Deep Convolutional Networks with Genetic Algorithm for Reinforcement Learning Problem

Mohamad Yani^{*1}, Naoyuki Kubota^{*2}

^{*1} Tokyo Metropolitan University, Hino, Tokyo, Japan
E-mail: mohamad-yani@ed.tmu.ac.jp
^{*2} Tokyo Metropolitan University, Hino, Tokyo, Japan
E-mail: kubota@tmu.ac.jp

The principles of reinforcement learning present a normative account firmly related to neuroscience and psychological viewpoints on how animals or humans survive and find their optimal state-action values of an environment. In order to use reinforcement learning successfully in conditions approaching real-world application complexity, however, agents are confronted with a challenging task: they need to obtain correct information of the environment from various kinds of sensors and manage these to generate optimal state-action values from experience to new conditions and also for long-term survival. Several dedicated approaches have been developed to improve reinforcement learning performances—DQN, Double DQN, SARSA name a few. Since reinforcement learning tasks require maximizing a reward function for the long term, we consider them as challenging optimization problems. Classical deep Q-network (DQN) with Backpropagation (BP) can solve challenging reinforcement learning problems like classic Atari games and robotics problems. However, DQN with BP usually takes a long time to train and difficult to get convergence in a short training. In this paper, we optimize DQN with Genetic Algorithms (GA) for a reinforcement learning problem. We also provide comparison results between DQN with GA and traditional DQN to show how efficient this method is compared to the standard DQN. The results demonstrate that DQN with GA has better results on CartPole problem.

Keywords: Deep Q-Networks, reinforcement learning, Genetic Algorithm.

1. INTRODUCTION

Developing the human and cognitive skills for agents such as robots and intelligent systems to perform challenging tasks in a dynamic and challenging environment is a primary goal of reinforcement learning in a real-world. Deep Q-network with experience replay has fascinated performance and reached the human-like level control to play classic Atari game environment in the previous work [1]. In the manipulation and navigation task, the reinforcement learning (RL) approach opens the chance for robotics and many other problems to have human abilities by directly learning manipulation and navigation from various sensory inputs such as image, force, and sound. Initial accomplishments in the field were promising; however, they showed some inherent problems for RL to solve real-world robotic manipulation applications.

Numerous deep-Q Networks and its modification have been developed to optimize and find the state-action values on reinforcement learning discrete problems, such as classical Atari games, collision avoidance in robotics navigation, and some manipulation problems with a monocular camera [1]–[6]. Even though this model may eventually generate impressive results, it leads to overestimating state-action values and needs a high computation cost to train.

They all approximate gradients in a Deep Neural Network (DNN) and optimize those parameters via stochastic gradient descent/ascent (though they do not need differentiating through the reward function, e.g. a simulator). Several results [2], [3], [7]–[9] from past decade have proven the effectiveness of BP applied to deep neural networks in conventional reinforcement learning tasks. However, DQN with BP usually takes a long time to train, and it is challenging to get convergence in a short time of training, and it may get trap in local optima. Stuck in local optima indicates when updating the weights, which minimize the temporal-difference error value, which is not necessarily the smallest one at all other possible value [10]. One of the techniques that can be used to overcome computational costs is Genetic Algorithms (GA). GA enables to train a Convolutional Neural Networks (CNN) and fully connected networks (FCN) for the Q-value estimation without any gradient-based computations required. Therefore, this paper investigates and compares GA performance as the gradient-free method to solve the reinforcement learning problem. We provide comparison results between deep Q-networks with GA and traditional DQN in order to show the effectiveness of each method for reinforcement learning problem from OpenAI gym

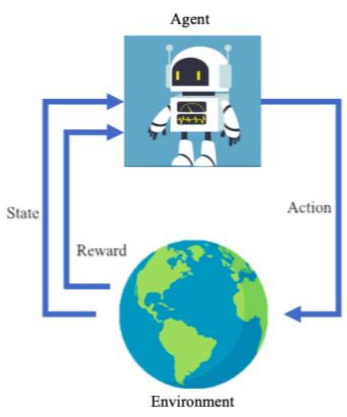


Figure 1. Agent and environment relationship in reinforcement learning, where the agent obtained the state of the environment and the reward, uses the information to decide an action to do.

simulation. This paper is organized as follows. Section 2 discuss the background and brief fundamental theory of Q-Convolutional neural Networks. Section 3 explains the Deep Q-Networks trained with GA. Section 4 shows numerical simulation results using OpenAI Gym environment. Finally, Section 5 discuss the essence of the proposed method and discusses the future direction of this research.

