Лекция 5: Обучение с подкреплением

Автор: Сергей Вячеславович Макрушин e-mail: SVMakrushin@fa.ru (mailto:SVMakrushin@fa.ru)

Финансовый университет, 2023 г.

При подготовке лекции использованы материалы:

- ...

v 0.3 01.11.23

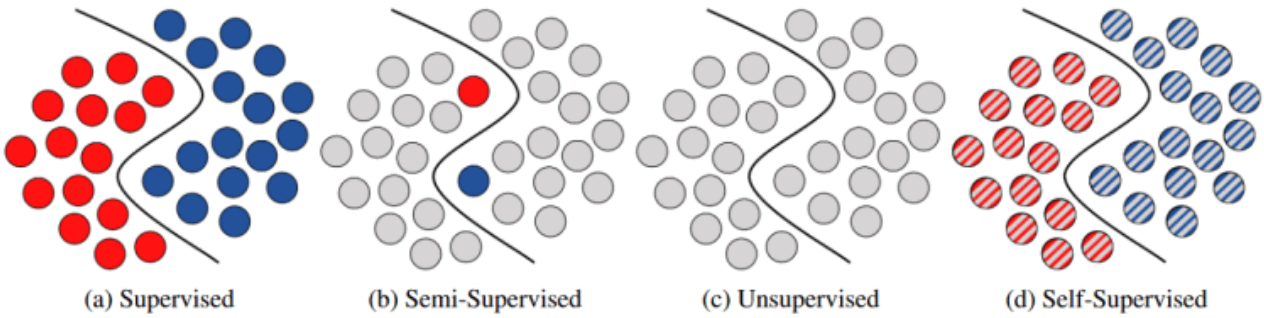
Разделы:

In [1]:
```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v2.css")
HTML(html.read().decode('utf-8'))
```

Out[1]:

Введение в обучение с подкреплением

Различные подходы к машинному обучению



Типы машинного обучения (Часть 1)

Обучение с учителем (Supervised Learning) - модель обучается **на размеченных данных**, где каждый входной пример имеет соответствующий целевой выход.

- **Цель** состоит в том, чтобы модель научилась **предсказывать целевой выход для новых**, ранее не виденных **данных**.
- **Задачи**: классификация и регрессия.

Обучение без учителя (Unsupervised Learning) - модель обучается на неразмеченных данных, где отсутствует целевой выход.

- **Цель** - найти внутренние структуры, закономерности или группировки в данных.
- **Задачи**: кластеризация, выявление аномалий, понижение размерности, поиск ассоциативных правил.

Обучение с частичным привлечением учителя (Semi-Supervised Learning) - комбинация обучения с учителем и обучения без учителя, где модель обучается на **небольшом наборе размеченных данных** и **большем наборе неразмеченных данных**.

- **Цель и задачи** аналогичны обучению с учителем, **идея** заключается в использовании неразмеченных данных для улучшения обучения модели и расширения ее способности к обобщению.

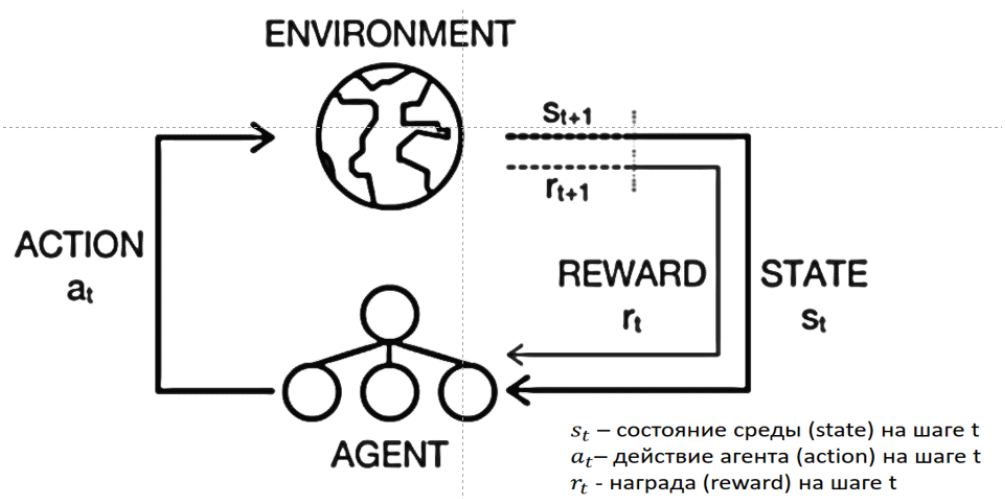
Самообучение (Self-Supervised Learning) - подходк к обучению, в котором модель обучается на неразмеченных данных, используя разнообразные задачи, создаваемые из самих данных, без необходимости внешних разметок. Вместо того, чтобы полагаться на учителя для предоставления меток (как в обучении с учителем), Self-Supervised Learning использует информацию, извлекаемую из данных, для самостоятельного обучения модели.

- **Цель и задачи** аналогичны обучению с учителем, входые данные - аналогичны обучению без учителя. (см.: https://dasha.ai/en-us/blog/self-supervised-machine-learning (https://dasha.ai/en-us/blog/self-supervised-machine-learning))

Принцип работы активного обучения

Активное обучение (Active Learning) - метод, применяемый в подходе **обучения с учителем**, обычно в случаях **когда получение меток дорого**, поэтому мы получаем **новые метки динамически**, определяя алгоритмическую **стратегию для максимизации полезности новых наблюдений**.

- Можно рассматривать активное обучение как **особый случай обучения с частичным привлечением учителя**.
- **Цель и задачи** аналогичны обучению с учителем, входые данные - без разметки, с возможностью (обычно дорогостоящей) динамической разметки.

