Politechnika Wrocławska

Wydział Matematyki

KIERUNEK:
Matematyka Stosowana

PRACA DYPLOMOWA INŻYNIERSKA

TYTUŁ PRACY:

Analiza efektywności metod uczenia przez wzmacnianie w grach komputerowych

AUTOR:

Adrian Galik

PROMOTOR:

dr hab. Janusz Szwabiński

WROCŁAW 2024

1 Wstęp

W ostatnich dekadach obserwuje się dynamiczny rozwój technologii, który przekracza pierwotne oczekiwania specjalistów. Zwiększona dostępność mocy obliczeniowej spowodowała, że algorytmy uczenia maszynowego stały się nieodłączną częścią codziennej aktywności. Zastosowania tych algorytmów można odnaleźć w robotyce, rozpoznawaniu obrazów, przetwarzaniu języka naturalnego, klasyfikacji spamu, systemach nawigacyjnych, diagnostyce chorób czy w sztucznej inteligencji dedykowanej grom komputerowym. Każda z wymienionych dziedzin oddziałuje na społeczeństwo zarówno pośrednio, jak i bezpośrednio.

Jedną z najciekawszych, a zarazem najdłużej rozwijanych poddziedzin uczenia maszynowego jest uczenie przez wzmacnianie. Metody tego typu, znane już od lat 50. ubiegłego wieku, stanowią centralny punkt rozważań niniejszej pracy.

Celem poniższej pracy inżynierskiej jest zbadanie efektywności wybranych metod uczenia przez wzmacnianie w grach komputerowych. Analiza skupia się w szczególności na porównaniu popularnych algorytmów pod kątem czasu uczenia oraz osiąganych wyników. W eksperymentach wykorzystano klasyczną grę Pong, często stosowaną w roli środowiska testowego do oceny zachowania agentów sztucznej inteligencji. Zaimplementowano dwie powszechnie używane metody: Deep Q-Learning (DQN), zaproponowaną przez Mnhia et al. [1], Advantage Actor-Critic (A2C), będącą uporoszczoną wersją asynchronicznych metod aktor-krytyk zaprezentowanych przez Mniha et al. [2] W kolejnych rozdziałach przedstawiono charakterystykę badanych algorytmów, omówiono przebieg eksperymentu oraz przeanalizowano otrzymane rezultaty.

2 Wprowadzenie do uczenia maszynowego

Uczenie maszynowe jest jedną z najważniejszych gałęzi sztucznej inteligencji. Jego istotę stanowi tworzenie algorytmów zdolnych do samodzielnego nabywania wiedzy na podstawie przetwarzanych danych, bez konieczności programowania konkretnych reguł. Według klasycznej definicji Arthura Samuela z 1959 roku, uczenie maszynowe to „dziedzina nauki dająca komputerom możliwość uczenia się bez konieczności ich jawnego programowania" [3]. Z kolei Tom Mitchell (1997) zwraca uwagę, że „program komputerowy uczy się na podstawie doświadczenia E w odniesieniu do zadania T i pewnej miary wydajności P, jeśli wydajność tego programu (mierzona za pomocą P) wobec zadania T poprawia się wraz z kolejnymi doświadczeniami E" [4].

W praktyce uczenie maszynowe opiera się na zbiorze danych uczących (ang. training set), którego elementy określa się mianem próbek lub przykładów uczących. Modelem zaś nazywa się część systemu odpowiedzialną za wyciąganie wniosków, w oparciu o dostarczone dane oraz proces uczenia. Przykładami modeli mogą być między innymi sieci neuronowe czy lasy losowe.

W przypadku zagadnienia klasyfikacji spamu, zadanie T polega na rozróżnianiu, czy dana wiadomość e-mail powinna zostać zakwalifikowana jako „spam" czy „nie spam". Doświadczeniem E jest tutaj zbiór wiadomości z odpowiednimi etykietami, a miarą wydajności P może być odsetek poprawnie zaklasyfikowanych wiadomości [5].

2.1 Podział uczenia maszynowego

Istnieje kilka kategorii uczenia maszynowego, wyróżnianych w zależności od rodzaju dostępnych danych i celu analizy. Najczęściej spotykanymi są:

2.1.1 Uczenie nadzorowane

Uczenie nadzorowane zakłada wykorzystanie zbioru danych z oznaczonymi przez człowieka etykietami. Metoda ta jest szeroko stosowana w zadaniach klasyfikacji (np. klasyfikacja spamu) oraz regresji (np. przewidywanie wartości liczbowych). Do popularnych algorytmów należą między innymi: regresja liniowa, drzewa decyzyjne oraz maszyny wektorów nośnych (SVM).

2.1.2 Uczenie nienadzorowane

W uczeniu nienadzorowanym algorytm otrzymuje dane bez dodatkowych etykiet, a celem jest odnajdywanie ukrytych wzorców i struktur. Główne zadania obejmują wizualizację danych, redukcję wymiarowości, analizę skupień (klastrów) oraz wykrywanie anomalii. Do typowych algorytmów zaliczają się K-Means oraz DBSCAN.

2.1.3 Uczenie częściowo nadzorowane

Uczenie częściowo nadzorowane stanowi wariant podejścia nadzorowanego, w którym etykiety nie są nadane ręcznie, lecz automatycznie wyznaczane na podstawie określonych heurystyk lub dodatkowych algorytmów. Metoda ta znajduje zastosowanie w sytuacjach, gdy proces ręcznego oznaczania danych jest kosztowny bądź czasochłonny, co często ma miejsce w diagnozach medycznych.

2.1.4 Uczenie przez wzmacnianie

Uczenie przez wzmacnianie (ang. reinforcement learning) było przez długi czas mniej popularne niż inne formy, jednak nabrało rozpędu dzięki sukcesom autorów projektu Google DeepMind [6], którzy zaprezentowali jego skuteczność w grach Atari. Charakteryzuje się tym, że algorytm (zwany agentem) uczy się optymalnych akcji poprzez interakcję z dynamicznym środowiskiem. Po wykonaniu każdej akcji agent otrzymuje nagrodę lub karę, co umożliwia stopniowe dopasowywanie strategii działania w celu maksymalizacji długoterminowego zysku. W ramach niniejszej pracy analizie zostały poddane właśnie takie algorytmy, z naciskiem na ich zastosowanie w środowisku gry Pong.

Podział uczenia maszynowego na te cztery kategorie jest szeroko akceptowany w literaturze i szczegółowo opisany w [7].

3 Teoretyczne podstawy uczenia przez wzmacnianie

3.1 Podstawowe pojęcia i definicje

W tej części pracy zastosowano standardowe oznaczenia i definicje stosowane w literaturze dotyczącej uczenia przez wzmacnianie. W szczególności terminologia i symbole (np. s, a, r_t, π) [8]

- **Agent** - Element systemu wchodzący w interakcje ze środowiskiem poprzez wykonywanie akcji (decyzji) oraz obserwowanie konsekwencji w postaci nagród i stanów. Głównym celem agenta jest maksymalizacja długoterminowej nagrody. Przykładowo w szachach agentem może być gracz lub program komputerowy.

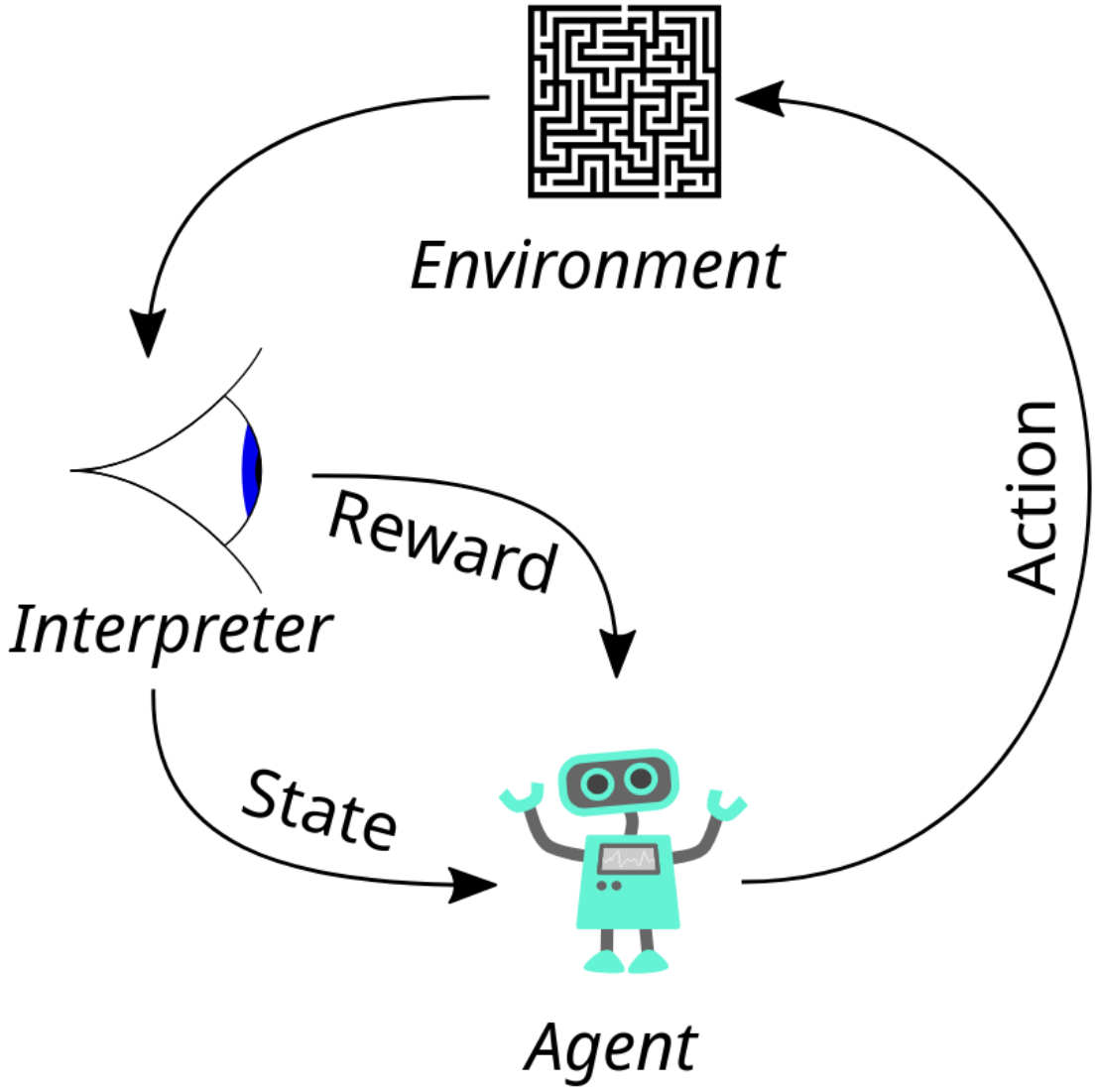
- **Środowisko** - Otoczenie, z którym agent ma styczność. Wymiana informacji ze środowiskiem obejmuje jedynie obserwacje (stany) i nagrody. Dla gry w szachy środowiskiem jest plansza szachowa wraz z aktualnym układem figur.

- **Stan** (s) - Informacje które środowisko dostarcza agentowi. Dają one wiadomości na temat tego co dzieje się wokół niego.

- **Akcje** (a) - Wszystkie czynności, które agent może podjąć w danym środowisku. Przykładową akcją w szachach jest przesunięcie pionka o jedno pole do przodu.

- **Nagroda** (r_t) - Informacja zwrotna otrzymywana od środowiska, wskazująca na korzystność (bądź niekorzystność) podjętej akcji. Nagroda ma z reguły charakter lokalny, czyli dotyczy wyłącznie niedawno wykonanej akcji, a nie całej historii działań agenta. Zadanie agenta polega na maksymalizacji skumulowanej nagrody w dłuższej perspektywie.

- **Polityka** (π) - Strategia agenta, która pomaga mu podejmować akcje w danych stanach. Polityka może być deterministyczna ($\pi(s) = a$) albo stochastyczna ($\pi(a|s)$)



Rysunek 1: Typowa struktura scenariusza uczenia przez wzmacnianie [9]

3.2 Modele Markowa (MDP)

Wzory i definicje użyte w tej sekcji zostały zaczerpnięte z książki „Reinforcement Learning: An Introduction" autorstwa Suttona i Barto [7]. Markowskie procesy decyzyjne (MDP) są formalizacją problemów decyzyjnych w warunkach niepewności, które zakładają spełnienie własności markowskiej: przyszłość (kolejny stan i otrzymana nagroda) zależy jedynie od bieżącego stanu i akcji, a nie od pełnej historii. W modelu MDP kluczowymi elementami są:

- Stany i akcje - Agent w chwili t obserwuje stan S_t ze zbiorów stanów S, a następnie wybiera akcję A_t z dostępnego zbioru akcji A.
- Funkcja przejścia i nagród - Po wykonaniu akcji A_t w stanie S_t agent przechodzi do stanu S_{t+1} i otrzymuje nagrodę R_{t+1}. Oba te elemenety opisuje funkcja:

$$p(s', r|s, a) = P(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a) \tag{3.1}$$

dzięki temu wiadomo z jakim prawdopodobieństwem przy danej akcji w stanie s agent znajdzie się w stanie s' oraz jaką otrzyma wtedy nagrodę r.

