

**Training a Bot to Play Games Using Reinforcement Learning**

Abdelrahman Ahmed 19105768

Mohamed Salama 19105743

Ahmed Dahy 19100643

Menna Essam 19104369

Mariam Aly 19104257

ITCS, NU

**Under Supervision of:**   
Dr. Ghada Khoriba

Eng. Amira Tarek

Contents

[1. Problem Statement 3](#_Toc124030285)

[2. Related Work 3](#_Toc124030286)

[3. Game Environment 4](#_Toc124030287)

[3.1 Observations 4](#_Toc124030288)

[3.2 Rewards 4](#_Toc124030289)

[4. Model Architecture 5](#_Toc124030290)

[*4.1 A2C Algorithm* 5](#_Toc124030291)

[*4.2 DQN Algorithm* 6](#_Toc124030292)

[4. Results 7](#_Toc124030293)

# 1. Problem Statement

Reinforcement learning (RL) is a branch of machine learning that focuses on agents operating in the environment to maximize some cumulative reward. The environment rewards an agent's actions to encourage the agent's correct behavior. In this situation, the agent can learn the best strategy on its own. Reinforcement learning methods appeared to be beneficial in a variety of AI-related fields, including robotics, industrial manufacturing, and video games. RL is typically used to address problems involving sequential decision-making. Following the success of the experiment, we will test the applicability of the episodic control RL algorithm in the Pong video game and a Car Game. Pong was one of the first computer games, released by Atari in the early 1970s as a type of arcade game. The game pits two players against each other. Each player controls his or her own paddle, and the goal is to return a ball by hitting it with the paddle while preventing the ball from escaping the playing area, while the goal of the Car Game is to control a car and try to traverse through the track without crashing into the lanes.

# 2. Related Work

This paper describes how to use FRIQ-learning to autonomously control the Pong game. A modest state-space problem can be solved using the FRIQ-learning approach. The final rule-base is built during the simulation based on a reward received for completing the assignment. The system starts with an empty knowledge base. By using the feedback, the environment provides, the approach can discover the necessary rules. The reward-function for the associated problem needs to be precisely defined to be solved (handling of the paddle in the Pong game in this case). once the actions and their results have been determined. The simulation application was created using the FRIQ-learning framework described in the paper. The primary objective of the paper is to demonstrate that the FRIQ-learning approach is appropriate for solving this issue by automatically creating a sparse rule-base for Pong.

# 3. Game Environment

To begin the training process in both games, we first must specify which component in the games the bot should focus on and adjust the weights for the network according to them which are called “Observations”. In the training process we also must specify a reward for every movement the agent takes to help it adjust the weights for the network.

## 3.1 Observations

In pong, the bot had to focus on getting the paddle to hit the ball to the other side. So, the observation parameters had the center of the bot’s paddle, the position of the ball and its velocity. When it comes to the Car game, the bot had to focus on not hitting any of the lanes. So, the observation parameters contained rays that check the distance between each side of the car and the lanes, the cars position, and the position of the next checkpoint on the map.

## 3.2 Rewards

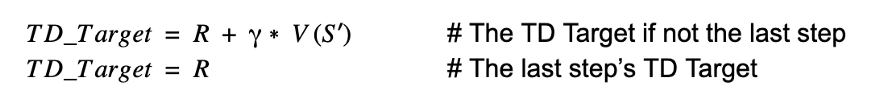
For pong, the bot had to try its best to hit the ball, so the reasonable choice for a reward was to check if the bot is trying to get away from the ball (going up when the ball is in the lower half of the screen or down when the ball is in the upper half), it should get a negative reward and gain a positive reward otherwise, while in the car game we specified specific points on the map as checkpoints. Whenever the bot reached one of these checkpoints it gains a positive reward and get a negative reward if it crashes. This was necessary as sometimes the bot had good runs and almost finishes but crashes at the end line. It used to get a negative reward for the whole run which wasn’t optimal. Only the last few moves had to be punished.

