МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения и применения ЭВМ

ОТЧЕТ

по Индивидуальному домашнему заданию
по дисциплине «Искусственные нейронные сети»
Тема: LunarLander-v2

Студенты гр. 7382 _____ Глазунов С. А.
_____ Лящевская А. П.
_____ Еременко А.
Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2020

Требования к выполнению работы.

1. Требования к разрабатываемой модели.

- Модель должна быть разработана на языке Python с использованием Keras API.
- Исходный код проекта должен быть быть в формате PEP8.
- В исходном коде должны быть поясняющие комментарии.
- Модель не должна быть избыточной (должен соблюдаться баланс между размером сети [кол-во слоев и нейронов] и качеством выдаваемого результата).
- Обучение модели должно быть стабильно (для предложенной архитектуры ИНС обучение должно приводить к примерно одним и тем же результатом, то есть не должно быть такого, что при обучении 10 сетей удовлетворительный результат дают только 5 из них).
- Плюсом будет анализ с использованием callback'а TensorBoard.
- Плюсом будет разработка собственных callback'ов.
- Плюсом будет создание модели из ансамбля ИНС.

2. Требования к отчету.

- В отчете должно быть описание датасета, а также описание решаемой
- В отчете должен быть проведен начальный анализ данных. Проведение статистического анализа, анализ на необходимость нормировки данных, обоснование применения определенного вида нормировки, и.т.д.
- В отчете необходимо отразить весь процесс разработки: с чего началась разработка модели, на основании чего проводились те или иные изменения/корректировки, обоснование выбора тех или иных изменений/корректировок. По сути выполнение должно быть разбито на итерации, каждая итерация должна сопровождаться результатами модели на

итерации, а также краткой выдержкой исходного кода, показывающая изменения на итерации

- В конце отчета должен быть приведен анализ результирующей модели, а также перечислены возникшие проблемы (и как они были решены) и проблемы, которые решить не удалось. Плюсом будет предложение по улучшению модели.
- В отчете должно быть указано, кто в бригаде за что отвечал (написание отчета не является зоной ответственности)
- В приложении должен быть исходный код
- Плюсом будет сравнение разработанной модели с методами решающими задачу и не относящимися к ИНС.

Цель работы.

Целью работы является разработка модели искусственной нейронной сети, которая сможет посадить корабль на место посадки.

Распределение обязанностей.

Глазунов С. А. – Архитектура dqn.

Ляшевская А. П. – Разработка модели.

Еременко А. – Взаимодействие со средой из gym.

Постановка задачи.

Посадочная площадка всегда находится в точке с координатами (0,0). Координаты – это первые два числа в векторе состояния. Награда за переход от верхней части экрана к посадочной площадке и нулевой скорости составляет около 100–140 баллов. Если посадочный аппарат отходит от посадочной площадки, он теряет награду назад. Эпизод заканчивается, если посадочный аппарат падает или останавливается, получая дополнительные -100 или +100 очков. Каждый контакт ноги с землей равен +10. Мощность главного двигателя составляет -0,3 балла за каждый кадр.

Задача считается решенной, когда будет набрано 200 баллов. Возможна посадка вне посадочной площадки. Топливо бесконечно, поэтому агент может научиться летать и приземлиться с первой попытки. Доступны четыре отдельных действия:

1. Ничего не делать.
2. Запустить левый двигатель.
3. Запустить главный двигатель.
4. Запустить правый двигатель.

Основные теоретические положения.

Данная задача решается обучением с подкреплением, а именно вид искусственных нейронных сетей, такой как Deep Q Network (DQN). Также использовалось гамма-система подкрепления. Гамма-система подкрепления – это правило изменения весовых коэффициентов некоторого элемента, при котором веса всех активных связей сначала изменяются на равную величину, а затем из их всех весов связей вычитается другая величина, равная полному изменению весов всех активных связей, деленному на число всех связей. Эта система обладает свойством консервативности относительно весов, так как у неё полная сумма весов всех связей не может ни возрастать, ни убывать.

Обучение с подкреплением.