- Oczekiwana nagorda - Bardzo często zamiast śledzić cały rozkład nagród, pracuje się z wartością oczekiwaną natychmiastowej nagrody:

$$r(s,a) = E[R_{t+1}|S_t = s, A_t = a] = \sum_{s',r} rp(s', r|s, a) \tag{3.2}$$

  Innymi słowy, jest to średnia nagroda, jakiej agent może oczekiwać w momencie przejścia ze stanu s do s' przy akcji a.

- Współczynnik dyskontowania - Aby modelować długofalowe konsekwencje podejmowanych decyzji, wprowadza się współczynnik $\gamma \in [0, 1]$. Określa on, jak silnie agent ceni przyszłe nagrody w porównaniu z bieżącymi. Gdy $\gamma = 0$, agent skupia się wyłącznie na nagrodach natychmiastowych, a gdy γ jest bliskie 1, uwzględnia głównie wpływ aktualnej decyzji na dalszą przyszłość.

Skumulowana nagroda

Dla każdego epizodu w uczeniu przez wzmacnianie definiuje się skumulowaną nagrodę w chwili t następująco:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.3}$$

Gdzie:

- G_t - całkowita (skumulowana) wartość nagród, którą agent otrzymuje od chwili t aż do zakończenia epizodu,
- γ - współczynnik dyskontowania z przedziału $[0, 1]$, który wyznacza, jak bardzo agent ceni przyszłe nagrody w stosunku do natychmiastowych, Na przykład:
  - Dla $\gamma = 0$, agent skupia się wyłącznie na nagrodach natychmiastowych,
  - Gdy γ jest blisko 1, agent korzysta z długoterminowych strategii co może przyniać się do bardziej sensownych akcji,
- R_{t+k+1} - Nagroda otrzymana przez agenta w kroku czasowym $t + k + 1$. Są to nagrody będące sygnałami zwrotnymi otrzymanymi od środowiska, mające na celu informowanie agenta o jakości jego działań,
- k - indeks czasowy który określa zasięg możliwości przyszłych decyzji agenta, dzięki któremu jest w stanie obliczyć skumulowaną nagrodę. Sumowanie zaczyna się od $k = 0$ co wskazuje nagrodzie otrzymanej po wykonaniu akcji w stanie S_t.

Skumulowana nagroda G_t jest ma kluczowe znaczenie w uczeniu przez wzmacnianie ponieważ określa ocenę jakości działań agenta. Stanowi ona podstawyw cel, który agent stara się maksymalizować poprzez optymalny wybór akcji.

Funkcje wartości i algorytmy uczenia

- Funckja wartości stanu $V_\pi(s)$ - Oczekiwana skumulowana nagroda przy założeniu, że agent znajduje się w stanie s i przestrzega polityki π:

$$V_\pi(s) = E_\pi[G_t|S_t = s] \tag{3.4}$$

- Funkcja wartości akcji $Q_\pi(s, a)$ - Oczekiwana skumulowana nagroda przy wykonaniu akcji a w stanie s, a następnie kontynuacji zgodnie z polityką π: a następnie postępując zgodnie z polityką π.

$$Q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] \tag{3.5}$$

Funkcje te są kluczowe w wielu algorytmach uczenia przez wzmacnianie. W podejściu Q-learning następuje iteracyjna aktualizacja wartości Q w celu wyznaczenia optymalnej polityki, natomiast w metodach typu aktor-krytyk (np. A2C) równocześnie modyfikuje się politykę (aktor) i funkcję wartości (krytyk).

Przykład dla gry Pong

W celu zilustrowania pojęcia skumulowanej nagrody można rozważyć epizod w grze Pong, gdzie agent otrzymuje w kolejnych krokach czasowych następujące nagrody:

- $r_1 = +1$ (zdobycie punktu)
- $r_2 = -1$ (utrata punktu)
- $r_3 = +1$
- $r_4 = +1$
- $r_5 = -1$

Zakładając że $\gamma = 0.9$, wtedy skumulowana nagroda G_0 zaczynając od chwili $t = 0$ będzie obliczana jako:

$$G_0 = \gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \gamma^4 r_5 + ...$$

$$G_0 = 1 * 1 + 0.9 * (-1) + 0.9^2 * 1 + 0.9^3 * 1 + 0.9^4 * (-1) + ...$$

$$G_0 = 1 - 0.9 + 0.81 + 0.729 - 0.6561 + ...$$

Maksymalizacja tej sumy wymusza na agencie podejmowanie decyzji zapewniających możliwie największy zysk nie tylko w bieżącym kroku, lecz także w dalszych etapach rozgrywki.

3.3 Równanie Bellmana

Wzory i definicje użyte w tej sekcji zostały zaczerpnięte z książki „Reinforcement Learning: An Introduction" autorstwa Suttona i Barto [7]. Równanie Bellmana pełni kluczową rolę w teorii uczenia przez wzmacnianie, umożliwiając sformalizowanie zależności między wartościami stanów a podejmowanymi akcjami. Używane jest głównie do iteracyjnego obliczania wartości funkcji stanów i akcji, co stanowi fundament wielu algorytmów. Jak zauważyli Sutton i Barto (2018), Równanie Bellmana stanowi podstawę dla większości algorytmów uczenia przez wzmacnianie, ponieważ pozwala na efektywne obliczanie wartości stanów i akcji poprzez iteracyjne aktualizacje [7].

3.3.1 Równanie Bellmana dla funkcji wartości stanu $V_\pi(s)$

Funkcja wartości stanu $V_\pi(s)$ wyraża oczekiwaną sumę zdyskontowanych nagród, jakie agent może uzyskać, rozpoczynając od stanu s i postępując zgodnie z polityką π. Równanie Bellmana w tym kontekście ma postać:

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma V_\pi(s')) \tag{3.6}$$

gdzie:

- $V_\pi(s)$ - Funckja stanu wartości dla polityki π, określająca sumę zdyskontowanych nagród od stanu s,
- E_π - wartość oczekiwana względem rozkładu wyznaczonego przez politykę π,
- R_{t+1} - nagroda otrzymywana po przejściu do stanu S_{t+1} w wyniku podjęcia akcji zgodnej z π,
- γ - Współczynnik dyskonotwania z przedziału $[0, 1]$,
- S_{t+1} - Stan osiągnięty po wykonaniu akcji w stanie s.

Równanie to można interpretować poprzez równość wartości stanu s a oczekiwanej nagrodzie otrzymanej po przejściu do kolejnego stanu plus zdyskontowanej wartości nowego stanu, zakładając, iż agent działa zgodnie z polityką π.

3.3.2 Równanie Bellmana dla funkcji wartości akcji $Q_\pi(s, a)$

Funkcja wartości akcji $Q_\pi(s, a)$ reprezentuje oczekiwaną sumę zdyskontowanych nagród, uzyskiwanych przez agenta w sytuacji, gdy w stanie ss podjęta zostanie akcja a, a w kolejnych krokach zastosowana zostanie polityka π. Odpowiednie równanie Bellmana ma postać:

$$Q_\pi(s, a) = E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma \sum_a \pi(a|s) Q_\pi(s', a')) \tag{3.7}$$

Zgodnie z tym równaniem wartość akcji a w stanie s zależy od natychmiastowej nagrody oraz zdyskontowanej wartości przyszłych akcji, które zostaną wybrane zgodnie z polityką π.

3.3.3 Równanie Bellmana dla polityki optymalnej $V_*(s)$ i $Q_*(s, a)$

Polityka optymalna π_* maksymalizuje funkcję wartości stanu. Oznacza to, że:

$$V_*(s) = \max_\pi V_\pi(s) \tag{3.8}$$

Równanie Bellmana dla optymalnej funkcji wartości stanu może zostać zapisane w postaci:

$$V_*(s) = \max_a E[R_{t+1} + \gamma V_*(S_{t+1}) | S_t = s, A_t = a] = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma V_*(s')) \tag{3.9}$$

Analogicznie, optymalna funkcja wartości akcji wyraża się wzorem:

$$Q_*(s, a) = E[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') | S_t = s, A_t = a] = \sum_{s',r} p(s', r|s, a)(r + \gamma \max_{a'} Q_*(s', a')) \tag{3.10}$$

Powyższe równania można interpretować następująco:

- $V_*(s)$ - Najlepsza możliwa wastość stanu s, uzyskana poprzez wybór najlepszej akcji.
- $Q_*(s, a)$ - Najlepsza możliwa wartość akcji a w stanie s, przy założeniu dalszego postępowania według optymalnych decyzji.

Opisane zależności leżą u podstaw algorytmów takich jak Value Iteration czy Q-learning, które dążą do znalezienia polityki maksymalizującej skumulowaną nagrodę.

3.3.4 Metoda iteracji wartości

Algorytm, który pozwala na uteracyjną aktualizację funkcji wartości stanu $V(s)$ zgodnie z równaniem Bellmana dla optymalnej polityki, do momentu osiągnięcia zbieżności. Składa się ona z poniższych kroków:

- Zainicjalizuj wszystkie stany V_i z pewnymi wartościami początkowymi. Zawyczaj $V(s) = 0$ dla wszystkich $s \in S$.
- Dla każdego stanu $s \in S$ w procesie decyzyjnym Markowa wykonaj aktualizacje:

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{3.11}$$

- Powtarzaj poprzedni krok poprzez wykonanie wielu iteracji do momentu gdy maksymalna zmiana $V(s)$ jest mniejsza niż zadany próg.

3.3.5 Metoda iteracji polityki

Algorytm składający się z dwóch głównych kroków: ewaluacji polityki i jej ulepszania. Składa się on z poniższych kroków:

- Zainicjalizuj początkową politykę $\pi(s)$ oraz $V(s)$
- Oblicz wartość $V(s)$ dla bierzącej polityki π za pomocą poniższego wzoru:

$$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))(r + \gamma V(s')) \tag{3.12}$$

- ulepszenie polityki poprzez wybór akcji a maksymalizującej wartość oczekiwaną dla każdego stanu s za pomocą poniższego wzoru:

$$pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{3.13}$$

- Jeżeli $\pi' = \pi$, kończy się proces w przeciwnym wypadku ustaw $\pi \leftarrow \pi'$ i ponów kroki 2-3.

3.4 Metoda entropii krzyżowej w uczeniu przez wzmacnianie

Wzory i definicje użyte w tej sekcji zostały zaczerpnięte z książki „Deep Reinforcement Learning Hands-On." autorstwa Maxima Lapana [8]. Entropia krzyżowa jest miarą różnicy pomiędzy dwoma rozkładami prawdopodobieństwa. W kontekście uczenia przez wzmacnianie bywa wykorzystywana do oceny jakości nowej polityki $\pi_{new}(a|s)$ względem idealnego rozkładu akcji, który ma maksymalizować skumulowaną nagrodę. Definicja entropii krzyżowej pomiędzy rozkładami $p(a)$ i $q(a)$ ma postać:

$$H(p,q) = -\sum_{a \in A} p(a) log(q(a)) \tag{3.14}$$

gdzie:

- $p(a)$ - Jest rozkładem prawdopodobieństwa akcji a według starej polityki $\pi_{old}(a|s)$.
- $q(a)$ - Jest rozkładem prawdopodobieństwa akcji a według nowej polityki $\pi_{new}(a|s)$.

