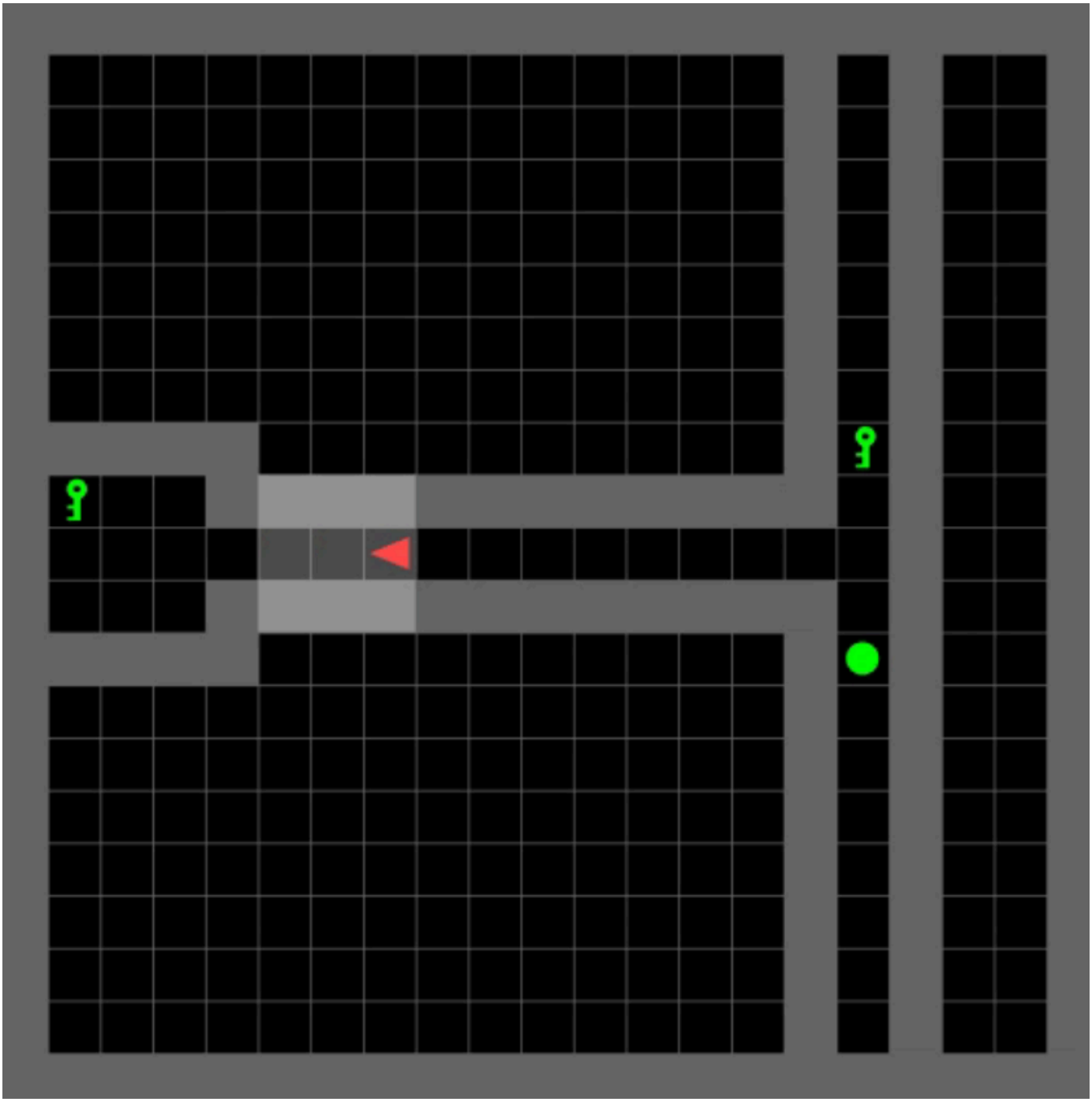
Основная проблема для задач, связанных с **POMDP** заключается в том, что модели (а далее - агенту) необходимо запоминать и проносить с собой контекст, который нужен для решения задачи. Основным препоном, разумеется, встаёт слово *нужен*, потому как совершенно непонятно какой из тех кусочков информации, что строит внутри себя агент, является *полезным* и *нужным*.

Помимо тенденции к работе с **POMDP** также четко видна проблема т.н. "Stack More Layers". Исследователям зачастую становится совершенно индефферентен вопрос о том, что именно строит агент в ходе решения задачи (на самом деле, разумеется, проблема кроется еще в отсутствии лёгкой интерпретируемости для всех нейросетевых моделей), и поэтому они расходуют память в любых количествах и размерах, не задумываясь о том, сколько ее *нужно вообще*.