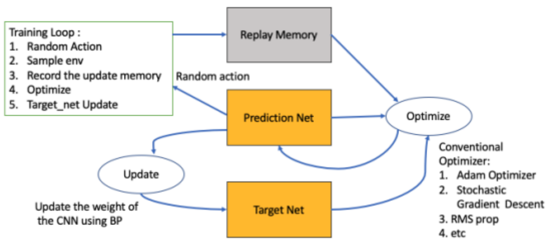


Figure 3. DQN with Experience Replay diagram.

2. BACKGROUND

A reinforcement learning mechanism comprises an agent that detects an environment in a specific state and takes actions to maximize future rewards [11]. The agent receives only a state and reward information through the high dimensional sensory input, and it can only affect the environment through its actions, as described in Figure 1. The goal is to maximize the total discounted reward it receives. Q-Learning is one approach for obtaining the optimized reward, in which the agent learns the action-value function, Q, of its policy and uses this to improve the policy iteratively[12]. Equation 1 is defined the temporal-difference (TD) error of Q :

$$\delta Q = Q(s_t, a_t) - \left(r_{t+1} + \gamma \max_a Q(s_{t1}, a)\right) \quad (1)$$

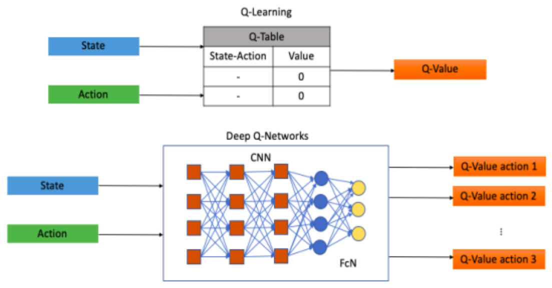


Figure 2. Q-learning and Deep Q-Networks schematics comparison.

s_t, a_t and r_t represent state, action and reward respectively in the time-step t, then the discount factor parameter is denoted by γ that will affect the long-term rewards on the δQ , and the estimated action-value function represents by $Q(s, a)$. Traditionally, the most basic Q-Learning algorithms are based on approximating the value function and updating with TD-error using tabular method. However, if we have high dimensional input with image and high dimensional discrete output of the Q function, therefore, we need to approximate the action-value function using Convolutional Neural Networks with BP for the standard Deep Q-Networks (DQN) that shown in Figure 2.

