535514: Reinforcement Learning

Lecture 28 — Model-Based RL

Ping-Chun Hsieh

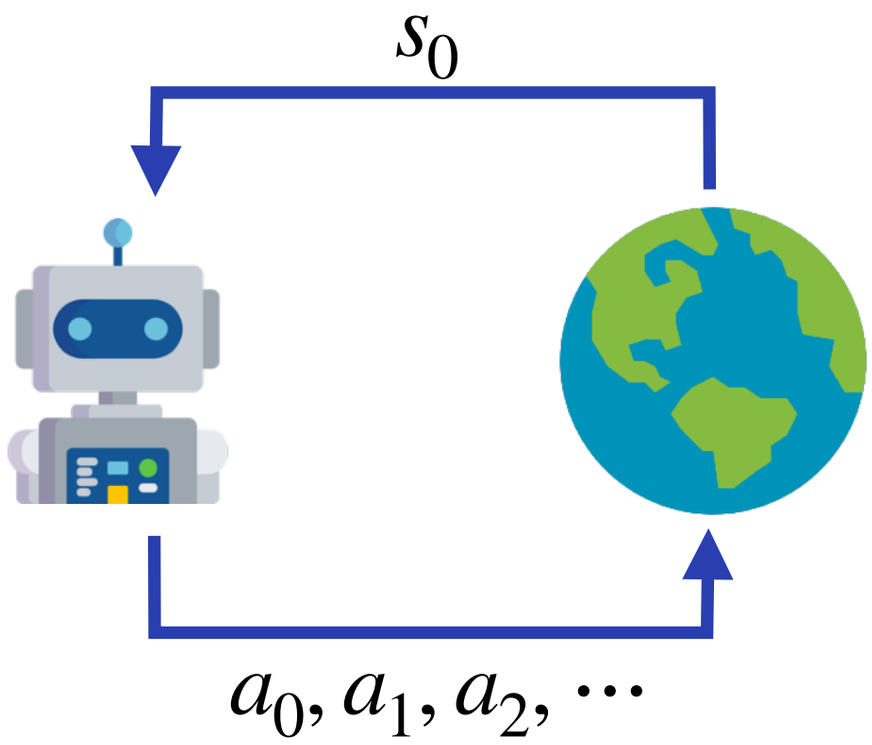
June 6, 2024

On-Policy vs Off-Policy Methods

	Policy Optimization	Value-Based	Model-Based	Imitation-Based
On-Policy	**Exact PG** **REINFORCE (w/i baseline)** **A2C** **On-policy DAC** **TRPO** **Natural PG (NPG)** **PPO-KL & PPO-Clip** **RLHF by PPO-KL**	**Epsilon-Greedy MC** **Sarsa** **Expected Sarsa**	**MCTS** **Model-Predictive Control (MPC)** **PETS** **TT**	**IRL** **GAIL** **WAIL**
Off-Policy	**Off-policy DPG & DDPG** **Twin Delayed DDPG (TD3)**	**Q-learning** **Double Q-learning** **DQN & DDQN** **Rainbow** **C51 / QR-DQN / IQN** **Soft Actor-Critic (SAC)**		

Review: Open-Loop Planning

- Open-loop planning: Given dynamics model and initial state s_0, determine the sequence of actions $a_0, a_1, a_2, \cdots, a_T$

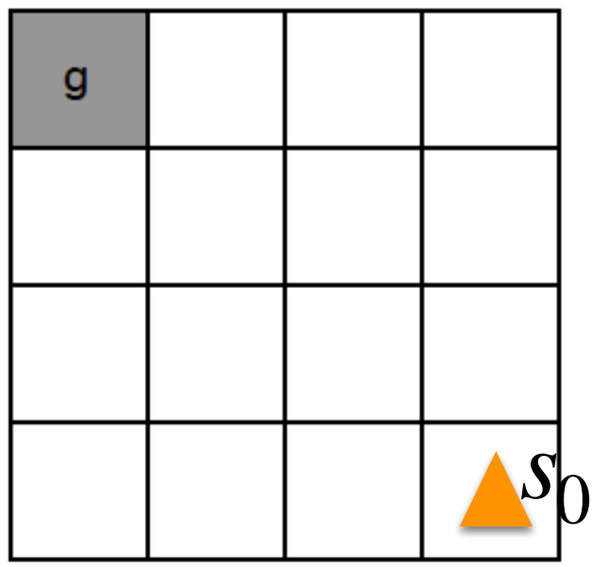


$$\arg\max_{a_0,a_1,\cdots,a_T} \mathbb{E}\left[\sum_{t=0}^{T} R(s_t, a_t)\right]$$

(For simplicity, let's assume $\gamma = 1$)

Example: Deterministic Gridworld

- Reward = -1, for each step
- Episode ends when reaching "g"



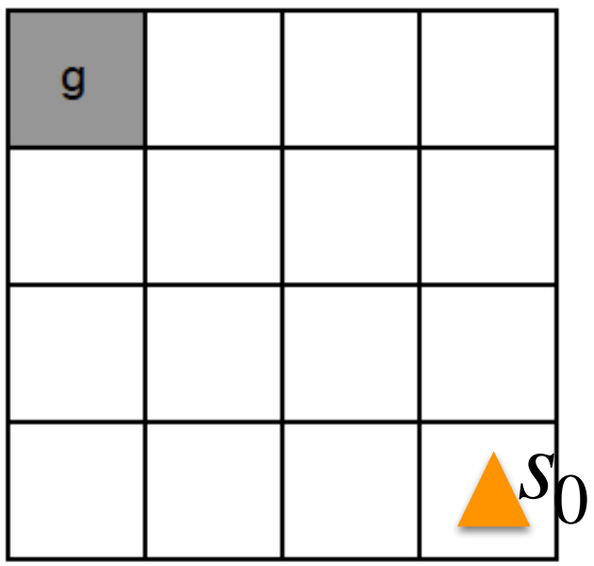
Review: Open-Loop Planning by Optimization

- Rethink open-loop planning as an optimization problem
  (this is often called "**trajectory optimization**")

$$\arg\max_{a_0,a_1,\cdots,a_T} \mathbb{E}\left[\sum_{t=0}^{T} R(s_t, a_t)\right] \xrightarrow{\theta := (a_0, \cdots, a_T)} \arg\max_{\theta} J(\theta)$$

(Here $J(\theta)$ is the total expected reward obtained under θ)

Example: Deterministic Gridworld



Suppose $T = 10$ and set θ as (L, U, D, L, D, L, U, U, U, D, R)

What's the corresponding $J(\theta)$?

Question: In general, given θ, how to obtain $J(\theta)$?

Solution 1: Random Shooting

- Rethink open-loop planning as an optimization problem
  (this is often called "**trajectory optimization**")

$$\arg\max_{a_0,a_1,\cdots,a_T} \mathbb{E}\left[\sum_{t=0}^{T} R(s_t, a_t)\right] \quad \theta := (a_0, \cdots, a_T) \quad \arg\max_{\theta} J(\theta)$$

(Here $J(\theta)$ is the total expected reward obtained under θ)

- **Random shooting** (the simplest open-loop planning approach)

Step 1: Randomly draw action sequences $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(K)}$ from some distribution (e.g., uniformly random)

Step 2: Choose $\theta^* = \arg\max_{k=1,\cdots,K} J(\theta^{(k)})$

Question: Is Random Shooting an efficient method? And why?

