Model-Based Methods in Reinforcement Learning

Part 1: Introduction & Learning Models

Igor Mordatch (Google) & Jessica Hamrick (DeepMind)

ICML 2020 Tutorial

Prerequisites

Deep Learning

Gradient-based optimization
MLPs
RNNs / LSTMs
CNNs
GNNs

Model-Free RL

MDPs
Q-learning
Policy gradient

Outline

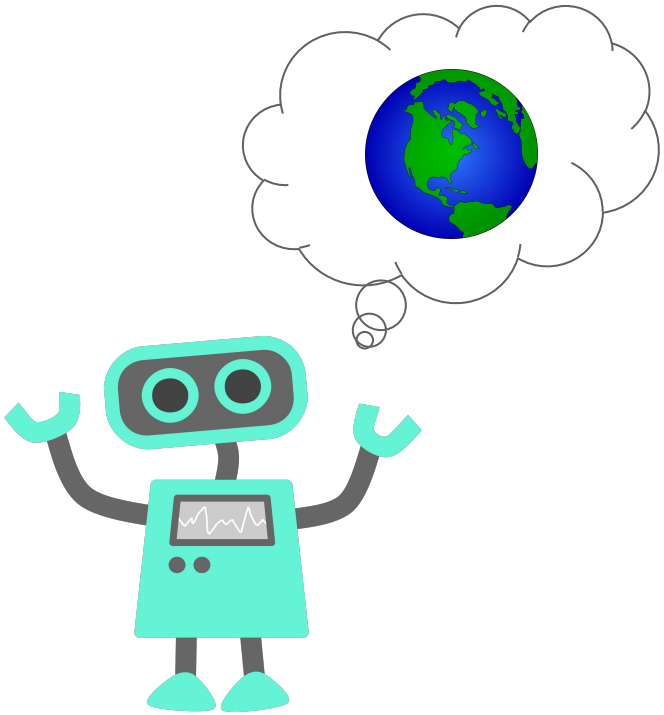
1. Introduction and motivation *(Jess)*
2. Problem statement *(Jess)*
3. What is a “model”? *(Jess)*
4. What is model-based control? *(Igor)*
5. Model-based control in the loop *(Igor)*
6. What else can models be used for? *(Jess)*
7. What’s missing from model-based methods? *(Jess)*
8. Conclusion *(Jess)*

Outline

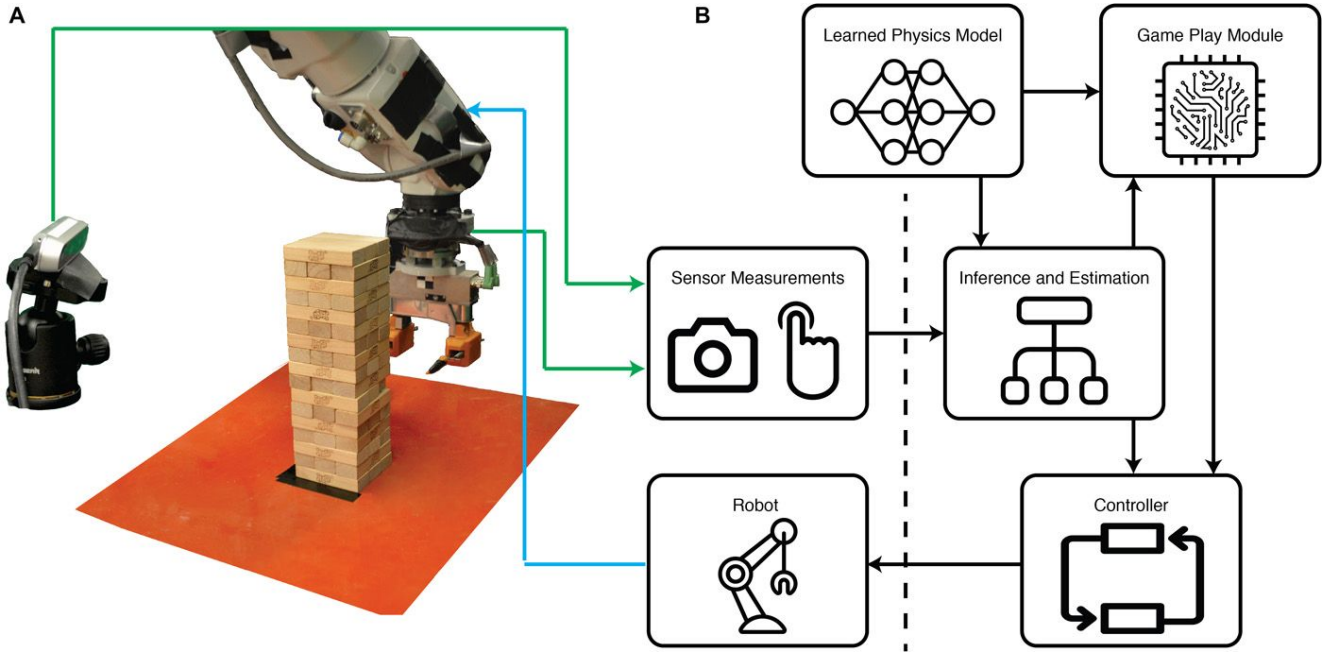
1. **Introduction and motivation**
2. Problem statement
3. What is a “model”?
4. What is model-based control?
5. Model-based control in the loop
6. What else can models be used for?
7. What’s missing from model-based methods?
8. Conclusion

“If the organism carries a **‘small-scale model’ of external reality** *and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilise the knowledge of past events in dealing with the present and future, and in every way to react in a much* **fuller, safer, and more competent** *manner to the emergencies which face it.”*

—Craik, 1943, *The Nature of Explanation*

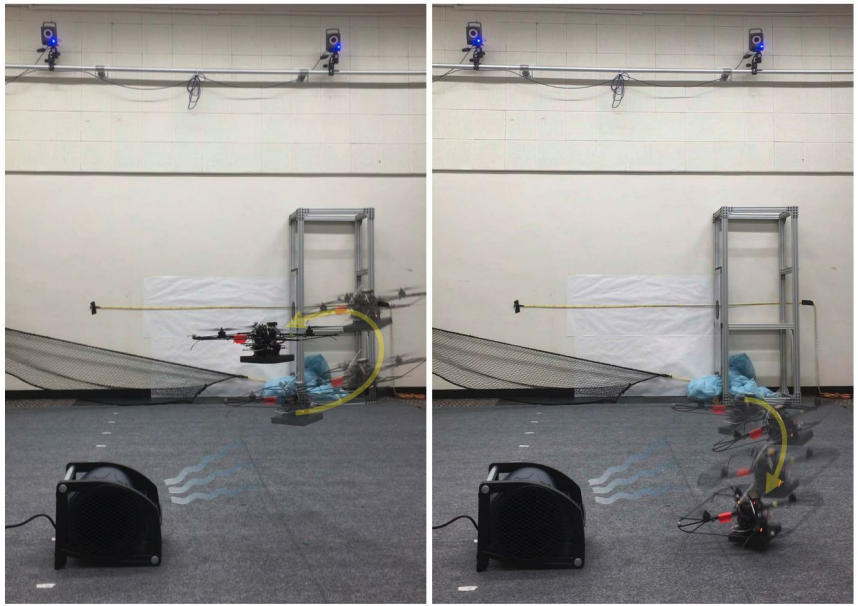


Model-based reasoning for ***robotic control***



Fazeli et al. (2019). See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. Science Robotics, 4(26).

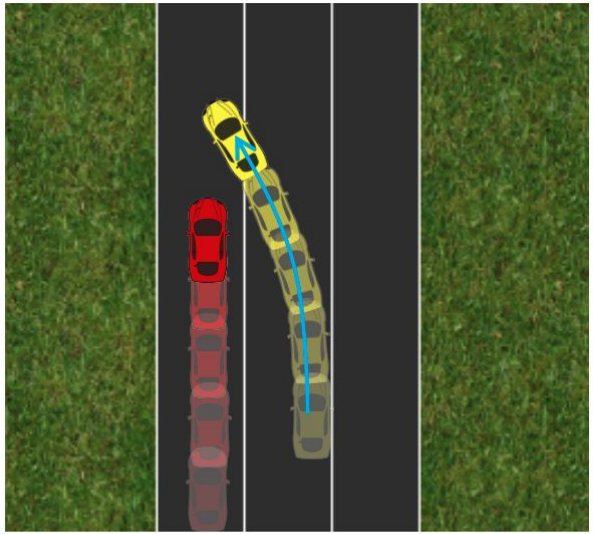
Model-based reasoning for ***safety***



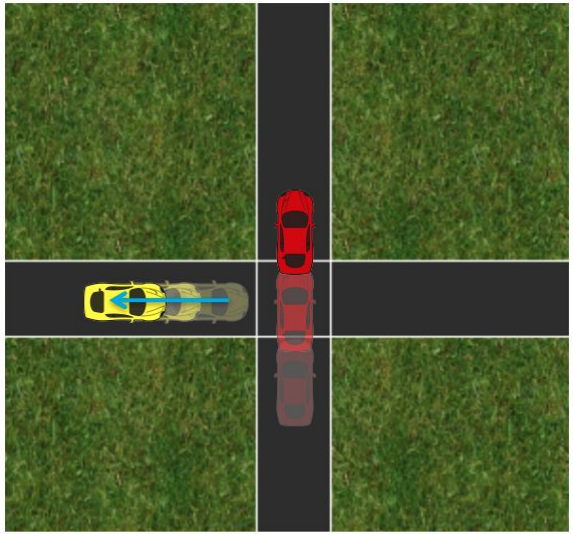
(a) With online guarantee validation (b) Without online guarantee validation

Fisac et al. (2019). A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems. IEEE Transactions on Automatic Control.

