A post-training quantization method applied to the game-environment-based Dueling-DQN model

Chenghao Yang

University of Nottingham Ningbo China, 199 Taikang East Road Ningbo, 315100, China

hnycy2@nottingham.edu.cn

Abstract. Reinforcement learning is a common method that focuses on the best behaviors that agents can get the highest rewards for in one particular environment. It was recently combined with deep learning to improve information perception. However, the traditional deep reinforcement model DQN is considered to be as computationally intensive as its layer deplumation. As a result, this paper proposes to include a post-training quantization method in an improved version of DQN to see if it is worthwhile to include. After some experiments, the outputs exactly prove that post-training quantization can deduce the computation with a tiny precision loss.

Keyword: deep reinforcement leaning, network, compression, game.

1. Introduction

In recent years, reinforcement learning (RL) has received much attention in the game field, which is now an established research field with multiple conferences and dedicated journals. Nonetheless, these works depend on massive networks with numerous of parameters. With limited resources, the reduction of storage and computational costs becomes critical. Thus, how to compress the models is worthy of study.

In general, RL is a branch of machine learning aiming to describe and resolve the problem of behaviour learning for an agent in a dynamic environment through trial-and-error interactions as Figure 1 shows [1]. Basically, RL includes two types: one is model-based RL, which attempts to model the environment and then, based on that model, choose the most appropriate policy, or Q value. The other is model-free RL, which attempts to learn the optimal policy or Q value in one step. Specifically, model-free RL can be classified based on value or policy. Value-based methods optimize the value-action function to get the best policy, while policy-based methods optimize the policy. Also, these two methods can be combined into a composite method called the actor-critic (AC) method.

Regarding the network compression methods, their objective is to compress models in deep networks without significantly reducing model performance. According to their different properties, they can be divided into four types [2]. The first one is low-rank factorization, which uses the matrix to compose informative parameters but is hard to imply due to its expensive computational operation. The second one is compact convolutional filters, which design special convolutional filters to save parameters while performing badly in deep networks. The third one is knowledge distillation, which trains a compact model to distill knowledge from a large model while a large model can only satisfy tasks with the SoftMax loss function. The last one is quantization, which reduces the number of bits. It is easy to implement and can result in a conspicuous speedup with minimal accuracy loss.

Our paper will focus on demonstrating the high efficiency of quantization on Dueling-DQN. We can prove whether our assumption is correct by comparing rewards between int8 post training quantization, fp16 post training quantization, and int8 quantization aware training in different DQN architectures.

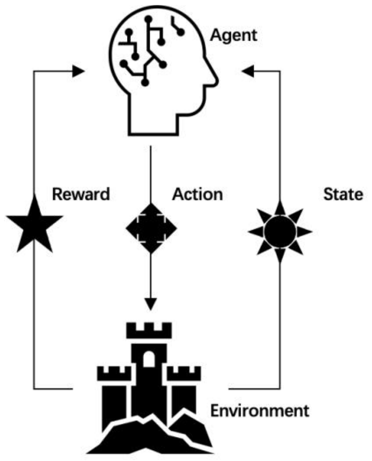

Figure 1. reinforcement Learning [12].

2. Related work

Nowadays, there are two types of RL: model-based RL and model-free RL. Model-based RL focuses on environment dynamics, which means to first study the environment by sampling, then use the learned data to model the environment, and finally optimize the policy of RL based on the model. However, in reality, agents can rarely get the true model of the environment, which leads to unpredictable performance errors of agents between the modelled environment and the true environment [3]. The other type is model-free RL, which does not use the environment to find the best rewards while optimizing the policy or Q value. In this context, Q value denotes the quality of a state–action combination, and policy denotes How does an agent select an action in different states? To optimize the model-free RL, convolutional neural networks have been imported, which help the model-free RL develop into two categories: Q value-based RL and policy-based RL. The initial Q value-based RL is DQN, the combination of the convolutional neural network and the Q-learning algorithm in traditional RL. As Watkins and Dayan claimed [4], Q-learning is a method that evaluates the Q Value of each action and then calculates the optimal policy based on the Q Value. However, it absolutely relies on the states, actions, and Q value in the past, which infers that Q-learning cannot handle undetermined states. With deep learning's significant success in the supervision of learning area, using a convolutional neural network to fit the Q value could be a necessary solution to give Q-learning prediction ability. Regarding policy-based RL, it is a direct way to optimize the policy, which is based on the policy gradient (PG) method. This method trains the agent to take actions with higher rewards using the supervised study idea, in which rewards are reflections of whether actions are good or bad [5]. However, PG converges on a local maximum rather than on the global optimum, which means it requires a long time for the training process. Additionally, A3C is one method and is the complexation of value-based and policy-based, which first optimizes the policy and takes one action, then evaluates the value function to define whether the action should be taken [6]. However, poor concurrency and difficulty imply that A3C has a high threshold.