Solution 2: Cross-Entropy Method (CEM)

- **Cross-Entropy Method** (a popular open-loop planning approach)

Step 1: Randomly sample action sequences $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(K)}$ from some distribution $\rho(\theta)$

Step 2: Evaluate $J(\theta^{(1)}), \cdots, J(\theta^{(K)})$ and choose $\theta^* = \arg \max_{k=1,\cdots,K} J(\theta^{(k)})$

Step 3: Select the top-N candidates $J(\theta^{(i_1)}), \cdots, J(\theta^{(i_N)})$

Step 4: Re-fit $\rho(\theta)$ to $\theta^{(i_1)}, \cdots, \theta^{(i_N)}$ by maximum likelihood estimation (MLE)

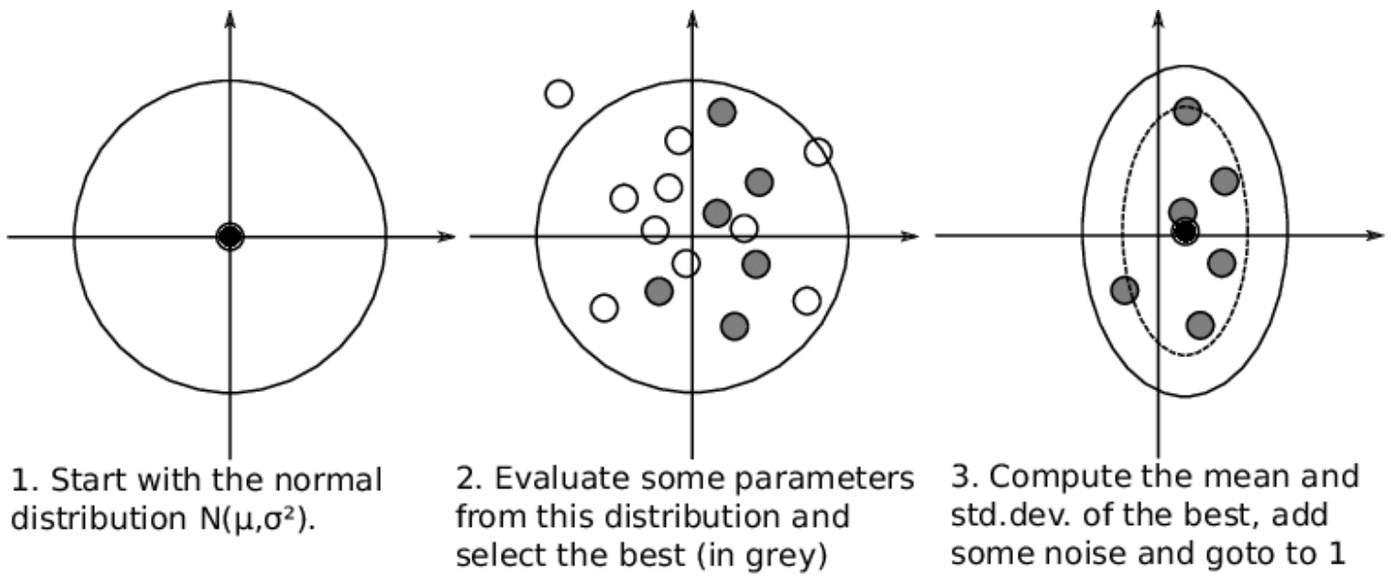


Figure Source: Towards fast and adaptive optimal control policies for robots: A direct policy search approach

Discussions on Open-Loop Planning

- Main advantages of open-loop planning:

(A1) Simple, zeroth-order optimization (no gradient at all!)

(A2) Very computationally efficient under parallelization

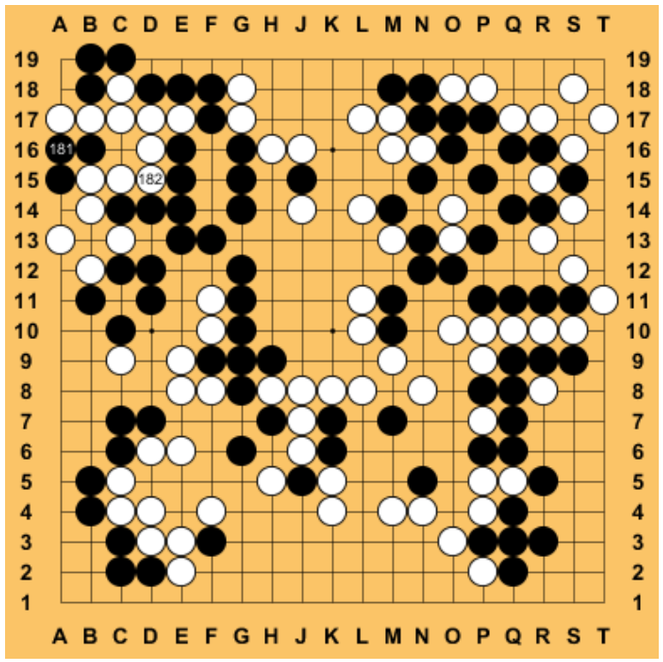
- Main drawbacks of open-loop planning:

(D1) Not scalable (dimensionally grows with T)

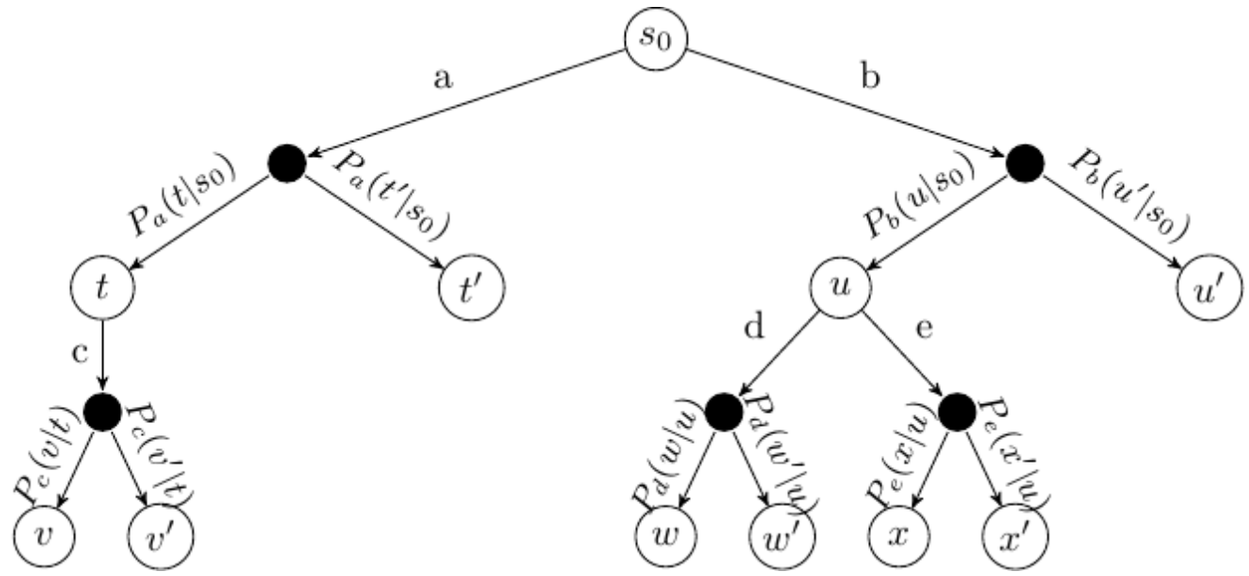
(D2) Can be sub-optimal (e.g., in stochastic environments)

Question: How to do more efficient “planning”?

For (D1): Monte-Carlo Tree Search (MCTS)



In discrete cases (state and action spaces are discrete), "θ" can be expressed as a tree



However, the number of nodes can grow exponentially with planning horizon T

Question: How to plan without building a full tree?

Intuition: Select the nodes with highest reward (exploit), but also spend some time visiting some unfamiliar nodes (explore)

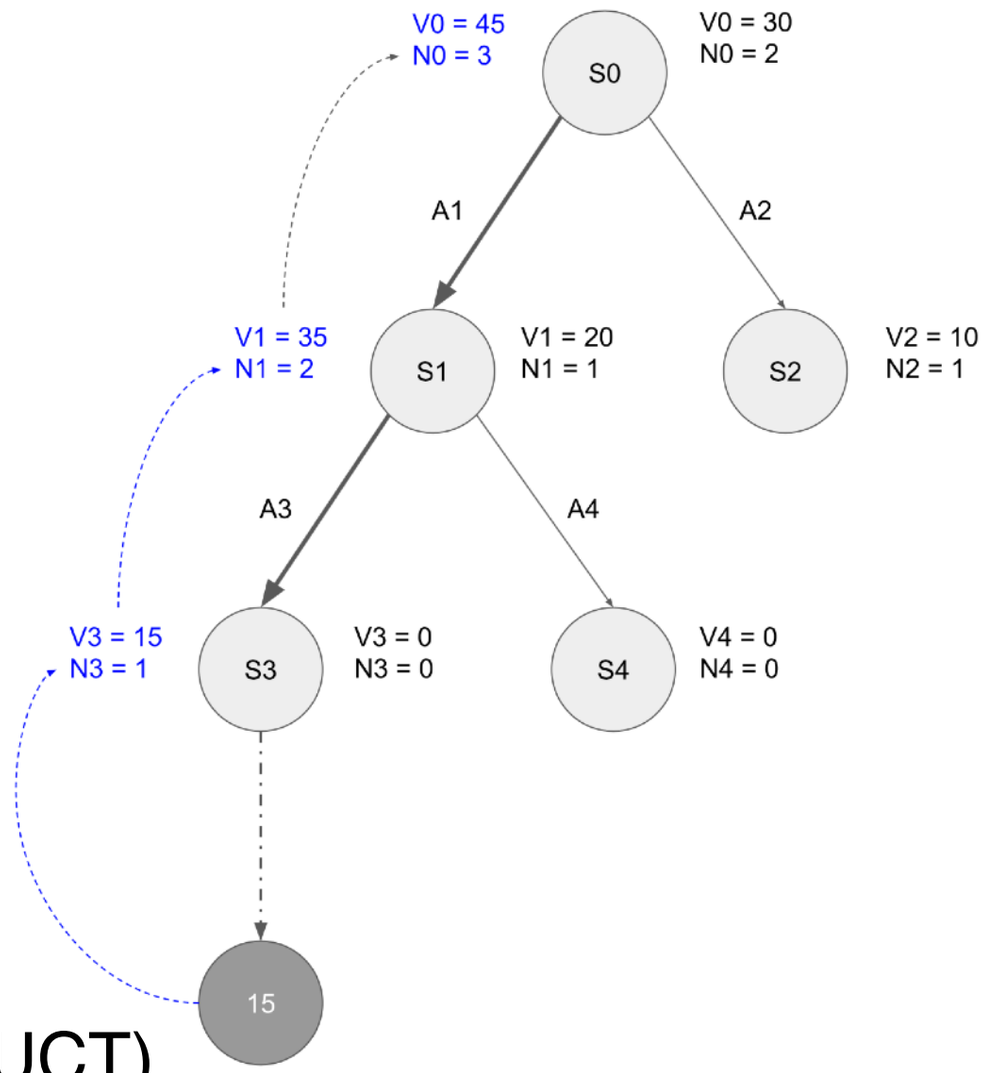
Case Study: Monte-Carlo Tree Search (MCTS)

