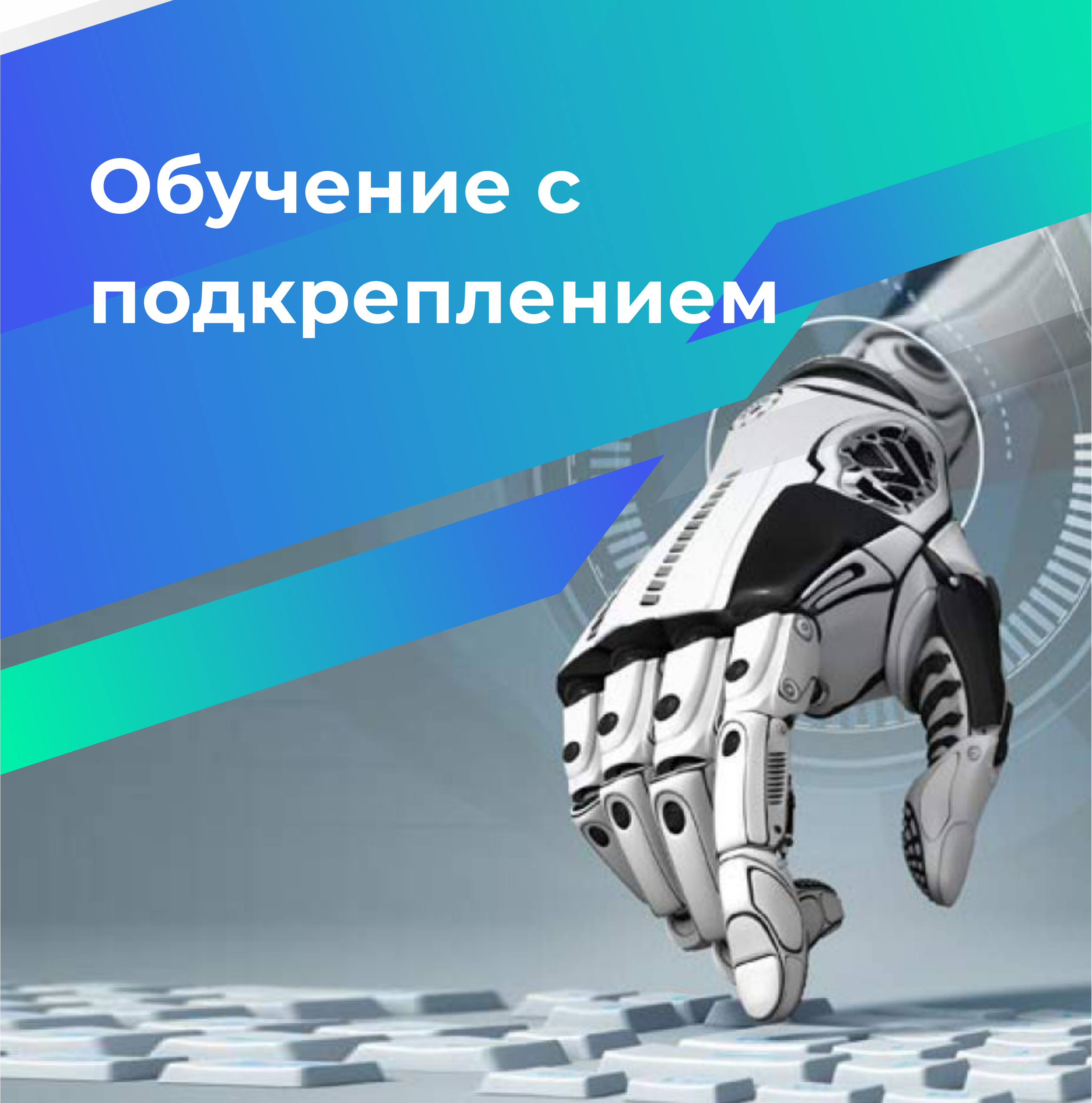
УНИВЕРСИТЕТ
искусственного
интеллекта

# Обучение с подкреплением



# Базовые понятия

**Обучение с подкреплением (англ. *reinforcement learning, RL*)** — один из способов машинного обучения, связанный с принятием решений и контролем моторики. В ходе обучения с подкреплением испытуемая система (*агент, в нашем случае им может выступать модель*) учится, взаимодействуя с некоторой *средой*. Основная цель данного способа — изучить, как агент может научиться достигать своих целей в сложной неопределенной среде.