3.4.1 Twierdzenie o próbkowaniu istotnościowym

Próbkowanie istotnościowe pozwala na wykorzystanie próbek pochodzących z pewnego rozkładu prawdopodobieństwa w celu oszacowania wartości oczekiwanej funkcji definiowanej względem innego rozkładu. W uczeniu przez wzmacnianie z próbkowaniem istotnościowym ma się do czynienia np. wtedy, gdy dane są zbierane na podstawie starej polityki $\pi_{old}(a|s)$, zaś celem jest szacowanie wartości dla nowej polityki $\pi_{new}(a|s)$. Zgodnie z twierdzeniem:
Twierdzenie o próbkowaniu istotnościowym:

$$E_{x \sim p(x)}[H(x)] = \int_x p(x)H(x)dx = \int_x q(x)\frac{p(x)}{q(x)}H(x)dx = E_{x \sim q(x)}[\frac{p(x)}{q(x)}H(x)] \tag{3.15}$$

gdzie:

- $p(x)$ - Rozkład próbkowania (np. stara polityka)
- $q(x)$ - Rozkład docelowy (np. nowa polityka)
- $H(x)$ - Funkcja entropi w stanie x często definiowana jako:

$$H(\pi) = -\sum_{a \in A} \pi(a|s) log(\pi(a|s)) \tag{3.16}$$

3.4.2 Dywergencja Kullbacka-Leiblera

Dywergencja Kullbacka-Leiblera (KL) mierzy odległość między dwoma rozkładami prawdopodobieństwa $p(x)$ i $q(x)$. W uczeniu przez wzmacnianie może służyć do kontrolowania, jak bardzo nowa polityka różni się od starej, co pomaga zapobiegać gwałtownym zmianom w trakcie treningu. Definicję KL przedstawia równanie:

$$KL(p(x)||q(x)) = \sum_x p(x)\frac{p(x)}{q(x)} \tag{3.17}$$

W kontekście uczenia przez wzmacnianie:

$$KL(\pi_{old}(a|s)||\pi_{new}(a|s)) = \sum_a \pi_{old}(a|s) log(\frac{\pi_{old}(a|s)}{\pi_{new}(a|s)}) \tag{3.18}$$

Dywergencja Kullbacka-Leiblera w kontekście uczenia przez wzmacnianie jest używana do:

- Regularizacji polityki - Ogranicza stopień smiany między starą a nową polityką, co pomaga uniknąć niestabilnych lub niepożądanych zachowań podczas trenowania,
- Kontrola eksploracji - Wspiera utrzymanie równowagi między eksploracją nowych akcji a wykorzystywaniem już poznanych strategii.

4 Implementacjia wybranych algorytmów uczenia przez wzmacnianie

Istnieje wiele algorytmów wykorzystywanych w uczeniu przez wzmacnianie, a konkretny wybór zależy głównie od charakterystyki środowiska oraz rodzaju problemu. Zwyczajowo dzieli się je na trzy główne kategorie:

- Algorytmy optymalizujące wartości (Value Optimization)
- Algorytmy optymalizujące politykę (Policy Optimization)
- Algorytmy imitacyjne (imitation)

4.1 Klasyfikacja algorytmów uczenia przez wzmacnianie

4.1.1 Algorytmy optymalizujące wartości (Value optimization)

W algorytmach tej grupy zasadniczym celem jest wyuczenie funkcji wartości, która pozwala oceniać jakości stanów lub akcji w danym czasie. Przykładem jest algorytm Q-Learning, koncentrujący się na wyznaczaniu funkcji $Q(s, a)$, wyrażającej oczekiwaną sumę zdyskontowanych nagród po wykonaniu akcji aa w stanie ss przy założeniu postępowania według polityki optymalnej.
Przykładowe algorytmy:

- Q-Learning
- Deep Q-Learning (DQN)
- double DQN
- dueling DQN

4.1.2 Algorytmy optymalizujące politkę (Policy Optimization)

Podejście to polega na bezpośredniej optymalizacji polityki, czyli reguły wyboru akcji w każdym stanie. Zamiast modelować funkcję wartości, dąży się do znalezienia strategii maksymalizującej oczekiwaną sumę nagród.
Przykładowe algorytmy:

- Policy Gradient Methods (REINFORCE)

- Advantage Actor-Critic (A2C)
- Asynchronous Adbantage Actor-Critic (A3C)
- Proximal Policy Optimization (PPO)

4.1.3 Algorytmy imitacyjne (imitation)

W podejściu imitacyjnym algorytm wzoruje się na działaniach eksperta (tzw. expert policy) i uczy się replikowania jego skutecznych zachowań bez konieczności przeprowadzania pełnej eksploracji środowiska.
Przykładowe algorytmy:

- Behavioral Cloning
- Inverse Reinforcement Learning (IRL)
- Generative Adversarial Imitation Learning (GAIL)

Podział algorytmów uczenia przez wzmacnianie na algorytmy optymalizujące wartości, algorytmy optymalizujące politykę oraz algorytmy imitacyjne jest powszechnie stosowany w literaturze naukowej [7].

4.2 Wybór algorytmów do implementacji

W ramach realizowanego projektu postanowiono skupić się na algorytmach wywodzących się z dwóch pierwszych grup, a więc Deep Q-Learning (DQN), Advantage Actor-Critic (A2C). Wybór wynika głównie z charakteru wybranego środowiska i celów badawczych związanych z grą Pong.

4.2.1 Dlaczego odrzucono klasyczną metodę Q-Learning?

Klasyczny Q-Learning, mimo że stanowi fundament uczenia przez wzmacnianie, bywa niewystarczający w bardziej złożonych środowiskach, na przykład w grach wideo. Istnieje kilka istotnych ograniczeń tej metody:

- Wysoka wymiarowość przestrzeni stanów - Gry Atari, w tym Pong, generują wielowymiarowe dane wejściowe, przez co tablicowe podejście do Q-funkcji staje się niepraktyczne.
- Brak generalizacji - Tablicowa wersja Q-Learningu nie potrafi przekładać wiedzy z jednych stanów na inne, co ogranicza szybkość i skuteczność nauki.
- Trudność z eksploracją - Klasowa metoda Q-Learningu wymaga rozbudowanej fazy eksploracyjnej, co powoduje duże zapotrzebowanie na czas i zasoby.
- Brak stabilności procesu uczenia - W dużych, dynamicznych przestrzeniach stanów uczenie metodą Q-Learning potrafi ulegać częstym fluktuacjom lub nawet nie osiągać zbieżności.

Z tych względów zdecydowano się na zastosowanie algorytmów, które wykorzystują sieci neuronowe w roli aproksymatora funkcji wartości. Tego rodzaju rozwiązania pozwalają na skuteczniejsze radzenie sobie z wysokowymiarowymi danymi.

4.3 Deep Q-Learning (DQN)

Deep Q-Learning (DQN) stanowi rozwinięcie klasycznej metody Q-Learning, w którym w miejsce tablicowej reprezentacji funkcji $Q(s, a)$ wykorzystuje się głęboką sieć neuronową. Pozwala to agentowi na efektywną naukę w środowiskach cechujących się złożonymi i licznie występującymi stanami.

4.3.1 Architektura modelu

Podstawą algorytmu DQN jest sieć neuronowa pełniąca funkcję aproksymatora wartości $Q(s, a; \theta)$. Najistotniejsze elementy tej architektury to:

- Sieć Q - Aproksymuje funkcję Q. Otrzymuje na wejściu reprezentację stanu (np. wycinek obrazu) i generuje estymowane wartości Q dla każdej możliwej akcji w tym stanie.
- Sieć docelowa - Kopia sieci Q, aktualizowana rzadziej niż zasadnicza sieć Q. Ma to na celu stabilizację treningu, ponieważ ogranicza wzajemne sprzężenie pomiędzy siecią Q a docelową wartością przy obliczaniu błędu.
- Bufor doświadczeń - Przechowuje pary $(s_t, a_t, r_{t+1}, s_{t+1})$, które pochodzą z kolejnych interakcji agenta ze środowiskiem. W trakcie uczenia próbki losuje się z bufora, dzięki czemu redukuje się korelację między kolejnymi próbkami.

Architektura została zaprezentowana na podstawie publikacji Mniha et al. [6]

4.3.2 Proces treningu algorytmu DQN

Trening DQN składa się z następujących kroków:

- Inicjalizacja:
  - Sieć Q - Losowa inizjalizacja wag θ,
  - Sieć docelowa - Ustawienie wag θ' równych θ,
  - Bufor doświadczeń - Utworzenie pustej struktury do magazynowania próbek (s, a, r, s')
- Wybór akcji (strategia ϵ-greedy):
  Akcja a_t w stanie s_t jest wybierana w sposób probabilistyczny:

$$a_t = \begin{cases} \text{Losowa akcja} & \text{z prawdopodobieństwem } \epsilon, \\ \arg\max_a Q(s_t, a; \theta) & \text{z prawdopodobieństwem } 1 - \epsilon. \end{cases}$$

- Interakcja ze środowiskiem:
  - Wykonanie akcji a_t, otrzymanie nagrody r_{t+1} i przejście do nowego stanu s_{t+1}
  - Zapis przejścia $s_t, a_t, r_{t+1}, s_{t+1}$ w buforze.
- Pobranie próbek z bufora:
  - Losowanie niewielkiej partii (mini-batch) przejść (s_i, a_i, r_i, s'_i)

– Jeżeli epizod zakończył się w tym kroku, to dla każdego z przejść oblicz wartość docelową $y = r$. W przeciwnym razie użyj wzoru $y_i = r_i + \gamma \max_{a'} Q(s'_i, a'_i; \theta')$, gdzie θ' to wagi sieci docelowej.

- Obliczanie straty (np. błędu średniokwadratowego):

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_i (Q(s_i, a_i; \theta) - y_i)^2 \tag{4.1}$$

- Zaktualizuj wartość $Q(s, a)$ za pomocą algorytmu stochastycznego spadku wzdłuż gradientu poprzez minimalizacje straty w odniesieniu do parametrów modelu.
- Aktualizacja wag sieci Q - Przypisanie θ do θ po ustalonej liczbnie kroków, co stabilizuje proces uczenia.
- Powtarzaj proces od punktu drugiego (Wybór akcji) do momentu osiągnięcia stabilnych wyników.

Opis procesu treningu został napisany na podstawie książki Maxima Lapana [8].

4.3.3 Przykładowa architektura sieci Q dla DQN

- Warstwa wejściowa - Wejście w postaci obrazu o rozmiarze 84x84 pikseli z 4 kanałami
- Pierwsza warstwa konwolucyjna - 32 filtry, rozmiar jądra 8x8, stride 4, aktywacja ReLU.
- Druga warstwa konwolucyjna - 64 filtry, rozmiar jądra 4x4, stride 2, aktywacja ReLU.
- Trzecia warstwa konwolucyjna - 64 filtry, rozmiar jądra 3x3, stride 1, aktywacja ReLU.
- Warstwa w pełni połączona - 512 neuronów, aktywacja ReLU.
- Warstwa wyjściowa - Liczbna neronów jest równa liczbie dostępnych akcji w środowisku, bez wykorzystania funkcji aktywacji

4.3.4 Zalety i wady DQN

Zalety:

- Skuteczne działanie w dużych przestrzeniach stanów dzięki zastosowaniu sieci neuronowych.
- Mechanizmy experience replay i target network wspierają stabilność procesu uczenia.

wady:

- Terning sieci DQN wymaga znaczących zasobów obliczeniowych
- Kluczowe w treningu sieci DQN jest dobranie odpowiednich hiperparametrów dla odpowiedniej optymalizacji, co może nieść za sobą spore trudności.
- Problemy z eksploracją. Model może utknąć w lokalnym minimum.
- Możliwość nadmiernego dopasowania do danych treningowych.

4.4 Advantage Actor-Critic (A2C)

Advantage Actor-Critc (A2C) to metoda, która łączy w sobie zalety metod opartych na polityce i opartych na wartściach. Pozwala ona na zmniejszczenie wariancji poprzez uzależnienie punktu odniesienia od stanu. Nagrodę można przedstawić jako wartość stanu plus przewaga akcji: $Q(s,a) = V(s) + A(s,a)$. A2C działa na zasadzie wykorzystania dwóch odzielnych sieci nauronowych: aktora odpowiedzialnego za wybór akcji oraz krytka oceniającego jakość stanu lub poprawność wyboru akcji, dostarczając użytecznych sygnałów uczących.

4.4.1 Architektura modelu

W modelu A2C zwykle wykorzystuje się jedną sieć neuronową (z ewentualnie współdzielonymi warstwami początkowymi), która rozgałęzia się na dwie części wyjściowe: Aktor, krytyk.

