Installing MuJoCo to Work With OpenAI Gym Environments

6 mins read Author Piotr Januszewski July 20th, 2021

In this article, I'll show you how to install MuJoCo on your Mac/Linux machine in order to run continuous control environments from OpenAI's Gym. These environments include classic ones like HalfCheetah, Hopper, Walker, Ant, and Humanoid and harder ones like object manipulation with a robotic arm or robotic hand dexterity. I'll also discuss additional agent diagnostics provided by the environments that you might not have considered before.

READ ALSO

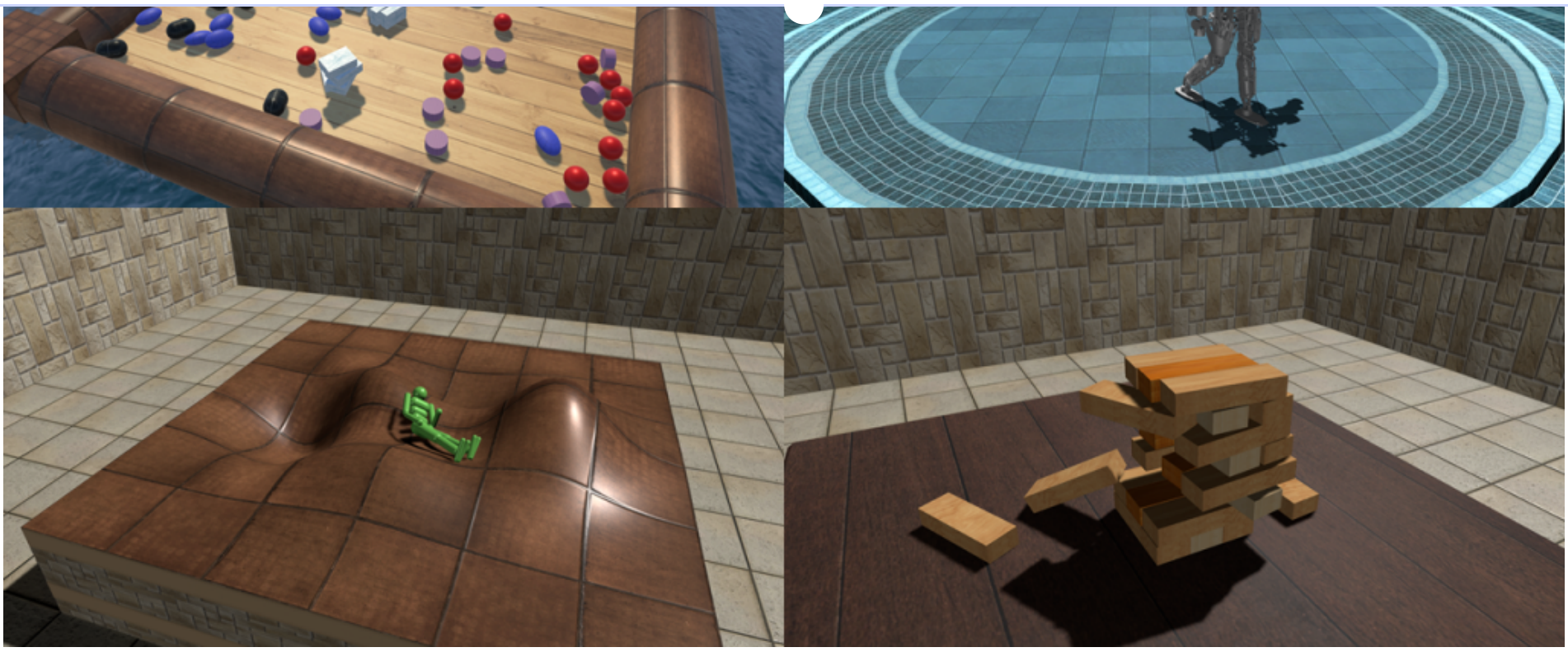
👉 The Best Tools for Reinforcement Learning in Python
👉 Best Benchmarks for Reinforcement Learning: The Ultimate List

How do you get MuJoCo?

