Inteligencja obliczeniowa - Procesy decyzyjne Markova i Gym

Grzegorz Madejski

Problemy decyzyjne Markova (MDP)

Część I: Problemy decyzyjne Markova (MDP)

W świecie ludzi...

W życiu wykonujemy wiele czynności prowadzących do określonego celu. Zależy nam, by proces wykonywania czynności był jak najbardziej optymalny i prowadził do celu. Przykłady:

- Trzymam szczoteczkę z pastą i chcę porządnie umyć zęby. Jakie ruchy i w jakiej kolejności wykonywać, żeby zęby w całości i dokładnie były umyte?
- Jadę samochodem do pracy i podejmuje wiele decyzji: jechać prosto, skręcać na skrzyżowaniach, zawracać. Jaka sekwencja decyzji wyznaczy najlepszą (najkrótszą, najszybszą) drogę?

W świecie technologii...

W świecie informatyki, matematyki czy robotyki, również mamy do czynienia z procesami mającymi prowadzić do określonego celu.

- Autopilot samolotu chce dolecieć do celu. Steruje kierunkiem samolotu, pułapem wysokości i szeregiem innych parametrów. Decyzje podejmuje co ułamek sekundy patrząc na zmieniającą się sytuację.
- Klimatyzator chce osiągnąć idealną temperaturę. Co ma robić?
- Sztuczna inteligencja chce wygrać z nami w szachy. Jakie ruchy ma wykonywać w reakcji na nasze ruchy?
- Przewrócony humanoidalny robot chce wstać. Jak ma sterować przegubami rąk i nóg by najszybciej i najsprawniej wstać?

W świecie technologii...

Jak opisać / zakodować akcje, tak żeby komputery je rozumiały i mogłyby się nauczyć? Potrzebny sensowny aparat matematyczny.

MDP

Proces decyzyjny Markova

Proces decyzyjny Markova (ang. *Markov Decision Process*, *MDP*) to matematyczny model do symulowania podejmowania decyzji w czasie (w ruchach, w krokach) w środowisku częściowo losowym. System podejmujący decyzję (*agent*) nie musi pamiętać przeszłości (np. historii ruchów i zmian środowiska) - przyszłość zależy tylko od teraźniejszości! Decyzje mogą mieć różne prawdopodobieństwa wykonania, a ich wykonanie sterowane jest systemem nagród i kar. Szukamy najbardziej optymalnej sekwencji decyzji.

Agenty

W definicji MDP pojawiło się wiele pojęć, które wypada szerzej objaśnić. Zacznijmy od agenta. Najprościej mówiąc:

Agent

Agent to jednostka obserwująca środowisko, podejmująca decyzje, by osiągać określone cele. Agent często może ulepszać podejmowanie decyzji w procesie uczenia.

Agentem może być zarówno człowiek (używa zmysłów, podejmuje różne decyzje, uczy się dzięki mózgowi), ale również klimatyzacja (używa sensorów, podejmuje decyzje o schładzaniu lub ogrzewaniu, być może się uczy albo jest wyuczona).

Środowisko

Środowisko

Środowisko (ang. *environment*) to otoczenie agenta, które jest on w stanie dostrzec swoimi sensorami. Inaczej mówiąc: to sytuacja, w której znajduje się aktualnie agent.

Środowisko

Środowisko może mieć różne własności:

- W pełni lub tylko częściowo obserwowalne. Albo sensory widzą wszystko, albo trzeba nimi sterować, by obserwować wycinki środowiska.
- Deterministyczne lub stochastyczne. Albo agent ma całkowity wpływ na przyszłość, albo przyszłość zależy też od pewnych zdarzeń losowych.
- Jedno- lub wieloagentowe. W środowisku znajduje się tylko jeden agent podejmujący decyzję, lub wiele (rywalizujących lub kooperujących).
- Dynamiczne lub statyczne. Może się zmieniać w trakcie gry, lub być cały czas takie samo. Pacman (dyn) vs Labirynt (stat).
- Dyskretne lub ciągłe. Skończony zestaw ruchów i sytuacji (np. szachy) czy ciągłość (jazda taksówką).

