Zaawansowane Metody Uczenia Maszynowego - projekt zespołowy

Jakub Jasiński
Oskar Rozwadowski
Oskar Wyłucki

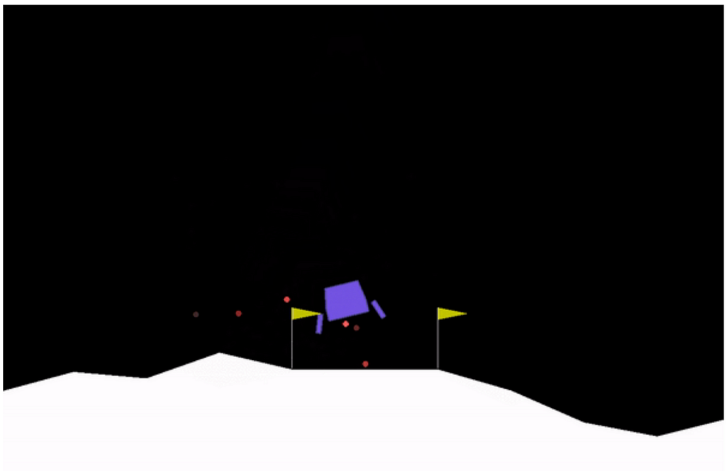
27 Styczeń 2025

Wstęp

Raport prezentuje wyniki projektu zespołowego polegającego na wykorzystaniu dwóch algorytmów uczenia przez wzmacnianie do rozwiązania problemu optymalizacji. Zostanie przedstawiony szczegółowy opis środowiska, opis działania algorytmów, proces trenowania modeli, wyniki oraz wnioski.

1 Opis Środowiska

Środowiskiem użytym w projekcie jest Lunar Lander[1] - klasyczny problem optymalizacji trajektorii rakiety. Zadaniem jest sterowanie wirtualnym lądownikiem wyposażonym w główny silnik oraz dwa boczne dopalacze, które pozwalają na manewrowanie obiektem. Celem jest bezpieczne posadzenie lądownika w wyznaczonym punkcie oznaczonym w środowisku graficznym Rys. 3 dwoma żółtymi flagami. Gra uwzględnia realizm fizyczny, w tym grawitację, wiatr oraz turbulencje, co czyni ją idealnym środowiskiem do testowania algorytmów uczenia ze wzmocnieniem.



Rysunek 1: Środowisko Lunar Lander

Istnieją dwie wersje środowiska: dyskretna lub ciągła. Paliwo jest nieskończone, więc agent może nauczyć się latać by potem wylądować przy pierwszej próbie.

Można wyróżnić cztery dyskretne akcje, które agent może wykonać:

- 0: nic nie robić
- 1: odpalenie lewego silnika
- 2: odpalenie środkowego silnika
- 3: odpalenie prawego silnika

Stan środowiska przechowywany jest w 8 wymiarowym wektorze, który zawiera: współrzędne lądownika w osiach x i y, jego prędkości liniowe w osiach x i y, kąt nachylenia, prędkość kątowa oraz dwie zmienne logiczne definiujące, czy każda z nóg ma kontakt z podłożem.

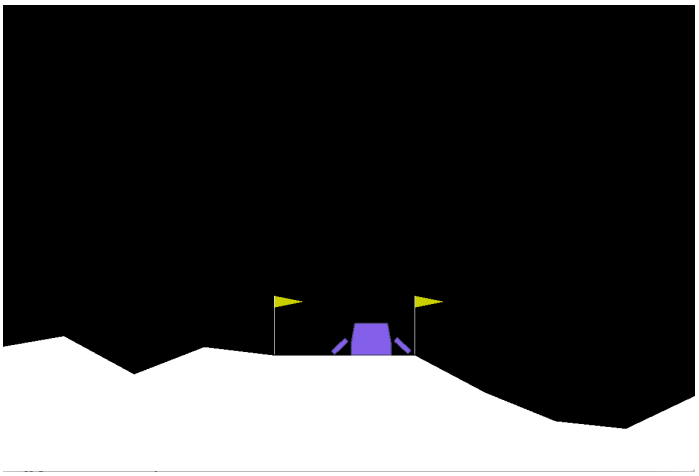
Lądownik rozpoczyna swoją podróż u góry ekranu po środku z wylosowaną losowo siłą z punktem zaczepienia w środku jego ciężkości. Następnie w każdym kroku otrzymywana jest nagroda/kara. Całkowity wynik obliczany jest po zsumowaniu wszystkich nagród/kar. Dla każdego kroku punkty obliczane są następująco:

- Nagroda zwiększa się lub zmniejsza w zależności od tego, jak blisko lądownik znajduje się platformy lądowania.
- Nagroda zwiększa się lub zmniejsza w zależności od tego, jak wolno lub szybko porusza się lądownik.
- Nagroda zmniejsza się, jeśli lądownik jest bardziej przechylony w bok.
- Nagroda zwiększa się o 10 punktów za każdą nogę, która ma kontakt z podłożem.
- Nagroda zmniejsza się o 0,03 punktu za każdą klatkę, gdy pracuje silnik boczny.
- Nagroda zmniejsza się o 0,3 punktu za każdą klatkę, gdy pracuje główny silnik.

