Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyczne Systemy Automatyki (ISA)**
Specjalność: **Inteligentne Systemy Przemysłu 4.0 (IPS)**

PRACA DYPLOMOWA MAGISTERSKA

Zastosowanie metod uczenia ze wzmocnieniem w środowisku gry Pac-Man

Paweł Pagacz
Maciej Siwicki
Grzegorz Smolak

Opiekun pracy
tytuł/stopień naukowy, imię i nazwisko opiekuna

Słowa kluczowe: Reinforcement Learning, Deep Q Learning, Advantage Actor Critic, Pac-Man, Machine Learning

WROCŁAW (rok złożenia pracy dyplomowej)

Streszczenie

Niniejsza praca skupia się na analizie wydajności dwóch popularnych algorytmów uczenia ze wzmocnieniem - Deep Q Network (DQN) i Advantage Actor-Critic (A2C) - w kontekście nauczania agenta gry w Pac-Mana. Praca podkreśla osiągnięcia metody DQN, która umożliwiła agentowi znalezienie efektywnej strategii w zdefiniowanym czasie nauki. Zgodnie z wynikami, agent zdołał zjeść około 30 pelletów, co odpowiada 20% całkowitego wyniku w grze. To świadczy o zdolności agenta do znajdowania optymalnych ścieżek i osiągania stałego przyrostu nagrody, nawet w ograniczonym okresie nauki. Jednakże, badania ujawniają również ograniczenia tych algorytmów. Głównym wyzwaniem okazała się duża przestrzeń stanów charakterystyczna dla gry Pac-Man, co znacząco wydłuża czas potrzebny do nauki. Dodatkowo, efektywność nauczania zależy w dużym stopniu od odpowiedniej implementacji zmiennych stanu oraz skutecznego projektowania funkcji nagrody. W pracy zasugerowano, że zmniejszenie przestrzeni stanów, na przykład poprzez ograniczenie pola widzenia agenta lub redukcję rozmiarów mapy, mogłoby przyspieszyć proces nauki. Mimo że agent DQN osiągnął znaczący przyrost nagrody, szybko przegrywał po osiągnięciu tego wyniku, co wskazuje na potrzebę dłuższego okresu treningu lub dalszych optymalizacji algorytmu. Podsumowując, praca ta wskazuje na potencjał algorytmów DQN i A2C w nauczaniu agentów gry w skomplikowane środowiska, jednocześnie zwracając uwagę na istotne wyzwania i ograniczenia, które wymagają dalszych badań i rozwoju w tej dziedzinie.

Abstract

This paper focuses on analyzing the performance of two popular reinforcement learning algorithms - Deep Q Network (DQN) and Advantage Actor-Critic (A2C) - in the context of training an agent to play Pac-Man. The work highlights the achievements of the DQN method, which enabled the agent to find an effective strategy within a defined learning period. According to the results, the agent managed to eat about 30 pellets, which corresponds to 20% of the total score in the game. This demonstrates the agent's ability to find optimal paths and achieve a steady increase in reward, even in a limited learning period. However, the research also reveals limitations of these algorithms. The main challenge turned out to be the large state space characteristic of the Pac-Man game, which significantly extends the learning time required. Additionally, the effectiveness of learning largely depends on the proper implementation of state variables and the effective design of the reward function. The paper suggests that reducing the state space, for example by limiting the agent's field of view or reducing the size of the map, could accelerate the learning process. Although the DQN agent achieved significant reward growth, it quickly lost after reaching this result, indicating the need for a longer training period or further algorithm optimizations. In summary, this work points to the potential of DQN and A2C algorithms in training agents for complex environments, while also highlighting significant challenges and limitations that require further research and development in this field.

Spis treści

1. Uczenie ze wzmocnieniem

1.1. Wstęp

Metoda uczenia maszynowego sztucznej inteligencji zwana uczeniem ze wzmocnieniem (DRL) opiera się na tworzeniu programów komputerowych, które mogą wykonywać zadania wymagające inteligencji. Unikalną cechą algorytmów DRL jest ich zdolność do uczenia się poprzez popełnianie błędów i wykorzystywanie informacji zwrotnych, które są próbkowane, sekwencyjne i oceniające jednocześnie, poprzez wykorzystanie silnego przybliżenia funkcji nieliniowych.

W informatyce sztuczna inteligencja (AI) to dziedzina zajmująca się opracowywaniem programów komputerowych wykazujących inteligencję. Chociaż sztuczna inteligencja obejmuje wszystkie programy komputerowe wykazujące inteligencję, nie wszystkie instancje sztucznej inteligencji są zdolne do uczenia się. Gałąź sztucznej inteligencji znana jako uczenie maszynowe koncentruje się na budowaniu programów komputerowych, które mogą wykorzystywać dane do uczenia się w celu rozwiązywania wielu problemów.

1.2. Podział uczenia maszynowego

Według źródła [12] uczenie maszynowe możemy podzielić na trzy główne obszary:

1. uczenie nadzorowane,
2. uczenie nienadzorowane,
3. uczenie ze wzmocnieniem.

Proces uczenia się na podstawie oznaczonych danych nazywany jest uczeniem nadzorowanym (SL). Człowiek wybiera, jakie dane chce zebrać i jak je oznaczyć. Klasycznym przykładem SL jest aplikacja do rozpoznawania cyfr pisanych odręcznie: człowiek zbiera obrazy z odręcznie zapisanymi cyframi, oznacza je etykietami i szkoli model, aby dokładnie rozpoznawał i klasyfikował cyfry na obrazach. Oczekuje się, że wyszkolony model będzie uogólniał i dokładnie identyfikował odręcznie zapisane cyfry na świeżych obrazach.