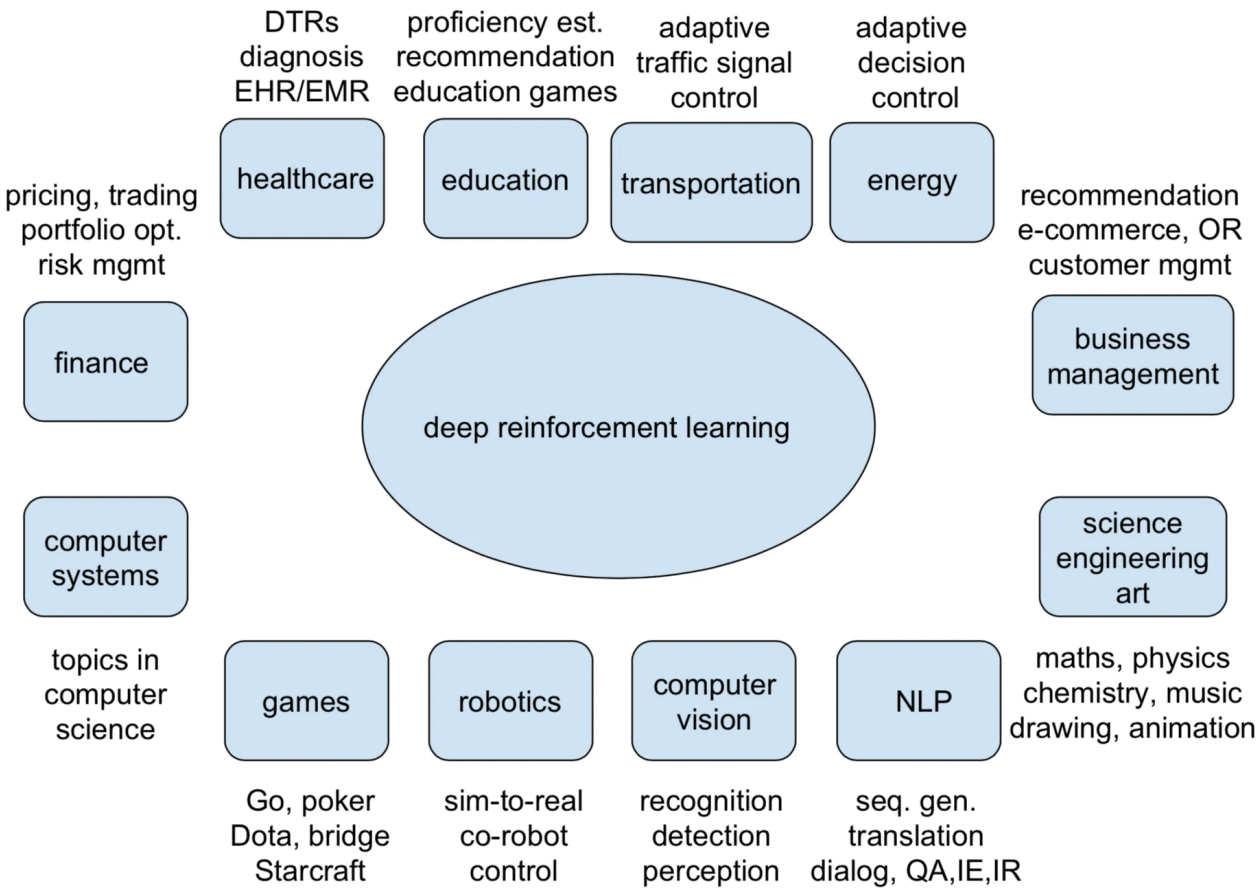


Принцип работы обучения с подкреплением

Обучение с подкреплением (Reinforcement Learning, RL) - модель обучается как **агент, взаимодействующий с окружающей средой** и максимизируют вознаграждение получаемое от системы.

- В RL агент самостоятельно исследует окружающую среду, принимая решения на основе своего опыта и обратной связи от среды.
- В RL нет меток:
  - нельзя использовать обучение с учителем
  - есть "вознаграждение", которое говорит насколько хорош текущий результат.Следовательно,
- В RL модель **учится стратегии** максимизации вознаграждения.
  - стратегия должна должна учитывать баланс между исследованием (получением новый информации о среде) и использованием существующих знаний о ней для максимизации вознаграждения.

Приложения обученяи с подкреплением



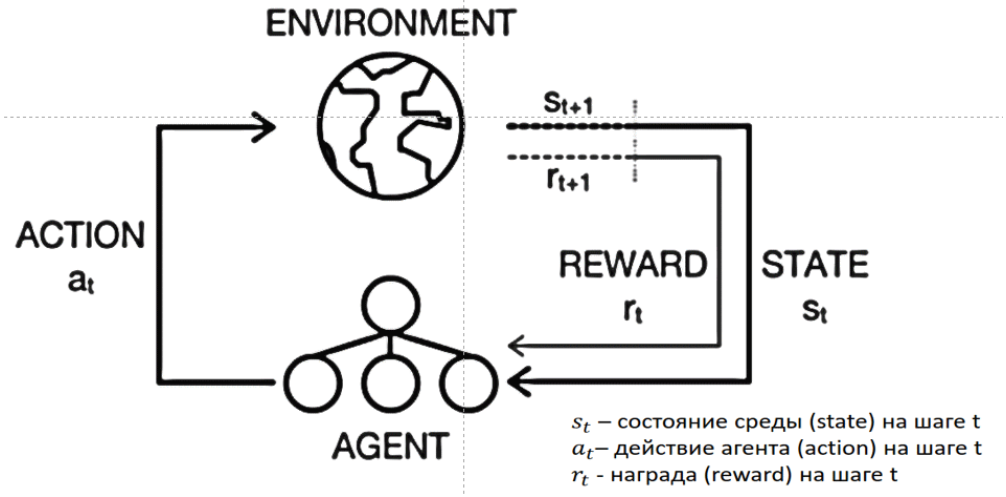
Yuxi Li, Deep Reinforcement Learning, arXiv, 2018

Приложения обученяи с подкреплением

Постановка задачи обучения с подкреплением

Обучение с подкреплением (reinforcement learning) — метод машинного обучения, при котором система обучается, взаимодействуя с некоторой средой. В обучении с подкреплением есть:

- **агент (agent)** кторый при помощи
- **действий (actions)** взаимодействует с
- окружающей **средой (environment)** описываемой
- внутренним **состоянием (state)**
- в ответ на действие среда возвращает **вознаграждение (reward)** за эти действия
- и меняет свое сотоятение.



