Reinforcement Learning Algorithms for Atari 2600 games

Fatma Betül Yazıcı
Department of Computer Engineering
İstanbul Technical University, Turkey
Email: betulyazicii@gmail.com

Çağrı Karahan
Department of Computer Engineering
İstanbul Technical University, Turkey
Email: cagrikarahan94@gmail.com

***Abstract*—There are lots of works about improving control actions with training agents in Atari games. In this work, we aim to improve one of these approaches to get higher scores from the game. First of all, like the general construction methods, we construct our network model with a convolutional neural network. Convolutional neural networks algorithm has remarkable performance for training Reinforcement Learning agents to learn optimal policy from high dimensional visual inputs. Then, we train our agents using an enhanced Deep Q-Network (DQN), which is supported by classical exploration strategies. We simulated our strategies in SpaceInvaders from Atari 2600 using the Gym atari library from the Open AI. Moreover, we compare the original approach and our enhanced approach according to scores that taken each step in the simulation.**

***Index Terms*—Reinforcement Learning, Artificial Intelligence, Markov Decision Process, Deep Q-Networks, Atari Games**

I. Introduction

Recently, deep reinforcement learning which combines deep learning and reinforcement learning has been paid much attention. DQN is one of the most famous methods of deep reinforcement learning. Many research showed the DQN algorithm has the ability to acquire the game playing skills superior to human experts in Atari 2600 games. It was difficult to deal with the high-dimensional image data as input data in the conventional reinforcement learning. DQN has remarkable features to directly learn the action policy from the high dimensional image data by using Convolutional Neural Network.

Most reinforcement learning applications has challange two problems. One of a scalar reward signal that is frequently sparse, noisy and delayed. Other issue is that most deep learning algorithms assume the data samples to be independent, while in reinforcement learning one typically encounters sequences of highly correlated states. In this paper we show that a convolutional neural network can overcome these problems to learn successful control policies from video data in RL environments. We trained with a DQN algorithm for Atari games, applying different action selection strategy and comparing results.

The paper is organized as follows. In Section II related work different techniques on reinforcemet learning will be mentioned. In Section III proposed methodology will be explained in detail with the subheadings Agent Description, DQN Algorithm and Enhancement of Work - Different Action Selection Strategy. Describes the preprocessing, CNN architecture, hyper-parameters, Open AI Gym and experimental results in Section IV. Conclusion and final remarks are given in Section V.

II. Related Work

The objective of reinforcement learning is to find a good strategy by maximizing the cumulative future reward from the environment [1]. It was widely used in game theory. In this section, we will examine the work done in this field.

Volodymyr Mnih and David Silver implement a deep learning model to control agents from the environment [2]. For these, they make agents learn from high dimensional inputs by using reinforcement learning. They applied their techniques to Atari 2600 games. They used seven different Atari 2600 games and trained agents that will become a human expert. They construct network model with a convolutional neural network like the result of general deep learning methods. They implemented a variant of Q-learning, which cooperates with using replay memory to ease the training of deep networks for controlling complex control policies in Atari 2600. Moreover, they take raw pixels and use the same type of output to show the game.

Y. Huang, G. Wei and Y. Wang propose an algorithm that is Variant of Double dueling DQN [3]. With this approach, they reduce the unnecessary estimations in action values while a program is learning. After demonstrating their approach, they compare their performance with the Double DQN algorithm and show us a better route planning domain. Moreover, they also show that their algorithm has the generalization ability of a dynamic environment.

A. Jeerige, D. Bein and A. Verma published a paper to let readers understand the deep reinforcement learning approaches that help create agents [4]. They simulate existing approaches and compare their results. They test these existing approaches in Atari 2600 game called Breakout. They aim to see achieving high scores when they were training agents control policies. They use two different deep reinforcement learning approaches that are Asynchronous Advantage, actor-critic and Deep Q-learning .

Seonghun Yoon and Kyung-Joong Kim propose to use deep Q Networks (DQN) for the visual fighting game AI competitions [5]. The number of actions was reduced to 11 and the sensitivity of several control parameters was tested using the visual fighting platform. The authors shows the potential of DQN approach for the two player real-time fighting game.

Manan Tomar and Yonathan Efroni propose multi step greedy reinforcement learning algorithm [6]. The author using K-Policy Iteration (K-PI) and K-Value Iteration (K-VI) algorithms. These methods iteratively compute the next policy and value function by solving a surrogate decision problem with a shaped reward and a smaller discount factor. They derive model-free RL algorithms based on K-PI and K-VI in which the surrogate problem can be solved by any discrete or continuous action RL method, such as DQN. They identify the importance of a hyperparameter that controls the extent to which the surrogate problem is solved and suggest a way to set this parameter. They evaluate the proposed algorithm Atari games and multi-step greedy algorithms significantly improve the performanse of DQN.