RL носит очень общий характер и охватывает все проблемы, связанные с принятием последовательности решений: например, управление двигателями робота, чтобы он мог бегать и прыгать, принятие деловых решений, таких как ценообразование и управление запасами, участие в видеоиграх и настольных играх. RL может даже применяться к контролируемым задачам обучения с последовательными или структурированными выходами.

Откликом среды (а не специальной системы управления подкреплением, как это происходит в обучении с учителем) на принятые решения являются *сигналы подкрепления*, поэтому такое обучение является частным случаем обучения с учителем, но учителем является среда или её модель. Также нужно иметь в виду, что некоторые правила подкрепления базируются на неявных учителях, например, в случае искусственной нейронной среды, на одновременной активности формальных нейронов, из-за чего их можно отнести к обучению без учителя.

**Агентом** мы будем называть наш алгоритм, который умеет

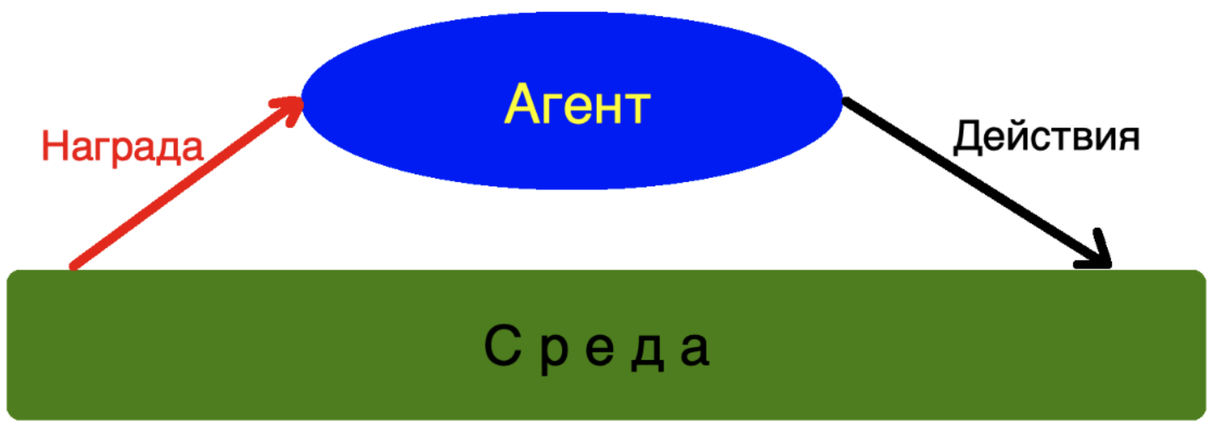
# Базовые понятия

анализировать состояние среды и совершать в ней какие-то действия.