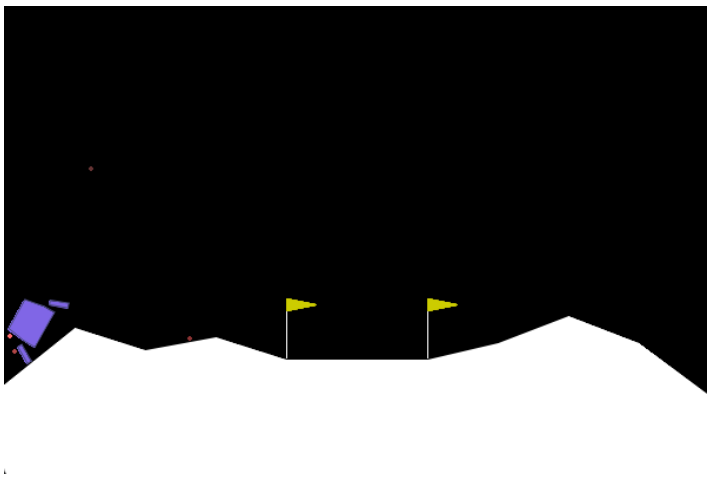
Cała symulacja otrzymuje dodatkową nagrodę: -100 punktów za rozbicie lądownika lub +100 punktów za bezpieczne lądowanie. Symulacja jest uznawana za zakończoną pozytywnie jeśli osiągnie co najmniej 200 punktów.

Symulacja zakończy się gdy wystąpi jeden z poniższych warunków:

1. Symulacja kończy się, jeśli lądownik ulegnie rozbiciu (korpus lądownika ma kontakt z powierzchnią księżyca).
2. Symulacja kończy się, jeśli lądownik wyjdzie poza widoczny obszar (współrzędna x jest większa niż 1).
3. Symulacja kończy się, jeśli lądownik nie jest aktywny. Z dokumentacji Box2D wynika, że ciało, które nie jest aktywne, to ciało, które się nie porusza i nie koliduje z żadnym innym obiektem.



Rysunek 2: Udane lądowanie



Rysunek 3: Rozbicie lądownika

2 Opis Algorytmów

2.1 Deep Q-Learning (DQN)

Q-learning jest algorytmem opartym na dynamicznym programowaniu, który iteracyjnie wyznacza wartości funkcji $Q(s, a)$. Równanie Bellmana dla Q-learningu jest definiowane jako:

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

gdzie:

- s — bieżący stan,
- a — wykonane działanie,
- r — otrzymana nagroda,

- s' — następny stan,
- $\gamma \in [0, 1]$ — współczynnik dyskontowania,
- $\alpha \in (0, 1]$ — współczynnik uczenia.

W przypadku DQN, funkcja $Q(s, a)$ jest aproksymowana za pomocą sieci neuronowej $Q_\theta(s, a)$, gdzie θ oznacza parametry sieci.

Komponenty

Aby skutecznie eksplorować środowisko, DQN wykorzystuje strategię epsilon-greedy. Z prawdopodobieństwem ϵ agent wybiera losowe działanie (eksploracja), a z prawdopodobieństwem $1 - \epsilon$ wybiera działanie maksymalizujące wartość $Q(s, a)$ (eksploatacja). Wartość ϵ jest stopniowo zmniejszana podczas treningu.

Kluczowe elementy algorytmu

Replay buffer. Aby unikać korelacji między kolejnymi próbkami, agent przechowuje swoje doświadczenia $(s, a, r, s', \text{done})$ w pamięci doświadczeń $\mathcal{D}$. Mini-batche próbek są losowo pobierane z tego bufora podczas procesu uczenia, co poprawia stabilność i wydajność algorytmu. Replay buffer rozbija zależności czasowe w danych, co zwiększa stabilność treningu [2].

