# Assignment 2

Due April 10

## 1 Теоретическая часть: аппроксимация стратегии

Данное теоретическое задание посвящено методам аппроксимации стратегии.

**Задание 1** Воспользуйтесь своими знаниями о клеточном мире и его динамике, чтобы найти точное символьное выражение для оптимальной вероятности выбора действия *right* в примере 13.1 книги Саттона и Барто.

## 2 Практическая часть: градиент стратегии

Цель задания — реализовать и поэкспериментировать с методом прямой оптимизации стратегии с помощью градиентного спуска и его вариациями.

Данное задание основано на втором задании курса по Deep RL Университета Беркли.

Код задания расположен по адресу

https://github.com/pkuderov/mipt-rl-hw-2022

Как и в первом задании, у вас есть возможность выполнять его как локально, так и в Google Colab. Подробности, в том числе по установке зависимостей, вы найдете в `README` ко второму заданию в репозитории.

### 2.1 Обзор реализации

#### 2.1.1 Файлы

Для реализации градиента стратегии мы частично воспользуемся кодом из первого практического задания. Однако, чтобы каждое практическое задание имело независимую реализацию, переиспользуемые файлы были скопированы в соответствующие директории второго задания. В этих файлах сохранены пропуски, которые вам нужно будет заполнить своей готовой реализацией из первого задания — такие пропуски помечены комментарием `# TODO: get this from hw1`.

После заполнения пропусков в компонентнах из первого практического задания, вы можете приступить к работе над кодом градиента стратегии, заполняя пропуски, помеченные `TODO`, в следующих файлах:

- `agents/pg_agent.py`,
- `policies/MLP_policy.py`

По аналогии с первым заданием скрипты запуска экспериментов [локально или в Google Colab] вы найдете в папке `hw2/scripts/`.

### 2.1.2 Цикл обучения

Основной цикл обучения реализован в `infrastructure/rl_trainer.py`, как и в первом практическом задании.

Алгоритм градиента стратегии использует 3 шага:

- Сбор траекторий агента в среде с использованием его текущей стратегии.
- Оценка отдач для траекторий и вычисление значений функции преимущества (advantage). Это реализовано в функции `train` в `pg_agent.py`.
- Шаг обучения параметров стратегии агента. Вычислительный граф (нейронная сеть), задающий стратегию, вычисление базового уровня (baseline), а также код их обновления вы найдете в `policies/MLP_policy.py`.

## 2.2 Реализация градиента стратегии

В этой части вам нужно будет реализовать в `pg_agent.py` два варианта оценки отдачи. Первый вариант ("Case 1" в функции `calculate_q_vals`) использует дисконтированную сумму вознаграждений за всю траекторию и относится к "vanilla" версии градиента стратегии:

$$r(\tau_i) = \sum_{t'=0}^{T-1} \gamma^{t'} r(s_{it'}, a_{it'}). \tag{1}$$

Вторая версия ("Case 2") использует "reward-to-go" формулировку:

$$r(\tau_i) = \sum_{t'=t}^{T-1} \gamma^{t'} r(s_{it'}, a_{it'}). \tag{2}$$

*Обратите внимание, что она отличается лишь начальной точкой при суммировании.*

Реализуйте также оставшиеся помеченные `TODO` секции, которые вам понадобятся для запуска небольших экспериментов в следующем пункте задания. А к остальным — связанным с подсчетом базового уровня — мы вернемся чуть позже.

## 2.3 Запуск простых экспериментов

В этом пункте задания от вас потребуется провести набор простых экспериментов с целью оценить, какие параметры и варианты реализации градиента стратегии влияют на его скорость и качество обучения.

**Эксперимент 1: CartPole.** Проведите серию экспериментов с методом градиента стратегии в дискретной среде *CartPole-v0*, используя команды для запуска из соответствующей секции `README` ко второму практическому заданию.

