Assignment 3

Due May 13

1 Теоретическая часть: среднее вознаграждение

Данное теоретическое задание посвящено таким понятиям как среднее вознаграждение и дифференциал полезности.

Задание 1. Упражнение 10.6 книги Саттона и Барто. Расмотрим марковский процесс (примечание: просто процесс, не МППР!), состоящий из трех состояний A, B, C; переход производятся детерминированно по циклу. Вознаграждение +1 начисляется по прибытии в состояние A, в остальных случаях оно равно 0. Каковы дифференциалы для полезностей во всех трех состояний?

Задание 2. Упражнение 10.8 из книги Саттона и Барто. Псевдокод на стр.296-297 (над упражнением 10.6 — Дифференциальный полуградиентный Sarsa для оценивания $\hat{q} = q_*$) обновляет $\hat{R}_{t+1}$, используя в качестве ошибки δ_t, а не просто $R_{t+1} - \hat{R}_{t+1}$. Годятся оба определения ошибки, но δ_t дает лучшие результаты. Чтобы понять, почему, рассмотрим круговой МППР с тремя состояниями из задания 1. Оценка среднего вознаграждения должна стремиться к своему истинному значению $\frac{1}{3}$. Предположим, что она уже принимала это значение и там и осталась. Какой тогда была бы последовательность $R_t - \hat{R}_t$? А какой была бы последовательность δ_t (вычисленная по формуле 10.10)? Какая последовательность давала бы более устойчивую оценку среднего вознаграждения, если бы оценке было разрешено изменяться в ответ на ошибки?

2 Практическая часть: Q-обучение и актор-критик

Цель задания — реализовать и поэкспериментировать с методами Q-обучения и актор-критик.

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

Код задания расположен по адресу

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

Как и в предыдущих заданиях, 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 Часть 1. Q-обучение

Часть 1 практического задания требует, чтобы вы реализовали и оценили качество работы Q-обучения для игры Atari. Так как алгоритм Q-обучения был рассмотрен на лекциях и семинарах, вам будет предоставлен стартовый код.

Это задание будет будет быстрее работать на графическом процессоре, хотя его можно выполнить и на процессоре. Тем не менее рекомендуется использовать Colab, если вы у вас нет доступного графического процессора.

2.1.1 Файлы

Код задания будет основываться на коде, который вы реализовали в первых двух заданиях. Все файлы, необходимые для запуска вашего кода, находятся в папке hw3, но будут некоторые пустые места, которые требуется заполнить своим решением из домашнего задания 1. Эти места отмечены `# TODO: get this from hw1 or hw2` и находятся в следующих файлах:

- `infrastructure/rl_trainer.py`,
- `infrastructure/utils.py`,
- `policies/MLP policy.py`

Ваша реализация Q-обучения будет расположена в следующих файлах:

- `agents/dqn_agent.py`,
- `critics/dqn_critic.py`,
- `policies/argmax policy.py`

2.1.2 Реализация Q-обучения

Первый этап задания заключается в реализации рабочей версии Q-learning. Код по умолчанию запустит игру *Ms. Pac-Man* с разумными настройками гиперпараметров. Найдите маркеры `# TODO` в файлах, перечисленных выше, для получения инструкций по реализации. Вы можете заглянуть внутрь `infrastructure/dqn_utils.py`, чтобы понять, как работает (оптимизированный для памяти) буфер воспроизведения. Вам не нужно будет его модифицировать.

Для ответа на некоторые вопросы в задании может потребоваться настройка гиперпараметров, архитектуры нейронной сети и даже смена игры. Это должно быть сделано путем изменения аргументов командной строки, переданных в `run_hw3_dqn.py`, или путем изменения параметров запука скрипта в Colab.

Чтобы определить, верна ли ваша реализация Q-learning, вы должны запустить ее с гиперпараметрами по умолчанию в игре *Ms. Pac-Man* на 1 миллион шагов, используя приведенную ниже команду. Эталонное решение достигает результата в 1500 за этот период. В Colab это займет примерно 3 часа GPU. Если вы видите, что обучение занимает намного больше времени, возможно, в вашей реализации есть ошибка.

