Learning Rewards for Long Horizon Rearrangement

Lucas Chen

Intro

We teach a robot to **rearrange** randomly scattered objects into a goal arrangement.

- We **learn a reward function** from pure Brownian noise and use it to guide policy learning.
- Brownian noise contains information about the *gradient* of positions that lead to the goal state.

We (attempt to) show that such a reward can meaningfully guide policy learning to solve simple and complex pusher environments.

Method

The reward function **estimates the current timestep** for a given configuration of cubes. The reward signal is then derived as the change in this estimate between consecutive states.

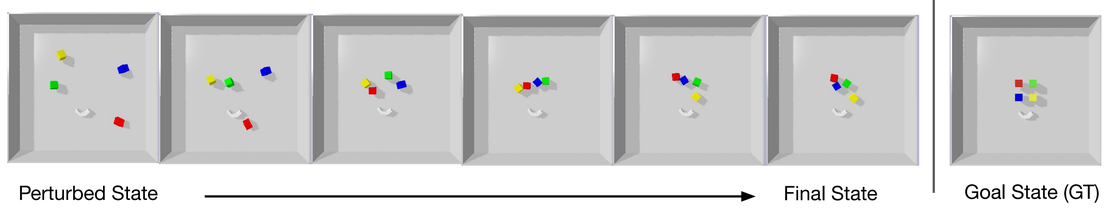
- Predicting the timestep forces the network to learn the route to the goal, even if there are walls or obstacles in the way.

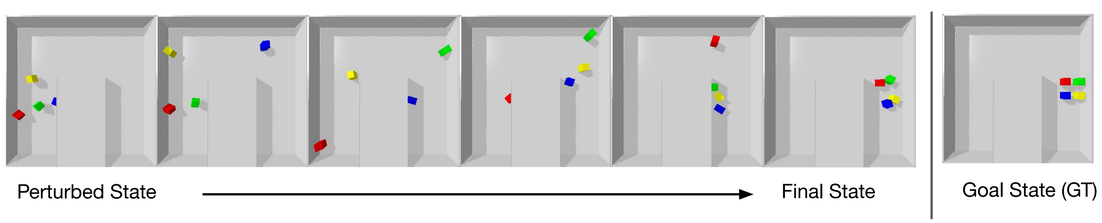
Why we cannot predict the gradient directly:
Brownian walks follow the Markov Property — for any current states, all possible next states are equally likely. As a result, directly predicting the gradient between two states is equivalent to predicting noise.

An issue with learning from random walks is that the chance a walk will travel far from the goal area is small. This causes a **sample imbalance** where most samples are clustered around the center. In this work, we use some ad-hoc methods to restart walks at states that need more samples (and this works due to Markovian Property). The proper way to do it requires estimating the distribution of the walks, which is difficult!

Experiments

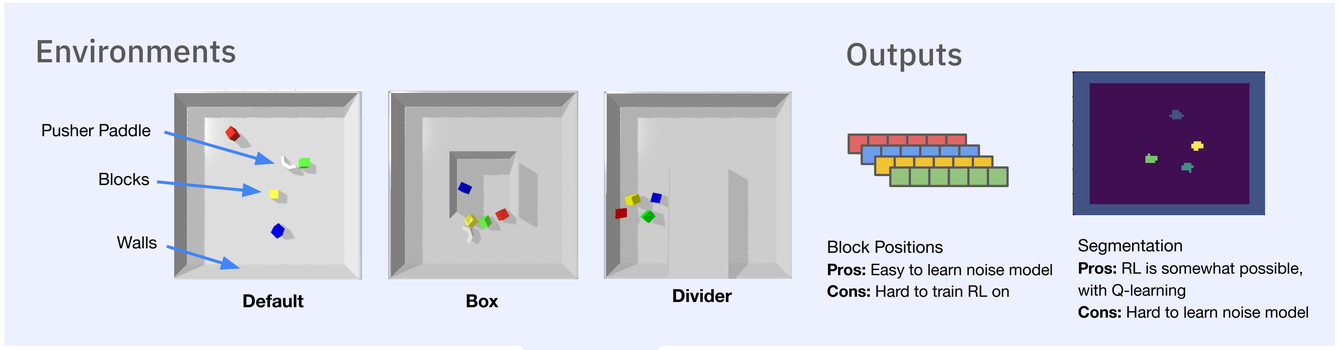
Visualizing the paths stored in the reward function. Here we apply an omniscient force on each cube according to the gradient of the reward function (the version that takes positions as inputs)

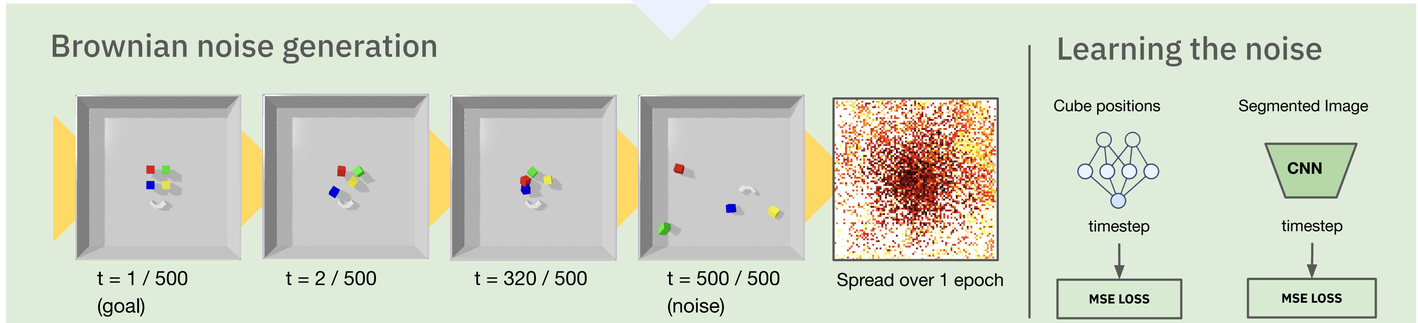




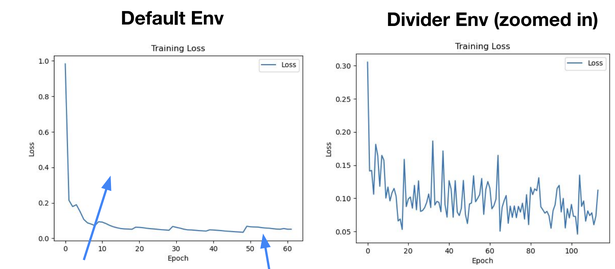
Results

No RL training results, yet :(





Noise Model Training Loss



Don't be fooled! Most of the learning happens in the valley

Sawtooth pattern caused by sample imbalances (despite our best efforts)

This env is much harder to learn since the extended walk lengths lead to more noise

RL Training

When the paddle could be far away from the desired cube, it could take a lot of actions to make any change to the reward function. So the action space was paired with a local controller to span these long gaps in a few actions.