- Część współdzielona
  - Warstwa wejściowa - Przyjmuje stan środowiska s. W przypadku gier Atari jest to np. znormalizowany obraz w skali szarości o wymiarach 84 x 84 piksele. W przypadku środowisk wektorowych warstwa wejściowa może mieć postać zwykłego wektora cech.
  - Warstwy ukryte - Zwykle stosuje się kilka warstw konwolucujnuch do wyodrębniania cech z obrazu. Przy środowiskach o wejściu wektorowym wystarczające mogą być 203 warsty wpełni połączone.
  - Wspólna reprezentacja cech - Wyjście z konwolucji stanowi rdzeń, spółdzielony zarówno przez aktora jak i krytyka. Dzięki temu aktor i krytyk uczą się wspólnej reprezentacji stanu, co często poprawia efektywność i stabilność trenowania.
- Sieć aktora - Odpowiada za generowanie rozkładu prawdopodobieństwa akcji $\pi(a,s)$ w danym stanie s:
  - Warstwa wejściowa - Przyjmuje jako dane wejściowe stan środowiska.
  - Warstwy ukryte - Zawierają kilka warstw połączonych lub konwolucyjnych, których celem jest ekstrakcja cech oraz transformacja informacji.
  - Warstwa wyjściowa - Zwykle kończy się funkcja softmax, wyprowadzającą prawdopodobieństwo każdej możliwej akcji:

$$\pi(a|s) = \frac{exp(f(a,s))}{\sum_{a'} exp(f(a',s))} \tag{4.2}$$

    gdzie $f(a,s)$ jest wynikiem ostatniej warstwy sieci aktora.
- Sieć Krytyka - Jest odpowiedzialna za szacowanie wartości stanu $V(s)$ (a także przewagi $A(s,a)$, jeśli zostanie odpowiednio zaprojektowana). Ma to kluczowe znaczenie przy dostarczaniu trafnej oceny jakości działań aktora.
- Warstwa wejściowa - Przyjmuje stan środowiska jako dane wejściowe podobnie jak sieć aktoram, czasem dodaje się też dodatkową warstwę połączoną.

- Warstwa ukryta - Z reguły zbliżone do tych w sieci aktora, przetwarzające dane wejściowe.
- Warstwa wyjściowa - Ma postać pojedynczej jednostki (neuronu), której wartość numeryczna reprezentuje $V(s)$, czyli estymowaną wartość stanu:

$$V(s) = f_{critic}(s) \tag{4.3}$$

Architektura została zaprezentowana na podstawie publikacji Mniha et al. [2] oraz oficjalnej dokumentacji modelu A2C [10]

4.4.2 Proces treningu algorytmu A2C

Poniżej przedstawiono zarys treningu metody A2C, uwzględniający interakcję agenta ze środowiskiem, gromadzenie doświadczeń, obliczanie przewagi oraz aktualizację parametrów.

- Inicjalizacja - Nadanie losowych wartości parametrom θ. Zwykle wyróżnia się:
  - θ_π - parametry sieci aktora,
  - θ_v - parametry sieci krytyka.
- Wykonanie N kroków - Przy użyciu bierzącej polityki π_{θ_π}, Agent wykonuje N kroków w środowisku. Dla każdego kroku t zapisywane są: stan s_t, akcja a_t oraz nagroda r_t.
- Obliczenie wartości końcowej R - Jeżeli epizod uległ zakończeniu, przyjmuje się $R = 0$. W przeciwnym wypadku oblicza się wartość stanu końcowego s_{t+1} przy użyciu sieci wartości:

$$R = V_{\theta_v}(s_{t+1}). \tag{4.4}$$

  W przypadku zakończenia epizodu na przykład gdy gra się kończy przerywamy zbieranie danych wcześniej.
- Przetwarzanie wstecz i aktualizacja parametrów - Rozważając kroki wstecz od $t = t_N, t_{N-1}, ..., t_{start}$, aktualizuje się wartość R:

$$R \leftarrow r_t + \gamma R \tag{4.5}$$

  Następnie należy aktualizować gradnienty aktora i krytyka:
  - Gradient polityki:

$$\nabla\theta_\pi \leftarrow \nabla\theta_\pi log\pi_{\theta_\pi}(a_t|s_t)(R - V_{\theta_v}(s_t)) \tag{4.6}$$

  - Gradient wartości:

$$\nabla\theta_v\mathcal{L}_v \leftarrow \frac{\partial}{\partial\theta_v}(R - V_{\theta_v}(s_t))^2 \tag{4.7}$$

  Sumujemy powyższe gradienty dla aktora i krytyka w pamięci co w praktyce zbiera się je wektorem przez N kroków.
- Aktualizacja parametrów - Zaktualizować parametry sieci wykorzystując zsumowane gradienty. Wektor $\nabla\theta_\pi$ dodajemy do θ_π (maksymalizacja polityki) oraz wektor $\nabla\theta_v$ odejmujemy od θ_v (minimalizacja błędu wartości).
- Powtarzamy procedure z kroku 2 do momentu osiągnięcia konwergencji lub uzyskania założonych winików.

Opis procesu treningu został napisany na podstawie książki Maxima Lapana [8].

4.4.3 Zaleti i wady A2C

Zalety:

- Łączenie mocnych stron algorytmów opartych na wartości i na polityce, poprzez równoczesne optymalizowanie polityki i oceny stanów.
- Stabilność uczenia się dzięki wykorzystaniu przewagi $A(s, a)$ oraz regularyzacji entropii.
- Skuteczna eksploracja - regularizacja entropii zapobiega zbyt wczesnemu zafiksowaniu się na jednej strategii.
- Skalowalność - możliwe jest implementowanie wersji wielowątkowych w różnych środowiskach.

Wady:

- Złożoność obliczeniowa - Algorytm A2C wymaga trenowania dwóch oddzielnych sieci neuronowych dla aktora i krytyka co zwiększa wymagania obliczeniowej
- Wrażliwość na parametry takie jak współczynnik regulacji entropii β, współczynnik dyskonotwania γ oraz współczynnik uczenia α.
- Potencjalne problemy z równowagą między aktorem a krytykiem – niewłaściwa synchronizacja tych dwóch komponentów potrafi prowadzić do niestabilności uczenia.

5 Eksperymenty i analiza wyników

5.1 Konfiguracja środowiska testowego

Głównym celem badań jest przetestowanie wybranych algorytmów uczenia przez wzmacnianie w prostej grze typu Atari. Z tego powodu podjęto decyzję o wyborze gry Pong jako środowiska testowego. W celu uproszczenia procesu implementacji oraz uniknięcia nadmiarowego kodu zastosowano zasoby biblioteki Gymnasium, zapewniającej ujednolicony interfejs API umożliwiający interakcję agenta z otoczeniem.

5.1.1 Środowisk testowe: Pong

Pong jest popularną grą wideo z kategorii gier Atari, które idealnie się nadają do testowania algorytmów takich jak uczenie przez wzmacnianie, dlatego zdecydowano się na użycie tej gry. Biblioteka Gymnasium (następca biblioteki Gym) oferuje szeroką gamę możliwości testowych poprzez gre Pong. W projekcie wykrzystano środowisko PongNoFrameskip-v4, dostarczone przez bibliotekę Arcade Learning Environment [11]. Wiele algorytmów zostało przetestowanych i wykorzystanych jako benchmark w uczeniu maszynowych ze względu na prostotę implementacji biblioteki. Warto też zwrócić uwagę iż pong wymaga od agenta skutecznego podejmowania decyzji w czasie rzeczywistym co nadaje się idealnie do testów.

5.1.2 Charakterystyka środowiska Pong:

- Stan Środowiska - Stan odzwierciedla aktualny obraz z gry, o wymiarach 210 x 160 x3 (wysokość, szerokość, kanały RGB). W celu zwiększenia efektywności uczenia przeprowadza się wstępne przetwarzanie, zmniejszając rozdzielczość oraz upraszczając dane wejściowe.

- Zbiór akcji - Dla gry Pong mamy możliwość wykonania trzech akcji: przesunięcie paletki w góre, przesunięcie paletki w dół oraz pozostawienie paletki w miejscu.

- Nagrody - Sposób przyznawania nagród dla naszego środowiska wyraża się w następujący sposób: +1: Agent zdobywa punkt w momencie odbica piłki w taki sposób aby przeciwnik nie był w stanie jej odbić, -1: W momencie gdy agent nie doła odbić piłki przeciwnik otrzymuje punkt, 0: Dla pozostałych przypadków np: w trakcie wymiany odbić.

- Warunek końca epizodu - Koniec jednego epizodu uczenia następuje w momencie, gdy agent lub jego przeciwnik osiągnie 21 punktów.

- Cel agenta - Maksymalizacja całkowitej zdyskontowanej nagrody w trakcie jednego epizodu, co jest równoznaczne z wygrywaniem większą liczbą punktów niż przeciwnik.

5.1.3 Język programowania: Python [12]

W ramach implementacji algorytmów uczenia przez wzmacnianie zdecydowano się na użycię języka programowania Python. Głównymi aspektami przemawiającym za wybraniem tego języka są przede wszystkim szeroka gama bilbiotek wspierających uczenie przez wzmacnianie np. Gymnasium, Pytorch. Python jest najpopularniejszym językiem sotosowanym w dziedzinie sztucznej inteligencji i uczenia przez wzmacnianie. Duża część współczesnych rozwiązań w AI jest implementowana właśnie w Pythonie, co przekłada się na bogate wsparcie społeczności.

5.1.4 Biblioteki wykorzystane do implementacji

- Gymnasium [13] - Biblioteka stanowiąca standard dla uczenia przez wzmacnianie służąca do symulacji środowsik. Zastosowaną ją głównie w celu dostarczenia środowiska gry pong, oraz łatwości w zapewnieniu interfejsu dla agenta który ma mu służyć jako interakcja ze środowiskiem. Biblioteka oferuje prostą interakcję z różnymi algorytmami uczenia przez wzmacnianie.

- PyTorch [14] - Pozwala na implementacje złożonych modeli uczenia przez wzmacnianie opartych na sieciach neuronowych za pomocą kilku linijek kodu. PyToch zapewnia dwie wysokopoziomowe funkcję: Obliczenia tensorowe z silną akceleracją przy pomocy wykorzystania procesorów graficznych, Wykorzystanie głębokich sieci neuronowych zaprojektowanych za pomocą taśmowych systemów automatycznego różnicowania.

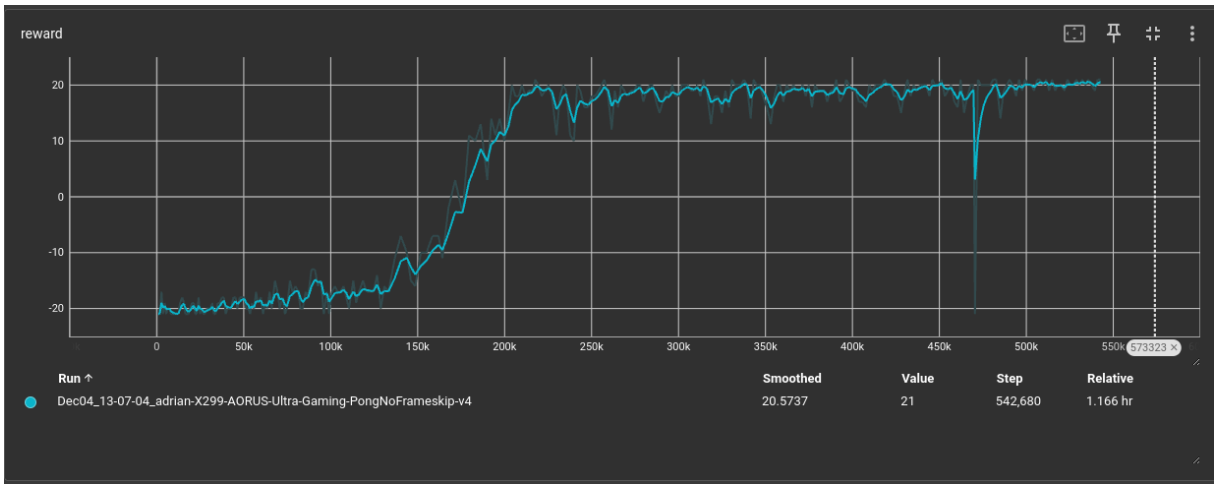
- OpenCV [15] - Zestaw narzędzi pozwalający na przetwarzanie obrazu oraz wizję komputerową wizję zadań. Dzięki wykorzystaniu tej biblioteki jesteśmy w stanie

osiągnąć szybkie iw wydajne przetwarzanie danych wizualizacyjnych przed przesłaniem ich do sieci neuronowej, dzięki czemu uzyskujemy pozytywny efekt w postaci czasu treningu oraz jego efektywności.

5.2 Wyniki dla modelu Deep Q-Learning

Model Deep Q-Learning (DQN) został zaimplementowany w celu wytrenowania agenta do gry Pong przy użyciu sieci neuronowej do aproksymacji funkcji Q. Architektura algorytmu opiera się na bibliotekach PyTorch, Gymnasium oraz OpenCV. Dodatkowo środowisko Pong poddano wstępnemu przetwarzaniu (tzw. wrappery), aby usprawnić proces uczenia.