# 4. Model Architecture

## *4.1 A2C Algorithm*

The actor critic algorithm consists of two networks (the actor and the critic) working together to solve a particular problem. At a high level, the Advantage Function calculates the agent’s TD Error or Prediction Error. The actor network chooses an action at each time step and the critic network evaluates the quality or the Q-value of a given input state. As the critic network learns which states are better or worse, the actor uses this information to teach the agent to seek out good states and avoid bad states. In Temporal Difference Learning, agents learn by making predictions about future rewards and adjusting their actions based on prediction error.

**Calculating TD Error:**



Text

Description automatically generated with medium confidenceTo calculate the Advantage Function (TD Error), we need to first calculate the TD Target. In the equation above, the TD Target is the predicted value of all future rewards from the current state S. The function V(s’) represents the Critic Network calculating the value of the next state S’.

In the Advantage Actor Critic algorithm, the Advantage is equal to the TD Error shown above. The Advantage can also be interpreted as the Prediction Error of our agent.Text

Description automatically generated with low confidence

**Actor Network:**

The actor network outputs a probability distribution corresponding to each action. We sample actions from this probability distribution according to each action’s probability. If the action to go left has a value of .8 and the action to go right has a value of .2, we will only choose the left action 80% of the time and the right action 20% of the time.

**Critic Network:**

The critic network maps each state to its corresponding Q-value. The Q-value represents the value of a state where Q represents the Quality of the state.

Unlike the Actor Network which outputs a probability distribution of actions, the Critic Network outputs the TD Target of the input state as a floating-point number.

## *4.2 DQN Algorithm*

Deep Q-Network (DQN) is a type of artificial intelligence algorithm used in reinforcement learning to learn and make decisions in environments with a large number of possible states. It works by using a neural network to approximate the optimal action-value function, which is a mathematical representation of the expected rewards for taking a particular action in each state. The DQN algorithm is based on the Bellman equation, which defines the relationship between the expected reward for an action and the future expected rewards that will result from that action. The Bellman equation can be written as follows:

**Q\*(s, a) = r + γ \* max (Q\*(s', a'))**

where Q\*(s, a) is the optimal action-value function, r is the reward for acting a in states, γ is a discount factor, and Q\*(s', a') is the optimal action-value function for the next state’s' after acting a'. To train the DQN algorithm, the network is given a set of state-action pairs and their corresponding rewards, and it uses these examples to learn the optimal action-value function. The network is then used to make predictions about the expected rewards for taking different actions in new states. To improve the performance of the DQN algorithm, several techniques have been developed, such as experience replay and fixed Q-targets. Experience replay involves storing the agent's past experiences in a memory buffer and sampling from this buffer to train the network, rather than training on the current experience. Fixed Q-targets involve using a separate network to calculate the target values for the training process, rather than using the same network for both prediction and target calculation.

# 4. Results

In Pong, we trained the bot using DQN and A2C each took 1 million iterations of training. At the beginning of training A2C had a good start and the bot started and managed to find a local minimum and stayed there for the rest of the iterations. On the other hand, DQN had a bad start and took longer iterations to train before it managed to find a local minimum. However, when it found that local minimum, it found a better local minimum at the end and managed to survive against the opponent.

