1. Classic control 1: cartpole

The first classic control I trained with a reinforcement learning algorithm is a Cart Pole environment. The goal of the CartPole environment is to keep the pole upright for as long as possible. A reward +1 is given for every step taken. The Cartople environment has a discrete action space, which consists of two actions, 0: pushing the cart to the left and 1: pushing the cart to the left. It has 4 observation spaces which are 0: cart position, 1: cart velocity, 2: pole angle, and 3:pole angular velocity.

1.1 RL algorithm: Deep Q Learning algorithm (DQN) Design

The Deep Q Learning algorithm (DQN) is used to train the CartPole. DQN is a value based approach which applies a deep neural network with θ as parameter to estimate the stateaction value Q, and utilise another neural network to generate the target Q value Q [1]
The observations which are also known as states are fed into the DQN's neural network. This produces the estimated Q-values for each action. [reference] The action is selected by the agent using an epsilon-greedy which is an exploration-exploitation strategy.
The agent receives an observation of the next state and a reward when the selected action is done. The previous state, action, reward, and the next state is stored in a replay buffer. The neural network of DQN is trained to minimise the difference between the target Q value and predicted Q value.

1.2 Justification

As mentioned above, the DQN estimates Q-values for each action, therefore it is suitable for discrete action space. The CartPole environment has a discrete action space and benefits from a value based approach. This was one of the reasons why I chose to implement DQN. Furthermore, the DQN algorithm is improved by using experience replay with the stabilised value function [2].
Exploration is where agents learn how to predict and control unknown environments and exploitation is making decisions in the known environments [3]. In the DQN algorithm used for the CartPole environment, epsilon-greedy is implemented as previously mentioned, to balance exploration and exploitation. The ability to balance exploration and exploitation effectively was one of the reasons why I chose to use it for the CartPole environment.

2. Classic control 2: mountain car continuous

The second classic control I trained is a Mountain Car Continuous. This environment has a continuous action space of Box (-1.0, 1.0, (1,), float32) which is different from the first environment, CartPole. The goal of training in this environment is to make the cart reach the goal with the minimum time step (which also means higher positive reward). A negative reward of -0.1* action is received at each timestep to penalise for taking actions of large magnitude and the positive reward of +100 is added to the negative reward for that timestep. It has two observation spaces which are 0: position of the car along the x-axis and 1: velocity of the car.

2.1 RL algorithm: Proximal Policy Optimisation (PPO) Design

The Proximal Policy Optimisation (PPO) algorithm is used to train the mountain car continuous environment. PPO is a policy gradient method [4]. Clipping is applied as a parser argument, which determines the maximum ratio between the probabilities of actions to limit the policy updates [4]. The policy function is updated using surrogate loss functions and the value function is updated using the value loss function.

2.2 Justification

The action space of the Mountain Car Continuous environment is continuous which tries to reach the goal (the top of the mountain) with continuous actions. The actions are not restricted to a fixed set of values like the CartPole environment. I chose to use the PPO algorithm which can maximise the expected cumulative reward using a policy based approach and handle the continuous action spaces well. Moreover, in the PPO algorithm I implemented, a replay memory is used for storing and sampling experiences. As explained in section 1.2, use of replay memory improves sample efficiency and outputs and I thought this would be beneficial for training continuous. Also, the clipping functionality mentioned above which prevents a big change and helps to converge was one of the reasons I used this PPO algorithm.

3. Atari game: Pong

The game I chose from Atari to train with the RL algorithm is an environment called Pong. The pong has a discrete action space which consists of 6 actions, 0: NOOP, 1:FIRE, 2:RIGHT, 3:LEFT, 4:RIGHTFIRE, 5: LEFTFIRE.
It gets points for getting the ball to pass the opponent's paddle and loses points if the ball passes your paddle.

