Empirical Analysis of Data Preprocessing and Reward Shaping for Deep Q-learning in Playing Atari Game

Jeremy Winston, Dae-Ki Kang*

Department of Computer Engineering, Dongseo University, Busan, South Korea

Received 15 June 2023; received in revised form 05 August 2023; accepted 10 September 2023

Abstract

Learning a policy directly from high-dimensional input, like a video game, is very challenging. Since the frames of the game act as a state, we can imagine that the data can be noisy and incomplete (model-free). Training the agent with this kind of data may not be effective and even can lead to complete failure. In this paper, we present our study to tackle this issue using different formats of (1) data preprocessing and (2) reward shaping. Data preprocessing addresses the problem by removing noise and redundant information as much as possible while reward shaping addresses the problem by providing an intermediate reward (reward shaping) to aid the agent in the vast environment. From the result of our experiment, we demonstrate how we can learn a suitable policy by properly preprocessing the data and adjusting the rewards. We also demonstrate a scenario in which faulty data preparation and reward structuring caused the training to fail.

Keywords: Reinforcement learning, Deep Q-learning, Data preprocessing, Reward shaping, Atari game

1. Introduction

Reinforcement learning research frequently employs video games as a training environment. This practice has grown in popularity [1-2] due to the availability of numerous libraries that supply the structure. This ready-to-use environment offers a range of challenges, even for human players, from the simple to the extremely difficult. This framework is ideally suited to train various Reinforcement Learning algorithms.

The objective of a reinforcement learning agent is to, given an environment, determine a course of action that will maximize the expected cumulative reward. This implies that the agent must learn what action to perform for each state in the environment. In the case of the Atari game, the environment is the game rule, and the state is represented by every frame of the game's visual output. Recent advances in computer vision [3-5] have made it feasible to reliably extract key features from raw images, making it viable to train an agent straight from the game's raw output. We saw the debut of this technique [6] where they essentially suggested using a convolutional neural network (CNN) to develop a policy straight from the game's raw vector graphics using a variant of Q-learning [7] which they called Deep Q-learning.

Since this high-dimensional input provides a huge number of potential states, data preparation and reward shaping are very important. Data preparation will help the agent learn efficiently by minimizing the noise and removing redundant information, while reward shaping helps the agent learn more quickly by providing guidance on how to complete the task. In this paper, we perform experiments on both issues by trying different configurations. We use OpenAI's Gym Retro as the framework for our Deep Q-learning agent and KungFu Nintendo Entertainment System (NES) game as the training ground.

* Corresponding author. E-mail address: dkkang@dongseo.ac.kr
Tel.: +82-10-75572944

2. Method

We consider a task in which an agent interacts with a stochastic environment, in this case, the KungFu NES game from Gym Retro OpenAI. In each time step, the agent observes a state represented by a vector of the raw pixels value from the game and then chooses an action among all possible actions. The environment will then reward the agent based on changes in the game's score and take the agent to the next state. The same procedure is repeated until the agent achieves a terminal condition, in which the agent succeeds in completing the game or loses.

The agent's objective is to learn a policy that maps a state into an action that will maximize the reward in the future. To learn the policy, we specifically used Deep Q-learning combined with experience replay. Formally, the policy maps $\mathbb{R}^n$ to $\mathbb{R}^m$ where the state is n-dimensional, and there are m number of actions. We attempt to empirically investigate how the agent's performance is influenced by data preparation and the use of various reward shaping. Our systematic experiments include:

- **Data Preprocessing**
  a. Convert the colored frame to greyscale and resize.
  b. Convert the colored frame to greyscale, resize, and crop the part of the frame which not playable as shown in the Fig. 2.
- **Reward Shaping**
  Three reward shaping: The reward for getting hit by the enemy with the value of -10, -50, and -100. A more detailed description will be explained in section 3.

2.1. KungFu NES Game

Kung-fu NES game is a beat-em-up game released in 1985. This game is based around a "kung fu" master who needs to save his friend after being kidnaped by a rival gang. The objective of this game is to defeat five bosses on five different levels, but for this experiment, we only consider the first level. On the first level, the boss is located on the very left of the map and is defended by a lot of enemies. Given this condition, we observe that the agent needs to learn to (1) move to the left to meet the boss while (2) defend itself from the defender's enemy and finally (3) kill the boss. The agent loses if it runs out of health or time, and the faster the agent completes the game the more point it receives.

