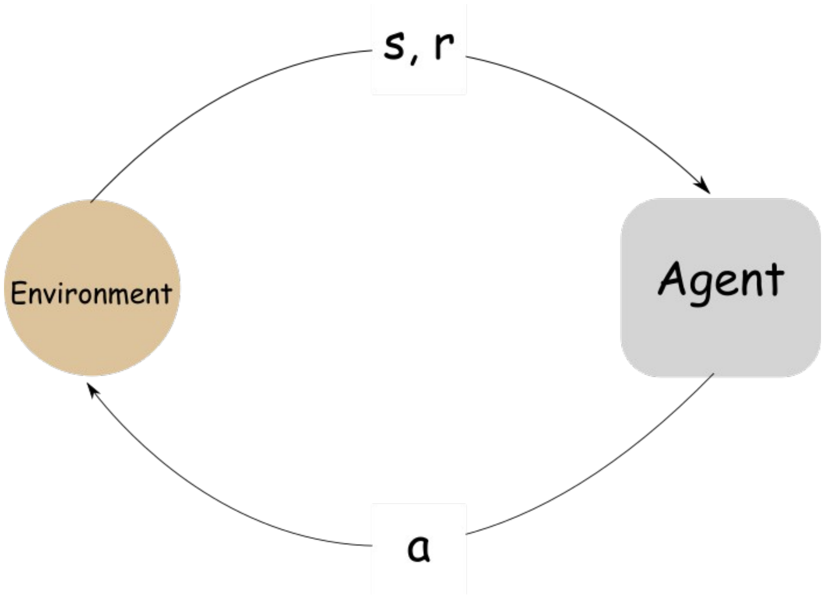
Learning Policies for Model-Based Reinforcement Learning using Distributed Reward Formulation

Presenter - Nikhil Agarwal

Committee:
Dr. Heni Ben Amor (Chair)
Dr. Mariano Phielipp (Member)
Dr. Hemanth DV (Member)

Arizona State University

Reinforcement Learning



States $S \in \mathbb{R}^{d_s}$

Actions $A \in \mathbb{R}^{d_a}$

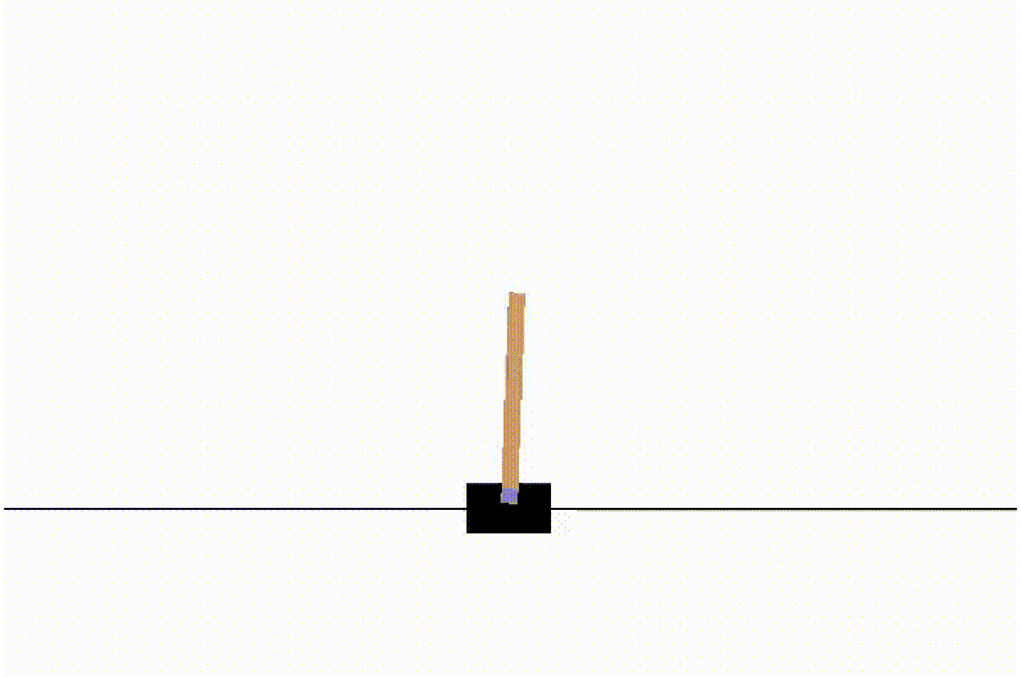
Reward Function $R : S \times A \to \mathbb{R}$

Transition Function $T : S \times A \to S$

Discount $\gamma \in (0, 1)$

Policy $\pi : S \to A$

Example – RL Env



OpenAI – Gym – CartPole Env

State:	Pole angle, dist. from center
Action:	+1 (left), -1 (right)
Reward:	+1, if pole is upright
Termination:	Pole angle > 15 from vertical

Source: gym.openai.com/envs/CartPole-v1/

Outline

1. C-51

   Distributional RL algorithm

2. PDDM

   Model-Based RL algorithm

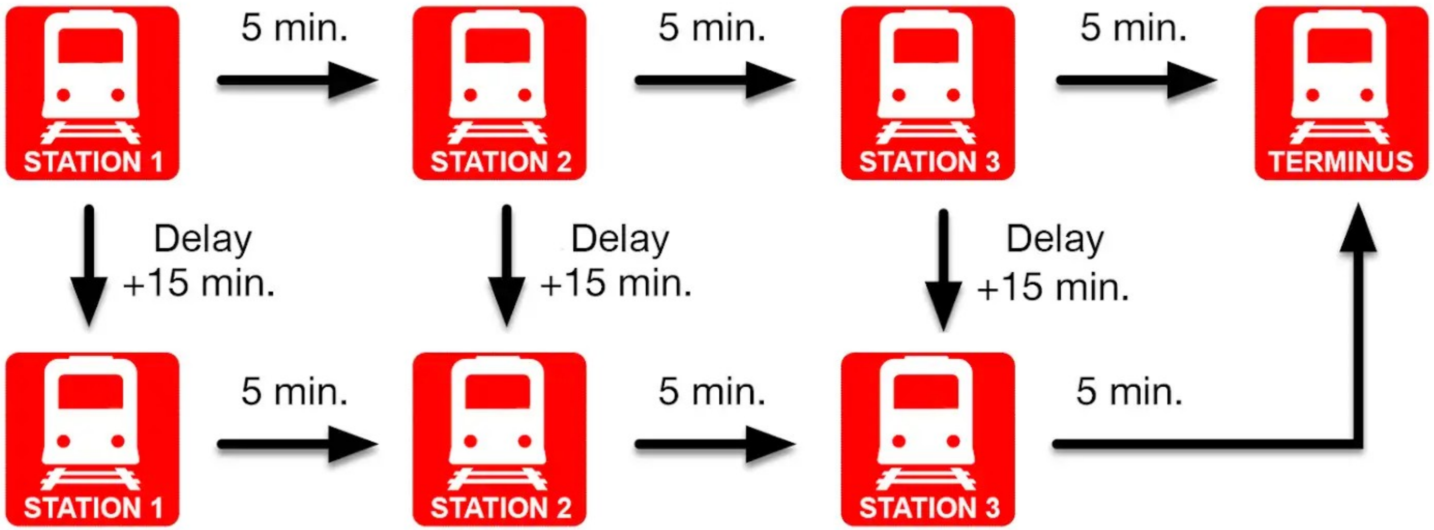
3. PDDM + C-51

   Model-Based Distributional RL algorithm

C-51: Distributional RL algorithm

Bellemare, Marc G., Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." International Conference on Machine Learning. PMLR, 2017.

