Keeping Your Distance: Solving Sparse Reward Tasks Using Self-Balancing Shaped Rewards

Alexander Trott, Stephan Zheng, Caiming Xiong & Richard Socher

Salesforce Research

NeurIPS 2019

Introduction

Solving RL tasks with sparse rewards is notoriously difficult.

Reward shaping can help agents solve sparse reward tasks, but often requires careful engineering.

We consider tasks

- in state spaces with a *distance* metric, e.g., Euclidean distance
- where task completion can be defined through goal states

It is intuitive to shape rewards using distance-to-goal, which measures progress towards success.

But this often fails due to the emergence of new *local optima*.

Sibling Rivalry: a simple and effective method to learn from distance-based shaped rewards and pairs of rollouts

SR uses a distance-based reward

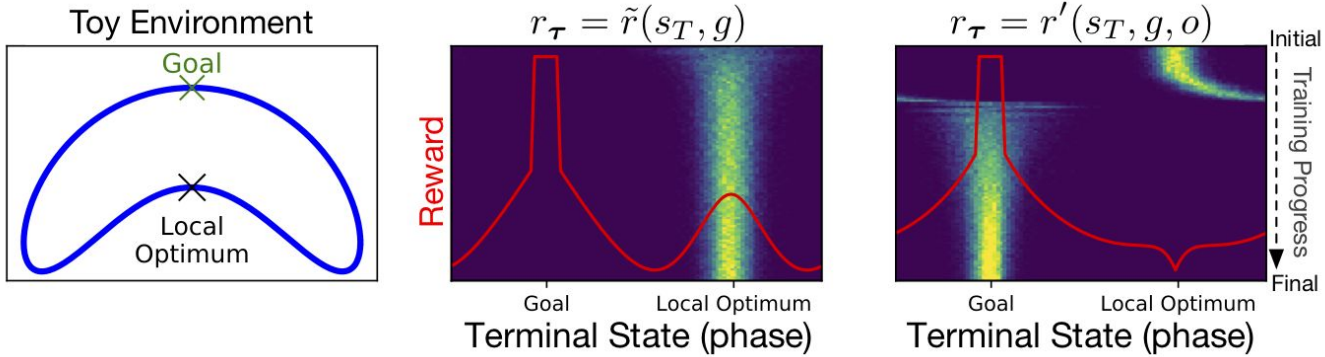
- to encourage progress towards task goals while avoiding local optima
- that converges to the original sparse reward as the agent learns to solve its task

Distance-based Rewards & Local Optima

In this toy example, the agent must navigate the warped circular track to reach the goal at the top (left panel) and receive the sparse reward. Exposing the distance-to-goal as a shaped reward fails because of the local optimum (middle panel). Augmenting the shaped reward to encourage avoiding the local optimum allows the agent to solve the task (right panel).

$$\tilde{r}(s,g) = \begin{cases} 1, & d(s,g) \leq \delta \\ -d(s,g), & \text{otherwise} \end{cases}$$

$$r'(s,g,\bar{s}) = \begin{cases} 1, & d(s,g) \leq \delta \\ \min\left[0, -d(s,g) + d(s,\bar{s})\right], & \text{otherwise} \end{cases}$$



Terminal state distributions during learning. The middle figure shows optimization using a distance-to-goal shaped reward. For the right figure, the shaped reward is augmented to include a hand-engineered distance-to-optimum bonus. The red overlay illustrates the reward at each phase of the track.

Distance-based Rewards & Local Optima

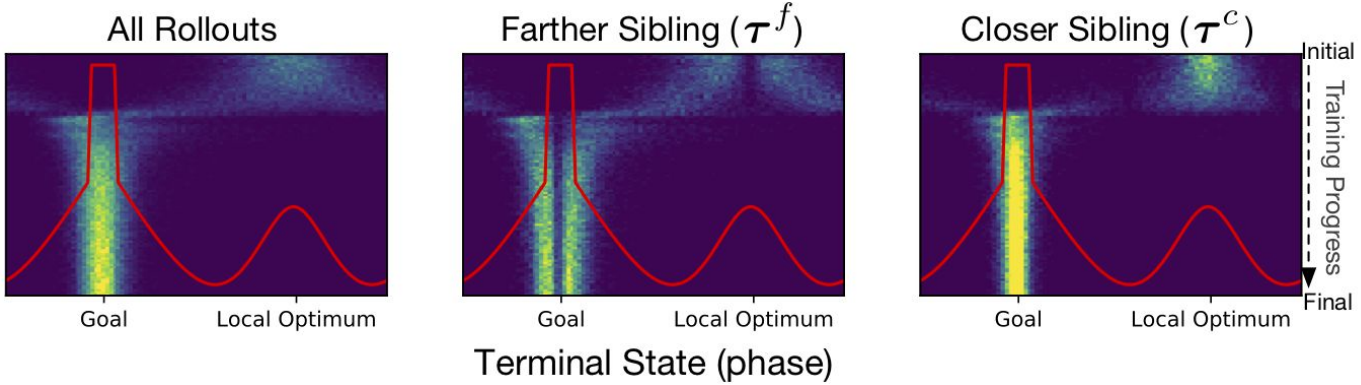
Rather than using a hand-engineered estimate of the local optima, we can use ***sibling rollouts***, independently sampled rollouts using the same policy, starting state, and goal:

$$\boldsymbol{\tau}^f, \boldsymbol{\tau}^c \sim \pi | g$$

And relabel each rollout to reward distance between the outcomes of sibling rollouts:

$$r'_{\boldsymbol{\tau}^f} = r'(s_T^f, g, s_T^c) \quad \& \quad r'_{\boldsymbol{\tau}^c} = r'(s_T^c, g, s_T^f)$$

And selectively ignore the closer-to-goal rollout when computing gradient updates.



Terminal state distributions during learning using **Sibling Rivalry**. The red overlay illustrates the reward at each phase of the track.

Sibling Rivalry

Sibling Rivalry samples pairs of *sibling rollouts* and uses the below class of shaped rewards to relabel each rollout, producing a reward bonus for sibling rollouts whose terminal states are far apart:

$$r'(s, g, \bar{s}) = \begin{cases} 1, & d(s, g) \leq \delta \\ \min\left[0, -d(s, g) + d(s, \bar{s})\right], & \text{otherwise} \end{cases}$$

This reward strategy, along with limiting the use of the closer-to-goal rollout for gradient estimation, helps prevent the policy from getting stuck in local optima created by the distance function.

As the agent learns to solve the task, the Sibling Rivalry reward converges to the original sparse reward.

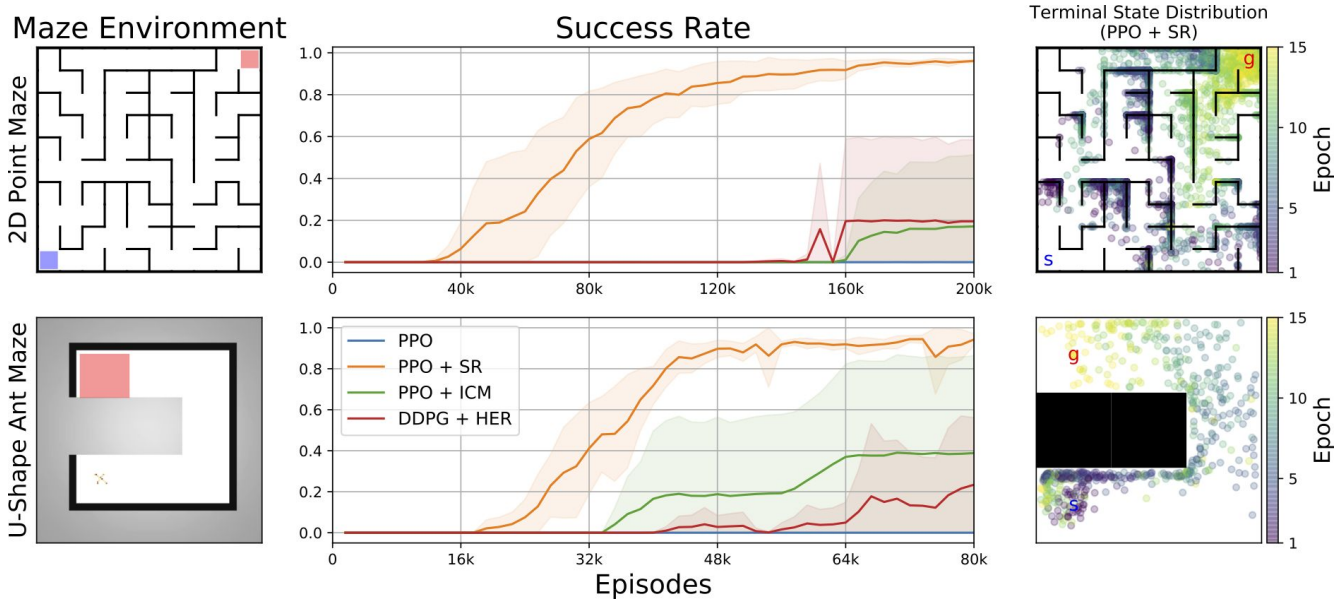
Algorithm 1: Sibling Rivalry

Given

- Environment, Goal-reaching task w/ $S, G, A, \rho(s_0, g), d(,), \delta$ and max episode length
- Policy $\pi : S \times G \times A \to [0, 1]$ and Critic $V : S \times G \times G \to \mathbb{R}$ with parameters θ
- On-policy learning algorithm $\mathbb{A}$, e.g., REINFORCE, Actor-critic, PPO
- Inclusion threshold ϵ

