Deep Surrogate Assisted Generation of Environments

This repository is the official implementation of "Deep Surrogate Assisted Generation of Environments."

Contents

Manifest

- `config/`: gin configuration files.
- `src/`: Python implementation and related tools.
- `scripts/`: Bash scripts.

Installation

1. **Install Singularity:** All of our code runs in a Singularity container. Singularity is a container platform (similar in many ways to Docker). Please see the instructions here for installing Singularity 3.6.
2. **Build the Container:** Build the Singularity container with

   ```
   sudo make container.sif
   ```

3. **Install NVIDIA Drivers and CUDA:** The node where the main script runs should have a GPU with NVIDIA drivers and CUDA installed (in the future, we may try to put CUDA in the container instead).

Running on HPC with Singularity

1. Follow all the instructions above, and after creating the container, copy `container.sif` to the HPC, e.g. with `scp`.

Instructions

Structure: Dask is the distributed compute library we use. When we run an experiment, we connect to a Dask scheduler, which is in turn connected to one or more Dask workers. Each component runs in a Singularity container.

Overview: The experiment script is located in `src/main.py`. The following instructions describe how to set up Dask locally or on a cluster running SLURM and run the script.

Note: We assume you have placed this project somewhere in your home directory. Singularity binds the home directory and executes code in it by default. If you need to use another directory, you may want to look into bind mounts for Singularity.

Note: The Makefile has many useful commands. Run `make` for a full command reference.

Logging Directory Manifest

Regardless of where the script is run, the log files and results are placed in a logging directory in `logs/`. The directory's name is of the form `%Y-%m-%d_%H-%M-%S_<dashed-name>_<uuid>`, e.g. `2020-12-01_15-00-30_experiment-1_ff1dcb2b`. Inside each directory are the following files:

```
- config.gin  # All experiment config variables, lumped into one file.
- seed  # Text file containing the seed for the experiment.
- reload.pkl  # Data necessary to reload the experiment if it fails.
- reload_em.pkl  # Pickle data for EmulationModel.
```

```
- reload_em.pth  # PyTorch models for EmulationModel.
- metrics.json  # Data for a MetricLogger with info from the entire run, e.g. QD score.
- hpc_config.sh  # Same as the config in the Slurm dir, if Slurm is used.
- archive/  # Snapshots of the full archive, including solutions and metadata,
            # in pickle format.
- archive_history.pkl  # Stores objective values and behavior values necessary
                       # to reconstruct the archive. Solutions and metadata are
                       # excluded to save memory.
- dashboard_status.txt  # Job status which can be picked up by dashboard scripts.
                        # Only used during execution.
- slurm_YYYY-MM-DD_HH-MM-SS/  # Slurm log dir (only exists if using Slurm).
                              # There can be a few of these if there were reloads.
  - config/
    - config.sh  # Possibly has a different name.
  - job_ids.txt  # Job IDs; can be used to cancel job (scripts/slurm_cancel.sh).
  - logdir  # File containing the name of the main logdir.
  - scheduler.slurm  # Slurm script for scheduler and experiment invocation.
  - scheduler.out  # stdout and stderr from running scheduler.slurm.
  - worker-{i}.slurm  # Slurm script for worker i.
  - worker-{i}.out  # stdout and stderr for worker i.
```

Running Locally

Single Run To run one experiment locally, use:

```
bash scripts/run_local.sh CONFIG SEED NUM_WORKERS
```

For instance, with 4 workers:

```
bash scripts/run_local.sh config/foo.gin 42 4
```

`CONFIG` is the gin experiment config for `src.main`.

Running on Slurm

Use the following command to run an experiment on an HPC with Slurm (and Singularity) installed:

```
bash scripts/run_slurm.sh CONFIG SEED HPC_CONFIG
```

For example:

```
bash scripts/run_slurm.sh config/foo.gin 42 config/hpc/test.sh
```

`CONFIG` is the experiment config for `src.main`, and `HPC_CONFIG` is a shell file that is sourced by the script to provide configuration for the Slurm cluster. See `config/hpc` for example files.

Once the script has run, it will output commands like the following:

- `tail -f ...` - You can use this to monitor stdout and stderr of the main experiment script. Run it.
- `bash scripts/slurm_cancel.sh ...` - This will cancel the job.
- `bash scripts/slurm_postprocess.sh ...` - This will move the slurm logs into the logging directory. Run it *after* the experiment has finished.

You can monitor the status of all your Slurm jobs by setting an alias like:

```
alias swatch="watch \"squeue -o '  %10i %.9P %.2t %.8p %.4D %.3C %.10M %30j %R' -u $USER\""
```

And then running `swatch`.