Understanding Distributional RL



Avg commute time: 3*5 + 15/5 = 18 mins

Actual commute times: 15 mins to 30 mins

Image Source: deepmind.com/blog/article/going-beyond-average-reinforcement-learning

Bellman Equation

$$Q^{\pi}(s,a) = \mathbb{E}R(s,a) + \gamma\mathbb{E}Q^{\pi}(s^{'},a^{'})$$

Classic bellman equation

Reward for reaching state s

Discounted reward from state s` to goal

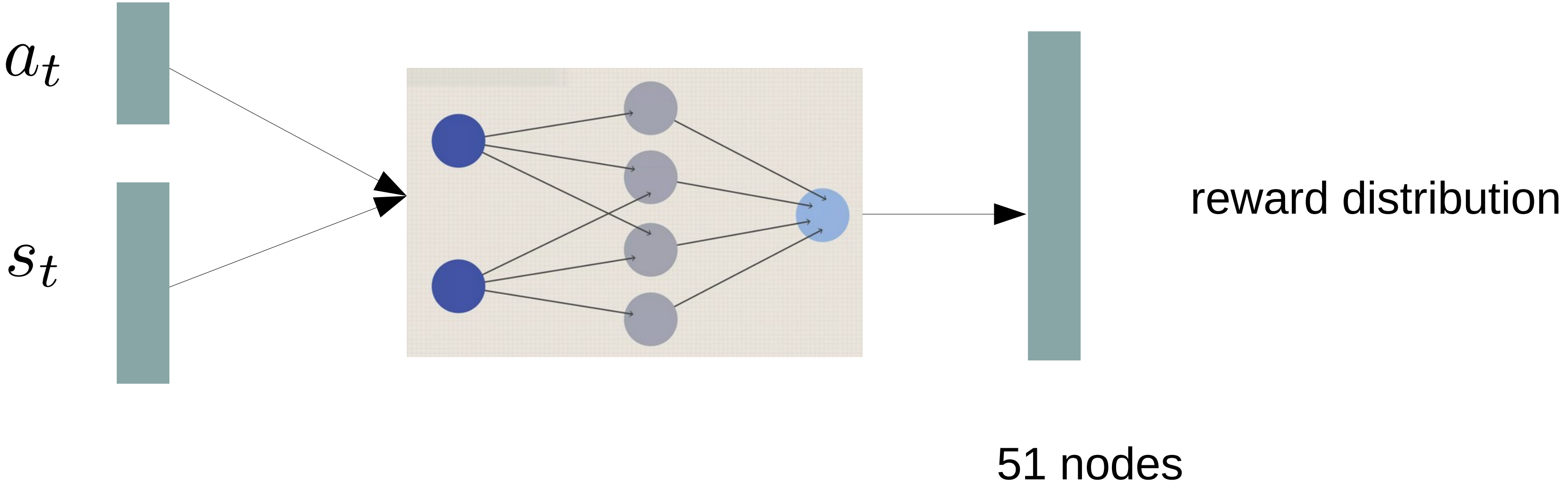
$$Z^{\pi}(s,a) = \mathbb{E}R(s,a) + \gamma\mathbb{E}Z^{\pi}(s^{'},a^{'})$$

Distributed bellman equation

Discounted reward distribution from state s` to goal

$$Q^{\pi}(s,a) := \mathbb{E}Z^{\pi}(s,a)$$

Learning Distributional Reward Representation



Parametric Distribution

$$Z_\theta(s,a) = z_i$$

Reward distribution for each node

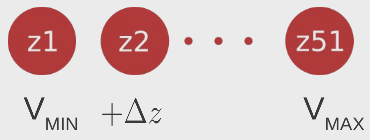
w.p. $$p_i(s,a) := \frac{e^{\theta_i(s,a)}}{\sum_j e^{\theta_j(s,a)}}$$

$$\Delta z = \frac{V_{MAX} - V_{MIN}}{N-1}$$

$$0 \leq i < 51$$

$$z_i = V_{MIN} + i\Delta Z$$

Histogram bucket sizes



Projected Bellman Update

$$\hat{\tau} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{MIN}}^{V_{MAX}}$$

Bellman Equation

Limits of our distribution

C-51 Algorithm

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$

$Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$

$a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$

$m_i = 0, \quad i \in 0, \ldots, N-1$

for $j \in 0, \ldots, N-1$ **do**

Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$

$\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$

$b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$ # $b_j \in [0, N-1]$

$l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$

Distribute probability of $\hat{\mathcal{T}} z_j$

$m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$

$m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$

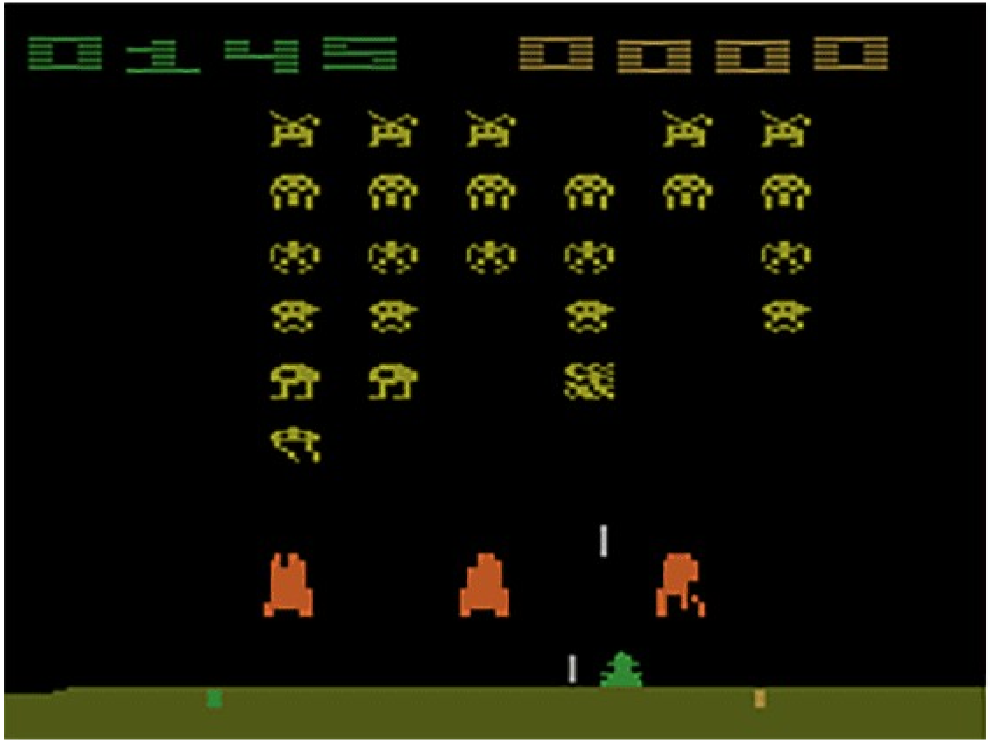
end for

output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

- For discrete action-spaces only

Bellemare, Marc G., Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." International Conference on Machine Learning. PMLR, 2017.

Example – Distributional RL



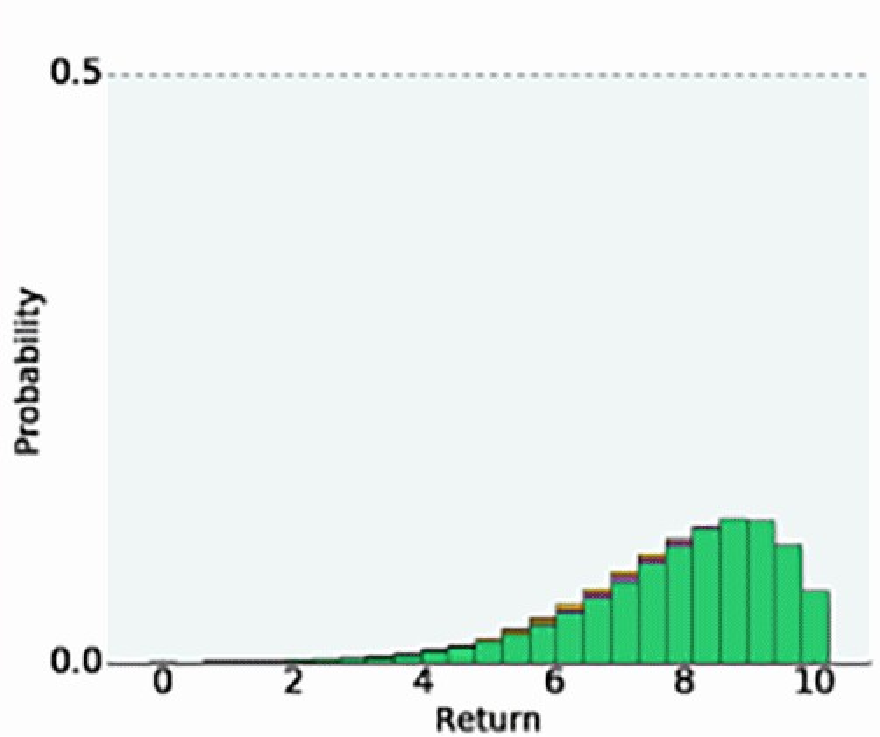
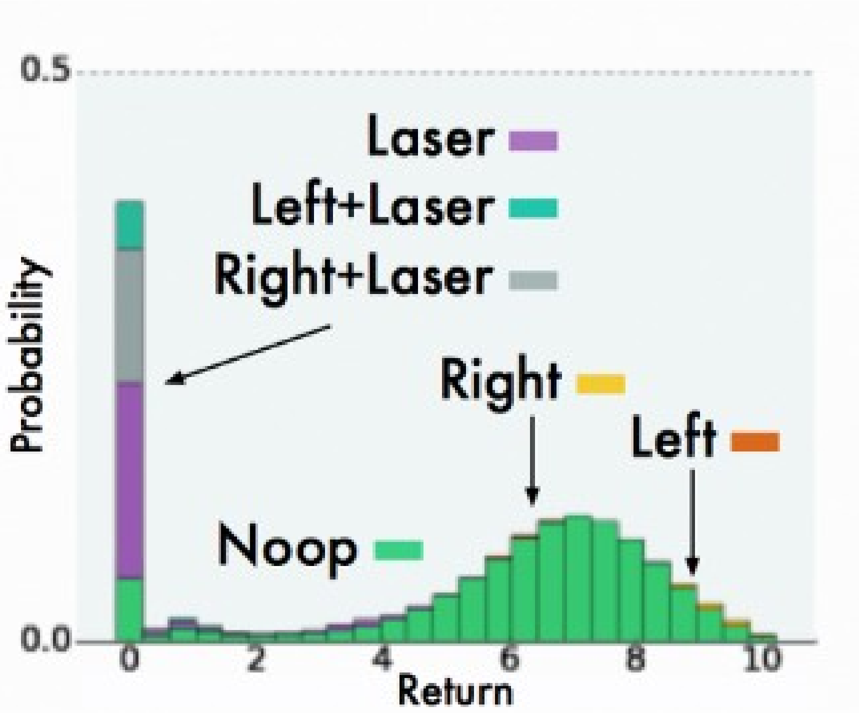


