Sprawozdanie

Zastosowanie uczenia maszynowego do gry wyścigowej

Mateusz Burczaniuk,
Patryk Fijałkowski

25 stycznia 2021

Spis treści

1 Wprowadzenie

1.1 Problem

Omawiana gra wyścigowa symuluje rywalizację na torze pomiędzy kilkoma samochodami wyścigowymi. Tor, na którym odbywa się rywalizacja jest zamkniętą pętlą otoczoną niezbyt wysokimi ścianami. Poza tym na torze występują także przeszkody. Samochód może poruszać się do przodu, do tyłu, skręcać w lewo, prawo lub wykorzystać nitro. Zderzenie samochodu gracza ze ścianą lub przeszkodą powoduje natychmiastową śmierć gracza i koniec gry. Zderzenie dwóch graczy ze sobą powoduje analogiczne skutki (śmierć i usunięcie z rozgrywki) dla obu graczy.

Gracz ma zadanie przejechać zadaną trasę jak najszybciej, wyprzedzając swoich rywali. Podczas rywalizacji gracz może znacznie zwiększyć swoją prędkość przy użyciu nitro. Jednak maksymalna dostępna w danej chwili ilość nitro jest ograniczona, więc gracz nie może używać go bez przerwy. Zużyte nitro stopniowo odnawia się i po pewnym czasie wraca do pierwotnego poziomu. Może ono odnawiać się nieskończoną liczbę razy.

Wszystkie rozważane pojazdy mają taki same parametry techniczne i osiągi, w związku z czym wygrana lub przegrana w wyścigu zależy wyłącznie od obranego przez nie sposobu przemieszczania się po torze.

Zatem postawiony problem składa się z kilku zagadnień - należy zoptymalizować politykę doboru kierunku poruszania się gracza oraz używania nitro tak, aby przejechać całą trasę bez żadnego zderzenia, a jednocześnie jak najszybciej, pozostawiając rywali za sobą. Zadania te w pewnym stopniu konfliktują ze sobą, gdyż optymalną strategią w celu uniknięcia zderzeń jest stanie w miejscu, natomiast dążenie do uzyskania optymalnego czasu przejazdu powoduje wzrost ryzyka zderzenia z przeszkodą lub innym pojazdem.

1.2 Stan wiedzy

Omawiany problem reprezentuje zagadnienie kontroli w ciągłej przestrzeni stanów. Istnieje kilka sposobów rozwiązywania tego typu problemów. Do czołowych algorytmów w tej dziedzinie należą obecnie zarówno metody on-policy, jak i off policy. W niniejszej pracy w kontekście omawianego zadania przetestowano dwie stosunkowo młode metody, ale znacznie różniące się metody: PPO oraz SAC.

1.3 Użyte technologie

Prezentowane rozwiązanie zostało wykonane w silniku Unity w wersji 2019.4.11f1. Wykorzystano również framework ml-agents w wersji 0.21.1, odpowiedzialny za proces uczenia agentów.

1.4 PPO

Proximal Policy Optimization, czyli PPO to nowa metoda, wprowadzona w 2017 roku. Wykorzystuje ona metodę aktor- krytyk, posługując się dwoma sieciami neuronowymi. Aktor jest reprezentowany przez sieć neuronową, która wykonuje zadanie klasyfikacji, wybierając, jaką akcję podjąć przy obserwowanym stanie środowiska. Krytyk to druga sieć, o podobnej strukturze, która uczy się oceniać, czy poszczególne akcje poprawiają, czy pogarszają stan. Krytyk zwraca Q-wartość akcji wybranej w poprzednim stanie. Na tej podstawie aktor modyfikuje swoją politykę, dbając, by nie przeprowadzić zbyt dużych zmian.

W metodzie PPO aktualna polityka wykorzystywana jest do przeprowadzenia pewnej liczby doświadczeń (wielkość tę określa się mianem rozmiaru batcha) wymagających interakcji ze środowiskiem, które wykorzystywane są do poprawy polityki. Jest to uczenie on-policy, jako, że zebrane dane są wykorzystywane tylko raz.