Algorithm 1 is our baseline performance to compare DQN with GA and the basis for incorporating GA as an optimizer for the NN weight update mechanism. The agent decides and acts by an epsilon greedy policy-based.

```
Algorithm 1 Deep Q-Network with experience replay
Initialize replay buffer $d$ to capacity $n$
Initialize action-value function $Q$ with weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode =1, $m$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\emptyset_1 = \emptyset(s_1)$
    For t =1, T do
        With probability $\mathcal{E}$ choose a random action $a_t$
        Otherwise select $a_t = \operatorname{argmax}_a Q(\emptyset(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_{t,} a_t, x_{t+1}$ and preprocess $\emptyset_{t+1} = \emptyset(s_{t+1})$
        Store transition $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ in $d$
        Sample random minibatch of transitions $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ from $d$
        Set $\gamma_j = \begin{cases} r_j & \text{if episode terminates ate step } j+1 \\ r_j + \gamma \max_{a`} \hat{Q}\left(\emptyset_{j+1}, a`; \theta^-\right) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(\gamma_j - Q(\emptyset_j, a_j, \theta))^2$ with respect to the network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```

Moreover, the agent's previous experience can also be reused through experience replay. Q-value action

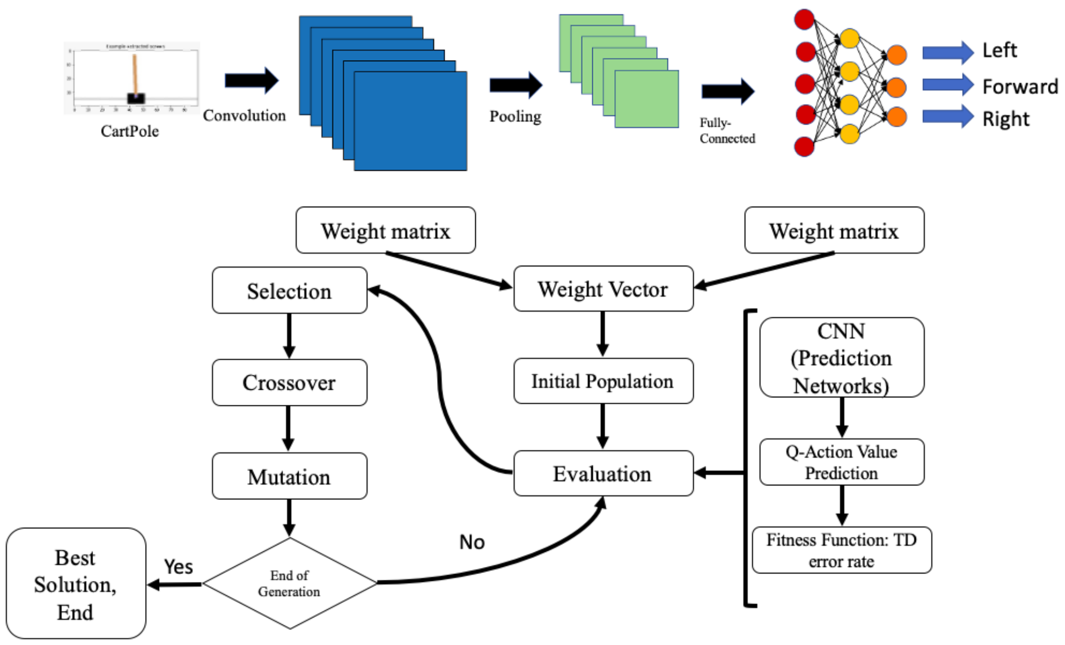


Figure 4. Deep Convolutional Q-Networks with Genetic Algorithm.

approximation in neural networks, especially at lower learning rates, needs several BP processes and optimization steps to get optimal values. Reusing previous experience data, particularly in mini-batches to update the neural network, dramatically aids in converging the Q-network to optimal action values. Figure 3. describes the flowchart of how the conventional Deep Q-Networks used the traditional optimizers to update the weights from the gradient calculation of BP.

3. Deep Q-Networks Trained With Genetics Algorithm

In this part, we will explain how DQN with GA methods can be implemented for reinforcement learning and how it become alternative to the conventional backpropagation methods of CNN training. The Convolutional Neural Network (CNN) has several layers, such as convolution, pooling, and fully connected layers. The weight parameter of the convolution filters and fully connected CNN layers affects the optimal result of Q-value action. Consequently, these weights play an essential role in Q-value accuracy in DQN. During the training phase of the DQN, the weights are updated continuously to achieve a minimum TD-error rate. Therefore, selecting suitable approach to optimize the weight in network architecture has become essential in this research field.

Furthermore, the challenge of applying a genetic algorithm to train a deep convolutional Q-network is interpreting the problem from the field of CNN to the mechanism of genetic algorithms. Figure 4 shows a block diagram of our proposed method and illustrates how we implement and map the problem and the model are explained below:

- *Input layer*: This layer fills the input images and passes them to the convolutional layer. In our scheme, inputs are used for Cartpole balancing. The details will be explained in the experimental results section.
- *Convolutional layer*: This layer connects the convolution method on input images using kernels (filters) to produce the feature map. These filters of weights are initialized randomly, applying either normal or uniform distribution.
- *Pooling layer*: This part is executed after the convolution step. An average pooling function is used to decrease the feature map size. This function computes the mean of each filter of the convolutional layer.
- Activation layer: we used ReLU activation function (with formulation form of $y = \max(0, x)$) to deal with non-linearity in all convolutional and pooling layers.
- *Fully connected layer*: after the convolution and pooling steps, the whole matrix is transformed into a single vector by applying the flattening procedure. This vector is then fed to the fully connected layer of the network as its input neurons.
