Курсова работа

по "Обучение по метода поощрение/наказание"

на Гасан Мохамад Назер,

фак № 2MI3400038

дата: 17.06.2022

Тема: „Приложение ма метода Q обучение за обект Breakout“

1. Избран обект от Gym AI, променливи и оптимизационна задача

За проекта избрах Gym AI средата "BreakoutNoFrameskip-v4", която пресъздава играта “Breakout”. В нея слой от тухли заема горната част от екрана, а целта е те да бъдат унищожени. Това става като многократно топче се отблъсква от плоскост в долната част на екрана и разбива тухли в горната. Плоскостта се измества наляво и надясно спрямо позицията на падащото към долната част на екрана топче. При разбиване на тухла се получава печалба, която се инкрементира с всяка унищожена тухла. След разбиране на всички тухли се преминава на следващо ниво.

1. Изображение на играта “Breakout”. Печалба(точките до този момент-003) от ляво, в центъра оставащи животи (5) и отдясно текущо ниво (1).

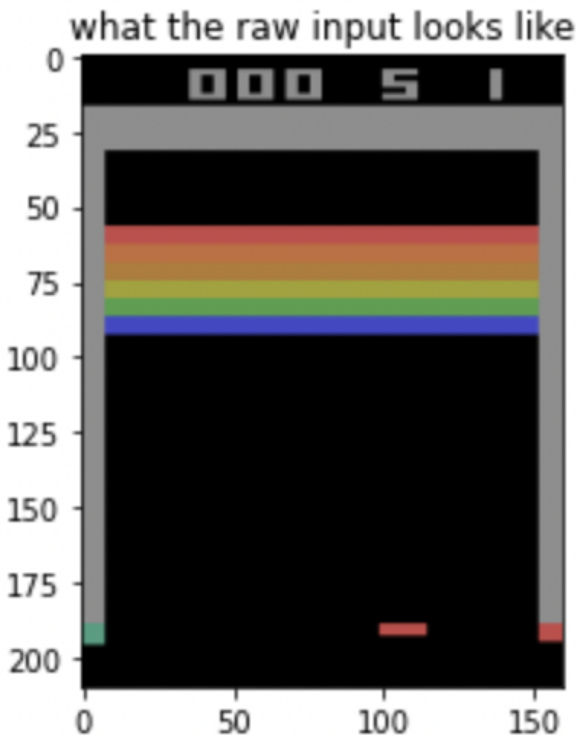
Играта започва с пет живота. Един живот приключва с падането на топчето извън долната част от екрана, след неуспешно отблъскване от плоскостта(плоскостта не е под топчето, когато то пада). Един епизод от играта представлява консумирането на петте живота.

В нея **възможните действия** са четири:

- NOOP - не се предприема действие
- FIRE - стреля се
- RIGHT - отмества се плоскостта надясно
- LEFT - отмества се плоскостта наляво

Състояния/Наблюдения (Observations):

По подразбиране средата връща RGB изображение, което се представя на играча като наблюдение/състояние. То има следните измерения: (210, 160, 3) - (височина, ширина, RGB канали).



2. Изображение на “observation” върнато от срещата.

Печалба:
Точките от разрушаване на туха се определят от цвета на тухлата:

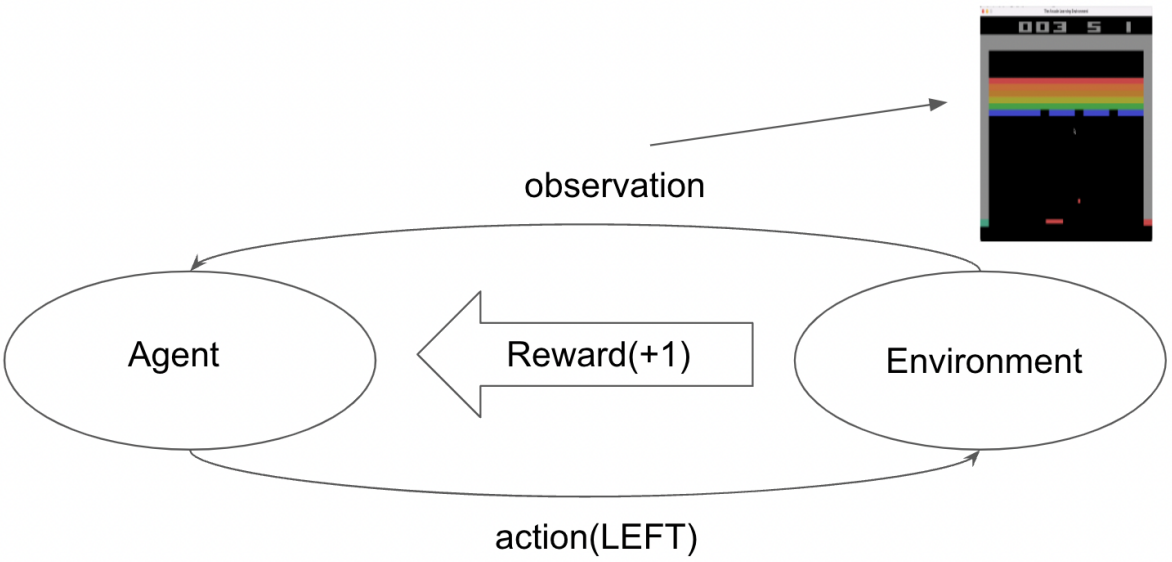
- синя тухла → 1 точка
- зелена тухла → 1 точка
- жълта тухла → 4 точка
- кафява тухла → 4 точка
- оранжева тухла → 7 точка
- червена тухла → 7 точка

Подобно на други игри целта в “Breakout” е получаването на максимално количество точки. Това е и оптимизационната задача- максимизиране на печалбата(По-надолу в текста е математическото фомулиране на оптимизационната задача). Оптималната стратегия е разбиване на тухли, образувайки тунел, през който да премине топчето, качвайки се над тухлите и разбивайки ги без необходимост от изместване на плоскостта.

2. Избран метод за обучение

Приложеният метод е Конволюционна невронна мрежа тренирана с вариант на Q обучение, където входа са пиксели, а изхода е функция, прогнозираща бъдещите печалби. Цялостната идея е базирана на [Playing Atari with Deep Reinforcement Learning].

Обучение се базира на интеракциите между Агент и Среда. В конкретния случай Средата ще връща състояние/наблюдение (observation), което ще представлява едно изображение от играта. Агента от своя страна трябва да върне действие, отговарящо на наличните от играта- пример: LEFT- премести плоскостта наляво. След това Средата трябва да върне печалба(+1/+4/+7 за разбита тухла, 0 в противен случай).



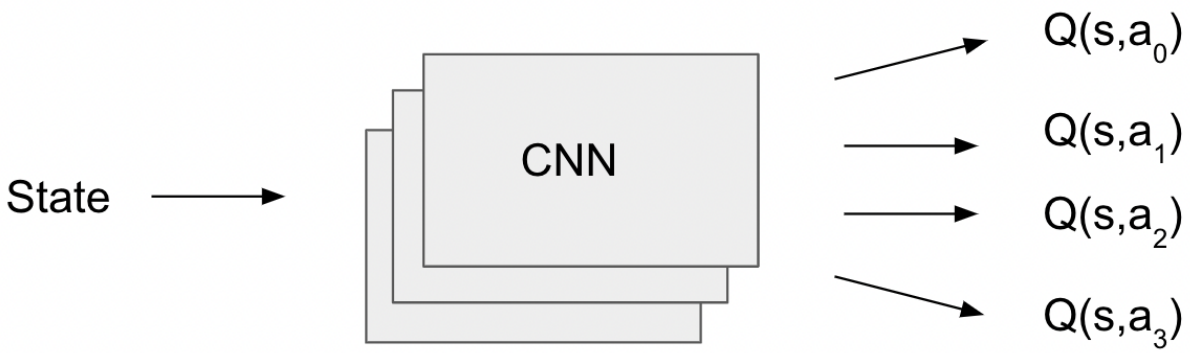
Изображение на взаимодействието между Агента и Средата.

