Multi-Agent Proximal Policy Optimization (MAPPO)

MAPPO is a **model-free**, **stochastic on-policy policy gradient** CTDE (centralized training, decentralized execution) **multi-agent** algorithm that uses a centralized value function to estimate a single value that is used to guide the policy updates of all agents, improving coordination and cooperation between them

Paper: The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games

---

Back to top

Algorithm

For each iteration do:
  For each agent do:

- Collect, in a rollout memory, a set of states s, actions a, rewards r, dones d, log probabilities $logp$ and values V on policy using π_θ and V_ϕ
- Estimate returns R and advantages A using Generalized Advantage Estimation (GAE(λ)) from the collected data $[r, d, V]$
- Compute the entropy loss $L_{entropy}$
- Compute the clipped surrogate objective (policy loss) with $ratio$ as the probability ratio between the action under the current policy and the action under the previous policy: $L^{clip}_{\pi_\theta} = \mathbb{E}[\min(A \; ratio, A \; \text{clip}(ratio, 1 - c, 1 + c))]$
- Compute the value loss L_{V_ϕ} as the mean squared error (MSE) between the predicted values $V_{_{predicted}}$ and the estimated returns R
- Optimize the total loss $L = L^{clip}_{\pi_\theta} - c_1 \, L_{V_\phi} + c_2 \, L_{entropy}$

Algorithm implementation

Main notation/symbols:
- policy function approximator (π_θ), value function approximator (V_ϕ)
- states (s), actions (a), rewards (r), next states (s'), dones (d)
- shared states ($s_{_{shared}}$), shared next states ($s'_{_{shared}}$)
- values (V), advantages (A), returns (R)
- log probabilities ($logp$)
- loss (L)

Learning algorithm

`compute_gae(...)`

def $f_{GAE}(r, d, V, V'_{_{last}}) \;\rightarrow\; R, A:$

$adv \leftarrow 0$

$A \leftarrow \text{zeros}(r)$

advantages computation

FOR each reverse iteration i up to the number of rows in r **DO**

IF i is not the last row of r **THEN**

$V'_i = V_{i+1}$

ELSE

$V'_i \leftarrow V'_{_{last}}$

$adv \leftarrow r_i - V_i +$ `discount_factor` $\neg d_i \; (V'_i -$ `lambda` $adv)$

$A_i \leftarrow adv$

returns computation

$R \leftarrow A + V$

normalize advantages

$$A \leftarrow \frac{A - \bar{A}}{A_\sigma + 10^{-8}}$$

1.1.0

`_update(...)`

FOR each agent **DO**

compute returns and advantages

$V'_{_{last}} \leftarrow V_\phi(s'_{_{shared}})$

$R, A \leftarrow f_{GAE}(r, d, V, V'_{_{last}})$

sample mini-batches from memory

$[[s, a, logp, V, R, A]] \leftarrow$ states, actions, log_prob, values, returns, advantages

learning epochs

FOR each learning epoch up to `learning_epochs` **DO**

mini-batches loop

FOR each mini-batch $[s, a, logp, V, R, A]$ up to `mini_batches` **DO**

$logp' \leftarrow \pi_\theta(s, a)$

compute approximate KL divergence

$ratio \leftarrow logp' - logp$

$KL_{_{divergence}} \leftarrow \frac{1}{N} \sum_{i=1}^{N} ((e^{ratio} - 1) - ratio)$

early stopping with KL divergence

IF $KL_{_{divergence}} >$ `kl_threshold` **THEN**

BREAK LOOP

compute entropy loss

IF entropy computation is enabled **THEN**

$L_{entropy} \leftarrow \, -$ `entropy_loss_scale` $\frac{1}{N} \sum_{i=1}^{N} \pi_{\theta_{entropy}}$

ELSE

$L_{entropy} \leftarrow 0$

compute policy loss

$ratio \leftarrow e^{logp' - logp}$

$L_{surrogate} \leftarrow A \; ratio$

$L_{clipped \; surrogate} \leftarrow A \; \text{clip}(ratio, 1-c, 1+c)$ with c as ratio_clip

