Sprawozdanie MIO

Aleksander Siekliński, Wiktor Szewczyk, Jan Gorczyński

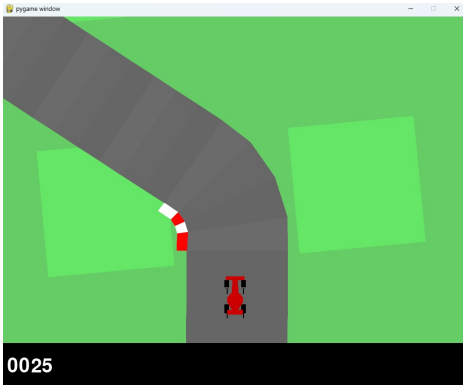
June 2024

1 Cel projektu

W naszym projekcie postanowiliśmy nauczyć sztuczną inteligencję grać w grę. Jako grę wybraliśmy 'CarRacing-v2' czyli jedną z gier dostępnych w bibliotece gymnasium. Jako algorytmów postanowiliśmy użyć algorytmów genetycznych oraz głębokiego uczenia ze wzmocnieniem.

2 Gra 'CarRacing-v2'

Gra 'CarRacing-v2' opiera się na pokonaniu toru przez samochód wyścigowy w jak najkrótszym czasie.W grze jako akcje przyjmujemy wektor trzech liczb, z których pierwsza odpowiada za skręt od -1 (w lewo) do 1 (w prawo), druga za gaz, a trzecia za hamulec. Po każdym wykonaniu akcji dostajemy z powrotem następny stan gry oraz nagrodę, która jest tym większa, im dalej i szybciej pojechaliśmy.



Rysunek 1: Gra 'CarRacing-v2'

3 Algorytm genetyczny

3.1 Wprowadzenie

Algorytm genetyczny jest zastosowany do optymalizacji sterowania samochodem w środowisku CarRacing-v2 z wykorzystaniem bibliotek takich jak DEAP oraz Gymnasium.

3.2 Inicjalizacja populacji

Algorytm rozpoczyna od inicjalizacji populacji osobników, gdzie każdy osobnik reprezentuje sekwencję akcji, które samochód ma wykonać w środowisku CarRacing-v2.

- **Reprezentacja chromosomu**: Każdy chromosom jest listą liczb zmiennoprzecinkowych, reprezentujących sekwencję działań (sterowanie samochodem). Wartości są generowane losowo w przedziale [-1, 1] przy użyciu funkcji `np.random.uniform`.

3.3 Ocena przystosowania

Każdy osobnik w populacji jest oceniany pod kątem przystosowania, które jest miarą jego zdolności do prowadzenia samochodu w środowisku testowym.

- **Funkcja przystosowania**: Dla każdego osobnika wykonywana jest symulacja jazdy, gdzie obliczana jest suma punktów nagród otrzymanych za każdy krok symulacji. Nagroda jest miarą sukcesu (prędkości, trzymania się toru oraz przejechanej odległości).

3.4 Selekcja

Algorytm wybiera osobników do reprodukcji na podstawie ich przystosowania (sukces przejazdu), stosując metodę selekcji turniejowej.

- **Selekcja turniejowa**: Losowo wybierana jest grupa osobników, a osobnik o najwyższym przystosowaniu w tej grupie jest wybierany do reprodukcji. Proces ten powtarza się, aby wyłonić pary rodziców do krzyżowania.

3.5 Krzyżowanie

Wybrani rodzice są krzyżowani w celu stworzenia nowych osobników (potomstwa) z prawdopodobieństwem `CXPB`.

- **Dwupunktowe krzyżowanie**: Chromosomy rodziców są dzielone w dwóch losowych punktach, a fragmenty są wymieniane, tworząc nowe chromosomy potomstwa.

3.6 Mutacja

Na potomstwo stosowana jest mutacja, aby wprowadzić różnorodność genetyczną i uniknąć zbieżności do lokalnego optimum.

- **Mutacja gaussowska**: Mała losowa zmiana jest wprowadzana do wartości w chromosomie, zgodnie z rozkładem normalnym. Szansa na mutację jest kontrolowana przez prawdopodobieństwo mutacji `MUTPB`.

3.7 Ewolucja

Proces selekcji, krzyżowania i mutacji powtarza się przez `NGEN`, aby ewoluować populację w kierunku coraz lepszych rozwiązań.

- **Pętla ewolucyjna**: Algorytm powtarza kroki selekcji, krzyżowania i mutacji przez określoną liczbę generacji.