Stany

- Środowisko zmienia się w czasie. W każdej jednostce czasu jest w jakimś *stanie* (ang. state) $s \in S$, który agent może zaobserwować.
- Stanem może być aktualna sytuacja na planszy szachowej (gra w szachy) lub na ulicy (autonomiczne auto).
- Wyróżniamy stan początkowy $s_0 \in S$ i stany terminalne.
- Bywa, ze stany nazywamy obserwacjami, ale to nie zawsze znaczy to samo, np. w częściowo obserwowalnych środowiskach.

Akcje

- Agenci podejmują decyzje i wykonują *akcje* (ang. actions) $a \in A$.
- Zwykle będąc w stanie s można wykonać tylko część akcji, ozn. $A(s)$.
- Akcją może być ruch wybraną szachową figurą (po przeanalizowaniu stanu gry) lub hamowanie samochodu autonomicznego.
- Bywa, ze stany nazywamy obserwacjami, ale to nie zawsze znaczy to samo, np. w częściowo obserwowalnych środowiskach.

Prawdopodobieństwa akcji

- Prawdopodobieństwo wykonania przejścia ze stanu s do stanu s' w punkcie czasu t za pomocą akcji a zapisujemy wzorem:

$$P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a) = Pr(s' | s, a)$$

- Zestaw prawdopodobieństw nazywamy modelem (tranzycji/przejść).
- W środowisku bez zakłóceń/poślizgów/turbulencji prawdopodobieństwa wynoszą albo 0, albo 1. Np. grając w szachy: jeśli wybierzemy akcję przesunięcia pionka o 1 do przodu, to znajdzie się on na polu o 1 do przodu z prawdopodobieństwem 1.
- W problemach związanych z życiem realnym, nie wszystko jest takie pewne. Pójście o krok do przodu na lodzie może (np. z prawdopodobieństwem 0.1) skutkować poślizgiem i pojechaniem o 3 kroki do przodu.

Nagrody

Po wykonaniu każdej akcji a prowadzącej ze stanu s do stanu s' algorytm otrzymuje natychmiastową *nagrodę* (ang. reward) $R_a(s, s')$. Nagroda może być karą (ang. penalty) jeśli ma wartość ujemną.

Polityka

- *Polityka* (ang. *policy*) to strategia wykonywania ruchów w oparciu o aktualny stan środowiska. Opisuje ona zachowanie agenta.
- Jeśli agent ma ruchy zdeterminowane, to polityka jest funkcją $\mu(s) = a$, która każdemu stanowi przyporządkowuje akcję. Zwykle jednak agent nie jest taki pewny...
- Wówczas politykę można rozumieć jako funkcję $\pi : A \times S \rightarrow [0, 1]$, która rozpatruje akcję i stan, i przypisuje takiej parze prawdopodobieństwo wykonania tej akcji $\pi(a, s) = Pr(a|s)$.
- Celem algorytmów uczących się jest znalezienie optymalnej (lub prawie optymalnej) polityki π^*. Przez optymalną politykę rozumiemy taką sekwencję akcji, która zmaksymalizuje sumę nagród po dojściu do stanu terminalnego.

Polityka

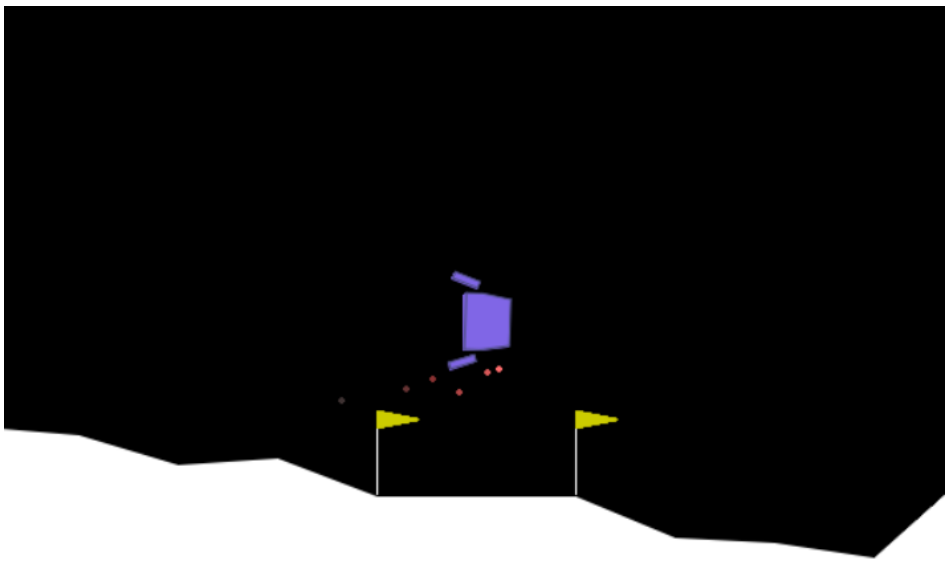
- W procesie uczenia, często agent losuje akcje, które uważa za najlepsze i wykonuje je.
- Szansa na wylosowanie danej akcji jest wskazana przez politykę $\pi(a, s) = Pr(a|s)$.
- Wylosowanie złej akcji nie zawsze jest złe - eksplorujemy inne możliwości! W niektórych algorytmach możemy dzieki temu znaleźć inną, lepszą politykę!
- Ważna jest różnica między modelem $Pr(s'|s, a)$ (szansa, z jaką dojdziemy do stanu s' za pomocą a), a polityką $Pr(a|s)$ (opłacalność akcji a wyliczona jakimś algorytmem).