Sieć docelowa (target network). Sieć docelowa działa jako mechanizm stabilizacji poprzez okresową synchronizację z siecią główną, co minimalizuje błędy sprzężenia zwrotnego [2]. Cel dla danej próbki jest obliczany jako:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a').$$

Funkcja straty. Główną sieć Q trenuje się poprzez minimalizację błędu średniokwadratowego (MSE) między wartością przewidywaną $Q_\theta(s, a)$ a wartością celu y:

$$L(\theta) = E_{(s,a,r,s') \sim \mathcal{D}} \left[\left(y - Q_\theta(s, a) \right)^2 \right].$$

Przebieg algorytmu

Algorytm DQN można opisać następująco:

1. Zainicjalizuj sieć Q Q_θ z losowymi wagami.
2. Skopiuj wagi do sieci docelowej Q_{θ^-}: $Q_{\theta^-} \leftarrow Q_\theta$.
3. Utwórz pusty replay buffer $\mathcal{D}$.
4. **Dla** każdego kroku treningowego **wykonaj**:
   (a) Wybierz działanie a zgodnie z polityką epsilon-greedy.
   (b) Wykonaj działanie a w środowisku, otrzymaj nagrodę r i następny stan s'.
   (c) Zapisz doświadczenie (s, a, r, s') w replay buffer $\mathcal{D}$.
   (d) Jeśli bufor zawiera wystarczającą liczbę próbek:

i. Pobierz losowy mini-batch próbek z $\mathcal{D}$.

ii. Oblicz cel y dla każdej próbki.

iii. Zaktualizuj parametry θ, minimalizując funkcję straty $L(\theta)$.

(e) Zaktualizuj sieć docelową: $Q_{\theta^-} \leftarrow Q_\theta$.

Zalety

Algorytm Deep Q-Network (DQN) wnosi znaczący wkład w dziedzinę uczenia wzmacniającego dzięki kilku istotnym zaletom:

- **Efektywna aproksymacja funkcji Q:** Dzięki wykorzystaniu głębokich sieci neuronowych algorytm jest w stanie aproksymować funkcję $Q(s, a)$ w wysokowymiarowych przestrzeniach stanów, co było trudne lub wręcz niemożliwe przy użyciu tradycyjnych metod.
- **Rozwiązanie problemu korelacji próbek:** Wprowadzenie *replay buffer* pozwala na losowe pobieranie próbek doświadczeń, co redukuje korelację między nimi i poprawia stabilność uczenia.
- **Stabilność uczenia:** Sieć docelowa (*target network*) wprowadza stabilność poprzez dostarczanie stałych wartości celu podczas aktualizacji wag sieci głównej.
- **Skalowalność:** Algorytm DQN może być stosowany do szerokiej gamy problemów o różnym stopniu złożoności, w tym gier wideo, systemów sterowania i innych aplikacji.

Wady

Pomimo swoich zalet, DQN posiada pewne ograniczenia, które należy uwzględnić w zastosowaniach praktycznych:

- **Wysokie wymagania obliczeniowe:** Trening sieci neuronowej w DQN wymaga znacznych zasobów obliczeniowych, zarówno pod względem mocy obliczeniowej, jak i pamięci.
- **Wrażliwość na hiperparametry:** Algorytm jest bardzo czuły na dobór hiperparametrów, takich jak współczynnik uczenia α, współczynnik dyskontowania γ, rozmiar replay buffer oraz częstotliwość aktualizacji sieci docelowej.
- **Trudności w eksploracji:** Strategia epsilon-greedy, stosowana w DQN, może być niewystarczająca w środowiskach o dużej przestrzeni akcji lub w środowiskach wymagających długoterminowego planowania.
- **Niestabilność i rozbieżności:** W niektórych przypadkach algorytm może napotkać problemy niestabilności, takie jak eksplozja wartości $Q(s, a)$, co prowadzi do rozbieżności w trakcie uczenia.

Podsumowanie

Algorytm DQN zrewolucjonizował uczenie wzmacniające, umożliwiając skuteczne działanie w środowiskach o złożonych przestrzeniach stanów. Dzięki kluczowym komponentom, takim jak replay buffer oraz sieć docelowa, DQN rozwiązuje problemy niestabilności tradycyjnego Q-learningu.

2.2 Advantage Actor Critic (A2C)

Algorytm **Advantage Actor-Critic (A2C)** jest popularnym podejściem do rozwiązywania problemów uczenia ze wzmocnieniem [3]. Łączy on dwa komponenty: *aktor* oraz *krytyk*, które współpracują w celu efektywnego uczenia się optymalnej polityki. Połączenie tych komponentów ma za zadanie ograniczenie wad tych metod podczas ich pojedynczego używania. A2C jest wersją *asynchronicznego algorytmu Advantage Actor-Critic (A3C)* uproszczoną do działania synchronicznego, co pozwala na łatwiejsze wdrożenie i stabilniejsze działanie.

