<https://github.com/rlworkgroup/garage/blob/f056fb8f6226c83d340c869e0d5312d61acf07f0/src/garage/_functions.py#L116>

<https://github.com/rlworkgroup/garage/blob/f056fb8f6226c83d340c869e0d5312d61acf07f0/src/garage/_functions.py#L73>

<https://github.com/openai/gym/blob/master/gym/core.py>

<https://github.com/rlworkgroup/garage/blob/master/src/garage/examples/step_gym_env.py>

<https://medium.com/swlh/getting-started-with-reinforcement-learning-mujoco-and-openai-gym-67243b78b599>

Misc setup and test notes

wget <https://www.roboti.us/file/mjkey.txt>

git clone https://github.com/rlworkgroup/garage.git

linux setup script with the --mjkey arg set to the path of the key

cd ~/garage && scripts/setup\_linux.sh --mjkey ../mjkey.txt

Dont need garage repo after this point

cd ~ && rm -rf garage

For installing on the vm. Had to run

pip install garage[mujoco,dm\_control]

pip install garage[dev]

python -m garage.examples.torch.maml\_trpo\_metaworld\_ml10.py

<https://github.com/rlworkgroup/garage/blob/master/src/garage/examples/torch/maml_trpo_metaworld_ml10.py>

Ahhhh….. So this is why it crashes

<https://garage.readthedocs.io/en/latest/user/sampling.html>

Plotting results

In garage, we use [TensorBoard](https://www.tensorflow.org/tensorboard) for plotting experiment results.

[This guide](https://garage.readthedocs.io/en/latest/user/monitor_experiments_with_tensorboard.html) will provide details how to set up tensorboard when running experiments in garage.

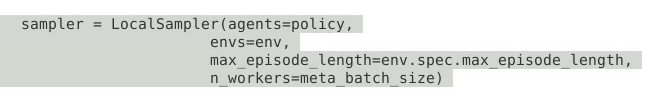
GARAGE NOTES

Has notes on how trainer works

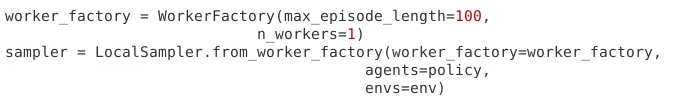
<https://garage.readthedocs.io/en/latest/user/ensure_your_experiments_are_reproducible.html>

Found the difference

Collin thinking you could just change the class name….



Correct way that Michael did:



You can also directly run an example by passing the fully qualified name to python -m, as follows:

python -m garage.examples.tf.dqn\_cartpole.py

The decorator for experiment wrapper is just for logs since -m will just treat this as a script above on any qualified experiment launcher file with run it

wget <https://www.roboti.us/file/mjkey.txt>

The garage cli command has a resume command which if you declared a log dir you should be able to cd to and run garage resume and start back up training

IMPORTANT

META-WORLD is a sub benchmark built into garage

It provides new envs(tasks) and a sampler for multiple tasks

Garage itself has benchmark tools and algos for meta RL tasks

Even though garage default has examples for meta world it is not included need to run garage[dev] to install right version

<https://github.com/rlworkgroup/metaworld#installation>

Experiment Launchers

<https://garage.readthedocs.io/en/latest/user/experiments.html>

All experiment launchers eventually call a function wrapped with a decorator called wrap\_experiment, which defines the scope of an experiment, and handles common tasks like setting up a log directory for the results of the experiment.

Experiment seems like what i would think of as the train loop in regular ol pytorch as it contains all the setup and init of a encapsulated training env, value function, optimizer and so on

IMPLEMENTING AN ALGO

<https://garage.readthedocs.io/en/latest/user/implement_algo.html>

All pytorch examples  
<https://github.com/rlworkgroup/garage/tree/f056fb8f6226c83d340c869e0d5312d61acf07f0/src/garage/examples/torch>

Example

<https://github.com/rlworkgroup/garage/tree/f056fb8f6226c83d340c869e0d5312d61acf07f0/src/garage/torch>

HAS VERY GOOD EXAMPLE OF METAWORLD EXPERIMENT LAUNCHER IN GARAGE

<https://garage.readthedocs.io/en/latest/user/meta_multi_task_rl_exp.html>

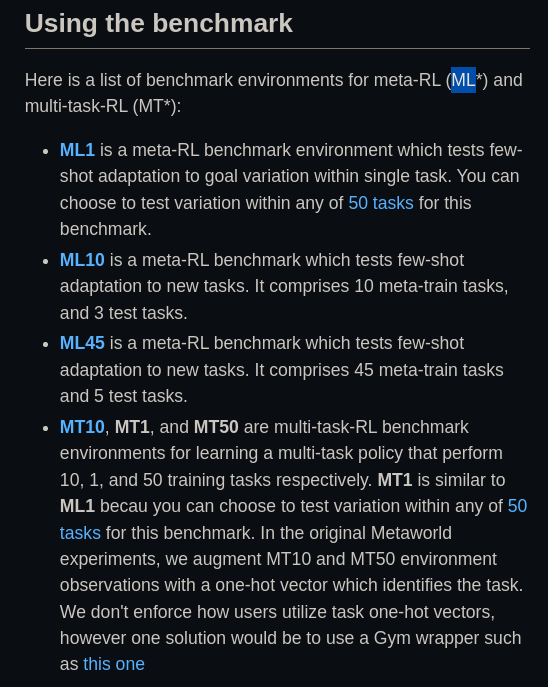
More examples of metaworld experiment launchers for various algos and differences in MT1 vs MT10

