Implementacja i analiza algorytmu Q-Learning

Mateusz Łopaciński

20 kwietnia 2025

Streszczenie

Celem raportu jest implementacja oraz analiza działania algorytmu Q-Learning na przykładzie klasycznego środowiska „CartPole-v1" dostępnego w bibliotece Gym. Projekt obejmuje cztery zadania, których celem jest zbudowanie w pełni funkcjonalnego agenta uczącego się, ocena jego skuteczności oraz poprawna wizualizacja wyników.

Spis treści

1 Wstęp

Q-Learning to jedna z podstawowych metod uczenia ze wzmocnieniem (Reinforcement Learning, RL), pozwalająca agentowi uczyć się optymalnej polityki działania w środowisku poprzez maksymalizację łącznej oczekiwanej nagrody. W niniejszym projekcie zadaniem agenta było opanowanie balansowania kijkiem na ruchomym wózku, wykorzystując do tego celu środowisko „CartPole-v1" z biblioteki Gym.

2 Implementacja algorytmu Q-Learning

Pierwszym krokiem w projekcie było dokończenie i poprawna implementacja algorytmu Q-Learning, w oparciu o uprzednio przygotowany szkielet w pliku `balance.py`. Początkowy kod zapewniał jedynie losowe operowanie agentem w środowisku, dlatego wymagał kilku kluczowych uzupełnień:

- **Dyskretyzacja stanu:** Aby możliwe było przechowywanie Q-wartości w słowniku, należało dokonać dyskretyzacji obserwacji środowiska. Umożliwiło to przypisywanie stanów do odpowiadających im kubełków.
- **Zasada wyboru akcji:** Zaimplementowano politykę -greedy, zapewniającą kompromis pomiędzy eksploracją nowych akcji a eksploatacją zdobytej wcześniej wiedzy.
- **Aktualizacja wiedzy:** Algorytm został wzbogacony o właściwą regułę aktualizacji Q-wartości zgodnie z formułą Q-Learning:

$$Q^{\text{new}}(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \right)$$

- **Parametry:** Dla poprawnego uczenia się kluczowe okazało się dobranie odpowiednich wartości parametrów: tempu uczenia się α, współczynnika dyskontowego γ oraz współczynnika eksploracji ε, a także liczby kubełków w dyskretyzacji.

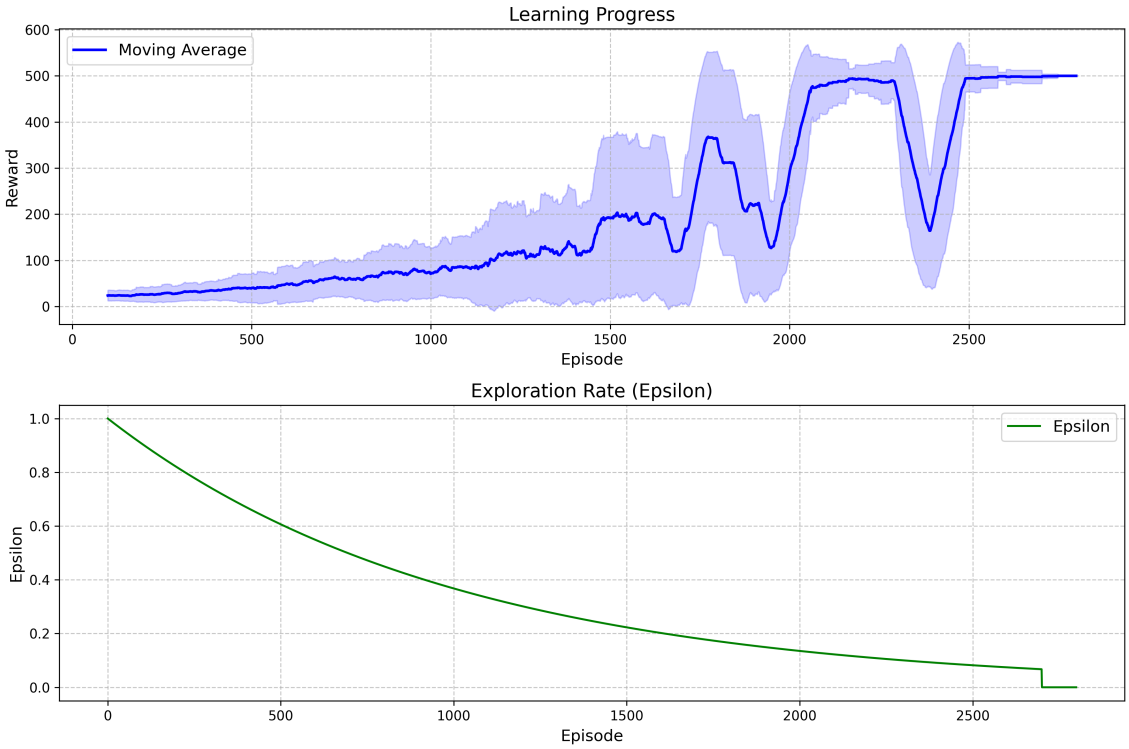
Po zaimplementowaniu wszystkich niezbędnych elementów, agent rozpoczął skuteczne uczenie się rozwiązywanego zadania.

