Human-like AI in a Fighting Game Using Reinforcement Learning

Riku Tanji s1270139 Supervised by Prof. Maxim Mozgovoy

Abstract

Fighting games usually include battles against an AI-controlled opponent for training. Such AI needs to resemble human players to serve as training AI, allowing players to have mock battles without other players. In this research, an attempt was made to create human-like AI in a fighting game using reinforcement learning. Similarity to demonstration data played by human player is used for human-like behavior. The proposed method based on DQN worked effectively compared to pure DQN.

1 Introduction

In this study, we aim to create human-like AI in a fighting game. An AI opponent in fighting games serves not only as an enemy to beat but also as a training partner to practice with. In a fighting game whose main game mode is against real players, it is important to also have battle mode against AI, as the players can practice in the same setup many times without other real players. The AI should behave as human players behave to become good training partners. Also, since fighting AI often has humanly impossible behaviors, such as fast reaction time and perfect precision of combos, this makes the players feel it is unfair, but human-like AI by itself solves this problem.

There has been much past research done on AI in gaming. In Oh et al., deep reinforcement learning is used to create AI in a commercial fighting game called Blade & Soul [1]. They employed reward shaping to generate three different styles of AI in order to utilize self-play. They successfully created AI that even surpassed the ability of professional players. Needless to say, our goal is human-like AI, not strong AI like this work.

In Dossa et al., they attempted to create a human-like AI that sustains high performance and tested it on three games [2]. In their method, based on reinforcement learning, the policy was updated, taking into account both human play-based policy and pure reinforcement learning-based policy. As a result, the created agent achieved both high performance and human-likeliness. Their method is one example of a method that attempts to use imitation learning using reinforcement learning, like ours.

2 Method

2.1 Algorithm

To imitate human players' behavior, one possible approach would be behavior cloning. In behavior cloning, the agent learns the expert's policy using supervised learning with collected data demonstrated by the expert. However, directly learning an expert's policy often has problems, such as requiring numerous samples and getting stuck in local optima.

Instead of directly imitating a human player's policy like behavior cloning, guiding the agent to human-like behavior while it is learning how to play on its own is also a common approach, especially in a more complex environment. We took this approach in this research. In our method, DQN [3] was used to let the agent learn how to play the game and another technique was introduced so that agent can follow some given demonstrations while learning the policy.

DQN is a reinforcement algorithm to learn a mapping from a particular state to the value of an action. This mapping function is called the Q function and is represented by a neural network. At each time step, the agent takes an action, a_t under the current state, s_t. Then the agent receives reward, r_t and observes the next state, s_{t+1}. The Q function is then updated using the obtained information and the Bellman equation to make it more similar to the actual Q function. This process is repeated until the performance of the agent improves.

To make its behavior closer to a human player's behavior, we used demonstration data. During training, if the agent chooses exploitation based on the ϵ-greedy algorithm, the agent only takes an action, a_t which satisfies:

$$a_t = \arg\max_a Q(s_t, a) \text{ where } L(s_t, a, D) \geq L_m$$

Here, $L(s, a, D)$ is a similarity function between the state-action pair and the demonstrations D, and L_m is a constant we manually set. $L(s, a, D)$ simply returns -1 if action a is not found in demonstrations D, otherwise, it returns the maximum cosine similarity value between state s and a state in the demonstrations D where the next action is the same as a. With this similarity constraint, an action taken by the agent is guaranteed to have some similarity to the demonstration data.

Table 2: State space

Elements of state space		Explanation
Agent information	isJumping	Boolean, true if the agent is jumping
	frame advantage	Frame information which indicates how the agent is safe or unsafe
	last move	The last move the agent did
	moving forward duration	Time duration from when the agent starts moving forward until the current frame
	moving back duration	Time duration from when the agent starts moving backward until the current frame
	crouching back duration	Time duration from when the agent starts crouching back until the current frame
Opponent information	isDown	Boolean, true if the opponent is down
	isJumping	Boolean, true if the opponent is jumping
	isBlocking	Boolean, true if the opponent is blocking
Global information	distance	Distance between two characters

Also, potential-based reward shaping was used for faster training. An additional reward f is added to r_t. The new reward signal is:

$$r_t' = r_t + f(s_t, a_t, s_{t+1}, a_{t+1}) \quad (1)$$

where f is defined as:

$$f(s_t, a_t, s_{t+1}, a_{t+1}) = \gamma L(s_{t+1}, a_{t+1}) - L(s_t, a_t)$$

Additionally, some common techniques such as target Q-network, Experience Replay, and annealing ϵ-greedy were used to make the training more stable and faster.

Finally, all parameters were manually adjusted. They are shown in Table 1.

Table 1: Parameters

learning rate	0.00025
mini batch size	32
initial epsilon	1
final epsilon	0.1
epsilon decay	0.0001
γ (discount factor for future reward)	0.999
replay buffer size	10000
update interval of target Q network's weights	10000
L_m	0.5, 0.9

2.2 Testbed

Universal Fighting Engine [4] was used as a testbed. UFE is a fighting game engine that works on Unity. We used a demo game in UFE which is already playable. The demo game is like a typical fighting game: you can move left and right, jump, attack, and block, but there are no throws.