There are three different enemies which are two defender enemies and a boss. We call the defender enemies after their vest color, which are the purple enemy and the blue enemy. The purple enemy hit the agent by punching, the blue enemy hit the agent by throwing a knife to the head or foot of the agent (the agent can dodge this knife by crouching or jumping respectively), and the boss hit the agent using an extended stick which has more range. In the NES controller, seven possible actions. Those seven action are punch, kick, jump, crouch, go left, go right, and do nothing.

2.2. Q-learning

In the typical reinforcement learning algorithm, given a state s_t, the agent will take an action a_t based on the policy π, then the environment will give the agent reward r_t and take the agent to the next state s_{t+1}. This same process is repeated until the agent reaches a terminal state. We assumed that each episode must have a terminal state so we set a limit on how long an episode could be. Given these sequences ($s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, ...$) we try to learn the best policy.

Generally speaking, Q-learning algorithm can be summarized into three steps: initialize Q-table, take an action, and update the Q-table with Bellman Optimality Equation. First, we initialize a Q-table, which stores the action value function $Q(s_t, a_t)$ for every state-action pair. We initialize every Q-value to zero, which means that the agent has not learned anything about the environment. Since Q-learning is a model-free method, the agent doesn't have access to the complete model of the environment. Therefore Q-learning will learn by trial and error. To encourage a balance between exploration end exploitation, Q-learning agent chooses an action based on ε greedy policy. After taking an action, Q-learning will update the Q-table by calculating the difference between the current Q-value, with the target Q-value: the sum of the reward it will get after taking action by following a policy π and the discounted next Q-value considering the agent taking the current optimum action (greedy action) action onwards. Eq. 1 shows the update rule of the Q-value, where α is the learning rate and γ is the discount factor. Performing this update rule for several episodes will make the Q-value converge since the target Q-value obeys the Bellman Optimality Equation which is denoted as $q_*(s,a) = E[R_{t+1} + \gamma \underset{a}{max}\, q_*(s_{t+1}, a_{t+1})]$.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) \; + \; \alpha[(r_t + \gamma \max_a Q(S_{t+1}, a)) \; - Q(S_t, A_t)] \tag{1}$$

2.3. Deep Q-learning

In Deep Q-learning, instead of using Q-table, we learn a function to map a state to an action. This function will be a convolution neural network (CNN) where the input is the raw pixels vector $\mathbb{R}^n$ from the game and the output is a vector of Q-value for every possible action $\mathbb{R}^m$. Fig. 1 shows the network architecture. To update the weight of the network, we use a similar bellman equation in Q-learning. The difference is that instead of calculating the difference between the current Q-value and the target Q-value, we calculate their squared difference, which is mean squared error (MSE). Eq. 2 denotes the loss function, where B is the batch size.

$$loss = \frac{1}{B}\sum_{i=1}^{B} [R_{i,t+1} + \gamma \max_a Q(S_{i,t+1}, a) \; - Q(S_{i,t}, A_{i,t})]^2 \tag{2}$$

To make the training more stable, Deep Q-leaning uses separate networks to calculate the target Q-value and the current Q-value namely the target network and policy network respectively. These networks share the same architecture but with slightly different weights. The loss function in Eq. 2 only optimizes the weights of the policy network W_p. Then, every k iteration, we update the weights of the target network W_t with the soft update [8] based on the policy network, as shown in Eq. 3, where τ is a hyperparameter with the value of τ << 1. The idea is the target Q-value is not always changing for every step which makes it produce a stable target Q-value for training the policy network. Additionally, the target Q-value are constrained to change slowly, adding more stability during training.

$$W_t \; \leftarrow \; \tau * W_p + (1 - \tau) * W_t \tag{3}$$

To determine what action an agent should take, Q-learning uses ε greedy policy. The agent will choose an optimal policy with a probability $1-\varepsilon$ and a random policy with a probability ε. The value of ε decreases from 1 to 0.1 over the course of training as denoted in Eq. 4, where p is the number of episodes and d is the rate by which ε to be decayed. This will make the agent do a lot of exploration early in the training and exploit its knowledge by the end of training.

$$\forall\{i \in \mathbb{N}, 0 < i < p\}, \varepsilon = 0.1 + (1 - 0.1) * e^{\frac{-i}{d}} \tag{4}$$

2.4. Experience Replay

In the experience replay setting [9], we store the agent's experiences at each time step in a replay memory. We represent the agent's experience at time t as e_t which is composed of the current state, action taken on that state, the reward it received, and the next state.