Wizualizacja rezultatów

Zamiast prezentować surowe wyniki treningu, zastosowano uśrednianie wyników z użyciem: (1) średniej kroczącej oraz (2) wielokrotnych przebiegów eksperymentu, obliczając odchylenie standardowe dla każdej wartości.

Poniżej przedstawiono wykresy obrazujące efekty działania agenta. Górny wykres prezentuje średnią uzyskanych nagród w funkcji numeru epizodu, z zaznaczonym korytarzem odchylenia standardowego. Widzimy, że agent stopniowo poprawia swoje wyniki, osiągając stabilną wartość nagrody równą 500 po ok. 2400 epizodach.

Dolny wykres przedstawia wartość współczynnika eksploracji epsilon, który maleje w miarę trwania uczenia – na początku agent eksploruje środowisko, a następnie stopniowo przechodzi do działań opartych na wiedzy.



Rysunek 1: Postępy w uczeniu się agenta (góra) oraz zmniejszanie eksploracji ε (dół).

3 Eksperyment z zestawami parametrów

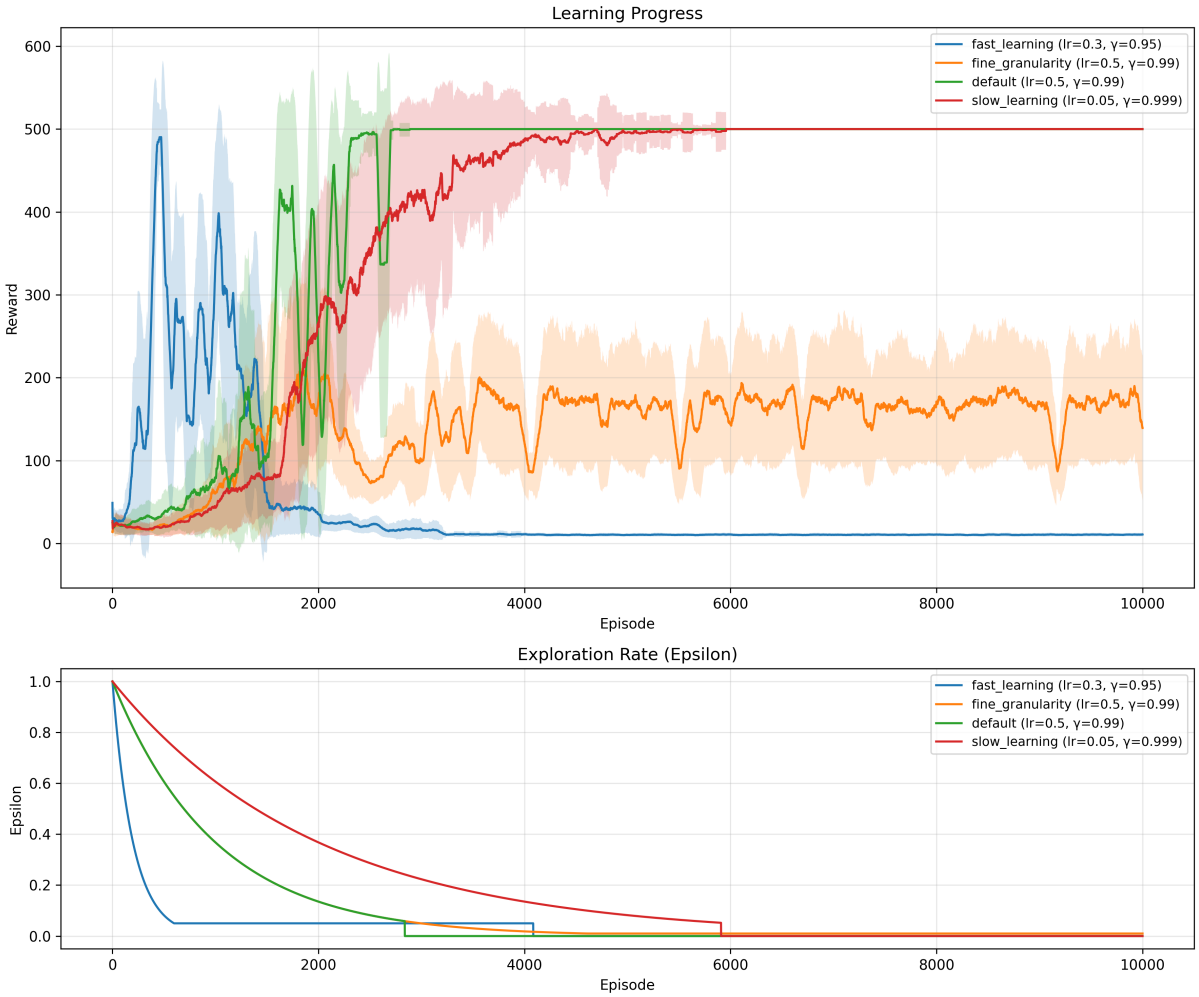
W tym zadaniu przeprowadziłem eksperyment mający na celu porównanie wpływu różnych zestawów parametrów na efektywność i dynamikę uczenia się algorytmu Q-Learning. Przetestowano cztery predefiniowane konfiguracje parametrów:

- **default** – domyślne wartości parametrów, takie jak użyte w zadaniu 1 (learning rate $\alpha = 0{,}5$, discount factor $\gamma = 0{,}99$, liczba kubełków: (1, 1, 6, 12)),
- **fast_learning** – szybkie uczenie się (większy współczynnik uczenia, niższy γ, szybki spadek ε, mniejsza liczba kubełków),
- **slow_learning** – bardzo ostrożne uczenie się (niski współczynnik uczenia α, wysoki współczynnik dyskontowy γ, wolniejszy spadek ε),