**Среда** — виртуальный мир, в котором существует наш Агент и своими действиями может менять его состояние...

**Награда** — обратная связь от Среды к Агенту как ответ на его действия.

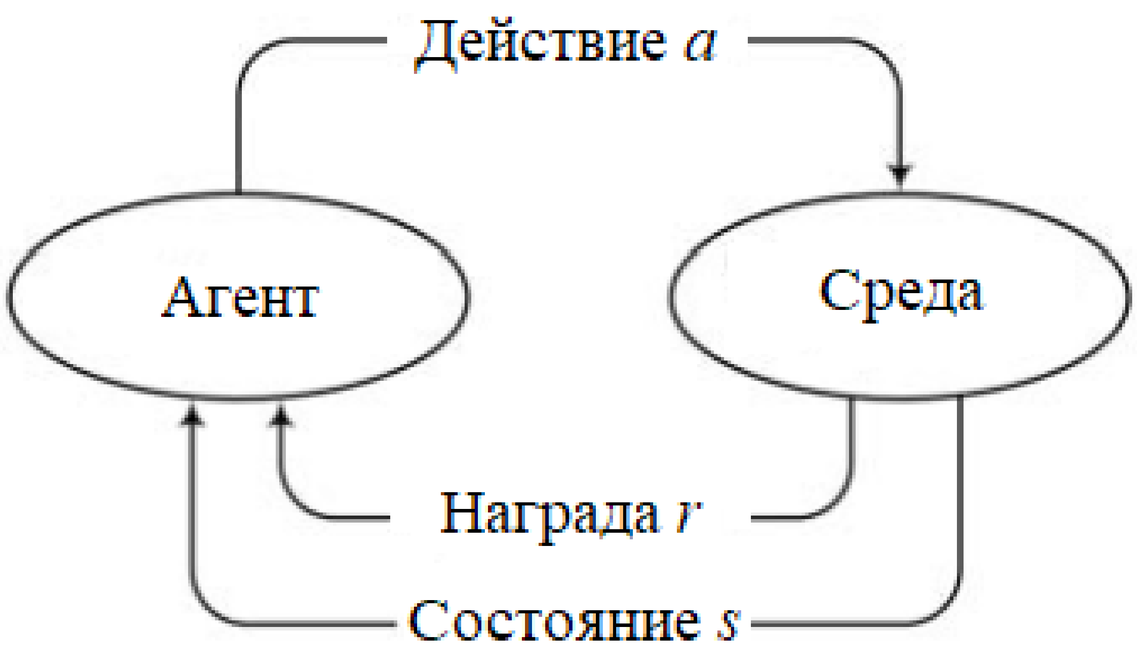
Агент воздействует на среду, а среда воздействует на агента. О такой системе говорят, что она имеет **обратную связь**. Мы можем рассмотреть это на примере среды:  гоночной машинки на дороге, которой управляет наша модель — автопилот (см. видео занятия). Такую систему нужно рассматривать как единое целое, и поэтому линия раздела между средой и агентом достаточно условна.




Среда, в которой обитает наш Агент, может быть сколь угодно сложной, агент может и не знать вовсе, как она устроена, чтобы принимать свои решения и выполнять действия. Для Агента важна лишь обратная связь в виде награды, которую он получает от среды.

Если более подробно рассмотреть процесс взаимодействия агента со средой, его можно выразить следующей схемой:

# Базовые понятия




Конечно, с анатомической или физической точек зрения между средой и агентом (организмом) существует вполне определённая граница, но если эту систему рассматривать с функциональной точки зрения, то разделение становится нечетким. Так, если смотреть примеры из жизни, резец в руке скульптора можно считать либо частью сложного биофизического механизма, придающего форму куску мрамора, либо частью материала, которым пытается управлять нервная система.

Алгоритмы RL начали достигать хороших результатов во многих сложных условиях. RL имеет долгую историю, но до недавних достижений в области глубокого обучения требовалось много инженерии, ориентированной на конкретные проблемы. Результаты DeepMind для Atari, BRETT из группы Питера Аббеля и AlphaGo использовали алгоритмы глубокого RL, которые не делали слишком много предположений об их среде и, следовательно, могут применяться в других условиях.

# Обучение с подкреплением

## Система подкрепления и её виды

Системой подкрепления называется любой набор правил, на основании которых можно изменять с течением времени матрицу взаимодействия (или состояние памяти) перцептрона.

Кроме классического метода обучения перцептрона — метода коррекции ошибки, который можно отнести к обучению с учителем, Розенблатт также ввёл понятие обучения без учителя, предложив несколько способов обучения.

***Альфа-системой*** подкрепления называется система подкрепления, при которой веса всех активных связей $c_{ij}$, которые ведут к элементу $u_j$, изменяются на одинаковую величину r, а веса неактивных связей за это время не изменяются.

***Гамма-системой*** подкрепления называется такое правило изменения весовых коэффициентов некоторого элемента, при котором веса всех активных связей сначала изменяются на равную величину, а затем из их всех весов связей вычитается другая величина, равная полному изменению весов всех активных связей, делённому на число всех связей. Эта система обладает свойством консервативности относительно весов, так как у неё полная сумма весов всех связей не может ни возрастать, ни убывать.

## Алгоритмы обучения с подкреплением

Основные два подхода к обучению с подкреплением, которые реализуем в наших занятиях - это Q-learning и Policy Gradient.

