CS 464 – Progress Report
Breakout Atari™ Game with Reinforcement Learning

Group 4
Abdullah Arda Aşçı (21702748), Alim Toprak Fırat (21600587),
Atahan Yorgancı (21702349), Tuna Alikaşifoğlu (21702125)

I. Introduction

In our term project we have decided on using reinforcement learning algorithms to learn how to play Atari™ Breakout game. In Breakout, the player controls a paddle at the bottom of the screen to score by destroying bricks above by hitting a ball in a manner similar to "pong". Upon destroying a brick player is awarded a single point of score, and if player misses and the ball goes out bounds the player's life is reduced. From a reinforcement learning perspective classic Atari games like Breakout are suitable RL environments with clearly defined action space, and reward structure for agents.

Fig. 1. Sample Atari™ Breakout Game with Random Agent [1]

More specifically in our project we design and train a deep neural network that can play the nostalgic Atari™ Breakout game. Deep Q-networks (DQN) combine Q-learning with deep neural networks to develop RL agents that can be applied to complex, high dimensional environments, like video games [2]. By receiving rewards and punishments, which are associated with certain in-game actions and outcomes, the agent is expected to develop "understanding" of game mechanics such as how to score.

II. Background Information

In machine learning, the process of learning is usually separated into 3 categories. Supervised, Unsupervised and Reinforcement learning. In supervised learning, a collection of data instances, and their respective labels are used by the learning algorithm to optimize weight values to be able to create a model which can predict label given features. In unsupervised learning, the unlabeled data is processed in order to perform clustering with respect to certain features, extract underlying patterns, and relations from data. Reinforcement learning doesn't use pre-collected data. Different from other learning methods in reinforcement learning, an agent is placed in an environment which can be interacted with using predefined set of actions, and the agent is trained by receiving rewards or punishments considering the actions of the agent in different states.

In reinforcement learning problems, the agents' past actions influence their current actions by changing the state of the environment. However,agents are not supplied information about which actions to take in a particular situation. The agents learn by maximizing reward (or minimizing punishment) by trying out different combinations from the set of all possible actions after observing the environment.

One of the key concepts in reinforcement learning is the *environment* which maintains the state, and at each time step the learning *agent* observers current

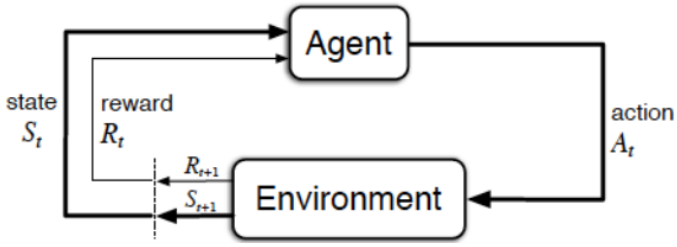

Fig. 2. RL agent, and environment [3]

state of the environment $s_t \in S$ where S is the set of all possible states, and s_t is the current state in time t. Then, the agent performs an action $a_t \in A$ depending on the state where a_t is the current action of the agent, and A is the possible action space. The environment updates its current state depending on the action of the agent, and outputs the next state s_{t+1}, and *reward* for the state next state s_{t+1}, r_{t+1}. This feedback loop continues until final condition is met, Figure 2 demonstrates this interaction between the environment, and the agent.

Further, *reward signal*, $r_a(s, s')$, is the reward for transitioning from state s to s' by taking action a. *Value functions*, $V_\pi(s)$, is the total amount of reward is expected from a particular state. While the reward signal is instantaneous, value function calculates the long-term future rewards that can be expected. *Policy*, π, is a mapping from the agent's observation of the environment to the set of actions. Informally, the policy of the agent can be considered as the "strategy" used by agent to receive more reward. The agent may choose to alter its policy based on how high/low reward it gains from its actions.

RL systems can be broken into two main categories based on whether a *model* of an actual environment is used that are model-based methods, and model-free methods. A *model* of the environment mimics the the behavior of the environment so that the agent can plan what action will result in what kind of outcome. Conversely, in model-free methods, the agent learns by trial-and-error instead of planning out actions [3].