- **General Recipe of MCTS**

Step 1: Traverse the tree from s_0 to a leaf node s_n by using TreePolicy

Step 2: Evaluate the leaf node s_n by some EvaluatePolicy

Step 3: Update the values of all the nodes between s_0 and s_n



A popular TreePolicy: Upper-Confidence Tree (UCT)

$$\text{Score of } (s,a) = \frac{V(s,a)}{N(s,a)} + C\sqrt{\frac{\log N(s)}{N(s,a)}}$$

Levente Kocsis and Csaba Szepesvari, “Bandit based Monte-Carlo Planning,” ECML 2016

The Classic UCT Paper

Bandit based Monte-Carlo Planning

Levente Kocsis and Csaba Szepesvári

Computer and Automation Research Institute of the
Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary
kocsis@sztaki.hu

[ECML 2006]

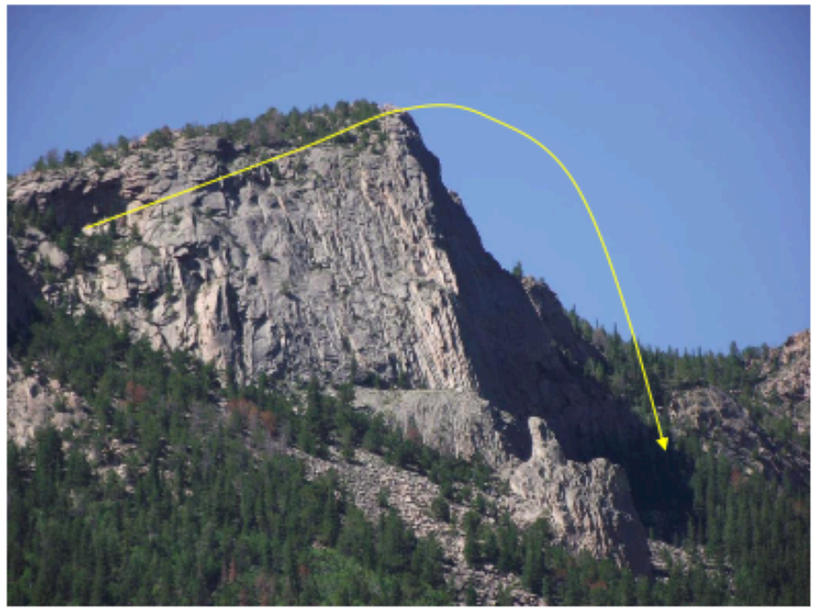
Abstract. For large state-space Markovian Decision Problems Monte-Carlo planning is one of the few viable approaches to find near-optimal solutions. In this paper we introduce a new algorithm, UCT, that applies bandit ideas to guide Monte-Carlo planning. In finite-horizon or discounted MDPs the algorithm is shown to be consistent and finite sample bounds are derived on the estimation error due to sampling. Experimental results show that in several domains, UCT is significantly more efficient than its alternatives.

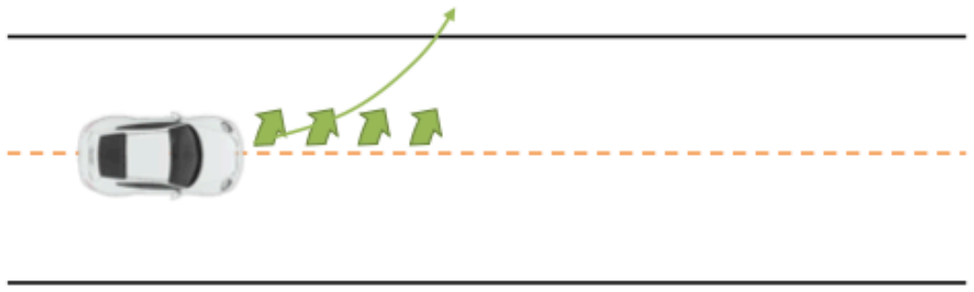
Levente Kocsis

Csaba Szepesvari

For (D2): Closed-Loop Planning

- Why is open-loop planning sub-optimal?





1. Stochastic transitions can put as in unexpected or even dangerous states (even if the dynamics model is fully known)

2. If the dynamics mode is NOT known, then we can make more mistakes

- Close-Loop Planning: Replan as you go (replan to fix the mistakes)

(Figure Source: Sergey Levine)

Closed-Loop Planning: Model-Predictive Control (MPC)

- **Model-Predictive Control**

At each time step $t = 0,1,2\cdots, T$

Step 1: Plan from current state s_t for a future horizon $H < T$
(e.g., by random shooting or CEM)

Step 2: Execute the first planned action and observe the next state s_{t+1}

Nice features:

1. Short-horizon planning would work
2. Replanning helps with model errors
3. Even random shooting would work sufficiently well!

(Figure Source: Sergey Levine)

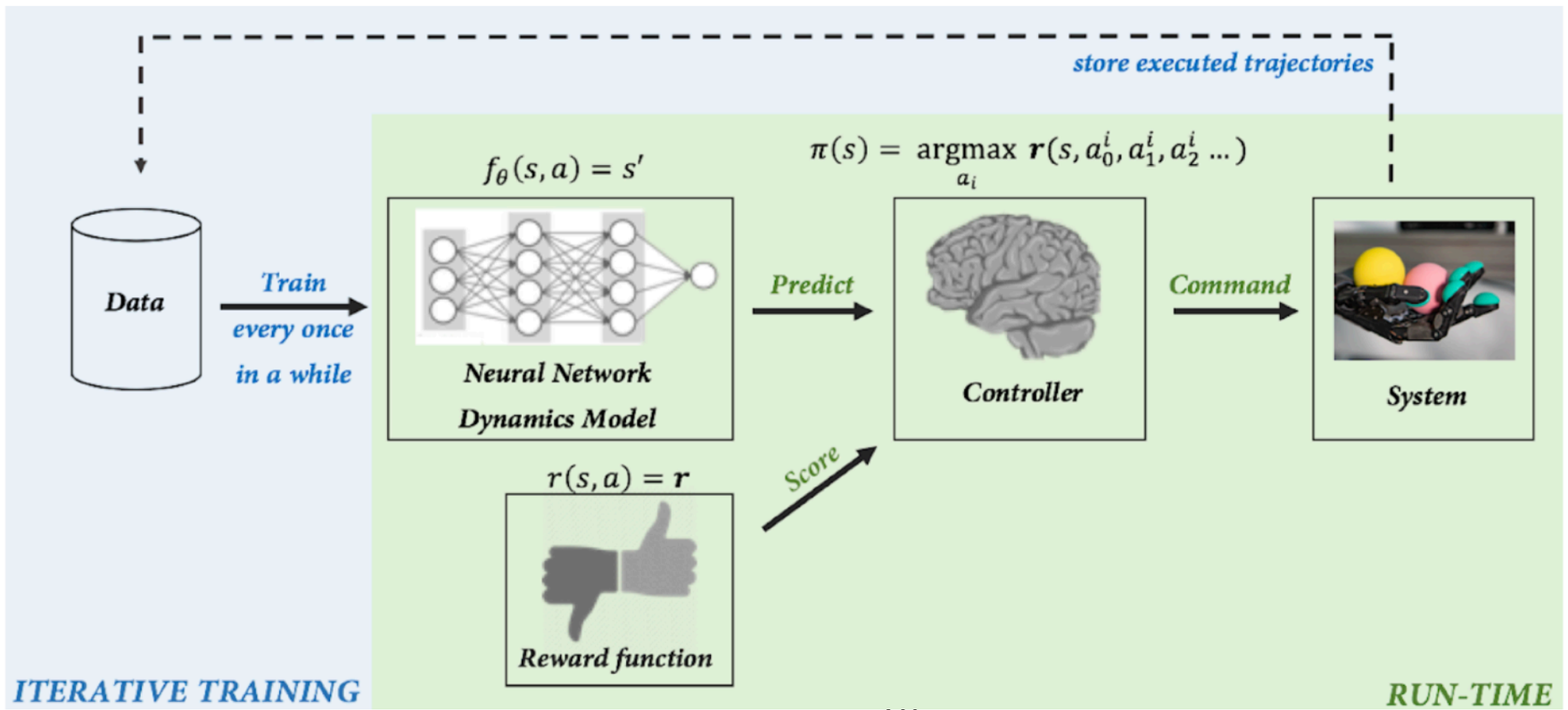
A Practical Example of MPC: PDDM for Robot Arm