Традиционная архитектура ИНС с обучением подкреплением показано на рис. 1

Действие (A): Все возможные ходы, которые может предпринять агент.

Состояние (S): текущая ситуация, возвращенная окружающей средой.

Награда (R): немедленный возврат, отправленный обратно из среды, чтобы оценить последнее действие

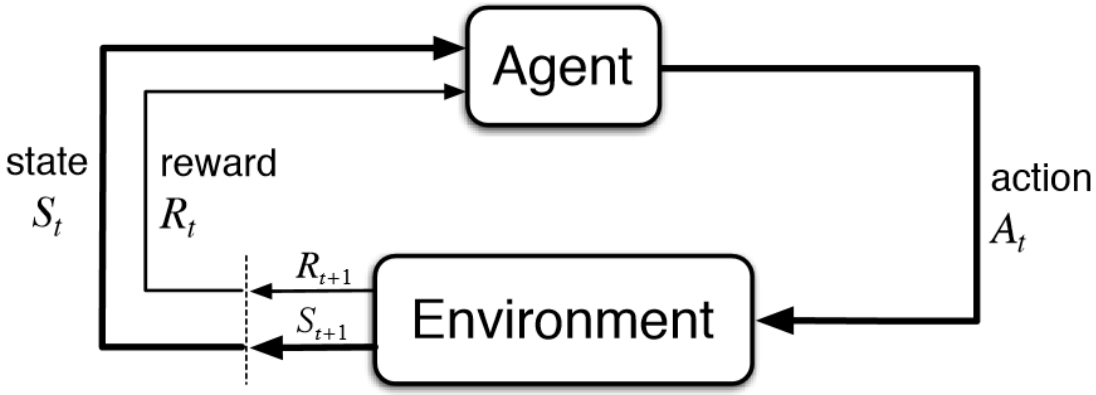

Рисунок 1 – архитектура ИНС

Тогда среда относится к объекту, на который действует агент (например, к самой игре в игре LunarLander-v2), в то время как агент представляет алгоритм RL. Среда начинается с отправки состояния агенту, который затем на основе своих знаний предпринимает действия в ответ на это состояние. После этого среда отправляет пару следующего состояния и вознаграждает агента обратно. Агент обновит свои знания с вознаграждением, возвращенным средой, чтобы оценить его последнее действие. Цикл продолжается до тех пор, пока среда не отправит состояние терминала, которое заканчивается эпизодом.

В нашем случае мы используем DQN. Поэтому более точная картинка видна на рис. 2

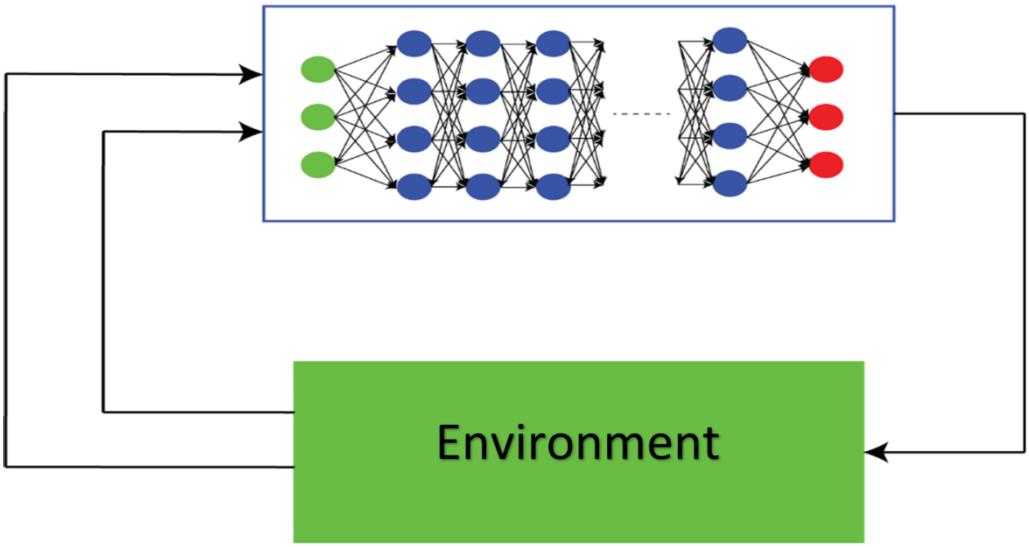

Рисунок 2 – Улучшенная архитектура ИНС

Ход работы.

