Uczenie Maszynowe: Laboratorium 02

Radosław Łazarz

1 Wstęp

Celem drugiego laboratorium będzie dokończenie implementacji algorytmu uczącego się typu aktor-krytyk (wariant TD(0) z-polityką), a następnie sprawdzenie jak spisze się w rozwiązywaniu dwóch epizodycznych problemów: prostego (balansowanie tyczką na ruchomym wagoniku) oraz trudniejszego (bezpieczne dotarcie lądownikiem na powierzchnię planety).

2 Narzędziownik

Do wykonania zadania wykorzystamy bibliotekę `tensorflow` wraz z jej podbiblioteką `keras` oraz rozszerzeniem `tensorflow_probability`. W szczególności zaś przydatne będą następujące klasy:

- `tf.keras.Model` — posłuży do skonstruowania duetu sieci neuronowych (a w tym konkretnym przypadku, przynajmniej początkowo: jednej sieci z "rozdwojoną" końcówką), które wykorzystamy potem jako aktora i krytyka; w szczególności warto zwrócić uwagę na argumenty `inputs` i `outputs`;
- `tf.keras.optimizers.Adam` — optymalizator gradientowy, który wykorzystamy w miejsce klasycznego SGD (jest w tej sytuacji nieco wydajniejszy);
- `tf.keras.layers.Input`, `tf.keras.layers.Dense` i `tf.keras.layers.LayerNormalization` — warstwy z których będzie budowana nasza prosta sieć;
- `tf.GradientTape` — klasa, która pozwoli obliczyć gradienty dla naszych nietypowych funkcji straty; nie będziemy przecież robić "zwykłej" klasyfikacji, a przybliżenie polityki i wartościowania stanów;
- `tfp.distributions.Categorical` — pomocnicza klasa, która pozwoli potraktować wartości zwracana przez aktora jak politykę i losować ak-

cję do wykonania zgodnie z aktualnym rozkładem ich prawdopodobieństwa; ponieważ jest to część frameworku `tensorflow` to łatwo będzie też policzyć odpowiednie gradienty.

3 Metoda aktor-krytyk

---

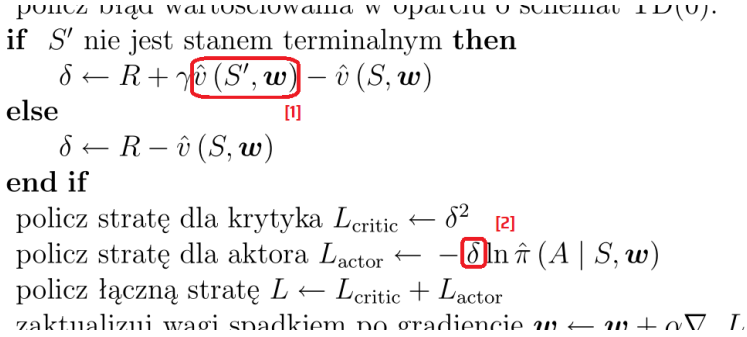
for każdy epizod **do**
  $S \leftarrow$ początkowy stan
  for każdy krok **do**
    wybierz A w oparciu o $\hat{\pi}(\cdot \mid S, \boldsymbol{w})$
    wykonaj akcję A
    zaobserwuj nagrodę R i nowy stan S'
    policz błąd wartościowania w oparciu o schemat TD(0):
    if S' nie jest stanem terminalnym **then**
      $\delta \leftarrow R + \gamma \hat{v}(S', \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w})$
    else
      $\delta \leftarrow R - \hat{v}(S, \boldsymbol{w})$
    end if
    policz stratę dla krytyka $L_{\text{critic}} \leftarrow \delta^2$
    policz stratę dla aktora $L_{\text{actor}} \leftarrow -\delta \ln \hat{\pi}(A \mid S, \boldsymbol{w})$
    policz łączną stratę $L \leftarrow L_{\text{critic}} + L_{\text{actor}}$
    zaktualizuj wagi spadkiem po gradiencie $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \nabla_{\boldsymbol{w}} L$
    $S \leftarrow S'$
  end for
end for

W tym schemacie uczenia mamy do czynienia z dwoma przybliżeniami funkcji: $\hat{\pi}(A \mid S, \boldsymbol{w})$ (czyli aproksymowaną polityką) oraz $\hat{v}(S, \boldsymbol{w})$ (czyli aproksymowanym wartościowaniem stanów). Pierwszą nazywa się zwyczajowo aktorem, drugą zaś krytykiem. Obie są parametryzowane przez pewien zbiór wag $\boldsymbol{w}$. W praktyce te zbiory wag są często rozłączne — tu jednak by przyśpieszyć proces uczenia pozwolimy by początkowo współdzieliły one część z nich.

