Курс «Машинне навчання»

Домашнє завдання 4: «Навчання з підкріпленням»

Як здавати роботу

Питання домашньої роботи вимагають певного обмірковування, але не вимагають довгих відповідей. Будь ласка, будьте якомога більш стислі.

1. Якщо ви маєте будь-які питання щодо цієї домашньої роботи, задавайте їх у Slack-чаті курсу та відповідайте на запитання інших.
2. Ви можете обговорювати домашні завдання в групах, але не показуйте іншим свої рішення і не користуйтесь готовими чужими.
3. Для теоретичних задач, можна надсилати або скановані рукописні відповіді, або підготувати електронні версії в Word чи LaTeX. Зберігайте ці звіти в форматі PDF.
4. Для задач, які вимагають написання програм, надсилайте ваш код (з коментарями) та графіки, якщо їх потрібно намалювати відповідно до умов задачі. Якщо ви виконуєте завдання в IPython Notebook, достатньо надіслати .ipynb-файл (переконайтеся, що він виконується без помилок).
5. Вкажіть ваше ім’я та прізвище у звіті.
6. Потрібно здати: PDF-звіт із теоретичними завданнями (якщо такі є), код програмних завдань (якщо такі є).

Технічні примітки

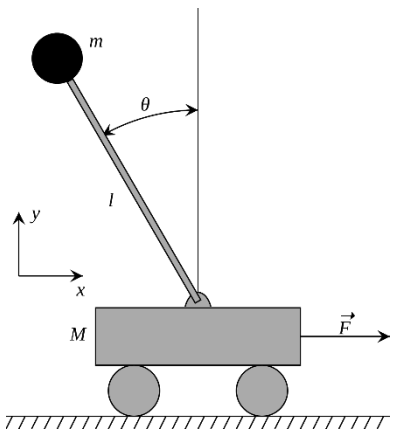
1. Для завдань з програмування використовуйте Python 3.5. Можете користуватися або офіційним дистрибутивом, або дистрибутивом Anaconda, що вже містить більшість заздалегідь встановлених пакетів.
2. Встановіть середовище OpenAI Gym та супровідні бібліотеки:
   - **Linux** (Ubuntu-based):
     ```
     $ apt install ffmpeg libav-tools
     $ pip3 install -r requirements.txt
     ```
   - **macOS** (використовуючи Homebrew):
     ```
     $ brew install ffmpeg libav
     $ pip3 install -r requirements.txt
     ```
   - **Windows** (використовуючи Chocolatey):
     ```
     > choco install ffmpeg
     > pip install -r requirements.txt
     ```
3. Запуск:
   ```
   $ python3 cartpole.py
   ```

1. Балансування оберненого маятника.

[25 балів]

У цьому завданні ви застосуєте Reinforcement Learning для навчання інтелектуального агента, що здатен керувати оберненим маятником. В сучасному житті вирішення цієї задачі дало можливість побудувати двоколісний самокат Segway та ракети, що здійснюють посадку самостійно — SpaceX Grasshopper.

Модель задачі оберненого маятника доволі проста. Візок рухається вздовж горизонтального столу — ліворуч або праворуч. До візка на шарнірі прикріплена жердина, яка може відхилятися на довільний кут. Проте, якщо жердина відхилиться надто далеко від центру, вона впаде. Ми можемо уникнути цього, штовхаючи візок у потрібному напрямку. Однак, візок також не може від'їжджати надто далеко від центру стола, оскільки упаде з нього.



В будь-який дискретний момент часу (0, 1, 2, ...) агенту доступні такі дані про стан:

1. x — положення візка на горизонтальній осі.
2. v — горизонтальна лінійна швидкість візка.
3. θ — кут нахилу жердини.
4. ω — кутова швидкість жердини.

Експеримент (епізод) вважається неуспішним і завершується передчасно за хоча б однієї з перелічених умов (див. документацію CartPole-v0):

1. x виходить за межі $[-2.4,\ 2.4]$.
2. θ виходить за межі $[-20.9°,\ 20.9°]$.

