Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, системы управления и искусственный интеллект

КАФЕДРА Системы обработки информации и управления

Лабораторная работа №6

По курсу

«**Методы машинного обучения в АСОИУ**»

«Обучение на основе глубоких Q-сетей»
ALE/AirRaid-v5

Выполнил:
Студент группы ИУ5-22М
Кириллов Д.С.
06.05.2024

Проверил:
Гапанюк Ю.Е.

Москва 2024 г.

Цель лабораторной работы

Ознакомление с базовыми методами обучения с подкреплением на основе глубоких Q-сетей.

Задание

На основе рассмотренных на лекции примеров реализуйте алгоритм DQN.

В качестве среды можно использовать классические среды (в этом случае используется полносвязная архитектура нейронной сети).

В качестве среды можно использовать игры Atari (в этом случае используется сверточная архитектура нейронной сети).

Ход работы

Реализуем вариант со сверточной НС.

Для данной ЛР подходит виртуальное окружении из предыдущих лабораторных 4 и 5, но необходимо установить еще пакеты.

Параметры обучения задаются в файле constants.py.

В коде программы для лабораторной сделали обработку аргументов.

Запуск обучения:

```
python main.py -m t
```

Запуск проигрывания среды (кол-во проигрываний – переменная PLAY_ROUNDS):

```
python main.py -m p
```

Сохранение результатов в pdf-файл по пути SAVES_PATH:

```
python main.py -s
```

1. Описание среды ALE/AirRaid-v5

Будем работать со средой ALE/AirRaid-v5 (из окружения игр Atari), для которой будем использовать сверточную архитектуру сети.

В данной среде в качестве агента выступает космический корабль, который может двигаться боком. Кораблю необходимо защитить два здания (которые постоянно двигаются) от вражеских кораблей, которые пытаются сбросить на них бомбы.

Агент имеет следующие возможные действия (пространство действий action_space = 6):

- 0: NOOP (no operations, т.е. ничего не делать);
- 1: FIRE (стрельба из центра корабля);
- 2: RIGHT (движение вправо);
- 3: LEFT (движение влево);
- 4: RIGHTFIRE (стрельба с правого борта корабля);
- 5: LEFTFIRE (стрельба с левого борта корабля).

Пространство состояний следующее: Box(0, 255, (250, 160, 3), uint8). Наблюдаемая высота области игры = 255.

Обученную модель можно запустить в среде, чтобы визуально оценить, как хорошо она обучилась. Среда выглядит следующим образом (рис. 1):

