A COMPARATIVE STUDY OF JITTER-MITIGATING METHODS FOR DEEP REINFORCEMENT LEARNING

Ajay Tulsan
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA
atulsyan@wpi.edu

Anthony Topper
Data Science
Worcester Polytechnic Institute
Worcester, MA
ajtopper@wpi.edu

Apiwat Ditthapron
Computer Science
Worcester Polytechnic Institute
Worcester, MA
aditthapron@wpi.edu

M. Caner Tol
Electrical and Computer Engineering
Worcester Polytechnic Institute
Worcester, MA
mtol@wpi.edu

Xiaosong Wen
Data Science
Worcester Polytechnic Institute
Worcester, MA
xwen2@wpi.edu

December 8, 2020

ABSTRACT

A Deep Q Learning (DQN) agent, trained based on exploration on the environment and aiming to maximize reward from the states, usually introduces unstable and unnecessary actions and makes *jittering* performance. In many cases, people can easily distinguish whether a performance is acted by a human player or a computer player. In this project, our goal is to prevent unnecessary actions and make the agent perform smoothly. To achieve our goal, we proposed adding bias into the cost function and adding penalties for unnecessary actions. To better visualize our work, we made comparison experiments on classic games like Flappy Bird and Super Mario.

1 Introduction

Motivation: In Deep Reinforcement Learning (DRL), the agent performs an action based on observation and policy. It is crucial to have a learning policy that controls a trade-off between exploration and exploitation. Although the agent gains environment and reward information through exploration, it introduces unstable action and produces a *"jittering"* motion. This motion, while not directly hindering the performance of the model, maybe undesirable. In many cases, it is unnecessary and can look unrealistic when compared with a human's performance. This study intends to prevent the agent from performing any extreme action that does not lead to reward gaining.

Previous work: Previous work improves the smoothness of action by proposing a noval learning policy or introducing a penalty term into the cost function. Shen *et al.* proposed a smooth regularized policy learning method by minimizing the difference of policy $\pi(s_i)$ and policy $\pi(s'_i)$, given that state s'_i is augmented from s_i with a small perturbation [1]. The method was evaluated in both on-policy and off-policy DRL and obtained a stable action.Yang *et al.* investigated various improvements to existing models for robotics, but in particular, describes a method of smoothing the learning process by modeling the reward based on the distance constraints in the various joints within the machinery [2].

In addition, there are imitation learning methods that learn smooth movements from human demonstration. Yu *et al.* proposed a meta-learning framework to perform one-shot imitation learning [3]. The model initially learns from human demonstration with an adaptation to robot demonstration. The agent is able to mimic human motion using only one video recording of human performing object manipulation. Another imitation learning proposed by Liu *et al.* learns contextual information from human demonstration using a transformer model before applying DRL with a reward calculated from a sequence of manipulation by robot [4]. Nair *et al.* introduced a learning method to tech robot to learn high-level representation of rope manipulation from human demonstration while learning low-level feature by itself [5].

Proposed method: Action stability can be achieved by introducing bias into the cost function - applying penalties for unnecessary actions. Some basic penalties are jerk, change in acceleration rate, and absolute distance in a given time. To this end, we propose a action-based penalty which prevents the agent from continuously choosing the same action. The introduced penalty mitigates the jittering at low computational cost. The proposed penalty is evaluated in FlappyBird and Super Mario simulators using Deep Q Networks (DQN).

The rest of this paper is organized as follows. Section 2 presents reinforcement algorithms that are core of this study, followed the proposed penalty and experiment setup in Section 3. The results are shown with a discussion in Section 4. Finally, this study is concluded in Section 5.

2 Related Works

2.1 Vanilla DQN

Before diving into complex models, it is important to understand some of the basic methodologies used for reinforcement learning. Deep Q networks (DQN) [6] focus on optimizing traditional Q-learning using neural networks. When there is a large state-action space, which is the case for many situations, it becomes unrealistic to store Q values for every possible state-action pair. Instead, a neural network can be used to approximate the optimal Q-function. This network can be trained based on a series of states which map to a series of actions, all from previously known experiences. The architecture of these networks often involves several convolutional layers. The states are often represented with minimal subsampling or processing, and in many cases are simply pixels from input images, making characteristics such as translation invariance imperative. Deep Q Learning uses the concept of experience replay, which involves randomly sampling states to avoid temporal correlations that would come with merely sampling data sequentially.

2.2 DQN with Prioritized Experience Replay

The concept of prioritized experience replay was first introduced in [7] and was incorporated into DQN in [8]. The prioritized replay works by considering important transitions more frequently based on the temporal-difference (TD) error.

