Master Reinforcement Learning 2022 Lecture 5: Model Based Methods

Aske Plaat

Different Approaches

- Model-free
  - Value-based [2,3]
  - Policy-based [4]
- Model-based
  - Learned [5]
  - Perfect; Two-Agent [6]
- Multi-agent [7]
- Hierarchical Reinforcement Learning (Sub-goals) [8]
- Meta Learning [9]

Motivation

Warning

- Model-free (value-based & policy-based) is mature RL (things work)

- Everything from now on starts to be more advanced, and less mature (research; not all things work; wild ride)

- More exciting, but less secure
  More questions, fewer answers

Overview

- Model Free is great, but sample complexity
- Model Free: Learn Policy (s,a)
- Model Based: Learn Transitions (s,s) [and use that to learn policy (s,a)]
- Classic: Dyna
- Model Learning:
  - Uncertainty, Ensembles
  - Latent Models
- Planning the Model:
  - Short Rollouts, replanning (MPC)
  - End-to-end Planning
- PlaNet, Dreamer
- MuZero

Sample Complexity

$\epsilon = 0.2$. We see that PPO outperforms the previous methods on almost all the continuous control environments.

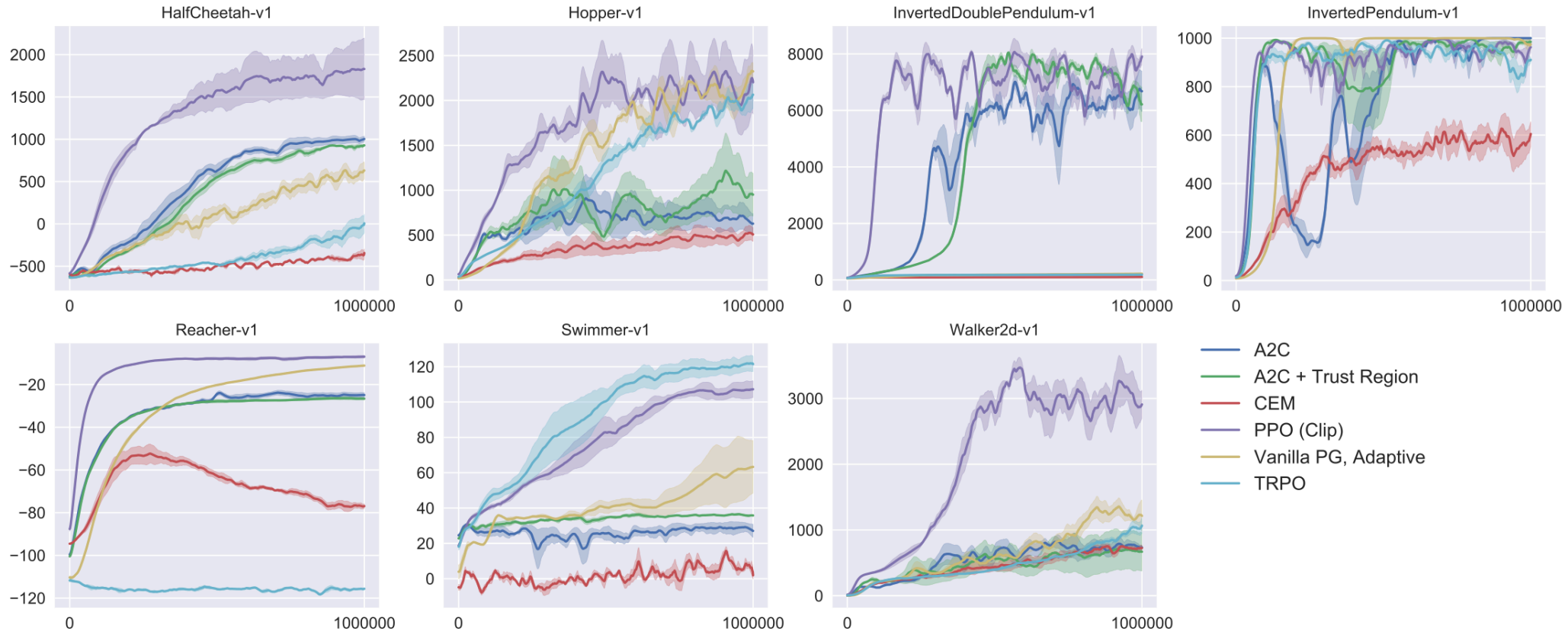
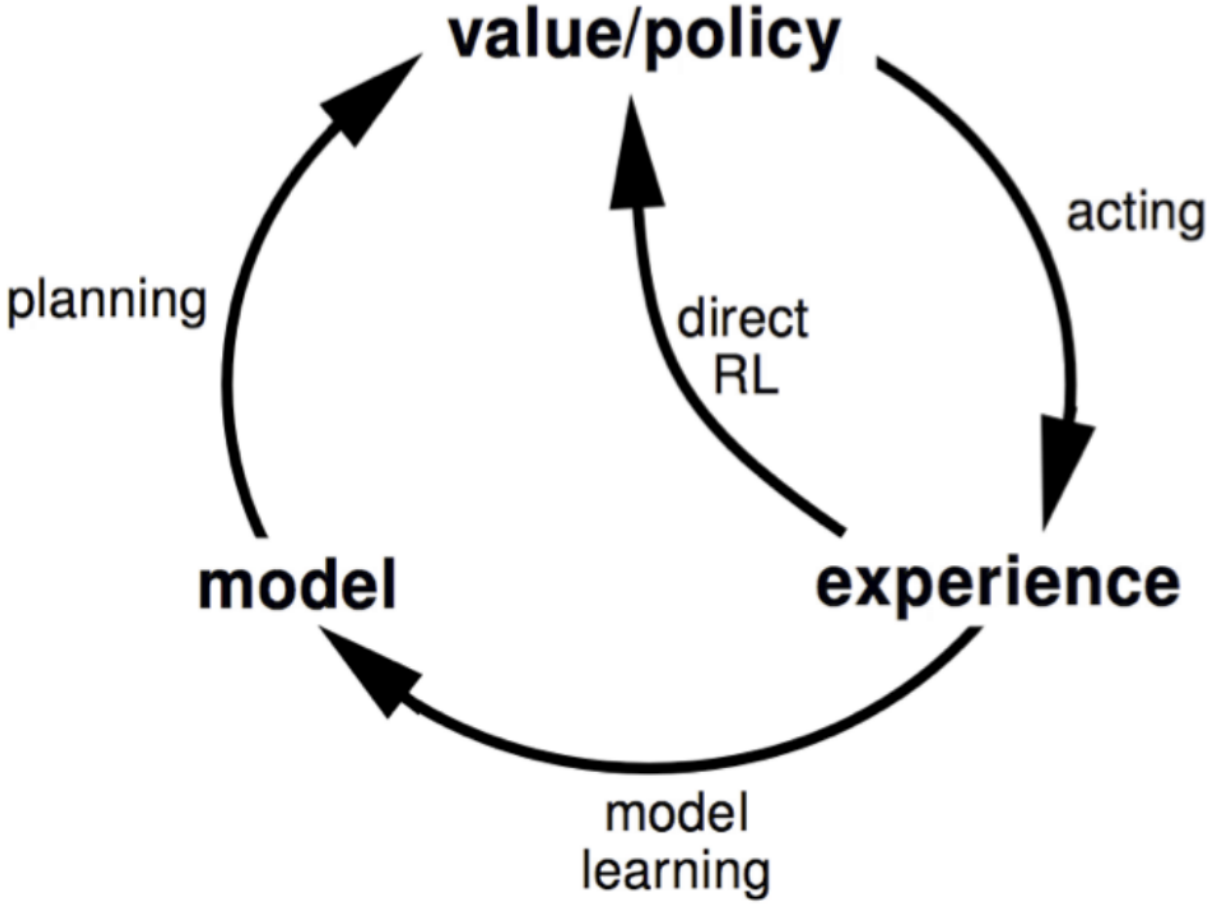


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

If simple MuJoCo environments take millions of time steps, how can we learn more complicated environments?

Model-free Model-based



- Learn Policy Direct or Learn Transition first and then policy?

Meaning of Learning Transition

- $s \rightarrow a \rightarrow s' \rightarrow a' \rightarrow s'' \rightarrow a'' \rightarrow s''' \rightarrow a''' \rightarrow s''''$

- $s \rightarrow a$
  policy: understanding how to react in an environment

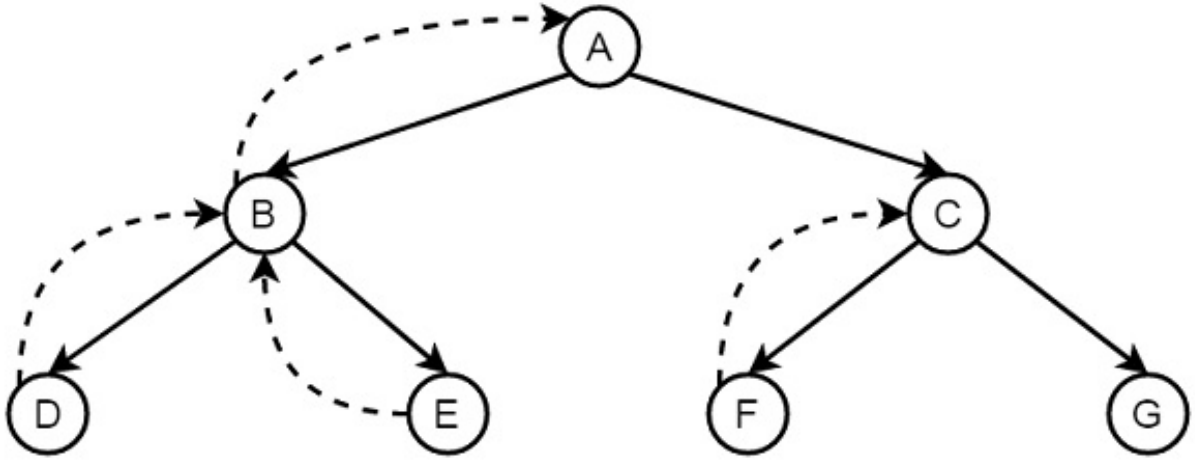
- $s \rightarrow a \rightarrow s'$
  transition: understanding an environment

