

Лекция 6

Обучение с подкреплением

Никита Юдин, iudin.ne@phystech.edu

Московский физико-технический институт
Физтех-школа прикладной математики и информатики

13 марта 2024



Что делали?

Q-функция по политике π

$$Q^\pi(s, a) = \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s, a_0 = a \right] = \mathbb{E}_\tau Z(s, a),$$

где $\tau \sim p(s_0) \cdot \prod_{t=0}^{\infty} (\pi(a_t | s_t) p(s_{t+1} | s_t, a_t))$. До этого момента вопрос о распределении Z нас несильно интересовал — мы искали сразу её матожидание.

Что делали?

Идея

Почему бы нам не искать распределение Z (т.е. распределение наград) и понимать о задаче больше?

Иными словами заставим нейронную сеть аппроксимировать функцию Z и будем выбирать оптимальное действие согласно

$$a^* = \arg \max_a \mathbb{E} Z(s, a)$$

Вопрос к залу

Каким образом заставить нейросеть возвращать распределение?

Что делали?

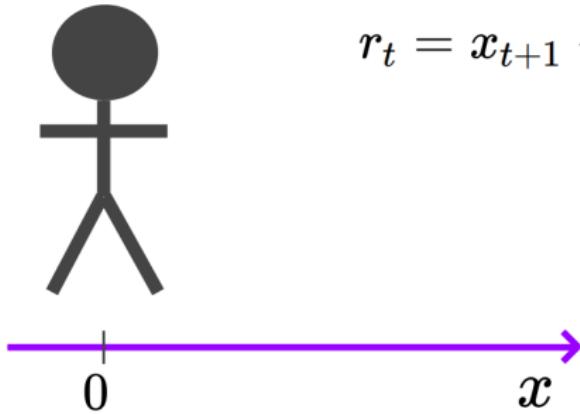
QR-DQN

Зафиксируем N и будем генерировать N дельта-функций. Их позиции будет выбирать нейронная сеть, каждая дельта-функция принимает максимальное значение $1/N$.

Следующий вопрос

Каким образом учить такую сеть? — Смотри предыдущую лекцию.

Жадность плохое качество

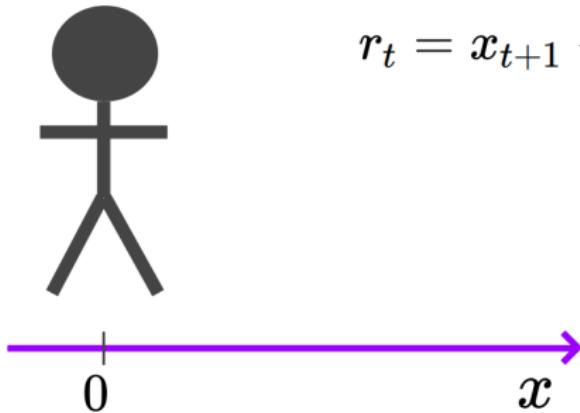


$$r_t = x_{t+1} - x_t$$

Жадная политика агента

- Упасть — маленькая награда
 $r_{\text{fall}} = \Delta x$

Жадность плохое качество

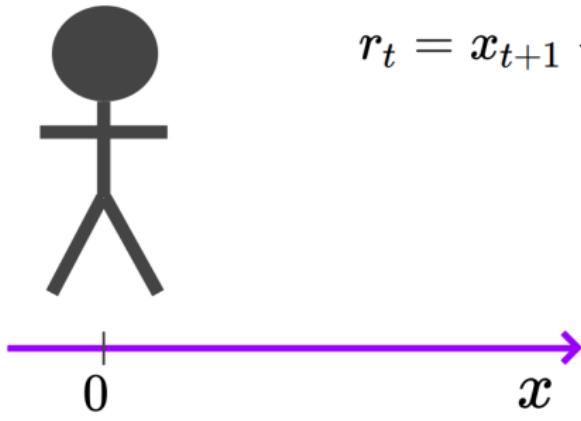


$$r_t = x_{t+1} - x_t$$

Жадная политика агента

- Упасть — маленькая награда
 $r_{\text{fall}} = \Delta x$
- Идти вперед — награда пропорциональна движению

