Лекция 4

Обучение с подкреплением

Никита Юдин, iudin.ne@phystech.edu

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

28 февраля 2024

ФПМИ

Что делали?

- Начали работать в условиях отсутствия полной информации о среде и решили применять *model-free* методы.
- При этом на среду сохранялись условия $|S| \ll \infty$ и $|A| \ll \infty$.
- Поняли, что оценивать Q-функцию можно несколькими способами: Monte Carlo / Temporal Difference / Q-learning.
- Получили, что для сходимости этих алгоритмов необходимо, чтобы была возможность полного исследования среды, т.е. $\forall s, a \implies \pi(a, s) > 0$.
- Решением проблемы исследования может послужить $\varepsilon-$*greedy* алгоритм.

Что делали?

$$Q^{\pi}_{k+1}(s,a) \leftarrow Q^{\pi}_{k}(s,a) + \alpha_k(y(s,a) - Q^{\pi}_{k}(s,a))$$

Temporal Difference

- $y(s,a) := r + \gamma Q^{\pi}_{k}(s',a')$
- Обновление происходит после каждого шага.
- Медленное распространение награды (посещает по паре состояний).

Monte Carlo

- $y(s,a) := r + \gamma r' + \gamma^2 r'' + \ldots$
- Обновление происходит в конце эпизода.
- Быстрое распространение награды (посещает несколько состояний в эпизоде).

Что делали?

Temporal Difference

- Маленькая дисперсия.
- Смещенная оценка.

Monte Carlo

- Большая дисперсия.
- Несмещенная оценка.

Что делали?

Q-learning

Идея такая же как в Value Iteration — давайте улучшать нашу оценку Q-функции с помощью TD и тут же улучшать нашу политику:

$$
\begin{aligned}
&Q^{\pi}_{k+1}(s,a) \leftarrow Q^{\pi}_{k}(s,a) + \alpha_k \left(r + \gamma Q^{\pi}_{k}(s',a') - Q^{\pi}_{k}(s,a)\right) = \\
&= Q^{\pi}_{k}(s,a) + \alpha_k (r + \gamma Q^{\pi}_{k}(s',\pi_k(s')) - Q^{\pi}_{k}(s,a)) = \\
&= Q^{\pi}_{k}(s,a) + \alpha_k (r + \gamma Q^{\pi}_{k}(s',\arg\max_{a'} Q^{\pi}_{k}(s',a')) - Q^{\pi}_{k}(s,a)) = \\
&= Q^{\pi}_{k}(s,a) + \alpha_k (r + \gamma \max_{a'} Q^{\pi}_{k}(s',a') - Q^{\pi}_{k}(s,a)).
\end{aligned}
$$

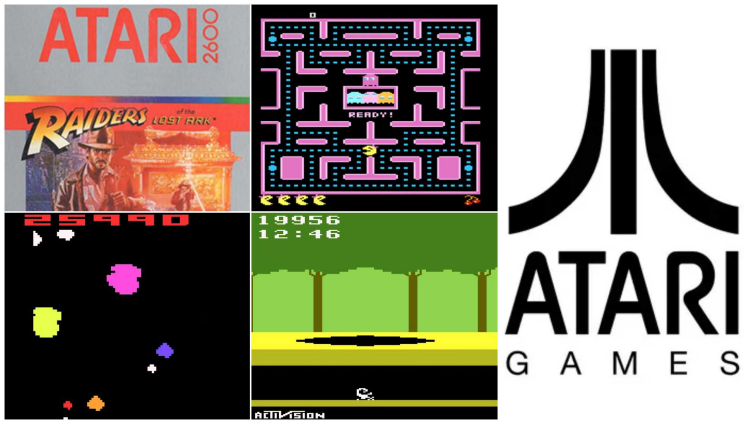
Состояний теперь много

Постановка задачи

Теперь в задаче удалим ограничение на множество состояний среды $\cancel{|S| \ll \infty}$.

Пример

Примером таких задач могут послужить игры ATARI (*benchmark RL* алгоритмов).



Препроцессинг состояний на практике

Action selection frequency

- Framestack
- Frameskip
- MaxAndSkip
- Sticky actions

Atari-specific preprocessing

- EpisodicLife
- FireReset

Standart Tricks

- Crop Image
- Rescale (84×84)
- Grayscale

Reward preprocessing

- Clip reward to $\{-1, 0, 1\}$

Состояний теперь много

Проблема

Так как теперь множество состоянии есть непрерывная величина, то *Q-learning* теперь применять нельзя, так как Q-функция не может быть описана таблицей.

Решение

Deep Q-learning: давайте искать оптимальную $Q^*(s, a)$ в виде нейронной сети!

Возможные имплементации DQN

Первый вариант

$$\text{Q-Net} := Q(s, a) \in \mathbb{R}$$

Второй вариант

$$\text{Q-Net} := (Q(s, a_1), \ldots, Q(s, a_A)) \in \mathbb{R}^A$$

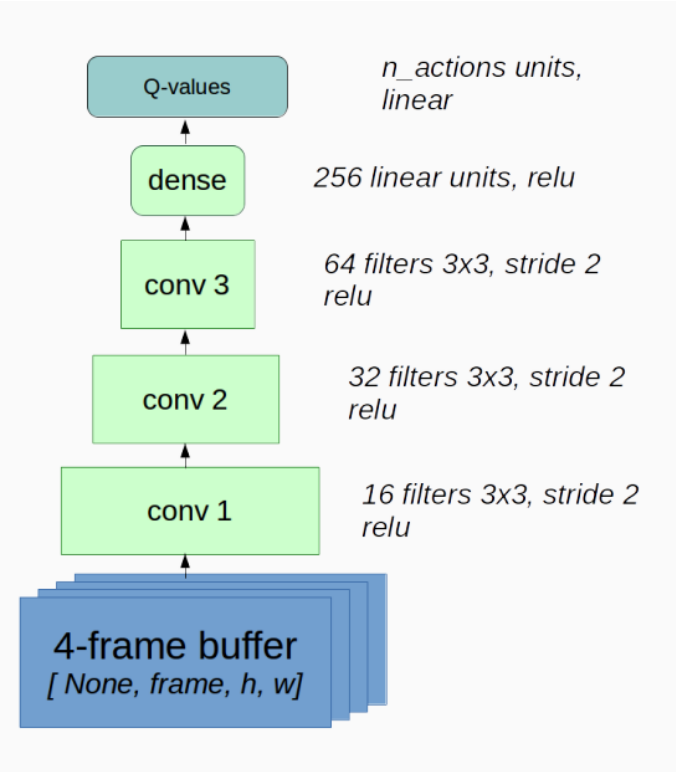
Вопрос

Что лучше использовать и почему?

Ответ

Если $|A| \ll \infty$, то удобнее воспользоваться вторым вариантом, поскольку так удобнее брать $\arg\max_a$.

