AI Project Updates

Group 15

4/7/2022

Control of a Quadrotor With Reinforcement Learning

***Abstract*—In this letter, we present a method to control a quadrotor with a neural network trained using reinforcement learning techniques. With reinforcement learning, a common network can be trained to directly map state to actuator command making any predefined control structure obsolete for training. Moreover, we present a new learning algorithm that differs from the existing ones in certain aspects. Our algorithm is conservative but stable for complicated tasks. We found that it is more applicable to controlling a quadrotor than existing algorithms. We demonstrate the performance of the trained policy both in simulation and with a real quadrotor. Experiments show that our policy network can react to step response relatively accurately. With the same policy, we also demonstrate that we can stabilize the quadrotor in the air even under very harsh initialization (manually throwing it upside-down in the air with an initial velocity of 5 m/s). Computation time of evaluating the policy is only 7 μs per time step, which is two orders of magnitude less than common trajectory optimization algorithms with an approximated model.**

Fig. 1. The quadrotor stabilizes from a hand throw. The motor was enabled after it left the hand.

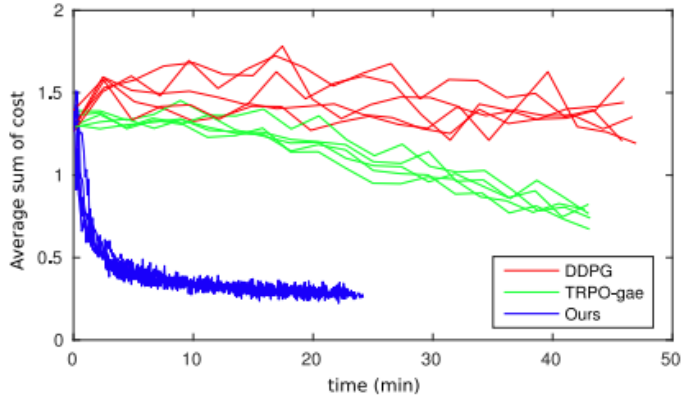


Fig. 4. Learning curves for optimizing the policy for three different algorithms and 5 different runs per each algorithm.

TABLE I
QUADROTOR PHYSICAL PARAMETERS

Parameter	value
mass	0.665 kg
dimension	0.44 m, 0.44 m, 0.12 m
I_{xx}, I_{yy}, I_{zz}	0.007, 0.007, 0.012 kgm^2

Hwangbo, J., Sa, I., Siegwart, R., & Hutter, M. (2017). Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, *2*(4), 2096-2103.