- **fine_granularity** – większa rozdzielczość reprezentacji stanów (większa liczba kubełków w procesie dyskretyzacji, przy zachowaniu pozostałych parametrów na poziomie domyślnym).

Każdy przypadek został przetestowany w osobnej sesji uczenia, a wyniki wizualizowano przy użyciu technik opisanych w Zadaniu 1 — tj. średnich kroczących oraz obliczonych odchyleń standardowych. W celu minimalizacji efektów losowości, każdy przebieg wykonywano z zastosowaniem tej samej logiki zatrzymania treningu oraz wygładzania wyników.

Wyniki eksperymentu

Na poniższym wykresie pokazano zależność nagrody od liczby epizodów (góra) oraz ewolucję wartości współczynnika eksploracji ε (dół) dla każdego z zestawów parametrów.



Rysunek 2: Porównanie różnych konfiguracji parametrów: przebieg uczenia się (góra) oraz malejąca eksploracja ε (dół).

Analiza i porównanie

Zaobserwowano wyraźne różnice w przebiegu procesu uczenia się pomiędzy poszczególnymi konfiguracjami:

- **fast_learning** – bardzo szybka eksploracja oraz gwałtowny spadek ε spowodowały, że agent zbyt wcześnie przeszedł do fazy eksploatacji. W rezultacie nie zdołał wystarczająco dobrze poznać środowiska, a jego wyniki były niskie i niestabilne.
- **fine_granularity** – zwiększona liczba kubełków znacznie zwiększyła rozmiar przestrzeni stanów, co spowodowało rozproszenie wiedzy w tablicy Q. Choć zachowano umiarkowane parametry uczenia i eksploracji, agent nie był w stanie osiągnąć stabilnej i wysokiej nagrody.
- **default** – domyślny zestaw parametrów zapewnił dobrą równowagę pomiędzy eksploracją a eksploatacją. Algorytm stosunkowo szybko osiągnął maksymalną wartość nagrody i poruszał się stabilnie. Ten wariant okazał się najbardziej efektywny w kontekście szybkości i jakości uczenia.