- The fully connected layer includes an input layer, some hidden layers, and one output layer. The hidden layer resides between the input and output layers and transforms the input data to the output layer. Finally, the output layer is a vector that depends on the action dimension of each agent.

The input of convolutional layers and the design of fully connected layers depend on each agent state (observation) and output dimension of the agent. After building the CNN, the entire method is trained. Training

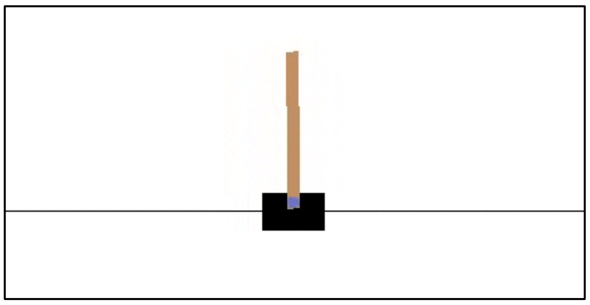
```
Algorithm 2: Deep Q-network with Genetic Algorithm
Initialize replay buffer $d$ to capacity $n$
Initialize replay population size $p$ and maximum number of generation $G$
Generate the random initial weights of the networks
Convert the weight matrix to the vector of initial population;
For 1 to maximum number of generation $G$:
    For each agent in population:
        Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\emptyset_1 = \emptyset(s_1)$
        While true:
            With probability $\mathcal{E}$ choose a random action $a_t$
            Otherwise select $a_t = \text{argmax}_a Q(\emptyset(s_t), a; \theta)$
            Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
            Set $s_{t+1} = s_{t,} a_t, x_{t+1}$ and preprocess $\emptyset_{t+1} = \emptyset(s_{t+1})$
            Store transition $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ in $d$
            Sample random minibatch of transitions $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ from $d$
            Set $\gamma_j = \begin{cases} r_j & \text{if generation terminates at step } j+1 \\ r_j + \gamma \max_{a`} \hat{Q}\left(\emptyset_{j+1}, a`; \theta^-\right) & \text{otherwise} \end{cases}$
    Select individual for new generation (selection)
    Crossover
    Mutation
End
```

refers to updating the weights of the CNN until find the most optimal Q-action value. The optimal output of Q-action value on the CNN highly depends on the weights in all layers. We improve this model using the gradient-free method such as GA. Because applying matrix form makes the computation of the CNN more convenient, all the weights of the network are collected in a matrix for further computation. However, the initial population of the GA is reshaped in one-dimensional vectors. The weight matrix is converted to a vector to be used as the initialization population of the GA. Algorithm 2 summarizes the rest of the algorithm of DQN with GA.

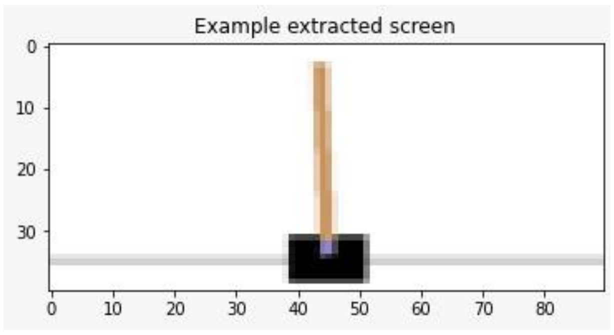
In this paper, we used steady-state selection to choose individual for new generation [13]. To realize the crossover between the selected parents, we used single point crossover and we used random mutation to changes the values to some genes randomly. Technically, we used all the evolutionary function using PyGad [14] and for the deep Q-networks frameworks we used Pytorch.

4. Simulation Results

This section experiment explained how to train DQN with GA agent on the CartPole-v0 problem from the OpenAI Gym [15] as shown in Figure 5(a). The CartPole-V0 has to choose between two moves - moving the cart left or right - so that the pole remains upright. While the agent observes the state of the environment and takes action, the environment shifts to a new position. It returns a reward that shows the results of the action. In this task, the feedback rewards are +1 for each incremental timestep, and the environment stops if the shaft drops over too far or the CartPole shifts more than 2.4 units apart from the middle position. In other words, better-



(a)



(b)

Figure 5. CartPole-V0 environment provided by OpenAI Gym; (a) The original screen of CartPole-V0;(b) The preprocessing results to get the desired position of the CartPole

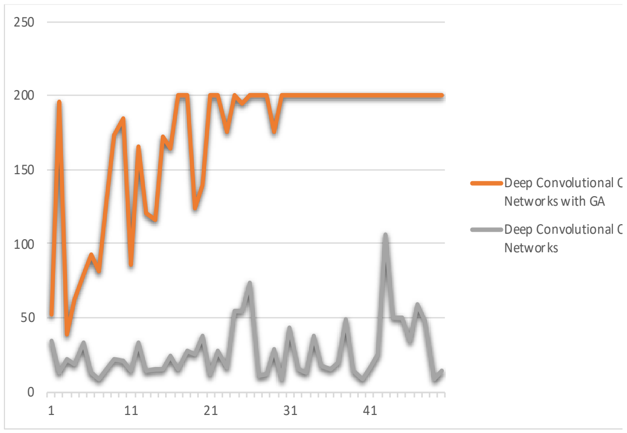


Figure 6. Training Result DQN with GA vs DQN.

performing scenarios will run for a longer duration, accumulating bigger rewards value. Therefore, we compared the conventional DQN to DQN with GA in this section.

The CartPole task is designed so that the inputs to the Multi-Layer Perceptron networks are 4 real values denoting the environment state (position, velocity, etc.). However, CNN can solve the task by looking at the scene, so we used a screen patch centered on the cart as an input of convolutional networks. Then, we use a basic image preprocessing step to reduce the input dimensionality and get a square image centered on a cart as shown in Figure.5(b). Therefore, we get the screen height and width are close to 40x90. We run the