Image Source: deepmind.com/blog/article/going-beyond-average-reinforcement-learning

Example – Distributional RL



Benefits of distributional reward formulation

- More stable learning agent
- Richer set of predictions
- Reduces Chattering

Chattering: When a policy converge to a region where it oscillates indefinitely

PDDM: Planning with Deep Dynamics Models

Nagabandi, Anusha, et al. "Deep dynamics models for learning dexterous manipulation." Conference on Robot Learning. 2020.

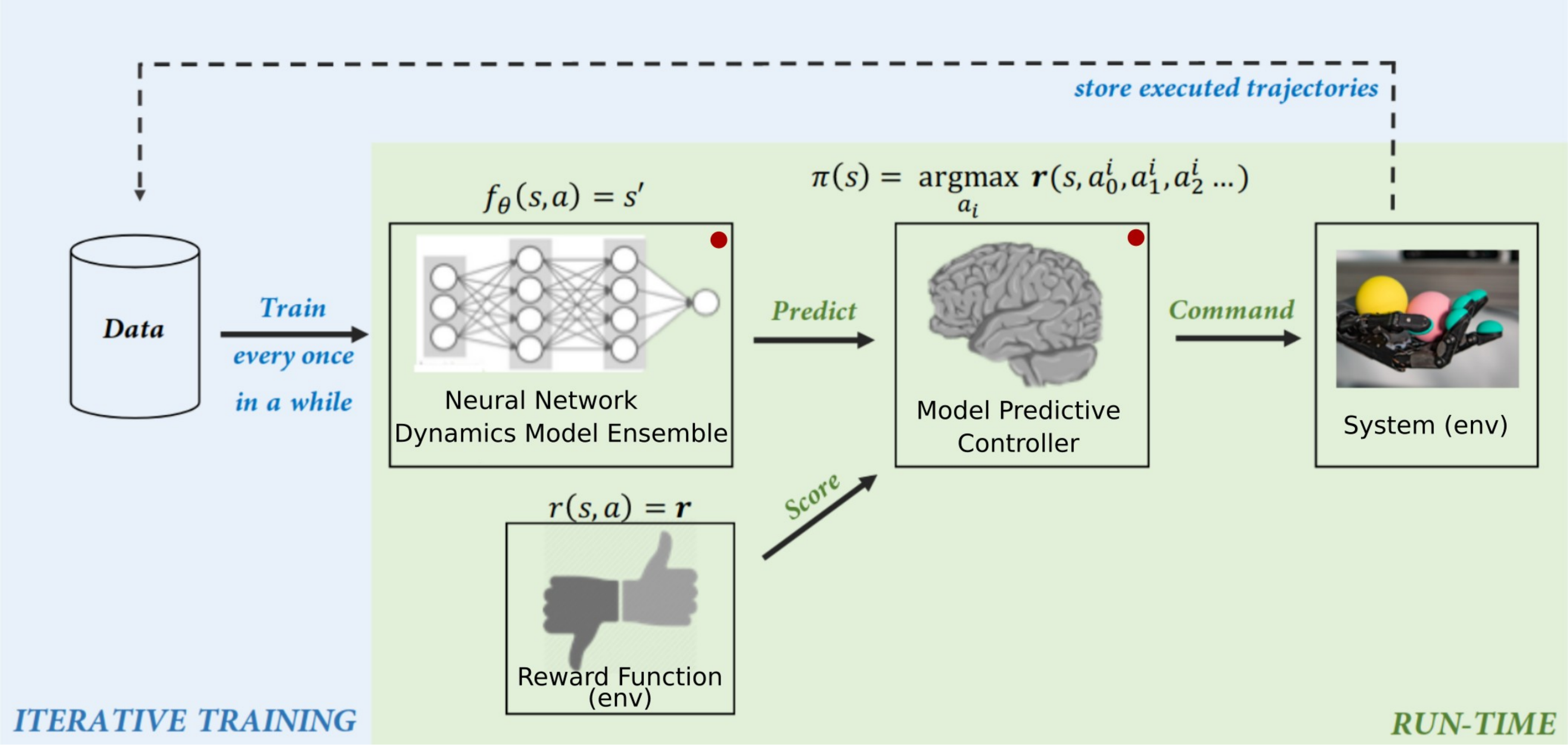
Types of Learned Models

- A transition/dynamics model: $s_{t+1} = f_s(s_t, a_t)$
- A model of rewards: $r_{t+1} = f_r(s_t, a_t)$
- An inverse transition/dynamics model: $a_t = f_s^{-1}(s_t, s_{t+1})$
- A model of distance: $d_{ij} = f_d(s_i, s_j)$
- A model of future returns: $G_t = V(s_t)$

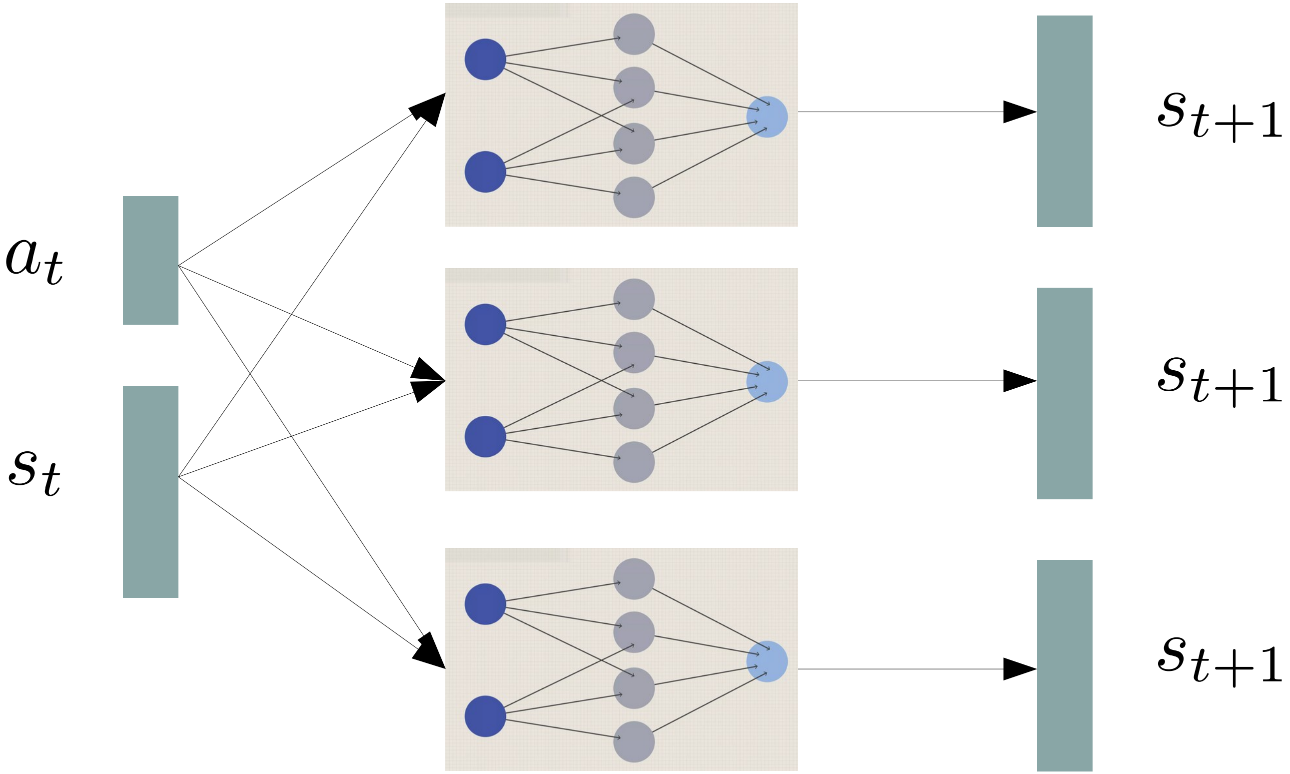
What does PDDM learn ?