Learning

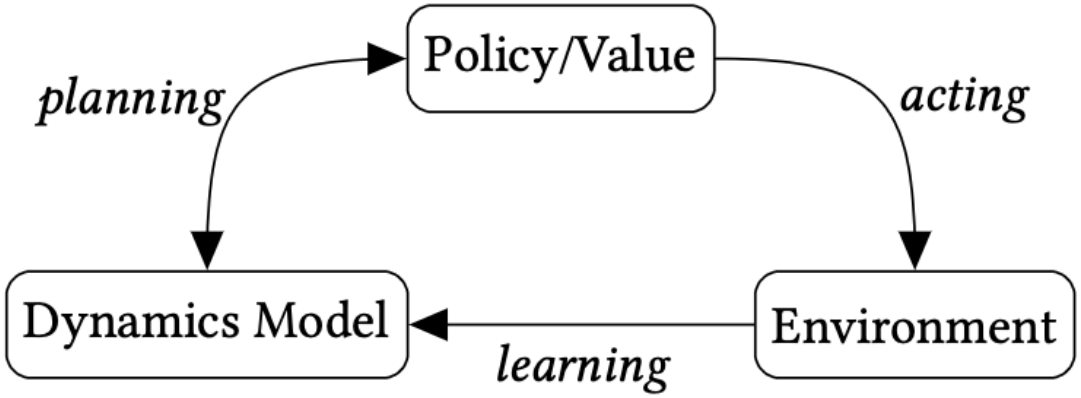
- Agent changing state in the environment
- Irreversible state change
- Forward Path
- $s \rightarrow a \rightarrow s' \rightarrow a' \rightarrow s'' \rightarrow a'' \rightarrow s''' \rightarrow a''' \rightarrow s''''$

Planning

- Agent changing own local state
- Reversible local state change
- Backtracking Tree
-



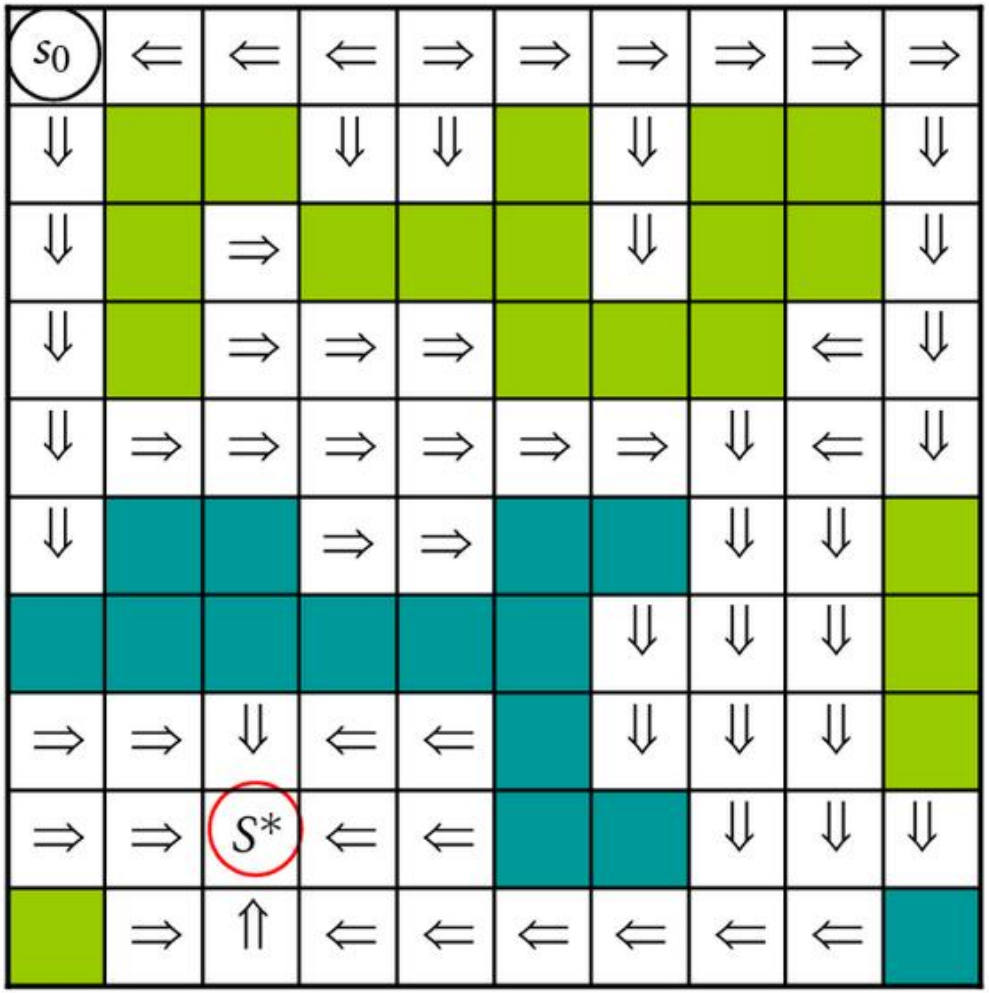
Model-based



repeat
  Sample environment E to generate data $D = (s, a, r', s')$
  Use D to learn $M = T_a(s, s'), R_a(s, s')$ ▹ learning
  for $n = 1, \ldots, N$ **do**
    Use M to update policy $\pi(s, a)$ ▹ planning
  end for
until π converges

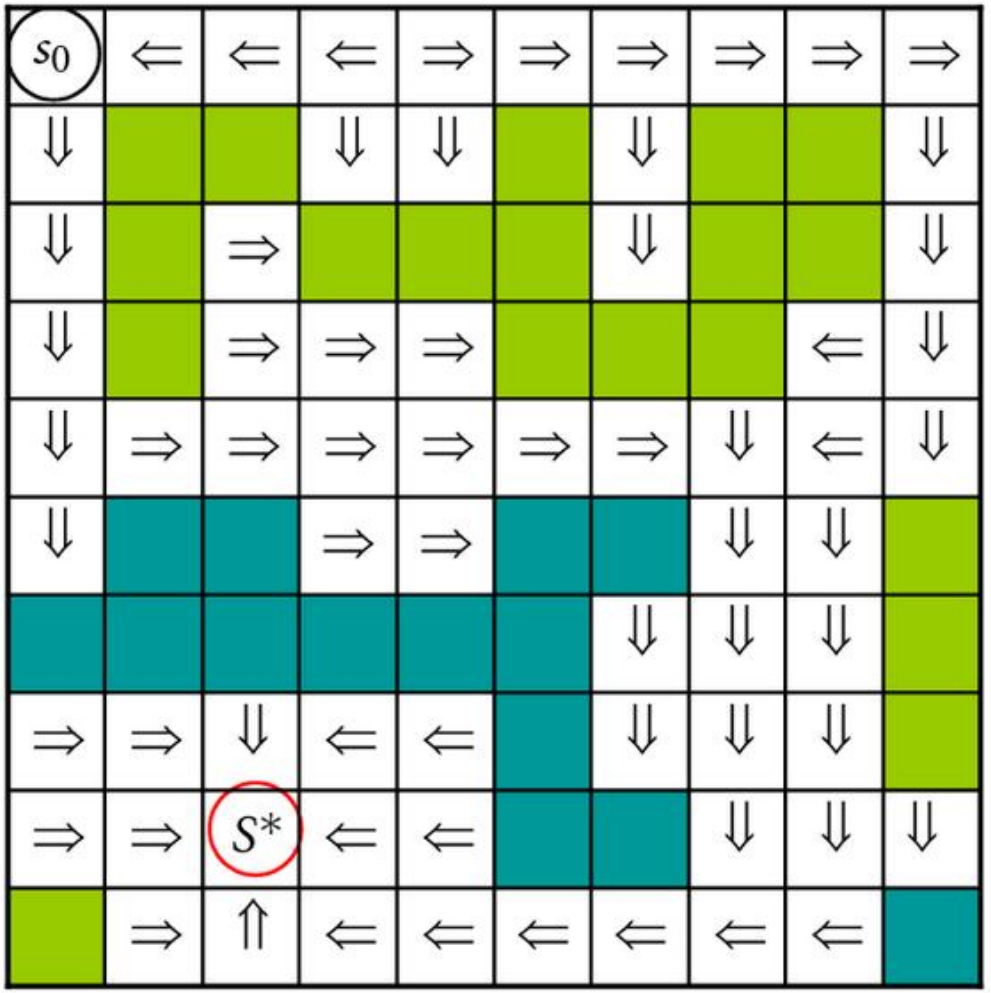
Example Model-free

- Initialize Q-function
- For All Episodes:
  - Initialize s
  - For All Time Steps in this Episode:
    - Select a ϵ-greedy from Q(s)
    - Perform a in Environment giving s' and r
    - $Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma \max_a Q(s',a)-Q(s,a)]$
    - $s \leftarrow s'$
- return Q



Example Model-based

- Initialize Q-function
- Repeat
  - Initialize s; a←π(s); (s',r)←Env(s,a) :: **Learn**
  - Q(s,a) ← Q(s,a)+α[r+γmax$_a$Q(s',$_a$)-Q(s,a)]
  - M(s,a) ← (s',r) :: **Model**
  - For n=1, …, N :
    - Select $\hat{s}$ and $\hat{a}$ randomly
    - (s',r) ← M($\hat{s}$,$\hat{a}$) :: **Plan for FREE!**
    - Q($\hat{s}$,$\hat{a}$) ← Q($\hat{s}$,$\hat{a}$)+α[r+γmax$_a$Q(s',$_a$)-Q($\hat{s}$,$\hat{a}$)]
- Until Q converges
- return Q