Model-based reasoning for ***human-AI interaction***



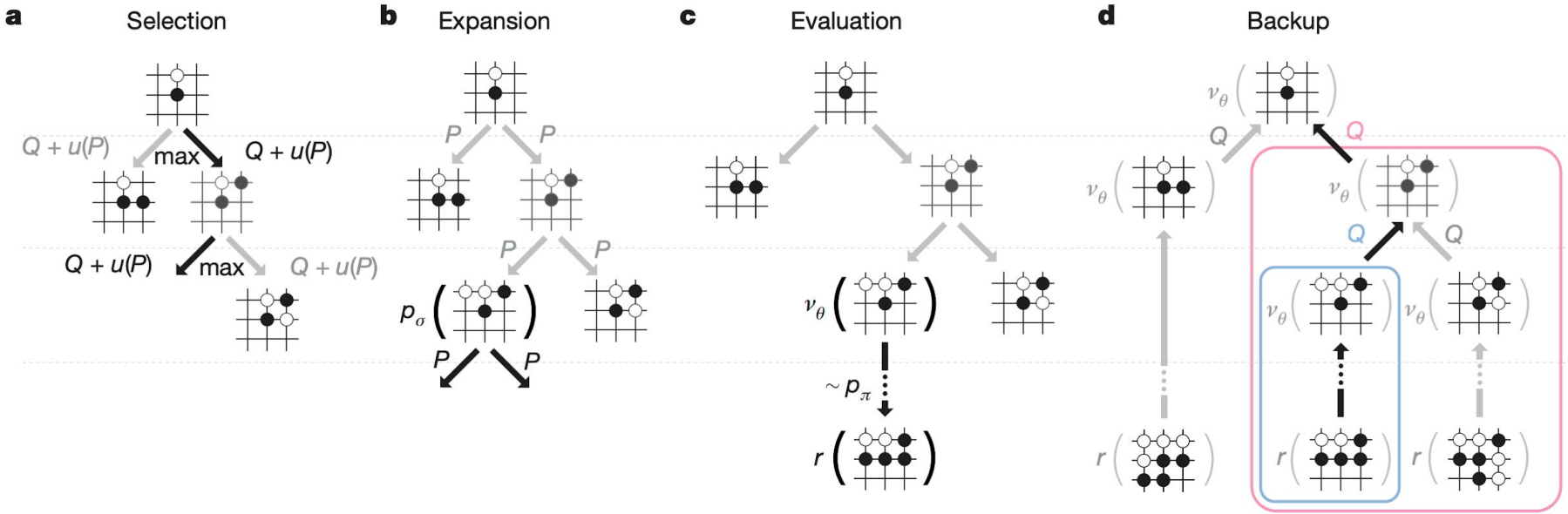
(a) Car merges *ahead* of human; anticipates human *braking*



(b) Car *backs up* at 4way stop; anticipates human *proceeding*

Sadigh et al. (2016). Planning for autonomous cars that leverage effects on human actions. RSS 2016.

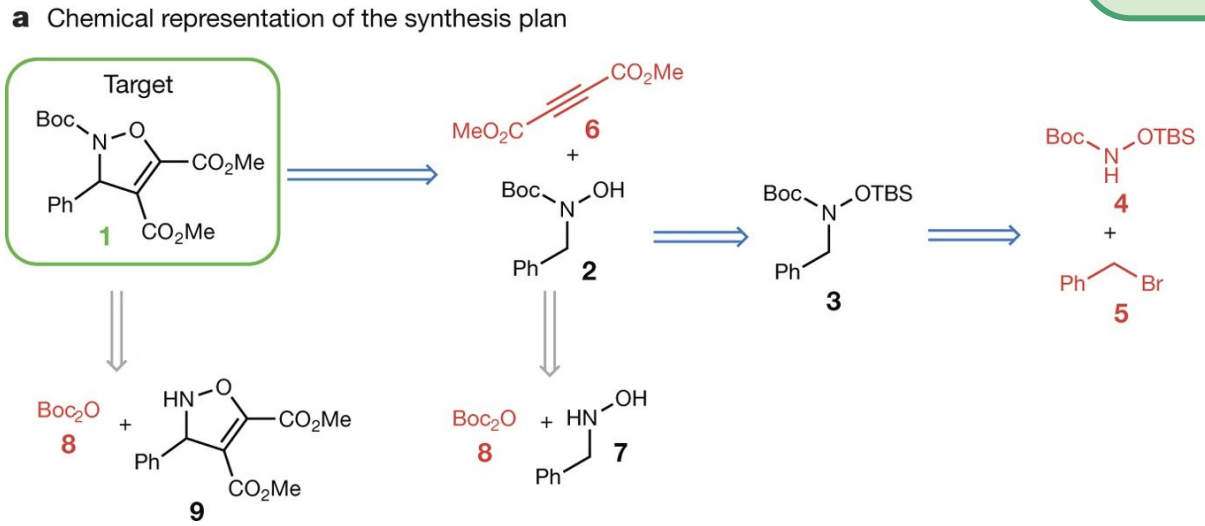
Model-based reasoning for ***games***



Silver et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484.

Model-based reasoning for ***science***

See also:
Cranmer, Brehmer, & Louppe (2020).
The frontier of simulation-based inference. *PNAS.*



Segler, Preuss, & Waller (2018). Planning chemical syntheses with deep neural networks and symbolic AI. Nature, 555(7698).

Model-based reasoning for ***operations research***

Figure from Warren Powell's 2017 ECSO tutorial, "A Unified Framework for Optimization under Uncertainty"

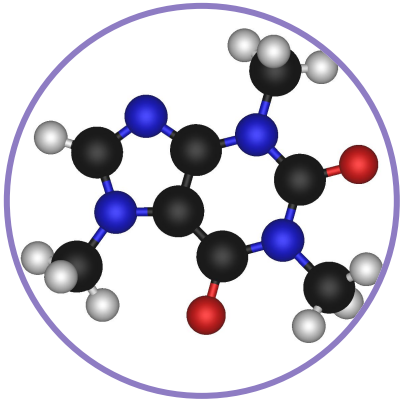
Salas & Powell (2013). Benchmarking a scalable approximate dynamic programming algorithm for stochastic control of multidimensional energy storage problems.

Robotics

Human/AI Interaction

Operations & Logistics

Games



Physical Sciences

Climate & Energy

What is “model-based RL”?

How does it differ from model-free RL?

What is a “model”?

How is the model used?