- A transition/dynamics model: $s_{t+1} = f_s(s_t, a_t)$ (learns)
- A model of rewards: $r_{t+1} = f_r(s_t, a_t)$ (assumes knowledge)
- An inverse transition/dynamics model: $a_t = f_s^{-1}(s_t, s_{t+1})$
- A model of distance: $d_{ij} = f_d(s_i, s_j)$
- A model of future returns: $G_t = V(s_t)$

PDDM: Model Overview

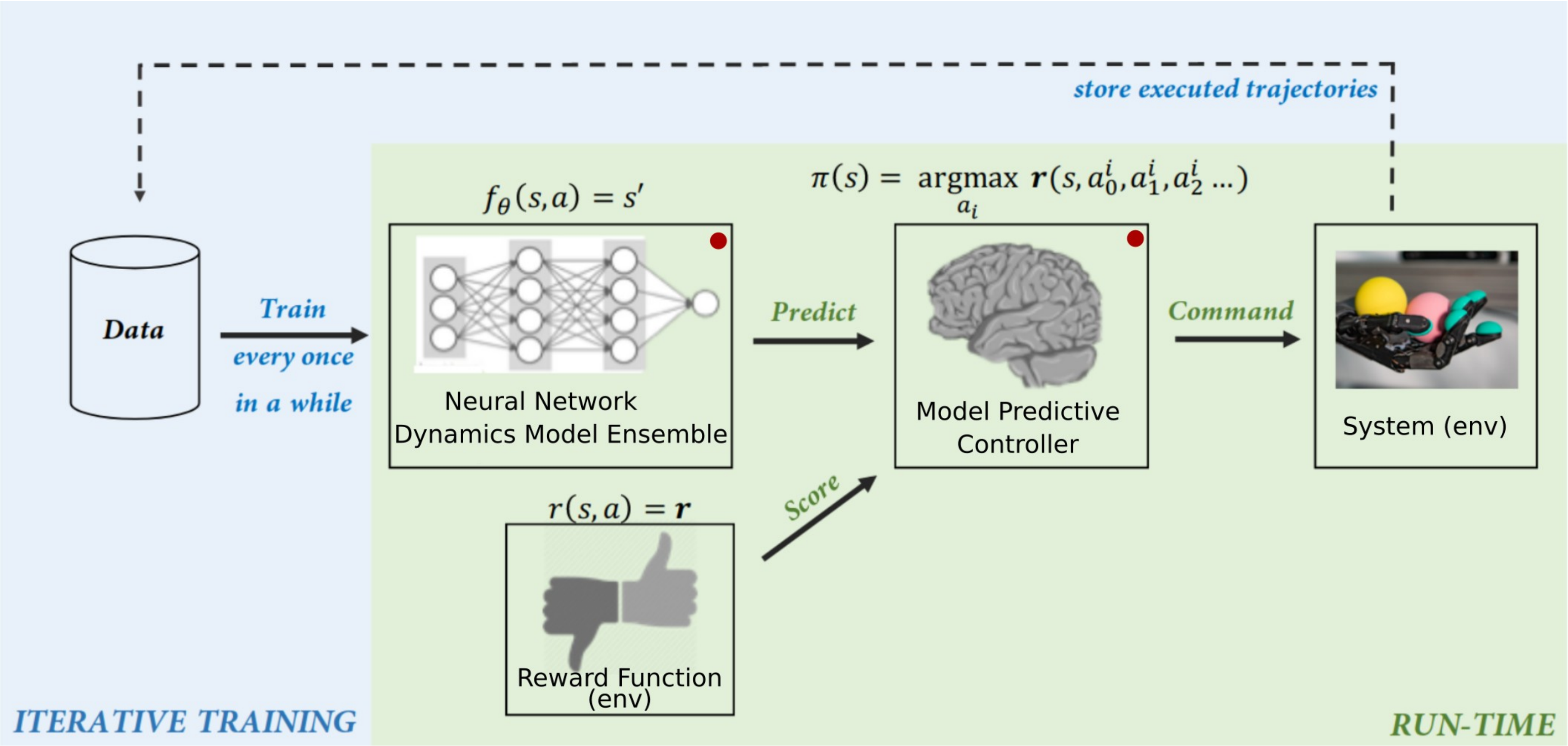


Learning State-Transition Model



- Ensemble of 3 NN models

PDDM: Model Overview



Policy Learning (Controller)

Gradient Free Optimization

- Tries to learn mean of the action distribution

$$\mu_t = \frac{\sum_{k=0}^{N} (e^{\gamma \cdot R_k})(a_t^k)}{\sum_{j=0}^{N} e^{\gamma \cdot R_j}} \forall t \in \{0..H - 1\}$$

Mean Action Update

N = Number of trajectories

R = Reward for that trajectory

H = Number of horizons per trajectory

Policy Learning (Controller)

Action Sampling and Smoothing

$$u_t^i \sim \mathcal{N}(0, \Sigma) \forall i \in \{0..N-1\}, t \in \{0..H-1\}$$

Sampling Gaussian Noise

$$n_t^i = \beta . u_t^i + (1-\beta) . n_{t-1}^i \qquad n_{t<0} = 0$$

Applies smoothing and filtering

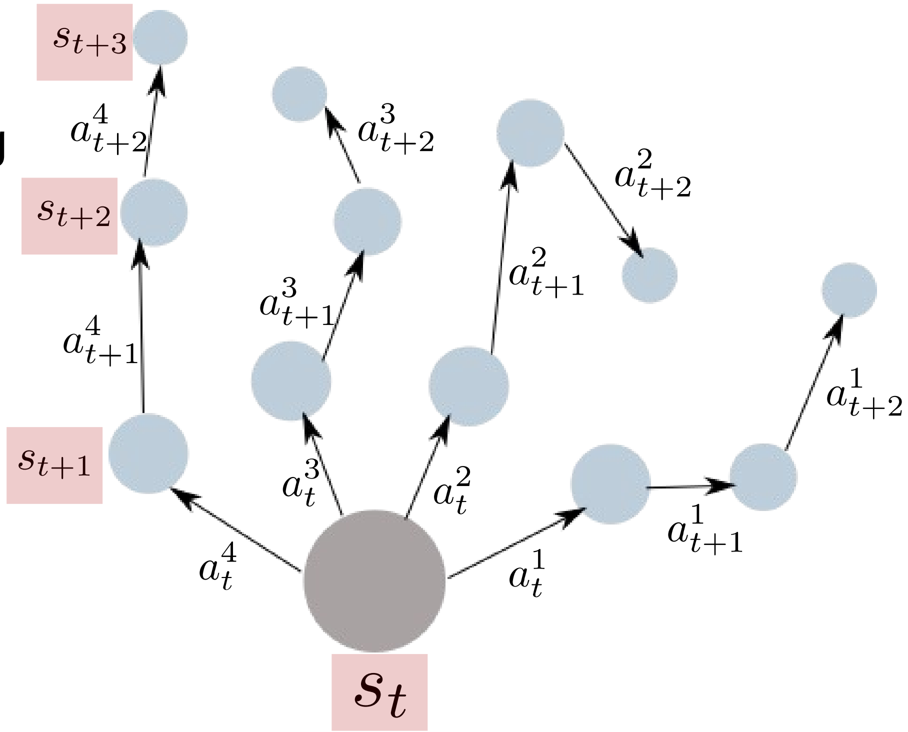
$$a_t^i = n_t^i + \mu_t$$

Action Sampling

Policy Learning (Controller)