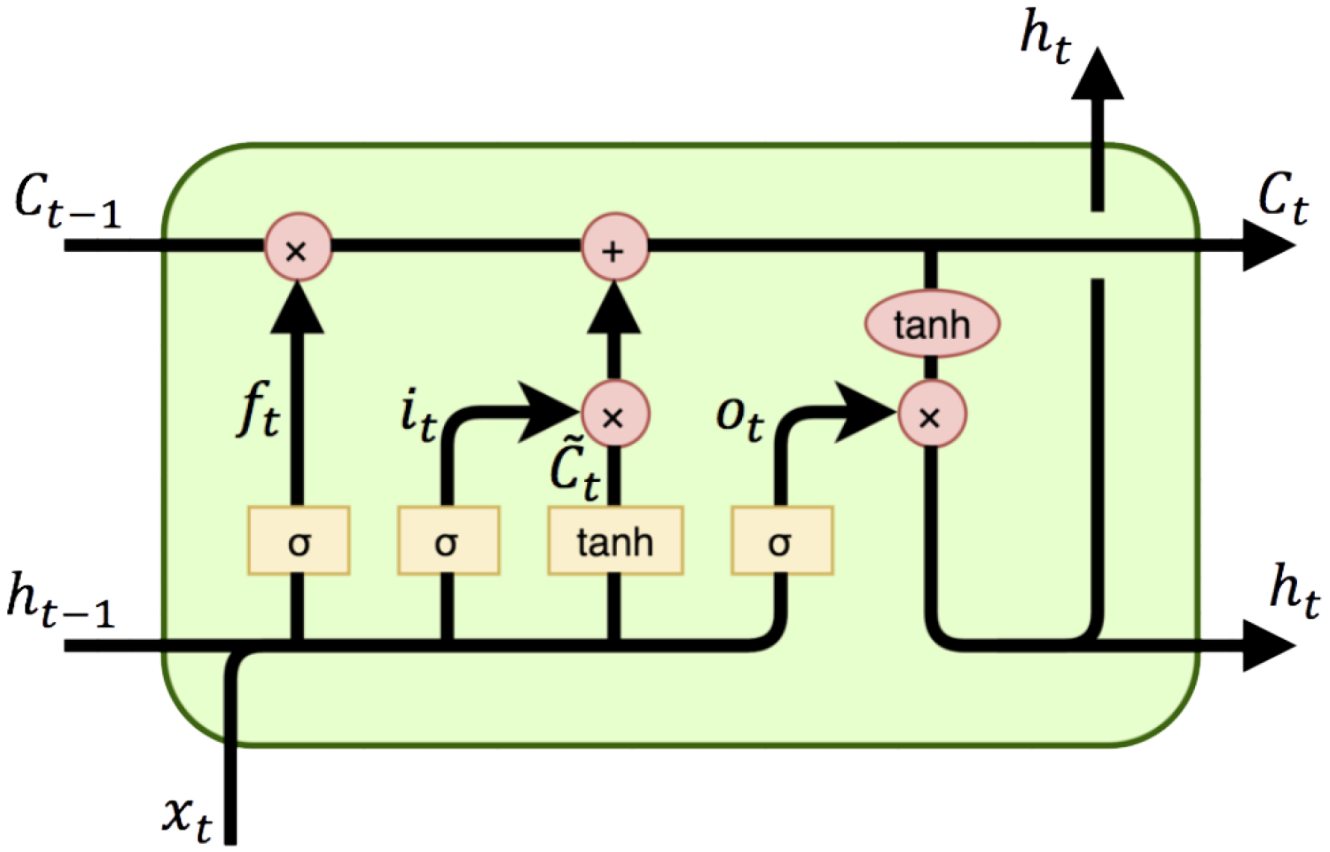
В силу этого, можно поднять вопрос о том, как оптимизировать память агента. Хочется каким-то образом "убирать" ненужную информацию из памяти агента. Звучит как построение фильтра для памяти агента, чтобы убирать некоторый шум, который будет препятствовать принятию адекватных решений агентом в ходе решения задач. Тяжело представить себе фильтр построенный на основе уже обученного агента в силу пункта выше - мы почти наверное не сможем проинтерпретировать и построить что-то рациональное в таком случае.

Формализация задачи

Потому как мы работаем с **POMDP**, то будет решаться задача *Reinforcement Learning* в среде крайне удобной для этого - `MinigridS13`. В ней перед агентом встаёт задача, формализуемая как чистое **POMDP** и требующая от него использования памяти: в начале агенту демонстрируется некоторый объект, по форме напоминающий ключ или сферу, после чего агенту нужно пройти по небольшому коридору и выбрать такой же объект. Сложность для агента заключается в том, что он не может запомнить в какую из сторон идти, поэтому алгоритм *PPO* не справится с этой задачей адекватно, сохранив метрику на уровне ~0.5 награды/эпизод.



Интуиция подсказывает, что для решения поставленной задачи можно применить классическую архитектуру для запоминания и работы с "бесконечной" памятью, именуемой LSTM. Агенту необходимо дать такую ячейку памяти для того, чтобы со временем он наткнулся на возможность поместить туда какие-то знания о том, *что* хранилось в начале карты.



Однако LSTM обладает существенным недостатком - ее память это "размазанное" скрытое состояние, которое мы не можем до конца интерпретировать как "память" в классическом для понимании человека, поскольку там содержатся не только какие-то эмбеддинги, описывающие *полезную* информацию в контексте опознания ключ-сфера, но также и какие-то неинтерпретируемые внутренние значения модели.

В противовес ей можно поставить архитектуру, представленную в работе "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context" в 2019 году [1]. Данная архитектура решает классические проблемы трансформера вроде ограниченности долговременной памяти в силу окна контекста, и вполне может соперничать с LSTM в плане сохранения контекста.