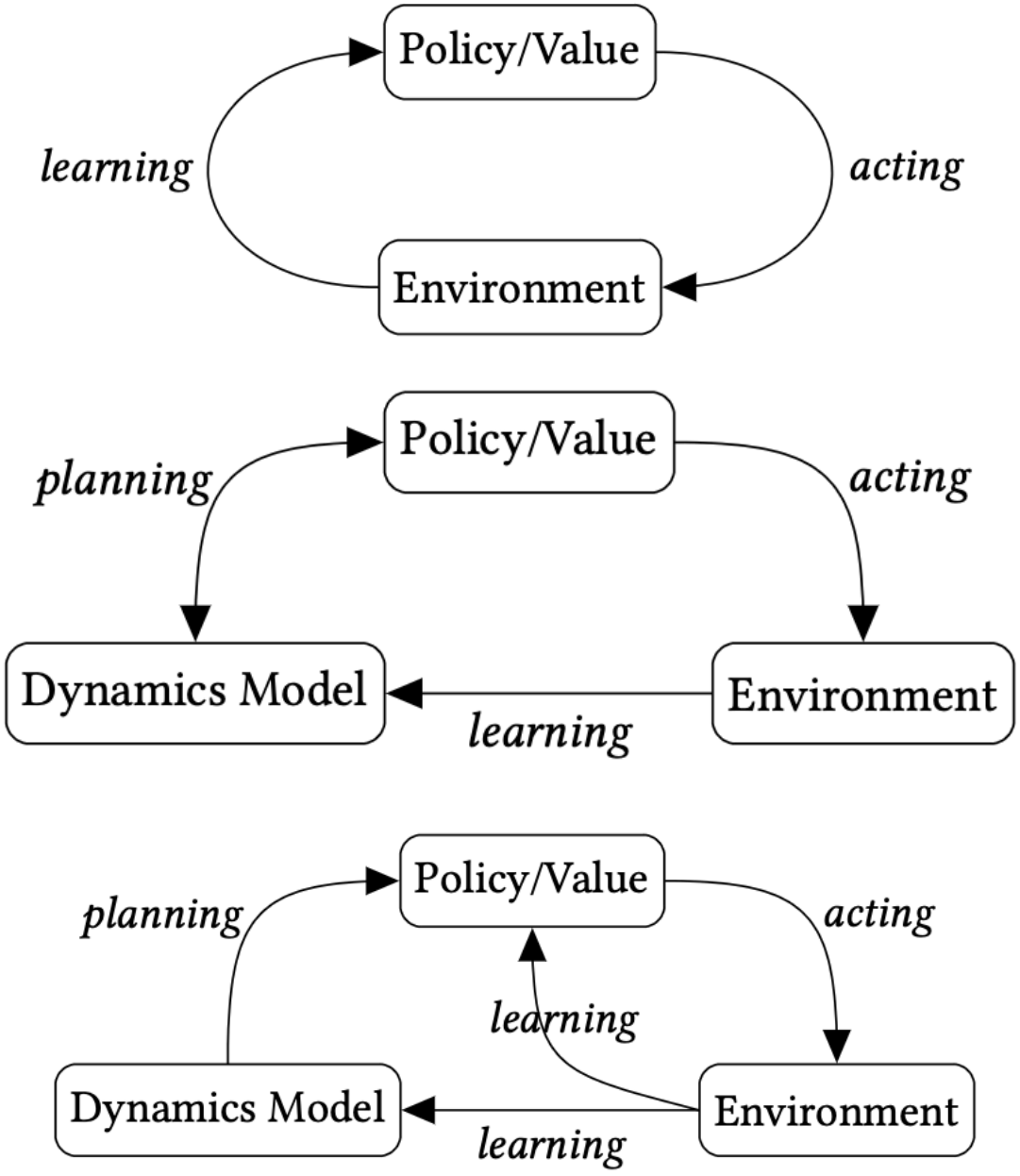


Dyna [Sutton]

- Initialize Q-function
- Repeat
  - Initialize s; a←π(s); (s',r)←Env(s,a) :: **Learn**
  - Q(s,a) ← Q(s,a)+α[r+γmax$_a$Q(s',$_a$)-Q(s,a)]
  - M(s,a) ← (s',r) :: **Model**
  - For n=1, …, N :
    - Select $\hat{s}$ and $\hat{a}$ randomly
    - (s',r) ← M($\hat{s}$,$\hat{a}$) :: **Plan for FREE!**
    - Q($\hat{s}$,$\hat{a}$) ← Q($\hat{s}$,$\hat{a}$)+α[r+γmax$_a$Q(s',$_a$)-Q($\hat{s}$,$\hat{a}$)]
- Until Q converges
- return Q

Dyna: Learning & Planning

- Learning
- Planning



Sample Complexity

- Model based reduces sample complexity
- As soon as Model has enough transition entries, the policy can be learned from the Model, for free
- This free learning is called planning. It does not involve environment samples, hence, “free”

Why “Planning”?

- Internal to Agent
- Agent has state
- Agent can undo; allows to retrace your steps
- Agent can backtrack to try another action
- Planning is reversible learning
- Learning changes Environment state that the Agent cannot undo

Imperfect Models

However

- Free planning concept only works if Model is perfect
- When Environment is high dimensional, it will never be fully sampled, so Model will be imperfect

Trade-off

- Learning T to **reduce** Env samples
- T is a function, a neural network
- With high dim problems, T is high capacity, so needs **many** samples to prevent overfitting
- Trade-off sample complexity vs quality of model

Dealing with Imperfect Models

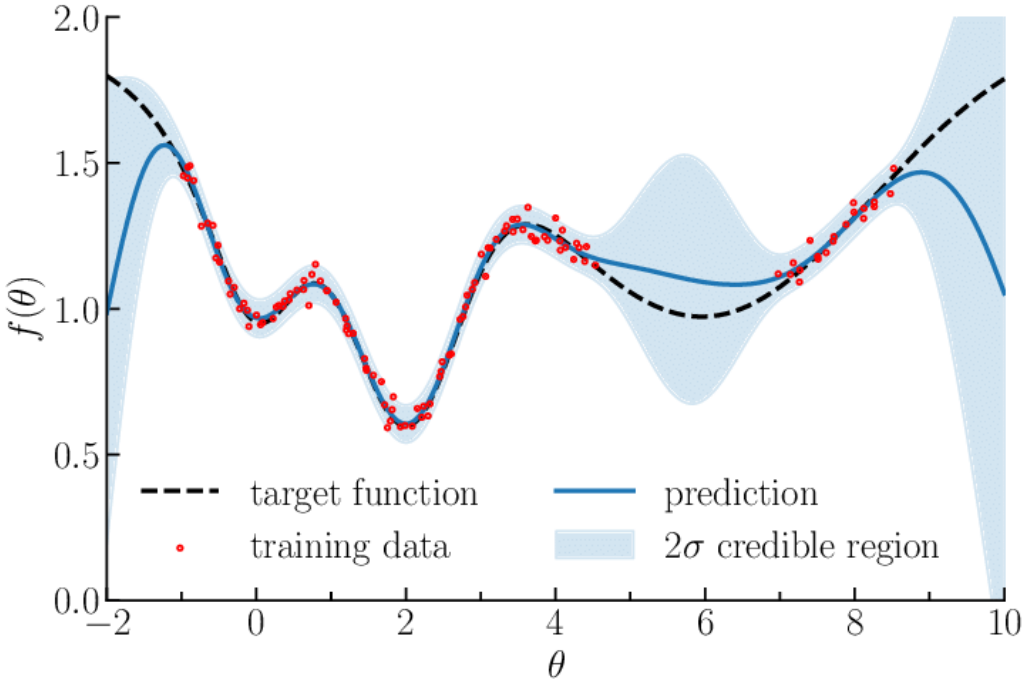
- Improving Model Learning
- Improving Planning with Imperfect Model

Improving Model Learning

- Modeling Uncertainty
- Latent Models

Modeling Uncertainty

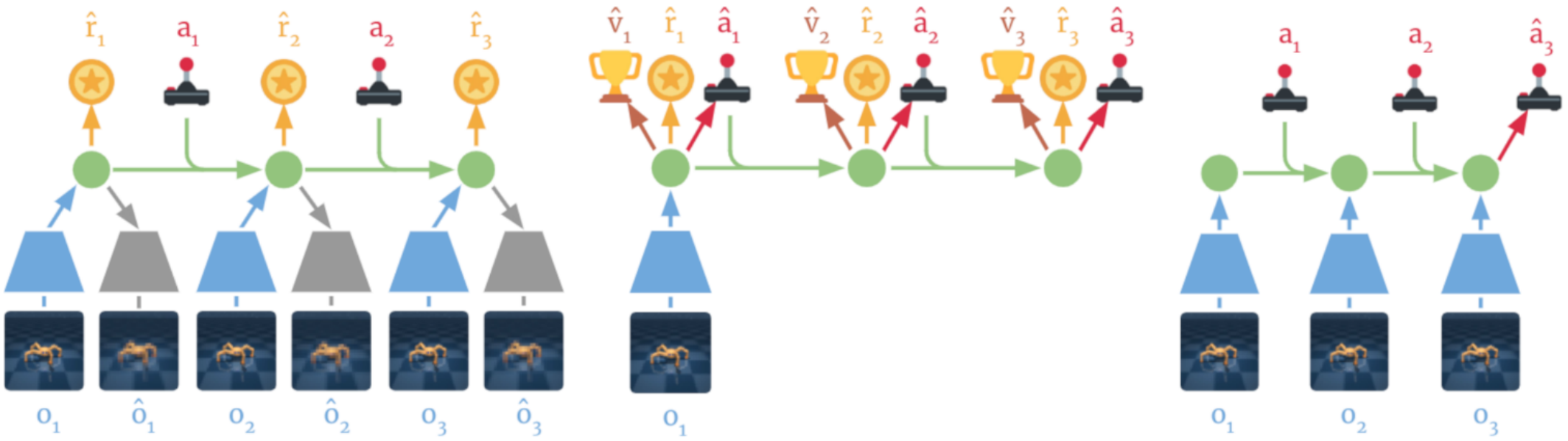
- Gaussian Processes
  - PILCO, GPS, SVG
  - Works, but Computationally expensive
- Ensembles
  - PETS



- Knowing uncertainty allows better planning
- Do Planning sampling from distribution, plan with locally-linear search or with stochastic trajectory optimizer
- Does not scale to high dimensional problems