Gradient Free Closed-Loop Planning

- Performs short-horizon rollouts (H=10) using learned model
- Employs gradient-free optimization to select best action at each step
- Chooses the trajectory with highest cumulative reward



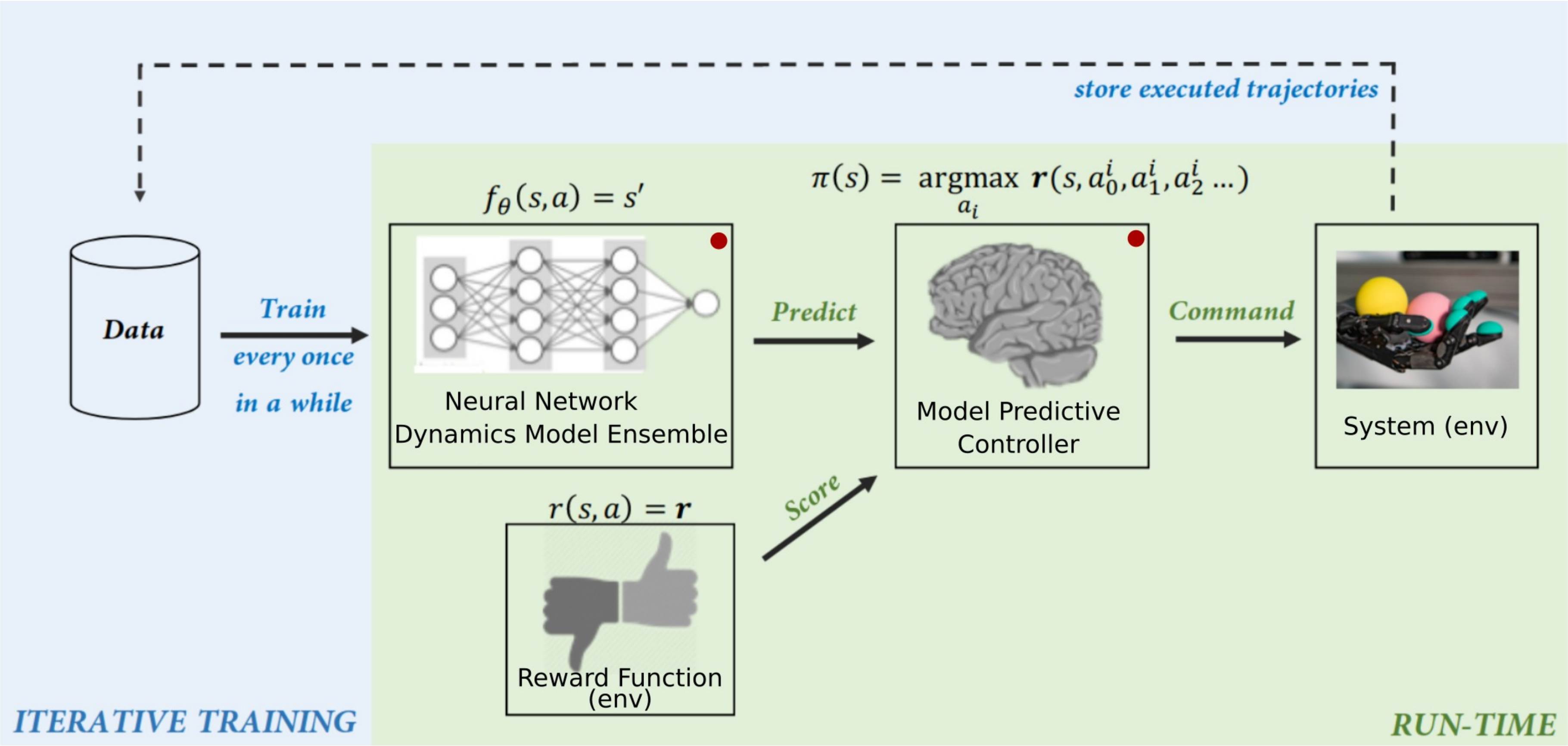
PDDM Algorithm

Algorithm 1 PDDM Overview

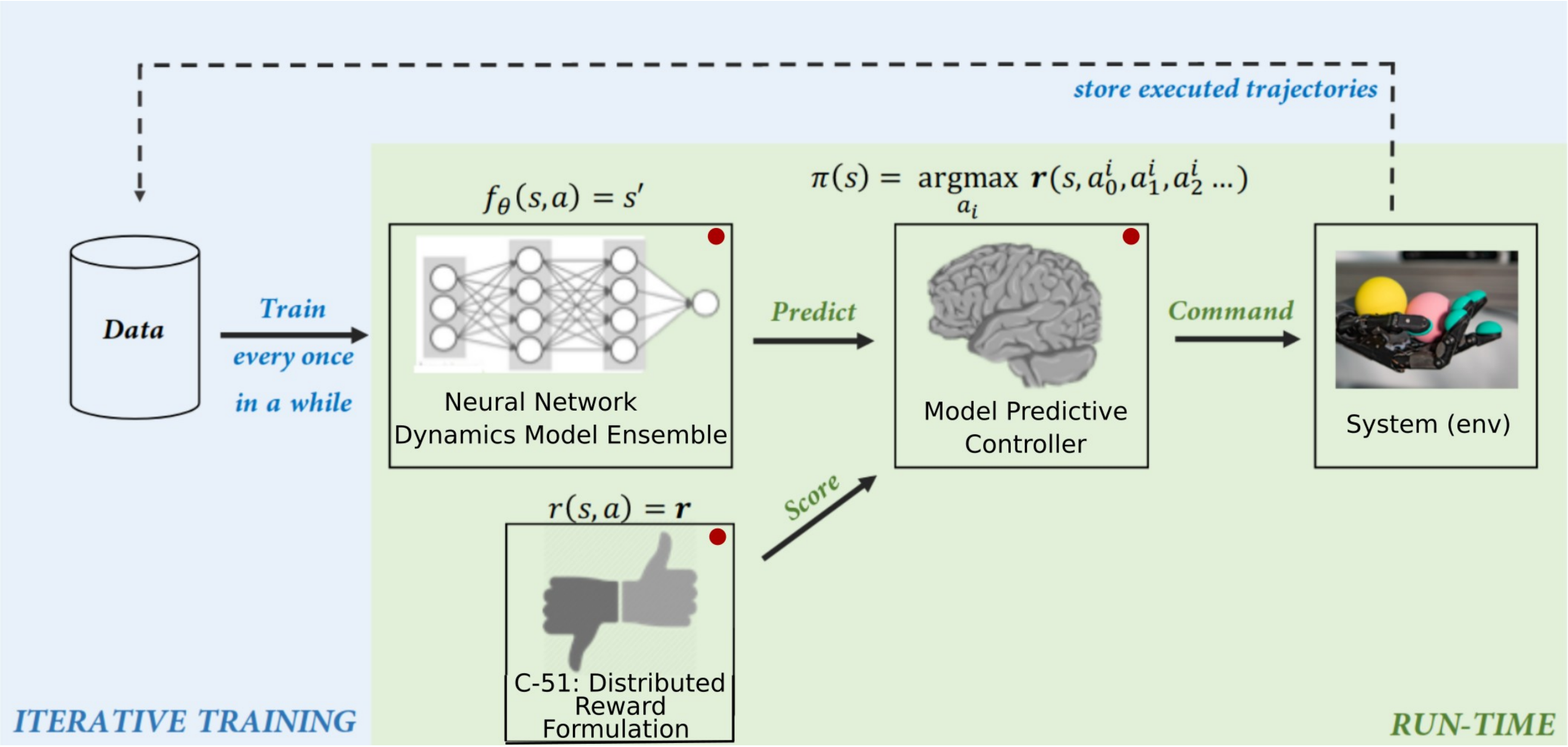
randomly initialize ensemble of models $\{\theta_0, \ldots, \theta_M\}$
initialize empty dataset $D \leftarrow \{\}$
for iter in range(I) **do**
    for rollout in range(R) **do**
        $s_0 \leftarrow$ reset env
        for t in range(T) **do**
            $a \leftarrow \text{PDDM}_{\text{MPC}}(s_t, \{f_{\theta_0}, \ldots, f_{\theta_M}\}, H, N, r, \gamma, \beta)$
            $s_{t+1} \leftarrow$ take action a
            $D \leftarrow (s_t, a_t, s_{t+1})$
        end for
    end for
    use D to train models $\{f_{\theta_0}, \ldots, f_{\theta_M}\}$ for E epochs each
end for