Dreamer

Dyna

Implicit planning

Locally-linear models

SVG

World Models

Background planning

Value iteration

CEM

MPC

PDDL

Pilco

MBPO

iLQR

DDP

MCTS

Shooting

Decision-time planning

MPPI

STOMP

iLQR

System identification

I2A

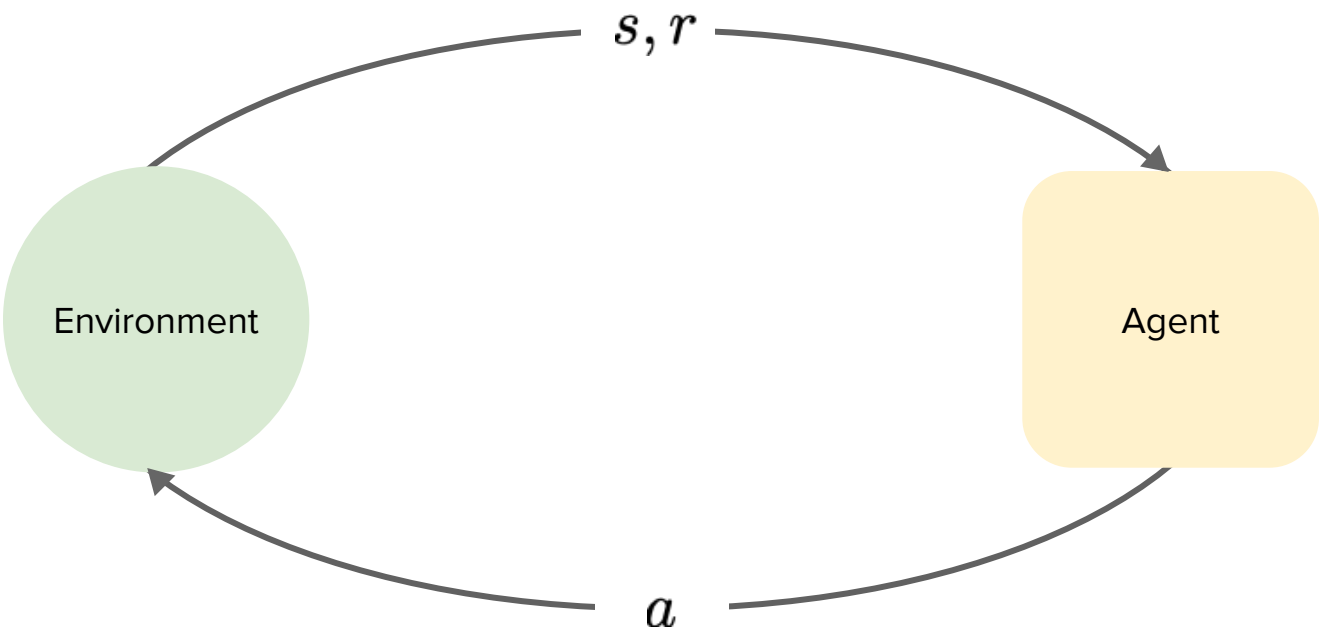
State-space models

Collocation

Outline

1. Introduction and motivation
2. **Problem statement**
3. What is a “model”?
4. What is model-based control?
5. Model-based control in the loop
6. What else can models be used for?
7. What’s missing from model-based methods?
8. Conclusion

Quick recap: Sequential decision making



Quick recap: Sequential decision making

- 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 $\pi : S \to A$

$$\arg\max_{\pi} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$$

Subject to

$$a_t = \pi(s_t)$$
$$s_{t+1} = T(s_t, a_t)$$

(infinite horizon, discounted MDP)

Model-free vs. model-based reinforcement learning

Collect data

$$\mathcal{D} = \{s_t, a_t, r_{t+1}, s_{t+1}\}_{t=0}^{T}$$

Model-free: learn policy directly from data

$$\mathcal{D} \rightarrow \pi$$ *e.g. Q-learning, policy gradient*

Model-based: learn model, then use it to learn or improve a policy

$$\mathcal{D} \rightarrow f \rightarrow \pi$$

What is a model?

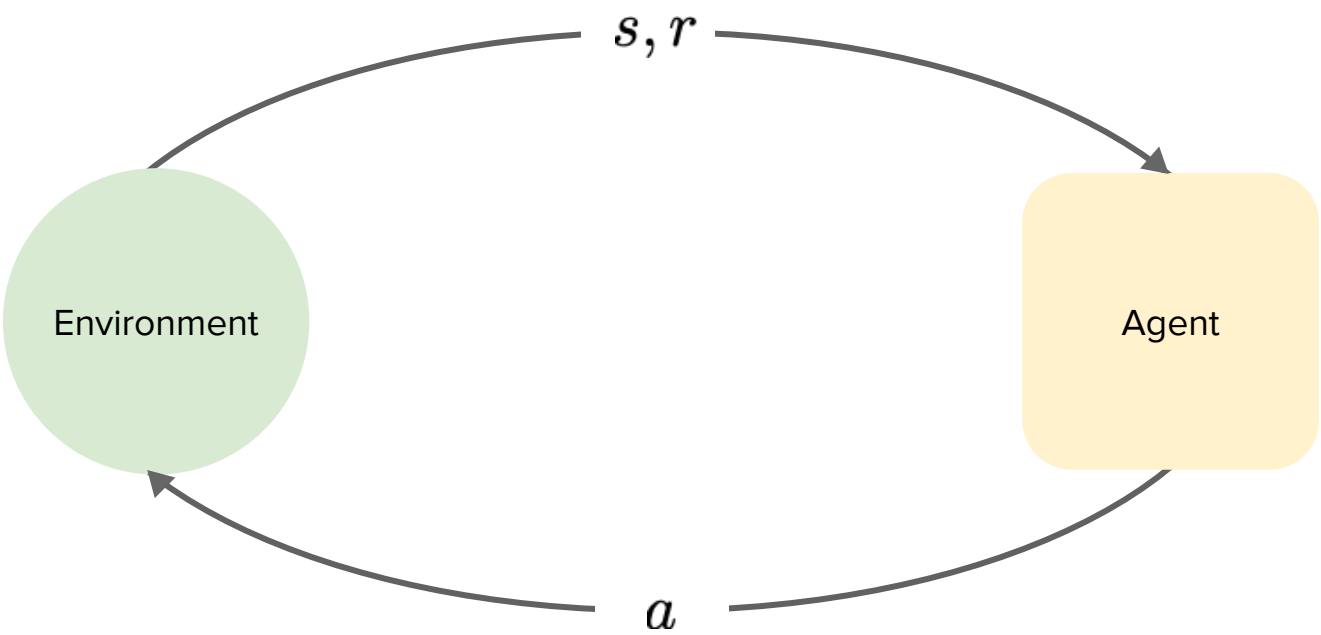
*Definition: a model is a representation that **explicitly** encodes knowledge about the structure of the environment and task.*

- A transition/dynamics model: $s_{t+1} = f_s(s_t, a_t)$
- A model of rewards: $r_{t+1} = f_r(s_t, a_t)$

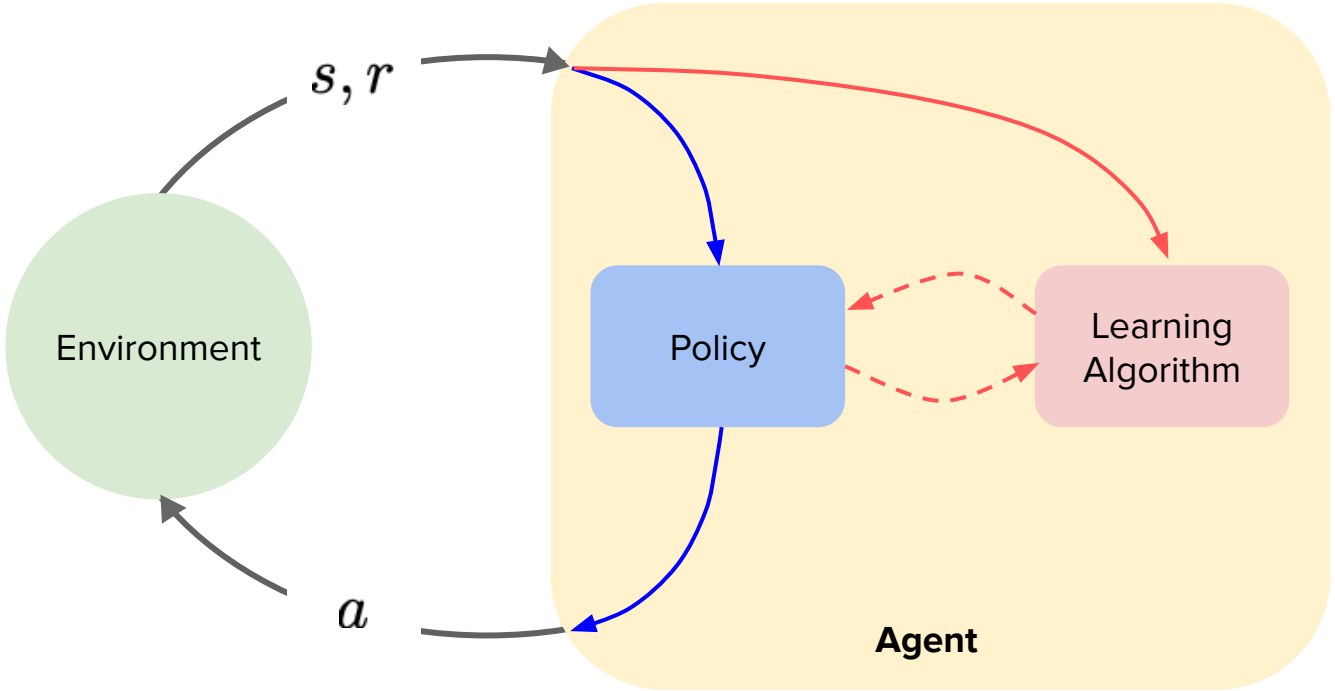
Typically what is meant by the model in model-based RL

- 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: $G_t = Q(s_t, a_t)$ or $G_t = V(s_t)$

Where does the model fit into the picture?

