A Practical Guide to Running Experiments in RL

Subho, April 2023

Overview

- A Typical Workflow for RL Experiments.
- Automating various aspects of the **experiment pipeline**.
  - Experiment Tracking (**Weights and Biases**)
  - Logging and Configs (**Hydra**)
  - Running hyper parameter sweeps (**Submitit plugin**)
- Reproducible experiments using Singularity containers.

Managing Experiments can be Cumbersome!

- Typical Empirical RL workflow:
  - Choose a simple problem setting that tests a specific property/answers a specific research question.
  - Identify set of baselines/approaches (**experiment**)
  - Repeat each experiment with multiple seeds, parameters (**run**)
  - Aggregate runs and compare results from experiments (**result**)

Managing Experiments can be Cumbersome!

This can be tricky to do in practice:

As the number of experiments grows:

- Organizing data becomes cumbersome.
- Managing multiple experiment configurations can become messy.

- Scheduling sweep jobs in CC is not straightforward.
  - Improper scheduling strategies affect priority of jobs!
  - Training environments are difficult to replicate across multiple CC clusters.

A good experiment workflow **must** make it easy to iterate over multiple ideas

My Current Workflow

1. Write a **Hydra** config file that specifies my experiment (combination of runs)
2. Schedule runs in Compute Canada using **submitit** (one run = one job).
3. Retrieve run data from **Wandb** Database using **Jupyter** Notebook.
4. Generate Results using **Matplotlib** + **Pandas**.

Experiment Tracking

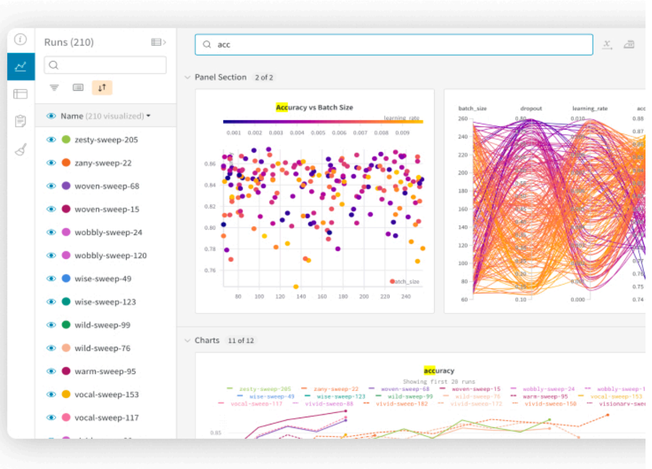
Available tools: Weights and Biases, MLFlow, Comet.ml

Features of a good experiment tracking tool:

- A central database to store data (configs, metrics, rollouts, etc) from all runs.
- Ability to filter and search by config parameters.
- Ability to tag, modify configs after the run,
- Secure storage, because you do not want to lose your valuable runs :)

Answer:

Weights & Biases



https://wandb.ai/

Simple Tracking Demo

00 main_start.py > ...

```python
import random
import numpy as np
import logging

logger = logging.getLogger("cool_project")
logger.setLevel(logging.INFO)
logging.basicConfig()

def single_run(config):
    # Your crazy RL experiment here
    for i in range(config['num_steps']):
        # Craete a dummy learning curve
        # Dummy exponential decay
        loss = np.exp(-i / config['decay_rate'])+random.random()
        # Log the loss
        logger.info("loss: %f", loss)


if __name__ == "__main__":
    config = {
        'seed': 42,
        'num_steps': 100,
        'decay_rate': 10
    }
    single_run(config)
```

Managing Configs and Log Outputs

Available tools: Hydra, python-fire

Features of a good config, log management toolkit:

- Ability to specify complex configs (parent/group configs) through json/yaml.
- Automatically organizes log outputs from runs.
- Ability to specify hyper parameter sweeps without code modifications.
- Ability to override config configurations.

Answer:

https://hydra.cc/docs/intro/

Hydra

- A config management tool developed by Facebook.
- Makes it super easy to define hierarchical configs.
- Configs can be overridden using the override keyword, or through CLI
- Automatically manages the run logs into folders.
- Ability to specify hyper-parameter sweeps.

In this example, we configure a server. The server can host multiple websites at the same time.

