Introducation of Unity ML-Agents Toolkit

xuehao zhang

JAN 8, 2021

Abstract

1 Introduction

The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents. We provide implementations (based on PyTorch) of state-of-the-art algorithms to enable game developers and hobbyists to easily train intelligent agents for 2D, 3D and VR/AR games. Researchers can also use the provided simple-to-use Python API to train Agents using reinforcement learning, imitation learning, neuroevolution, or any other methods. These trained agents can be used for multiple purposes, including controlling NPC behavior (in a variety of settings such as multi-agent and adversarial), automated testing of game builds and evaluating different game design decisions pre-release. The ML-Agents Toolkit is mutually beneficial for both game developers and AI researchers as it provides a central platform where advances in AI can be evaluated on Unity's rich environments and then made accessible to the wider research and game developer communities.

2 How to train an agent

With ML-Agents, there are various methods to train an agent. The basic idea is simple, We need to define three instances at each moment of the game (called the environment):

Observation - the agent's perception of the environment.Observations can be in digital and/or visual form. Digital observation will measure the properties of the environment from the perspective of the agent.Depending on the game and the agent, the digital observation data can be either in the form of discrete or continuous.For most interesting environments, agents will require a number of continuous digital observations, while for simple environments with a few unique configurations, discrete observations will suffice. On the other hand, visual observation is the image generated by the camera attached to the agent, which represents the content seen by the agent at this point in time.We often confuse the agent's observations with the state of the environment (or the game).The environment state represents information about the entire scene that contains all the game characters.However, the agent observation contains only the information that the agent knows, which is usually a subset of the environment state.For example, agent observations cannot include stealth enemy information that is not known to the agent.

Actions - Actions that the agent can take. Similar to observation, actions can be either continuous or discrete depending on the complexity of the environment and agent. If the environment is a simple grid

world based only on position, then a discrete action with one of the four values (up, down, left and right) is sufficient.However, if the environment is more complex and the medic can move freely, it is more appropriate to use two consecutive actions (one for direction and one for speed).

Reward - A scalar value that represents an agent's behavior. Note that a reward does not need to be provided at every moment, but only if the agent performs a good or bad action. The reward represents how the goal of the task is communicated to the agent, so we set it up in such a way as to ensure that when the reward is maximized, the agent will produce the behavior we expect most.

Although it's called ML-agents, the main content is Reinforcement Learning. Actually, ML mainly includes three paradigms of supervised learning, unsupervised learning and reinforcement learning, but there is no content of supervised learning and unsupervised learning.

In the concept of reinforcement learning, the learned behavior is called **policy**, and the policy is essentially a mapping from every possible observation to the optimal action under the observation. The process of learning a policy by running a simulation is called the training process, and the process that an agent play the game with the policy it has learned is called the prediction.

ML-Agents provide all the tools necessary to use Unity as a simulation engine to learn the policies of different objects in the Unity environment

3 Main component

ML-Agents is a Unity plugin that contains three advanced components:

Learning Environment - contains the Unity scene and all the game characters.

Python API - This contains all machine learning algorithms for training (learning a behavior or policy). Unlike the learning environment, the Python API is not part of Unity, but is External and communicates with Unity via External Communicator.

External Communicator - This connects the Unity environment to the Python API.It's in the Unity environment.

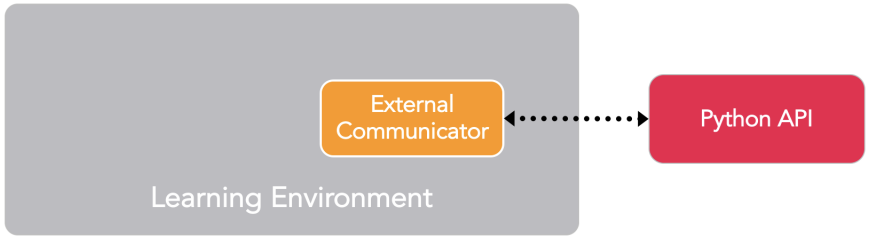

Figure 1: Simplified block diagram of ML-Agents

3.1 Learning Environment

The learning environment contains three additional components that help the organization unity scenario:

Agent - it can be attached to a unity game object (any role) in the scene), responsible for generating its observation results, executing its actions, and assigning rewards (positive/negative) in time. Each agent is associated only with a brain associated.

Brain-it encapsulates the agent's decision logic. In essence, the policy making of each agent in the brain determines the action that the agent should take in each case. More than that, it is a component of the agent receiving observations and rewards and returns actions.

Academy - it directs the agent's observation and decision-making process. Within the academy, several environmental parameters can be specified, such as rendering quality and environment running speed parameters. External communicator is located in the academy.

Each learning environment has a global Academy and multiple agents corresponding to each role in the game.Although each Agent must be associated with a Brain, multiple agents with similar observations and actions can be associated with the same Brain.In other words, the Brain defines the space of all possible observations and actions, and the agents can have their own unique values of observations and actions.

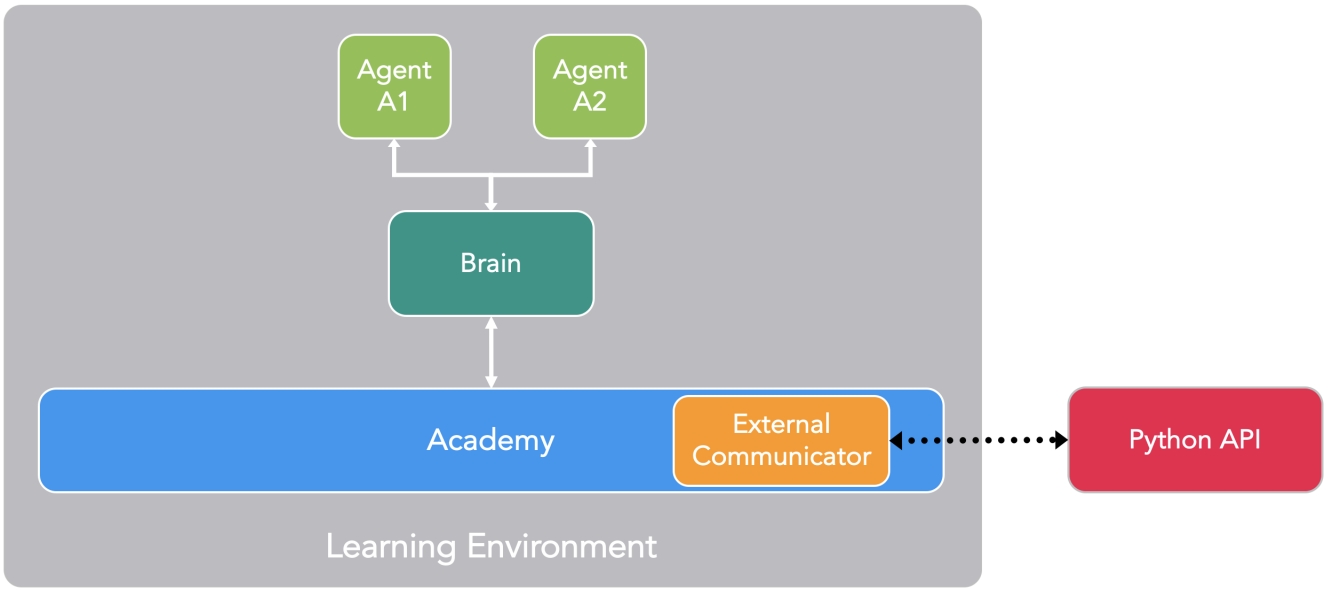

Figure 2: Block diagram of Learning environment for ML-Agents

In brief,Each game character has an Agent attached to it, and each Agent is connected to a Brain.Brain receives the observation and reward from the Agent and returns the action.In addition to being able to control the environment parameters, Academy also ensures that all Agent and Brain are in synchronization.

3.2 Brain

There are four different types of BRAIN that can be used to train and predict a wide range of situations:

External - Use the Python API for decision making.In this case, the observations and rewards collected by Brain are forwarded to the Python API via External Communicator.The Python API then returns the appropriate action that the Agent needs to take.

Internal - Use the TensorFlow model for decision making.The TensorFlow model contains the learned policy, which Brain directly uses to determine the actions of each Agent.

Player - Use the actual input from the keyboard or controller to make decisions.In this case, the human player is responsible for controlling the Agent, and the observations and rewards collected by Brain are

not used to control the Agent.

Heuristic - Decision-making with fixed a logical action, as most of game agents currently do. This type is useful for debugging agents with fixed logic behavior. It is also helpful to compare the Agent commanded by fixed logic with the trained Agent. In our example, once we have trained Brain for Mario, we can assign trained Brain for Multiple Mario, and Heuristic Brain for another Mario with fixed logic action.Then, we can assess which Mario is more effective.

According to the above description, it seems that External Communicator and Python API can only be used by External Brain. However, the Internal, Player, and Heuristic types of Brain can be configured, so that they can also send observations, rewards and actions to the Python API through the External Communicator (a function called broadcasting).

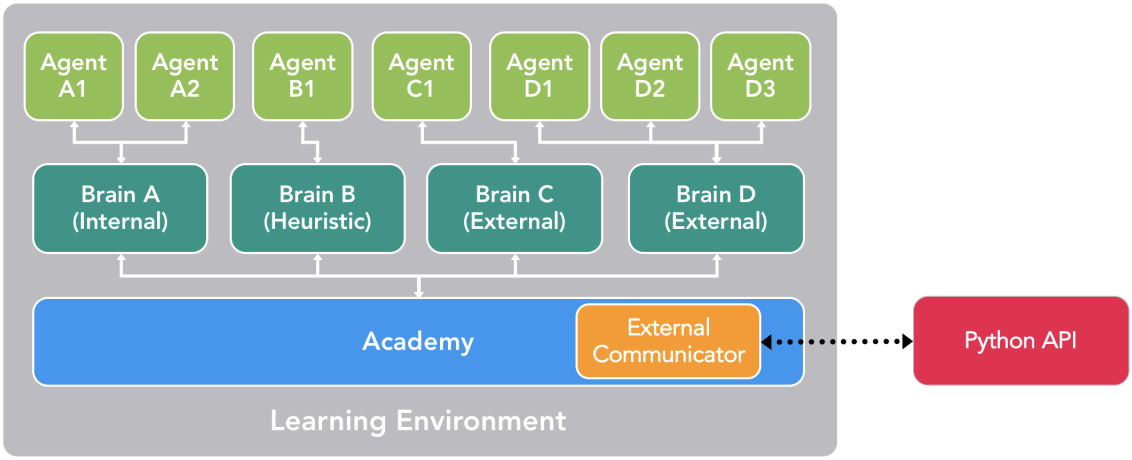

Figure 3: An example of a possible configuration for a scenario involving multiple agents and brains

4 Training model

4.1 Built-in training and prediction

As mentioned earlier, ML-Agents come with implementations of a variety of state-of-the-art algorithms for training intelligent Agents. In this mode, the BRAIN type is set to External during training and Internal during prediction. More specifically, during training, all Agents in the scenario send their observations to the Python API via the External Communicator (this is the behavior with the External Brain).The Python API processes these observations and sends back the actions to be taken by each agent. During the training, these actions were mostly exploratory, designed to help the Python API learn the best policies for each agent. At the end of the training, the policy learned by each agent can be derived.Since all of implementations are based on TensorFlow, the Policy learned is just a TensorFlow model file. Then, in the prediction phase, we switched the Brain type to Internal and added the TensorFlow model generated from the training phase. Now, during the prediction phase, Agents continue to generate their observations, but instead of sending the results to the Python API, they feed the results into their embedded TensorFlow model to generate the optimal action that each agent will take at each point in time.

In brief, Our implementation is based on TensorFlow, so during training, the Python API uses the observations received to learn the TensorFlow model. The model is then embedded into the Internal Brain during the prediction process to generate the best actions for all agents connected to the Brain.Please note that our Internal Brain is currently experimental, as it is limited to the TensorFlow model and will utilize

the third-party TensorFflowSharp library.

4.2 Curriculum Learning

This pattern is an extension of built-in training and prediction and is particularly useful for training complex behaviors in complex environments. Curriculum Learning is a way of training a machine learning model in a way that gradually introduces the more difficult aspects of a problem so that the model is always best challenged. This kind of thinking has been around for a long time, and it's how we humans usually learn. For example, any primary education in childhood, curricula and topics will be sorted. For example, arithmetic is taught first, then algebra. Again, you teach algebra first, then calculus.The skills and knowledge gained in the early courses provide the foundation for the later courses.The same goes for machine learning, where training on easier tasks can provide the foundation for harder tasks in the future.

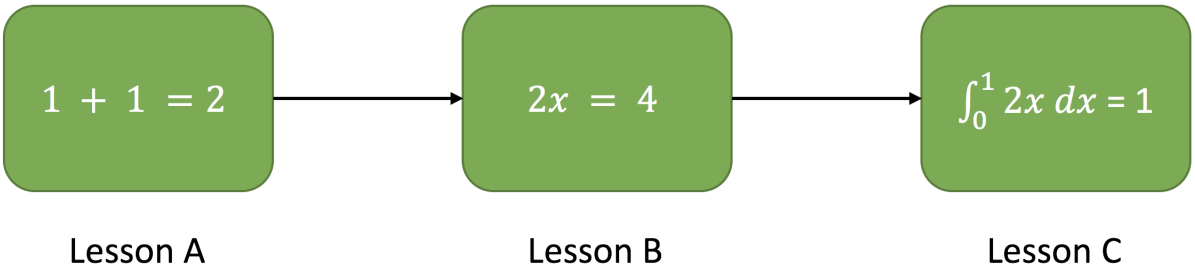

Figure 4: An example of Curriculum Learning

When we consider the actual principle of reinforcement learning, learning signals are rewards that are occasionally received throughout the training process. The starting point for training the Agent to complete this task will be a random policy. This starting policy will make the agent go around in circles, and such behavior may never be rewarded or rarely rewarded in a complex environment. Therefore, by simplifying the environment at the beginning of training, we can allow agents to quickly update random policies to more meaningful policies, that is, as the environment becomes more complex, the policies will continue to improve. In our example, we can consider training Mario in different levels ranging from a very simple level without any gap, without any enemy, to the one with various components. ML-Agents support setting custom environment parameters within the Academy.Thus, environmental elements (such as game objects) that are related to difficulty or complexity can be dynamically adjusted according to the training schedule.