Обучение ИИ с помощью Q-Learning в среде Assault

Антон Никифоров 11 В

Апрель 2024

1. Abstract

В данной работе представлено решение игры Assault (Atari) с помощью глубокого Q-обучения (DQN), сравнивается эффективность реализованных методов для решения среды.

2. Введение

Assault – это аркадная видеоигра, разработанная компанией Atari. В игре игрок управляет космическим кораблем, цель которого – защититься от волн нападающих инопланетных кораблей.

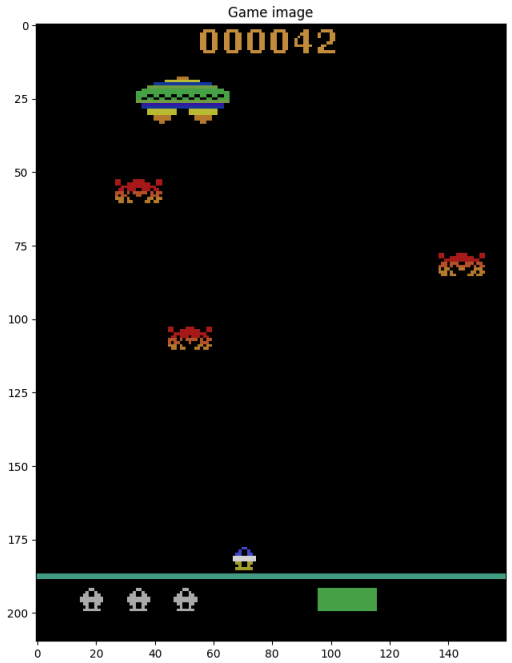


Figure 1: Пример кадра из игры Assault

В данной работе представлено решение среды с помощью Q-Learning (greedy-espilon action selecting) с использованием сверточных нейронных сетей. В качестве входных данных для агента используются только кадры из среды. Реализованы алгоритмы DQN и Dueling DQN. Для ускорения обучения используются различные реплеи: Prioritized Replay Buffer и Experience Replay Buffer.

В работе в качестве среды использовалась версия “ALE/Assault-v5” из библиотеки gym.

3. DQN и DDQN

Агент использует epsilon-greedy сратегию. То есть с шансом ε $(0 \leq \varepsilon \leq 1)$ агент делает случайное действие, а с шансом $1 - \varepsilon$ делает то действие, которое сейчас оптимально, то есть то, за которое он в конце игры получит наибольшую награду. Случайные действия совершаются для “исследования” среды.

3.1. DQN

Пусть $Q(s, a)$ - это прогнозируемая награда, которую получит агент в конце игры, если из состояния s сделает действие a. Тогда для выбора оптимального дейсвтия, давайте выбирать то, у которого $Q(s, a)$ максимальна. Как считать Q(s, a)?

По уравнению Беллмана:

$$Q(s, a) = r + \gamma \cdot Q_{\text{expected}}$$
$$Q_{\text{expected}} = (1 - \text{done}) \cdot \max_{a'} Q(s', a')$$

Давайте функцию $Q(s, a)$ считать с помощью нейронных сетей policy network (с помощью сверточных нейронных сетей и полносвязных нейронных сетей).

Тогда чтобы обучать нейронную сеть, будем называть ошибкой: $\text{Loss} = \left(Q(s, a) - Q_{\text{expected}}\right)^2$

3.2. Target nework

Однако у нас есть проблема: при использовании глубоких нейронных сетей наш агент будет “забывать” то, что он делал раньше: сначала награда будет расти, но в какой-то момент, агент как будто разучится играть и награда сильно упадет, и так по кругу. И так он ничему не научится.

Чтобы этого избежать давайте ведем еще одну нейронную сеть “target network” и будем обновлять target network каждые Δ итераций (копируем параметры policy network).

Тогда теперь Q_{expected} будет выражаться так:

$$Q_{\text{expected}} = (1 - \text{done}) \cdot \max_{a'} Q_{\text{target}(s', a')}$$

И на практике это решает проблему того, что агент забывает чему он научился, так как мы меняем сеть периодически и постепенно приближаемся к оптимальному предсказаню Q функции.

