Обучение с подкреплением

Deep Q-network

Ограничения

1) ~~Должны знать T и R~~
2) Дискретное пространство состояний
3) Дискретное пространство действий

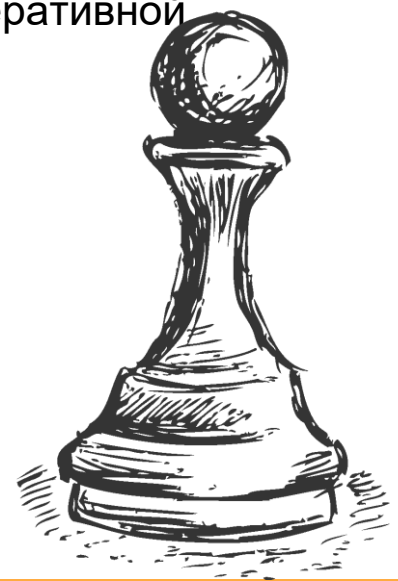
Пусть все оценки храним во float. Размер 4 байта. Пусть есть 16 Гб оперативной памяти. Сколько всего можем хранить пар состояние-действие?

$$\frac{16 \cdot 2^{30}}{4} = 4 \cdot 2^{30} \approx 4 \cdot 10^9$$

Сколько всего возможных позиций в игре в шахматы?

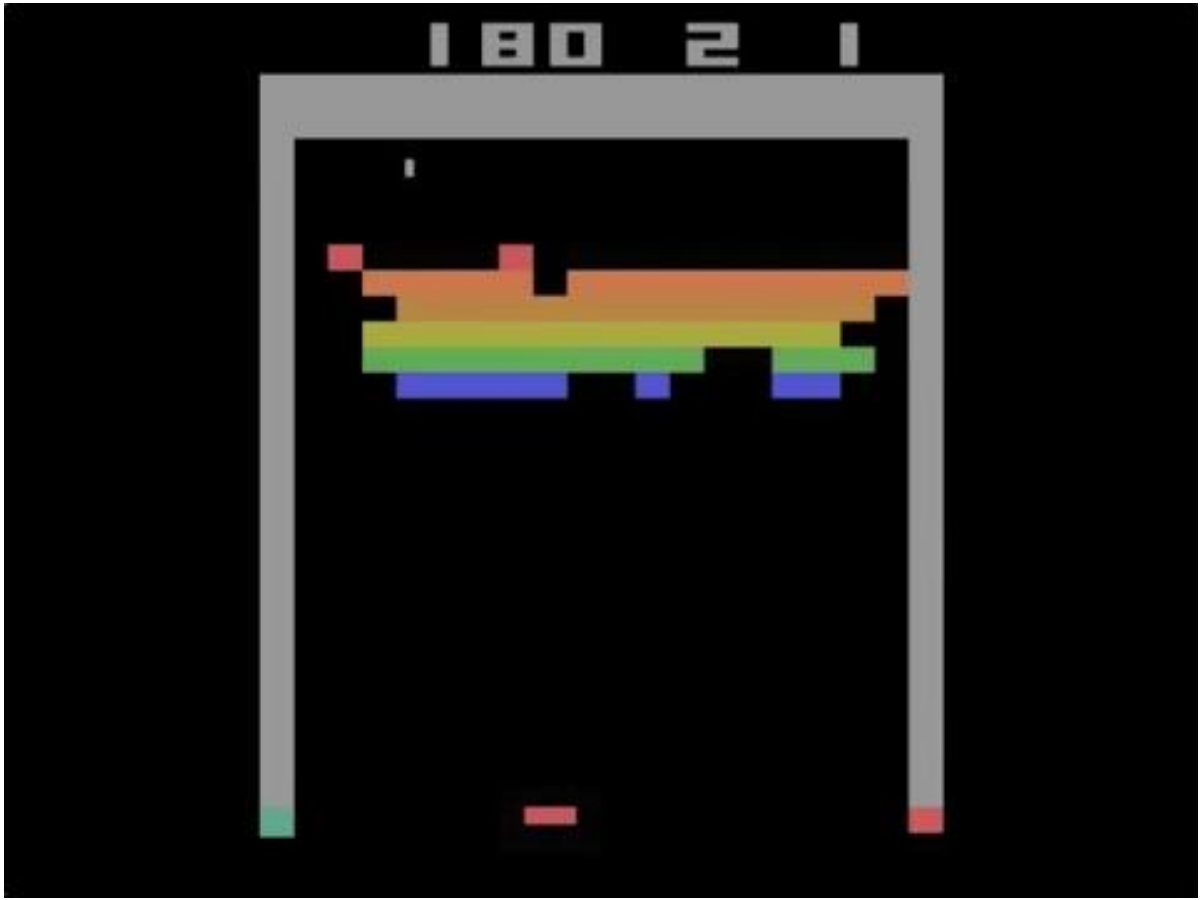
Оценка сверху (число Шеннона): 10^{43}

После первых 10 ходов (точное число): $69352859712417 \approx 69 \cdot 10^{12}$



Дискретное пространство состояний

Deep Reinforcement Learning



Q-learning

Алгоритм:

1. Инициализируем **Q(s, a)**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
   1. **a** := argmax **Q(s, .)** с вероятностью 1-**eps**, иначе **a** := random(**A**)
   2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
      $$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \cdot \max_{a'} Q(s',a'))$$
   3. **s** := **s'** если не **d**, иначе реинициализируем среду

Deep Q-Network

Алгоритм:

1. Инициализируем **Q(s, a)**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
   1. **a** := argmax **Q(s, .)** с вероятностью 1-**eps**, иначе **a** := random(**A**)
   2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
      $$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \cdot \max_{a'} Q(s',a'))$$
   3. **s** := **s'** если не **d**, иначе реинициализируем среду

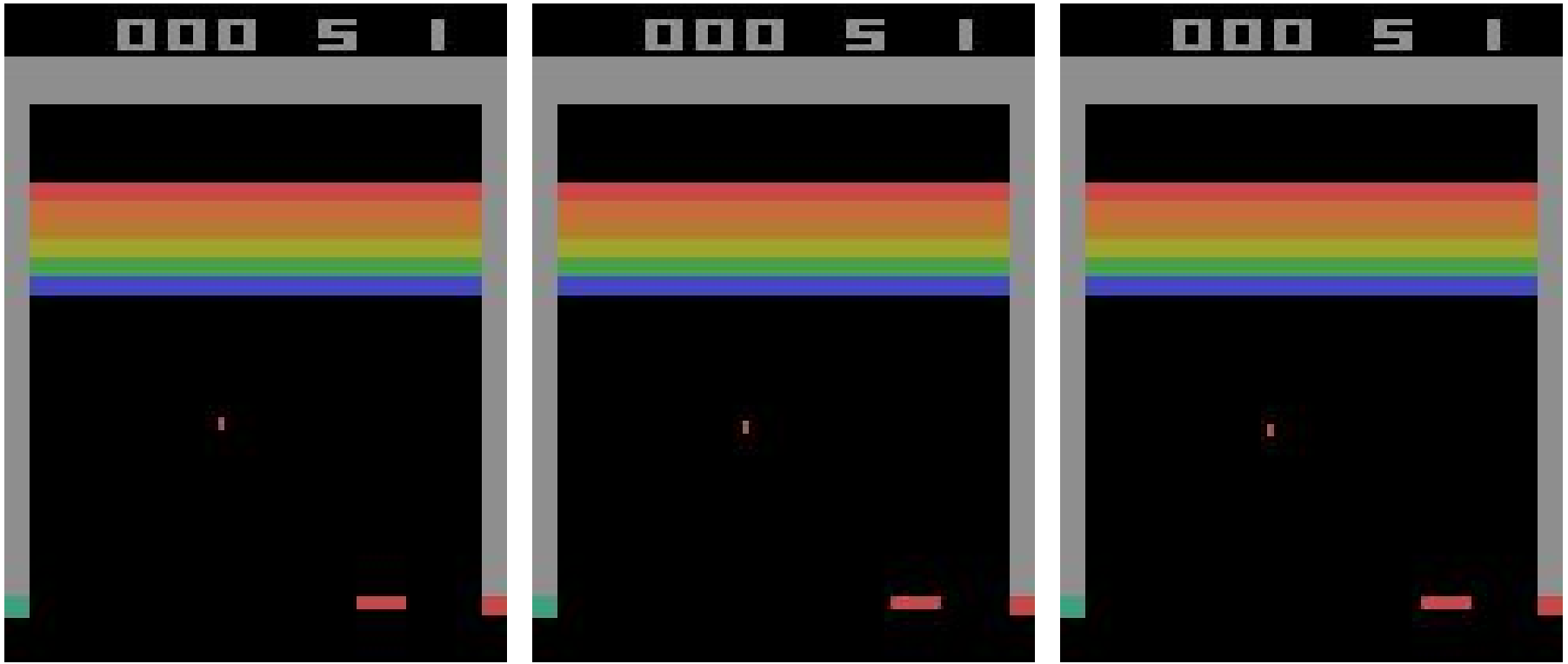
Теперь Q(s, a) будем приближать нейронной сетью.

