Projekt Zespołowy: Zaawansowane Metody Uczenia Maszynowego

Alicja Osam-Gyaabin Mikołaj Zawada Karol Kociołek

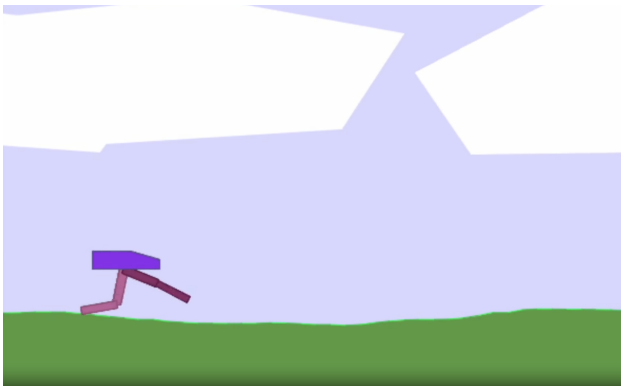
***Streszczenie*—Celem projektu jest opracowanie dwóch modeli uczenia przez wzmacnianie w środowisku BipedalWalker-v3. Łącznie z analizą środowiska, wybraniem 2 algorytmów oraz wytrenowaniem 2 modeli.**

I. Wybór i Opis Środowiska

A. Wybór Środowiska

W celu realizacji projektu zdecydowaliśmy się wybrać środowisko Bipedal Walker a konkretnie BipedalWalker-v3 w wersji Normal (Nie zawierającej dodatkowych przeszkód jak schody czy dziury).

Dokładna specyfikacja wybranego środowiska dostępna jest pod adresem: gymnasium.farama.org/environments/box2d/bipedal_walker/



Rysunek 1. Przykładowy obraz ze środowiska podczas sterowania Walkerem przez agenta wytrenowanego na bazie PPO

B. Opis Środowiska

Bipedal Walker to środowisko należące do rodziny środowisk Box2D. Przedstawia prostego robota kroczącego o czterech stawach (2xbiodra i 2xkolana) po terenie o niewielkich nierównościach (w wersji normal) lub z dodatkowymi przeszkodami (w wersji Hardcore). Wymaga sterowania ciągłego.

C. Przestrzeń stanów i akcji

Stan środowiska opisuje wektor o 24 wymiarach, który zawiera:

- Kąt i prędkość kątową korpusu
- Prędkość poziomą i pionową
- Pozycje stawów oraz ich prędkości kątowe
- Informację o kontakcie nóg z podłożem (od wersji V1)
- 10 pomiarów z lidara

Walker może wykonać 4 akcje. Każda odpowiada ustawiniu wartości w przedziale [-1, 1] na jednym z 4 stawów.

1) Nagrody: Agent otrzymuje nagrodę za poruszanie się do przodu. Maksymalna suma nagród wynosi 300+ punktów, jeśli agent dotrze do końca terenu. Upadek robota powoduje karę w wysokości -100 punktów. Wykorzystywanie silników jest mało, ale lekko karane, tak aby walker nie tylko nauczył się chodzić ale robił to również optymalnie.

2) Stan początkowy: Robot rozpoczyna symulację stojąc na lewej krawędzi terenu. Jego korpus jest ustawiony poziomo, a obie nogi znajdują się w takiej samej pozycji.

3) Zakończenie epizodu: Epizod kończy się w następujących sytuacjach:

- Korpus robota dotknie podłoża (Kara -100 pkt)
- Robot przekroczy prawą krawędź terenu
- Skończy się czas

II. Opis algorytmu Proximal Policy Optimization (PPO)

Proximal Policy Optimization należy do klasy algorytmów uczenia ze wzmocnieniem opartych na optymalizacji strategii. Głównym celem algorytmu jest znalezienie optymalnej polityki $\pi_\theta(a|s)$, która maksymalizuje skumulowaną wartość nagrody oczekiwanej:

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{1}$$

gdzie:

- $\pi_\theta(a|s)$ – polityka parametryzowana przez θ, opisująca prawdopodobieństwo wyboru akcji a w stanie s,
- γ – współczynnik dyskontowania. Określa w jakim stopniu decyzje z przeszłości wpływają na podejmowaną decyzję. Jest to hiperparamter, $\gamma \in [0, 1]$,
- r_t – nagroda otrzymana w kroku t.

