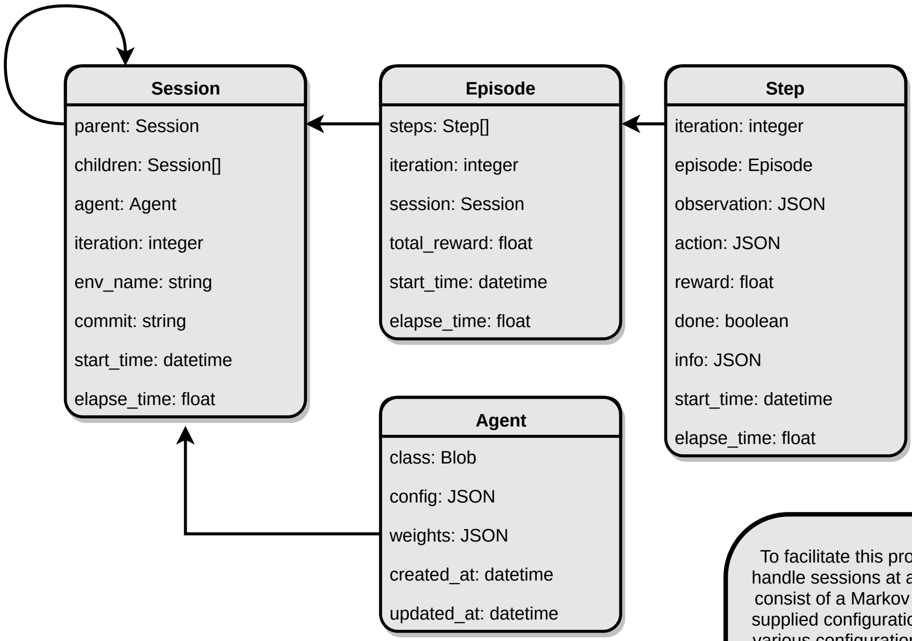


Session

parent: Session
children: Session[]
agent: Agent
iteration: integer
env_name: string
commit: string
start_time: datetime
elapse_time: float

Episode

steps: Step[]
iteration: integer
session: Session
total_reward: float
start_time: datetime
elapse_time: float

Step

iteration: integer
episode: Episode
observation: JSON
action: JSON
reward: float
done: boolean
info: JSON
start_time: datetime
elapse_time: float

Agent

class: Blob
config: JSON
weights: JSON
created_at: datetime
updated_at: datetime

Start with an initial session. This will be a set of data (**Session** table)

For each session, run a number of episodes. Each episode will be a set of data (**Episode** table)

For each episode, run steps until done. Each step will be a set of data (**Step** table)

Running a Session

- **--parent**
  - **by default, the parent session is selected based on the --session-rule argument**
  - **if an integer is supplied, then the parent_id is set to this**
- **--rule**
  - **by default, 'max-reward'**
  - **keyword can be supplied to change this**
- **--agent**
  - **by default, 'use-last'**
  - **if an integer is supplied, then the agent_id is set to this**
- **--env**
  - **by default, 'CartPole-v1'**
  - **make the Gym environment**

To facilitate this process, we provide tools to handle sessions at a high level. A session will consist of a Markov Decision Process given a supplied configuration. With an initial session, various configurations must be made, such as supplying an environment and an agent. When continuing a session, every parameter for configuration will have default values provided from the parent session. This will make the process of continuous training possible (e.g., keep training indefinitely).

On top of training configurations, we can also set session configurations. These configurations will dictate how sessions branch, what criteria is used to determine healthy/dead nodes, directions for changing configurations, etc. We will supply a number of default parameters to suit general RL tasks.

The goal is to provide an organized process for training RL agents. This process should be agnostic to the actual agents/environments being used, and should be robust enough to handle considerable changes to our process (e.g., training a resource allocation agent). The data will be stored in a database, and everything suggested above (including serializing weights/classes) should be possible using most databases (e.g., MySQL, SQLite, Postgres).

Once this architecture is setup, we can then create an interface (via an online app) to monitor and even start/modify training.

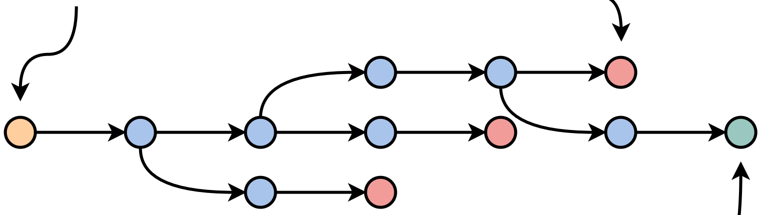
A Session is a collection of episodes (trained in sequence) for a given agent configuration. The goal of a session is to allow training to be reproducible. We store enough data to generate the same starting agent and environment.

Once a training session is complete, the model is saved and the session is over. The session, at this point, is should not be edited beyond this point.

When we want to begin training again, we specify the previous session that we want to branch from, adjust configurations, and run through episodes in a newly created session.

Through this process, we can keep track of each training session including changes we've made to code and configuration as well as model checkpoints.

We can than navigate our session tree to look at how training progressed through various stages.



The root session is the first session of any instance. The root session has no parent session and no episodes. Instead, the root session keeps track of initialization variables, such as weights and configuration. We can then begin training starting with a new branch from the root.

Branches may die if predefined criteria are not hit (e.g., an increase in reward since the last session). To continue training, a new branch is created at the last healthy session node, and we continue.

The goal is to reach a session with satisfactory results. Through this process, we may find multiple sessions which meet our criteria.

Start

A Session is initialized. The user either selects a Session to branch from, or provides a configuration to start a new graph.

An episode is then spawned from this configuration. This includes a provided Agent class, Environment class, and Markov Decision Process class. This episode is spawned as a subprocess, which ends after a provided iteration count is reached.

During the episodes iterations, steps are generated and logged through the Markov Decision Process.

Continue

The Session spawns Episodes according to it's configuration. By default, this will be to spawn *cpu_count* parallel episodes (and to maintain this number of episodes running in parallel until end conditions are met).

Spawned Episodes are initialized with the current configuration of the Session (stored in the Sessions table). The Episode then begins a Markov Decision Process, collecting necessary data along the way.

Each Episode runs until a Step's *done* value is *True*. When an episode is complete, the episode compute's it's gradients and posts them to the database.

Finish

A Session periodically checks the status of it's episodes. When it finds a completed episode, it processes it's gradients and updates the master weights.

If end conditions aren't met, the Session will then spawn a new episode with the new configuration.

If end conditions *are* met, then the Session computes any final statistics and runs finishing steps.

Based on the Session's meta-configuration, another Session might start from this node, repeating the process.