<https://github.com/rlworkgroup/garage/blob/f056fb8f6226c83d340c869e0d5312d61acf07f0/src/garage/examples/torch/mtsac_metaworld_mt1_pick_place.py>

<https://github.com/rlworkgroup/garage/blob/f056fb8f6226c83d340c869e0d5312d61acf07f0/src/garage/examples/torch/maml_trpo_metaworld_ml10.py>

One with a replay buffer it seems

<https://github.com/rlworkgroup/garage/blob/f056fb8f6226c83d340c869e0d5312d61acf07f0/src/garage/examples/torch/mtsac_metaworld_mt50.py>



WHEN LOOKING AT FILE NAMES IN EXAMPLE EXPERIMENT LAUNCHERS REFER TO THIS FOR FILE NAME MEANINGS

BENCHMARKS

<https://garage.readthedocs.io/en/latest/user/benchmarking.html#write-garage-benchmark-scripts>

when talking about a garage benchmark (not metaworld) are a layer on top of experiments for comparing multiple algos against each other

(for instance one experiment in garage may run tasks via a metaworld sampler and is thus running a RL benchmark on the algo defined in that class. But that is not the same as a garage benchmark which may be running multiple such algos all on a specific RL benchmark)

<https://github.com/rlworkgroup/garage/blob/b4abe07f0fa9bac2cb70e4a3e315c2e7e5b08507/benchmarks/src/garage_benchmarks/benchmark_algos.py>

So this is the def of the iterate experiments it seems like this is for testing multiple experiments eg configs of env and algo and sampler separately but in an automatic fashion and this handles logging and plotting of each of those experiments

<https://github.com/rlworkgroup/garage/blob/b4abe07f0fa9bac2cb70e4a3e315c2e7e5b08507/benchmarks/src/garage_benchmarks/helper.py>

Lmao even has an upload to GCP storage function

<https://github.com/rlworkgroup/garage/blob/b4abe07f0fa9bac2cb70e4a3e315c2e7e5b08507/benchmarks/src/garage_benchmarks/helper.py#L243>

CUDA SUPPORT

import torch

from garage.torch import set\_gpu\_mode

if torch.cuda.is\_available():

set\_gpu\_mode(True)

else:

set\_gpu\_mode(False)

algo.to()

COMMAND LINE RUNNER FOR GARAGE PIP PACKAGE

<https://github.com/rlworkgroup/garage/blob/master/scripts/garage>

ORG OF GARAGE

We currently support [PyTorch](https://pytorch.org/) and [TensorFlow](https://www.tensorflow.org/) for implementing the neural network portions of RL algorithms, and additions of new framework support are always welcome. PyTorch modules can be found in the package [garage.torch](https://github.com/rlworkgroup/garage/tree/master/src/garage/torch) and TensorFlow modules can be found in the package [garage.tf](https://github.com/rlworkgroup/garage/tree/master/src/garage/tf). Algorithms which do not require neural networks are found in the package [garage.np](https://github.com/rlworkgroup/garage/tree/master/src/garage/np).

Installing garage as an editable package (dont need to do this for your experiments and algos)

* <https://garage.readthedocs.io/en/latest/user/setting_up_your_development_environment.html#installing-garage-as-an-editable-package>

MISC/trash

<https://garage.readthedocs.io/en/latest/user/benchmarking.html#write-garage-benchmark-scripts>

<https://github.com/rlworkgroup/metaworld/blob/a0009ed9a208ff9864a5c1368c04c273bb20dd06/scripts/demo_sawyer.py#L357>

LOOK AT META WORLD SCRIPTS FOLDER

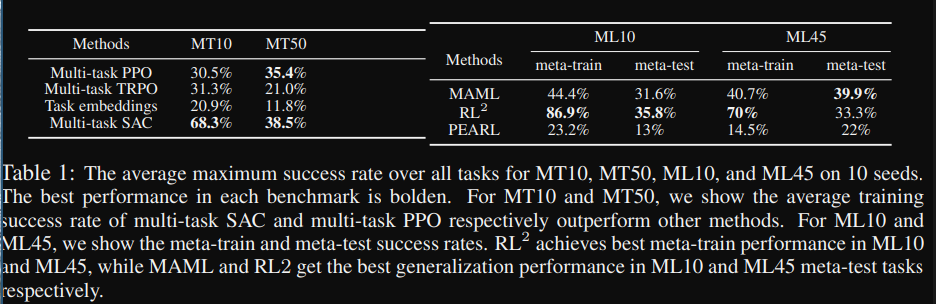
I think all they are saying in the parametric variability stuff from the paper is they have domain randomization on the tasks eg moved goals for cups so structure can be learned with assured overlap

Note that this kind of parametric variation, which we introduce for each task, essentially represents the entirety of the task distribution for previous meta-RL evaluations [10, 22], which test on single tasks (e.g., running towards a goal) with parametric variability (e.g., variation in the goal position). Our full task distribution is therefore substantially broader, since it includes this parametric variability for each of the 50 tasks.

