Wrocław, 22 października 2024

Przemysł 4.0

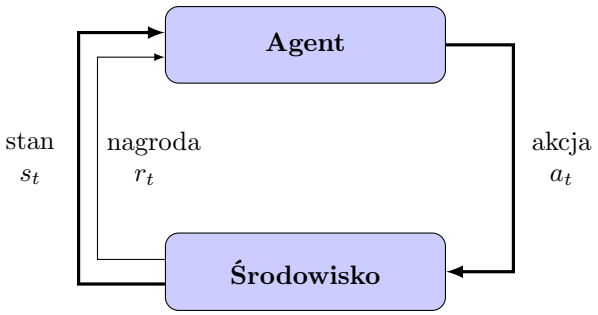
Laboratorium 3

Uczenie ze wzmocnieniem

prowadzący: *Dr inż. Radosław Idzikowski*

1 Wprowadzenie

Celem laboratorium jest zapoznanie się z uczeniem ze wzmocnieniem (ang. *Reinforcement Learning*, RL). RL służy do rozwiązywania problemów, które można opisać za pomocą agenta podejmującego decyzje w ramach środowiska (Rysunek 1). Decyzje podjęte przez agenta wiążą się z otrzymywaniem sygnału nagrody. W przeciwieństwie do innych typów uczenia maszynowego, agent otrzymuje pewną informację o "dobroci" podejmowanych decyzji. Nie jest to więc przypadek uczenia nienadzorowanego, gdzie nie ma zupełnie informacji o oczekiwanym wyjściu (w tym wypadku agent nie miałby dostępu do informacji o nagrodzie). Jednocześnie nie jest to przypadek uczenia nadzorowanego, gdzie przy treningu wykorzystuje się pary: wejście oraz przypisane oczekiwane wyjście (w tym wypadku agent trenowany nie byłby przez sygnał nagrody, ale przez przykłady idealnego zachowania w każdym z kroków decyzyjnych). Tym samym uczenie ze wzmocnieniem można stosować w przypadku środowisk, gdzie nieznane są konkretne kroki konieczne do osiągnięcia zadanego celu, ale znany jest co najmniej sposób oceny finalnego rezultatu (lub co poprawi działanie samego agenta).



Rysunek 1: Schematyczne przedstawienie podstawowych pojęć w ramach RL: agent wchodzi w interakcję z środowiskiem poprzez podjęcie akcji a_t, ta natomiast prowadzi do zmiany stanu środowiska na s_t oraz otrzymaniem nagrody lub kary r_t przez agenta za podjętą akcję – proces ten następnie powtarza się w kolejnych iteracjach.

Za źródłem [1] z zalecanej literatury, w ramach RL program komputerowy odpowiedzialny za podejmowanie skomplikowanych decyzji w warunkach niepewności nazywany jest agentem (ang. *agent*). Analizując zawarty przykład: jeśli trenowany jest program do sterowania ramieniem robotycznym w celu podnoszenia przedmiotu to samo w sobie ramie nie stanowi części agenta (pomimo, że agent podejmuje decyzje na nie wpływające) – jedynie część kodu odpowiedzialna za podejmowanie decyzji określana jest jako agent.

Następnym podstawowym pojęciem jest środowisko (ang. *environment*) – wszystko co znajduje się poza agentem jest środowiskiem. Kontynuując przykład ramienia robotycznego z [1] – zarówno obiekt który ma być podniesiony, jak i podłoga na której obiekt się znajduje, tak samo

jak samo ramie jest częścią środowiska. Choć agent ma kontrolę nad ramieniem i może je poruszać to, to tym bardziej ze względu na istnienie zakłóceń oraz nieidealne odwzorowanie decyzji agenta na stan ramienia, ramie jest częścią środowiska.

Niestety, pełna informacja dotycząca stanu środowiska nie może być w praktyce wykorzystana (po części również dlatego, że nie cały stan środowiska ma znaczenie w ramach zadanego problemu). Przez co samo środowisko operacjonalizuje się w postaci zmiennych liczbowych je opisujących w postaci stanu (ang. *state*). Zbiór wszystkich możliwych stanów tworzy przestrzeń stanów (ang. *state space*). Kontynuując przykład ramienia robotycznego w ramach pojedynczego stanu można zawrzeć informację o dokładnej lokalizacji ramienia, prędkościami przegubów, kątów pomiędzy przegubami, lokalizację obiektu który ramie ma podnieść. Stanu środowiska które agent jest w stanie bezpośrednio odbierać w postaci danych wejściowych nazywana jest obserwacją (ang. *observation*) – co ważne informacja w ramach obserwacji może być niepełna lub być zniekształcona. Wracając do przykładu z [1], agent może nie mieć dostępu do bezpośrednich wartości lokalizacji czy prędkości ramienia robotycznego, a jedynie dostęp do wartości pikseli z obrazu kamery skierowanej na ramie robotyczne oraz obiekt który należy podnieść.

Formalny zapis problemu w ramach RL wymagałby wykorzystania *procesu decyzyjnego Markowa* – jednak nie będzie to koniecznie w kontekście laboratoriów, zainteresowanych tematem zapraszam do literatury: [1, 2, 3].