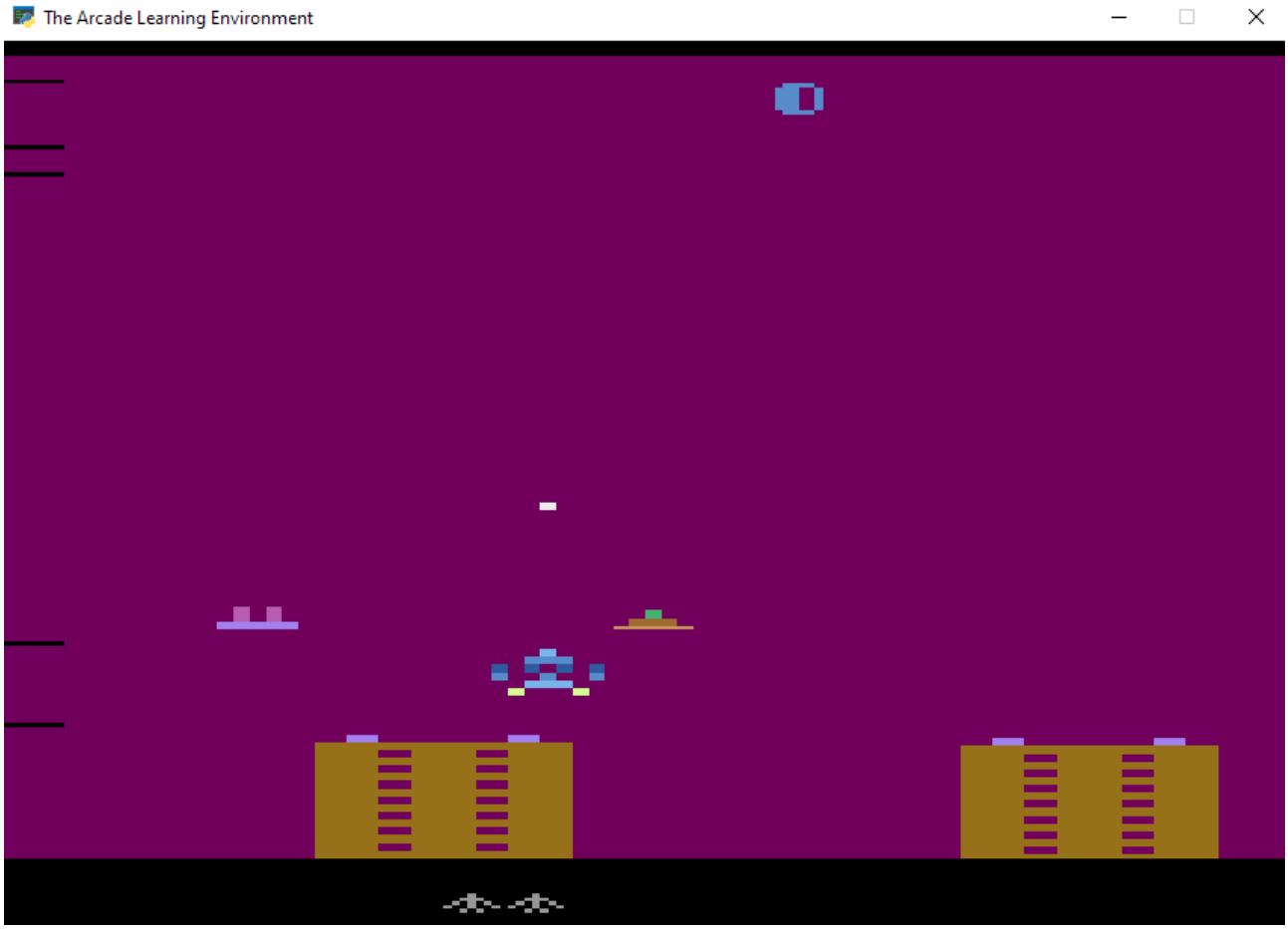

Рис.1. Окно с демонстраций работы обученного агента в среде Gym[Atari] ALE/AirRaid-v5.

2. Описание алгоритма Deep Q Network

Deep Q Network (DQN) — один из наиболее известных алгоритмов глубокого обучения с подкреплением (Deep RL). Является разновидностью Q-обучения. Основная идея состоит в том, чтобы построить приближении Q-функции с некоторым параметром θ в форме Q (s, a; θ) ≈ Q* (s, a). Используется нейронная сеть, чтобы аппроксимировать значение Q для всех возможных действий в каждом состоянии, ее принято называть Q-сетью.

Также для данного алгоритма реализовали технику Replay Memory. Это метод запоминания повторений, используемый в обучении с подкреплением, когда сохраняется опыт агента на каждом шаге в наборе данных, объединенных по многим эпизодам в память воспроизведения.

Данные из памяти воспроизведения выбираются случайно и используются для обучения в методах на основе value-based подхода (который подразумевает,

что алгоритм ищет не саму стратегию, а оптимальную Q-функцию). Такой подход решает проблему автокорреляции, приводящую к нестабильному обучению, делая проблему более похожей на обучение с учителем.

