Collaboration and Competition (Tennis) — MADDPG Report

Author: Marco Abramo

Date: 2025-09-26

Overview

We solve the Unity *Tennis* environment using **MADDPG** with decentralized actors and centralized critics. Two agents cooperate to keep the ball in play. Per Udacity's criterion, the project is considered *solved* when the 100-episode moving average of the *per-episode maximum* agent score is ≥ 0.5.

1 Environment

- **Agents:** 2
- **State (per agent):** 24 dims (8 observations $\times$ 3-frame stack)
- **Action (per agent):** 2-D continuous, range $[-1, 1]$
- **Reward:** +0.1 for a successful return; small negatives if the ball hits the ground or goes out
- **Episode termination:** when either agent is done
- **Metric:** for each episode, take max of the two agents' scores; track a 100-episode moving average

2 Learning Algorithm: MADDPG

For each agent $i \in \{1, 2\}$, we learn an actor $\mu_{\theta_i}(s_i)$ mapping local state s_i to a continuous action a_i, and a *centralized* critic $Q_{\phi_i}(s_1, s_2, a_1, a_2)$ that observes the joint state-action.

Critic target (TD):

$$y_i \;=\; r_i \;+\; \gamma\, Q_{\phi_i'}(s_1', s_2', a_1', a_2'), \qquad a_j' \;=\; \mu'_{\theta_j}(s_j'), \tag{1}$$

and we minimize the MSE: $\mathcal{L}_{\text{critic},i} = \big(Q_{\phi_i}(s_1, s_2, a_1, a_2) - y_i\big)^2$.

Actor update (deterministic policy gradient):

$$\nabla_{\theta_i} J \;\approx\; \mathbb{E}\Big[\nabla_{a_i} Q_{\phi_i}(s_1, s_2, a_1, a_2)\big|_{a_i=\mu_{\theta_i}(s_i)}\; \nabla_{\theta_i}\mu_{\theta_i}(s_i)\Big]\,, \tag{2}$$

treating the other agents' actions as fixed during agent i's policy gradient.

Target networks (soft updates):

$$\phi_i' \leftarrow \tau\phi_i + (1-\tau)\phi_i', \qquad \theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'. \tag{3}$$

3 Network Architectures

Let $S_{\text{all}} = 2 \times 24$ and $A_{\text{all}} = 2 \times 2$.

- **Actor** (per agent): Linear(24 $\to$ 256) $\to$ ReLU $\to$ Linear(256 $\to$ 256) $\to$ ReLU $\to$ Linear(256$\to$2) $\to$ `tanh` (actions scaled to $[-1, 1]$).
- **Centralized Critic** (per agent): Linear($S_{\text{all}}$$\to$256) $\to$ ReLU $\xrightarrow{\text{concat } A_{\text{all}}}$ Linear(256+$A_{\text{all}}$$\to$256) $\to$ ReLU $\to$ Linear(256$\to$1).

4 Implementation Details

- **Replay Buffer (shared):** stores concatenated joint states/actions and *per-agent* reward/done vectors.
- **Credit assignment (fix):** each critic trains with its own (r_i, done_i); averaging rewards over agents stalled learning.
- **Exploration:** OU noise per agent with σ decaying over episodes; a 200-episode random warmup (uniform actions in $[-1, 1]$) seeds diverse transitions.
- **Stability:** 4 updates per env step; critic gradient clipping ($\|g\| \le 1$); critic L2 regularization; actions cast to `float32` before stepping into Unity.
- **Checkpoints:** saved only when the success criterion is reached, to keep the folder clean.

5 Hyperparameters

Component	**Value**
Discount γ	0.99
Soft update τ	1e$-$3
Replay buffer size	1e6
Batch size	256
Actor learning rate	1e$-$4
Critic learning rate	1e$-$3
Critic L2 (weight decay)	1e$-$5
Updates per step	4
Random warmup	200 episodes (uniform $[-1, 1]$)
OU noise σ decay	0.30 $\to$ 0.05 over first 1000 episodes
Critic grad clip	1.0
Seed	1

6 Results

- **Solved in 995 episodes**.
- **100-episode moving average at solve:** 0.501.