3.8 Nagrywanie

Najlepszy osobnik z ostatniej generacji jest wybrany i jego akcje są nagrywane podczas symulacji wideo, aby wizualizować jego zachowanie.

- **Nagrywanie wideo**: Przy użyciu OpenCV, klatki z symulacji są zapisywane do pliku wideo, pokazując możliwości najlepszego osobnika.

4 Uczenie przez wzmocnienie

4.1 Proximal Policy Optimalization

W algorytmie PPO (Proximal Policy Optimization) sieć neuronowa uczy się wybierać najbardziej optymalną akcję w danym stanie gry. Model ten opiera się na dwóch głównych komponentach: 'actor' i 'critic'. Model 'actor' (aktor) to sieć neuronowa, która na podstawie stanu gry (wejścia w postaci kilku kolejnych ramek z gry, przetworzonych przez sieć konwolucyjną) zwraca wektor trzech liczb, które reprezentują najbardziej optymalną akcję do podjęcia w danym momencie. Model 'critic' (krytyk) to również sieć neuronowa, która przewiduje wartość (oczekiwaną sumę nagród) dla danego stanu gry. Krytyk pomaga aktorowi ocenić, jak dobre są podejmowane przez niego akcje. Podczas procesu uczenia, 'critic' jest trenowany do przewidywania oczekiwanej nagrody na podstawie stanu gry, a 'actor' uczy się wybierać akcje, które maksymalizują tę przewidywaną nagrodę, korzystając z informacji zwrotnej od 'critic'. Algorytm PPO stosuje techniki optymalizacji, które zapobiegają zbyt dużym zmianom w polityce (czyli strategii działania) aktora, co pomaga w stabilnym i skutecznym procesie uczenia. Po zakończeniu treningu, model 'actor' jest używany do przewidywania najbardziej korzystnej akcji w trakcie rzeczywistej gry. Do implementacji algorytmu PPO wykorzystaliśmy bibliotekę 'stable baselines3'.

4.2 Deep Q learning

W algorytmie DQN staramy się przewidzieć wartość Q, czyli oczekiwaną skumulowaną nagrodę dla danego stanu i akcji w grze. Algorytm wymaga skończonej liczby akcji, więc zdyskretyzowaliśmy przestrzeń akcji do 5 możliwych:

- (0,0,0) - nic nie rób,
- (0.25,0,0) - skręć lekko w prawo,
- (-0.25,0,0) - skręć lekko w lewo,
- (0,0.1,0) - użyj jedną dziesiątą mocy,
- (0,0,0.25) - lekko zahamuj.

Wyjściem sieci neuronowej jest 5 neuronów, z których każdy pokazuje oczekiwaną nagrodę dla danej akcji w obecnym stanie gry. Podczas trenowania sieci neuronowej, algorytm początkowo wykonuje losowe akcje, aby eksplorować przestrzeń stanów i akcji. W miarę postępu treningu, algorytm coraz częściej wybiera akcje proponowane przez sieć, zgodnie z polityką eksploracyjno-eksploatacyjną (epsilon-greedy). Po każdym epizodzie zapisujemy pary stan-akcja-nagroda oraz wynikające z nich następne stany do pamięci doświadczeń (replay buffer). Następnie, losowo wybieramy mini-batch z pamięci doświadczeń i uczymy model przewidywać wartości Q. Używamy do tego tzw. *target network*, czyli drugiego modelu, który stabilizuje proces uczenia. *Target network* ma taką samą architekturę jak główny model (*policy network*), ale jego wagi są aktualizowane rzadziej (co 5 kroków), aby przeciwdziałać niestabilnościom w uczeniu. Podczas uczenia optymalizujemy funkcję strat, która minimalizuje różnicę między przewidywaną wartością Q a docelową wartością Q wyznaczoną na podstawie *target network* i aktualnie otrzymanej nagrody. Wzór na docelową wartość Q (target Q-value) jest zazwyczaj dany przez:

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

gdzie r to otrzymana nagroda, γ to współczynnik dyskontowania, s' to następny stan, a a' to akcja maksymalizująca wartość Q w następnym stanie s'. Zaimplementowaliśmy algorytm DQN od zera w PyTorchu, ale nie udało nam się wyjść z minimum lokalnego, jakim jest jazda w przód.

5 Podsumowanie

Proces treningu tego typu algorytmów jest bardzo czasochłonny i trudny do zrównoleglenia. Udało nam się osiągnąć pewien stopień skuteczności, jednak nie wystarczyło nam czasu na udoskonalenie naszych algorytmów.