

Лекция 3: Q-обучение, аппроксимация, Rainbow

Панов А.И.
panov.ai@mipt.ru

Центр когнитивного моделирования МФТИ
Институт проблем искусственного интеллекта ФИЦ ИУ РАН

2020 – Машинное обучение с подкреплением
Курс для Сбербанка



План лекции

- 1 Обучение по чужому опыту
- 2 Q-обучение
- 3 Аппроксимация
- 4 Инкрементальные методы
- 5 Глубокое обучение с подкреплением
- 6 Rainbow

SARSA алгоритма для управления по актуальному опыту (on-policy)

Algorithm 1 SARSA с ϵ -жадной стратегией

```
1:  $MDP = \langle S, A, \gamma \rangle$                                 ▷ задача МППР
2:  $Q \leftarrow [0, \dots]$ ;                                     ▷ таблица значений полезности действий
3: for all эпизодов do
4:    $s$  - начальное состояние;
5:    $a \leftarrow \epsilon\text{-жадная } \pi(s)$ 
6:   for all шагов эпизода do
7:      $r, s' \leftarrow$  применение  $a$ ;
8:      $a' \leftarrow \epsilon\text{-жадная } \pi(s');$ 
9:      $Q[s, a] \leftarrow Q(s, a) + \alpha(r + \gamma Q[s', a'] - Q[s, a]);$ 
10:     $s \leftarrow s', a \leftarrow a';$ 
```

Обучение по отложенному опыту (off-policy)

Строим целевую стратегию $\pi(a|s)$ для вычисления $V^\pi(s)$ или $Q^\pi(s, a)$ по следующей поведенческой стратегии $\mu(a|s)$:

$$(s_t, a_t, r_t, \dots) \sim \mu.$$

- Обучение по наблюдениям за человеком или другими агентами
- Переиспользование опыта, полученного по старым стратегиям $\pi_1, \pi_2, \dots, \pi_{t-1}$
- Конструирование оптимальной стратегии, следуя поисковой стратегии
- Конструирование нескольких стратегий, следуя одной

Выборка по значимости

Оценка матожидания другого распределения (importance sampling):

$$\begin{aligned}\mathbb{E}_{X \sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\ &= \mathbb{E}_{X \sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

Выборка по значимости для Монте-Карло по отложенному опыту

- Используем отдачи, полученные по μ , для вычисления π
- Взвешиваем отдачу R_t в соответствии со сходством стратегий
- Считаем поправки для выборки значимости по всему эпизоду:

$$R_t^{\pi/\mu} = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \frac{\pi(a_{t+1}|s_{t+1})}{\mu(a_{t+1}|s_{t+1})} \dots \frac{\pi(a_T|s_T)}{\mu(a_T|s_T)} R_t$$

- Обновляем стратегию на основе скорректированной отдачи:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t^{\pi/\mu} - V(s_t))$$

- Не применяем, если μ – нулевая, а π – ненулевая
- Выборка по значимости может существенно увеличить дисперсию

Выборка по значимости для TD по отложенному опыту

- Используем TD показатели, полученные по μ , для вычисления π
- Взвешиваем TD показатель $R + \gamma V(s')$ в соответствии выборкой по значимости
- Необходима только одна правка по значимости

$$V(s_t) \leftarrow V(s_t) + \alpha \left(\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} (r_{t+1} + \gamma V(s_{t+1})) - V(s_t) \right)$$

- Намного более низкая дисперсия по сравнению с Монте-Карло
- Стратегии должны быть схожи только на одном шаге итерации

Q-обучение

- Теперь рассмотрим обучение по отложенному опыту для функции полезности действий $Q(s_t, a_t)$
- Для Q-значения выборка по значимости не требуется!
- Следующее действие a_{t+1} выбирается по поведенческой стратегии $\mu(\cdot|s_t)$
- Рассмотрим альтернативное последующие действия $a' \sim \pi(\cdot|s_t)$
- Тогда обновление $Q(s_t, a_t)$ в соответствии с полезностью альтернативного действия будет выглядеть как:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a') - Q(s_t, a_t))$$

Управление по отложенному опыту с Q-обучением

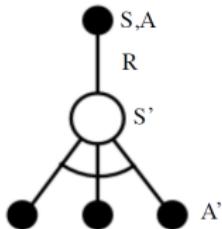
- Будем улучшать как целевую, так и поведенческую стратегии
- Целевая стратегия π улучшается жадно с учетом $Q(s, a)$:

$$\pi(s_{t+1}) = \arg \max_{a'} Q(s_{t+1}, a').$$

- Поведенческая стратегия μ , например, ϵ -жадно, также с учетом $Q(s, a)$.
- Показатель Q-обучения упрощается:

$$\begin{aligned}
 & r_{t+1} + \gamma Q(s_{t+1}, a') \\
 &= r_{t+1} + \gamma Q(s_{t+1}, \arg \max_{a'} Q(s_{t+1}, a')) \\
 &= r_{t+1} + \max_{a'} \gamma Q(s_{t+1}, a').
 \end{aligned}$$

Алгоритм управления с Q-обучением (SARSAMAX)



$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

Алгоритм Q-обучения для управления по отложенному опыту

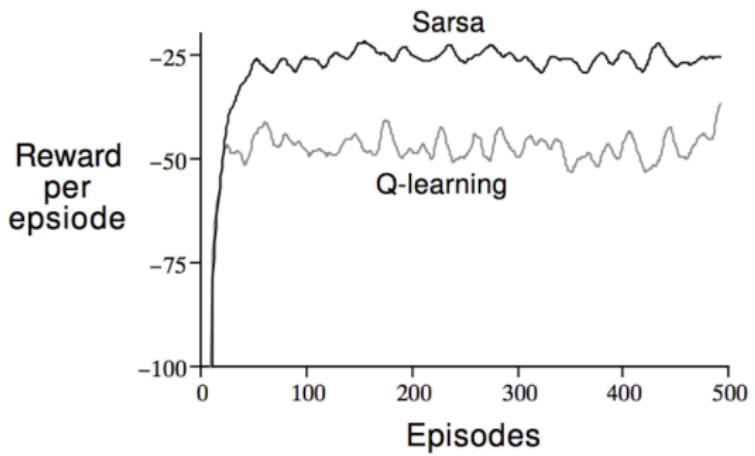
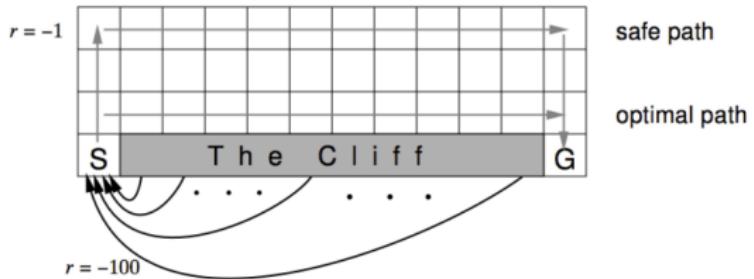
Algorithm 2 Q-обучение с ϵ -жадной стратегией

```

1:  $MDP = \langle S, A, \gamma \rangle$                                 ▷ задача МППР
2:  $Q \leftarrow [0, \dots]$ ;                               ▷ таблица значений полезности действий
3: for all эпизодов do
4:    $s$  - начальное состояние;
5:   for all шагов эпизода do
6:      $a \leftarrow \epsilon\text{-жадная } \pi^Q(s)$ 
7:      $r, s' \leftarrow$  применение  $a$ ;
8:      $a' \leftarrow \epsilon\text{-жадная } \pi(s');$ 
9:      $Q[s, a] \leftarrow Q(s, a) + \alpha(r + \gamma \max_a Q[s', a'] - Q[s, a]);$ 
10:     $s \leftarrow s', a \leftarrow a';$ 

