Master's Degree in Data Science and Scientific Computing

Training an AI Agent in the Game of Briscola

Reinforcement Learning Exam Project

Matteo Nunziante **Noemi Ippolito** **Paolo Vizzo**

Overview

Introduction

The Project

- We will formalize as a RL problem the two - player variant of Briscola, one of the most popular card games in Italy.

- The agent will be trained using the Deep Q Networks algorithm.

- The win rate and average reward will be used as a performance measure to compare the final results.

Briscola Rules

Number of players

Between 2 or 6. They can either play individually or in teams. We will only present the 2 players version.

Rules

Trick-taking game. A player leads by playing a card and the others take turns and try to beat it with a higher card in that suit or a card from the same suit of the "briscola"

Winner

Whoever wins each round leads the next one. To win the game a player, or a team, have to accumulate more points than the other players, or teams.

Steps

Environment

The first necessary step was to clearly define the Reinforcement Learning environment

Agent

Then the choosen algorithm was implemented through the use of the agent

Evaluation

At last, the agent was trained and different tests were performed to evaluate its performance

RLCard

Overview

RLCard is a Python library for reinforcement learning in card games, providing a standardized environment and a collection of pre-implemented algorithms and agents.

Why did we use it?

We chose the RLCard framework because, by creating a new game environment compatible with the toolkit's specifications, we gained access to the extensive repository of pre-existing code, functions, and agents within RLCard.

Each game in the toolkit shared common base classes, each embodying a distinct abstract concept.

Game Design

Player

- **Definition**: The active participant in the game
- **Player Class Role**:
    - Player initialization
    - Keeping track of personal information (player_id/score)

Game

- **Definition**: A complete sequence from non-terminal to terminal states
- **Game Class Role**:
  - **Initialization** (players/deck)
  - **Evolution Management** (round progression, payoffs)

Round

- **Definition**: A segment within the game sequence
- **Round Class Role**:
    - Execution of single round
    - Player turn management
    - Score update at round end

Dealer

- Definition: Handles deck management
- Dealer Class Role:
    - Initialization (Italian card deck)
    - Card shuffling
    - Briscola setting
    - Card dealing management

Judger

- **Definition**: Major decision-maker in a round or end of a game
- **Judger Class Role**:
    - Implementing scoring system
    - Determining Round Winner
    - Determining Game Winner

State Representation

State Representation

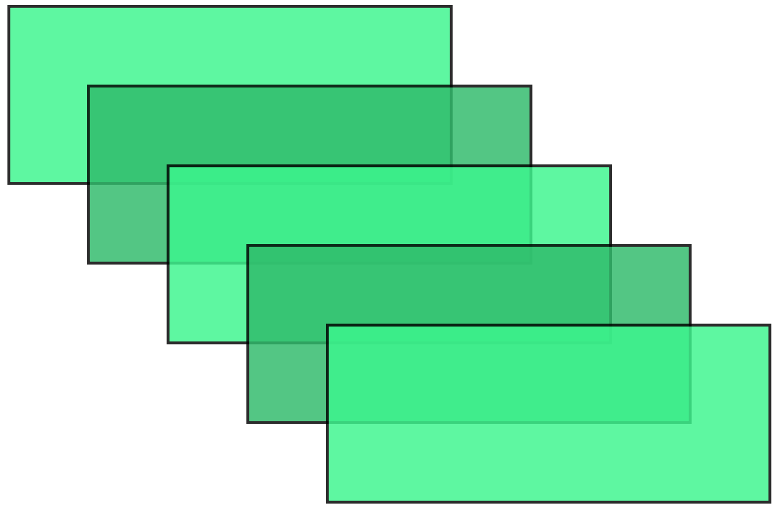
- **Definition**: Information observable by one player at a specific game time step
- **Toolkit Standard:** State represented as a dictionary with two values
    - Observations
    - Legal Actions list

Observation Encoding

Observations were encoded into a Matrix of 5 Card Planes, one-hot encoded in a 4 by 10 grid

- **Player's current Hand**
- **Table Cards**
- **Briscola Card**
- **Face down pile in front of Player 0**
- **Face down pile in front of Player 1**

5x4x10



Action Encoding

Action Encoding

- **Definition**: Conversion of specific game actions into action indices
- **Format:** Positive integers, from 0 to 39
- **Correspondence:** Each index represents a unique game action
- **Legal Actions:** Represented as a list of action indices. They are the only actions an agent can choose from and are comprised of the cards in the player’s hand

Rewards

Per Game

- Rewards are assigned when the game is over.
    - +1 for Winning
    - -1 for Losing
    - 0 for a Draw

Per Round

- Rewards are assigned at the end of each round. The points of the cards on the table are normalized and then they are assigned as follows

    - **+ Normalized points** for the Winner of the Round
    - **- Normalized points** for the Loser of the Round

Setting

Setting

Incomplete information: a game in which players do not have full knowledge of the state of the game.