for *iteration = 1...K* **do**
  Initialize transition buffer D
  for *episode = 1...M* **do**
    Sample $s_0, g \sim \rho$
    $\boldsymbol{\tau}^a \leftarrow \pi_\theta(...)|_{s_0, g}$ # Collect rollout
    $\boldsymbol{\tau}^b \leftarrow \pi_\theta(...)|_{s_0, g}$ # Collect sibling rollout
    Relabel $\boldsymbol{\tau}^a$ reward using r' and $\bar{s} \leftarrow s_T^b$
    Relabel $\boldsymbol{\tau}^b$ reward using r' and $\bar{s} \leftarrow s_T^a$
    if $d(s_T^a, g) < d(s_T^b, g)$ **then**
      $\boldsymbol{\tau}^c \leftarrow \boldsymbol{\tau}^a$
      $\boldsymbol{\tau}^f \leftarrow \boldsymbol{\tau}^b$
    else
      $\boldsymbol{\tau}^c \leftarrow \boldsymbol{\tau}^b$
      $\boldsymbol{\tau}^f \leftarrow \boldsymbol{\tau}^a$
    if $d(s_T^c, s_T^f) < \epsilon$ ***or*** $d(s_T^c, g) < \delta$ **then**
      Add $\boldsymbol{\tau}^f$ and $\boldsymbol{\tau}^c$ to buffer D
    else
      Add $\boldsymbol{\tau}^f$ to buffer D
  Apply on-policy algorithm $\mathbb{A}$ to update θ using examples in D

Continuous Navigation Tasks

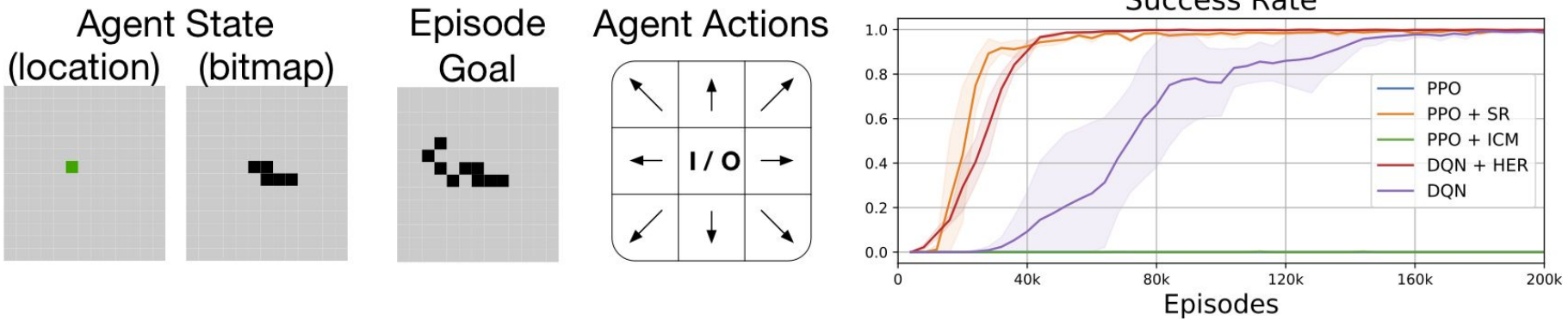


(Left) Maze environments. Top row illustrates our 2D point maze; bottom row shows the U-shaped Ant Maze in a Mujoco environment. Start/goal locations are sampled uniformly from the blue/red square regions, respectively. In the ant maze, the agent starts near its pictured location.

(Middle) Learning progress. Lines show rate of goal completion averaged over 5 experiments (shaded area shows mean ± SD, clipped to [0, 1]). Only our method (PPO+SR) allows the agent to discover the goal in all experiments. Conversely, PPO with the naive distance-to-goal reward never succeeds. Methods to learn from sparse rewards (PPO+ICM [Intrinsic Curiosity Module] and DDPG+HER [Hindsight Experience Replay]) only rarely discover the goals.

