Avrupa Bilim ve Teknoloji Dergisi
Özel Sayı 28, S. 539-544, Kasım 2021

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
Special Issue 28, pp. 539-544, November 2021

Research Article

Setting Reward Function of Sensor Based DDQN Model

Mehmet Gökçay Kabataş [1*], Sevinç İlhan Omurca [2]

[1*] Kocaeli University, Faculty of Engineering, Department of Computer Engineering, Kocaeli, Turkey, (ORCID: *0000-0002-9628-4890*), gkcy.kbts@gmail.com
[2] Kocaeli University, Faculty of Engineering, Department of Computer Engineering, Kocaeli, Turkey, (ORCID: *0000-0003-1214-9235*), *silhan@kocaeli.edu.tr*

(1st International Conference on Applied Engineering and Natural Sciences ICAENS 2021, November 1-3, 2021)

(**DOI:** 10.31590/ejosat.1008702)

ATIF/REFERENCE: Kabataş, M. G., Omurca S. İ., (2021). Setting Reward Function of Sensor Based DDQN Model. *European Journal of Science and Technology*, (28), 539-544.

Abstract

In this study, it is aimed to determine the appropriate reward function of the agent which trained to pass 100 obstacles/objects in Reinforcement Learning (RL) with Double Deep Q Network (DDQN) model. To train the agent, environment is split into sub problems. Several rules and different reward functions defined for the sub problems. A developed mini deep learning library which is called gNet is used for the training.

Keywords: artificial intelligence, deep learning, DDQN, gNet, reward function, sensor based.

Sensör Tabanlı DDQN Modeline Ödül Fonksiyonu Belirleme

Öz

Bu çalışmada DDQN Modeli ile Pekiştirmeli Öğrenme içerisinde 100 engeli/nesneyi geçmek üzere eğitilen ajanın uygun ödül fonksiyonunun belirlenmesi amaçlanmaktadır. Ajanı eğitmek için çevre alt problemlere bölümüştür. Alt problemler için çeşitli kurallar ve farklı ödül fonksiyonları tanımlanmıştır. Eğitim için gNet adında geliştirilmiş mini derin öğrenme kütüphanesi kullanılmıştır.

Anahtar Kelimeler: yapay zeka, derin öğrenme, DDQN, gNet, ödül fonksiyonu, sensor tabanlı

[*] Corresponding Author: gkcy.kbts@gmail.com

1. Introduction

Today, Artificial intelligence (AI) makes it possible for computer to learn from past experience, adjust to new inputs and perform several human-like tasks. The Reinforcement learning [1] is one of the most popular artificial intelligence research areas applied for this purpose. It aims to learn potential good policies for sequential agent behaviours by optimizing a cumulative future reward value.

Setting the reward function effects what the agent will learn. Therefore, knowing of conditions of environment and determining the appropriate reward function have a serious impact on success.

Ratner [2] shows how to set reward function in robot arm which has 7-DOF motion ability to have done required/intended motion and effects of set reward functions in their study. The most important result of Ratner's study is dividing the problem into sub-problems with sub-reward functions. This sub-rewarding approach has contribution to the success of agent.

Hu [3] show effects of arranging reward function. By increasing positive rewards (or only rewards) and decreasing negative rewards (or penalty) makes agent converge more quickly.

In the study, the dividing approach proposed in [2] is used for the rules of the environment and different reward functions are set according to the approach proposed in [3]. Thereby, it is examined how to determine the reward function which is an important factor for Reinforcement Learning.

2. Environment

The environment created in Python with pygame library from scratch. It is 2D game which main purpose is passing over obstacles/objects. In our environment, the agent drives a car which has a constant speed in 5 different actions which are increasing speed, decreasing speed, move left, move right and do nothing.

In Fig. 1, the screen of the game is shown. The mowing agent interacts with the environment as it is seen in the screen. The created objects have random sizes and locations in the screen. This randomness brings a stochastic property to the environment.

There is no moving entity which has an ability to move. The explanations below the screen are as follows:

- "Passed Object" represents the passed objects from beginning of the episode,
- "Reward" represents the result of reward function at the moment,
- "Epsilon" represents the willing of selecting random action of model,
- "Score" represents the summation of rewards from beginning of the episode,
- "Sensors" represents the instantaneous sensor values,
- "Action" represents the selected action,
- "Action Type" shows whether the selected action is random or prediction of the model.