5.2.1 Analiza wykresu nagordy

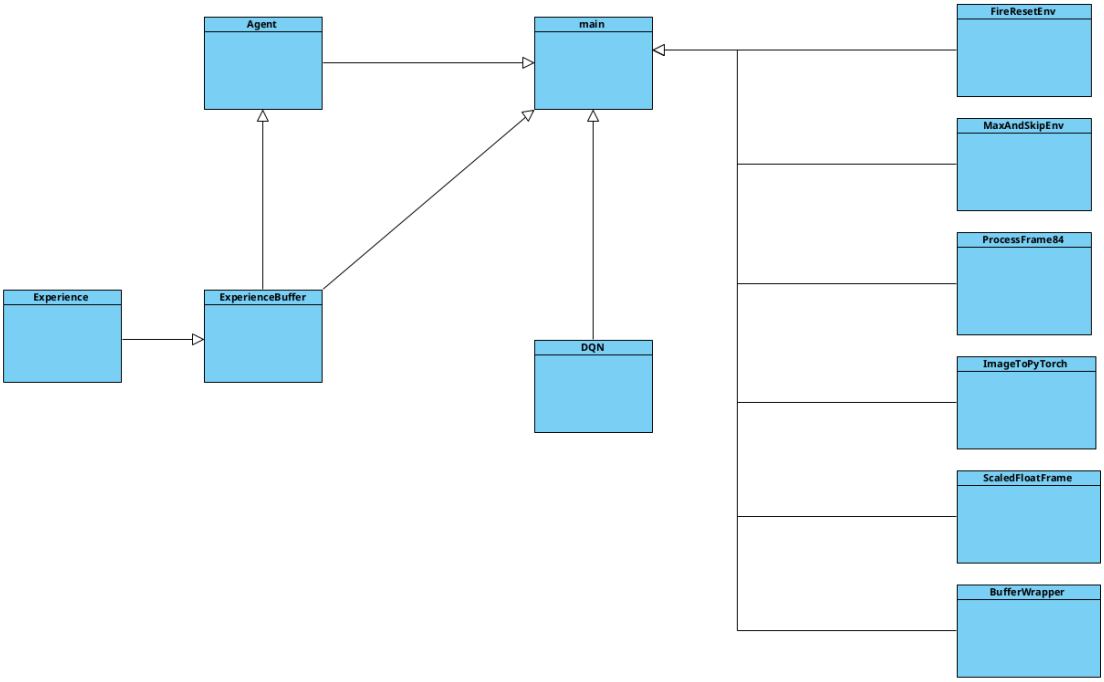


Rysunek 2: Opis obrazka

Na podstawię poniższego wykresu przedstawiającego proces uczenia modelu Deep Q-Learning widzimy, że na początku treningu agent wykonuje losowe ruchy czyli ekspoloruje środowisko, co jest adekwatne do otrzymywania nagród na poziomie około -21 czyli maksymalnej możliwej przegranej. Następnie następuje powolny wzrost wartości nagród co wskazuje na powolne szukanie podstawowych strategii przez agenta. W momencie około 100 000 kroków treningowych następuje gwałtowny wzrost otrzymywanej średniej nagrody agenta co wskazuje na stopniową naukę strategii gry. W okolicach około 175 000 kroków średnia nagroda zaczyna przekraczać punkt 0 co oznacza iż agent zaczyna wygrywać więcej razy w trakcie jednego epizodu gry. W momencie około 300 000 kroków można zaobserwować osiągnięcie stopniowej stablności wyników, zbliżając się do maksymalnej średniej nagrody wynoszącej +21. W kolejnych krokach widać niewielkie wachania wyników co jest związane z stochastyczną naturą dynamicznego środowiska gry Pong. Proces treningu został zakończony w ciągu około 540 000 kroków, przy czasie trwania procesu uczenia wynoszącym 1,166 godziny czasu rzeczywistego.

5.2.2 Strukutra projektu

W niniejszej części przedstawiono uproszczony diagram klas (w notacji UML) ilustrujący kluczowe elementy architektury implementacji algorytmu Deep Q-Learning dla gry Pong. Diagram zawiera zarówno klasy związane z gromadzeniem i przetwarzaniem danych (m.in. bufor doświadczeń i tak zwane wrappery), jak również samą sieć neuronową (DQN) i logikę podejmowania decyzji przez agenta. Dzięki tak zorganizowanej strukturze możliwa jest wieloetapowa analiza sekwencji obserwowanych stanów w środowisku oraz generowanie akcji w oparciu o metodologie uczenia ze wzmocnieniem.



Rysunek 3: Diagram stuktury projektu

- Main - Moduł główny odpowiada za inicjalizację środowiska Pong, wykorzystując dedykowane wrappery ułatwiające proces uczenia. W jego obrębie tworzone są również wszystkie obiekty niezbędne do trenowania modelu: sieć DQN, bufor doświadczeń oraz agent. Moduł zarządza pętlą treningową - w każdej iteracji agent podejmuje akcje w środowisku, kolekcjonuje nowe doświadczenia i przekazuje je dalej do procesu uczenia. Ponadto, w pliku tym ustala się podstawowe hiperparametry (np. współczynnik uczenia, początkowy poziom eksploracji ϵ czy rozmiar partii treningowej w buforze doświadczeń) oraz definiuje sposób rejestrowania postępów treningu.

- Agent - Klasa Agent stanowi łącznik między środowiskiem a siecią Q. Jej zadaniem jest wybór akcji na bazie przyjętej polityki (np. ϵ-greedy) oraz monitorowanie bieżącego stanu gry, w tym gromadzenie nagród i informacji o zakończeniu epizodów. Agent organizuje również zapisywanie doświadczeń w buforze, co umożliwia efektywniejsze uczenie modelu w oparciu o różne fragmenty rozgrywki.

- ExperienceBuffer - zapewnia strukturę do przechowywania doświadczeń generowanych przez agenta. Dzięki temu możliwe jest wielokrotne wykorzystanie zapisanych informacji podczas uczenia, co wspomaga stabilność procesu treningowego i poprawia jakość nabywanych przez sieć neuronową reprezentacji.

- Experience - Pojedynczy krok w środowisku (tzw. transition) został zdefiniowany w klasie Experience. Zawiera ona kluczowe dane opisujące stan przed podjęciem akcji, stan po jej wykonaniu, otrzymaną nagrodę oraz informację o ewentualnym zakończeniu epizodu. Każdy obiekt tej klasy wykorzystywany jest następnie przez bufor doświadczeń podczas przygotowywania próbek treningowych dla sieci DQN.

- DQN - Sieć DQN służy do aproksymacji funkcji Q. W opisywanym projekcie wykorzystano sieć konwolucyjną, zaprojektowaną do analizy sekwencji obrazów pochodzących z rozgrywki. Przetworzone przez sieć dane wejściowe umożliwiają wyznacze-

nie wartości Q dla wszystkich możliwych akcji. Agent, na podstawie tych wartości, wybiera ruch dążący do maksymalizacji skumulowanej nagrody w dłuższym horyzoncie czasowym.

- MaxAndSkipEnv - Mechanizm pozwalający pominąć część klatek i jednocześnie zsumować odpowiadające im nagrody. Wykorzystuje on maksymalne wartości pikseli z dwóch ostatnich obserwacji, co prowadzi do redukcji liczby kroków przetwarzanych podczas treningu i tym samym zwiększa efektywność obliczeń.

- FireResetEnv - odpowiada za inicjalizację rozgrywki przez wymuszenie akcji FIRE (oraz ewentualnie innej) na początku każdego epizodu. Dzięki temu mechanizmowi agent może od razu rozpocząć właściwą interakcję z grą.

- ProcessFrame84 - Ten komponent przetwarza pojedynczą klatkę na obraz w odcieniach szarości oraz skaluje go do rozmiaru 84x84. Pozwala to znacząco zmniejszyć wymiarowość danych wejściowych, co korzystnie wpływa na szybkość i stabilność treningu

- ImageToPyTorch - Zadaniem ImageToPyTorch jest zmiana kolejności wymiarów klatki z postaci (wysokość, szerokość, kanały) na (kanały, wysokość, szerokość).Ułatwia to dalsze przetwarzanie obrazu w bibliotece PyTorch, w której standardem jest inna konwencja wymiarów tensora.

- ScaledFloatFrame - Normalizuje wartości pikseli do zakresu [0,1].

- BufferWrapper - Umożliwia zbudowanie stosu ostatnich kilku klatek, co pozwala sieci neuronowej wyłapać krótkoterminową dynamikę obiektów w grze (ruch piłki i paletki). Dzięki temu agent dysponuje kontekstem czasowym, niezbędnym w zadaniach związanych z przetwarzaniem serii obrazów.

Współdziałanie powyższych elementów skutkuje iteracyjnym procesem treningowym, w którym agent stopniowo udoskonala strategię gry w Pong, korzystając zarówno z bieżących obserwacji, jak i zróżnicowanego zasobu doświadczeń przechowywanych w buforze. Sieć DQN oparta na warstwach konwolucyjnych czerpie informacje z wielu typów danych, co pozwala ostatecznie na wypracowanie stabilnej i efektywnej polityki rozgrywki.

Pełną implementację kodu można znaleźć na repozytorium github [16] Kod został zainspirowany rozwiązaniami przedstawionymi w książce Maxima Lapana [8] i dostosowany do najnowszych wersji bibliotek.

5.2.3 Problem przetrenowania modelu

W trakcie testowania modelu poprzez bezpośrednią obserwację rozgrywki w formie aplikacji można zauważyć, że modele osiągające średni wynik w przedziale 10–21 przejawiają charakterystyczny styl gry. Mimo wysokiej skuteczności w środowisku Pong, agent wykonuje ruchy przewidujące zachowanie przeciwnika – tego samego, który służył do trenowania. Takie zjawisko jest przejawem nadmiernego dopasowania modelu do danych treningowych, powszechnie określanego jako przetrenowanie (overfitting).

Przyczyny przetrenowania modelu DQN:

- Ograniczona różnorodność danych w buforze powtórki - Rozmiar bufora (10 000) sprawia, że może on zostać zdominowany przez powtarzające się wzorce rozgrywki, co ogranicza ekspozycję modelu na bardziej nietypowe sytuacje. W momencie gdy agent dominuję daną strategię gry, bufor może być wypełniony głównie przykładami wspierającymi taką startegię, co prowadzi do utraty różnorodności danych. W praktyce agent uczy się przewidywania konkretnych scenariuszy, które często występują w buforze, co skutkuje brakiem przygotowania na bardziej niestandardowe sytuacje.

- Eksploatacja kosztem eksploracji - Podczas późniejszych etapów uczenia, gdy wartość ϵ w strategii epsilon-greedy spada do 0.01, gent rzadko wybiera akcje losowe, co powoduje utrwalenie się wyuczonych schematów i brak odkrywania alternatywnych strategii.

- Brak elementu stochastyczności w wyborze akcji - Wybór akcji w modelu DQN dokonuje się na podstawie maksymalizacji wartości Q, co sprzyja sztywnemu dopasowaniu do konkretnych sekwencji stan - akcja.

- Brak mechanizmów zapobiegających przetrenowaniu - Ze względu na swoją naturę model DQN nie uwzględnia mechanizmów regulujących eksplorację (np. entropii polityki)

Zastosowanie Generatywnych Sieci Przeciwstawnych (GAN) jako możliwe rozwiązanie problemu przetrenowania - Jedną z możliwych strategii radzenia sobie z przetrenowaniem jest poszerzenie zakresu danych treningowych przy pomocy generatywnych sieci przeciwstawnych (GAN). W tym kontekście mogą one posłużyć do tworzenia nowych, trudniejszych do przewidzenia trajektorii rozgrywki w środowisku Pong. W rezultacie agent zostaje zmuszony do wypracowania bardziej uniwersalnych działań. Wprowadzenie losowości w stanach i akcjach, które rzadko pojawiają się w standardowym treningu, pomaga uniknąć zbyt wąskiego dopasowania. Mimo to, wdrożenie GAN w połączeniu z modelem DQN jest z technicznego punktu widzenia złożonym zadaniem, wymagającym precyzyjnie dobranych hiperparametrów. Z tego powodu w niniejszym projekcie zdecydowano się na opracowanie innego modelu (A2C), który w praktyce okazał się łatwiejszy do implementacji przy zachowaniu zadowalającej jakości działania.