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1}) \tag{5}$$

During the training, all this data is stored in a replay memory with a finite size, so this memory only contains several latest experiences. Then after *L* step, we sample random experiences from the replay memory and use that to train the network. This will help the network's convergence by breaking the consecutive frame's correlation.

3. Data and Experiments

3.1. Model architecture

The model we use for the policy and target network is three layers of a convolution network followed by two fully connected layers. The first convolution layer has 32 filters with kernel size = 8 and stride = 4. The second convolution layer has 64 filters with kernel size = 4 and stride = 2. The third convolution layer has 64 filters with kernel size = 3 and stride = 1. The first and the second hidden layers have 512 and 7 neurons respectively, where 7 is the number of possible actions. We use zero padding for all the convolution layers and use ReLu activation function for all layers except for the output layer. All this setting follows the practice of [6].

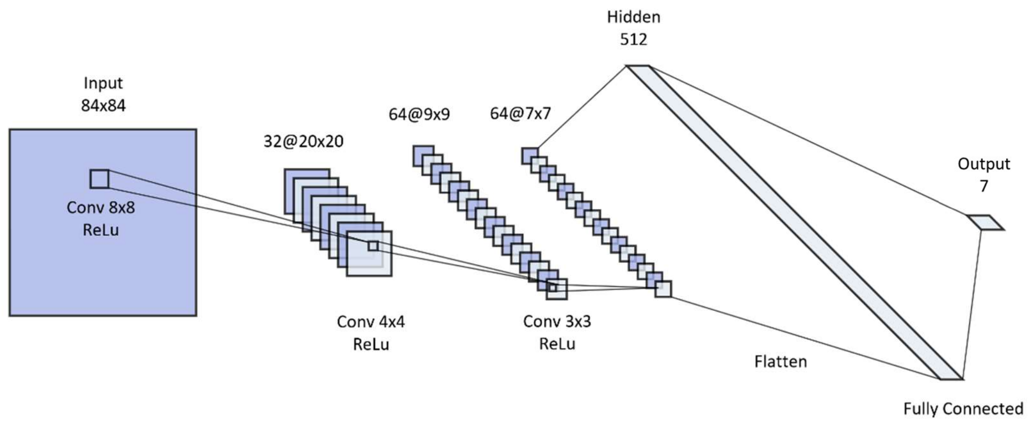
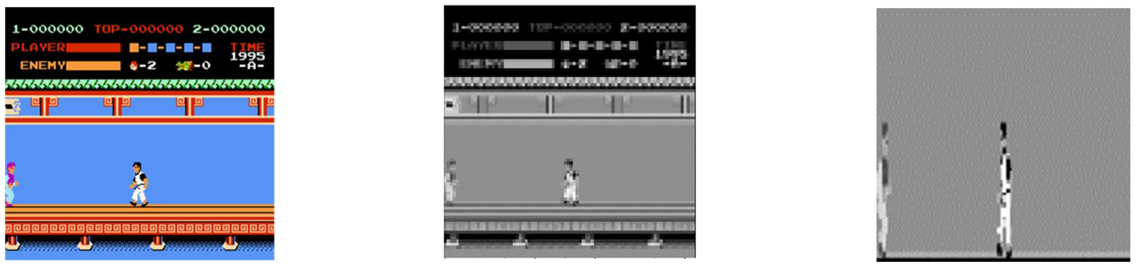


Fig. 1 Neural network architecture for policy and target network.

3.2. Preprocessing

The output video of the KungFu NES game from Gym Retro has a size 224x240 with 128 color palettes. As part of our experiments, we perform two different preprocessing. First, we convert the 128 color palettes e to grayscale and resize it to 84x84. The second is the same as the first experiment, but we crop the uninformative part of the frame which is the area that displays the score, health, lives, and time. The result of preprocessing step can be seen in Fig. 2. For all experiments, we also perform frame stacking which stacks four consecutive frames together. Frame stacking is important to give the agent a sense of motion.



(a) Raw frame (b) Preprocessed frame without cropping (c) Preprocessed frame with cropping

Fig. 2 The difference between preprocessed frame and non-preprocessed frame.