Архитектура



Не стоит использовать

- max pooling — ведет к инвариантности сети относительно сдвигов объектов
- batch norm — см. drop out
- drop out — *train* и *test* стадии не отличимы для алгортма

Обучение DQN

Задача регрессии

Пусть s, a — входные данные, $f \in \mathbb{R}$ — искомые значения, которые в идеале хочется брать из уравнения Беллмана:

$$f(s,a) := r(s,a) + \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s', a', \theta_k)$$

где θ_k — параметры сети.

Проблема

Так же как и в табличном Q-обучении матожидание по s' берется из неизвестного (в общем случае) нам распределения. Нужно «обходить» эту операцию. Построим оптимизационную задачу, вычислим градиент и поймем, где можно выполнить приближение.

Обучение DQN

Задача регрессии

Пусть s, a — входные данные, $f \in \mathbb{R}$ — искомые значения, которые в идеале хочется брать из уравнения Беллмана:

$$f(s,a) := r(s,a) + \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s', a', \theta_k)$$

Функция потерь — *MSE Loss function*: $(f - y_{\text{pred}})^2/2$. Итого задача оптимизации:

$$\frac{1}{2}\mathbb{E}_{(s,a,f)}(f - Q^*(s,a,\theta_{k+1}))^2 \to \min_{\theta_{k+1}}$$

Данная задача на самом деле и является Q-обучением.

Обучение DQN

Вычисление градиента

$$\nabla_\theta \frac{1}{2}(f - Q^*(s,a,\theta))^2 = \underbrace{(f - Q^*(s,a,\theta))}_{\text{скаляр со знаком}} \overbrace{\nabla_\theta Q^*(s,a,\theta)}^{\text{направление увеличения значения Q}} =$$

$$= \left(r + \gamma \underline{\mathbb{E}_{s'}} \max_{a'} Q^*(s', a', \theta_k) - Q^*(s,a,\theta) \right) \nabla_\theta Q^*(s,a,\theta) \approx$$

$$\approx \left(r + \gamma \max_{a'} Q^*(\underline{s'}, a', \theta_k) - Q^*(s,a,\theta) \right) \nabla_\theta Q^*(s,a,\theta),$$

где $s' \sim p(s'|s,a)$, то есть приближаем математическое ожидание по одному сэмплу. Первая скобка полученного есть и в алгоритме Q-обучения.

Обучение DQN

Итоговая задача регрессии

Если s, a — входные данные, $y \in \mathbb{R}$ — наблюдаемые значения, которые будут получаться как несмещ. оценка f :

$$y(s,a) := r(s,a) + \gamma \max_{a'} Q^*(s', a', \theta_k)$$

Где $s' \sim p(s'|s,a)$.

Вывод

С помощью приближения уравнения Беллмана сэмплированием мы получил задачу, которая является уже изученным *Q-learning*, только вместо таблицы приближаем искомое Q^* нейросетью.

Обучение DQN

Итого мы поняли, что будем решать задачу с целевыми значениями $y : \mathbb{E}_{s'} y = f(s, a)$. Почему алгоритм сойдется к f?

Теорема

Пусть $Q_{\theta_{k+1}}(s, a)$ — достаточно ёмкая модель, выборка неограниченно большая, а оптимизатор идеальный. Тогда решением задачи регрессии выше будет:

$$Q_{\theta_{k+1}}(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_k}(s', a').$$

Обучение DQN

Доказательство

Найдем оптимальное значение $Q_{\theta_{k+1}}(s, a)$ для позиции (s, a), которое минимизирует MSE:

$$\frac{1}{2}\mathbb{E}_{s'}(y(s, a) - Q_{\theta_{k+1}}(s, a))^2.$$

Вспоминаем как мы брали градиент и строили его несмещенную оценку, приравниваем несмещенную оценку градиента (а значит, в среднем, и сам градиент) к нулю откуда получаем:

$$Q_{\theta_{k+1}}(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_k}(s', a').$$

Обучение DQN

Замечание

Построенная и обоснованная задача регрессии на самом деле является не очень типичной для машинного обучения: наши целевые значения зависят от аппроксимирующей функции с фиксированными весами: $y(s,a) := r(s,a) + \gamma \max\limits_{a'} Q^*(s',a',\theta_k)$ (градиенты в целевые значния не текут!).

Схема обучения: один шаг простой итерации

1. Фиксируются веса θ_k, вычисляются целевые значения y_k.
2. Модель $Q^*(s,a,\theta)$ обучается на эти целевые значения.
3. Меняются веса $\theta_k \to \theta_{k+1}$, вычисляются новые целевые значения y_{k+1}.

Обучение DQN

Схема обучения: один шаг простой итерации

1. Фиксируются веса θ_k, вычисляются целевые значения y_k.
2. Модель $Q^*(s, a, \theta)$ обучается на эти целевые значения.
3. Меняются веса $\theta_k \to \theta_{k+1}$, вычисляются новые целевые значения y_{k+1}.

Проблема

Переход со второго пункта на третий пункт обманчиво прост. Доколе нам обучать сетку с фиксированными таргетами? Как понять, что пора обновить веса сетки и пересчитать целевые значения?

Обучение DQN

Возможные варианты

1. *SGD*: Можно пересчитывать целевые значения после каждого градиентного шага обучения.

Недостаток

Подход не стабилен. Градиентный шаг может немного «поломать» модель, ведь *SGD* улучшает модель лишь в среднем, но не гарантирует этого на каждом шаге.

2. *До сходимости*: будем обучаться, пока веса модели не перестанут значимо меняться.

Недостаток

Ждать сходимости слишком долго.