3.1 RL algorithm: Deep Q Learning algorithm (DQN) Design

As same as the CartPole environment, the DQN algorithm is used to train the Pong game. The convolutional neural network is used in the DQN I implemented. The agents select actions based on the epsilon-greedy exploration strategy. The agent receives an observation of the next state and a reward when the selected action is done. The next observations, rewards, and termination information is stored in a replay buffer.
It takes a sample of experiences from the replay buffer and performs Q-learning updates using the loss function and optimiser.

3.2 justification

As mentioned above, the DQN is well suited for discrete action space. The Pong game has a discrete action space of 6, so I chose to use DQN for training the Pong game.
The DQN algorithm benefits from using experience replay [2] and balances exploration and exploitation using epsilon-greedy. These abilities are also the reasons why I chose to use the DQN algorithm.

4. Results and Evaluation

4.1 CartPole

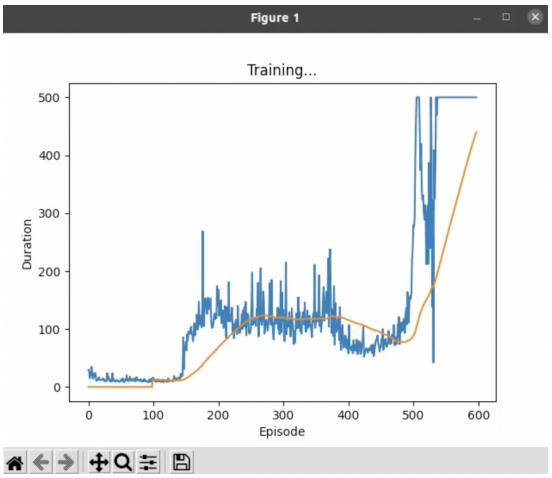

Fig 1. Cartpole graph

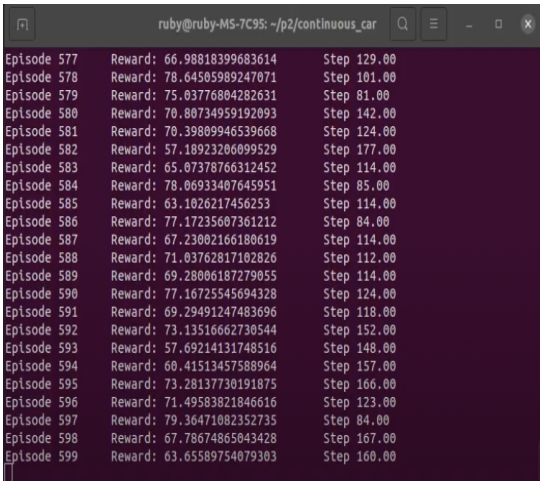

Fig 1. Cartpole rewards

The Fig 1. and Fig .2 show that the cartpole reaches maximum rewards 500 with 500 steps. As the goal is to keep the pole upright as long as possible (max 500 steps), the training went

well, reaching the maximum, stabilising after around 520 episodes. There were some periodic drops observed, which could be caused by overfitting, but it managed to get the constant values at the end. If I had time, I would tune the parameters to check what caused the drops.