Zadanie uczenia się na podstawie nieoznakowanych danych jest znane jako uczenie się bez nadzoru (UL). Dane nie wymagają już etykietowania. Klasycznym przykładem uczenia nienadzorowanego jest aplikacja do klasyfikacji klientów - człowiek zbiera dane o klientach i szkoli model, aby klasyfikować klientów w klastry. Kompresując dane, klastry te ujawniają podstawowe relacje klientów.

Proces uczenia się poprzez popełnianie błędów nazywany jest uczeniem się przez wzmacnianie (RL). Żaden człowiek nie etykietuje danych w tego rodzaju zadaniach. Agent grający w Ponga jest klasycznym przykładem uczenia ze wzmocnieniem. Agent uczy się podejmując działania i obserwując ich efekty poprzez wielokrotne interakcje z otoczeniem. Od wyszkolonego agenta oczekuje się, że będzie działał w sposób umożliwiający mu skuteczną grę.

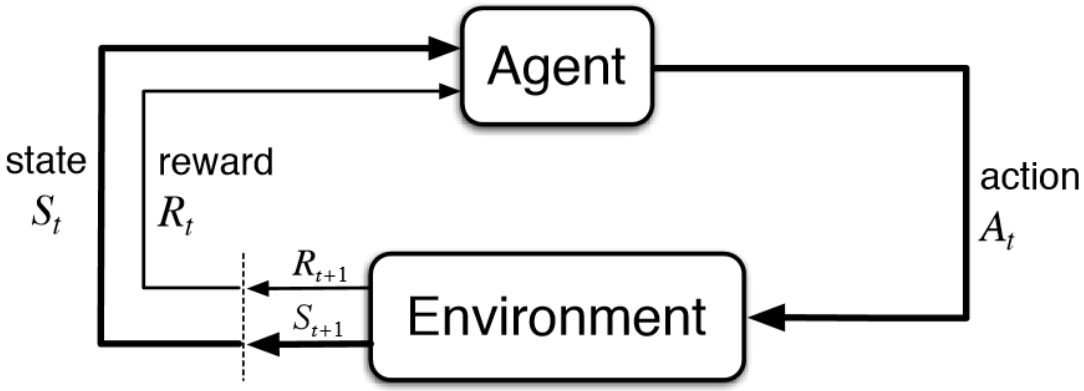
1.3. Uczenie głębokie

Głębokie uczenie (DL), potężna metoda uczenia maszynowego, wykorzystuje wielowarstwowe nieliniowe przybliżenie funkcji, zwykle sieci neuronowe. Ponieważ uczenie głębokie jest podzbiorem uczenia maszynowego, jego zadania są identyczne z omówionymi wcześniej. Uczenie głębokie ze wzmocnieniem to po prostu zastosowanie uczenia głębokiego do uczenia ze wzmocnieniem w rozwiązywaniu problemów.

1.4. Uczenie głębokie ze wzmocnieniem

Uczenie głębokie ze wzmocnieniem zajmuje się przede wszystkim trudnymi kwestiami związanymi z podejmowaniem decyzji sekwencyjnych w obliczu niepewności. Jest to jednak obszar zainteresowań wielu dyscyplin. Na przykład teoria sterowania bada strategie sterowania dla znanych złożonych systemów dynamicznych. w teorii sterowania dynamika systemów, które staramy się kontrolować, jest zazwyczaj z góry określona. Innym przykładem są badania operacyjne, które badają również podejmowanie decyzji w warunkach niepewności. Jednakże przestrzenie akcji są zazwyczaj znacznie większe niż w przypadku problemów uczenie głębokiego ze wzmocnieniem. Zgodnie ze źródłem [13] podstawowymi elementami uczenia ze wzmocnieniem jest:

1. Środowisko - to zadanie/symulacja, z którym agent wchodzi w interakcję.
2. Agent - poprzez interakcję ze środowiskiem uczy się, jak najkorzystniejszego z nim oddziaływać. Funkcję odpowiedzialną za wybranie odpowiedniej akcji przez agenta nazywamy polityką.
3. Nagroda - wartość zwracana przez środowisko po wykonaniu akcji wybranej przez agenta.
4. Akcja - działanie agenta na środowisko. Akcja może być wybrana ze zbioru lub określoną wartością albo wektorem wartości.
5. Stan - zmienna określająca stan środowiska.



Rysunek 1.1: Schemat uczenia ze wzmocnieniem

Agent wchodzi w interakcję ze środowiskiem poprzez podjęcie akcji A_t, ta natomiast prowadzi do zmiany stanu środowiska na S_t oraz otrzymaniem nagrody lub kary R_t przez agenta za podjętą akcję – proces ten następnie powtarza się w kolejnych iteracjach.

Zadanie, które agent próbuje wykonać, mogą być różne. Niektóre zadania posiadają naturalny koniec - wygranie gry. Innym przykładem może być nauka chodzenia do przodu w określonym środowisku. Osiągnięcie biegłości w zadaniu może wymagać od agenta wielu etapów i epizodów. Agenci uczą się metodą prób i błędów: próbują czegoś, obserwują, zdobywają wiedzę, próbują czegoś innego i tak dalej.