Komponenty

A2C opiera się na dwóch komponentach:

- **Aktor** (ang. *actor*) – odpowiada za wybór akcji na podstawie obecnego stanu środowiska. Reprezentuje politykę $\pi(a|s)$, która jest funkcją parametrów θ. Celem aktora jest maksymalizacja skumulowanego wzmocnienia.
- **Krytyk** (ang. *critic*) – odpowiada za ocenę jakości stanu poprzez estymację funkcji wartości $V(s)$. Krytyk dostarcza aktorowi informacji zwrotnej w postaci wzmocnienia przewagi (ang. *advantage signal*).

Gradient Polityki (Aktor)

Polityka jest optymalizowana za pomocą gradientu polityki:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=0}^{N} \nabla_\theta \log \pi_\theta(a_i|s_i) \cdot A(s_i, a_i)$$

Gdzie:

- $J(\theta)$ – oczekiwany zwrot (skumulowana nagroda) w ramach polityki π_θ.
- $\pi_\theta(a|s)$ – funkcja polityki, która określa prawdopodobieństwo wyboru akcji a w stanie s.
- N – liczba próbek (trajektorii) wykorzystanych do estymacji gradientu.
- $A(s, a)$ – funkcja przewagi (ang. *advantage function*), która mierzy korzyść z wykonania akcji a w stanie s, w porównaniu do oczekiwań.
- i – indeks próbki w zbiorze danych.

Funkcja przewagi $A(s, a)$ może być obliczona jako różnica:

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

gdzie $Q(s, a)$ to wartość akcji, a $V(s)$ to wartość stanu.

Aktualizacja Funkcji Wartości (Krytyk)

Funkcja wartości stanu $V_w(s)$ jest optymalizowana za pomocą minimalizacji błędu średniokwadratowego (*Mean Squared Error, MSE*) pomiędzy przewidywaną wartością a rzeczywistymi zwrotami:

$$\nabla_w J(w) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_w \big(V_w(s_i) - Q_w(s_i, a_i)\big)^2$$

Gdzie:

- $\nabla_w J(w)$ – gradient funkcji błędu względem parametrów w krytyka.
- $V_w(s_i)$ – estymacja wartości stanu s przez krytyka z parametrami w.
- $Q_w(s_i, a_i)$ – estymacja wartości akcji a w stanie s.
- N – liczba próbek.
- i – indeks próbki w zbiorze danych.

Reguły Aktualizacji

Algorytm A2C aktualizuje parametry aktora i krytyka w następujący sposób:

Aktualizacja Aktora

Parametry aktora θ są aktualizowane zgodnie z regułą wstępu gradientowego:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t)$$

Gdzie:

- α – współczynnik uczenia (*learning rate*) aktora.
- t – krok czasowy.

Aktualizacja Krytyka

Parametry krytyka w są aktualizowane zgodnie z regułą zstępu gradientowego:

$$w_{t+1} = w_t - \beta \nabla_w J(w_t)$$

Gdzie:

- β – współczynnik uczenia (*learning rate*) krytyka.
- t – krok czasowy.

Główne Kroki Algorytmu

Algorytm A2C składa się z następujących kroków:

1. **Inicjalizacja**:
   - Zainicjalizuj sieci neuronowe dla aktora i krytyka z losowymi wagami.
   - Ustal hiperparametry: współczynnik uczenia (α), współczynnik dyskontowania (γ), liczbę epizodów i maksymalną długość epizodu.
2. **Interakcja ze środowiskiem**:
   - Rozpocznij epizod w środowisku i zainicjalizuj stan początkowy s_0.

- Dla każdego kroku t w epizodzie:
  (a) Wykonaj akcję a_t wybraną przez aktora na podstawie rozkładu $\pi(a|s_t;\theta)$.
  (b) Otrzymaj nagrodę r_t oraz następny stan s_{t+1}.
  (c) Zapisz logarytm prawdopodobieństwa akcji $\log \pi(a_t|s_t)$, nagrodę r_t oraz estymację wartości stanu $V(s_t)$.

3. **Obliczanie zwrotów i przewag**:
   - Oblicz skumulowane zwroty G_t dla każdego kroku w epizodzie:
   $$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$
   - Wyznacz przewagę (ang. *advantage*) dla każdego kroku:
   $$A_t = G_t - V(s_t)$$
4. **Aktualizacja sieci neuronowych**: Zaktualizuj wagi aktora i krytyka przy użyciu optymalizatora (np. Adam).
5. **Powtórz kroki 2–4 dla kolejnych epizodów**.

