Politechnika Wrocławska

Algorytmy sztucznej inteligencji w Przemyśle 4.0

Uczenie ze wzmocnieniem

Dr inż. Radosław Idzikowski

HR EXCELLENCE IN RESEARCH

Definicja

Uczenie ze wzmocnieniem (*Reinforcement Learning*, RL)

to gałąź uczenia maszynowego, w której agent uczy się przez interakcję ze środowiskiem, aby maksymalizować sumę nagród. W odróżnieniu od uczenia nadzorowanego, gdzie model uczony jest na podstawie danych z etykietami, RL nie ma wstępnie zdefiniowanych poprawnych odpowiedzi. Agent samodzielnie poznaje, które działania prowadzą do najlepszych wyników.

RL opiera się na psychologicznej koncepcji warunkowania instrumentalnego — podobnie jak w przypadku ludzi czy zwierząt, agent uczy się przez konsekwencje swoich działań.

Podstawowe zastosowania

Robotyka

Programowanie robotów do wykonywania złożonych sekwencji ruchów, takich jak chodzenie, omijanie przeszkód czy manipulacja obiektami.

Automatyzacja gier

Rozwój algorytmów, które uczą się wygrywać w grach komputerowych i strategicznych, jak np. szachy, `Go`, `StarCraft`.

Optymalizacja procesów logistycznych i produkcyjnych

Wykorzystanie `RL` do usprawniania operacji magazynowych, tras dostaw czy zarządzania produkcją, gdzie ważne jest dynamiczne dostosowywanie decyzji do zmieniających się warunków.

Koncepcja

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.

Uczenie ze wzmocnieniem

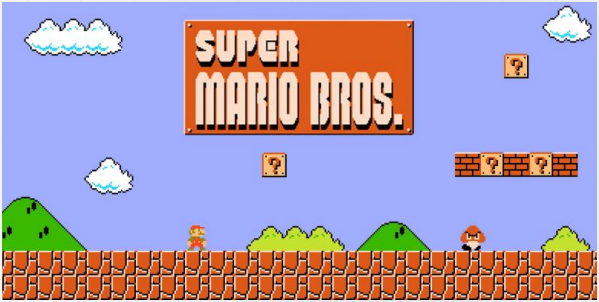
Proces decyzyjny Markowa

Proces decyzyjny Markowa (*Markov Decision Process*, MDP)

Ciąg zdarzeń, w którym prawdopodobieństwo każdego zdarzenia zależy jedynie od wyniku poprzedniego. W ujęciu matematycznym, procesy Markowa to takie procesy stochastyczne, które spełniają własność Markowa, że warunkowe rozkłady prawdopodobieństwa przyszłych stanów procesu są zdeterminowane wyłącznie przez jego biedacy stan, bez względu na przeszłość. [a]

[a] wikipedia

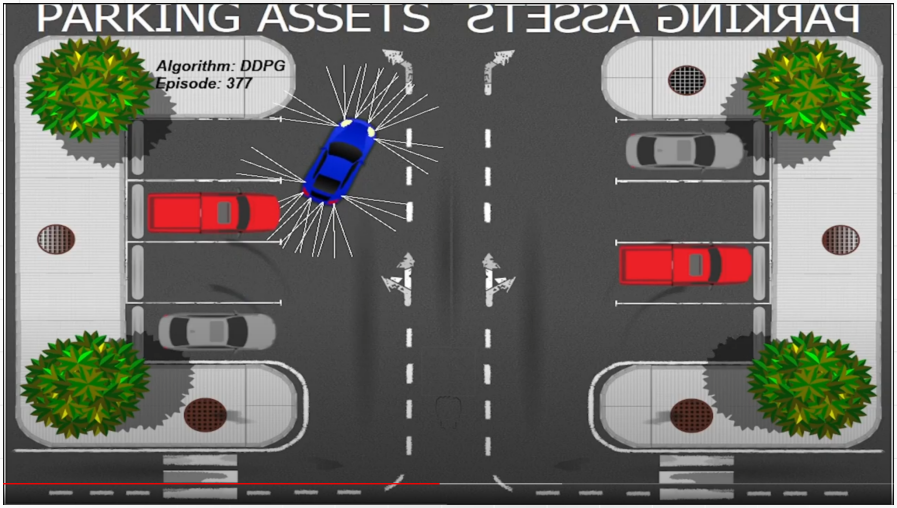
Super Mario Bros



Rysunek 1: Super Mario Bros

- Tsay, Jyh-Jong, Chao-Cheng Chen, and Jyh-Jung Hsu. "Evolving intelligent mario controller by reinforcement learning." 2011 International Conference on Technologies and Applications of Artificial Intelligence. IEEE, 2011.
- Shu, Tianye, Jialin Liu, and Georgios N. Yannakakis. "Experience-driven PCG via reinforcement learning: A Super Mario Bros study." 2021 IEEE Conference on Games (CoG). IEEE, 2021.