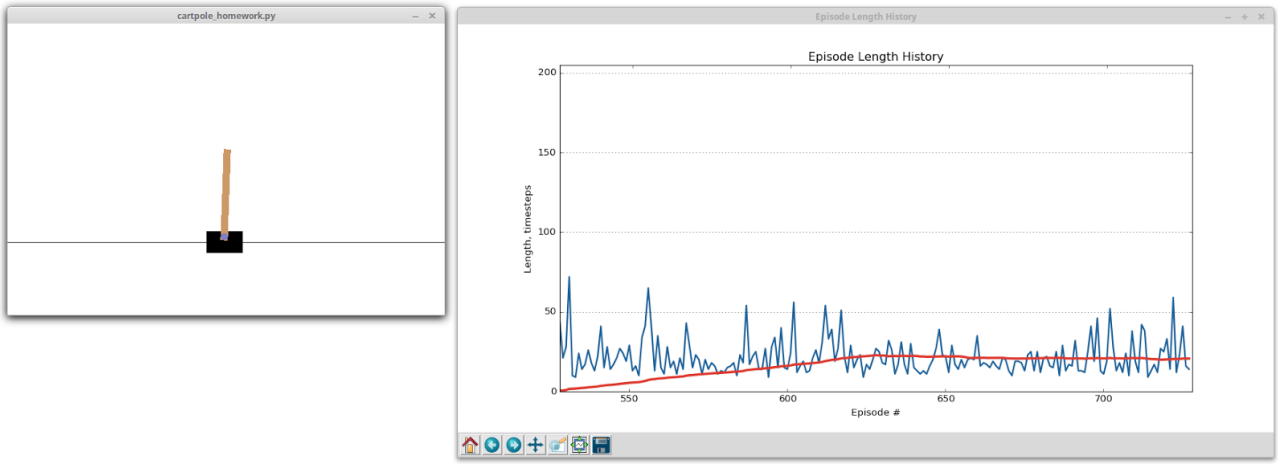
Епізод вважається **успішним** і завершується, якщо він протривав 200 моментів часу (timesteps).

Задача оберненого маятника вважається **розв'язаною**, якщо агент зміг протримати жердину в середньому хоча б 195 моментів часу впродовж останніх 100 епізодів.

Ваше завдання — натренувати агента, що здатен буде керувати візком, не знаючи нічого про внутрішню фізичну модель цієї задачі, а лише спостерігаючи стан: (x, v, θ, ω). В якості основного алгоритму навчання ви використаєте табличну форму Q-Learning, а для симуляції середовища будемо використовувати відкриту платформу OpenAI Gym.

В термінах Reinforcement Learning, на кожному кроці агент отримує винагороду $+1$ і може обрати одну з двох дій: штовхнути візок ліворуч або праворуч (дії «не робити нічого» немає). Якщо дія призвела до падіння візка чи жердини, агент отримує негативну винагороду -200.

Файл `cartpole.py` уже містить реалізацію основного процесу роботи та моніторинг якості навчання. Проте, за замовчуванням агент робить хаотичні випадкові дії та не навчається:



Вам потрібно реалізувати логіку навчання. Відповідні місця позначені в коді, з інструкціями, що потрібно зробити. Всього таких місць у коді є 6:

a. **[5 балів]** В конструкторі класу `CartPoleQLearningAgent` створіть порожню Q-таблицю $Q(s, a)$. Для цього вам потрібно придумати спосіб, як представити стан (x, v, θ, ω) у вигляді єдиного числа. Множина усіх можливих таких числових станів становитиме рядки Q-таблиці.

b. **[4 балів]** Щоразу, коли агент спостерігає новий стан, нам потрібно дізнатися, який рядок Q-таблиці відповідає цьому стану. Вам потрібно реалізувати в методі `_build_state` перетворення 4-вимірного спостереження в ціле число.

c. **[1 балів]** На початку кожного епізоду агент опиняється в довільному стані. Вам потрібно обрати в методі `begin_episode` найкращу можливу першу дію, користуючись Q-таблицею.

d. **[2 балів]** Оскільки ми не маємо відразу повного розуміння даного MDP, ми мусимо час від часу робити випадкові дії, щоб краще дослідити невідомий світ. Вам потрібно в методі `act` реалізувати цю логіку, яка відповідно до коефіцієнтів обирає або випадкову дію (exploration), або найкращу (exploitation). З часом цей коефіцієнт дослідження затухатиме (див. метод `begin_episode`).

e. **[9 балів]** Коли агент переходить зі стану в стан, він отримує винагороду та обирає наступну дію. В алгоритмі Q-Learning в цей момент відбувається оновлення Q-таблиці, внаслідок чого агент навчається. Вам потрібно в методі `act` реалізувати правило оновлення Q-таблиці.

f. **[4 балів]** Тепер, коли ми реалізували логіку агента, ми створюємо його екземпляр в основній програмі. Вам потрібно дізнатися, як впливають на Q-Learning екстремально великі чи екстремально малі параметри, і, відповідно, підібрати оптимальні для даної задачі.

