Implementacja i Uczenie Gry Snake za Pomocą Sieci Neuronowej DQN w Pythonie

Bartek Grabowski

Numer Indeksu: 325467

27 maja 2025

Streszczenie

Niniejszy dokument stanowi sprawozdanie z realizacji projektu polegającego na implementacji klasycznej gry Snake oraz opracowaniu agenta opartego na sztucznej inteligencji, zdolnego do autonomicznej nauki optymalnej strategii gry. Kluczowym elementem projektu było zastosowanie algorytmu Deep Q-Network (DQN). Środowisko gry zostało zaimplementowane w języku Python z wykorzystaniem biblioteki Pygame, a agent DQN przy pomocy frameworka TensorFlow/Keras. W sprawozdaniu przedstawiono architekturę systemu, implementację kluczowych komponentów, definicję stanu, akcji i systemu nagród. Raport zawiera analizę przebiegu procesu treningowego oraz dyskusję uzyskanych wyników. Kod źródłowy projektu dostępny jest na platformie GitHub: https://github.com/GrabowskiB/PythonSnakeProject.git.

Spis treści

Spis rysunków

1 Wstęp

1.1 Cel Projektu

Głównym celem projektu było zaprojektowanie, implementacja oraz ewaluacja systemu opartego na uczeniu przez wzmacnianie, umożliwiającego agentowi autonomiczną naukę gry w Snake. Wykorzystano algorytm Deep Q-Network (DQN) do podejmowania decyzji i optymalizacji strategii.

1.2 Motywacja i Kontekst

Gry komputerowe, takie jak Snake, są popularnymi platformami testowymi dla algorytmów AI. Algorytm DQN zrewolucjonizował podejście do rozwiązywania złożonych problemów decyzyjnych. Projekt ten stanowi próbę praktycznego zastosowania mechanizmów DQN.

1.3 Zakres Projektu

Realizacja projektu obejmowała: opracowanie środowiska gry Snake (Pygame), implementację agenta DQN (TensorFlow/Keras) z mechanizmami Experience Replay i Target Network, definicję przestrzeni stanów, akcji i funkcji nagrody, integrację agenta ze środowiskiem, przeprowadzenie treningu oraz analizę wyników.

2 Podstawy Teoretyczne Uczenia przez Wzmacnianie i DQN

Uczenie przez wzmacnianie (RL) to dziedzina uczenia maszynowego, gdzie agent uczy się optymalnych decyzji poprzez interakcję ze środowiskiem, dążąc do maksymalizacji sumarycznej nagrody. Formalnym modelem są Procesy Decyzyjne Markowa (MDP).

Algorytm Q-Learning uczy optymalnej funkcji wartości akcji $Q^*(s, a)$ za pomocą równania Bellmana. W przypadku złożonych problemów, Deep Q-Networks (DQN) aproksymują tę funkcję za pomocą głębokich sieci neuronowych. Kluczowe techniki w DQN to:

- **Sieci Neuronowe jako Aproksymatory Funkcji Q:** Umożliwiają generalizację na niewidziane wcześniej stany.
- **Pamięć Powtórek (Experience Replay):** Przechowywanie i losowe próbkowanie doświadczeń agenta $(s, a, r, s', \text{done})$ stabilizuje uczenie poprzez przełamywanie korelacji między próbkami.

- **Sieci Docelowe (Target Networks):** Oddzielna sieć, której wagi są okresowo kopiowane z głównej sieci, używana do obliczania wartości docelowych, co dodatkowo stabilizuje proces uczenia.

3 Implementacja Szczegółowa

3.1 Środowisko Gry Snake (‘snake_game_ml.py‘)

Środowisko gry zaimplementowano w Pygame. Plansza to siatka 800×600 pikseli, bloki 40×40. Wąż zbiera jedzenie, wydłuża się, gra kończy się po kolizji. Wizualizacja obejmuje węża, jedzenie i wynik.