- **slow_learning** – ostrożne uczenie się, charakteryzujące się niewielką wartością α oraz wolniejszym spadkiem ε, prowadziło do bardzo stabilnych i powtarzalnych wyników. Choć konwergencja wymagała większej liczby epizodów, agent był w stanie osiągnąć optymalne zachowanie i utrzymywać je w sposób spójny.

Wnioski z eksperymentu

Z przeprowadzonej analizy wynika, że:

- zbyt szybkie tempo uczenia i eksploracji może utrudnić prawidłowe poznanie środowiska,
- zwiększenie liczby kubełków wymaga większej liczby epizodów i danych, aby zbudować efektywną reprezentację wiedzy,
- stabilne i umiarkowane wartości parametrów (default, slow_learning) sprzyjają skutecznemu i powtarzalnemu uczeniu agenta.

4 Implementacja i porównanie algorytmu SARSA

W tym zadaniu zmodyfikowano kod algorytmu Q-Learning, tak aby realizował podejście **SARSA** (State–Action–Reward–State–Action). W przeciwieństwie do Q-Learningu, SARSA używa nie maksymalnej wartości przyszłych akcji w stanie s_{t+1}, ale rzeczywistej, wykonanej przez agenta akcji a_{t+1}. Aktualizacja wiedzy agenta następuje zgodnie z następującym wzorem:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Powyższe podejście wymaga, aby w trakcie jednej iteracji przechowywać zarówno bieżącą akcję a_t, jak i następną a_{t+1}, co znacząco odróżnia je od klasycznego Q-Learningu.

Różnice koncepcyjne

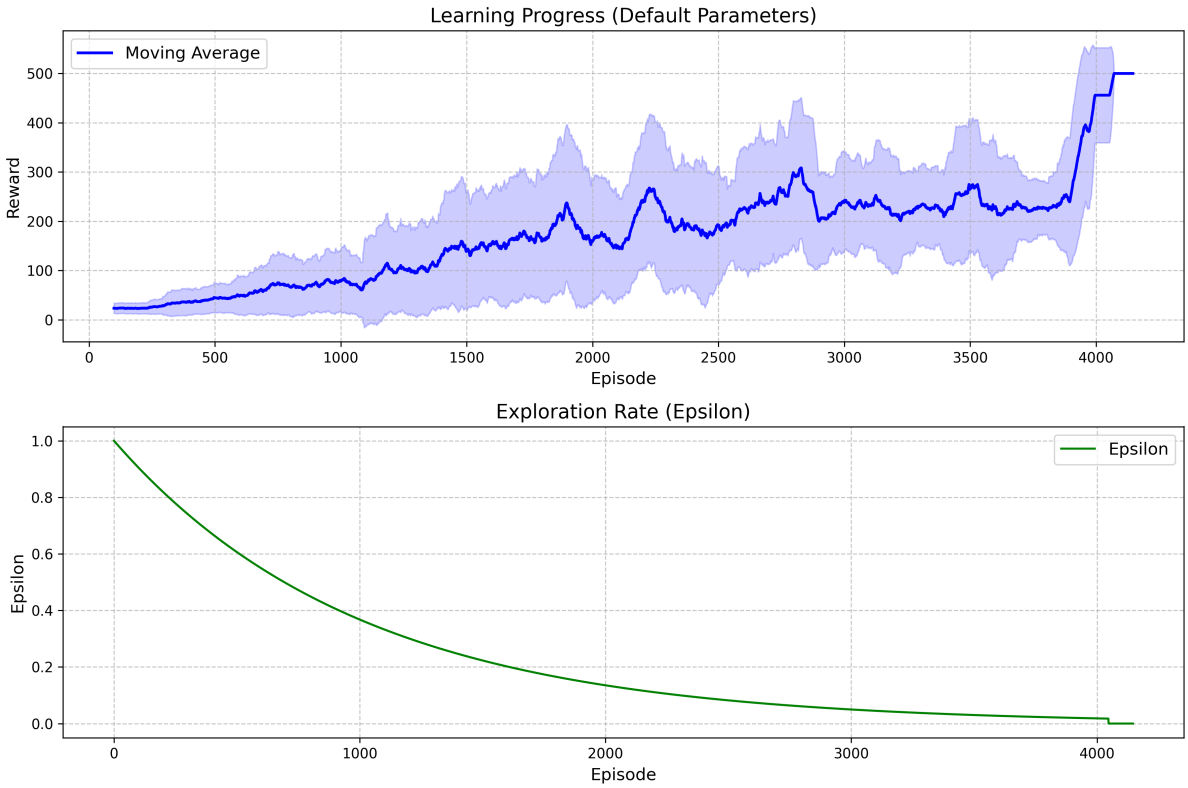
W związku z wykorzystaniem rzeczywiście wybranej akcji w kolejnym kroku, SARSA lepiej odwzorowuje zachowanie agenta w środowiskach, w których eksploracja wiąże się z ryzykiem. Przykładowo, jeśli przestrzeń stanów zawiera obszary niebezpieczne (np. „krawędź przepaści"), to SARSA – jako algorytm *on-policy* – będzie częściej ich unikać.