Мы рекомендуем отладить ваше решение сначала в среде *LunarLander-v3*. Эталонное решение с гиперпараметрами по умолчанию достигает результата около 150 после 350 тыс шагов в среде. Однако нужно быть готовым к большой дисперсии результатов в этой среде, и без трюка с двойным Q-обучением средняя отдача часто уменьшается после достижения результата 150.

2.1.3 Тестирование Q-обучения

После того, как у вас есть работающая реализация Q-learning, выполните указанные ниже эксперименты и подготовьте отчет. Для каждого задания ниже ожидается получить от вас по одному графику.

Задание 1. Качество базовой версии Q-обучения (DQN)

Включите график кривой обучения вашей реализации на *Ms. Pac-Man*. Ось X должна соответствовать количеству временных шагов, а ось Y должна показывать среднее вознаграждение за эпоху, а также лучшее среднее вознаграждение на данный момент. Расчет данных величин уже есть в начальном коде. Вы можете визуализировать их с помощью Tensorboard, как и в предыдущих заданиях. Вам не нужно изменять гиперпараметры по умолчанию, чтобы получить хорошую производительность. Если же вы изменяете какие-либо параметры, перечислите их в подписи к графику.

Для запуска эксперимента отталкивайтесь строки, указанной на странице README к третьему домашнему заданию (секция *Experiment 1. Base Q-learning*)

Задание 2. Качество двойного Q-обучения (DDQN)

Используйте двойную оценку, чтобы повысить точность обучаемых Q-значений. Это означает использование онлайновой Q-сети [вместо целевой] для выбора **наилучшего действия** при вычислении целевых значений. Сравните производительность DDQN с базовым DQN. Проведите как минимум три запуска с различными инициализациями генератора случайных чисел как для DQN, так и для DDQN. Вы можете использовать *LunarLander-v3* в этом задании. Для запуска эксперимента используйте строки запуска, указанные на странице README к третьему домашнему заданию (секция *Experiment 2. DQN vs DDQN*).

Приложите к сдаче логи данных запусков. В отчете приведите график со сравнением средних результатов алгоритмов.

Задание 3. Эксперименты с гиперпараметрами

Теперь проанализируем чувствительность Q-обучения к гиперпараметрам. Произвольно выберите один гиперпараметр и запустите алгоритм с не менее чем тремя различными его значениями в дополнение к тому значению, который был использован ранее в задании 1. Приведите график с кривыми обучения для все четырех значений. В подписи поясните ваш выбор и опишите влияние этого гиперпараметра на производительность. Для запуска эксперимента используйте строки запуска, указанные в README к третьему домашнему заданию (секция *Experiment 3. DQN hyperparameters*). Вы можете взять в качестве тестовой среды что-то отличное от Lunar Lander.

Примеры гиперпараметров для исследования: скорость обучения; архитектура нейронной сети, например, количество слоев, размер скрытого слоя и т. д.; расписание исследования или стратегия исследования (например, вы можете реализовать альтернативу эпсилон-жадности и установить разные значения гиперпараметров) и т. д. Желательно найти и выбрать такой параметр, который существенно повлияет на производительность.

2.2 Часть 2. Актор-критик

Во второй часть практического задания потребуется изменить реализованный в прошлом практическом задании алгоритм градиента стратегии до метода актор-критик. Обратите внимание, что обучение критика может занять больше времени, чем обучение

актора.

Вспомним, как мы считали градиент стратегии в прошлом задании:

$$\nabla_\Theta J(\Theta) \approx \mathbf{E}\left[\sum_{t=1}^{T} \nabla_\Theta \log \pi_\Theta(a_t|s_t) A^\pi(s_t, a_t)\right], \tag{1}$$

$$A^\pi(s_t, a_t) \approx \left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\right) - V^\pi_\Phi(s_t) \tag{2}$$

где мы использовали "rewards-to-go" оценку Q-функции и дополнительно вычитали полезность состояния в качестве базового уровня, тем самым получая оценку функции преимущества $A^\pi(s_t, a_t)$.