Проблема 1: не можем использовать формулу из пункта 3.3

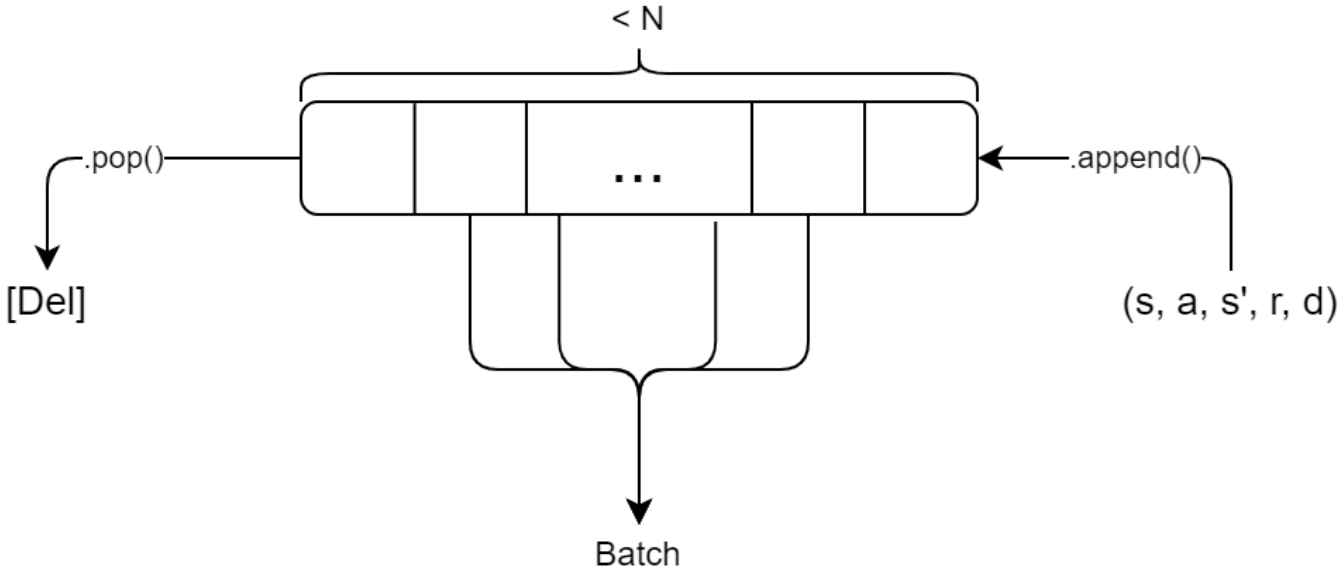
Проблема 2: градиентный спуск использует обучение батчами

Deep Q-network

Три последовательных семпла в игре Breakout:



Replay Buffer



- Новые данные добавляем в конец, из начала удаляем старые данные
- Батчи собираем из случайных элементов

Deep Q-Network

Алгоритм:

1. Инициализируем **Q(s, a)**, создаем пустой буфер **B**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
   1. **a** := argmax **Q(s, .)** с вероятностью 1-**eps**, иначе **a** := random(**A**)
   2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
   3. Добавляем **(s, a, s', r, d)** в **B**
   $$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \cdot \max_{a'} Q(s',a'))$$
   4. **s** := **s'** если не **d**, иначе реинициализируем среду

Проблема 1: не можем использовать формулу из пункта 3.4

Решение: Надо заменить на обновление по батчу из **B** с помощью MSE

$$L = MSE(Q(s,a), r + \gamma \cdot \max_{a'} Q(s',a'))$$

Deep Q-Network

Алгоритм:

1. Инициализируем **Q(s, a)**, создаем пустой буфер **B**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
   1. **a** := argmax **Q(s, .)** с вероятностью 1-**eps**, иначе **a** := random(**A**)
   2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
   3. Добавляем **(s, a, s', r, d)** в **B**

      $$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \cdot \max_{a'} Q(s',a'))$$
   4. **s** := **s'** если не **d**, иначе реинициализируем среду