$L^{clip}_{\pi_\theta} \leftarrow -\frac{1}{N} \sum_{i=1}^{N} \min(L_{surrogate}, L_{clipped \; surrogate})$

compute value loss

$V_{predicted} \leftarrow V_\phi(s_{shared})$

IF clip_predicted_values is enabled **THEN**

$V_{predicted} \leftarrow V + \text{clip}(V_{predicted} - V, -c, c)$ with c as value_clip

$L_{V_\phi} \leftarrow$ value_loss_scale $\frac{1}{N} \sum_{i=1}^{N} (R - V_{predicted})^2$

optimization step

reset $\text{optimizer}_{\theta,\phi}$

$\nabla_{\theta, \, \phi} (L^{clip}_{\pi_\theta} + L_{entropy} + L_{V_\phi})$

$\text{clip}(\|\nabla_{\theta, \, \phi}\|)$ with grad_norm_clip

step $\text{optimizer}_{\theta,\phi}$

update learning rate

IF there is a learning_rate_scheduler **THEN**

step $\text{scheduler}_{\theta,\phi}(\text{optimizer}_{\theta,\phi})$

Usage

Standard implementation

```python
# import the agent and its default configuration
from skrl.multi_agents.torch.mappo import MAPPO, MAPPO_DEFAULT_CONFIG

# instantiate the agent's models
models = {}
for agent_name in env.possible_agents:
    models[agent_name] = {}
    models[agent_name]["policy"] = ...
    models[agent_name]["value"] = ...  # only required during training

# adjust some configuration if necessary
cfg_agent = MAPPO_DEFAULT_CONFIG.copy()
cfg_agent["<KEY>"] = ...

# instantiate the agent
# (assuming a defined environment <env> and memories <memories>)
agent = MAPPO(possible_agents=env.possible_agents,
              models=models,
              memory=memories,  # only required during training
              cfg=cfg_agent,
              observation_spaces=env.observation_spaces,
              action_spaces=env.action_spaces,
              device=env.device,
              shared_observation_spaces=env.shared_observation_spaces)
```

Configuration and hyperparameters

Note

The specification of a single value is automatically extended to all involved agents, unless the configuration of each individual agent is specified using a dictionary. For example:

```python
# specify a configuration value for each agent (agent names depend on environment)
cfg["discount_factor"] = {"agent_0": 0.99, "agent_1": 0.995, "agent_2": 0.985}
```

```python
MAPPO_DEFAULT_CONFIG = {
    "rollouts": 16,                 # number of rollouts before updating
    "learning_epochs": 8,           # number of learning epochs during each update
    "mini_batches": 2,              # number of mini batches during each learning epoch

    "discount_factor": 0.99,        # discount factor (gamma)
    "lambda": 0.95,                 # TD(lambda) coefficient (lam) for computing returns and

    "learning_rate": 1e-3,                  # learning rate
    "learning_rate_scheduler": None,        # learning rate scheduler class (see torch.optim
    "learning_rate_scheduler_kwargs": {},   # learning rate scheduler's kwargs (e.g. {"step_

    "state_preprocessor": None,             # state preprocessor class (see skrl.resources.p
    "state_preprocessor_kwargs": {},        # state preprocessor's kwargs (e.g. {"size": env
    "shared_state_preprocessor": None,      # shared state preprocessor class (see skrl.reso
    "shared_state_preprocessor_kwargs": {}, # shared state preprocessor's kwargs (e.g. {"size
    "value_preprocessor": None,             # value preprocessor class (see skrl.resources.p
    "value_preprocessor_kwargs": {},        # value preprocessor's kwargs (e.g. {"size": 1})

    "random_timesteps": 0,          # random exploration steps
    "learning_starts": 0,           # learning starts after this many steps

    "grad_norm_clip": 0.5,              # clipping coefficient for the norm of the gradients
    "ratio_clip": 0.2,                  # clipping coefficient for computing the clipped sur
    "value_clip": 0.2,                  # clipping coefficient for computing the value loss
    "clip_predicted_values": False,     # clip predicted values during value loss computatio

    "entropy_loss_scale": 0.0,      # entropy loss scaling factor
    "value_loss_scale": 1.0,        # value loss scaling factor

    "kl_threshold": 0,              # KL divergence threshold for early stopping

    "rewards_shaper": None,         # rewards shaping function: Callable(reward, timestep, t
    "time_limit_bootstrap": False,  # bootstrap at timeout termination (episode truncation)

    "experiment": {
        "directory": "",            # experiment's parent directory
        "experiment_name": "",      # experiment name
        "write_interval": 250,      # TensorBoard writing interval (timesteps)

        "checkpoint_interval": 1000,        # interval for checkpoints (timesteps)
        "store_separately": False,          # whether to store checkpoints separately

        "wandb": False,             # whether to use Weights & Biases
        "wandb_kwargs": {}          # wandb kwargs (see https://docs.wandb.ai/ref/python/ini
    }
}
```