Eksperyment i wizualizacja

W celu porównania skuteczności Q-Learningu oraz SARSA, wykonano dwa zestawy eksperymentów:

- porównanie działania SARSA dla parametrów **domyślnych** ($\alpha = 0{,}5$, $\gamma = 0{,}99$, liczba kubełków: (1, 1, 6, 12)),
- zestawienie wyników SARSA i Q-Learningu dla tych samych zestawów parametrów: `default`, `slow_learning`, `fine_granularity`, `fast_learning`.

Poniższe dwa wykresy przedstawiają rezultaty wspomnianych eksperymentów.

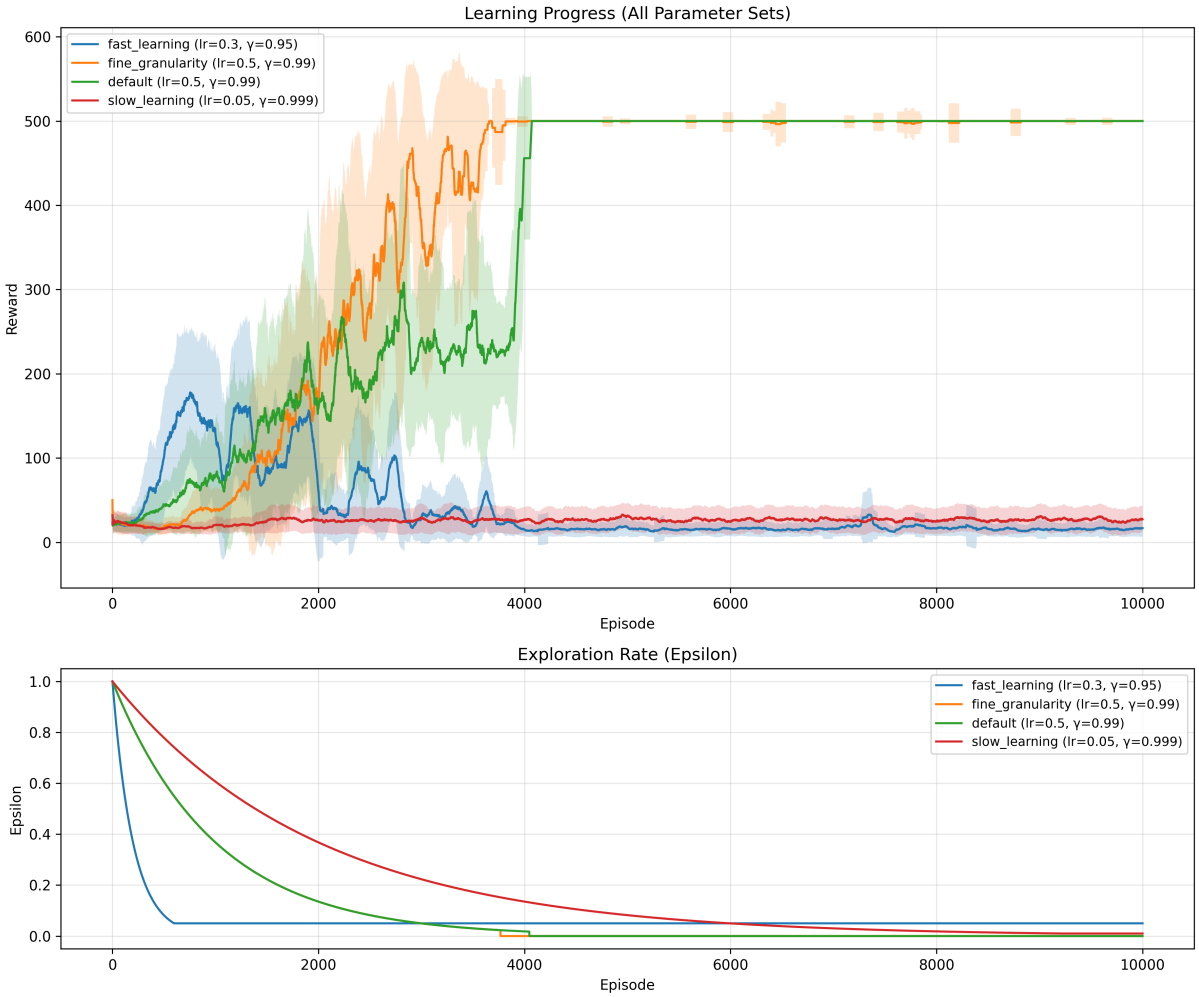


Rysunek 3: Uczenie z algorytmem SARSA przy domyślnych parametrach.

Analiza

SARSA okazała się być bardziej konserwatywna w eksploracji przestrzeni stanów, co miało wpływ na dynamikę procesu uczenia:

- Przy domyślnych parametrach, SARSA wykazywała wolniejszy wzrost wyników niż Q-Learning, jednak osiągała stabilne rezultaty i ostatecznie zbliżyła się do wartości nagrody równej 500.
- W konfiguracji `fine_granularity`, SARSA radziła sobie lepiej niż Q-Learning, który miał trudność ze stabilnym opanowaniem przestrzeni stanów przy większej liczbie kubełków.
- W przypadku `fast_learning`, oba algorytmy (SARSA i Q-Learning) osiągały niskie i niestabilne wyniki, co potwierdziło, że zbyt szybkie zmniejszenie eksploracji prowadzi do słabego uczenia.