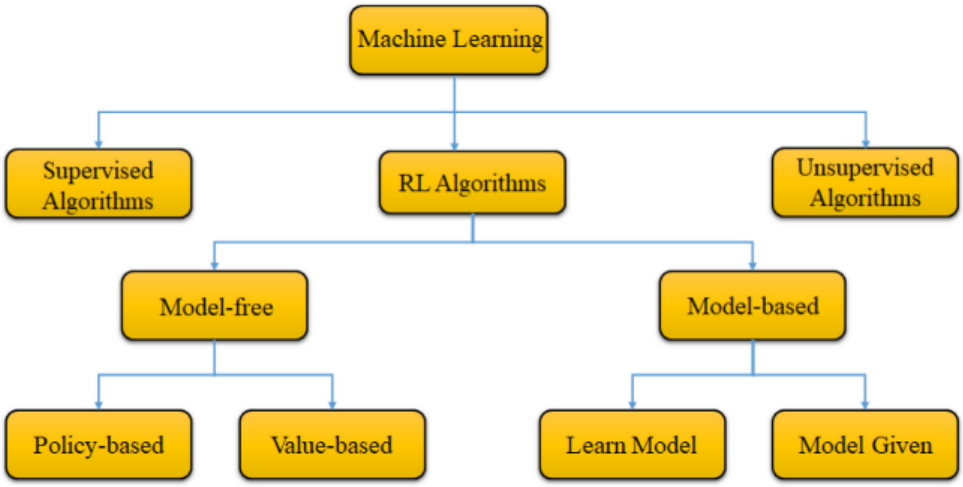
1.5. Unikalne cechy podejścia

Według źródła [12] uczenie ze wzmocnieniem polega na osiąganiu dobrych wyników w określonych zadaniach. w przeciwieństwie do uczenia nadzorowanego, gdzie celem jest generalizacja, uczenie ze wzmocnieniem przoduje w konkretnych zadaniach. Kluczem jest to, że jasno określone indywidualne zadania stanowią o sile uczenia ze wzmocnieniem.

Naturalnie, uczenie ze wzmocnieniem nie jest bezbłędny. Jednym z największych problemów jest to, że agenci wymagają milionów próbek, aby poznać skuteczne zasady dotyczące większości problemów. Jednym z najważniejszych aspektów uczenia ze wzmocnieniem, który można ulepszyć, jest wydajność próbki. Zrozumienie celu nagród i funkcji nagród to kolejny problem związany z uczeniem ze wzmocnieniem. Czy lepiej, aby nagroda była możliwie rzadka, co podkreśli wyjątkowość i ekscytację rozwiązaniami, czy też możliwie gęsta, co przyspiesza naukę?

1.6. Rodzaje podejścia w uczeniu ze wzmocnieniem

W rozdziale tym skupimy się na różnych rodzajach podejść w uczeniu ze wzmocnieniem (RL). Uczenie ze wzmocnieniem stanowi szeroką dziedzinę sztucznej inteligencji, więc nie dziwi też fakt, że wyodrębniły się różne podejścia do tego tematu. Zgodnie z [7], istnieją dwa główne nurty w podejściach do RL: model-free (bezmodelowe) i model-based (z modelowaniem). Każde z tych podejść ma swoje własne cechy, zalety i ograniczenia, co stanowi istotny obszar badań w rozwoju inteligentnych systemów zdolnych do samodzielnego uczenia się i podejmowania optymalnych decyzji. w niniejszym rozdziale przedstawimy główne aspekty obu podejść oraz omówimy, w jakich kontekstach i warunkach jedno podejście może być bardziej efektywne niż drugie.



Rysunek 1.2: Podział uczenia maszynowego i Uczenia ze wzmocnieniem według autorów: [14]

Model-based - Głównym założeniem uczenia ze wzmocnieniem z modelowaniem jest próba odtworzenia lub przybliżenia środowiska. w tym podejściu zadaniem agenta jest zbudowanie modelu który będzie starał się odzwierciedlić zachowanie środowiska na jego akcje. Takie podejście pozwala agentowi na akcje, które nie byłyby możliwe w przypadku podejścia model-free m. in. wyznaczanie trajektorii (Trajektoria, na podstawie artykułu [5], to zestaw akcji podejmowanych jedna po drugiej w kierunku osiągnięcia jakiegoś celu. Ponieważ podejście Model-based posiada swój model środowiska, to może podjąć próbę wyznaczenia sekwencji akcji, gdyż jest niejako w stanie przewidzieć zachowanie środowiska).

Uczenie ze wzmocnieniem bazujące na modelu można podzielić na dwie kategorie: Learn model i Model given. Jak sama nazwa mówi, learn model polega na nauczeniu się modelu przez agenta samemu na podstawie obserwacji środowiska, natomiast Model given bazuje na stworzonym ówcześnie modelu. Metoda ucząca się modelu często może w nieoczekiwany sposób zinterpretować środowisko, natomiast nadrabia elastycznością (dostosowaniem algorytmu to różnorakich problemów), której brakuje w przypadku odgórnie nadanego modelu.
Uczenie ze wzmocnieniem wykorzystujące podejście Model-based potrafi dać bardzo dobre rezultaty gdy uda się poprawnie zamodelować środowisko (przykładowo sukces AlphaGo, opisany szczegółowo w artykule [16]), jednak nie zawsze jest to proste zadanie, co wpływa na to, że to podejście jest złym wyborem do dynamicznych i złożonych środowisk.

