Uczenie ze Wzmocnieniem

Implementacja Q-learning (Taxi-v3) oraz Deep Q-Network "from scratch"(LunarLander-v3)

Marcin Przybylski

2 maja 2025

Spis treści

1 Wstęp

Niniejszy dokument stanowi sprawozdanie z realizacji projektu dotyczącego uczenia ze wzmocnieniem (Reinforcement Learning - RL). Celem projektu było praktyczne zapoznanie się z podstawowymi koncepcjami RL poprzez implementację, trening i analizę dwóch popularnych algorytmów na klasycznych środowiskach z biblioteki Gymnasium (następcy OpenAI Gym).

Projekt został podzielony na dwie główne części:

1. **Rozwiązanie środowiska Taxi-v3 przy użyciu algorytmu Q-learning:** Środowisko to charakteryzuje się dyskretną przestrzenią stanów i akcji, co czyni je idealnym kandydatem do zastosowania tablicowego algorytmu Q-learning. Agent (taksówka) uczył się optymalnej strategii odbierania i dostarczania pasażera.
2. **Rozwiązanie środowiska LunarLander-v3 przy użyciu algorytmu Deep Q-Network (DQN) implementowanego "from scratch":** Środowisko lądowania na Księżycu posiada ciągłą przestrzeń stanów, co uniemożliwia użycie prostego Q-learningu. W tym przypadku zaimplementowano algorytm DQN, w którym funkcja wartości Q jest aproksymowana przez sieć neuronową. Implementacja została wykonana od podstaw przy użyciu biblioteki NumPy, aby lepiej zrozumieć wewnętrzne mechanizmy algorytmu, w tym koncepcje takie jak pamięć powtórek (Experience Replay) i sieć docelowa (Target Network).

Prace zostały zrealizowane w środowisku Google Colaboratory, wykorzystując standardowe biblioteki języka Python, takie jak ‘gymnasium‘, ‘numpy‘, ‘matplotlib‘, ‘PIL‘ i ‘imageio‘. Projekt obejmował etapy przygotowania i eksploracji środowisk, implementacji algorytmów, przeprowadzenia procesu treningu agentów oraz ewaluacji ich działania za pomocą odpowiednich metryk i wizualizacji.

Szczegółowe instrukcje do projektu oraz kod źródłowy notatnika Colab są dostępne pod poniższymi linkami:

- Link do instrukcji: Instrukcje Projektu
- Link do notatnika roboczego Google Colab: Colab Notebook

Sprawozdanie przedstawia metodykę, opis zaimplementowanych algorytmów, przebieg eksperymentów, uzyskane wyniki wraz z ich interpretacją oraz wnioski końcowe.

2 Metodologia

Proces realizacji projektu obejmował następujące kroki:

2.1 Przygotowanie Środowisk

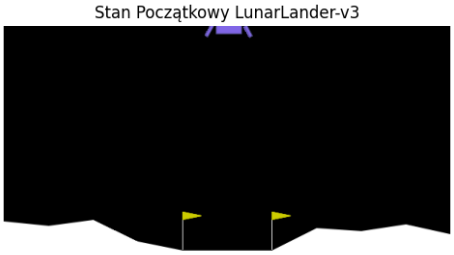
Wykorzystano dwa środowiska z biblioteki `gymnasium`:

- **Taxi-v3:** Środowisko z dyskretną przestrzenią stanów (500) i akcji (6). Stan koduje pozycję taksówki (siatka 5x5), lokalizację pasażera (4 dedykowane + w taksówce) i cel podróży (4 dedykowane). Akcje obejmują ruchy w czterech kierunkach, podniesienie i wysadzenie pasażera. Środowisko zostało zainicjalizowane z

`render_mode='rgb_array'` w celu umożliwienia wizualizacji. Przykładowy stan początkowy przedstawiono na Rysunku 1a.