III. PROPOSED METHOD

For computational interaction with the environment, we have to define the environment computationally. The Markov decision process can be used to model the RL problem.

A. Agent Description

State Space S : The state of a reinforcement learning environment is the set of all information that an agent requires to make the correct choice of the control action to take [7]. The state space is the set of all states in the environment. Environments do not provide direct access to the state. Instead we use visual observations, typically 210 × 160 RGB images. A single image does not determine the state. In order to reduce environment's partial observability, we stack N consecutive frames and use it as the observation. In this experiment we will take the hyperparameter N = 4.

Action Space A: The action space is the union of all possible action sets for all the states in the environment. $a_t = a_t^1, ..., a_t^K \in A$ indicates the action that agent taken by agent at time t. The number of action size (K) is the based on problem. There are six actions can be taken for the Space Invaders atari game: noop, fire, right, left, right fire and left fire.

Action Selection Strategy: The agent selects and executes an action according to an ϵ-greedy policy. An ϵ-greedy selection policy where a random action is selected with a probability $\epsilon \in [0, 1]$. The purpose of this is to encourage exploration. Figure 1 shows the Epsilon-Greedy action selection strategy.

$$\begin{cases} argmax_a Q(s, a; \theta) & \text{With probability } 1\text{-}\varepsilon \\ \text{select a random action } a & \text{With probability } \varepsilon \end{cases}$$

Fig. 1. Epsilon-Greedy Action Selection

Reward R: A reward r_t representing the change in game score. The goal of the agent is to interact with the environment by selecting actions in away that maximizes future rewards. We make the standard assumption that future rewards are discounted by a factor of γ per time-step (γ taken 0.99 as a hyperparameter), and define the future discounted return at time t Eq.(1) as follows:

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

in which T is the time-step at which the game terminates. In this work fixed positive rewards to be 1,negative rewards to be -1 and leaving 0 rewards unchanged situations.

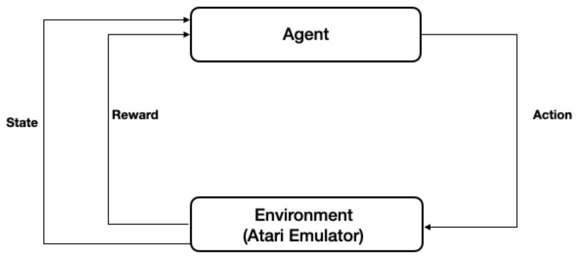

Fig. 2. Agent-Environment Interactions in MDP

Figure 2 shows the agent-environment interactions in MDP formulation. We consider tasks in which an agent interacts with an environment, in this case the Atari emulator, in a sequence of actions, observations and rewards. At each time-step the agent selects an action a_t from action space. The action is passed to the environment and modifies its internal state and the game score. The emulator's internal state is not observed by the agent; instead the agent observes an image x_t. In here $x_t \in R^d$ which is a vector of pixel values representing the current screen.

The agent only observes the current screen because of that the task is partially observed. Also it is impossible to fully understand the current situation from only the current screen x_t.

Therefore, sequences of actions and observations, $s_t = x_1, a_1, x_2, ..., a_{t-1}, x_t$ are input to the algorithm, which then learns game strategies depending upon these sequences. All sequences in the environment are assumed to terminate in a finite number of timesteps.

B. DQN Algorithm

DQN algorithm uses the Q-learning method which is one of the most widely-used reinforcement learning and also uses CNN in order to approximate the action-value function called Q-function. Reinforcement learning is known to be sometimes unstable or even to diverge when the nonlinear function approximator such as neural network is used to represent the action-value function (Q-function).

There are several reasons for this instability such as i) the correlations present in the sequence of state observations s_t

and s_{t+1} ii) small updates of Q-value may significantly change the policy and therefore change the data distribution.

In order to overcome the above problems, DQN uses the method called experience replay. In order to perform experience replay, the agent's experiences $e_t = s_t, a_t, r_t, s_{t+1}$ at each time step t are stored in the data set $D = e_1, ..., e_N$. The data set D is also called replay memory. During learning, we apply Q-learning updates on samples of experience (s_j, a_j, r_j, s_{j+1}) drawn uniformly at random from the data set D.