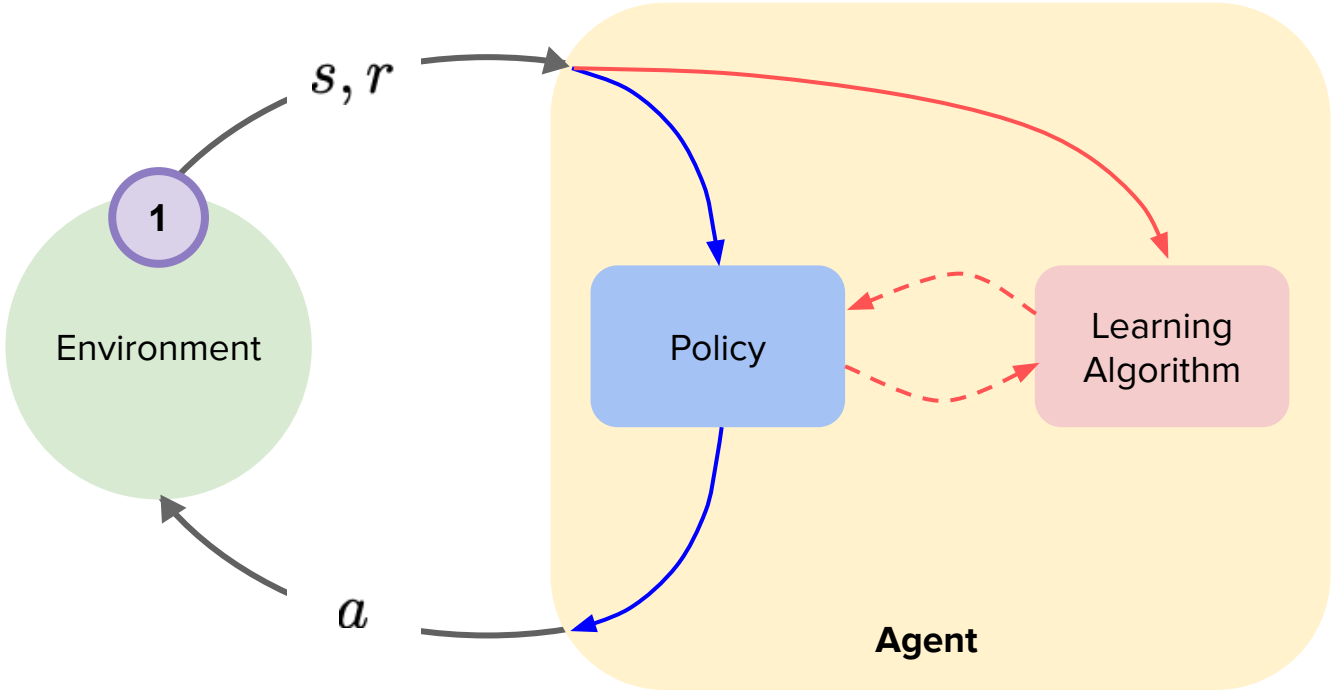


Where does the model fit into the picture?



Where does the model fit into the picture?

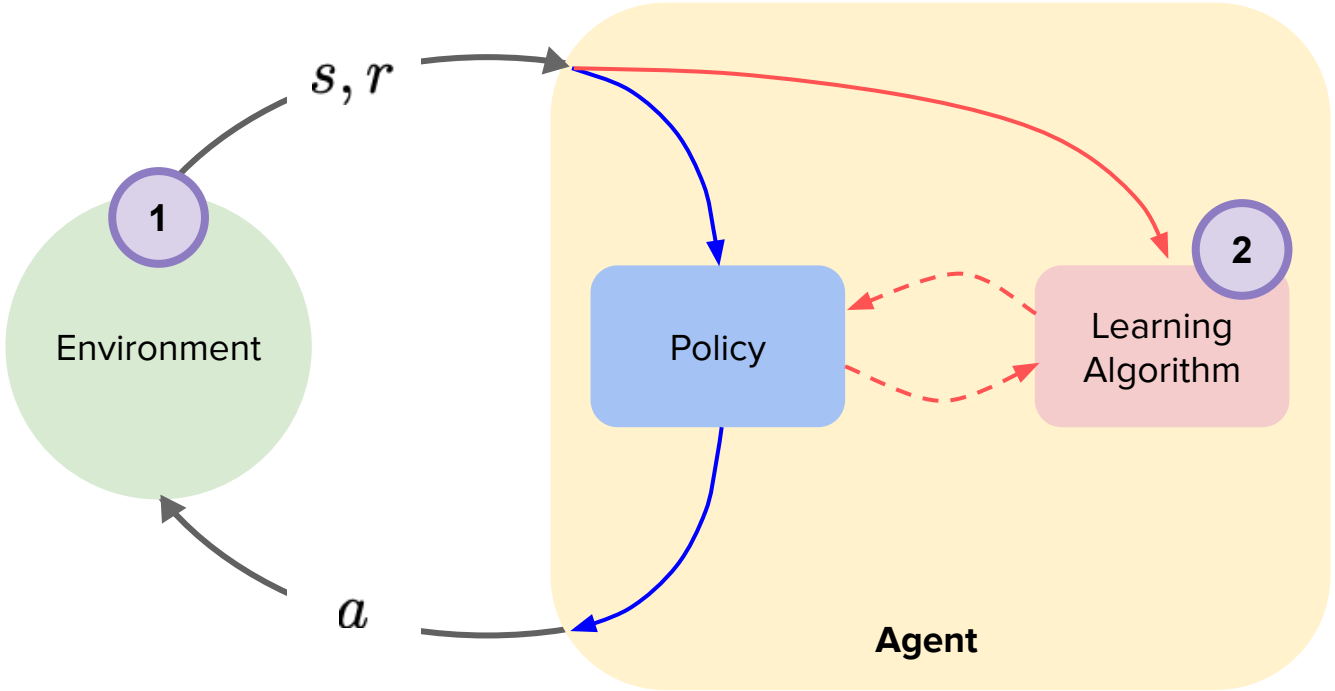
1. Simulating the environment



Where does the model fit into the picture?

1. Simulating the environment

2. Assisting the learning algorithm

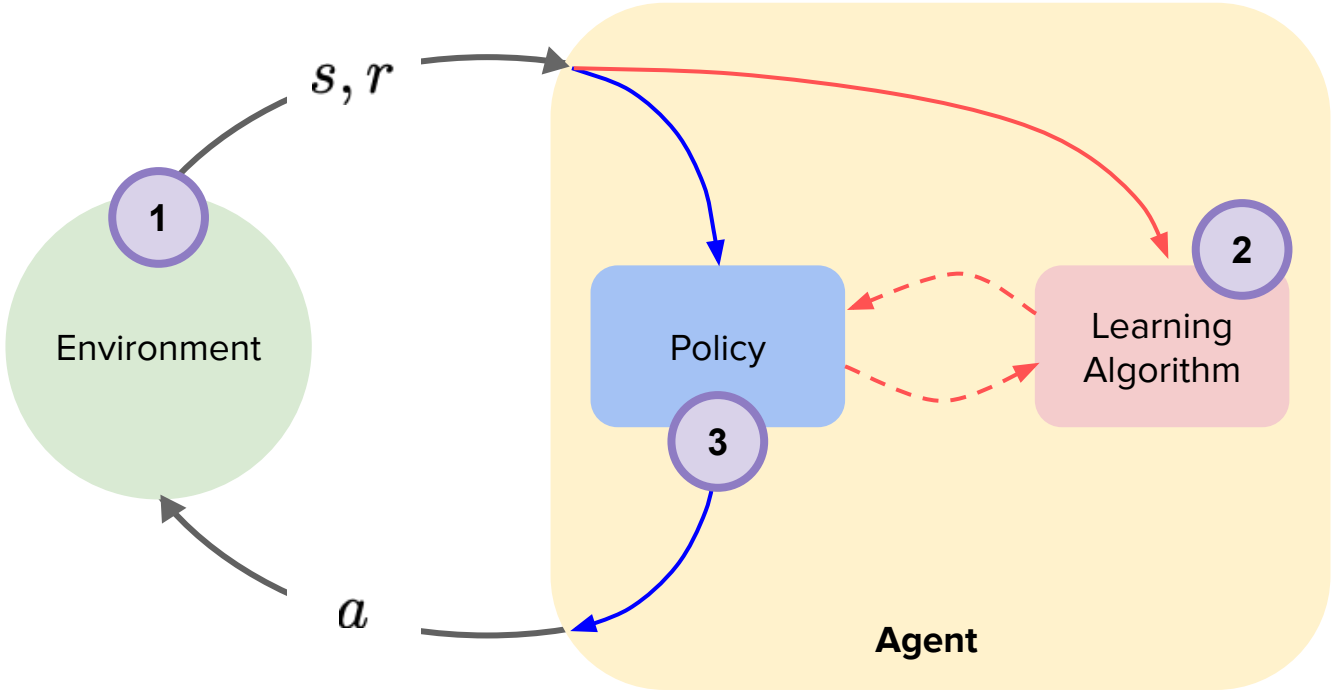


Where does the model fit into the picture?