PDDM = Online **P**lanning with **D**eep **D**ynamics **M**odels

Deep Dynamics Models for Learning Dexterous Manipulation

Anusha Nagabandi, Kurt Konoglie, Sergey Levine, Vikash Kumar
Google Brain

Abstract: Dexterous multi-fingered hands can provide robots with the ability to flexibly perform a wide range of manipulation skills. However, many of the more complex behaviors are also notoriously difficult to control: Performing in-hand object manipulation, executing finger gaits to move objects, and exhibiting precise fine motor skills such as writing, all require finely balancing contact forces, breaking and reestablishing contacts repeatedly, and maintaining control of unactuated objects. Learning-based techniques provide the appealing possibility of acquiring these skills directly from data, but current learning approaches either require large amounts of data and produce task-specific policies, or they have not yet been shown to scale up to more complex and realistic tasks requiring fine motor skills. In this work, we demonstrate that our method of online planning with deep dynamics models (PDDM) addresses both of these limitations; we show that improvements in learned dynamics models, together with improvements in online model-predictive control, can indeed enable efficient and effective learning of flexible contact-rich dexterous manipulation skills – and that too, on a 24-DoF anthropomorphic hand in the real world, using just 4 hours of purely real-world data to learn to simultaneously coordinate multiple free-floating objects. Videos can be found at https://sites.google.com/view/pddm/

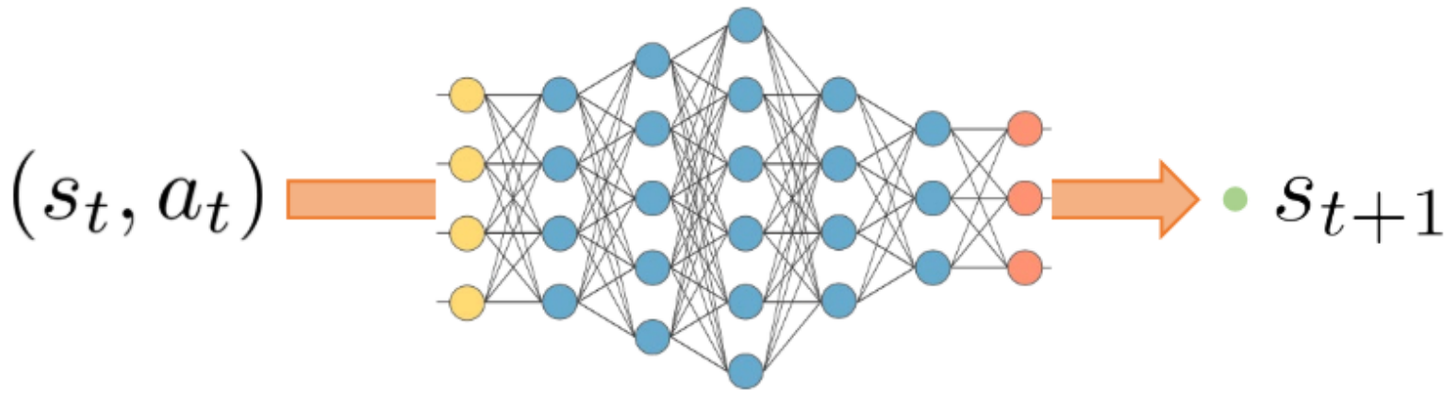


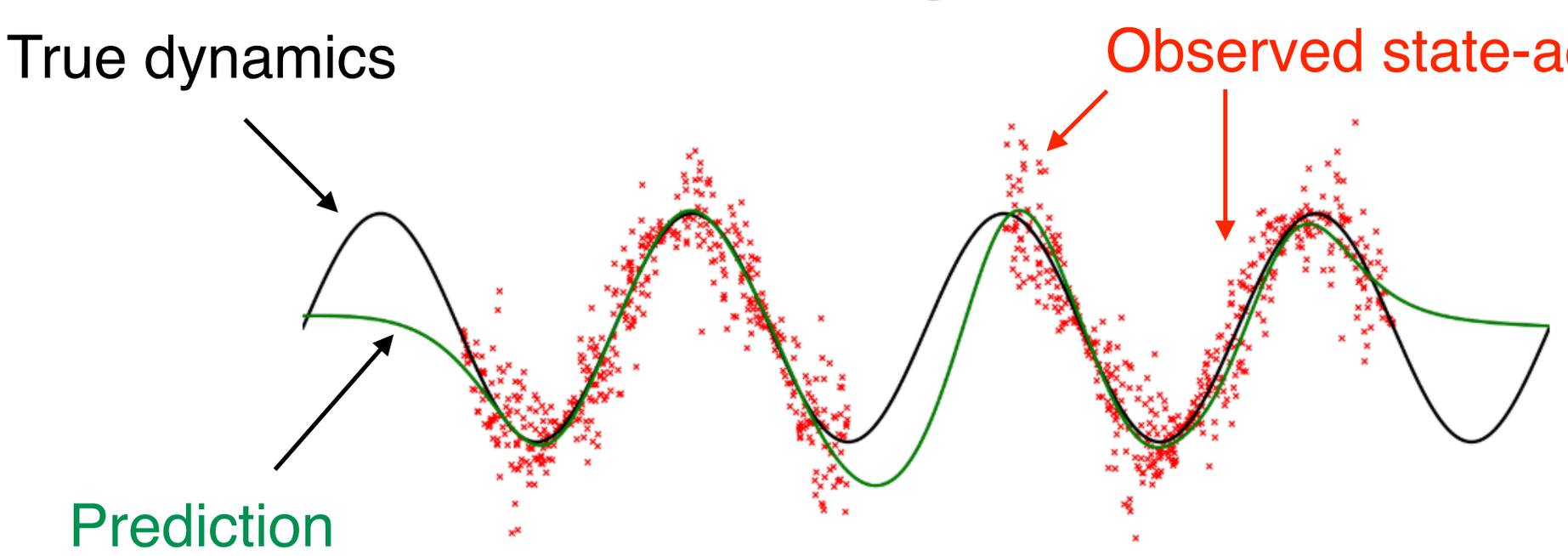
https://sites.google.com/view/pddm/

Next Question: How to learn the dynamics model?

