Dokumentacja końcowa projektu nr 10

Jarosław Nachyła, Marta Wojtkowska

20.01.2023

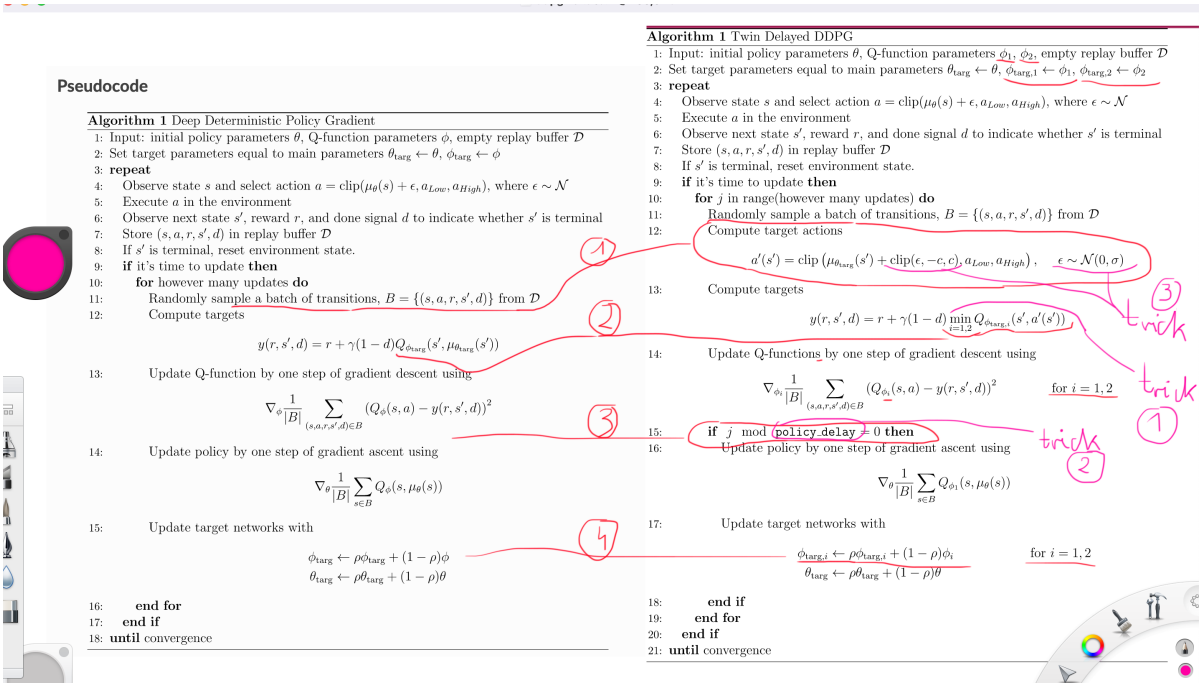
1 Temat projektu

Zaimplementować algorytm TD3, zoptymalizować hiperparametry algorytmów TD3 i FastACER na środowisku HalfCheetach. Porównać z wynikami i odpowiednikami z PyBullet.

2 Implementacja algorytmu TD3

2.1 Koncepcja

Zadanie zaimplementowania algorytmu TD3 podzielone zostało na kilka etapów. Na początku zaimplementowano algorytm Aktor - Krytyk. Następnie, w kroku drugim algorytm przerobiono na DDPG. Ostatnim krokiem było przerobienie algorytmu DDPG na TD3. Pseudokod dla tych dwóch algorytmów pobrano z OpenAI Spinning up. Poniżej przedstawiono ich porównanie - na kolorowo zaznaczono różnice.



Rysunek 1: Porównanie pseudokodu algorytmów DDPG oraz TD3.

Algorytm Twin Delayed DDPG w stosunku do DDPG wprowadza trzy modyfikacje:

1. Wykorzystanie dwóch krytyków oraz dwóch target krytyków. Razem z aktorem i target aktorem TD3 korzysta zatem z aż sześciu sieci neuronowych. Do aktualizowania wag sieci krytyków wykorzystywane jest minimum z wyjść target krytyków. Pozwala to uniknąć "zawyżania" wartości stanów (ang. overestimation bias).

2. Opóźnienie aktualizacji polityki względem aktualizacji funkcji Q. To, jak często następuje aktualizacja polityki zależy od hiperparametru algorytmu, polecaną wartością jest jedna aktualizacja polityki na dwie aktualizacje funkcji Q. Ta modyfikacja daje krytykom czas na zbiegnięcie.