Model-free - w przeciwieństwie do podejścia Model-based, podejście Model-free nie próbuje zamodelować środowiska na jakim jest stosowane, a dynamicznie podejmuje decyzje w zależności od stanu w jakim się znajduje. Oznacza to, że to podejście nie będzie planować z wyprzedzeniem gdyż nie jest wstanie przewidzieć zachowania środowiska na jego akcję, a jest jedynie obserwatorem, który podejmuje decyzję zaraz po otrzymaniu odpowiedzi ze środowiska.

w uczeniu ze wzmocnieniem z modelowaniem wyodrębniły się dwa rodzaje algorytmów: oparte na strategii (*ang. policy-based*) i oparte na wartości (*ang. value-based*). Podejście

policy-based polega na wyznaczeniu strategii dla danego stanu, która polega na stochastycznym wybieraniu akcji. Dzięki takiemu podejściu algorytm wyodrębni działania, które będą wykonywane z danym prawdopodobieństwem w określonym stanie. w przypadku podejścia value-based algorytym będzie podejmował deterministycznie zawsze najlepszą akcję w oparciu o funkcję wartości. Aby jednak agent mógł uczyć się potrzebuje eksplorować różne akcje dla danych stanów. w przypadku policy-based eksploracja jest oczywista i odbywa się poprzez stochastyczne wykonywanie akcji. w przypadku value-based agent posiada parametr chciwości, który oznacza szanse wykonania losowej akcji, a nie akcji o najlepszej wartości.

Podejście model-free, pomimo braku możliwości przeanalizowania i zastosowania sekwencji akcji będzie często lepszym podejściem od uczenia z modelowaniem, gdyż poradzi ono sobie lepiej w dynamicznie zmiennych czy złożonych środowiskach, lub takich, dla których trudne jest sporządzenie modelu.

Obydwa podejścia mają swoje plusy i minusy. Nie można wyróżnić podejścia lepszego lub gorszego, ponieważ każde z nich sprawdzi się lepiej lub gorzej w zależności od środowiska. w sytuacji gdy środowisko jest łatwe do zamodelowania, nie jest dynamiczne i/lub agent będzie odnosił duże korzyści z rozumienia i planowania wprzód - lepiej sprawdzi się podejście z modelowaniem. w przypadku przeciwstawnym, gdy sporządzenie modelu jest kłopotliwe - lepszym wyjściem będzie uczenie bez modelu. w praktyce często spotyka się implementacje łączące te dwa podejścia, by zyskać z nich jak najwięcej plusów, a zniwelować minusów.

2. Podstawy matematyczne

Większość rzeczywistych problemów decyzyjnych można wyrazić jako środowiska RL. Powszechnym sposobem reprezentowania procesów decyzyjnych w RL jest modelowanie problemu za pomocą ram matematycznych znanych jako procesy decyzyjne Markowa (MDP).

W książce [12], autor opisał środowisko jako zbiór zmiennych związanych z problemem. Kombinacja wszystkich możliwych wartości, jakie może przyjąć ten zestaw zmiennych, nazywana jest ***przestrzenią stanów***. Agenci mogą, ale nie muszą, mieć dostęp do rzeczywistego stanu środowiska; jednakże w ten czy inny sposób mogą obserwować zmienne z otoczenia. w każdym stanie środowisko udostępnia agentowi zestaw akcji. Zbiór wszystkich akcji we wszystkich stanach nazywany jest ***przestrzenią akcji***. Agent poprzez te działania próbuje wpłynąć na otoczenie. Środowisko może zmieniać stany w odpowiedzi na działanie agenta. Funkcja odpowiedzialna za to przejście nazywa się ***funkcją przejścia***. Po przejściu środowisko emituje nową obserwację. w odpowiedzi otoczenie może również dostarczyć sygnału nagrody. Funkcja odpowiedzialna za to odwzorowanie nazywa się ***funkcją nagrody***. Zbiór funkcji przejścia i nagrody nazywany jest ***modelem środowiska***.

Stany muszą zawierać wszystkie zmienne niezbędne do uniezależnienia ich od wszystkich innych stanów. Prawdopodobieństwo następnego stanu (S_{t+1}), biorąc pod uwagę bieżący stan (S_t) i akcję (A_t), jest niezależne od historii. Ta pozbawiona pamięci właściwość MDP jest znana jako właściwość Markowa (2.1):

$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, \ldots) \tag{2.1}$$

2.1. Funkcja przejścia

Autor [12] opisuje również sposób, w jaki środowisko zmienia się w odpowiedzi na działania, nazywany jest prawdopodobieństwem przejścia stanu lub prościej funkcją przejścia i jest oznaczany przez $T(s, a, s')$. Funkcja przejścia T odwzorowuje krotkę przejścia s, a, s' na prawdopodobieństwo, to znaczy przekształca stan s, akcję a i następny stan s' na prawdopodobieństwo przejścia ze stanu s do stanu s' podczas podejmowania akcji a (2.2).

$$p(s' \mid S, a) = P(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) \tag{2.2}$$

Funkcja przejścia T opisuje rozkład prawdopodobieństwa $p(\cdot \mid s, a)$ określający ewolucję systemu w cyklu interakcji począwszy od wybrania działania w stanie s. Całkując po kolejnych stanach s', jak każdy rozkład prawdopodobieństwa, suma tych prawdopodobieństw musi być równa jedności (2.3).

