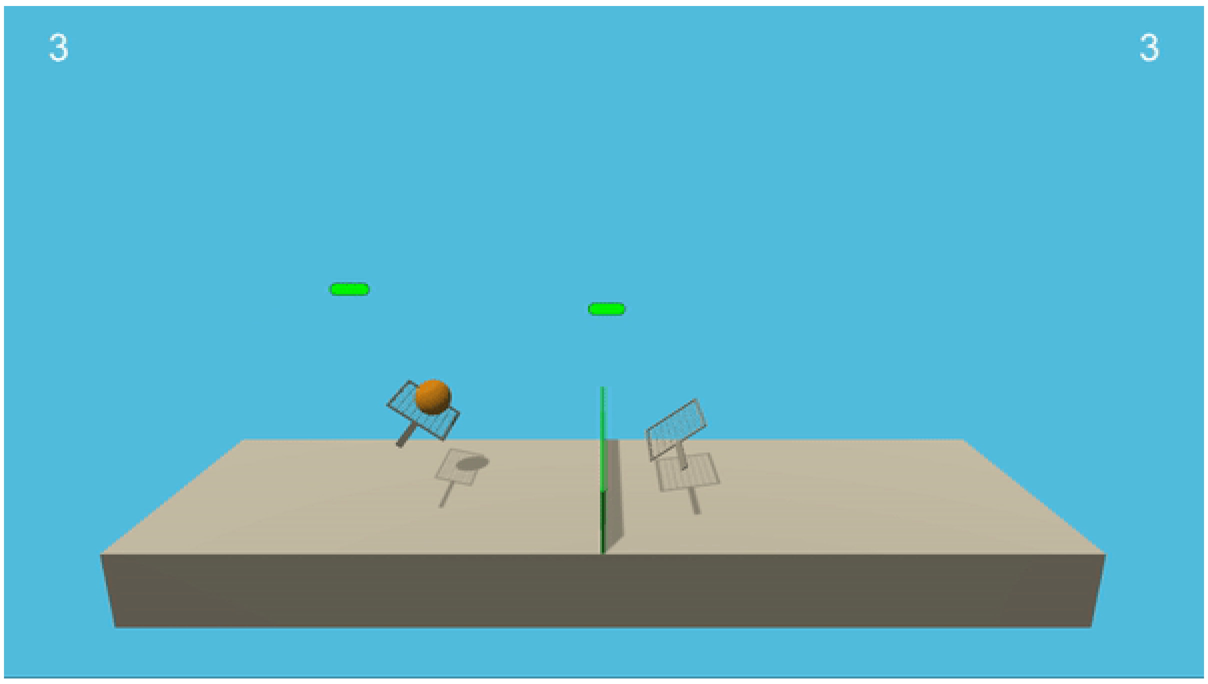
Deep Reinforcement Learning

Project 3: Collaboration and Competition



Simon FERRAND

School : Udacity

Course name : [Deep Reinforcement Learning anodegree]

21/02/24

Abstract

In this project, we tackled the challenge of the "Tennis" environment from the Udacity Deep Reinforcement Learning Nanodegree, employing a Multi-Agent Deep Deterministic Policy Gradient (MADDPG) approach. This involved adapting DDPG algorithms to support cooperative and competitive interactions between two agents with the goal of maintaining a ball in play. Our approach, grounded in shared learning and exploration of advanced multi-agent coordination techniques, culminated in achieving the target average score, showcasing the effectiveness of collaborative reinforcement learning strategies in a controlled setting and setting a foundation for further exploration in complex multi-agent scenarios.

Introduction

In the evolving landscape of Artificial Intelligence, Deep Reinforcement Learning (DRL) has emerged as a pivotal methodology for training agents capable of making autonomous decisions. Project 3, "Collaboration and Competition," from the Deep Reinforcement Learning Nanodegree by Udacity, presents a unique challenge that extends beyond individual agent performance to include multi-agent dynamics. This project utilizes the "Tennis" environment, where two agents control rackets to hit a ball over a net. The objective is not just to keep the ball in play but to do so in a manner that is both cooperative and competitive, illustrating the complex interplay between agents.

Building on the foundation laid in Project 2, this report explores the adaptation of the Deep Deterministic Policy Gradient (DDPG) algorithm to a Multi-Agent Deep Deterministic Policy Gradient (MADDPG) framework. Through extensive experimentation with various hyperparameters, this study seeks accelerates the learning process towards achieving the project goal: an average score of +0.5 over 100 consecutive episodes over (after taking the maximum over both agents).