Combining PDDM with C-51

PDDM: Model Overview



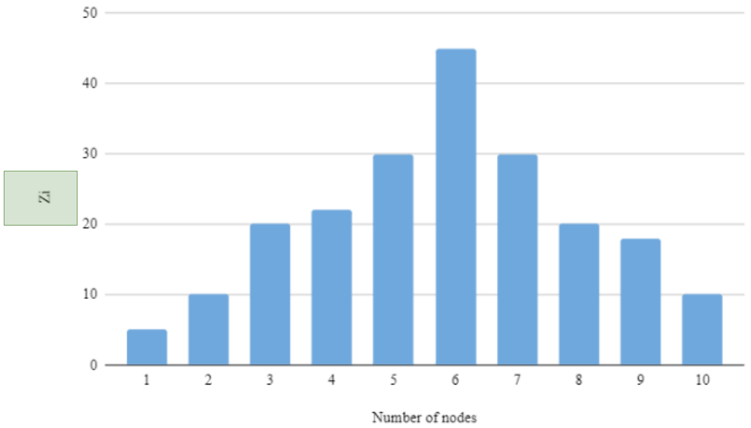
PDDM + C-51: Model Overview



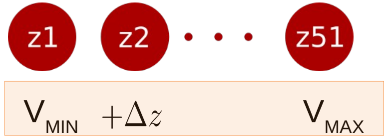
Updated Controller

$$\mu_t = \frac{\sum_{k=0}^{N} (e^{\gamma \cdot R_k})(a_t^k)}{\sum_{j=0}^{N} e^{\gamma \cdot R_j}} \forall t \in \{0..H-1\}$$

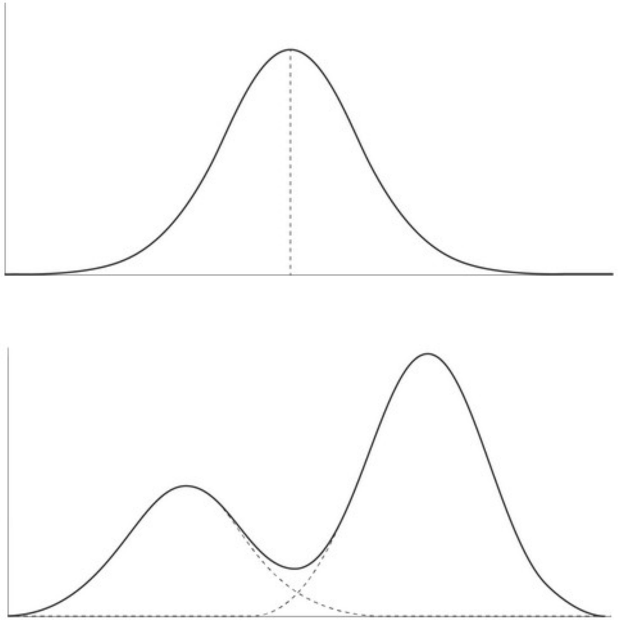
Mean Action Update



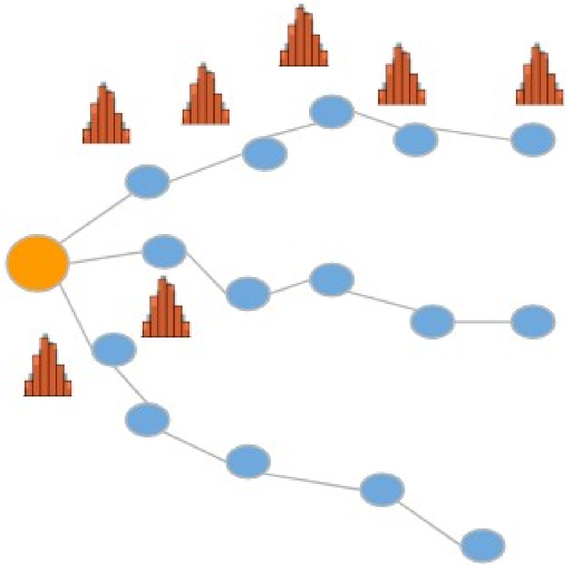
$$R_k = sum(weight\_value\_node\_i \times Z_i)$$



Benefits



Distributions with same estimates no longer similar



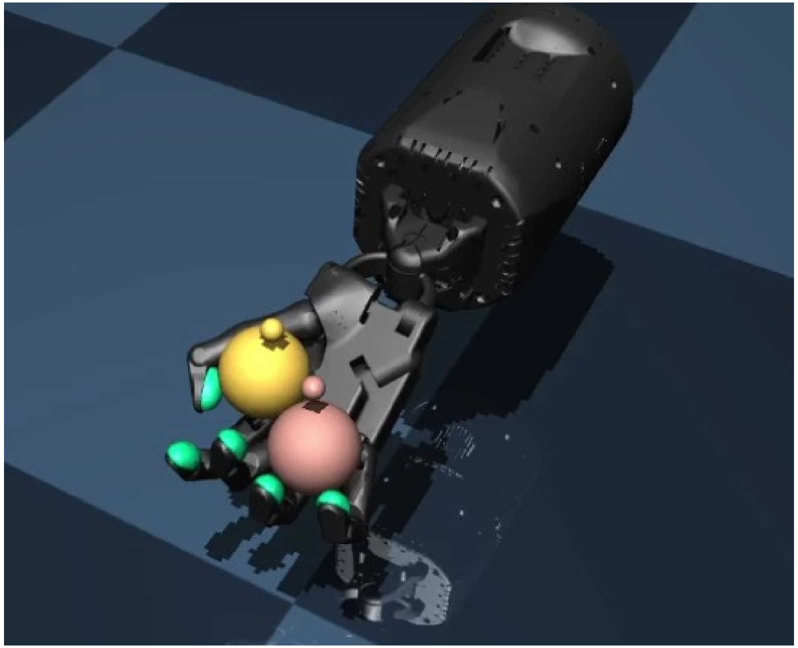
Learner gets access to both - future states and reward distributions

Expectations

- Enables learning in stochastic environments
- Chosen actions are more risk-averse
- Execute episodes with longer rollouts