Жадность плохое качество

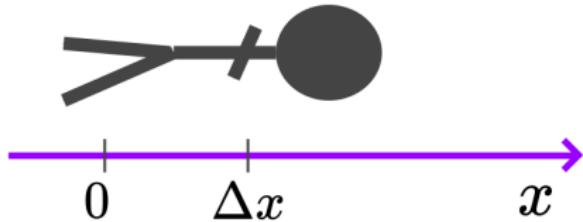


$$r_t = x_{t+1} - x_t$$

Жадная политика агента

- Упасть — маленькая награда
 $r_{\text{fall}} = \Delta x$
- Идти вперед — награда пропорциональна движению
- Init оценка Q -функции:
 $Q(s_0, a_{\text{fall}}) = 0$
 $Q(s_0, a_{\text{step}}) = 0$

Жадность плохое качество

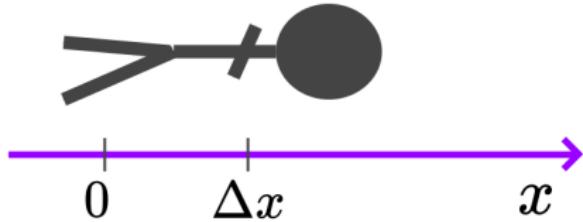


$$r_t = x_{t+1} - x_t$$

Жадная политика агента

- Если политика жадная и нам не повезло выбрать первым действием падение, то награда всегда будет Δx , потому что агент не исследует!

Жадность плохое качество



$$r_t = x_{t+1} - x_t$$

Жадная политика агента

- Если политика жадная и нам не повезло выбрать первым действием падение, то награда всегда будет Δx , потому что агент не исследует!
- Вывод: нужно добавить *exploration*, подобно тому как мы делали это в *DQN*.

Одно «но»

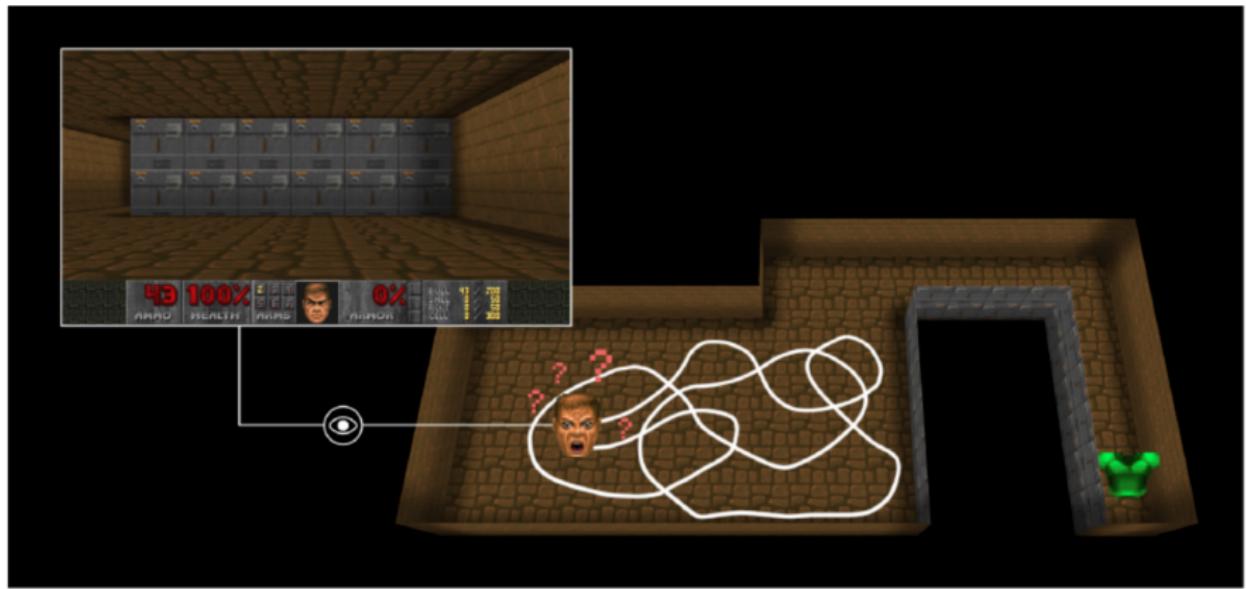
Внимание

Несмотря на то что для DQN с ε -жадной стратегией есть целая теорема о сходимости, на практике сходимость может быть ужасно плохой. Это связано прежде всего с тем, что в теореме используется бесконечное количество взаимодействий со средой.