- Было изучено как работать с системой, которая состоит из агента и окружающей среды в gym.
- Была выбрана и изучена архитектура DQN.
- Был введен гипер параметр epsilon, который позволял изначально нейронной сети обучиться просто летать и в целом обучил ее чему-то, что позволило получить более лучшие результаты
- Был добавлен Monitor, что позволило сохранять видео для каждой эпохи.

В качестве оптимизатора был выбран оптимизатор Adam с параметром LEARNING_RATE = 10^{-3}. Исходный код программы предоставлен в приложении А, Б, В, Г.

В ходе работы были рассмотрены 3 вариации модели:

1. «Модель 1» и график точности для данной модели представлены на рис. 3 и рис. 4 соответственно. Она обучалась 256 эпох с параметром BATCH_SIZE = 64 и дала следующий результат Average reward: 164.89.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	1152
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 4)	516

Рисунок 3 – Архитектура «Модели 1»

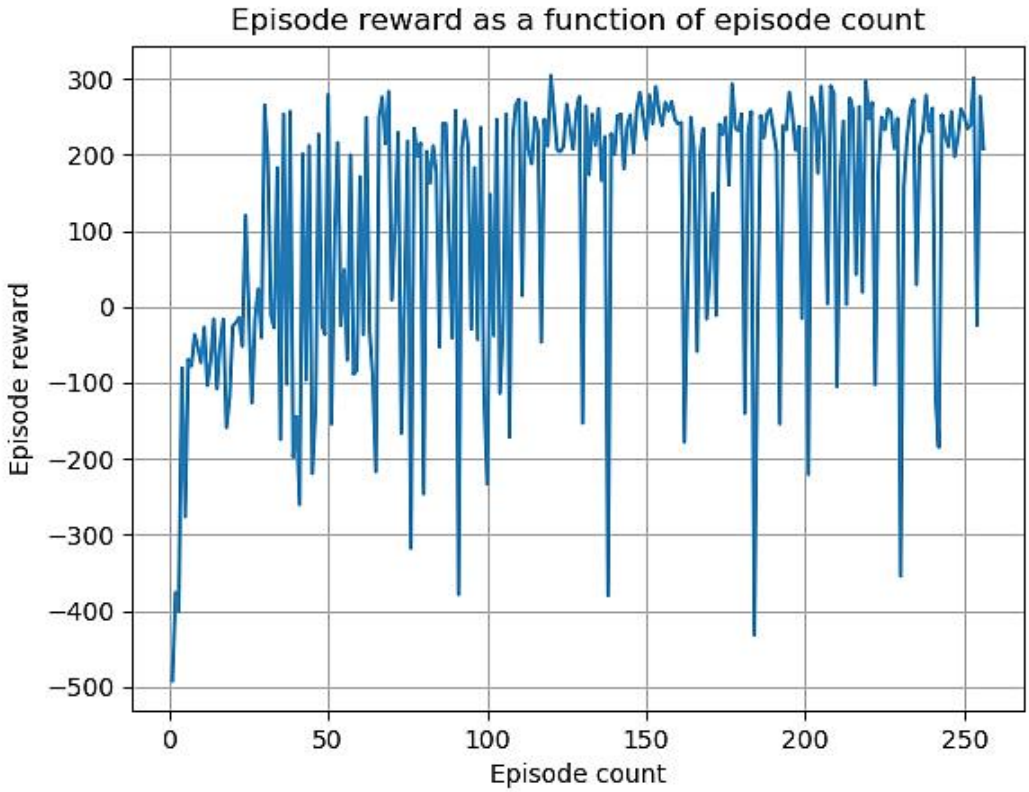

Рисунок 4 – График точности «Модели 1»