3 Methodology

3.1 Penalty Terms to Mimic Human Action

The explanation in this section is based on the action in FlappyBird game for simplicity. There are two actions: 1 (jump) and 0 (not to jump).
Penalty A: Giving a action history ($\boldsymbol{h}$) contains n previous actions. Penalty a can be defined as

$$P_A = \left| \frac{1}{n} \sum_{i=1}^{n} h_i - 0.5 \right| \tag{1}$$

Penalty A increases the value in loss function if the distributions of previous actions are not equal. For instance, if the agent has a history action of $[00111]$, P_A will be 0.1. This penalty will prevent the agent from continuously performing the same action. However, n needs to selected wisely to archive the goal.

3.2 Baseline

We include vanila DQN and DQN with prioritized experience replay using unmodified loss function as baselines. The tuned hyperparameters are shown in Table 1.

3.3 Environment setup

We conducted our experiments on the FlappyBird game and Mario Bros that is part of PyGame-Learning-Environment (PLE) [9] project.

3.3.1 FlappyBird

The goal of FlappyBird game is to navigate the bird, using only a jump action, to pass through a gap between pipes without hitting any surface. The game is also terminated if the bird flying too high and hit the top or too low and hit the

Parameter	Value
Learning rate	0.0001
Optimizer	Adam
Loss function	SmoothL1Loss
Starting epsilon	1.0
Min epsilon	0.1
Epsilon decay duration	1E6 iterations
Buffer replay size	10000

Table 1: Hyperparameter Configuration for FlappyBird

ground. The game simulator operates at 30 frames per second. The original game increment the score once the bird passing the pipe, but in this study, we give a reward of 1 for every frames that the game runs and a negative reward of -5 when the bird dies.

3.3.2 MarioBros

Super Mario Bros. is a platform video game in which one has to control Mario, as he travels inside the Mushroom Kingdom to rescue princess Toadstool from Bowser. The game is a side-scrolling plat-former, the player moves from the left side of the screen to the right side to reach the flag pole at the end of each level. Mario loses a life if he takes damage while small, falls in a bottomless pit, or runs out of time or get hit by enemies. The game simulator operates at 30 frames per second. Since our goal is to get to move as far right as possible, the reward function consist of three parts: first, mario gets positive reward when moves right and negative reward when moves to left; second, mario gets negative reward with time elapsing; third, negative 15 points if mario dies. In this project we chose the 'COMPLEX_MOVEMENT' action spaces, which including 12 actions: [Nothing, Right, Right + A, Right + B,Right + A + B, A, Left, Left + A, Left + B, Left + A + B, Down, Up], here A, B, and Up means fire, Sprint and jump respectively.

3.4 Model

PLE framework capture a screen shot of FlappyBird game at 30 frames per second. Each frame contains 144x256 pixels in an RGB format. We combine three previous frames and current frame into a single observation, resulting in a tensor with dimension of (4,3,144,256). This tensor is reshaped to (12,144,256) and feeded to the DNN model that comprises three Convolutional Neural Network (CNN) layers and two Fully-Connected (FC) layers with RelU activation applied on each. The model acchitecture can be summarized as shown in Table 2. Pytorch [10] library was used to train the DQN with Adam optimization.

Layers
Input of shape (12,144,256)
CNN with 32 filters, kernel size of (8,8), and a stride of (4,4)
Relu activation function
CNN with 64 filters, kernel size of (4,4), and a stride of (2,2)
Relu activation function
CNN with 64 filters, kernel size of (3,3), and a stride of (1,1)
Relu activation function
FC layer of 512 units

Table 2: DNN architecture for FlappyBird

For Super Mario Bros, the model used worked directly with in-game frames as input, so some image preprocessing was done in order to reduce dimensionality and lessen the computational load. Each frame was down sampled from the original 256 × 240 pixels to an 84×84 pixels image. The convolutional neural network architechture is described below. It mirrors the network used in the Atari Learning Environment, but with a few alterations to the size of each layer as well as minor tweaks of the filter itself.

1. Input: Four RGB frames with a resolution of 84 × 84 pixels
2. Hidden Layer: Convolves 32 8 × 8 filters of stride 4 with the input image and applies a rectifier nonlinearity.
3. Hidden Layer: Convolves 64 4 × 4 filters of stride 2 and applies a rectifier nonlinearity.

4. Hidden Layer: Convolves 64 3×3 filters of stride 1 and applies a rectifier nonlinearity
5. Hidden Layer: Fully connected layer that consists of 512 rectifier units
6. Output: Action probability from Softmax

4 Results

4.1 Flappy Bird

The Pygame and Gym environments for Flappy Bird and Mario are shown in Figure 1 and 4. Figure 2 shows reward gaining from DQN with prioritized experience replay. The agent starts to learn important features after 26,000 episodes. Figure 1 shows the trained agent that is capable of making timely actions against the obstacles. At around 35,000 epochs, the bird survives five minutes of game playing. The demonstration video can be downloaded at `https://github.com/aditthapron/RL_project/blob/main/FlappyBird_DQN_prioritizedReplay_1/DQN_prioritized.avi`. Unfortunately, the proposed penalty prevents the learning, in such a way that, the bird cannot stay in a gap of the first pipe and results in a maximum reward of 32. A delay and weighted penalty were experimented, but failed to improve the learning.