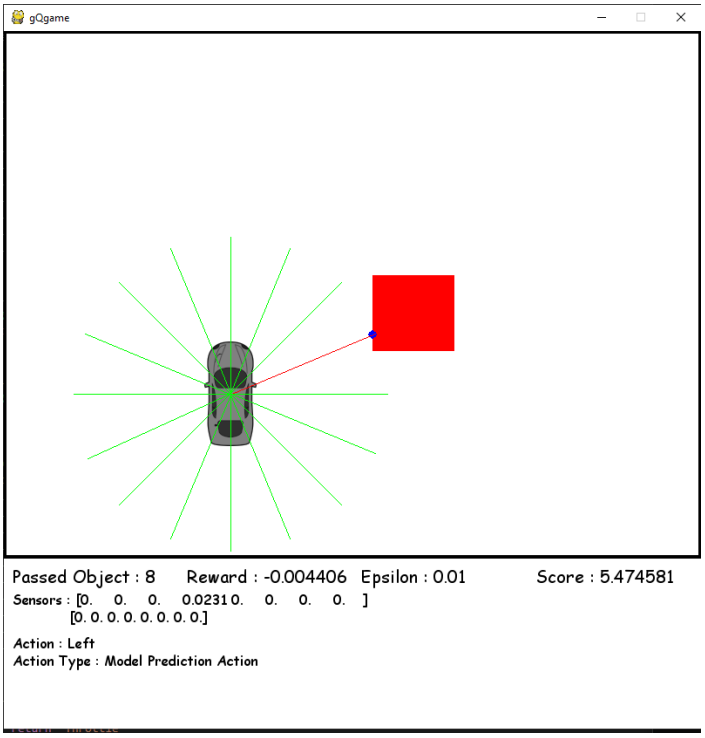

Fig. 1 Screenshot of Environment.

An environment is created to implement RL problems with Double Deep Q Network (DDQN) to decide which action does the agent next. Aim of the study is setting the reward function that can pass over 100 objects with the applied DDQN model. To achieve this, agent should obey some rules. If the agent hit the border of the screen or an object, then agent get penalty and start next episode. If any of the sensors of the agent is turn into active, then agent get penalty depends on how many sensors are turned into active. Purpose of this rule are selecting the action which has minimum risk; because if there is an active sensor, it means that there is a probability of hitting and the agent get penalty. Agent gets reward when it stays alive. Agent gets reward when pass over an object. Yet, there is a constraint to accept passing over object. The constraint is that, object should be perceived by one of the sensors. If constraint has not occurred, then the agent gets meaningless reward. To handle this possibility, that constraint is considered. If agent pass over 100 objects, agent gets the big reward and start next episode. Agent should have minimum speed to move on.

Because of the agent perceiving the environment by its sensors, the environment become partial observable. Also, the actions of the agent are in discrete form.

The agent moves in pixelwise motion thus it doesn't have a dynamic motion model in the environment. If the agent select to move left, instead of turning front of the car, car moves along x axis of screen in rigid form. This situation effects reward function and inputs of the environment. The inputs will be explained in next chapters.

3. Methodology

Main purpose of the RL is reaching maximum reward. To get maximum reward, agent should select appropriate actions and observes the action results.

3.1. Reward Function

Reward functions generally based on Markov Decision Process (MDP) rules. MDP, prosses around Markovian property. The meaning of Markovian property explained as 'next state is depends on current state, not the previous states' in verbally. In mathematical form of explanation shown as $P[S_{t+1}|S_t] = P[S_{t+1}|S_t, \dots, S_1]$. P is probability of changing of state in time t to time t+1.

Founded reward function with this structure of the MDP, convert the $P[S_{t+1}|S_t, a_t]$ probability in form of $R_t = E[R_{t+1}|S_t, a_t]$ reward function. It means calculation of reward which is taken in next state depends on the current state and selected action.

The main purpose is reaching maximum reward. Therefore, collection of reward in the future (G_t) should be also in maximum. Total reward in the future can be calculated as $G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum \gamma_{t+k+1}^{kR}$ where discount factor γ in$(0 < \gamma < 1)$. The idea of the discount factor is there is no guarantee that next reward will be more important than the current one.