```

Пример: случайные блуждания над обрывом



Пространства большой размерности в RL

Обучение с подкреплением может быть использовано для задач большой размерности, например:

- нарды: 10^{20} состояний,
- игра Го: 10^{170} состояний,
- вертолет: непрерывное множество состояний

Как можно масштабировать безмодельные методы *предсказания и управления*, которые мы рассматривали ранее?

Аппроксимация функции полезности

- Мы рассматривали функцию полезности в виде таблицы (lookup table):
 - ▶ для каждого s состояния была запись $V(s)$,
 - ▶ или для каждой пары состояние-действие s, a была запись $Q(S, a)$
- Проблема с большими MDP:
 - ▶ слишком много состояний и/или действий, чтобы хранить их в памяти,
 - ▶ слишком медленное обучение функции полезности для каждого состояния в отдельности
- Решение для больших MDP:
 - ▶ Оценка функции полезности с помощью *аппроксимации функции*:

$$\hat{V}(s, w) \approx V^\pi(s),$$

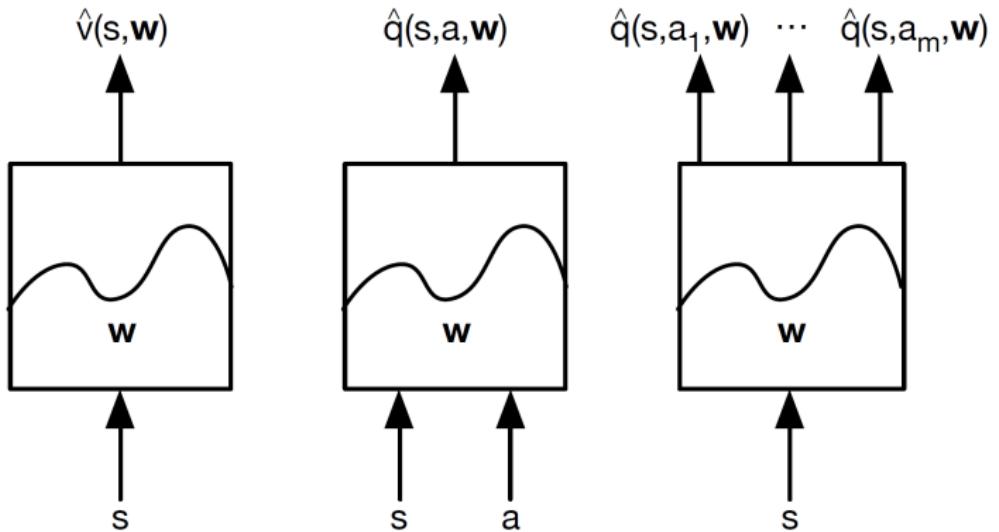
$$\hat{Q}(s, a, w) \approx Q^\pi(s, a)$$

- ▶ Обобщение наблюдаемых состояний на ненаблюдаемые
- ▶ Обновление параметров w с использованием МС или TD обучения

Преимущества обобщения

- Сокращение используемой памяти для $(T, R)/V/Q/\pi$
- Сокращение вычислительных затрат при работе с $(T, R)/V/Q/\pi$
- Сокращение количества необходимых данных (опыта) для поиска оптимальных $(T, R)/V/Q/\pi$

Виды аппроксимации функции полезности



Аппроксиматоры

Будем рассматривать **дифференцируемые** аппроксиматоры:

- **Линейная комбинация признаков**
- **Нейронные сети**
- Деревья принятия решений
- Ближайшие соседи
- Фурье/вейвлет разложения
- ...

Нам нужны методы обучения, подходящие для анализа
нестационарных и неразмеченных данных

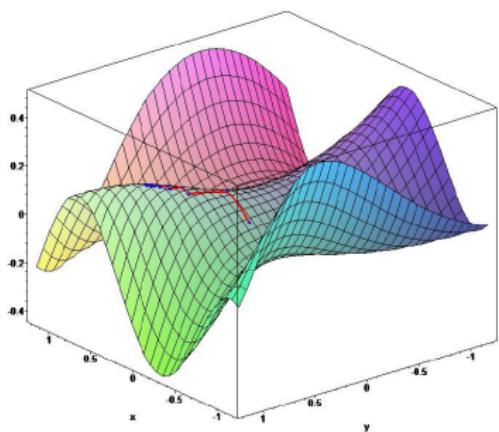
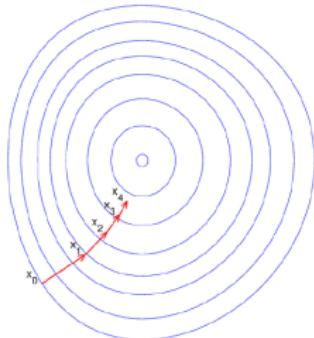
Градиентный спуск

- Пусть $J(w)$ – дифференцируемая функция вектора параметров w
- Запишем градиент функции $J(w)$:

$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$

- Будем искать локальный минимум функции $J(w)$
- Будем обновлять w в направлении антиградиента (α – параметр длины шага) (gradient descent):

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w)$$



Стохастический градиентный спуск

- Цель: найти вектор параметров w , минимизируя среднеквадратичную ошибку в предположении, что мы знаем истинные значения $V^\pi(s)$:

$$J(w) = \mathbb{E}_\pi[(V^\pi(s) - \hat{V}(s, w))^2]$$

- Градиентный спуск позволяет найти локальный минимум:

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w) = \alpha \mathbb{E}_\pi[(V^\pi(s) - \hat{V}(s, w)) \nabla_w \hat{V}(s, w)]$$

- Стохастический градиентный спуск (stochastic gradient descent) производит спуск по подвыборке:

$$\Delta w = \alpha(V^\pi(s) - \hat{V}(s, w)) \nabla_w \hat{V}(s, w)$$

- Матожидание обновления равно полному градиентному обновлению

Вектор признаков

Представим состояние в виде вектора признаков:

$$x(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

Примеры:

- Расстояние от робота до ключевых точек местности
- Тренды на бирже
- Сильные и слабые конфигурации в шахматах

Линейная аппроксимация функции полезности

- Представим функцию полезности в виде линейной комбинации признаков:

$$\hat{V}(s, w) = \mathbf{x}(s)^\top w = \sum_{j=1}^n x_j(s)w_j$$

- Целевая функция квадратична по w :

$$J(w) = \mathbb{E}_\pi[(V^\pi(s) - \mathbf{x}(s)^\top w)^2]$$

- Стохастический градиентный спуск сходится к глобальному оптимуму
- Правило обновления:

$$\nabla_w \hat{V}(s, w) = \mathbf{x}(s)$$

$$\Delta w = \alpha(V^\pi(s) - \mathbf{x}(s)^\top w)\mathbf{x}(s)$$

- Обновление = длина шага \times ошибка предсказания \times значение признака

Табличные признаки

- Табличное представление функции полезности – частный случай линейного аппроксиматора:

$$x^{table}(s) = \begin{pmatrix} 1(s = s_1) \\ \vdots \\ 1(s = s_n) \end{pmatrix}$$

- Вектор параметров представляет собой значения полезности каждого состояния:

$$\hat{V}(s, w) = \begin{pmatrix} 1(s = s_1) \\ \vdots \\ 1(s = s_n) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

Инкрементальные алгоритмы предсказания

- Мы предполагали, что истинные значения функции полезности нам доступны
- Однако в нашей задаче нет учителя, только вознаграждения
- На практике мы заменяем учителя оценкой функции $V^\pi(s)$ с предыдущего этапа:
 - ▶ для МС оценка – это отдача R_t :

$$\Delta w = \alpha(R_t - \hat{V}(s_t, w)) \nabla_w \hat{V}(s_t, w),$$

- ▶ для TD(0) оценка – это показатель $r_{t+1} + \gamma \hat{V}(s_{t+1}, w)$:

$$\Delta w = \alpha(r_{t+1} + \gamma \hat{V}(s_{t+1}, w) - \hat{V}(s_t, w)) \nabla_w \hat{V}(s_t, w)$$

Монте-Карло с аппроксимацией функции полезности

- Отдача R_t – несмещенная, зашумленная подвыборка истинного значения $V^\pi(s_t)$
- Можем применить обучение с учителем для “обучающей выборки”:

$$\langle s_1, R_1 \rangle, \langle s_2, R_2 \rangle, \dots, \langle s_T, R_T \rangle$$

- Пример: использование линейной Монте-Карло оценки стратегии

$$\Delta w = \alpha(R_t - \hat{V}(s_t, w)) \nabla_w \hat{V}(s_t, w) = \alpha(R_t - \hat{V}(s_t, w)) x(s_t)$$

- Монте-Карло оценка сходится к локальному оптимуму
- Это верно и при использовании нелинейных аппроксиматоров

TD-обучение с аппроксимацией функции полезности

- TD показатель $r_{t+1} + \gamma \hat{V}(s_{t+1}, w)$ – смещенная подвыборка истинного значения $V^\pi(s_t)$
- Можем применить обучение с учителем для “обучающей выборки”:

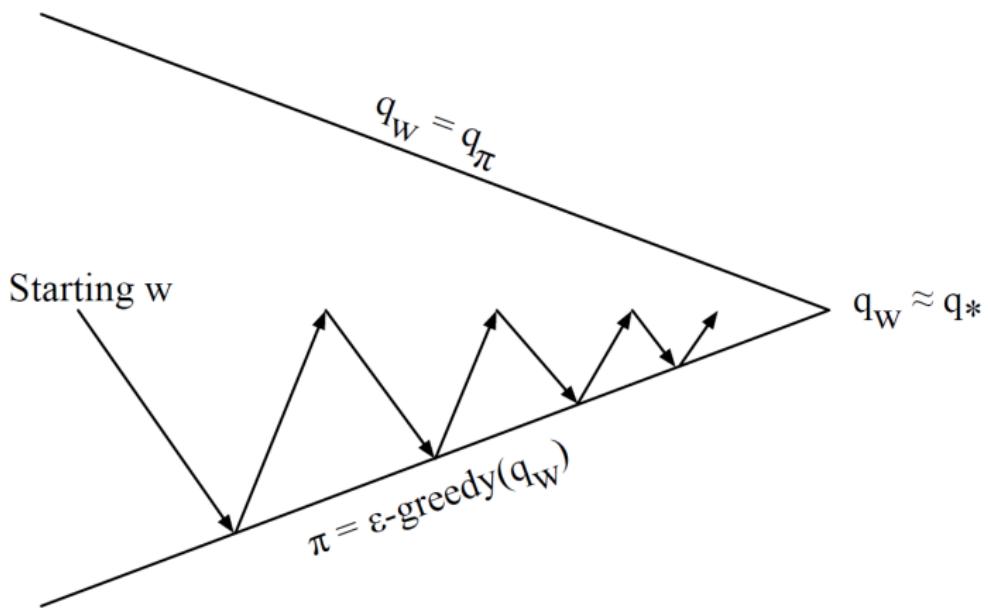
$$\langle s_1, r_2 + \gamma \hat{V}(s_2, w) \rangle, \langle s_2, r_3 + \gamma \hat{V}(s_3, w) \rangle, \dots, \langle s_{T-1}, r_T \rangle$$

- Пример: использование линейного TD(0)

$$\Delta w = \alpha(r + \gamma \hat{V}(s', w) - \hat{V}(s_t, w)) \nabla_w \hat{V}(s_t, w) = \alpha \delta x(s)$$

- Линейное TD(0)-обучение сходится к глобальному оптимуму

Управление с аппроксимацией функции полезности



Оценка стратегии: приближенная оценка стратегии $\hat{Q}(\cdot, \cdot, w) \approx Q^\pi$.

Улучшение стратегии: ϵ -жадное улучшение стратегии

Аппроксимация функции полезности действия

- Аппроксимация функции полезности действия:

$$\hat{Q}(s, a, w) \approx Q^\pi(s, a)$$

- Минимизация среднеквадратичной ошибки:

$$J(w) = \mathbb{E}_\pi[(Q^\pi(s, a) - \hat{Q}(s, a, w))^2]$$

- Использование стохастического градиентного спуска для поиска локального минимума:

$$-\frac{1}{2} \nabla_w J(w) = (Q^\pi(s, a) - \hat{Q}(s, a, w)) \nabla_w \hat{Q}^\pi(s, a, w),$$

$$\Delta w = \alpha(Q^\pi(s, a) - \hat{Q}(s, a, w)) \nabla_w \hat{Q}^\pi(s, a, w)$$

Линейная аппроксимация функции полезности действия

- Будем представлять и состояние и действие в виде вектора признаков:

$$x(s, a) = \begin{pmatrix} x_1(s, a) \\ \vdots \\ x_n(s, a) \end{pmatrix}$$

- Представим функцию полезности действия в виде линейной комбинации признаков:

$$\hat{Q}(s, a, w) = x(s, a)^\top w = \sum_{j=1}^n x_j(s, a)w_j$$

- Обновление стохастического градиентного спуска:

$$\nabla_w \hat{Q}(s, a, w) = x(s, a),$$

$$\Delta w = \alpha(Q^\pi(s, a) - \hat{Q}(s, a, w))x(s, a)$$

Инкрементальный алгоритмы управления

- Как и в предсказании, мы должны чем-то заменить оценку для $Q^\pi(s, a)$:
 - для МС оценка – это отдача R_t :

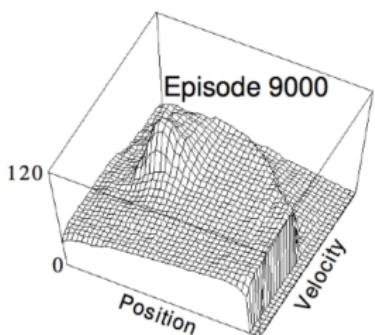
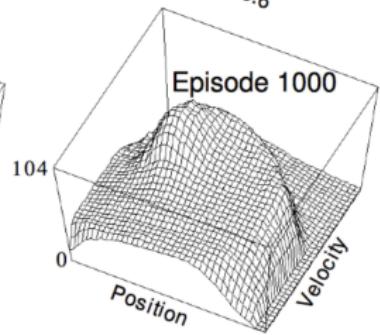
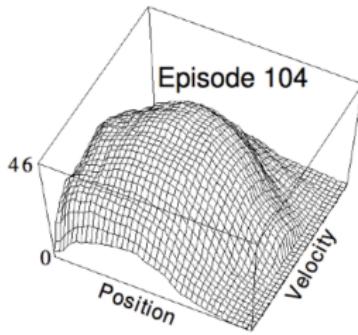
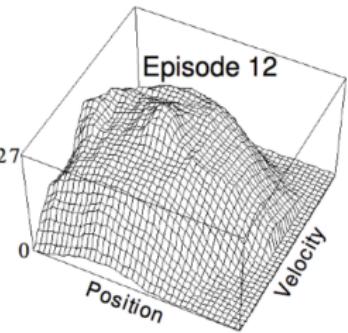
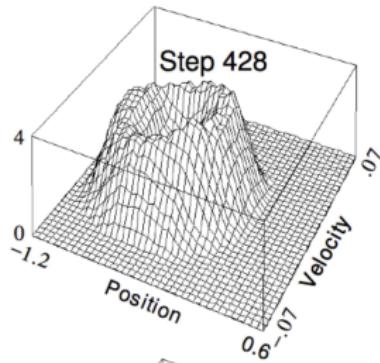
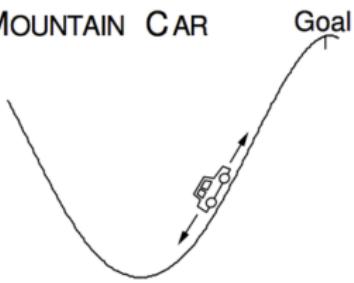
$$\Delta w = \alpha(R_t - \hat{Q}(s_t, a_t, w)) \nabla_w \hat{Q}(s_t, a_t, w),$$

- для TD(0) оценка – это показатель $r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, w)$:

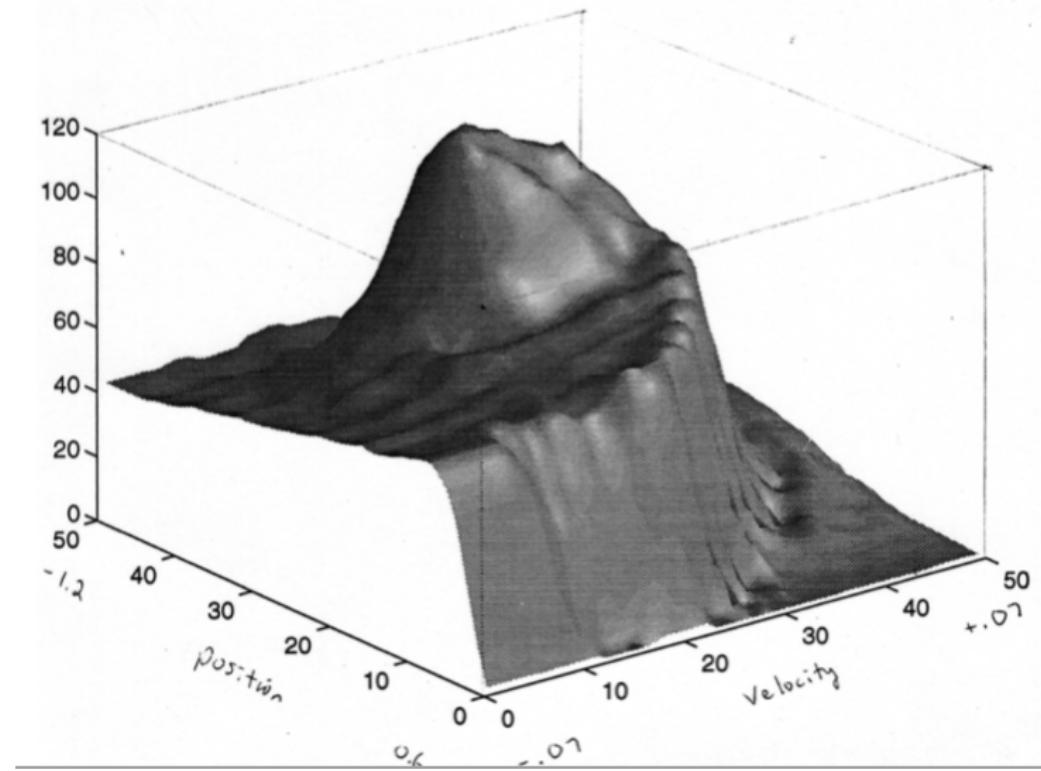
$$\Delta w = \alpha(r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}, w) - \hat{Q}(s_t, a_t, w)) \nabla_w \hat{Q}(s_t, a_t, w)$$

Линейная SARSA для задачи “Внедорожник”

MOUNTAIN CAR



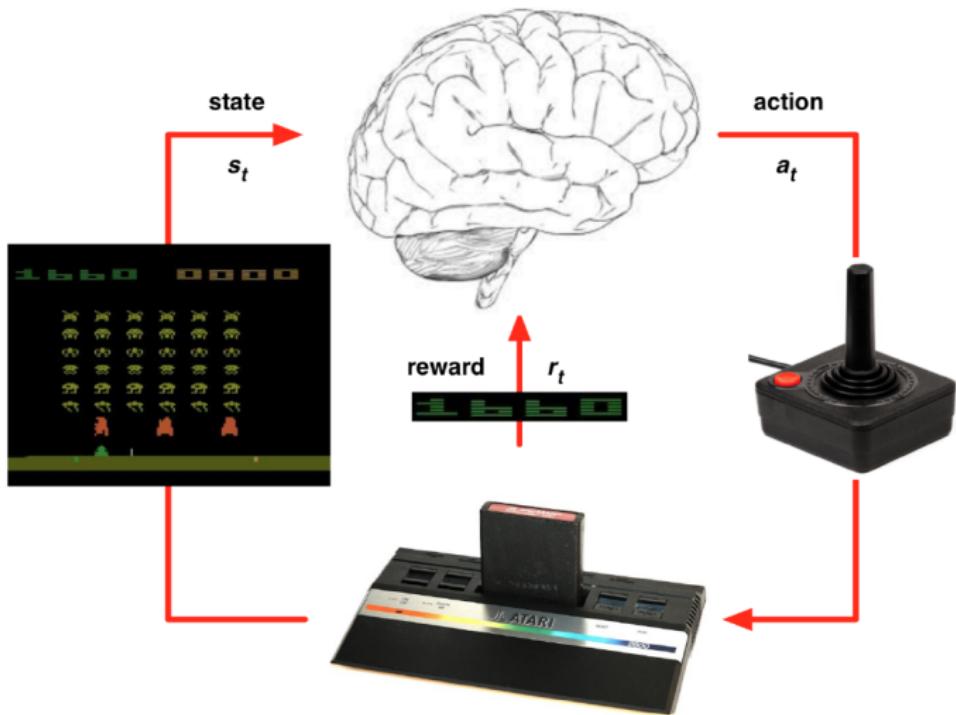
Линейная SARSA с радиальными базисными функциями для задачи “Внедорожник”



Аппроксимация глубокими нейронными сетями

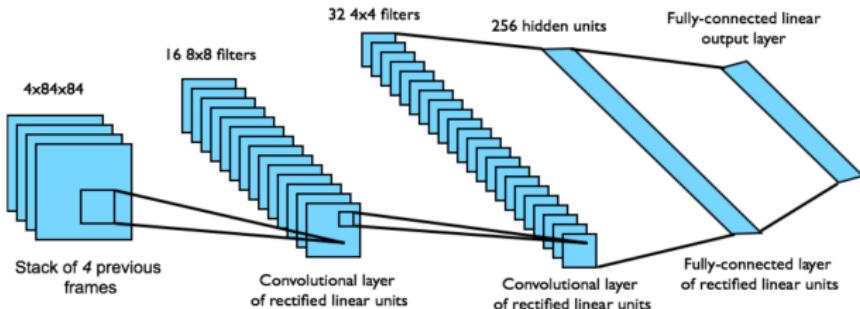
- Композиция множества функций $y = h_n(h_{n-1} \dots h_1(x))$
- Комбинация линейного и нелинейного преобразования:
 $h_n = wh_{n-1}$ и $h_n = f(h_{n-1})$
- Используем цепное правило обратного распространения для вычисления градиента
- Для обучения необходимо задание функции потерь L

Применим нейросетевую аппроксимацию к Atari



DQN в играх Atari

- Сквозное (end-to-end) обучение значений $Q(s, a)$ по пикселям s .
- Входное состояние – 4 склеенных изображения из каждого 4-го кадра (frameskip, framestack).
- Выход – значения $Q(s, a)$ для 18 позиций джойстика/кнопки
- Вознаграждение – изменение в счете за этот шаг



Проблемы сходимости Q-обучения с аппроксимацией:

- Высокая корреляция между примерами в выборке
- Нестационарные целевых значений

Сохранение опыта в глубоких Q-сетях (DQN)

DQN использует память прецедентов (replay buffer) и фиксированные Q-показатели

- Выбираем действие a_t в соответствии с ϵ -жадной стратегией
- Сохраняем наблюдаемый переход $(s_t, a_t, r_{t+1}, s_{t+1})$ в память переигровок \mathcal{D}
- Выбираем случайный мини-пакет (mini-batch) переходов (s, a, r, s') из памяти \mathcal{D}
- Используем один из вариантов стохастического градиентного спуска:

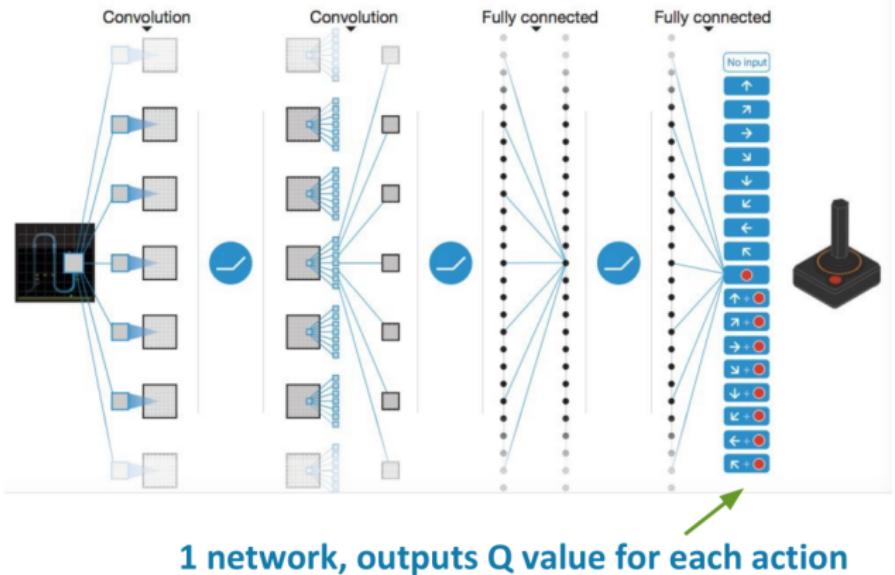
$$\Delta w = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a', w) - \hat{Q}(s, a, w)) \nabla_w \hat{Q}(s, a, w)$$

Фиксирование Q-показателя

- Зафиксируем значение весов используемых для вычисления Q-показателя на несколько шагов обновления
- Используем отдельное множество параметров w^- , которое затем обновляется
- Вычисляем показатели Q-обучения на основе предыдущего фиксированного значения параметров w^-
- Оптимизируем ошибку между Q-сетью и показателями Q-обучения:

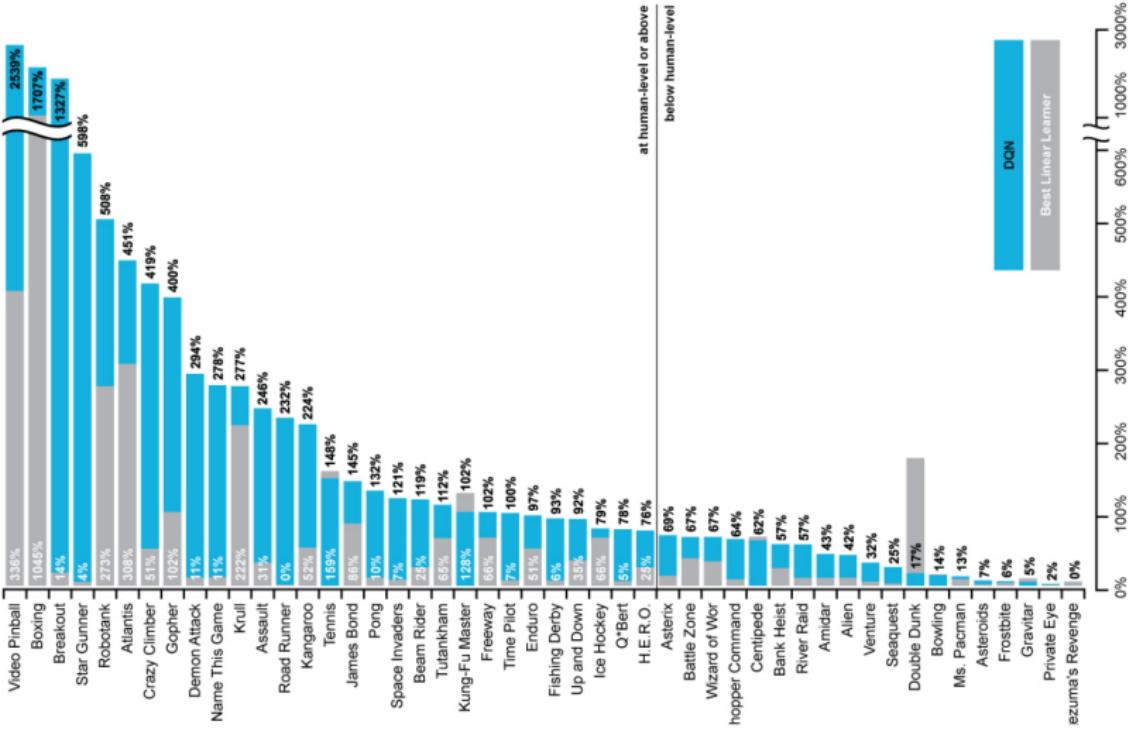
$$\mathcal{L}_i = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w_i^-) - Q(s, a, w_i) \right)^2 \right]$$

Глубокая Q-сеть



DQN: результаты обучения

Результаты DQN в играх Atari



Результаты DQN в играх Atari

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

Развитие DQN

Вслед за успешным применением DQN было предложено много различных улучшений:

- Двойной DQN - Double DQN (AAAI 2016)
<http://arxiv.org/abs/1509.06461>
- Приоритизированная память прецедентов - Prioritized Replay (ICLR 2016) <https://arxiv.org/abs/1511.05952>
- Дуэльный DQN - Dueling DQN (ICML 2016)
<https://arxiv.org/abs/1511.06581>
- Радуга - Rainbow (AAAI 2018)
<http://arxiv.org/abs/1710.02298>

Двойное Q-обучение (DDQN)

```

1: Initialize  $Q_1(s, a)$  and  $Q_2(s, a), \forall s \in S, a \in A$   $t = 0$ , initial state  $s_t = s_0$ 
2: loop
3:   Select  $a_t$  using  $\epsilon$ -greedy  $\pi(s) = \arg \max_a Q_1(s_t, a) + Q_2(s_t, a)$ 
4:   Observe  $(r_t, s_{t+1})$ 
5:   if (with 0.5 probability True) then
6:

```

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha(r_t + Q_1(s_{t+1}, \arg \max_{a'} Q_2(s_{t+1}, a')) - Q_1(s_t, a_t))$$

```

7:   else
8:

```

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha(r_t + Q_2(s_{t+1}, \arg \max_{a'} Q_1(s_{t+1}, a')) - Q_2(s_t, a_t))$$

```

9:   end if
10:   $t = t + 1$ 
11: end loop

```

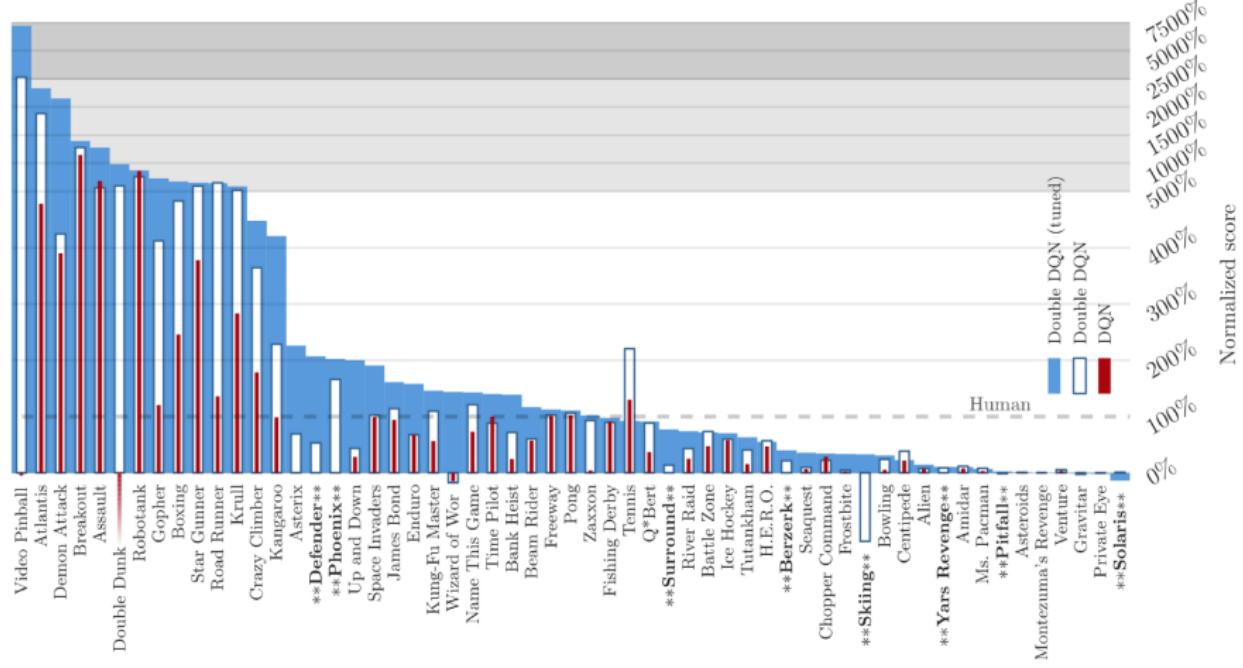
Двойной DQN

- Текущая Q-сеть с параметрами w используется для выбора действий
- Вторая Q-сеть с параметрами w^- используется для оценки действий

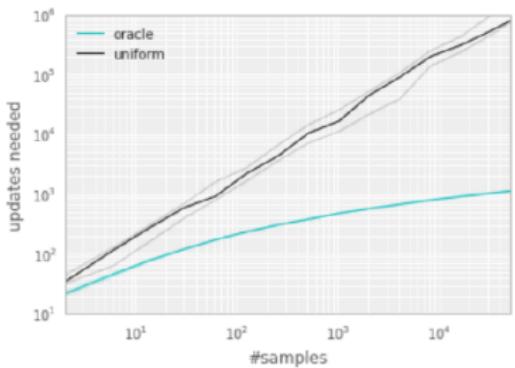
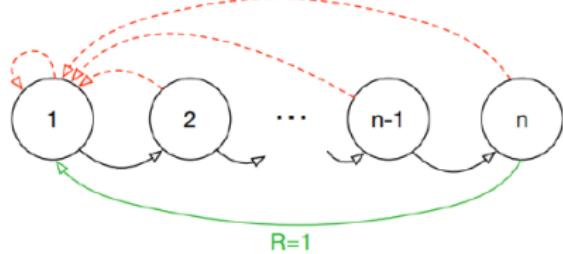
$$\Delta w = \alpha(r_{t+1} + \gamma \underbrace{\hat{Q}(s_{t+1}, \arg \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}, w), w^-)}_{\text{выбор действия}} - \hat{Q}(s_t, a_t, w))$$

оценка действия

Двойной DQN на играх Atari



Порядок выбора в базе прецедентов



- Теоретический пример: пусть оракул, который подсказывает в каком порядке выбирать прецеденты
- При удачном порядке нужно в экспоненциальное количество раз меньшее количество шагов до сходимости

Приоритизированная память прецедентов

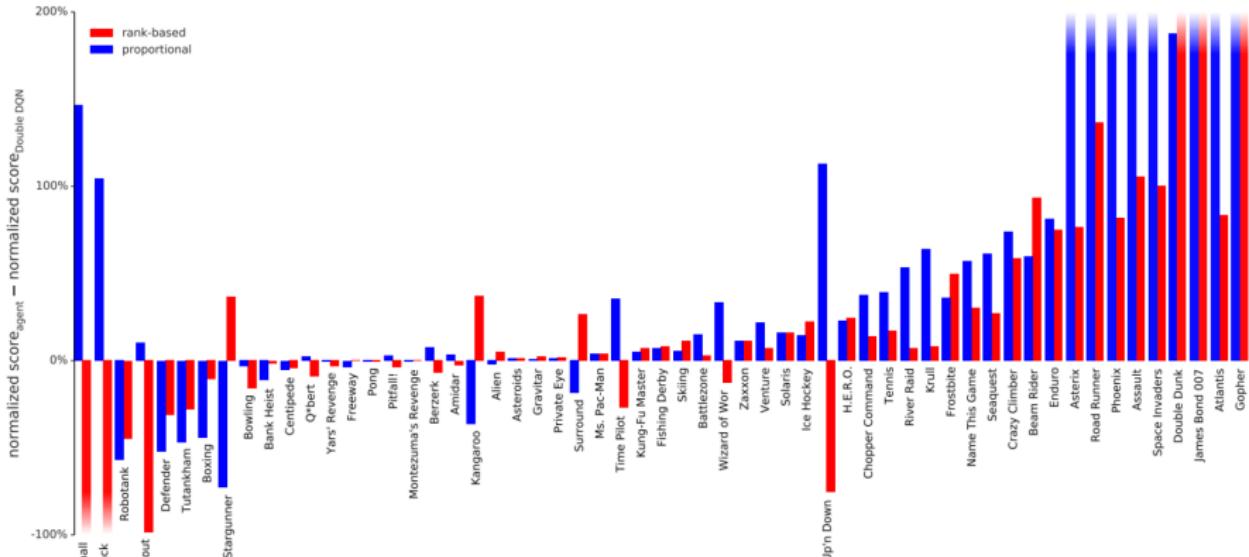
- i -ый прецедент из памяти (s_i, a_i, r_i, s_{i+1})
- Выбираем прецеденты для обновления весов по приоритету, пропорциональному ошибке DQN

$$p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a', w^-) - Q(s_i, a_i, w) \right|$$

- Обновляем приоритеты после каждого обновления весов
- Для новых прецедентов приоритет равен 0
- Стохастическая приоритизация

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

PRB vs DDQN

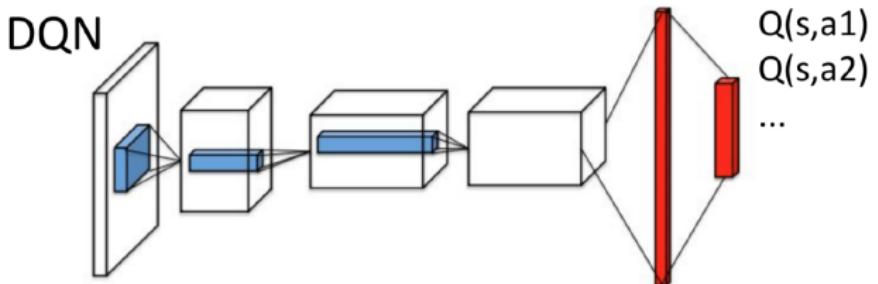


Функция преимущества

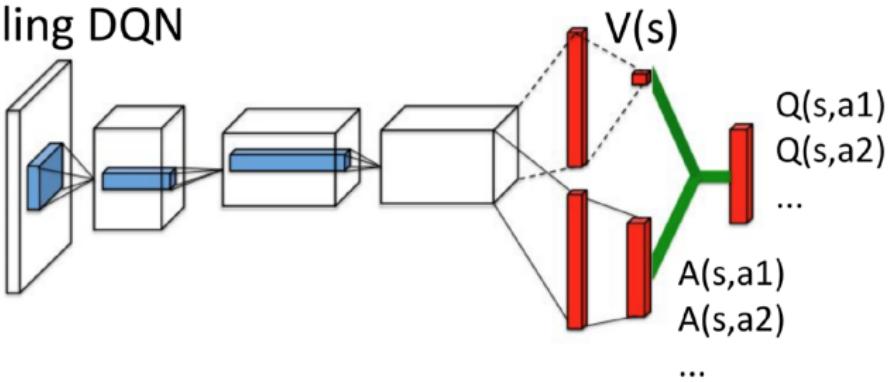
- Идея: признаки, используемые при определении полезности состояния, могут не совпадать с признаками, определяющими успешность действия
- Пример: счет играет роль в определении полезности состояния, но не нужен для выбора действия
- Функция преимущества (advantage function):

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

Дуэльный DQN



Dueling DQN



Дуэльный DQN

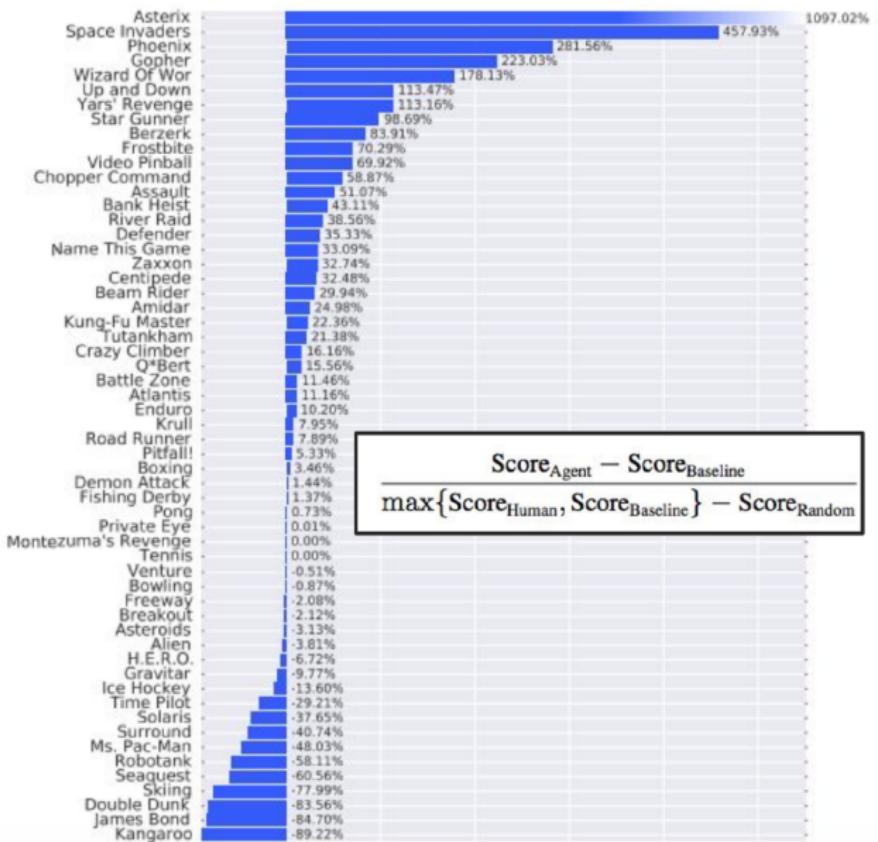
- Назначаем $A(s, a) = 0$ для выбранного действия a

$$\hat{Q}(s, a, w) = \hat{V}(s, w_1) + \left(\hat{A}(s, a, w_2) - \max_{a'} \hat{A}(s', a', w_2) \right)$$

- Используем среднее в качестве базового значения

$$\hat{Q}(s, a, w) = \hat{V}(s, w_1) + \left(\hat{A}(s, a, w_2) - \frac{1}{|A|} \sum_{a'} \hat{A}(s', a', w_2) \right)$$

Дуэльный DQN на Atari



Новые улучшения DQN

- DQN и другие улучшения показали, что удается успешно сочетать глубокое обучение и обучение с подкреплением
- Кроме DDQN, PRB, Dueling предлагались и другие улучшения: многошаговое обучение (multi-step learning), обучение по распределениям (distributional RL), шумовые слои (noisy net)
- Оказывается, что многие улучшения могут быть применены одновременно \Rightarrow алгоритм «радуги» (Rainbow)

Многошаговое обучение

- Вместо жадного (одношагового) подхода, будем использовать более «дальнюю» оценку полезности, которая будет получена после n шагов:

$$R_t^{(n)} = \sum_{k=1}^{n-1} \gamma^{(k)} r_{t+k+1}$$

- В этом случае функция потерь будет выглядеть так:

$$\mathcal{L}_i = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(R_t^{(n)} + \gamma^{(n)} \max_{a'} Q(s', a', w_i^-) - Q(s, a, w_i) \right)^2 \right]$$

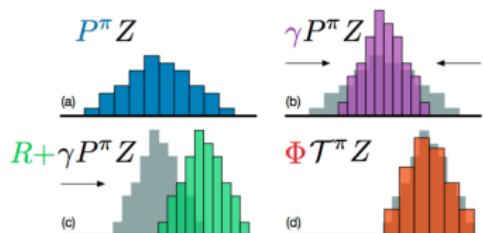
- Такое многошаговое обучение ускоряет процесс обучения в том случае, когда является изменяемым гиперпараметром

Обучение по распределениям

- Вместо оценки ожидаемой отдачи $Q(s, a)$ будем оценивать **распределений отдач** с матожиданием $Q(s, a)$. Пусть z – носитель распределения вероятностей полезности (распределение по категориям):

$$z^i = V_{min} + (i - 1) \frac{V_{max} - V_{min}}{N - 1}, \quad i \in \{1, \dots, N\}$$

- Наша цель – настроить параметры сети w , задающие распределение $d_t = (z, p(s_t, a_t, w))$, где $p(s_t, a_t, w)$ – вероятностная мера для каждого атома i , так, чтобы оно было близко к целевому распределению $d'_t = (r_{t+1} + \gamma z, p(s_{t+1}, a_{t+1}^*, w'))$
- В данном случае минимизируется KL-дивергенция $D_{KL}(\Phi_z d'_t \| d_t)$



Шумовые слои

- Идея – заменить линейный слой шумовым линейным слоем, в котором комбинируются детерминированные и случайные параметры:

$$y = (b + Wx) + (b_{noisy} \odot \epsilon^b + (W_{noisy} \odot \epsilon^W)x)$$

- После некоторого обучения, сеть приобретет способность игнорировать шумовой поток, но в различной степени в зависимости от состояний среды
- Таким образом мы реализуем условное исследование среды
- Это замена -жадному алгоритмы: мы привносим шум в параметрическое пространство для поддержания необходимого уровня исследования среды

Алгоритм Rainbow

Собираем все улучшения вместе:

- Используем обучения по распределениям для оценки распределения отдачи вместо ожидаемой отдачи
- Заменяем одношаговую функцию потерь по распределениям на многошаговую: $d_t^{(n)} = (R_{t+1}^{(n)} + \gamma^{(n)} z, p(s_{t+n}, a_{t+n}^*, w'))$
- Комбинируем эту функцию потерь со стандартной функцией потерь двойного -обучения
- Используем идею приоритезированной памяти прецедентов для выборки переходов с учетом приоритета по функции потерь $D_{RL}(\Phi_z d_t^{(n)} \| d_t)$
- Возвращаемые распределения используются в дульной архитектуре сети

$$p^i(s, a, w) \propto \exp \left(\hat{V}^i(s, w_1) + \left(\hat{A}^i(s, a, w_2) - \frac{1}{|A|} \sum_{a'} \hat{A}^i(s', a', w_2) \right) \right)$$

- Привносим шум заменой линейных слоев на шумовые

Rainbow: производительность и значимость