Принцип работы обучения с подкреплением

- s_t - сотояние среды (state) на шаге t
- a_t - действия агента (agent) на шаге t
- r_t - вознаграждение (reward) на шаге t

- $s \in S$ - сотояния
- $a \in A$ - действия
- $p(s'|a,s)$ - модель переходов между состояниями (в общем случае - вероятностная)
  - модель переходов неизвестна агенту и в зависимости от подхода к решению задачи агент может пытаться, а может не пытаться узнать модель переходов.
- $\pi(a|s)$ - стратегия (policy, политика)
- $r(s,a)$ - вознаграждение
  - для многих задач построение функции вознаграждения сложная задача. Пример: шахматы.

Стратегия в RL

Стратегия (policy, политика) в RL - это стратегия или набор правил, которые определяют, какое действие должен выбирать агент должен при заданном состоянии среды.

- Формально может быть записано как распределение: $\pi(a|s)$
- Статегия определяет, как агент будет взаимодействовать с окружающей средой, и важна для достижения целей обучения с подкреплением.

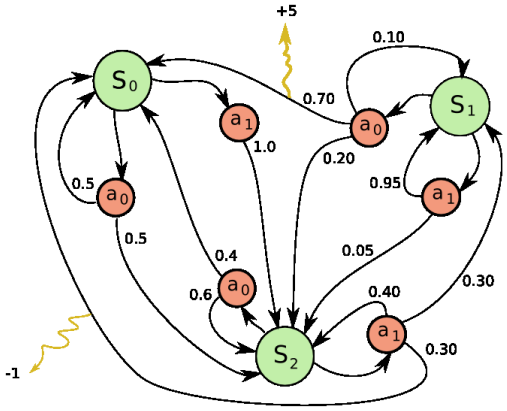
Стратегия может быть определена различными способами:

- **Детерминированная стратегия** - для каждого состояния среды существует конкретное действие, которое агент должен выполнить.
  - Например, если агент находится в состоянии A, он всегда выполняет действие X.
- **Случайная стратегия** - агент выбирает действия с определенными вероятностями для каждого состояния.
- **Параметризованная стратегия** - например, стратегия может быть параметризованной с использованием нейронных сетей, где параметры модели настраиваются в процессе обучения. В записи $\pi(a|s,\theta)$ θ это векторо параметров модели, например веса нейронной сети.

- **Цель обучения** с подкреплением заключается в **нахождении оптимальной стратегии**, которая максимизирует ожидаемое вознаграждение (reward) в долгосрочной перспективе.
- **Агент обучается, итеративно взаимодействуя с средой и обновляя свою стратегию**, чтобы лучше адаптироваться к окружающей среде и достигать своих целей.

Определение вознаграждения

- Траектория агента: $(s_0, a_0), (s_1, a_1), \ldots$



It is a special case of a Markov Decision Process (MDP): A graph where each node is a particular game state and each edge is a possible (in general probabilistic) transition. Each edge also gives a reward, and the goal is to compute the optimal way of acting in any state to maximize rewards.

- Суммарное дисконтированное вознаграждение тогда:
$$R_t = \gamma^0 r(s_t, a_t) + \gamma^1 r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \ldots$$
где $\gamma \in [0, 1]$ - дисконтирующий множитель

$$R_0 = \sum_{t=0}^{\infty} \gamma^t r_t$$

где:

- R - дисконтированное вознаграждение.
- t - шаг времени.
- γ - коэффициент дисконтирования, обычно $\gamma \in [0, 1]$.
- r_t - награда (reward) на шаге времени (t).

Это выражение представляет собой бесконечную сумму наград, взвешенных коэффициентами дисконтирования (\gamma^t), где (\gamma) определяет, как сильно будут учитываться будущие награды по сравнению с текущей.

Функции V и Q

Функция значения состояния (value function, $V^{\pi}(s)$) - ожидаемая сумма дисконтированных будущих вознаграждений при следовании политике (\pi) если начать из состоянии s:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]$$

Где:

- $\mathbb{E}_{\pi}$ - оператор математического ожидания по политике π.
- r_t - награда (reward) на временном шаге t.
- s_0 - начальное состояние.

Q-функция - ожидаемая сумма дисконтированных будущих вознаграждений при следовании политике (\pi) если начать из состоянии s при выполнении действия a:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right]$$

- Если бы мы знали функции V и Q то можно было бы просто выбирать то действие a, которое максимизирует $Q(s, a)$.

- Т.е. функции V и Q — это как раз то, что нам нужно оценить.

Уравнение Беллмана для функции значения состояния

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}r_{t} \mid s_{0} = s\right] =$$

первый член - это награда r_0, которая получается после выполнения действия a_0 в состоянии s_0:

$$= \mathbb{E}_{\pi}\left[r_{0} + \gamma\sum_{t=0}^{\infty}\gamma^{t}r_{t+1} \mid s_{0} = s\right]$$

распишем матожидание при выполнении первого шага стратегии:

$$V^{\pi}(s) = \sum_{a}\pi(a|s)\sum_{s'}P(s'|s,a)\left[R(s,a,s') + \gamma\mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}r_{t+1} \mid s_{0} = s'\right]\right]$$

выражение во вторых квадратных скобках это функция значения состояния для нового состояния s'

$$= \sum_{a}\pi(a|s)\sum_{s'}P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

Т.е. получаем рекурентную зависимость которая является уравнением Беллмана:

$$V^{\pi}(s) = \sum_{a}\pi(a|s)\sum_{s'}P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

Уравнение Беллмана играет важную роль в динамическом программировании: оно позволяет **рекурсивно вычислять оптимальные значения для всех состояний на основе оптимальных значений для более коротких траекторий**.

- Это ключевой элемент для поиска оптимальных стратегий в марковских процессах принятия решений и обучении с подкреплением.