The Simplest Case: Deterministic Neural Nets as Models

- Model learning as a regression problem

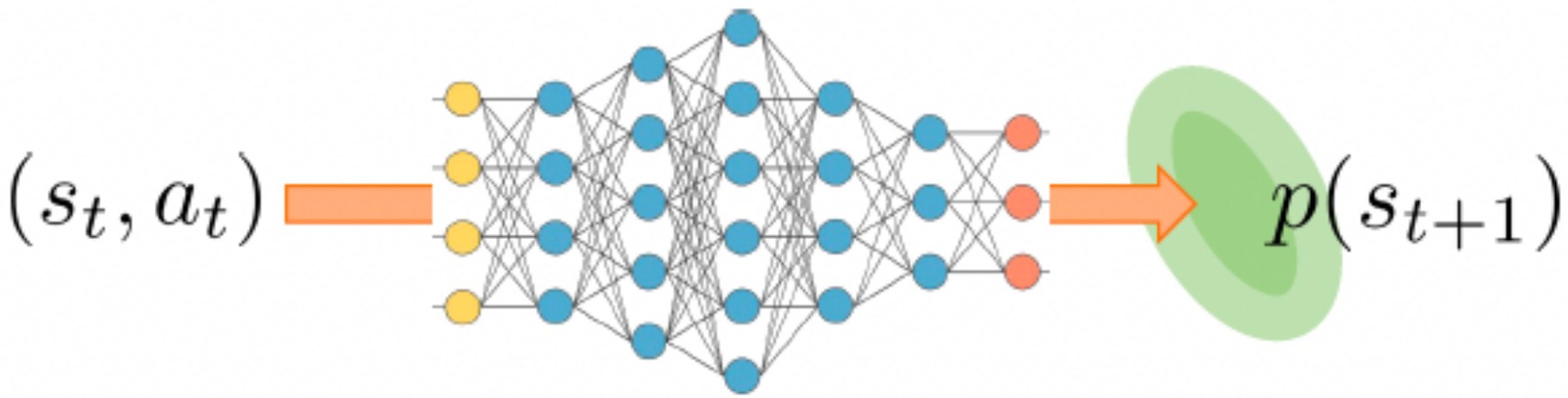


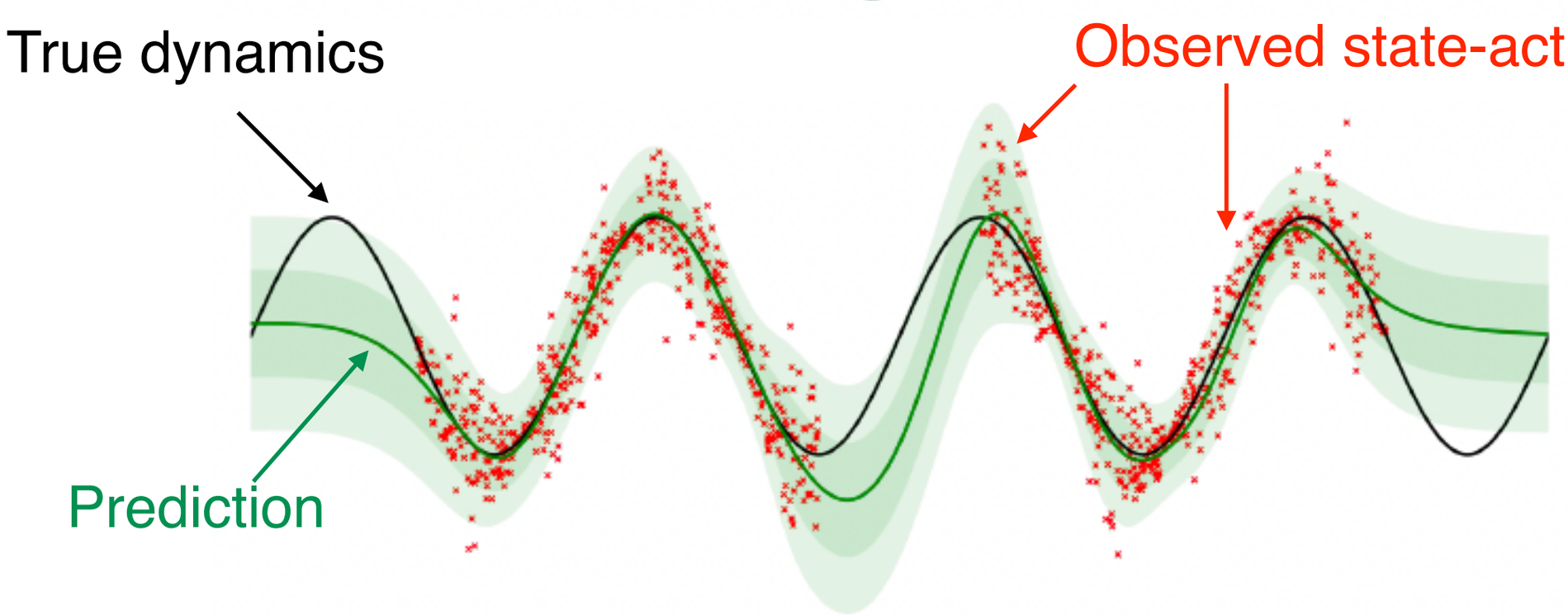


(Figure Source: Kurtland Chua)

