

Codes README

DuRND: Dual Random Networks Distillation

The codes for our proposed **D**ual **R**andom **N**etworks **D**istillation (DuRND) algorithm. The following figure illustrates an overview of the DuRND algorithm to shape rewards:

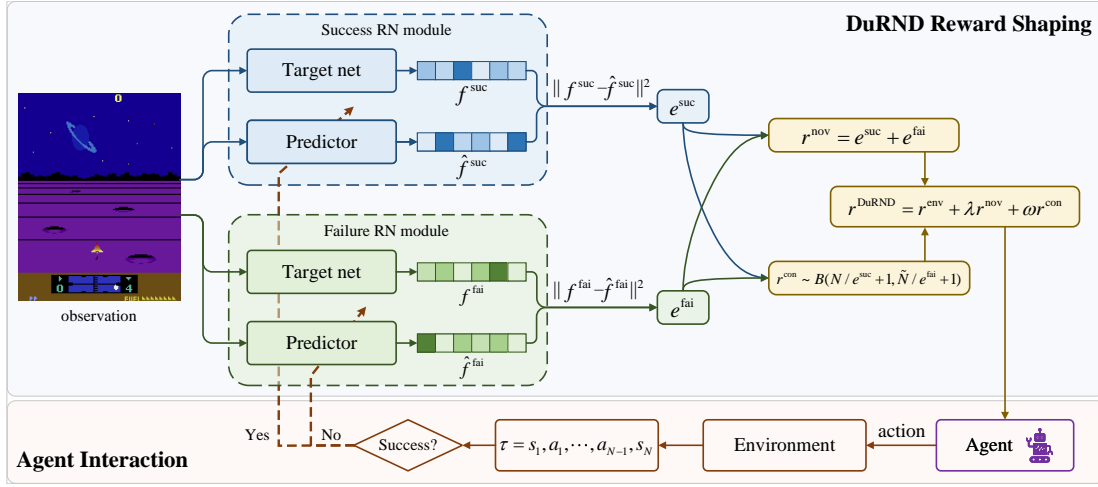


Figure 1: An overview of the Dual Random Networks Distillation (DuRND) framework. The observation is processed through both Success and Failure RN modules to derive errors that reflect its novelty in successful and failed scenarios, respectively. The two errors jointly form the DuRND shaping rewards (**novelty reward** and **contribution reward**) used to train the agent. At the end of each trajectory, the corresponding RN module is updated based on the trajectory’s outcome, as indicated by the sparse environmental reward.

1 Requirements

- The code is only supported for Python 3.6 to 3.10. (Due to the PyBullet rendering package, the code is not supported for Python higher than 3.11.)
- The code is tested on Ubuntu 20.04, with PyTorch 2.0.1 and cuda 11.7.
- All required packages can be installed using the `requirements.txt` file:

```
pip install -r requirements.txt
```

2 Run DuRND Algorithm

The DuRND algorithm is implemented in the `./DuRND/DuRND_Algo.py` file. We separately provide two variants of the DuRND algorithm:

- The original DuRND algorithm for task-completion-indication episodic rewards, e.g., *ThreeRooms* and *TMaze* environments.

```
python run-DuRND.py --env-id <ENV-ID>
```

- The DuRND algorithm for common sparse rewards, using the T_{max} to divide sub-trajectories, e.g., *Atari* games and *VizDoom* games.

```
python run-DuRND-Tmax.py --env-id <ENV-ID>
```

All tasks evaluated in the paper are listed below:

- **Atari games:** ALE/Freeway-v5, ALE/Frogger-v5, ALE/Solaris-v5, ALE/BeamRider-v5.
- **VizDoom (and LevDoom) games:** VizdoomDefendLine-v0, VizdoomDefendCenter-v0, Levdoom/HealthGatheringLevel0-v0, Levdoom/SeekAndSlayLevel0-v0.
- **MiniWorld:** MiniWorld-ThreeRooms-v0, MiniWorld-TMazeLeft-v0.

Some important hyperparameters are listed as follows (for the full list, please refer to the running scripts):

```
--exp-name: The experiment name for logging.
--env-id: The environment ID.
--seed: The random seed.
--rollout-length: the rollout length for the backbone PPO algorithm.
--num-mini-batches: the number of mini-batches for the backbone PPO algorithm.
--lr: the learning rate for the agent.
--rnd-lr: the learning rate for the random network modules.
--total-timesteps: the total training steps.
--max-sub-tra: the maximum steps for sub-trajectories, i.e.,  $T_{\max}$ , only for the DuRND
               algorithm with sub-trajectories.
```

3 Plot Experimental Results

All experimental data are saved in the `./Experiments/data.zip` file, to plot the results in our paper, you can unzip the data and run the following commands:

1. To plot the results of comparing the DuRND algorithm with the baseline algorithms:

```
python ./Experiments/comparison.py
```

2. To plot the results of the ablation study:

```
python ./Experiments/ablation-study.py
```

3. To plot the results of the novelty and contribution rewards through the training process:

```
python ./Experiments/rewards.py
```