Представьте

А что если у нас среда — трехмерный дум????

Как я буду эксплорить-то?



Self-Supervision

Напоминание

Марковский процесс принятия решений — кортеж из (S, A, R, T, γ) , где:

A — множество действий,

S — множество состояний,

$R : S \times A \rightarrow R$ — функция наград,

$T : S \times A \times S \rightarrow [0, 1]$ — функция вероятности перехода

Для которого $T(s, a, s') = \mathbb{P}(s_{t+1} = s' | S_t = s, A_t = a)$, то есть вероятность оказаться в следующем состоянии зависит только от предыдущего состояния и действия из него.

Для решения нашей проблемы с *exploration* попробуем изменить функцию награды R

Markov decision process

- S — множество состояний
- A — множество действий
- $S \times A \times S \rightarrow [0, 1]$ —
функция вероятности
перехода
- $R^{\text{extr}} : S \times A \rightarrow R$ —
функция наград для
исходной задачи (внешняя)

Self-Supervision

Markov decision process

- S — множество состояний
- A — множество действий
- $S \times A \times S \rightarrow [0, 1]$ — функция вероятности перехода
- $R^{\text{extr}} : S \times A \rightarrow R$ — функция наград для исходной задачи (внешняя)

Auxiliary task

- S — множество состояний
- A — множество действий
- $S \times A \times S \rightarrow [0, 1]$ — функция вероятности перехода
- $R^{\text{intr}} : S \times A \rightarrow R$ — функция наград для лучшего *exploration* (внутренняя)

На практике

В реальных задачах такое раздвоение MDP означает добавление к основной награде придуманной нами награды за исследование.

Вся сложность

Основная задача теперь создать хорошую функцию «внутренней мотивации» агента к исследованиям даже для тех сред, о которых нам самим не особо много известно.

Basic Idea

Самая простая идея

Начнем выдавать агенту дополнительную награду за посещение нового состояния среды. Данные награды будут пересоздаваться перед началом каждого эпизода. Однако такая функция наград поощряет не «умное» исследование, а просто исследование всего пространства состояний на каждом эпизоде.

Basic Idea

Самая простая идея

Начнем выдавать агенту дополнительную награду за посещение нового состояния среды. Данные награды будут пересоздаваться перед началом каждого эпизода. Однако такая функция наград поощряет не «умное» исследование, а просто исследование всего пространства состояний на каждом эпизоде.

Модификация (часто применяется на практике)

Не пересоздаём награды в посещенных ячейках между эпизодами, а заводим счетчики посещения состояний агентом за всё время обучения и, когда агент заходит в состояние, поощряем его обратно пропорционально счётчику (В теории такое решение делает *MDP* неприменимым, поскольку среда «подстраивается» под агента, но на практике всё хорошо).

Недостатки самой простой идеи

- Данное решение невозможно применить в средах с непрерывным количеством состояний (среду придется бить на конечное число кусков и присваивать кускам награды; или нужно иметь некоторые дополнительные знания о среде, характерные только для неё)

Недостатки самой простой идеи

- Данное решение невозможно применить в средах с непрерывным количеством состояний (среду придется бить на конечное число кусков и присваивать кускам награды; или нужно иметь некоторые дополнительные знания о среде, характерные только для неё)
- Такой подход награждает все состояния одинаково (не выявляет важные или похожие друг на друга состояния)

Недостатки самой простой идеи

- Данное решение невозможно применить в средах с непрерывным количеством состояний (среду придется бить на конечное число кусков и присваивать кускам награды; или нужно иметь некоторые дополнительные знания о среде, характерные только для неё)
- Такой подход награждает все состояния одинаково (не выявляет важные или похожие друг на друга состояния)
- Нужно помнить в каких состояниях мы уже были

