Unity ML-Agents SAC Policy

Harry Li ‡, Ph.D., Chee Vang§
Computer Engineering Department, San Jose State University
San Jose, CA 95192, USA

Email: harry.li@ctione.com†, chee.vang02@sjsu.edu§

***Abstract*—This note describes the reinforcement learning policies employed by Unity ML-Agents, specifically Soft Actor Critic (SAC). [1] provides an overview of the ML-Agent Toolkit and its training methods.**

I. Introduction

Unity ML-Agents toolkit is an open-source projects that allows training intelligent agents in simulated environments. The agents are trained using reinforcement learning and other machine learning algorithms through a Python API. For this paper, a 6 DoF robot arm from [2] was trained using ML-Agents toolkit. Unity ML-Agents toolkit provides several deep reinforcement learning algorithms, including

- Soft Actor Critic (SAC)
- Proximal Policy Optimization (PPO)

Unity ML-Agents toolkit also provides other DRL environment-based training methods to aid in the training behaviors of the agents for specific environments and can be found in [1]. The reinforcement learning algorithm implemented in this project is SAC and will be explained in further details.

II. Training Configuration File

The backend Python implementations for the trainers can be found in [4] from the github repository in [3].

A. Common Trainer Configurations

To define the type of training done on the agents in Unity, a training configuration file is created which defines the policy (PPO, SAC, POCA) and other hyperparameters. Table I consists of common settings between PPO and SAC in a training configuration yaml file defined in [5]. More detailed explanations including the default values and typical range can be found in [5].

TABLE I
Common Trainer Configurations

Setting	Description
trainer_type	The type of trainer: ppo, sac, or poca.
summary_freq	Number of experiences before generating and displaying training statistics.
time_horizon	Amount of steps of experience per-agent to collect before adding to experience buffer.
max_step	Total number of steps (observations and actions) that must be taken in the environment before ending training process.
keep_checkpoints	Maximum number of model checkpoints to keep.
checkpoint_interval	Number of experiences collected between each checkpoints. Checkpoints saved in .onnx file.
init_path	Initialize trainer from previously saved model.
threaded	Ability to update the model while the environment is being stepped.
hyperparameters: learning_rate	Initial learning rate for gradient descent.
hyperparameters: batch_size	Number of experiences in each iteration of gradient descent.
hyperparameters: buffer_size	PPO: Number of experience to collect before updating policy model SAC: Max size of experience buffer.
hyperparameters: learning_rate_schedule	Determines the learning rate changes over time: linear or constant. Suggested constant for SAC.
network_settings: batch_size	Number of units in hidden layers of NN.
network_settings: batch_size	Number of hidden layers in the NN.
network_settings: batch_size	Determines if observations inputs are normalized.
network_settings: batch_size	Encoder type for encoding visual observations.

B. Soft Actor Critic (SAC)

An off-policy reinforcement learning algorithm. [1] and [6] describes SAC as an algorithm that learns from experiences collected at any time during the past. SAC was developed at UC Berkeley and Google for their robotics experiments. The objective function is based on maximum entropy reinforcement learning framework described as

$$J(\pi) = E_{\pi}\left[\sum_{t} r(s_t, a_t) - \alpha log(\pi(a_t|s_t))\right] \quad (1)$$

where expectation is taken over the policy and actual dynamics of the system with the state s_t and action a_t. [6] explains that the first sum maximizes the expected return and the second sum maximizes the expected entropy of itself. The

non-negative temperature parameter (entropy coefficient) α is used to control the trade-off between the two. The following parameters in table II are exclusive to the SAC training policy. More detailed explanations including the default values and typical range can be found in [5].

TABLE II
SAC-SPECIFIC CONFIGURATIONS

Setting	Description
hyperparameters: buffer_init_steps	Number of experiences to collect into the buffer before updating policy model.
hyperparameters: init_entcoef	How much the agent should explore in the beginning of training.
hyperparameters: save_replay_buffer	Determines whether to save or load the experience replay buffer as well as the model when quitting and re-starting training.
hyperparameters: tau	Corresponds to the magnitude of the target Q update in SAC which determines how aggressively to update the target network used for bootstrapping value estimation.
hyperparameters: steps_per_update	Average ratio of the agent steps (actions) taken to updates made of agent's policy. A lower value will improve training speed but increases CPU time to perform updates. For most environments, this value is equal to number of agent's in the scene.
hyperparameters: reward_signal_num_update	Number of steps per mini batch sampled and used for updating the reward signals.