### Q-Learning

Этот алгоритм обучения с подкреплением возьмём как базовый, от которого стоит отталкиваться при рассмотрении остальных. В простейшем варианте в нём даже не используются нейронные

# Обучение с подкреплением

сети или функции, агент предпринимает действия, выбирая их из lookup-таблицы, так называемой Q-table. Величины Q представляют собой математические ожидания полной суммы вознаграждений, которые будут получены к концу эпизода. Вначале таблица пуста, обучение сводится к заполнению таблицы по некоторому алгоритму на основании полученного опыта.

Формальное описание алгоритма из литературы:

**Random-sample one-step tabular Q-planning**

Loop forever:

1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send $S, A$ to a sample model, and obtain
   a sample next reward, $R$, and a sample next state, $S'$
3. Apply one-step tabular Q-learning to $S, A, R, S'$:
   $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

Основное отличие данного метода в том, что на основе награды формируется функция полезности для агента, что впоследствии дает ему возможность уже не случайно выбирать стратегию поведения, а учитывать опыт предыдущего взаимодействия со средой. При таком подходе, просто нет необходимости анализировать среду, и так ясно, как действовать для получения награды.

С этой точки зрения алгоритм Q-learning можем записать так:

1. Инициализация функции полезности.
2. Наблюдение: запоминание прежнего состояния, прежнего действия, получить текущее состояние, а также вознаграждение за предыдущее действие.
3. Выбор действия.
4. Обновление функции полезности.
5. Повторение цикла.

Данный алгоритм годится только для очень простых, «игрушечных» задач с небольшим количеством состояний и небольшим дискретным набором возможных действий. Это связано с тем, что размер Q таблицы очень быстро растет

# Обучение с подкреплением

с увеличением сложности модели и исследование всех возможных состояний становится невозможным в разумное время.

## Deep Q-Learning

Этот алгоритм является усовершенствованием предыдущего, в нем Q-таблица или функция заменяется на нейронную сеть, которая по заданному начальному состоянию предсказывает величины Q для каждого возможного действия. Далее остается либо выбрать действие с максимальной величиной Q (так называемое «жадное» поведение), либо продолжить исследование среды, выбрав случайное действие. На начальных этапах обучения выбираются случайные действия, так как сеть еще не умеет с достаточной точностью предсказывать последствия действий. По мере обучения можно начинать действовать по рекомендациям сети. Алгоритм также не лишен недостатков: для него характерно обучение с резкими «провалами памяти», когда модель хорошо обучается, а потом вдруг резко забывает все, чему научилась. Еще один недостаток: алгоритм годится только для сред с дискретным набором действий. К настоящему времени разработано несколько усовершенствованных версий алгоритма, призванных устранить «провалы памяти».

## Policy Gradient

Данный алгоритм использует совершенно другой подход, в нем нейронная сеть получает на вход состояние и выдает policy — распределение вероятностей принятия того или иного действия. Агенту следует разыграть случайную величину в соответствии с полученным распределением, получить действие и передать его в среду. Так сеть работает в режиме предсказания действия. В режиме обучения используется несколько модифицированная сеть — она использует те же слои и те же веса, что и «рабочая» сеть, но имеет дополнительный вход для подачи величины advantage, которая представляет собой сумму дисконтированных наград, полученных в ответ на действия, предпринятые в прошлом в этом состоянии. Обучение

# Обучение с подкреплением

происходит на данных о прошлых шагах, которые агент запоминает и использует для обучения. Обучение проводится после завершения каждого эпизода, либо, как вариант, после N эпизодов. В качестве функции потерь используется кастомная функция — произведение логарифма вероятности действия на суммарную дисконтированную величину reward (т. е. advantage). Фактически в качестве потери используется математическое ожидание суммарной награды, получаемой к концу эпизода, взятое со знаком минус. Градиентный спуск ищет локальный минимум этой величины по отношению к весам сети, т. е. локальный максимум суммарной награды. Таким образом оптимизируется стратегия агента — стремление к максимальной конечной сумме награды. Ниже приведен алгоритм в формальной математической записи из литературы:

**REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for $\pi_*$**

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\quad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Основное отличие данного подхода — это именно то, что мы максимизируем награду, при этом в отдельных случаях возможно представить так, что лосса как такового нет.

Policy Gradient можем сравнить с тренировкой точной мышечной реакции. Последовательность действий в упрощённом для понимания виде при этом такая:

1. Агент наблюдает состояние среды.
2. Он совершает предсказанное действие (Policy).
3. Он двигается, оппоненты реагируют на действие. Формируется новое состояние среды.
4. Переходим к п.1.