TABLE I. CNN LAYERS SIZE AND HYPERPARAMETERS

Env	Convolution Layers *(in_channels, out_channels, kernel_size, stride)*	FCN Layers (input, hidden, outputs)	Input screen (h x w)	Epsilon Decay	Batch Size	Gamma	Epsilon Start	Epsilon End	Target Update	Mem Replay

Cartpole-v0	Conv1(3, 16, kernel_size=5, stride=2)
	
	Conv2(16, 32, kernel_size=5, stride=2)
	
	Conv3(32, 32, kernel_size=4, stride=2)
		(32 x 32, 512, 2)	40 x 90	200	128	0.999	0.9	0.05	10	10000

simulation training in Intel i7-8750H 2.20GHz, GeForce RTX 2070 Max-Q Mobile Version 16GB of RAM.
The details convolutional layers, fully connected layers and the hyperparameters are defined in Table I. For GA parameter, we used 50 number of generations, 5 number of parents mating, and 10% for probability of mutation parent genes. we used the PyGad Evolutionary library and Pytorch with Python 3.7 version. We can see from Figure 6. that the DQN with GA has shown impressive performance during the training process. Our proposed method suddenly reached a 200 reward score in the initial episode and then drops to below 50. After that, the reward increased with fluctuation progress until 31 episodes. After 31 episodes, the reward score got the constant maximum score by 200. On the other hand, the traditional DQN with experience replay is difficult to reach the maximum reward score. In this method, we used the Stochastic Gradient Descent optimizer to optimize the weight of the CNN. The highest score in this method only reached 123. Furthermore, the DQN with GA can spend shorter computation time process at 22.267 seconds than conventional DQN at 29.66 seconds.

5. CONCLUSION

In this work, we have shown a method to optimize the Q-action value of the deep Q network for the RL problem. We used DQN with GA to find the based solution RGB images input on Cartpole problem. We compared our agent against Traditional DQN with experience replay. Furthermore, we showed that DQN with GA has a better average reward score and can reach the average score faster than the DQN with the experience replay. The optimization method inspires us to optimize the real-world robotics manipulation and navigation in reinforcement learning problem with input pixel in future work.

Acknowledgements

This work was partially supported by JST [Moonshot RnD][Grant Number JP- MJMS2034].

REFERENCES:

[1] S. L. Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," in *2nd Conference on Robot Learning (CoRL)*, 2018, no. CoRL, pp. 6244–6251.

[2] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] S. Iqbal *et al.*, "Directional Semantic Grasping of Real-World Objects: From Simulation to Reality," 2019.

[4] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning," 2017.

[5] Y. Hou, L. Liu, Q. Wei, X. Xu, and C. Chen, "A novel DDPG method with prioritized experience replay," in *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*, 2017.

[6] C. Wilson, A. Riccardi, and E. Minisci, "A novel update mechanism for Q-Networks based on extreme learning machines," *IEEE World Congr. Comput. Intell. 2020*, 2020.

[7] Y. Huang, "Deep Q-networks," in *Deep Reinforcement Learning: Fundamentals, Research and Applications*, 2020.

[8] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Advances in Neural Information Processing Systems*, 2016.

[9] R. Arakawa, S. Kobayashi, Y. Unno, Y. Tsuboi, and S. ichi Maeda, "DQN-TAMER: Human-in-the-loop reinforcement learning with intractable feedback," *arXiv*, 2018.

[10] J. Li, J. Cheng, J. Shi, and F. Huang, "Brief Introduction of Back Propagation (BP) Neural Description of BP Algorithm in Mathematics," *Adv. Comput. Sci. Inf. Eng.*, vol. 2, pp. 553–558,

2012.
[11] R. S. Sutton and A. G. Barto, *Reinforcement Leaning*. 2018.
[12] B. J. A. Kröse, “Robotics and Autonomous Systems Learning from delayed rewards,” *Rob. Auton. Syst.*, vol. 15, pp. 233–235, 1995.
[13] C. W. Ahn, *Advances in Evolutionary Algorithms*, vol. 18. Berlin/Heidelberg: Springer-Verlag, 2006.
[14] A. F. Gad, “PyGAD: An Intuitive Genetic Algorithm Python Library,” no. April 2020, 2021.
[15] A. Rana, “Introduction: Reinforcement Learning with OpenAI Gym,” *Towards Data Science*, 2018.
.