During training, an in-game AI called Random AI was used as an opponent. Random AI is a simple rule-based AI that approaches to within attack range and strikes a reachable move. Also, the opponent and the agent used the same character for a fair evaluation.

2.3 A model for learning

The model consists of state space, action space and reward function.

Table 2 shows the structure of state space. State space consists of 10 elements. All state information is from in-game, not visual information. Each element was discretized or normalized accordingly.

There are 25 actions such as basic moves (walk forward and backward, crouch, jump, etc.), attack moves (light standing punch, heavy crouching kick, etc.), and special moves (dash, heavy fireball, etc.). Note that actual moves are used as actions, not physical button inputs. This helps the agent learn faster as different button inputs are likely to cause

different moves depending on the character's state.

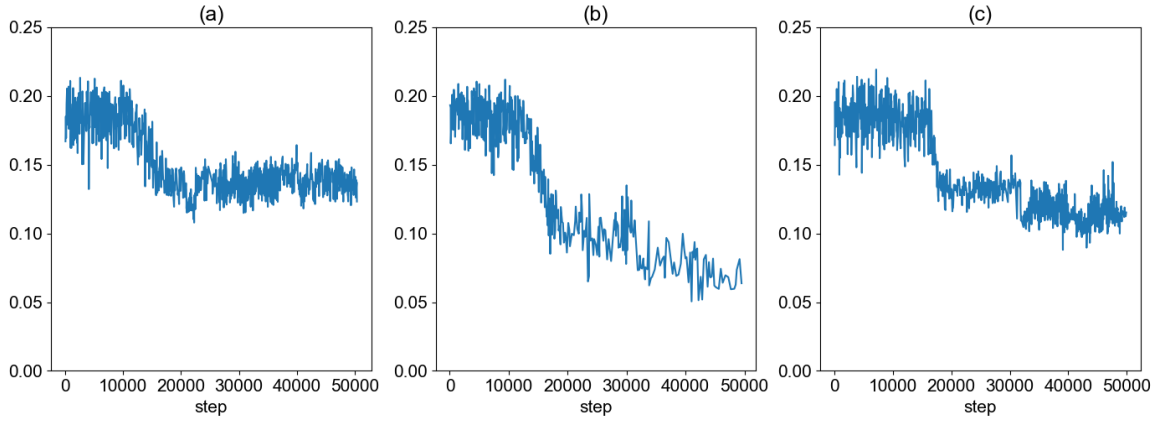

Figure 1: Action frequency errors of (a) L_m=0.5, (b) L_m=0.9, and (c) pure DQN

Decision timing is as important as state and action space. In our implementation, the agent decides the next action immediately following the previous action is executed. The chosen action will be executed when it becomes executable. For example, when a character is down or blocking an attack, all moves are not executable. The agent waits until such a state ends and then casts the move. This reduces unnecessary decisions in which the chosen action doesn't affect the environment and makes learning faster.

Finally, the environment reward (the term denoted as r_t in (1)) is calculated as follows:

$$r_t = \text{damage dealt} - \text{damage taken} + \text{chip damage taken} / 2$$

As shown in the formula, the agent is rewarded or punished according to the damage dealt and the damage taken. Each damage value was normalized between 0 to 1. However, if an agent blocks an attack but takes a little damage (chip damage), it is rewarded with half of the damage received. This is intended to prevent agents from perceiving blocking as a bad thing due to the chip damage of an attack.

3 Results and discussion

In the experiment, three types of training are compared, namely our DQN with L_m of 0.5 and 0.9, and pure DQN. The agents were trained five thousand steps with the same parameters except for L_m. In each episode, frequencies of each action were calculated and compared to the frequencies in demonstration data used in the training. The frequency is a vector with the size of the number of actions and normalized between 0 to 1. Mean absolute error is used to calculate the difference between two vectors. The used demonstration data is play data by a single player against Random AI. The data contains eight episodes, approximately 10 thousand steps. We also collected total reward the agent gained per episode to see its performance.

Figure 1 shows the action frequency errors for the three trainings. Action frequency errors were smallest when L_m was 0.9. Errors when L_m was 0.5 did not contribute to the reward received, indicating that the similarity constraint at L_m =0.9 works effectively for having human-like behavior.

Figure 2 also shows the frequency for each action. (a) represents the frequency of actions within the demonstration data used, and (b)-(d) represent the frequency of actions within the last episode during training. As can be seen from this figure, the case where L_m is 0.9 is the closest to demonstration. In particular, action 0 (Neutral, i.e., doing nothing) is the most obvious, as it is the most frequently selected action in both (a) and (c) despite the fact that it does not contribute to the reward received, indicating that the similarity constraint at L_m =0.9 works effectively for human-like behavior.