Zalety Algorytmu A2C

- **Stabilność**: Krytyk zapewnia bardziej stabilny proces uczenia dzięki dokładniejszej ocenie wartości stanu.
- **Efektywność obliczeniowa**: A2C wykorzystuje synchroniczne aktualizacje, co czyni algorytm prostszym w implementacji w porównaniu do A3C.
- **Efektywność danych**: Użycie przewagi jako sygnału wzmocnienia poprawia efektywność uczenia się.
- **Szybka zbieżność**: Jednoczesne uaktualnianie zarówno aktora jak i krytyka przyczynia się do szybkiej zbieżności podczas uczenia.

Wady Algorytmu A2C

- **Ograniczona skalowalność**: A2C jest mniej wydajny w wykorzystaniu wielu wątków w porównaniu do A3C.
- **Wymóg stabilnych hiperparametrów**: Algorytm wymaga starannego doboru hiperparametrów, takich jak γ czy α.

Podsumowanie

Algorytm A2C jest wydajnym i stabilnym podejściem do uczenia ze wzmocnieniem, które skutecznie łączy politykę aktora z estymacją wartości krytyka. Dzięki temu stanowi solidną podstawę dla bardziej zaawansowanych metod RL, takich jak PPO (Proximal Policy Optimization).

3 Trenowanie modeli

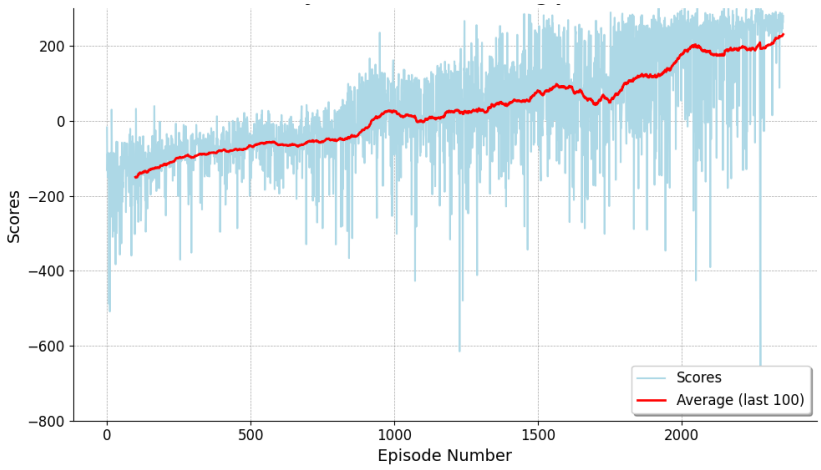
Porównanie modeli

Oba algorytmy różnią się pod względem podejścia do procesu uczenia się i podejmowania decyzji. DQN to algorytm uczenia się poza polityką, który wykorzystuje bufor odtwarzania, co pozwala mu uczyć się na podstawie zapisanych doświadczeń niezależnie od bieżącej polityki. Z drugiej strony algorytm A2C uczy się w ramach polityki, co oznacza, że aktualizacje modelu są dokonywane na podstawie bieżących interakcji z otoczeniem. Dzięki temu A2C jest stabilniejszy w procesie uczenia i powinien lepiej dostosować się do dynamicznych środowisk[4].

W środowisku Lunar Lander cechy obu algorytmów mogą być przydatne w różnych aspektach. DQN dzięki zapisie historii działań może efektywnie korzystać z doświadczeń, co sprawdza się w zadaniach z ograniczoną liczbą epizodów lub dużą zmiennością stanów. Z kolei A2C może lepiej adaptować się do zmian dynamiki lądownika. Zdolność A2C do oszczędzania zasobów i unikania nadmiernego użycia działań jak np. ograniczenie użycia silników, może prowadzić do bardziej efektywnego zarządzania zasobami, co jest istotne w zadaniach takich jak minimalizacja zużycia paliwa.

Proces trenowania

Trening obu modeli ma wspólny cel: doprowadzenie agenta do osiągnięcia średniej wyniku wynoszącej 230, co wskazuje na skuteczne rozwiązanie zadania. Oba algorytmy wykorzystują iteracyjne podejście, gdzie agent uczy się poprzez interakcję ze środowiskiem, zbieranie nagród i optymalizacje swoich parametrów. Kluczowym elementem w obu metodach jest monitorowanie wyników z ostatnich 100 epizodów, co pozwala na ocenę postępów agenta.