Уравнение Беллмана для функции значения состояния $V(s)$ выглядит следующим образом:

Пусть функция значения состояния при оптимальной стратегии: $V^{*}(s) = \max_{\pi}\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^{t}r_{t}\right]$, тогда:

$$V^{*}(s) = \max_{a}\sum_{s'}P(s'|s,a)[R(s,a,s') + \gamma V^{*}(s')]$$

Где:

- $V(s)$ - значение состояния s, то есть ожидаемая сумма наград при следовании оптимальной стратегии из состояния s.
- a - действие, выбранное в состоянии s в соответствии с оптимальной стратегией.
- s' - следующее состояние.
- $P(s'|s,a)$ - вероятность перехода из состояния s в состояние s' при выполнении действия a.
- $R(s,a,s')$ - вознаграждение (reward), получаемая после выполнения действия a в состоянии s и перехода в состояние s'.
- γ - коэффициент дисконтирования.

Два подхода к нахождению оптимальной стратегии

Теоретически для того чтобы найти значения $V(s)$, можно просто решить систему линейных уравнений, неизвестными в которых являются $V(s)$ для разных состояний s.

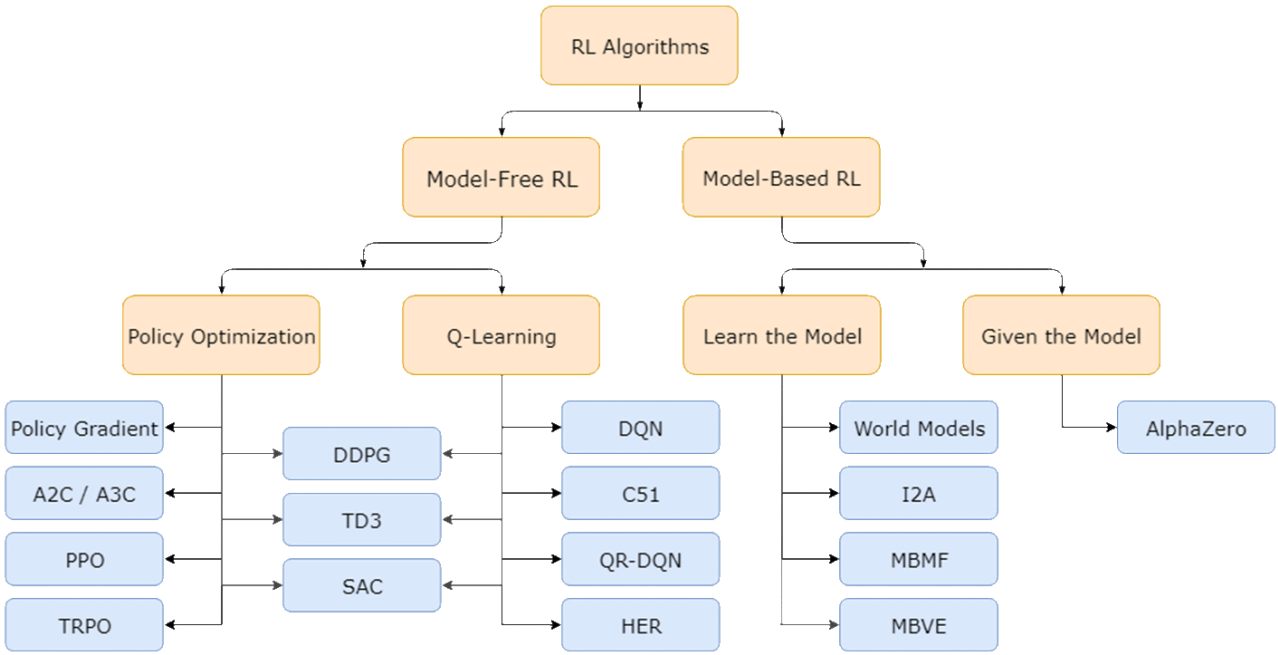
Но **для этого нужно знать все параметры марковского процесса**, то есть функции R и P, а с этим в реальных ситуациях все сложно.

- Все, что у нас обычно есть на входе — **окружающая среда, выдающая вознаграждения как черный ящик**. Т.е. в реальных задачах функции R и P тоже приходится обучать.

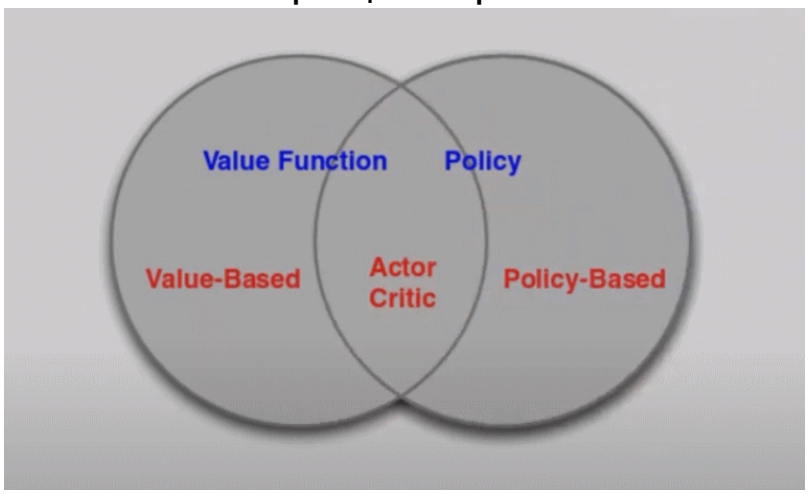
Методы обучения с подкреплением делятся на те, которые:

- обучают функции R и P в явном виде
- обходятся без этого и сразу обучают V и/или Q.

Классификация алгоритмов RL



Классификация алгоритмов RL



Подходы к RL

Метод Policy Gradient

Введение в метод Policy Gradient

Policy gradient Класс методов reinforcement learning в которых стратегию $\pi_\theta(a|s)$ оптимизируют напрямую (в отличие от **Q-learning**).