Enhancement #1: Probabilistic Neural Nets as Models

- Model learning as a (distributional) regression problem

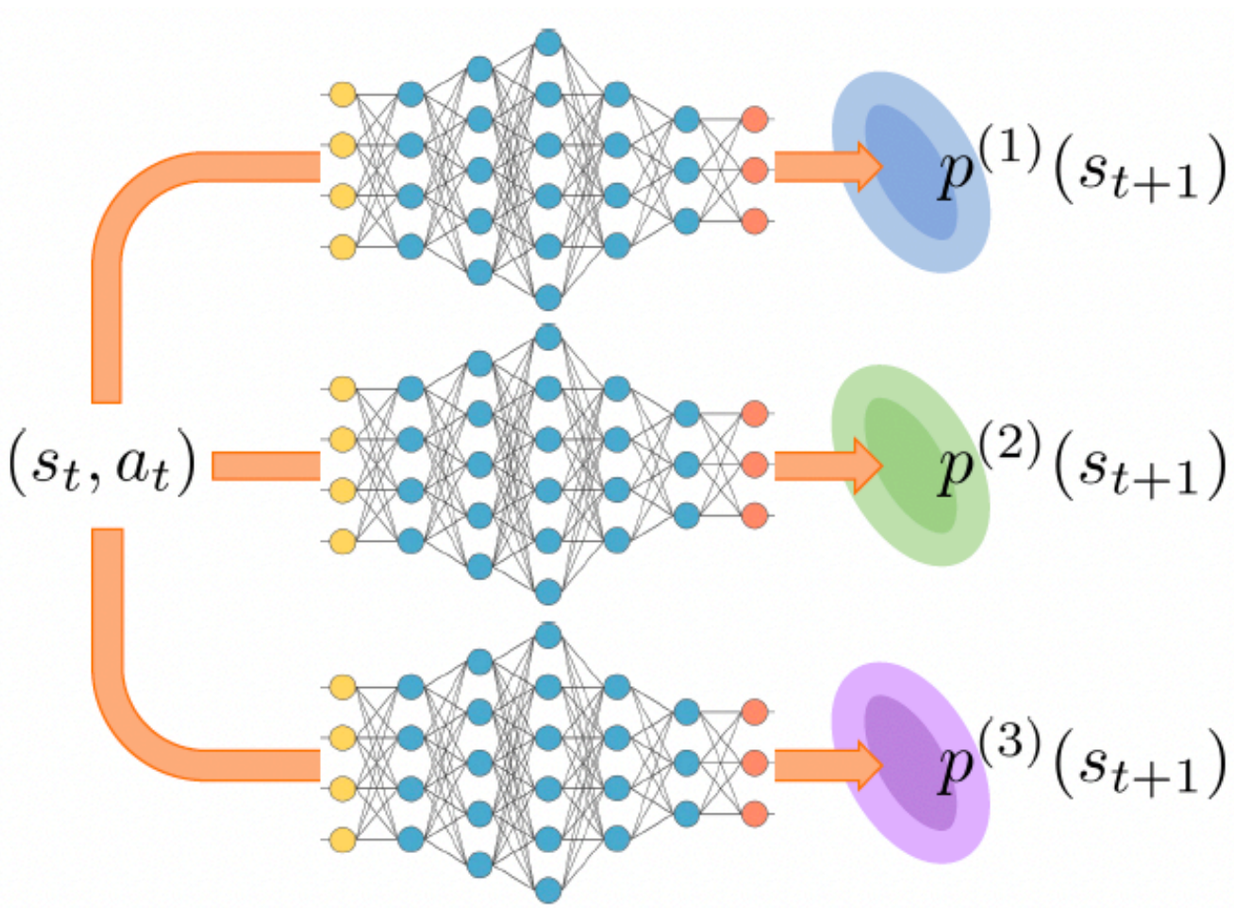


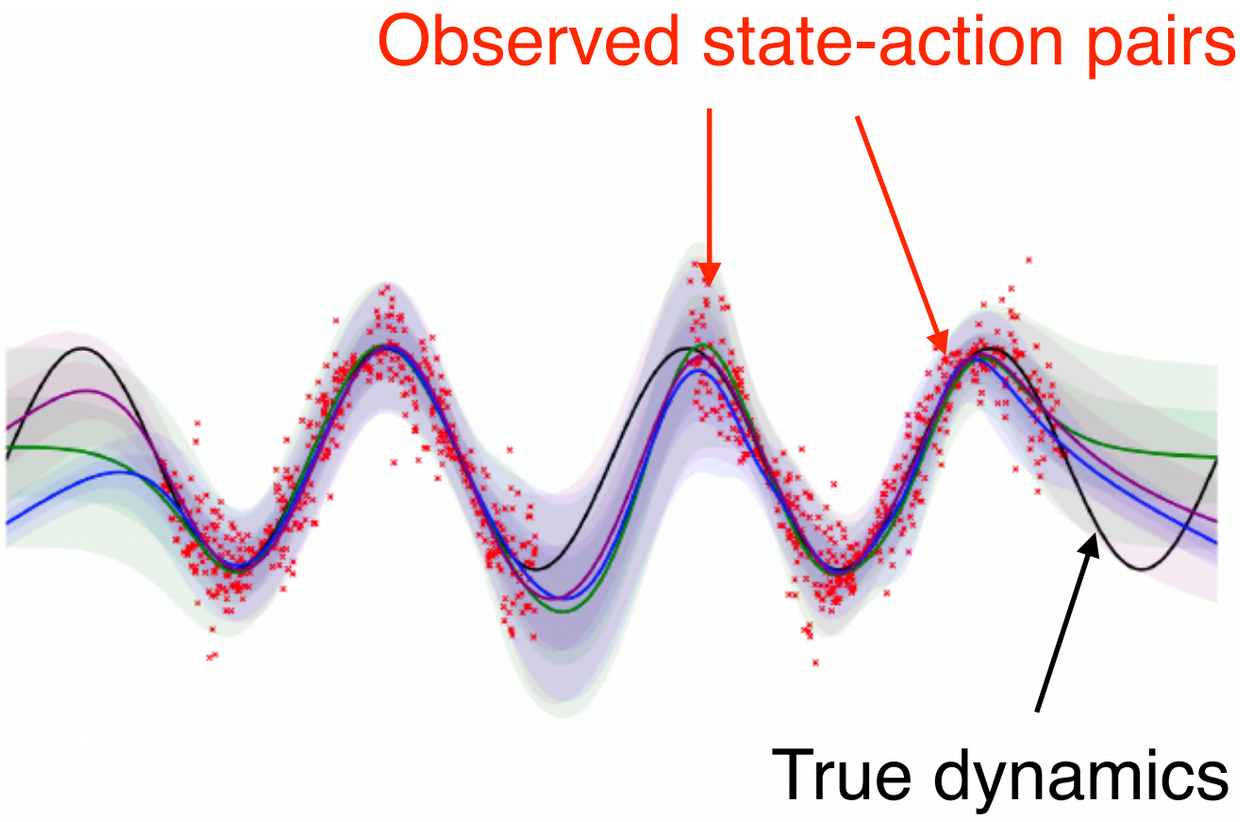


(Figure Source: Kurtland Chua)

Enhancement #2: Probabilistic Ensembles as Models

- Model learning as a (distributional) regression problem

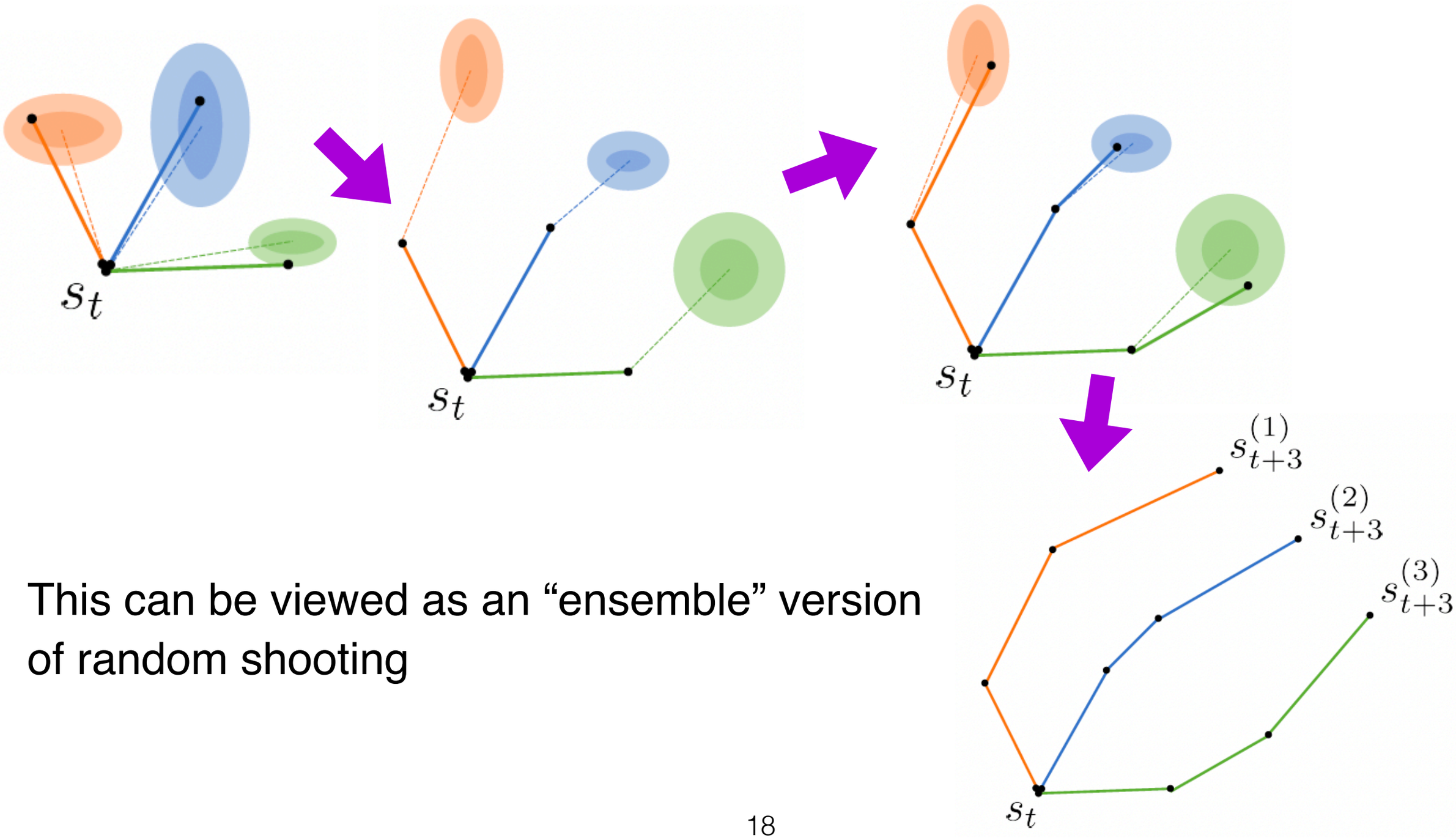




(Figure Source: Kurtland Chua)

Probabilistic Ensembles with Trajectory Sampling (PETS)

- **Trajectory sampling** for utilizing ensembles of models



This can be viewed as an “ensemble” version of random shooting

Model-Predictive Control With PETS

- **MPC with PETS**

At each time step $t = 0,1,2\cdots, T$

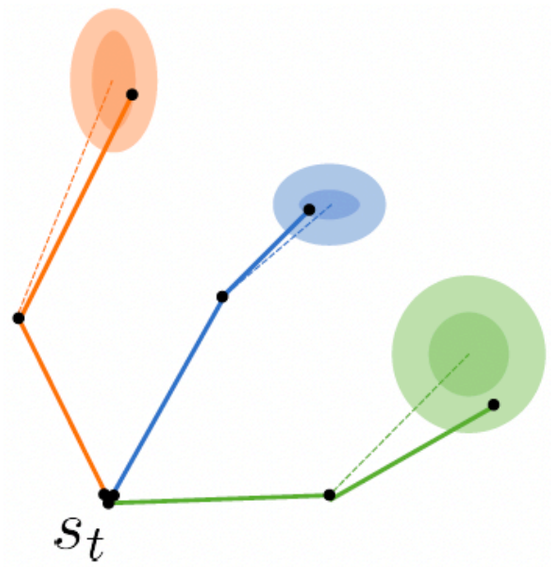
Step 1: Plan from current state s_t for a future horizon $H < T$ by the ensemble models of PETS (e.g., by random shooting or CEM)