1. Simulating the environment

2. Assisting the learning algorithm

3. Strengthening the policy



Model-free vs. model-based RL

Model-free	Model-based	
+	?	Asymptotic rewards*
+	- / +	Computation at deployment*
-	+	Data efficiency*
-	+	Speed to adapt to changing rewards*
-	+	Speed to adapt to changing dynamics*
-	+	Exploration*

*but it depends a lot on the specific method!

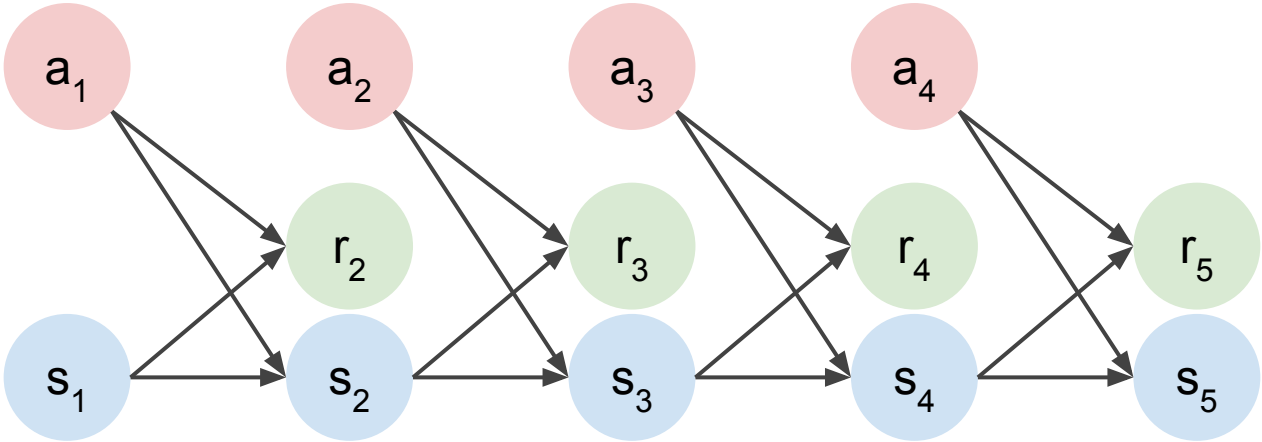
Outline

1. Introduction and motivation
2. Problem statement
3. **What is a “model”?**
4. What is model-based control?
5. Model-based control in the loop
6. What else can models be used for?
7. What’s missing from model-based methods?
8. Conclusion

Models in sequential decision making

*Definition: a model is a representation that **explicitly** encodes knowledge about the structure of the environment and task.*

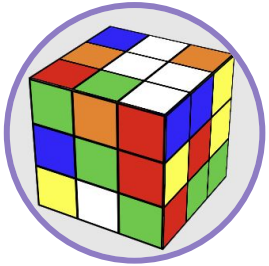
Transition model: $s_{t+1} = f_s(s_t, a_t)$ **Reward model:** $r_{t+1} = f_r(s_t, a_t)$



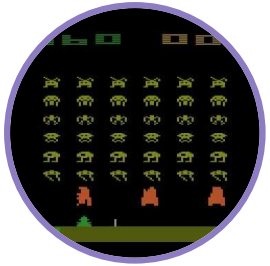
Environment simulators

Sometimes we know the ground truth dynamics and rewards.
Might as well use them!

Silver et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature.

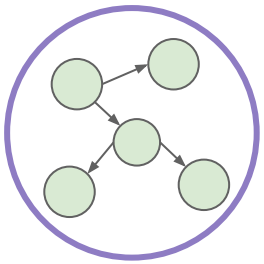


Agostinelli et al. (2019). Solving the Rubik's cube with deep reinforcement learning and search. Nature Machine Intelligence.



Bellemare et al. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. JAIR.

Todorov et al. (2012). MuJoCo: A physics engine for model-based control. IROS.



Shen et al. (2019). M-Walk: Learning to Walk over Graphs using Monte Carlo Tree Search. NeurIPS.

Ellis et al. (2019). Write, Execute, Assess: Program Synthesis with a REPL. NeurIPS.

Why do we want to learn a model?

Planning with real robots

(too expensive, too risky)

Simulating complex physical dynamics

(too expensive)

Interactions with humans

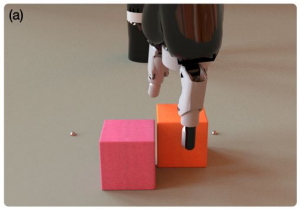
(no access)

Dimensions of learned models

States vs. **Observations** vs. **Latent States**

$$s = [x, \dot{x}, \phi]$$

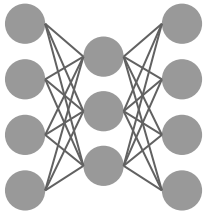
$$o =$$



$$z \in \mathbb{R}^{d_z}$$

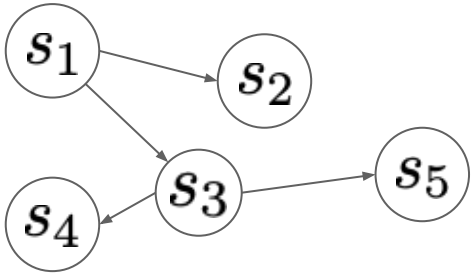
---

s_t
a_t



s_{t+1}

Parametric vs. **Non-Parametric**

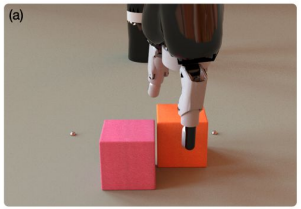


Dimensions of learned models

States vs. **Observations** vs. **Latent States**

$$s = [x, \dot{x}, \phi]$$

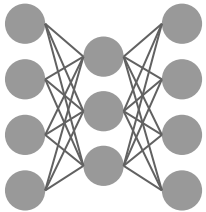
$$o =$$



$$z \in \mathbb{R}^{d_z}$$

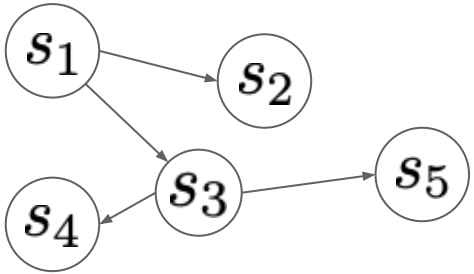
---

s_t
a_t



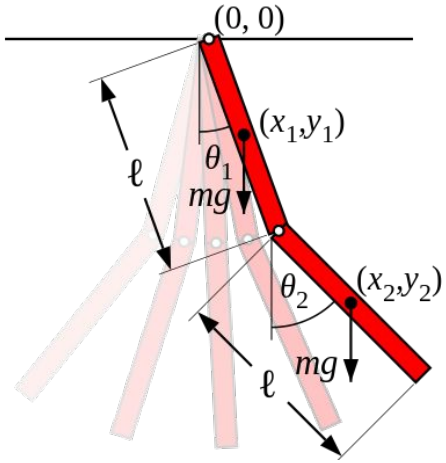
s_{t+1}

Parametric vs. **Non-Parametric**



State transitions: dynamical systems

Double pendulum (acrobot)



$$x_1 = \frac{l}{2}\sin\theta_1 \qquad x_2 = l\left(\sin\theta_1 + \tfrac{1}{2}\sin\theta_2\right)$$