Проблема 1: не можем использовать формулу из пункта 3.4

Решение: Надо заменить на обновление по батчу из **B** с помощью MSE

$$L = MSE(Q(s,a), r + \gamma \cdot \max_{a'} Q_T(s',a'))$$

Фиксированная target-сеть

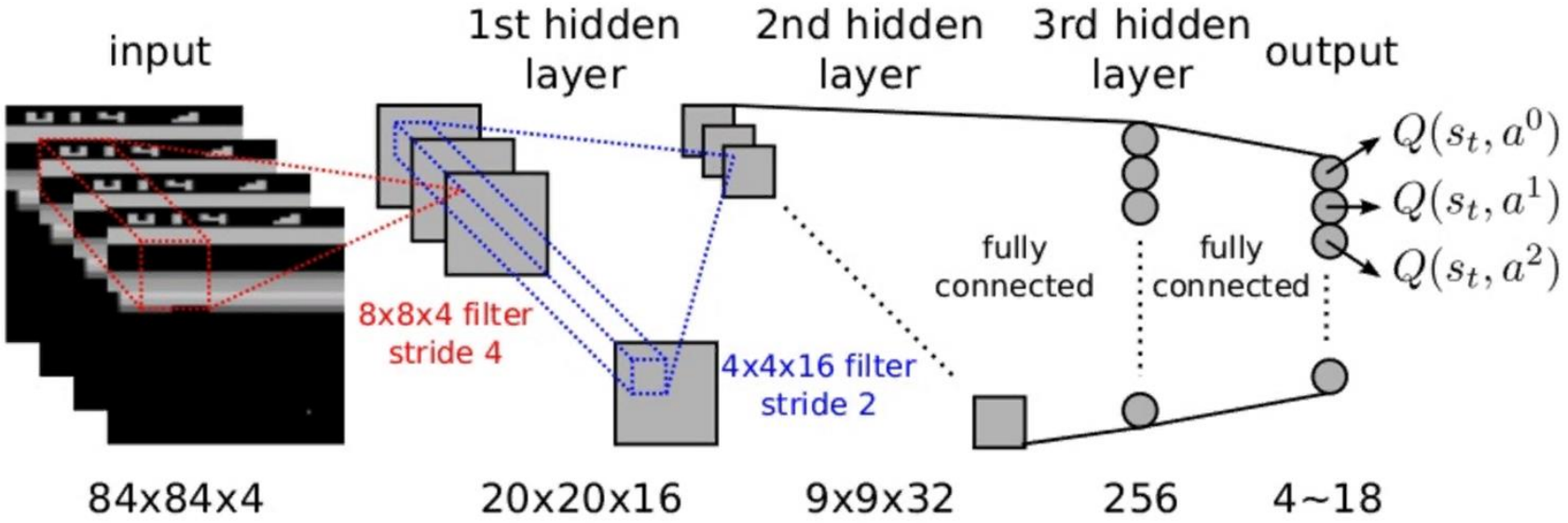
Deep Q-Network

Алгоритм:

1. Инициализируем **Q(s, a)**, создаем пустой буфер **B**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
   1. **a** := argmax **Q(s, .)** с вероятностью 1-**eps**, иначе **a** := random(**A**)
   2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
   3. Добавляем **(s, a, s', r, d)** в **B**
   4. Сэмплим батч **b ~ B**
   5. Обновляем Q с помощью $L = MSE(Q(s,a), r + \gamma \cdot \max_{a'} Q_T(s', a'))$
   6. **s** := **s'** если не **d**, иначе реинициализируем среду
   7. Если **i % N** = 0, то $\mathbf{Q_T}$:= **Q,** где **i** - номер шага

Deep Q-Network

Deep Q-Network (DQN) - алгоритм, использующий для приближения Q-function нейронную сеть. Он приближает значения Q-function сразу для всех действий.



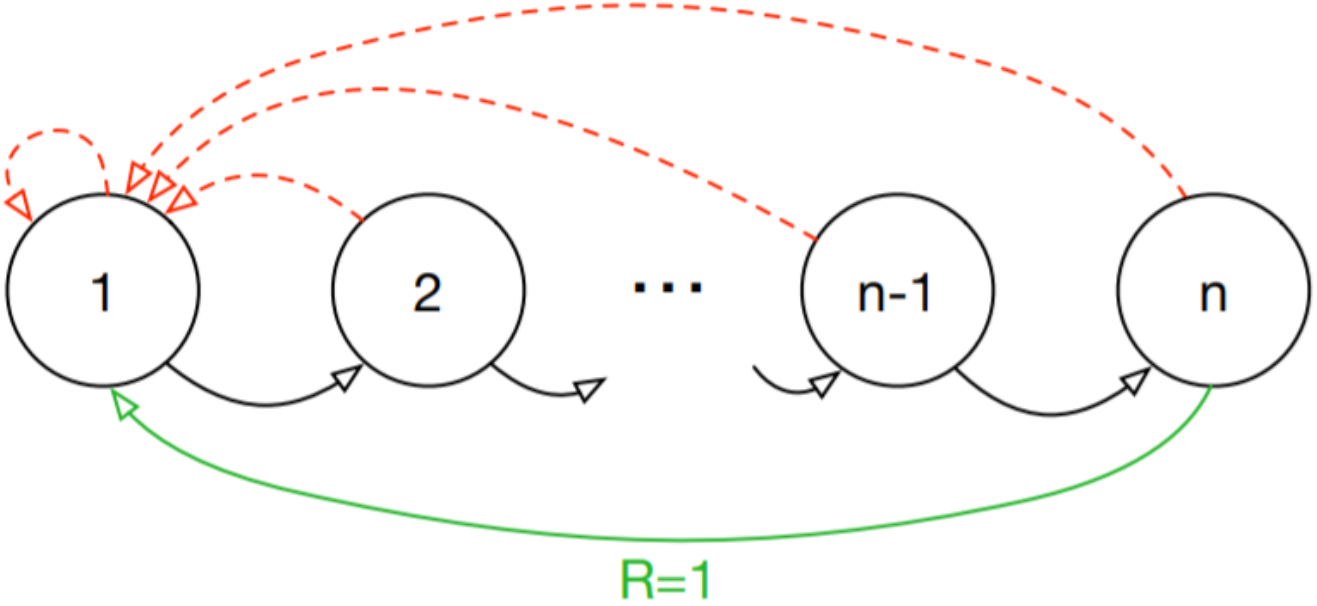
Deep Q-Network

1) Как и при обучении любой нейронной сети, для обновления весов используются батчи.
2) В качестве датасета для обучения сети используется очередь фиксированного размера, в которую складывается опыт, полученный агентом в процессе взаимодействия со средой.
3) На самом деле, есть две сети: Q и Q_T. Веса второй сети приравниваются весам первой сети раз в N батчей.
4) Первая сеть обновляется при помощи такой функции потерь:

$$L = MSE(Q(s,a), r + \gamma \cdot \max_{a'} Q_T(s',a'))$$

1) В случае, если s’ - конечное состояние, считаем, что $Q_T = 0$.