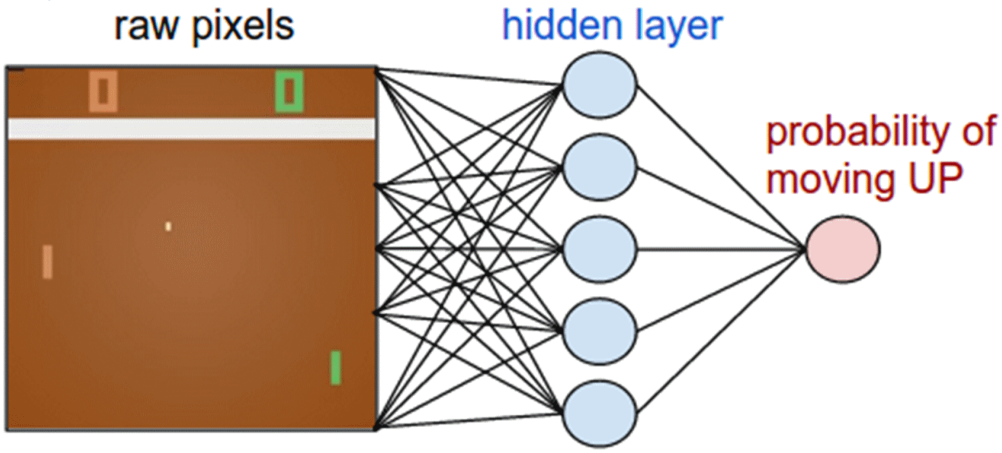
Интуитивное объяснение принципа работы:

$p_\theta(\tau)$ - вероятность того, что будет реализован сценарий τ при условии параметров модели θ, т. е. функция правдоподобия. Нам хочется:

- увеличить правдоподобие "хороших" сценариев (обладающих высоким R_τ)
- понизить правдоподобие "плохих" сценариев (с низким R_τ).

Игра Pong

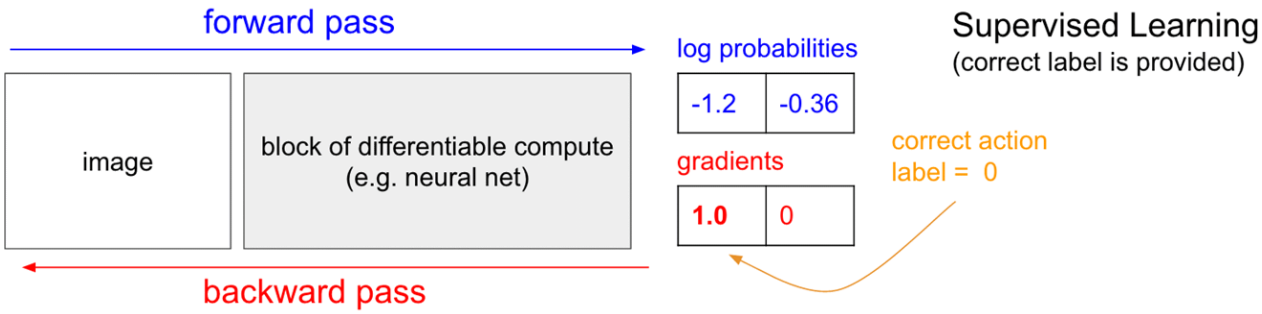
Игра Pong: https://www.youtube.com/watch?v=fiShX2pTz9A (https://www.youtube.com/watch?v=fiShX2pTz9A)



Нейронная сеть для обучения игре Pong

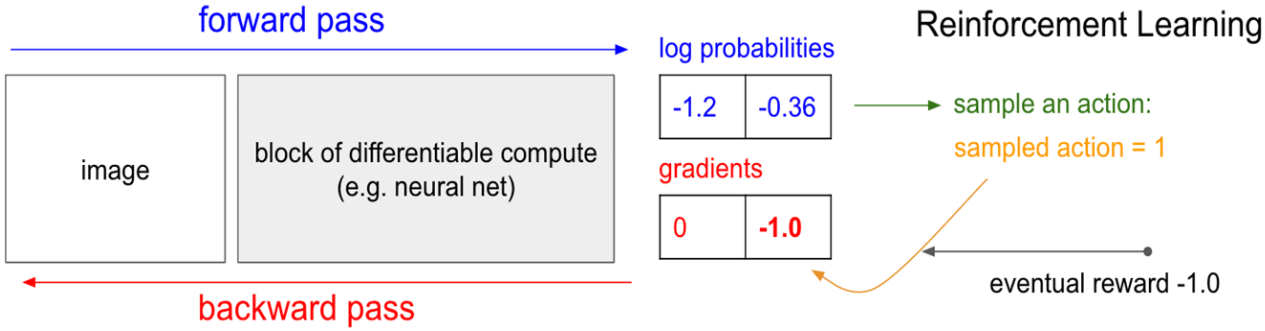
- На нейроны подается информация **о разности** значений пикселей в последующих кадрах игры.
- Выходной нейрон - вероятность двигать ракетку вверх (иначе двигаем ракетку вниз).

Принцип обучения методом Policy Gradient



Процесс обучения при обучении с учителем

При обучении с учителем на каждом шаге мы знаем правильный ответ (правильный класс: "вверх" или "вниз"), на основании которого и проводим обратное распространение ошибки.