В таком случае, мы определились с условиями задачи - у нас есть агент, построенный на базе LSTM/TransformerXL, мы обучаем агента с помощью алгоритма **PPO** на среде `MinigridS13`, параллельно пытаясь некоторым образом вмешаться в его память во время обучения для того, чтобы построить фильтрацию памяти у агента. Следующим встает вопрос о том, как именно фильтровать память агента.

Типы фильтраций

Идейно несложно придти к тому, что для фильтрации подходят такие методы как, например, низкоранговые разложения. Низкоранговые разложения позволяют нам оценить вклад интересующих нас компонент за счет построения проекций в некоторых латентных подпространствах. Как это принято в научной литературе, для прямоугольных матриц обычно используется сингулярное разложение, которое позволяет нам представить матрицу как произведение трех. Любую матрицу мы можем прдетсавить M в виде произведения трёх других матриц: $M = U\Sigma V^T$

- U и V^T — это матрицы вращения.
- Σ — это диагональная матрица, содержащая так называемые сингулярные значения.
  Singular Value Decomposition (SVD) разделяет исходную матрицу на три компонента: вращение, масштабирование и ещё одно вращение. Важнейшая часть, это матрица Σ, сингулярные значения которой расположены в порядке убывания. Самые большие значения содержат ключевую информацию о структуре матрицы, а самые маленькие, информацию о мелких деталях.

Все что мы можем представить в виде матрицы (изображения, веса модели), с помощью SVD мы можем эффективно сжимать или восстанавливать.

Идея заключается в том, чтобы использовать не все сингулярные значения из матрицы Σ, а только k самых больших из них. Это называется **низкоранговой аппроксимацией**.

Например, сжатие можно наблюдать на разложении картины Виктора Васнецова "Три богатыря". Мы можем четко заметить, что увеличение количества сингулярных чисел ведет к увеличению детализации изображения.

Конечно, есть и другие методы низкоранговых разложений - например, предложенный Иваном Оселедцем в 2009 году Поезд Тензоров [2], который позволяет раскладывать тензоры высшего порядка для более компактного представления. Действие разложения *Tensor Train* (TT) почти незаметно на глаз в сравнении с SVD. Это мы можем пронаблюдать на разложении картины "Утро в сосновом бору".

Кратко TT-разложение можно записать следующим образом:

$$[\mathcal{A}(i_1, i_2, \ldots, i_d) = \sum_{\alpha_0=1}^{r_0} \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_d=1}^{r_d} G_1(\alpha_0, i_1, \alpha_1) \cdot G_2(\alpha_1, i_2, \alpha_2) \cdots G_d(\alpha_{d-1}, i_d, \alpha_d)] \quad (1)$$

где:

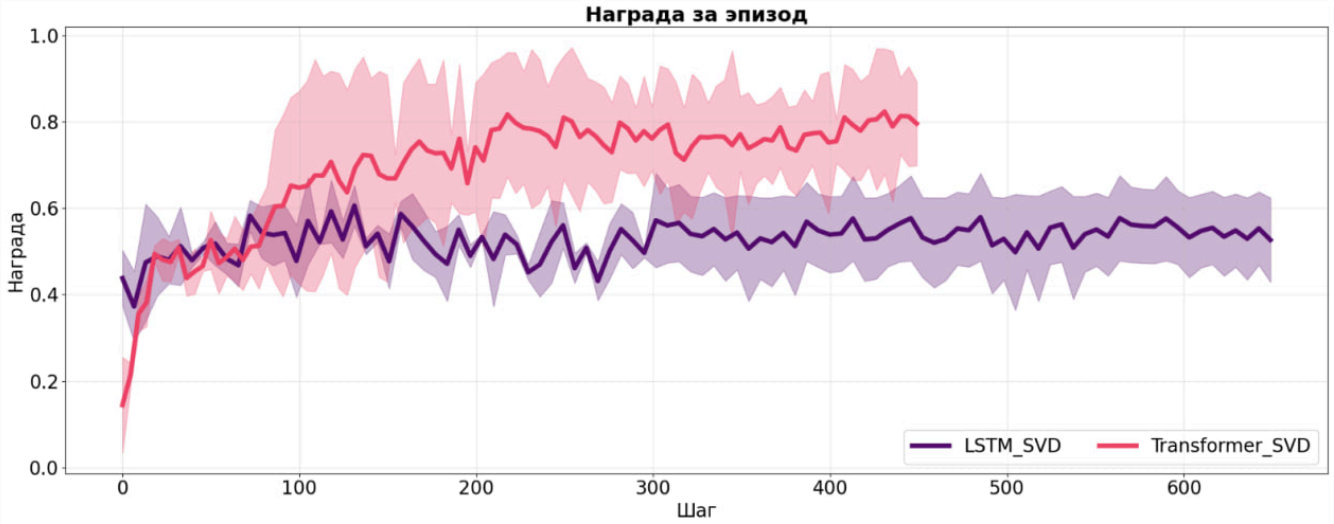
- $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times \ldots n_d}$ - исходный d - мерный тензор.
- $G_k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ - TT-ядра (тензоры 3-го порядка).
- $r_0 = r_d$ - граничные условия TT-рангов.

Помимо этого, также есть методы из классической обработки сигналов - различные фильтры, вейвлеты и методы, однако нас интересуют два самых ярко выделяющихся: фильтр Гаусса и фильтр Лапласа, которые соответственно размывают и увеличивают резкость картинки. Таким образом при обработке памяти агента, мы сможем увидеть как противоположное воздействие на память влияет на то, как он взаимодействует со средой.