Step 2: Execute the first planned action and observe the next state s_{t+1}

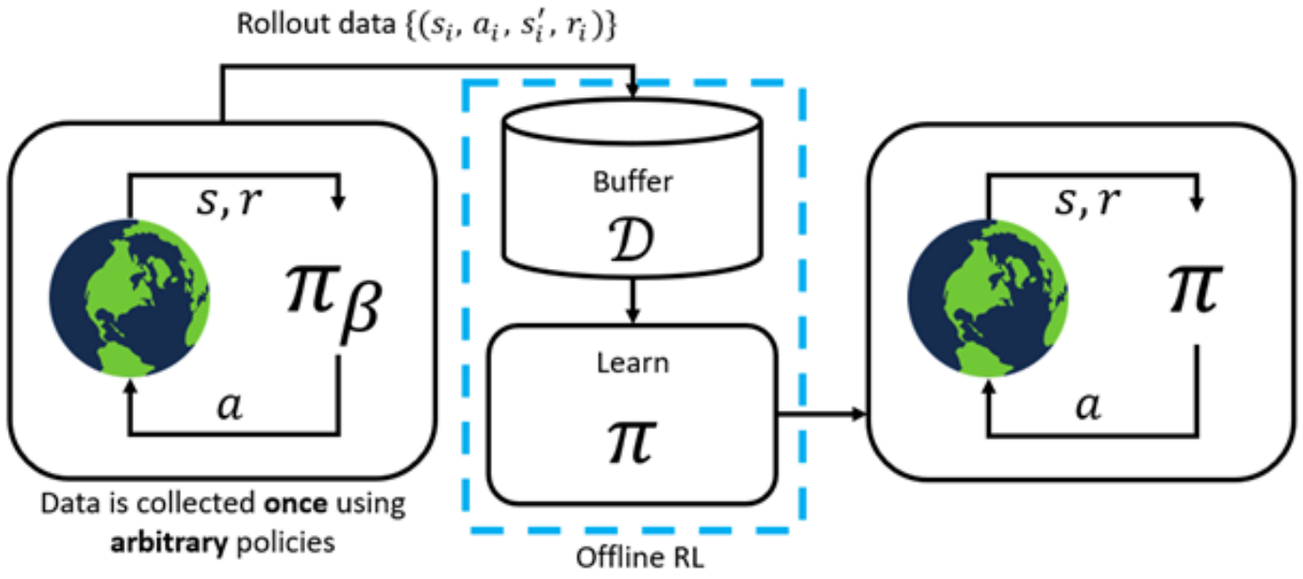
Nice features:

1. Ensemble models are **uncertainty-aware**

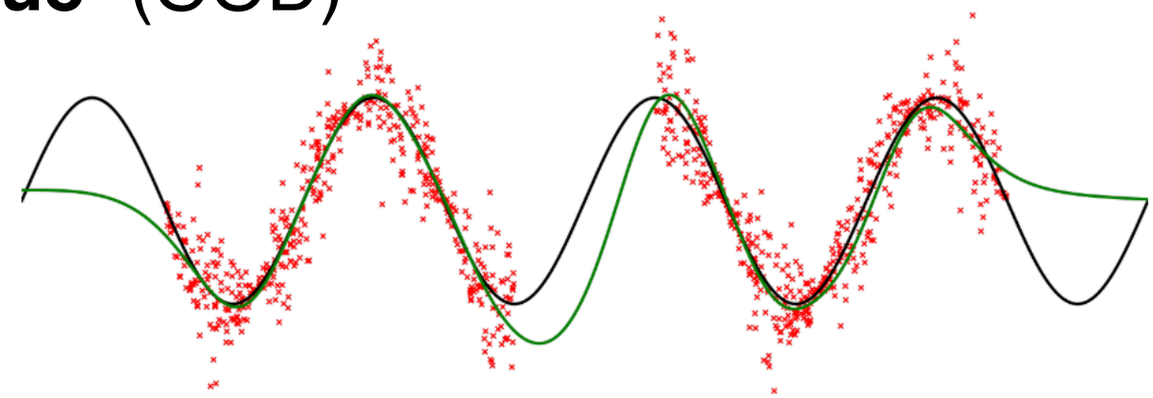
2. More efficient than simple random shooting



Why MBRL is Popular for Offline RL?

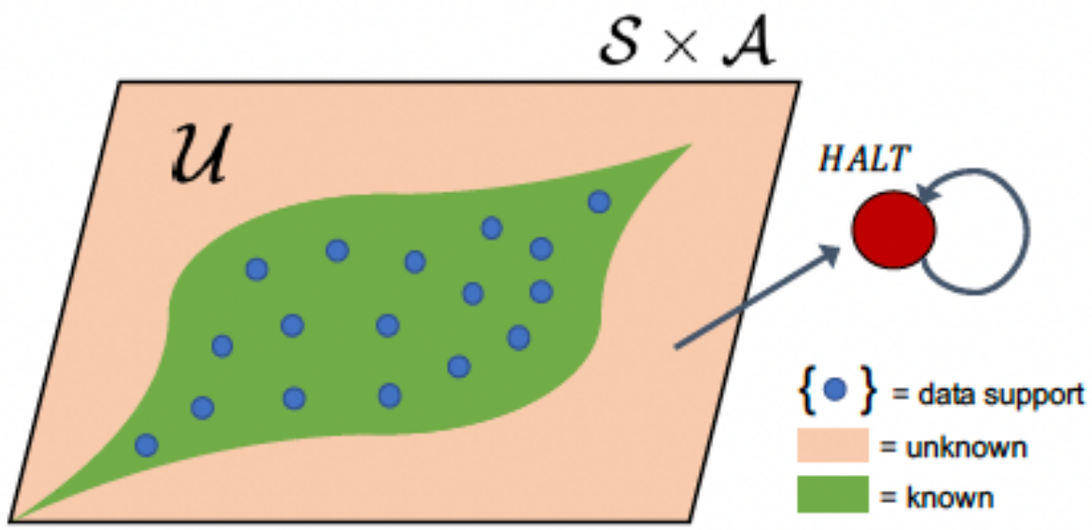


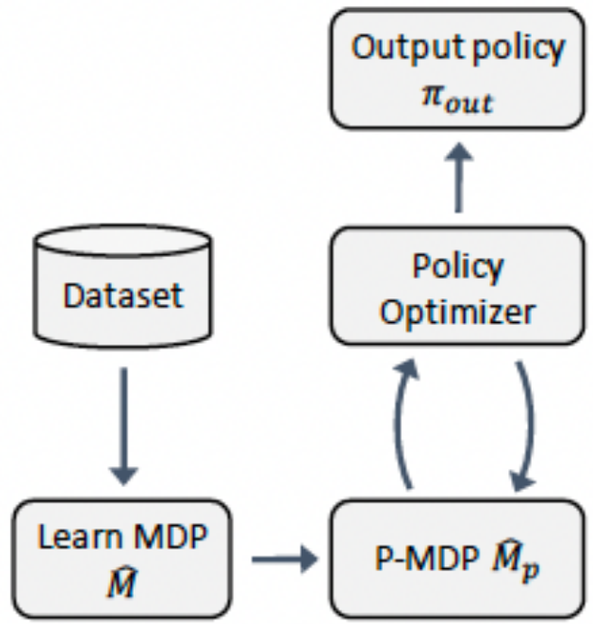
- In offline RL, the dataset can be collected by an arbitrary policy
- Therefore, it can be difficult to estimate value functions (Q or V)
- Nevertheless, we can still learn the dynamics model and use MBRL
- Challenge: "**Out-of-distribution issue**" (OOD)



MOReL: A Simple Offline MBRL Method

- A Straightforward Idea: Prevent the learned policy from visiting those OOD regions by "Pessimism"