Обучение DQN

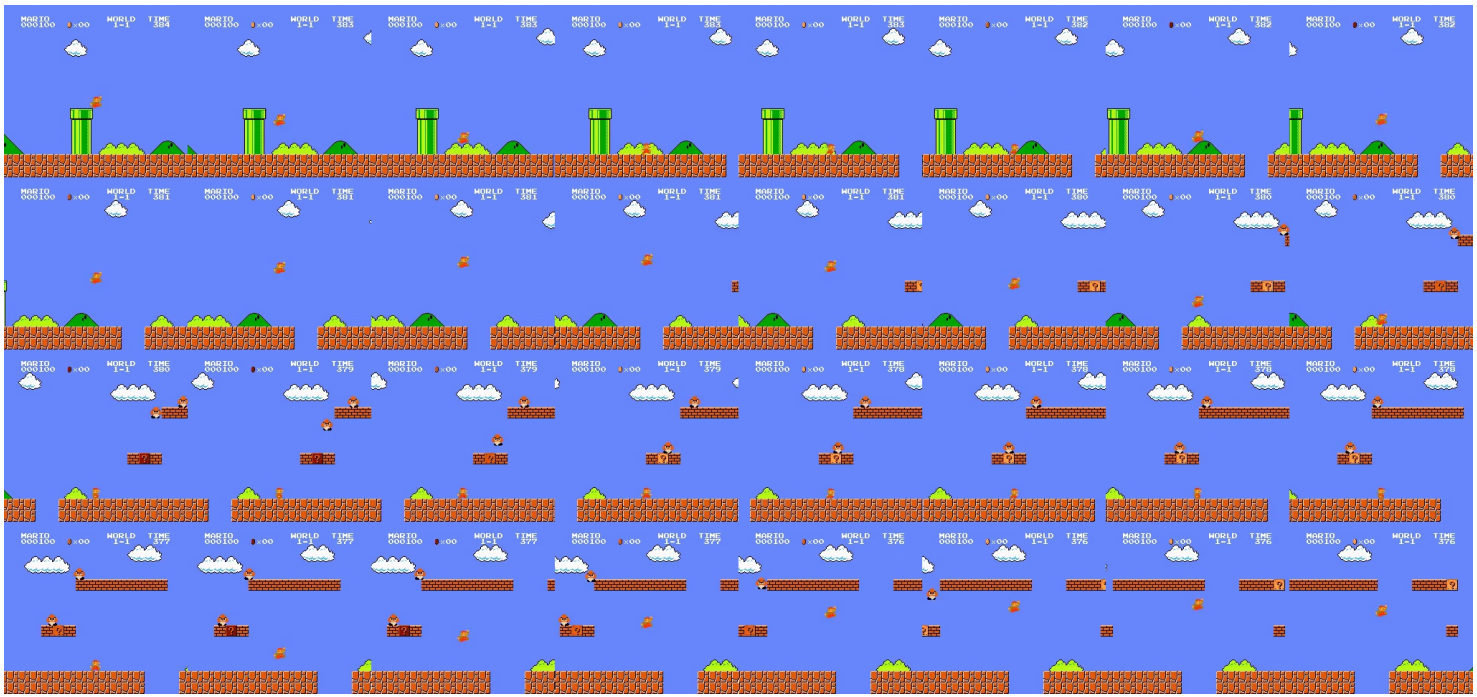
Возможны варианты

3. *Что-то между*. Предлагается выполнять условно $100 - 1000$ итераций градиентного обучения сети прежде, чем менять целевые значения.

Декорреляция сэмплов

Заметим

Обучать *DQN* мы будем по батчам, но есть нюанс: при последовательной игре четверки (s, a, r, s') часто очень коррелированы, что ломает обучение сети.



Декорреляция сэмплов

Пути решения

1. Запуск параллельных агентов.
2. Буфер реплеев.

Target Network

Имплементация

Предложенный алгоритм фиксации целевых значений и последующего градиентного спуска реализован с помощью *Target Net*, генерирующая целевые значения, на которых несколько итераций обучается *Deep Q-Network*, после чего все веса *DQN* копируются в *Target Net* и так далее.

Замечание

Веса *Target Net* часто обозначаются за θ^-.

Замечание

Полученный результат всё так же является *model-free off-policy* алгоритмом, в котором всё так же необходим *exploration*.

Финальная схема *Deep Q-learning*

Инициализируем $Q^*(s, a, \theta)$ произвольно, $\theta^- := \theta, \mathcal{D} = \varnothing$;
Получаем s_0 : **for** $k = 0, 1, 2, \ldots$

1. Выбрать действие $a_k \sim \varepsilon - \text{greedy}(Q^*(s_k, a, \theta))$;
2. Пронаблюдать $r_k, s_{k+1}, \text{done}_{k+1}$ и сохранить $(s_k, a_k, r_k, s_{k+1}, \text{done}_{k+1})$ в $\mathcal{D}$, $\text{done}_{k+1} := \mathbb{1}_{\{s_{k+1} \text{ терминальное}\}}$;
3. Засэмплировать батч $\mathbb{T} = \left\{(s_{i_j}, a_{i_j}, r_{i_j}, s_{i_j+1}, \text{done}_{i_j+1})\right\}_{j=1}^{B}$ из $\mathcal{D}$;
4. $y_j := r_{i_j} + \gamma(1 - \text{done}_{i_j+1}) \max\limits_{a'} Q^*(s_{i_j+1}, a', \theta^-)$;
5. Совершить шаг градиентного спуска:

$$\theta \leftarrow \theta - \frac{\alpha_k}{2B} \sum_{j=1}^{B} \nabla_\theta (Q^*(s_{i_j}, a_{i_j}, \theta) - y_j)^2, \ \alpha_k \text{ согласно Robbins-Monro;}$$

6. Обновляем *Target Net*, если $k \mod K = 0 : \theta^- \leftarrow \theta$.

Зачем вообще что-то модифицировать?

Проблема

Выученная функция склонна к переоцениванию будущей награды, то есть Q-функция начинает неограниченно расти. Эта проблема называется *overestimation bias*.

Почему это происходит?

Это возникает из-за оператора максимума в формуле построения целевых значений:

$$y(\mathbb{T}) = r + \gamma \max_{a'} Q_{\theta^-}(s', a').$$

Зачем вообще что-то модифицировать?

Природа *overestimation*

При обучении сетки ошибка появляется из-за неточности аппроксимации и случайной шумовой ошибки, но функции max это индифферентно — он всегда выбирает лучшее, поэтому случайно или аппроксимационно завышенные оценки будут накапливаться.

Зачем вообще что-то модифицировать?

Формальное утверждение

Пусть s' фиксировано, $Q^*(s', a')$ — истинная Q-функция, а $Q(s', a') \approx Q^*(s', a')$ — её приближение:

$$Q(s', a') = Q^*(s', a') + \varepsilon(a'),$$

где $P(\varepsilon(a') > 0) = P(\varepsilon(a') < 0) = 0.5$ и этот шум не зависит от выбора a' в вероятностном смысле. Тогда:

$$P(\max_{a'} Q(s', a') > \max_{a'} Q^*(s', a')) > 0.5.$$

Action decoupling

Отделение выбора действия от оценки

$$\max_{a'} Q^*(s', a') = \overbrace{Q^*(s', \underbrace{\arg\max_{a'} Q^*(s', a')}_{\text{action selection}})}^{\text{action evaluation}}$$

Возможные улучшения

Name	Networks	Targets
Double* Q-learning	Q_1 Q_2	$y_1 = r + \gamma Q_2(s', \arg\max_{a'} Q_1(s', a'))$ $y_2 = r + \gamma Q_1(s', \arg\max_{a'} Q_2(s', a'))$
Double Q-learning	Q Q_- — TN	$y = r + \gamma Q_-(s', \arg\max_{a'} Q(s', a'))$
Twin Q-learning	Q_1 Q_2	$y_1 = r + \gamma \min_{i=1,2} Q_i(s', \arg\max_{a'} Q_1(s', a'))$ $y_2 = r + \gamma \min_{i=1,2} Q_i(s', \arg\max_{a'} Q_2(s', a'))$

Prioritized Experience Replay

Проблема

В буфере реплеев чаще всего оказываются тривиальные случаи, когда агент сделал шаг и не получил никакой награды, что плохо влияет на обучение, с другой стороны случаи с наградой довольно редки, а они на начальных порах дают агенту намеки, где искать улучшение.