Config directory

```
├── config.yaml
└── server
    ├── apache.yaml
    └── site
        ├── amazon.yaml
        ├── fb.yaml
        └── google.yaml
```

config.yaml

```yaml
defaults:
  - server/apache
```

server/apache.yaml

```yaml
defaults:
  - site:
    - fb
    - google

host: localhost
port: 443
```

server/site/amazon.yaml

```yaml
amazon:
  domain: amazon.com
```

server/site/fb.yaml

```yaml
fb:
  domain: facebook.com
```

server/site/google.yaml

```yaml
google:
  domain: google.com
```

Output:

$ python my_app.py

```yaml
server:
  site:
    fb:
      domain: facebook.com
    google:
      domain: google.com
  host: localhost
  port: 443
```

Specifying Sweeps

- Sweeps can be specified using Hydra and run in parallel across:
  - Multiple processes
  - CC array jobs
- All we need to do is write a config file. No code change required!
- The same config can also launch parallel jobs in CC.

```
$ python my_app.py hydra.mode=MULTIRUN db=mysql,postgresql schema=warehouse,support,school

[2021-01-20 17:25:03,317][HYDRA] Launching 6 jobs locally
[2021-01-20 17:25:03,318][HYDRA]        #0 : db=mysql schema=warehouse
[2021-01-20 17:25:03,458][HYDRA]        #1 : db=mysql schema=support
[2021-01-20 17:25:03,602][HYDRA]        #2 : db=mysql schema=school
[2021-01-20 17:25:03,755][HYDRA]        #3 : db=postgresql schema=warehouse
[2021-01-20 17:25:03,895][HYDRA]        #4 : db=postgresql schema=support
[2021-01-20 17:25:04,040][HYDRA]        #5 : db=postgresql schema=school
```

Reproducibility using Singularity Containers

- Replicating exact set of dependencies in CC can be challenging:
  - Packages are often outdated/unavailable.
  - Some dependencies require sudo access.
  - Need to redo installations across all CC clusters

Answer:

Singularity Containers 

Singularity

- Singularity is a way to run Docker containers in multi-user clusters such as CC.
- It gives us a way to **package** scientific software and **deploy** them to *different* clusters having the *same* architecture.
- Steps to run Singularity in CC:
  - Start with a Dockerfile (a recipe to build an image)
  - Build a Docker image.
  - Use a different Linux machine to convert Docker image to .sif
  - Copy .sif to CC clusters and issue commands using:

```
$ singularity run -B /home -B /project -B /scratch -B /localscratch:/temp myimage.simg some-program
```

https://docs.alliancecan.ca/wiki/Singularity#Overview

But, what is a Dockerfile?

Its a recipe to build a Linux image.

```
# System
FROM nvidia/cuda:11.4.2-cudnn8-devel-ubuntu20.04
```

START WITH BASE IMAGE

```
RUN apt-get update && apt-get install -y \
  ffmpeg git python3-pip vim libglew-dev \
  x11-xserver-utils xvfb \
  && apt-get clean
RUN pip3 install --upgrade pip

RUN sh scripts/install-dmlab.sh
RUN sh scripts/install-atari.sh
RUN sh scripts/install-minecraft.sh

# Agent
RUN pip3 install jax[cuda11_cudnn82] -f https://storage.googleapis.com/jax-releases/jax_cuda_releases.html
RUN pip3 install jaxlib
RUN pip3 install tensorflow_probability
RUN pip3 install optax
RUN pip3 install tensorflow-cpu
ENV XLA_PYTHON_CLIENT_MEM_FRACTION 0.8
```

RUN REGULAR LINUX COMMANDS
TO INSTALL DEPENDENCIES

```
(rlt) → Practical Guide Materials docker build -f  Dockerfile -t dreamerv3 . --no-cache --platform=linux/arm64
```

BUILD RECIPE INTO A NEW IMAGE

Useful Resources

- Weights and Biases: https://docs.wandb.ai/quickstart
- Hydra: https://hydra.cc/docs/intro/
- Submitit: https://github.com/facebookincubator/submitit
- Submitit Hydra Launcher Plugin: https://hydra.cc/docs/plugins/submitit_launcher/
- Docker getting started: https://docker-curriculum.com/
- Singularity installation in Ubuntu 20.04: https://github.com/apptainer/singularity/issues/5099
- Singularity in CC: https://docs.alliancecan.ca/wiki/Singularity

Summary

- Having the right set of tools and pipeline can make experimentation easy.
- Various tools could be used to automate various aspects of the experiment pipeline:
  - Experiment Tracking (**Weights and Biases**)
  - Logging and Configs (**Hydra**)
  - Running hyper parameter sweeps (**Submitit plugin**)
- The set of tools often depends on your use-case.