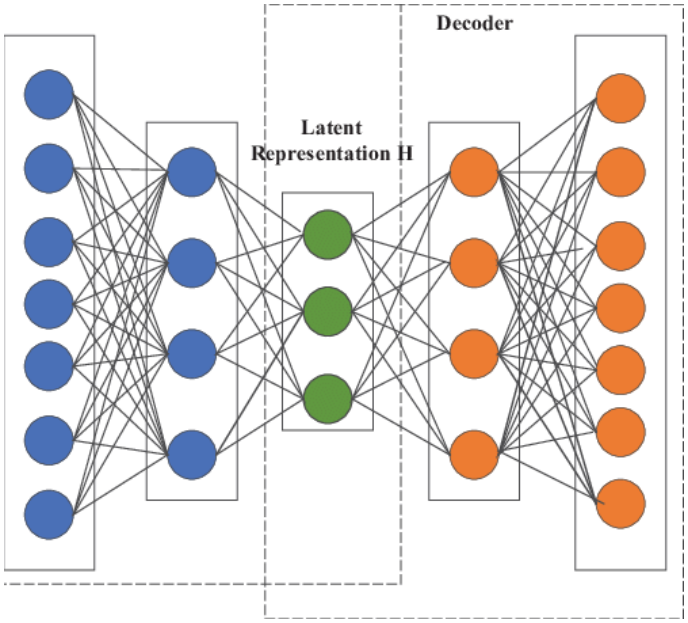
Latent Model

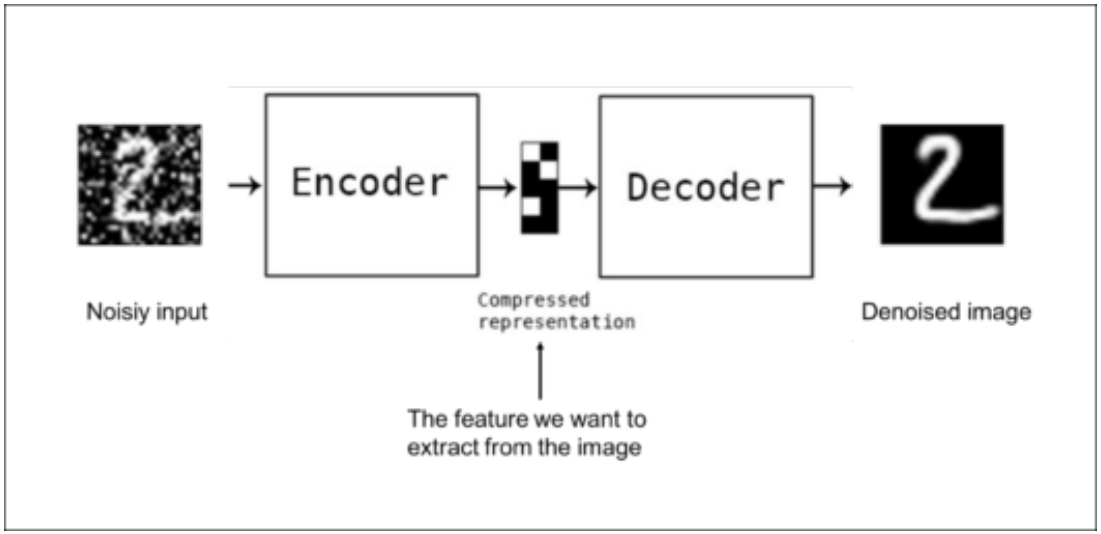
- Compress Observation Space into (smaller) Latent Space by modeling on value prediction
- Plan in small Latent Space [Dreamer]



Latent Model

- Compression: autoencoder
- Many networks, Complicated architecture, Good performance
- PlaNet, Dreamer, VPN





How to Plan with a weak Model

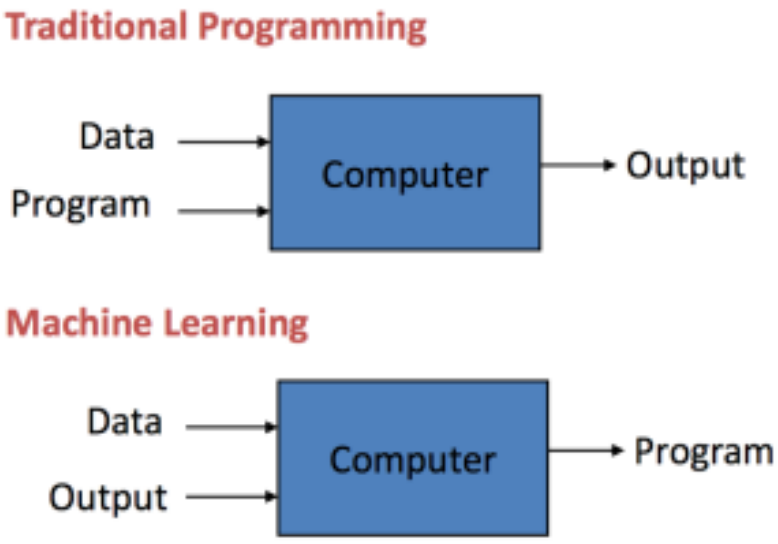
- Trajectory Rollouts and Model-Predictive Control
- End-to-end learning and planning

Trajectory MPC

- Short trajectory rollouts
  - reduce lookahead depth
  - splits rollouts in near future (planned) and far-future (model-free) -> MVE
- Model-predictive Control
  Decision-time planning
  - Highly non-linear function are often locally linear
  - MPC: optimize model over limited time, and re-learn -> PETS

End-to-end learning/ planning

- Trend for programming by example
- Can a neural network do planning? (with backtrack?)
- Learn differentiable planning



Value Iteration

Initialize V(s) to arbitrary values

Repeat until V(s) converge

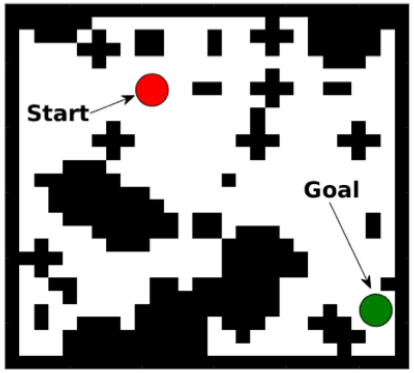
For all states

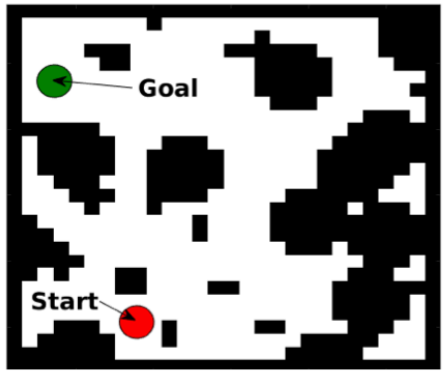
For all actions

$$Q(s,a) \leftarrow \sum_{s} P_{ss'}^{a}(r(s,a) + \gamma V(s'))$$

$$V(s) \leftarrow \max_{a} Q(s,a)$$

End-to-end learning/ planning



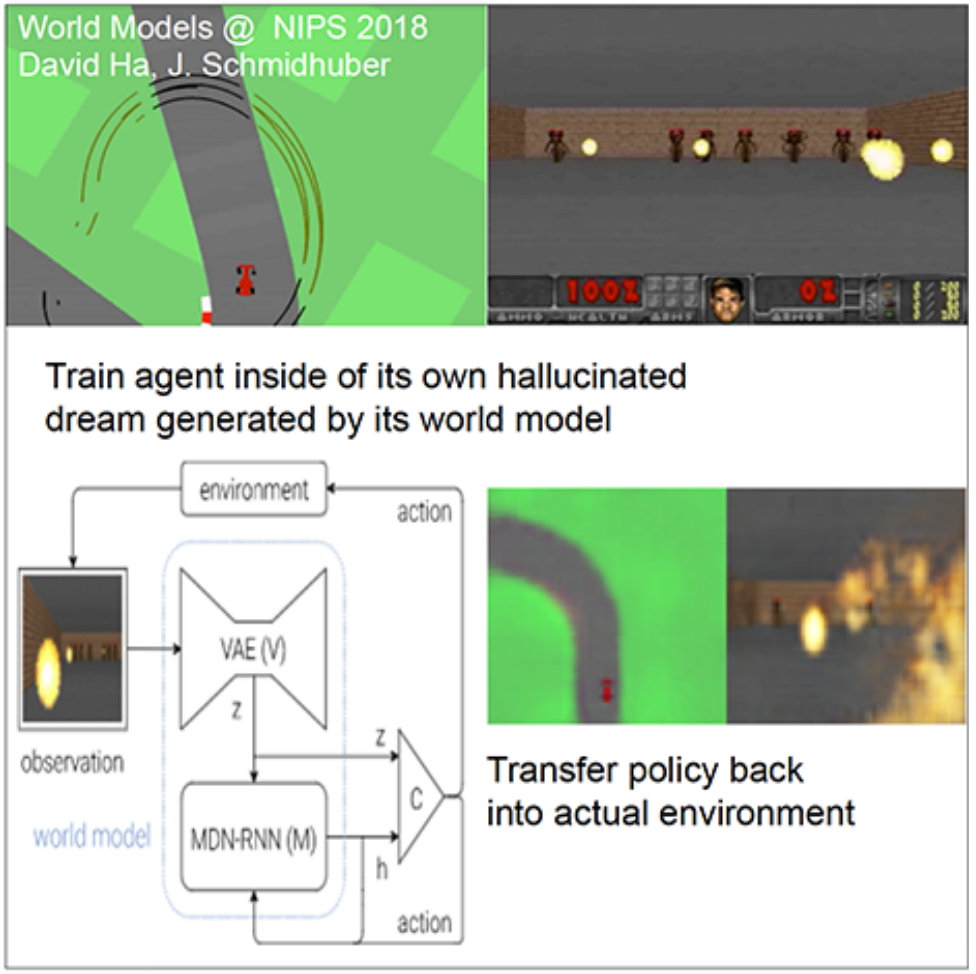


- Value Iteration Network
- Each layer one step of Value Iteration
- Learn $\sum_{s' \in S} T_a(s, s')(R_a(s, s') + \gamma V[s'])$
- Learn different Mazes

Our main observation is that each iteration of VI may be seen as passing the previous value function Vn and reward function R through a convolution layer and max-pooling layer. In this analogy, each channel in the convolution layer corresponds to the Q-function for a specific action, and convolution kernel weights correspond to the discounted transition probabilities.