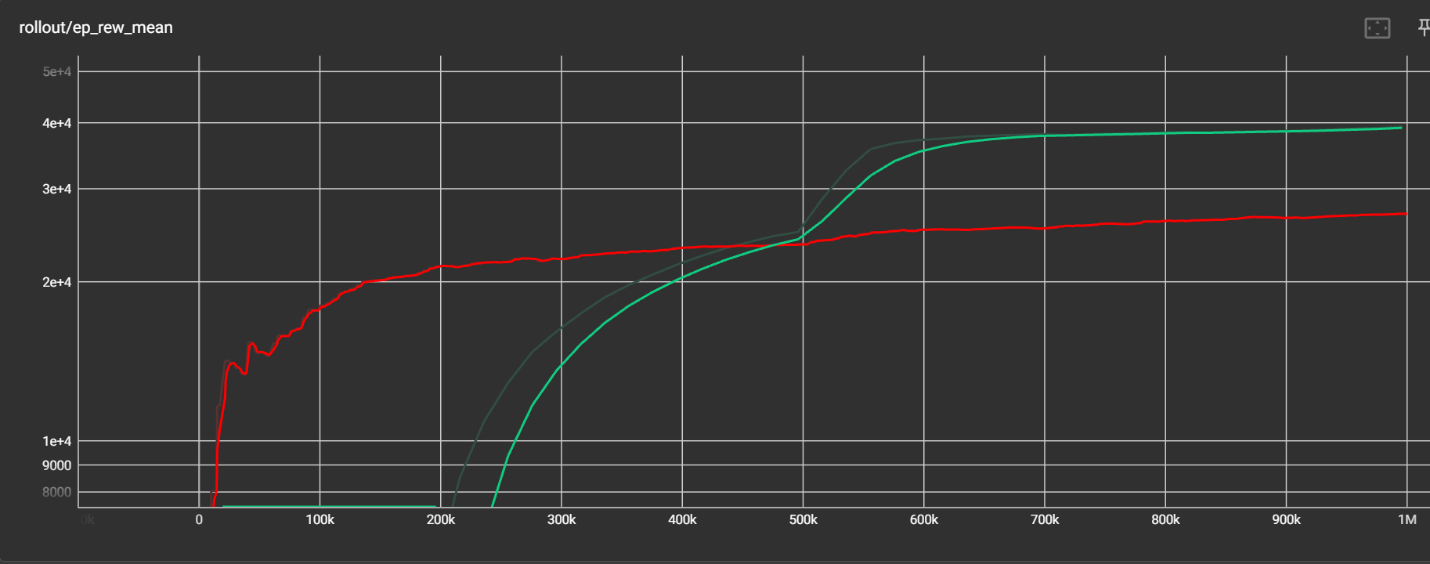


Fig 1: Reward Mean for DQN and A2C

A2C DQN

Diagram

Description automatically generated with medium confidence

Fig 2: Before Training

A picture containing shape

Description automatically generated

Fig 3: After Training

In the Car Game, we used another algorithm called PPO and trained the bot for another 200 thousand iterations. It had a good start, but it kept crashing at the very end. Weighting all the reward for reaching the end didn’t help the bot with training, that’s why we had to add several check points (green lines inside the game).