5.2.4 Tabela z wynikami dla różnych hiperparametrów

Hiperparametr	Zmiana	Czas	Liczba kroków	Uwagi
γ	0.95	$\sim$ 50min	$\sim$ 450000	Szybsza konwergencja, mniejsze nagrody.
BATCH_SIZE	64	$\sim 2,6$ godziny	$\sim$ 850000	Stabilniejsze wyniki, wolniejsze uczenie.
REPLAY_SIZE	50,000	$\sim 2,5$ godziny	$\sim$ 1100000	Większa różnorodność danych, dłuższy czas treningu.
LEARNING_RATE	0.0002	N/A	N/A	Brak konwegencji.
LEARNING_RATE	0.00015	N/A	N/A	Brak konwegencji.
EPSILON_DECAY	300,000	$\sim 2,2$ godziny	$\sim$ 1000000	Większa eksploracja, dłuższy czas do konwergencji.
SYNC_TARGET_FRAMES	2,000	N/A	N/A	Brak konwergencji.

Tabela 1: Wyniki eksperymentów dla różnych hiperparametrów.

5.2.5 Wnioski

W przeprowadzonych eksperymentach algorytm Deep Q-Learning (DQN) wykazał się wysoką skutecznością w nauce gry Pong, osiągając maksymalną średnią nagrodę na poziomie +21. Taki wynik potwierdza, że agent zdołał w pełni opanować reguły i dynamikę środowiska. Proces uczenia przebiegał zgodnie z założeniami początkowe wyniki były niskie z uwagi na losową eksplorację, a w miarę postępu treningu agent sukcesywnie rozwijał strategię, aż do zaobserwowania stabilizacji po około 300 000 kroków. Istotną rolę odegrał bufor powtórki, który pozwolił na wielokrotne wykorzystanie zebranych doświadczeń. Zastosowanie polityki ϵ -greedy przyczyniło się natomiast do zachowania równowagi między eksploracją nowych możliwości a eksploatacją już wytrenowanych schematów postępowania. Mimo zadowalających rezultatów, w późniejszych etapach pojawiło się jednak zjawisko przetrenowania, przejawiające się w postaci nienaturalnie przewidujących ruchów agenta. Głównym powodem było ograniczenie różnorodności danych w buforze (zwłaszcza w obliczu malejącego parametru ϵ), co z czasem prowadziło do zawężenia zakresu eksplorowanych strategii. W odpowiedzi na problem przetrenowania podjęto decyzję o użyciu bardziej zaawansowanego algorytmu, takiego jak A2C (Advantage Actor-Critic), który umożliwia lepsze wyważenie eksploracji i eksploatacji dzięki wprowadzeniu elementu sto-

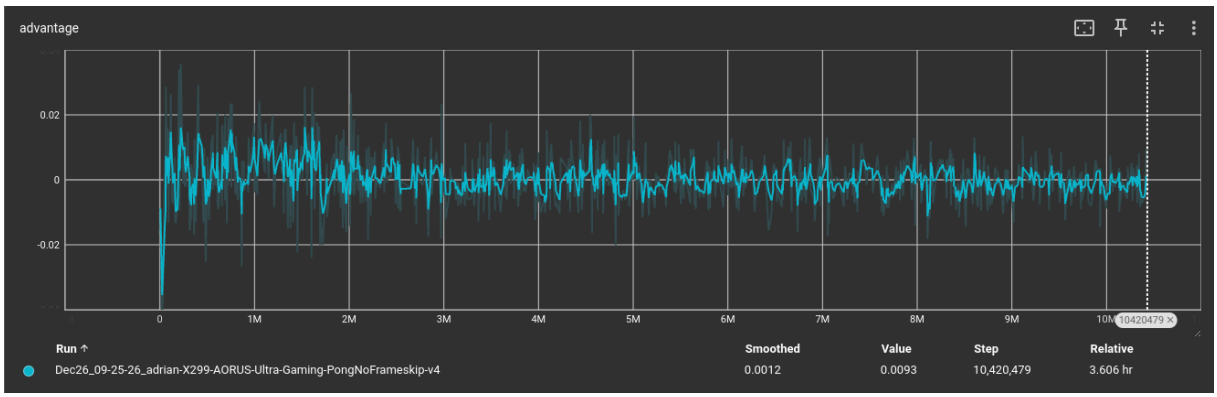
chastyczności w polityce. Podsumowując, DQN pozostaje wartościowym punktem wyjścia w badaniach nad grami Atari, niemniej jego praktyczne zastosowania wymagają zarówno dodatkowych mechanizmów ograniczających przetrenowanie, jak i starannej kalibracji hiperparametrów. Te wymogi w sposób naturalny przekładają się na konieczność zaangażowania znacznych zasobów obliczeniowych oraz planowania eksperymentów w sposób maksymalnie metodyczny.

5.3 Wyniki dla modelu Advantage Actor-Critic (A2C)

Poniżej przedstawiono główne wyniki i wizualizacje procesu uczenia modelu A2C, takie jak wykresy przewagi, zmiany funkcji nagrody, gradienty oraz rozmaite funkcje strat. Każdy z tych elementów pozwala z innej perspektywy ocenić, w jaki sposób agent przyswaja strategię gry Pong.

5.3.1 Analiza wykresów dla modelu A2C

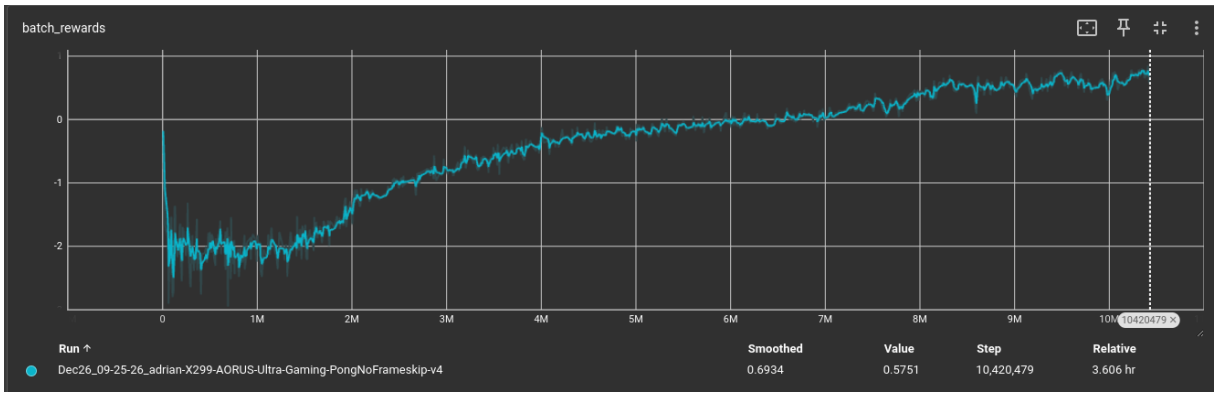
Wykres przewagi ($A(s, a)$)



Rysunek 4: Opis obrazka

Wartość przewagi reprezentuje różnicę między wartością konkretnej akcji w danym stanie a uogólnioną wartością tego stanu. Odgrywa kluczową rolę w uczeniu A2C, ponieważ pomaga ograniczyć wariancję w estymacji wartości akcji. Wahania w początkowej fazie świadczą o tym, że agent wciąż uczy się oceniać akcje w poszczególnych stanach, co skutkuje zmiennością przewag. Stopniowa stabilizacja sugeruje, że model z czasem poprawnie identyfikuje wartości akcji, co przekłada się na lepszą politykę w dłuższej perspektywie.

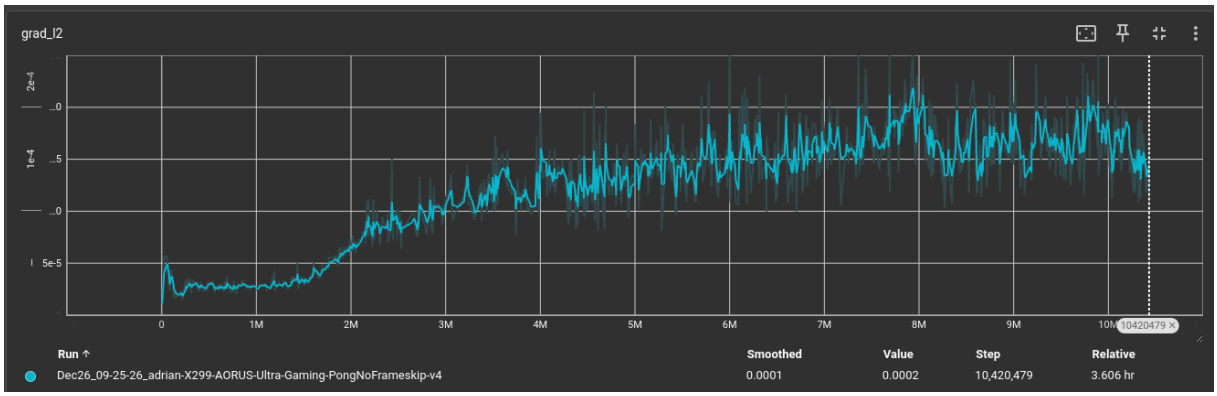
Wykres średniej wartości paczki



Rysunek 5: Opis obrazka

Ilustruje uśrednione nagrody osiągane przez agenta w kolejnych epizodach: Początkowy wzrost wskazuje na szybkie dostosowywanie się modelu do wymagań środowiska. Stały wzrost wartości nagród w dalszej fazie sugeruje, że agent coraz lepiej rozumie otoczenie i opracowuje skuteczniejsze taktyki gry.

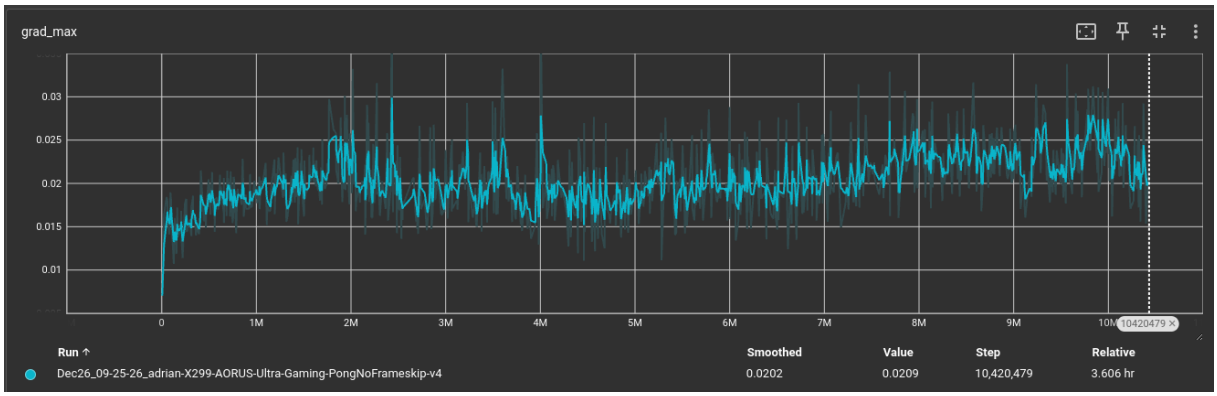
Wykres Normy L2 gradnientu



Rysunek 6: Opis obrazka

Norma L2 gradientów określa siłę aktualizacji parametrów modelu. Zbyt duże wartości mogą prowadzić do niestabilności lub znacznych fluktuacji wag.Zbyt małe wartości utrudniają osiągnięcie konwergencji.Zaobserwowana stabilizacja norm gradientów w trakcie treningu świadczy o prawidłowo przeprowadzanym procesie optymalizacji.

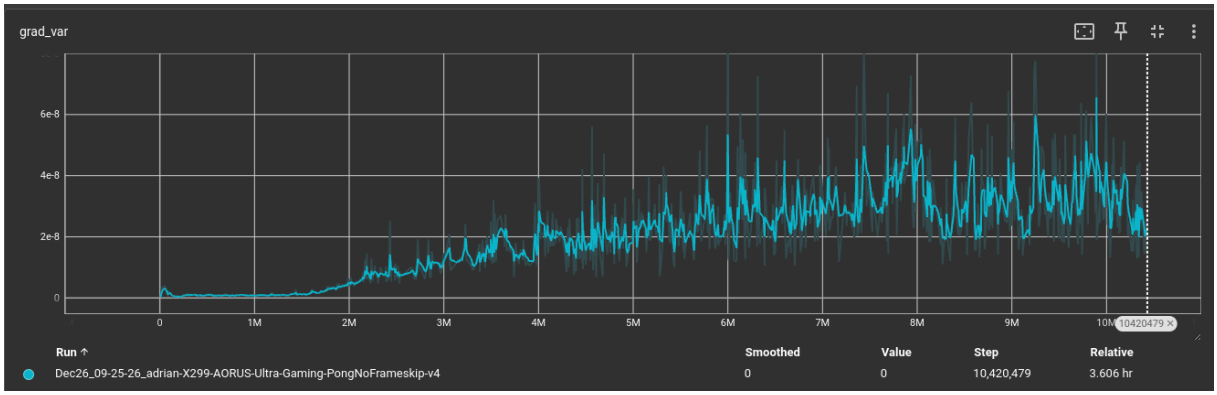
Wykres maksymalnych gradientów



Rysunek 7: Opis obrazka

Maksymalne wartości gradientów wskazują na bardziej znaczące zmiany parametrów podczas trenowania modelu. Stabilizacja ich w późniejszym procesie treningu sugeruje, iż model zaczyna osiągać równowagę w uczeniu oraz dostosowywaniu się do środowiska.