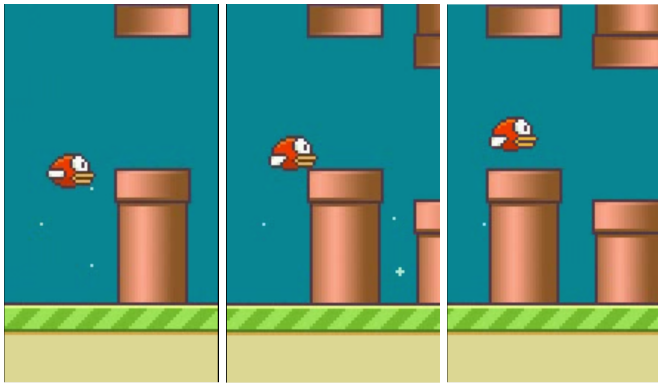

Figure 1: Trained model is able to make Flappy Bird do successive up actions to fly above the obstacle.

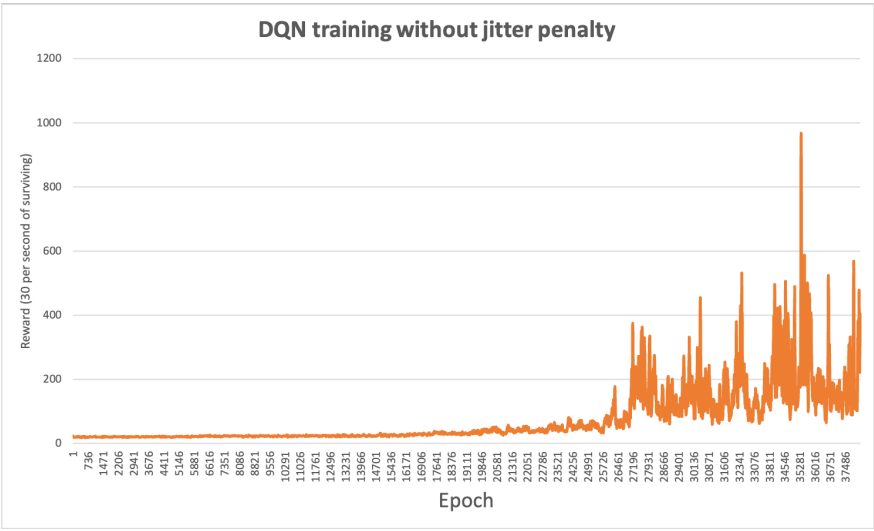

Figure 2: A plot showing reward in FlappyBird over 40000 episodes without jittering penalty

Performance Reduction Depending on the nature of the environment, the smoothing penalty can lead to hindrances to the agent's performance during a game. For example, in our simulations for Flappy Bird, the penalty ended up making the game impossible — the agent could not even surpass a score of one. The nature of some environments leads

to the necessity for sporadic and non-smooth behavior. In some cases, this performance decrease can be mitigated by applying a coefficient to the penalty value in the cost function. Instead of having the penalty weigh severely against the agent, it can be limited using a value less than 1.

4.2 Super Mario

In Super Mario, agent has an objective to reach a finishing point as fast as possible. Our proposed penalty prevents the mario from continuously moving forward. A reward function with an average are plotted in Figure 3.

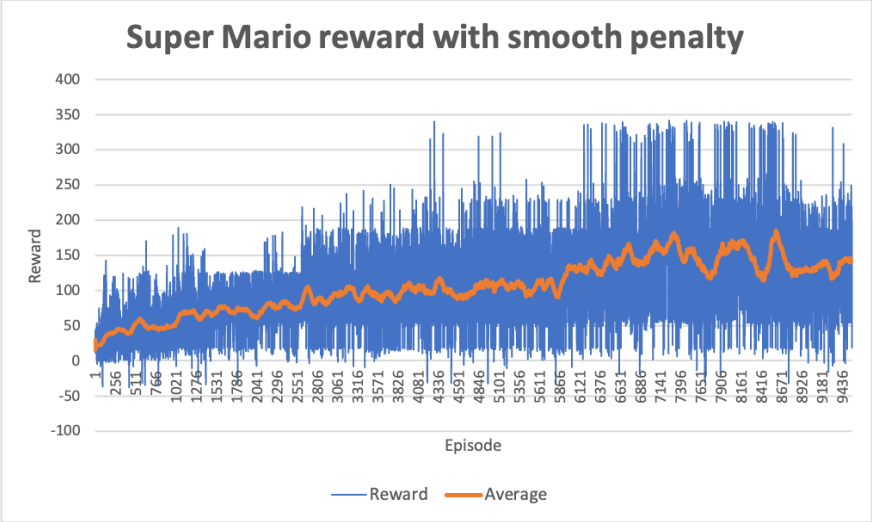

Figure 3: A plot showing reward in Super Mario game over 10000 episodes with jittering penalty

Due to a time constraint, this model was trained for only 24 hours. The Mario has not yet reach the finishing point, but is very close to, as shown in Figure 5.