- **LunarLander-v3:** Środowisko z ciągłą, 8-wymiarową przestrzenią stanów (pozycja x, y, prędkość x, y, kąt, prędkość kątowa, kontakt nogi lewej, kontakt nogi prawej) i dyskretną przestrzenią 4 akcji (nic, odpalenie lewego silnika, głównego, prawego). Środowisko również zainicjalizowano z `render_mode='rgb_array'`. Przykładowy stan początkowy przedstawiono na Rysunku 1b.

(a) Stan początkowy środowiska Taxi-v3.



(b) Stan początkowy środowiska LunarLander-v3.

Rysunek 1: Wizualizacje stanów początkowych użytych środowisk.

Dla obu środowisk przeprowadzono wstępną eksplorację, aby zrozumieć ich dynamikę, system nagród i warunki zakończenia epizodu. Sprawdzono również działanie agenta wykonującego losowe akcje jako punkt odniesienia.

2.2 Implementacja Algorytmów

2.2.1 Q-learning (dla Taxi-v3)

Zaimplementowano standardowy algorytm Q-learning. Wartości Q dla wszystkich par stan-akcja przechowywano w tablicy NumPy o wymiarach (500, 6), zainicjalizowanej zerami. Do eksploracji wykorzystano strategię ϵ-greedy, gdzie ϵ (prawdopodobieństwo wyboru losowej akcji) liniowo zanikało od 1.0 do 0.01 w trakcie treningu. Aktualizacja wartości Q odbywała się zgodnie ze wzorem:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right]$$

gdzie α to współczynnik uczenia, γ to współczynnik dyskontowania, S to bieżący stan, A to wykonana akcja, R to otrzymana nagroda, a S' to następny stan.

2.2.2 Deep Q-Network (DQN) (dla LunarLander-v3)

Z uwagi na ciągłą przestrzeń stanów, dla środowiska LunarLander zaimplementowano DQN "from scratch". Kluczowe komponenty:

- **Sieć neuronowa:** Zbudowana przy użyciu NumPy, składająca się z warstwy wejściowej (8 neuronów), dwóch warstw ukrytych z aktywacją ReLU (256 i 128 neuronów) oraz warstwy wyjściowej liniowej (4 neurony - po jednym dla każdej akcji). Wagi inicjalizowano metodami He (dla ReLU) i Glorot/Xavier (dla liniowej).

- **Pamięć powtórek (Experience Replay):** Bufor typu `deque` o rozmiarze 100 000, przechowujący krotki $(S, A, R, S', \text{done})$.
- **Sieć docelowa (Target Network):** Osobna kopia sieci neuronowej, której wagi były okresowo synchronizowane z wagami sieci głównej (co 1000 kroków). Służyła do stabilizacji obliczeń wartości docelowych Q.
- **Aktualizacja (Trening):** Co 4 kroki losowano minibatch (rozmiar 64) z pamięci powtórek. Obliczano docelowe wartości Q (y_i) oraz wartości Q przewidziane przez sieć główną ($Q(S_i, A_i; \theta)$). Błąd (zwykle MSE lub Huber loss między y_i a Q) był propagowany wstecznie w celu aktualizacji wag sieci głównej (θ). W tej implementacji zastosowano błąd $Q_{\text{current}} - Q_{\text{target\_bp}}$ z clippingiem do $[-1, 1]$.

Podobnie jak w Q-learningu, zastosowano strategię ϵ-greedy z liniowym zanikiem ϵ oraz dodatkowo zanikiem współczynnika uczenia (learning rate).

2.3 Proces Treningu

- **Q-learning (Taxi):** Trening przez 15 000 epizodów. Parametry: $\alpha = 0.1$, $\gamma = 0.9$, ϵ zanikające od 1.0 do 0.01 przez 80% epizodów.
- **DQN (LunarLander):** Trening przez 2 000 epizodów. Parametry: $\gamma = 0.99$, rozmiar batcha $= 64$, ϵ zanikające od 1.0 do 0.01 przez ≈ 600000 kroków, LR zanikające od 5×10^{-4} do 1×10^{-5} przez ≈ 600000 kroków. Sieć docelowa aktualizowana co 1000 kroków, trening sieci głównej co 4 kroki (po zebraniu 640 próbek w pamięci).