2. Посадка ракети багаторазового використання

[40 балів + 15 додаткових балів]

Багаторазове використання першого ступеня космічної ракети-носія – шлях до суттєвого здешевлення виводу вантажів на орбіту. На сьогодні таку технологію мають SpaceX і Blue Origin. Дві інші приватні космічні компанії знаходяться в процесі її розробки.

У цій задачі ви використаєте reinforcement learning для того, щоб навчити контролер садити ракету-носій на морську платформу з використанням симулятора за авторством Ігоря Костюка. На відміну від інших популярних симуляторів для цієї задачі, цей підтримує вітер і коливання платформи на хвилях (хоча і має, на відміну від них, дуже просту графіку).

Встановіть симулятор, слідуючи інструкції. Зверніть увагу, що це посилання веде на форк оригінального репозиторію Ігоря – важливо використовувати саме цей форк, оскільки його код був змінений для роботи з Python 3.6.

Якщо ви використовуєте Windows, вам буде потрібно здійснити додаткові активності.

- **Встановлення pygame для Windows**

  Завантажте wheel потрібної вам версії за цим посиланням. Зверніть увагу: навіть якщо у вас не встановлюється 64-бітна версія бібліотеки, вам потрібно завантажти і встановити 32-бітну, незалежно від версії вашої операційної системи. Встановіть її за допомогою pip. Наприклад:

  ```
  >pip3 install pygame-1.9.3-cp36-cp36m-win32.whl
  ```

- **Встановлення Box2D для Windows**

  Завантажте wheel потрібної вам версії за цим посиланням. Зверніть увагу: навіть якщо у вас не встановлюється 64-бітна версія бібліотеки, вам потрібно завантажити і встановити 32-бітну, незалежно від версії вашої операційної системи. Встановіть її за допомогою pip. Наприклад:

  ```
  >pip3 install Box2D-2.3.2-cp36-cp36m-win32.whl
  ```

a. **[10 балів]** В поточній версії симулятора існує лише два стани, коли контроллер отримує винагороду: у випадку, коли ракета успішно приземлилася (винагорода: +100), і коли ракета розбилася (винагорода: -100). Для застосування Q-learning цього буде замало: для оновлення функції Q позитивним значенням контроллер мусить хоча би один раз успішно посадити ракету. Імовірність, що це вдасться зробити випадково, надзвичайно мала (ви можете переконатися, спробувавши посадити ракету вручну). Тому важливо мати функцію винагороди для будь-якого стану, так, щоб вона слугувала евристикою того, наскільки поточний стан ракети є близьким до бажаного.

Подумайте, якою може бути ця евристика, імплементуйте її в функції `getReward()` в файлі `F9utils.py` і опишіть, чому ця евристика має сенс, на вашу думку, у письмовому звіті. Зверніть увагу, що хибна евристика може сильно сповільнити навчання. Наприклад, в оригінальній версії симулятора евристика залежить від відстані до платформи. Це призводить до того, що контролер доволі швидко вивчає, що розбитися пізніше краще, ніж розбитися раніше (за кожен крок, коли ракета знаходиться в польоті, нараховується нагорода) і напрацьовує стратегію: «висіти» в повітрі, поки не закінчиться пальне. Реалізувати цю стратегію набагато простіше, ніж вдало посадити ракету, і вивести політику з цього «плато» дуже важко.

b. **[30 балів + 15 додаткових балів]** В поточній версії файлу `F9LanderClientCORE.py` реалізований контролер, який намагається тримати ракету вертикально. Ви можете запустити його з візуалізацією і подивитися, як він працює.

   Ваша задача – реалізувати контроллер за допомогою Q-learning з апроксимацією функції, який буде садити ракету на платформу.