In figure 3 summarizes the algorithm of DQN with experience replay.

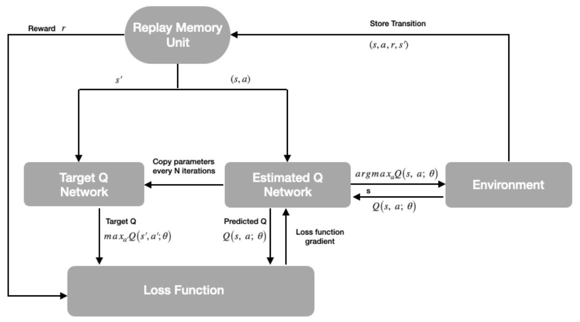

Fig. 3. A data flow diagram for a DQN with a replay buffer and a target network

In the reinforcement learning community this is typically a linear function approximator, but sometimes a non-linear function approximator is used instead, such as a neural network. We refer to a neural network function approximator with weights as a Q-network.

First using the state of the agent as the input to the neural network. According to each state of the agent, the Q values of all current actions are obtained by analyzing the convolutional neural network. We define this network estimated q network or prediction network. Q(s,a;θ) used to evaluate the value function of the current state action pair. Also, we use another network to generate the target Q value through the Q-learning algorithm. We use Q(s,a;θ') to indicate the output of target q network The target Q value is calculated by Eq.(2).

$$target = r + \gamma max_{a'} Q(s', a'; \theta') \tag{2}$$

The parameter θ of the estimated q network is updated in real time. Every iteration of N iterations, the parameters of the estimated q network are copied to the target q network. Then, the stochastic gradient descent method can be used to optimize the weights of DQN so that the loss function is minimized and the estimated q value gradually approaches the target q value. The loss function is:

$$L_i(\theta_i) = E[\,(r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^{\,2}] \tag{3}$$

A Q-network can be trained by minimising a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i.

C. *Enhancement of Work - Different Action Selection Strategy*

One drawback of ϵ-greedy action selection strategy is that the action is selected uniform randomly from the set of possible actions. Therefore, it is as likely to choose the worst appearing action as it is to choose the second-best appearing action if an exploration action is selected. For this reason, different action selection strategies examined.

1) Boltzmann Exploration Approach: Boltzmann exploration uses the learned Q values, and agents find their actions with this exploration [8]. Boltzmann exploration regulated by a temperature parameter T. In this approach uses Boltzmann distribution function to assign a probability $\pi(s_t, a)$ to the actions in order to create a weighted function of estimated value:

$$\pi(s_t, a) = \frac{e^{Q_t(s_t,a)/T}}{\sum_{i=1}^{K} e^{Q_t(s_t,a^i)/T}} \tag{4}$$

$\pi(s_t, a)$ denotes the probability the agent selects action a in state s_t and $T \geq 0$ is the temperature parameter used in the Boltzmann distribution. When T = 0 the agent does not explore at all, and when $T \rightarrow \infty$ the agent selects random actions.

Boltzmann exploration does not always get the optimal action or random action. This approach contains choosing an action with probabilities include actions weighted. We use a softmax over the network's estimates of value for each action to succeed in this. In this case, the agent estimates to be optimal is most likely (but is not guaranteed) to be chosen. The most significant advantage over ϵ-greedy is that information about the potential value of the other actions can also be taken into consideration.

In practice, we use a temperature parameter and this parameter controls the spread of softmax distribution. If the temperature parameter too big, all actions results will have the same probability. If the parameter's value is too small, the computer will select the action that has the greatest probability. In this experiment we will take the temperature parameter T=500.

IV. EXPERIMENTAL RESULTS

Preprocessing: Atari game frames are 210 × 160 pixel images with a 128-color palette. We convert the RGB representation to gray-scale and crop an 160 × 160 region of the image that captures the playing area and then the cropped image is down-sampled to 84 × 84 in order to reuse DQN's CNN architecture. This procedure is applied to the last 4 frames associated with a state and stacked to produce a 84×84×4 preprocessed input representation for each state.

CNN Architecture: We use the same deep neural network architecture as DQN for our agents. Our network consists of three hidden layers. The input to the neural network is an 84 × 84 × 4 image produced by the preprocessing procedure step. The first hidden layer convolves 16, 8 × 8, filters with stride 4 with the input image and applies a rectifier nonlinearity. The second hidden layer convolves 32, 4 × 4, filters with stride 2 again followed by a rectifier nonlinearity. The final hidden layer is fully connected and consists of 256 rectifier units. The output layer is a fully connected linear layer with a single output for each valid action.