# Обучение с подкреплением

Преимуществом алгоритма Policy Gradient является возможность его использования в задачах с непрерывным пространством действий, где не применим алгоритм Deep Q-Learning. Недостатками являются медленная сходимость и потенциальная возможность свалиться в процессе обучения в локальный минимум, не являющийся оптимальной политикой. Deep Q-Learning и Policy Gradient можно рассматривать как основные базовые алгоритмы. Предложен ряд усовершенствований для этих алгоритмов.

## Как мы поступаем с обратной связью, подкреплением

Чтобы наш агент обучался, в обучении с подкреплением обязательна обратная связь. В лекции хорошо показано это на примере машинки. Когда машинка наезжает на препятствие, это, конечно, должно приводить к наказанию, награда отрицательна. А когда машинка подбирает что-то полезное, то это хорошо, награда положительна. Но на самом деле, действие или набор действий, которые привели к удаче или неудаче, начались существенно раньше. Это произошло в тот момент, когда машинка поменяла направление движения. И при этом у неё ещё была возможность выправить траекторию своего движения. Но чем ближе к развязке, тем труднее избежать ситуации. Поэтому награда должна распространяться в те эпизоды времени, которые происходили до события. Но логично, что чем дальше от самого события, тем быстрее убывает награда. В ноутбуке занятия обратная связь реализована через коэффициент, меньший 1, на который происходит умножение на каждом новом шаге «в прошлое».

# Обучение с подкреплением

## Известные реализации — библиотеки обучения с подкреплением:

OpenAI Gym — платформа для разработки и сравнения алгоритмов обучения с подкреплением от OpenAI, язык Python, лицензия MIT.

MMLF (Maja Machine Learning Framework) — библиотека алгоритмов обучения с подкреплением и набор тестовых сред для их проверки, язык Python, лицензия GPL.

PyBrain — библиотека алгоритмов машинного обучения, язык Python, лицензия BSD.

RLPy — библиотека для проведения экспериментов по обучению с подкреплением, язык Python, 3-х пунктовая лицензия BSD.

Teachingbox — инструментарий для разработки алгоритмов обучения с подкреплением, язык Java, лицензия GPL.

BURLAP (Brown-UMBC Reinforcement Learning and Planning) — библиотека одно- и многоагентных алгоритмов планирования и обучения с подкреплением, язык Java, лицензия LGPL.

В нашей реализации обучения с подкреплением используется библиотека Gym от OpenAI.

## Начало работы с OpenAI Gym

**Gym** — это набор инструментов для разработки и сравнения алгоритмов обучения с подкреплением. Он не делает никаких предположений о структуре вашего агента и совместим с любой библиотекой численных вычислений, такой как TensorFlow или Theano.

Библиотека **Gym** — это набор тестовых задач-сред, которые вы можете использовать для отработки алгоритмов обучения с подкреплением. Эти среды имеют общий интерфейс, позволяющий писать общие алгоритмы.

# Обучение с подкреплением

Установка Gym

Для установки необходим Python версии 3.5+. Просто установите gym, используя pip:

pip install gym

## Альтернативный способ — сборка из исходного кода

При желании вы также можете напрямую клонировать Git-репозиторий Gym. Это особенно полезно, когда вы работаете над изменением самого Gym или добавлением окружения. Загрузите и установите библиотеку, используя код:

**git clone https://github.com/openai/gym**
**cd gym**
**pip install -e**

После этого можете запустить полную инсталляцию, чтобы установить все возможные среды из библиотеки:

pip install -e .[all]

при этом потребуется установить больше зависимостей, включая cmake и свежую версию установщика pip.

Как устанавливаются среды

Вот минимальный пример запуска чего-либо. Это запустит экземпляр среды CartPole-v0 для 1000 временных шагов, визуализируя среду на каждом шаге. Вы должны увидеть всплывающее окно с классической проблемой тележки:

**import gym**
**env = gym.make(‘CartPole-v0’)**

# Обучение с подкреплением

```
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
env.close()
```

Видео эпизода с пинг-понгом Вы можете увидеть в ноутбуке занятия.

Обычно мы завершаем симуляцию до того, как тележка выйдет за пределы экрана. Пока не обращайте внимания на предупреждение о вызове step (), даже если эта среда уже вернула done = True.