Также можно попробовать обучить вариационный автокодировщик для того, чтобы фильтр являлся *обучаемым*. Смотрим.

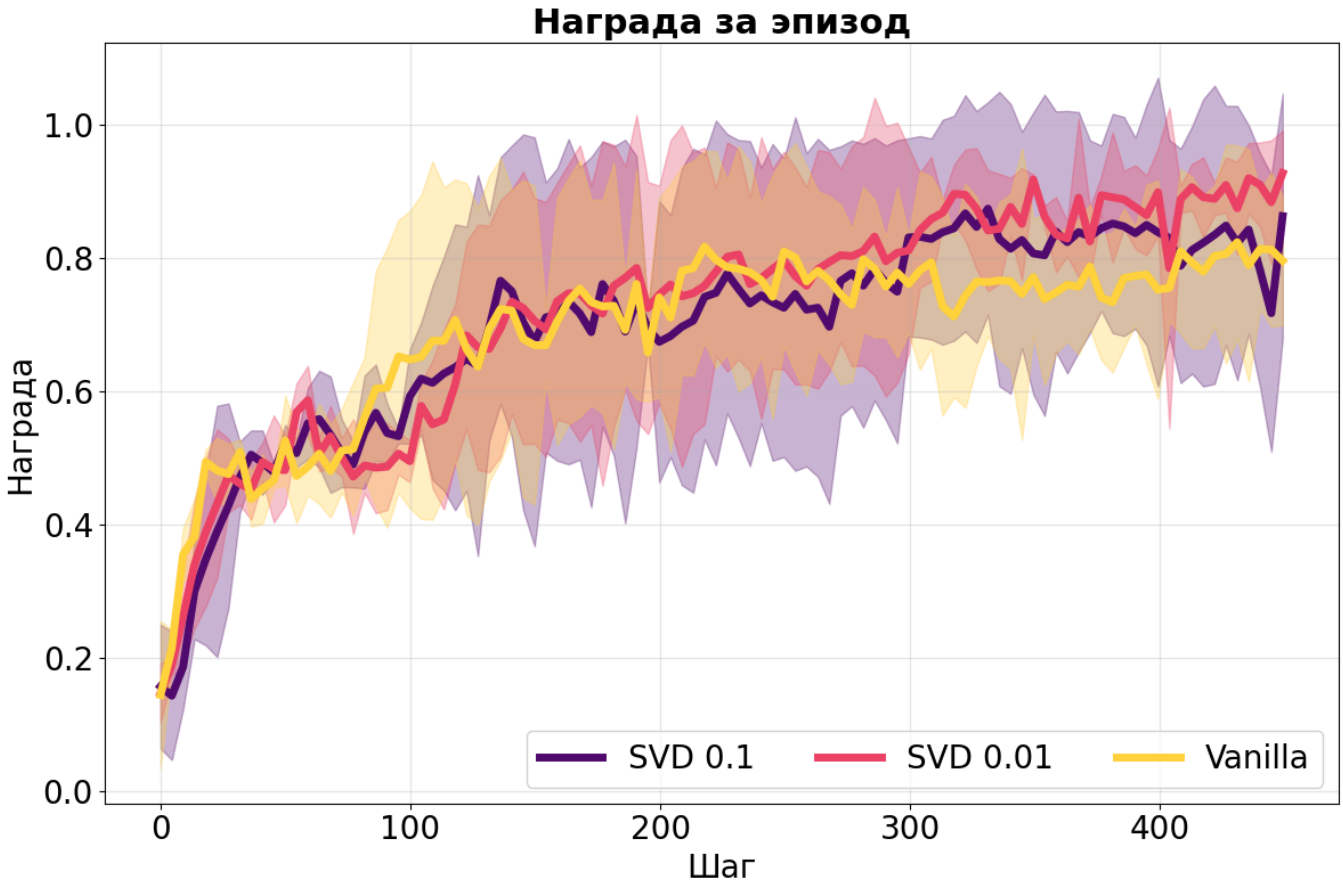
Эксперименты

За основу был взят факт, что обе модели способны сойтись на среде `Minigrid13S` без каких-либо внедрений в их память. Для начала попробуем сравнить между собой сходимость для LSTM-based и Transformer-based архитектуры с применением SVD.