Watkins [4] presents Q-Learning (QL) for appropriate action selection by agent. QL means 'Quality Learning'. Agent observe the state $x_n \in X$ in its environment and select action $a_n \in A$. After that, the agent get reward from the environment's reward function $R(x_n, a_n)$.

The environment state change possibility according to $\text{Prob}[y_n = y|x_n, a_n] = P_{x_n y}[a_n]$ rule. Because of the change, agent need to determine optimal policy ($\pi^\star$) to take maximum reward. This policy where agent get rewards even reward discounted by γ factor in time to get maximum reward.

Value of the state x in policy π is calculated by $V^\pi(x) \equiv R_x(\pi(x)) + \gamma \sum_y P_{xy}[\pi(x)] V^\pi(y)$ In Dynamic Programming (DP) [5] theory, optimal policy $\pi^\star$ should be found. With this policy, optimal value $V^\star \equiv V^{\pi^*}(x) = \max_a \{R_x(a) + \gamma \sum_y P_{xy}[a] V^{\pi^*}(y)\}$ can be found. This equation called Belmann Equation. Yet, in the equation, assumed R_x and $P_{xy}[a]$ are known. The QL classified as an incremental DP problem; therefore, Q-value or action-value become in the form of $Q^\pi(x, a) = R(a) + \gamma \sum P_{xy}[a] V^\pi(y)$.

3.2. Deep Q Network

Mnih [6] combine QL and Deep Learning (DL) approaches. Their study effects in the Deep Reinforcement Learning (DRL) which using RL with deep models and show its success. Network structure which they establish in study called Deep Q Network (DQN).

To find optimal Q-value, Belmann Equation still used. There is an assumption for the Belmann Equation. If optimal value $Q^\star(s', a')$ known at next state(s'), optimal action (a')should be selected. Therefore, expected value of $r + \gamma Q^\star(s', a')$ become maximum. This makes $Q^\star(s, a) = E_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^\star(s', a')|s, a]$ equality. This equality is theoretically effective, but practically impractical because it can not to generalize. To calculate Q-value, DL model is used. In mathematical form $Q(s, a; \theta) \approx Q^\star(s, a)$. After that, DL model become function approximator of Q value.

θ is parameters of Q Network. To update these parameters, loss function is selected as

$$L_i(\theta_i) = E_{s,a \sim \rho(.)}\left[\left(y_i - Q(s, a; \theta_i)\right)^2\right]$$
$$y_i = E_{s' \sim \varepsilon}[r + \gamma max_{a'} Q(s', a'; \theta_{i-1})|s, a]$$

and tried minimize at every iteration. $P(s, a)$ is a probability distribution over environment ε and depending variables of s and a. If DL model parameters updates at every iteration, model became QL which present by Watkins.

Experience Replay Mechanism [7] is used for decrease correlation between states which train the model. By using Experience Replay Mechanism, DL model become general function approximator. Mechanism works as putting actions and states at the past into buffer memory which also called Replay Buffer, and take some random samples (or batches) from buffer during training.

The most critical point in DQL algorithm is calculation of loss function L_i. y_i is a label which also used in supervised learning. This label calculated by Q-value function. To calcualte Q-value function, reward is used. Therefore, appropriate reward function and appropriate DQN model is effects success of agent.

3.3. Double Deep Q Network

Van Hasselt [8] present a developed version of DQN. Instead of using one Q Network, Van Hasselt suggest two Q Networks. This approach called Double Deep Q Network (DDQN). In the structure of DDQN, there is twin Q Networks. Algorithm is very similar to DQN algorithm, and the difference are calculation of y_i and equalize parameters of twin networks at some points.

y_i function for DDQN in form of

$$y_i = E_{s_{i+1} \sim \varepsilon}\left[r_{i+1} + \gamma Q^T\left(s_{i+1}, argmax_{a'} Q^E(s_{i+1}, a')\right)\right].$$

Q^E and Q^T represents evaluation network and target network respectively. Training is provided through the evaluation network. At some points, the evaluation network's parameters transferred to the target network. With this structure, state evaluated by the evaluation network and its value calculated by the target network. These twin networks makes more stabilized policy development. Therefore, DDQN selected in this study.