2. «Модель 2» и график точности для данной модели представлены на рис. 5 и рис. 6 соответственно. Она обучалась 256 эпох с параметром BATCH_SIZE = 64 и дала следующий результат Average reward: 214.38.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	576
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 64)	4160
dense_3 (Dense)	(None, 4)	260

Рисунок 5 – Архитектура «Модели 2»

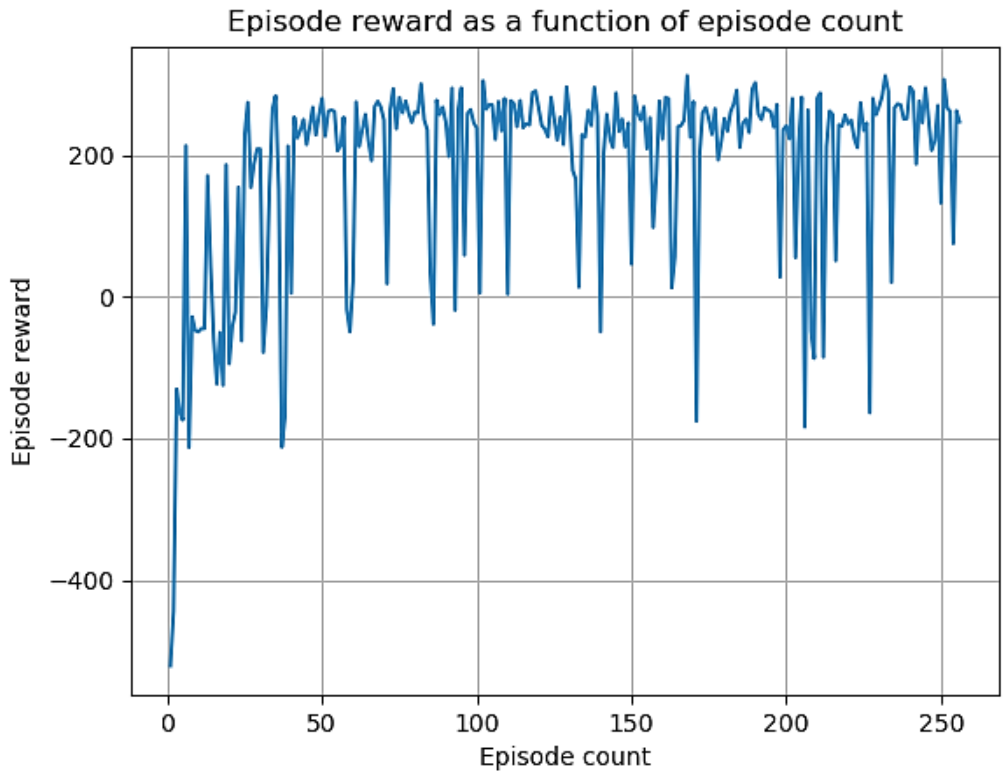

Рисунок 6 – График точности «Модели 2»

3. “Модель 3” и график точности для данной модели представлены на рис. 7 и рис. 8 соответственно. Она обучалась 256 эпох с параметром BATCH_SIZE = 32 и дала следующий результат Average reward: 153.88.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	1152
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 4)	516

Рисунок 7 – Архитектура «Модели 3»