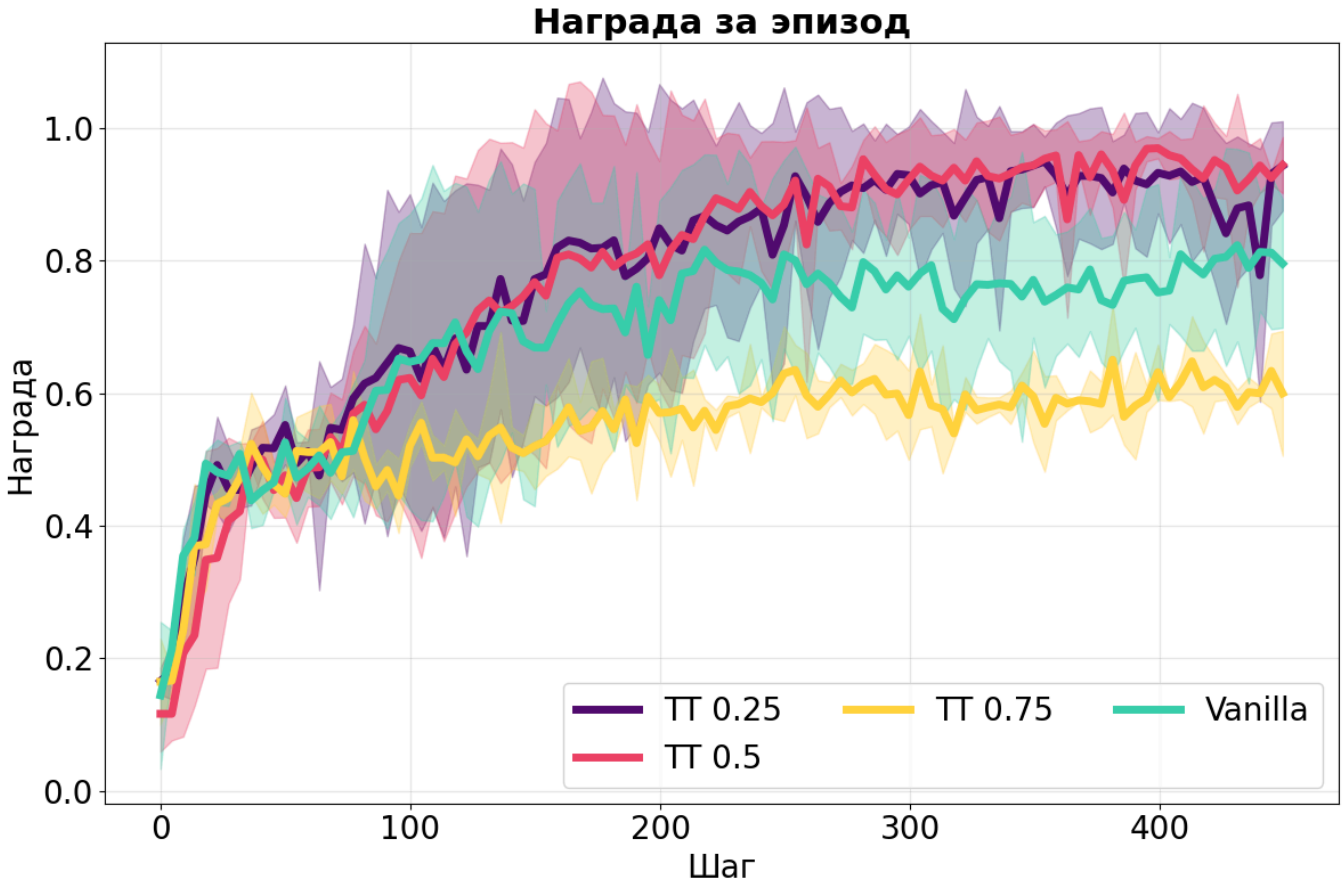
Модель с использованием LSTM очень быстро сошла с дистанции в силу того, что для нее любые разложения, похоже, являются разрушающими структуру скрытых состояний. В таком случае, модель не обучается использовать память вовсе, алгоритм PPO сводит агента к развилке, на которой агент начинает выбирать рандомно, и совершенно не учится строить связи и запоминать. Дополнительные шаги обучения также не помогли модели LSTM сойтись.

В качестве следующего эксперимента попробуем дойти до предела сжатия памяти трансформера. Есть обоснованное предположение, что в столь простой задаче такое количество памяти избыточно и не требуется. Попробуем сжимать память до каких-то разумных пределов, в качестве такого возмём 1% от изначально ранга.



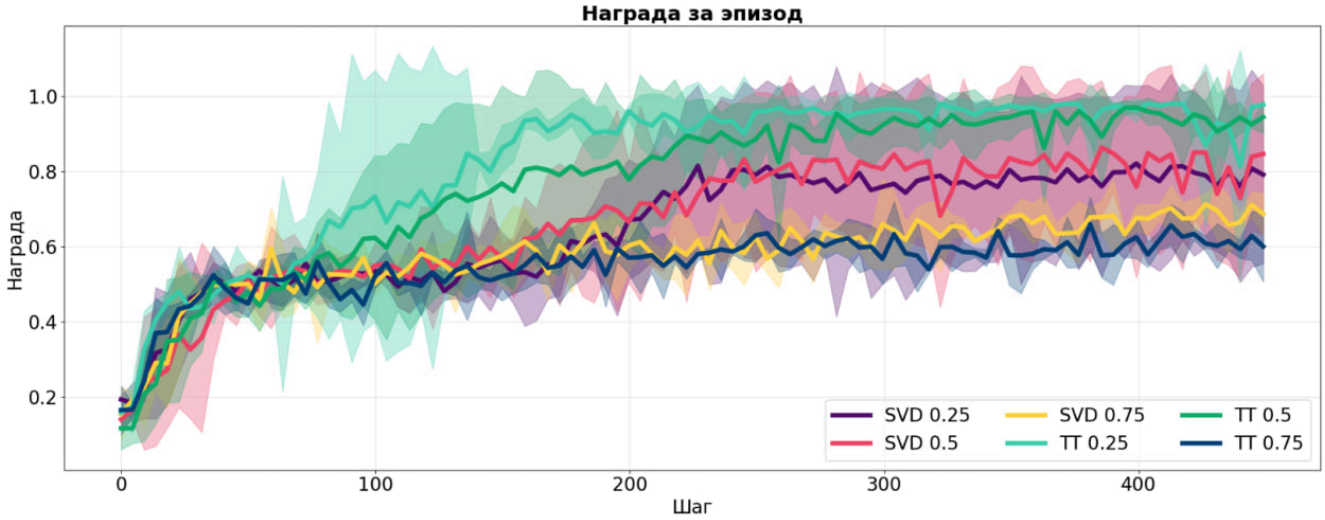
Как мы можем наблюдать выше, агент не просто сошёлся, а смог немного улучшить результат базовой модели, которая обучалась без использования каких-либо сжатий памяти.