$$\sum_{s' \in S} p(s' \mid s, a) = 1, \forall s \in S, \forall a \ \in A(s) \tag{2.3}$$

2.2. Funkcja nagrody

Funkcja nagrody R odwzorowuje krotkę przejściową s, a, s' na skalar. Funkcja nagrody daje numeryczny sygnał poprawności przejść. Kiedy sygnał jest pozytywny, możemy nagrodzić agenta za wykonany ruch. W przeciwnym przypadku (ujemny sygnał), agent zostanie ukarany (negatywna wartość nagrody). Funkcja nagrody może być zdefiniowana następująco (2.4):

$$r(s, a, s') = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] \tag{2.4}$$

2.3. Horyzont planowania

Autor [12] stosuje również podział na dwa rodzaje zadań - epizodyczne (zadanie,w którym występuje skończona liczba etapów czasowych) oraz ciągłe (w których nie ma stanów końcowych, więc agent musi zostać zatrzymany ręcznie). W przypadku zadań, w których istnieje duże prawdopodobieństwo, że agent utknie w pętli i nigdy się nie zakończy (zadania ciągłe), powszechną praktyką jest dodawanie sztucznego stanu końcowego na podstawie kroku czasowego. Aby zrealizować tą metodę należy dyskontować (obniżać) wartości nagród w czasie. Zwykle używa się dodatniej wartości rzeczywistej mniejszej niż jeden, aby wykładniczo dyskontować wartość przyszłych nagród. Im dalej w przyszłość otrzymamy nagrodę, tym mniej będzie ona wartościowa w teraźniejszości. Liczba ta nazywana jest ***współczynnikiem dyskontowym*** (***gamma***). Jeśli zastosujemy współczynnik dyskontowy to suma wszystkich nagród prezentuje się następująco (2.5):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \ldots + \gamma^{T-1} R_T \tag{2.5}$$

Równanie to można uprościć do równania (2.6):

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1} \tag{2.6}$$

Często jednak nie można znaleźć wskazówek dotyczących właściwej wartości gamma, którą należy zastosować w danym środowisku. Dzieje się też tak dlatego, że gamma jest również używana jako hiperparametr zmniejszający wariancję i dlatego pozostawia się ją agentowi do dostrojenia.

2.4. Rodzaje metod MDP

W skład metody MDP wchodzą przestrzenie stanów S, przestrzenie akcji A, funkcje przejścia T, sygnały nagrody R, zbiór rozkładów stanów początkowych S_θ , współczynnik dyskontowy γ i horyzont H. Autor [12] wspomina również o rozszerzeniach metody MDP i są to między innymi:

1. **Częściowo obserwowalny proces decyzyjny Markowa (POMDP)** – gdy agent nie może w pełni obserwować stanu środowiska - $POMDP(S, A, T, R, S_\theta, \gamma,$

H, O, ϵ). Jak widać, w definicji pojawiają się dodatkowe dwa argumenty O - przestrzeń obserwacji oraz ϵ - prawdopodobieństwo emisji, określające prawdopodobieństwo pokazania obserwacji O dla danego stanu s_t,

2. **Faktoryzowany proces decyzyjny Markowa (FMDP)** – umożliwia bardziej zwięzłe przedstawienie funkcji przejścia i nagrody, dzięki czemu możemy reprezentować duże MDP,
3. **Ciągły Proces decyzyjny Markowa** – gdy czas, akcja, stan lub dowolna ich kombinacja są ciągłe,
4. **Relacyjny proces decyzyjny Markowa (RMDP)** – umożliwia połączenie wiedzy probabilistycznej i relacyjnej,
5. **Proces decyzyjny semi-Markowa (SMDP)** – umożliwia włączenie abstrakcyjnych działań, których wykonanie może wymagać wielu etapów czasowych,
6. **Wieloagentowy proces decyzyjny Markowa (MMDP)** – umożliwia włączenie wielu agentów do tego samego środowiska,
7. **Zdecentralizowany proces decyzyjny Markowa (Dec-MDP)** – umożliwia współpracę wielu agentów i maksymalizację wspólnej nagrody.

3. Powiązane prace

Gry Atari (takie jak Pac-Man, Pong, Breakout) są znanymi przykładami środowisk, w których można zaimplementować uczenie ze wzmocnieniem. Ciekawymi i rozbudowanymi pracami są [10], [11], gdzie autorzy zaimplementowali metodę głębokiego uczenia ze wzmocnieniem opartej na sieciach neuronowych (Deep Q-Network) na wielu grach opracowanych przez Atari. Metodę opartą na DQN w grze Pac-Man zastosowali również autorzy [1].
W przedstawionych powyżej artykułach zastosowane zostało podejście *model-free*. Podejście *model-based* zostało zastosowane w artykule [8] gdzie autorzy skupili się na minimalizacji interakcji agenta ze środowiskiem. Podobne podejście zostało zastosowane w artykule [6], gdzie autorzy zastosowali metodę Monte Carlo Tree-search w celu planowania trajektorii agenta.