Experiments

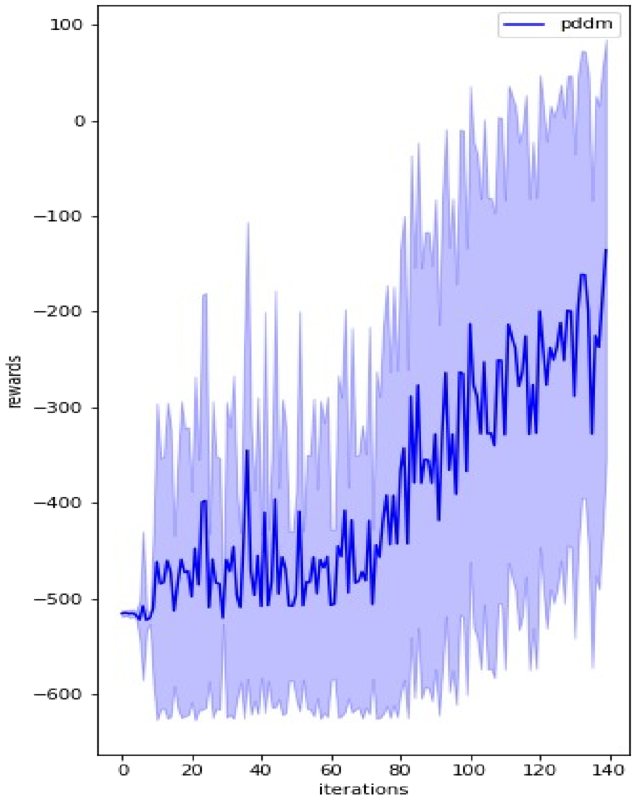
Simulator



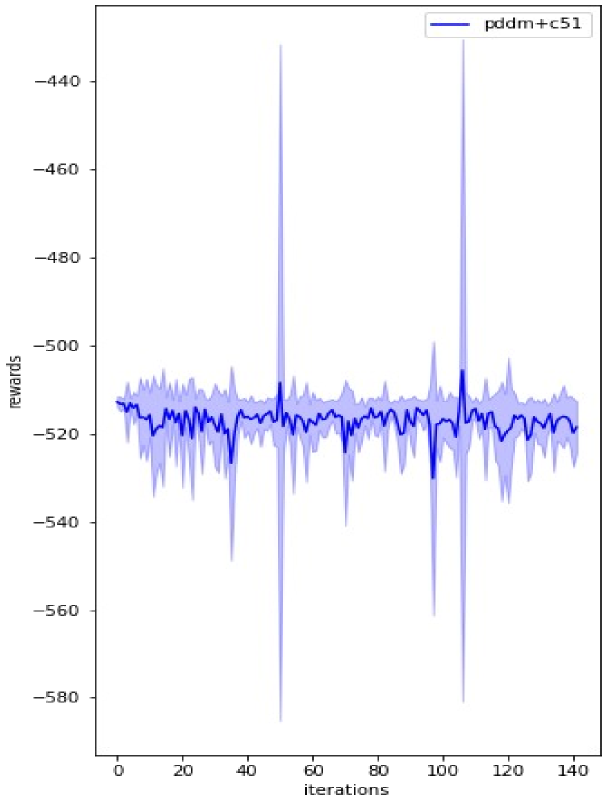
Baoding Balls Manipulation

- State Size: $\mathbb{R}^{40}$
- Action Size: $\mathbb{R}^{24}$
- Reward Formulation: rotating both balls in robot's palm (without any ball falling and robotic wrist < 25 degrees)
- Deterministic environment

Experiment 1

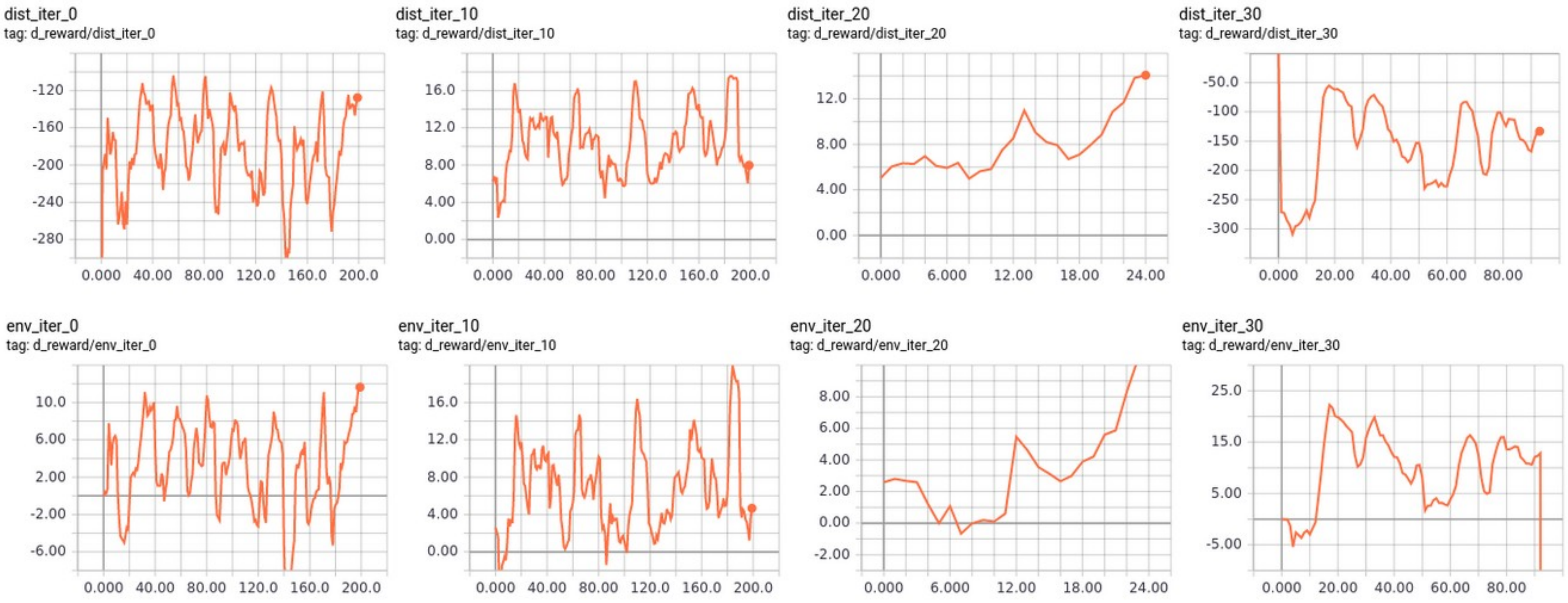


Baoding Balls - PDDM



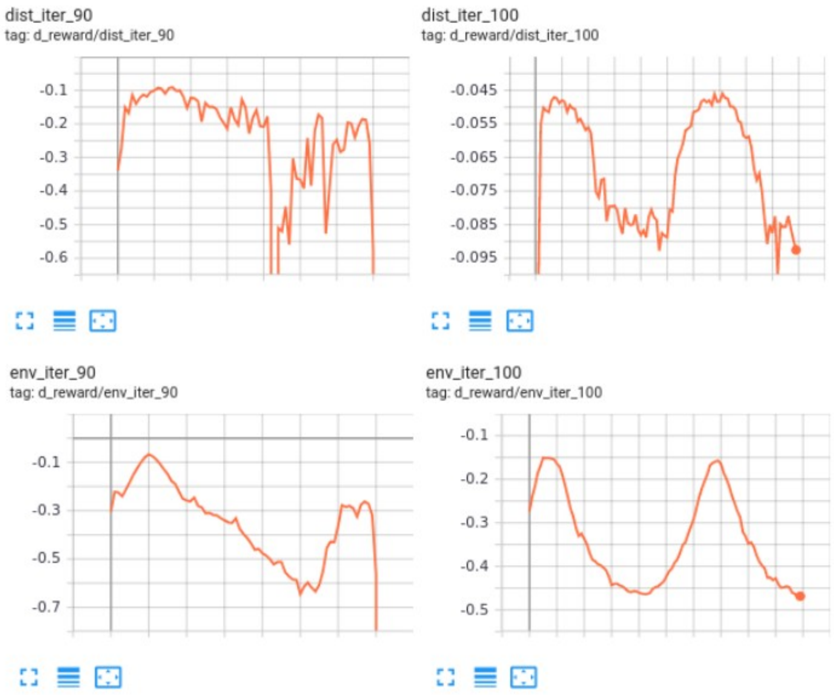
Baoding Balls - PDDM + C51

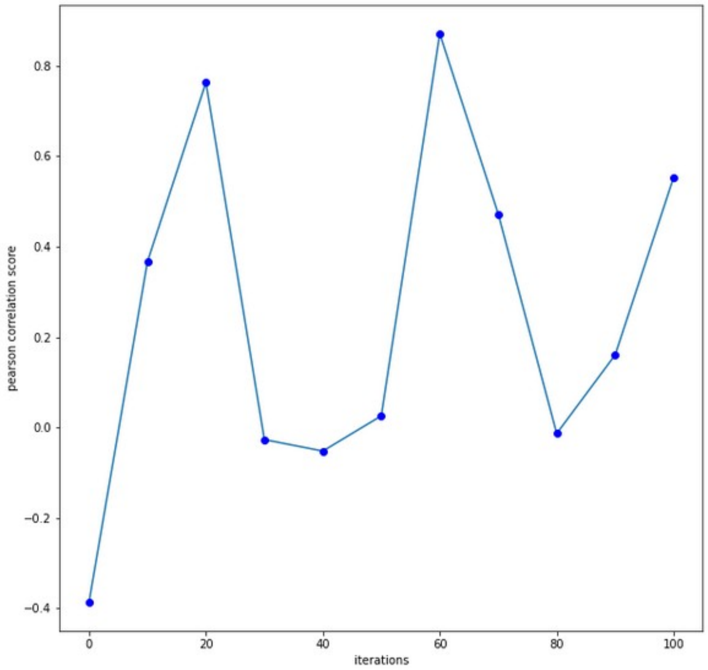
Experiment 1 (contd.)