In the introduction part, there are also four types of network compression methods. Firstly, low-rank factorization uses matrix/tensor decomposition to estimate the informative parameters, which can improve the compression rate as a speedup. For instance, Lu and Yang mentioned that some simple

DNN models which apply with low-rank factorization can consequently accomplish a 2-fold speedup with a 1% reduction in classification accuracy [7]. However, the implementation of this type of method is not that easy since it involves decomposition operations, which are computationally expensive. Another problem is that, in order to obtain convergence, factorization necessitates extensive model retraining. The second type is compact convolutional filters, as Lawhern et al [8]. proposed, which are a special structure based on compact convolutional filters designed to reduce memory and computation. Research suggests that this method can reduce the number of parameters while maintaining a high classification accuracy. However, in thin or deep architecture, this method performs terribly, and in some situations, it has a significant effect on the learning, making the results unstable. The third type is knowledge distillation, discussed by Hinton, Vinyals, and Dean [9]. It seeks to reduce deep and large networks to smaller ones., and then the compressed model will imitate the function learned by the complex model. Knowledge distillation can make deeper models shallower and help conspicuously reduce computational costs; however, without the SoftMax loss function in tasks, knowledge distillation cannot be applied. The fourth type is quantization, as concluded by Gholami et al. [10], which reduces the number of bits required to represent each weight. Specifically, based on the different functions for quantization, quantization can be divided into two categories: uniform quantization and ununiform quantization. They can both represent 32-bit floating-point numbers as low-precision (e.g., 4-bit or 8-bit) fixed-point numbers. Nonetheless, ununiform quantization can be difficult to implement in mainstream hardware such as GPU or CPU.

3. Combination of Dueling-DQN and post-training quantization

This paper proposes a method combining the training process of Dueling-DQN with post-training quantization for readers unfamiliar with Dueling-DQN and quantization. Section 3.1 gives a brief introduction to reinforcement learning. In Section 3.2, we introduce the deep reinforcement architecture of Dueling-DQN. Section 3.3 presents the training section of Dueling-DQN. In 3.4, we discuss the various quantization methods available.

3.1. Reinforcement learning

Reinforcement is a branch of machine learning that aims to reveal the solution for agents, the entities that run the reinforcement learning algorithm, to execute the action for reaching the goal and maximizing the return in a single environment. For instance, in a video game called Flappy Bird, we needed simple clicks to control the bird, evade various pipes, and fly as far as possible, because the further we flew, the higher the bonus points. Under this circumstance, the agent is the bird, the target is the longest possible fly distance, the environment is the evasion from pipes, the action is the click to command the bird to fly a little bit, and the response is the points. While there are various types of RL, the paper employs Q-value-based RL to collaborate with the quantization method. Initially, this type of RL will have an empty Q-table for looking up the actions and states and scoring the expected future reward. Afterward, we will select an action based on Epsilon's greedy strategy and perform the action. Finally, we use the temporal difference approach to update Q based on states and action based on Q-function:

$$New\ Q(s,a) = Q(s,a) + \ \alpha[R(s,a) + \gamma maxQ^{'}(s^{'},a^{'}) - Q(s.a)] \tag{1}$$

Here α is the learning rate, γ is the discount rate, and $maxQ^{'}(s^{'},a^{'})$ are the maximum expected future rewards given the new state(s') and all possible actions at the new state. By repeating these steps, the Q value will update, and we can find the most suitable action based on the Q value.