3.3. Dueling DQN

Dueling DQN[2] - это тип сети, в которой есть два потока для раздельной оценки (скалярной) значения состояния (Value) и преимущества для каждого действия (Avantage).

Обычно оба потока используют общий модуль изучения сверточных функций. Эти два потока объединяются с помощью специального агрегирующего слоя для получения оценки функции Q “состояние-действие”, как показано на рисунке справа.

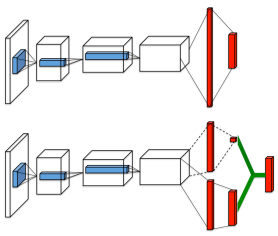


Figure 1. A popular single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values for each action.

Figure 2: Пример dueling сети

Теперь Q функция пересчитывается следующим образом:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|}\sum A(s,a')\right)$$

4. Реплеи

4.1. Experience Replay

Experience Replay - это метод запоминания опыта, используемый в обучении с подкреплением, при котором мы сохраняем опыт агента на каждом временном шаге $e_i = (\text{state}, \text{action}, \text{reward}, \text{next\_state})$ в виде набора данных $D = e_1, e_2, ..., e_N$.

Для обучения модели мы проводим выборку памяти из D случайным образом для получения $\text{min-batch} = e_{\text{random\_1}}, e_{\text{random\_2}}, ..., e_{\text{random\_K}}$ и уже на этих данных обучаем агента.

Это решает проблему автокорреляции, приводящую к нестабильному обучению.

4.2. Prioritized Experience Replay

Prioritized Experience Replay[3] - это тип воспроизведения опыта в обучении с подкреплением, при котором мы чаще воспроизводим переходы с высоким ожидаемым прогрессом в обучении, измеряемым величиной ошибки их временной разницы (TD Error).

Давайте определим приоритет i-го элемента реплея как:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

$$p_i = \left|Q_{\text{value}} - Q_{\text{expected}}\right| + \varepsilon$$

Однако стохастическая расстановка с приоритетом приводит к смещению данных, поскольку оно изменяет распределение неконтролируемым образом и, следовательно, изменяет решение, к которому будут сходиться модели. Мы можем исправить это смещение, используя веса выборки по важности (Importance Sampling):

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$$

5. Preprocessing Environment

Так как изначально среда представляет из себя rgb картинку формата 210x160, то агенту может быть трудно обучаться на ней. Поэтому для ускорения обучения сожмем картинку до размера 64x64 и переведем ее из rgb формата в черное-белое изображение.

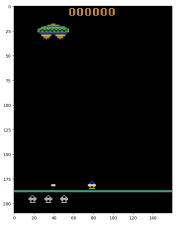


Figure 3: Исходное изображение

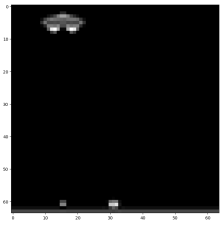
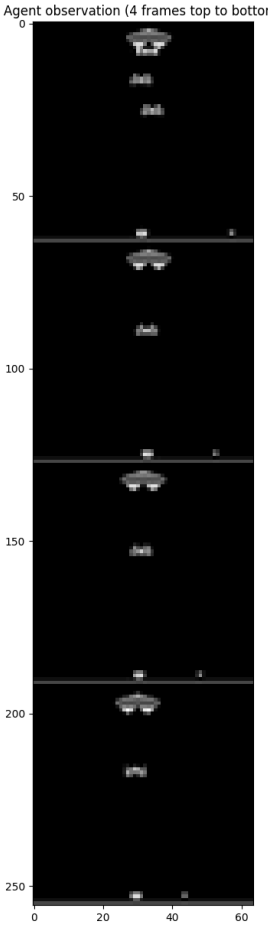


Figure 4: Преобразованное изображение

Так как по одному кадру непонятен характер движения объектов, то в качестве входных данных для агента используются 4 последних кадра среды. То есть формат входных данных для агента имеет вид $(4, 64, 64)$.



Пример входных данных агента

6. Обучение

Изначально обучение было реализовано так: агент играл x эпизодов, а затем обучался на батче размером s.