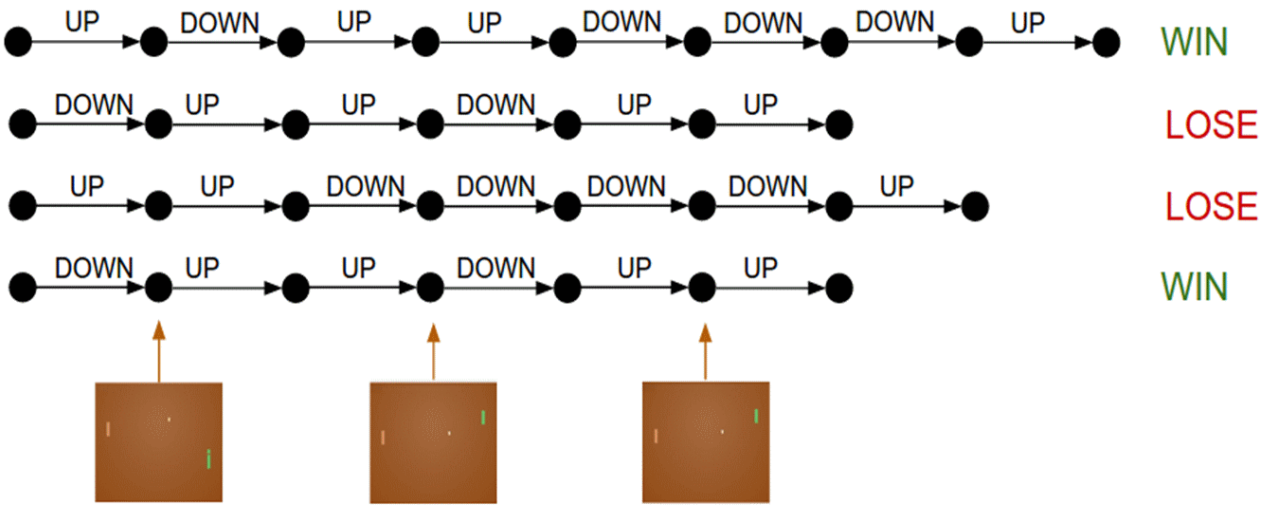
Процесс обучения в Policy Gradient

- У нас нет правильного ответа для каждого шага игры, есть только ответ для всей игры: "выиграл" или "проиграл".
- **В случае выигрыша** действие на каждом шаге (выбранный класс) поощряется: устанавливается что оно было верным.

- **В случае проигрыша** действие на каждом шаге (выбранный класс) штрафуется:

Пример сценариев обучения в Policy Gradient

- **Каждая точка** представляет собой некоторое игровое **состояние** (три примера состояния визуализированы внизу)
- **Каждая стрелка** - это переход (шаг по времени): действие (указано у стрелки) и изменение состояния.



Пример 4 тректорий игры в Pong

- В этих примерах мы выиграли 2 игры и проиграли 2 игры.

Используя Policy Gradients:

- мы поощряем две выигранные игры: слегка поощряем каждое действие, которое мы совершили в этом эпизоде.
- мы штрафуем две проигранные игры: слегка штрафуем каждое действие, которое мы совершили в этом эпизоде.

- Pong AI with Policy Gradients https://www.youtube.com/watch?v=YOW8m2YGtRg&list=TLGG2Hi1VyMnZ7gwMjExMjAyMw (https://www.youtube.com/watch?v=YOW8m2YGtRg&list=TLGG2Hi1VyMnZ7gwMjExMjAyMw)
  - The learned agent (in green, right) facing off with the hard-coded AI opponent (left).

Loss Policy Gradients

Обозначения:

- $p(x) = p(x|s, \theta)$ - policy function, θ - веса сети
- $f(x)$ - вознаграждения

По

$$
\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) && \text{definition of expectation} \\
&= \sum_x \nabla_\theta p(x) f(x) && \text{swap sum and gradient} \\
&= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x) && \text{both multiply and divide by } p(x) \\
&= \sum_x p(x) \nabla_\theta \log p(x) f(x) && \text{use the fact that } \nabla_\theta \log(z) = \frac{1}{z} \nabla_\theta z \\
&= E_x[f(x) \nabla_\theta \log p(x)] && \text{definition of expectation}
\end{aligned}
$$

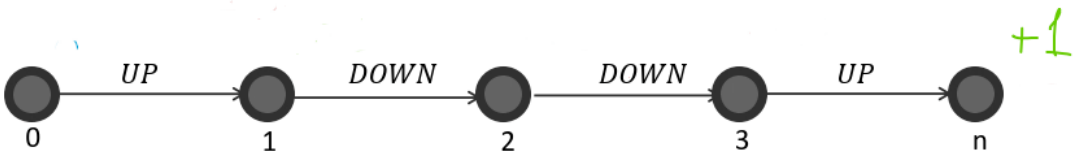
...

- Что весьма похоже на Cross-entropy loss

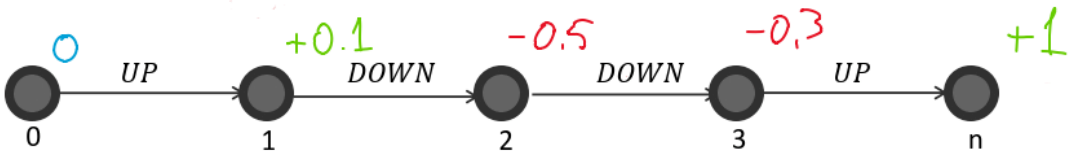
Алгоритм обучения с помощью Policy Gradients

1. Разыгрываем N игр при помощи имеющейся нейросети реализующей стратегию $\pi(a|s)$ - получаем сценарии игр для обучения.
2. На основе Loss Policy Gradients по полученным сценариям игр обучаем нейросеть стратегии.
3. Меняем нейросеть стратегии $\pi(a|s)$ на обученную, переходим к п.1

Ссылка: http://karpathy.github.io/2016/05/31/rl/ (http://karpathy.github.io/2016/05/31/rl/)



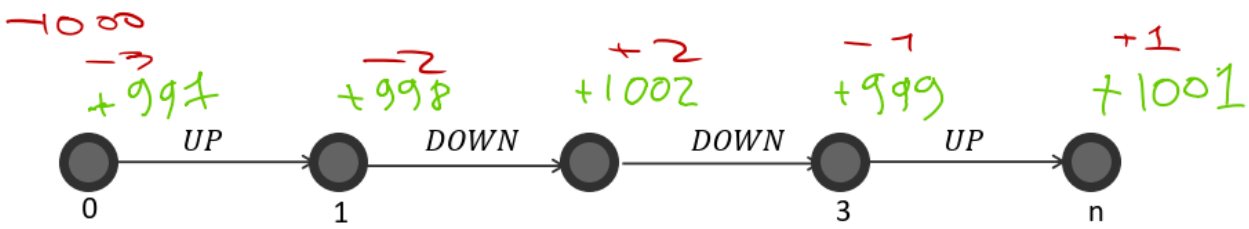
...



$$R_0 = r_0 + r_1 + \cdots + r_n \qquad R_2 = r_2 + r_3 + r_4 + \cdots + r_n$$

$$R_0 = r_0 + \gamma\, r_1 + \gamma^2\, r_2 + \cdots + \gamma^n r_n \qquad R_2 = r_2 + \gamma\, r_3 + \gamma^2\, r_3 + \cdots + \gamma^{n-2} r_n \qquad \gamma < 1$$

...