3.2. Deep reinforcement learning architecture

With the development of deep learning, it is a common idea to apply deep learning to reinforcement learning. A typical example of deep reinforcement is DQN, which was published in Nature in 2015. The experiment shows that DQN can perform better than other algorithms in most games. While DQN is a commonly used architecture now, it has several drawbacks, such as speed. Dueling-DQN is a

DQN follow-up improvement that adds a dueling network to DQN. Instead of outputting the Q value for |A| number of actions, this new network will separate the features collected from the convolutional network into two steams, representing the state-value function and the advantage function. Consequently, the Q value is represented by these two streams:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \tag{2}$$

Where $V(s; \theta, \beta)$ is the state-value function, and $A(s, a; \theta, \alpha)$ is the advantage function. Additionally, θ is the parameter of the convolution network, and a and β are parameters of the fully connected layer with two streams, respectively. Yet, this formula may be unidentifiable sometimes, so for practical use, we can force the sum of the output vectors of the advantage function to be set to 0, and the Q value is represented as:

$$Q(s, a; \theta, a, \beta) = (s; \theta, \beta) + A(s, a; \theta, \alpha) - \Sigma_{a'} A(s, a'; \theta, \alpha) \tag{3}$$

In Figure 2, we can see the complete architecture, which begins with the common CNN input filters and flatten layers. After the common parts, the network bifurcates into separately, densely connected layers. The state-action function is represented by a single output from the first densely connected layer. The second densely connected layer produces n outputs representing n actions, which are the expressions of the advantage function. These value-state and advantage functions are then aggregated in the Aggregation layer and finally generate Q values based on the formula as mentioned. The agent can identify the correct action to take faster during policy evaluation.

3.3. *Training process*

The previous content discussed some drawbacks of DQN. Thus, in Dueling-DQN, the training process is the same as one of the improved versions of DQN called Double DQN (DDQN) to avoid overestimation [11]. The training process is shown in Figure 3, which includes two new mechanisms: a loss function and quantization. We will mainly loop through six steps in the whole process. We begin by resetting the environment and initializing the state Tensor. Then, we sample an action, execute it with quantized nets, observe the next screen and the reward (always 1), and optimize our model once. When the episode ends, we restart the loop.

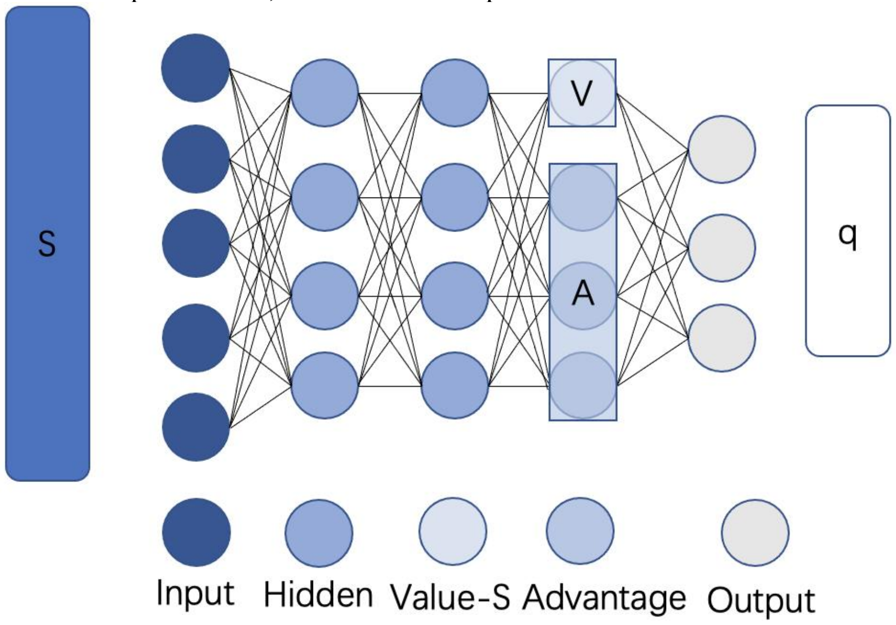

Figure 2. The architecture of Dueling-DQN (recreated from [13]).