Prioritized Experience Replay



Prioritized Experience Replay

Идея: Будем чаще обучаться на том опыте, на котором больше ошибка

Берем опыт из буфера с вероятностью:

$$p((s, a, s', r, d)_i) = \frac{p_i^\alpha}{\sum p_j^\alpha}$$

А затем корректируем изменение распределения с помощью взвешенного обновления нейронной сети:

Importance Sampling Сглаживание

$$w_i = \frac{[N \cdot p(s, a, s', r, d)_i))]^{-\beta}}{\max_j w_j}$$

Нормализация

Prioritized Experience Replay

Определить p_i можно разными способами. Например:

$$p_i = \frac{1}{rank(i)}$$

Или

$$p_i = |Q(s_i, a_i) - (r_i + \gamma \cdot \max_a Q(s'_i, a))|$$

Замечание 1: Для семплирования используется дерево отрезков

Замечание 2: Веса обновляются только для тех элементов, для которых обновляется сеть

Deep Q-Network with PER

Алгоритм:

1. Инициализируем **Q(s, a)**, создаем пустой буфер **B**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
    1. **a** := argmax **Q(s, .)** с вероятностью 1-**eps**, иначе **a** := random(**A**)
    2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
    3. Добавляем **(s, a, s', r, d)** в **B** с максимальным приоритетом
    4. Сэмплим батч **b ~ B** с учетом приоритетов
    5. Обновляем Q с помощью $L = MSE(Q(s,a), r + \gamma \cdot \max_{a'} Q_T(s', a'))$ с учетом весов элементов из **b**
    6. Обновляем приоритеты для элементов из b
    7. **s** := **s'** если не **d**, иначе реинициализируем среду
    8. Если **i % N** = 0, то $\mathbf{Q_T}$:= **Q,** где **i** - номер шага

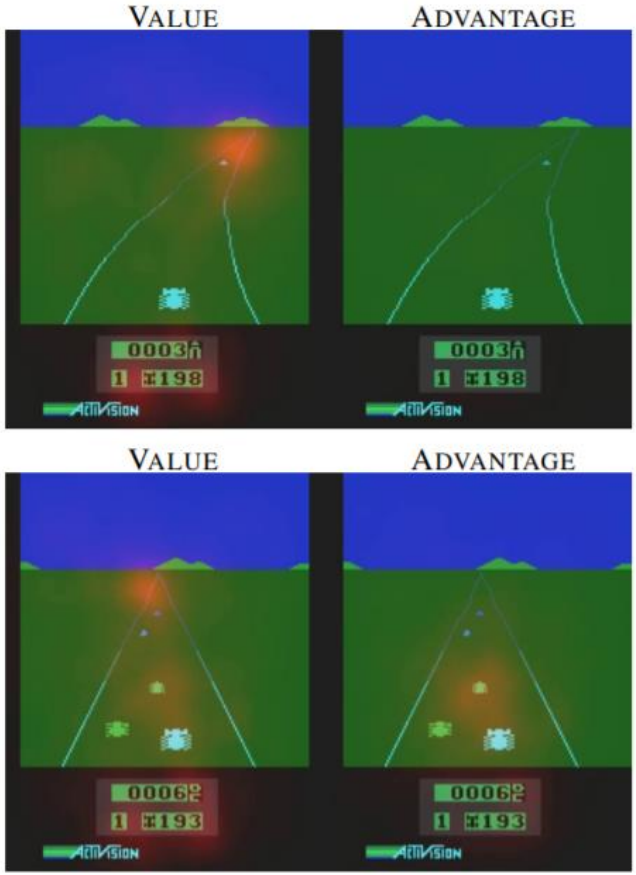
Dueling Deep Q-Network

Advantage function - это функция, которая определяет, насколько совершение определенного действия меняет максимальную суммарную награду

$$A(s, a) = Q(s, a) - V(s)$$

Замечание: значение функции всегда не положительное

Идея: давайте учить advantage и value отдельно



Dueling Deep Q-Network

Идея: давайте учить advantage и value отдельно

$$Q(s,a) = V(s) + A(s,a),\ A(s,a) \leq 0$$

Переформулируем:

$$Q(s,a) = V(s) + (A(s,a) - \max_{a'} A(s,a'))$$

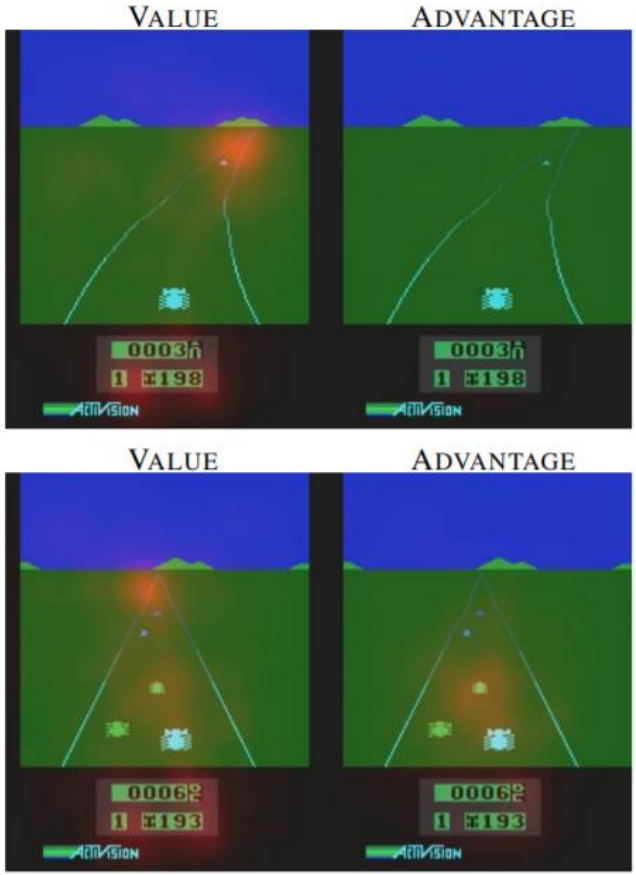
Максимальный advantage для состояния

Проблема: через максимум градиент проходит только к значению для одного действия

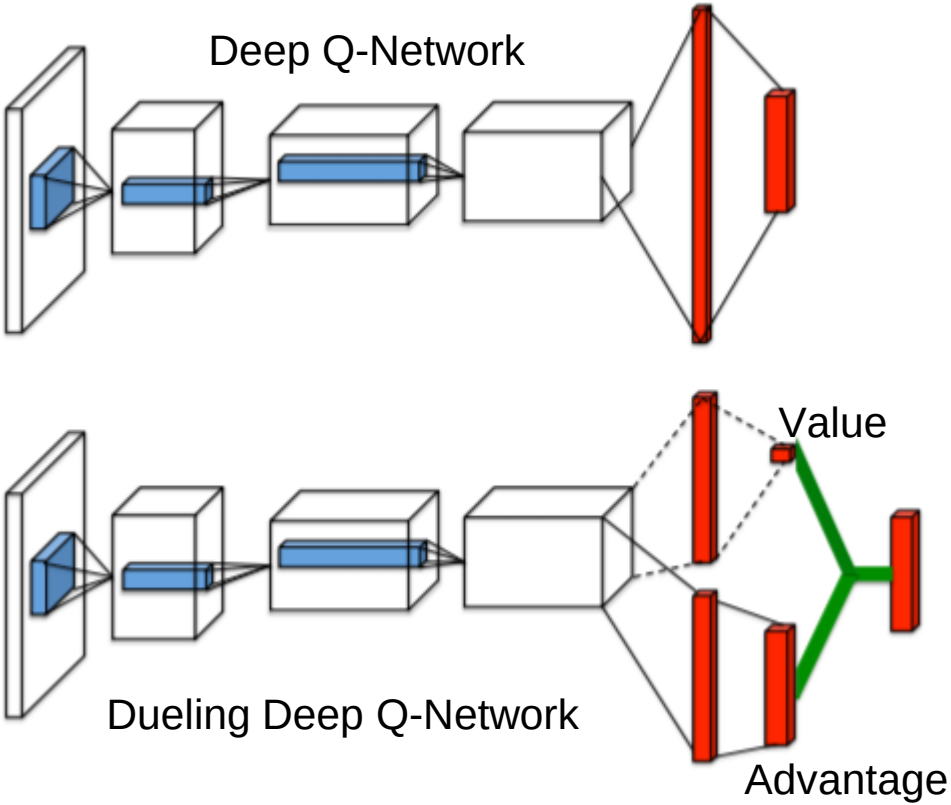
Решение:

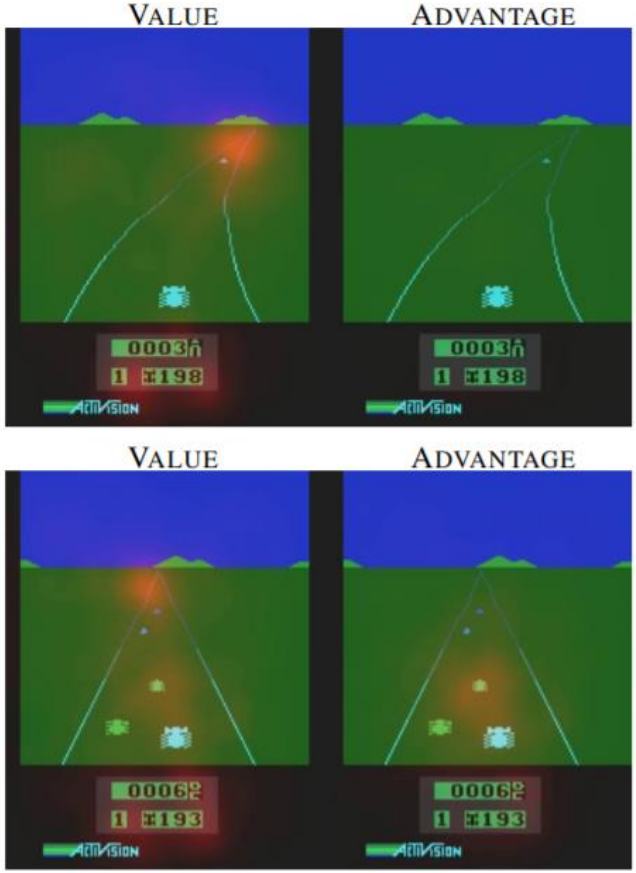
$$Q(s,a) = V(s) + (A(s,a) - \frac{1}{|A|}\sum_{a'} A(s,a'))$$

