

# Training RL Agents Remotely

Reinforcement learning agents can take a long time to train. In this datalab we will learn how to train reinforcement learning agents remotely on the ADSAI servers. This will allow us to send the training job to the cloud and continue working on other tasks while the agent trains. We will also learn how to run multiple training jobs in parallel with different hyperparameters to find the best performing agent.

We will use the ClearML platform to manage the remote training jobs. We will also use the Weights and Biases platform to track the experiments and log the results.

## Remote Training Jobs: Pendulum Control with RL

First lets start with a simple example to get familiar with the process. We will use the Stable Baselines 3 library to train a reinforcement learning agent to control the pendulum environment.

Use the following code to train the agent locally first:

```
from stable_baselines3 import PPO
import gym
import time

env = gym.make('Pendulum-v1', g=9.81)

model = PPO('MlpPolicy', env, verbose=1)

model.learn(total_timesteps=10000, progress_bar=True)
```

This time we leave out the test code since we will be running the agent remotely, after testing it locally. We need to test the agent locally first to make sure it is working correctly before we send it to the cloud to train. This will save us time.

## Add Weights and Biases for Experiment Tracking (Optional)

Set your API key using os environment variables:

```
import os

os.environ['WANDB_API_KEY'] = 'INSERT_API_KEY_HERE'
```

Initialize the wandb project and add the wandb callback to the model:

```
import wandb
from wandb.integration.sb3 import WandbCallback

# initialize wandb project
run = wandb.init(project="sb3_pendulum_demo", sync_tensorboard=True)

# add tensorboard Logging to the model
model = PPO('MlpPolicy', env, verbose=1, tensorboard_log=f"runs/{run.id}")

# create wandb callback
wandb_callback = WandbCallback(model_save_freq=10000,
                               model_save_path=f"models/{run.id}",
                               verbose=2,
                               )

# add wandb callback to the model training
model.learn(total_timesteps=timesteps, callback=wandb_callback, progress_bar=True, tb_log_name=f"runs/".
```

## Save the model incrementally

Since these models can take a long time to train, it is a good idea to save the model periodically so that you can resume training later if needed. We can also use the saved models to test the agent in the environment and visualize the progress.

Simply include the learn function in a for loop and save the model after each iteration:

```
# variable for how often to save the model
time_steps = 100000
for i in range(10):

    # add the reset_num_timesteps=False argument to the Learn function to prevent the model from resetting
    # add the tb_log_name argument to the Learn function to log the tensorboard data to the correct folder
    model.learn(total_timesteps=timesteps, callback=wandb_callback, progress_bar=True, reset_num_timesteps=False)
    # save the model to the models folder with the run id and the current timestep
    model.save(f"models/{run.id}/{timesteps*(i+1)}")
```

## Add command line arguments to the training script for the hyperparameters

This will allow you to easily change the hyperparameters without having to edit the training script. It will also allow you to run multiple training jobs with different hyperparameters in parallel, by simply cloning job in the ClearML dashboard and changing the hyperparameters through the UI.

Define the arguments:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--learning_rate", type=float, default=0.0003)
parser.add_argument("--batch_size", type=int, default=64)
parser.add_argument("--n_steps", type=int, default=2048)
parser.add_argument("--n_epochs", type=int, default=10)

args = parser.parse_args()
```

Add the arguments to the model:

```
model = PPO('MlpPolicy', env, verbose=1,
            learning_rate=args.learning_rate,
            batch_size=args.batch_size,
            n_steps=args.n_steps,
            n_epochs=args.n_epochs,
            tensorboard_log=f"runs/{run.id},)
```

To run the training script with the command line arguments, use the following command in the terminal:

```
python train_pendulum.py --learning_rate 0.0001 --batch_size 64 --n_steps 2048 --n_epochs 10
```

You will need to replace `train_pendulum.py` with the name of your training script, and replace the hyperparameters with the ones you want to use.

If this runs successfully, you are ready to start training the agent remotely.

# Remote Training with ClearML

## Installation and Setup

If you haven't already, install the ClearML package and initialize the configuration file. If you are using remote JupyterLab, you will have already done this and can skip to running the training job. Run the following commands in the terminal (make sure you are in the correct virtual environment):

```
pip install clearml
```

```
clearml-init
```

Copy these credentials when prompted:

```
api {
    web_server: http://194.171.191.227:8080
    api_server: http://194.171.191.227:8008
    files_server: http://194.171.191.227:8081
    # Students
    credentials {
        "access_key" = "UJODYOFVELU1XCB70FM2FKU7XY48K"
        "secret_key" = "OKCS8xT-vngmYWgpIMYsu_GbS2fLgMmMp1MbzqyLZdWZtA-FGx1UJ5KGISFGPMdcDDK"
    }
}
```

Leave all other settings as default.

Note: You need to be connected to the VPN to queue jobs and access the dashboard.

## Github

ClearML uses git to clone your training script and any other files you need to run the job. Make sure you have the latest version of your training script committed to your github repository.

If you have permission errors related to cloning your github repository when running remote jobs a simple fix is to create a public repository on github and push only the files you need to run the job to the public repository.

## Running Remote Training Jobs

Add the following code to the top of your training script after the imports:

```
from clearml import Task

# Replace Pendulum-v1/YourName with your own project name (Folder/YourName, e.g. 2022-Y2B-RoboSuite/Mic
task = Task.init(project_name='Pendulum-v1/YourName', # NB: Replace YourName with your own name
                  task_name='Experiment1')

#copy these Lines exactly as they are
#setting the base docker image
task.set_base_docker('deanis/2023y2b-r1:latest')
#setting the task to run remotely on the default queue
task.execute_remotely(queue_name="default")
```



If you also want to save the model and upload it to ClearML for later download add the following code to your training script (the saved model file will appear in the artifacts tab):

```
model.save("model_name")
task.upload_artifact("model", artifact_object="model_name.zip")
```

The above code does the following:

- Initializes the task in the ClearML platform using the project name and task name you specify. This will create a new task in the ClearML dashboard.
- Sets the base docker image to use for the training job. This is the docker image that will be used to run the training job. A docker image is a virtual machine that contains all the libraries and dependencies needed to run the training job. One has been created for you and is available on dockerhub. Just use the provided code to load the docker image.
- Sets the task to run remotely on the default queue. This will send the training job to the cloud to run on the default queue. You can also specify a specific queue to run the job on. For

example, if you want to run the job on the GPU queue, you would use

`task.execute_remotely(queue_name="gpu")`. You can see what queues are available in the ClearML dashboard.

## How to Run the Training Job

To run the training job, simply run the training script in the terminal with the desired hyperparameter arguments. For example:

```
python train.py --learning_rate 0.0001 --batch_size 64 --n_steps 2048 --n_epochs 10
```

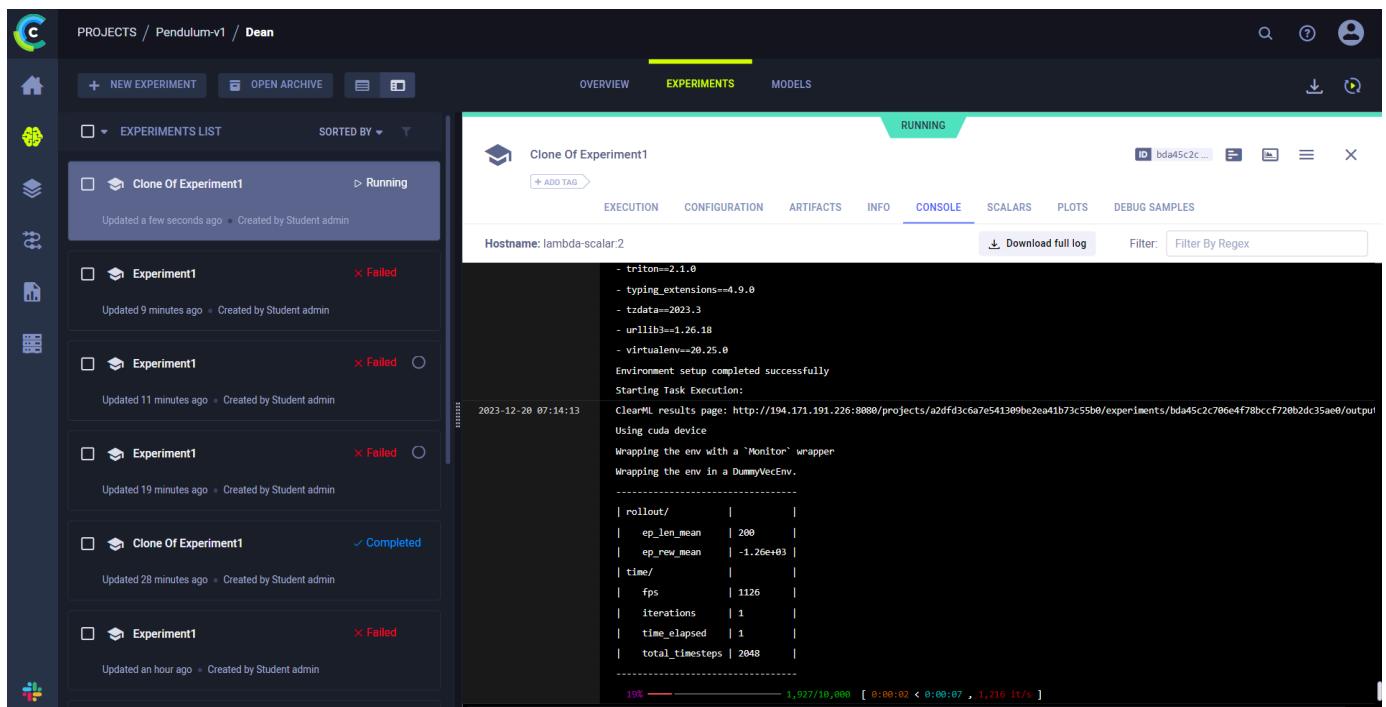
- Note: Replace `train.py` with the name of your training script, and replace the hyperparameters with the ones you want to use.

## How do I know if it is working?

Go to the ClearML [dashboard](#). Navigate to the project you created and click on the task you just created. You should see the task running in the dashboard. You can click on the task to see the logs and other information about the task. The console tab will show you the output from the training script.

Your login credentials *THE SAME AS YOUR CODER CREDENTIALS*.

If it is working correctly, you should see the following output in the console tab:



If there are errors, you will see them in the console tab, and you can use the error message to debug the issue. If you are having trouble debugging the issue.

# There are missing dependencies in the docker image! What do I do?

If you get an error saying there are missing dependencies in the docker image like in the screenshot below:

The screenshot shows the Remote RL Training interface. On the left, the 'EXPERIMENTS LIST' sidebar shows several experiments, most of which are failed. On the right, the 'Experiment1' details page is open, specifically the 'FAILED' tab. The 'CONSOLE' tab is selected, displaying the following error message:

```
- virtualenv==20.25.0
Environment setup completed successfully
Starting Task Execution:
2023-12-20 07:05:07
ClearML results page: http://194.171.191.226:8880/projects/a2df3c6a7e541309be2ea41b73c55b0/experiments/f376bb90ced4f25956a13b5053c8c1c/output
Using cuda device
Wrapping the env with a 'Monitor' wrapper
Wrapping the env in a DummyVecEnv.
Traceback (most recent call last):
File "/root/clearml/venvs-builds/3.10/task_repository/2023-y2b-clearml-test.git/pendulum_clearml.py", line 18, in <module>
    model.learn(total_timesteps=10000, progress_bar=True)
File "/root/clearml/venvs-builds/3.10/lib/python3.10/site-packages/stable_baselines3/ppo/ppo.py", line 315, in learn
    return super().learn(
File "/root/clearml/venvs-builds/3.10/lib/python3.10/site-packages/stable_baselines3/common/on_policy_algorithm.py", line 264, in learn
    total_timesteps, callback = self._setup_learn(
File "/root/clearml/venvs-builds/3.10/lib/python3.10/site-packages/stable_baselines3/common/base_class.py", line 434, in _setup_learn
    callback = self._init_callback(callback, progress_bar)
File "/root/clearml/venvs-builds/3.10/lib/python3.10/site-packages/stable_baselines3/common/base_class.py", line 378, in _init_callback
    callback = CallbackList([callback, ProgressBarCallback()])
File "/root/clearml/venvs-builds/3.10/lib/python3.10/site-packages/stable_baselines3/common/callbacks.py", line 690, in __init__
    raise ImportError(
ImportError: You must install tqdm and rich in order to use the progress bar callback. It is included if you install stable-baselines with the
Process failed, exit code 1
2023-12-20 07:05:22
```

You can simply clone the experiment and add the missing dependencies to the list of installed packages in the execution tab. For example, if you get an error saying the package `tqdm` is missing, you can add `tqdm` to the list of installed packages and rerun the job.

The screenshot shows the Remote RL Training interface. The 'EXPERIMENTS LIST' sidebar shows the same set of experiments as the previous screenshot. On the right, the 'Clone Of Experiment1' details page is open, specifically the 'DRAFT' tab. The 'EXECUTION' tab is highlighted with a red circle. Below it, the 'INSTALLED PACKAGES' section is shown, listing the following packages:

```
attrs==22.2.0
certifi==2023.11.17
charset-normalizer==3.3.2
clearml==1.13.2
cloudpickle==3.0.0
contourpy==1.1.2
cycler==0.12.1
```

A red circle also highlights the 'EDIT' button at the bottom right of the package list.

## Cloning Jobs and Changing Hyperparameters

You can clone jobs in the ClearML dashboard to run multiple training jobs with different hyperparameters in parallel. Simply click the clone button on the job you want to clone and change the hyperparameters in the UI. You can also just rerun the job with different hyperparameters in your terminal.

- Clone job:

The screenshot shows the ClearML dashboard interface. On the left, there's a sidebar with various icons and a list of experiments. The main area shows a detailed view of an experiment named 'Experiment1'. A context menu is open over 'Experiment1', with the 'Clone' option highlighted by a red arrow. The main panel displays the experiment's configuration, including its source code (repository: https://github.com/deanis/RLDemo.git, commit ID: 18df89b354e2c174e9bdec46130251109e268b1b), uncommitted changes (no changes logged), and installed packages (Python 3.8.10, ale\_py 0.7.4, clearml 1.7.1, stable\_baselines3 1.6.2, tensorboard 2.10.0, wandb 0.13.3).

- Select cloned job and change hyperparameters on the configuration tab:

The screenshot shows the ClearML dashboard after cloning 'Experiment1' to 'Clone Of Experiment1'. The 'CONFIGURATION' tab is selected. The configuration table shows the following hyperparameters:

HYPER PARAMETERS	ARGS
batch_size	64
learning_rate	0.0003
n_epochs	10
n_steps	4096

To start running the jobs right click on the cloned job and select enqueue.

## ClearML Tutorials and Documentation (Optional)

If you want more information about ClearML, you can check out the tutorials and documentation here:

- [ClearML Docs](#)

## ClearML Onboarding Walkthrough - PART 1: Expe...



<iframe width="480" height="270" src="https://www.youtube.com/embed/PDQQxKeuK0A" title="ClearML Onboarding Walkthrough - PART 2: Remote Task Execution and Automations" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>

,Applied Data Science and Artificial Intelligence @ Breda University of Applied Sciences

Contact : [Frank Peters](#)