W trakcie uczenia aktor i krytyk wpływają na siebie wzajemnie. Krytyk wartościuje stany w oparciu o aktualną politykę, więc wszelkie zmiany w aktorze zmuszą go do aktualizacji estymat, gdyż zmianie ulega wtedy właśnie sama polityka. Z drugiej strony — aktor poszukuje optymalnej polityki w oparciu o bootstrapping TD(0), wykorzystując krytyka jako funkcję wartościującą. Jeżeli krytyk zacznie inaczej oceniać stany, to inne akcje mogą okazać się tymi właściwymi.

Sam algorytm przybliża podany wcześniej pseudokod (γ to discount factor, a α to rozmiar kroku uczącego).

Strata krytyka to zwykły błąd średniokwadratowy — chcemy po prostu by jego predykcje były jak najbardziej zgodne z obserwowaną rzeczywistością. Strata aktora wynika wprost z twierdzenia o gradiencie polityki. Warto zauważyć, że efektywnie nie jest to strata, a zysk (chcemy maksymalizować efektywność polityki, nie minimalizować) — dlatego konieczne jest przemnożenie straty przez -1, by mogła być wykorzystywana we frameworku opartym o spadek (a nie wznoszenie po gradiencie).

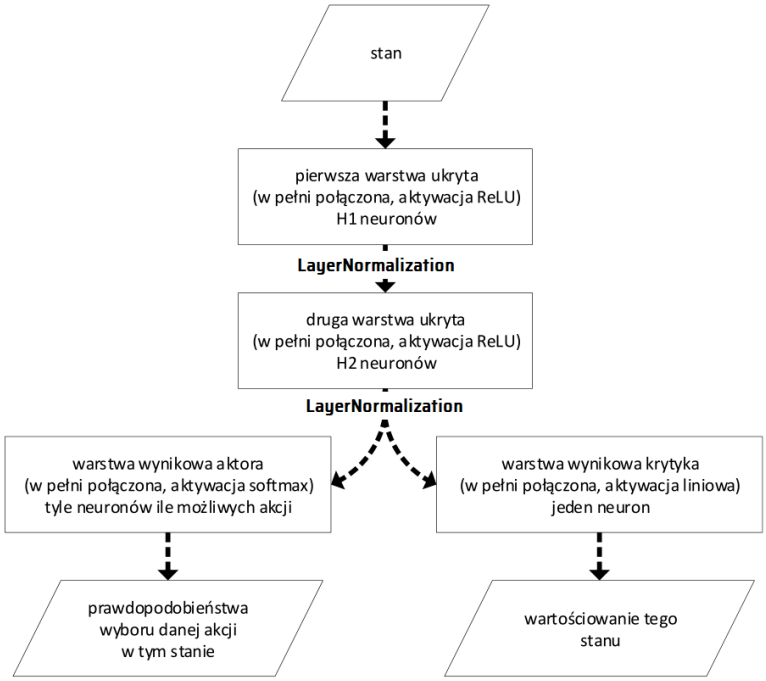


Rysunek 1: Tutaj trzeba uważać (patrz: szczegóły w tekście)!

Należy też pamiętać, iż niektóre części powyższych wzorów (`[1]` w przypadku krytyka, `[2]` w przypadku aktora — patrz: Rysunek 1) powinny być traktowane jak stałe i w ten sposób brać udział w wyliczaniu funkcji straty (i późniejszych pochodnych z niej). W przeciwnym wypadku aktor poprawi swoją sytuację zmieniając wagi krytyka tak, by ten dobrze go oceniał (zamiast zmienić swoje wagi — i swoje zachowanie). Bardzo życiowa sytuacja — i potencjalnie duża przeszkoda w efektywnym uczeniu `;]`. W `TensorFlow` można to zrobić poprzez np. brutalną zamianę (śledzonego przez `GradientTape`) tensora `x` na (bezpieczną i nieśledzoną) liczbę zmiennoprzecinkową `float(x.numpy())`.