Przykłady problemów i Gym

Część II: Przykłady problemów i Gym

LunarLander

LunarLander to gra, w której musimy sterować sondą i wylądować pomyślnie na księżycu w miejscu pomiędzy flagami. Należy to zrobić powoli, żeby sonda się nie rozbiła.



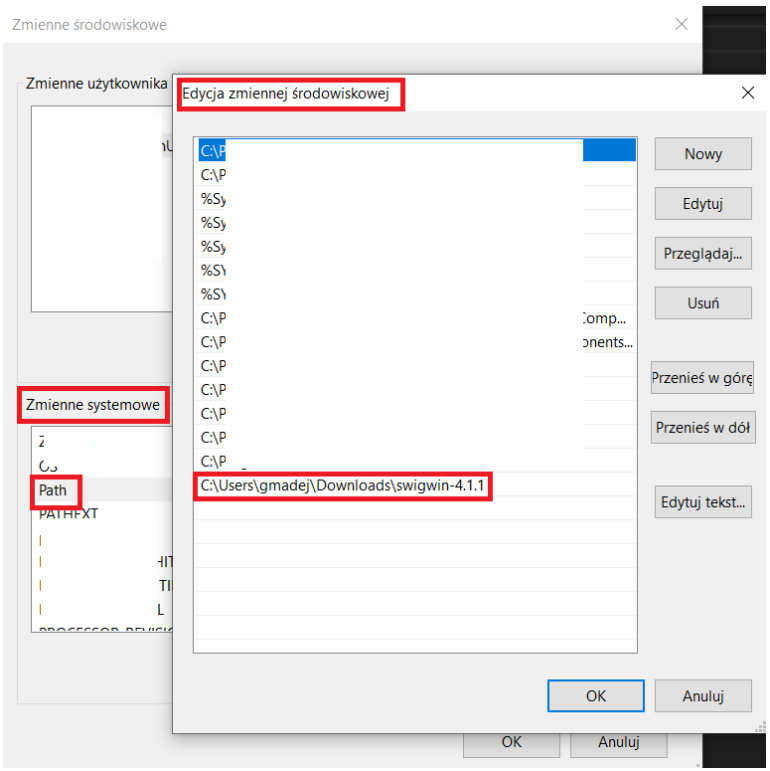
(Rzuć okiem na lunarlander.gif)

Gym/Gymnasium

- Do nauki rozwiązywania problemów (np. lądowania sondą) potrzebne jest wirtualne środowisko, w którym będziemy robić symulacje.
- Przez jakiś czas takie środowisko zwane *Gym* było prowadzone przez znaną organizację *OpenAI*.
- Projekt zarzucono, ale *Gym* nadal jest rozwijane przez społeczność, choć coraz bardziej niemrawo. Linki: `https://github.com/openai/gym` i `https://www.gymlibrary.dev/`.
- Powstają też narzędzia na bazie poprzedniego np. *Gymnasium* - `https://github.com/Farama-Foundation/Gymnasium`, `https://gymnasium.farama.org/`
- Oprócz tych dwóch gymów istnieje szereg innych alternatywnych narzędzi.

Gym/Gymnasium

Import Gym do Pythona może przynieść kilka trudności. Związane są one zwłaszcza z próbą pokazania graficznej symulacji (renderowanie). Być może trzeba doinstalować środowisko (pip install gym[box2d]) i Swig `https://www.swig.org/`. Na Windowsie, rozpakowałem paczkę swig i dodałem folder do path.