Algorytm korzysta z mechanizmu Uogólnionej Estymacji Przewagi (ang. *Generalized Average Estimation*), aby ustalić, które akcje przyniosły, jakie skutki. Liczona jest ona od ostatnich wykonywanych akcji. Obliczane są kolejno wartości:

$$\delta = r_t + \gamma V(s_{t+1}) m_t - V(s_t) \tag{1}$$

$$gae_t = \delta + \gamma \lambda m_t gae_{t+1} \tag{2}$$

$$R_t(s_t, a_t) = gae_t + V(s_t) \tag{3}$$

gdzie:

- t - numer kroku analizowanego testu,
- s_t - stan w chwili t,
- a_t - akcja podejmowana w chwili t,
- $V(s)$ - wartość stanu s,
- r_t - nagroda w chwili t.
- m_t - maska w chwili t, jeśli w stan t jest końcowy, to 0, w przeciwnym wypadku 1.

Wystąpiły tutaj dwa hiperparametry algorytmu:
- γ - parametr dyskontowy, standardowo 0.99,
- λ - parametr wygładzający, standardowo 0.95.

Wyliczane jest ratio, które mówi, jak bardzo polityka uległa zmianie.

$$ratio = \pi_{new}/\pi_{old} \tag{4}$$

Polityka jest oceniana przez aktora i krytyka. Ocena krytyka to błąd średniokwadratowy pomiędzy wartością zwracaną a oszacowaniem krytyka.

$$critic\_loss = (R - V(s))^2 \tag{5}$$

Natomiast błąd aktora wyraża się wzorem:

$$actor\_loss = min(ratio * advantage, clip(ratio, 1-\varepsilon, 1+\varepsilon) * advantage) \tag{6}$$

ε to parametr obcinania, który zapewnia, że jednorazowo zmienimy politykę o co najwyżej $\varepsilon\%$. Domyślna wartość tego parametru wynosi 0.2.

Ostatecznie ocena polityki wyraża się wzorem:

$$total\_loss = critic\_loss * critic\_discount + actor\_loss - entropy \tag{7}$$

critic_discount to współczynnik mający na celu zrównoważenie wpływu na wynik oceny aktora i oceny krytyka. Entropia to natomiast hiperparametr metody, który domyślnie przyjmuje wartość 0.005.

Na podstawie powyższych funkcji oceny używane sieci uczone są przy użyciu metody stochastycznego przyrostu gradientu. Sieć jest uczona na danych z pojedynczego batcha przez pewną liczbę epok, będącą hiperparametrem metody.

Opis działania metody PPO został oparty na artykułach C. Trivediego [2],[3].

1.5 SAC

Soft Actor-Critic, czyli SAC to też stosunkowo nowa metoda do rozwiązywania tego typu problemów. W przeciwieństwie do PPO jest to metoda off-policy, czyli wyniki uzyskane przy użyciu jednej polityki są wykorzystywane do oceny następnych polityk.

Algorytm ten modyfikuje funkcję celu, aby maksymalizować nie tylko nagrody, ale również entropię polityki, aby wspierać eksplorację przestrzeni stanów. Funkcja ta jest zdefiniowana następująco:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha H(\pi(\cdot|s_t))] \tag{8}$$

α to parametr, który określa wagę entropii.

SAC używa trzech sieci neuronowych:

- określającej wartość stanu V z parametrem ψ,
- określającej funkcję Q z parametrem θ,
- określającej funkcję polityki z parametrem ϕ.

Sieć V jest uczona tak, aby zminimalizować błąd kwadratowy pomiędzy predykcją wartości zwracanej przez sieć a wartością oczekiwaną sumy funkcji Q i entropii:

$$J_V(\psi) = \mathbb{E}_{s_t \sim D}\Big[\frac{1}{2}\big(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}[Q_\theta(s_t, a_t) - log\pi_\phi(a_t|s_t)]\big)^2\Big] \tag{9}$$

Sieć Q minimalizuje błąd średniokwadratowy pomiędzy predykcją funkcji Q a natychmiastową nagrodą zsumowaną ze zdyskontowaną predykcją wartości następnego stanu.

$$J_Q(\theta) = (E)_{(s_t,a_t)\sim D}\Big[\frac{1}{2}\big(Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma \mathbb{E}_{s_{t+1}\sim p}[V_{\bar{\psi}}(s_{t+1})]\big)\Big] \tag{10}$$

Sieć π ma za zadanie minimalizować różnicę pomiędzy wartością zwracaną przez sieć a wartością $\frac{exp(Q_\theta(s_t,\cdot))}{Z_\theta(s_t)}$, gdzie Z to funkcja normalizująca. Matematyczny opis tej zależności wykorzystuje dywergencję Kullbacka-Leiblera i można go przedstawić przy użyciu wzoru:

$$J_\pi(\phi) = (E)_{s_t\sim D}\Big[D_{KL}\big(\pi_\phi(\cdot|s_t)||\frac{exp(Q_\theta(s_t,\cdot))}{Z_\theta(s_t)}\big)\Big] \tag{11}$$

Przy wyliczaniu tej wartości używana jest też pewna technika reparemetryzacji.