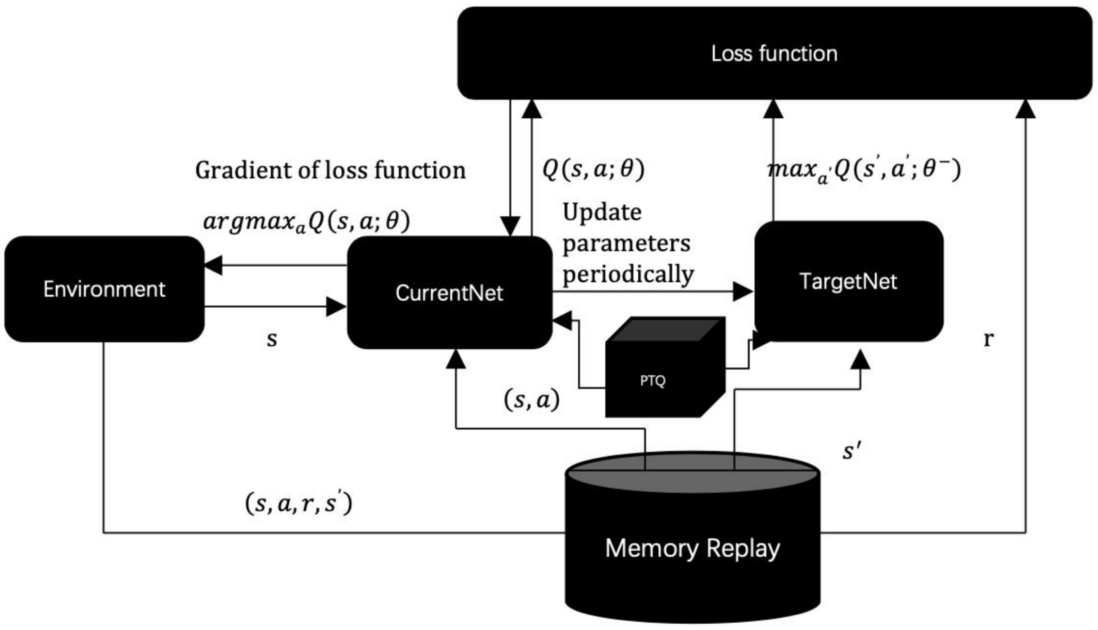

Figure 3. The training processes of DQN[14].

3.3.1. Two new mechanisms. Two new mechanisms are applied. The first mechanism is the TargetNet, which uses an older set of parameters than the CurrentNet. The CurrentNet is updated at each step of training, while the parameters of the TargetNet are synchronized with the training network only once every C steps. Specifically, $Q(s,a;\theta_t)$ is the output of CurrentNet, and $Y_t = R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta_t')$ c a represents the target function of TargetNet. Doing so makes the target network more stable relative to the training network and results in less overestimation. The second one is prioritized experiment replay, which is built on top of experiment replay. The old replay method maintains a playback buffer and stores the four-tuple data (state, action, reward, and next state) sampled from the environment each time into the playback buffer. It then randomly samples some data from the playback buffer for training the Q network. The improved method uses experience markers to make replay more efficient than simply randomly sampled replay, and the experiment shows that this mechanism can enhance the efficiency of sampling while maintaining the independence of selection.

3.3.2. Loss function. We can update network parameters by minimizing the mean square error between Q values in CurrentNet and TargetNet. The loss function could be: $L(\theta_t) = E_{s,a,r,s'}[(Y_t - Q(s,a;\theta_t))^2$. We use partial derivative to the loss function to get the gradient about parameter θ.

3.4. Quantization

The process of converting a model into an equivalent representation is known as quantization. It uses parameters and computations with lower precision and can squeeze a small range of floating-point values into a fixed number of information buckets, as Figure 4 shows. Thus, prepared quantization can significantly decrease the computation and memory required for Dueling DQN. Depending on the position of quantization, there are two ways of quantization we need to consider. One conversion technique is post-training quantization. It can quantize an already-trained model by replacing the weights of a neural network with quantized ones, which will reduce the model's size while also improving CPU and hardware accelerator latency. With development, there are different sizes of post-training quantization, e.g., 8-bit post-training quantization, and fp16 post-training quantization.

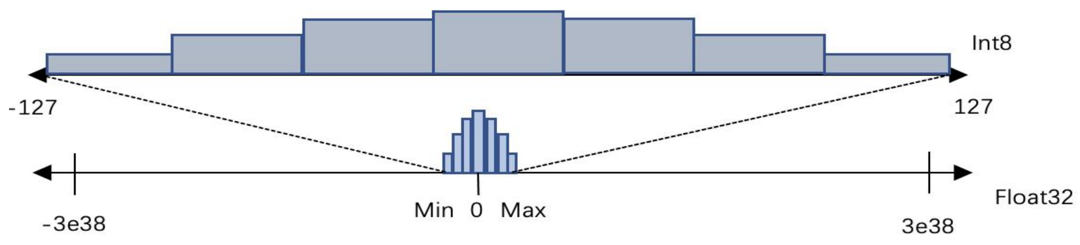

Figure 4. How Post-training quantization works [15].