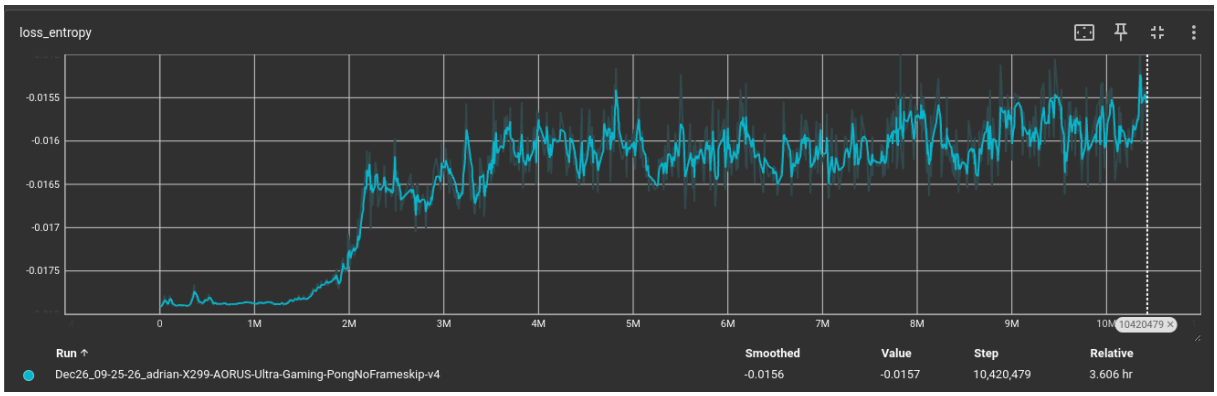
Wykres wariancji gradientów



Rysunek 8: Opis obrazka

Wariancja gradientów obrazuje, jak bardzo zróżnicowane są gradienty w kolejnych aktualizacjach. Wzrost wariancji może świadczyć o intensywnym poszukiwaniu nowych, lepszych strategii. Stopniowy wzrost pod koniec treningu oznacza większą pewność modelu co do wypracowanych rozwiązań.

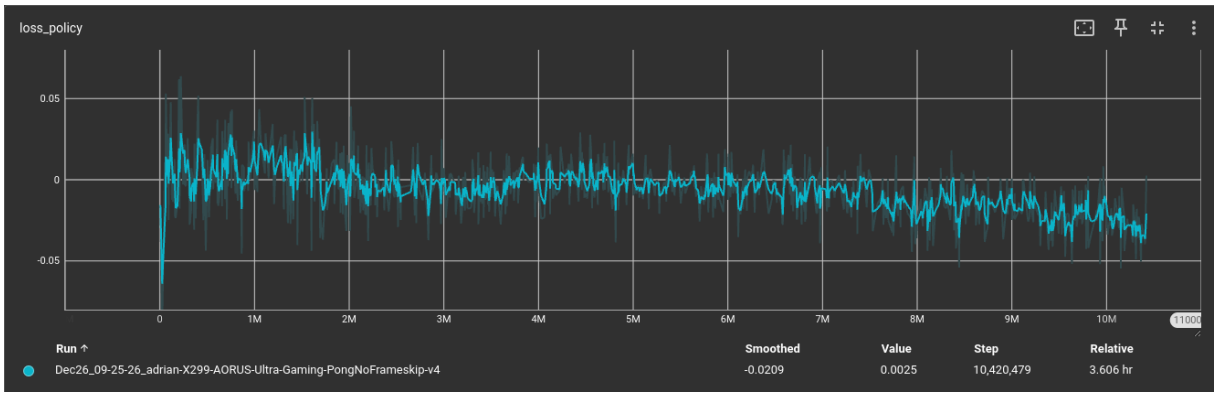
Wykres straty entropii



Rysunek 9: Opis obrazka

Strata entropii mierzy poziom eksploracji w polityce agenta. Zmniejszająca się wartość entropii w procesie treningu modelu oznacza, że model staje się bardziej pewny w procesie podejmowania decyzji. Początkowo wysoka entropia wskazuje na eksplorację, następnie jej spadek w późniejszych etapach wskazuje na stabilizację polityki. Zasadniczo oznacza to, że podczas gdy polityka zaczyna się zmieniać, agent staje się coraz bardziej pewny akcji, które wykonuje.

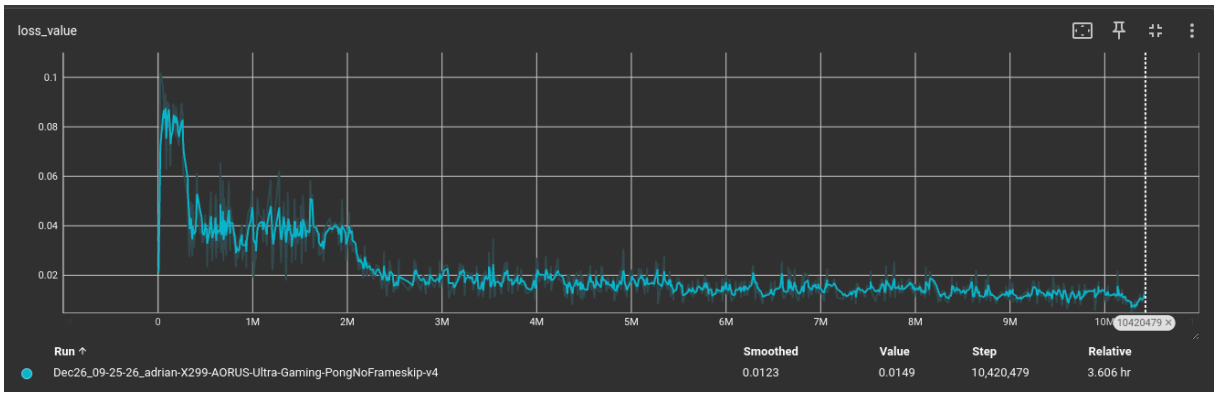
Wykres straty polityki



Rysunek 10: Opis obrazka

Przedstawia, w jakim tempie i w jakim kierunku dostosowują się parametry sieci aktora. Znaczne zmiany w początkowej fazie typowe dla etapu intensywnej eksploracji i poszukiwania lepszych akcji. Stabilizacja w późniejszej fazie świadczy o zbliżaniu się do polityki zoptymalizowanej (lub lokalnie zoptymalizowanej). Jest to zjawisko pozytywne.

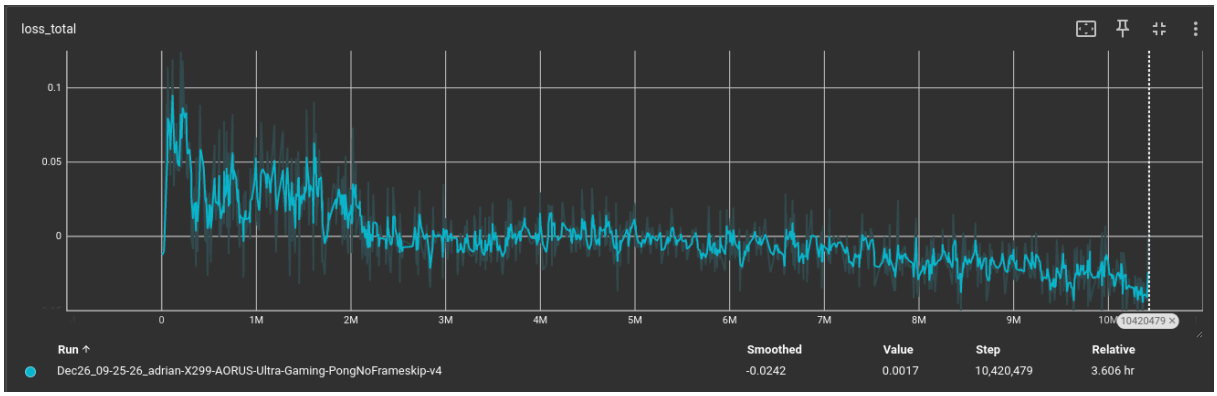
Wykres straty wartości



Rysunek 11: Opis obrazka

Odzwierciedla różnicę między przewidywaną a rzeczywistą wartością stanu. Systematyczny spadek: wskazuje, że sieć krytyka coraz trafniej przewiduje wartość stanu. Poprawa estymacji wskazuje na lepiej wyestymowana wartość stanu umożliwia szybszą i bardziej precyzyjną aktualizację polityki.

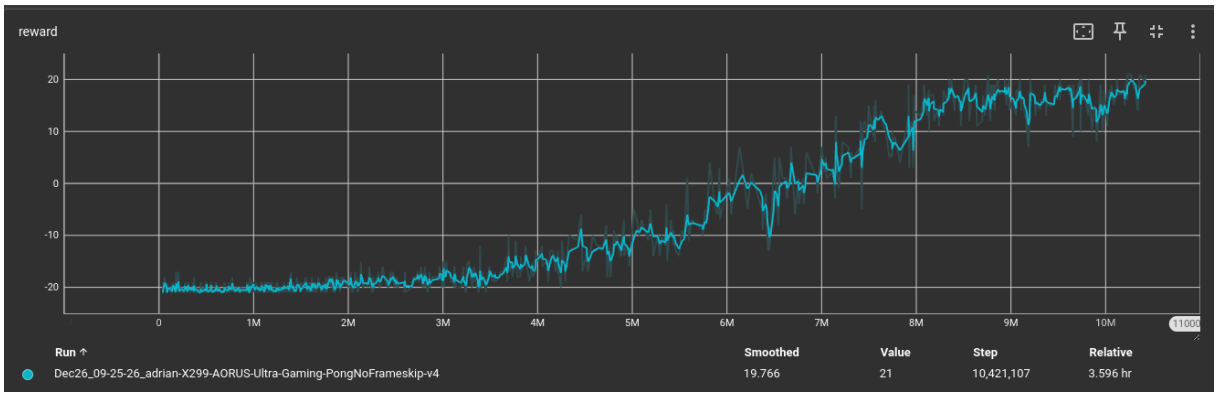
Wykres całkowitej straty



Rysunek 12: Opis obrazka

Łączna strata to kombinacja strat polityki, wartości oraz entropii. Ma za zadanie ona odzwierciedlić ogólny koszt optymalizacji procesu. Malejący trend pokazuje, że algorytm skutecznie minimalizuje błąd funkcji kosztu. Stabilizacja pod koniec sugeruje zbliżanie się do równowagi między eksploracją a eksploatacją.

Wykres średnich wartości nagród

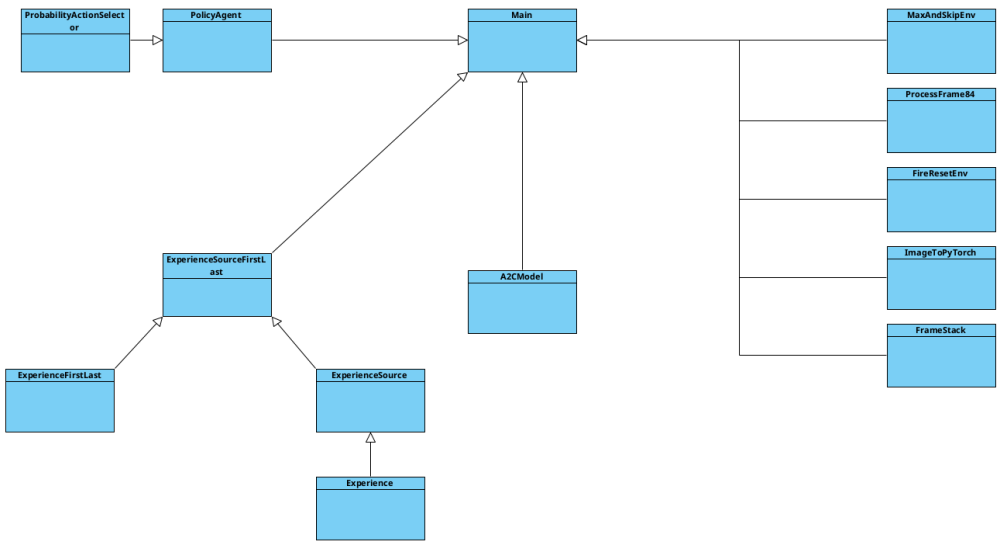


Rysunek 13: Opis obrazka

Wartość nagrody dla każdego kolejnego epizodu procesu uczenia ilustruje efektywność modelu. Niskie wartości na początku są typowe dla fazy eksploracji i braku wyuczonej strategii. Stopniowy wzrost świadczy o ciągłym ulepszaniu polityki, co skutkuje wyższą liczbą punktów zdobywanych przez agenta. Osiągnięcie maksymalnych wartości (ok. +21) oznacza pełne opanowanie środowiska Pong (lub osiągnięcie wyniku zbliżonego do perfekcji).

