

# Обучение с подкреплением II

Сергей Николенко



Центр Речевых Технологий, 2012



# Outline

- 1 **Агенты с несколькими состояниями**
  - Марковские процессы принятия решений
  - Поиск оптимальных стратегий в известной модели
- 2 Поиск оптимальных стратегий без модели
  - Стохастические алгоритмы



## И спросила кроха

- Вернёмся теперь к задаче с несколькими состояниями.
- Вознаграждения (rewards) – на каждом шаге:  $r_t, r_{t+1}, \dots$
- Но что такое «хорошо» in the long run? Как оценивать поведение алгоритма в приведённом выше сеттинге?
- Если есть естественное конечное число шагов (партия), то это *эпизодическая задача* (episodic task), и логично суммировать вознаграждение по эпизоду (до терминального состояния).
- А что с продолжающимися задачами?



## Разные модели

- Ваши версии?



## Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие  $h$  шагов:  $E \left[ \sum_{t=0}^h r_t \right]$ .



## Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие  $h$  шагов:  $E \left[ \sum_{t=0}^h r_t \right]$ .
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?



## Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие  $h$  шагов:  $E \left[ \sum_{t=0}^h r_t \right]$ .
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?



$$E \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right],$$

где  $\gamma$  — некоторая константа (discount factor).



## Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие  $h$  шагов:  $E \left[ \sum_{t=0}^h r_t \right]$ .
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?



$$E \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right],$$

где  $\gamma$  — некоторая константа (discount factor).

- Модель *среднего вознаграждения* (average-reward model):

$$\lim_{h \rightarrow \infty} E \left[ \frac{1}{h} \sum_{t=0}^h r_t \right].$$





## Что мы будем использовать

- Все модели разные, приводят к разным результатам.
- Обычно используется модель бесконечного горизонта с некоторым фиксированным discount factor. Её и мы будем использовать.
- Кроме того, её можно обобщить на эпизодические задачи: достаточно просто положить  $\gamma = 1$  и добавить одно лишнее состояние с вознаграждением 0, которое будет замкнуто на себя. Так что отныне навсегда

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$



## Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?



## Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.



## Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.
- Сходится с большой скоростью. Два подхода:
  - Скорость сходимости к какой-то фиксированной доле оптимальности. Какой?
  - Насколько хорошо себя ведёт алгоритм после фиксированного времени. Какого?



# Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.
- Сходится с большой скоростью. Два подхода:
  - Скорость сходимости к какой-то фиксированной доле оптимальности. Какой?
  - Насколько хорошо себя ведёт алгоритм после фиксированного времени. Какого?
- Минимизировать цену (regret), т.е. уменьшение общей суммы выигрыша по сравнению с оптимальной стратегией с самого начала. Это очень хорошая мера, но результаты о ней получить очень сложно.



# Марковские процессы

- *Марковский процесс принятия решений* (Markov decision process) состоит из:
  - множества состояний  $S$ ;
  - множества действий  $A$ ;
  - функции поощрения  $R : S \times A \rightarrow \mathbb{R}$ ; ожидаемое вознаграждение при переходе из  $s$  в  $s'$  после  $a - R_{ss'}^a$ ;
  - функции перехода между состояниями  $p_{ss'}^a : S \times A \rightarrow \Pi(S)$ , где  $\Pi(S)$  — множество распределений вероятностей над  $S$ . Вероятность попасть из  $s$  в  $s'$  после  $a - P_{ss'}^a$ .
- Модель *марковская*, если переходы не зависят от истории предыдущих переходов.



# Подкрепление и значение состояния

- Главный момент – разница между *reward function* (непосредственное подкрепление) и *value function* (общее подкрепление, ожидаемое, если начать с этого состояния).
- Суть многих методов обучения с подкреплением – в том, чтобы оценивать и оптимизировать *value function*.
- Для марковских процессов можно формально определить:

$$V^\pi(s) = \mathbb{E}_\pi [R_t \mid s_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right].$$



## Подкрепление и значение состояния

- Можно более детализированно – общее подкрепление, ожидаемое, если начать с состояния  $s$  и действия  $a$ :

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi [R_t \mid s_t = s, a_t = a] = \\ &= \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right]. \end{aligned}$$

- Функции  $V$  и  $Q$  – это как раз то, что нам нужно оценить; если бы мы их знали, можно было бы просто выбирать то  $a$ , которое максимизирует  $Q(s, a)$ .