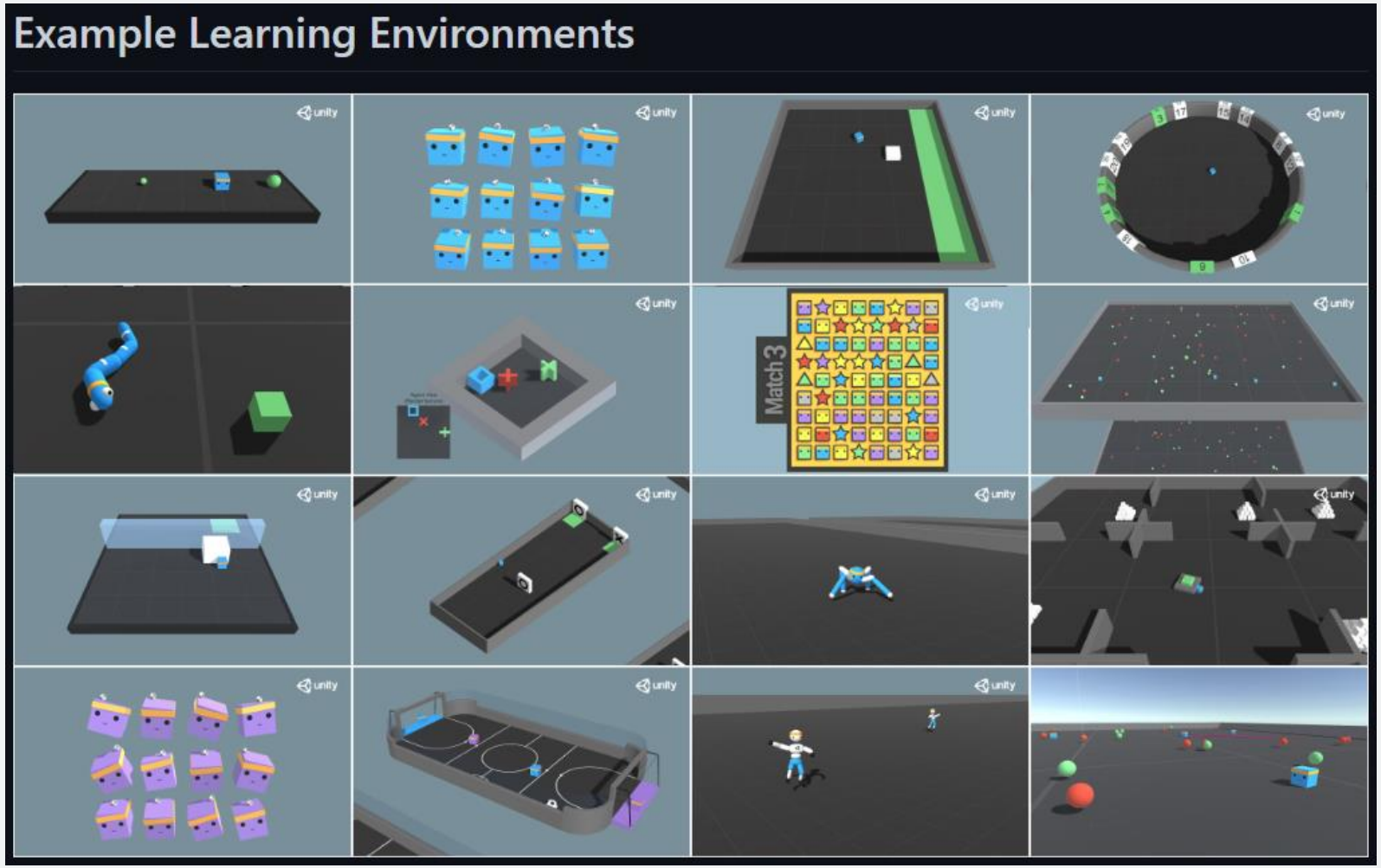
ml-agents Unity asset package for RL

Features

- 18+ example Unity environments
- Support for multiple environment configurations and training scenarios
- Flexible Unity SDK that can be integrated into your game or custom Unity scene
- Support for training single-agent, multi-agent cooperative, and multi-agent competitive scenarios via several Deep Reinforcement Learning algorithms (PPO, SAC, MA-POCA, self-play).
- Support for learning from demonstrations through two Imitation Learning algorithms (BC and GAIL).
- Easily definable Curriculum Learning scenarios for complex tasks
- Train robust agents using environment randomization
- Flexible agent control with On Demand Decision Making
- Train using multiple concurrent Unity environment instances
- Utilizes the Unity Inference Engine to provide native cross-platform support
- Unity environment control from Python
- Wrap Unity learning environments as a gym