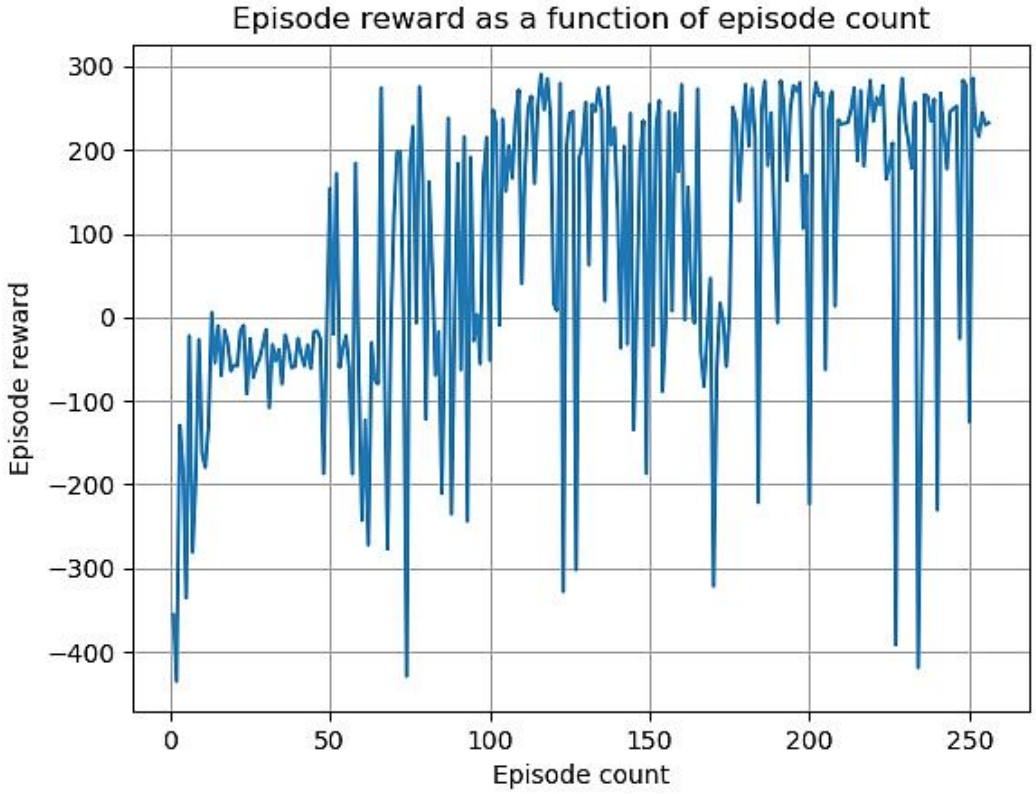

Рисунок 8 – График точности «Модели 3»

Обработка результатов.

В результате было решено использовать «Модель 2» как оптимальную. Она проще и в среднем дает лучше результаты, чем остальные модели.

Проблемы и их решения.

1. Без использования гипер параметра epsilon нейронная сеть переобучалась. А именно в ходе наблюдений за обучений было видно, что ИНС запоминает что надо делать. Используя epsilon мы предотвращаем переобучение, потому что каждый следующий с определенной вероятностью может быть выполнен ИНС, или быть случайным, тем самым мы усложняем задачу для ИНС, что избавляет нас от переобучения. Также использование гипер параметра позволило обучить ИНС быстрее - это было выявлено экспериментально. Это обуславливается тем, что первые эпохи(эпизоды) по

большей своей части выбирают следующие действия рандомно(в силу того, что гипер параметр еще принимает большое значение), тем самым ИНС получает разнородные данные, некоторые из которых выдают хороший результат(случайно), тем самым ИНС быстрее “понимает что лучше”. Без использования гипер параметра выходная оценка очень слабо менялась от эпохи к эпохе.

2. Проблема с использованием колбеков. Из за того, что мы используем обучение для каждой эпохи отдельно, от колбеков нет особого смысла. В планах на будущее найти способ подключить колбеки для такого рода ИНС.

```
action_state_value[[indices], [actions]] = next_action_state_value
self.model.fit(states, action_state_value, epochs=1, verbose=0)
if self.epsilon > self.epsilon_min:
```

Демонстрация обучения.

В ходе обучения видно, как корабль на каждой эпохе пытается приземлиться на место посадки (см. рис. 9,10). Используя всего 3 двигателя: влево, вправо и вниз.