End-to-end learning/ planning

- Later: RNN/LSTM for state: VProp
- Latent & End-to-end: TreeQN, Predictron, MuZero, I2A, World Model
- Elaborate, Complex systems



Diagram

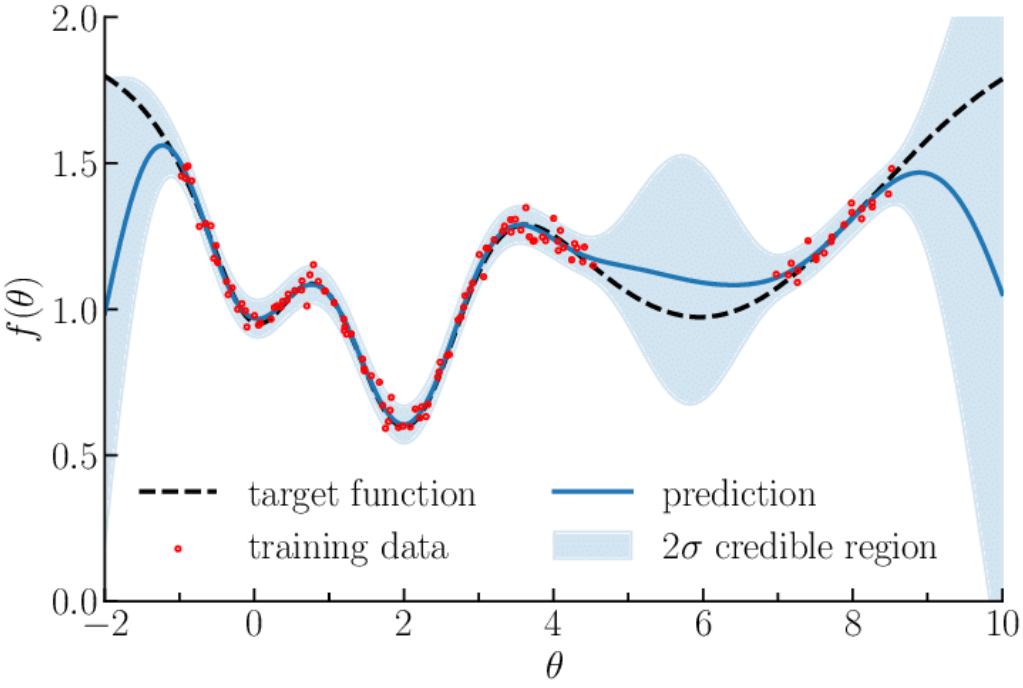
		LEARNING	
		Uncertainty Ensembles CNN	**Latent Models**
PLANNING	**Trajectory Model-predictive control**	PILCO GPS SVG Local PETS MVE Meta	Dreamer Plan2Explore L3P VPN SimPLe Dreamer-v2
	End-to-end	VIN VProp Network-Planning	TreeQN I2A Predictron World Model MuZero

An Overview of Model-Based Approaches

Name	Learning	Planning	Environment
PILCO	Uncertainty	Trajectory	Pendulum
iLQG	Uncertainty	MPC	Small
GPS	Uncertainty	Trajectory	Small
SVG	Uncertainty	Trajectory	Small
VIN	CNN	e2e	Mazes
VProp	CNN	e2e	Mazes
Planning	CNN/LSTM	e2e	Mazes
TreeQN	Latent	e2e	Mazes
I2A	Latent	e2e	Mazes
Predictron	Latent	e2e	Mazes
World Model	Latent	e2e	Car Racing
Local Model	Uncertainty	Trajectory	MuJoCo
Visual Foresight	Video Prediction	MPC	Manipulation
PETS	Ensemble	MPC	MuJoCo
MVE	Ensemble	Trajectory	MuJoCo
Meta Policy	Ensemble	Trajectory	MuJoCo
Policy Optim	Ensemble	Trajectory	MuJoCo
PlaNet	Latent	MPC	MuJoCo
Dreamer	Latent	Trajectory	MuJoCo
Plan2Explore	Latent	Trajectory	MuJoCo
L^3P	Latent	Trajectory	MuJoCo
Video-prediction	Latent	Trajectory	Atari
VPN	Latent	Trajectory	Atari
SimPLe	Latent	Trajectory	Atari
Dreamer-v2	Latent	Trajectory	Atari
MuZero	Latent	e2e/MCTS	Atari/Go

PILCO uncertainty/ trajectory

- Gaussian Processes
- Computationally Expensive



PETS ensemble/mpc

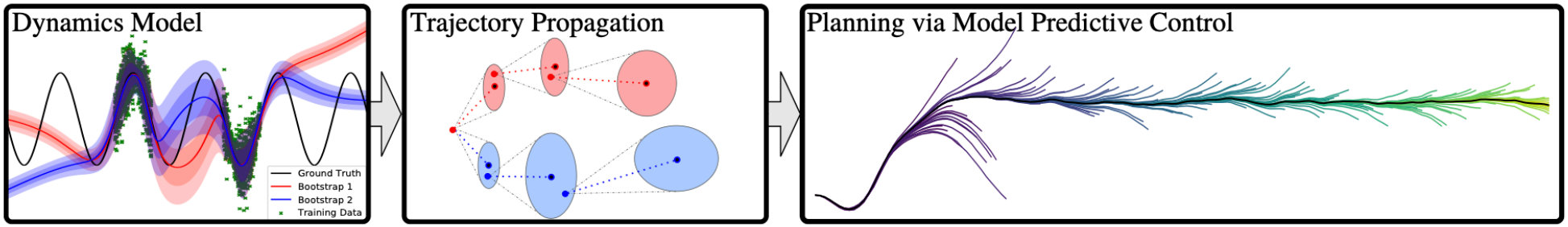
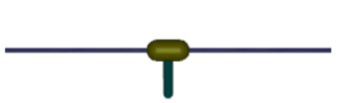
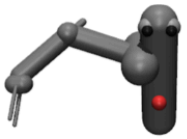


Figure 1: Our method (PE-TS): **Model**: Our probabilistic ensemble (PE) dynamics model is shown as an ensemble of two bootstraps (bootstrap disagreement far from data captures epistemic uncertainty: our subjective uncertainty due to a lack of data), each a probabilistic neural network that captures aleatoric uncertainty (inherent variance of the observed data). **Propagation**: Our trajectory sampling (TS) propagation technique uses our dynamics model to re-sample each particle (with associated bootstrap) according to its probabilistic prediction at each point in time, up until horizon T. **Planning**: At each time step, our MPC algorithm computes an optimal action sequence, applies the first action in the sequence, and repeats until the task-horizon.