Если вы хотите увидеть в действии некоторые другие среды, попробуйте заменить CartPole-v0, приведенный выше, чем-то вроде MountainCar-v0, MsPacman-v0 (требуется зависимость от Atari) или Hopper-v1 (требуются зависимости MuJoCo). Все среды происходят от базового класса Env.

Обратите внимание: если у вас отсутствуют какие-либо зависимости, вы должны получить полезное сообщение об ошибке, сообщающее, что вы упустили. Установить отсутствующую зависимость, как правило, довольно просто. Вам также понадобится лицензия MuJoCo для Hopper-v1.

## Наблюдения

Если мы когда-нибудь захотим добиться большего, чем случайные действия на каждом этапе, вероятно, было бы хорошо знать, что наши действия делают с окружающей средой.

Функция step среды возвращает именно то, что нам нужно. Фактически, step возвращает четыре значения. Это

***observation (object)***: зависящий от среды объект, представляющий ваше наблюдение за окружающей средой. Например, пиксельные данные с камеры, совместные углы и совместные скорости робота или состояние доски в настольной игре.

# Обучение с подкреплением

***reward (float)***: сумма награды, полученная за предыдущее действие. Масштаб варьируется в зависимости от среды, но цель всегда — увеличить вашу общую награду.

***done (boolean)***: пришло ли время снова сбросить среду. Большинство (но не все) задач разделены на четко определенные эпизоды, и значение «Выполнено» означает, что эпизод завершился. (Например, возможно, полюс слишком сильно наклонился, или вы потеряли свою последнюю жизнь.)

***info (dict)***: диагностическая информация, полезная для отладки. Иногда он может быть полезен для обучения (например, он может содержать необработанные вероятности последнего изменения состояния среды). Однако официальные оценки вашего агента не позволяют использовать это для обучения.

Это просто реализация классического цикла «агент-среда». На каждом временном шаге агент выбирает действие, а среда возвращает наблюдение и награду.

Процесс стартует с вызова reset(), возвращающего первоначальное наблюдение (observation). Таким образом, правильнее написать предыдущий код, используя флаг done:

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```

# Обучение с подкреплением

Это должно привести к воспроизведению видео картинки и выводу на печать подобных данных. Вы увидите момент вызова reset:

[-0.061586  -0.75893141  0.05793238  1.15547541]
[-0.07676463 -0.95475889  0.08104189  1.46574644]
[-0.0958598  -1.15077434  0.11035682  1.78260485]
[-0.11887529 -0.95705275  0.14600892  1.5261692 ]
[-0.13801635 -0.7639636   0.1765323   1.28239155]
[-0.15329562 -0.57147373  0.20218013  1.04977545]
Эпизод завершается после 14 отсчетов времени:
[-0.02786724  0.00361763 -0.03938967 -0.01611184]
[-0.02779488 -0.19091794 -0.03971191  0.26388759]
[-0.03161324  0.00474768 -0.03443415 -0.04105167]

## Spaces (пространства)

В приведенных выше примерах мы отбирали случайные действия из пространства действий среды. Но что это за действия на самом деле? Каждая среда поставляется с action_space и observation_space. Эти атрибуты относятся к типу Space, и они описывают формат допустимых действий и наблюдений:

```
import gym
env = gym.make('CartPole-v0')
print(env.action_space)
#> Discrete(2)
print(env.observation_space)
#> Box(4,)
```

Дискретное пространство допускает фиксированный диапазон неотрицательных чисел, поэтому в этом случае

# Обучение с подкреплением

допустимыми действиями являются либо 0, либо 1. Пространство прямоугольника представляет собой n-мерный прямоугольник, поэтому допустимые наблюдения будут массивом из 4 чисел. Мы также можем проверить границы окна:

```
print(env.observation_space.high)
#> array([ 2.4       ,         inf,  0.20943951,         inf])
print(env.observation_space.low)
#> array([-2.4       ,        -inf, -0.20943951,        -inf])
```

Этот самоанализ может быть полезен при написании универсального кода, который работает во многих различных средах. Box и Discrete — самые распространенные пространства. Вы можете взять образец из пространства или проверить, что ему принадлежит:

```
from gym import spaces
space = spaces.Discrete(8) # Set with 8 elements {0, 1, 2, ..., 7}
x = space.sample()
assert space.contains(x)
assert space.n == 8
```

Для CartPole-v0 одно из действий применяет силу влево, а другое применяет силу вправо. (Вы можете понять, какое куда?)