Model Free Learning: unavailable model of the environment

Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

$$A_t = \begin{cases} arg\max_a Q(s_t, a) & \text{with probability } 1 - \epsilon \\ \text{random } a & \text{with probability } \epsilon \end{cases}$$

Number of states

- It is crucial in RL to determine the **number** of states for the given problem:

$$\#\text{States} > 40 \cdot \binom{39}{3} \cdot 37 \cdot 2 \sum_{i=1}^{18} i \binom{36}{2i} \approx 10^{17}$$

Deep Q Networks Algorithm

Benefits

- Train RL agents with large neural networks
- Learn directly from high dimensional raw inputs
- Decrease unstability and divergence of NN training in RL framework

Experience Replay

- Store the agent's **experience** at each time step in the **replay memory**
- Apply **offline minibatch** updates to **samples** of experience

Data efficiency

Breaks correlations

Smoothing out Learning

Target Network

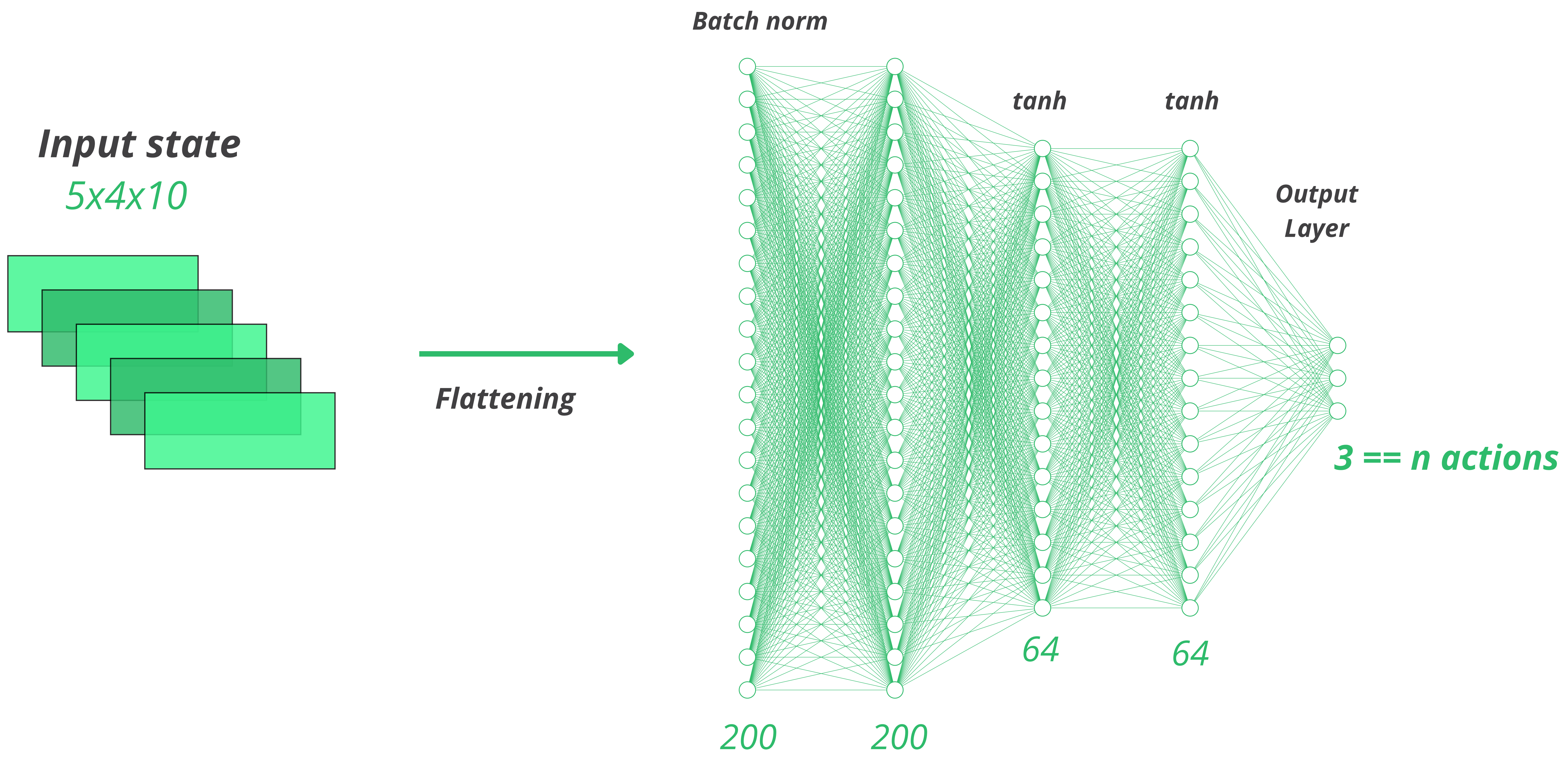
- **Use separate net to generate the targets**
- **Clone the Q network every C updates**