Однако оказалось, что лучше обучении агента реализовать “по кадрам”, то есть каждые 4 итерации агент обучался на минибатче размером 16.

Обучение агента производилось на 3M кадров.

Все награды были отнормированы от -1 до 1 для ускорения обучения.

Так как использовалась greedy-epsilon стратегия для выбора действия, то в процессе обучения ε равномерно уменьшался: $1 \to 0.1$ на протяжении первых 500K кадров, в дальнейшем оставался равным 0.1.

Каждые $\Delta = 5000$ кадров оценивалась средняя награда агента за 3 игры и обновлялась target network.

Так как в среде Assault есть 4 жизни, то для стабилизации обучения эпизод считался завершенным, когда агент терял одну жизнь, но при этом в самой среде игра продолжалась до потери всех жизней (то есть за одну игру было 4 эпизода).

В случае Prioritized Experience Replay: были выбраны $\alpha = 0.6$, а β равномерно увеливался: $0.4 \to 1$ на протяжении первых 750K кадров, а затем оставался равным 1.

7. Эксперименты

7.1. Frame skip

Изначально агент обучался в среде, где frame_skip был равен 1 (то есть действия агента не повторялись). Однако результат был чуть лучше, чем если бы агент выбирал каждый раз случайные действия.

Поэтому стало интересно посмотреть, как агент обучается, когда frame_skip равен 4 (то есть все действия повторяются 4 раза). Ниже приведены графики при разных frame_skip. Во всех моделях использовался стандартный DQN.