Basic Idea

Mario Oracle

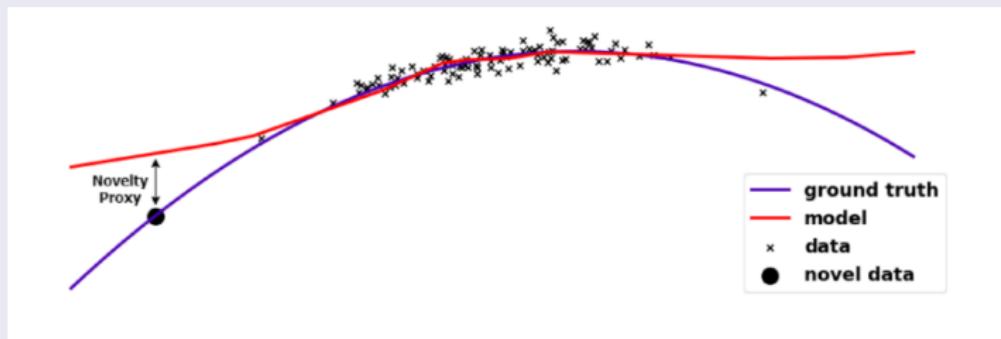
Пример хорошего *reward-shaping* на конкретном примере



Будем поощрять движение агента вправо и наказывать за движение влево.

Детектируем новые состояния

Рассмотрим некоторую задачу регрессии на картинке:



Задача обучения Q -функции — тоже задача регрессии! Поэтому новые состояния можно оценивать по этой задаче регрессии.

Замечание

Обучать Q -функцию может быть трудно учить, особенно тяжело оценить насколько хорошо мы её выучили (иными словами — нужен таргет), а значит смотреть на зазор ошибки Q -функции — плохая идея

Другая задача регрессии!

Заведем вспомогательную нейронную сеть, выходы этой нейронной сети на наших состояниях и есть «новизна» состояний.

Недостатки такого метода

- Мы можем иметь большую ошибку не только на новых состояниях, которые модель ещё не видела, но и в том случае, если модель не обладает достаточной гибкостью.

Недостатки такого метода

- Мы можем иметь большую ошибку не только на новых состояниях, которые модель ещё не видела, но и в том случае, если модель не обладает достаточной гибкостью.
- Мы можем столкнуться с большими проблемами, если целевые значения в задаче регрессии — случайные величины (недетерминированный таргет).

Недостатки такого метода

- Мы можем иметь большую ошибку не только на новых состояниях, которые модель ещё не видела, но и в том случае, если модель не обладает достаточной гибкостью.
- Мы можем столкнуться с большими проблемами, если целевые значения в задаче регрессии — случайные величины (недетерминированный таргет).
- Ещё хуже, если в процессе обучения какие-то целевые значения меняются (нестационарный таргет).

Всё обсудили

Теперь перейдем к возможному варианту реализации проверки состояния на «новизну».

Random Network Distillation

1. Заведем нейросеть, называемую *Target Network*,
заподлицоизируем ей веса (случайным образом) и зафиксируем
их навсегда.

Random Network Distillation

1. Заведем нейросеть, называемую *Target Network*,
заподлицоизируем ей веса (случайным образом) и зафиксируем
их навсегда.
2. Заведем вторую нейросеть, называемую *Predictor Network*,
которую уже будем обучать. Она будет пытаться спрогнозировать
выход *Target Network* на состояниях среды.

Random Network Distillation

1. Заведем нейросеть, называемую *Target Network*, инициализируем ей веса (случайным образом) и зафиксируем их навсегда.
2. Заведем вторую нейросеть, называемую *Predictor Network*, которую уже будем обучать. Она будет пытаться спрогнозировать выход *Target Network* на состояниях среды.
3. Невязку между значениями *Target Network* на данном состоянии и *Predictor Network* на данном состоянии будем интерпретировать как новизну этого состояния:
$$r^{\text{intr}}(s) := \|\varphi^{\text{predictor}}(s) - \varphi^{\text{target}}(s)\|_2^2 / 2.$$

Но есть нюансы!

- Случайная *Target Network* должна давать *различные* ответы на *различных* состояниях (аккуратная инициализация)

Но есть нюансы!

- Случайная *Target Network* должна давать *различные* ответы на *различных* состояниях (аккуратная инициализация)
- Эмбеддинги состояний хорошо бы отнормировать, прежде чем давать их сетям. Для этого перед всем действием запускаем агента в среду, он собирает для нас состояния, по этому пакету считаем среднее и отклонение и далее используем их для нормировки при обучении (больше статистики не меняются — получим нестационарность таргета).

Но есть нюансы!

- Первые несколько итерации обучения лучше вообще не использовать внешнюю награду, а пообучать внутреннюю (когда *Predictor* выйдет на асимптоту обучения)

Но есть нюансы!