Для следующего эксперимента было взято уже TT разложение с несколькими порогами для того, чтобы его влияние на сходимость модели при обучении с таким "препятствием".



Можно обратить внимание, что оба TT разложения рангом ниже 0.75 от изначального, смогли подняться выше по метрике, чем базовый трансформер. Вероятно, что информация, отсекаемая при работе с 0.75 от изначального ранга в TT разложении слишком важна для решения задачи из-за чего сходимости выше случайной от PPO нет.

Было бы также интересно сравнить TT разложение и SVD разложенте между собой для того, чтобы понимать, какое увеличит метрику сильне в поставленной задаче. Для этого провёдем также по 3 эксперимента и усредним результаты.



Исходя из полученной выше сходимости, можно сделать вывод о том, что агент гораздо лучше сходится при использовании TT с рангом 0.5 от изначального. Также заметно, что SVD с использованием ранга в районе 0.75 следует похожему паттерну и не сходится.

Для следующих экспериментов было использовано функциональное фильтрование - с помощью сглаживания Гаусса и выделения резкости Лапласа. В данном случае фильтр Гаусса был задан как:

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{2}$$

где:

- σ - стандартное отклонение.
  В свою очередь фильтр Лапласа можно записать посредством суммы вторых производных,

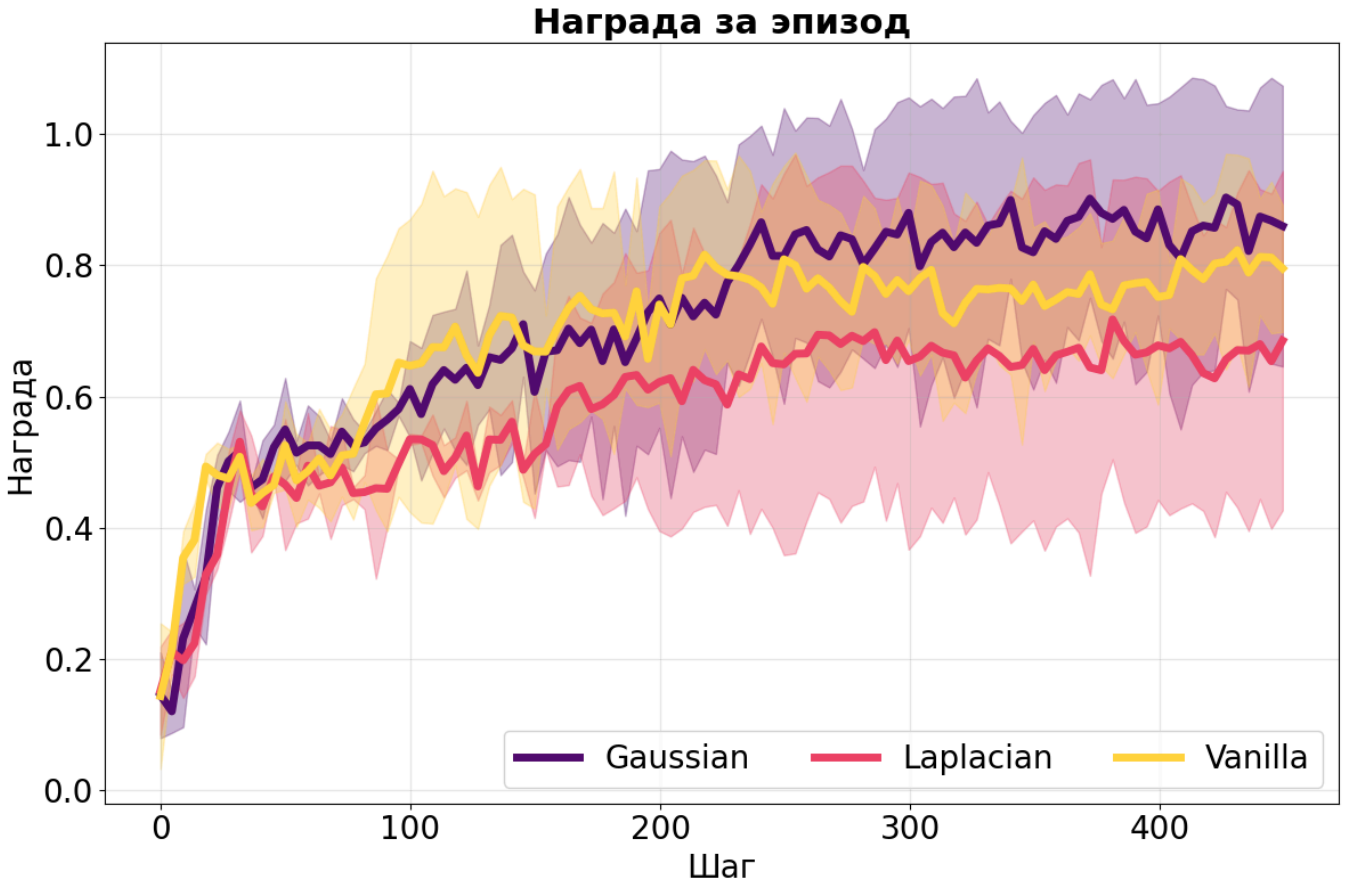
  $$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3}$$