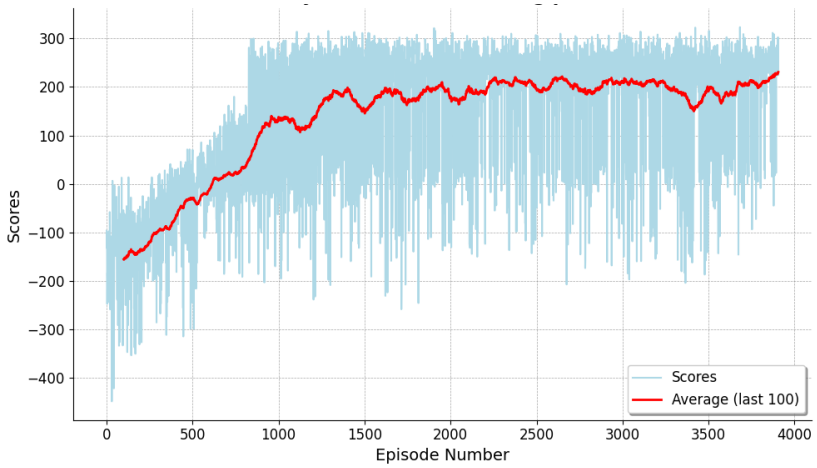
Rysunek 4: Trenowanie modelu z użyciem algorytmu DQN

Wykres 4 przedstawia wyniki treningu modelu z użyciem algorytmu DQN. Na początku treningu wyniki są bardzo niestabilne, a większość epizodów kończy się niskimi punktami, oscylującymi w przedziale od -400 do 0. Widoczny jest duży rozrzut wyników, co świadczy o intensywnej eksploracji środowiska przez model.

Czerwona linia, powoli wzrasta, co sugeruje, że agent zaczyna zdobywać podstawowe umiejętności niezbędne do poprawy wyników.

W miarę postępu treningu widoczne jest stopniowe zmniejszanie niestabilności wyników, choć wciąż pojawiają się epizody, które kończą się bardzo niskimi wynikami. Średnia (czerwona linia) zaczyna rosnąć w sposób bardziej stabilny, osiągając wartość około 0 w okolicach 1000 epizodu. To wskazuje, że model zaczyna coraz skuteczniej optymalizować swoje działania, unikając katastrof i lepiej zarządzając zasobami, takimi jak paliwo czy orientacja lądownika.

Po około 2000 epizodach średnia punktacja przekracza próg 200 punktów, co jest wymaganym kryterium sukcesu w tym środowisku. Rozrzut wyników pojedynczych epizodów pozostaje stosunkowo duży. Jest to naturalna cecha DQN, wynikająca z faktu, że model wciąż eksploruje pewne przestrzenie akcji, aby unikać lokalnych minimów.



Rysunek 5: Trenowanie modelu z użyciem algorytmu A2C

Wykres 7 przedstawia wyniki treningu modelu z użyciem algorytmu A2C. Podobnie jak w przypadku treningu DQN, wyniki (niebieska strefa) na początku treningu są niskie, z wyraźną niestabilnością. Widoczny jest duży rozrzut wyników od -400 do około 0, co wynika z intensywnej eksploracji środowiska przez agenta i poszukiwania optymalnej strategii.

W porównaniu do DQN, wyniki A2C charakteryzują się mniejszymi skokami między epizodami. Wynika to z natury algorytmu on-policy, w którym aktualizacje polityki i wartości stanu opierają się bezpośrednio na bieżących interakcjach z otoczeniem. Dzięki temu A2C jest bardziej stabilny w procesie uczenia, chociaż wciąż występują epizody o znacząco niższych wynikach, co może być związane z adaptacją polityki do nowych wyzwań w środowisku.

W odróżnieniu od wykresu treningu DQN, w przypadku A2C widoczny jest bardziej przewidywalny wzorzec w wynikach po osiągnięciu średniej w okolicach 200 punktów. Algorytm utrzymuje tę wartość z mniejszą zmiennością, co sugeruje, że agent jest w stanie bardziej spójnie lądować na platformie. Rozbieżność wyników staje się węższa w późniejszych epizodach, co oznacza, że model Actor-Critic bardziej skutecznie optymalizuje swoją politykę niż DQN, dla którego zmienność wyników pozostawała większa nawet po osiągnięciu progu

sukcesu.