System parkowania



Rysunek 2: *Automatyczne parkowanie z wykorzystaniem metod sztucznej inteligencji* – praca magisterska A. Sobecki (2023)

Multi-arm bandits problem

Maze

Wyobraźmy sobie prosty labirynt, w którym agent chce znaleźć drogę do wyjścia. Labirynt składa się z siatki 6x6, w której:

- Startujemy w górnym lewym rogu (0, 0),
- Wyjście znajduje się w prawym dolnym rogu (5, 5),
- Agent może poruszać się w czterech kierunkach: w górę, w dół, w lewo i w prawo.

Agent otrzymuje nagrodę za dotarcie do wyjścia (np. $+1$), a w każdym innym kroku nagroda wynosi 0. Agent nie zna układu labiryntu ani nagrody związanej z wyjściem – musi więc uczyć się przez eksplorację.

Q-learning

Metoda uczenia ze wzmocnieniem, która pozwala agentowi uczyć się optymalnej polityki, prowadzącej do maksymalizacji nagrody, nawet bez znajomości modelu środowiska. Agent uczy się przez eksplorację środowiska i modyfikację tzw. wartości Q (wartości funkcji Q) dla każdej kombinacji stanu i akcji.

Wartość Q

liczba, która odzwierciedla jakość danej akcji w danym stanie. Jest to oczekiwana skumulowana nagroda, którą agent uzyska, jeśli podejmie akcję z danego stanu i będzie kontynuował działanie zgodnie z najlepszą strategią.

Q-learning

Aktualizacja wartości Q

Za każdym razem, gdy agent podejmie akcję i otrzyma nagrodę, aktualizujemy wartość Q dla tej kombinacji stanu i akcji. Aktualizacja wartości Q dla stanu s i akcji a wygląda następująco:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)), \tag{1}$$

gdzie:

- α to współczynnik uczenia (reguluje tempo, z jakim uczymy się nowych wartości),
- r to nagroda za przejście do nowego stanu s',
- γ to współczynnik dyskontowy, który określa, jak bardzo przyszłe nagrody wpływają na aktualne decyzje.

Q-learning

Eksploracja i eksploatacja

Aby znaleźć optymalną politykę, agent równoważy eksplorację (próbowanie nowych akcji) z eksploatacją (wybieranie najlepszej obecnie znanej akcji). Używa się np. strategii ϵ-zachłannej, w której agent wybiera losową akcję z prawdopodobieństwem ϵ, a najlepszą znaną akcję z prawdopodobieństwem $1 - \epsilon$.

Po zakończeniu uczenia tablica Q przechowuje wartości optymalnych akcji dla każdego stanu. Algorytm Q-learning w tym labiryncie pozwala agentowi uczyć się, jak najlepiej dotrzeć do wyjścia (prawy dolny róg) poprzez ciągłe eksplorowanie i eksploatowanie przestrzeni stanów.

maze

Narzędzia

- `Google Colab` – darmowe środowisko (choć bez gwarancji dostępności zasobów) w chmurze, pozwala wykonywać kod `PYTHON` oraz jednocześnie dokumentować ten proces w ramach `Jupyter notebooks`.
- `Gym` – biblioteka zawierająca ustandaryzowaną implementację prostych, podstawowych środowisk dla RL.
- `Stable-Baselines3` – biblioteka zawierająca implementację algorytmów odpowiedzialnych za trening i decyzje agenta.
- `Anaconda` – zalecany menadżer środowiska programistycznego (nie w znaczeniu środowiska w ramach RL) dla języka `PYTHON` do zarządzania bibliotekami.

Stable-Baselines3

A2C (Advantage Actor-Critic)

A2C to algorytm actor-critic, który stosuje synchronizowaną wersję metody Actor-Critic. Składa się z dwóch sieci:

- Actor decyduje, którą akcję wybrać.
- Critic ocenia, jak dobra była ta akcja, bazując na funkcji wartości stanu (V).

A2C różni się od klasycznego Actor-Critic przez równoległe, zsynchronizowane przetwarzanie wielu agentów, co pozwala na bardziej stabilne aktualizacje. Każdy agent dostarcza próbki z różnych epizodów, co zmniejsza korelację pomiędzy kolejnymi stanami i poprawia stabilność treningu.

Stable-Baselines3

DQN (Deep Q-Network)

`DQN` to algorytm `Q-Learningu`, który wykorzystuje sieci neuronowe do aproksymacji wartości Q dla dużych przestrzeni stanów i akcji. Stosuje replays buffer oraz sieć docelową (target network) dla stabilności uczenia:

- Replay buffer przechowuje wcześniejsze doświadczenia agenta, aby podczas treningu używać różnorodnych danych.
- Target network służy jako stabilny cel dla wartości Q w czasie uczenia, zmniejszając oscylacje i rozbieżności w oszacowaniach.