3.2 Agent DQN (‘dqn_agent.py‘)

3.2.1 Architektura Sieci Neuronowej

Model sieci (Keras Sequential API): warstwa wejściowa (11 cech stanu), warstwa ukryta (128 neuronów, ReLU), warstwa wyjściowa (3 neurony, liniowa, wartości Q). Kompilacja: strata MSE, optymalizator Adam.

```
model = models.Sequential([
    layers.Input(shape=(self.state_size,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(self.action_size, activation='linear')
])
model.compile(loss='mse', optimizer=optimizers.Adam(learning_rate=self.learning_rate))

```

Listing 1: Definicja modelu sieci DQN

3.2.2 Hiperparametry Treningowe

Tabela 1: Hiperparametry agenta DQN.

Parametr	Wartość
Współczynnik dyskontowania (γ)	0.95
Początkowa ϵ	1.0
Minimalna ϵ	0.01
Zanik ϵ	0.995
Współczynnik uczenia	0.001
Batch size	64
Pojemność pamięci	20000
Aktualizacja sieci docelowej	co 5 epizodów

3.3 Integracja Agenta ze Środowiskiem ('snake_game_ml.py')

3.3.1 Definicja Stanu Środowiska

Stan gry to 11-elementowy wektor binarny: 3 bity dla kolizji (prosto, lewo, prawo), 4 bity dla kierunku ruchu (one-hot), 4 bity dla względnej pozycji jedzenia.

3.3.2 Definicja Przestrzeni Akcji

3 akcje: idź prosto, skręć w lewo, skręć w prawo (względem obecnego kierunku).

3.3.3 System Nagród

- **+10 punktów:** za zjedzenie jedzenia.
- **-100 punktów (kara):** za kolizję (koniec gry).
- **0 punktów:** za każdy inny krok.

3.3.4 Pętla Treningowa i Zapis Postępów

Trening w epizodach. W każdym kroku: obserwacja stanu, wybór akcji (ϵ-greedy), wykonanie akcji, nagroda, nowy stan, zapis do pamięci, krok uczenia ('replay'). ϵ maleje, sieć docelowa aktualizowana. Logowanie metryk do CSV, okresowy zapis modelu.