LunarLander w Gymie

Wypróbujmy kod lunar1.py:

```
import gym
env = gym.make("LunarLander-v2", render_mode="human")
env.reset(seed=42)

for _ in range(300):
    action = 1 #odpal lewy silnik
    env.step(action)
env.close()
```

Zmień 1 na jakiś random sample: env.action_space.sample()

LunarLander w Gymie

Wypróbujmy kod lunar3.py:

```
import gym
env = gym.make("LunarLander-v2", render_mode="human")
observation, info = env.reset(seed=42)

for _ in range(300):
   action = env.action_space.sample()
   observation, reward, terminated, truncated, info = env.step(action)

   if terminated or truncated:
      observation, info = env.reset()
env.close()
```

LunarLander

- Środowisko to wszystko, co sonda dostrzega swoimi sensorami tzn. środowisko obserwowalne (np. położenie i prędkości sondy) oraz ukryte przed agentem (grawitacja, wiatr, turbulencje, ukształtowanie terenu).
- W każdym kroku czasowym środowisko może być w innym stanie.
- W naszym przypadku stan środowiska obserwowalnego to wektor 8 parametrów:
  - współrzędne x i y
  - prędkości w poziomie i pionie v_x i v_y
  - kąt nachylenia sondy θ wyrażony w radianach
  - prędkość kątowa ω [radiany/sec]
  - dwie flagi bitowe wykrywające czy lewa i prawa noga dotykają podłoża
- Jak wyglądają stany początkowe i terminalne? Odpal lunar4.py.

LunarLander

- Agent może wykonywać 4 różne akcje: nie rób nic (0), odpal silnik lewy (1), odpal silnik prawy (3), odpal silnik dolny/główny (2).
- Jakie akcje wykonuje sonda? Odpal lunar5.py.

LunarLander

- Funkcja nagrody w algorytmach uczących powinna premiować: pomyślne lądowanie, wolne prędkości, oszczędzenia paliwa, szybkie lądowanie.
- Jak wygląda funkcja nagrody w LunarLander? `https://gymnasium.farama.org/environments/box2d/lunar_lander/#rewards`
- Przykładowy wzór, który ma szansę zadziałać:

$$R(s_t, s_{t-1}) = -100 \cdot (d_t - d_{t-1}) - 100 \cdot (v_t - v_{t-1}) - 100 \cdot (\omega_t - \omega_{t-1})$$
$$+ hasLanded(s_t)$$

gdzie d_t to odległość od lądowiska, v_t to prędkość, ω_t to prędkość kątowa, *hasLanded* sprawdza było lądowanie (pomyslne lub niepomyślne)

FrozenLake

W grze FrozenLake mamy planszę z zamarzniętym jeziorem. Musimy dojść od lewego górnego rogu, do prezentu w prawym dolnym. Należy omijać dziury w lodzie. Uwaga, czasem można poślizgnąć się na lodzie i pójść nie tam gdzie się chciało. Link: `https://gymnasium.farama.org/environments/toy_text/frozen_lake/`

FrozenLake

- Środowisko to cała plansza gry (płytki: start, goal, hole, frozen) i współrzędne agenta. Stan środowiska zmienia się, gdy agent ma inne współrzędne. Obserwacja składa się tylko ze współrzędnych agenta, przy czym ujęta jest za pomocą jednej liczby: $mojWiersz \cdot liczbaWierszy + mojeKolumna$ (np. agent we współrzędnych 2, 1 na planszy 4x4 ma pozycję 2*4+1=9).
- Agent to ludzik biegający po lodzie.
- Agent może wykonywać 4 różne akcje, ruch w lewo, prawo, do góry, w dół.
- Funkcja nagrody premiuje za dojście do prezentu.
- Jezioro jest śliskie (is_slippery), więc ruch nie zawsze może się udać.

FrozenLake w Gymie

Wypróbujmy poniższy kod:

```
import gym
import numpy as np

env = gym.make('FrozenLake8x8-v1', render_mode="human")
observation, info = env.reset(seed=42)

for _ in range(10):
   action = env.action_space.sample()
   observation, reward, terminated, truncated, info = env.step(action)

   if terminated or truncated:
      observation, info = env.reset()
env.close()
```