Console at convergence:

```
 Solved in 995 episodes! 100-episode average: 0.501
Saving checkpoints...
```

Evaluation. A separate script loads the saved checkpoints and runs 100 deterministic episodes (noise OFF). If available, the plot `scores_eval.png` is included below.

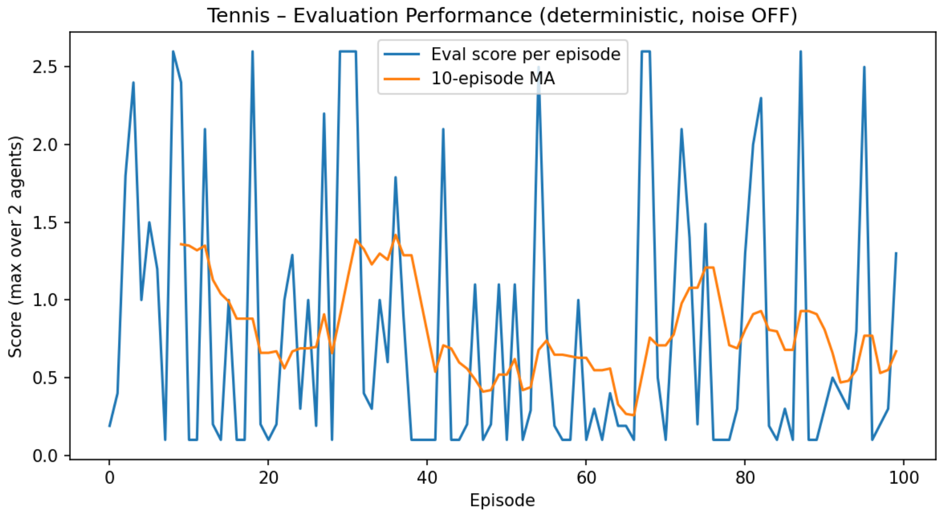

Figure 1: Deterministic evaluation over 100 episodes (noise OFF).

7 Ablations / What Did Not Work Initially

- **Averaging rewards** across agents for critic targets ⇒ flat learning (≈ 0.03–0.05).
- Passing **float64** actions to Unity ⇒ long stretches of zero scores; casting to `float32` fixed it.
- **Too-aggressive actor LR** destabilized coordination; reducing to 1e−4 improved stability.

8 Reproducibility & How to Run

- Dependencies: `numpy`, `torch`, `matplotlib`, `unityagents`
- Place the Unity binary at `envs/Tennis_Windows_x86_64/Tennis.exe` (or update `ENV_PATH`).
- Train:

  ```
  python train.py
  ```

- Evaluate (deterministic, noise OFF):

  ```
  python eval.py
  ```

9 Further Work

Algorithmic upgrades.

- **TD3-style MADDPG:** twin critics, target policy smoothing, and delayed policy updates to reduce overestimation bias and stabilize training.
- **Parameter sharing** across actors (and optionally critics) with *role/agent encodings* to improve sample efficiency and generalization.
- **Prioritized Experience Replay (PER)** with importance sampling to focus updates on informative transitions.
- **n-step returns / TD(λ)** to propagate reward information more quickly through time.
- **Normalization layers & value scaling:** LayerNorm in hidden layers; PopArt/mean–std value normalization for robust critic targets.
- **Exploration variants:** parameter-space noise, adaptive OU σ, or clipped Gaussian noise with schedule.
- **Attention-based critics (MAAC)** for scalable centralized value functions when the number of agents grows.

Stability & regularization.

- Entropy or action regularization on actors; spectral norm or weight decay sweeps for critics.
- Observation/reward normalization (running mean–std), and target network update schedule ablations.

Partial observability.

- **Recurrent policies/critics** (LSTM/GRU) to capture temporal dependencies beyond frame stacking and improve credit assignment.

Data efficiency & scaling.

- Increase #updates per env step with smaller batches; try distributed rollouts or asynchronous data collection.
- Population-Based Training (PBT) or automated schedules for LR, noise σ, and τ.

Evaluation & reproducibility.

- Multiple random seeds with confidence intervals; component ablations (PER, twin critics, normalization, recurrence); standardized logging and checkpoints.

10 References

- Ryan Lowe et al., *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (MADDPG)*, NeurIPS 2017.
- Timothy P. Lillicrap et al., *Continuous control with deep reinforcement learning (DDPG)*, 2015.