In two Q Network structure, there is an $\epsilon - \text{greedy}$. ϵ is an probability of exploration of agent. Generally, it starts with higher values such as 1, then decrease. This helps the agent explore the environment at the beginning of the training. By decreasing the ϵ during training, agent starts deciding action by using its policy.

4. Experiments

In the experiments, Python selected as programming language, NumPy used as a linear algebraic operation library and

Algoritma 1 DQN with Experience Replay Mechanism

Initialize D memory with N capacity
Initialize Q value function (model) with random weights
for $episode = 1, M$ **do**
  s_1 state initialization
  for $t = 1, T$ **do**
    Action selection : random action a_t with ϵ probability or $a_t = max_a Q(s_t, a; \theta)$ calculation
    Apply a_t action in the enviorenment then get r_t, s_{t+1} ve terminal (boolean) values
    Put $(s_t, a_t, r_t, s_{t+1}, terminal_t)$ values into memory D
    Get random sample (batch) from memory D as $(s_j, a_j, r_j, s_{j+1}, terminal_j)$
    $y_j = r_j + \gamma max_{a'} Q(s_{j+1}, a'; \theta) * (1 - terminal_j)$
    $L_j = (y_j - Q(s_j, a_j; \theta))^2$ train network with loss function
    $s_t = s_{t+1}$
    $\epsilon = \beta\epsilon$, update ϵ with β
  end for
end for

Algoritma 2 DDQN with Experience Replay Mechanism

Initialize D memory with N capacity
Initialize Evaluation Network Q^E with random weights
Initialize Target Network as $Q^T(\theta) = Q^E(\theta)$
C, target network equalization counter
for $episode = 1, M$ **do**
  s_1 state initialization
  for $t = 1, T$ **do**
    Action selection : random action a_t with ϵ probability or $a_t = argmax Q^E(s_t, a)$ calculation
    Apply a_t action in the enviorenment then get r_t, s_{t+1} ve terminal (boolean) values
    Put $(s_t, a_t, r_t, s_{t+1}, terminal_t)$ values into memory D
    Get random sample (batch) from memory D as $(s_j, a_j, r_j, s_{j+1}, terminal_j)$
    $y_j = r_j + \gamma Q^T(s_{j+1}, argmax_{a'} Q^E(s_{t+1}, a')) * (1 - terminal_j)$
    $L_j = (y_j - Q^E(s_j, a_j))^2$ train network with loss function
    $s_t = s_{t+1}$
    $\epsilon = \beta\epsilon$, update ϵ with β
    if $mod(T, C) == 0$ **then**
      $Q^T(\theta) = Q^E(\theta)$
    end if
  end for
end for

the whole mathematical applications applied with gNet [9] which is a CPU based mini deep learning library developed by the author.

The parameters selected during the application can be listed as follows. Mini batch size is 64, learning rate is 5e-6, initial ϵ is 1, β is 0.996, ϵ_{min} is 0.01, γ is 0.99, N is 1e6. DL model for DDQN structure as follow.

Layer	Neuron Number
Flatten	
Dense: ReLU	256
Dense: ReLU	256
Dense: None	5

Output layer has 5 neuron which is equal to number of possible actions. Accepted input by model is combination of 16 sensor data and 1 normalized passed object number. This combination makes an input vector which has 17 elements in it. 16 sensor data are generated when the agent detects the object or environment boundaries/border and generates values depending on the distance between [0,1]. To these data, $\mathcal{N}(0,1)$ noise which represents faulty measurement also added. Also, there is a 15% possibility of adding $\mathcal{N}(0.01,1)$ noise which represents malfunction of sensors. With these noises, possibility of faulty sensor measurement and mulfunctionality can modelled.

For loss function in DDQN, "Mean Square Error (mse)" which built-in in gNet selected. Adam [10] selected as optimizer.

4.1. Setting Reward Function

Reward function depends on the environment rules and they are divided into sub-problems. There are two reward functions.

R_1 is standard reward function. The rewards and penalties are defined as:

- If agent hit boundaries of the environment or objects, agent gets 1 penalty points and start next episode,
- If agent's sensors are become active, agent gets $\frac{0.1}{\#act\_sen}$ penalty points,
- Agent gets 0.01 reward points for each moment of being alive,
- Agent gets 1 reward points for passing an object,
- If agent pass over 100 objects, gets 100 reward points and start next episode.