   Задача буде оцінюватися наступним способом.

   - Менше 5 успішних посадок зі 100 симуляцій: 0 балів.
   - 5-10 успішних посадок зі 100 симуляцій: 20 балів.
   - Більше 10 успішних посадок зі 100 симуляцій: 30 балів.

   Студентам, чиїм контролерам вдасться посадити ракету найбільшу кількість разів (але більше 10) зі 100 симуляцій (окремо для бакалаврів і магістрів) буде присвоєно додаткові бали за таким принципом:

   - 1 місце: 15 додаткових балів;
   - 2 місце: 10 додаткових балів;
   - 3 місце: 5 додаткових балів.

3. Управління віртуальною інфраструктурою в хмарі

[50 балів]

Популярність хмарної інфраструктури зростає дуже високими темпами. Amazon AWS звітував $17.459 млрд. виторгу в 2017 році, що на 43% більше за виторг цих сервісів в 2016. За останній квартал 2017 року чистий прибуток від хмарних сервісів Amazon складав 73% чистого прибутку

компанії загалом. Станом на 31 березня 2018 року Amazon є четвертою за капіталізацією компанією світу з оціночною вартістю $700 млрд.

У цій задачі ви використаєте reinforcement learning для управління хмарною інфраструктурою, яку продає Amazon.

В часи до хмарних технологій компанії (особливо маленькі та середні) потребували власної інфраструктури: кожна з них мала якусь кількість фізичних серверів, на яких працювали інформаційні системи компанії (бухгалтерська звітність, поштовий сервер, веб-сайт). Ці сервери вимагали обслуговування, яке було однією зі статей витрат таких компаній. Більші компанії мали окрему посаду системного адміністратора, в обов'язки якого входило в тому числі обслуговування серверів. Менші часто укладали контракти з компаніями, які спеціалізувалися на таких послугах (аутсорсили адміністрування інфраструктури). Веб-розробники мали наступний вибір.

- Використовувати хостинги з визначеним наперед набором технологій. Скажімо, ви могли купити хостинг PHP + MySQL, але замінити в ньому MySQL на PostgreSQL вже не могли.
- Купувати сервер в colocation. Colocation - послуга дата-центрів, в яких можна було купити окремий сервер або навіть просто місце для встановлення своєї машини. Тоді такий веб-девелопер мав можливість повністю контролювати, що встановлено на цій машині, але не перейматися охолодженням, живленням та мережевим з'єднанням. При цьому, якщо машина виходила з ладу фізично, це вже, як правило, ставало проблемою веб-розробника, а не дата-центру.
- Купувати фізичний сервер і ставити його у власній серверній.

Одною з перших пропозицій Amazon AWS була купівля віртуальних машин. Ви маєте віддалений доступ до них, не переймаючись ні живленням, ні зв'язком, ні охолодженням. Ви можете отримати нову віртуальну машину потрібної вам конфігурації протягом дуже короткого часу (від декількох секунд). І ви платите тільки за час, коли вона працює.

В цій задачі ви працюватимете з двома видами віртуальних машин від Amazon: EC2 і Spot.

EC2 працює, як звичайна віртуальна машина. Ви платите фіксовану ціну за час її роботи. Spot – віртуальні машини набагато дешевші від EC2, за які ви торгуєтеся з іншими користувачами.

Spot – метод, яким Amazon продає незатребувані ресурси. Скажімо, якщо в дата-центрі є вільні потужності і ніхто не купує їх в цей момент за ціною EC2, Amazon віддає їх тому, хто дасть найбільшу ціну (згадайте баланс попиту і пропозиції з економіки). Але оскільки кількість Spot є фіксованою (вона визначається кількістю вільних ресурсів), якщо хтось дасть за вашу Spot віртуальну машину більшу вартість, ніж платите ви, Amazon миттєво забере її у вас і передасть користувачу, який платить найбільше.

Звісно, попит на віртуальні машини в різний час різний. Скажімо, ми можемо уявити ситуацію, коли вночі їх потрібно набагато менше, ніж вдень. Відповідно, імовірність, що у вас заберуть ваш Spot вночі набагато менша, ніж вдень.