3.3. Reward Shaping

We assign the reward for the agent based on the changes in the game score. Kill the purple enemy will be rewarded 100 points, kill the blue enemy will be rewarded 500 points, kill the boss will be rewarded 2,000 points. Next, we give a punishment to the agent if it gets hit by the enemy. As part of our experiment, we use three different punishment value which is -10, -50, and -100. We specifically use small (-10), medium (-50), and high (-100) punishment values to study which value is sufficient for the agent to learn effectively. Additionally, in the game, the agent will receive more reward if it can complete the game faster.

4. Experiments

For training the network, we use RMSProp [10] with a minibatch size of 32, learning rate of 0.00025, and momentum with a value of 0.95. We use ε-greedy policy where ε = 1 for the first 100 episodes then degraded linearly from 1 to 0.1 over the training episode. We use replay memory with a size of 10^6 and sample the experiences from replay buffer every 50 frames (L = 50). To update the target policy, we use a soft update with τ = 0.0001. We use discount factor with a value of 0.99. For each experiment, we train the agent for 700 episodes. We run our experiment on an ubuntu 18.04 machine with a NVIDIA GTX 1080Ti GPU.

4.1. Experiments on Data Preprocessing

First, we studied the effect of data preprocessing on the performance of the agent. Specifically, we compared the agent performance trained on the cropped frame and uncropped frame. Fig. 3 shows that the agent trained with cropped frame outperforms but still does not deviate too much from the agent trained with uncropped data. This shows that the agent trained with uncropped images still manages to deal with noisy input, which is the upper part of the frame that contains the health, score, etc. However, training the network with cropped frame is still more effective and intuitive. Note that after the 600th episode, the score is stagnant or does not show improvement. This will be further explained in the next experiments.

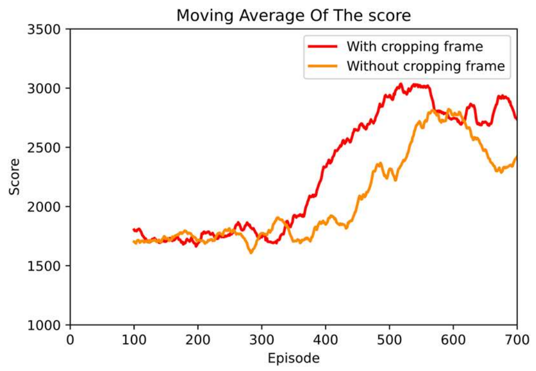
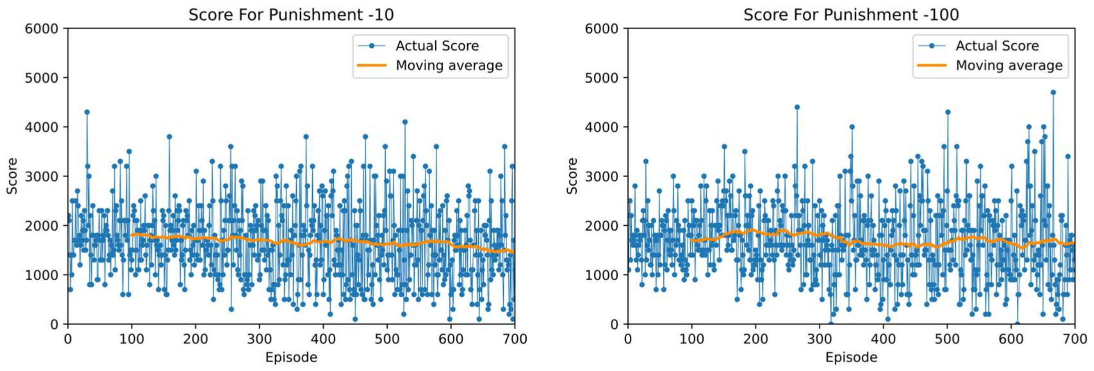


Fig. 3 Moving average of the score of data preprocessing experiments

4.2. Experiments on Reward Shaping

Next, we studied the effect of reward shaping on the performance of the agent. Specifically, we use three different punishments if the agent gets hit as described in section 3. Fig. 4 shows that using high and low punishment shares a similar result: the trend of the score is stagnant or even declining slightly. This is most likely happening because, with low punishment, the agent did not get enough punishment signals to learn to defend itself. On the other hand, with high punishment, the agent learns that getting hit by the enemy is really bad, so the agent tries to avoid the enemy at all costs. But this is not always the case. Especially at the beginning of the training, getting hit is good if the agent is finally able to kill the enemy which generates a reward signal.