Целта на Агентът е да взаимодейсва със Средата, избирайки действия по такъв начин, че да се максимизира бъдещата печалба. Тя се представя математически със следната формула:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

,която изразява, че печалбата е равна на сумата от печалбите за всички стъпки, намалени спрямо фактора γ. Тази печалба се открива благодарение на Q функция в Q обучението.
В него се тренира Q функцията (Q(s, a) → R). Тя представлява функция, приемаща състояние (s), действие (a) и връща каква ще бъде бъдещата печалба (R), ако в конкретното състояние се предприеме съответното действие.

Както беше споменато по- горе използвах Конволюционна невронна мрежа, която служи за споделен енкодър между състоянията и Q функцията за всяко действие. Чрез Q функцията се изразява, колко стойностно е съответното действие в конкретното състояние.
Този вид невронна мрежа е известна с името Deep Q-Networks (DQN).



Изображение на Deep Q-Networks. За вход се приема конкретно състояние, а изоходът е Q функция за всяко действие.

Използваният метод е вид онлайн обучение, защото докато се играе играта, се събират данни (състояния, действия, печалби), образуващи трениращо множество, и в същото време се тренира и невронната мрежа.

Формално:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a \right]$$

Изразява как за Q функцията, винаги ще се взима действието, което максимизира печалбата. Преминаването от едно състояние в друго, ще се базира на действието, което ще донесе най- голяма бъдеща стойност (печалба).

За оптимизиране се ползва следната функцият (loss function):

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[\left(y_i - Q(s,a;\theta_i) \right)^2 \right]$$

, където y_i е:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \right]$$

С нея се изразява разликата между печалбата получена от Q функцията за бъдещото състояние, при конкретно действие (тази печалба се взима от исторически данни θ_{i-1}) и печалбата от Q фунцкията за сегашното състояние, при конкретно действие.

Градиента на тази функция е:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Използван алгоритъм:

Algorithm 1 Deep Q-learning with Experience Replay

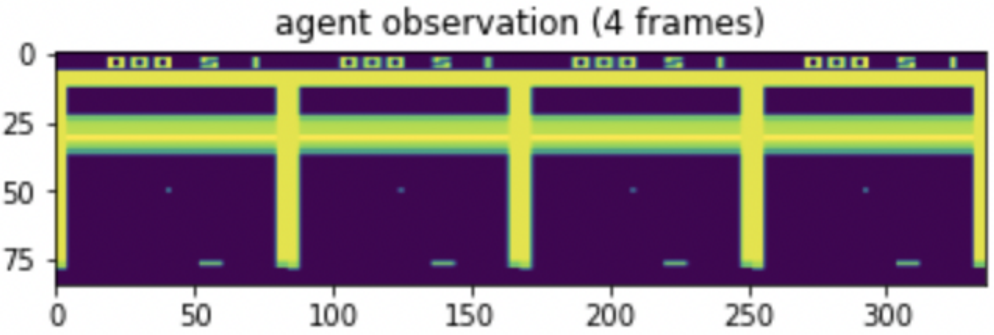
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

За алгоритъмът е изключително важна ролята на буфера за повторно възпроизвеждане (replay buffer, произлизаш от техниката experience replay). Той се използва за да съхрани четворки от вида (състояние, действие, печалба, бъдещо състояние). Това позволява много подобни фреймове да бъдат запазени, но в следствие семплирани произволно. Произволното семплиране премахва възможността от засядане в локални минимуми или драстични разминавания в параметрите. Също така използването на тази техника, усреднява дистрибуцията на предишни състояния, изглаждайки обучението и премахвайки колебания или отклонения в параметрите.
Алгоритъмът селектира произволно действие с вероятност $\in$, прилагайки greedy стратегия. Цели се равновесие между експлоатация или изследване(exploitation vs exploration). В противен случай се взима действието, което максимизира Q

функцията. Подава се новото действие на средата и се получава ново състояние. Записва се четворката в буфера. От него се семплира четворка и се прави градиентно спускане спрямо производната на целевата функция (loss function). Това се повтаря за T брой времеви стъпки и M брой епизода.