5.3.2 Struktura projektu

Poniższy diagram UML ilustruje przykładową implementację algorytmu Advantage Actor-Critic (A2C), bazując na analizie załączonych plików. Diagram koncentruje się na głównych klasach i ich wzajemnych relacjach, nie wchodząc w szczegółowe opisy wszystkich metod wewnętrznych. Po przedstawieniu diagramu zamieszczono opis zasadniczych elementów projektu.



Rysunek 14: Diagram stuktury projektu modelu A2C

- Main - W tym module inicjalizowane jest środowisko i koordynowany zostaje proces uczenia. Tworzone są tu liczne obiekty (m.in. A2CModel, PolicyAgent, ExperienceSourceFirstLast), a następnie uruchamiana jest główna pętla treningowa. Pod-

czas tej pętli agent iteracyjnie gromadzi doświadczenia, oddziałując ze środowiskiem gry Pong, zaś sieć (A2C) jest regularnie aktualizowana przy użyciu optymalizatora (funkcja Adam). Moduł odpowiada również za zapisywanie wyników oraz ewentualne wczytywanie wytrenowanych modeli.

- A2CModel - Jest to sieć neuronowa łącząca w jednej architekturze funkcje aktora i krytyka. Sieć konwolucyjna służy do przetwarzania surowych obrazów pochodzących z gry. Rozdzielenie wyjść (dla warstwy aktora oraz krytyka) umożliwia równoczesne wyznaczanie prawdopodobieństw akcji i estymację ich oczekiwanej wartości.

- PolicyAgent - Klasa pełniąca rolę pośrednika między środowiskiem a modelem. Na podstawie bieżącej obserwacji otrzymuje z sieci A2C rozkład prawdopodobieństwa wybrania poszczególnych akcji i dokonuje ostatecznego wyboru ruchu, który zostaje wykonany w środowisku.

- ProbabilityActionSelector - Obiekt ten otrzymuje rozkład prawdopodobieństwa akcji wygenerowany przez sieć i na tej podstawie losuje konkretną akcję do wykonania. Rozwiązanie takie wprowadza element stochastyczności, wspierając eksplorację środowiska i redukując ryzyko zbyt wczesnej stabilizacji strategii.

- Experience - Struktura przechowująca informacje dotyczące pojedynczego kroku w środowisku: obserwacji, wybranej akcji oraz otrzymanej nagrody. Upraszcza zarządzanie historią rozgrywki i umożliwia przejrzyste gromadzenie danych.

- ExperienceFirstLast - Rozszerzona wersja rekordu doświadczenia, w której oprócz standardowych informacji przechowywane jest także odniesienie do stanu końcowego. Ułatwia to obliczanie zdyskontowanych sum nagród w podejściu A2C, szczególnie przydatnych do treningu krytyka.

- ExperienceSource - Podstawowy generator strumienia doświadczeń. Scala on dane z wielu środowisk oraz integruje działania agenta, zarządzając uruchamianiem kolejnych epizodów, gromadzeniem sygnałów zwrotnych oraz przygotowywaniem sekwencji kroków niezbędnych do dalszej obróbki.

- ExperienceSourceFirstLast - Specjalizowana wersja generatora, wykorzystująca strukturę ExperienceFirstLast. Pozwala na pobieranie niewielkich sekwencji kroków, jednocześnie automatycznie obliczając zdyskontowane sumy nagród. Zabieg ten wpisuje się w schemat A2C, gdzie istotne jest uwzględnienie wartości przyszłych nagród.

- MaxAndSkipEnv - Mechanizm pozwalający pominąć część klatek i jednocześnie zsumować odpowiadające im nagrody. Wykorzystuje on maksymalne wartości pikseli z dwóch ostatnich obserwacji, co prowadzi do redukcji liczby kroków przetwarzanych podczas treningu i tym samym zwiększa efektywność obliczeń.

- FireResetEnv - odpowiada za inicjalizację rozgrywki przez wymuszenie akcji FIRE (oraz ewentualnie innej) na początku każdego epizodu. Dzięki temu mechanizmowi agent może od razu rozpocząć właściwą interakcję z grą.

- ProcessFrame84 - Ten komponent przetwarza pojedynczą klatkę na obraz w odcieniach szarości oraz skaluje go do rozmiaru 84x84. Pozwala to znacząco zmniejszyć wymiarowość danych wejściowych, co korzystnie wpływa na szybkość i stabilność treningu

- ImageToPyTorch - Zadaniem ImageToPyTorch jest zmiana kolejności wymiarów klatki z postaci (wysokość, szerokość, kanały) na (kanały, wysokość, szerokość).Ułatwia to dalsze przetwarzanie obrazu w bibliotece PyTorch, w której standardem jest inna konwencja wymiarów tensora.

- FrameStack - Utrzymuje stos kilku ostatnich klatek (np. czterech) i traktuje je jako pojedynczą obserwację, co pomaga sieci uchwycić krótkoterminową dynamikę obiektów w grze (ruch piłki czy paletki).

Opisana struktura umożliwia agentowi otrzymywanie przetworzonych danych o stanie gry, na podstawie których sieć neuronowa A2C (działająca w roli aktora i krytyka) wyznacza optymalne akcje. Trening polega na iteracyjnym aktualizowaniu wag w celu minimalizacji błędu estymacji wartości oraz maksymalizacji skumulowanej nagrody zdobywanej przez agenta. Połączenie architektury aktor - krytyk ze starannie zorganizowanym mechanizmem gromadzenia obserwacji zapewnia stabilną optymalizację i umożliwia efektywne opanowanie gry Pong.

Pełną implementację kodu można znaleźć na repozytorium github [17] Kod został zainspirowany rozwiązaniami przedstawionymi w książce Maxima Lapana [8] i dostosowany do najnowszych wersji bibliotek.

5.3.3 Tabela z wynikami dla różnych hiperparametrów

Hiperparametr	Zmiana	Czas	Liczba kroków	Uwagi
γ	0.95	~ 3 godziny	~ 8500000	Szybsza konwergencja, mniejsze nagrody.
BATCH_SIZE	64	~ 2 godziny	~ 6000000	Mniej stabilne wyniki, szybsze uczenie.
BATCH_SIZE	32	$\sim 1,7$ godziny	~ 4300000	Mniej stabilne wyniki, szybsze uczenie.
REPLAY_SIZE	50,000	$\sim 2,5$ godziny	~ 1100000	Większa różnorodność danych, dłuższy czas treningu.
LEARNING_RATE	0.002	5500000	1,8 godziny	Szybsza konwergencja, większe wahania wyników.
LEARNING_RATE	0.003	N/A	N/A	Brak konwegencji.
ENDROPY_BETA	0.03	$\sim 4,5$ godziny	$\sim$ 13500000	Większa eksploatacja kosztem eksploracji, dłuższy czas konwergencji.

Tabela 2: Wyniki eksperymentów dla różnych hiperparametrów.

5.3.4 Wnioski

Przedstawiony algorytm A2C pozwala na pomyślne wytrenowanie agenta w środowisku Pong i uzyskanie optymalnej lub zbliżonej do optymalnej polityki gry. W porównaniu z algorytmem DQN, A2C oferuje bardziej stabilny proces treningowy, co w dużej mierze wynika z równoległego wykorzystywania wielu środowisk i regularyzacji entropii. Architektura aktor-krytyk pozwala na efektywniejsze zarządzanie informacjami zwrotnymi, dzięki czemu agent w mniejszym stopniu ulega przetrenowaniu i szybciej przystosowuje się do wymagań zadania.

6 Podsumowanie

Głównym celem przeprowadzonych badań była analiza efektywności wybranych algorytmów uczenia przez wzmacnianie w kontekście gry Pong, która stanowi często stosowane środowisko testowe dla metod sztucznej inteligencji. W ramach pracy przedstawiono kluczowe założenia teoretyczne dotyczące procesów decyzyjnych Markowa (MDP) oraz zaprezentowano znaczenie równań Bellmana w procesie optymalizacji polityki.

Praktyczna część prac koncentrowała się na porównaniu dwóch podejść: Deep Q-Learning (DQN) oraz Advantage Actor-Critic (A2C). Dokonano szczegółowego omówienia ich architektur i sposobu treningu, a także wpływu doboru hiperparametrów na stabilność i szybkość konwergencji. Wyniki eksperymentów wskazały, że DQN, choć w wielu przypadkach pozwala osiągać zadowalające rezultaty, bywa podatny na przetrenowanie. Z kolei A2C, dzięki architekturze aktor-krytyk, okazał się bardziej stabilny i skuteczny w osiąganiu długoterminowych celów.

W trakcie badań zidentyfikowano również znaczenie właściwego dostrajania hiperparametrów (takich jak współczynnik uczenia czy entropia), które okazało się niezbędnym warunkiem do uzyskania wysokich wyników. Odpowiednia konfiguracja parametrów bywa równie ważna jak sam wybór algorytmu i często decyduje o ostatecznej efektywności metody.

Uzyskane rezultaty potwierdzają, że algorytmy uczenia przez wzmacnianie stanowią obiecujące narzędzie do rozwiązywania problemów decyzyjnych, jednak ich wydajność silnie zależy od właściwego przygotowania środowiska, umiejętnego doboru architektury sieciowej oraz przemyślanej selekcji hiperparametrów. Wskazane konkluzje mogą być wykorzystane jako punkt wyjścia do dalszych badań nad bardziej zaawansowanymi podejściami, takimi jak Proximal Policy Optimization (PPO) czy Deep Deterministic Policy Gradient (DDPG), i ich potencjalnym zastosowaniem nie tylko w grach komputerowych, lecz także w robotyce czy analizie danych.

Dzięki połączeniu wiedzy teoretycznej z weryfikacją praktyczną przedstawione w niniejszej pracy wyniki umożliwiły uzyskanie spójnego obrazu efektywności algorytmów uczenia przez wzmacnianie w środowisku Pong i wskazały najważniejsze kierunki dalszych poszukiwań badawczych.

7 Bibliografia

References

[1] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[2] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1602.01783* (2016).

[3] Arthur L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* (1959).

[4] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[5] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 3rd Edition*. O'Reilly Media, 2022.

[6] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* (2015).

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, Second Edition*. The MIT Press, 2018.

[8] Maxim Lapan. *Deep Reinforcement Learning Hands-On. Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more - Second Edition*. Packt Publishing, 2020.

[9] Wikipedia contributors. *Reinforcement learning*. Accessed: 2025-01-21. 2025. URL: https://en.wikipedia.org/wiki/Reinforcement_learning.

[10] Stable-Baselines contributors. *A2C (Advantage Actor Critic) - Stable-Baselines Documentation*. Accessed: 2025-01-21. 2025. URL: https://stable-baselines.readthedocs.io/en/master/modules/a2c.html.

[11] Marc G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279. URL: https://github.com/mgbellemare/Arcade-Learning-Environment.

[12] Python Software Foundation. *Python Language Reference, version 3.12*. https://www.python.org. [Online; accessed 22-January-2025]. 2023.

[13] Gymnasium contributors. *Gymnasium: A Standard API for Reinforcement Learning Environments*. Accessed: 2025-01-21. 2025. URL: https://gymnasium.farama.org/.

[14] PyTorch contributors. *PyTorch: An Open Source Machine Learning Framework*. Accessed: 2025-01-21. 2025. URL: https://pytorch.org/.

[15] OpenCV contributors. *OpenCV: Open Source Computer Vision Library*. Accessed: 2025-01-21. 2025. URL: https://opencv.org/.

[16] Adrian Galik. *Repozytorium z implementacją DQN dla Pong*. https://github.com/Vexus1/engineer_project/tree/main/pong_ai/deep_q_learning_model. dostęp: styczeń 2025. 2025.

[17] Adrian Galik. *Repozytorium z implementacją A2c dla Pong*. https://github.com/Vexus1/engineer_project/tree/main/pong_ai/A2C_model. dostęp: styczeń 2025. 2025.