4 Jakiej sieci neuronowej użyjemy?

Do konkretnej implementacji algorytmu użyjemy jednej z najbardziej uniwersalnych technik aproksymacji funkcji — (niezbyt) głębokiej sieci neuronowej. Nasza realizacja będzie posiadać dwie warstwy ukryte, współdzielone przez aktora i krytyka. Jej schematyczną strukturę zaprezentowano na zamieszczonym diagramie. Warstwa wyjściowa aktora korzysta z aktywacji softmax — podobnie jak przy problemach klasyfikacji potrzebujemy przypisać praw-



Rysunek 2: Schemat sieci neuronowej, której użyjemy w zadaniu.

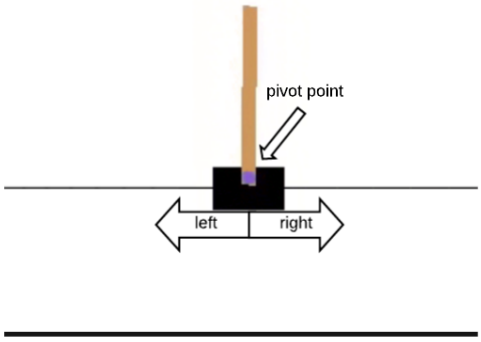
dopodobieństwa do elementów pewnego zbioru. Warstwa wyjściowa krytyka korzysta z atywacji liniowe, czyli efektywnie nie używa żadnej aktywacji: chcemy zapewnić jej pełną siłę wyrazu w całym zakresie pracy.

W podstawowej wersji nasza sieć powinna mieć H1 = 1024 neurony w pierwszej warstwie ukrytej i H2 = 256 neuronów w drugiej warstwie ukrytej. Można przyjąć *discount factor* $\gamma = 0.99$. Układy aktor-krytyk są niezwykle niestabilne, a dodatkowo uczenie naszej sieci opiera się na pojedynczych przykładach (a nie ich batchach) — musimy więc modyfikować wagi bardzo powoli. W tym celu przyjmujemy rozmiar kroku uczącego $\alpha = 0.00001$ (tak, tu są cztery zera po przecinku!).

5 Problem balansowania kijkiem

Celem jest utrzymanie kijka w pionie jak najdłużej. Za każdy krok bez przewrócenia kijka otrzymujemy nagrodę 1. Epizod kończy się po wywrotce lub po 500 krokach.

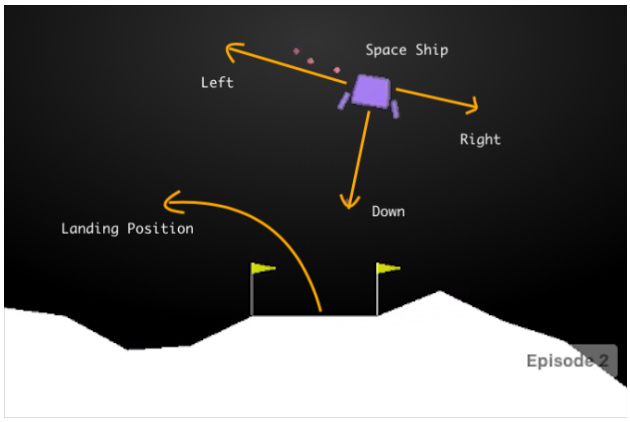
Stan składa się z czterech elementów: pozycji wagonika na którym stoi



kijek, jego prędkości, kąta wychylenia kijka oraz jego prędkości kątowej. Do wyboru są tylko dwie akcje: jedź w lewo i jedź w prawo.

Nie jest to tak eskcytujący problem jak obsługa lądownika, będzie jednak robił za dobry punkt startowy dla debugowania algorytmu. Ten problem da się oczywiście rozwiązać szybciej z użyciem bardziej prymitywnych metod, warto jednak zauważyć, że nasz aktor-krytyk nie korzysta z żadnych założeń odnośnie reprezentacji stanu (poza jego rozmiarem) oraz żadnej wiedzy domenowej o problemie — a to bardzo wygodne!

6 Problem lądownika



Celem jest wylądowanie na płaskim obszarze między chorągiewkami. Za

zbliżenie się do celu dostajemy niewielką nagrodę, za oddalanie się dostajemy niewielką karę. Suma tych nagród dla pełnej drogi do lądowiska wynosi około 100–140 (zależy od tego jak daleko od celu rozpoczniemy epizod). Dostajemy również nagrodę równą 10 gdy każda z nóg złapie kontakt z podłożem oraz analogiczną karę gdy kontakt ten straci. Lądownik dostaje też końcowy bonus 100 za bezpieczne lądowanie (gdziekolwiek) i karę -100 za roztrzaskanie się. Dodatkowo każde użycie głównego, dolnego silnika wiążę się z karą -0.3 za marnowanie paliwa.