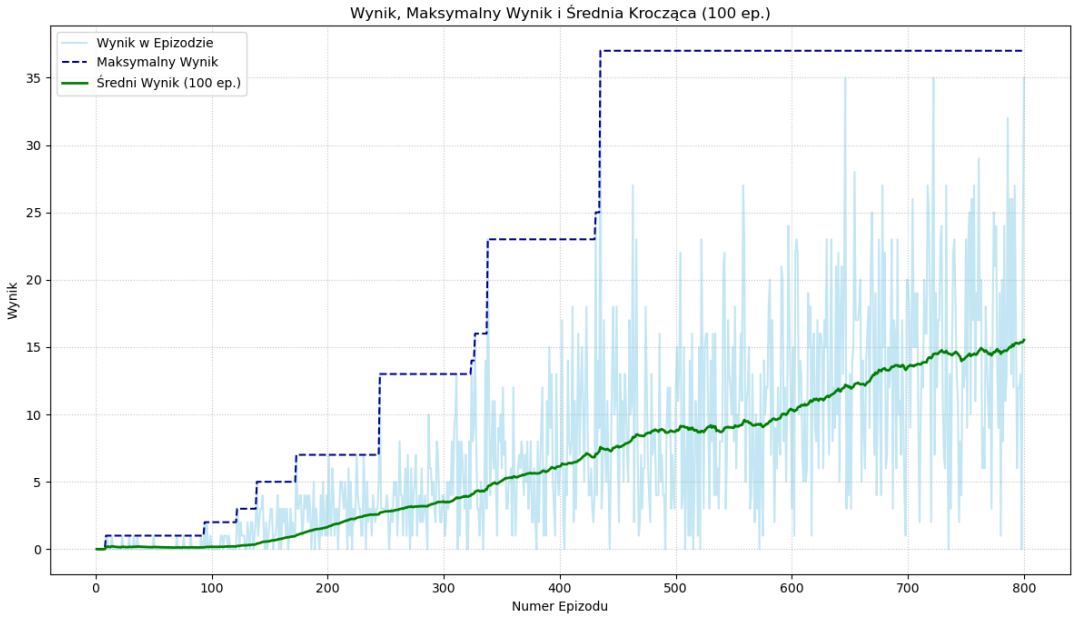
4 Eksperymenty i Wyniki

4.1 Konfiguracja Środowiska Treningowego

Trening agenta przeprowadzono dla 800 epizodów. Ograniczenie liczby epizodów wynikało z wydłużającego się czasu ich trwania w miarę postępów uczenia. Analiza opiera się na danych z pliku `snake_dqn_training_log2_01.csv`. Wykorzystano TensorFlow 2.10.1. *Informacja o konfiguracji sprzętowej: Trening przeprowadzono z wykorzystaniem GPU NVIDIA GeForce RTX 4050 Laptop GPU.*

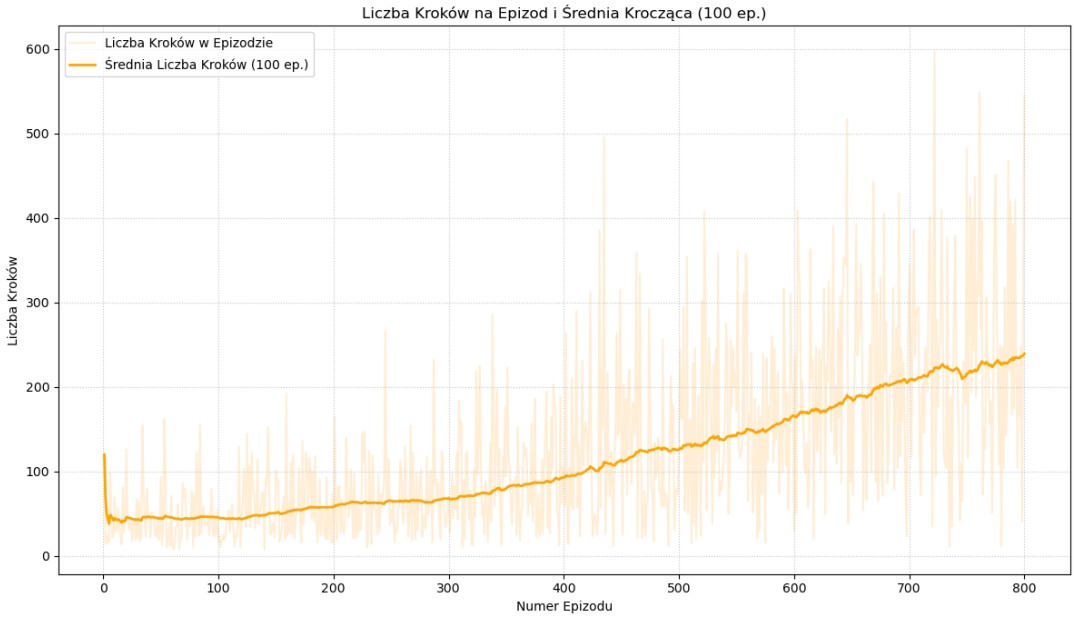
4.2 Analiza Metryk Treningowych

Poniżej przedstawiono wykresy ilustrujące kluczowe metryki zebrane podczas procesu treningowego agenta DQN.