In figure 4 summarizes the CNN architecture.

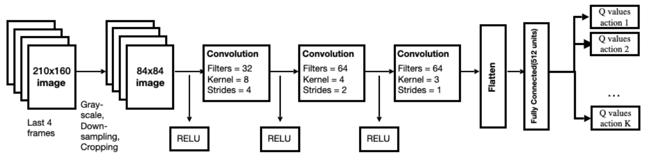

Fig. 4. CNN Architecture

Open AI Gym: We used OpenAI Gym (Brockman et al., 2016) for simulating Atari environment [9]. It is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games Gaming from Pixels like Pong or Breakout.

Hyper-parameters:

In all experiments, the discount factor was set to γ= 0.99 and the Adam optimizer was used with the learning rate α= 0.00001. The target network was updated every 40.000 steps. The size of experience replay memory was 3000000 tuples and batch size was 64. ϵ-greedy was used with the epsilon decreasing 1 to 0.1 over 1000000 steps.

In figure 5, Boltzmann's exploration performance is better than epsilon greedy exploration in training. Moreover, in the test case of one hundred games, Boltzmann's exploration gets the better score in almost every game. Furthermore, the maximum score of Boltzmann's exploration is more than 700. However, the maximum score of epsilon greedy is almost 500.

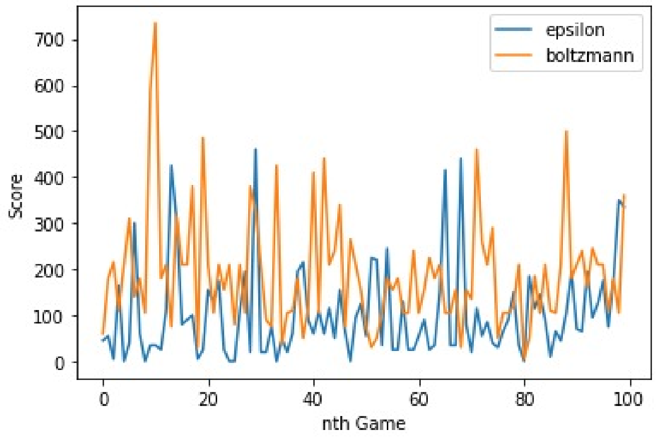

Fig. 5. Comparison of Epsilon and Boltzmann Strategies

When an agent selects an action in training with epsilon greedy exploration, it is chosen as the highest q-value or random action. The problem with epsilon greedy is that it chooses all actions equally when deciding on what action to take. Maybe some actions might look more encouraging than others. Boltzmann exploration does that. This strategy exploration will select the action that has the best probability.

V. CONCLUSION

In this paper, we propose a DQN algorithm with Boltzmann exploration approach. Experimental results compared DQN with epsilon greedy approach. In the future study, it is aimed to apply different reinforcement learning methods with different action selection strategy.

REFERENCES

[1] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction (no. 1). MIT press Cambridge, 1998.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," Computer Science, 2013.

[3] Y. Huang, G. Wei and Y. Wang, "V-D D3QN: the Variant of Double Deep Q-Learning Network with Dueling Architecture," 2018 37th Chinese Control Conference (CCC), Wuhan, China, 2018, pp. 9130-9135, doi: 10.23919/ChiCC.2018.8483478.

[4] A. Jeerige, D. Bein and A. Verma, "Comparison of Deep Reinforcement Learning Approaches for Intelligent Game Playing," 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2019, pp. 0366-0371, doi: 10.1109/CCWC. 2019.8666545.

[5] S. Yoon and K. Kim, "Deep Q networks for visual fighting game AI," 2017 IEEE Conference on Computational Intelligence and Games (CIG), 2017, pp. 306-308, doi: 10.1109/CIG.2017.8080451.

[6] Manan Tomar, Yonathan Efroni,"Multi-step Greedy Reinforcement Learning Algorithms" Proceedings of the 37th International Conference on Machine Learning, PMLR 119:9504-9513, 2020.

[7] M. M. Kokar and S. A. Reveliotis, 'Reinforcement learning: Architectures and algorithms' Int. J. Intell. Syst., vol. 8, no. 8, pp. 875–894, 1993

[8] K. Azizzadenesheli, E. Brunskill and A. Anandkumar, "Efficient Exploration Through Bayesian Deep Q-Networks," 2018 Information Theory and Applications Workshop (ITA), 2018, pp. 1-9, doi: 10.1109/ITA.2018.8503252.

[9] Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech https://openai.com/blog/, 2016