Podczas treningu monitorowano sumę nagród w każdym epizodzie oraz (dla DQN) średni loss na minibatchach.

2.4 Ewaluacja Modeli

Po zakończeniu treningu przeprowadzono ewaluację agentów:

- **Metryki ilościowe:** Obliczono średnią nagrodę uzyskaną przez agenta w ostatnich 100 epizodach treningu. Przeprowadzono również dedykowaną fazę ewaluacji (kilka epizodów bez eksploracji ϵ-greedy), obliczając średnią nagrodę, średnią liczbę kroków i odsetek pomyślnie zakończonych epizodów.
- **Wizualizacje:** Wygenerowano wykresy przedstawiające średnią kroczącą nagrody (i loss dla DQN) w funkcji epizodów treningowych. Wygenerowano animacje GIF (zapisując klatki z jednego epizodu ewaluacyjnego) pokazujące działanie nauczonych agentów w ich środowiskach.

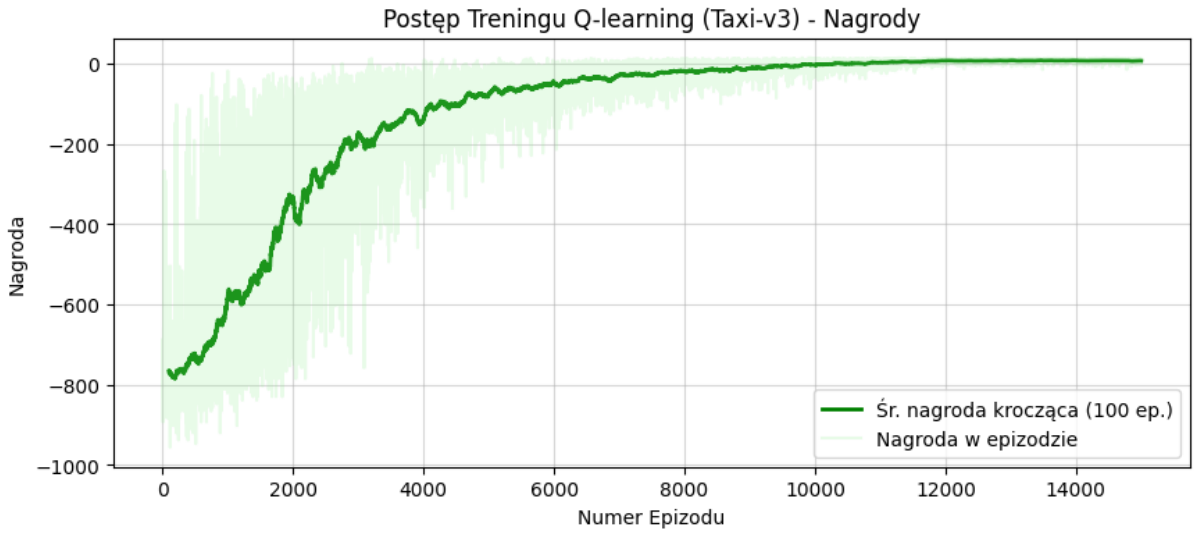
3 Wyniki i Interpretacja

3.1 Q-learning (Taxi-v3)

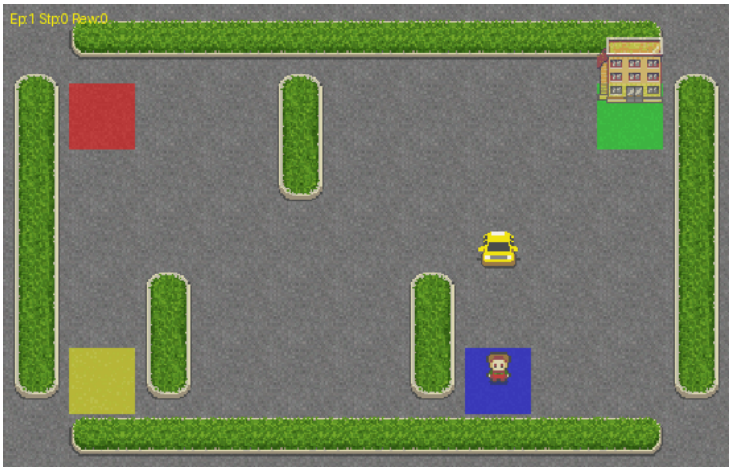
Agent trenowany algorytmem Q-learning osiągnął bardzo dobre wyniki.

- **Wyniki treningu:** Po 15 000 epizodów, średnia nagroda w ostatnich 100 epizodach ustabilizowała się na poziomie **7.12**. Cały trening zajął około 37 sekund.

