Projekt - Deep Q Learning (DQN)

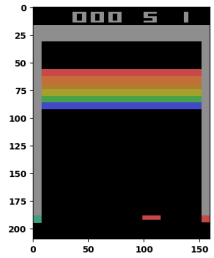
Piotr Matusiak

1 Wstęp

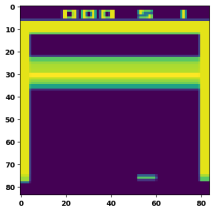
Celem projektu było zaimplementowanie metody Reinfocement Learningu - DQN oraz zastosowanie jej na grach Atari. W trakcie realizacji projektu dodatkowo dowiedziałem się o dwóch modyfikacjach tej metody, których implementacja była względnie prosta, gdy miałem już gotowy model DQN. Te dwie metody to Double DQN oraz Dueling DQN. Po zaimplementowaniu tych metod wytestowałem ich działanie na grach Atari.

2 Prerekwizyty

Przed samą implementacją modelu musiałem przygotować środowisko w którym agent DQN będzie mógł działać. W tym celu skorzystałem z modułu gymnasium (kiedyś gym). Mając już gotowe środowisko zwracające potrzebne mi dane (obserwacje, nagrody itp) musiałem zaimplementować preprocessing danych otrzymywanych ze środowiska. W tym celu skorzystałem również z narzędzi oferowanych przez gymnasium, a konkretnie wrapperów (częściowo wbudowanych, częściowo własnych), aby dostosować obserwacje ze środowiska do swoich potrzeb. Mianowicie przeskalowałem otrzymany obraz, przesłałem do sklali szarości, przeskalowałem wartości, aby uniknąć flickeringu wybierałem maksiumum 2 z 4 kolejnych klatek, oraz na końcu stackowałem klatki, tak aby na końcu otrzymać wejście o wymiarach $(4, 84, 84)$ z wartościami w przedziale $(0, 1)$.



(a) Bez wrapperów

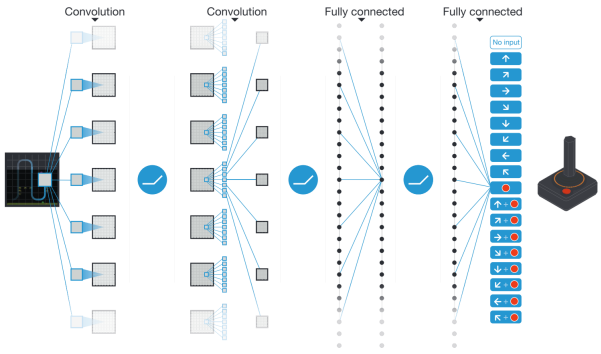


(b) Po nałożeniu wrapperów

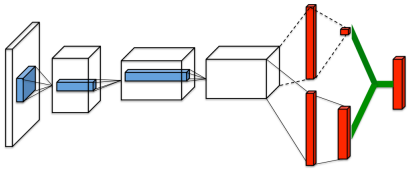
Kolejną rzeczą było zaimplementowanie experience buffera, czyli listy cyklicznej, pamiętającej ostatnie n (w moim przypadku 50000) kroków.

3 Implementacja

Mając już przerekwizyty mogłem przejść do implementacji. Do gotowego modelu potrzebowałem sieci, estymującej Q - funkcję, agenta, oraz pętli uczenia. Architektura sieci DQNa prezentuje się następująco:



Z tą różnicą, że na obrazku widoczne są jedynie dwie warstwy konwolucyjne, natomiast w zaimplementowanej sieci są 3. Tak więc mamy 3 warstwy konwolucyjne, z aktywacją ReLu, a następnie dwie warstwy liniowe fully connected, pierwszą również z aktywacją ReLu i drugą bez aktywacji. Z takiej sieci korzystają DQN i DDQN, natomiast DuelingDQN korzysta ze zmodyfikowanej sieci:



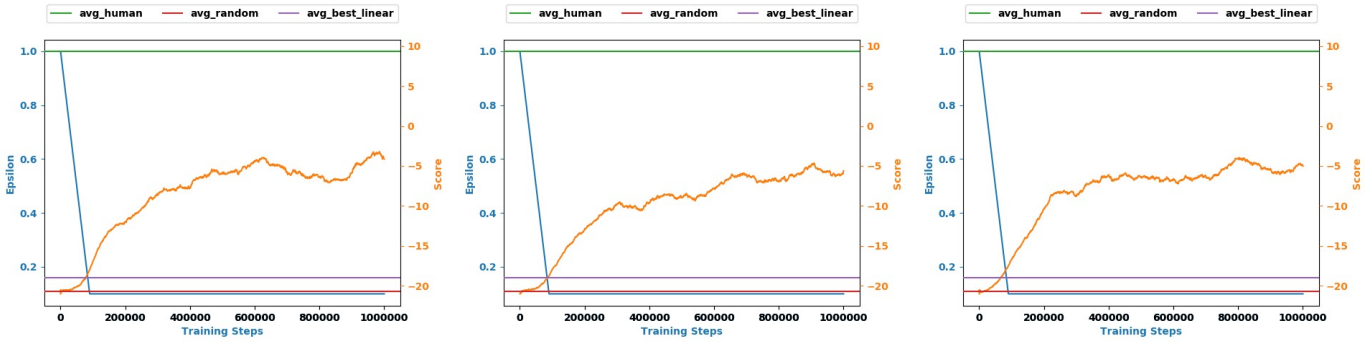
Gdzie po wyjściu z konwolucji rozdzielamy się na dwa strumienie do warstw liniowych, które następnie łaczymy. Algorytm DQNa prezentuje się następująco:

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

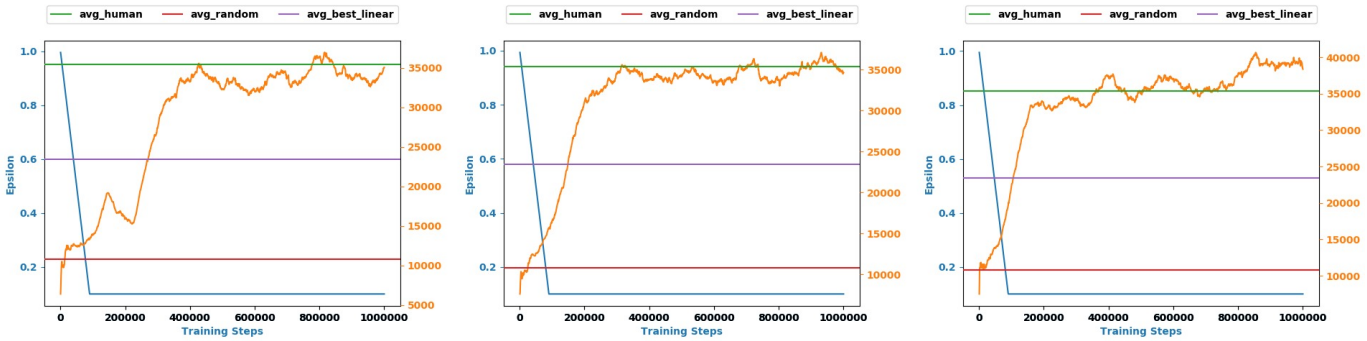
DDQN jest jego modyfikacją w której targety są wyliczane nie wprost z sieci targetów (tej aktualizowanej co określona liczba kroków), ale indeksy maksimów wyznaczane są z sieci online (tej aktualizowanej na bierząco), a następnie ich wartości pobierane z predykcji sieci targetów. Natomiast DueliungDQN, jak wspomniano wcześniej posiada zmodyfikowaną sieć.

4 Wyniki

Modele przetestowałem głównie na dwóch grach: Pongu, oraz Crazy Climberze, wyniki jakie otrzymałem widoczne są poniżej:

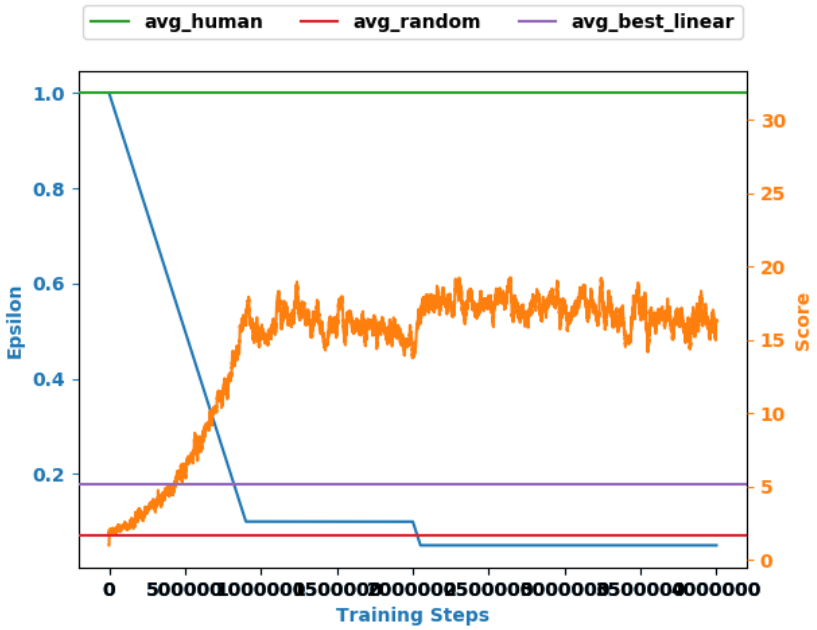


Rysunek 2: Wyniki w Pongu, od lewej odpowiednio DQN, DDQN, DuelingDQN



Rysunek 3: Wyniki w Crazy Climberze, od lewej odpowiednio DQN, DDQN, DuelingDQN

Pomarańczowa linia to średnia krocząca ostatnich 100 gier, linia niebieska, to epsilon czyli prawdopodobieństwo wykonania przez agenta losowego ruch, natomiast pozostałe trzy pionowe linie to średnie: losowa, ludzka (gry profesjonalnych testerów), oraz najlepszego w danym czasie liowego modelu rozwiązującego ten sam problem (dane na temat tych średnich pochodzą z tego papera). Dodatkowo wykonałem jeszcze jeden eksperyment, gdzie próbowałem uczyć DuelingDQNa grać w grę Breakout przez 4 miliony kroków (czyli 4 razy dłużej, niż w pozostałych przypadkach). Jednek wyniki nie były zbyt imponujące:



Moja hipoteza na temat tego, dlaczego wynik tego eksperymentu był niezbyt imponujący jest taka, że wielkość mojego experience buffera była znacznie mniejsza, niż w paperach na których się opierałem (50 tysięcy w moim przypadku 1 milion w przypadku paperów).

5 Konkluzje

Podsumowując całokształt projektu, jestem z niego zadowolony. Moje wyniki niestety nie dorastają do pięt tym z paperów, jednak dla mnie są satysfakcjonujące i jestem zadowolony, że udało mi się w przynajmniej jednym przypadku doprowadzić do tego, że agent stablinie radził sobie lepiej, niż ludzki gracz. Dodatkowo dzięki temu projektowi nauczyłem się sporo na temat podstaw reinforcement learningu.