A. Q-Learning

Q-learning is a model-free reinforcement learning algorithm. It gives the agents the ability to learn to take optimal actions to move around Markovian state space [4].

Assume there exists an agent and an environment. Let $\pi : A \times S \rightarrow [0, 1]$ be the policy map which gives to probability of taking an action in a state. As previously mentioned for time step t the state of the agent is $s_t \in S$, and it chooses the action $a_t \in A$. Define γ, where $0 \geq \gamma \geq 1$, to be the *discount factor*. Then, the future discounted return at time t is discounted sum of all future rewards.

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

where T is the time step that the game comes to an end. Also define the optimal action-value function as the maximum expected return achievable by taking any action.

$$Q^*(s, a) = max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{2}$$

Since estimating the real value of the action-value function is too costly, a function approximator is used. $Q(s, a; \theta) \approx Q^*(s, a)$.

Neural networks are an example for a non-linear approximator, with weights θ. A Q-network is trained to minimize the loss function L_i which changes at each each epoch.

$$L_i(\theta_i) = \mathbb{E}_{(s,a)}[(y_i - Q(s, a; \theta_i))^2] \tag{3}$$

$$y_i = \mathbb{E}_{s'}[r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) | s, a] \tag{4}$$

where y_i is the target which depends on the weights of the neural networks, θ. Gradient of the loss function with respect to θ, we get.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{(s,a),s'}[(r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a : \theta_i)) \tag{5}$$

The expected value's calculation can also be too costly, rather than calculating it directly, *stochastic gradient descent* can be used [5].

III. Work Done & Reaming Work

A. Work Done

In our proposal, we stated that we would conduct a research on reinforcement learning, deep neural networks and more specific algorithms like Deep Q-networks (DQN) that combines Q-learning with

deep neural networks to let reinforcement learning to be applied to complex, high dimensional environments, like video games [2]. We conducted the promised research to find the optimal approaches to the problem in hand. We informed ourselves on the basics of reinforcement learning and especially on Q-learning with DQN implementations. There are more recent solutions like DeepMind's *Agent57* [6] that can outperform DQN based solutions for atari games. However, these type of approaches are necessary for games that have a credit assignment problem, which means that if we choose an action, and we only win or lose hundreds of actions later, leaving us with no idea as to which of our actions led to this win or loss, making it difficult to learn from our actions. The chosen Breakout game is not that complex nor that far-fetched. Thus, our prior research concludes that DQN based solution will be more than sufficient to outperform the atari human benchmark for the Breakout game.

In addition to our background research, we declared that we would obtain a working, playable version of the Breakout game which is suitable for training and testing our artificial agent. In this context, we are utilizing *OpenAI*'s "Gym" environments for the implementation of the Breakout atari game. Basically there are two versions of the Breakout game present as a Gym environment. One of the versions provides information of the RAM throughout the game [1], and the other provides 3-channel image for each frame throughout the game [7]. Currently both versions of the Breakout game can be initiated from our implementation, and we will decide which one of these versions is the optimum for our case after some experimentation.

At this point of the project, we obtained a playable version of the Breakout game with an human agent, that we call the "keyboard agent" by utilizing the example provided in the *GitHub* repo of `openai/gym` [8]. With this aspect of the program, an human agent can play the Breakout atari game by using the keyboard. Version of the game, either RAM based or image based, and the game speed can be adjusted using CLI arguments. In addition to our playable version, we also managed to initialize an artificial agent based version which is suitable for training and testing. Currently, this agent only takes random actions from the action space. However, this environment is crucial for the remaining of the project, and the training and testing will be conducted on this environment, details are provided in the next section.

B. Remaining Work

At this point we have both RAM and image based Breakout atari game, with an artificial agent and keyboard controlled versions. Currently, artificial agent only takes random actions, but we are planning to thrive on obtaining an artificial agent that outperforms human benchmark. The environments for the reinforcement learning aspect of the project are built, and they are operating successfully. The remaining work is to generate DQN based solution to improve the performance of the artificial agent. In the light of our background research, we believe we can provide an elegant approach. Finally, our research unveils that we should obtain an artificial agent that meets our requirements approximately within 70-96 hours of training.