You might wonder, what's so special about installing MuJoCo that it needs a guide? Well, getting a license and properly installing it might be relatively easy, but **the big problems start when you're matching MuJoCo and OpenAI Gym versions, and installing the mujoco-py package**. It took me many hours to get it right the first time I tried!

To save you the trouble, **I'll walk you through the installation process step by step.** Then I'll discuss some useful diagnostics to keep an eye on, we'll take a look at example diagnostics from Humanoid training. Finally, I'll link the code that lets you train agents on MuJoCo tasks and watch the diagnostics using Neptune. To start, I'll give you a bit of context about MuJoCo and OpenAI Gym environments.

MuJoCo – **Mu**lti-**Jo**int dynamics with **Co**ntact



Source: MuJoCo Plugin and Unity Integration

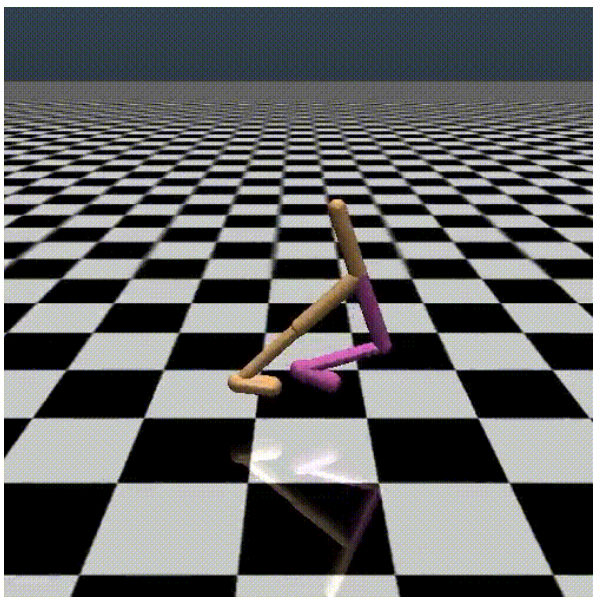
MuJoCo is a **fast and accurate physics simulation engine** aimed at research and development in robotics, biomechanics, graphics, and animation. It's an engine, meaning, it doesn't provide ready-to-use models or environments to work with, rather it **runs environments** (like those that OpenAI's Gym offers).

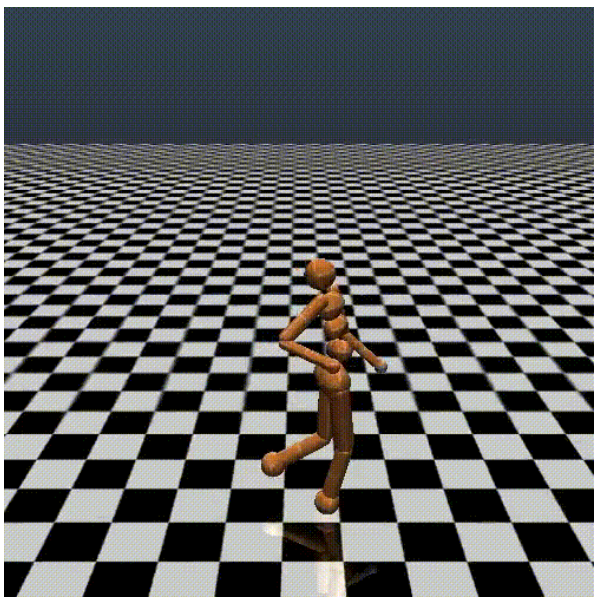
What is OpenAI Gym?

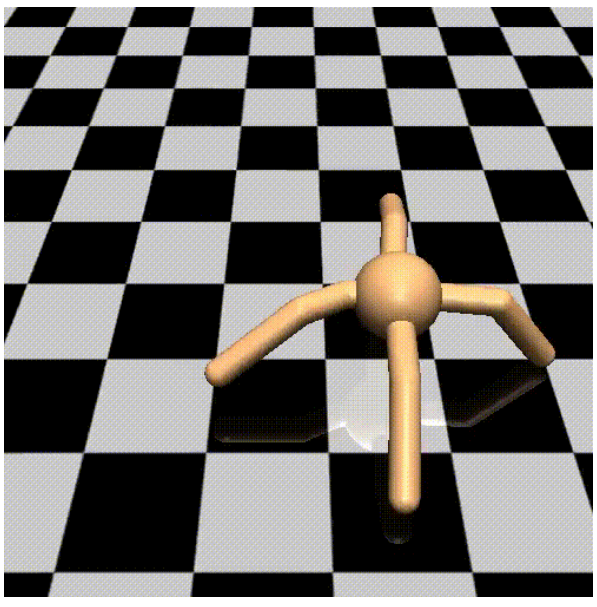
OpenAI Gym (or Gym for short) is a collection of environments. Some of them called continuous control in general, run on the MuJoCo engine. All the environments share two important characteristics:

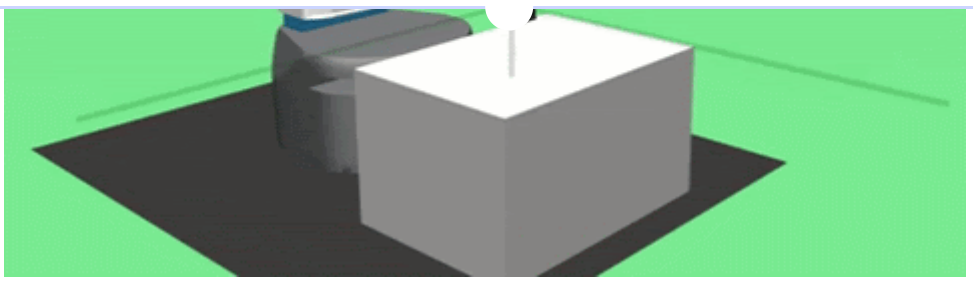
1. An agent observes vectors that describe the kinematic properties of the controlled robot. This means that the state space is continuous.
2. Agent actions are vectors too and they specify torques to be applied on the robot joints. This means that the action space is also continuous

Gym MuJoCo environments include classic continuous control, objects manipulation with a robotic arm, and robotic hand (Shadow Hand) dexterity. There are multiple tasks available for training in these environments. Some of them are presented in the figures below. You can find details about all of them in the Gym environments list. This post is especially useful for robo-arm and robo-hand environments. If you don't know the Gym API yet, I encourage you to read the documentation – the two short sections "Environments" and "Observations" should be enough to start.

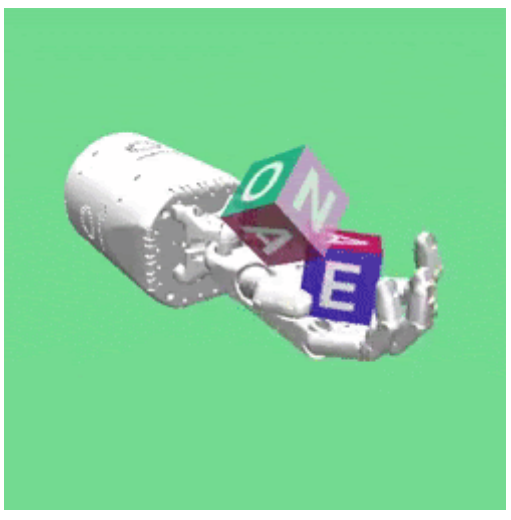








Objects manipulation with a robotic arm – the pick and place task.
Source: Overcoming exploration in RL from demos



Shadow Hand dexterity – the hand manipulate block task.
Source: OpenAI Gym Robotics

Installing MuJoCo and OpenAI Gym

In this section, I'll show you where to get the MuJoCo license, how to install everything required, and also how to troubleshoot a common macOS problem.

License

You can get a 30-day free trial on the MuJoCo website or—if you're a student—a free 1-year license for education. The license key will arrive in an email with your username and password. If you're not a student, you might try to encourage the institution you work with to buy a license.

Installing mujoco-py

Here are step-by-step instructions, and below I added some explanations and troubleshooting tips:

1. Download the MuJoCo version 1.50 binaries for Linux or macOS.
2. Unzip the downloaded `mjpro150` directory into `~/.mujoco/mjpro150`, and place your license key (the `mjkey.txt` file from your email) at `~/.mujoco/mjkey.txt`.
3. Run `pip3 install -U 'mujoco-py<1.50.2,>=1.50.1'`
4. Run `python3 -c 'import mujoco_py'`

If you see warnings like **"objc[...]: Class GLFW... is implemented in both...", then ignore them**. If you're on macOS and see "clang: error: unsupported option '-fopenmp'" or any other compilation-related error, then go to the *Troubleshooting* subsection. If you wonder why MuJoCo 1.5, then go to the *Version* subsection. If you have no more concerns, then you can jump into Gym installation!

Troubleshooting

more than one version.

3. Run `brew install llvm boost hdf5`

4. Add this to your `.bashrc / .zshrc`

```
export PATH="/usr/local/opt/llvm/bin:$PATH"
export CC="/usr/local/opt/llvm/bin/clang"
export CXX="/usr/local/opt/llvm/bin/clang++"
export CXX11="/usr/local/opt/llvm/bin/clang++"
export CXX14="/usr/local/opt/llvm/bin/clang++"
export CXX17="/usr/local/opt/llvm/bin/clang++"
export CXX1X="/usr/local/opt/llvm/bin/clang++"
export LDFLAGS="-L/usr/local/opt/llvm/lib"
export CPPFLAGS="-I/usr/local/opt/llvm/include"
```

5. Don't forget to source your `.bashrc / .zshrc` (e.g. relaunch your cmd) after editing it and make sure your python environment is activated.

6. Try to uninstall and install mujoco-py again.

See this GitHub issue for more information. You should also see the *Troubleshooting* section of the mujoco-py README.

Version

Here we bump into the first trap! **The newest OpenAI Gym doesn't work with MuJoCo 2.0**, see this GitHub issue if you want to know the details. This is why you need to download MuJoCo version 1.50 binaries. Alternatively, if you really need to use MuJoCo 2.0, you can download the MuJoCo 2.0 binaries for Linux or OSX, install the newest mujoco-py, and then install the last Gym that supports MuJoCo 2.0: `pip install -U gym[all]==0.15.3`

Installing OpenAI Gym Environments (tutorial)

Here, it's important to install the OpenAI Gym package with the "mujoco" and "robotics" extras or simply all extras:

1. Run `pip3 install gym[mujoco,robotics]` or `pip3 install gym[all]`
2. Check the installation by running:

```
python3 -c "import gym; env = gym.make('Humanoid-v2'); print('\nIt is OKAY!' if env.reset() is not None else '\nSome problem here...')"
```

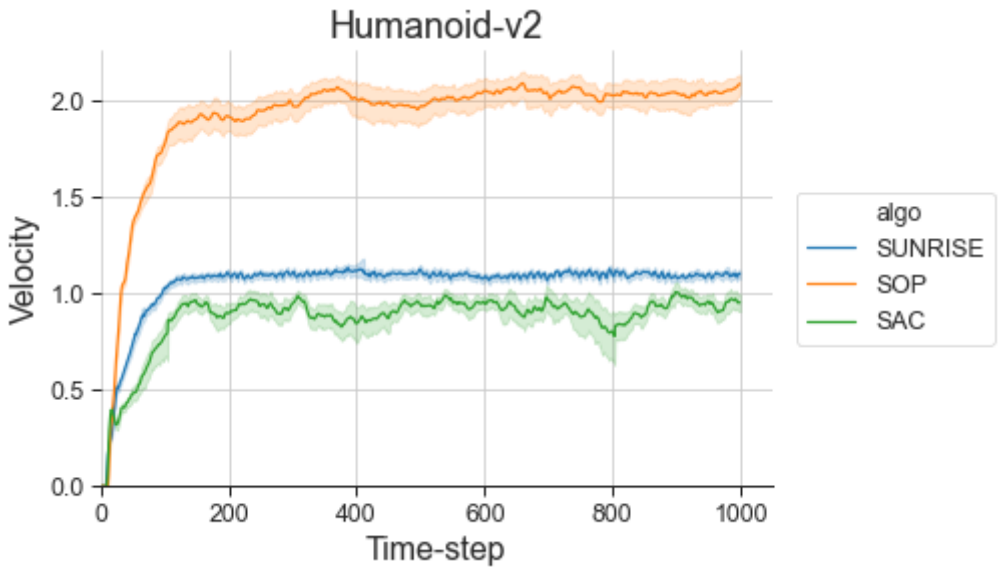
If you see "It is OKAY!" printed at the end of the cmd, then it's OKAY! Again, **you can ignore warnings like "objc[...]: Class GLFW... is implemented in both..."**.

MuJoCo diagnostics

Now I'll talk about useful metrics provided by the OpenAI Gym MuJoCo environments. They depend on an environment version, so I divide them into v2 and v3 diagnostics. You can access these metrics in an "info" dictionary provided by the environment step method: observation, reward, done, **info** = env.step(action). See Gym documentation for more. **The table below presents keys that allow you to access the metrics in the dictionary and metrics short descriptions**

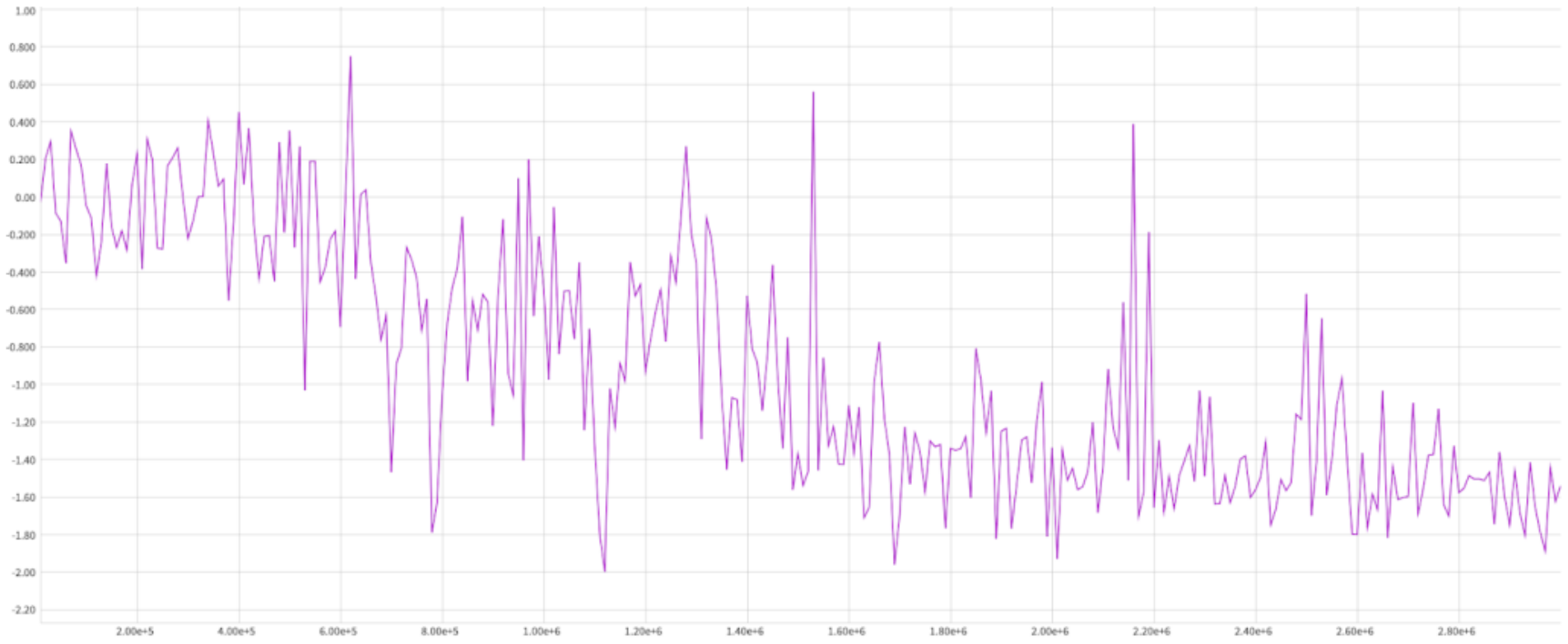
	v3	x_position	Position in the X-axis.
		x_velocity	Velocity in the X-axis (forward velocity).
Hopper	v3	x_position	Position in the X-axis.
		x_velocity	Velocity in the X-axis (forward velocity).
Walker2d	v3	x_position	Position in the X-axis.
		x_velocity	Velocity in the X-axis (forward velocity).
Ant	v2 / v3	reward_forward	The positive reward for the robot forward velocity.
		reward_ctrl	The negative reward for the robot action vector magnitude.
		reward_contact	The negative reward for the contact force magnitude between the robot and the ground.
		reward_survive	The constant positive reward at each time step when the robot is alive (until the end of an episode or the robot falls).
	v3	x_position	Position in the X-axis.
		x_velocity	Velocity in the X-axis.
		y_position	Position in the Y-axis.
		y_velocity	Velocity in the Y-axis.
		distance_from _origin	Distance from the robot starting position, (0, 0).
Humanoid	v2 / v3	reward_linvel	The positive reward for the robot forward velocity.
		reward_quadctrl	The negative reward for the robot action vector magnitude.
		reward_impact	The negative reward for the contact force magnitude between the robot and the ground.
		reward_alive	The constant positive reward at each time step when the robot is alive (until the end of an episode or the robot falls).
	v3	x_position	Position in the X-axis.
		x_velocity	Velocity in the X-axis.
		y_position	Position in the Y-axis.
		y_velocity	Velocity in the Y-axis.