- **Wykres uczenia:** Rysunek 2 pokazuje wyraźny wzrost średniej nagrody kroczącej, która szybko osiąga wartości dodatnie i stabilizuje się, wskazując na efektywną naukę polityki.
- **Wyniki ewaluacji:** W 3 epizodach ewaluacyjnych bez eksploracji agent osiągnął średnią nagrodę **10.33** i pomyślnie ukończył wszystkie epizody (100% sukcesu), potrzebując średnio tylko **10.67** kroków. Potwierdza to, że agent nauczył się optymalnej lub bliskiej optymalnej strategii.
- **Wizualizacja działania:** Animacja GIF (Rysunek 3) pokazuje agenta sprawnie nawigującego po planszy i wykonującego zadanie.



Rysunek 2: Krzywa uczenia (średnia krocząca nagrody) dla Q-learning w środowisku Taxi-v3.



Rysunek 3: Przykładowa klatka z animacji GIF pokazującej działanie nauczonego agenta Taxi-v3. Pełna animacja dostępna w wynikach Colab.

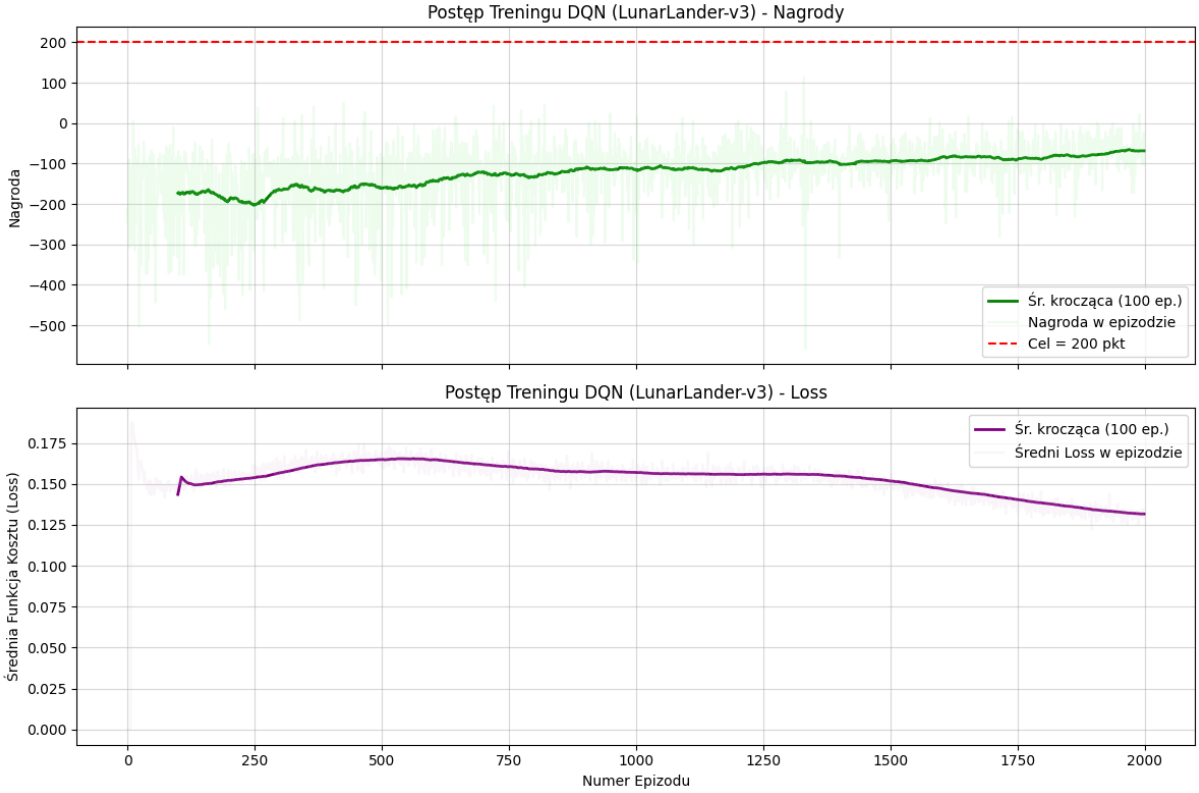
Interpretacja: Algorytm Q-learning doskonale poradził sobie ze środowiskiem Taxi-v3. Dyskretna i stosunkowo niewielka przestrzeń stanów pozwoliła na efektywne wypełnienie