- Первые несколько итерации обучения лучше вообще не использовать внешнюю награду, а пообучать внутреннюю (когда *Predictor* выйдет на асимптоту обучения)
- Внутреннюю награду можно шкалировать по величине в зависимости от того, какие внутренние награды были на предыдущих шагах (скользящим оцениванием статистик по ходу обучения)

Curiosity

Любопытство

Идея: давайте смотреть не на «новизну» состояния, а на то насколько наш агент хорошо понимает как устроен мир. Иными словами давайте учить среду. $f(s, a)$ — функция, которая по текущим s, a предсказывает следующее состояние:

$$\frac{1}{2} \|f(s, a) - s'\|_2^2 \rightarrow \min_f$$

Ошибку этой модели будем использовать как внутреннюю награду:

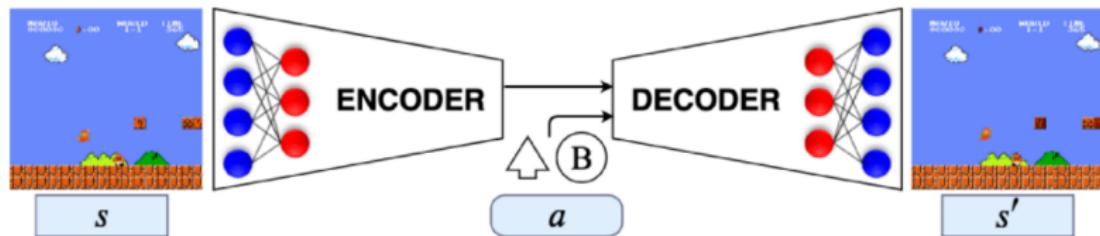
$$r^{\text{intr}}(s, a, s') := \frac{1}{2} \|f(s, a) - s'\|_2^2$$

То есть нам неважно новое это состояние или нет; если мы можем предсказать, что будет дальше, то нам неинтересно.

Curiosity



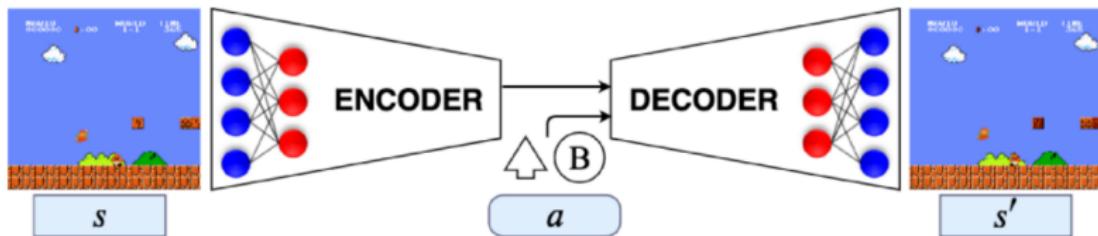
Curiosity: naive approach



Проблемы

- Генерация картинок — дорогой процесс

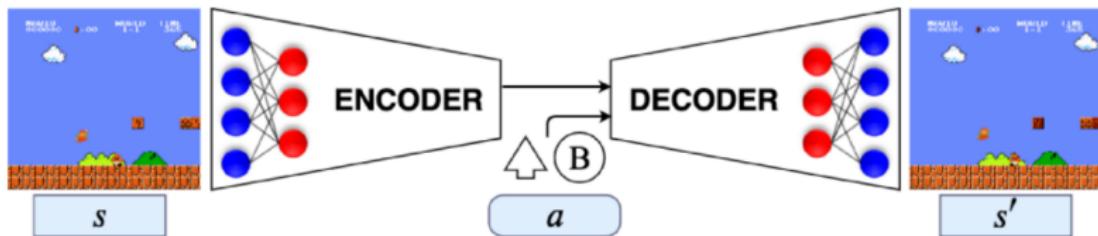
Curiosity: naive approach



Проблемы

- Генерация картинок — дорогой процесс
- Выучиваем большое количество ненужной информации
(выучиваем как двигается вся сцена)

