Exploring the use of data preprocessing reinforcement learning to play Pong

Xingzhi Lu

Concord College

1 Introduction

In recent years, there has been a boom in the use of computer vision models for reinforcement learning. However, as seen in other computer vision models, it can often be difficult for the model to extract the key information from the image. This is often done through training convolutional neural networks, which takes up significant computing memory and time. Therefore, the research aims to explore whether there is a more efficient alternative to CNN models through manual feature extraction.

2 Literature Review

2.1 Previous work

Since DeepMind developed the first deep reinforcement learning that automatically learns to play various Atari games in 2013 [1] with a Deep Q-learning Network (DQN), there has been many researches that adopts deep reinforcement learning, due to how it demonstrated the viability of training a completely computer-vision model that acts as an smart agent. It is worth noting that there were at least 30 papers published from 2014 to 2017 after DeepMind's breakthrough in DQN in 2013 [2].

Some of the key researches after 2013 include how Levine et al. trained a robot arm to complete manipulation tasks simply using a computer vision model, [3] and how Gu et al. extended deep reinforcement learning to a continuous action space in robot arm control.[4]

The use of self-playing in training was inspired by DeepMind's 2016 research on AlphaGo, which was the first Chinese Go playing AI that defeated a world champion in Go. [5] OpenAI subsequently trained a model that solves Rubik's Cube with a robot hand. [6] This research showed how self-playing could solve the issue of having insufficient training data.

3 Methodology

In this investigation, we would look at how a model that takes in the last few frames of the gameplay directly performs as compared to a model that takes in some features extracted from the last few frames of the gameplay in various games. We would measure the performance of the models by comparing number of epochs used to reach a target level of reward for 10 epochs in a row.

3.1 Games considered

In this experiment, I will consider the following games listed. Those games are either single-player or the agent is being played against an AI.

3.1.1 Pong

I have used a simplified version of Pong, removing the more realistic physical simulations, so that the game consists solely of reflections of the ball when hitting the paddles or the boundaries. The game is rendered in 0s and 1s using a 160×210 array in computer vision mode, and returns the current position, speed of the balls and the paddles when using feature extractions. To make the reward less sparse, I have altered the reward function so that there is reward for each time the agent hits the ball with the paddle. In order to make the game less deterministic, I slowed down the speed of the ball so that it is guaranteed that the player can catch the ball. The opponent AI will also hit the ball each time so that there is as little randomness as possible. To simplify the game for training, I have also removed most of the physics engine of the game, leaving only ball reflections when it hits the wall or the paddle.

The challenge was the difficulty for the AI to successfully hit the ball with the paddle when it was acting randomly, hence it was hard for the AI to get a positive reward. As such, I increased the learning rate for the model.

The target of the training is to not lose a single point against AI for 1200 frames, which equates to 20 seconds.

3.1.2 Flappy bird

I used the external libraries available on GitHub. The rewards are: +0.1 for every frame it stays alive; +1.0 for successfully passing a pipe; -1.0 for dying; −0.5 for touching the top of the screen. [7], [8]

For the feature extraction observation, it returns a tensor with:

- The last pipe's horizontal position
- The last top pipe's vertical position
- The last bottom pipe's vertical position
- The next pipe's horizontal position
- The next top pipe's vertical position
- The next bottom pipe's vertical position
- The next next pipe's horizontal position
- The next next top pipe's vertical position
- The next next bottom pipe's vertical position
- Player's vertical position
- Player's vertical velocity
- Player's rotation

Similar to Pong, the computer vision observation is composed of a screenshot of the current gameplay.

3.2 Control variables

Both models used will have a roughly equal number of learnable parameters and are all trained under the same learning rate and other hyperparameters. They will all be trained under the same virtual gym environment.

3.3 Model architectures

Both models take in the data from the last 3 frames of the gameplay. The frame data were either taken in as Tensors. In the computer model, the tensor is just a grayscale or RGB array which is a screenshot of the gameplay.