tablicy Q i znalezienie optymalnej polityki w rozsądnym czasie treningu. Wyniki ewaluacyjne potwierdzają wysoką skuteczność nauczonego agenta.

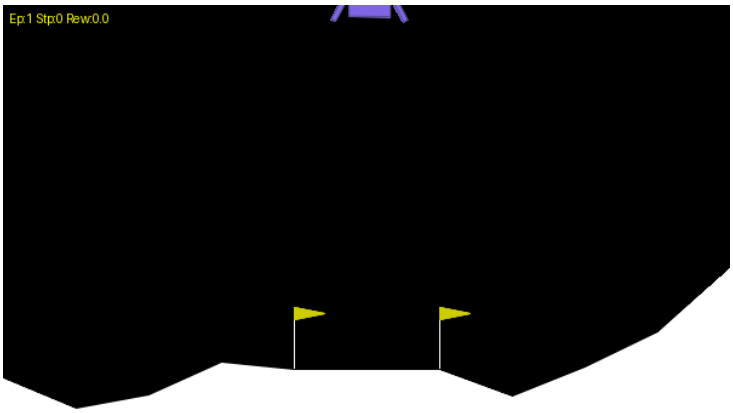
3.2 Deep Q-Network (LunarLander-v3)

Agent trenowany algorytmem DQN "from scratch"wykazał postępy w nauce, ale nie osiągnął jeszcze pełnego rozwiązania problemu w ramach przeprowadzonego treningu.

- **Wyniki treningu:** Po 2 000 epizodów (ok. 209 tys. kroków), średnia nagroda w ostatnich 100 epizodach wyniosła **-68.32**. Trening trwał około 578 sekund (prawie 10 minut).
- **Wykresy uczenia:** Rysunek 4 pokazuje tendencję wzrostową średniej nagrody kroczącej (choć pozostaje ona ujemna) oraz tendencję spadkową średniego lossu. Wskazuje to, że agent uczy się i poprawia swoją politykę, ale proces jest znacznie wolniejszy niż w przypadku Q-learningu dla Taxi.
- **Wyniki ewaluacji:** W 3 epizodach ewaluacyjnych bez eksploracji agent osiągnął średnią nagrodę **-132.40** i nie udało mu się pomyślnie wylądować ani razu (0% sukcesu przy progu >190 pkt). Wyniki te są spójne z fazą treningu - agent jest lepszy niż losowy, ale daleki od optymalnego rozwiązania.
- **Wizualizacja działania:** Animacja GIF (Rysunek 5) pokazuje próby lądowania agenta. Choć nie są one w pełni udane, często widać próby stabilizacji i kontrolowanego opadania, co sugeruje, że sieć nauczyła się pewnych podstawowych zależności.