I) Methodology

1.1) Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm

The Multi-Agent Deep Deterministic Policy Gradient (MADDPG) is an extension of the DDPG algorithm, tailored for environments where multiple agents are learning simultaneously. This algorithm maintains a central critic that learns a value function based on the observations and actions of all agents, promoting stationary policies during training. Each agent utilizes its own local observation to execute its policy during training and deployment.

In this project, we employ the DDPG agent developed in Project 2, leveraging shared experience replay, and the model architectures for the critic and actor networks, as outlined in the project's detailed algorithmic description. This approach allows for coordinated learning among agents, with each agent contributing to a shared knowledge base, thereby enhancing the overall system's efficiency.

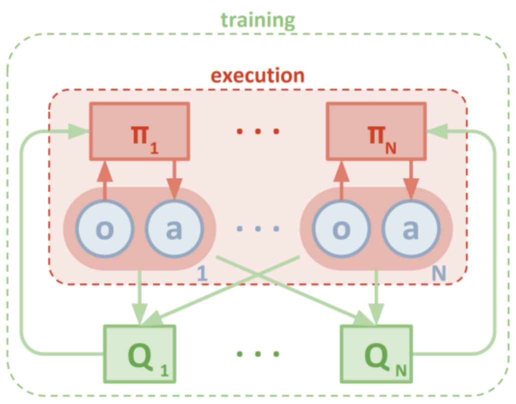


Figure: Overview of our multi-agent decentralized actor, centralized critic approach

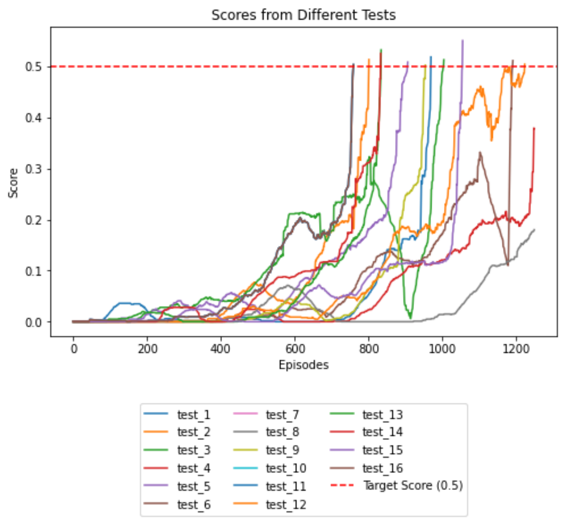
1.2) Code Evolution and Feature Integration

In the "Code Evolution and Feature Integration" section of the project report, we highlight the enhancements made from the Project 2 codebase. Retaining functionalities such as exponential noise decay and Learning Iteration Enhancement, we've integrated Prioritized Experience Replay (PER) and robust Performance Indicators Monitoring. New additions include the control of exponential noise decay over time, a centralized plotting utility in utils.py, and a systematic approach for saving each training session in the 'tests' directory. This allows for detailed subsequent performance analysis, ensuring that every aspect of the training process is captured and can be evaluated to inform future improvements. The scoring method has been modified to fit the requirements of this project.

II) Results

2.1) Experimentation and Testing

The sixth test achieves the goal the fastest, completing it in just 760 episodes. The model and the hyperparameters are presented in the next section.



1.2) Presentation of the best training (test_6)

The detailed presentation of the model architecture and hyperparameters of the algorithm that achieved the objective most rapidly will be provided.

a) Model Architecture

Actor Network: Designed to determine the optimal action given the current state, featuring a fully connected neural network with two hidden layers. The first layer contains 164 units, while the second encompasses 128 units, both utilizing the Leaky ReLU activation function for non-linearity. The output layer consists of 4 nodes, applying the tanh activation function to ensure the output action values are within the required range of -1 to 1.

Critic Network: Evaluates the potential of the chosen actions by estimating their Q-values. Similar to the actor, it employs a fully connected neural network with two hidden layers of 164 and 128 units, respectively. The critic's output layer is designed to produce a single value, indicating the expected return of the state-action pair, without applying any activation function to the final output.

b) Training Configuration and Hyperparameters

Maximum Episodes and Timesteps: The training regimen is structured to run for a maximum of 3500 episodes, with each episode allowing up to 1000 timesteps. This setup ensures ample opportunity for the agent to interact with the environment, learn from a wide range of situations, and refine its policy over time.

Experience Replay: The model utilizes an experience replay buffer with the capacity to store 1,000,000 tuples, facilitating efficient learning by breaking the correlation of sequential experiences.

Batch Size: 1024 - At each learning step, 1024 tuples are sampled from the experience replay buffer to update the network weights.

Learning Rate: Both the actor and critic networks are trained with a learning rate of 0.001, enabling gradual but steady learning.

Soft Updates: The model employs soft updates for the target networks with a tau (τ) of 0.001, ensuring the target networks are gradually adjusted to the learned networks, promoting stability.

Discount Factor (Gamma): Set at 0.99, this factor discounts future rewards, balancing immediate and future gains.

Noise: To encourage exploration, a noise process with an initial sigma value of 0.2 is incorporated into policy execution. This exploration is exponentially tapered, progressively shifting the emphasis towards exploitation over time, while maintaining a baseline level of exploration by setting a minimum sigma threshold of 0.01 reached after 800 episodes.

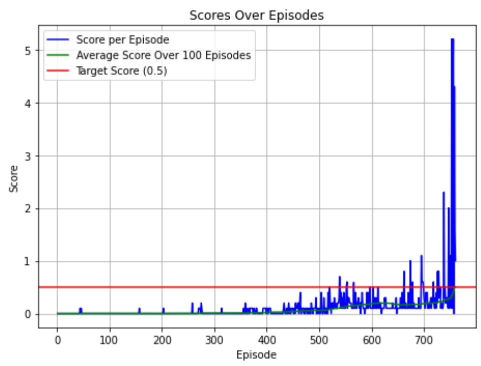
Prioritized Experience Replay (PER): The dynamic Beta parameter in the PER mechanism, starting at 0.4 and increasing to 1.0 over 800 episodes, strategically modifies the focus from predominantly learning from experiences with high TD errors to a more balanced approach over time.

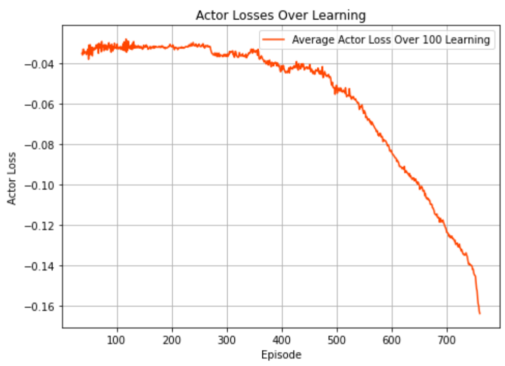
Learning Frequency: The agent initiates the learning process every 2 timesteps (from 2 agents in parallel). This is achieved by accumulating experiences in the environment and storing them in a prioritized experience replay buffer. Once there are enough samples in the buffer (at least equal to the batch size of 1024), the agent begins the learning process. During this phase, it randomly samples a batch of experiences from the buffer to update both the actor and critic networks.

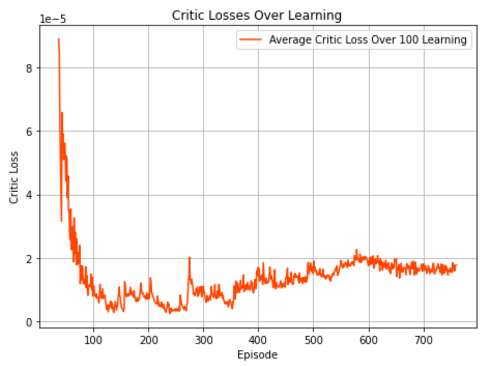
Weight Decay: For both the actor and critic networks, a weight decay parameter is set to 0, indicating that no additional regularization is applied through weight decay. This choice focuses the learning updates purely on the optimization of the loss functions, without the influence of weight size penalties.

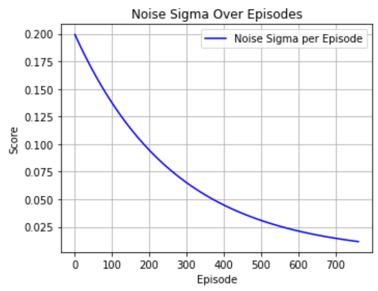
c) Detailed presentation of the best training

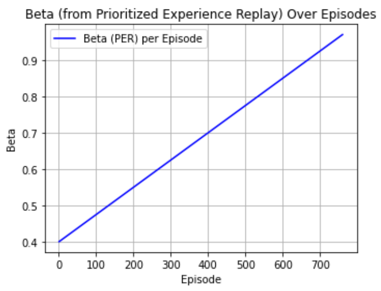
This section reports the metrics. It seems that the training is unstable and has not yet reached convergence. Indeed, there is still room for learning.