$$R_s = \sum_{i=0\ldots n-s} \gamma^i r_{s+i} \qquad \gamma < 1$$

$$R_s = \sum_{i=0\ldots n-s} \gamma^i r_{s+i} - b \qquad b = E_{rollouts}\left[\sum_i \gamma^i r_{s+i}\right]$$

Baseline

Преимущества Policy gradient:

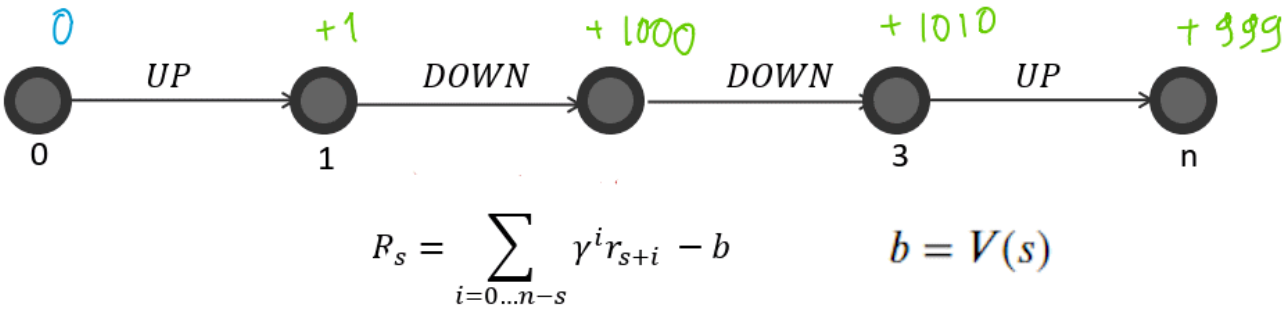
- **Легко обобщается на задачи с большим множеством действий_**, в том числе на задачи с непрерывным множеством действий;
- По большей части **избегает конфликта между эксплуатацией (exploitation) и исследованием (exploration)**, так как оптимизирует напрямую стохастическую стратегию $\pi_\theta(a|s)$;
- Имеет **более сильные гарантии сходимости**: если Q-learning гарантированно сходится только для МППР с конечными множествами действий и состояний, то policy gradient сходится к локальному оптимуму всегда, в том числе в случае бесконечных множеств действий и состояний.

Недостатки Policy gradient:

- **Очень низкая скорость работы** - требуется большое количество вычислений для оценки $\nabla_\theta J(\theta)$ по методу Монте-Карло, так как:
  - для получения всего одного семпла требуется произвести T взаимодействий со средой;
  - случайная величина $\nabla_\theta \log p_\theta(\tau) R_\tau$ имеет большую дисперсию поэтому для точной оценки $\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R_\tau \right]$ требуется много семплов;
    - **семплы**, собранные для предыдущих значений θ, **никак не переиспользуются на следующем шаге**, семплирование нужно делать заново на каждом шаге градиентного спуска.
- В случае конечных марковских процессов принятия решений (МППР) Q-learning сходится к глобальному оптимуму, тогда как **policy gradient может застрять в локальном**.

Actor-Critic

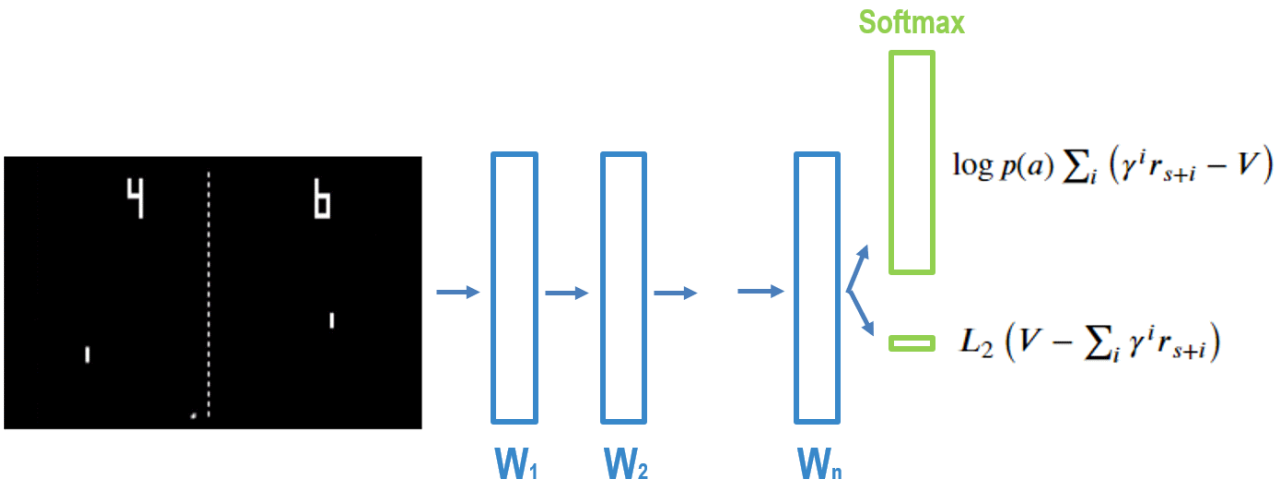
В модели Actor-Critic обучается функции значения состояния $b = V(s)$ предсказывающая дисконтированное вознаграждение по текущему состоянию:



$$R_s = \sum_{i=0 \ldots n-s} \gamma^i r_{s+i} - b \qquad b = V(s)$$

Baseline

Обучение $b = V(s)$ проводится параллельно собучением Policy gradient рассматривающей не исходое вознаграждение а дельту вознаграждения ($\log p(a) \sum_i \left(\gamma^i r_{s+i} - V \right)$) и ожидаемого значения состояния для текущего состояния:



Baseline

Q-Learning

Уравнение Беллмана для функции значения состояния

Уравнение Беллмана для функции значения состояния $V(s)$ выглядит следующим образом:

Пусть функция значения состояния при оптимальной стратегии: $V^*(s) = \max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, тогда:

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

Где:

- $V(s)$ - значение состояния s, то есть ожидаемая сумма наград при следовании оптимальной стратегии из состояния s.
- a - действие, выбранное в состоянии s в соответствии с оптимальной стратегией.
- s' - следующее состояние.
- $P(s'|s,a)$ - вероятность перехода из состояния s в состояние s' при выполнении действия a.
- $R(s,a,s')$ - вознаграждение (reward), получаемая после выполнения действия a в состоянии s и перехода в состояние s'.
- γ - коэффициент дисконтирования.