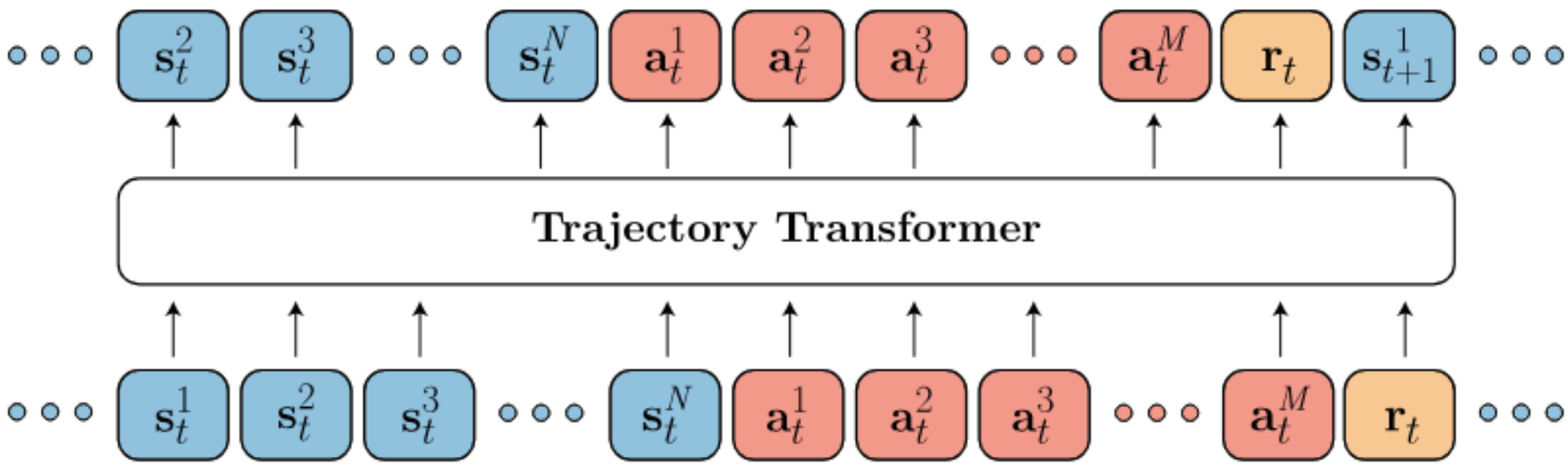
$$\hat{P}_p(s'|s,a) = \begin{cases} \delta(s' = \text{HALT}) & \text{if } U^\alpha(s,a) = \text{TRUE} \\ & \text{or } \ s = \text{HALT} \\ \hat{P}(s'|s,a) & \textit{otherwise} \end{cases} \qquad r_p(s,a) = \begin{cases} -\kappa & \text{if } s = \text{HALT} \\ r(s,a) & \textit{otherwise} \end{cases}$$

$$U^\alpha(s,a) = \begin{cases} \textit{FALSE (i.e. Known)} & \text{if } D_{TV}\left(\hat{P}(\cdot|s,a), P(\cdot|s,a)\right) \leq \alpha \ \textit{can be guaranteed} \\ \textit{TRUE (i.e. Unknown)} & \textit{otherwise} \end{cases}$$

Kidambi et al., "MOReL: Model-Based Offline Reinforcement Learning," NeurIPS 2020

Next Question: How to go beyond one-step prediction?

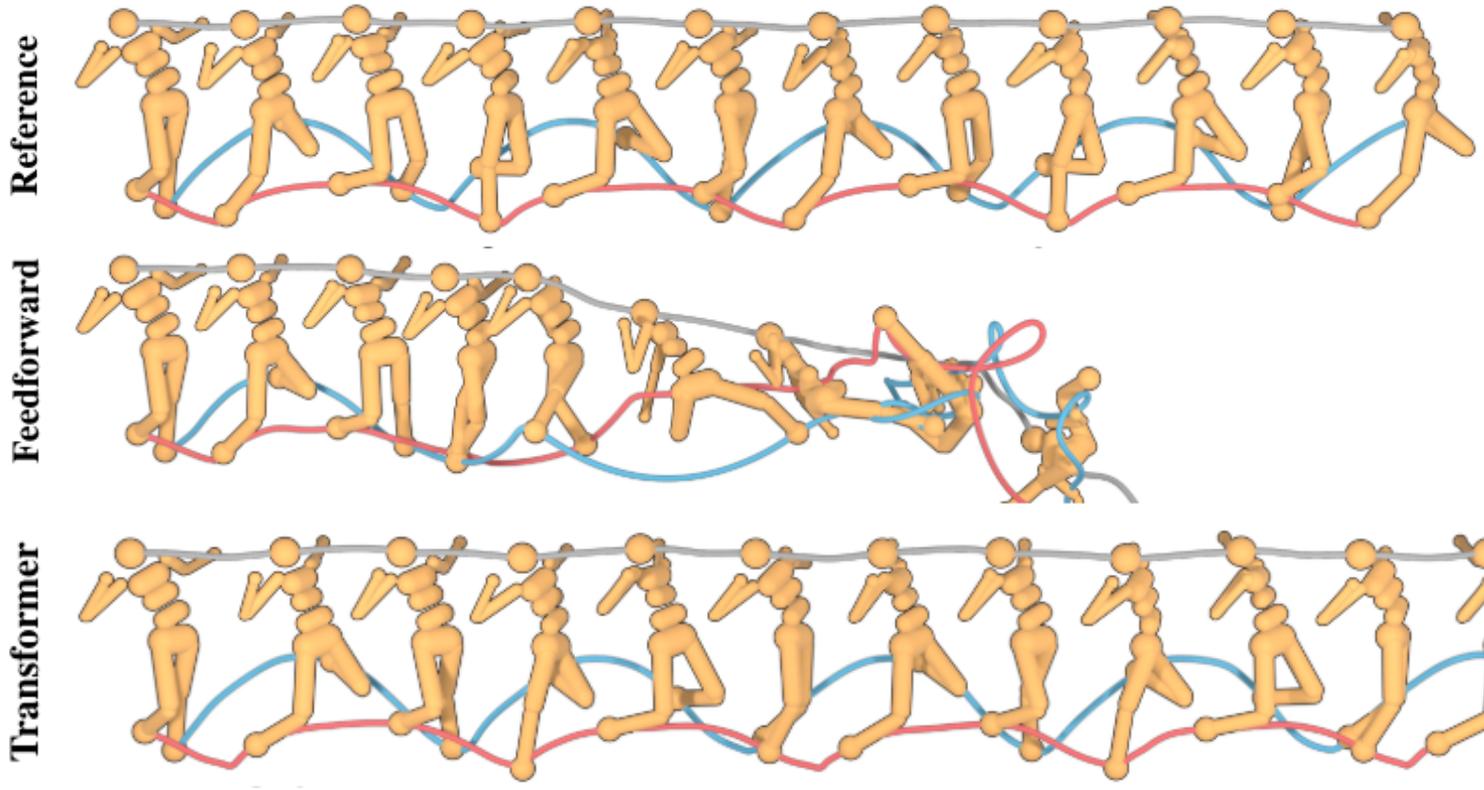
Trajectory Transformer (TT)

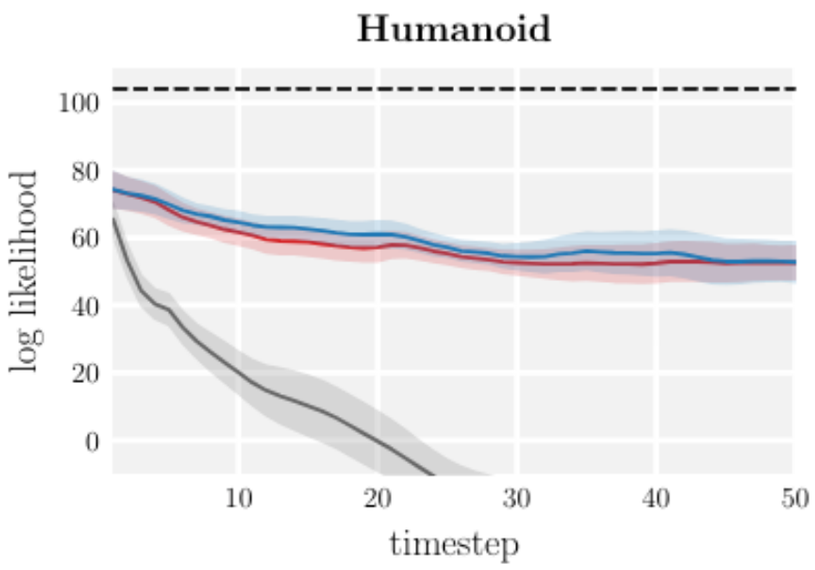


- Trajectory transformer is trained on sequences of (autoregressively discretized) states, actions, and rewards.
- TT is an offline model-based method

Janner et al., "Offline Reinforcement Learning as One Big Sequence Modeling Problem," NeurIPS 2021

Trajectory Transformer





Hope that you find RL research appealing (at least a bit) through this course

Still, we may forget about most of the details in about 2 months…

Despite this, we will still know how to think like an RL researcher

Takeaway

1. RL ≈ (Model-Free) Generalized Policy Iteration

2. Advantage function is the key to policy improvement

3. Off-policy learning could be very useful

4. Exploration does matter

5. Model + planning can be efficient