  Однако, поскольку мы всё таки работаем с дискретным случаем, то нет никакого смысла использовать вторые частные производные, и можно переписать это более естественным образом:

  $$\Delta f(x,y) \approx f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{4}$$

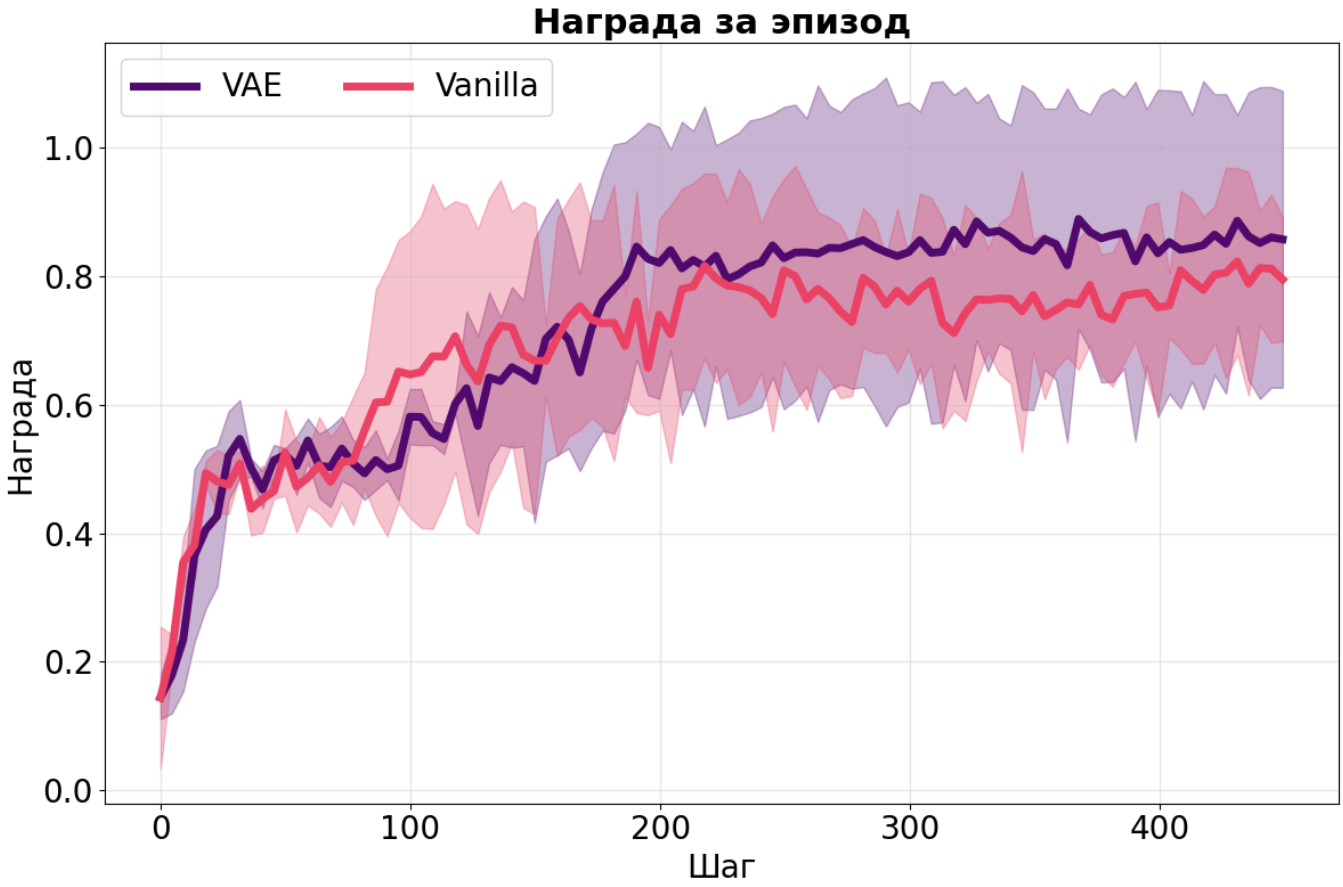
  Или, собирая коэффициенты у этого дифференциала, перепишем в виде матрицы