IV. Work Breakdown Structure

TABLE I
Task Sharing

Worker	Task
Abdullah Arda Aşçı	Setting OpenAI, and developing wrappers for `gym` en
Atahan Yorgancı	Background research about RL, and DQN
Alim Toprak Fırat	Background research Q-Learning
Tuna Alikaşifoğlu	Developing CLI for running, and training using `gym`

References

[1] OpenAI. (Oct. 2019). "Gym," [Online]. Available: https://gym.openai.com/envs/Breakout-ram-v0/. [Accessed: Mar. 3, 2021].

[2] ——, (Sep. 2020). "Openai baselines: Dqn," [Online]. Available: https://openai.com/blog/openai - baselines - dqn/. [Accessed: Mar. 3, 2021].

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[6] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, *Agent57: Outperforming the atari human benchmark*, 2020. arXiv: 2003.13350 [cs.LG].

[7] OpenAI. (Oct. 2019). "Gym," [Online]. Available: https://gym.openai.com/envs/Breakout-v0/. [Accessed: Mar. 3, 2021].

[8] Openai, *Openai/gym*, Apr. 2020. [Online]. Available: https://github.com/openai/gym/blob/master/examples/agents/keyboard_agent.py.

APPENDIX A
PYTHON CODE

```python
import gym
import time
import click


@click.group()
def cli():
    pass


@cli.command()
@click.option('--env_name', default='Breakout-v0',
              help='Name of the gym environment.')
@click.option('--it_count', default=0,
              help='Bound number of ')
@click.option('--delay', default=0.0,
              help='Delay duration to set the game speed.')
def random_agent(env_name, it_count, delay):
    env = gym.make(env_name)
    observation = env.reset()

    iteration = 0
    while iteration <= it_count:
        env.render()
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            observation = env.reset()
        if it_count > 0:
            iteration += 1
        time.sleep(delay)
    env.close()


@cli.command()
@click.option('--env_name', default='Breakout-v0',
              help='Name of the gym environment.')
@click.option('--delay', default=0.1,
              help='Delay duration to set the game speed.')
def keyboard_agent(env_name, delay):
    env = gym.make(env_name)
    ACTIONS = env.action_space.n
    SKIP_CONTROL = 0

    def key_press(key, mod):
        global human_agent_action, \
            human_wants_restart, \
            human_sets_pause
        if key == 0xff0d:
            human_wants_restart = True
        if key == 32:
            human_sets_pause = not human_sets_pause
        a = int(key - ord('0'))
        if a <= 0 or a >= ACTIONS:
            return
        human_agent_action = a

    def key_release(key, mod):
        global human_agent_action
        a = int(key - ord('0'))
        if a <= 0 or a >= ACTIONS:
            return
        if human_agent_action == a:
            human_agent_action = 0

    env.render()
    env.unwrapped.viewer.window.on_key_press = key_press
    env.unwrapped.viewer.window.on_key_release = key_release

    def rollout(env):
```

```
        global human_agent_action, \
            human_wants_restart, \
            human_sets_pause
        human_wants_restart = False
        obser = env.reset()
        skip = 0
        total_reward = 0
        total_timesteps = 0
        while True:
            if not skip:
                a = human_agent_action
                total_timesteps += 1
                skip = SKIP_CONTROL
            else:
                skip -= 1

            obser, r, done, info = env.step(a)
            if r != 0:
                print('reward %0.3f' % r)
            total_reward += r
            window_still_open = env.render()
            if not window_still_open:
                return False
            if done or human_wants_restart:
                break
            while human_sets_pause:
                env.render()
                time.sleep(delay)
            time.sleep(delay)
        print((f'timesteps {total_timesteps} '
               f'reward {total_reward:0.2f}'))

    print(f'ACTIONS={ACTIONS}')
    print('Press keys 1 2 3 ... to take actions 1 2 3 ...')
    print('No keys pressed is taking action 0')

    while True:
        window_still_open = rollout(env)
        if window_still_open == False:
            break


if __name__ == '__main__':
    # Keyboard Global Attributes
    human_agent_action = 0
    human_wants_restart = False
    human_sets_pause = False
    cli()
```