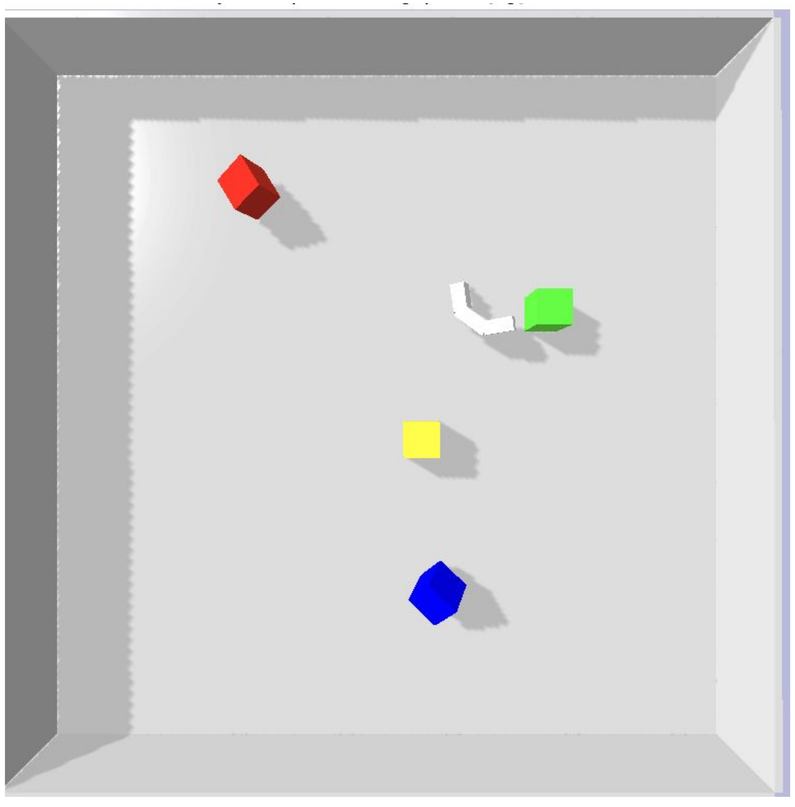
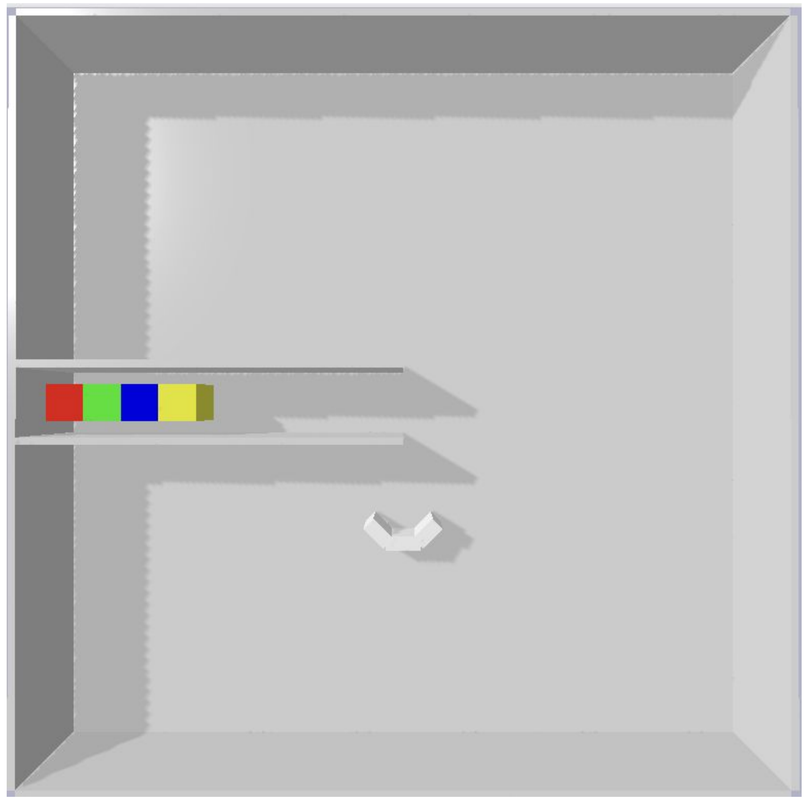
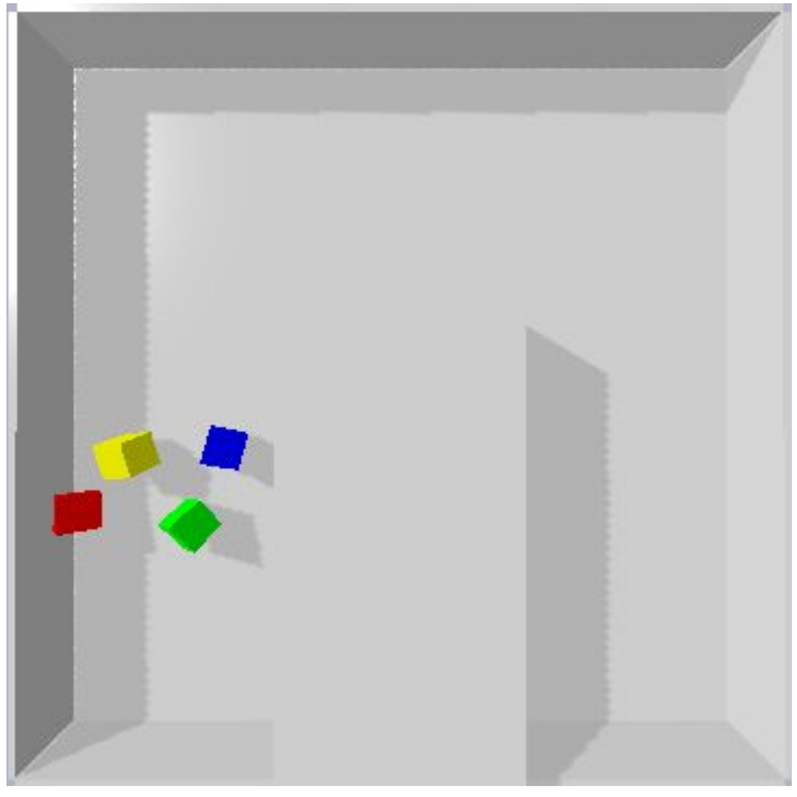
Action Space X and Y coordinates where to send the paddle to

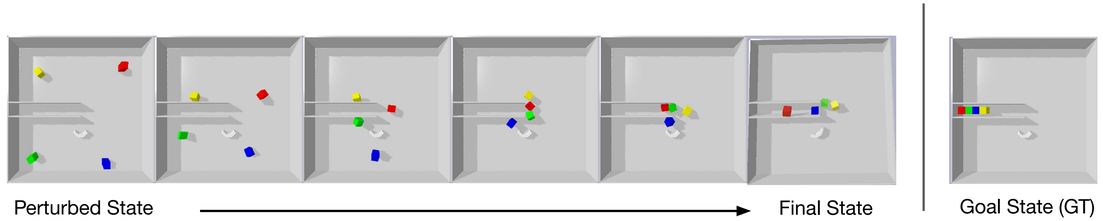
The **Reward** is set as

$$\begin{cases} \hat{t}_{k-1} - \hat{t}_k - \lambda, & \text{if not reached} \\ 1, & \text{when reached (termination)} \end{cases}$$

and λ is a "hurry up" factor to to incentivize the policy to finish sooner, usually set to somewhere in [0.01, 0.05]

Algorithm We can use Double DQN to predict a probability map of whether to go to each pixel. This allows us to avoid learning a regression task, which is quite hard given the large size of our environment.


Perturbed State
Final State
Goal State (GT)