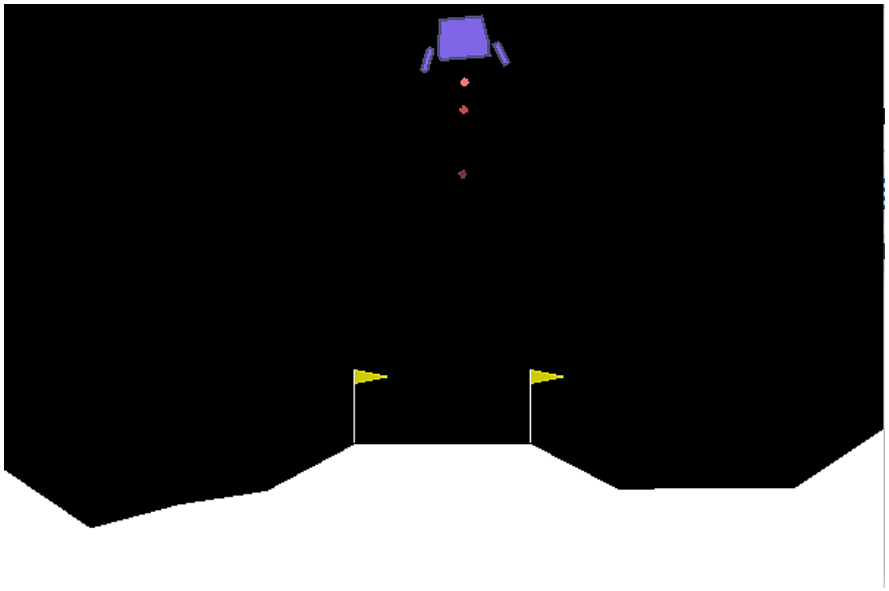

Рисунок 9 – Демонстрация 1

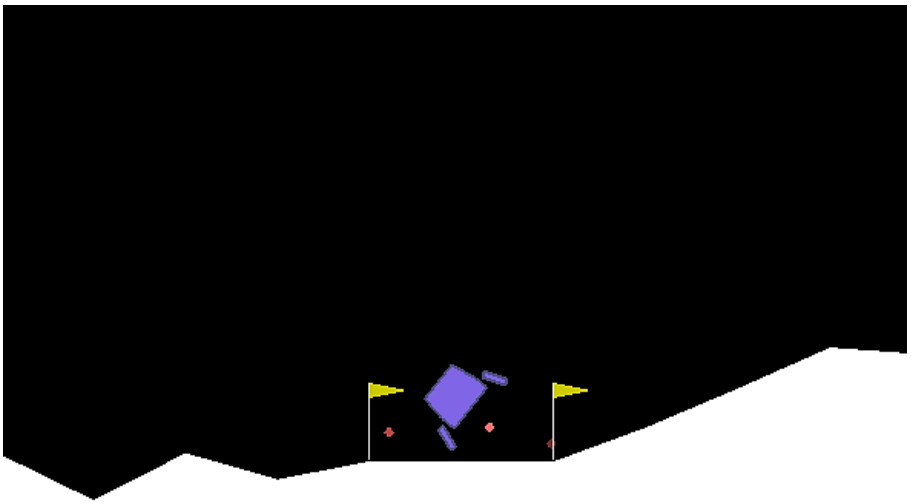

Рисунок 10 – Демонстрация 2

Выводы.

В ходе выполнения работы была разработана и обучена модель искусственной нейронной сети, которая может посадить корабль на место посадки. При этом она должна набрать больше 200 очков, чтобы задача считалась выполненной.