Finally, the `slurm_dashboard.sh` script may be helpful for monitoring jobs. It also includes the job listing using the `squeue` command shown above.

```
watch scripts/slurm_dashboard.sh
```

Since the dashboard output can be quite long, it can be useful to be able to scroll through it. For this, consider an alternative to `watch`, such as viddy.

Reloading

While the experiment is running, its state is saved to `reload.pkl` in the logging directory. If the experiment fails, e.g. due to memory limits, time limits, or network connection issues, `reload.pkl` may be used to continue the experiment. To do so, execute the same command as before, but append the path to the logging directory of the failed experiment.

```
bash scripts/run_slurm.sh config/foo.gin 42 config/hpc/test.sh -r logs/.../
```

The experiment will then run to completion in the same logging directory. This works with `scripts/run_local.sh` too.

For convenience, `scripts/slurm_reload.sh` will also continue an experiment. Note that unlike the command above, which uses configs in the `config` dir, this script uses configs in the logging directory. This may be useful if configs have changed since the job was originally started. For example:

```
bash scripts/slurm_reload.sh logs/.../
```

Testing

There are some tests alongside the code to ensure basic correctness. To run these, start a Singularity container with:

```
make shell
```

Within that container, execute:

```
make test
```

Reproducing Paper Results

The `config/` directory contains the config files required to run the experiments shown in the paper. Below is a brief description of each config:

```
config/
    maze/
        dsage.gin: DSAGE
        dsage_only_anc.gin: DSAGE-Only Anc
        dsage_only_down.gin: DSAGE-Only Down
        dsage_basic.gin: DSAGE Basic
        me.gin: MAP-Elites

        dsage_rsample.gin: DSAGE with random sampling instead of downsampling
        new_measures/
            dsage_rep_explore.gin: DSAGE with "repeated visits" and "maze exploration" measures
            dsage_block_explore.gin: DSAGE with "number of wall cells" and "maze exploration" measures
            dsage_block_rep.gin: DSAGE with "number of wall cells" and "repeated visits" measures

    mario/
        dsage.gin: DSAGE
        dsage_only_anc.gin: DSAGE-Only Anc
        dsage_only_down.gin: DSAGE-Only Down
        dsage_basic.gin: DSAGE Basic
        cma_me.gin: CMA-ME

        dsage_rsample.gin: DSAGE with random sampling instead of downsampling
        new_measures/
            dsage_sky_enemies.gin: DSAGE with "sky tiles" and "enemies killed" measures
            dsage_jumps_enemies.gin: DSAGE with "number of jumps" and "enemies killed" measures
```

Running with the configs above will produce multiple logging directories, which can be assembled and compiled into results with the instructions in `src/analysis/figures.py`. The results should look like the following:

Maze

	QD-score	Archive Coverage
DSAGE	16,446.60 ± 42.27	0.40 ± 0.00
DSAGE-Only Anc	14,568.00 ± 434.56	0.35 ± 0.01
DSAGE-Only Down	14,205.20 ± 40.86	0.34 ± 0.00

	QD-score	Archive Coverage
DSAGE Basic	11,740.00 ± 84.13	0.28 ± 0.00
MAP-Elites	10,480.80 ± 150.13	0.25 ± 0.00

Mario

	QD-score	Archive Coverage
DSAGE	4,362.29 ± 72.54	0.30 ± 0.00
DSAGE-Only Anc	2,045.28 ± 201.64	0.16 ± 0.01
DSAGE-Only Down	4,067.42 ± 102.06	0.30 ± 0.01
DSAGE Basic	1,306.11 ± 50.90	0.11 ± 0.01
CMA-ME	1,840.17 ± 95.76	0.13 ± 0.01

Miscellaneous

- In our code, we use the term `EmulationModel` to refer to the deep surrogate model.
- We use the terms `blocks`, `tiles`, `cells` interchangeably in the Maze domain.
- We also use the term `cells` to denote the cells of the archives, but they are either distinguished by context or we explicitly specify `archive cell`
- We use the term `augmented data/level` or `aug data/level` in the code to refer to the ancillary agent behavior data (the occupancy grid).

License

This code is currently not for public release. However, it uses the following components which are released under various licenses:

- `src/mario/Mario.jar` is adapted from the Mario AI Framework, which is released for research purposes.
- `src/mario` is adapted from the MarioGAN-LSI repo.
- `level_replay/`, `src/maze/agents/common.py`, `src/maze/agents/distributions.py`, `src/maze/agents/multigrid_network` `src/maze/agents/rl_agent.py`, and `src/maze/envs/` are adapted from the PAIRED codebase, which is released under the Apache-2.0 License.
- The pre-trained ACCEL agent in `src/maze/agents/saved_models/accel_seed_1/` was obtained directly from the original authors with their consent.
- Various infrastructure files in this repo are adapted from the dqd-rl repo, which is released under the MIT License.