Comparing distributional reward with actual reward function

Experiment 1 (contd.)

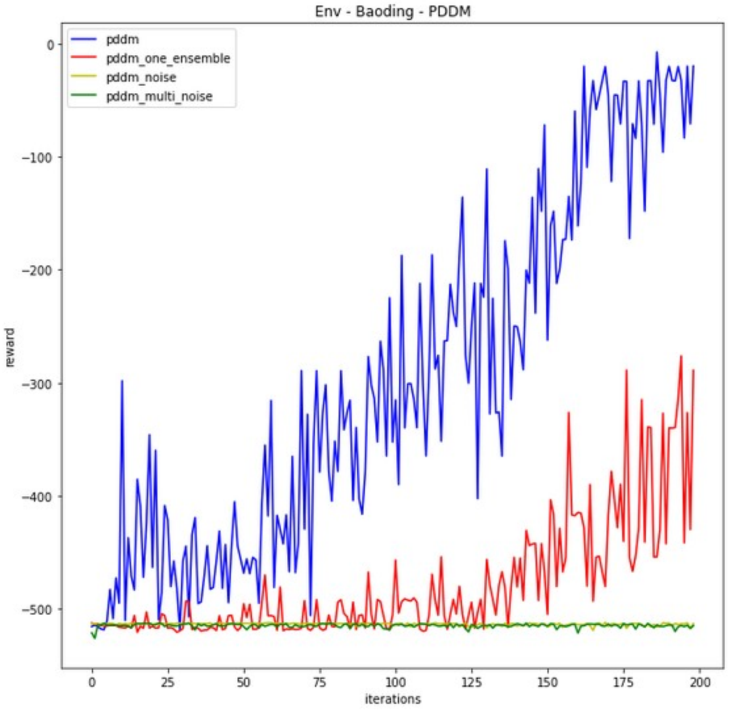


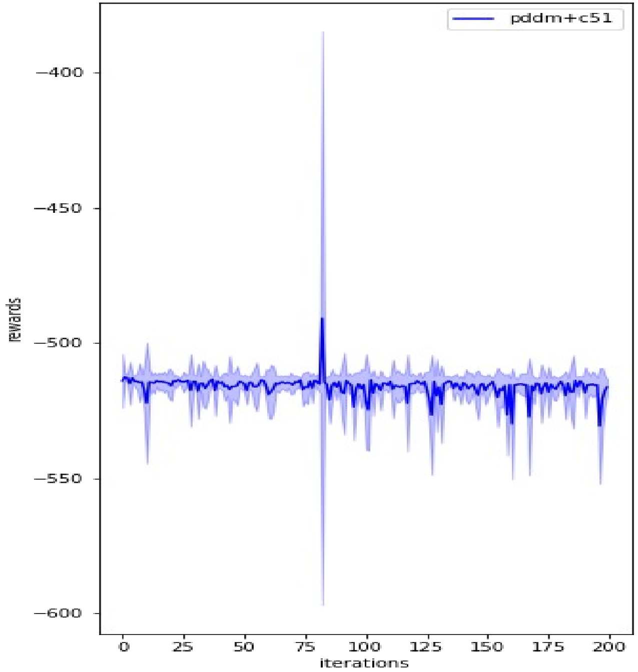


Pearson Co-relation score b/w env and dist rewards

Experiment 2

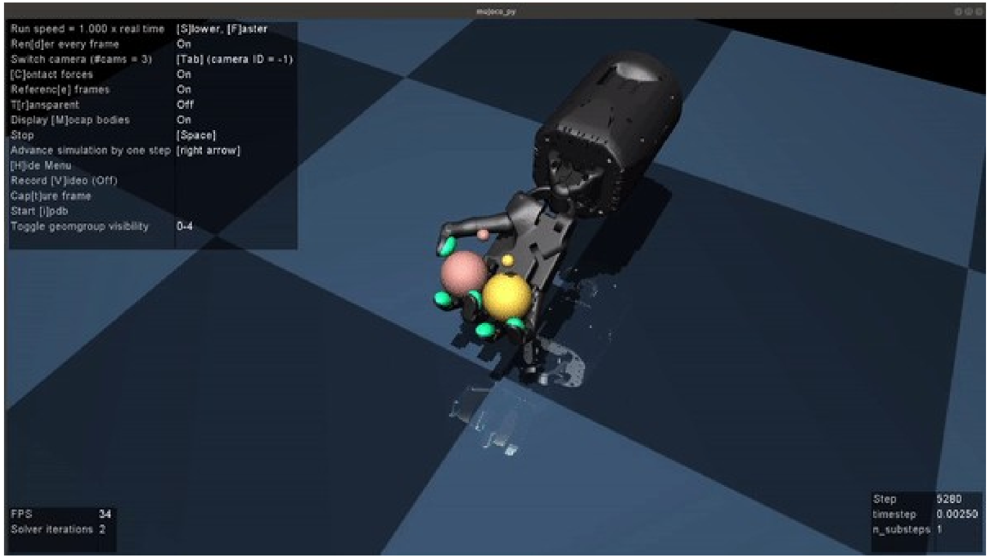
- Adding Gaussian Noise to baoding balls env





Analysis

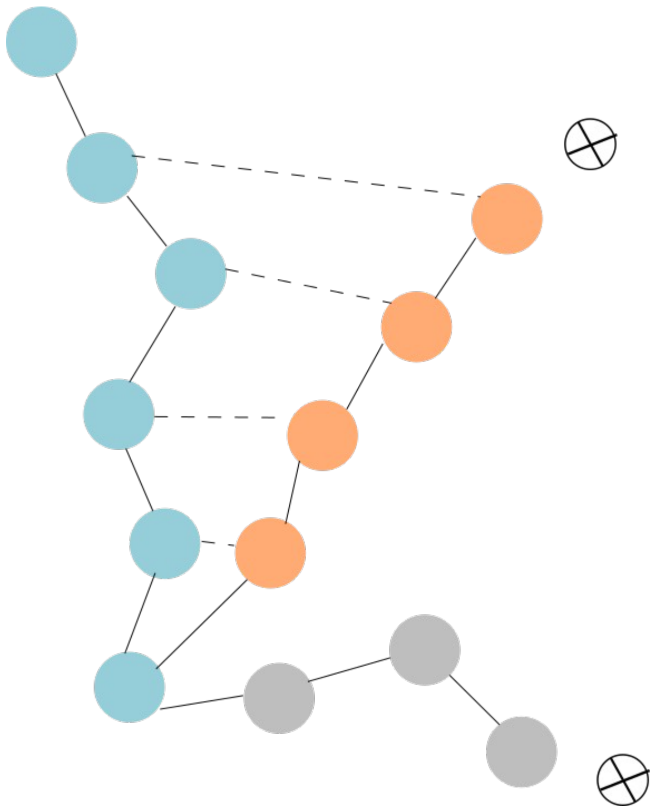
Baoding balls simulation after 5M steps



Pddm (using env reward function)

Pddm (using distributed reward function)

Analysis



Optimal policy

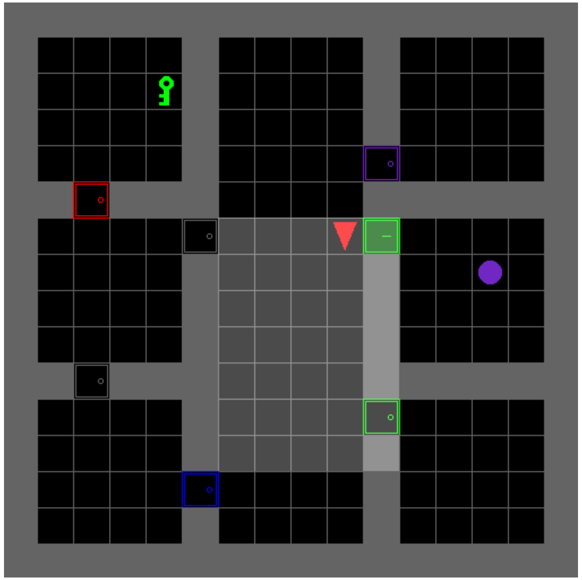
Error due to state-transition model

Compounded error due to state-transition model + distributional reward model

Further Work

Further analysis on small state-space envs

- Gym – minigrid env



Thank You

Questions ?