Rysunek 4: Krzywe uczenia (średnia krocząca nagrody i loss) dla DQN w środowisku LunarLander-v3.



Rysunek 5: Przykładowa klatka z animacji GIF pokazującej działanie (próbę lądowania) nauczonego agenta LunarLander-v3. Pełna animacja dostępna w wynikach Colab.

Interpretacja: Implementacja DQN "from scratch"działała poprawnie i agent wykazywał postępy w nauce. Jednak środowisko LunarLander jest znacznie bardziej złożone niż Taxi, a osiągnięcie dobrych wyników (średnia nagroda > 200) wymaga zazwyczaj znacznie dłuższego treningu (więcej epizodów/kroków) i potencjalnie dalszego strojenia hiperparametrów (np. harmonogramu zaniku ϵ i LR, rozmiaru sieci, częstotliwości aktualizacji Target). Wynik -68.32 po 2000 epizodów jest typowym wynikiem dla początkowej fazy uczenia w tym środowisku. Implementacja "from scratch"pozwoliła jednak na zrozumienie kluczowych komponentów algorytmu DQN.

4 Podsumowanie i Wnioski

Projekt ten obejmował implementację i porównanie dwóch fundamentalnych algorytmów uczenia ze wzmocnieniem: Q-learningu dla środowiska z dyskretną przestrzenią stanów (Taxi-v3) oraz Deep Q-Network (DQN) "from scratch"dla środowiska z ciągłą przestrzenią stanów (LunarLander-v3).

Główne wnioski płynące z projektu to:

1. **Skuteczność Q-learningu w środowiskach dyskretnych:** Algorytm Q-learning okazał się bardzo efektywny i szybko zbieżny do optymalnej polityki w środowisku Taxi-v3, co potwierdza jego przydatność dla problemów o stosunkowo niewielkiej, dyskretnej przestrzeni stanów.
2. **Konieczność i złożoność DQN dla środowisk ciągłych:** W przypadku środowiska LunarLander-v3 z ciągłą przestrzenią stanów, konieczne było zastosowanie aproksymacji funkcji wartości za pomocą sieci neuronowej (DQN). Choć zaimplementowany agent DQN wykazywał postępy w nauce, osiągnięcie wysokiej wydajności wymaga znacznie więcej czasu treningu i potencjalnie dalszej optymalizacji w porównaniu do Q-learningu w prostszym środowisku.
3. **Wyzwania implementacji "from scratch":** Implementacja DQN od podstaw przy użyciu NumPy, choć bardzo pouczająca w kontekście zrozumienia działania

algorytmu (pamięć powtórek, sieć docelowa, proces aktualizacji), jest bardziej złożona i podatna na błędy niż korzystanie z gotowych, wysokopoziomowych bibliotek RL.

4. **Znaczenie hiperparametrów i czasu treningu:** Wyniki dla DQN podkreślają kluczową rolę odpowiedniego doboru hiperparametrów (współczynnik uczenia, harmonogram zaniku epsilon, rozmiar pamięci, architektura sieci) oraz wystarczająco długiego czasu treningu, zwłaszcza w bardziej złożonych środowiskach.

5. **Praktyczne aspekty pracy z Gymnasium:** Projekt wymagał dostosowania do aktualnego API biblioteki 'gymnasium' oraz obsługi zależności (np. Box2D dla LunarLander) i konfiguracji środowiska (np. wirtualny wyświetlacz w Colab).