Іншими словами, Spot – дуже дешеві віртуальні машини, які можуть пропасти в будь-який момент з імовірністю, обернено пропорційною ціні, яку ви згодні за них платити.

Уявіть собі, що ви керуєте інфраструктурою великої інтернет-компанії. Ви використовуєте ваші віртуальні машини, щоб опрацьовувати запити від ваших користувачів. Скажімо, кожен пошуковий запит в Google має бути направлений до конкретного серверу чи віртуальної машини, опрацьований і повернутий користувачу з результатами пошуку. Аналогічно – кожен запит на купівлю на Amazon. Для цього вам потрібно, щоб ваші у вас були віртуальні машини, готові опрацьовувати такі запити.

Ви платите за час роботи цих машин. Таким чином, якщо вони простоюють, і кількість доступних потужностей набагато більша від кількості запитів від користувачів, вам є сенс зупинити якусь кількість ваших віртуальних машин заради економії. Якщо ж ви бачите, що кількість запитів дуже висока, і машини майже повністю загружені, варто запустити кілька нових, щоб не вийшло так, що нові запити від користувачів не буде кому опрацьовувати.

Ви можете купувати як EC2, так і Spot машини, але важливо звернути увагу на специфіку роботи всього цього середовища.

- Від моменту, коли ви запускаєте віртуальну машину, до моменту, коли вона готова опрацьовувати запити, проходить якийсь час. Він потрібен для запуску самої машини, запуску операційної системи та запуску вашого власного коду. Через природу технологій віртуалізації цей час буде щоразу трошки відрізнятися.
- Віртуальна машина може зупинитися через помилку (crash). Будь-яка машина в будь-який момент часу має якусь імовірність зламатися і зупинитися.

- Spot машини дешеві, але окрім імовірності зупинитися через помилку, є також імовірність, що Spot машину у вас заберуть, і в кожен момент часу ця імовірність різна.

Вам потрібно написати контролер, який буде керувати такою інфраструктурою. В 2016 році Google впровадив подібну в своїх дата-центрах, що допомогло компанії зекономити 40% вартості їх охолодження.

Завантажте симулятор для цього завдання за цим посиланням.

Зверніть увагу: ви – перші студенти, які працюють на цьому симуляторі. Попри те, що команда інструкторів робила все, щоб гарантувати його стабільність, під час роботи з ним можливі помилки, зокрема тому, що він дає вам дуже велику свободу вибору reinforcement learning підходів. У разі виникнення будь-яких помилок чи питань по роботі з ним, будь ласка, задавайте їх на Piazza. Успіх виконання цієї задачі напряму залежить від комунікації з інструкторами.

Архітектурно, система складається з контролера і симулятора. Контролер реалізує алгоритм управління інфраструктурою і взаємодіє з нею через інтерфейс. Симулятор емулює цей інтерфейс, під час взаємодії з ним поведінка AWS симулюється.

Загалом, система складається з таких модулей.

- **`simulator/interfaceSimulator.py`** реалізує симулятор хмарної інфраструктури. Він підтримує два режими: ручний і автоматичний. В автоматичному режимі симулятор симулює поведінку системи, закладену в файл з історичними даними (в вашому випадку – `data/full_balancer_model_normal.csv`). Ви будете використовувати цю версію симулятора для навчання моделі. В ручному режимі ви самі регулюєте параметри інфраструктури. Цей режим ви будете використовувати для демонстрації роботи вашого контролера як доказ того, що він може справлятися навіть з випадками, яких не було в історичних даних.
- **`controller/controller.py`** реалізує контролер, який керує віртуальною інфраструктурою.
- **`data/full_balancer_model_normal.csv`** – опис поведінки користувачів системи, яка симулюється. Система має генерувати кількість запитів, які приходять на цю систему від користувачів кожну хвилину. В цьому файлі для кожної хвилини доби зберігається середня кількість

запитів від користувачів, які ми бачили в минулому, та стандартне відхилення цього значення.
- **`control_log.csv`** – логи вашої поточної симуляції.
- **`control.py`** – скрипт запуску симуляції.

Відкрийте **`control.py`** і запустіть симуляцію в ручному та автоматичному режимі з різним verbose, щоб ознайомитися з роботою симулятора.

Класи і методи, які вам знадобляться в завданні.