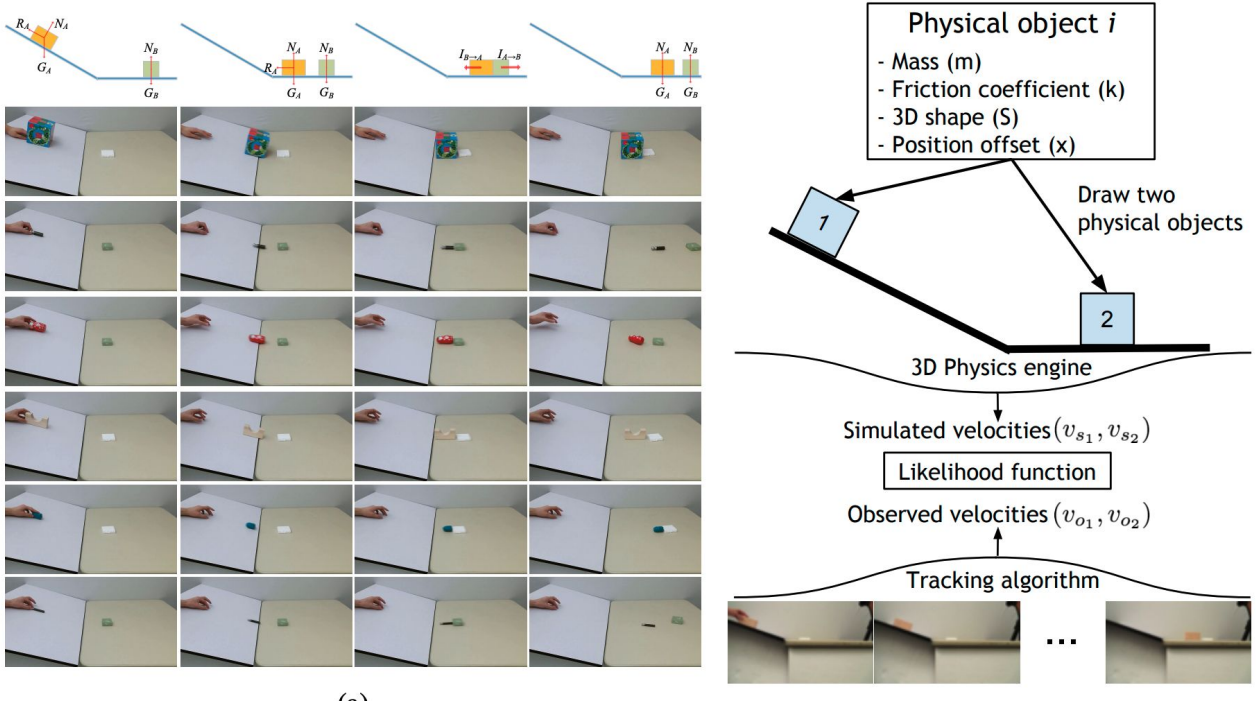
$$y_1 = -\frac{l}{2}\cos\theta_1 \qquad y_2 = -l\left(\cos\theta_1 + \tfrac{1}{2}\cos\theta_2\right)$$

Equations of motion are assumed known.

Reward usually assumed known.

Use system identification to estimate unknown parameters (e.g. mass)

State transitions: dynamical systems

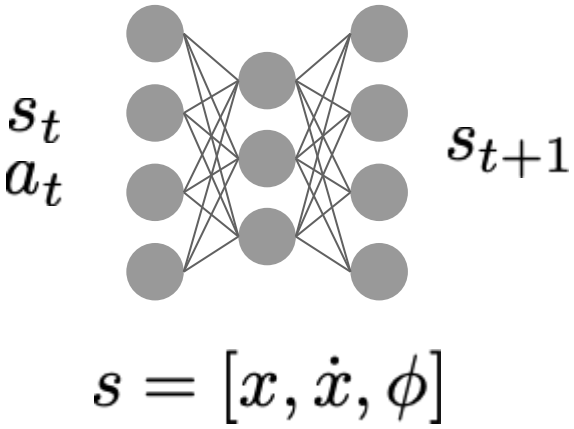


Physics engine assumed known.

Train CNN to estimate unknown parameters (e.g. mass)

Wu et al. (2015). Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning.

State transitions: MLP-based

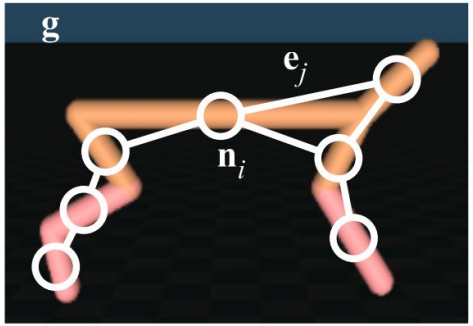


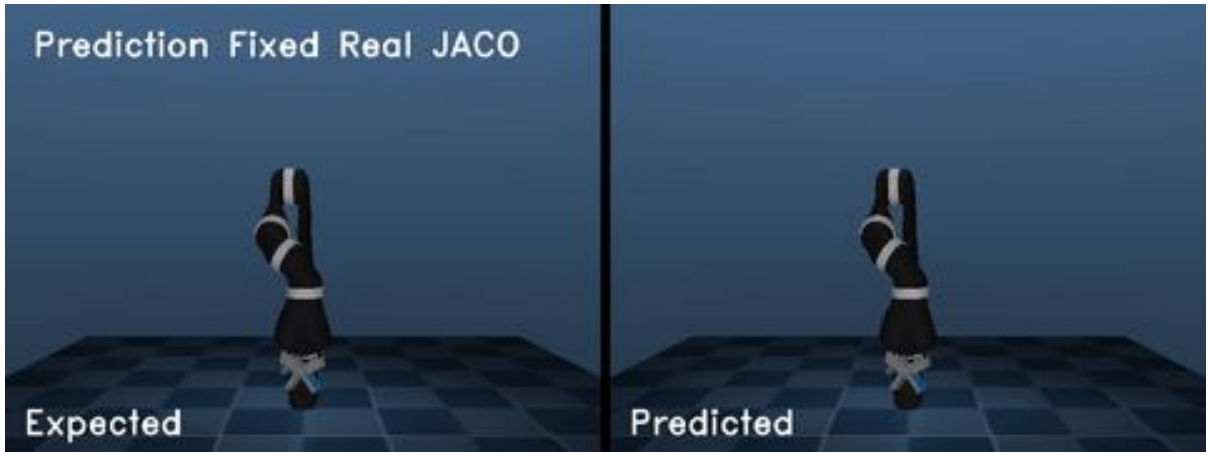
Note: typically better to predict the derivative (change in *s*), and then integrate to obtain s_{t+1}

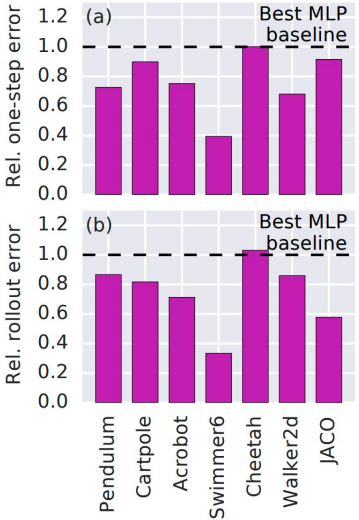
This is the same idea behind using skip connections / residual blocks!

State transitions: GNN-based

Represent relationships between state variables with a graph, and then process with a graph neural network

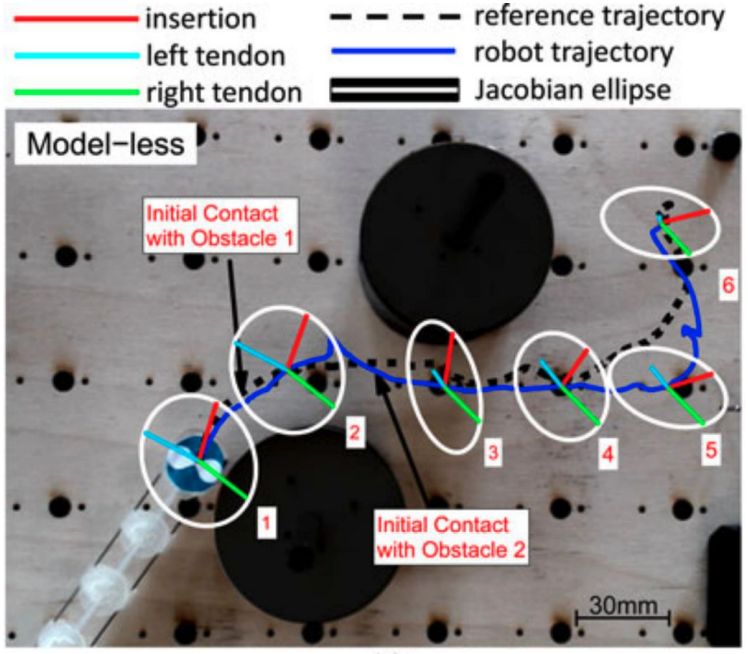






Sanchez-Gonzalez et al. (2018). Graph Networks as Learnable Physics Engines for Inference and Control. ICML 2018.

State transitions: locally-linear models



1. Assume locally-linear state transitions:

$$\dot{s}_{t+1} = J_{s_t} \dot{a}_t$$

2. Plan with locally-linear model.

3. Execute planned control.

4. Update locally-linear model with recursive least squares.

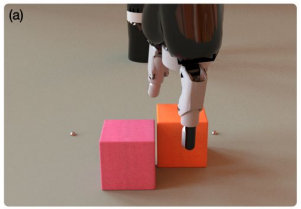
Yip & Camarillo (2014). Model-Less Feedback Control of Continuum Manipulators in Constrained Environments. IEEE Transactions on Robotics, 30(4).