Для этого было изучено и практически применено обучение с подкреплением. В процессе были рассмотрены различные вариации моделей нейронной сети и ее параметров. Лучшей оказалась «Модель 2» см. рис. 5, она оказалась проще и давала лучшие результаты по сравнению с другими моделями.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД AGENT.PY

```
import random
import numpy as np
from collections import deque
from .model import Model
from .config import *

class Agent:
    def __init__(self, action_space, state_space):
        self.action_space = action_space
        self.batch_size = BATCH_SIZE
        self.epsilon = 1.0
        self.epsilon_min = EPSILON_MIN
        self.epsilon_decay = EPSILON_DECAY
        self.gamma = GAMMA
        self.memory = deque(maxlen=MEMORY_SIZE)
        print(action_space)
        print(state_space)
        self.model = Model(
            action_space,
            state_space,
            HIDDEN_NODES,
            HIDDEN_LAYERS,
            LEARNING_RATE
        ).build()

    def act(self, state):
        if np.random.rand() <= self.epsilon:
            actions = range(self.action_space-1)
            action = np.random.choice(actions)
        else:
            policy = self.model.predict(state)
            action = np.argmax(policy[0])
        return action

    def observe(self, environment, action):
        return environment.step(action)
```

```
    def remember(self, observation):
        self.memory.append(observation)

    def learn(self):
        if len(self.memory) > self.batch_size:
            batch = random.sample(self.memory, self.batch_size)
            states = np.array([i[0] for i in batch])
            actions = np.array([i[1] for i in batch])
            rewards = np.array([i[2] for i in batch])
            next_states = np.array([i[3] for i in batch])
            dones = np.array([i[4] for i in batch])
            states = np.squeeze(states)
            next_states = np.squeeze(next_states)
            action_state_value = self.model.predict_on_batch(states)
            next_action_state_value         =         rewards         +
self.gamma*np.amax(self.model.predict_on_batch(next_states),
axis=1)*(1-dones)
            indices = np.array([i for i in range(self.batch_size)])
            action_state_value[[indices],         [actions]]         =
next_action_state_value
            self.model.fit(states,    action_state_value,    epochs=1,
verbose=0)
            if self.epsilon > self.epsilon_min:
                self.epsilon *= self.epsilon_decay

    def load(self, directory):
        self.model.load_weights(directory)

    def save(self, directory):
        self.model.save_weights(directory)
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД MAIN.PY

```
import numpy as np
import gym
from gym.wrappers import Monitor
from lib.agent import Agent
from lib.summary import summary
from lib.config import *

def main():
    environment = gym.make(ENVIRONMENT)
    if RECORD:
        environment = Monitor(
            env=environment,
            directory=VIDEO_DIRECTORY,
            video_callable=lambda episode_id: True,
            force=True
        )
    environment.seed(0)
    np.random.seed(0)
    action_space = environment.action_space.n
    state_space = environment.observation_space.shape[0]
    agent = Agent(action_space, state_space)
    rewards = []
    for episode in range(EPISODES):
        state = environment.reset()
        state = np.reshape(state, (1, state_space))
        score = 0
        for _ in range(STEPS):
            environment.render()
            action = agent.act(state)
            next_state, reward, done, _ = agent.observe(environment,
action)
            next_state = np.reshape(next_state, (1, state_space))
            observation = (state, action, reward, next_state, done)
            agent.remember(observation)
            state = next_state
            agent.learn()
            score += reward
```

```
            if done:
                print("Episode:             {}/{}.             Reward:
{:.2f}".format(episode+1, EPISODES, score))
                break
        rewards.append(score)
        average_reward = np.mean(rewards[-100:])
        print("Average reward: {:.2f}\n".format(average_reward))
    environment.close()
    summary(rewards)

if __name__ == "__main__":
    main()
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД MODEL.PY

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import relu, linear
from tensorflow.keras.optimizers import Adam

class Model():
    def __init__(self, action_space, state_space, hidden_nodes,
layers, learning_rate):
        self.action_space = action_space
        self.state_space = state_space
        self.nodes = hidden_nodes
        self.layers = layers
        self.learning_rate = learning_rate

    def build(self):
        model = Sequential()
        model.add(Dense(self.nodes, input_dim=self.state_space,
activation=relu))
        for _ in range(self.layers):
            model.add(Dense(self.nodes, activation=relu))
        model.add(Dense(self.action_space, activation=linear))
        model.compile(
            optimizer=Adam(lr=self.learning_rate),
            loss='mse',
            metrics=['accuracy']
        )
        model.summary()
        return model
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД CONFIG.PY

```
ENVIRONMENT = 'LunarLander-v2'
FIGURE_DIRECTORY = './summary.png'
VIDEO_DIRECTORY = './video'
EPISODES = 256
STEPS = 1024
MEMORY_SIZE = 1000000
BATCH_SIZE = 64
HIDDEN_NODES = 128
HIDDEN_LAYERS = 2
LEARNING_RATE = 0.001
GAMMA = 0.98
EPSILON_MIN = 0.02
EPSILON_DECAY = 0.98
RECORD = True
```