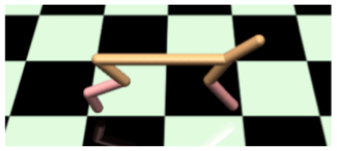


(a) Cartpole

(b) 7-dof Pusher



(c) 7-dof Reacher



(d) Half-cheetah

Value Prediction Network latent/traj

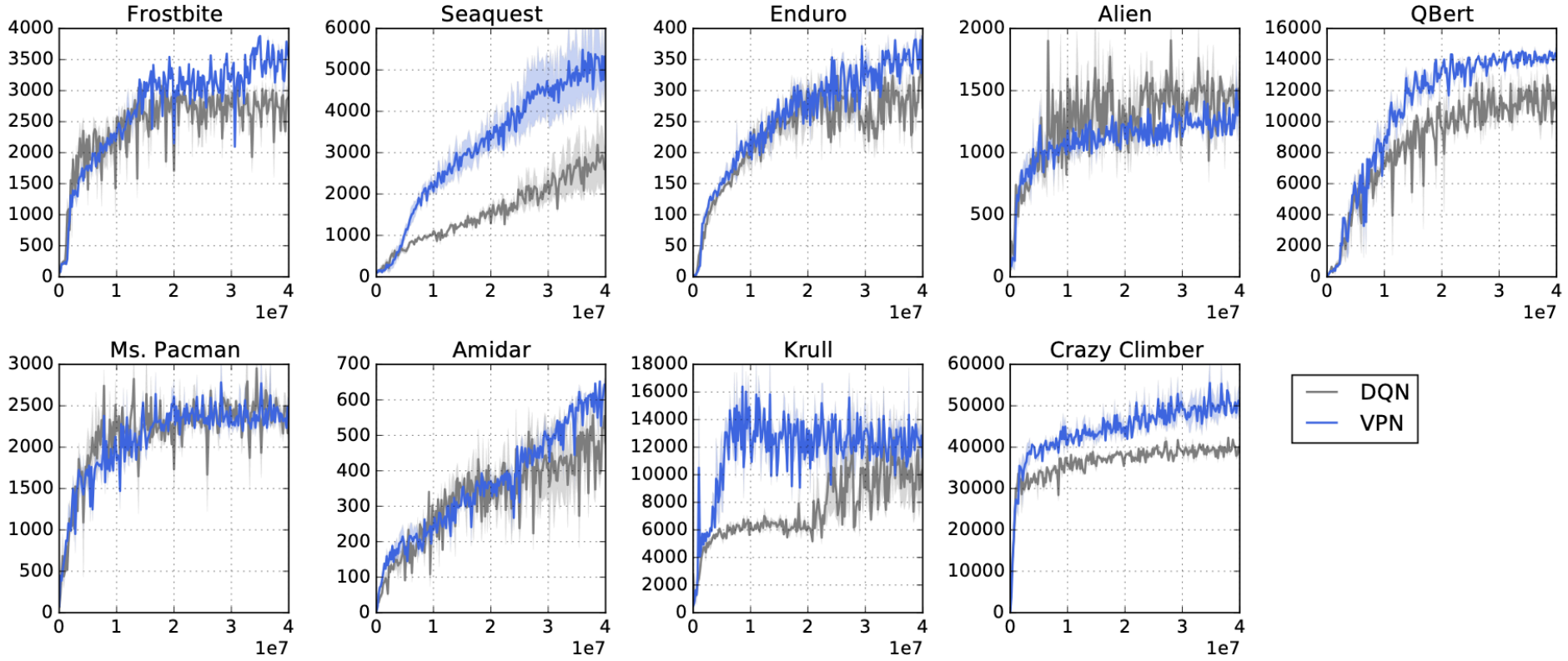
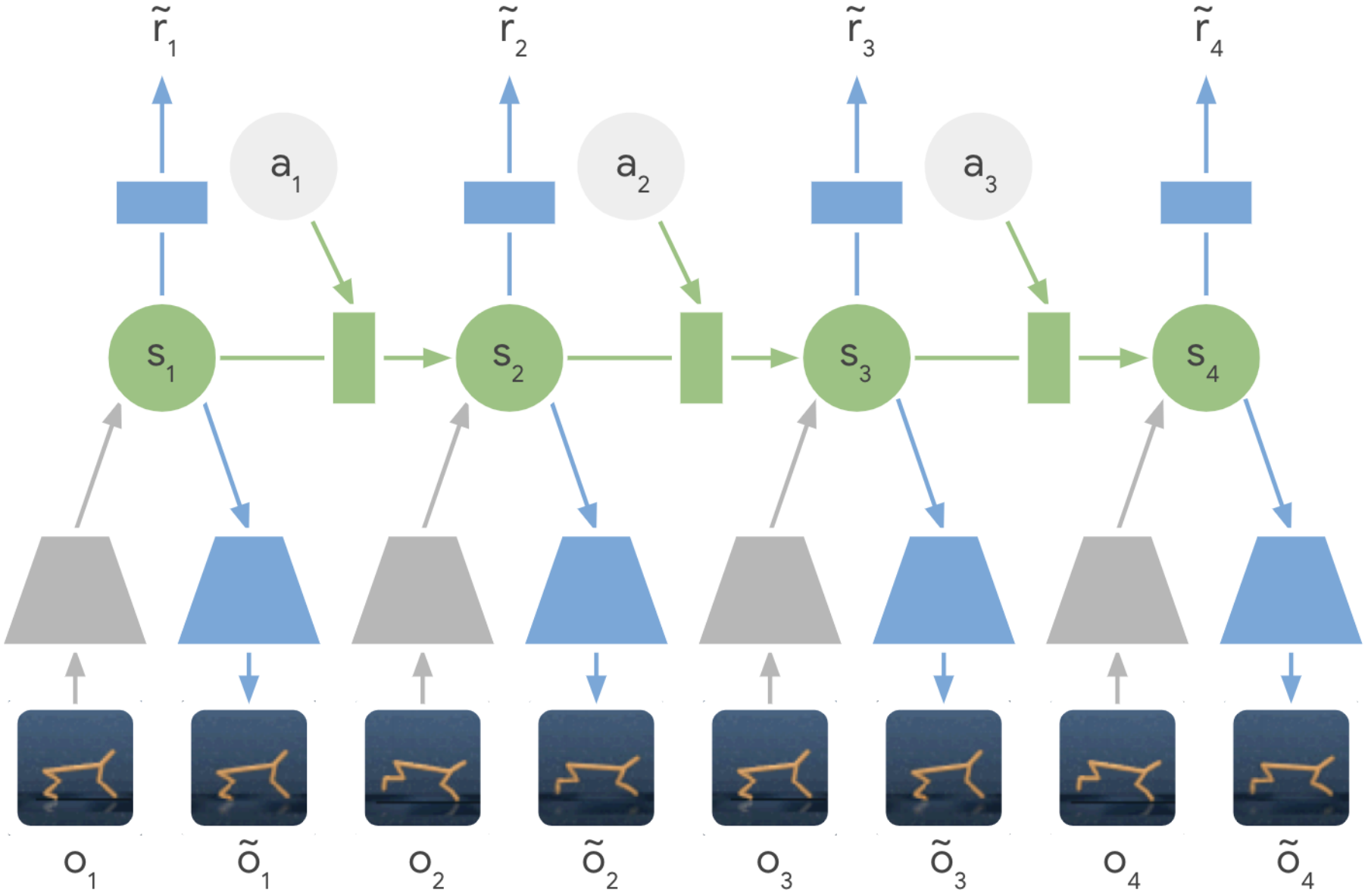
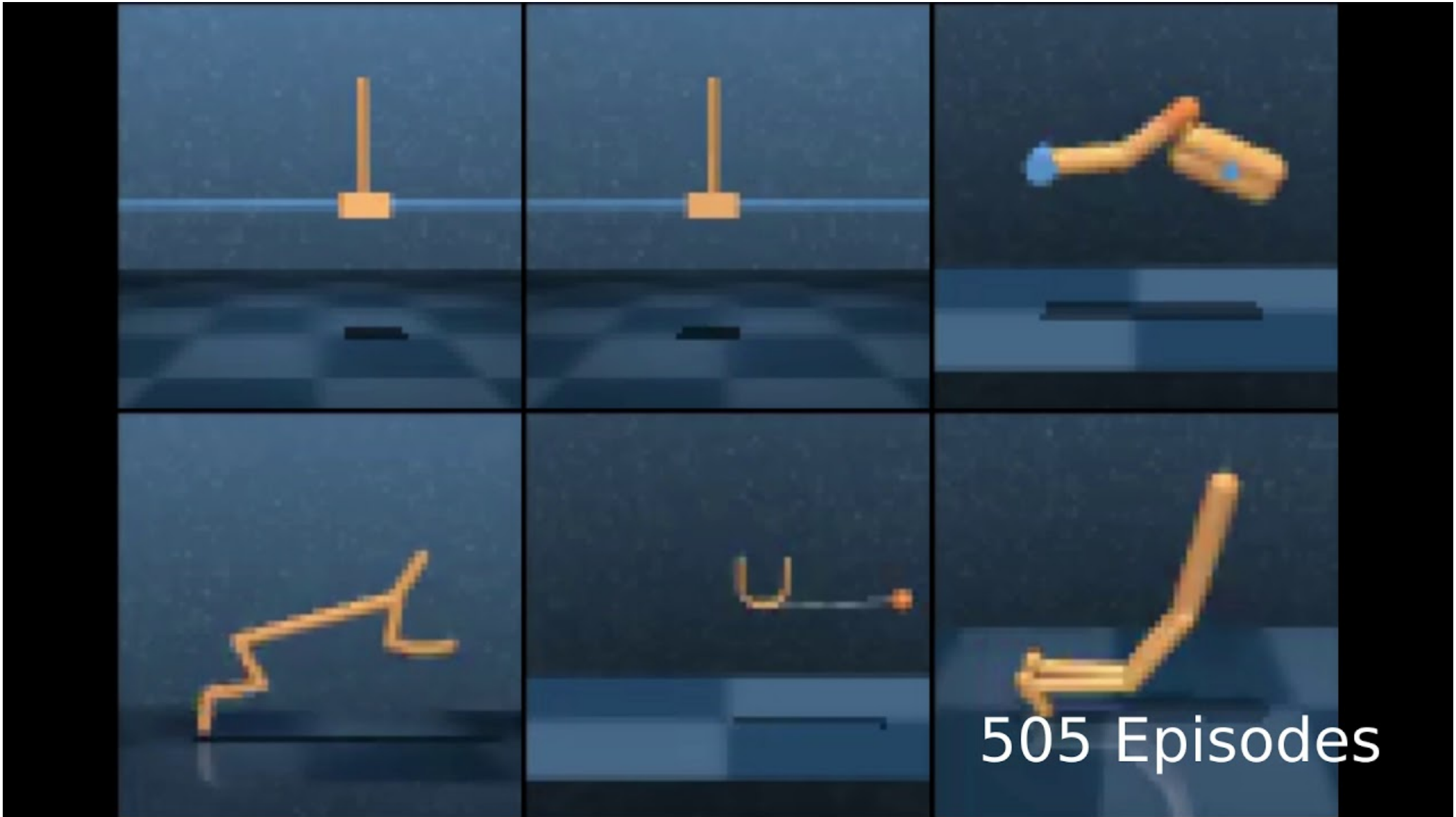


Figure 8: Learning curves on Atari games. X-axis and y-axis correspond to steps and average reward over 100 episodes respectively.