4.2 Mountain Car Continuous

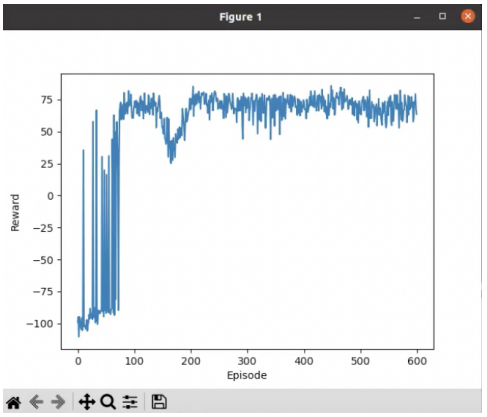

Fig 3. Mountain Car graph

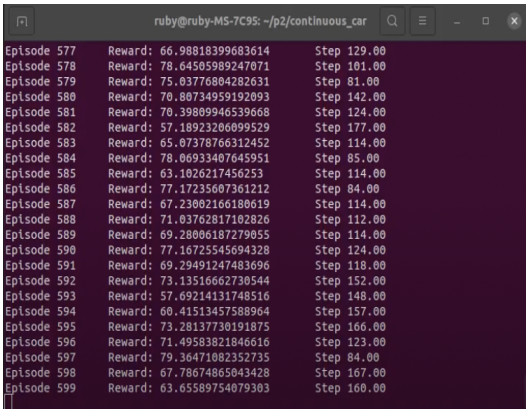

Fig 4. Mountain Car rewards

The Fig 3. and Fig 4. show that the Mountain Car Continuous reaches the top of the mountain with small steps (between around84~167, untrained one has 999 steps), with average of 75 rewards. The reward is not a fixed value unlike the CarPole. I think this is caused by randomness of the starting position and velocity of the car even though the agent is following an optimal policy. While researching, I found https://github.com/lantunes/mountain-car-continuous,converges to an average 90 reward while mine had an average 75 reward. If I had more time, I would train with more episodes and try with different parameters (changing the exploration-exploitation rate, learning rate) to reach the higher maximum reward, find the reason why it was converging to a slightly lower value, and find the reason of the non-constant rewards.

4.3 Pong

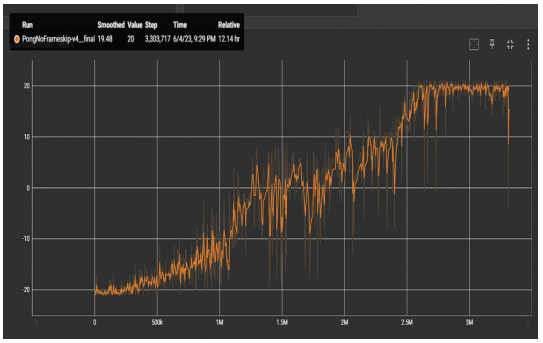

Fig 5. Pong graph

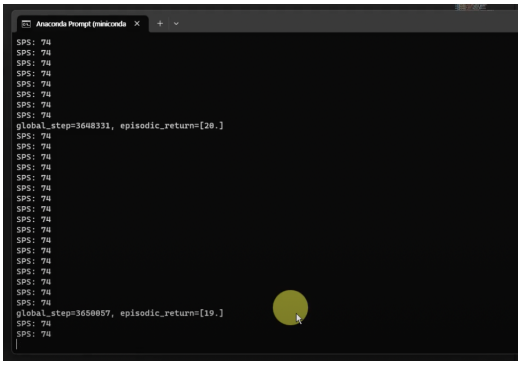

Fig 6. Pong rewards

The Fig 5. And Fig 6. graph show that the Pong game reaches the maximum average reward 20. The goal of the Pong game is to win against the opponent by reaching a score of 21, and the reward is the difference between the scores. The reward converges to 20, and it is stabilised after around 2.6M. I ran out of time so didn't get to train up to 10M. I would like keep training it up 10M and see if there is significant features to notice.

References

[1] EV charging bidding by multi-DQN reinforcement learning in electricity auction market Yang Zhanga , Zhengfeng Zhang b , Qingyu Yanga,c,∗ , Dou Ana,c , Donghe Li a , Ce Li

[2] PRIORITIZED EXPERIENCE REPLAY Tom Schaul, John Quan, Ioannis Antonoglou and David Silver, Available: https://arxiv.org/abs/1511.05952

[3] Susan Amin, Maziar Gomrokchi *, Harsh Satija *, Herke van Hoof *, Doina Precup A Survey of Exploration Methods in Reinforcement Learning, Sep 2021, Available: https://arxiv.org/abs/2109.00157

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, Proximal Policy Optimization Algorithms, Aug 2017, https://arxiv.org/abs/1707.06347v2