## Подкрепление и значение состояния

- Для известной стратегии  $\pi$   $V^\pi$  удовлетворяют уравнениям Беллмана:

$$\begin{aligned}
 V^\pi(s) &= \mathbb{E}_\pi [R_t \mid s_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right] = \\
 &= \mathbb{E}_\pi \left[ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s \right] = \\
 &= \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left( R_{ss'}^a + \gamma \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s' \right] \right) = \\
 &= \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V^\pi(s')).
 \end{aligned}$$



## Две основные задачи

- Предположим, что мы уже точно знаем нашу модель.  
Задача — найти оптимальную стратегию поведения для агента в этой модели.
- Реальная ситуация: модель не знаем, надо и модель узнать, и оптимально себя поведи.
- Сначала решим первую задачу, без неё все равно не будет второй.



# Оптимальные значения состояний

- *Оптимальное значение состояния* — ожидаемая суммарная прибыль, которую получит агент, если начнёт с этого состояния и будет следовать оптимальной стратегии:

$$V^*(s) = \max_{\pi} E \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right].$$

- Эту функцию можно определить как решение уравнений

$$V^*(s) = \max_a \sum_{s' \in S} P_{ss'}^a (R_{ss'}^a + \gamma V^*(s')),$$

а затем выбрать оптимальную стратегию

$$\pi^*(s) = \arg \max_a \sum_{s' \in S} P_{ss'}^a (R_{ss'}^a + \gamma V^*(s')).$$

- Как решать уравнения?



# Policy evaluation

- Чтобы посчитать value functions для данной стратегии  $\pi$ , можно просто итеративно пересчитывать по уравнениям Беллмана:

$$V^\pi(s) := \sum_a \pi(s, a) \sum_{s' \in \mathcal{S}} P_{ss'}^a (R_{ss'}^a + \gamma V^\pi(s')),$$

пока не сойдётся.

- Соответственно, для оптимального надо решать уравнения с максимумами:

$$V^*(s) := \max_a \sum_{s' \in \mathcal{S}} P_{ss'}^a (R_{ss'}^a + \gamma V^*(s')).$$



## Итеративное решение (по значениям)

- Можно то же самое по  $Q$ : пока не сойдётся,

$$Q(s, a) := \sum_{s' \in S} P_{ss'}^a \left( R_{ss'}^a + \gamma \sum_{a'} \pi(s, a') Q(s, a') \right).$$

- А потом просто  $V(s) := \max_a Q(s, a)$ .
- Оптимальное  $Q^*(s, a)$  тоже нехитро:

$$Q^*(s, a) := \sum_{s' \in S} P_{ss'}^a \left( R_{ss'}^a + \gamma \max_{a'} Q^*(s, a') \right).$$



## Другой вариант

- Пересчёт в предыдущем алгоритме использует информацию от всех состояний–предшественников. Можно сделать другой вариант:

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)).$$

- Он работает, если каждая пара  $(s, a)$  встречается бесконечное число раз,  $s'$  выбирают из распределения  $P_{ss'}^a$ , а  $r$  сэмплируют со средним  $R(s, a)$  и ограниченной дисперсией.
- Но ведь на самом деле нам не  $V$  и не  $Q$  нужно, а оптимальная стратегия...



# Улучшение стратегий

- Мы ищем  $V^\pi$ , чтобы улучшить  $\pi$ . Как улучшить  $\pi$ ?
- Естественная идея: давайте жадно выбирать  $a$  в  $s$  как  $\arg \max_a Q^\pi(s, a)$  после вычисления  $Q^\pi$ .
- Policy improvement theorem: для  $\pi$  и  $\pi'$ , если для всех  $s$

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s),$$

то  $\pi'$  не хуже  $\pi$ , т.е.  $\forall s V^{\pi'}(s) \geq V^\pi(s)$ .