Latent PlaNet



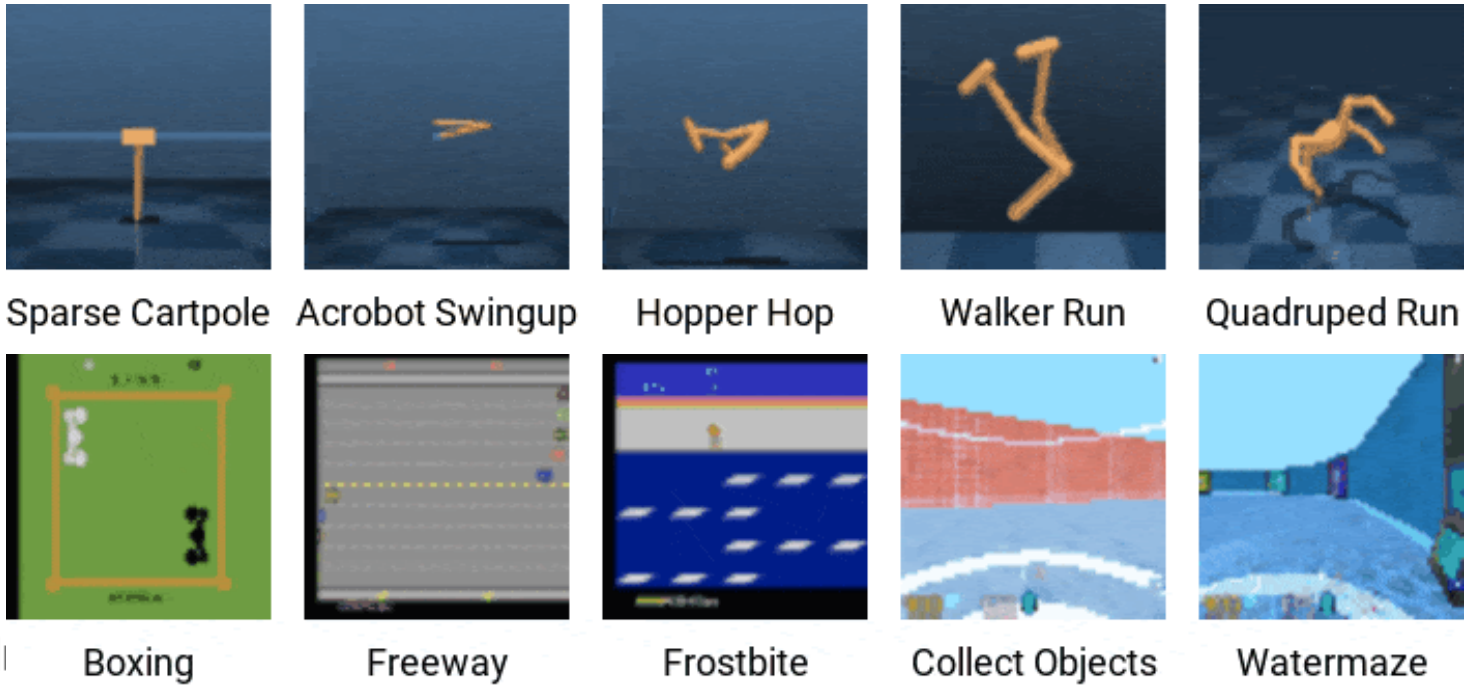
PlaNet latent/mpc

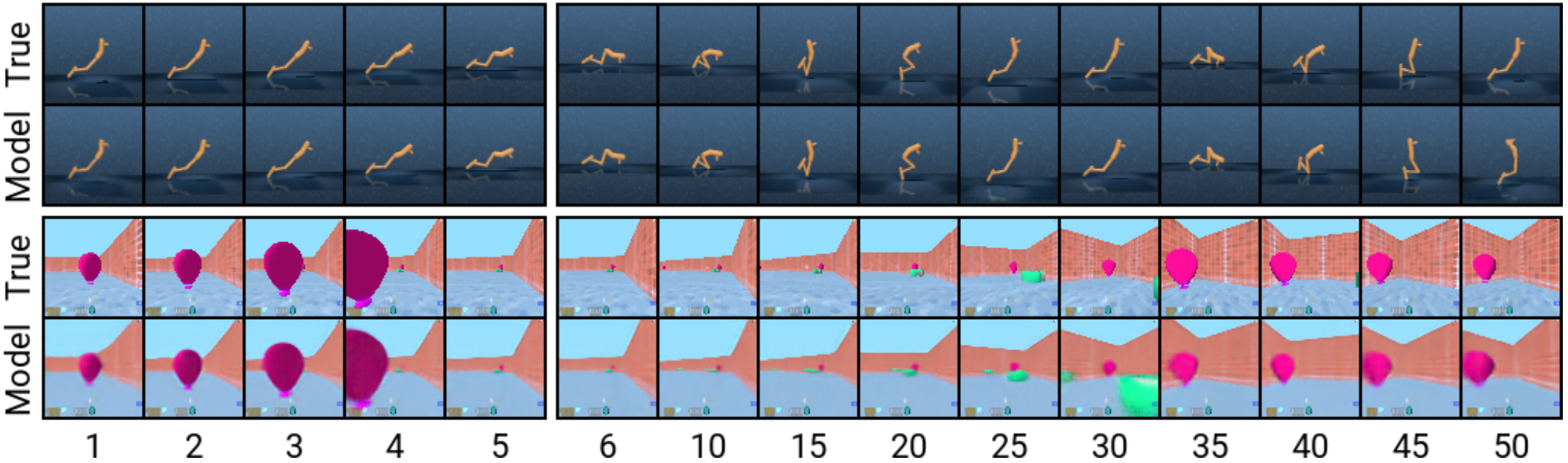


Dreamer latent/traj

The three processes of the Dreamer agent. The world model is learned from past experience. From predictions of this model, the agent then learns a value network to predict future rewards and an actor network to select actions. The actor network is used to interact with the environment.

Dreamer latent/traj





I2A latent/e2e

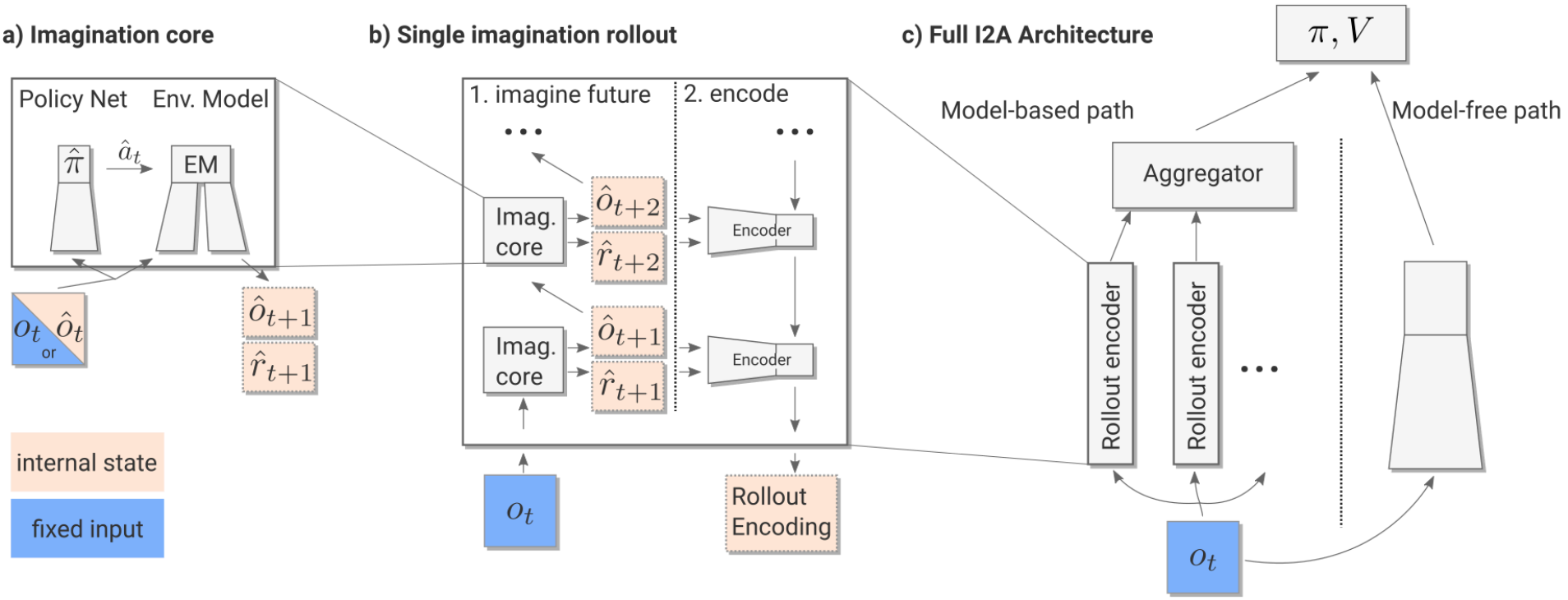


Figure 1: *I2A architecture*. $\hat{\cdot}$ notation indicates imagined quantities. *a)*: the imagination core (IC) predicts the next time step conditioned on an action sampled from the rollout policy $\hat{\pi}$. *b)*: the IC imagines trajectories of features $\hat{f} = (\hat{o}, \hat{r})$, encoded by the rollout encoder. *c)*: in the full I2A, aggregated rollout encodings and input from a model-free path determine the output policy π.

I2A latent/e2e

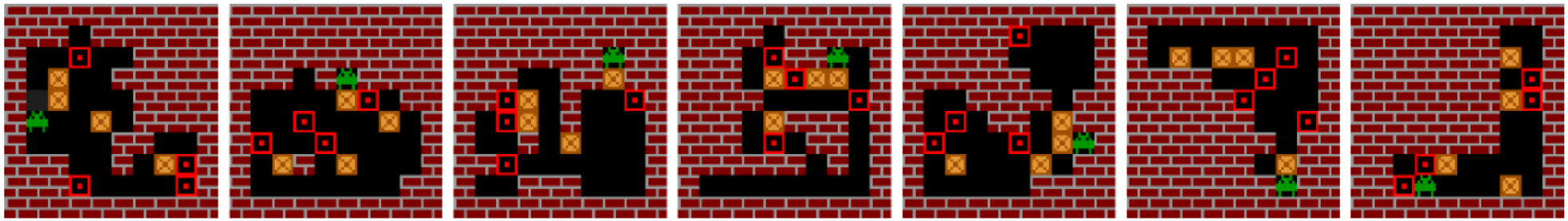


Figure 3: *Random examples of procedurally generated Sokoban levels.* The player (green sprite) needs to push all 4 boxes onto the red target squares to solve a level, while avoiding irreversible mistakes. Our agents receive sprite graphics (shown above) as observations.

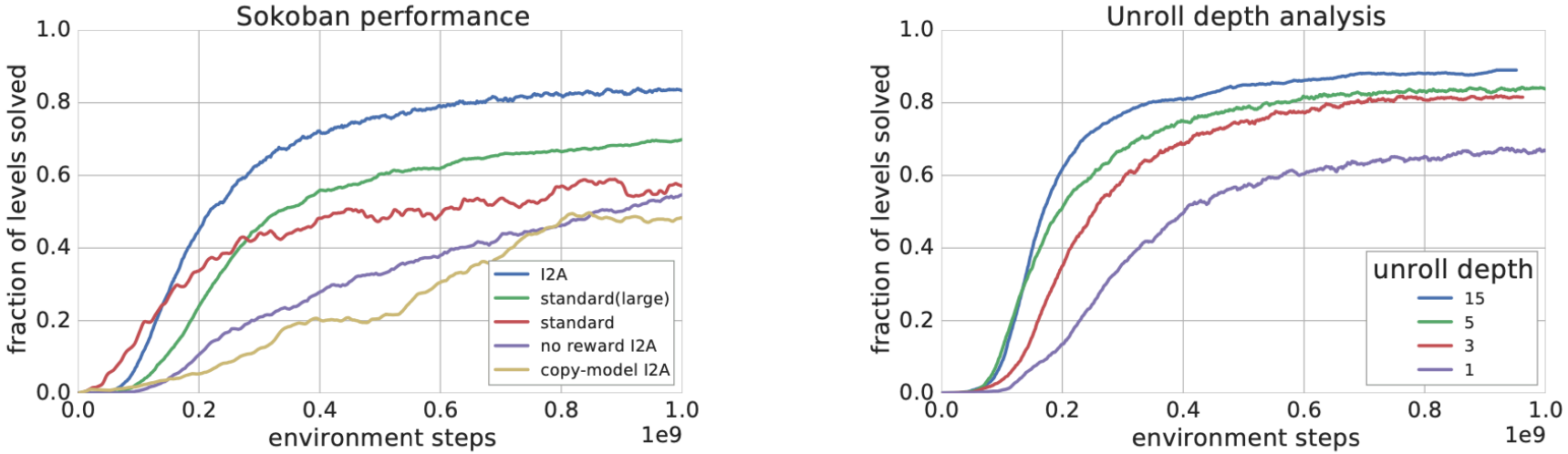
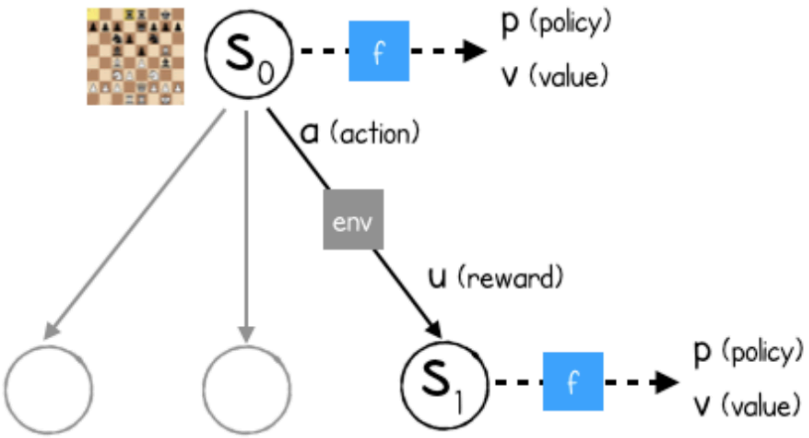


Figure 4: *Sokoban learning curves. Left:* training curves of I2A and baselines. Note that I2A use additional environment observations to pretrain the environment model, see main text for discussion. *Right:* I2A training curves for various values of imagination depth.