PPO modyfikuje istniejącą politykę, aby poprawić jej skuteczność, ograniczając jednocześnie nadmierne zmiany w parametrach θ (wagi i biasy), co stabilizuje proces uczenia.

A. Funkcja celu PPO

Kluczowym elementem PPO jest funkcja celu. Funkcja ta opiera się na współczynniku prawdopodobieństwa:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}, \tag{2}$$

gdzie $\pi_{\text{old}}(a_t|s_t)$ to polityka przed aktualizacją parametrów.

Funkcja celu ogranicza zmiany w polityce za pomocą operatora *clipping*:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right], \tag{3}$$

gdzie:

- A_t – korzyść, która mierzy, jak korzystna jest dana akcja w stanie s_t,
- ϵ – parametr ograniczający zmiany w polityce, typowo $\epsilon = 0.2$.

Dzięki tej funkcji algorytm unika zbyt dużych zmian w polityce, wybierając bardziej konserwatywne aktualizacje.

B. Obliczanie korzyści

PPO korzysta z metody Generalized Advantage Estimation (GAE) do obliczania korzyści A_t:

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots, \tag{4}$$

gdzie:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{5}$$

GAE pozwala kontrolować kompromis między wariancją a obciążeniem estymacji korzyści poprzez parametr $\lambda \in [0, 1]$.

C. Uczenie krytyka

Drugi komponent PPO to sieć krytyka, która uczy się wartości stanu $V(s_t)$. Optymalizacja krytyka polega na minimalizacji błędu między przewidywaną wartością stanu a zwrotem:

$$L^{\text{value}}(\phi) = \mathbb{E}_t \left[\max \left((V_\phi(s_t) - R_t)^2, (V_{\text{clip}}(s_t) - R_t)^2 \right) \right], \tag{6}$$

gdzie:

- $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$ – skumulowany zwrot,
- $V_{\text{clip}}(s_t)$ – estymacja wartości stanu ograniczona do przedziału $[V_{\text{old}}(s_t) - \epsilon, V_{\text{old}}(s_t) + \epsilon]$.

D. Regularizacja entropii

Aby zachęcić agenta do eksplorowania środowiska, do funkcji celu PPO dodawany jest dodatkowy składnik związany z entropią polityki:

$$H(\pi_\theta) = -\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s). \tag{7}$$

Ostateczna funkcja celu PPO przyjmuje postać:

$$L(\theta) = L^{\text{CLIP}}(\theta) - c_1 L^{\text{value}}(\phi) + c_2 H(\pi_\theta), \tag{8}$$

gdzie c_1 i c_2 to wagi regulujące wpływ poszczególnych składników.

E. Proces treningu

Proces treningu algorytmu PPO przebiega w następujących krokach:

1) Agent zbiera doświadczenia w środowisku (stany, akcje, nagrody).
2) Obliczane są zwroty R_t i korzyści A_t.
3) Aktualizowane są parametry:
   - Polityki aktora poprzez maksymalizację $L^{\text{CLIP}}(\theta)$,
   - Sieci wartości poprzez minimalizację $L^{\text{value}}(\phi)$.

F. Podsumowanie

Algorytm PPO to stabilna i skuteczna metoda optymalizacji polityki w uczeniu ze wzmocnieniem. Dzięki ograniczeniom wprowadzanym przez funkcję celu oraz regularizacji entropii, PPO pozwala na efektywne uczenie się polityki przy zachowaniu stabilności.

III. Drugi algorytm

Ze względu na potencjalnie słabe wyniki standardowym podejściem Q-learning w złożonych środowiskach o ciągłej przestrzeni akcji, zastosowano jego ulepszoną werjsę wykorzystującą sieć Q z wieloma głowicami akcji oraz mechanizmem Double Q-Network.