Schemat algorytmu przedstawiono na Listingu 1.

Opis algorytmu sporządzono w oparciu o artykuły [1],[4].

Algorithm 1 SAC

$\psi, \bar{\psi}, \theta, \phi$ - dane parametry
for $i = 0, \ldots, max\_iteration$ **do**
  for ruch środowiska **do**
    $a_t \sim \pi_\phi(a_t|s_t)$
    $_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end for
end for
Popraw parametry sieci przy użyciu gradientów poszczególnych funkcji dopasowania.

2 Eksperymenty

Przeprowadzone testy podzielono na dwie fazy. Pierwsza ma za zadanie określić najlepsze środowisko i zasady uczenia dla agentów. Z kolei druga faza skupia się na dobraniu jak najbardziej optymalnych parametrów w metodach PPO i SAC. W fazie pierwszej agenci byli uczeni do 100000 kroków, a w przypadku fazy drugiej – do 200000 kroków.

Za ocenę każdego z nauczonych modeli przyjęto dwie miary: k, u. Każda z miar mieści się w zakresie $[0, 1]$ i ma ocenić skuteczność wyuczonego modelu w różnych warunkach. Obie miary to stosunek średniej liczby przejechanych punktów kontrolnych (po 50 podejściach agenta do wyścigu) do pewnej empirycznie ustalonej wartości. W każdym z podrozdziałów wartość była inna, co dało możliwość znormalizowania wyników i w konsekwencji przejrzystszą ich analizę.

- Miara k (ang. *known*) - określa, jak dobrze model radzi sobie na torze, na którym przebiegało uczenie.
- Miara u (ang. *unknown*) - określa, jak dobrze model radzi sobie na nowym torze.

2.1 Faza pierwsza

2.1.1 Wypłaty

Sprawdzono różne podejścia nagradzania agentów. Trzy zasadnicze podejścia do udzielania wypłat:

- wypłata za przejechanie punktu kontrolnego,
- wypłata zwiększana z czasem życia agenta,
- wypłata zależna od aktualnej prędkości.

W ramach zachęcenia agentów do przemierzania toru zgodnie z założeniami, na obu torach testowych umieszczono 22 punkty kontrolne. Agent w momencie przejechania przez dany punkt dostaje stałą wypłatę. Wyniki przeprowadzonych testów zostały zaprezentowane w tabeli 1.

Analizując wyniki eksperymentu, zauważamy że warto wprowadzać wysokie wypłaty za punkty kontrolne, ponieważ to jednoznacznie poprawia obie miary. W przypadku nagradzania agenta za długość życia nie zauważono znaczącej poprawy w kontekście osiągów na torach testowych.

Prędkość	Długość życia	Punkty kontrolne	Miara k	Miara u
0.15	0	10	1	0.41
0.15	0	6	0.64	0.15
0.15	0	2	0.68	0.18
0	0.1	10	0.47	0.14
0	0.2	10	0.52	0.17
0	0.3	10	0.55	0.08

Tab. 1: Testy wypłat

2.1.2 Raycasty

Postanowiono sprawdzić wpływ liczby obserwacji odległości agenta od najbliższej przeszkody. W każdym z przypadków wysyłano raycasty z dwóch przednich narożników samochodu. Promienie były ustawione do siebie pod takim kątem, by pierwszy z nich celował z frontu samochodu, a ostatni był pod kątem 60°. Zgodnie z tabelą 2, zbadano cztery możliwości – wysyłanie odpowiednio 6, 8, 10 i 12 raycastów przez agenta.

Na podstawie uzyskanych wyników można stwierdzić, że najoptymalniej jest wysyłać 10 promieni. Zmniejszenie ich liczby do 6 i 8 nie wiąże się z istotnym pogorszeniem jakości wynikowych polityk. Jednak zwiększenie liczby promieni do 12 powoduje zauważalny spadek skuteczności agenta zarówno na torze znanym, jak i nowym.