- Konfiguracja `slow_learning`, mimo bardzo wysokiego współczynnika dyskontowego i niskiego tempa uczenia, **nie pozwoliła agentowi SARSA zbudować skutecznej polityki działania**. Pomimo dłuższego okresu eksploracji (ε), agent nie był w stanie znacząco zwiększyć otrzymywanych nagród, a wyniki pozostawały niskie przez wszystkie epizody. Świadczy to o zbyt wolnym uczeniu i zbyt zachowawczym działaniu w tej konfiguracji.



Rysunek 4: Porównanie algorytmów SARSA i Q-Learning dla różnych zestawów parametrów.

Wnioski

Algorytm SARSA, w przeciwieństwie do Q-Learningu, stosuje bardziej ostrożną strategię uczenia się, dostosowaną do faktycznie wybieranych akcji – uwzględniając możliwość eksploracji. Może to skutkować większym bezpieczeństwem, ale jednocześnie dłuższym czasem konwergencji.

Eksperyment pokazał, że:

- SARSA bywa skuteczniejsza w konfiguracjach z większą granularnością przestrzeni stanów,
- zbyt agresywna eksploracja (fast_learning) negatywnie wpływa na oba algorytmy,
- domyślne parametry zapewniają dobrą równowagę pomiędzy bezpieczeństwem a skutecznością,
- konfiguracja `slow_learning` nie doprowadziła do poprawnego wyuczenia się polityki działania — poziom nagród pozostał niski, co wskazuje na niewystarczające tempo uczenia przy zbyt dużym zaufaniu do długoterminowych przyszłych nagród.