4. Experiments

In the following, we evaluate the quantization method with the Dueling-DQN architecture

4.1. Datasets(scores)s

We conduct the experiments in two environments provided by OpenAi Gym, allowing us to directly compare the outcomes of various variations. The first one is called LunarLander, as Figure 5 shows. In this environment, our goal is to guide a rocket from the top of the screen to the marked area at the bottom of the screen. To control the rocket, there are four available actions: do nothing, fire the left orientation engine, fire the main engine, and fire the right orientation engine. The reward for landing on the landing pad and coming to rest is about 100-140 points. The lander forfeits its prize if f the lander moves away from the landing pad. The landing craft loses an additional 100 points if it crashes. It gains an additional 100 points if it stops. Each leg with ground contact is worth 10 points. The main engine costs -0.3 points per frame to fire. the side engine is -0.03 points per frame. Solved is worth 200 points.

To avoid the contingency, we are also including another environment with a different mechanism named MountainCar, as Figure 6 shows. In this environment, a car is randomly positioned at the bottom of a sinusoidal valley. The only possible action for this car is the acceleration that can be applied in either orientation. The environment's objective is to strategically accelerate the vehicle to the target state at the top of the mountain. So the agent is penalized with a reward of -1 for each timestep.

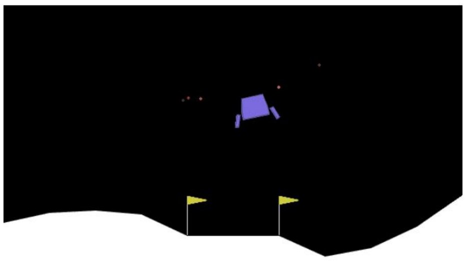

Figure 5. LunarLander[16].

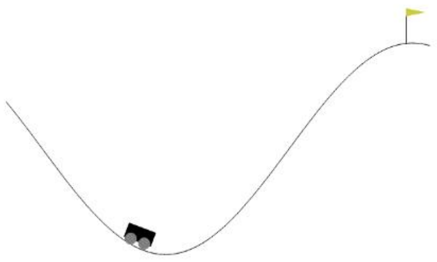

Figure 6. MountainCar[17].

4.2. Experiment settings

Policy: We use a modified policy model with a 64-size neural network, a multi-layer perceptron as feature extraction, and a relu function as activation function.

learning rate and batch size: 0.0001 and 50000, respectively, as the default settings of the package.

4.3. baseline

For composing, we evaluate two post-training quantization method groups and one non-quantization group in two environments separately. The former is referred to as 8-bit post-training quantization, while the latter is referred to as fp16 post-training quantization.

4.4. analysis

As shown in Tables 1 and 2, in both environments, the different DQN models based on different quantization methods are used to train the agents to obtain higher rewards. Because the batch size is large enough, the final rewards of Dueling-DQN, DDQN, and DQN are the same if the quantization methods are the same. However, the quantization methods significantly decrease the parameter sizes in the neural network; also, the mean rewards for the quantized environment are only slightly lower than the original environment. In Table 1, the accuracy of the Int8 PTQ groups is 96% compared to the original groups, and the Int16 PTQ groups are 98%. Besides, in Table 2, the accuracy of Int8 PTQ groups is 97.5% compared to the original groups, and the accuracy of Int16 PTQ groups is 99.9%. Thus, this experiment proves the idea that the quantization of Dueling-DQN can enormously decrease memory utilization while maintaining high precision as figure 7 presents.

Table 1. Comparison results on bitwise, time, rewards in LunarLander.

Method	Quantization Bitwise	Parameter size	Score reward
DuelingDQN	Int8 PTQ	512	264.55
DuelingDQN	FP16 PTQ	1024	269.33
DuelingDQN	NO	2048	274.21
DDQN	Int8 PTQ	512	264.55
DDQN	FP16 PTQ	1024	269.33
DDQN	NO	2048	274.21
DQN	Int8 PTQ	512	264.55
DQN	FP16 PTQ	1024	269.33
DQN	NO	2048	274.21

Table 2. Comparison results on bitwise, time, rewards in MountainCar.

