Collaboration and Competition Project: Tennis Environment

Marc Poirier

May 9, 2025

1 Introduction

This report presents the implementation of a multi-agent reinforcement learning solution for the Tennis environment, a Unity ML-Agents simulation where two agents control rackets to keep a ball in play. The objective is to train the agents to achieve an average score of at least +0.5 over 100 consecutive episodes, using the maximum score between the two agents per episode. The solution employs the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm, leveraging PyTorch for neural network implementation and training.

2 Implementation Description

The implementation is structured into several key components:

- **Environment Setup**: The Unity Tennis environment is initialized, providing a state space of 24 dimensions per agent (position and velocity of the ball and racket) and a continuous action space of 2 dimensions (movement toward/away from the net and jumping), bounded between -1 and 1.
- **Neural Networks**: Each agent uses an actor network to map states to actions and a critic network to estimate Q-values based on states and actions of all agents.
- **Replay Buffer**: A shared replay buffer stores experiences (states, actions, rewards, next states, dones) for both agents, enabling off-policy learning.
- **MADDPG Framework**: The MADDPG class coordinates two agents, managing action generation, experience storage, and learning updates.
- **Training Loop**: Agents interact with the environment over episodes, updating their policies and tracking performance until the success criterion is met.

The environment is solved when the average score over 100 episodes reaches +0.5, after which model weights are saved, and a plot of scores is generated.

3 Learning Algorithm

The MADDPG algorithm adapts the Deep Deterministic Policy Gradient (DDPG) method for multi-agent scenarios, allowing each agent to learn a decentralized policy while accounting for the actions of the other agent. Key features include:

- **Actor-Critic Framework**: Each agent has an actor network (policy) and a critic network (value function). The actor outputs continuous actions, while the critic evaluates Q-values using the full state and actions of both agents.
- **Shared Replay Buffer**: A common buffer of size 100,000 stores experiences, from which batches of 256 are sampled for training.

- **Exploration**: Gaussian noise is added to actions, with a reduction rate of 0.99 applied after 500 episodes, scaling from 1.0 to a minimum of 0.1.
- **Soft Updates**: Target networks are updated with a parameter $\tau = 0.001$ to stabilize learning.
- **Learning Frequency**: Updates occur every step after 500 episodes, with three updates per step to enhance learning efficiency.

3.1 Hyperparameters

The chosen hyperparameters are:

- Actor Learning Rate: 1×10^{-4}
- Critic Learning Rate: 3×10^{-4}
- Discount Factor (γ): 0.99
- Soft Update Parameter (τ): 0.001
- Replay Buffer Size: 100,000
- Batch Size: 256
- Noise Reduction Rate: 0.99 (after 500 episodes)
- Initial Noise Scale: 1.0
- Minimum Noise Scale: 0.1
- Episodes Before Training: 500
- Update Frequency: Every 1 step, with 3 updates per step

4 Model Architectures

4.1 Actor Network

The actor network maps an agent's 24-dimensional state to a 2-dimensional action:

- **Input Layer**: 24 units (state size)
- **Hidden Layer 1**: 256 units, ReLU activation, batch normalization
- **Hidden Layer 2**: 128 units, ReLU activation, batch normalization
- **Output Layer**: 2 units (action size), tanh activation

Weights are initialized uniformly using a fan-in-based range, with the output layer between -3×10^{-3} and 3×10^{-3}.

4.2 Critic Network

The critic network estimates Q-values from the full 48-dimensional state (concatenated states of both agents) and actions:

- **Input Layer**: 48 units (full state size)
- **Hidden Layer 1**: 256 units, ReLU activation, batch normalization
- **Hidden Layer 2**: 128 units, ReLU activation (after concatenating 4 action units)
- **Output Layer**: 1 unit (Q-value)

Weight initialization mirrors the actor network.

5 Results

The agents were trained for up to 3,000 episodes. The environment was solved in approximately 2,050 episodes, achieving an average score of +0.5 over 100 consecutive episodes (using the maximum score of the two agents per episode). The training loop tracked both individual episode scores and the 100-episode average, as shown below:

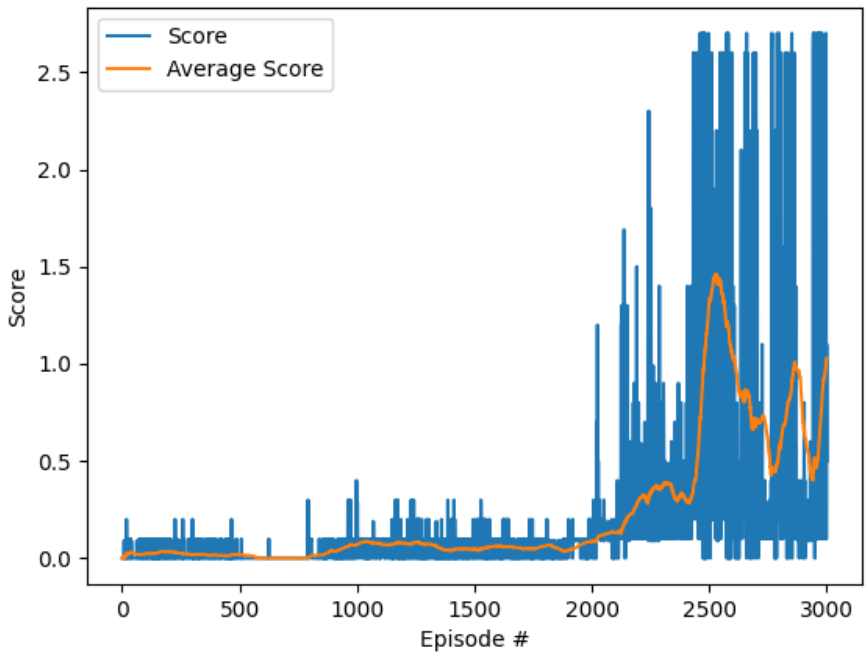

Figure 1: Plot of rewards per episode (blue) and 100-episode average score (orange), with the environment solved in 2449 episodes.

Figure 1 illustrates steady improvement, with the average score surpassing +0.5 at episode 2449 even reaching an average score of 1.46260 at episode 2530, indicating successful learning and coordination between the agents.

6 Future Ideas for Improvement

To enhance performance, the following strategies could be explored:

- **Hyperparameter Optimization**: Adjust learning rates, batch sizes, or noise parameters to accelerate convergence or improve stability.
- **Prioritized Experience Replay**: Prioritize sampling of impactful experiences to increase learning efficiency.
- **Advanced Noise Strategies**: Replace Gaussian noise with Ornstein-Uhlenbeck noise or parameter space noise for better exploration.
- **Agent Communication**: Enable agents to share information (e.g., intended actions) to improve coordination.
- **Curriculum Learning**: Start with simpler conditions (e.g., slower ball speed) and progressively increase difficulty.

These enhancements could reduce the episodes required to solve the environment, improve final scores, or enhance robustness.