5 Lądowanie na księżycu (LunarLander-v2)

Ostatnim etapem projektu było zastosowanie algorytmu SARSA do rozwiązania trudniejszego i bardziej realistycznego środowiska symulacyjnego – **LunarLander-v2** z biblioteki `OpenAI Gym`. Symulacja ta polega na bezpiecznym sprowadzeniu lądownika księżycowego na powierzchnię przy zachowaniu równowagi, właściwej orientacji i zużyciu jak najmniejszej ilości paliwa.

Charakterystyka środowiska

Stan środowiska LunarLander-v2 reprezentowany jest przez 8 zmiennych:

- pozycja w osi X i Y,
- prędkości w obu kierunkach,
- kąt nachylenia i prędkość kątowa kadłuba,
- oraz dwa sygnały logiczne (czy lewa/prawa noga lądownika dotyka powierzchni).

Dostępne są cztery akcje dyskretne:

- 0 – brak działania,
- 1 – strumień z lewej dyszy,
- 2 – strumień z głównej (dolnej) dyszy,
- 3 – strumień z prawej dyszy.

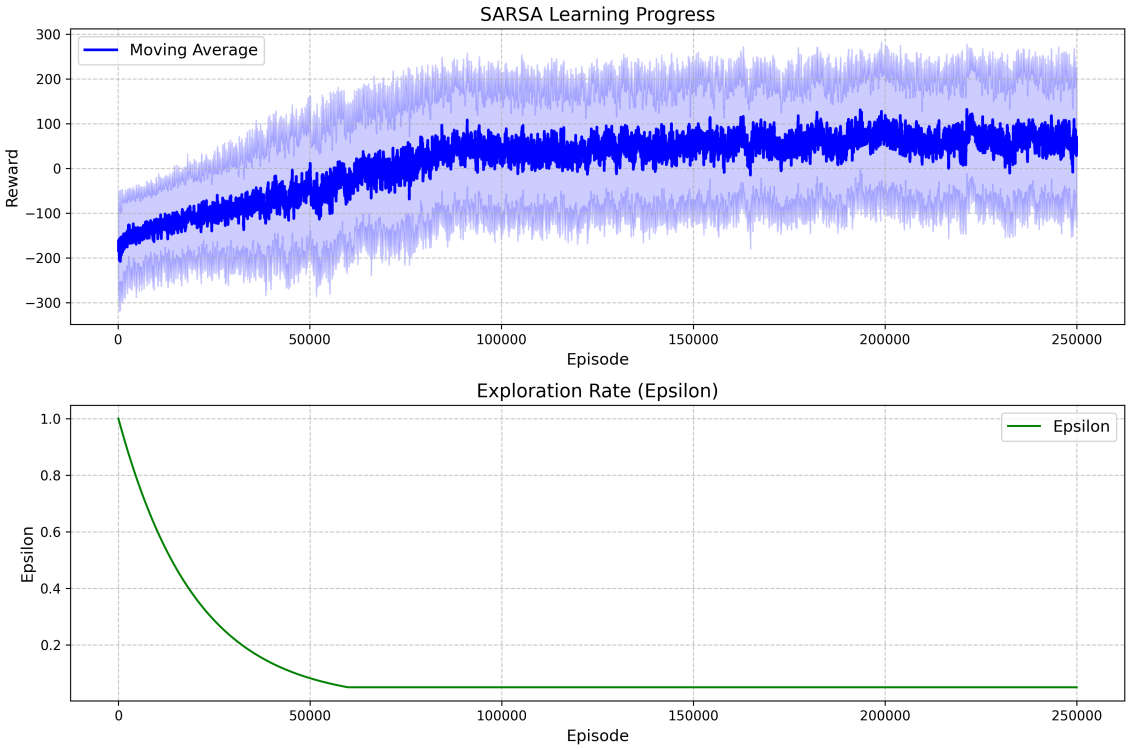
Implementacja i parametry

Do sterowania agentem zastosowano SARSA z reprezentacją wartości-Q w postaci tablicy wielowymiarowej. Ze względu na ciągły charakter przestrzeni obserwacji, zrealizowano dyskretyzację każdego z ośmiu wymiarów do ustalonej liczby kubełków:

- pozycja/prędkości: po 12 kubełków na wymiar,
- nogi lądownika (wartości logiczne): po 2 kubełki.