Уравенение Беллмана для функции Q

Q-функция - ожидаемая сумма дисконтированных будущих вознаграждений при следовании политике (\pi) если начать из состоянии s при выполнении действия a:

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right]$$

- Если бы мы знали функции V и Q то можно было бы просто выбирать то действие a, которое максимизирует $Q(s,a)$.
- Т.е. функции V и Q — это как раз то, что нам нужно оценить.

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a\right]$$

$Q^*(s,a)$ – Q-функция оптимальной стратегии

$\mathcal{E}$ – пространство возможных следующих состояний

Итеративное приближение:

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

Приближаем Q^* нейросетью с параметрами θ:

$$Q(s,a;\theta) \approx Q^*(s,a)$$

Формулируем функцию ошибки как приближние Q^* :

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a\right]$$

—

Алгоритм DQL

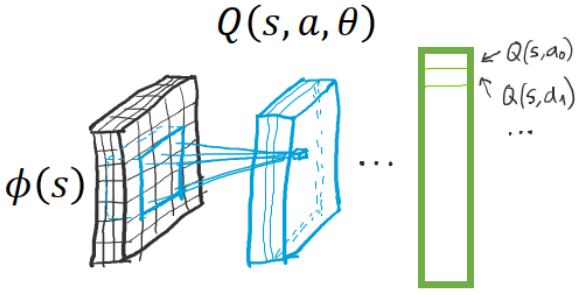
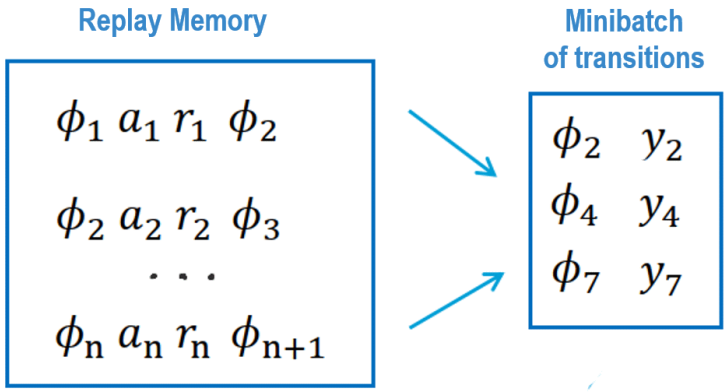


Схема нейросети Q для игр на Atari

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for



Работа с памятью эпизодов

Сравнение подходов

Policy Gradient	Q-learning
On-policy	Off-policy
Эффективнее исследует	Эффективнее сэмплирует
Проще	Сложнее
Обычно дольше	Обычно быстрее
Чаще работает	Когда работает, сходится стабильнее

Сравнение подходов

Спасибо за внимание!

Технический раздел:

In [ ]:

- https://neerc.ifmo.ru/wiki/index.php?title=Методы_policy_gradient_и_алгоритм_асинхронного_актора-критика (https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D1%8B_policy_gradient_%D0%B8_%D0%B0%D0%BA%D1%80%D0%B8%D1%82%D0%B8%D0%BA%D0%B0)
- https://neerc.ifmo.ru/wiki/index.php?title=Обучение_с_подкреплением (https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D0%

методы, которые позволяют оптимизировать стратегию $\pi_\theta(s|a)$ напрямую. Такие алгоритмы относятся к классу алгоритмов "policy gradient".

Интуитивное объяснение принципа работы:

$p_\theta(\tau)$ - вероятность того, что будет реализован сценарий τ при условии параметров модели θ, т. е. функция правдоподобия. Нам хочется увеличить правдоподобие "хороших" сценариев (обладающих высоким R_τ) и понизить правдоподобие "плохих" сценариев (с низким R_τ).

In [ ]:

Policy gradient Класс методов reinforcement learning в которых стратегию $\pi_\theta(s|a)$ оптимизируют напрямую (в отличие от *Q-learning*).

Интуитивное объяснение принципа работы:

$p_\theta(\tau)$ - вероятность того, что будет реализован сценарий τ при условии параметров модели θ, т. е. функция правдоподобия. Нам хочется увеличить правдоподобие "хороших" сценариев (обладающих высоким R_τ) и понизить правдоподобие "плохих" сценариев (с низким R_τ).

Двигаясь вверх по градиенту функции полного выигрыша мы повышаем логарифм функции правдоподобия для сценариев, имеющих больший положительный R_τ.

In [ ]:

Преимущества Policy gradient:

- Легко обобщается на задачи с большим множеством действий, в том числе на задачи с непрерывным множеством действий;
- По большей части избегает конфликта между эксплуатацией (exploitation) и исследованием (exploration), так как оптимизирует напрямую стохастическую стратегию $\pi_\theta(a|s)$;
- Имеет более сильные гарантии сходимости: если Q-learning гарантированно сходится только для МППР с конечными множествами действий и состояний, то policy gradient сходится к

локальному оптимуму всегда, в том числе в случае бесконечных множеств действий и состояний.

Недостатки Policy gradient:

- Очень низкая скорость работы - требуется большое количество вычислений для оценки $\nabla_\theta J(\theta)$ по методу Монте-Карло, так как:
  - для получения всего одного семпла требуется произвести T взаимодействий со средой;
  - случайная величина $\nabla_\theta \log p_\theta(\tau) R_\tau$ имеет большую дисперсию поэтому для точной оценки $\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R_\tau \right]$ требуется много семплов;
    - семплы, собранные для предыдущих значений θ, никак не переиспользуются на следующем шаге, семплирование нужно делать заново на каждом шаге градиентного спуска.
- В случае конечных марковских процессов принятия решений (МППР) Q-learning сходится к глобальному оптимуму, тогда как policy gradient может застрять в локальном

In [ ]:

next **Q:** qs line
next **A:** an line
next **Note:** an line
next **Def:** df line
next **Ex:** ex line
next **+** pl line
next **-** mn line
next **±** plmn line
next ⇒ hn line

In [ ]: