Projekt 2: Lunar Lander

Dawid Roszman

Czerwiec 2024

1 Opis zadania

Chcemy pomyślnie wyladować sonda na ksieżycu. Napisz program, który to wykona. Do rozpatrzenia sa trzy możliwości:

- własny skrypt,
- algorytm reinforcement learning (w tym także z siecia neuronowa)
- kontroler rozmyty (wymaga napisania bazy i reguł)
- strategie metaheurystyczne (PSO lub Algorytm Genetyczny)

Na końcu zademonstruj działanie algorytmów na animacjach (możesz zapisać jaki video lub gify)

2 Pierwsze podejście - własny skrypt

W swoim pierwszym podejściu napisałem skrypt, który na podstawie nagrody otrzymanej za ruch zwiekszał lub zmniejszał prawdopodobieństwo wykonania tego ruchu w kolejnych krokach. To podejście miało jednak swoje wady, w szczególności opierało sie wyłacznie na ostatniej nagrodzie, pomijajac długoterminowe skutki.

W pierwszej próbie stworzyłem skrypt uczacy sie przez wzmacnianie pozytywnych akcji. Niestety, metoda ta działała tylko w oparciu o ostatnia nagrode i nie uwzgledniała długoterminowych efektów.

Poprawiłem ten problem, stosujac strategie Epsilon-Greedy. Ta strategia uczy zarówno na podstawie natychmiastowych nagród, jak i poprzez eksploracje nowych możliwości. Dodatkowo, wprowadzono mechanizm cofania sie do udanych wyborów z poprzednich iteracji, jeśli aktualna próba okazuje sie mniej efektywna. W miare upływu czasu agent mniej eksploruje, a bardziej polega na zdobytej wiedzy, zwiekszajac prawdopodobieństwo sukcesu.

3 Reinforcement Learning

W drugim podejściu zdecydowałem sie skorzystać z biblioteki stable_baselines3, która umożliwia trenowanie agenta za pomoca zaawansowanych algorytmów, takich jak PPO (Proximal Policy Optimization) oraz A2C (Advantage Actor-Critic). Na poczatku wytrenowałem model przy użyciu algorytmu A2C. Chociaż rakieta była w stanie ladować, proces ten był stosunkowo powolny i nie zawsze kończył sie sukcesem. Nastepnie wytrenowałem model za pomoca algorytmu PPO. Ten model uczył sie znacznie szybciej i z wieksza precyzja, co przełożyło sie na bardziej stabilne i efektywne ladowania.

3.1 Algorytm PPO

Algorytm Proximal Policy Optimization (PPO) jest jednym z najnowszych i najskuteczniejszych algorytmów uczenia przez wzmacnianie. Został opracowany przez OpenAI i jest szeroko stosowany ze wzgledu na swoja prostote, efektywność i stabilność.

Cel PPO

PPO daży do poprawy polityki agenta (czyli strategii działania) w sposób stabilny i skuteczny, unikajac dużych kroków aktualizacji, które moga prowadzić do niestabilności.

Podejście do aktualizacji

Zamiast bezpośrednio maksymalizować funkcje celu, PPO wprowadza ograniczenia na wielkość zmian polityki, co pomaga utrzymać stabilność procesu uczenia.

Proces uczenia

1. **Generowanie doświadczeń**: Agent zbiera doświadczenia z interakcji ze środowiskiem, rejestrujac stany, akcje, nagrody i stany nastepne.
2. **Obliczanie przewagi**: Na podstawie zebranych danych oblicza sie wskaźnik przewagi dla każdej akcji.
3. **Aktualizacja polityki**: Aktualizacja parametrów polityki odbywa sie przy użyciu funkcji celu PPO, co zapewnia, że zmiany sa kontrolowane i stabilne.

Zalety PPO

- **Stabilność**: Dzieki mechanizmowi ograniczania zmian polityki, PPO jest bardziej stabilny w porównaniu do innych metod.
- **Prostota**: PPO jest prostszy do implementacji niż metody takie jak Trust Region Policy Optimization (TRPO), które również daża do stabilnych aktualizacji.
- **Efektywność**: PPO efektywnie wykorzystuje dane, co czyni go wydajnym w praktyce.

PPO jest szeroko stosowany w różnych aplikacjach uczenia przez wzmacnianie ze wzgledu na swoja efektywność i stabilność, co czyni go jednym z najpopularniejszych algorytmów w tej dziedzinie.

3.2 A2C

Algorytm Advantage Actor-Critic (A2C) jest jednym z popularnych algorytmów uczenia przez wzmacnianie. A2C łaczy w sobie elementy metod policy gradient (aktora) i value-based (krytyka), co pozwala na efektywne trenowanie agenta.

Cel A2C

Celem A2C jest znalezienie optymalnej polityki działania (strategii), która maksymalizuje skumulowana nagrode otrzymywana przez agenta w długim okresie. Algorytm osiaga to poprzez jednoczesne aktualizowanie dwóch sieci neuronowych: aktora i krytyka.

Aktor i Krytyk

- **Aktor**: Sieć neuronowa, która generuje polityke (policy), czyli prawdopodobieństwa wyboru konkretnych akcji w danym stanie. Aktor aktualizuje swoje parametry na podstawie wskaźnika przewagi (advantage).
- **Krytyk**: Sieć neuronowa, która estymuje funkcje wartości (value function), czyli oczekiwana skumulowana nagrode dla danego stanu. Krytyk pomaga w ocenie jakości polityki generowanej przez aktora.

Wskaźnik przewagi

Wskaźnik przewagi (*advantage*) jest miara jakości podjetej akcji w danym stanie w porównaniu do oczekiwań. Jest on obliczany jako różnica miedzy rzeczywista skumulowana nagroda a estymowana wartościa stanu:

$$A(s, a) = Q(s, a) - V(s)$$

gdzie:

- $Q(s, a)$ to wartość akcji a w stanie s (estymowana skumulowana nagroda).
- $V(s)$ to wartość stanu s (oczekiwana skumulowana nagroda).

Proces uczenia

1. **Generowanie doświadczeń**: Agent zbiera doświadczenia z interakcji ze środowiskiem, rejestrujac stany, akcje, nagrody i stany nastepne.
2. **Obliczanie przewagi**: Na podstawie zebranych danych oblicza sie wskaźnik przewagi dla każdej akcji.
3. **Aktualizacja aktora i krytyka**: Aktualizacja parametrów aktora i krytyka odbywa sie przy użyciu odpowiednich funkcji celu, co zapewnia, że polityka i estymacje wartości staja sie coraz dokładniejsze.

Zalety A2C

- **Efektywność**: A2C efektywnie łaczy zalety metod policy gradient i value-based, co przyspiesza proces uczenia.
- **Stabilność**: Dzieki jednoczesnemu uczeniu aktora i krytyka, algorytm jest bardziej stabilny w porównaniu do metod opartych wyłacznie na policy gradient.
- **Elastyczność**: A2C może być stosowany w różnych środowiskach i problemach uczenia przez wzmacnianie.

A2C jest szeroko stosowany w aplikacjach uczenia przez wzmacnianie, oferujac skuteczność i stabilność w trenowaniu agentów.

4 Kolejne kroki

Po udanym wytrenowaniu modelu na algorytmie PPO postanowiłem zmienić parametry środowiska, aby sprawdzić czy model jest w stanie nauczyć sie ladować w innych warunkach. Zmieniłem m.in. grawitacje, oraz dodałem wiatr. Obecnemu modelowi czasami udaje sie ladować, jednak nie jest to zbyt skuteczne. Postanowiłem go dotrenować na nowych warunkach. Trenowałem ten sam model zmieniajać parametry środowiska. Po całkiem długim uczeniu model zaczał ladować w nowych warunkach. Obecnie model jest w stanie ladować w różnych warunkach (nawet ekstremalnych).

5 Algorytm genetyczny

Algorytmy genetyczne (AG) sa technikami optymalizacyjnymi, które imituja procesy biologiczne. Populacja rozwiazań jest iteracyjnie ulepszana poprzez selekcje najlepszych osobników, krzyżowanie ich cech oraz wprowadzanie mutacji. W kontekście Lunar Lander, każde rozwiazanie (osobnik) reprezentuje sekwencje działań, które agent ma wykonać.

Kroki algorytmu genetycznego

1. **Inicjalizacja**: Generowanie poczatkowej populacji losowych rozwiazań (chromosomów).
2. **Selekcja**: Wybór najlepszych rozwiazań na podstawie funkcji oceny (fitness function).
3. **Krzyżowanie (Crossover)**: Łaczenie par wybranych rozwiazań w celu stworzenia nowych rozwiazań (potomstwa).
4. **Mutacja**: Wprowadzenie losowych zmian do nowych rozwiazań w celu zachowania różnorodności genetycznej.
5. **Ocena**: Obliczenie wartości funkcji oceny dla nowych rozwiazań.
6. **Zastapienie**: Wybór najlepszych rozwiazań do nowej populacji.
7. **Iteracja**: Powtarzanie kroków 2-6 przez określona liczbe generacji.

Funkcja Oceny (Fitness Function)

Funkcja oceny jest kluczowym elementem algorytmu genetycznego, ponieważ określa jakość poszczególnych rozwiazań. W naszym przypadku funkcja oceny ocenia sekwencje działań agenta na podstawie sumy nagród uzyskanych w trakcie jednego epizodu ladowania.

Kryteria oceny

- **Nagroda za ladowanie**: Agent otrzymuje pozytywne punkty za prawidłowe ladowanie.
- **Kara za duża predkość**: Jeżeli predkość pionowa przekracza pewien próg, agent otrzymuje kare, aby zniechecić do zbyt szybkiego opadania.
- **Kara za nachylenie**: Jeżeli nachylenie rakiety przekracza pewien kat, agent otrzymuje kare, aby promować stabilne ladowanie.