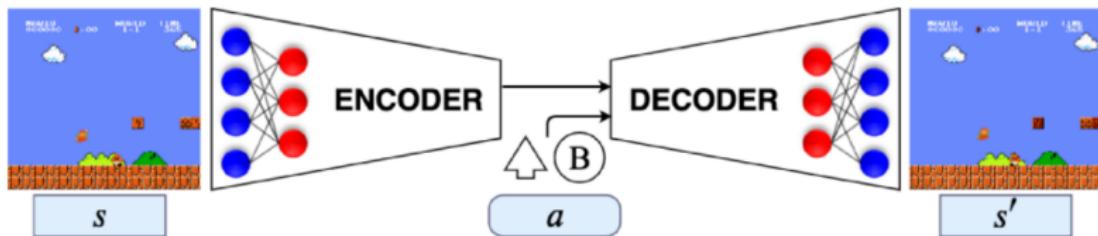
Curiosity: naive approach



Проблемы

- Генерация картинок — дорогой процесс
- Выучиваем большое количество ненужной информации (выучиваем как двигается вся сцена)
- Учим модель в пространстве пикселей!

Curiosity: naive approach



Проблемы

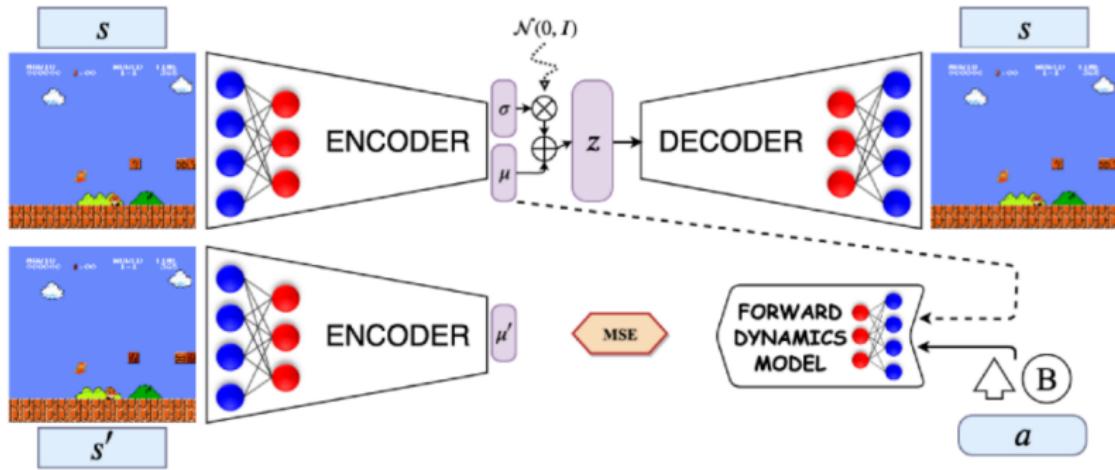
- Генерация картинок — дорогой процесс
- Выучиваем большое количество ненужной информации (выучиваем как двигается вся сцена)
- Учим модель в пространстве пикселей!
- Что если среда не является детерминированной?

Curiosity: less naive approach

Идея

Возьмем VAE и обучим его на нашем буфере, зафиксируем его. Затем на каждом новом состоянии (картинке) VAE будет выдавать нам эмбеддинг этого состояния, который будет отправляться в *Forward Dynamics Model*. Таким образом мы получим *предсказание эмбеддинга следующего состояния*, остаётся только посчитать с помощью VAE эмбеддинг реального следующего состояния и вычислить $MSELoss$ между двумя величинами.

Curiosity: less naive approach



Вывод

Перевод состояний в эмбеддинги это хорошо! Всё, что нам нужно — хорошие эмбеддинги :)

Noisy TV Problem



All you need is good embedding

«Проблема шумного телевизора»

Часто в средах на фоне происходят случайные бесполезные события, которые никак не помогают игроку достичь цели. Такие события для *RL*-агентов очень вредны, ведь на их восприятие тратится очень много ресурсов, поэтому нам хотелось бы, чтобы такие события отсекались на уровне эмбеддингов. Но как?