Средний advantage для состояния

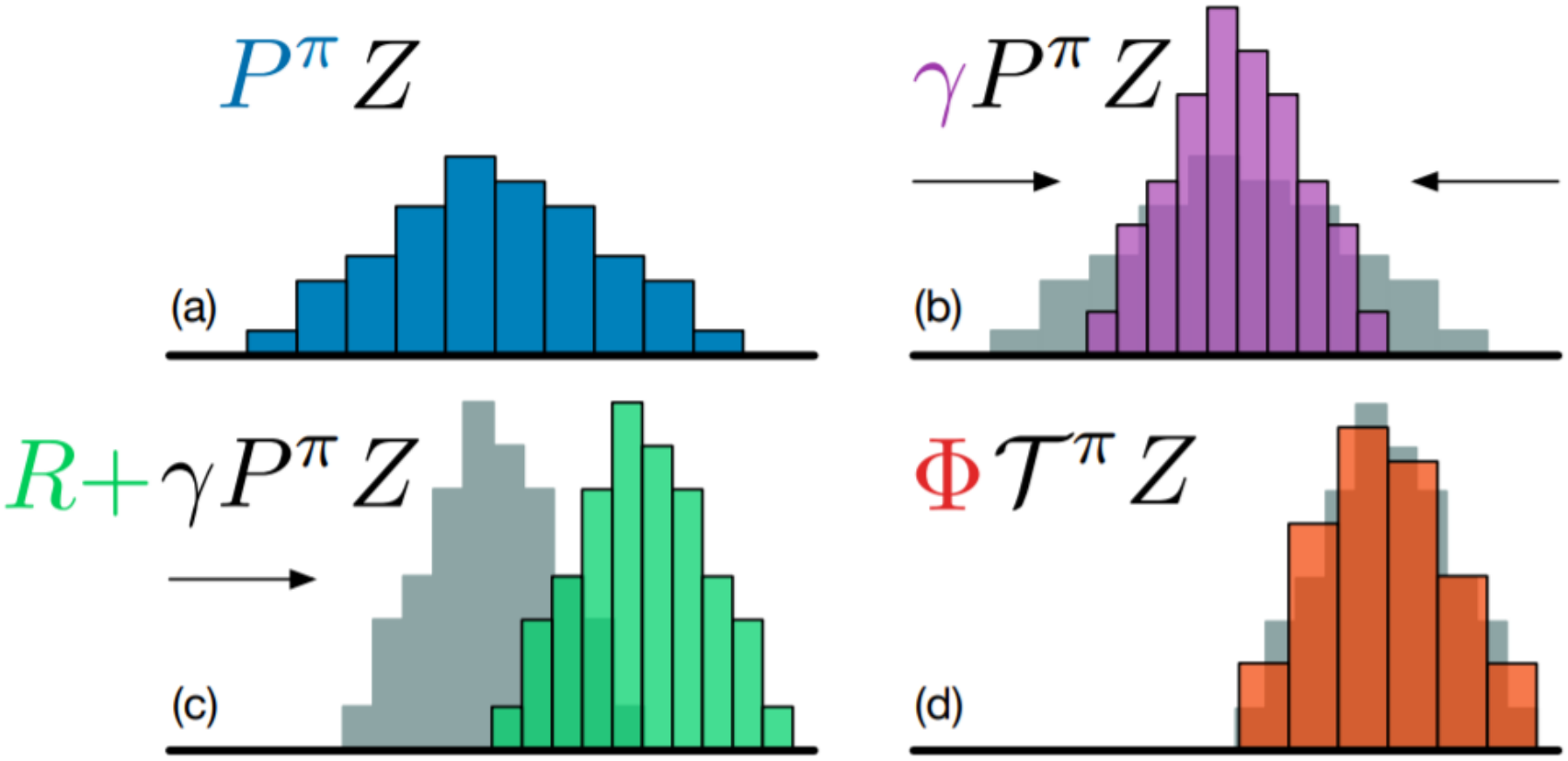


Dueling Deep Q-Network





Distributional Deep Q-Network



Distributional Deep Q-Network

Обновление оценки Q:

- Перед обучением определяем максимальное и минимальное значение Q и фиксируем число делений

- Во время обучения корректируем распределение для следующего состояния и сдвигаем распределение для s в сторону полученного распределения

Детали реализации:

- Во время выбора действия используем математическое ожидание

- Нейронная сеть теперь для каждого действия возвращает заданное число делений к которым применяется softmax. В результате получается некоторое дискретное распределение

Еще улучшения

N-step replay

Идея: давайте суммировать награды за несколько шагов. Это должно помочь быстрее учиться

$$L = MSE(Q(s,a), \sum_{i=0}^{n-1} \gamma^i r_i + \gamma^n \max_{a'} Q_T(s', a'))$$

Double Deep Q-Network

Идея: будем использовать текущую policy для выбора действия в целевой функции как в SARSA

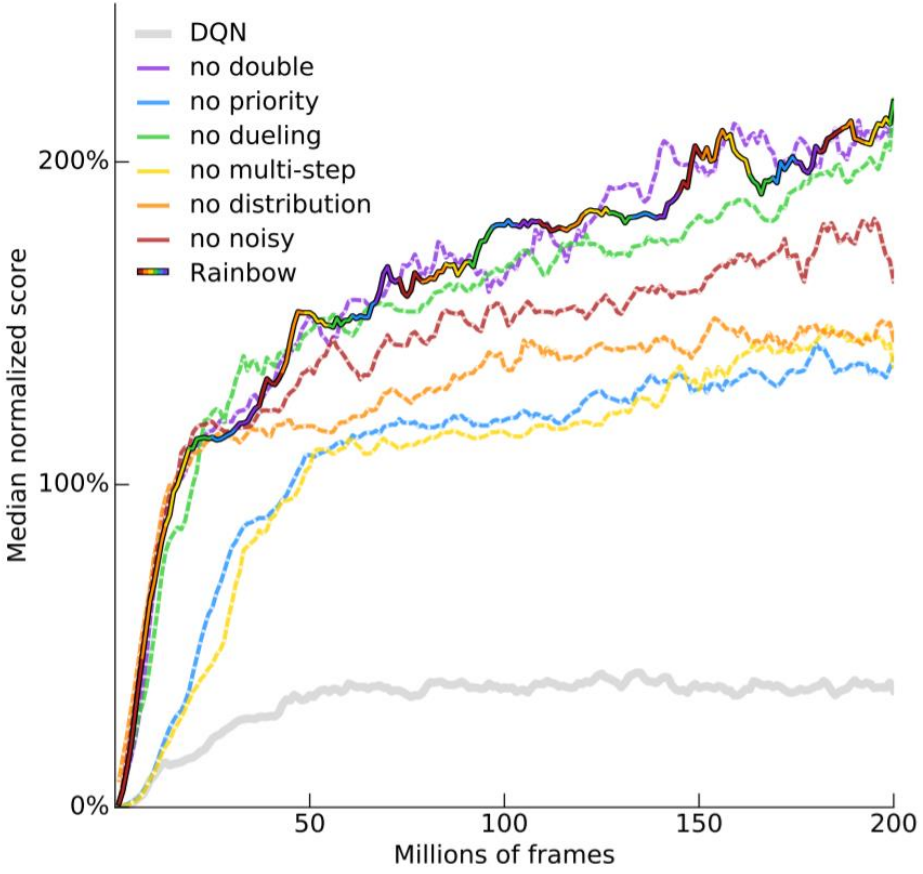
$$L = MSE(Q(s,a), r + \gamma Q_T(s', \arg\max_{a'} Q(s', a')))$$

Layer noise

Идея: зашумляем веса на слоях нейронной сети

$$W_i = (\mu_i + \epsilon \cdot \sigma_i), \quad \epsilon \sim \mathcal{N}(0, 1)$$

Rainbow DQN


DQN
no double
no priority
no dueling
no multi-step
no distribution
no noisy
Rainbow
Median normalized score
200%
100%
0%
50
100
150
200
Millions of frames


kaifolog.ru

Ограничения

1) ~~Должны знать T и R~~
2) ~~Дискретное пространство состояний~~
3) Дискретное пространство действий