Решение

Сэмплировать повторы из буфера будем не равномерно, а приоритезировано с помощью весов сэмплирования:

$$P(\mathbb{T}) \propto |y(\mathbb{T}) - Q^*(s, a, \theta)|^\alpha.$$

Prioritized Experience Replay

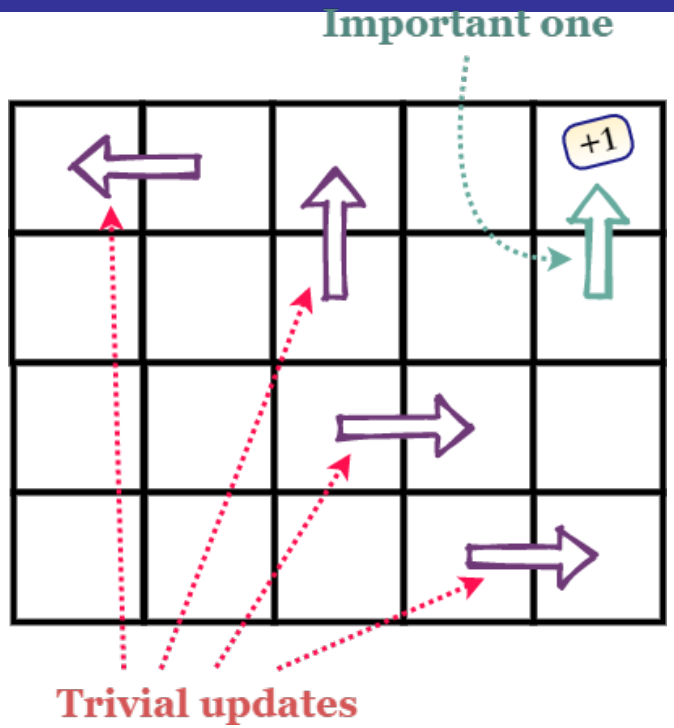
Недостаток

После неконтролируемой замены равномерного сэмплирования на какое-то другое, могло случиться так, что для наших переходов $s' \not\sim p(s'|s,a)$. Иными словами, приоритизированное сэмплирование приводит к *смещению*.

Утешение

Этот эффект не так страшен поначалу обучения, когда распределение, из которого приходят состояния, всё равно скорее всего не сильно разнообразно. Более существенно нивелировать этот эффект по ходу обучения, в противном случае процесс обучения может полностью дестабилизироваться или где-нибудь застрять.

Prioritized Experience Replay



Проблема: много тривиальных обновлений.

Цель: распространить подкрепление (награду) из будущего в прошлое.

Сэмплирование с приоритетами из буфера:

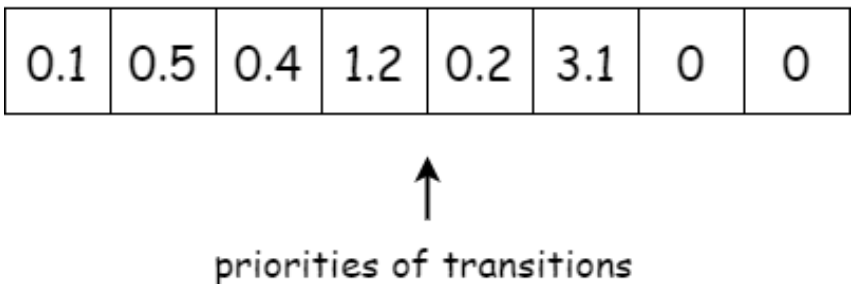
$$\mathrm{P}(\mathbb{T}) \propto |y(\mathbb{T}) - Q(s, a, \theta)|^{\alpha}.$$

Проблемы? Теперь $s' \not\sim p(s' \mid s, a)$.

$$\mathbb{E}_{\mathbb{T} \sim \mathrm{Uniform}} \mathrm{Loss}(\mathbb{T}) \approx \mathbb{E}_{\mathbb{T} \sim \mathrm{P}(\mathbb{T})} \underbrace{\left(\frac{1}{\mathrm{P}(\mathbb{T})}\right)^{\beta(t)}}_{\text{вес}} \mathrm{Loss}(\mathbb{T}),$$

где для $\beta(t)$ выполняется отжиг от 0 (смещённая оценка) до 1 (несмещённая оценка).

Техническая деталь: структура SumTree



Использование SumTree:

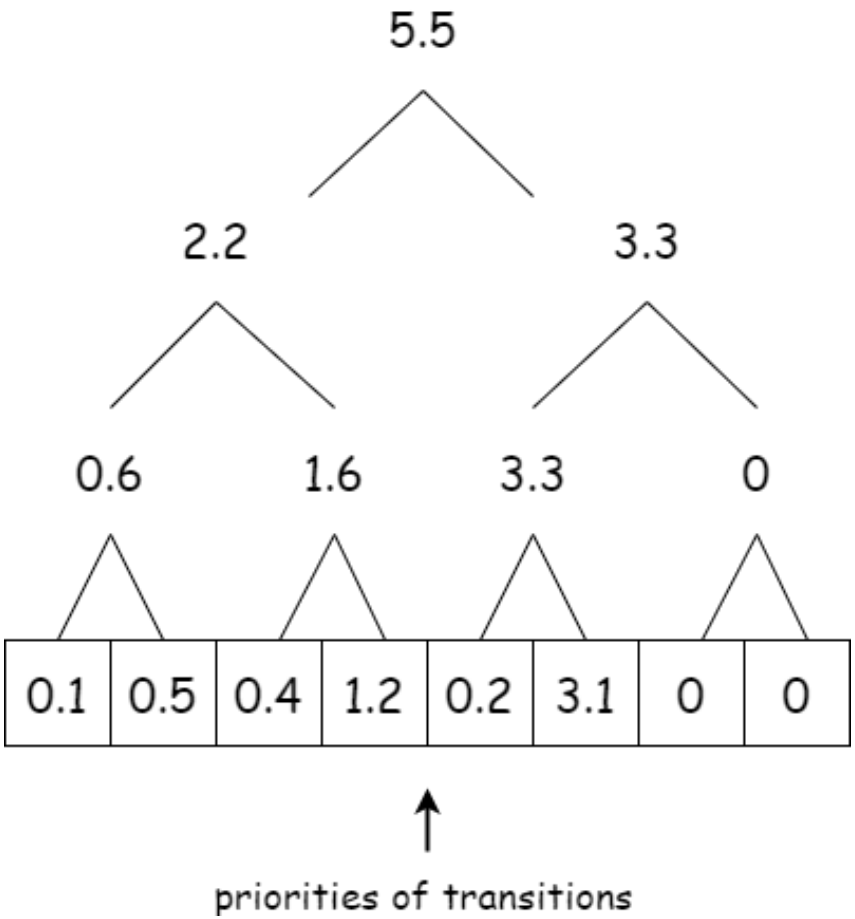
- листья – значения приоритета сэмплирования;
- каждый узел – сумма двух детей;

Буферизация:

- сэмплировать из $\mathrm{P}(\mathbb{T})$ с помощью равномерного распределения на $[0, \sum \mathrm{P}(\mathbb{T})]$;
- обновление параметров;
- пересчёт приоритетов **только для текущего батча**;
- обновить узлы SumTree;

Сложность: $O(\log N)$, где N размер буфера.