$$Ker_{laplace} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{5}$$



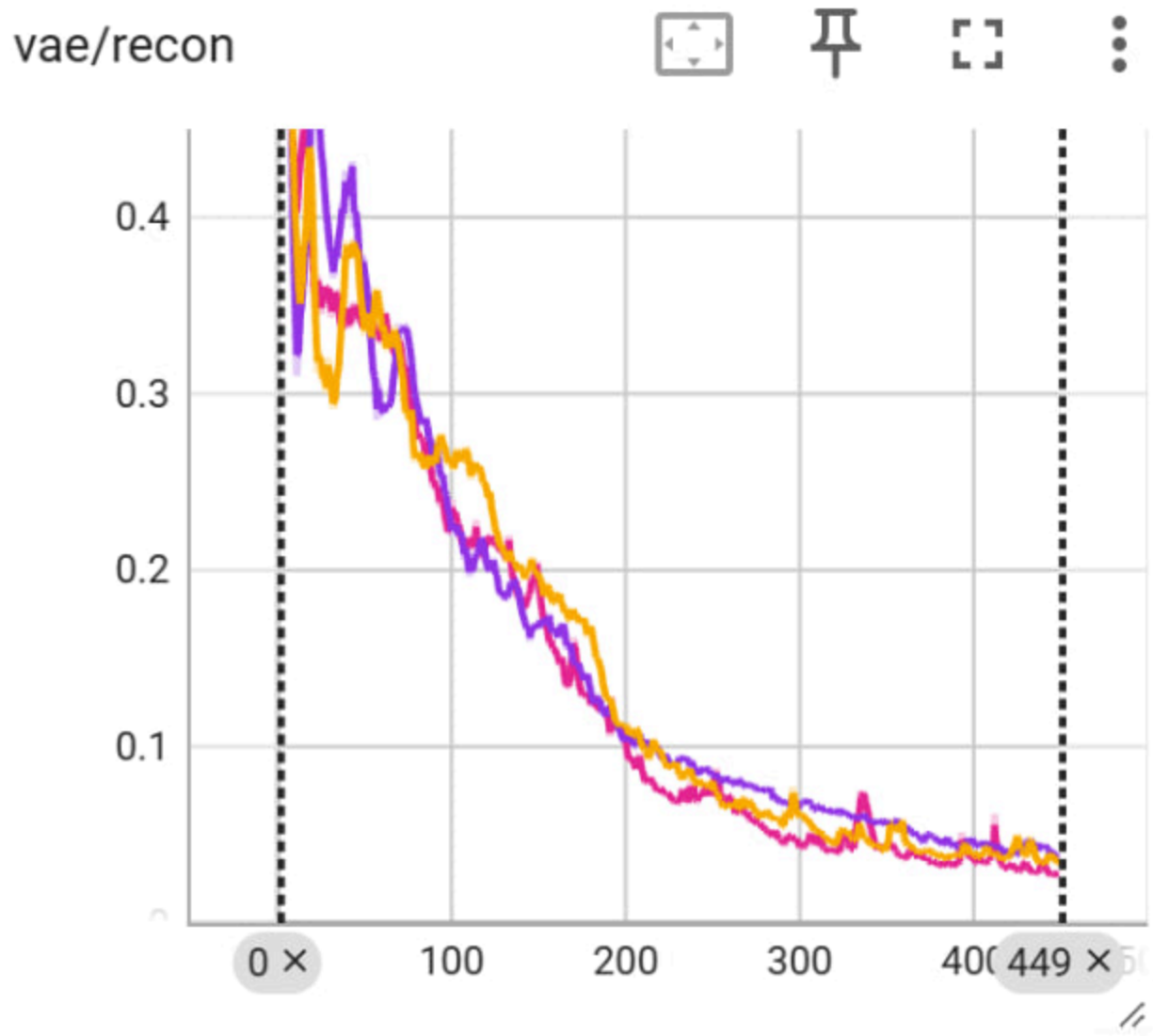
Исходя из этой зависимости, можно сделать предположение о том, что небольшое размытие в памяти агента скорее помогает сходимости - вероятно, что это влияет на гладкость политики агента из-за чего его действия могут становится более консистентыми в соотвествии с тем, что было в начале эпизода. Попытка нарастить разницу между ячейками памяти модели привела к плохой сходимости, скорее всего, это ведет к дополнительной сходимости, и, как следствие, плохой сходимости агента.

Помимо таких простых "автономных вариантов", можно попробовать модель обучить заниматься фильтрацией. В качестве такой модели был взяь вариационный автоэнкодер, а именно три полносвязных слоя, занимающиеся удалением шума из памяти.



Поскольку вариацонный автокодировщик был обучен в онлайн режиме, то это занимает много времени, и по этой причине не был доучен кодировщик, судя по

отсутствию плато у него.



Результаты

В ходе проекта были проведены эксперименты по обучению агента с различными архитектурами памяти. Несмотря на технические сложности и необходимость адаптации внешнего кода, удалось провести 65 запусков и получить широкий спектр поведенческих стратегий, визуализированных в нескольких тысячах анимаций.

Модель на основе трансформера показала высокую эффективность даже при крайне ограниченной памяти (до 1%), однако эмбеддинги, формируемые в процессе обучения, содержат значительный шум. Это указывает на потенциальные резервы для оптимизации.

В дальнейшем планируется исследовать альтернативные методы, такие как **Stable Hadamard Memory** и **Fast and Forgetful Memory**, а также изучить проекцию памяти

в пространстве представлений.

Ссылки

[1] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., & Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. *In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)* (pp. 2978–2988). https://doi.org/10.18653/v1/P19-1285

[2] Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, *33*(5), 2295–2317. https://doi.org/10.1137/090752286