Chart, line chart

Description automatically generated

Fig 4: PPO Reward Mean

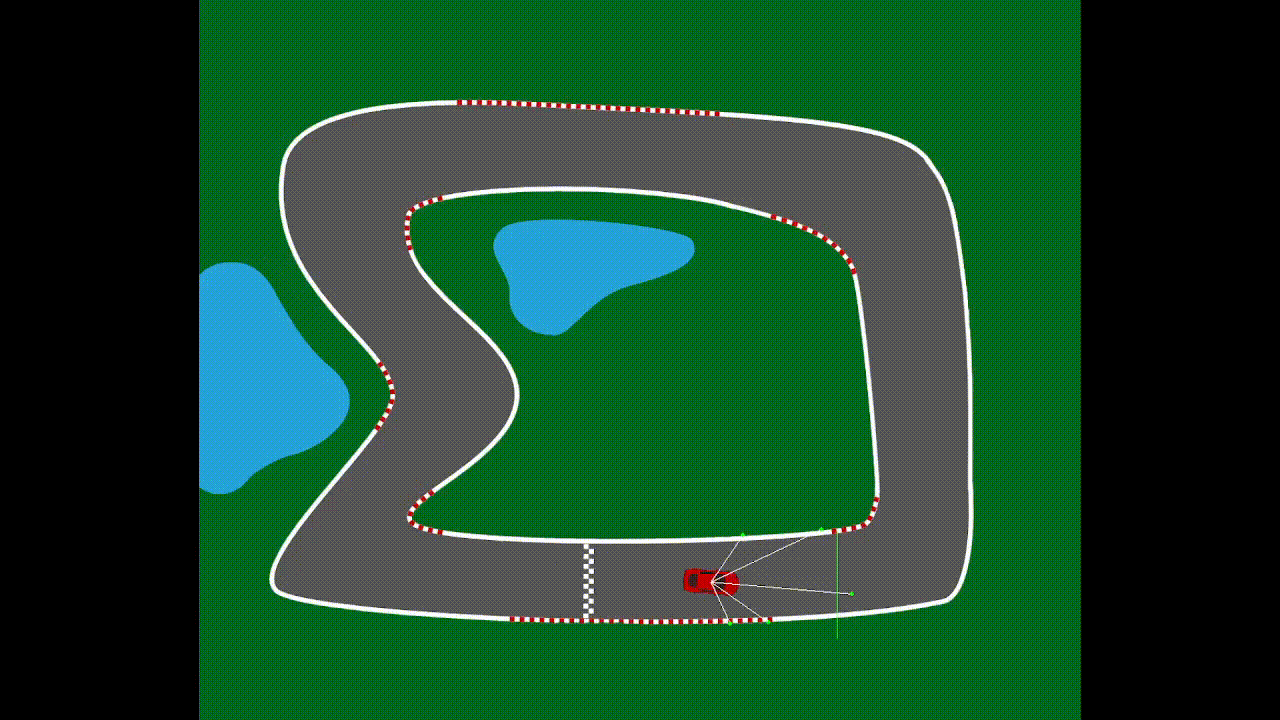


Fig 5: First 50k Iteration of Training

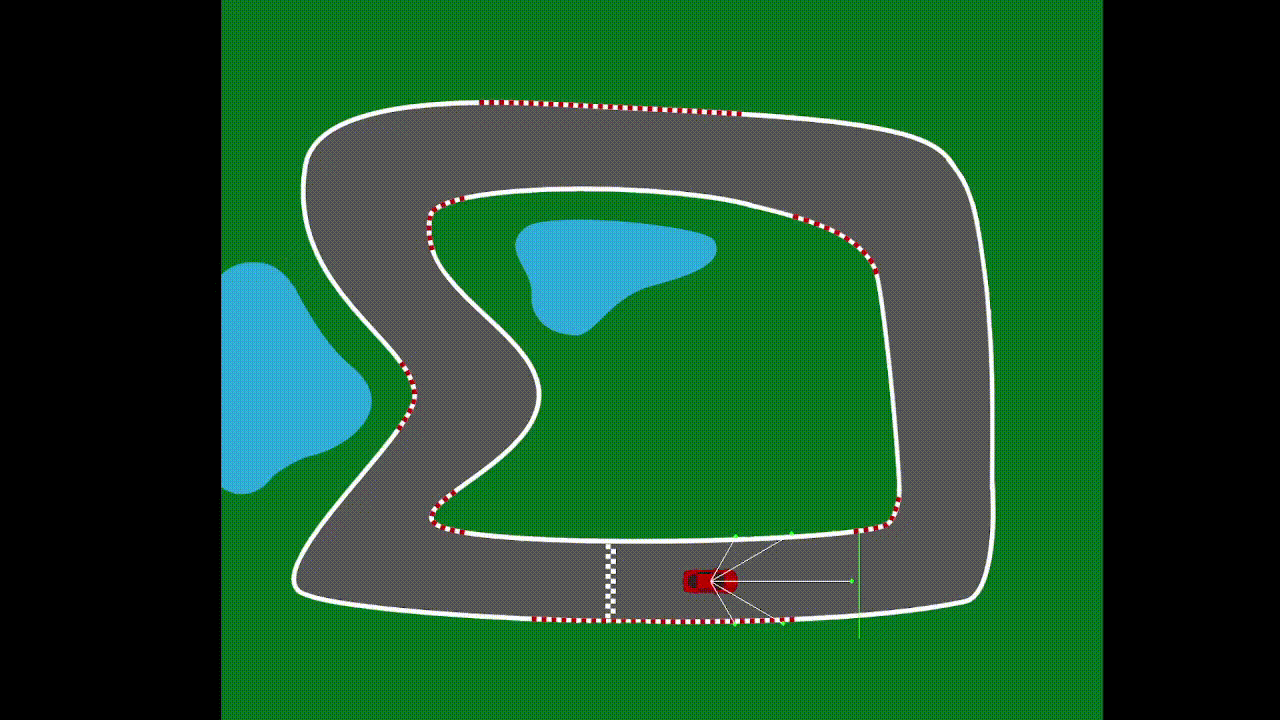


Fig 6: After Training