Figure 4: Mario game in Pygame environment

Figure 5: A snapshot of Super Mario game with the highest reward

Evaluation We created a survey collecting an opinion of Super Mario game play in terms of smoothness. The survey and videos are accessible at `https://forms.gle/JBeXqgP1YJXDW26w8`. Twenty-five participants were asked to watch ten videos, five of them from the DQN with smoothing penalty and five of them from the DQN without smoothing penalty and rate the game smoothness in a scale of 1 to 10 without knowing which one belongs to which class. The results are averaged for each video and summed for each of both classes, no-smoothing and smoothing. Figure 6 shows that the DQN with smooth penalty has an average smoothness score of 6.13 (out of 10), while the DQN without smooth penalty an average smoothness score of 4.95 (out of 10). The survey result indicates that the proposed smooth penalty mitigate a jittering movement that is usually manifested in reinforcement learning. the participants think the agent trained with smoothing penalty acts 20% more smoothly while playing the Super Mario Bros game.

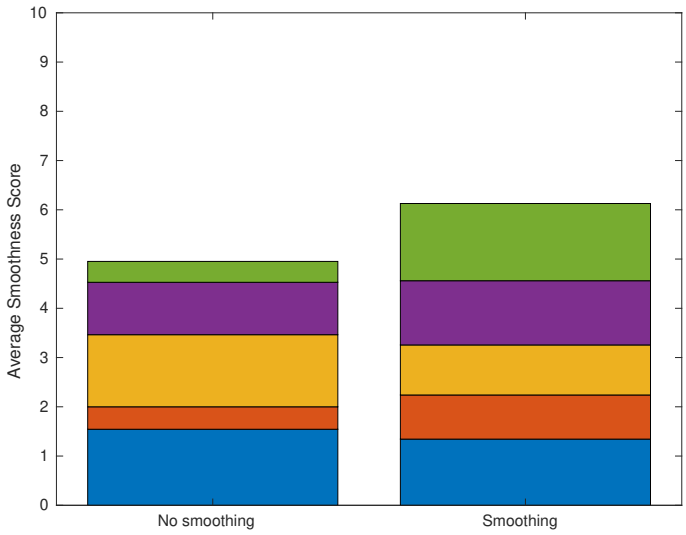

Figure 6: Survey results for the agent trained without smoothing (left) and with smoothing(right). Each color in a column represents a different video from the corresponding agent.

5 Conclusion

The proposed penalty term based on the action history improves the smoothness of game playing, but hinder a fast learning process of reinforcement learning. Some environments that requires a quick action, e.g. FlappyBird, might not benefit from the proposed penalty and prevents the learning from converging while open-world game, e.g. Super Marios, that has many actions and most action can be delayed gains an learning advantage and smoothly controls the agent like human.

6 Schedule

Our tentative plan until Dec 10,2020 is shown in Table 3.

Table 3: Weekly Schedule

Due Date	Objective	Status
Nov 12	Deliver Proposal Report	Done
Nov 15	Paper review and research state-of-art IL/IRL algorithms	Done
Nov 19	Set up environment for Super Mario & Flappy Bird	Done
Nov 26	Run DQN models for Flappybird	Done
Nov 28	Determine Jittering Penalty term	Done
Nov 30	Deliver Progress Report	Done
Dec 2	Add smooth policy, IL/IRL algorithms	Done
Dec 3	Run DQN models for Super Mario	Done
Dec 4	Hyperparameter Tuning	Done
Dec 5	Comparison on different avatar game	Done
Dec 7	DEMO: Smoothing vs no smoothing	Done
Dec 8	Deliver Final Report	Done
Dec 10	Poster and presentation	In-progress

References

[1] Qianli Shen, Yan Li, Haoming Jiang, Zhaoran Wang, and Tuo Zhao. Deep reinforcement learning with robust and smooth policy, 2020.

[2] Jin Yang and Gang Peng. Deep reinforcement learning with stage incentive mechanism for robotic trajectory planning. *arXiv preprint arXiv:2009.12068*, 2020.

[3] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning, 2018.

[4] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation, 2018.

[5] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation, 2017.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[7] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[8] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[9] Norman Tasfi. Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.