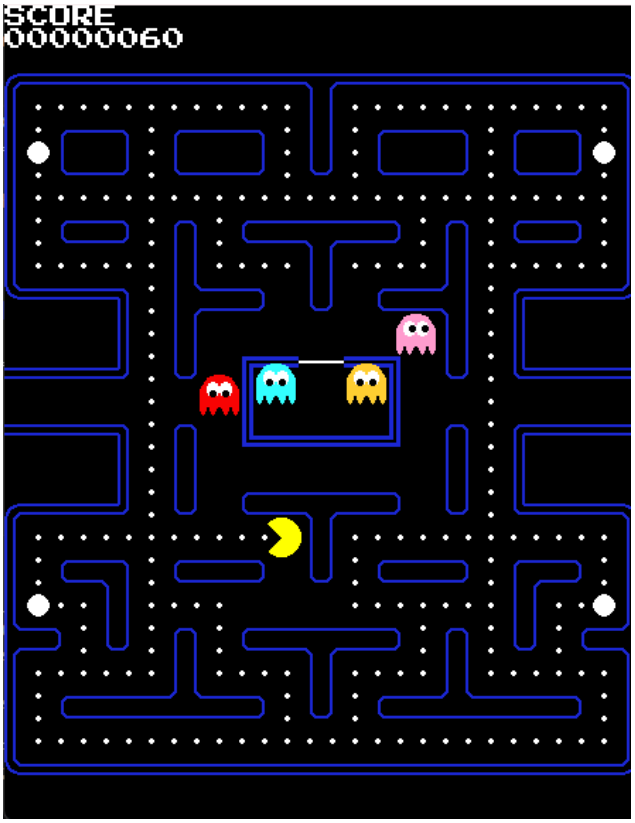
Uczenie ze wzmocnieniem zostało również zastosowane do nauczenia agenta grania w innych grach niż atari. Przykładem takiego zastosowania może być praca [9], gdzie autorzy skupili się na zastosowaniu uczenia ze wzmocnieniem w grach typu FPS (first person shooter) lub praca [2], gdzie autorzy skupili się na zastosowaniach w grach typu RTS(real-time strategy). Bardziej złożonych zastosowań wykorzystania uczenia ze wzmocnieniem (RL) do stworzenia agenta zdolnego grać w gry można znaleźć w artykule [17], gdzie autorzy opracowali algorytm AlphaStar, który po okresie uczenia był zdolny pokonać 99,8% graczy na świecie osiągając poziom zawodowych graczy. Podobna sytuacja została opisana w artykule [4] gdzie autorzy byli w stanie stworzyć nauczoną za pomocą uczenia ze wzmocnieniem sztuczną inteligencję, która była w stanie pokonać mistrzów świata w grze Dota2.
Powstało wiele artykułów, w których podsumowywane są osiągnięcia w dziedzinie uczenia ze wzmocnieniem w grach komputretowych. Dobrym przykładem takiego artykułu jest [15], gdzie autorzy skupili się na analizie zarównoodkryć w grach 2D i 3D jak i systemach jedno- i wielo-agentowych.

W ninejszej pracy autorzy skupili się na zastosowaniu algorytmów A2C i DQN do nauczenia agenta gry w pac-mana. Algorytmy te zostały dobrze opisane w artykułach przeglądowych takich jak [3], [18]

4. Badania

Do zaimplementowanego ręcznie środowiska Pac-Man (Rysunek 4.1) zaimplementowano dwie metody:

1. Metoda DQN, (4.1. Metoda DQN)) składającą się z trzech warstw gęstych: dwóch warstw ukrytych z funkcją aktywacji ReLU oraz warstwy wyjściowej z funkcją liniową,
2. Metoda A2C, dostępna w bibliotece *stable_baselines* (rozdział 4.2. Metoda A2C).



Rysunek 4.1: Schemat uczenia ze wzmocnieniem

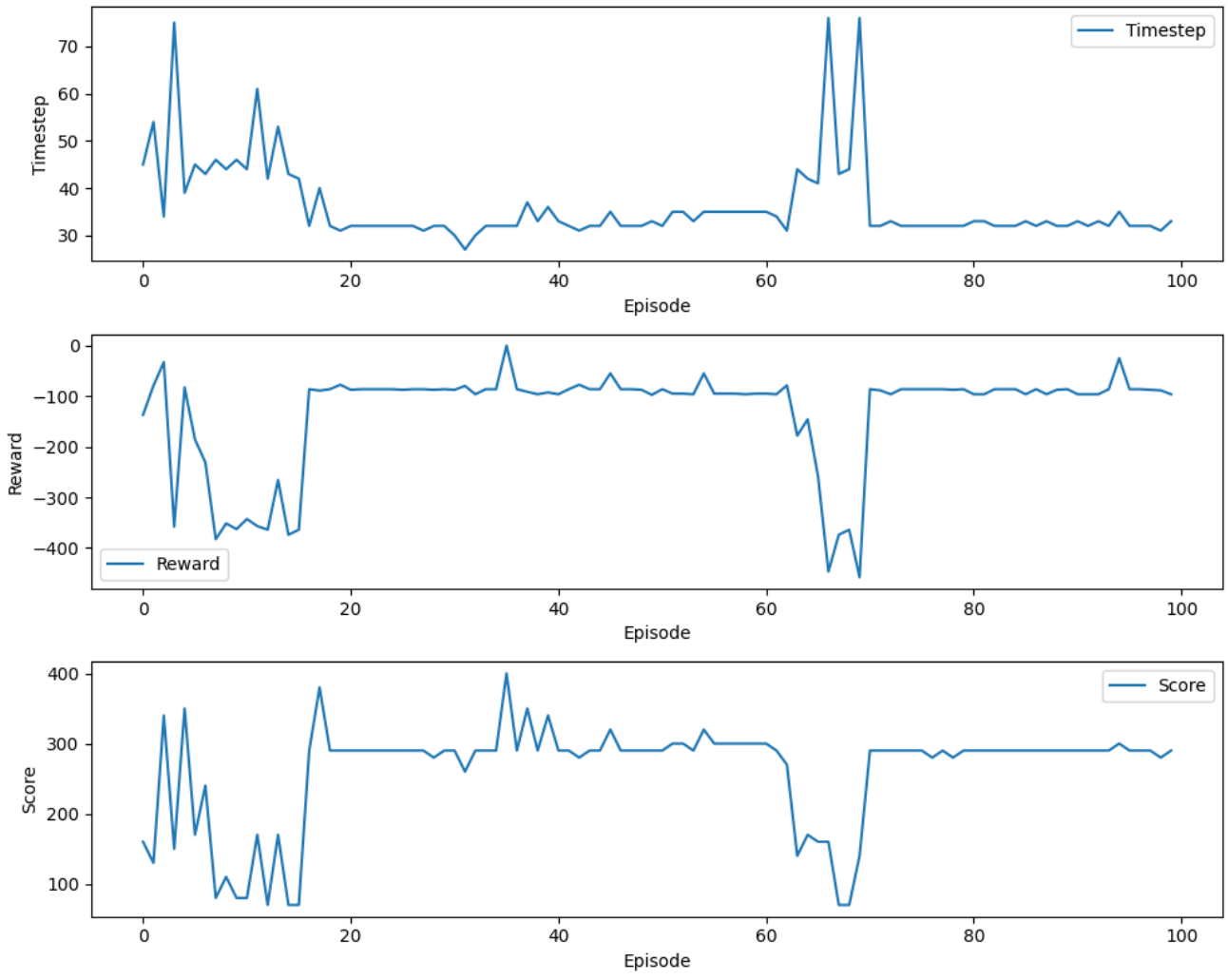
Zakładając, że nasze zadanie uczenia się przez wzmacnianie przebiega według skończonego procesu decyzyjnego Markowa (MDP) (2.1), zdefiniować można:

- Przestrzeń stanów S: pozycje (X, Y), kierunek ruchu wszystkich duchów (blinky, pinky, inky, clyde) oraz Pac-Mana. Dodatkowo, aby obserwować skuteczność zjadania pożywienia przez Pac-Mana, do przestrzeni stanów dodana jest również informacja o ilości zjedzonego pożywienia,

- Przestrzeń akcji A: Wszystkie możliwe ruchy Pac-Mana (ruch w prawo, lewo, w górę, w dół),
- Funkcja nagrody: Na podstawie testów środowiska dla zdefiniowanej przestrzeni stanów, ustalono następującą nagrodę:
  - Nagroda za jedzenie pożywienia +1,2;
  - Nagroda za zwycięstwo +150;
  - Kara za przegraną -20;
  - Kara za wykonywanie niedozwolonych poprzez architekturę mapy akcji -10;

4.1. Metoda DQN

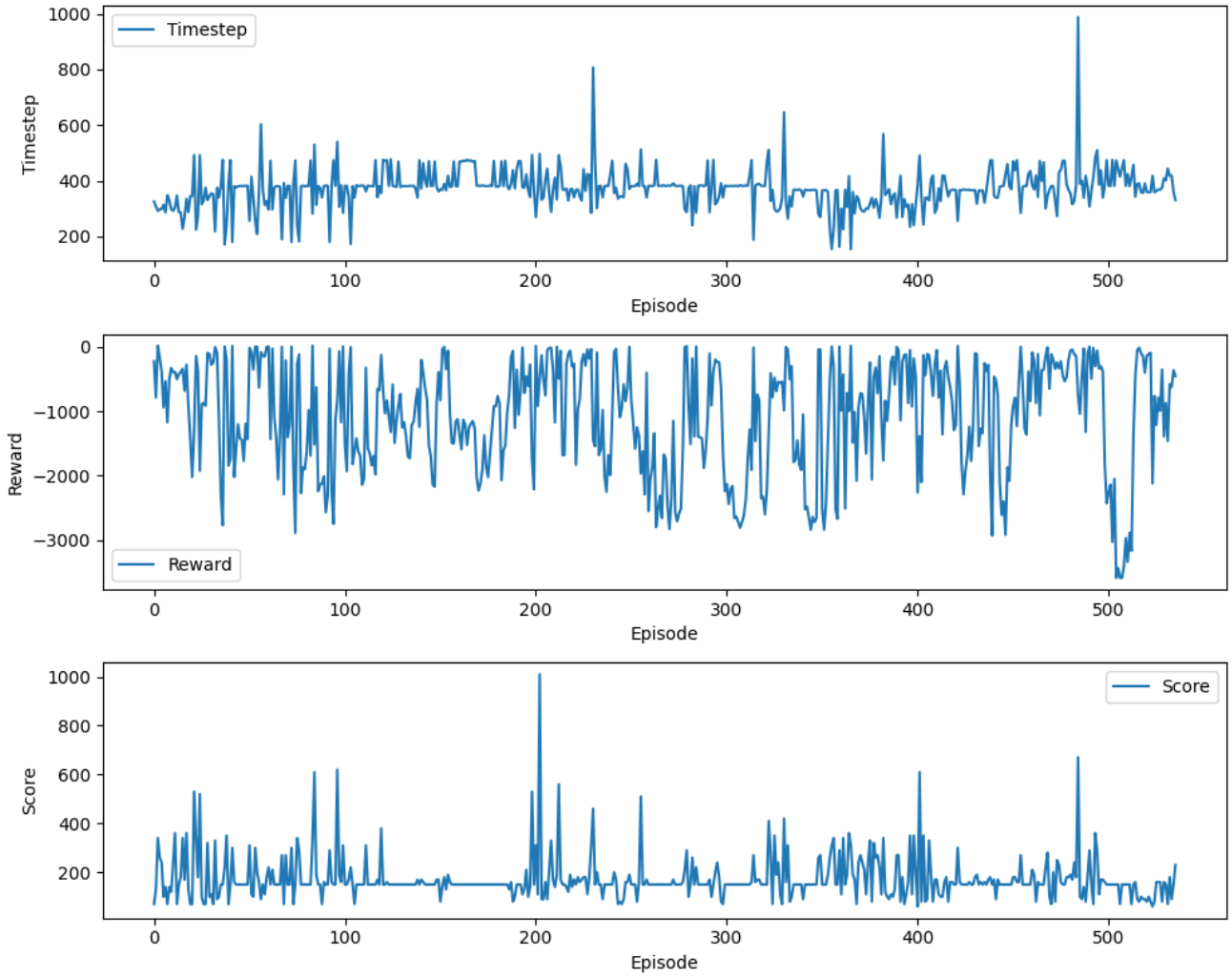
W celu przebadania metody DQN, podczas uczenia modyfikowano wartość epsilon decay, zwiększając tym samym czas uczenia. Co każdy epizod (czyli co każdy restart środowiska) zbierano informację na temat ilości kroków czasowych, wartości nagrody oraz ilość punktów zebranych podczas gry. Przykładowy przebieg wykresów zaprezentowano poniżej (Rysunek 4.2):