Spaces

The implementation supports the following Gym spaces / Gymnasium spaces

Gym/Gymnasium spaces	Observation	Action
Discrete	□	■
MultiDiscrete	□	■
Box	■	■
Dict	■	□

Models

The implementation uses 1 stochastic (discrete or continuous) and 1 deterministic function approximator. These function approximators (models) must be collected in a dictionary and passed to the constructor of the class under the argument `models`

Notation	Concept	Key	Input shape	Output shape	Type
$\pi_\theta(s)$	Policy	`"policy"`	observation	action	Categorical / Multi-Categorical / Gaussian / MultivariateGaussian
$V_\phi(s)$	Value	`"value"`	observation	1	Deterministic

Features

Support for advanced features is described in the next table

Feature	Support and remarks		
Shared model	for Policy and Value	■	□
RNN support	-	□	□

API (PyTorch)

skrl.multi_agents.torch.mappo.**MAPPO_DEFAULT_CONFIG**

alias of {'clip_predicted_values': False, 'discount_factor': 0.99, 'entropy_loss_scale': 0.0, 'experiment': {'checkpoint_interval': 1000, 'directory': '', 'experiment_name': '', 'store_separately': False, 'wandb': False, 'wandb_kwargs': {}, 'write_interval': 250}, 'grad_norm_clip': 0.5, 'kl_threshold': 0, 'lambda': 0.95, 'learning_epochs': 8, 'learning_rate': 0.001, 'learning_rate_scheduler': None, 'learning_rate_scheduler_kwargs': {}, 'learning_starts': 0, 'mini_batches': 2, 'random_timesteps': 0, 'ratio_clip': 0.2, 'rewards_shaper': None, 'rollouts': 16, 'shared_state_preprocessor': None, 'shared_state_preprocessor_kwargs': {}, 'state_preprocessor': None, 'state_preprocessor_kwargs': {}, 'time_limit_bootstrap': False, 'value_clip': 0.2, 'value_loss_scale': 1.0, 'value_preprocessor': None, 'value_preprocessor_kwargs': {}}

```
class skrl.multi_agents.torch.mappo.MAPPO(possible_agents: Sequence[str], models: Mapping[str, Model], memories: Mapping[str, Memory] | None = None, observation_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None, action_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None, device: str | torch.device | None = None, cfg: dict | None = None, shared_observation_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None)
```

Bases: `MultiAgent`

```
__init__(possible_agents: Sequence[str], models: Mapping[str, Model], memories:
    Mapping[str, Memory] | None = None, observation_spaces: Mapping[str, int] |
    Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None,
    action_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str,
    gymnasium.Space] | None = None, device: str | torch.device | None = None, cfg:
    dict | None = None, shared_observation_spaces: Mapping[str, int] |
    Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None) → None
```

Multi-Agent Proximal Policy Optimization (MAPPO)

https://arxiv.org/abs/2103.01955

PARAMETERS:

- **possible_agents** (*list of str*) – Name of all possible agents the environment could generate
- **models** (*nested dictionary of skrl.models.torch.Model*) – Models used by the agents. External keys are environment agents' names. Internal keys are the models required by the algorithm
- **memories** (*dictionary of skrl.memory.torch.Memory, optional*) – Memories to storage the transitions.
- **observation_spaces** (*dictionary of int, sequence of int, gym.Space or gymnasium.Space, optional*) – Observation/state spaces or shapes (default: `None`)
- **action_spaces** (*dictionary of int, sequence of int, gym.Space or gymnasium.Space, optional*) – Action spaces or shapes (default: `None`)
- **device** (*str or torch.device, optional*) – Device on which a tensor/array is or will be allocated (default: `None`). If None, the device will be either `"cuda"` if available or `"cpu"`
- **cfg** (*dict*) – Configuration dictionary
- **shared_observation_spaces** (*dictionary of int, sequence of int, gym.Space or gymnasium.Space, optional*) – Shared observation/state space or shape (default: `None`)

```
_update(timestep: int, timesteps: int) → None
```

Algorithm's main update step

PARAMETERS:

- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

`act(states: Mapping[str, torch.Tensor], timestep: int, timesteps: int) → torch.Tensor`

Process the environment's states to make a decision (actions) using the main policies

PARAMETERS:

- **states** (*dictionary of torch.Tensor*) – Environment's states
- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

RETURNS:

Actions

RETURN TYPE:

torch.Tensor

`init(trainer_cfg: Mapping[str, Any] | None = None) → None`

Initialize the agent

`post_interaction(timestep: int, timesteps: int) → None`

Callback called after the interaction with the environment

PARAMETERS:

- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

`pre_interaction(timestep: int, timesteps: int) → None`

Callback called before the interaction with the environment

PARAMETERS:

- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

record_transition(states: Mapping[str, torch.Tensor], actions: Mapping[str, torch.Tensor], rewards: Mapping[str, torch.Tensor], next_states: Mapping[str, torch.Tensor], terminated: Mapping[str, torch.Tensor], truncated: Mapping[str, torch.Tensor], infos: Mapping[str, Any], timestep: int, timesteps: int) → None

Record an environment transition in memory

PARAMETERS:

- **states** (*dictionary of torch.Tensor*) – Observations/states of the environment used to make the decision
- **actions** (*dictionary of torch.Tensor*) – Actions taken by the agent
- **rewards** (*dictionary of torch.Tensor*) – Instant rewards achieved by the current actions
- **next_states** (*dictionary of torch.Tensor*) – Next observations/states of the environment
- **terminated** (*dictionary of torch.Tensor*) – Signals to indicate that episodes have terminated
- **truncated** (*dictionary of torch.Tensor*) – Signals to indicate that episodes have been truncated
- **infos** (*dictionary of any supported type*) – Additional information about the environment
- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

API (JAX)

skrl.multi_agents.jax.mappo.**MAPPO_DEFAULT_CONFIG**

alias of {'clip_predicted_values': False, 'discount_factor': 0.99, 'entropy_loss_scale': 0.0, 'experiment': {'checkpoint_interval': 1000, 'directory': '', 'experiment_name': '', 'store_separately': False, 'wandb': False, 'wandb_kwargs': {}, 'write_interval': 250}, 'grad_norm_clip': 0.5, 'kl_threshold': 0, 'lambda': 0.95, 'learning_epochs': 8, 'learning_rate': 0.001, 'learning_rate_scheduler': None, 'learning_rate_scheduler_kwargs': {}, 'learning_starts': 0, 'mini_batches': 2, 'random_timesteps': 0, 'ratio_clip': 0.2, 'rewards_shaper': None, 'rollouts': 16, 'shared_state_preprocessor': None, 'shared_state_preprocessor_kwargs': {}, 'state_preprocessor': None, 'state_preprocessor_kwargs': {}, 'time_limit_bootstrap': False, 'value_clip': 0.2, 'value_loss_scale': 1.0, 'value_preprocessor': None, 'value_preprocessor_kwargs': {}}

class skrl.multi_agents.jax.mappo.**MAPPO**(possible_agents: Sequence[str], models: Mapping[str, Model], memories: Mapping[str, Memory] | None = None, observation_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None, action_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None, device: str | jax.Device | None = None, cfg: dict | None = None, shared_observation_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None)