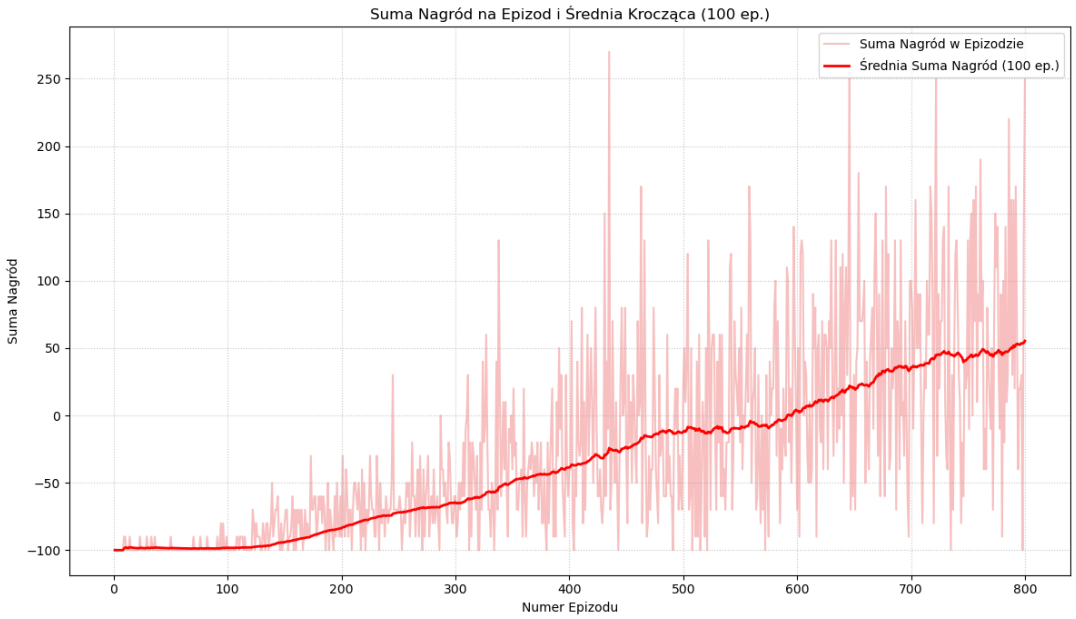
Rysunek 1: Wynik w epizodzie, maksymalny osiągnięty wynik oraz krocząca średnia wyniku z 100 ostatnich epizodów.

Rysunek 1 prezentuje ewolucję wyniku. Linia jasnoniebieska (*Wynik w Epizodzie*) ukazuje dużą wariancję. Linia ciemnoniebieska, przerywana (*Maksymalny Wynik*) systematycznie rośnie. Zielona linia (*Średni Wynik (100 ep.)*) pokazuje wyraźny trend wzrostowy, potwierdzając naukę efektywnej strategii. Po około 400 epizodach średni wynik zaczyna stabilnie przekraczać 5 punktów, a pod koniec analizowanego okresu zbliża się do 15 punktów.



Rysunek 2: Liczba kroków wykonanych w epizodzie oraz krocząca średnia liczby kroków z 100 ostatnich epizodów.

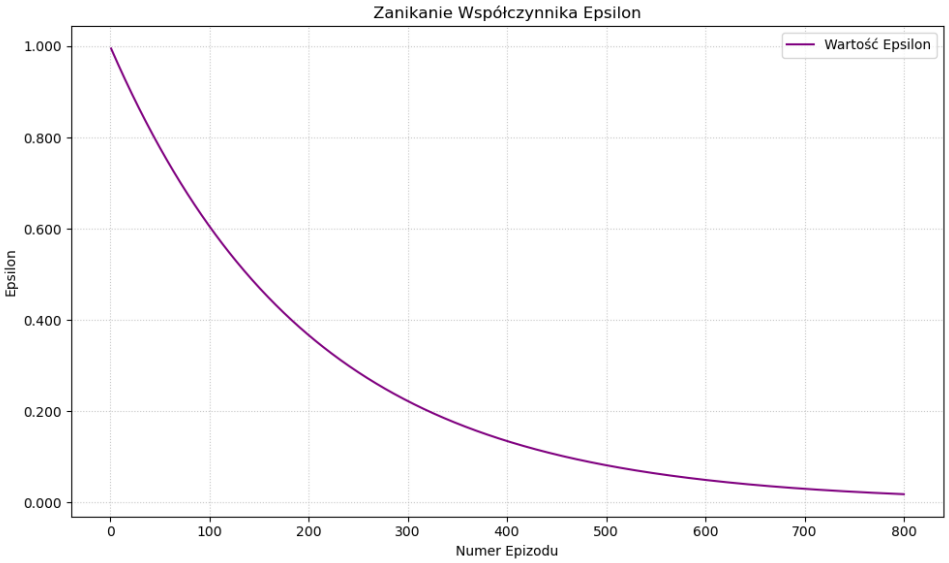
Rysunek 2 ilustruje liczbę kroków agenta w epizodzie. Obserwujemy wzrost średniej liczby kroków (linia pomarańczowa), co koreluje z nauką strategii pozwalającej na dłuższe przetrwanie. Wzrost ten jest szczególnie widoczny po około 300-400 epizodach.