Требования к эмбеддингам

Фиксируем желания

Прежде чем решать главную проблему с шумом, давайте определим что такое «хороший» эмбеддинг по нашему мнению:

- Не содержит бесполезного шума среды

Требования к эмбеддингам

Фиксируем желания

Прежде чем решать главную проблему с шумом, давайте определим что такое «хороший» эмбеддинг по нашему мнению:

- Не содержит бесполезного шума среды
- Потенциально может содержать всю информацию необходимую для оптимального решения задачи

Требования к эмбеддингам

Фиксируем желания

Прежде чем решать главную проблему с шумом, давайте определим что такое «хороший» эмбеддинг по нашему мнению:

- Не содержит бесполезного шума среды
- Потенциально может содержать всю информацию необходимую для оптимального решения задачи
- Желательно, чтобы латентное пространство эмбеддингов было невелико и мы могли считать $MSELoss$

Требования к эмбеддингам

Фиксируем желания

Прежде чем решать главную проблему с шумом, давайте определим что такое «хороший» эмбеддинг по нашему мнению:

- Не содержит бесполезного шума среды
- Потенциально может содержать всю информацию необходимую для оптимального решения задачи
- Желательно, чтобы латентное пространство эмбеддингов было невелико и мы могли считать $MSELoss$
- Эмбеддинги не должны меняться со временем или хотя бы менялись достаточно медленно (этот пункт мы выполнить не сможем)

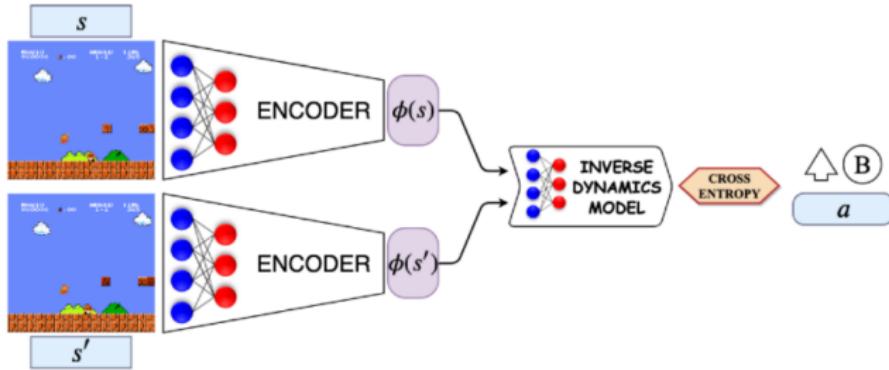
Уменьшаем хотелки до выполнимых

Хороший эмбеддинг

Ищем функцию $\varphi(s) : S \rightarrow \mathbb{R}^d$ — перевода состояния в эмбеддинг такую, что:

- 1) Не содержит бесполезного шума среды
- 2) Потенциально может содержать всю информацию необходимую для оптимального решения задачи
- 3) Желательно, чтобы латентное пространство эмбеддингов было невелико и мы могли считать *MSELoss*

Inverse Dynamics Model



Объяснение

Помочь достичь желаемое может модель, которая по двум соседним состояниям среды предсказывает действие между ними ($\varphi(s)$ — тоже обучается).

Inverse Dynamics Model

На чём основан такой подход?

Для того, чтобы понять какое действие совершилось, фоновая шумовая информация нам не нужна, поэтому она отмечается сразу самой моделью.

Бонусы

На практике ещё часто прибавляют невязку между предсказанным такой моделью действием и действительно выбранным в буфере (такие состояния кажутся более интересными, чем те в которых предсказания модели и выбор агента совпали).

Inverse Dynamics Model

Замечание к бонусу

Невязка может быть большой не только потому что состояние интересное, но и потому что некоторые два различных действия из одного и того же состояния могут приводить в одно и то же состояние. Например, упор в стену в играх: стоять на месте и идти в сторону стены приводят к одному и тому же состоянию (упор в стену).

И это всё?

Inverse Dynamics Model — не единственный способ достичь желаемое.

Intrinsic Curiosity Model

Идея

Сначала обучим эмбеддинги в *Inverse Dynamics Model*, а затем выкинем из неё блок под названием *Inverse Dynamics Model* (оставив только эмбеддинги). Заместо выкинутой части вставим модель прямой динамики и будем снова считать *MSELoss* в латентном пространстве эмбеддингов.