All of our tasks are performed by a simulated Sawyer robot. The action space is a 2-tuple consisting of the change in 3D space of the end-effector followed by a normalized torque that the gripper fingers should apply. The actions in this space range between −1 and 1.

For all tasks, the robot must either manipulate one object with a variable goal position, or manipulate two objects with a fixed goal position. The observation space is represented as a 6-tuple of the 3D Cartesian positions of the end-effector, a normalized measurement of how open the gripper is, the 3D position of the first object, the quaternion of the first object, the 3D position of the second object, the quaternion of the second object, all of the previous measurements in the environment, and finally the 3D position of the goal.

ranging from the level of current goal-centric meta-RL benchmarks to a setting where methods must learn distinctly new, challenging manipulation tasks based on diverse experience across 45 tasks.



UNDERSTANDING GARAGE

Garage is the general RL training and simulation platform

An environment in reinforcement learning is a task, or simulation, that an agent interacts with. Environments in garage are very similar to Open AI’s [Gym](https://gym.openai.com/) environments. One of the main differences is that garage uses [akro](https://akro.readthedocs.io/en/latest/) to describe input and output spaces, which is an extension of the gym.Space API.

<https://github.com/rlworkgroup/garage>

garage is a toolkit for developing and evaluating reinforcement learning algorithms, and an accompanying library of state-of-the-art implementations built using that toolkit.

The toolkit provides wide range of modular tools for implementing RL algorithms, including:

Starting from version v2020.10.0, garage comes packaged with examples. To get a list of examples, run:

garage examples

Ok has a listing of algos implemented and frameworks they are implemented in

<https://github.com/rlworkgroup/garage#algorithms>

Misc / trash

<https://github.com/openai/mujoco-py/issues/44>

import os

os.environ['LD\_LIBRARY\_PATH'] = os.environ['LD\_LIBRARY\_PATH'] + ":/root/.mujoco/mujoco200/bin"

## Install Environment Dependencies (Optional)

Generally speaking, system dependencies of garage are minimal, and likely already installed. However, many of the environments garage is used with have additional dependencies, and we provide “setup scripts” for installing those dependencies and working around known problems on common platforms.

If you can already use the environments you need, you can skip this section.

THE GOSH DARN MUJOCO LICENSE KEY

A MuJoCo key is required to run these install scripts. You can get one here: <https://www.roboti.us/license.html>

In order to use those scripts, please do the following:

Clone our repository (<https://github.com/rlworkgroup/garage>) and navigate to its directory.

Then, from the root directory of the repo, run the script.

* On Linux:

./scripts/setup\_linux.sh --mjkey path-to-your-mjkey.txt

sudo apt-get install libosmesa6-dev

Rendering open gl / gym envs in collab

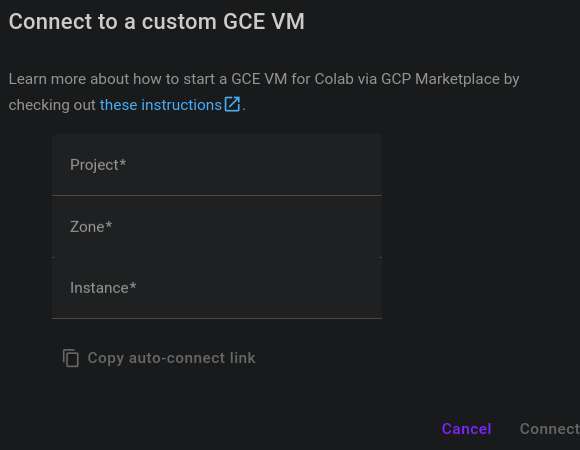
<https://davidrpugh.github.io/stochastic-expatriate-descent/openai/binder/google-colab/2020/04/16/remote-rendering-gym-envs.html>

Video recording system

<https://pypi.org/project/moviepy/>

<https://medium.com/analytics-vidhya/rendering-openai-gym-environments-in-google-colab-9df4e7d6f99f>

<https://stackoverflow.com/questions/53472940/nameerror-name-base-is-not-defined-openai-gym>



This may be all we need to get it running for longer than free collab limits

Didnt think you could do this but lit

<https://towardsdatascience.com/colab-free-gpu-ssh-visual-studio-code-server-36fe1d3c5243>

GPU allocation per user is restricted to maximum 12 hours at a time. The next time you can use it will probably be after 12 hours or once a …

Colab is able to provide free resources in part by having dynamic usage limits that sometimes fluctuate, and by not providing guaranteed or unlimited resources.