Improve stability

Reduce correlations with targets

Avoid divergence

Q Networks Architecture



Deep Q Networks

Pseudocode

- **Preprocess** function
- **Stocastic** gradient updates
- **Uniform** sampling
- **Limited** memory size

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Training

Steps*

1. **Train a intial model vs random agent for 500.000 games**

2. **Learning the final model from scratch against the initial model (500.000 games)**

3. **Evaluation of final trained model:**
   - **500 games versus random and initial agent**
   - **30 games versus human agent**

*Repeated for each version of payoffs

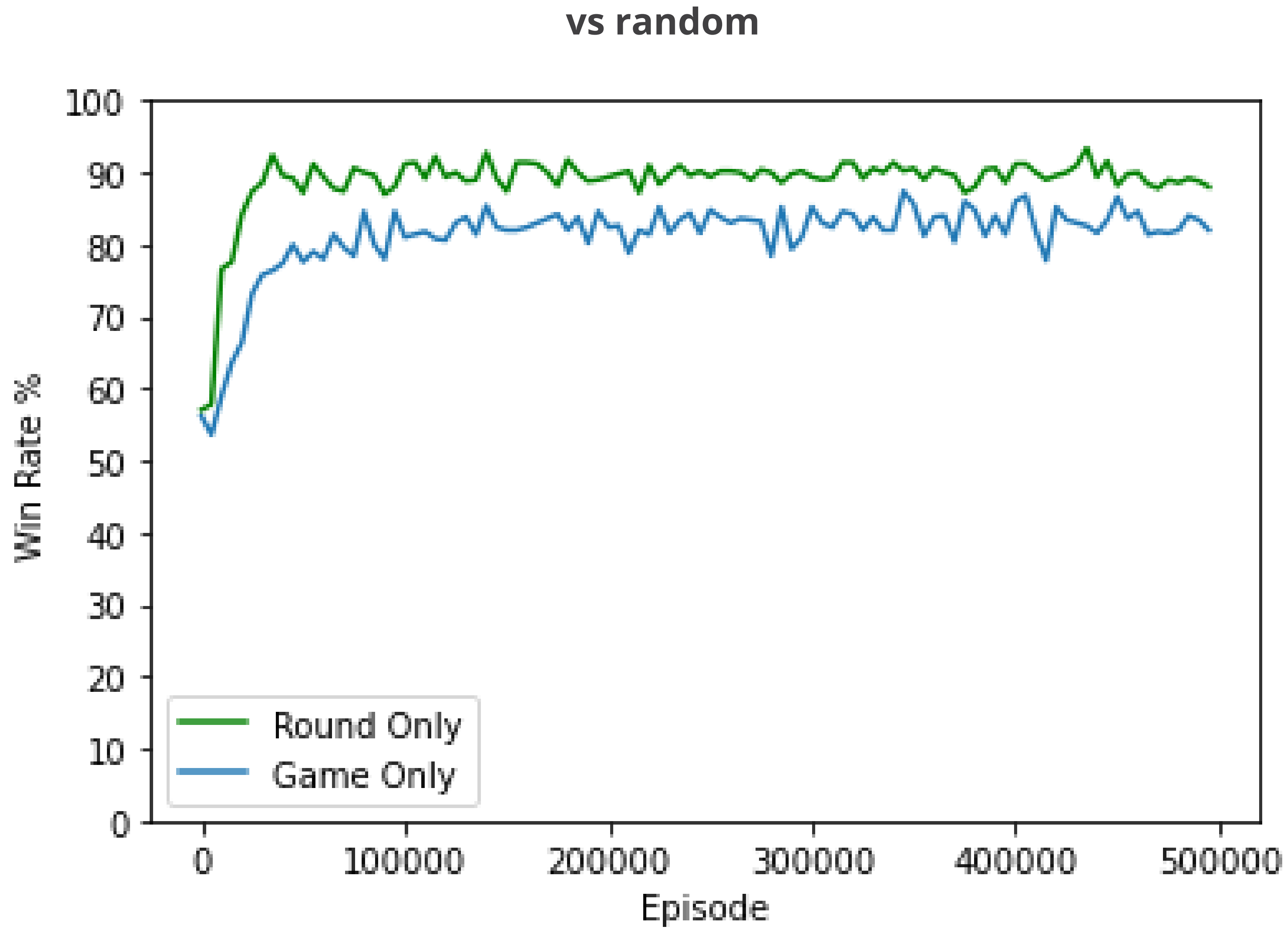
Hyperparameters

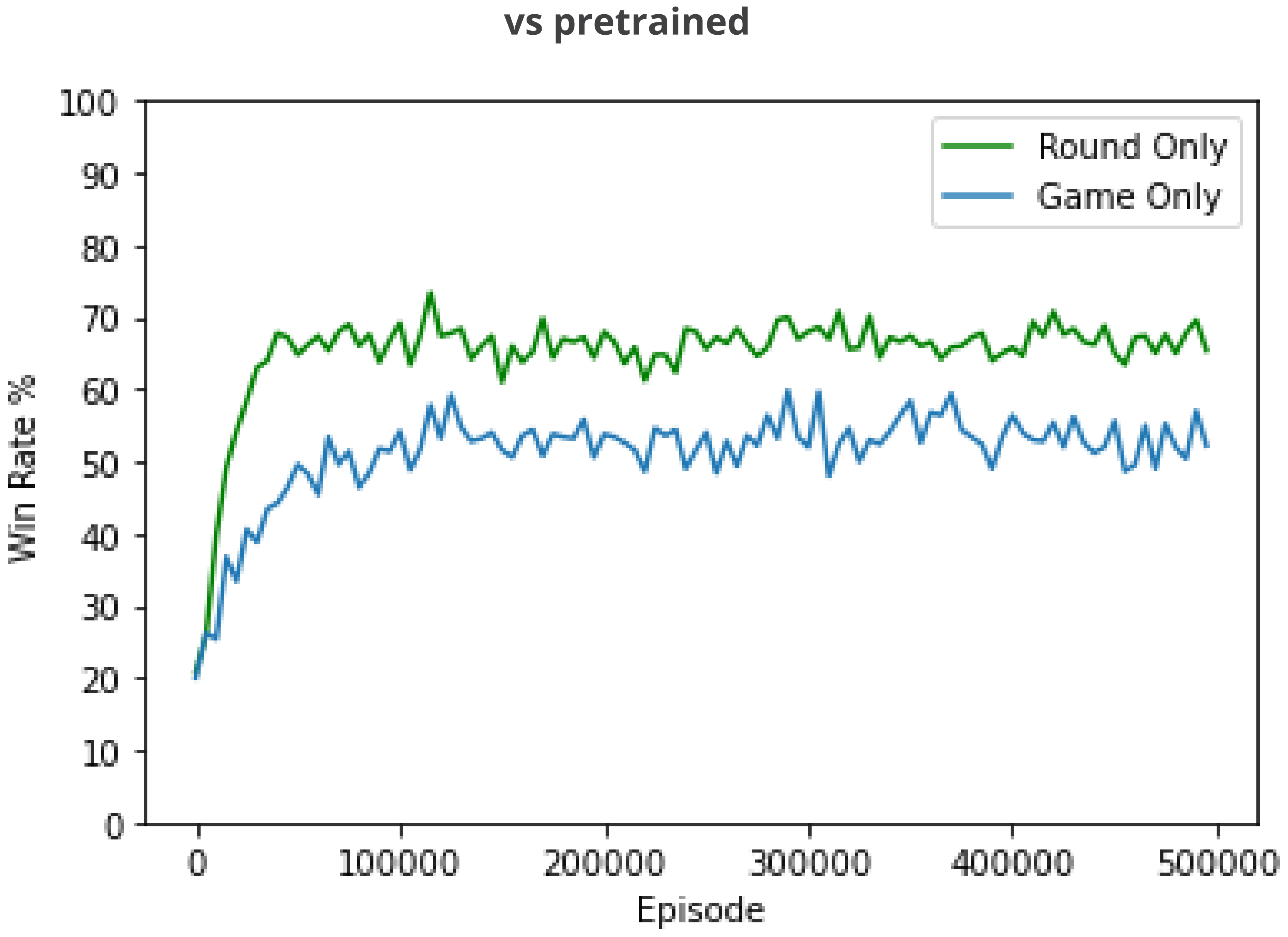
- **Replay memory size** = 500000
- **Target network update** = 10000
- **Discount factor** = 1
- **Epsilon start** = 1.0
- **Epsilon end** = 0.1
- **Epsilon decay steps** = 1000000
- **Batch size** = 64
- **Learning rate** = 0.00025