Постройте два графика:

1. Кривые обучения средней отдачи на каждой итерации для экспериментов с маленьким размером пакета (batch) — эксперименты с префиксом `q1_sb_`.
2. Кривые обучения средней отдачи на каждой итерации для экспериментов с большим размером пакета (batch) — эксперименты с префиксом `q1_lb_`.

Ответьте письменно в краткой форме на следующие вопросы:

- Какой из вариантов оценки отдачи имеет лучшие результаты без нормализации значения преимущества?
- Помогает ли нормализация значения преимущества?
- Как влияет размер пакета на качество обучения?

В пояснениях к данному пункту приведите точные команды, использованные вами для запуска экспериментов.

*Ожидается, что корректная реализация метода будет сходиться к максимальному значению результата (200).*

**Эксперимент 2: InvertedPendulum.** Проведите серию экспериментов с методом градиента стратегии в непрерывной среде *InvertedPendulum-v2*, используя команды для запуска из соответствующей секции `README` ко второму практическому заданию.

Вашей задачей будет подобрать наименьший размер пакета `b*` и наибольший коэффициент скорости обучения `r*`, при которых алгоритм успевает достичь максимального результата в 1000 за 100 итераций. Флуктуации около максимального результата — это нормально. Требуемая точность оценки значений `b*` и `r*` — один знак после запятой.

В отчет к этому пункту включите следующую информацию:

- График обучения для полученных значений `b*` и `r*`. По желанию, это может быть как результат на одной случайной инициализации (random seed), так и среднее по нескольким запускам.
- Точную команду, использованную вами для запуска эксперимента.

## 2.4 Нейросетевые реализации

В этом задании от вас потребуется реализовать базовый метод с помощью аппроксимации функции полезности состояния нейронной сетью. Для этого необходимо дозаполнить оставшиеся пропуски, помеченные `TODO`. В частности:

- Добавить обучение данной нейронной сети в функции `update` класса `MLPPolicyPG` к уже реализованному обучению стратегии.
- Добавить вычисление оценки преимущества в `pg_agent.py:estimate_advantage` путем вычитания из "reward-to-go" оценки отдачи обученной оценки полезности состояния: $\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{it'}, a_{it'})\right) - V_\phi^\pi(s_{it})$.

## 2.5 Сложные эксперименты

*Обратите внимание, что обучение в данной части потребует значительного вычислительного времени, поэтому, пожалуйста, не оттягивайте с началом! Также, для всех оставшихся экспериментов используйте "reward-to-go" версию оценки отдачи.*

**Эксперимент 3: LunarLander.** Используйте полученную реализацию градиента стратегии для обучения оптимальному управлению в среде `LunarLanderContinuous-v2`. В данной части задания мы протестируем вашу реализацию базового метода.

Запустите эксперимент (см. команду соответствующей третьему эксперименту секции `README`). Постройте график кривой обучения. *Ожидается, что средняя отдача к концу обучения окажется примерно на уровне 180.*

**Эксперимент 4: HalfCheetah.** Используйте полученную вами реализацию градиента стратегии для обучения оптимальному управлению в среде `HalfCheetah-v2` с уменьшенной длительностью эпизода (150 вместо 1000 по умолчанию). Такое уменьшение длительности эпизода выбрано с целью существенно сократить время обучения.

Проведите серию экспериментов поиска по размеру пакета $b \in [10000, 30000, 50000]$ и величине коэффициента скорости обучения $r \in [0.005, 0.01, 0.02]$ (см. команду в первой части секции 4-го эксперимента в `README`). Постройте единственный график, на котором будут отображены полученные кривые обучения. Опишите словами, как размер пакета и коэффициент скорости обучения влияют на результаты.

Получив наилучшие значения `b*` и `r*`, используйте их для проведения следующей серии экспериментов (см. список команд во второй части секции 4-го эксперимента `README`). Постройте единственный график с кривыми обучения на полученной серии запусков. *Ожидается, что вариант с "reward-to-go" и вариант с базовым методом оба получат результаты, близкие к 200.*

## 2.6 Реализация обобщенной оценки преимущества (GAE)

Данная часть задания посвящена реализации упрощенную версию алгоритма обобщенной оценки преимущества (GAE-$\lambda$).