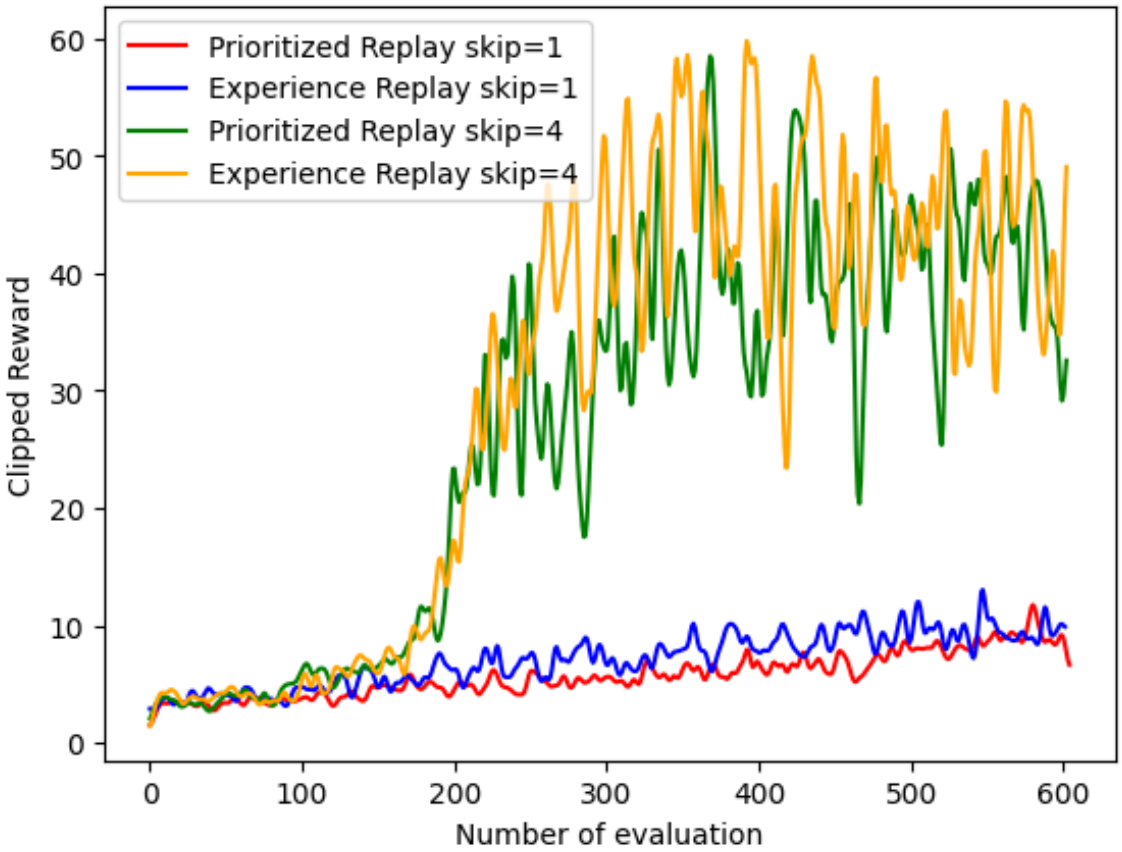


Figure 5: График средней награды во время обучения с разными frame_skip

Как видно на графике при frame_skip = 4, модель набирает больше, чем без, поэтому везде в дальнейшем использовался frame_skip = 4.

7.2. Dueling DQN vs DQN

Дальше были обучены Dueling DQN с Experience Replay Buffer и Prioritized Replay Buffer.

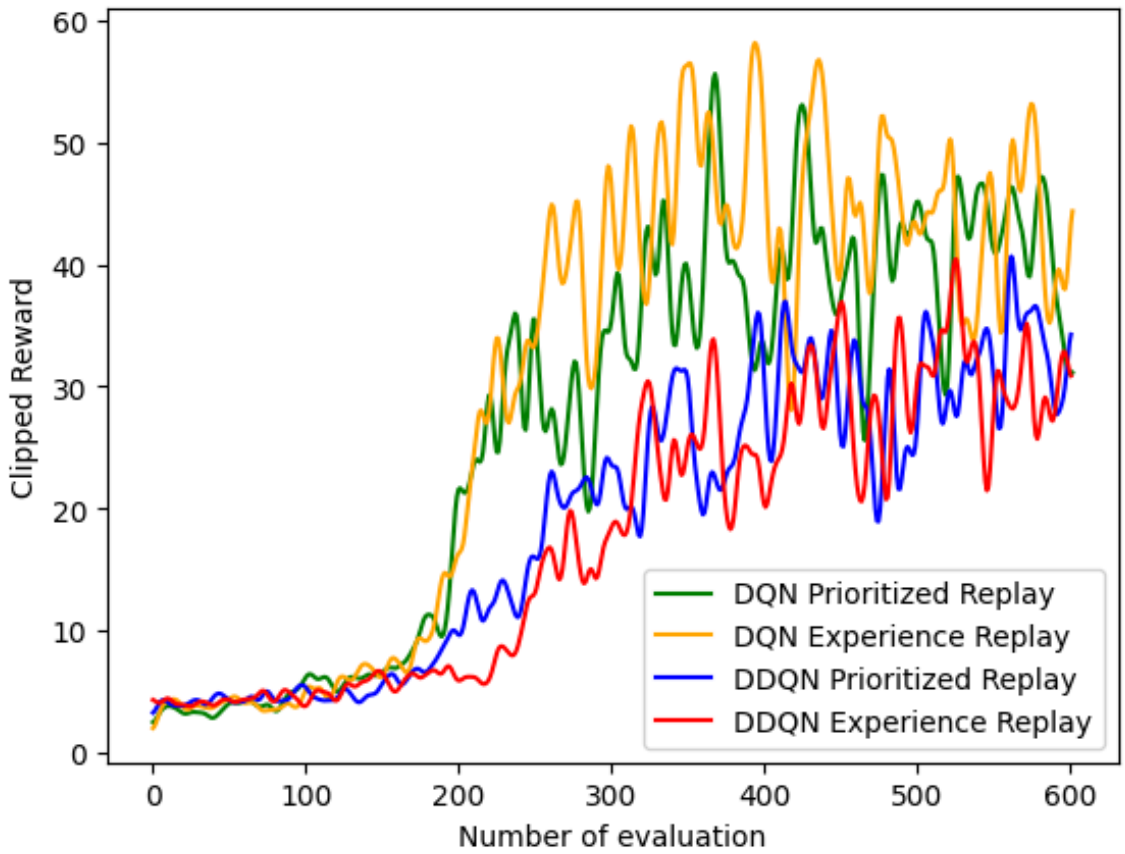


Figure 6: График средней награды во время обучения

Сравенение моделей на реальной игре со всеми жизнями

Модель	**Средняя награда за 10 игр**	**Максимальная награда за 10 игр**
Random	233.1	399.0
Human	1784.3	3223.0
DQN with Experience Replay	5680.1	6723.0
DQN with Prioritized Replay	5226.3	8276.0
Dueling DQN with Experience Replay	3965.0	7187.0
Dueling DQN wiht Prioritized Replay	3559.8	5436.0