Bases: `MultiAgent`

```
__init__(possible_agents: Sequence[str], models: Mapping[str, Model], memories: Mapping[str, Memory] | None = None, observation_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None, action_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None, device: str | jax.Device | None = None, cfg: dict | None = None, shared_observation_spaces: Mapping[str, int] | Mapping[str, gym.Space] | Mapping[str, gymnasium.Space] | None = None) → None
```

Multi-Agent Proximal Policy Optimization (MAPPO)

https://arxiv.org/abs/2103.01955

PARAMETERS:

- **possible_agents** (*list of str*) – Name of all possible agents the environment could generate
- **models** (*nested dictionary of skrl.models.jax.Model*) – Models used by the agents. External keys are environment agents' names. Internal keys are the models required by the algorithm
- **memories** (*dictionary of skrl.memory.jax.Memory, optional*) – Memories to storage the transitions.
- **observation_spaces** (*dictionary of int, sequence of int, gym.Space or gymnasium.Space, optional*) – Observation/state spaces or shapes (default: `None`)
- **action_spaces** (*dictionary of int, sequence of int, gym.Space or gymnasium.Space, optional*) – Action spaces or shapes (default: `None`)
- **device** (*str or jax.Device, optional*) – Device on which a tensor/array is or will be allocated (default: `None`). If None, the device will be either `"cuda"` if available or `"cpu"`
- **cfg** (*dict*) – Configuration dictionary
- **shared_observation_spaces** (*dictionary of int, sequence of int, gym.Space or gymnasium.Space, optional*) – Shared observation/state space or shape (default: `None`)

```
_update(timestep: int, timesteps: int) → None
```

Algorithm's main update step

PARAMETERS:

- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

```
act(states: Mapping[str, ndarray | jax.Array], timestep: int, timesteps: int) → ndarray | jax.Array
```

Process the environment's states to make a decision (actions) using the main policies

PARAMETERS:

- **states** (*dictionary of np.ndarray or jax.Array*) – Environment's states
- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

RETURNS:

Actions

RETURN TYPE:

np.ndarray or jax.Array

```
init(trainer_cfg: Mapping[str, Any] | None = None) → None
```

Initialize the agent

```
post_interaction(timestep: int, timesteps: int) → None
```

Callback called after the interaction with the environment

PARAMETERS:

- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

```
pre_interaction(timestep: int, timesteps: int) → None
```

Callback called before the interaction with the environment

PARAMETERS:

- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps

```
record_transition(states: Mapping[str, ndarray | jax.Array], actions: Mapping[str, ndarray | jax.Array], rewards: Mapping[str, ndarray | jax.Array], next_states: Mapping[str, ndarray | jax.Array], terminated: Mapping[str, ndarray | jax.Array], truncated: Mapping[str, ndarray | jax.Array], infos: Mapping[str, Any], timestep: int, timesteps: int) → None
```

Record an environment transition in memory

PARAMETERS:

- **states** (*dictionary of np.ndarray or jax.Array*) – Observations/states of the environment used to make the decision
- **actions** (*dictionary of np.ndarray or jax.Array*) – Actions taken by the agent
- **rewards** (*dictionary of np.ndarray or jax.Array*) – Instant rewards achieved by the current actions
- **next_states** (*dictionary of np.ndarray or jax.Array*) – Next observations/states of the environment
- **terminated** (*dictionary of np.ndarray or jax.Array*) – Signals to indicate that episodes have terminated
- **truncated** (*dictionary of np.ndarray or jax.Array*) – Signals to indicate that episodes have been truncated
- **infos** (*dictionary of any type supported by the environment*) – Additional information about the environment
- **timestep** (*int*) – Current timestep
- **timesteps** (*int*) – Number of timesteps