A. Q-Wartość

Q-wartość jest kluczowym pojęciem w drl. Reprezentuje ona oczekiwaną sumę nagród, jakie agent może uzyskać, wykonując określoną akcję w danym stanie środowiska i postępując zgodnie z określoną strategią. Formalnie:

$Q(s, a) =$ oczekiwana suma nagród po wykonaniu akcji a w stanie s

B. Replay Buffer

Bufor Doświadczeń przechowuje doświadczenia agenta w postaci przejść $(s, a, r, s', done)$, gdzie:

- s – obecny stan,
- a – wykonana akcja,
- r – otrzymana nagroda,
- s' – następny stan,
- $done$ – flaga zakończenia epizodu.

Bufor umożliwia losowe próbkowanie mini-batchów doświadczeń do treningu, co pomaga w stabilizacji procesu uczenia poprzez redukcję korelacji między kolejnymi danymi treningowymi.

C. Double Q-Network

Sieć Q jest siecią neuronową, która przyjmuje stan środowiska jako wejście i przewiduje Q-wartości dla wszystkich możliwych akcji. W przedstawionym modelu agent zarządza procesem uczenia się, korzystając z dwóch sieci Q:

- Główna sieć Q: Służy do przewidywania Q-wartości i podejmowania decyzji.
- Sieć docelowa Q: Używana do stabilizacji treningu poprzez dostarczanie stałych wartości docelowych, które są aktualizowane rzadziej.

D. Strategia Eksploracji-Eksploatacji (Epsilon-Greedy)

Strategia epsilon-greedy balansuje między eksploracją nowych akcji (wybór losowy) a eksploatacją znanych, najlepiej ocenianych akcji (wybór na podstawie sieci Q). Parametr ϵ decyduje o prawdopodobieństwie wyboru akcji losowej.

E. Działanie algorytmu

- Agent rozpoczyna epizod, resetując środowisko i otrzymując początkowy stan s.

Wybór Akcji:

- Na podstawie strategii epsilon-greedy, agent decyduje, czy wykonać akcję losową (eksploracja) czy wybrać akcję o najwyższej przewidywanej Q-wartości (eksploatacja) przy użyciu głównej sieci Q.
- W przypadku eksploatacji, sieć Q przetwarza stan s i generuje Q-wartości dla wszystkich możliwych akcji. Agent wybiera akcje z najwyższymi wartościami.

Wykonanie Akcji i Zbieranie Doświadczeń:

- Wybrana akcja a jest wykonana w środowisku, co skutkuje przejściem do nowego stanu s', otrzymaniem nagrody r oraz informacją o zakończeniu epizodu ($done$).
- Przejście $(s, a, r, s', done)$ jest dodawane do Replay Buffer.

Próbkowanie i Trening Sieci Q:

- Gdy Replay Buffer jest wystarczająco pełny (np. więcej niż 5000 przechowywanych przejść), agent losowo wybiera mini-batch doświadczeń do treningu.
- Sieć Q przetwarza stany s z mini-batcha, przewidując obecne Q-wartości dla wykonanych akcji.
- Sieć docelowa Q przetwarza następne stany s', przewidując przyszłe Q-wartości. Używa się ich do obliczenia wartości docelowych:

$$\text{target} = r + \gamma \cdot \max_{a'} Q_{\text{target}}(s', a') \cdot (1 - done)$$

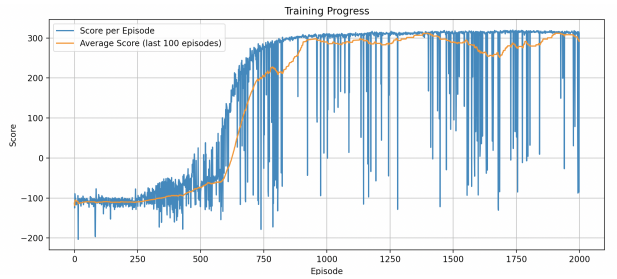
- Funkcja straty (średniokwadratowy błąd) porównuje obecne Q-wartości z wartościami docelowymi.
- Optymalizator (Adam) aktualizuje wagi głównej sieci Q na podstawie obliczonych gradientów.