Humanoid diagnostics

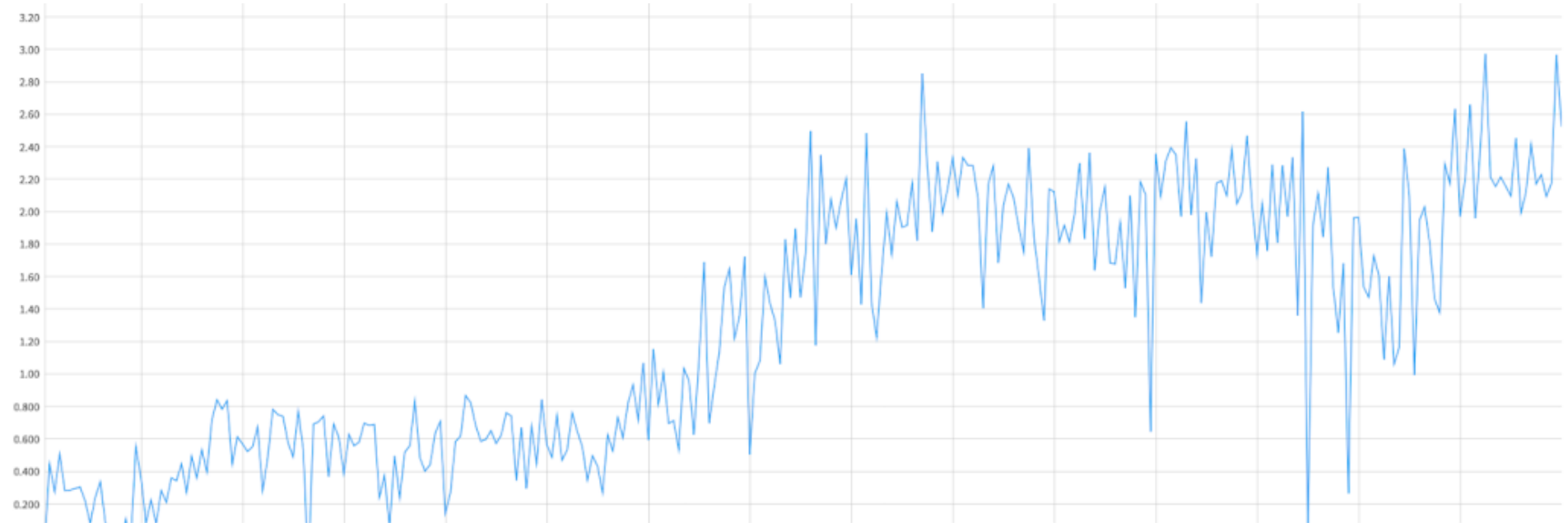


Comparison of velocities of three different DRL algorithms: SAC, SOP, and SUNR

The figure above compares velocities of three different DRL algorithms: SAC, SOP, and SUNRISE. The velocities are plotted for fully trained agents at different points of the episode. You can see that the SOP agent runs the fastest, which is the goal of this task. In the figures below we investigate the positions of the SAC agent at the end of episodes at different stages of training.



SAC final positions in the X-axis across training on the Humanoid task. Neptune experiment.



direction at some point of training, which is shown in the figures below.

SAC final positions in the X-axis across training on the Humanoid task. It changes the run direction in one-third of training. Neptune experiment.

SAC final positions in the Y-axis across training on the Humanoid task. It changes the run direction late in the training. Neptune experiment.

Conclusions

Congratulations, you've got MuJoCo up and running! Now you'll be interested in training agents in these environments—check out this repository. It includes an easy-to-understand code of DRL algorithms implemented in modern TF2. This code is based on the newcomer-friendly SpinningUp codebase. Moreover, **it includes the ability to log into the Neptune.ai platform**, which is very convenient to store and analyze the training results! I use it in my research and you have to give it a try too, especially that they shipped their new, amazing API.

Piotr Januszewski

A Ph.D. student at the Gdańsk University of Technology. Conducts his research on Deep Reinforcement Learning in the AwareLab group. His recent work with this team, "Structure and randomness in planning

READ NEXT

How to Structure, Organize, Track and Manage Reinforcement Learning (RL) Projects

7 mins read | Vladimir Lyashenko | Posted December 23, 2020