Шаги алгоритма:

1. Найти политику, которая оптимизирует награду:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s,a)$$

2. Поскольку Q∗ неизвестна, то нужно аппроксимировать ее нейронной сетью. Q-функция задается уравнением Беллмана:

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s', \pi(s'))$$

3. Минимизировать ошибку временных различий:

$$\delta = Q(s,a) - (r + \gamma \max_a Q(s',a))$$

В качестве функции потерь будем рассматривать функцию потерь Хьюбера:

$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} \mathcal{L}(\delta)$$

$$\text{where} \quad \mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \le 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

3. Программная часть

Произведем обучение с помощью алгоритмов SARSA, Q-обучения и двойного Q-обучения.

Для обучения модели будем использовать следующую архитектуру сверточной нейронной сети. Конфигурация НС задается в файле main.py:

```
airraid_dqn_layer_configs = (
	LayerConfig(Conv2D, dict(
		kernel_size=(8, 8), filters=16, strides=4,
		activation="relu",
		name="conv1"
	)),
	LayerConfig(Conv2D, dict(
```

```
        kernel_size=(4, 4), filters=32, strides=1,
        activation="relu",
        name="conv2"
        )),
        LayerConfig(Flatten, {}),
        LayerConfig(Dense, dict(units=256, activation="relu", name="fc1")),
        LayerConfig(Dense, dict(units=gym.make(CONST_ENV_NAME).action_space.n,
        activation=None, name="q_layer"))
)
```

Данная сеть состоит из 5ти слоев: 2х сверточных с функцией активации relu, преобразующего слоя Flatten и 2х полносвязных слоев.

Первое обучение произведем со следующими параметрами. Параметры обучения задаются в файле constants.py:

```
CONST_ENV_NAME = 'ALE/AirRaid-v5'
SAVES_PATH = "./model_saves"
NUM_TIME_STEPS = 100000
PLAY_ROUNDS = 5
REPLAY_CAPACITY = 1024
BATCH_SIZE = 32
TIME_STEPS_PER_TRAIN = 1000
TRAIN_STEPS_PER_Q_SYNC = 10
TIME_STEPS_PER_SAVE = TRAIN_STEPS_PER_Q_SYNC * TIME_STEPS_PER_TRAIN * 3
MAX_NUM_TIME_STEPS = 500000
WARM_START = BATCH_SIZE * 2
REWARD_GAMMA = 0.9
INITIAL_EPSILON = 0.5
FINAL_EPSILON = 0.01
```

Оценкой качества модели является episode length: чем дольше длина эпизода, тем лучше, т.к. самолётику надо как можно дольше продержаться живым.

При просмотре результата стало ясно, что агент обучился плохо, т.к. он пытался лишь спастись, практически не уничтожая вражеские корабли, и постоянно находился в левой части карты. По результатам на рис.1. видим, что ошибку уменьшить не удалось:

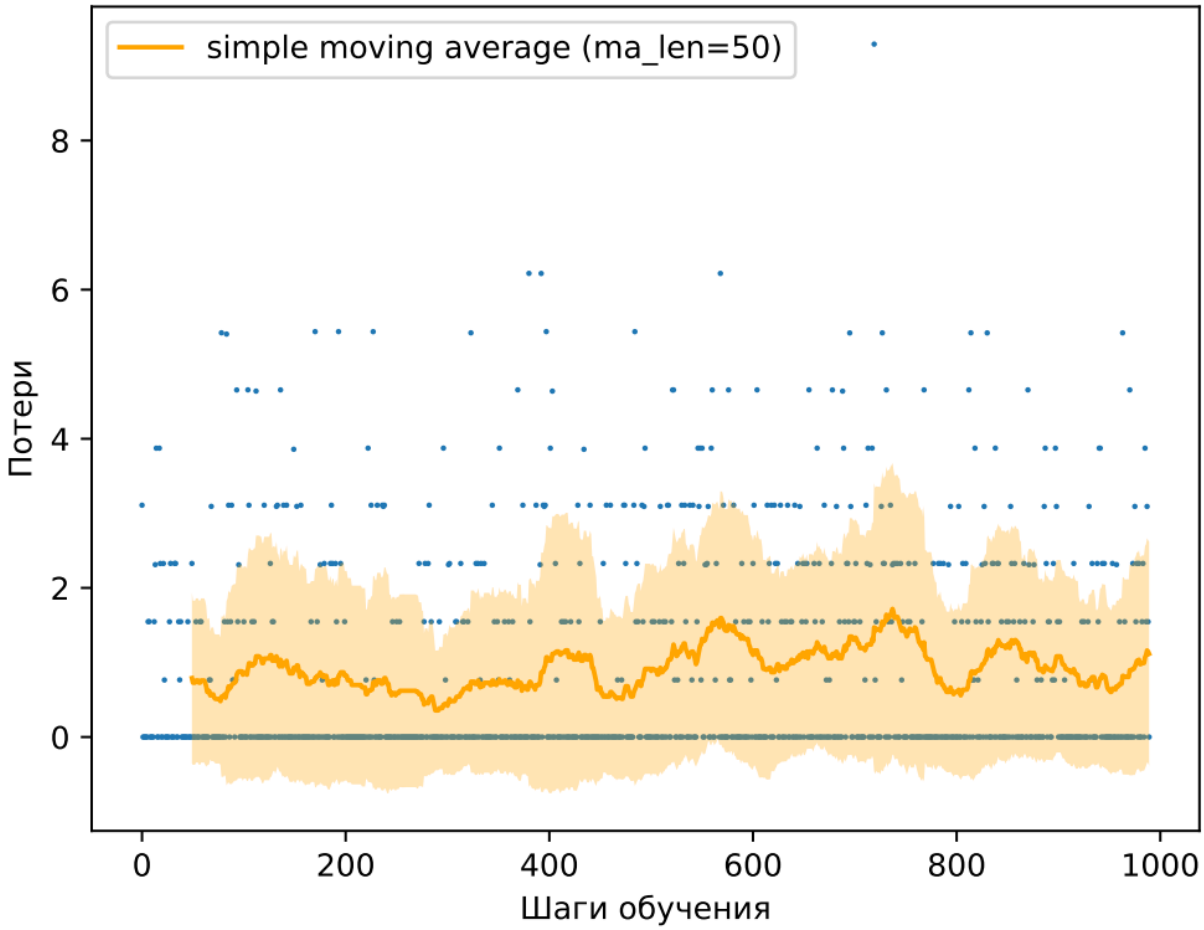

Рис. 2. Обучение первой модели.

Вторая модель.

Поменяли размер батча с 32 на 64. Здесь агент сработал более-менее хорошо, однако уничтожал корабли только справа, но, при этом, достаточно успешно от них уворачиваясь. На этот раз на графике немного отслеживается уменьшение ошибки:

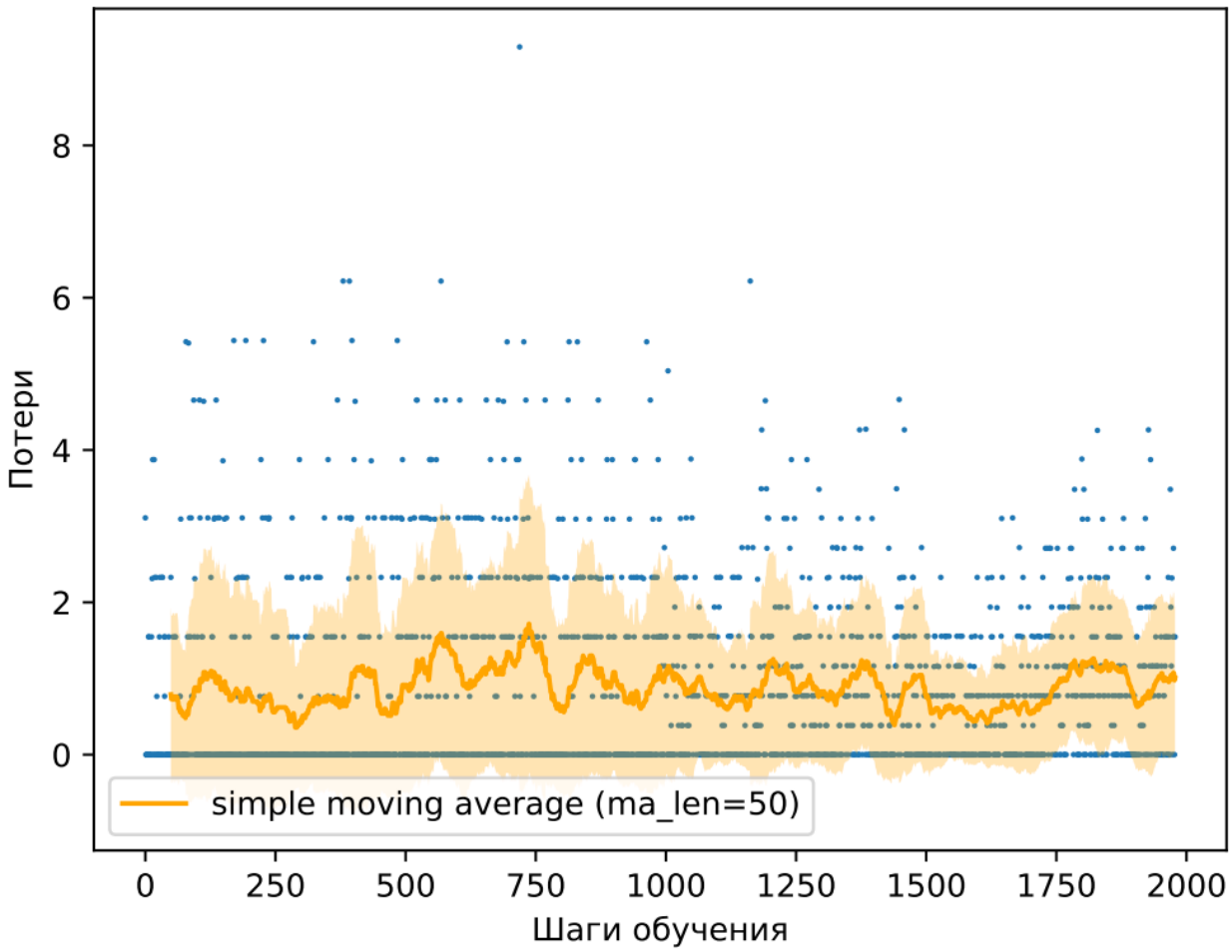

Рис. 3. Обучение с размером батча = 64.

Третья модель.

Поменяли размер батча на 16. Для данного обучения агент работал плохо и постоянно стоял на одном месте в центре. По графику видим, что ошибка только увеличилась:

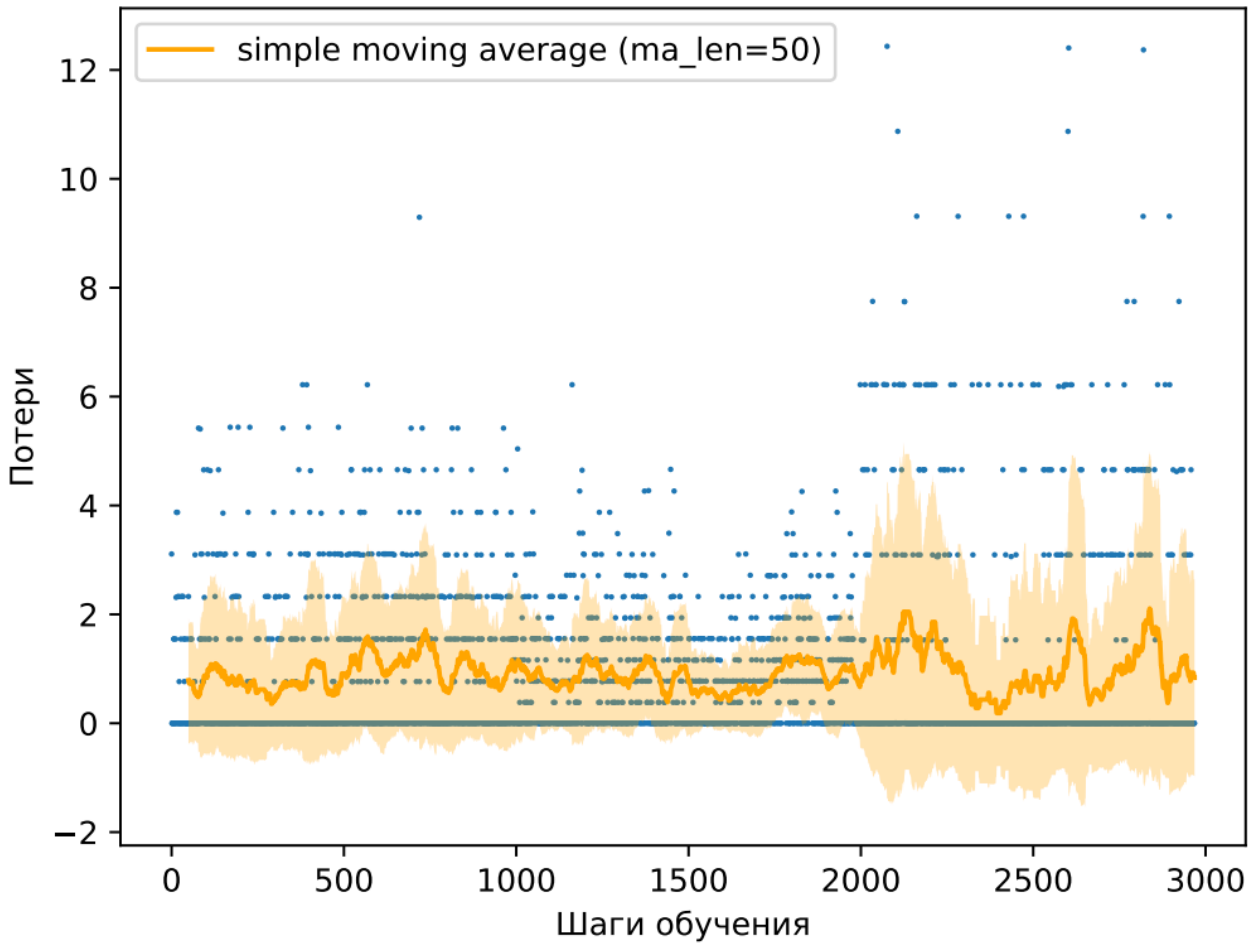

Рис. 4. Обучение с размером батча = 16.

Четвертая модель.

Поменяли TIME_STEPS_PER_TRAIN на 1000 (было 100), batch_size установили, как 32. На этот раз агент уже как-то пытался уничтожать космические корабли, однако постоянно находился слева и часто умирал. На этот раз график стал сильно обрезанным:

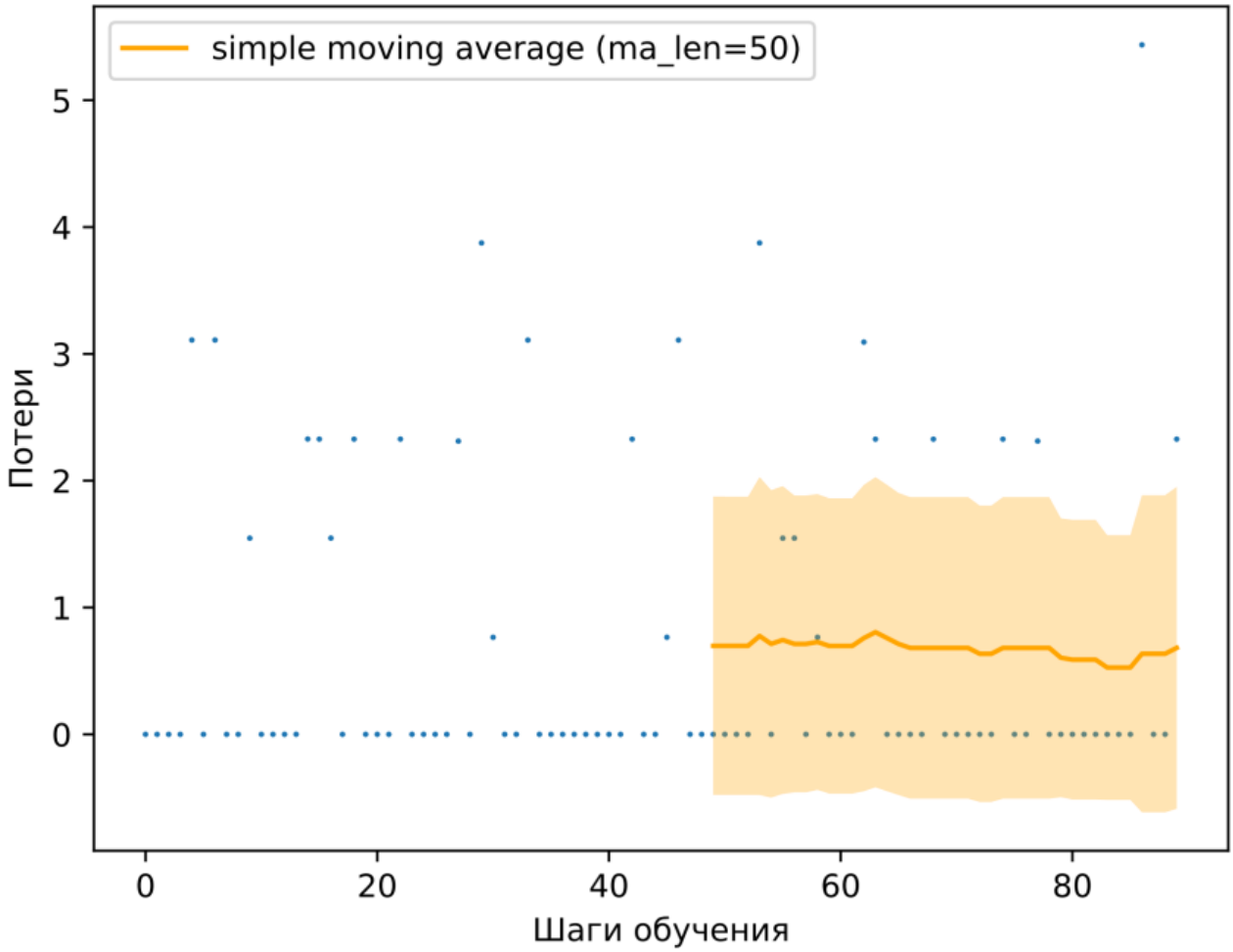

Рис. 5. Обучение с размером батча = 32, time_steps_per_train = 1000.

Вернули значение TIME_STEPS_PER_TRAIN (равное 100), установили батча как 64 (т.к. это был наилучший размер батча) и попробовали изменить REPLAY_CAPACITY с 1024 на 2048. Получили результаты, представленные на рис. 5. В этот раз у нас возникло переобучение, т.к. к концу графика ошибка начала расти. При проигрывании наш агент постоянно находился слева, думая, что лучше ничего не делать, не попадаясь под огонь вражеских кораблей, т.к. это быстрее приведет гибели.

