2. Assignment - Mechanism Design

Zoltán Barta, PhD student, Department of Artificial Intelligence

10 min read
March 12, 2025
Collective Intelligence

Task Description

This assignment focuses on implementing a **Multi-Agent Proximal Policy Optimization (MAPPO)** algorithm using **TorchRL** within a dynamic, **graph-based MARL environment**. Based on the famous board game "Scotland Yard", where agents operate in **asymmetric roles** - one as a **target (Mr. X)** and others as **cooperative pursuers (Policemen)**.

Project GitHub Link

The current project utilizes:

- A **custom graph environment** implemented in the PettingZoo AEC format.
- A centralized value function and shared policy (IDQN) for Policemen.
- A dynamic adversary (Mr. X), controlled by DQN or PPO.
- Meta-learning capabilities that adjust difficulty during training.
- Experimental **Graph Neural Networks (GNNs)** to process local agent neighborhoods and guide movement strategies.
- **Stable-Baselines3** for initial training, with TensorBoard + WandB logging and trajectory visualizations.

The new assignment goal is to replace the Policemen's current IDQN policy with a **MAPPO-based TorchRL implementation**, supporting **Centralized Training with Decentralized Execution (CTDE)**. Policemen must learn to coordinate efficiently on the graph using only local views, while Mr. X attempts to evade detection for as long as possible. An Introduction to Centralized Training for Decentralized Execution in Cooperative Multi-Agent Reinforcement Learning

While the final goal is to use **TorchRL**'s native environment structure (`EnvBase`, `TensorDict`, etc.), you may initially use **PettingZoo environments** with the official `PettingZooWrapper` provided by TorchRL, if helpful for bootstrapping.

> **Example**: Consider a team of police drones tracking a stealthy target in an urban environment. Each drone has local neighborhood information but no global view. By learning together during training but acting independently at test time, they must corner and capture the intruder with optimal coverage.

Environment Phases

1. **Spawn Phase**: All agents are randomly placed on graph nodes.
2. **Pursuit Phase**: Policemen must collaborate to locate and catch Mr. X using local views.
3. **Evasion Phase** (Mr. X): Uses stealthy navigation to maximize survival time.

Assignment Directions

You may choose between the following development directions for this assignment:

- **Option 1: Incremental Migration**
  Maintain the current implementation based on **Stable-Baselines3**, **PettingZoo**, and **Supersuit**, and gradually migrate to TorchRL. This involves:
  - Converting the current environment to TorchRL's `EnvBase` format.
  - Replacing IDQN with MAPPO using **TorchRL's PPO implementation**.
  - Retaining logging (TensorBoard + WandB), curriculum integration, and Docker support.
  - Adding structured **unit tests**, visualization tools, and configuration logic.
- **Option 2: Reimplementation Using Native TorchRL**
  Build a new version of the environment directly with **TorchRL-native components**, including:
  - MAPPO-based CTDE pipeline.
  - GNN-encoded local observations.
  - Fully decentralized execution logic.
  - Structured logging, evaluation, and visualization tools.

Elements to Preserve:

- **MAPPO-style PPO Algorithm**: Centralized critic, shared policy for Policemen, trained with PPO-Clip. PPO-Clip
- Map topology represented as a graph (nodes = positions, edges = moves).
- **Adversarial Setup**: Separate policy for Mr. X (e.g., DQN or PPO).

Elements to Improve or Redesign:

- **GNN-Based State Encoding**. Graph Spaces
- Encourage **coverage, triangulation, and proximity coordination**. Penalize isolated movement or redundant overlaps.
- **Curriculum Learning**: Gradually increase graph size, node degrees, and evasion intelligence. Curriculum Learning
- **Evaluation Metrics**:
  - **Capture rate**
  - **Policemen clustering entropy**
  - **Average distance to Mr. X**
  - **Coverage heatmaps**

A Possible Structured Plan for Reimplementation Using Native TorchRL

0. Possible Directory Structure

```
marl-task/
├── configs/                         # Hydra or YAML configs for experiment control
│   ├── base.yaml
│   ├── env/
│   │   ├── task.yaml
│   ├── algo/
│   │   ├── ppo.yaml
│   ├── agent/
│   │   ├── default.yaml
│   └── experiment/
│       ├── exp.yaml
│
├── docker/                          # Dockerfile and entrypoints
│   ├── Dockerfile
│   └── entrypoint.sh
│
├── logs/                            # TensorBoard / WandB logs (auto-created)
│
├── outputs/                         # Visualizations (e.g., GIFs, videos)
│   ├── gifs/
│   └── metrics/
│
├── models/                          # Trained model checkpoints
│   ├── ppo/
│   │   ├── seed_1.pt
│   │   ├── seed_2.pt
│
├── src/                             # Source code
│   ├── envs/
│   │   ├── env.py
│   │   └── metrics.py               # Custom metrics
│   │
│   ├── agents/
│   │   ├── ppo_agent.py             # PPO policy/training logic
│   │   └── utils.py
│   │
│   ├── rollout/
│   │   ├── evaluator.py             # Evaluation logic
│   │   └── visualizer.py
│   │
│   └── main.py                      # Entry point: loads config and runs training
│
├── test/                            # Unit tests
│   ├── test_env.py
│   ├── test_metrics.py
│
├── .gitignore
├── requirements.txt
├── README.md
└── LICENSE
```

1. Environment Setup

Define a Custom TorchRL-Compatible Environment

Create a class `Env(EnvBase)` in `src/envs/env.py` with the following methods:

- `reset(self) -> TensorDict`
- `step(self, actions: TensorDict) -> TensorDict`

Define:

- `observation_spec`
- `action_spec`
- `reward_spec`
- `done_spec`

Ensure all I/O uses `TensorDict`. Observations should be partial and relative, including distance to the shape center and nearest neighbor. Use `torchrl.envs.transforms` for normalization or preprocessing.