3. Dodanie szumu do wyjścia target aktora i podwójne clipowanie. Wyznaczanie akcji dla nowych stanów z pamięci odbywa się przez dodanie do wyjścia target aktora clipowanego szumu, a następnie clipowanie wyniku do skrajnych wartości akcji dostępnych w danym środowisku. Clipowanie to operacja "przycięcia" zmiennej - jeśli wartość mieści się w dostępnym przedziale pozostaje bez zmian, natomiast jeśli przekracza granicę obszaru wartość clipowanej zmiennej ustawiana jest jako granica przedziału, poza którą wybiega.

$$a'(s') = clip(\mu_{\theta targ}(s') + clip(\epsilon, -c, c), a_{low}, a_{high}), \quad \epsilon \sim N(0, \sigma) \tag{1}$$

2.2 Omówienie kodu

Instrukcja stworzenia wirtualnego środowiska oraz instalacji wymaganych bibliotek znajduje się w pliku README w repozytorium na GitHub (https://github.com/jnachyla/usd-proj).

Metody get_actor oraz get_critic stanowią "wzorzec", na podstawie których tworzone są odpowiednio sieci aktora, target aktora oraz sieci krytyków i target krytyków. Zgodnie z algorytmem sieć aktora przyjmują na wejściu stan i na wyjściu dają akcję do wykonania. Krytyk natomiast przyjmuje stan i akcję i ocenia jak dobrym wyborem było wykonanie danej akcji w danym stanie.

Klasa OUActionNoise pozwala na dodanie szumu Ornstein–Uhlenbeck'a do akcji zwróconej przez aktora. Ten rodzaj szumu występował w oryginalnej implementacji algorytmu.

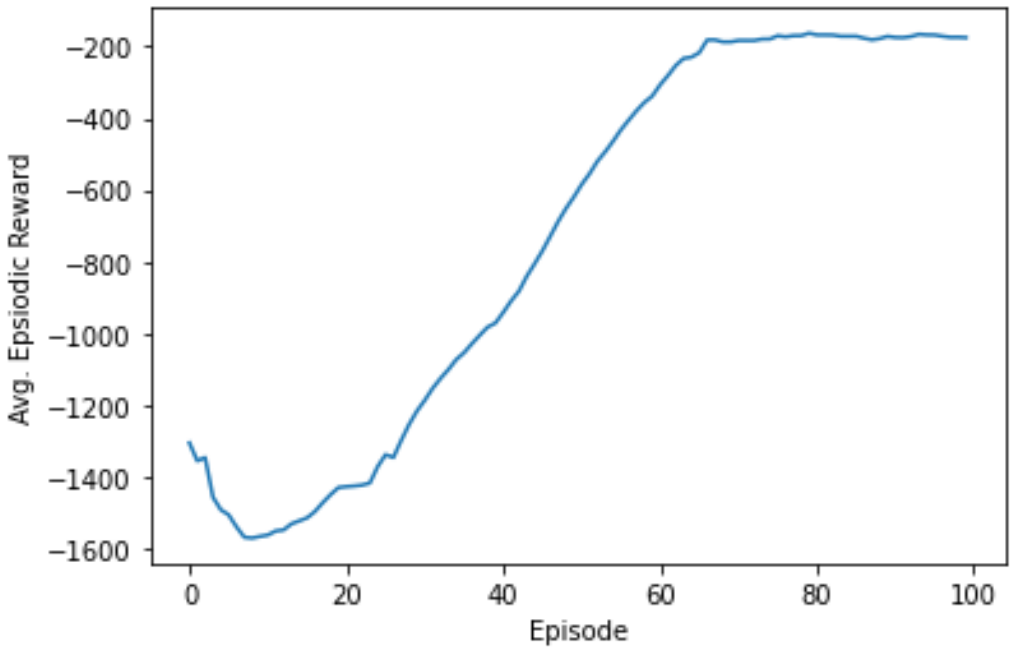
Klasa Buffer stanowi pamięć agenta, przechowywane są w niej stany, wykonane akcje, otrzymane nagrody oraz nowe stany, w których agent znalazła się po wykonaniu akcji. Metoda update służy do aktualizowania parametrów sieci aktora i sieci krytyka. Metoda noised_actions pozwala na dodanie szumu do akcji target aktora, czyli realizację modyfikacji oznaczonej numerem 3 na rys.1. Metoda double_target_critic zwraca minimum z wyjść obu target krytyków, które zostaje wykorzystane do realizacji modyfikacji oznaczonej numerem 1 na rys.1.

Metoda update_target jest wykorzystywana do aktualizacji wag sieci target. Wagi te aktualizowane są przez tak zwany "soft update", czyli bardzo powoli, zależnie od parametru tau.

Niektóre z działań wykonywanych w mainie to: definiowanie problemu, tworzenie sieci, przypisywane wartości parametrom oraz rysowanie wykresu.