Figure 3 shows the accumulated rewards for each episode during each training. Although there is not that much difference between the three figures, we can see that (a) is the most stable in terms of rewards. Since (a) does not affect action selection as discussed above, and since reward shaping speeds up learning, it can be assumed that performance simply increased due to reward shaping. Plus, the fact that performance in (b) is slightly lower than in (a) suggests that the similarity constraint may have a negative impact on performance.

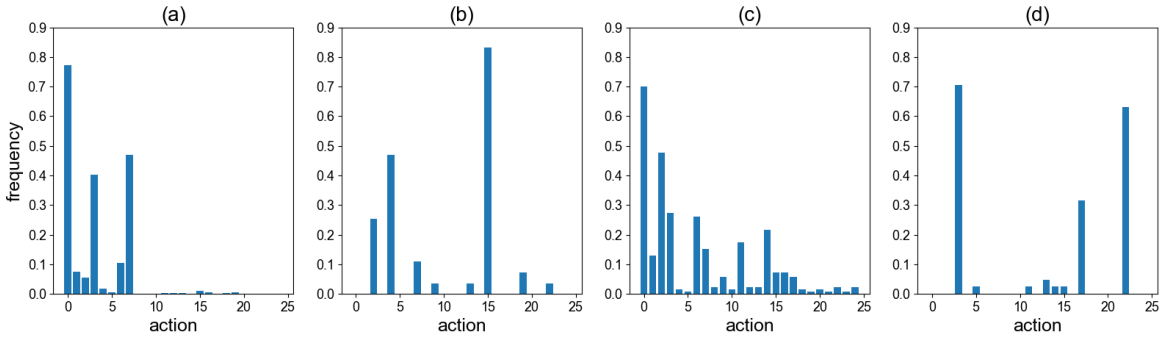

Figure 2: Action frequencies of (a) demonstrations, (b) L_m=0.5, (c) L_m=0.9, and (d) pure DQN

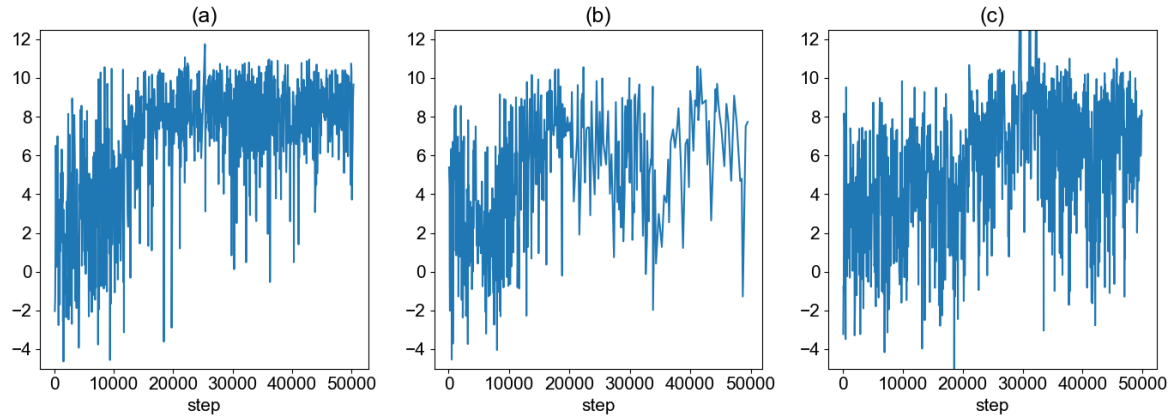

Figure 3: Accumulated reward per episode of (a) L_m=0.5, (b) L_m=0.9, and (c) pure DQN

4 Conclusion

This study proposes a method for creating human-like AI in a fighting game based on reinforcement learning. Similarity to demonstration data is used for human-like behavior, and reward shaping using the demonstration data is used to speed up training. It was shown that our method affected human-like behavior in terms of action frequency at L_m=0.9. It was also found that the similarity constraint may have a slightly negative impact on performance.

However, due to computational time and resources, it is possible that the training could not be done for a sufficient number of steps to know the real impact of the algorithm. In addition, in order to compare the human-like characteristics of agents, it may be necessary not only to simply compare the frequency of their actions, but also to consider multiple angles, such as comparing a sequence of moves and the state when the move is selected. Also, the weakness of our method is the extra computational time compared to pure DQN, since not only the output of the model but also the similarity with the demonstration data is required for action selection during inference.

References

[1] I. Oh, S. Rho, S. Moon, S. Son, H. Lee, and J. Chung, "Creating pro-level AI for a real-time fighting game using deep reinforcement learning," *IEEE Transactions on Games*, vol. 14, no. 2, pp. 212-220, 2022.

[2] R. F. J. Dossa, X. Lian, H. Nomoto, T. Matsubara, and K. Uehara, "A human-like agent based on a hybrid of reinforcement and imitation learning," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1-8.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning", *arXiv preprint arXiv:1312.5602*, 2013.

[4] "UFE [Universal Fighting Engine]", ufe3d.com http://www.ufe3d.com/doku.php (accessed Dec. 12, 2022).