When the low and high punishments fail to make the agent learn properly, we try to pick a value between them, which is punishment with the value of -50. Fig. 5 shows that the score trend is increasing, which is a sign that now the agent learns properly. However, the score did not increase that much from the beginning of the training. We carefully observe the agent's behavior during training, and we found two main problems.



(a) Punishment -10 (b) Punishment -100

Fig. 4 Comparison of the score between using a punishment -10 (a) and -100 (b)

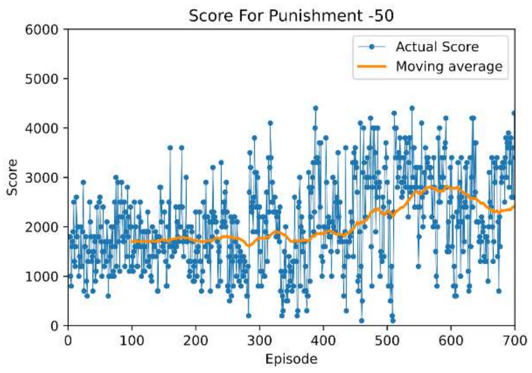


Fig. 5 The score plot of using punishment -50

To begin with, the agent never meets the boss as described in section 2. This is because the agent stays in the initial position and kills the oncoming enemies. In an ideal situation, we may assume that the algorithm would yield a greater Q-value for taking the go left action. However, there is no reward signal to encourage the agent to do so. Some Retro Gym games provide the *x* and *y* coordinate of the agent within the map of the game. With that information, we could punish the agent if it is stuck in the same place for successive frames. However, in this game, we don't have access to that data. Without it, it is quite difficult for the agent to meet the boss because there are a lot of sequences of actions that the agent must perform to meet the boss. The chance for this to happen by using a random policy at the beginning of training is fairly impossible.

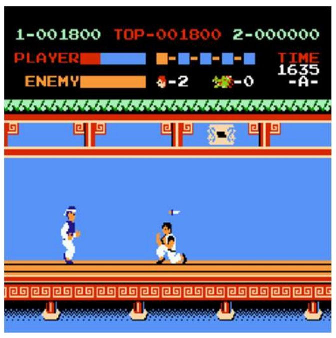


Fig. 6 The Knife thrown by the blue enemy

Second, we observe that the agent often dies prematurely because of the knife thrown by the blue enemy as shown in Fig. 6. This knife has a lot of damage and a long-range, so learning to dodge the knife is very important to prevent the agent keep dying. Therefore, giving the agent a small reward for dodging the knife might help to solve this issue. However, it is hard to add this kind of reward. Moreover, if the knife is thrown at the head level, the agent needs to take crouching action for several consecutive frames until the knife is completely past the agent. To help solve this problem, we can set the agent to take one action for several consecutive frames. This will make the action of the agent more stable and stabilize the training process [11]. However, there is a tread-off because it will make the agent unresponsive, so we need to find the right balance.

5. Conclusion

We have done experiments on training reinforcement learning agent to play the KungFu NES game by directly using the raw pixels from the game. We found that data preprocessing and reward shaping affects agent behavior, especially in such complex tasks. We show that failing to set up the proper data preprocessing and reward function could lead to ineffective learning or even failure to learn at all. From the experiments, we explore different data preprocessing and reward shaping and found a working setup that generates useful learning for the agent. We also analyze our experiment result and show that there are several interesting directions to improve the result of our experiment as mentioned in section 4.

Acknowledgment

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2022R1A2C2012243).

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] K. A. ElDahshan, H. Farouk, and E. Mofreh, "Deep Reinforcement Learning based Video Games: A Review," in *2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, 2022, pp. 302–309.
[2] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*, 1st ed. Springer Publishing Company, Incorporated, 2018.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012, vol. 25. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.
[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
[5] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, pp. 730–734.
[6] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[7] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach Learn*, vol. 8, no. 3, pp. 279–292, 1992.
[8] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[9] M. Andrychowicz *et al.*, "Hindsight Experience Replay," in *Advances in Neural Information Processing Systems*, 2017, vol. 30. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf
[10] G. Hinton, N. Srivastava, and K. Swersky, "Neural Networks for Machine Learning Lecture 6a Overview of mini-batch gradient descent," 2012.
[11] M. G. Bellemare, G. Ostrovski, A. Guez, P. Thomas, and R. Munos, "Increasing the action gap: New operators for reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016, vol. 30, no. 1.