Техническая деталь: структура SumTree



Использование SumTree:

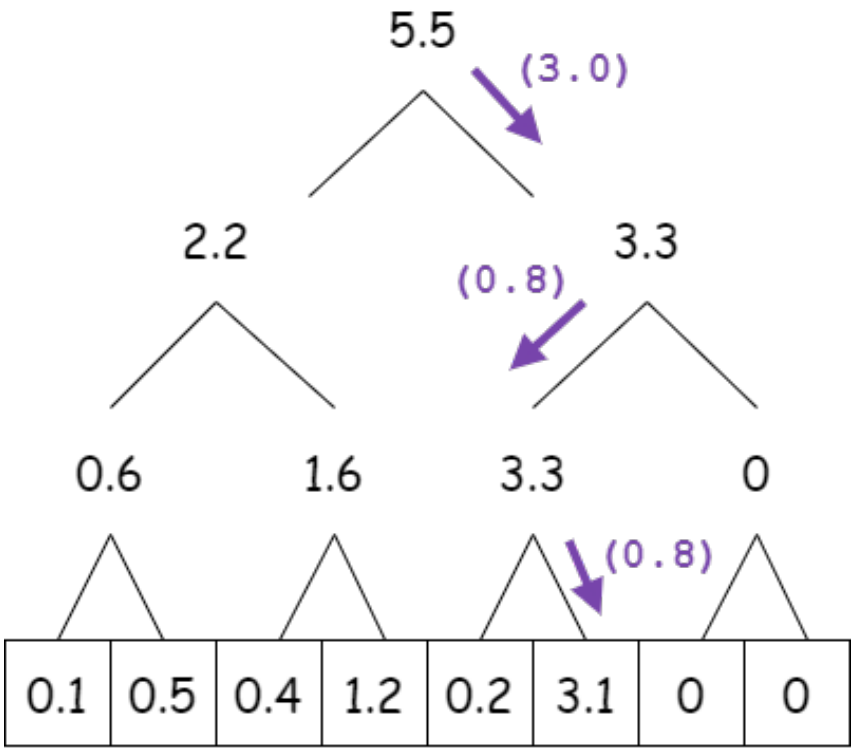
- листья – значения приоритета сэмплирования;
- каждый узел – сумма двух детей;

Буферизация:

- сэмплировать из $\mathrm{P}(\mathbb{T})$ с помощью равномерного распределения на $[0, \sum \mathrm{P}(\mathbb{T})]$;
- обновление параметров;
- пересчёт приоритетов **только для текущего батча**;
- обновить узлы SumTree;

Сложность: $O(\log N)$, где N размер буфера.

Техническая деталь: структура SumTree



Использование SumTree:

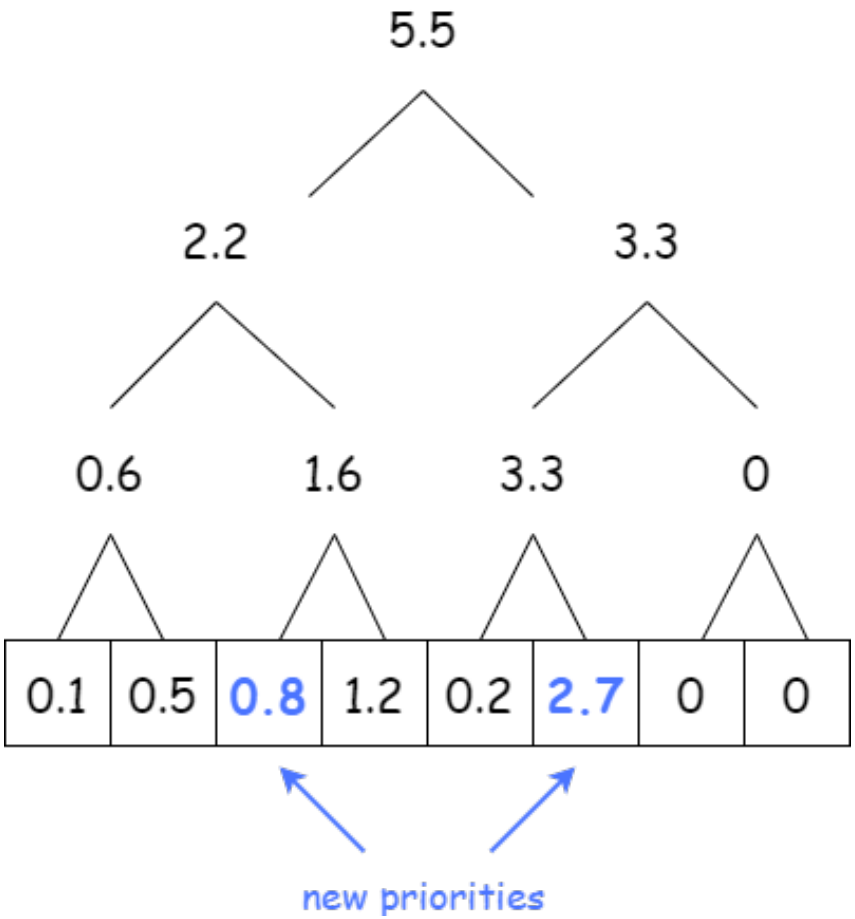
- листья – значения приоритета сэмплирования;
- каждый узел – сумма двух детей;

Буферизация:

- сэмплировать из $\mathrm{P}(\mathbb{T})$ с помощью равномерного распределения на $[0, \sum \mathrm{P}(\mathbb{T})]$;
- обновление параметров;
- пересчёт приоритетов **только для текущего батча**;
- обновить узлы SumTree;

Сложность: $O(\log N)$, где N размер буфера.