Dimensions of learned models

States vs. **Observations** vs. **Latent States**

$$s = [x, \dot{x}, \phi]$$

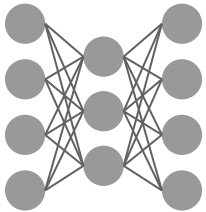
$$o =$$



$$z \in \mathbb{R}^{d_z}$$

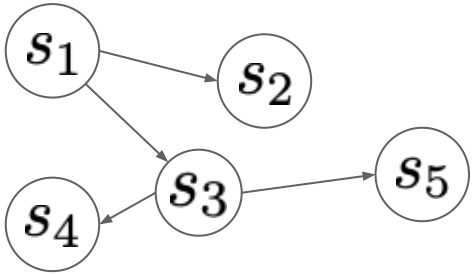
---

s_t
a_t

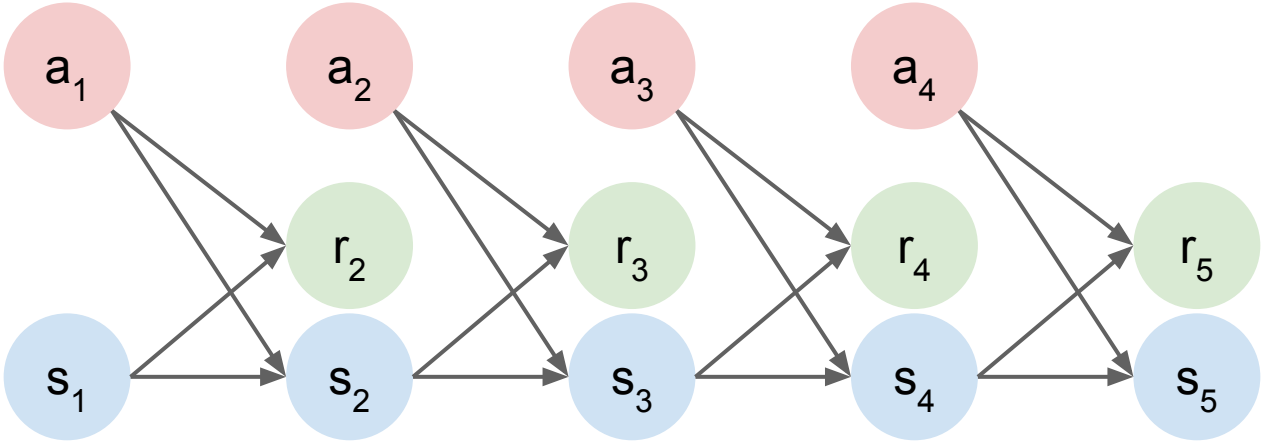


s_{t+1}

Parametric vs. **Non-Parametric**



Revisiting our MDP

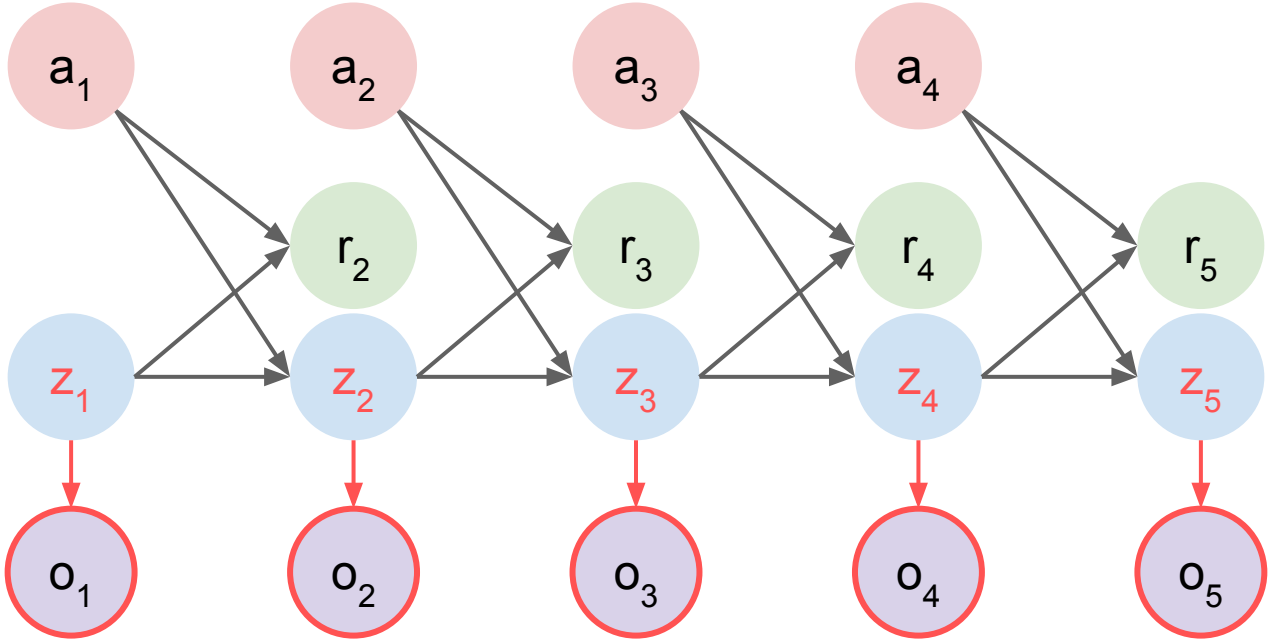


Revisiting our MDP

This is not necessarily a POMDP!