На практике такая оценка значения преимущества отличается высокой дисперсией. В метод актор-критик решается проблема высокой дисперсии путем использования сети критика для оценки полезности. Существуют разные варианты, какую функцию полезности оценивать критиком. Один из них — функцию полезности состояния. Тогда оценка преимущества может быть задана следующим образом:

$$A^\pi(s_t, a_t) \approx r(s_t, a_t) + \gamma V^\pi_\Phi(s_{t+1}) - V^\pi_\Phi(s_t) \tag{3}$$

В данном задании в качестве основы для сети критика мы будем использовать ту же сеть оценки полезности, что и во втором домашнем задании. Обучать же критика предлагается через минимизацию среднеквадратичной TD-ошибки по стандартному TD-правилу:

$$y_t = r(s_t, a_t) + \gamma V^\pi_\Phi(s_{t+1}), \tag{4}$$

$$\min_\Phi \mathrm{MSE}(V^\pi_\Phi(s_t) - y_t) \tag{5}$$

В теории, шаг минимизации необходимо выполнять каждый раз, когда обновляется стратегия агента, чтобы стратегия и функция полезности, которая ее оценивает, были синхронизированы. Однако на практике эта операция обычно чересчур дорогостоящая. Компромиссным решением будет выполнять лишь несколько шагов градиентного спуска для обновления критика на каждой итерации. Обратите внимание, что это не равносильно кратному увеличению гиперпараметра скорости обучения, так как на каждом шаге градиентного спуска (или раз в несколько шагов, как в текущем задании) мы будем заново пересчитывать целевые значения. В целом, процесс обучения критика представляет собой итеративный процесс, в котором мы постоянно обновляем его оценку полезности, чтобы она соответствовала целевым значениям для меняющейся стратегии актора.

2.2.1 Реализация метода актор-критик

Ваш код будет основан на вашем решении из второго домашнего задания. Вам потребуется заполнить пропуски, помеченные `TODO` для следующих частей кода:

- В `policies/MLP_policy.py` реализуйте функцию `update` для класса `MLPPolicyAC`. Следует отметить, что класс стратегии актор-критика фактически совпадает с

классом стратегии из домашнего задания по градиенту стратегии (за исключением отсутствия базового уровня).

- В `agents/ac_agent.py` дополните функцию `train`. Она должна содержать реализацию всех требуемых шагов по обучению критика, оценке преимущества, а затем обновлению стратегии актора. Залогируйте функцию потерь в конце, чтобы отслеживать ее значения во время обучения.

- В `agents/ac_agent.py` дополните функцию `estimate_advantage`. Эта функция использует сеть критика для оценки значения преимущества. Не забудьте корректно обработать окончания эпизода.

- В `critics/bootstrapped_continuous_critic.py` заполните пропуски в функции `update`. Обратите внимание на переменные `num_grad_steps_per_target_update` и `num_target_updates`, которые задают число шагов градиентного спуска и требуемую частоту обновления целевых значений в процессе.

2.2.2 Тестирование метода актор-критик

После того, как у вас есть работающая реализация актор-критика, выполните указанные ниже эксперименты и подготовьте отчет.

Задание 4. Проверка правильности реализации на CartPole

Цель этого задания убедиться в правильности вашей реализации на примере простой среды *CartPole-v0*. Для запуска эксперимента используйте строки запуска, указанные на странице README к третьему домашнему заданию (секция *Experiment 4. Sanity check with CartPole*). Вам потребуется сравнить качество работы агента с разными частотами обучения критика и обновления целевых функций. Агент с наилучшим набором гиперпараметров должен полностью решать данную среду, набирая результат 200. Подготовьте отчет со сравнением сделанных запусков в виде графика кривых обучения и кратким текстовым пояснением.

Задание 5. Проверка метода актор-критик на более сложных средах

Используя наилучшие значения гиперпараметров из предыдущего задания, протестируйте качество работы вашей реализации в средах *InvertedPendulum* и *HalfCheetah*. Для запуска эксперимента используйте строки запуска, указанные в README к третьему домашнему заданию (секция *Experiment 5. Run actor-critic with more difficult tasks*).

Ваши результаты должны примерно совпадать с результатами градиента стратегии из второго домашнего задания. После 150 итераций в *HalfCheetah* результат должен быть около 150. После 100 итераций в *InvertedPendulum* результат должен быть около 1000. В начале обучения отдача должна начать расти немедленно. Например, после 20 итераций отдача в HalfCheetah должна быть выше -40, а ваш доход InvertedPendulum должен быть около или выше 100. Однако, конечно, нужно учитывать, что результаты между запусками могут сильно варьироваться.

Результатом работы в этом разделе являются графики результатов для каждой из сред.

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.