8. Выводы

Модели DQN превосходят человека примерно в 1.7 раз, а Dueling DQN в 1.2 раза, и работают гораздо лучше, чем если бы агент совершал случайные действия, поэтому можно считать, что модели решили среду.

Как ни странно, но лучшую среднюю награду достигает обычный target DQN с Experience Replay, однако наилучший максимальный счет достигает DQN с Priortized Replay.

Также модели с Prioritized Experience Replay набирают в среднем меньшую награду, чем модели с обычным Experience Replay, однако максимальный счет набирает именно модель с Prioritized Replay.

Dueling DQN не дало особого ощутимого улучшения, однако возможно модель не дообучилась из-за недостаточно числа кадров, так как на графике средней награды видна тенденция роста средней награды у Dueling.

8.1. Интересное наблюдение

Если посмотреть на видео [6] визуализации игры агента, то можно заметить, что агент выбрал тактику запираться в левый угол и отстреливаться из него по врагам, иногда выезжая на середину поля.

Также интересно, что агент научился понимать когда в него летит горизонтальный снаряд и с какой стороны.

9. Заключение

В данной работе была решена среда Assault с помощью DQN и Dueling DQN (with Experience and Prioritized Replay).

Все модели достигли неплохого счета и превзошли человека в игре Assault, однако обычная DQN справилась с этим лучше, чем Dueling DQN.

Если брать максимальный счет, как оценку модели, то на PapersWithCode [4], модель DQN with Prioritized Replay занимает 21 место, что является вполне неплохим результатом.

Одной из возможностью улучшения является увелечение количества кадров (например до 200M), на которых обучаются агенты.

10. Appendix

Исходный код проекта доступен на Github [5]. Также есть видео [6] визуализации игры агента.

Небольшое количество кадров для обучения обусловлено отсутствием достаточной вычислительной мощности. Все модели обучались на одном компьютере с поддержкой cuda. Из-за этого обучение одной модели на 3M кадров занимало около одного дня.

Во всех моделях использовался $\text{learning\_rate} = 2.5 \cdot 10^{-4}$ и $\text{buffer\_size} = 2 \cdot 10^4$ для реплея.

Архитектура нейронной сети:

- input 4x64x64
- Convolutinal layer (out_channels = 32, kernel_size = 8, stride = 4)
- Convolutinal layer (out_channels = 64, kernel_size = 4, stride = 2)
- Convolutinal layer (out_channels = 32, kernel_size = 3, stride = 1)
- Linear Layer (input = 1024, output = 512)
- Linear Layer (input = 512, output = 512)
- Linear Layer (input = 512, output = n_actions)

К каждому слою нейронной сети (кроме последнего применяется функция ReLU).

В Dueling DQN Value и Advantage имели одинаковые сверточные слои, а затем разделялись на две сети полносвязных слоев с архитектурой как указано выше.

11. Источники

[1] Playing Atari with Deep Reinforcement Learning // 2013
URL: https://arxiv.org/abs/1312.5602
[2] Dueling Network Architectures for Deep Reinforcement Learning // 2015
URL: https://arxiv.org/abs/1511.06581
[3] Prioritized Experience Replay // 2015

URL: https://arxiv.org/abs/1511.05952
[4] PapersWithCode: Atari Games on Atari 2600 Assault
URL: https://paperswithcode.com/sota/atari-games-on-atari-2600-assault
[5] Исходный код проекта на Github
URL: https://github.com/qualdoom/rl-assault
[6] Визуализация игры агента URL: https://youtu.be/yzIsHN5WWdM