Архитектура 1:

В оригиналния “paper” се използва двуслойна CNN, приемаща изображение във формата 84 × 84 × 4 (височина, ширина, 4 фрейма, изразяващи “sticky actions”/”frame stacking”- по- лесно може да се определи посоката на топчето, ако има повече от 1 фрейм за действие).



Изображение на входния формат на CNN

Двата конволюционни слоя имат следните характеристики:

- 16 филтъра 8 x 8 със страйд 4 и релу за активационна функция
- 32 филтъра 4 x 4 със страйд 2 и релу за активационна функция

Последния скрит слой е свързан(fully connected) и има 256 нода с релу за активационна функция. Последния слой е също свързан(fully connected) с линейна активационна функция и един изход за всяко валидно действие (четири за играта “Breakout”).

Архитектура 2:

Допълнително изпробвах втори вид архитектура, отново CNN със следните характеристики:

Входеният слой е същия- 84 × 84 × 4
Конволюционните слоя са три:

- 32 филтъра 8 x 8 със страйд 4 и релу за активационна функция
- 64 филтъра 4 x 4 със страйд 2 и релу за активационна функция
- 64 филтъра 3 x 3 със страйд 1 и релу за активационна функция

Последния скрит слой е свързан(fully connected) и има 512 нода с релу за активационна функция.
Последният слой е непроменен спрямо оригиналната архитектура.

3. Симулационни резултати

Програмен код: https://github.com/GasanNazer/ReinforcementLearning

- CNN(modified architecture).ipynb → модифицирана архитектура
- CNN(original architecture).ipynb → оригинална архитектура
- Load model with saved weights.ipynb → за зареждане на вече тренирани модели

За оригиналната архитектура, поради малко итерации(епизоди) на трениране резултатите са ненадежни за сравняване с модифицирана версия.

В модифицираната версия тренирах две версии:

- с 20000 епизода
- с 25000 епизода

Всички други използвани параметри са идентични и затова не са споменати. Може да бъдат видяни в нотбуковете.

Изображението по-долу е от версията с 20000 епизода. На него се вижда как успешно Агента разбива тухлите, постигайки резултат от 90 точки. Също така той е успял да научи оптималната стратегия- да изгради тунел през тухлите. Въпреки тунела, Агентът не успява да прекара топчето през него и да натрупа допълнително точки, без необходимост от предвижване на плоскостта.

Долното изображение е от версията с 25000 епизода. При него Агентът бележи 82 точки. По- малко от тези при Агентът трениран на 20000 епизода. Въпреки това, ясно се вижда, че в този случай Агентът научава оптималната стратегия и

се опитва да я приложи повече от веднъж. Подобно на предходния, тук също не успява да прекара топчето в тунела.

Демо:

- с 20000 епизода: https://youtu.be/0JMP0nwACQU
- с 25000 епизода: https://youtu.be/Fl8g_B8M3qE

Извод:
Q обучение и в частност DQN успешно може да се приложи за научаването на Агент да играе атари игри и по- конкретно Breakout. При тренировка с по-комплексна архитектура и повече епизоди, Агентът успешно научава различни стратегии, дори такава считана за най- оптимална.

4. Използвана литература

   a. Playing Atari with Deep Reinforcement Learning
   b. [Classic] Playing Atari with Deep Reinforcement Learning (Paper Explained)
   c. https://github.com/drVlasov/Breakout/blob/main/breakout.ipynb
   d. Playing Atari Breakout with Deep Reinforcement Learning