Rysunek 3: Suma nagród uzyskanych w epizodzie oraz krocząca średnia sumy nagród z 100 ostatnich epizodów.

Dynamika sumy nagród (Rysunek 3) jest powiązana z wynikiem. Wzrost średniej sumy

nagród (linia czerwona) potwierdza, że agent efektywniej zbiera nagrody i unika kar. Początkowe epizody charakteryzują się dominacją kar.



Rysunek 4: Zmiana wartości współczynnika eksploracji epsilon w trakcie treningu.

Rysunek 4 przedstawia zanikanie ϵ. Z $\epsilon = 1.0$ (pełna eksploracja), wartość maleje, prowadząc do przejścia agenta do fazy eksploatacji. Po 800 epizodach ϵ osiąga około 0.018.

4.3 Analiza Wyuczonych Strategii Behawioralnych

Obserwacja agenta w trybie ewaluacji ($\epsilon \approx 0.01$) wykazała wykształcenie się spójnych strategii:

- **Efektywne zbieranie pożywienia:** Agent sprawnie kieruje się w stronę jedzenia.
- **Unikanie kolizji:** Agent poprawnie identyfikuje zagrożenia i wykonuje manewry wymijające.
- **Zachowanie w pobliżu krawędzi:** Agent nauczył się poruszać wzdłuż krawędzi planszy.
- **Decyzje w trudnych sytuacjach:** Analiza wartości Q w specyficznych scenariuszach (np. jedzenie w rogu, bliskość przeszkód) potwierdziła, że agent często podejmuje logiczne decyzje, maksymalizując oczekiwaną przyszłą nagrodę. Na przykład, w sytuacji bezpośredniego zagrożenia kolizją ze ścianą przy jednoczesnej możliwości skrętu w kierunku jedzenia, agent konsekwentnie wybierał bezpieczniejszy manewr prowadzący do celu, co odzwierciedlały odpowiednio wysokie i niskie wartości Q dla poszczególnych akcji.

W bardzo złożonych sytuacjach, przy długim wężu i ograniczonej przestrzeni, zdarzały się decyzje suboptymalne, co może wynikać z ograniczeń prostego wektora stanu.

5 Wnioski i Perspektywy Dalszego Rozwoju

5.1 Podsumowanie Osiągnięć

Projekt implementacji agenta DQN do gry Snake zakończył się pomyślnie. Agent wykazał zdolność do nauki i systematycznej poprawy strategii, co potwierdziła analiza metryk. Agent nauczył się unikać kolizji i aktywnie poszukiwać jedzenia.

5.2 Ograniczenia Projektu

Głównym ograniczeniem była liczba epizodów treningowych (800), podyktowana rosnącym czasem ich trwania. Prosta reprezentacja stanu może nie ujmować wszystkich niuansów gry.

5.3 Propozycje Dalszych Prac

Możliwe kierunki rozwoju:

- Kontynuacja treningu i jego optymalizacja.
- Eksperymenty z bardziej złożoną reprezentacją stanu (np. na podstawie obrazu gry z użyciem CNN).
- Modyfikacja systemu nagród (np. "reward shaping").
- Zastosowanie zaawansowanych algorytmów RL (np. Double DQN, Dueling DQN).

Literatura

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction (2nd ed.)*. MIT Press.

[2] Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.

[3] Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[4] Wang, Z., et al. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. PMLR.

A Wymagania Systemowe i Zależności

Do uruchomienia projektu wymagane są następujące biblioteki (wersje użyte w projekcie podano w nawiasach):

- Python (3.9.21)
- Pygame (2.6.1)
- TensorFlow (2.10.1)
- Keras (2.10.0, jako część TensorFlow)
- NumPy (1.26.4)

Pełna lista zależności znajduje się w pliku `requirements.txt` w repozytorium projektu.