К счастью, чем лучше ваш алгоритм обучения, тем меньше вам придется пытаться интерпретировать эти числа самостоятельно.

# Обучение с подкреплением

## Доступные среды

В Gym есть множество различных сред, от простых до сложных и включающих множество различных типов данных. Просмотрите полный список сред (full list of environments), чтобы увидеть их с высоты птичьего полета.

Классический контроль и игрушечный текст (Classic control, toy text): выполняйте небольшие задания, в основном из литературы по RL. Они здесь, чтобы Вам было проще начать.

Алгоритмический (Algorithmic): выполняйте вычисления, такие как добавление многозначных чисел и изменение последовательностей. Кто-то может возразить, что эти задачи легки для компьютера. Задача состоит в том, чтобы изучить эти алгоритмы исключительно на примерах. У этих задач есть приятное свойство, заключающееся в том, что их легко варьировать, варьируя длину последовательности.

Atari: играйте в классические игры Atari. Была интегрирована среда обучения Arcade (которая оказала большое влияние на исследования в области обучения с подкреплением) в простой в установке форме.

2D и 3D роботы (2D and 3D robots): управляйте роботом в симуляции. В этих задачах используется физический движок MuJoCo, который был разработан для быстрого и точного моделирования роботов. Включены некоторые среды из недавнего теста, проведенного исследователями Калифорнийского университета в Беркли (которые, кстати, присоединятся к нам этим летом). MuJoCo является проприетарным программным обеспечением, но предлагает бесплатные пробные лицензии.

Реестр

Основная цель gym — предоставить большой набор сред, которые открывают общий интерфейс и имеют версии для сравнения. Чтобы перечислить среды, доступные в вашей установке, просто спросите gym.envs.registry:

# Обучение с подкреплением

```
from gym import envs
print(envs.registry.all())
#> [EnvSpec(DoubleDunk-v0), EnvSpec(InvertedDoublePend
ulum-v0), EnvSpec(BeamRider-v0), EnvSpec(Phoenix-ram-v0),
EnvSpec(Asterix-v0), EnvSpec(TimePilot-v0), EnvSpec(Alien-v0),
EnvSpec(Robotank-ram-v0), EnvSpec(CartPole-v0),
EnvSpec(Berzerk-v0), EnvSpec(Berzerk-ram-v0),
EnvSpec(Gopher-ram-v0), ...
```

Это даст вам список объектов EnvSpec. Они определяют параметры для конкретной задачи, включая количество запускаемых испытаний и максимальное количество шагов. Например, EnvSpec (Hopper-v1) определяет среду, цель которой — заставить двухмерного смоделированного робота прыгать; EnvSpec (Go9x9-v0) определяет игру го на доске 9x9.

Эти идентификаторы среды обрабатываются как непрозрачные строки. Чтобы гарантировать корректные сравнения в будущем, среды никогда не будут изменены таким образом, чтобы это влияло на производительность, а только заменены более новыми версиями. В настоящее время мы добавляем к каждой среде суффикс v0, так что будущие замены естественно называть v1, v2 и т. д.

Добавить собственные среды в реестр и, таким образом, сделать их доступными для gym.make () очень легко: просто зарегистрируйте () их во время загрузки.

## Предыстория. Почему gym (тренажерный зал)? (2016)

Исследования RL также тормозятся двумя факторами:

1. Потребность в лучших тестах. В контролируемом обучении прогресс был обеспечен большими помеченными наборами данных, такими как ImageNet. В RL ближайшим эквивалентом будет большой и разнообразный набор сред. Однако

# Обучение с подкреплением

существующие коллекции сред RL с открытым исходным кодом недостаточно разнообразны, и их часто даже сложно настроить и использовать.

2. Отсутствие стандартизации сред, используемых в публикациях. Незначительные различия в определении проблемы, такие как функция вознаграждения или набор действий, могут кардинально изменить сложность задачи. Эта проблема затрудняет воспроизведение опубликованных исследований и сравнение результатов из разных статей.

Gym — это попытка решить обе проблемы.