- Как доказать?



## Улучшение стратегий

- Просто будем разворачивать  $V^\pi$ :

$$\begin{aligned}
 V^\pi &\leq Q^\pi(s, \pi'(s)) = \mathbb{E}_{\pi'} [r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_{t+1} = s] \\
 &\leq \mathbb{E}_{\pi'} [r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) \mid s_{t+1} = s] \\
 &\leq \dots \leq \\
 &\leq \mathbb{E}_{\pi'} [r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \mid s_{t+1} = s] = V^{\pi'}(s).
 \end{aligned}$$





# Итеративное решение (по стратегиям)

- Ищем оптимальную стратегию итеративным алгоритмом.
- PolicyIteration – инициализировать  $\pi$ , потом, пока  $\pi \neq \pi'$ , повторять:

- вычислить значения состояний для стратегии  $\pi$ , решив систему линейных уравнений

$$V^\pi(s) := \sum_a \pi(s, a) \sum_{s' \in S} P_{ss'}^a (R_{ss'}^a + \gamma V^\pi(s'));$$

- улучшить стратегию на каждом состоянии:

$$\pi'(s) := \arg \max_a Q^\pi(s, a) = \arg \max_a P_{ss'}^a (R_{ss'}^a + \gamma V^\pi(s'));$$

- Почему оно сходится?



# Итеративное решение (по стратегиям)

- Сходится, т.к. на каждом шаге строго улучшаем целевую функцию, а всего существует конечное число  $(|A|^{|S|})$  стратегий.
- Но, конечно, это медленно, надо  $V^\pi$  пересчитывать; проще делать на каждой итерации ровно один шаг пересчёта  $V^\pi$ , а потом сразу выбирать жадную стратегию:

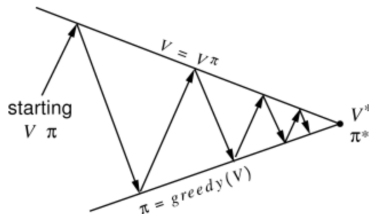
$$V_{k+1}(s) := \max_a \sum_{s' \in S} P_{ss'}^a (R_{ss'}^a + \gamma V_k(s')) .$$

- Это называется value iteration.



# Итеративное решение (по стратегиям)

- Есть другие похожие методы – их всех объединяет подход, основанный по сути на динамическом программировании. Оно может быть достаточно эффективно даже для больших задач (с трюками, позволяющими не всё пространство исследовать).





# Outline

- 1 **Агенты с несколькими состояниями**
  - Марковские процессы принятия решений
  - Поиск оптимальных стратегий в известной модели
- 2 **Поиск оптимальных стратегий без модели**
  - Стохастические алгоритмы



# Метод Монте-Карло

- Теперь будем обучать одновременно и модель, и оптимальную стратегию; вознаграждения и переходы не даны.
- Начнём со стохастических алгоритмов (метода Монте-Карло); но начнём опять с простой задачи.
- Как обучить вознаграждения  $V^\pi(s)$ , ожидаемые от состояния  $s$  в эпизодической задаче?



# Метод Монте-Карло

- Да очень просто: будем накапливать данные и усреднять.
  - 1 Сгенерировать эпизод по стратегии  $\pi$ .
  - 2 Для каждого состояния  $s$  из эпизода запомнить вознаграждение в конце эпизода.
  - 3  $V(s) := \text{Avg}(\text{rewards})$ .
- Есть тонкая разница между first-visit и every-visit.



# Метод Монте-Карло

- Но вообще без модели гораздо удобнее оценивать  $Q^*$ .
- Точно так же – но накапливаться будут только некоторые действия, особенно если  $\pi$  детерминированная.
- Возникает проблема, аналогичная многоруким бандитам.
- Но вообще смысл именно такой:

$$\pi_0 \rightarrow^E Q^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E Q^{\pi_1} \rightarrow^I \pi_2 \rightarrow \dots$$



# Метод Монте-Карло

- Надо гарантировать, что достаточно много эпизодов (легко) и что мы исследуем всё пространство (труднее).
- Monte Carlo ES (exploring starts): инициализировать случайные  $Q(s, a)$  и  $\pi(s)$ , потом повторять:
  - 1 сгенерировать эпизод по стратегии  $\pi$ ;
  - 2 для каждой пары  $(s, a)$  из эпизода запомнить вознаграждение в конце эпизода;
  - 3  $Q(s, a) := \text{Avg}(\text{rewards})$ ;
  - 4  $\pi(s, a) := \arg \max_a Q(s, a)$ .





# On-policy Monte Carlo

- Monte Carlo ES гарантирует успех, если мы как-то исследуем всё пространство  $(s, a)$ ; как?
- On-policy Monte Carlo – надо держать стратегию мягкой, с исследованием ( $\epsilon$ -жадную стратегию в каждом  $s$  или ещё что-то).

**Упражнение.** Докажите аналог policy improvement theorem для  $\epsilon$ -жадных стратегий.



# Off-policy Monte Carlo

- Однако удобнее было бы сделать так: следовать  $\epsilon$ -жадной стратегии, но  $Q^*(s, a)$  учить настоящие, а не  $\epsilon$ -мягкие.
- К счастью, так можно: пусть мы сгенерировали кучу эпизодов по стратегии  $\pi'$ , а обучить хотим  $Q^\pi$  и  $V^\pi$  для стратегии  $\pi$ .
- Предположим, что если  $\pi(s, a) > 0$ , то и  $\pi'(s, a) > 0$  (иначе, конечно, не получится).
- Что делать?



## Off-policy Monte Carlo

- Надо перевзвесить наблюдения: для  $n_s$  попаданий в  $s$  с вероятностями  $p_i$  и  $p'_i$

$$V(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)}}$$

- $p_i(s)$  и  $p'_i(s)$  неизвестны, но их отношение можно найти:

$$\frac{p_i(s)}{p'_i(s)} = \frac{\prod_{k=t}^{t_i(s)-1} \pi(s_k, a_k) P_{s_k s_{k+1}}^{a_k}}{\prod_{k=t}^{t_i(s)-1} \pi'(s_k, a_k) P_{s_k s_{k+1}}^{a_k}} = \prod_{k=t}^{t_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

**Упражнение.** А как оценивать  $Q(s, a)$ ?



# Off-policy Monte Carlo

- Соответственно, off-policy Monte Carlo control:
  - 1 выбрать  $\pi'$ , сгенерировать эпизод  
 $s_0, a_0, r_1, s_1, a_1, r_2, \dots, r_T, s_T$ ;
  - 2 пусть  $\tau$  – последнее время, когда  $a_\tau \neq \pi(s_\tau)$ ;
  - 3 для каждого первого визита в  $(s, a)$  в момент  $t > \tau$ :
    - $w := \prod_{k=t+1}^T \frac{1}{\pi'(s_k, a_k)}$ ;
    - $N(s, a) := N(s, a) + wR_t$ ;
    - $D(s, a) := D(s, a) + w$ ;
    - $Q(s, a) := N(s, a)/D(s, a)$ ;
  - 4 переопределяем жадную как  $\pi(s) := \arg \max_a Q(s, a)$ .



# TD-обучение

- Общий принцип TD-обучения тот же: давайте обучать состояния на основе обученных нами оценок для последующих состояний.
- $TD(0)$ -обучение: инициализировать  $V(s)$  и  $\pi$  произвольно, затем на каждом эпизоде обучения:
  - 1 инициализировать  $s$ ;
  - 2 для каждого шага в эпизоде:
    - выбрать  $a$  по стратегии  $\pi$ ;
    - сделать  $a$ , пронаблюдать результат  $r$  и следующее состояние  $s'$ ;
    - $V(s) := V(s) + \alpha[r + \gamma V(s') - V(s)]$ ;
    - $s := s'$ .



# TD-обучение

- Смысл в том, чтобы использовать уже обученные закономерности для поиска более глубоких закономерностей.
- В результате обучение получится целенаправленным, обучается гораздо быстрее, чем другие стратегии.



## TD-обучение

- $TD(0)$  смотрит на один шаг вперёд. Можно сделать алгоритм, учитывающий много шагов,  $TD(\lambda)$ :

$$V(u) := V(u) + \alpha(r + \gamma V(s') - V(s))\epsilon(u),$$

применяемый к каждому состоянию  $u$  на основе его *eligibility*  $\epsilon(u)$ .

$$\epsilon(s) = \sum_{k=1}^t (\lambda\gamma)^{t-k} [s = s_k].$$

- Eligibility — то, насколько часто это состояние посещалось в недавнем прошлом. Если  $\lambda = 0$ ,  $TD(\lambda)$  — это  $TD(0)$ . Eligibility можно пересчитывать в реальном времени:

$$\epsilon(s) := \gamma\lambda\epsilon(s) + [s = \text{текущему состоянию}].$$



## TD-обучение

- Sarsa – on-policy TD-обучение. По каждому данному  $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$  обновить

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)].$$

- При этом стратегия должна быть мягкой, например  $\epsilon$ -жадной с уменьшающимся  $\epsilon$ .
- Аналогично, Sarsa( $\lambda$ ) обновляет значения на всех парах  $(s, a)$  с учётом eligibility traces  $\epsilon(s, a)$ :

$$Q(s, a) := Q(s, a) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s, a)] \epsilon_t(s, a)$$

$$\epsilon_t(s, a) := \gamma \lambda \epsilon_{t-1}(s, a) + [s = s_t, a = a_t].$$





## TD-обучение

- Q-обучение – off-policy TD-обучение. Сразу решаем уравнения Беллмана относительно максимумов:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right].$$

- Теперь  $Q$  напрямую аппроксимирует оптимальную функцию  $Q^*$ , независимо от стратегии.
- Аналогично,  $Q(\lambda)$ :

$$Q(s, a) := Q(s, a) + \alpha \left[ r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \epsilon_t(s, a)$$

$$\epsilon_t(s, a) := \gamma \lambda \epsilon_{t-1}(s, a) + [s = s_t, a = a_t],$$

только теперь ещё надо забывать следы, если мы не следуем стратегии:  $\epsilon_t(s, a) := 0$ , если

$$Q(s_t, a_t) \neq \max_a Q(s_t, a).$$



# Адаптивный эвристический критик

- Основная идея — пусть один элемент алгоритма ищет стратегию, а другой — функцию значений, и пусть они друг с другом соперничают.
- Адаптивный эвристический критик (adaptive heuristic critic) состоит из *критика АНС* и *компонента обучения с подкреплением RL*.
- На месте RL может быть любой алгоритм обучения с подкреплением для решения задачи о  $k$  бандитах.



# Адаптивный эвристический критик

- RL максимизирует не прибыль, а значение эвристики  $v$ , вычисляемое критиком.
- АНС в это время использует полученное от RL значение для пересчёта ожидаемых значений прибыли от состояний.
- По очереди: фиксируем стратегию  $\pi$  и подсчитываем функцию значений  $V_\pi$ . Затем фиксируем  $V_\pi$  и учим новую стратегию  $\pi'$ , которая максимизирует  $V_\pi$ . И так далее.



# Адаптивный эвристический критик

- Как АНС выучит  $V_\pi$  по  $\pi$ ?
- $\langle s, a, r, s' \rangle$  — *experience tuple* ( $s$  — состояние перед переходом,  $a$  — действие,  $r$  — поощрение,  $s'$  — следующее состояние).
- Значение можно выучить по правилу  $TD(0)$ :

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s)).$$



# Certainty equivalence

- Можно просто сначала собрать опыт, по нему построить модель, а затем по этой модели вычислять оптимум.
- Недостатки:
  - Нужно проводить какую-то границу между обучением и действием.
  - Сбор данных может быть ужасно неэффективен.
  - Что, если окружающая среда вдруг изменится?
- Идея — одновременно строить модель, по опыту и по модели оптимизировать стратегию.



- При подаче на вход experience tuple  $\langle s, a, r, s' \rangle$  Dyna работает так:
  - Обновить модель, добавив статистику для перехода из  $s$  в  $s'$  под действием  $a$  и для получения награды за  $a$  в состоянии  $s$ . Получим  $\hat{T}$  и  $\hat{R}$ .
  - Обновить стратегию в состоянии  $s$ :

$$Q(s, a) := \hat{R}(s, a) + \gamma \sum_{s'} \hat{P}_{ss'}^a \max_{a'} Q(s', a').$$

- Сделать ещё  $k$  обновлений: для  $k$  случайно выбранных пар  $(s_k, a_k)$  ( $k$  произвольное — на сколько ресурсов хватит)

$$Q(s_k, a_k) := \hat{R}(s_k, a_k) + \gamma \sum_{s'} \hat{P}_{s_k s'}^{a_k} \max_{a'} Q(s', a').$$

- Выбрать  $a'$  в  $s'$  на основе значений  $Q$  (возможно, исправленных стратегией изучения среды).



# Уборка по приоритетам

- Дуна хороша, но бесцельна: обновляет случайные пары.
- Алгоритм prioritized sweeping делает то же, но у каждого состояния есть *приоритет*, и для каждого состояния запоминаются предшественники — состояния, из которых можно попасть в это состояние с ненулевой вероятностью.



# Уборка по приоритетам

- Алгоритм: точно такой же, как в Дупа, но значения теперь присваиваются состояниям, а не парам (состояние, действие), и вместо  $k$  случайных пар мы обновляем значения  $k$  наиболее приоритетных следующим образом:
  - Запомнить текущее значение  $V_{old} = V(s)$ .
  - Обновить

$$V(s) := \max_a \left( \hat{R}(s, a) + \gamma \sum_{s'} \hat{P}_{ss'}^a V(s') \right).$$

- Выставить состоянию  $s$  приоритет 0.
- Вычислить  $\Delta = |V_{old} - V(s)|$ .
- Для каждого состояния, для которого  $\hat{P}_{us}^a \neq 0$ , увеличить его приоритет на  $\Delta \hat{P}_{us}^a$ .





# Afterstates

- Часто обучают не по состояниям  $V(s)$  и не по парам  $Q(s, a)$ , а по т.н. *afterstates* – состояниям после действия.
- Это хорошо, когда действия имеют немедленный эффект, а случайный процесс происходит уже потом.
- Крестики-нолики – многие пары приводят к одной и той же позиции.

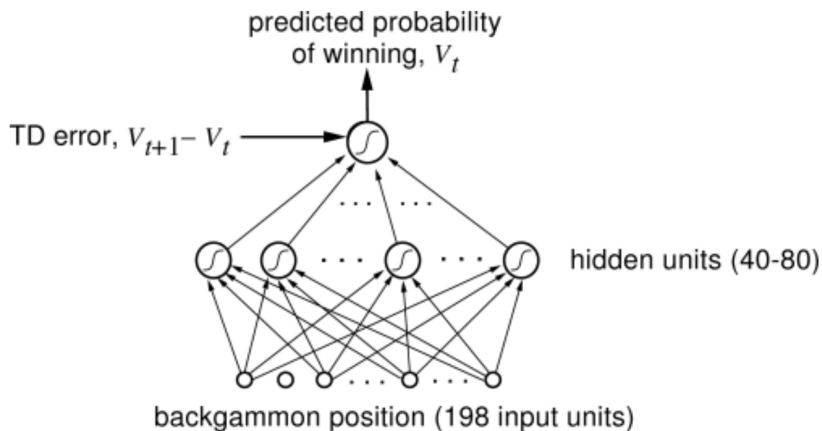


## Как обучить функцию

- Всё, что мы до сих пор говорили, использует значения вида  $Q(s, a)$  или  $V(s)$ , но этих самых состояний обычно астрономическое число.
- Поэтому надо обучать  $Q(s, a)$  методами машинного обучения, используя значения, предоставляемые RL-обучением.
- Обычно – нейронные сети.



## Пример: архитектура TD-Gammon





Thank you!

**Спасибо за внимание!**