R_2 reward function is application of Zijian et al. approach which increase rewards and decrease penalty. The rewards and penalties are defined as:

- If agent hit boundaries of the environment or objects, agent gets $\left(1-\frac{\#pass_{obj}}{100}\right)$ penalty points and start next episode,
- If agent's sensors are become active, agent gets $\frac{0.1}{\#act_{sen}} * \sqrt{\sum sen_{val}}$ penalty points,
- Agent gets $0.01 * \frac{(100+\#pass_{obj})}{100}$ reward points for each moment of being alive,
- Agent gets $\frac{\#pass_{obj}}{100}$ reward points for passing an object,
- If agent pass over 100 objects, gets 100 reward points and start next episode.

$\#act\_sen$, $\#pass_{obj}$ and sen_{val} represents active sensor number, passes object number and sensor values respectively. In this reward function, increasing rewards and decreasing penalty during time by using $\#pass_{obj}$ and makes them dynamic.

5. Results and Discussion

In this study, the agent's environment is created from scratch and the agent is put into the environment to learn under required/intended conditions then understand how to set appropriate reward function. Also, understand the effects of reward function. Therefore, the effect of R_1 and R_2 reward functions on success are examined over 4 different variables.

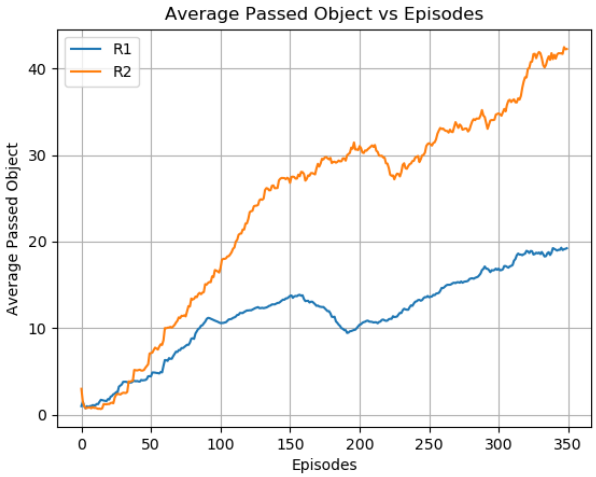

Fig. 2Average Passed Object.

Fig. 2 shows average passed objects with respect to episodes. The orange curve represents R_2 reward function, and it shows how effects increasing the rewards and decreasing the penalty for agent performance. General trend of two curves are increasing over episodes; yet, agent with R_2 reward function has better performance. In the study, the training process has 350 episodes. If the episodes are increased, then there is a probability of having state-of-art performance. The state-of-art performance is defined as 99 average passed objects. Here, average means that the value calculated in the last 100 episodes.

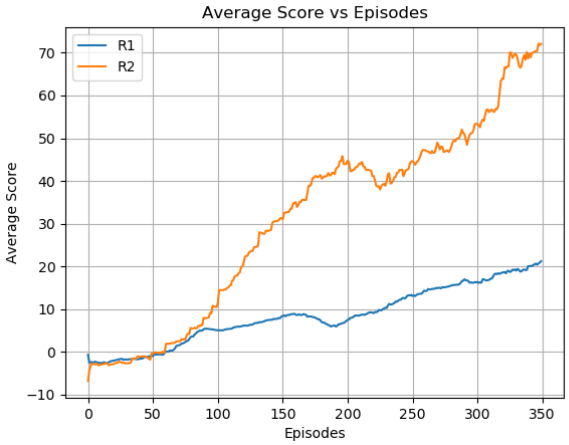

Fig. 3Average Score.

Fig. 3 shows how the obtained scores change during the episodes. Here, the score defined the total reward of the desired episode. The orange curve represents the R_2 reward function. General trend of two curves are increasing over episodes; yet, agent with R_2 reward function has better performance.