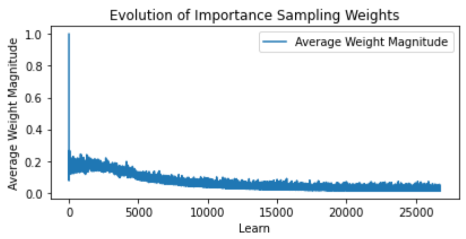


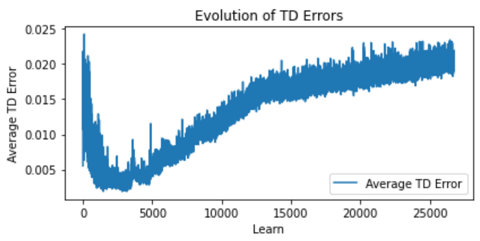


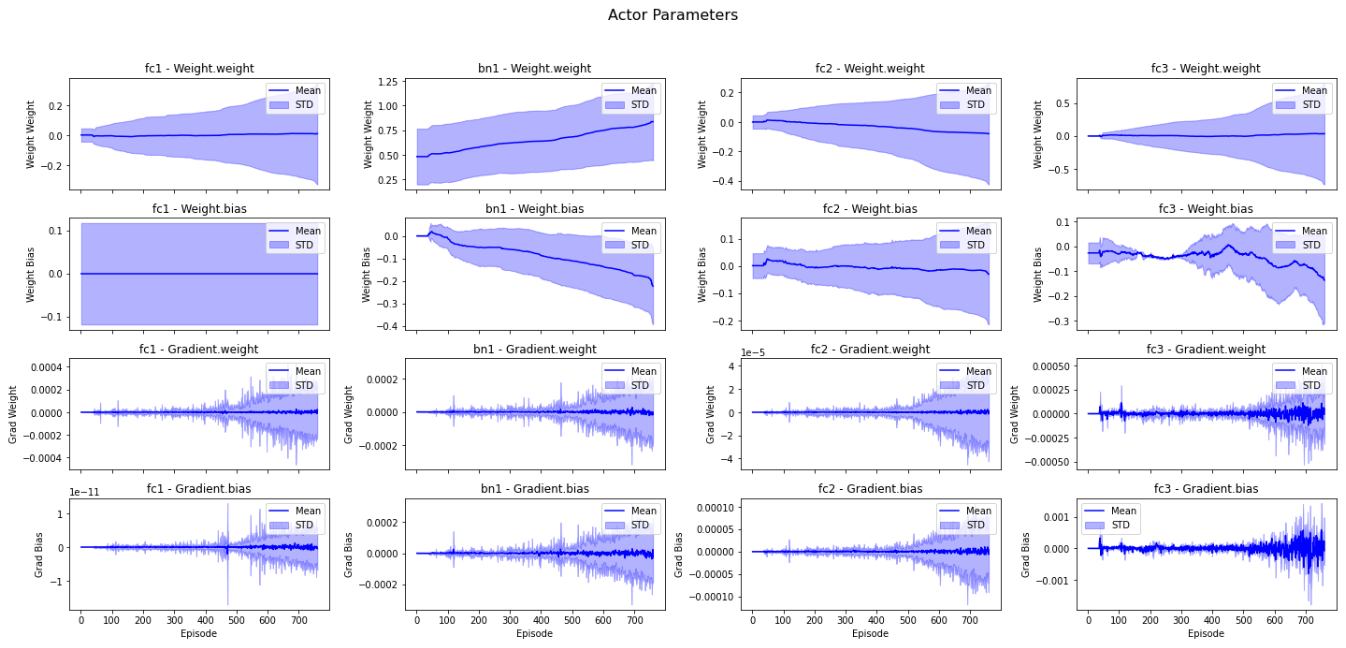


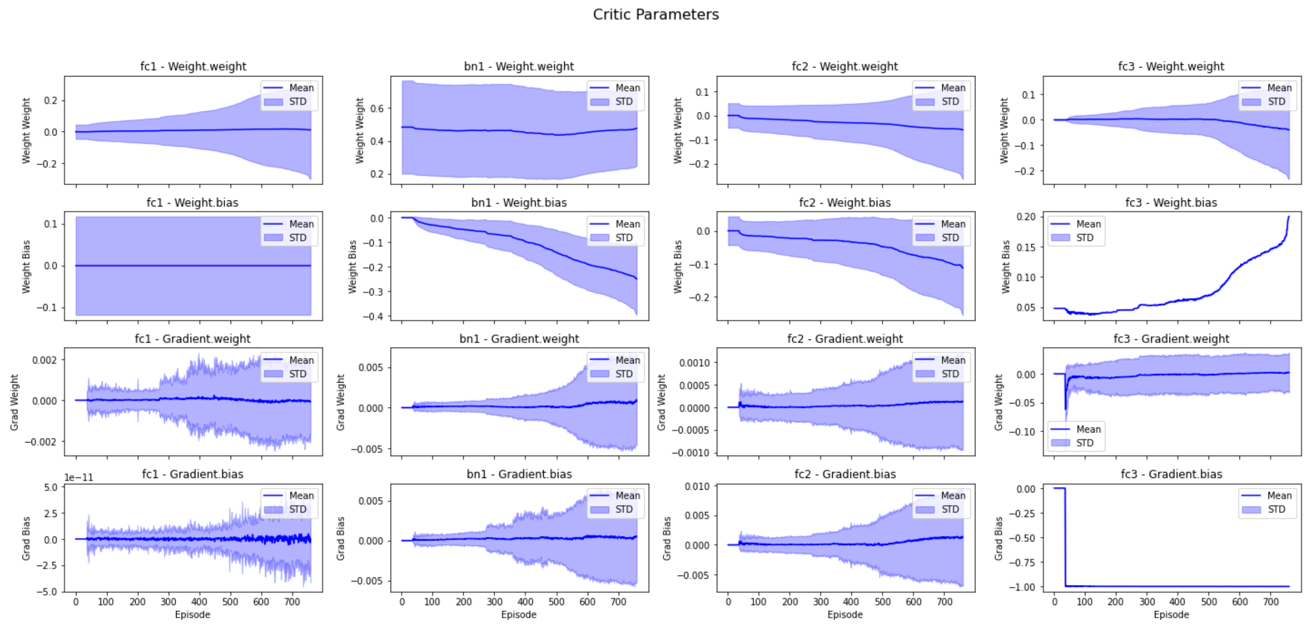












Conclusion

In this report, we have demonstrated the successful adaptation and optimization of the MADDPG algorithm, achieving a significant milestone by reaching an average score of +0.5 over 100 consecutive episodes in just 760 episodes. This accomplishment not only highlights the effectiveness of the chosen architecture and hyperparameters but also marks a pivotal step towards mastering complex multi-agent environments. The journey ahead remains exciting as we continue to push the boundaries of what is possible in the realm of deep reinforcement learning.

Future Work

We maintain the same future work recommendations as in Project 2 for optimizing non-stationary algorithms (refer to Project 2 for details). Here are future work suggestions specifically tailored for stationary environments:

- **Advanced Multi-Agent Coordination Techniques**: Implementing algorithms designed for enhanced communication and coordination between agents, such as differentiable inter-agent learning protocols or shared policy models.
- **Curriculum Learning for Multi-Agent Scenarios**: Gradually increasing the complexity of tasks or the sophistication of agent interactions to improve adaptability and collaboration skills in progressively challenging scenarios.
- **Adversarial Training Methods**: Introducing competitive elements in training, where agents are trained against progressively stronger adversaries to foster robustness and adaptability.