(Right) State distributions sampled over the course of training. Colored points illustrate terminal states achieved by the policy after each of the first 15 evaluation checkpoints. PPO+SR allows the agent to discover increasingly good optima without becoming stuck in them.

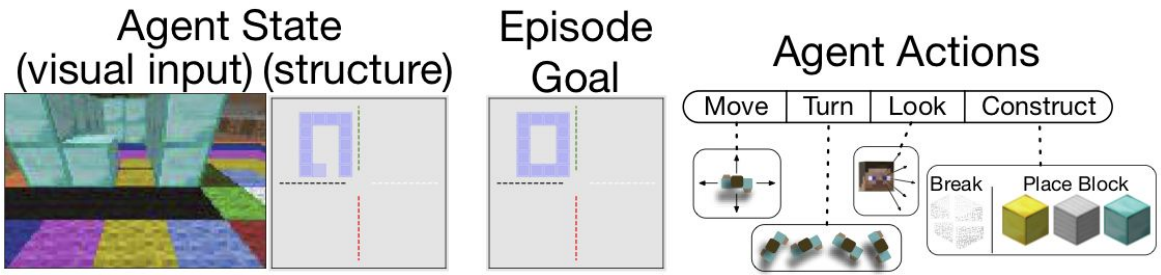
Discrete 2D Bit-Flipping Task

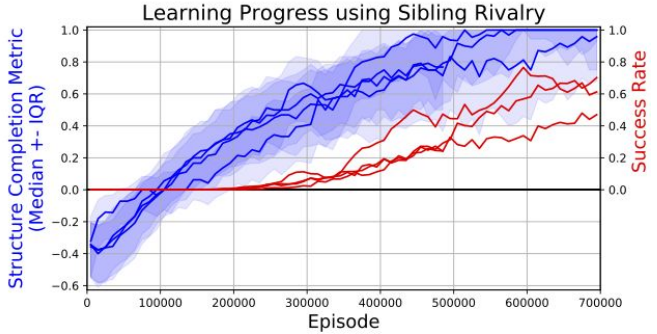


(Left panels) The agent begins in a random location on a 13x13 grid with all pixels off and must move and toggle pixels to produce the goal bitmap. The agent sees its current location (1-hot), the current bitmap, and the goal bitmap. The left 3 panels show an example of what the agent might observe as it completes the example goal shown. The agent succeeds when the bitmap exactly matches the goal (0-distance).

(Right) Learning progress. Lines show rate of goal completion averaged over 5 experiments (shaded area shows mean ± SD, clipped to [0, 1]). Using a naive distance-to-goal shaped reward (PPO), the agent learns to avoid toggling the bitmap and never makes progress. Sibling Rivalry (PPO+SR) avoids this outcome, allowing the agent to discover useful behavior and eventually solve the task. HER exhibits comparable performance, but ICM fails because it does not distinguish between task-useful and task-irrelevant exploration.

3D Construction in a Minecraft Environment





(Left panels) The agent must control its location/orientation and break/place blocks in order to produce the goal structure. The agent observes its first-person visual input, the discrete 3D cuboid of the construction arena (representing the structure it has built so far), and the corresponding cuboid of the goal structure. The agent succeeds when the structure it has built exactly matches the goal structure (0-distance). In the illustrated example, the agent has nearly constructed the goal, which specifies a height-2 diamond structure near the top left of the construction arena. Goal structures vary in height, dimensions, and material (4806 unique combinations).

(Right) Learning progress. The Structure Completion Metric (shown in blue) is the difference between the number of correctly and incorrectly placed blocks divided by the number of blocks in the goal structure. The Success Rate (shown in red) measures how often the agent completes the goal structure perfectly. The plot shows an overlay of the training progress of 3 experiments using Sibling Rivalry (using IMPALA as the backbone algorithm). Using a naive distance-to-goal shaped reward leads the agent to stop placing any blocks within ~1000 episodes (not pictured).

Conclusion

- We introduce **Sibling Rivalry**, a simple and effective method for learning sparse reward goal-reaching tasks from a generic class of distance-based shaped rewards.
- Sibling Rivalry makes use of *sibling rollouts* and self-balancing rewards to prevent the learning dynamics from stabilizing around local optima.
- By leveraging the distance metric used to define the underlying sparse reward, our technique enables robust learning from shaped rewards without relying on carefully-designed, problem-specific reward functions.
- Our experiments show that Sibling Rivalry can be readily applied to both continuous and discrete domains, incorporated into hierarchical RL, and scaled to demanding environments.
- Additional details, experiments, analysis, and discussion are available in the full paper!