Structuring and managing machine learning projects can be a tricky thing.

When you dive into a project, you may quickly realize that you're drowning in an ocean of Python scripts, data, algorithms, functions, updates, and so on. At some point, you just lose track of your experiments, and can't even say which script or update led to the best result.

So, **structuring your project and keeping track of experiments** is a crucial part of success.

From this point of view, working on an **ML** project might be challenging in general, but some fields are more complicated than others. **Reinforcement Learning** (**RL**) is one of the complicated ones**.**

This article is dedicated to **structuring and managing RL projects**. I'll try to be as precise as possible and provide a comprehensive step-by-step guide and some useful tips.

We'll cover:

- **General tips** – project directory structure, **Cookiecutter**, keeping track of experiments using **Neptune**, proper evaluation
- **Defining a problem as an RL problem** – Reinforcement Learning, Supervised Learning, optimization problem, maximization and minimization
- **Picking an RL environment** – OpenAI Gym
- **Picking an RL library and algorithm** – RL_Coach, Tensorforce, Stable Baselines, RL_Coach guidelines
- **Testing the performance of the agent**
- **Preparing for publishing** – README, requirements, readable code, visualizations

10 Real-Life Applications of Reinforcement Learning

by Derrick Mwiti, July 22nd, 2020

Read more

How to Make Sense of the Reinforcement Learning Agents? What and Why I Log During Training and Debug

by Piotr Januszewski, September 29th, 2020

Read more

Developing AI/ML Projects for Business – Best Practices

by Sundeep Teki, May 6th, 2021

Read more

Logging in Reinforcement Learning Frameworks – What You Need to Know

by Piotr Januszewski, December 2nd, 2020

Read more