Aktualizacja Sieci Docelowej:

- Co określoną liczbę kroków, wagi sieci docelowej są synchronizowane z wagami głównej sieci Q, co pomaga w stabilizacji procesu uczenia.

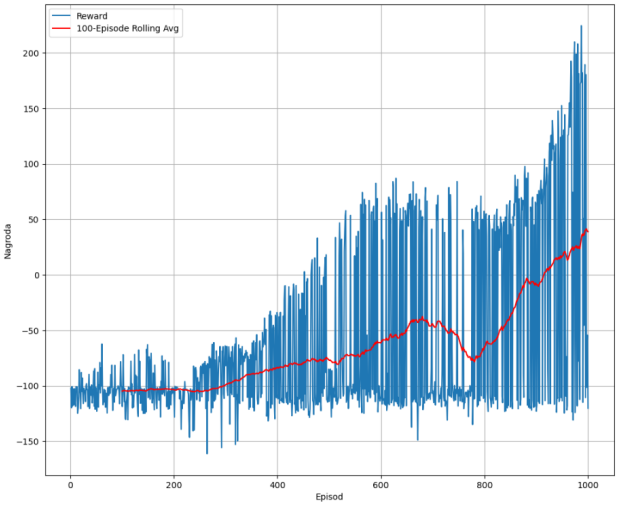
IV. Wyniki trenowanie Modeli

W celu porównania skuteczności modeli przeanalizowaliśmy nagrodę uzyskiwaną na epizod oraz średnią nagród na 100 episodów w określonym momencie trenowania. Wynik działania poszczególnych modeli zamieszczony w osobnych plikach.



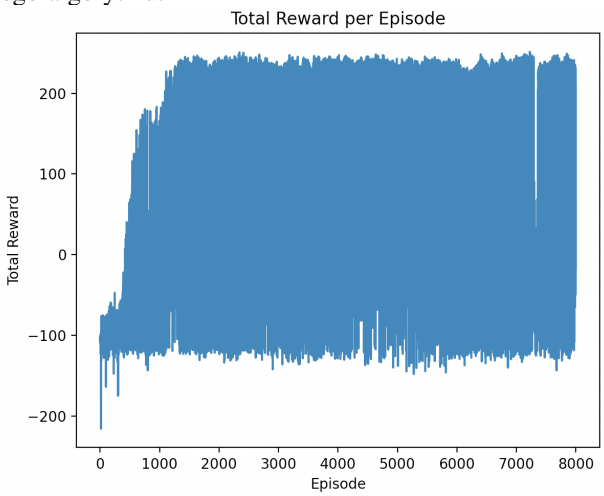
Rysunek 2. Trening model DQN

Z wykresu wartości wyniku dla episodu widać, że swoje maksimum algorytm osiągnął już po 1000 episodach. Dalsze trenowanie nie przynosi większych zmian i jest bezcelowe. Dla tak skonstruowanego algorytmu Walker nauczył się przemieszczać do przodu jednak blisko nie tak dobrze jak modele zaproponowane w BipedalWalker-v3 leaderboard.



Rysunek 3. Trening agenta oparty o PPO

Podczas uruchamiania agenta opartego o algorytm PPO zauważyliśmy, że już również po 1000 episodów walker nauczył się przemieszczać do przodu. Jednak dopiero test na większej ilości episodów (8000) pokazuje pełne możliwości tego algorytmu.



Rysunek 4. Trening agenta oparty o PPO - 8000 episodów

Tak zaimplementowany algorytm osiąga swoje maksimum po 1500 episodach. Co, w połączeniu z poprzednimi obserwacjami sugeruje, że podejście oparte o DQN nadaje się lepiej do tego problemu ponieważ uczy się lepiej i szybciej a jego maksymalny wynik jest większy niż możliwy maksymalny wynik oferowany przez agenta opartego o PPO. Warto jednak zaznaczyć, że w obu modelach jest duża przestrzeń do optymalizacji, od modyfikacji hiperparametrów po zwiększenie głębokości sieci neuronowych.