2.3 Wyniki, analiza i wnioski

Poniżej przedstawiono wykres średniej nagrody z ostatnich 40 epizodów dla zaimplementowanego algorytmu TD3.



Rysunek 2: Wykres średniej nagrody w epizodzie w funkcji liczby epizodów dla TD3.

Rezultaty są zgodne z przewidywaniami - wraz ze wzrostem liczby epizodów średnia nagroda w epizodzie rośnie - agent uczy się wybierać lepsze akcje i zbiera coraz lepsze, czyli mniej ujemne nagrody. Po około 60 epizodach widać wypłaszczenie krzywej - nagrody zbierane przez agenta utrzymują się w przybliżeniu na stałym poziomie. Można uznać, że agent osiągnął najlepszy rezultat jaki był w stanie dla danego problemu i że dalsza nauka nie przyniesie poprawy wyników.

Wyniki algorytmu poddano ewaluacji za pomocą kodu umieszczonego w pliku evaluators.py i klasy Td3EvalModel. Otrzymano następujący rezultat:

Evaluation over 100 episodes: -156.063194 with standard deviation: 2.323086.

3 Tuning hiperparametrów

3.1 Założenia i środowisko

Tuning hiperparametrów wykonywany jest metodą random search. Parametry algorytmów losowane były z rozkładu normalnego, stosując jako średnie wartości bliskie lub minimalnie zmienione wartości domyślne algorytmu TD3 dla implementacji stable baselines 3 (https://stable-baselines3.readthedocs.io/en/v1.0/modules/td3.htmlparameters). Jako referencyjny wynik użyliśmy wyniku dla TD3 ze źródła: https://huggingface.co/sb3/td3-HalfCheetah-v3. Referencyjną wartością, którą staraliśmy się poprawić było zatem: 9709.01 (+/- 104.84) dla HalfCheetah v3.

W skrócie eksperyment został przeprowadzony z nastepującymi parametrami:

- liczba epizodów trenowania: 50
- długość epizodu: 1000
- liczba kroków trenowania: 500e3
- liczba epizodów ewaluacji: 100

3.2 Tuning TD3

Poniższa tabela przedstawia listę parametrów algorytmu TD3, wraz z wartościami służącymi do losowania (średnia i odchylenie), a także wartościami minimalnymi i maksymalnymi.

Nazwa Parametru	Średnia	Odchylenie stand.	Minimum	Maksimum
learning_rate	1.0e-04	1.0e-04	1.0e-07	1.0e-03
buffer_size	1000000	10000	10000	100000000
train_frequency	1	4	1	10
tau	0.01	0.05	0.0001	1.0
policy_delay	2	4	1	10
target policy std noise	0.1	0.05	0.0001	2.0

Tak więc optymalizowano learning rate wspólny dla sieci krytyków i aktora, rozmiar bufora powtarzania, częstotliwość trenowania, parametr tau (aktualizacji sieci target), opóźnienie aktualizacji polityki i odchylenie standardowe szumu dla sieci target polityki.

3.3 Implementacja

Implementacja tuningu hiperparametrów TD3 znajduje się w pliku: td3_ht.py. Znajduje sie tam funkcja random_hiperparameter służąca do losowania poszczególnych parametrów i ograniczania (clipping) ich na końcu. Tunning wykonywany był w środowisku Google Colab ze względu na darmowe zasoby obliczeniowe GPU.

3.4 Rezultaty TD3 tuningu hiperparametrów.

lr	buffer_size	train_frequency	tau	policy_delay	tar.policy std noise	Res. Mean	Std
2.06e-05	1e6	1	0.01	2	0.1	2254.41	343.67
0.00091	959817	1	0.052	2	0.13	9180.29	90.61
1e-06	984590	1	0.0001	2	0.149	510	320.45

Najlepszy rezultat uzyskano dla drugiego zestawu parametrów przedstawionego w tabeli. Wynik jest nie znacznie gorszy od referencyjnego. Należałoby przeprowadzić grid search wokół uzyskanych wartości w celu znalezienia dokładniejszego maksimum. Część wartości nie została zaprezentowana w tabeli algorytm dla nich nie był w stanie uzyskać wartości powyżej 100 lub miał wartości ujemne (naukę wtedy przerywano).

4 Źródła

1. https://www.youtube.com/watch?v=1lZOB2S17LUt=802s
2. https://keras.io/examples/rl/ddpg_pendulum/introduction
3. https://huggingface.co/models?library=stable-baselines3sort=downloadssearch=Pendulum
4. https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic
5. https://deepai.org/publication/controlling-an-inverted-pendulum-with-policy-gradient-methods-a-tutorial
6. https://www.gymlibrary.dev/