Данный алгоритм по своей идее близок к алгоритму TD($\lambda$), только вместо оценки функции полезности состояния в нем оценивается функция преимущества:

$$A^{\pi}_{GAE}(s_t, a_t) = \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta^V_{t+k} = \sum_{k=0}^{\infty} (\gamma\lambda)^k [r_{t+k} + \gamma V(s_{t+k+1}) - V(s_{t+k})] \tag{3}$$

Параметр $\lambda$ используется в нем для подсчета взвешенной суммы $n$-шаговых Монте-Карло оценок преимущества (для разных $n$). Значению $\lambda = 0$ соответствует смещенная оценка с низкой дисперсией: $r_t + \gamma V(s_{t+1}) - V(s_t)$, по аналогии с TD(0), а значению $\lambda = 1$ — несмещенная Монте-Карло оценка с высокой дисперсией: $\sum_{k=0}^{\infty} \gamma^k r_{t+k} - V(s_t)$. Таким образом, подбирая данный параметр, можно найти баланс между смещением и дисперсией оценки преимущества. Детали данного метода приведены в оригинальной статье.

Заполните недостающую часть реализации в `pg_agent.py:estimate_advantage`.

**Experiment 5: Hopper**. Используйте полученную версию градиента стратегии с обобщенной оценкой преимущества для обучения оптимальному управлению агентом в среде `Hopper-v2`. Обучите агента с разными значениями параметра $\lambda \in [0, 0.95, 0.99, 1]$ (см. команду запуска в соответствующей секции `README`). *Обратите внимание, что при корректной реализации GAE-$\lambda$ случай $\lambda = 1$ эквивалентен случаю с "vanilla" оценкой отдачи с обученным базовым методом.*

Приведите единственный график кривых обучения для проведенных вами запусков. Опишите словами, как параметр $\lambda$ повлиял на качество обучения. *Ожидается, что наилучший результат обучения окажется в районе 400.*

### 2.7 Дополнительные эксперименты

В качестве бонусного задания выберите любой из пунктов (или оба):

- Получение опыта в среде зачастую является одним из важнейших вычислительно затратных мест. В `infrastructure/rl_trainer.py` траектории собираются последовательно в однопоточном режиме, однако этот процесс возможно распараллелить по нескольким потокам. Реализуйте многопоточную версию сбора опыта в среде и напишите краткий отчет с графиками о том, какой разницы в скорости вам удалось достичь.

- В методе градиента стратегии мы собираем один пакет данных для обучения, выполняем шаг градиентного спуска и затем отбрасываем эти данные, двигаясь дальше. Можем ли мы потенциально ускорить метод, делая несколько шагов градиентного спуска на одном пакете данных? Опробуйте этот вариант и напишите отчет с графиками о полученных результатах. Проведите сравнение на одной из использованных в задании MuJoCo сред.

Не забывайте включать команды запуска экспериментов в отчет, чтобы вашу работу можно было воспроизвести при проверке.

## 3 Формат отправки

*Формат сдачи совпадает с форматом сдачи первого практического задания.*

Сдача предполагается в виде предложений изменения кода (pull request) в репозитории по ссылке в начале практического задания.

Ожидается, что предложение будет содержать непосредственно код заполненных вами недостающих частей выданного решения и логи финальных запусков (для каждого задания и каждой из использованных сред).

Оригинально все логи лежат в папке `data`. Логи финальных запусков скопируйте из `data` в отдельную папку `run_logs` и отправьте вместе с вашим решением.

В сообщение к предложению необходимо добавить результаты/описание/решение по каждому из пунктов задания (в соответствии с тем, что оно требует). Разметка (markdown) позволяет и вставку картинок, и оформление табличек. Опционально, вы можете оформить результаты в виде отдельного файла .doc или .pdf и добавить их в посылку (commit), а в сообщении сослаться на этот файл. Не забудьте также добавить к каждому пункту задания код запуска, чтобы можно было воспроизвести ваши результаты.