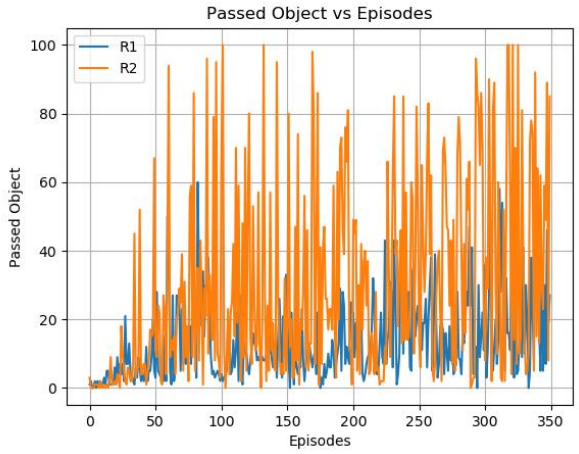

Fig. 4 Passed Object for Each Episode.

Fig. 4 shows how many object the agent passed in each episode. The orange curve represents R_2 reward function. At the initial episodes, it was observed that both R_1 and R_2 curves are overlapped and then diverged over time. Also, agent with R_2 reward function reach to big reward in most of the time.

Fig. 5 shows the obtained scores for each episode. The orange curve represents R_2 reward function. In the beginning, it was observed that both lines overlapped and diverged over time. Also, agent with R_2 reward function has a lot score most of the times.

6. Conclusion

Consequently, it is detected that, differences in reward functions has serious impact on training. And also, setting appropriate reward function which is sensitive and challenging task is very important. In this study, it is concluded that dividing sub-problems and arrange reward function by increased reward and decreasing penalty is effective on the performance of agent. As a future work, we intend to train a vehicle which has RL model with dynamic motion model. In this way, we aim to get one step closer to the solution of real problems with the vehicle with

dynamic motion model. In addition, it is also aimed to find a general solution by making the environment more stochastic.

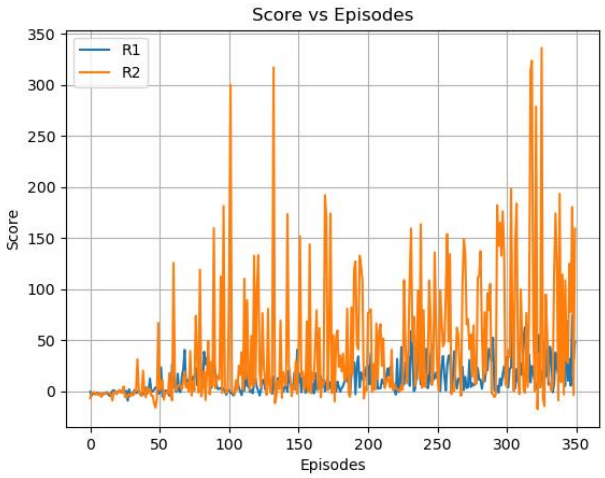

Fig. 5 Scores for Each Episode.

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, MIT Press, 1998.

[2] E. Ratner, D. Hadfield-Menell and A. D. Dragan, "Simplifying Reward Design through Divide-and-Conquer," *CoRR*, vol. abs/1806.02501, 2018, *[Online]* http://arxiv.org/abs/1806.02501.

[3] Z. Hu, K. Wan, X. Gao, and Y. Zhai, "A Dynamic Adjusting Reward Function Method for Deep Reinforcement Learning with Adjustable Parameters," *Mathematical Problems in Engineering*, vol. 2019, pp. 1-10, DOI: 10.1155/2019/7619483.

[4] C. J. C. H. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, 1992, pp. 279-292.

[5] R. E. Bellmann and S. E. Dreyfus, *Applied Dynamic Programming*, Princeton, NJ, USA: Princeton University Press, 1962.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5602, 2013, *[Online]* https://arxiv.org/abs/1312.5602

[7] L. Lin, "Reinforcement Learning for Robots Using Neural Networks," Ph.D. dissertation, School of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, USA, 1993.

[8] H. van Hasselt, A. Guez, D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 30, No.1, 2016, *[Online]* https://arxiv.org/abs/1509.06461

[9] gNet, *Avalaible:* https://github.com/MGokcayK/gNet.

[10] D. P. Kingma, J. Ba, (2014, 12), Adam: A Method for Stochastic Optimization in *International Conference on Learning Representations*, *[Online]* https://arxiv.org/abs/1412.6980.