Техническая деталь: структура SumTree



Использование SumTree:

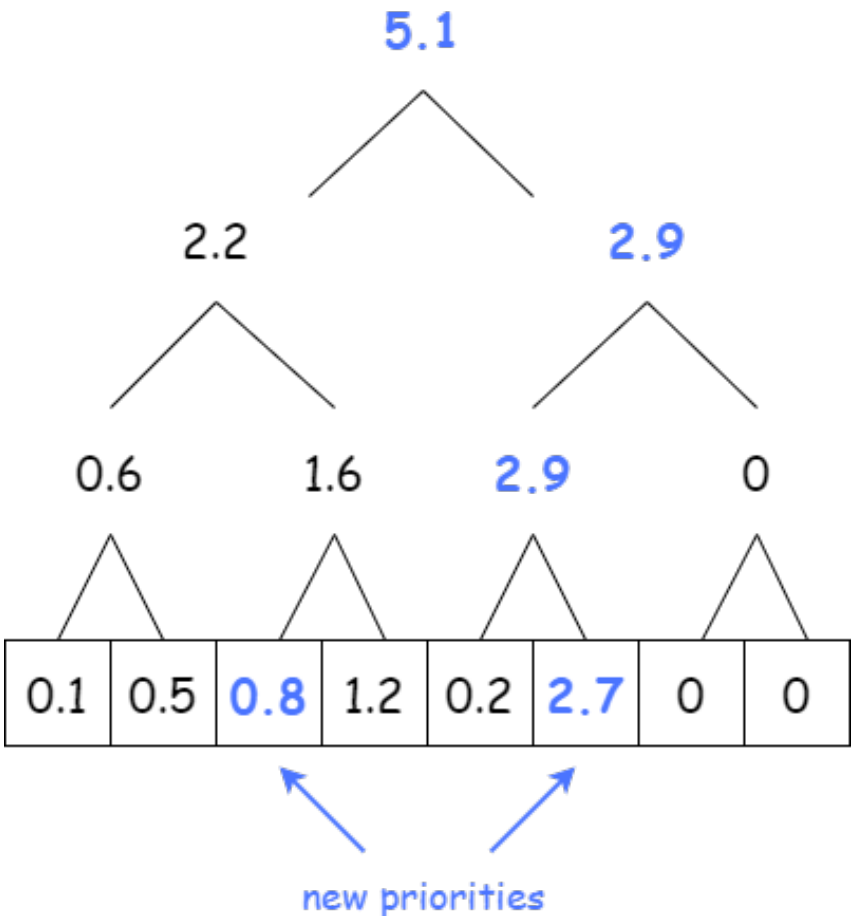
- листья – значения приоритета сэмплирования;
- каждый узел – сумма двух детей;

Буферизация:

- сэмплировать из $\mathrm{P}(\mathbb{T})$ с помощью равномерного распределения на $[0, \sum \mathrm{P}(\mathbb{T})]$;
- обновление параметров;
- пересчёт приоритетов **только для текущего батча**;
- обновить узлы SumTree;

Сложность: $O(\log N)$, где N размер буфера.

Техническая деталь: структура SumTree



Использование SumTree:

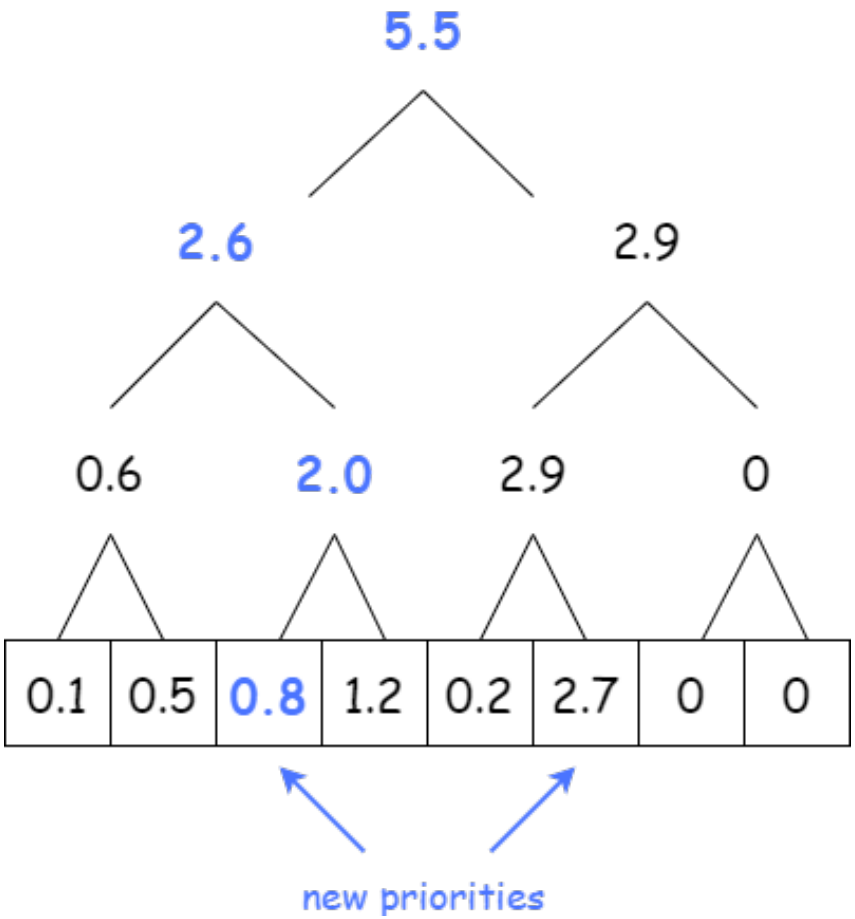
- листья – значения приоритета сэмплирования;
- каждый узел – сумма двух детей;

Буферизация:

- сэмплировать из $\mathrm{P}(\mathbb{T})$ с помощью равномерного распределения на $[0, \sum \mathrm{P}(\mathbb{T})]$;
- обновление параметров;
- пересчёт приоритетов **только для текущего батча**;
- обновить узлы SumTree;

Сложность: $O(\log N)$, где N размер буфера.

Техническая деталь: структура SumTree



Использование SumTree:

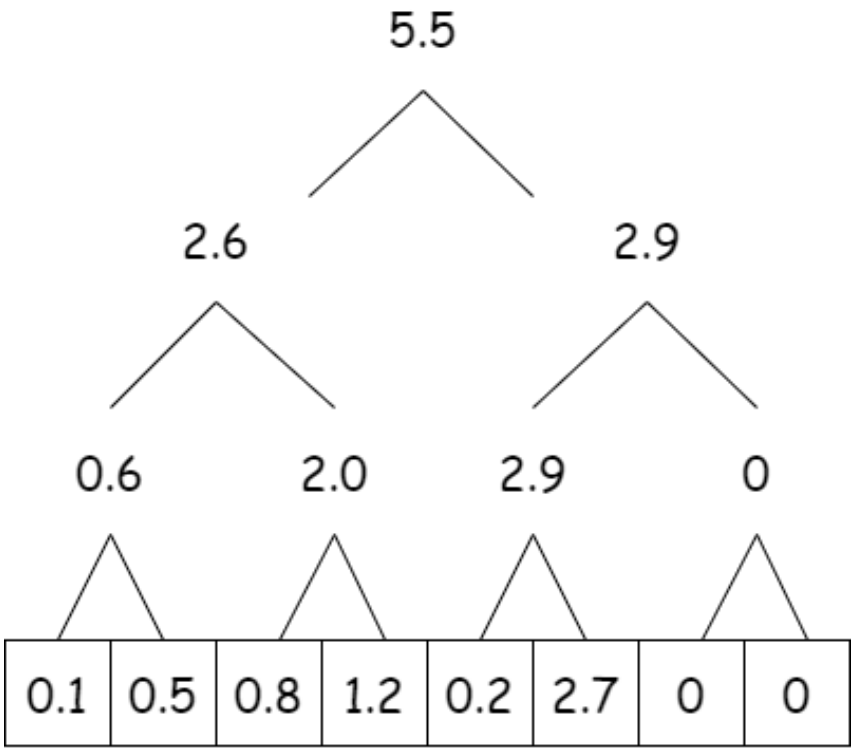
- листья – значения приоритета сэмплирования;
- каждый узел – сумма двух детей;

Буферизация:

- сэмплировать из $\mathrm{P}(\mathbb{T})$ с помощью равномерного распределения на $[0, \sum \mathrm{P}(\mathbb{T})]$;
- обновление параметров;
- пересчёт приоритетов **только для текущего батча**;
- обновить узлы SumTree;

Сложность: $O(\log N)$, где N размер буфера.

Техническая деталь: структура SumTree



Использование SumTree:

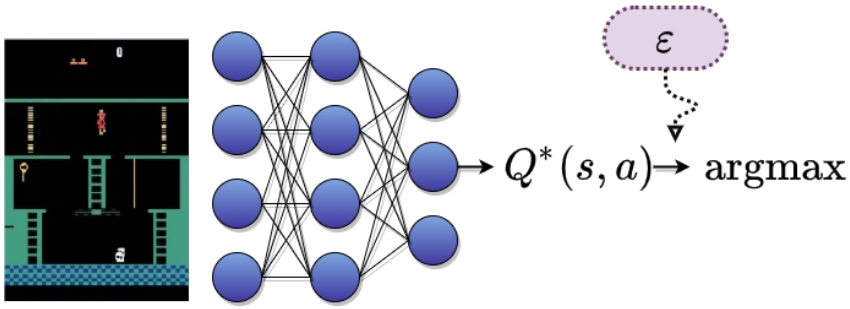
- листья – значения приоритета сэмплирования;
- каждый узел – сумма двух детей;

Буферизация:

- сэмплировать из $\mathrm{P}(\mathbb{T})$ с помощью равномерного распределения на $[0, \sum \mathrm{P}(\mathbb{T})]$;
- обновление параметров;
- пересчёт приоритетов **только для текущего батча**;
- обновить узлы SumTree;

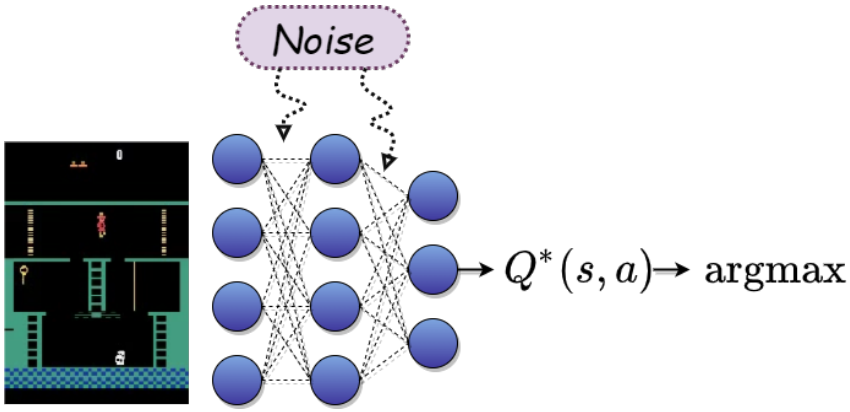
Сложность: $O(\log N)$, где N размер буфера.

Noisy Networks



Проблема: ε-жадная стратегия наивна:

- × неудобный гиперпараметр;
- × exploration вне зависимости от состояния.

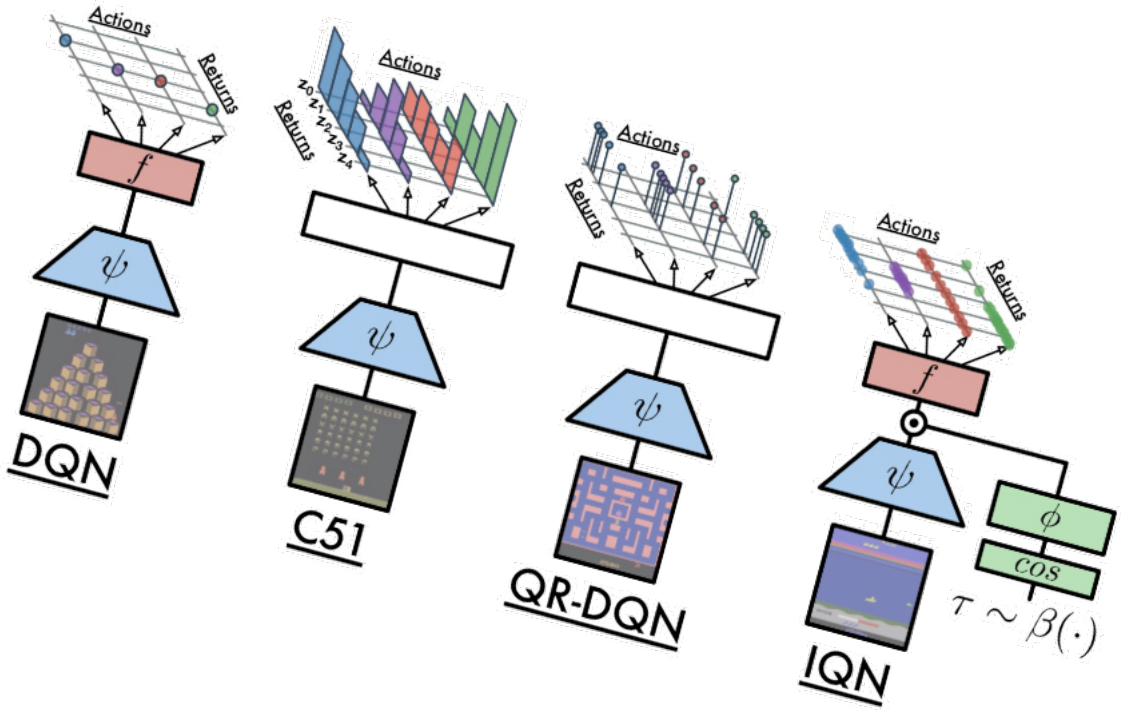


Решение: добавить шум на веса:

$$w := \mu + \sigma\varepsilon, \quad \varepsilon \sim \mathcal{N}(0,1).$$

- ✓ нет гиперпараметров;
- ✓ зависит от состояния (не интерпретируемо);
- × дорогое вычисление.

Distributional RL



Dueling DQN

Проблема: оценка пары состояние-действие требует знание о *всех* действиях.

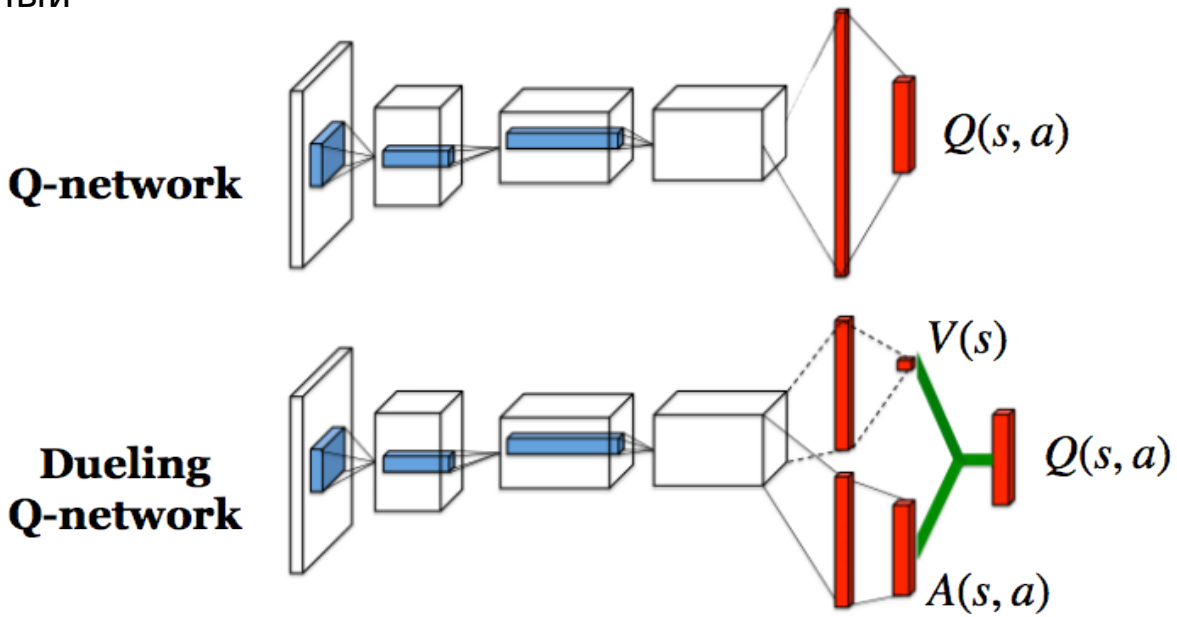
$$Q^*(s,a) := V^*(s) + \underbrace{A^*(s,a)}_{\text{не произвольный}}$$

Аппроксимация Q-функции через уравнение Беллмана относительно V-функции.

Внимание!

Добавляется **лишняя степень свободы!**

$$Q_\theta(s,a) := V_\theta(s) + A_\theta(s,a) - \underbrace{\underset{a}{\text{mean}}\, A_\theta(s,a)}_{\text{согласно теории: взять максимум}}$$

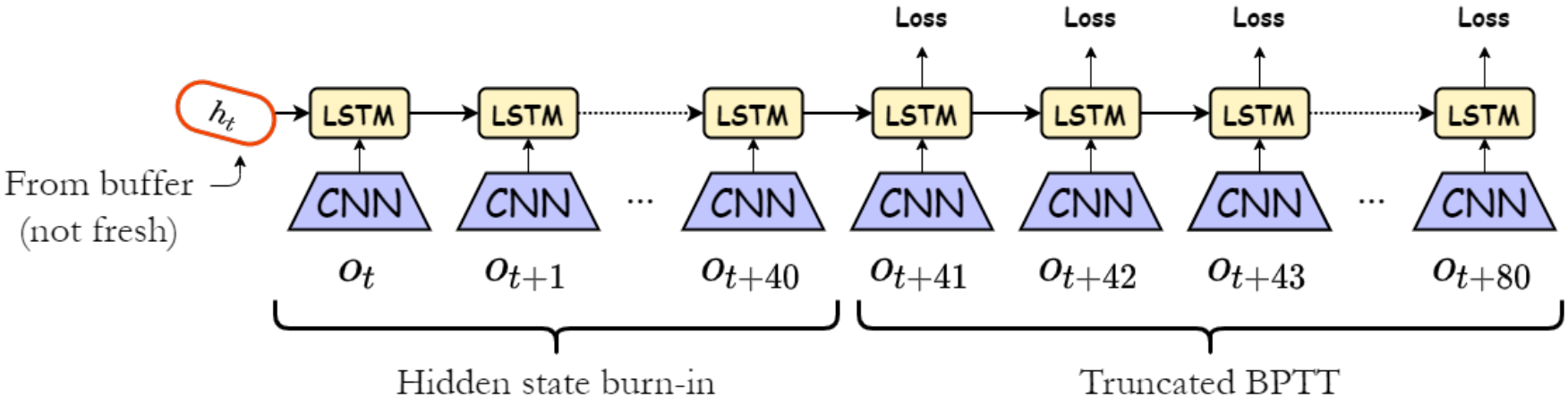


Partially Observable MDP

Проблема: есть доступ только к наблюдениям:

$$o_t \sim p(\cdot \mid s_t).$$

Теория: смотри PoMDP (частично наблюдаемые MDP);
Практика: с помощью сети LSTM.



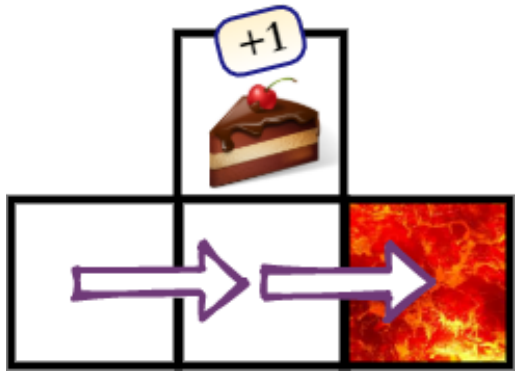
Multi-step DQN

Проблема: отложенные награды + ошибки вычисления.

$$y := \underbrace{r + \gamma r' + \gamma^2 r'' + \dots}_{\approx\text{«оптимальное поведение»?}} + \gamma^N \max_{a^{(N)}} Q(s^{(N)}, a^{(N)}).$$

Внимание!

- ✓ может *значительно* помочь;
- × в теории не для off-policy.



Практика: хранить в буфере кортежи вида как справа.

$$\left(s, \quad a, \quad \sum_{n=0}^{N-1} \gamma^n r^{(n)}, \quad s^{(N)}, \quad \bigvee_{n=1}^{N} \text{done}_n \right)$$

Передний край науки и индустрии

Rainbow DQN
(2017)

- Double DQN
- Буфер с приоритетами
- Multi-step DQN
- Noisy Nets
- Dueling DQN
- *Distributional RL*

(в следующий раз)

R2D2
(2018)

- Double DQN
- Буфер с приоритетами
- Multi-step DQN
- \+ LSTM
- \+ Массовая параллелизация

Agent 57
(2020)

- Double DQN
- Буфер с приоритетами
- Multi-step DQN
- LSTM
- Массовая параллелизация
- \+ Retrace
- \+ Внутренняя мотивация
- \+ Управление гиперпараметрами

Источники:

- Playing Atari with Deep Reinforcement Learning (2013);
- *Ссылки на модификации DQN перечислены на предыдущем слайде.*