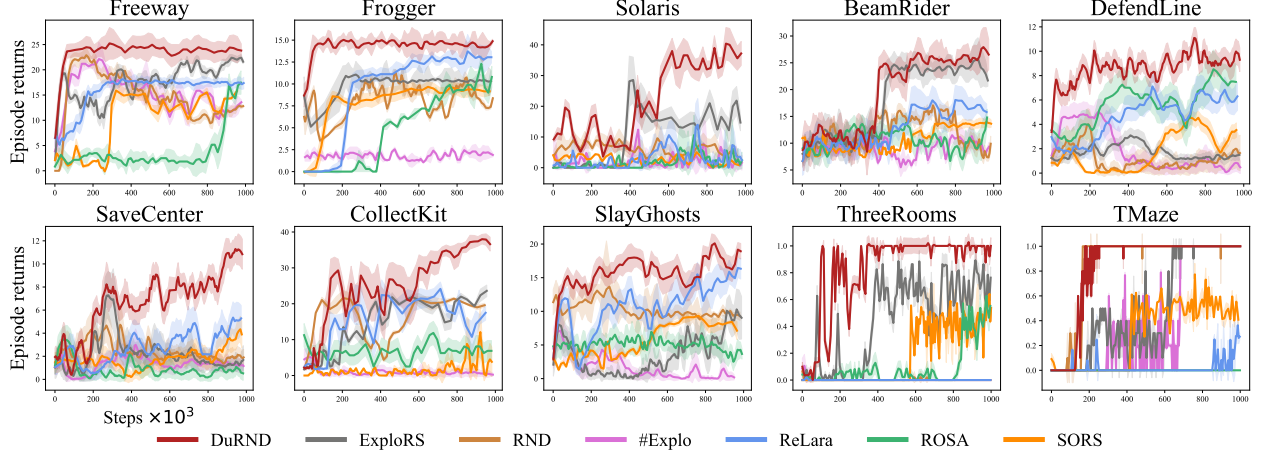


Figure 2: The learning performance of DuRND compared with baselines.

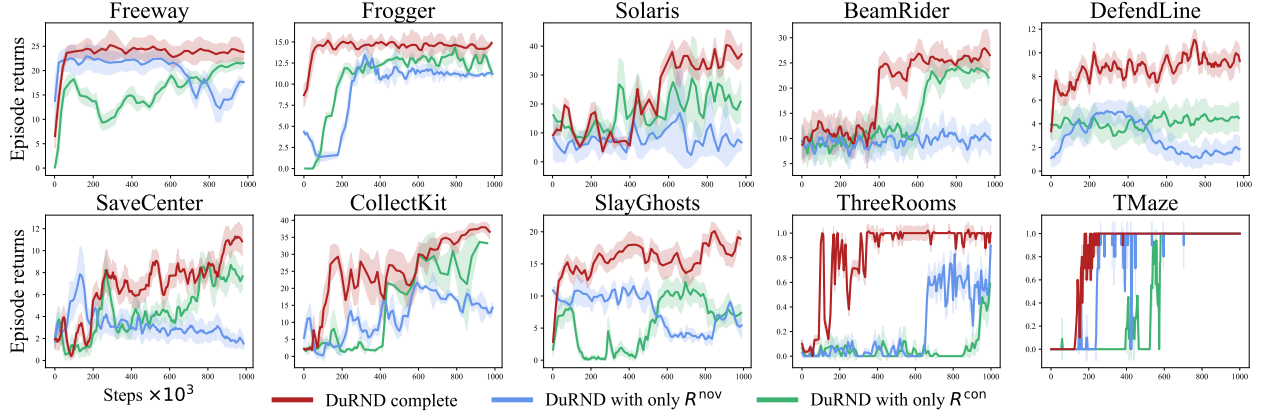


Figure 3: Ablation study: the learning performance of DuRND with a single type of reward.

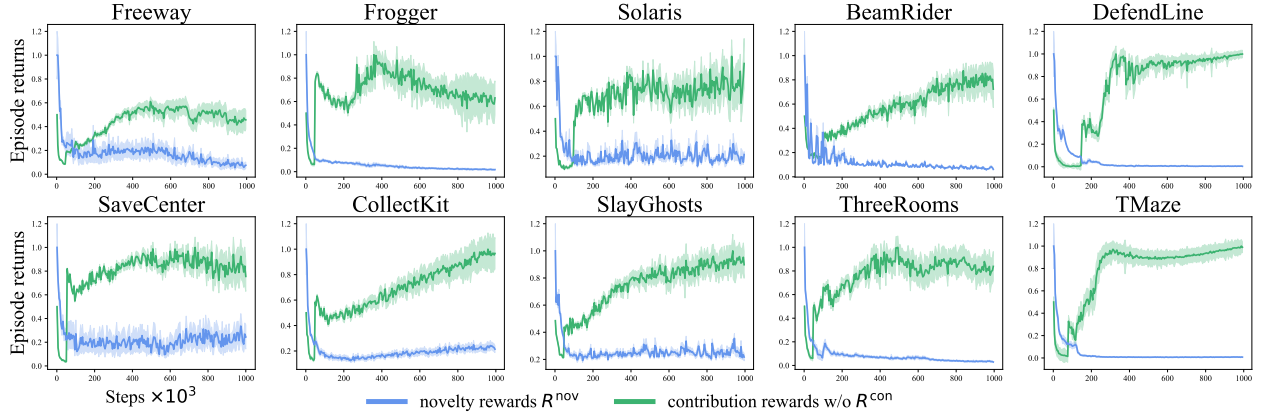


Figure 4: The novelty and contribution rewards learned in DuRND framework.