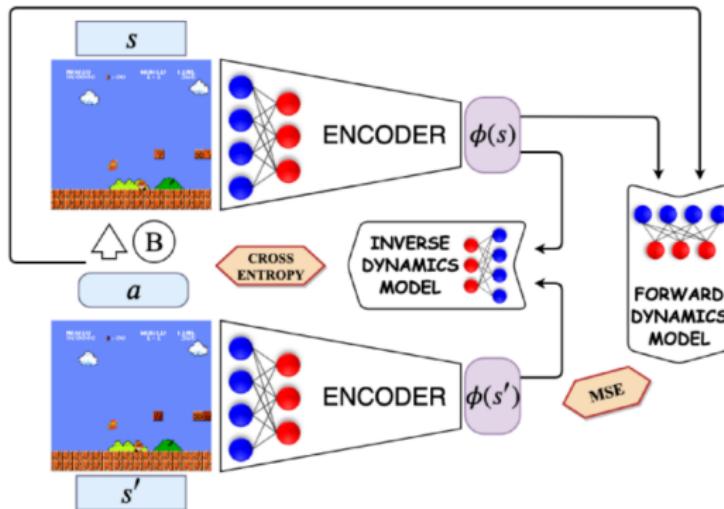
Вопрос

Не совсем понятно где и когда учить *Inverse Dynamics Model* на самом первом шаге.

Решение

Давайте не будем выбрасывать *Inverse Dynamics Model*, а будем таскать её с собой и доучивать вместе с моделью прямой динамики.

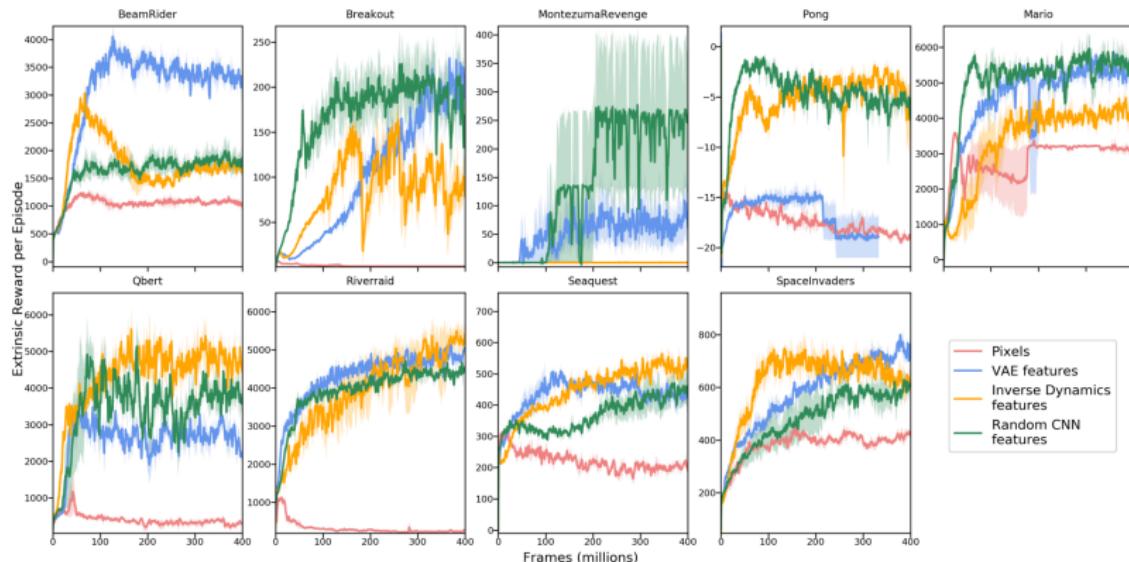
Intrinsic Curiosity Model



Замечание

На этой схеме градиент затекает во все блоки! Появляется регуляризация на модель прямой динамики с помощью модели обратной динамики.

Сравнение



Large-Scale Study of Curiosity-Driven Learning (2018)

Наглядный пример

Curiosity: AI agents exploring without looking at any scores (video)

Телевизор и пульт — слабость *ICM* или нечто большее?

Проблема прокрастинации

ICM настроен фильтровать шум неподконтрольный агенту, но если вдруг, агент может совершать действие и каждый раз получать неожиданный результат, то он залипнет...

Суть проблемы

Кажется, что данная уязвимость не привязана конкретно к алгоритму *ICM*, а является концептуально более важной задачей. Вдруг среда такова, что нужно долистать до сотового канала и остаться на нем? Вдруг случилось так, что мы не можем предсказать будущее не потому что это нерелевантный шум, а потому что мы слишком глупы?

Телевизор и пульт — слабость *ICM* или нечто большее?

Вывод

По риторическим вопросам выше становится понятно, что управляемые шумные телевизоры придется оставить.