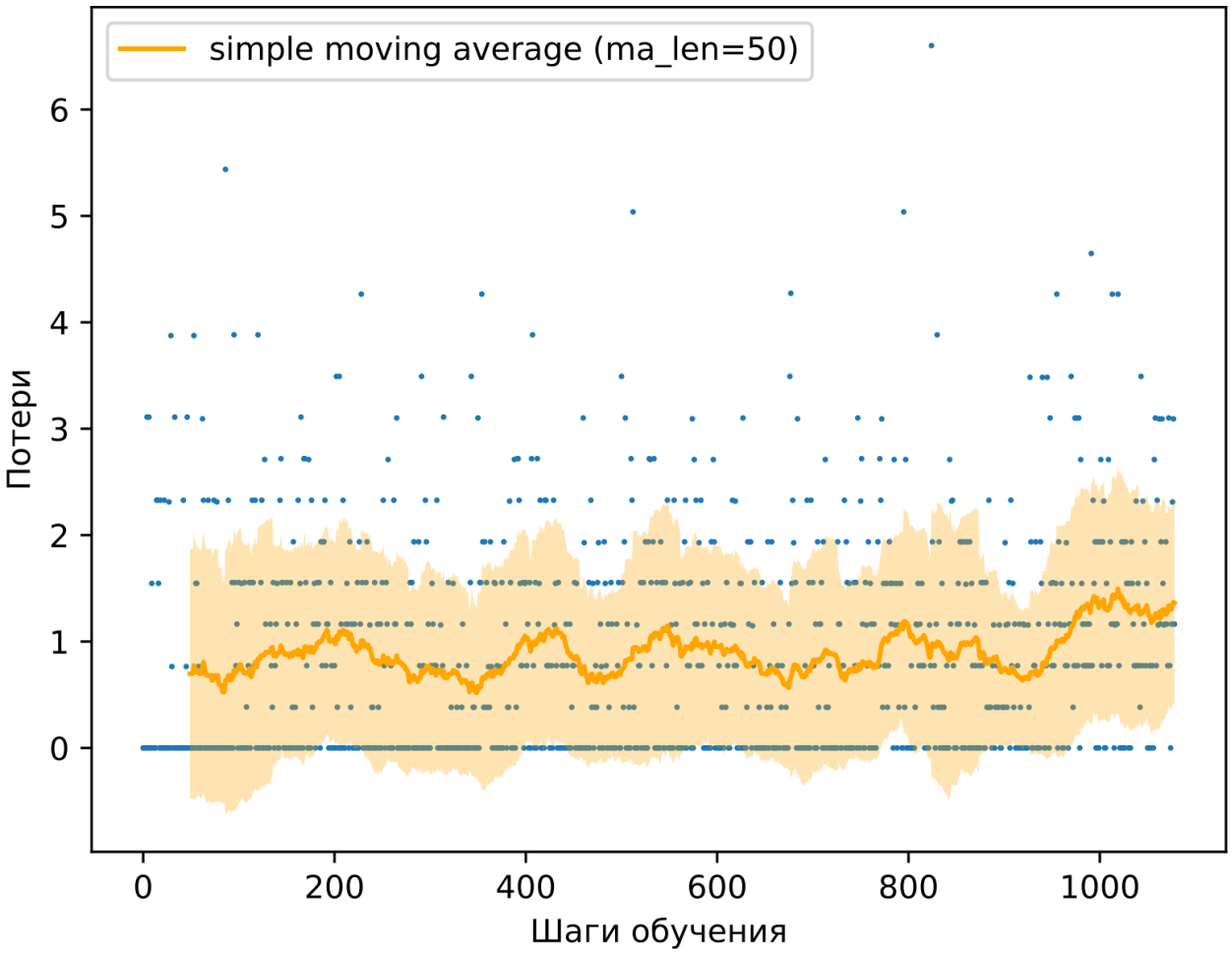

Рис. 5. Обучение с размером батча = 64, replay_capacity = 2048.

В итоге, наилучший результат получили для батча 64 с исходными параметрами.

Выводы

В ходе выполнения работы ознакомились с методами обучения с подкреплением на основе глубоких Q-сетей с помощью библиотеки Gym.