MuZero latent/e2e

MuZero latent/e2e

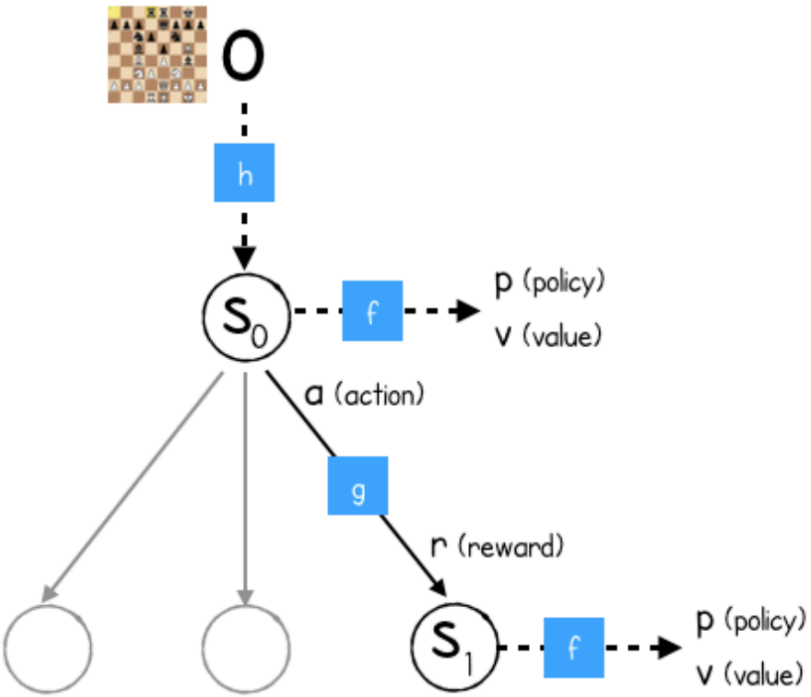
AlphaZero



AlphaZero has 1 network

		from	to
prediction	f:	s	p, v

MuZero



MuZero has 3 networks

		from	to
prediction	f:	s	p, v
dynamics	g:	s, a	r, s
representation	h:	o	s

MuZero latent/e2e

representation $h_\theta(o_1, .., o_t) = s^0$

prediction $f_\theta(s^k) = \mathbf{p}^k, v^k$

dynamics $g_\theta(s^{k-1}, a^k) = r^k, s^k$

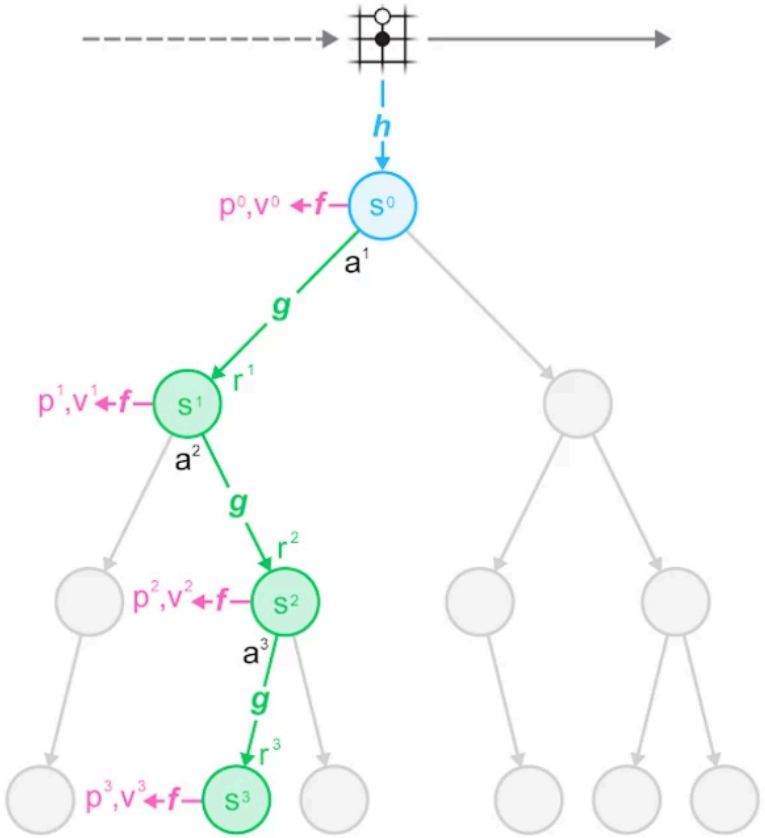
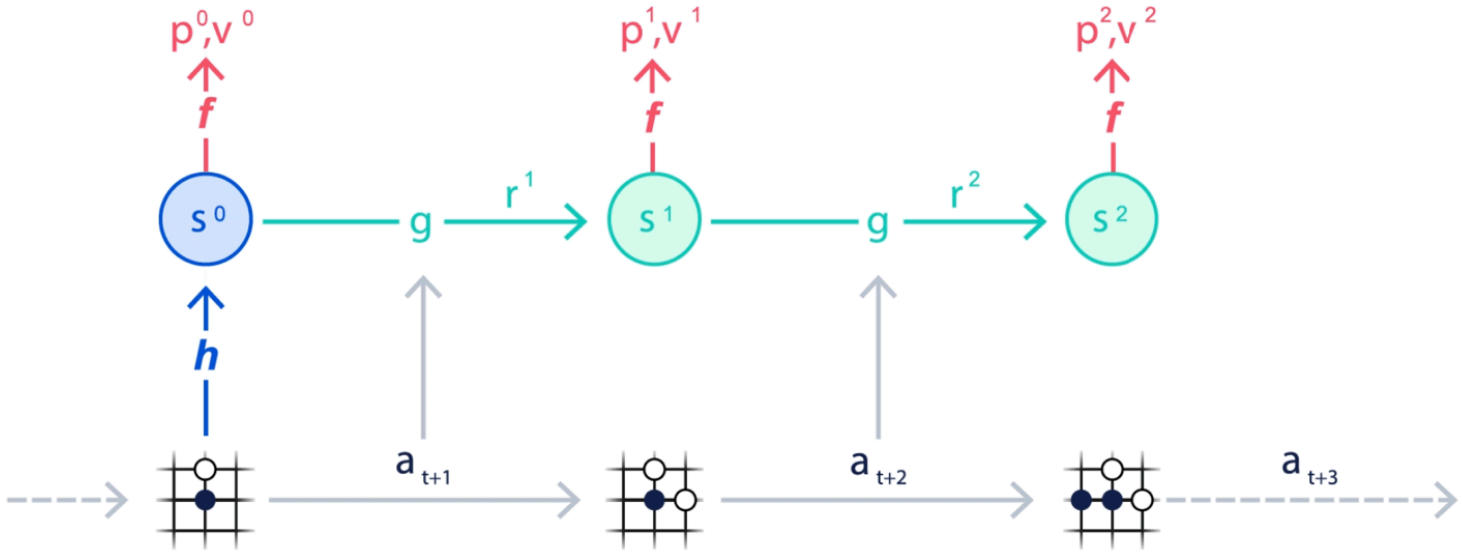


Illustration of how Monte Carlo Tree Search can be used to plan with the MuZero neural networks. Starting at the current position in the game (schematic Go board at the top of the animation), MuZero uses the representation function (h) to map from the observation to an embedding used by the neural network (s0). Using the dynamics function (g) and the prediction function (f), MuZero can then consider possible future sequences of actions (a), and choose the best action.

MuZero latent/e2e



During training, the model is unrolled alongside the collected experience, at each step predicting the previously saved information: the value function v predicts the sum of observed rewards (u), the policy estimate (p) predicts the previous search outcome (π), the reward estimate r predicts the last observed reward (u).

Benchmark

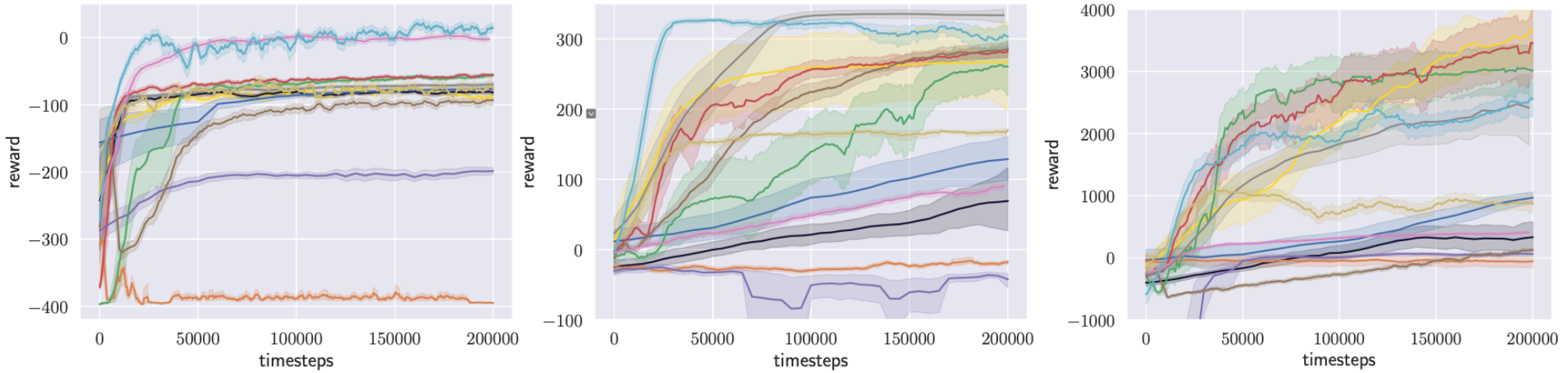


Figure 1: A subset of all 18 performance curve figures of the bench-marked algorithms. All the algorithms are run for 200k time-steps and with 4 random seeds. The remaining figures are in appendix C.

- SAC and TD3 are model-free baselines
- Model-based is sometimes better

Conclusion

- What does the overview tell us?
- Accuracy of transition model: crucial
- Sample complexity trade-off: success
- Results sometimes better than model-free
- Still brittleness, sensitive to hyperparameters
- Still active field of research

Questions?