Czas uczenia

W tabeli poniżej zostały przedstawione uśrednione wyniki dla uczenia 1, 100, 1000 epizodów oraz całkowitego czasu poświęconego na trening dla obydwu algorytmów. (Eksperymenty były przeprowadzone na procesorze: CPU Procesor 13th Gen Intel(R) Core(TM) i7-13700H, 2400 MHz, Rdzenie: 14, Procesory logiczne: 20)

Epizody \Algorytm	DQN [2355 epizody]	A2C [3906 epizody]
1	0.81s	0.59s
100	1min 22s	59.13s
1000	13min 36s	9min 59s
Całkowity trening	32min 04s	38min 30s

Tabela 1: Porównanie czasu treningu

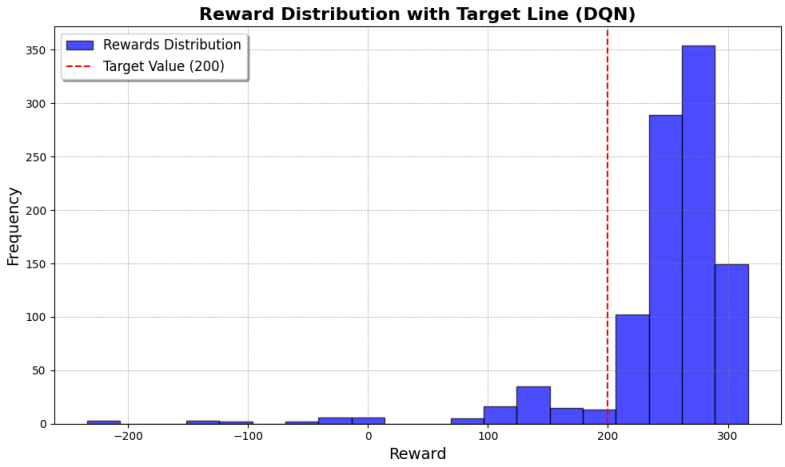
Algorytm A2C charakteryzuje się krótszym czasem przetwarzania pojedynczych epizodów w porównaniu do DQN, co wynika z jego architektury on-policy. A2C unika dodatkowych operacji, takich jak przechowywanie i przetwarzanie doświadczeń w buforze, co sprawia, że szybciej podejmuje decyzje w bieżących interakcjach ze środowiskiem. Ta przewaga czasowa jest widoczna szczególnie przy krótkich seriach epizodów, gdzie A2C lepiej wykorzystuje dostępne zasoby obliczeniowe.

Jednak całkowity czas treningu dla A2C jest dłuższy, ponieważ algorytm wymaga większej liczby epizodów, aby osiągnąć porównywalne wyniki z DQN. Z kolei DQN, mimo że potrzebuje więcej czasu na każdy epizod z powodu dodatkowych operacji, wymaga ich znacznie mniej, aby zakończyć trening. W efekcie A2C sprawdza się lepiej w scenariuszach, gdzie istotna jest szybkość przetwarzania danych w krótkim horyzoncie, podczas gdy DQN oferuje bardziej zrównoważony czas treningu w dłuższej perspektywie.

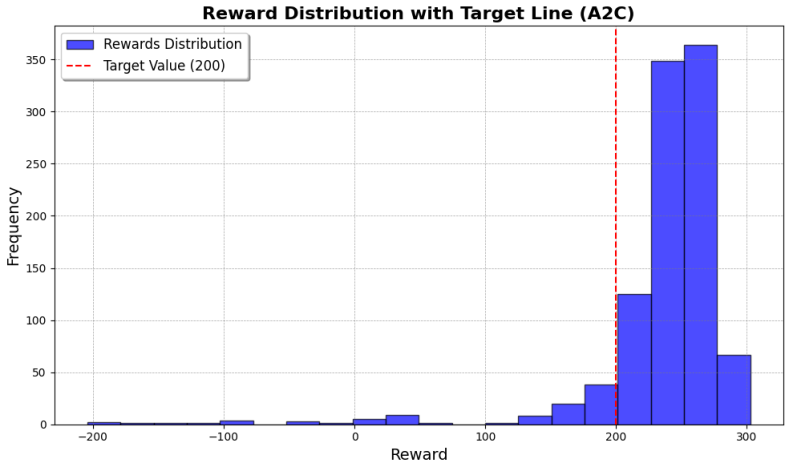
4 Wyniki

Testowanie modeli

Modele zostały przetestowane przez odpalenie procesu lądownika 1000 razy na każdy model. Poniżej przedstawiono 2 histogramy rozkładu nagród dla obu algorytmów z zaznaczoną czerwoną, przerywaną linią na wartości 200, która reprezentuje wartość docelową.



Rysunek 6: Testowanie modelu DQN



Rysunek 7: Trenowanie modelu A2C

Widzimy, że oba modele w znacznym stopniu przekraczają barierę 200 punktów. Z wizualnego punktu widzenia ciężko określić, który model sprawuje się lepiej w fazie testowej dlatego potrzebujemy bardziej zaawansowanych testów statystyczych.

Statystyka	DQN	A2C
Średnia	247.19	236.77
Maksimum	316.74	302.64
Minimum	-234.06	-204.20
Odchylenie standardowe	63.97	54.60
Procent powyżej 200	89.80	90.70
Mediana	261.79	248.26
25. percentyl	240.20	230.05
75. percentyl	279.72	261.46

Tabela 2: Porównanie statystyk nagród dla A2C i DQN

DQN osiąga wyższe wyniki średnie i maksymalne, co wskazuje na jego zdolność do uzyskiwania lepszych wyników w optymalnych warunkach. Z kolei A2C charakteryzuje się bardziej stabilnym zachowaniem, co można zauważyć w mniejszym rozrzucie wyników. Oznacza to, że model oparty na DQN ma potencjał do uzyskiwania wyższych nagród, ale robi to kosztem większej zmienności i większego ryzyka niskich wyników w niektórych epizodach.

Jeśli chodzi o stabilność, A2C wypada lepiej, uzyskując bardziej przewidywalne wyniki z mniejszym odchyleniem standardowym. Ponadto ten algorytm częściej przekracza ustalony próg sukcesu, co może być kluczowe w zadaniach, w których istotna jest konsekwencja działania.

Ilość kroków

Kolejnym istotnym czynnikiem w badaniu jak działają podane modele będzie analiza statystyk dotyczących ilości kroków potrzebnych do zakończenia działania programu.

Statystyka	DQN	A2C
Średnia	305.108	346.963
Maksimum	1000	890
Minimum	82	232
Odchylenie standardowe	194.67	96.88
Mediana	246.0	312.0
25. percentyl	230.0	290.0
75. percentyl	268.0	359.0

Tabela 3: Porównanie długości epizodów dla DQN i A2C

Wyniki wskazują, że algorytm A2C średnio prowadzi do dłuższych epizodów w porównaniu do DQN. Dłuższe epizody w przypadku A2C oznaczają, że algorytm częściej ma więcej czasu na korekty i precyzyjne dostosowanie swojej polityki, co może prowadzić do bardziej stabilnych wyników. Krótsze epizody w DQN sugerują, że algorytm szybciej osiąga stany końcowe, ale może to być związane z bardziej ryzykownymi decyzjami, które jak widać potrafią się opłacić np. zmniejszeniem zużycia paliwa.

Podsumowanie

W środowisku Lunar Lander kluczowe znaczenie mają zdolność algorytmu do precyzyjnego sterowania lądownikiem oraz efektywne zarządzanie zasobami, takimi jak paliwo. Analiza wyników pokazuje, że oba algorytmy różnią się podejściem do tych wyzwań, co znajduje odzwierciedlenie w ich wynikach.

DQN częściej osiąga wyższe nagrody i ma większy potencjał do uzyskiwania lepszych wyników w optymalnych warunkach. Jest jednak bardziej zmienny, co oznacza, że jego działania mogą prowadzić zarówno do bardzo dobrych rezultatów, jak i do wyjątkowo słabych epizodów.

A2C wykazuje bardziej stabilne działanie, co oznacza, że częściej uzyskuje wyniki zbliżone do optymalnych i rzadziej generuje skrajnie niskie wyniki. Algorytm ten charakteryzuje się większą regularnością, co czyni go lepiej dostosowanym do zadań wymagających oszczędnego zarządzania zasobami w sposób przewidywalny i zrównoważony.

Wybór między nimi zależy od wybranego celu tj. DQN sprawdzi się tam, gdzie kluczowa jest maksymalizacja wyników, podczas gdy A2C będzie lepszym wyborem w sytuacjach wymagających stabilności

Bibliografia

[1] The Farama Foundation. *Środowisko Lunar Lander w OpenAI Gymnasium*. Dostęp: 2025-01-22. 2023. URL: https://gymnasium.farama.org/environments/box2d/lunar_lander/.

[2] Jianqing Fan i in. *A Theoretical Analysis of Deep Q-Learning*. https://proceedings.mlr.press/v120/yang20a/yang20a.pdf. Proceedings of Machine Learning Research, Vol. 120:1–4. 2020.

[3] Shujin Qin i in. "An Optimized Advantage Actor-Critic Algorithm for Disassembly Line Balancing Problem Considering Disassembly Tool Degradation". W: *Mathematics* 12.6 (2024). ISSN: 2227-7390. DOI: 10.3390/math12060836. URL: https://www.mdpi.com/2227-7390/12/6/836.

[4] Khadijeh Alibabaei i in. *Comparison of On-Policy Deep Reinforcement Learning A2C with Off-Policy DQN in Irrigation Optimization*. Computers, 11(7):104, DOI: 10.3390/computers11070104. 2022. URL: https://www.mdpi.com/2073-431X/11/7/104.