https://github.com/Unity-Technologies/ml-agents

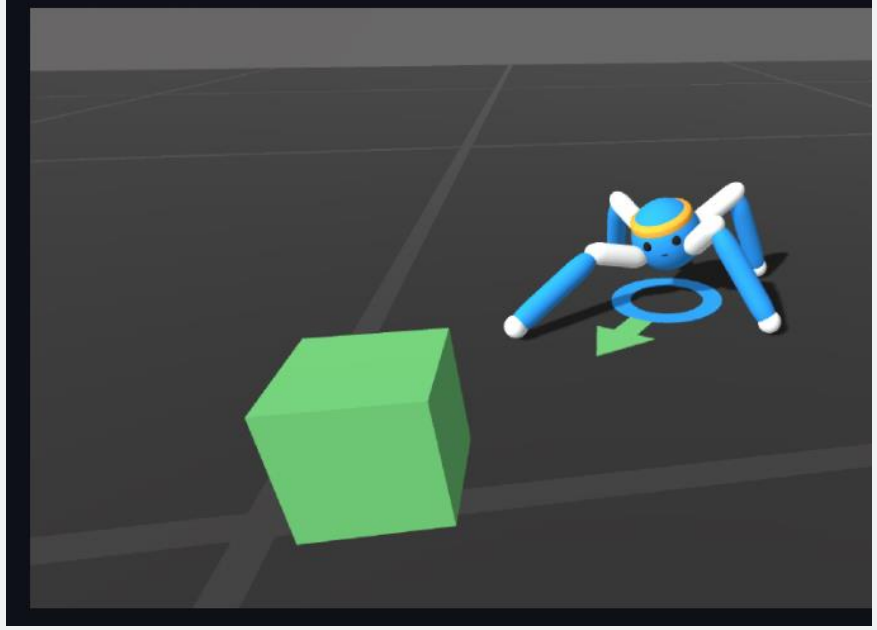
Example Environments



https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Examples.md

Environment Examples

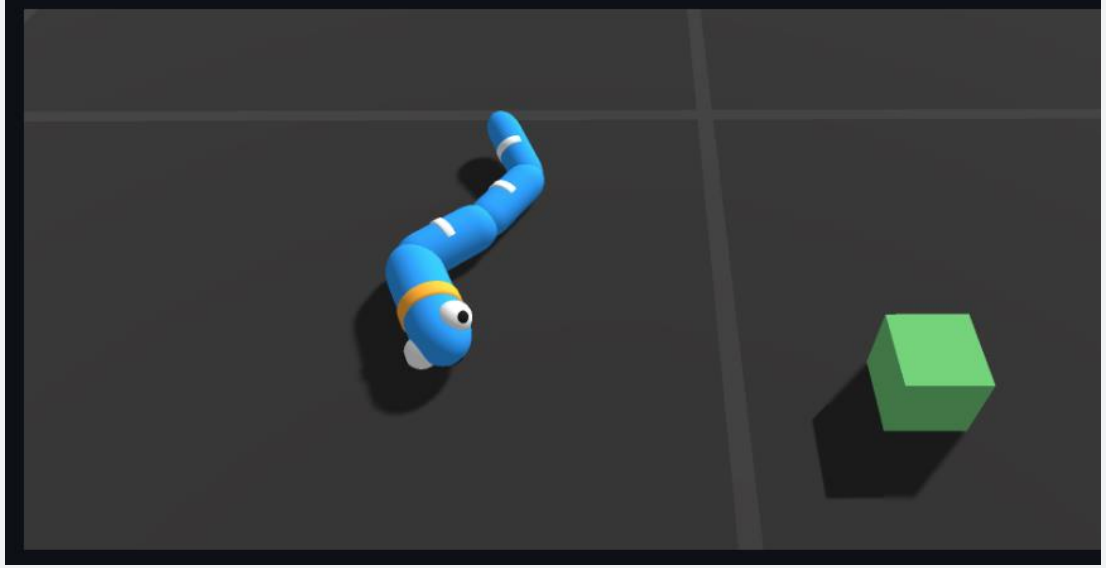
Crawler



- Set-up: A creature with 4 arms and 4 forearms.
- Goal: The agents must move its body toward the goal direction without falling.
- Agents: The environment contains 10 agents with same Behavior Parameters.
- Agent Reward Function (independent): The reward function is now geometric meaning the reward each step is a product of all the rewards instead of a sum, this helps the agent try to maximize all rewards instead of the easiest rewards.
  - Body velocity matches goal velocity. (normalized between (0,1))
  - Head direction alignment with goal direction. (normalized between (0,1))
- Behavior Parameters:
  - Vector Observation space: 172 variables corresponding to position, rotation, velocity, and angular velocities of each limb plus the acceleration and angular acceleration of the body.
  - Actions: 20 continuous actions, corresponding to target rotations for joints.
  - Visual Observations: None
- Float Properties: None
- Benchmark Mean Reward: 3000