Stan składa się z ośmiu elementów: położenia w poziomie, położenia w pionie, prędkości w pionie, prędkości w poziomie, kąta nachylenia, prędkości kątowejm, informacji o tym czy lewa noga ma kontakt z ziemią oraz o tym czy ma go noga prawa. Do wyboru są cztery akcje: nie robienie niczego, odpalenie lewego silnika manewrowego, odpalenie głównego silnika na spodzie, odpalenie prawego silnika manewrowego.

Problem jest znacznie trudniejszy, a proces uczenia może wydawać się niestabilny (okresy udanych prób agenta przeplatane są niekiedy kompletnymi porażkami). Korzystamy więc z najlepszego spośród wariantów systamu uczącego (ustalonego wcześniej dla problemu z kijkiem). Zapisujemy też regularnie model, by móc powrócić do dobrze funkcjonującej wersji w przypadku destabilizacji aktualnych wag.

7 Co jest do zrobienia?

W ramach laboratorium wykonujemy następujące zadania.

- Ściągamy kod z załącznika, uzupełniamy luki w oparciu o sugestie w komentarzach i załączony wyżej opis algorytmu.
- Uruchamiamy procedurę uczenia dla problemu z kijkiem i obserwujemy postępy. Pierwsze kilkaset iteracji może być lekko opornych, ale potem proces uczenia powinnien przyśpieszyć i zakończyć sie sukcesem najpóźniej w okolicach iteracji numer 1000-2000 (w praktyce w okolicach 500 powinno być realnie nieźle). Obserwujemy wykresy pokazujące przebieg procesu — czy widzimy jakąś zależność między błędem wartościowania, a otrzymywanymi nagrodami? Pamiętajmy, by na koniec uczenia zapisać finalny model na dysku!
- Połączony początek sieci neuronowej w praktyce bardziej nam przeszkadza, niż pomaga. Zamieńmy go na dwie rozłączne części (czyli: aktor ma swóje własne warstwy początkowe, a krytyk swoje — rozmiar warstw ukrytych pozostaje taki sam, po prostu istnieje ich dwa razy więcej). Powtórz eksperyment z poprzedniego punktu. Czy teraz system zachowuje się stabilniej?

- Przetestujmy część krytykującą wyuczonego modelu dla kilku ręcznie dobranych wartości. Jak wartościuje stan, w którym kijek jest pionowy, a prędkości zerowe? Jak wartościuje stan, w którym kijek zaczyna się szybko przechylać? Jak stan, w którym wagonik jest blisko krawędzi obszaru?
- Wykorzystana sieć zawiera sporo neuronów. Czy tak duża ich liczba jest konieczna? Uruchom trening ponownie, ale tym razem dla H1 = 128 i H2 = 32. Czy uczenie przebiega równie skutecznie?
- Przyjęty rozmiar kroku uczącego jest bardzo mały. Zwiększmy go do $\alpha = 0.01$. Jak teraz przebiega uczenie?
- Pora zabrać się za problem z lądowaniem. W tym celu wróćmy do pierwotnego kroku uczącego i rozmiaru warstw ukrytych. Pamiętajmy też o odpowiednim zmniejszeniu kroku uczącego (powinien być zauważalnie mniejszy, niż dla kijka). Zmodyfikujmy też odpowiednie rozmiary wejścia i wyjścia. Przeprowadźmy proces uczenia, zapiszmy wynikowy model i wykres przedstawiający przebieg procesu (tutaj uczenie może trwać dość długo — nawet do 3000 kroków (choć znów: koło 500 kroków wyniki są zwykle bardzo, bardzo przyzwoite), a każdy z nich jest dłuższy niż w przypadku kijka — warto zarezerwować na to czas). Jeżeli mamy problemy z czasem trenowania, to można zapisać też pośredni model i wrócić do treningu później, wczytując go jako stan startowy.
- Ponownie przetestuj dla ręcznie wybranych przypadków wyuczonego aktora i krytyka. Jaką strategię lądowania przyjął (obejrzyj animację)? W jakich sytuacjach uruchamia dolny silnik? Jak wartościuje stan tuż przed lądowaniem? Jak stan, w którym lądownik jest silnie przechylony? Jaką akcję proponuje, by zredukować przechył?
- Wykresy z przebiegów uczenia i odpowiedzi na zadane pytania wrzucamy do krótkiego raportu, a ten wraz z kodem rozwiązania w odpowiednie miejsce na platformie e-learningowej.