Parametry nauki zostały dobrane eksperymentalnie:

- `learning rate` $\alpha = 0{,}1$, malejące do 0,01,
- `epsilon-greedy` $\varepsilon = 1{,}0$ z redukcją do 0,05,
- `discount factor` $\gamma = 0{,}99$,
- czas trwania treningu: `250 000 epizodów`.



Rysunek 5: Uczenie SARSA w środowisku LunarLander-v2: postęp nagród oraz redukcja eksploracji.

Wyniki i przebieg uczenia

Poniżej przedstawiono uzyskane wyniki w postaci nagród uśrednionych krocząco (góra) oraz wartość eksploracji ε w czasie (dół).

Wyraźnie widoczna jest stopniowa poprawa jakości działań agenta – średnia nagród wzrosła z poziomów silnie ujemnych (poniżej -200) do wartości dodatnich (około 50–100) w końcowej fazie treningu. Warto podkreślić, że:

- problem był znacznie bardziej wymagający niż poprzednie środowiska (CartPole),
- przestrzeń stanów miała znacznie większy wymiar i zakres wartości,
- uczenie było wyraźnie wolniejsze i wymagało kilkuset tysięcy prób.

Choć agent nie osiągał idealnych lądowań w każdym epizodzie, to nauczył się unikać scenariuszy prowadzących do ekstremalnych kar (np. twarde upadki). Udawało się okazjonalnie uzyskać nagrody przekraczające 200, co świadczy o znaczącym postępie w nauce.

Wnioski

Uczenie lądownika księżycowego z użyciem SARSA jest możliwe, ale obarczone wieloma trudnościami wynikającymi z wysokowymiarowego i chaotycznego środowiska. Kluczowe obserwacje to:

- SARSA skutecznie uczy się unikać największych kar, koncentrując się na działaniach bezpiecznych,

- model bazujący na dyskretyzacji wymaga długiego czasu uczenia, ale może przynosić sensowne efekty nawet w trudniejszych zadaniach,
- dalsze ulepszenia (np. zmiana reprezentacji Q lub użycie funkcji aproksymujących) mogłyby znacząco przyspieszyć naukę oraz zwiększyć jakość końcowego rozwiązania.

6 Wnioski

W ramach projektu zaimplementowano i przeanalizowano działanie algorytmów Q-Learning i SARSA w różnych środowiskach. W pierwszym zadaniu potwierdzono skuteczność Q-Learningu w zadaniu równoważenia kijka – przy odpowiednim doborze parametrów agent osiągał maksymalne nagrody. Kluczowe okazały się tu dyskretyzacja obserwacji oraz dobrze dobrana strategia eksploracji.

Drugie zadanie wykazało, że zrównoważone parametry (np. default) prowadzą do najlepszych wyników. Zbyt szybka eksploracja lub zbyt duża szczegółowość stanu obniżają skuteczność uczenia, a nadmierne spowolnienie (slow_learning) uniemożliwia konwergencję.

Trzecie zadanie, dotyczące SARSA, pokazało, że algorytm ten może lepiej unikać ryzykownych zachowań, ale uczy się wolniej. W niektórych konfiguracjach działał równie dobrze lub lepiej niż Q-Learning, szczególnie w sytuacjach wymagających ostrożności.

W ostatnim zadaniu zastosowano SARSA do trudnego środowiska LunarLander. Mimo wysokiej złożoności problemu i prostoty użytej metody, agent stopniowo poprawiał swoje działania i zaczął uzyskiwać dodatnie nagrody, unikając skrajnych kar.

Reasumując, klasyczne metody oparte na wartościach Q sprawdzają się w wielu zadaniach, jednak ich skuteczność silnie zależy od doboru parametrów i jakości reprezentacji stanu. W bardziej złożonych środowiskach warto rozważyć zastosowanie metod funkcjonalnych, np. sieci neuronowych.