Results

Results

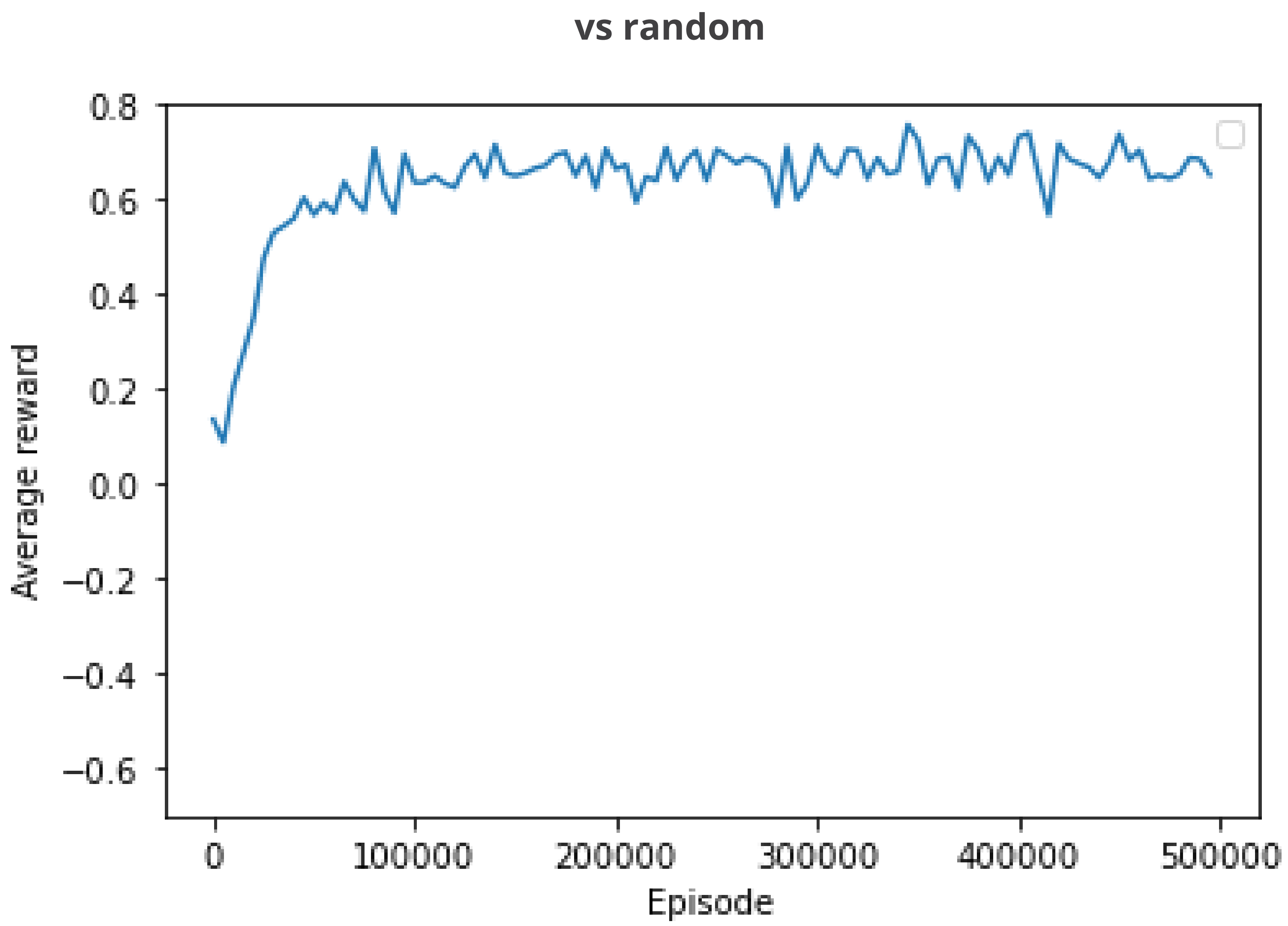
Win Rate

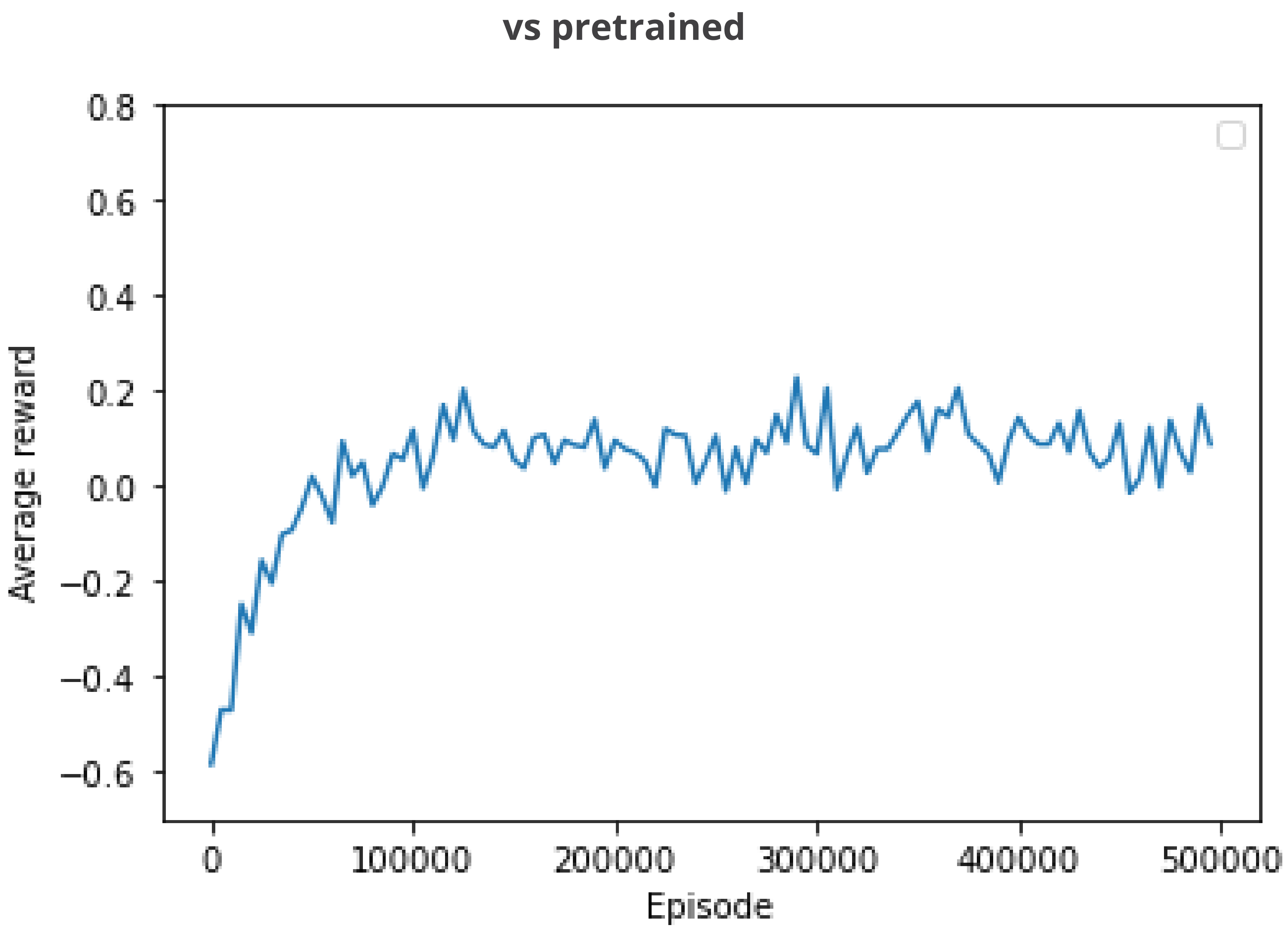




Results

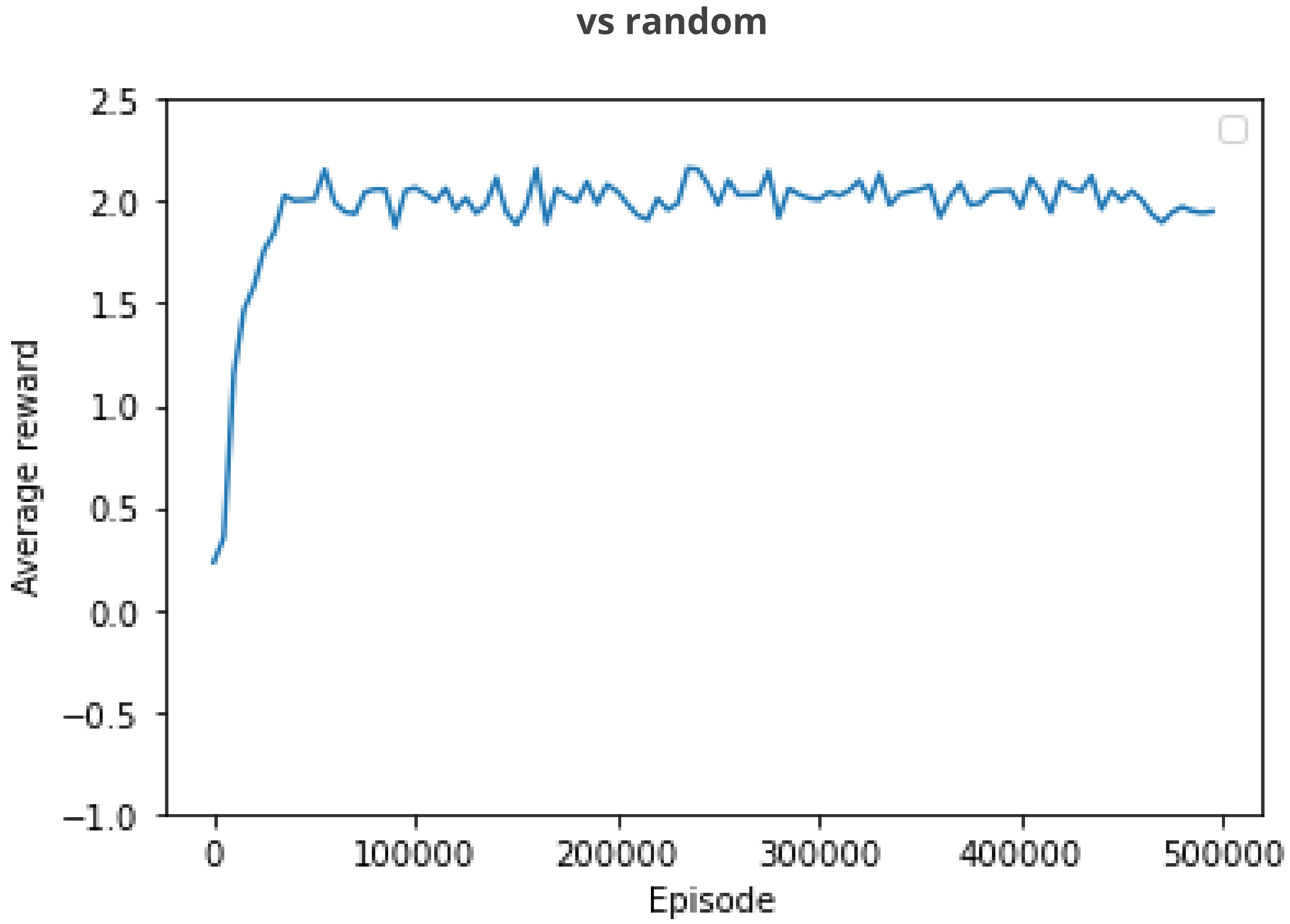
Average game-only reward

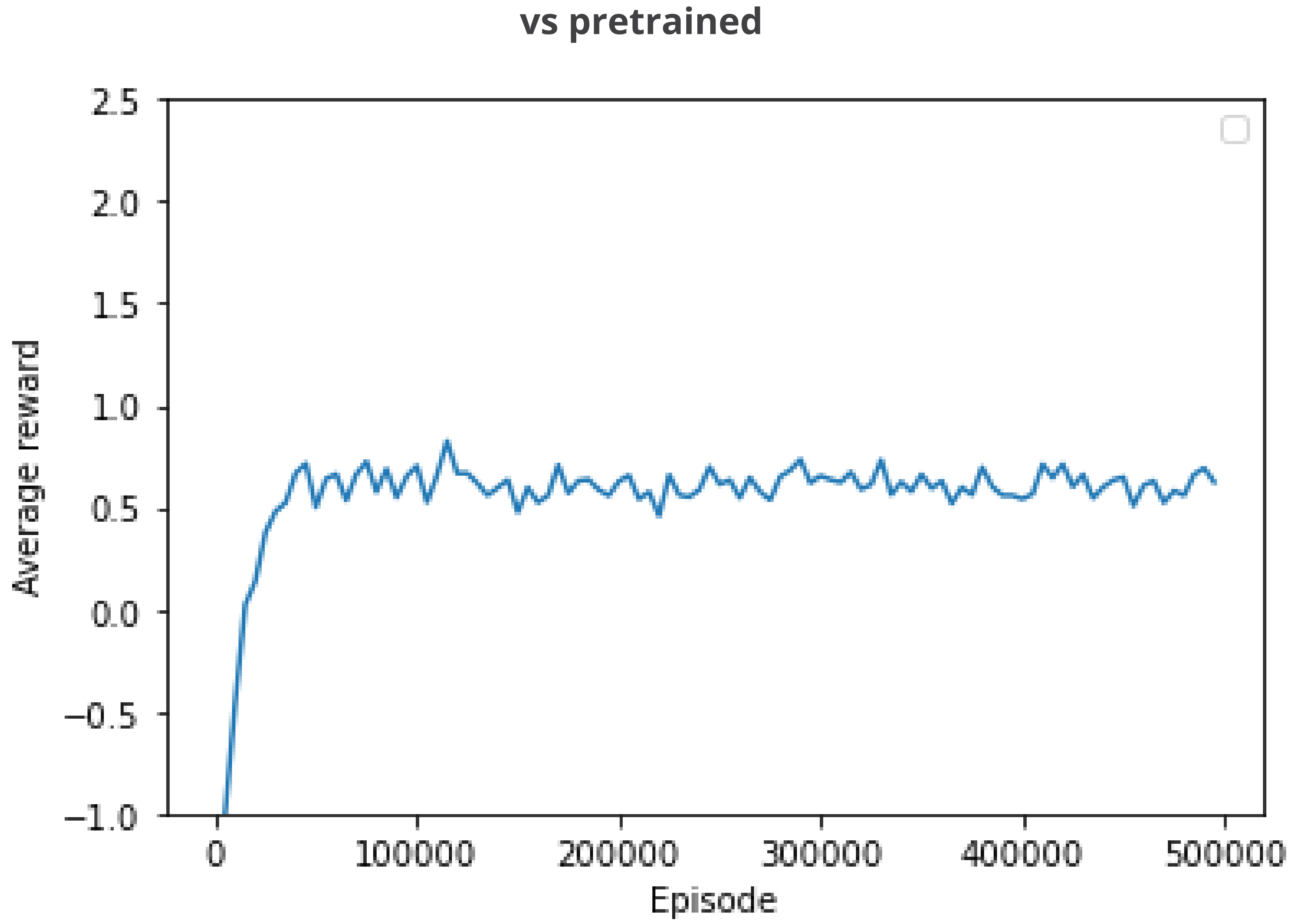




Results

Average round-only reward





Human Trials

Human Evaluation

- We introduced a single-agent interface, where the other player is simulated by pre-trained models
- Some tests were performed. 50 games were played against our best performing agent
- The results showed our agent winning 37% of games

Try it Yourselves!

Reference

01 **RLCard: A Platform for Reinforcement Learning in Card Games**, D. Zha et al. (2020)

02 **Reinforcement Learning: An Introduction**, A. Barto and R.S. Sutton (2018)

03 **Playing Atari with Deep Reinforcement Learning**, Google Deepmind (2013)

04 **Human-level control through deep reinforcement learning**, Google Deepmind (2015)

05 **Mastering Cooperative, Incomplete Information Board Games by Self-Play**, W. van der Weij (2022)