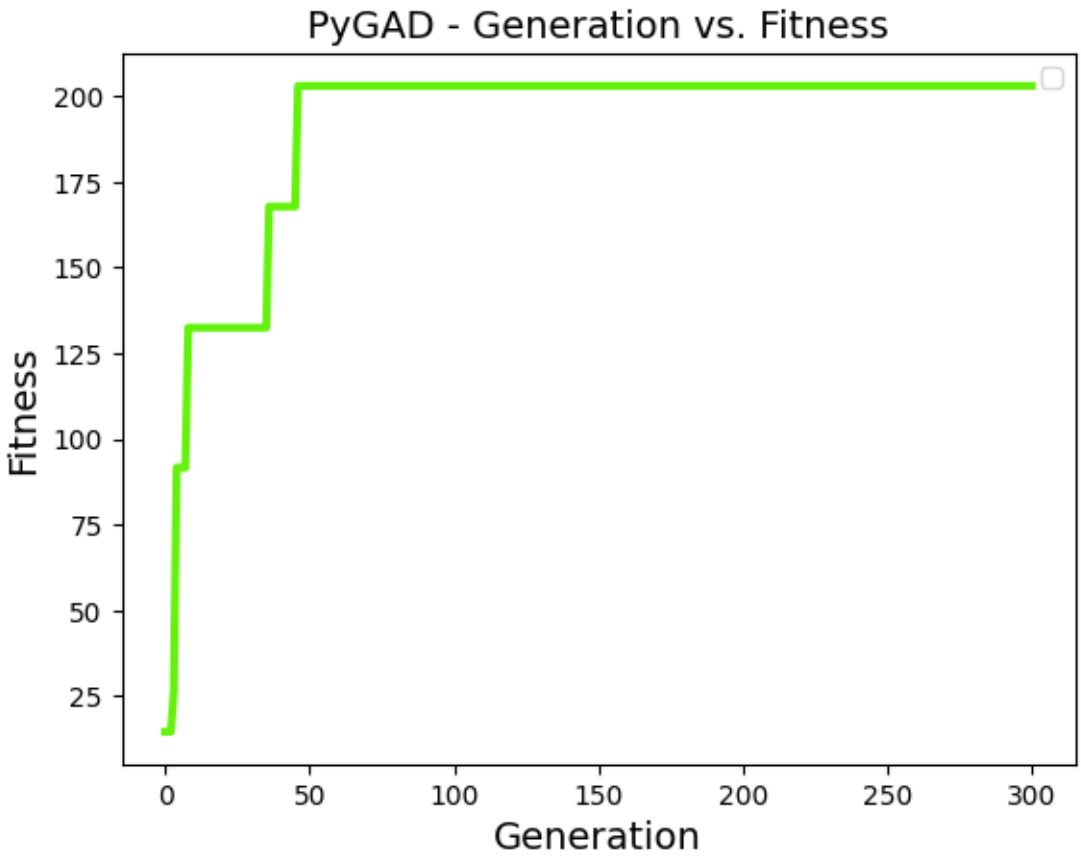
Łaczac powyższe kryteria, funkcja oceny weryfikuje, jak dobrze agent radzi sobie z zadaniem ladowania rakieta, nagradzajac stabilne i kontrolowane ladowanie, jednocześnie karzac za ryzykowne manewry.

Parametry Algorytmu

Używamy nastepujacych parametrów w naszym algorytmie genetycznym:

- **Populacja**: 100 chromosomów (rozwiazań).
- **Geny**: Każdy chromosom zawiera 200 genów, reprezentujacych maksymalna liczbe ruchów.
- **Liczba pokoleń**: Algorytm jest uruchamiany przez 300 generacji.
- **Selekcja rodziców**: 50
- **Typ krzyżowania**: Używamy jednokrotnego krzyżowania (single-point crossover).
- **Mutacja**: Używamy typu mutacji *swap*, z domyślnym procentem genów podlegajacych mutacji.

Wyniki i Wizualizacja



6 Podsumowanie

Z przeprowadzonych eksperymentów wynika, że najlepsze wyniki osiagnieto stosujac algorytmy uczenia przez wzmacnianie. Algorytmy te potrafiły efektywnie dostosować sie do zmiennych warunków środowiskowych, takich jak zmiany grawitacji i wiatru. W przeciwieństwie do tego, algorytm genetyczny nie wykazał sie równie dobra adaptacyjnościa. Definiował on stały zestaw ruchów, który nie był wystarczajaco elastyczny, aby poradzić sobie z różnorodnościa warunków poczatkowych, takich jak różne predkości startowe czy zmienne nachylenie.

7 Materiały

- https://www.youtube.com/watch?v=nRHjymV2PX8 YouTube - Reinforcement Learning
- https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning GeeksforGeeks - Epsilon-Greedy Algorithm in Reinforcement Learning
- https://gymnasium.farama.org/environments/box2d/lunar_lander Gymnasium - Lunar Lander Environment
- https://github.com/sudharsan13296/Hands-On-Reinforcement-Learning-With-Python/blob/master/11.
- https://stable-baselines.readthedocs.io/en/master Stable Baselines Documentation
- https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html Stable Baselines3 - RL Tips
- https://codecrucks.com/mamdani-fuzzy-inference-method-example Codecrucks - Mamdani Fuzzy Inference Method Example