Environment Examples

Worm



- Set-up: A worm with a head and 3 body segments.
- Goal: The agents must move its body toward the goal direction.
- Agents: The environment contains 10 agents with same Behavior Parameters.
- Agent Reward Function (independent): The reward function is now geometric meaning the reward each step is a product of all the rewards instead of a sum, this helps the agent try to maximize all rewards instead of the easiest rewards.
  - Body velocity matches goal velocity. (normalized between (0,1))
  - Body direction alignment with goal direction. (normalized between (0,1))
- Behavior Parameters:
  - Vector Observation space: 64 variables corresponding to position, rotation, velocity, and angular velocities of each limb plus the acceleration and angular acceleration of the body.
  - Actions: 9 continuous actions, corresponding to target rotations for joints.
  - Visual Observations: None
- Float Properties: None
- Benchmark Mean Reward: 800

Environment Examples

- Set-up: Physics-based Humanoid agents with 26 degrees of freedom. These DOFs correspond to articulation of the following body-parts: hips, chest, spine, head, thighs, shins, feet, arms, forearms and hands.
- Goal: The agents must move its body toward the goal direction without falling.
- Agents: The environment contains 10 independent agents with same Behavior Parameters.
- Agent Reward Function (independent): The reward function is now geometric meaning the reward each step is a product of all the rewards instead of a sum, this helps the agent try to maximize all rewards instead of the easiest rewards.
  - Body velocity matches goal velocity. (normalized between (0,1))
  - Head direction alignment with goal direction. (normalized between (0,1))
- Behavior Parameters:
  - Vector Observation space: 243 variables corresponding to position, rotation, velocity, and angular velocities of each limb, along with goal direction.
  - Actions: 39 continuous actions, corresponding to target rotations and strength applicable to the joints.
  - Visual Observations: None
- Float Properties: Four
  - gravity: Magnitude of gravity
    - Default: 9.81
    - Recommended Minimum:
    - Recommended Maximum:
  - hip_mass: Mass of the hip component of the walker
    - Default: 8
    - Recommended Minimum: 7
    - Recommended Maximum: 28
  - chest_mass: Mass of the chest component of the walker
    - Default: 8
    - Recommended Minimum: 3
    - Recommended Maximum: 20
  - spine_mass: Mass of the spine component of the walker
    - Default: 8
    - Recommended Minimum: 3
    - Recommended Maximum: 20
- Benchmark Mean Reward : 2500

Unity Control using Python

The basic command for training is:

```
mlagents-learn <trainer-config-file> --env=<env_name> --run-id=<run-identifier>
```

- `<trainer-config-file>` is the file path of the trainer configuration YAML. This contains all the hyperparameter values. We offer a detailed guide on the structure of this file and the meaning of the hyperparameters (and advice on how to set them) in the dedicated Training Configurations section below.
- `<env_name>` **(Optional)** is the name (including path) of your Unity executable containing the agents to be trained. If `<env_name>` is not passed, the training will happen in the Editor. Press the **Play** button in Unity when the message *"Start training by pressing the Play button in the Unity Editor"* is displayed on the screen.
- `<run-identifier>` is a unique name you can use to identify the results of your training runs.

```python
from mlagents_envs.environment import UnityEnvironment
# This is a non-blocking call that only loads the environment.
env = UnityEnvironment(file_name="3DBall", seed=1, side_channels=[])
# Start interacting with the environment.
env.reset()
behavior_names = env.behavior_specs.keys()
...
```

3DBallTrainDefault.yaml

```yaml
3DBall:
  trainer: ppo
  batch_size: 64
  buffer_size: 12000
  learning_rate: 0.0003
  beta: 0.001
  epsilon: 0.2
  lambd: 0.99
  num_epoch: 3
  learning_rate_schedule: linear
  keep_checkpoints: 5
  max_steps: 5000000
  time_horizon: 1000
  summary_freq: 12000

# memory
  use_recurrent: true
  sequence_length: 64
  memory_size: 256

  #network_settings:
  normalize: true
  hidden_units: 128
  num_layers: 2
  vis_encode_type: simple

  reward_signals:
    extrinsic:
      gamma: 0.99
      strength: 1.0
```

https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Python-API.md

Tasks

- (Everyone) Set up Unity on individual PC's & run the ml-agents examples
- Prepare a custom training project package in Unity
- Create a trainable agent model for the project (drone) to be imported
- Create a custom learning environment in Unity for training our agent:

  https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Create-New.md
- Train the agent using default algorithm configurations (PPO, SAC)
- Implement a classical path planning algorithm for comparison with RL methods (use one from the AI course)

Environment Examples