Rysunek 4.2: Zależność kroków czasowych, nagrody oraz wyniku gry od epizodu

4.2. Metoda A2C

Metoda A2C przebadana została w podobny sposób co metoda DQN (podczas procesu uczenia), jednakże ze względu na architekturę gotowej funkcji dostępnej w bibliotece *stable_baselines*, parametrem zmiennym była całkowita ilość kroków czasowych. Modyfikacja tej wartości zmieniała ilość epizodów gry, a tym samym długość nauki. Przykładowy przebieg wykresów dla metody A2C przedstawiono poniżej (Rysunek 4.3):



Rysunek 4.3: Zależność kroków czasowych, nagrody oraz wyniku gry od epizodu

5. Wnioski

1. Z badań wynika, że algorytmy takie jak Deep Q Network (DQN) czy Advantage Actor-Critic (A2C), mają ograniczony potencjał do efektywnego nauczania agenta gry w skomplikowanym środowisku. Wynika to z wielkości przestrzeni stanów do przeanalizowania co zwiększa czas nauki,

2. Przy tworzeniu modelu uczenia ze wzmocnieniem kluczowymi aspektami okazały się implementacja zmiennych stanu oraz zaprojektowanie odpowiedniej funkcji nagrody. Dobór odpowiednich parametrów pozwoliłby na efektywniejszą naukę agenta,

3. W celu przyspieszenia nauki warto byłoby zastanowić się nad zmniejszeniem przestrzeni stanów poprzez ograniczenie pola widzenia agenta lub zmniejszenie mapy i zmiennych.

4. Na wykresach przedstawionych na rysunku 4.2 (metoda DQN) można zauważyć, że agentowi udało się znaleźć optymalną (dla tak zdefiniowanego czasu uczenia) ścieżkę, zapewniającą mu stały przyrost nagrody. Analizując wynik, można zauważyć, że udało mu się zjeść około 30 pelletów, co równoważne jest z 20% gry. Niestety chwilę później ginął, co świadczy o tym, że nauka trwała zbyt krótko.

Bibliografia

[1] J. An, J. Feliu, and A. Gnanasekaran. Reinforcement learning in pacman. 2017.

[2] P.-A. Andersen, M. Goodwin, and O.-C. Granmo. Deep rts: a game environment for deep reinforcement learning in real-time strategy games. In *2018 IEEE conference on computational intelligence and games (CIG)*, pages 1–8. IEEE, 2018.

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[5] Y. Efroni, N. Merlis, and S. Mannor. Reinforcement learning with trajectory feedback. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 7288–7295, 2021.

[6] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[7] Q. Huang. Model-based or model-free, a review of approaches in reinforcement learning. In *2020 International Conference on Computing and Data Science (CDS)*, pages 219–221, 2020.

[8] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[9] G. Lample and D. S. Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, and Charle. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[12] M. Morales. *Deep Reinforcement Learning*. Manning Publications Co., 20 Baldwin Road, Shelter Island, NY 11964, (2020).

[13] A. G. B. Richard S. Sutton. *Reinforcement Learning*. The MIT Press, Cambridge, Massachusetts, 2 edition, (2018).

[14] A. Seyyedabbasi, R. Aliyev, F. Kiani, M. U. Gulle, H. Basyildiz, and M. A. Shah. Hybrid algorithms based on combining reinforcement learning and metaheuristic methods to solve global optimization problems. *Knowledge-Based Systems*, 223, 2021.

[15] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint arXiv:1912.10944*, 2019.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan. 2016.

[17] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[18] H.-n. Wang, N. Liu, Y.-y. Zhang, D.-w. Feng, F. Huang, D.-s. Li, and Y.-m. Zhang. Deep reinforcement learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 21(12):1726–1744, 2020.

Spis rysunków