Liczba promieni	Miara k	Miara u
6	0.93	0.22
8	0.92	0.33
10	1	0.42
12	0.77	0.21

Tab. 2: Testy racyastów

2.2 Faza druga

2.2.1 PPO

W kontekście PPO postanowiono sprawdzić wpływ parametru obcinania ε oraz liczbę epok uczenia sieci neuronowej. Wyniki ukazano w tabeli 3.

Liczba epok	Parametr obcinania	Miara k	Miara u
3	0.1	0.89	0.32
3	0.2	0.88	0.33
3	0.3	1	0.37
5	0.3	0.96	0.31
10	0.3	0.93	0.34

Tab. 3: Testy PPO

2.2.2 SAC

W przypadku algorytmu SAC badano wpływ na wyniki początkowego ustawienia wartości parametru α związanego z entropią. Zbadano również działanie parametru opisującego liczbę doświadczeń zbieranych do bufora przed aktualizacją polityki. Domyślnie wartość ta wynosi 0, ale powszechne jest używa się w tym miejscu 1000 - 10000 doświadczeń. Wyniki zostały zaprezentowane w tabeli 4.

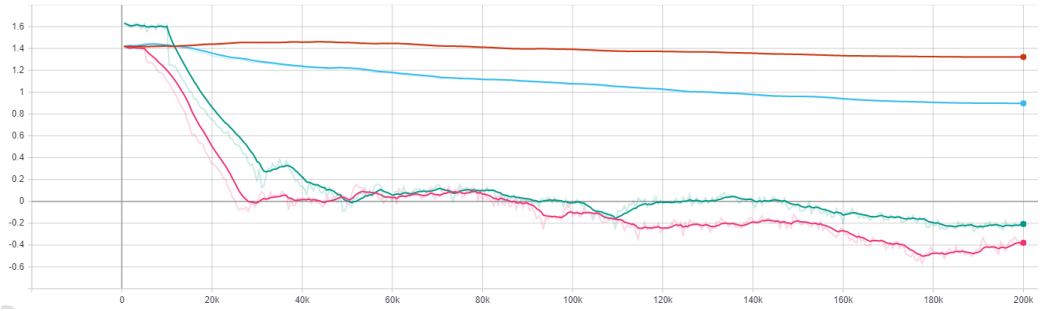
Wartość α	Liczba doświadczeń	Miara k	Miara u
0.5	0	0.89	0.11
0.7	0	1	0.22
1	0	0.82	0.19
0.7	1000	0.66	0.12
0.7	10000	0.54	0.21

Tab. 4: Testy SAC

3 Obserwacje

3.1 Entropia

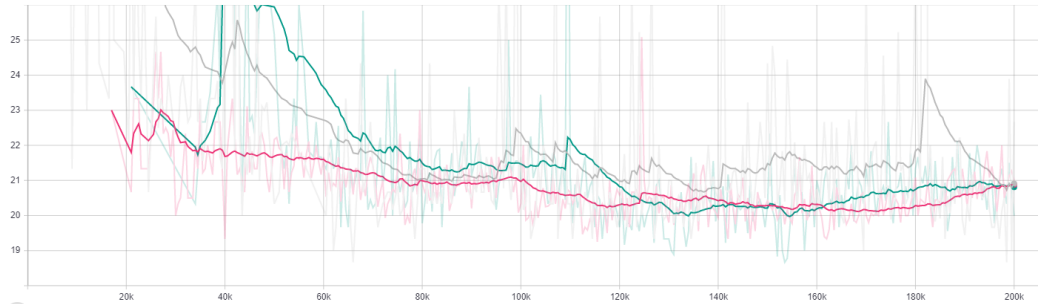
Podczas analizy wyników zaobserwowano istotną różnice między metodami uczenia PPO i SAC. Wykres ukazujący zmiany entropii został ukazany na rysunku 1. Dla przejrzystości zamieszczono 4 krzywe - 2 górne odpowiadają pewnym modelom metody PPO, a 2 dolne – SAC. Zgodnie z teoretycznymi założeniami, entropia w procesie uczenia dla PPO zmniejsza się wolniej i stabilniej.