- `controller.control()` – запускає симуляцію під управлінням вашого алгоритму.
- `controller.takeAction()` – виконує одну з можливих дій по управлінню інфраструктурою.
- `controller.getNextState()` – виконує один крок симуляції. Повертає стан, в якому опинилася система після виконання вибраної дії і отриману нагороду.
- `controller.getCost()` – повертає отриману нагороду (в нашому випадку це буде від'ємна нагорода – кошти, витрачені на роботу системи).
- `controller.getFeatures()` – повертає ознаки стану та дії. Так само, як в лінійній і логістичній регресії, ви можете самі визначати, які саме ознаки використовуватимене ваша модель (скажімо, площа квартири, квадрат площі квартири і т.д.).
- `controller.estimateBestAction()` – повертає дію, яку потрібно виконати в цьому стані.
- `controller.setSpotFailureProbability()` – задає імовірність для кожної Spot машини бути забраної у вас в наступну хвилину.
- `controller.setSpotHourlyPrice()` – задає вартість години роботи Spot машини.
- `controller.setOverloadCost()` – задає вартість одного запиту користувача, який буде оброблений з затримкою чи не буде оброблений взагалі через перевантаження вашої інфраструктури.

Цей симулятор підтримує всі можливі алгоритми reinforcement learning включно з deep reinforcement learning і PEGASUS.

Ваша задача – навчити контролер управляти інфраструктурою так, щоб мінімізувати витрати на її роботу. Витрати складають вартість інфраструктури, плюс вартість запитів користувачів, опрацьованих з затримкою через перевантаження.

Ви можете почати з методу `controller.estimateBestAction()`. Там вже реалізований алгоритм управління, який запускає одну EC2 машину, якщо завантажено більше 80% поточних ресурсів і зупиняє одну EC2 машину, якщо завантажено менше 50%. В іншому випадку він нічого не робить.

Ваш контролер повинен працювати в умовах різної вартості Spot машин і різної імовірності виходу їх з ладу. Для цього вам знадобляться методи `controller.setSpotHourlyPrice()` і `controller.setSpotFailureProbability()`.

Ваш контролер також повинен працювати в умовах різної вартості запиту, опрацьованого з затримкою. Скажімо, для онлайн гри і для торгівлі на біржі його вартість буде дуже різною. Для цього вам знадобиться метод `controller.setOverloadCost()`.

Ваш контролер також повинен працювати на довільному навантаженні, яке можна згенерувати за допомогою ручного управління. Він не повинен оверфітити автоматичну симуляцію. Враховуйте це, коли створюєте ознаки стану для контролера.

Ваш контроллер буде оцінюватися на основі того, наскільки добре він працює:

- на автоматичній симуляції з фіксованою імовірністю виходу з ладу Spot машин, їх ціни і вартості затримки опрацювання запиту;
- на автоматичній симуляції зі змінною імовірністю виходу з ладу Spot машин, їх ціни і вартості затримки опрацювання запиту;
- на ручній симуляції з фіксованою імовірністю виходу з ладу Spot машин, їх ціни і вартості затримки опрацювання запиту;
- на ручній симуляції зі змінною імовірністю виходу з ладу Spot машин, їх ціни і вартості затримки опрацювання запиту.

4. Рекомендаційна система.

[15 балів]

Рекомендаційні системи – одне з найбільш цінних для бізнесу застосувань машинного навчання. Такі компанії, як Amazon та Netflix мають рекомендаційну систему в самому серці своїх технологій, їх прибуток напряму залежить від якості роботи цієї системи.

В цій задачі ви побудуєте систему рекомендацій фільмів, використовуючи item-item collaborative filtering.

У записнику **Problem Set 4 - Recommender system.ipynb** уже підготовлений загальний каркас коду та реалізована підготовка даних.

a. **[10 балів]** Використовуючи item-item collaborative filtering, визначте попарну схожість між кожною парою фільмів.

b. **[5 балів]** Використовуючи обраховану схожість між фільмами, порекомендуйте 5 фільмів глядачу, якому сподобалися такі фільми (іншими словами, для кожного з фільмів зі списку виведіть 5 найбільш схожих):

- Matrix, The (1999)
- Toy Story (1995)
- From Dusk Till Dawn (1996)
- Gone with the Wind (1939)
- Iron Man (2008)