2 Zadania

W ramach zajęć należy w zespołach wykonać następujące zadania:

1. Zapoznanie się z narzędziami `Google Colab`, `Gym`, `Stable-Baselines3` oraz `gnwrapper`.
2. Sprawdzenie różnych algorytmów w środowisku `Lunar Lander`.
3. Wizualizacja postępu procesu uczenia.

Za wykonanie zadania nr 1 jest ocena dostateczna, za każde kolejne zadanie jest +1 do oceny. Na ocenę bardzo dobrą (5.0) należy wykonać wszystkie trzy zadania. Na koniec zajęć należy pobrać `notebook` i przesłać go mailowo do prowadzącego.

3 Opis zadań

3.1 Zadanie 1

W ramach zadania należy zapoznać się z przygotowanym programem w `Google Colab` oraz z dokumentacją bibliotek. Do zajęć przygotowano `notebook` składający się z trzech komórek (ang. *cell*). W pierwszej komórce przebiega instalacja wszystkich niezbędnych bibliotek. Proszę zwrócić uwagę, że wszystkie linii są poprzedzone zdankiem "`!`", co oznacza wykonanie komendy bezpośrednio w terminalu. Na listingu 1 przedstawiono proces uczenia agenta. Biblioteka `gym` zawiera ustandaryzowaną implementację prostych środowisk, które będziemy używać podczas zajęć. Podczas instalacji zostały zainstalowane dwa zestawy (środowisko `Cart Pole` zawiera się w zadaniach `classic_control`, a potrzebny w kolejnym zadaniu `Lunar Lander` w `box2d`). Biblioteka `Stable_Baselines3` zawiera implementację typowych algorytmów do RL. W przykładzie użytko algorytmu `AC2`. Ponadto w celu przyśpieszenia obliczeń zastosowano `make_vec_env` wektor wirtualnych środowisk do uczenia równoległego.

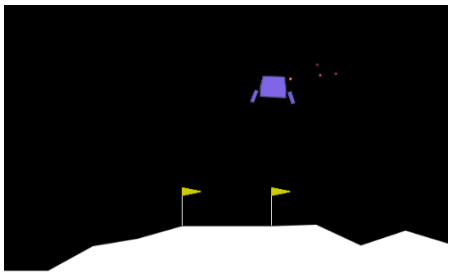
```
import gym
from stable_baselines3 import A2C
from stable_baselines3.common.env_util import make_vec_env

env = make_vec_env("CartPole-v1", n_envs=10) # gym.make("CartPole-v1") dla
    przypadku pojedynczego środowiska
model = A2C("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=5000, log_interval=10)
```

Listing 1: Proces uczenia agenta

`A2C` (*Advantage Actor-Critic*) to algorytm uczenia wzmacniającego, który łączy dwie kluczowe strategie: politykę (aktor) i funkcję wartości (krytyk). Aktor uczy się, jak podejmować decyzje, a krytyk ocenia te decyzje, przewidując przyszłe nagrody. Zaletą `A2C` jest jego stabilność dzięki synchronizacji aktualizacji, co pozwala na bardziej efektywne uczenie się w porównaniu do standardowego algorytmu `Actor-Critic`. Poleceniem kluczowym do uruchomienia treningu jest metoda `model.learn`, którym kluczowym parapetem jest liczba kroków.

3.2 Zadanie 2

Celem zadania jest skuteczne przeprowadzenie treningu w środowisku `Lunar Lander` z użyciem wybranego algorytmu do uczenia agenta. W przypadku zmiany środowiska proszę zwrócić uwagę, aby był tworzony cały ich wektor `make_vec_env`, w przeciwnym razie wybrany model zwróci błąd. Należy przetestować różne z dostępniej puli (zalecane: `A2C`, `DQN`, `PPO`). Jako efekt końcowy należy pokazać nauczony model, który poprawnie ląduje w wyznaczonej strefie między flagami. Przykład działania środowiska pokazano na rysunku 2.



Rysunek 2: Środowisko `Lunar Lander`

3.3 Zadanie 3

Dla wybranego algorytmu w zadaniu drugim, należy przeanalizować jak przebiega proces uczenia. W tym celu należy narysować wykres (biblioteka `matplotlib.pyplot`) z poziomu `Google Colab` z przebiegiem procesu uczenia w zależności od liczby epizodów dla wybranego algorytmu. Warto w pierwszej kolejności dostroić odpowiednie parametry. Na listingu 2 pokazano prosty sposób sposób na narysowanie wykresu.

```
import matplotlib.pyplot as plt
x = [i for i in range(1,11)]
y = [i**2 for i in range(1,11)]
plt.plot(x, y, label="Chart")
plt.legend()
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

Listing 2: Podstawy kod to rysowania wykresów

Literatura

[1] M. Morales, *Grokking deep reinforcement learning*, Manning Publications, 2020

[2] R. Sutton and A. Barto, *Reinforcement Learning, second edition: An Introduction*, Adaptive Computation and Machine Learning series, MIT Press, 2018. link

[3] Darmowy kurs na stronie Kaggle https://www.kaggle.com/learn/intro-to-game-ai-and-reinforcement-learning