C. Reward Signals

The reward is one of the components that are necessary to train the behaviors of an agent. Generally, the reward signal can be described as either extrinsic or intrinsic. Extrinsic rewards are defined by the environment and are external of the learning algorithm. In this case, the behavior of the agent is moved towards a specific reward outcome. While, intrinsic rewards are defined outside of the environment to encourage the agent to to behave a certain way such as a specific demonstration. Unity ML-Agents toolkit allows four reward signals to help shape the agent's behaviour:

- **extrinsic**: rewards defined in environment
- **gail**: intrinsic rewards defined by Generative Adversarial Imitation Learning (GAIL) which rewards the agent for behaviour similar to a set of demonstration
- **curiosity**: an intrinsic reward to encourage exploration in sparse-reward environments following the paper [7]
- **rnd**: similar to curiosity but follows the paper [8].

Each reward signal has at least two parameters: strength and gamma. For extrinsic rewards, strength defines the factor to multiply the reward given by the environment. Gamma is the discount factor for future rewards coming from the environment. For intrinsic rewards, strength is the magnitude of the curiosity reward generated by the module, and gamma is the discount factor for future rewards. The reward signal used in the Robot Arm demo is extrinsic since the rewards are defined

III. UNITY ML-AGENT TRAINING

The training configurations are saved onto a YAML file where ML-Agent toolkit will use during the training. In [9], the basic cammand to start training ML-Agents is:

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

where

- `<trainer-config-file>` is the full path to the trainer configuration YAML file
- `<env_name>` (optional) is the name of the Unity executable containing the agents to be trained
- `<run-identifier>` is a unique name to identify the results of the training

Training will take several minutes (even hours) depending on the parameters defined in the YAML file. Lower the max_step to reduce the time to training, but this will also reduce the accuracy of the trained model. From [2] the robot arm tends to get stuck in what they called the "cobra pose", where the robot arm has reclined into itself. To prevent robot arm from getting stuck in this position in a future training session, increase the batch_size and num_layers. The batch_size determine the number of nodes in each layer and num_layers is the number of hidden layers in a NN.

Once the command to start training is entered, a list of configurations will be displayed onto the the terminal shown in Fig. 1 and 2. In Fig 1, the trainer-type is ppo which includes configurations specifically for PPO in the red box. Similarly, there are SAC-specific configurations for the training done in Fig. 2 (in red box). In addition, some changes were made to the common configuration such as the max_step.

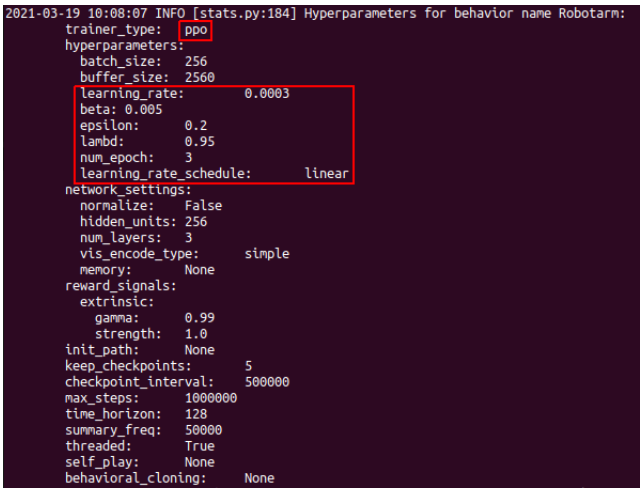

Fig. 1. Configuration output when training with PPO

From [9], SAC requires 5-10 times less samples than PPO for the same task; hence, the amount of steps taken was

reduced to 500,000 compared to 1,000,00 for PPO. Other configurations in Fig. 2, were changed to speed up the training process.

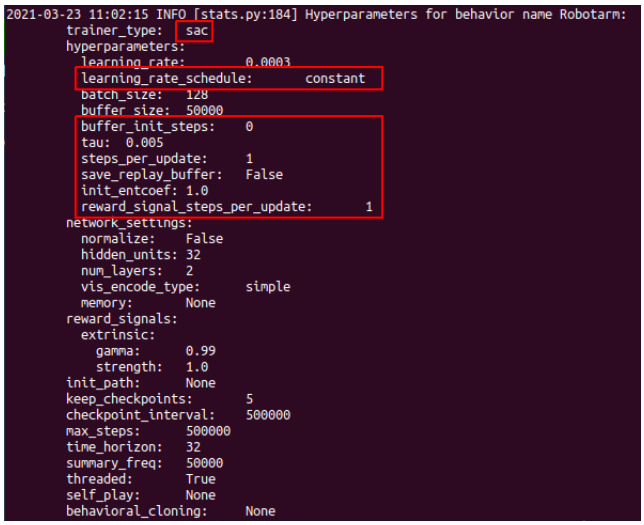

Fig. 2. Configuration output when training with SAC

The results can be monitors graphically through Tensorboard by running the code

```
tensorboard --logdir results --port 6006
```

It will provide a link where tensorboard will include all the training results in that directory. Fig. 3 are the training statistics for the robot arm using PPO (blue) and SAC (pink). As previously stated, PPO consists of 1M steps while SAC has 500k as seen in the figure. Notice how the rate of the cummulative reward for PPO continues to rise while SAC seems to stagnant. This might be due to the changes made to reduce training time since there were less nodes and layers in the NN for SAC.

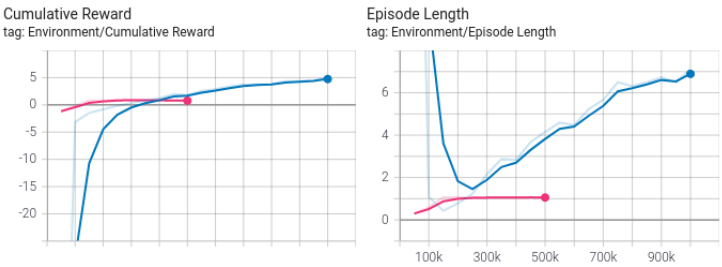

Fig. 3. Training statistics on Tensorboard

REFERENCES

[1] "ML-Agents Toolkit Overview" Unity-Technologies: ML-Agents. https://github.com/Unity-Technologies/ml-agents/blob/main/docs/ML-Agents-Overview.md

[2] Raju K. "How to train your Robot Arm?" Medium. https://medium.com/xrpractices/how-to-train-your-robot-arm-fbf5dcd807e1 (Accessed March 15, 2021).

[3] "." Unity-Technologies: ML-Agents. https://github.com/Unity-Technologies/ml-agents

[4] Unity-Technologies: ML-Agents. Github repository for ML-Agents trainers. https://github.com/Unity-Technologies/ml-agents/tree/main/ml-agents/mlagents/trainers

[5] "Training Configuration File." Unity-Technologies: ML-Agents. https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md

[6] T. Haarnoja, V. Pong, K. Hartikain, A. Zhou, M. Dalal, S. Levine. "Soft Actor Critic - Deep Reinforcement Learning with Real-World Robots." Berkeley Artificial Intelligence Research. 2018. https://bair.berkeley.edu/blog/2018/12/14/sac/. (Accessed March 21, 2021).

[7] D. Pathak, P. Agrawal, A. A. Efros, T. Darrell. "Curiosity-driven Exploration by Self-supervised Prediction." Univeristy of California, Berkeley. ICML 2017. https://blogs.unity3d.com/2018/06/26/solving-sparse-reward-tasks-with-curiosity/. (Accessed March 21, 2021).

[8] Y. Burda, H. Edwards, A. Storkey, O. Klimov. "Exploration by Random Network Distillation." Cornell University. 2018. https://arxiv.org/abs/1810.12894. (Accessed March 21, 2021).

[9] "Training ML-Agents." Unity-Technologies: ML-Agents. https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-ML-Agents.md