Rys. 1: Entropia

3.2 Szybkość ukończenia wyścigu

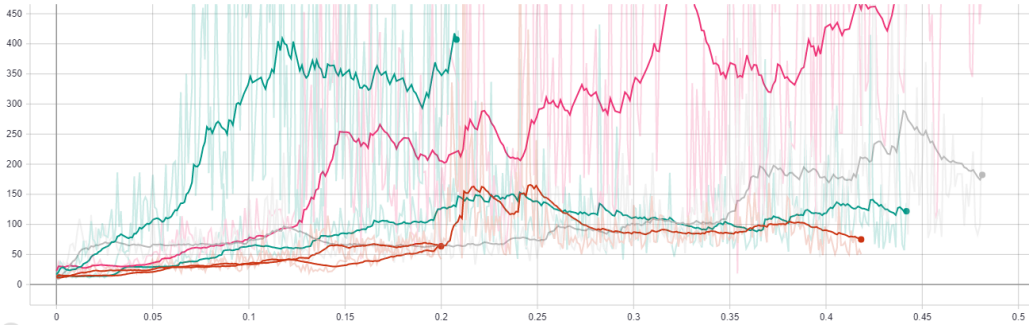
Po przeprowadzeniu testów zauważono również, że mimo delikatnej tendencji spadkowej, średnie czasy ukończenia wyścigów w trakcie uczenia nie zmieniają się znacznie, co ukazano na wykresie z rysunku 2. To oznacza, że agenci wraz z procesem uczenia nie uczą się przejeżdżać toru szybciej, a jedynie skuteczniej omijać przeszkody, co daje im dłuższy czas życia.



Rys. 2: Szybkość ukończenia wyścigu

3.3 Długość epizodów

Długość epizodów uczenia w przypadku obu metod jednoznacznie rośnie, co również wykazuje poprawny charakter uczenia agentów. W przypadku metody SAC (która sprawdziła się lepiej w badanym problemie) długość epizodów rośnie szybciej i uzyskuje wartości $1.5-3$ razy wyższe. Trzy najwyższe krzywe z rysunku 3 odpowiadają lepszej metodzie SAC, a trzy niższe – PPO.



Rys. 3: Długość epizodów

4 Wnioski

Analizując wszystkie wyniki eksperymentów, zauważono że polityki utworzone metodą PPO, mimo że uzyskiwały niższe wyniki na torach, to relatywnie lepiej odnajdywały się w nowym środowisku - miały wyższą miarę u. Zdaje się, że agenci korzystający z polityk metody PPO ostrożniej zachowywali się na zakrętach, unikając w ten sposób kolizji z przeszkodami. W obu przypadkach agenci podejmowali dobre decyzje w stałych fragmentach, tj. używali nitro na prostych odcinkach oraz modyfikowali tor jazdy, gdy w zakresie widzenia pojawiały się przeszkody.

Kluczowe w kontekście narzucenia dobrego rytmu uczenia agentom było nagradzanie agentów za osiąganie kolejnych punktów kontrolnych (co wykazano w fazie pierwszej testów). Ponadto, dostosowując parametry obu metod w drugiej fazie testów można istotnie wpłynąć na proces uczenia, co przełoży się na widocznie inne polityki. Co za tym idzie, warto dostosowywać hiperparametry różnych metod, by uzyskane wyniki były jak najbardziej optymalne.

Po ukończeniu eksperymentów, ustawiono 4 najlepsze modele agentów (2 dla PPO i 2 dla SAC) do wspólnego wyścigu. Wyuczone polityki agentów pozwoliły na przeprowadzenie wyrównanych i niejednoznacznych rozgrywek. Potwierdziły się wnioski związane z dysproporcją miar k i u dla obu metod. Na znanym torze metoda SAC sprawdzała lepiej – agenci korzystający z polityki wyuczonej metodą SAC jeździli ryzykowniej i lepiej wykorzystywali swoje nitro. Z kolei w nowym środowisku lepiej wypadali agenci uczeni metodą PPO – zachowywali ostrożność w nieznanych elementach toru.

Literatura

[1] Kumar V., *Soft Actor-Critic Demystified*, https://towardsdatascience.com/soft-actor-critic-demystified-b8427df61665. 2019.

[2] Trivedi C., *Proximal Policy Optimization Tutorial (Part 1/2: Actor-Critic Method)*, https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6, 2019.

[3] Trivedi C., *Proximal Policy Optimization Tutorial (Part 2/2: GAE and PPO loss)*, https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-2-2-gae-and-ppo-loss-fe1b3c5549e8, 2019.

[4] *SAC (soft actor-critic) algorithm for reinforcement learning*, https://www.programmersought.com/article/77705189397/