`DQN` działa najlepiej w przestrzeniach akcji dyskretnych, takich jak gry `Atari`.

Stable-Baselines3

DDPG (Deep Deterministic Policy Gradient)

DDPG jest algorytmem uczenia z krytykiem, zaprojektowanym do pracy w przestrzeniach akcji ciągłych. Łączy elementy Deep Q-Learning (DQN) i Actor-Critic:

- Critic to sieć Q, która ocenia wartość pary stan-akcja.
- Actor wybiera akcję, którą następnie ocenia Critic.

Aby zapewnić stabilność, DDPG wykorzystuje technikę replay buffer (bufor przechowywania przeszłych interakcji) oraz „powolne" kopiowanie parametrów sieci głównej do sieci docelowej (target network). Dzięki tym technikom DDPG jest w stanie lepiej optymalizować modele w środowiskach z akcjami ciągłymi.

Stable-Baselines3

HER (Hindsight Experience Replay)

`HER` to technika modyfikująca sposób korzystania z replay buffer, zaprojektowana do działania ze środowiskami, w których występuje wiele celów (np. zadania manipulacyjne w robotyce). `HER` dodaje do replay buffer alternatywne cele zrealizowane przez agenta nawet wtedy, gdy początkowy cel nie został osiągnięty. Dzięki temu agent może uczyć się, jak zrealizować różne cele, a nie tylko te konkretne, których wymaga dane zadanie, co zwiększa efektywność nauki w środowiskach z wysoką niepewnością.

Stable-Baselines3

PPO (Proximal Policy Optimization)

PPO to algorytm, który jest zoptymalizowaną wersją metody Actor-Critic, wykorzystującą zasadę ograniczenia kroków w celu zapewnienia stabilnych aktualizacji polityki:

- PPO stosuje funkcję straty z klipowaniem wartości gradientu, co ogranicza wielkość aktualizacji polityki, aby uniknąć gwałtownych zmian.
- PPO posiada dwie wersje: PPO-clip (klipowanie gradientów) oraz PPO-Penalty (dodatkowy składnik kary).

PPO działa skutecznie zarówno w przestrzeniach akcji dyskretnych, jak i ciągłych, i jest jednym z najczęściej stosowanych algorytmów RL, ponieważ łączy stabilność i efektywność.

Stable-Baselines3

SAC (Soft Actor-Critic)

`SAC` to algorytm z rodziny *Actor-Critic*, zaprojektowany do optymalizacji w środowiskach z akcjami ciągłymi. Jego kluczowym aspektem jest połączenie maksymalizacji nagrody z maksymalizacją entropii:

- Entropia pozwala agentowi zachować większą różnorodność wyborów (eksploracja), co pomaga w unikaniu zapętlenia w lokalnych optimum.
- SAC stosuje strategię off-policy, co oznacza, że uczy się na podstawie próbek z replay buffer.

Dzięki maksymalizacji entropii, `SAC` zapewnia lepszą eksplorację, co czyni go bardziej skutecznym w środowiskach, gdzie eksploracja jest kluczowa.

Stable-Baselines3

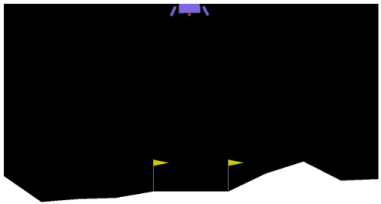
TD3 (Twin Delayed Deep Deterministic Policy Gradient)

`TD3` jest rozwinięciem `DDPG`, które poprawia jego stabilność i wydajność w środowiskach z akcjami ciągłymi. *TD3* wprowadza trzy główne ulepszenia:

- Dwukrotne krytyki: `TD3` stosuje dwie sieci Q do oceny wartości akcji, aby zmniejszyć pozytywną stronniczość w oszacowaniach wartości.
- Opóźniona aktualizacja aktora: `Actor` jest aktualizowany rzadziej niż `Critic`, co prowadzi do bardziej stabilnych aktualizacji.
- Dodawanie szumu do akcji docelowych: Szum jest dodawany do akcji w celu uniknięcia nadmiernego dostosowania się do błędnych wartości akcji docelowych.

`TD3` jest bardzo skuteczny w środowiskach z ciągłymi przestrzeniami akcji, gdzie wymagana jest precyzyjna optymalizacja.

Gym is a standard API for reinforcement learning, and a diverse collection of reference environments



The Gym interface is simple, pythonic, and capable of representing general RL problems:

```
import gym
env = gym.make("LunarLander-v2", render_mode="human")
observation, info = env.reset(seed=42)
for _ in range(1000):
   action = policy(observation)  # User-defined policy function
   observation, reward, terminated, truncated, info = env.step(action)

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