3.3.1 Multi-frame computer vision model

The model includes 3 convolutions with ReLU in between, followed by 3 fully connected layers, and each layer has a ReLU function after it as the activation function.

3.3.2 Feature extraction model

The model includes 3 fully connected layers, and each layer has a ReLU function after it as the activation function. I have made the stride of the image greater than one so that the image becomes smaller as it goes through the convolution layers, which help to save computational power and time and uses up less memory.

3.4 Training and evaluation

For the training cycle, I use the deep Q-learning algorithm to train the model, where I sample 32 frames for each back propagation. After each training epoch is completed, I then play the model deterministically for a set number of frames, to discover the level of reward it receives. Using this, I will be able to determine when the model achieves a target loss or reward.

4 Evaluation of various gaming methods

4.1 Experiment data

Game	Target	Epoch needed to achieve	
		Computer vision	Feature extraction
Pong	0 points lost in 20 seconds	>500	390

4.2 Conclusion

As seen, in the games considered, the computer vision model took much longer to complete the training. Also considering the significantly higher amount of computation time required for the computer vision model to inference due to the large input tensor (200×300), it would be much less efficient both time and memory-wise than the feature extraction model.

5 Limitations and future work

5.1 Limitations

Due to the limited availability of computational resources (only one A100 GPU on Google Colab), I could not test out models with a larger number of trainable parameters due to the shortage of memory and the unacceptable long training time. The research was solely done on video games where feature extraction is straightforward, and the game play is not complex. Therefore, this research might not apply to the more complex games where the current game state cannot be solely represented via a tensor consisting of a few numbers. Furthermore, the games all have straightforward reward function designs, and the reward functions of the games were specifically designed to ease the training process and to make the rewards less sparse. In games where there reward of the game is sparse or difficult to compute, such as when the target of the game was to accomplish specific target tasks, the reinforcement learning method might fail to learn to play the game effectively. The game also only focused on games where the player is facing a single enemy AI or environment.

5.2 Future works

The work can be extended to games where players are set a specific target to beat instead of just playing a set enemy or environment. For example, this can be extended to games where the player is asked to complete certain tasks. These

games would be much harder to model since there is usually not a way to easily describe the tasks using numbers.

Bibliography

[1] V. Mnih *et al.*, ‘Playing Atari with Deep Reinforcement Learning’, Dec. 19, 2013, *arXiv*: arXiv:1312.5602. doi: 10.48550/arXiv.1312.5602.

[2] H. A. Pierson and M. S. Gashler, ‘Deep Learning in Robotics: A Review of Recent Research’, July 22, 2017, *arXiv*: arXiv:1707.07217. doi: 10.48550/arXiv.1707.07217.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, ‘End-to-End Training of Deep Visuomotor Policies’, Apr. 19, 2016, *arXiv*: arXiv:1504.00702. doi: 10.48550/arXiv.1504.00702.

[4] S. Gu, E. Holly, T. Lillicrap, and S. Levine, ‘Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates’, Nov. 23, 2016, *arXiv*: arXiv:1610.00633. doi: 10.48550/arXiv.1610.00633.

[5] D. Silver *et al.*, ‘Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm’, Dec. 05, 2017, *arXiv*: arXiv:1712.01815. doi: 10.48550/arXiv.1712.01815.

[6] OpenAI *et al.*, ‘Solving Rubik’s Cube with a Robot Hand’, Oct. 16, 2019, *arXiv*: arXiv:1910.07113. doi: 10.48550/arXiv.1910.07113.

[7] M. Kubovčík, *markub3327/flappy-bird-gymnasium*. (Aug. 25, 2025). Python. Accessed: Aug. 26, 2025. [Online]. Available: https://github.com/markub3327/flappy-bird-gymnasium

[8] G. Nogueira, *Talendar/flappy-bird-gym*. (May 31, 2025). Python. Accessed: Aug. 26, 2025. [Online]. Available: https://github.com/Talendar/flappy-bird-gym