- **Exploration of Implicit Coordination Mechanisms**: Investigating how agents can develop their own non-communicative coordination strategies through shared experiences and joint policy optimization.
- **Cross-Agent Learning Transfer**: Evaluating techniques for transferring learning from one agent to another to accelerate the training process without each agent having to learn from scratch.
- **Dynamic Environment Adaptation**: Enabling agents to adapt to dynamic changes within the environment, such as varying the physics or introducing unpredicted elements, to test the flexibility of the learned policies.
- **Integration of Heterogeneous Agents**: Assessing the performance when agents of different architectures or capabilities are required to collaborate or compete, to understand the dynamics of heterogeneous multi-agent learning.

References

Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments

DDPG Deep Deterministic Policy Gradient - Continuous Action-space

Q-Prop: Sample-Efficient Policy Gradient

GAE Generalized Advantage Estimation

Proximal Policy Optimization Algorithms

NAF

OpenAI. (2024, February 14). Deep Deterministic Policy Gradient. Spinning Up in Deep RL. https://spinningup.openai.com/en/latest/algorithms/ddpg.html

Stable-baselines3. (2024, February 14). Reinforcement Learning Tips and Tricks. Retrieved from https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html

Annexe

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  for t = 1, T do
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
      $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$
    Update the target networks:
      $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
      $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
  end for
end for
```