Optional: PettingZoo Wrapper

Use `PettingZooWrapper` from `torchrl.envs.libs.pettingzoo` if adapting from existing environments:

```
from torchrl.envs.libs.pettingzoo import PettingZooWrapper
wrapped_env = PettingZooWrapper(pettingzoo_env)
```

2. Agent and Model Definition

Define Policy and Critic Modules

In `src/agents/ppo_agent.py`, implement:

- A shared `TensorDictModule` policy:

```
policy = TensorDictModule(network, in_keys=[...], out_keys=["action"])
```

- A centralized critic using `ValueOperator`:

```
critic = ValueOperator(critic_network, in_keys=[...])
```

This supports the CTDE paradigm: centralized critic with decentralized policy execution.

3. PPO Training Setup

Collector Configuration

Use `SyncDataCollector` or `MultiSyncDataCollector`:

```
collector = SyncDataCollector(
    create_env_fn=env_fn,
    policy=policy,
    frames_per_batch=2048,
    total_frames=...
)
```

Loss Function

Use `ClipPPOLoss`:

```
loss_module = ClipPPOLoss(
    actor=policy,
    critic=critic,
    clip_epsilon=0.2,
    entropy_coef=0.01
)
```

4. Training Loop

Training in `main.py`

Set up the training loop using `collector`, `replay_buffer`, `loss_module`, and `optimizer`:

```
for batch in collector:
    for _ in range(ppo_epochs):
        loss = loss_module(batch)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

5. Evaluation and Logging

Logging

Use TensorBoard or W&B:

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter(log_dir=...)
writer.add_scalar("reward/mean", mean_reward, step)
```

Evaluation

Run trained policies with local observations only (CTDE) and export GIFs using `pygame`, `matplotlib`, or `imageio`. Store results in `outputs/`.

6. Configuration Management

Hydra Integration

Use Hydra or structured YAML configs in `configs/`:

- `configs/env/task.yaml`
- `configs/algo/ppo.yaml`
- `configs/experiment/sweep.yaml`

Launch with:

```
python src/main.py +experiment=task +algo=ppo
```

7. Testing

Unit Tests

Place tests in `test/`:

```
def test_env_reset():
    env = Env(...)
    td = env.reset()
    assert "observation" in td
```

8. CTDE Framework Details

- The shared policy is trained with access to a centralized value function.
- Execution uses only local observations per agent.
- During inference, policies should operate without access to the global state or other agents' observations.
- Ensure the actor's input keys are restricted to local observations, while the critic receives richer information.

9. Docker for Reproducibility

Add Docker Support

Create a `docker/` folder with the following:

- `Dockerfile`:

```
FROM python:3.12-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["python", "src/main.py"]
```

- `entrypoint.sh` (optional launcher script)

Build and run:

```
docker build -t marl-task .
docker run --rm marl-task
```

PowerPoint Presentation

While presenting your work is not mandatory, **not presenting will limit your maximum grade to 3**. The presentation serves as a concise overview of your project.

Duration

- Aim for **a few well-organized slides** that complement your documentation without repeating it verbatim.

Suggested Structure

1. **Title & Objective**
   - Briefly state the objective.
   - Mention which direction you chose (migration or reimplementation).
2. **System Architecture**
   - Give a high-level overview of your system (environment, agent setup, training loop).
   - Highlight the use of **TorchRL**, and explain your training logic (CTDE, PPO-Clip).
   - Optionally include a block diagram of the pipeline (env → collector → buffer → PPO → evaluation).
3. **Environment & Task Setup**
   - Describe the environment design:
     - Custom vs. PettingZoo-based
     - Agent count and spawn logic
     - Obstacles and dynamics (if any)
   - Explain how agents observe the world and what actions they take.
4. **Key Design Choices**
   - Discuss reward structure, curriculum learning, and logic.
   - Explain any metric(s) you implemented for evaluation
   - Mention logging strategy (WandB, TensorBoard) and how configuration and reproducibility are handled.
5. **Results & Visualizations**
   - Show GIFs or short clips of trained agents forming shapes.
   - Present reward curves, training stability plots, or metric graphs.
   - Provide insights into what worked, what didn't, and what improved after tuning.
6. **Conclusion & Future Work**
   - Summarize key takeaways.

Important Notes

- The **core of your submission is your documentation and code**, which will be the primary basis for grading.
- The presentation is an opportunity to highlight your contributions and insights.

Assignment Submission and General Rules

- All development must be carried out within a **GitHub repository**.

- If working as a **team**:
  - The **collaboration strategy** (e.g., shared or individual branches) can be determined by the team.
  - **Task division must be clearly defined** and documented in the project's `README.md` file (e.g., who worked on the environment, training logic, visualization, etc.).
- If working **individually**, each student must develop their solution on a **separate branch** within the repository.
- Once development is complete, you (or your team) must upload a **single ZIP file** to Canvas containing:
  - The **entire project repository** (excluding large model files or checkpoints to keep the size manageable).
  - The **presentation in PDF format**.
- **Collaboration is highly encouraged**, as this is a larger-scale assignment that benefits from cooperative design and debugging.

👤 Zoltán Barta

📅 March 12, 2025