Method	Quantization Bitwise	Parameter size	Score reward
DuelingDQN	Int8 PTQ	128	131.73
DuelingDQN	FP16 PTQ	256	134.92
DuelingDQN	NO	512	134.98
DDQN	Int8 PTQ	128	131.73
DDQN	FP16 PTQ	256	134.92
DDQN	NO	512	134.98
DQN	Int8 PTQ	128	131.73
DQN	FP16 PTQ	256	134.92
DQN	NO	512	134.98

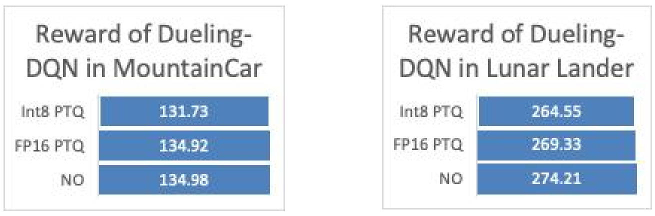

Figure 7. High precision of quantization.

5. Conclusion

While previous DQN applications in game environments struggled with low speed and high computation costs, this paper presents a suitable approach to improve efficiency. By applying a bit's quantization to the DQN model before training, the parameters of both LunarLander and MountainCar can be reduced to 1/2 or even 1/4, which consequently leads to a novel enhancement in efficiency compared to the original DQN model. In the future, we may explore the possibility of applying the quantization method to other deep reinforcement models, such as A3C or PPO2.

References

[1] Kaelbling, L. P., Littman, M. L., & Moore, A. W. 1996 Reinforcement learning: A survey. Journal of artificial intelligence research, 4, 237-285.

[2] Cheng, Y., Wang, D., Zhou, P., & Zhang, T. 2017 A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282.

[3] Moerland, T. M., Broekens, J., & Jonker, C. M. 2018 Emotion in reinforcement learning agents and robots: a survey. Machine Learning, 107(2), 443-480.

[4] Watkins, C. J., & Dayan, P. 1992 Q-learning. Machine learning, 8(3), 279-292.

[5] Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. 1999 Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12.

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. 2015 Human-level control through deep reinforcement learning. nature, 518(7540), 529-533, 2015.

[7] Lu, Y., & Yang, J. 2015 Notes on Low-rank Matrix Factorization. arXiv preprint arXiv:1507.00333.

[8] Lawhern, V. J., Solon, A. J., Waytowich, N. R., Gordon, S. M., Hung, C. P., & Lance, B. J. 2018 EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces. Journal of neural engineering, 15(5), 056013.

[9] Hinton, G., Vinyals, O., & Dean, J. 2015 Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2(7).

[10] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. 2021 A survey of quantization methods for efficient neural network inference. arXiv preprint arXiv:2103.13630.

[11] Van Hasselt, H., Guez, A., & Silver, D. 2016 Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, Vol. 30, No. 1.

[12] Mahoney C. 2021 Reinforcement Learning: A Review of Historic, Modern, and Historic. Developments Towards Data Science Available from: https://towardsdatascience.com/reinforcement-learning-fda8ff535bb6

[13] Jiang, F., Ma, R., Gao, Y., & Gu, Z. 2021 A reinforcement learning-based computing. offloading and resource allocation scheme in F-RAN. EURASIP Advances in Signal Processing, 91 (2021). https://doi.org/10.1186/s13634-021-00802-x

[14] Cai Q, Cui C, Xiong Y, et al. 2017 A Survey on Deep Reinforcement Learning for Data. Processing and Analytics. arXiv preprint arXiv:2108.04526, 2021.

[15] Tensorflow.org 2020 Quantization Aware Training with TensorFlow Model. Optimization Toolkit - Performance with Accuracy. Available from: https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html

[16] Lunar Lander - Gym Documentation 2022 Gymlibrary.dev. Available at: https://www.gymlibrary.dev/environments/box2d/lunar_lander/

[17] Mountain Car - Gym Documentation 2022 Gymlibrary.dev. Available at: https://www.gymlibrary.dev/environments/ classic_control/mountain_car/