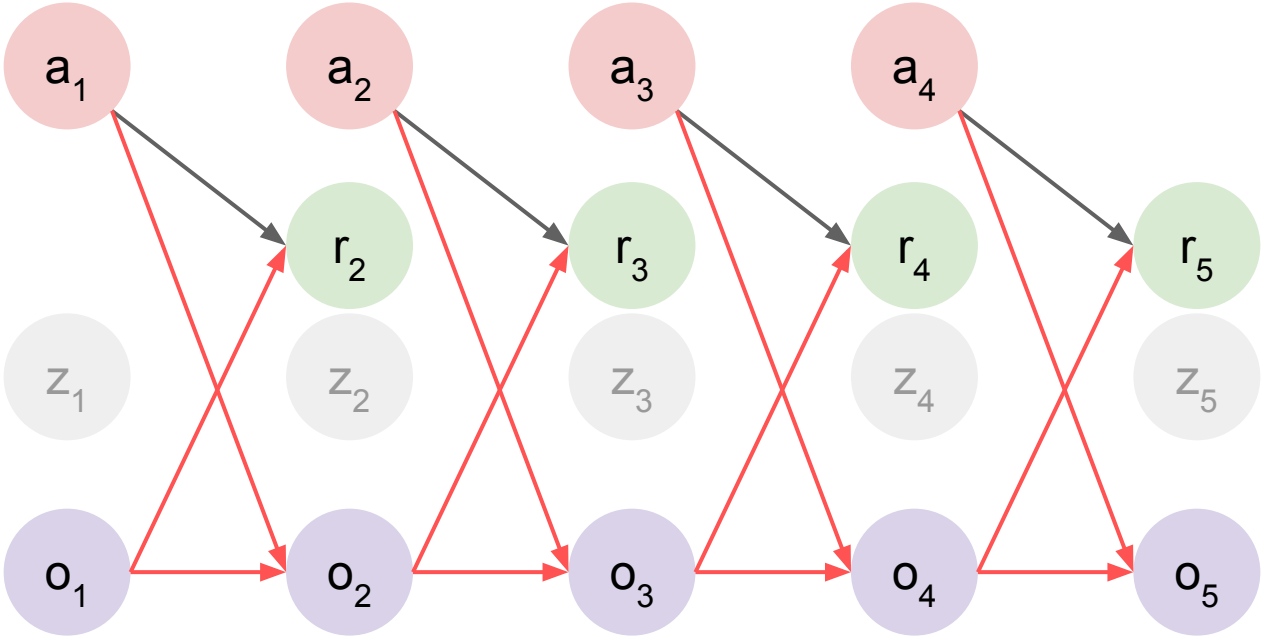
z = low dim
o = high dim

Think 2D Mujoco joints versus pixel observations



Observation-transition models

Directly predict transitions between observations

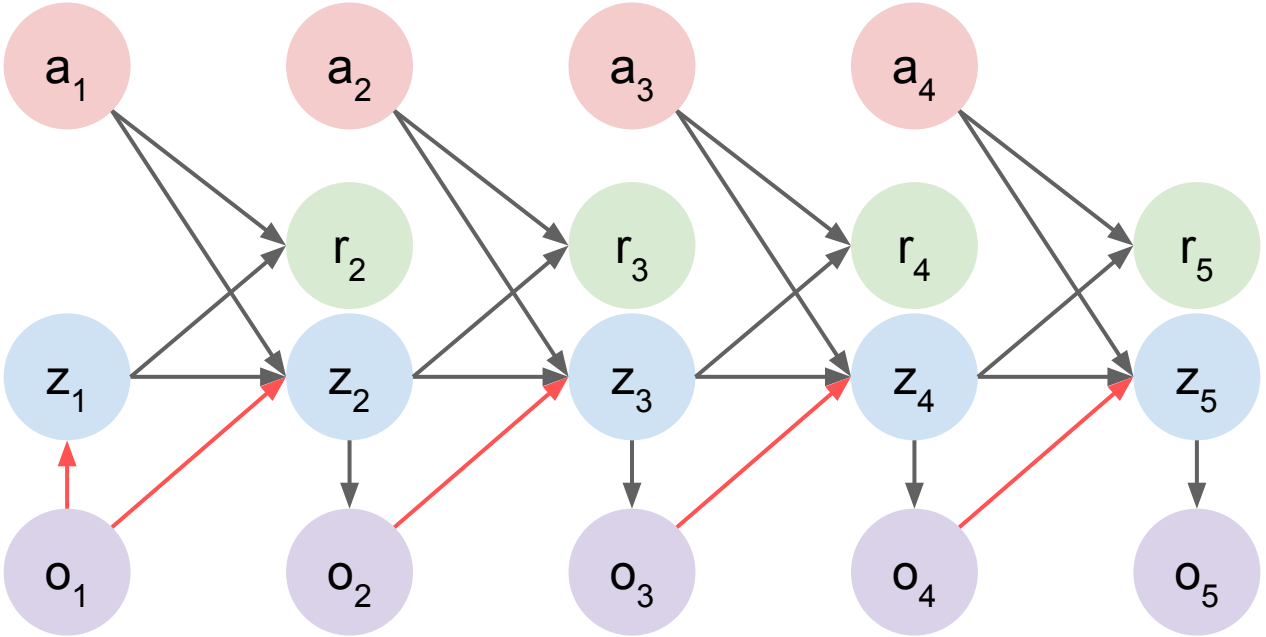


Observation-transition models

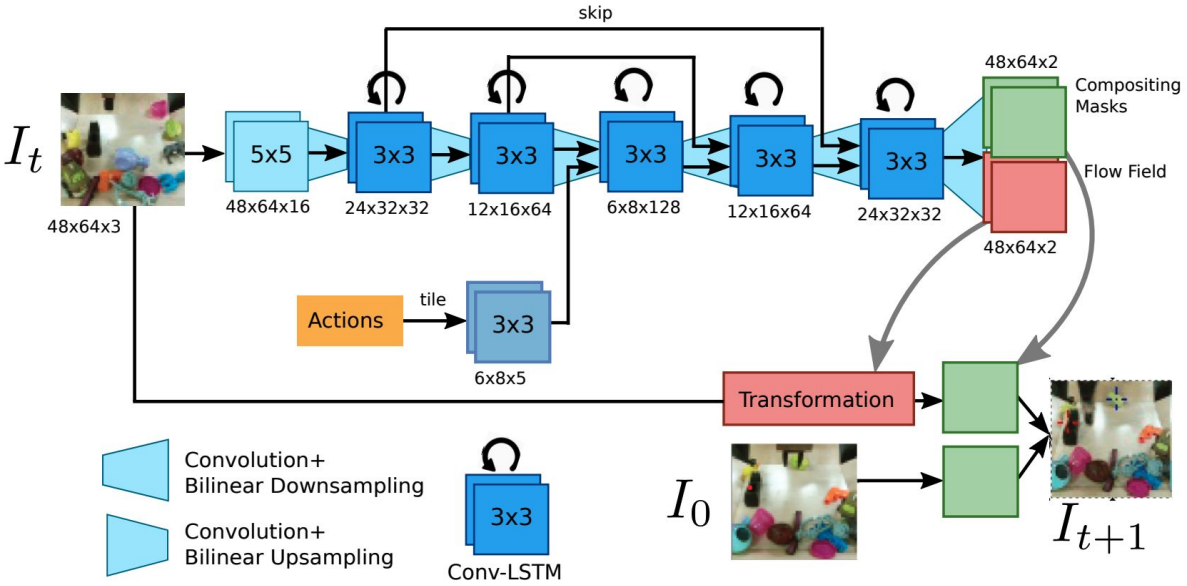
Directly predict transitions between observations

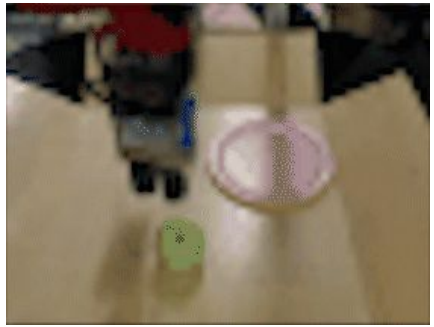
OR

Reconstruct observations at every timestep



Observation-transition models





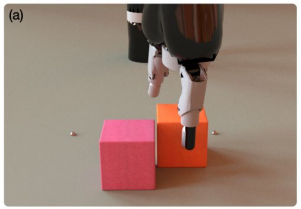
Ebert, Finn, et al. (2018); Finn & Levine (2017); Finn, Goodfellow, & Levine (2016)
https://bair.berkeley.edu/blog/2018/11/30/visual-rl/

Dimensions of learned models

States vs. **Observations** vs. **Latent States**

$$s = [x, \dot{x}, \phi]$$

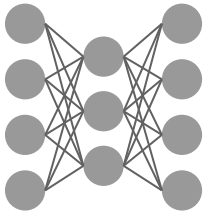
$$o =$$



$$z \in \mathbb{R}^{d_z}$$

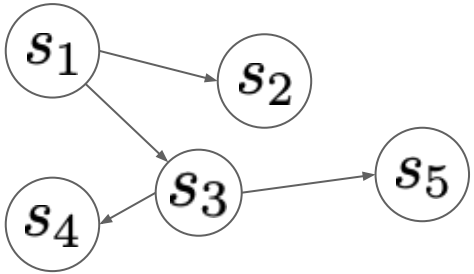
---

s_t
a_t



s_{t+1}

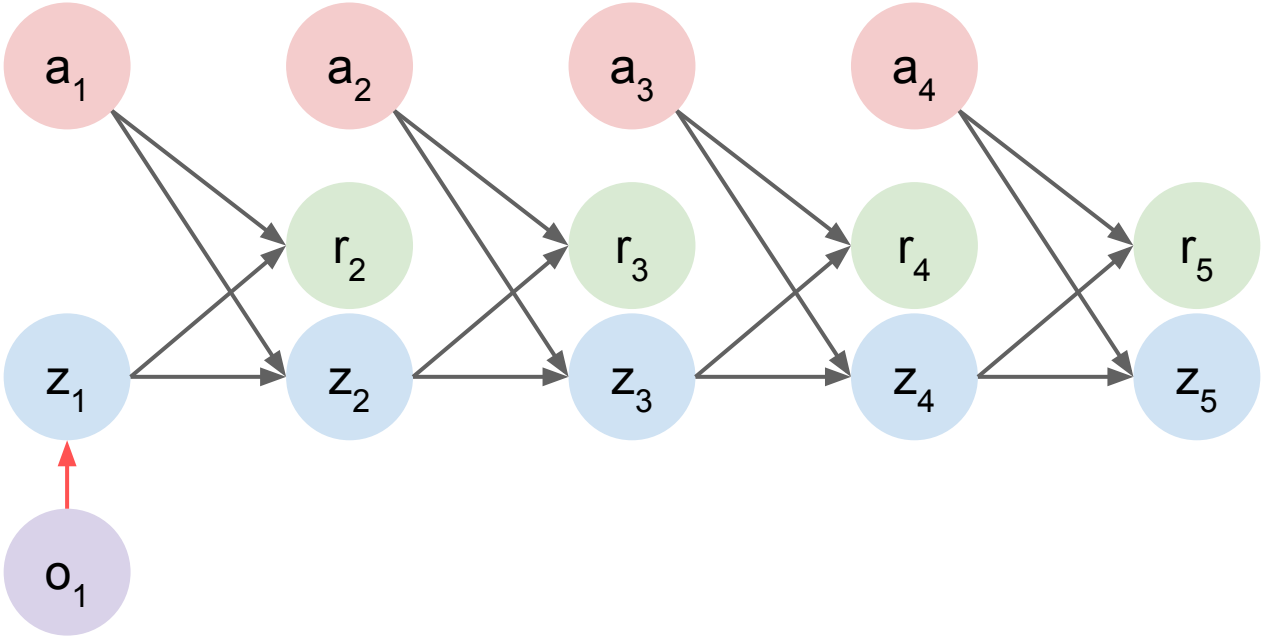
Parametric vs. **Non-Parametric**



Latent state-transition models (a.k.a state-space models)

Reconstructing high dimensional observations is computationally expensive!

Embed initial observation, and then roll out in the latent space.

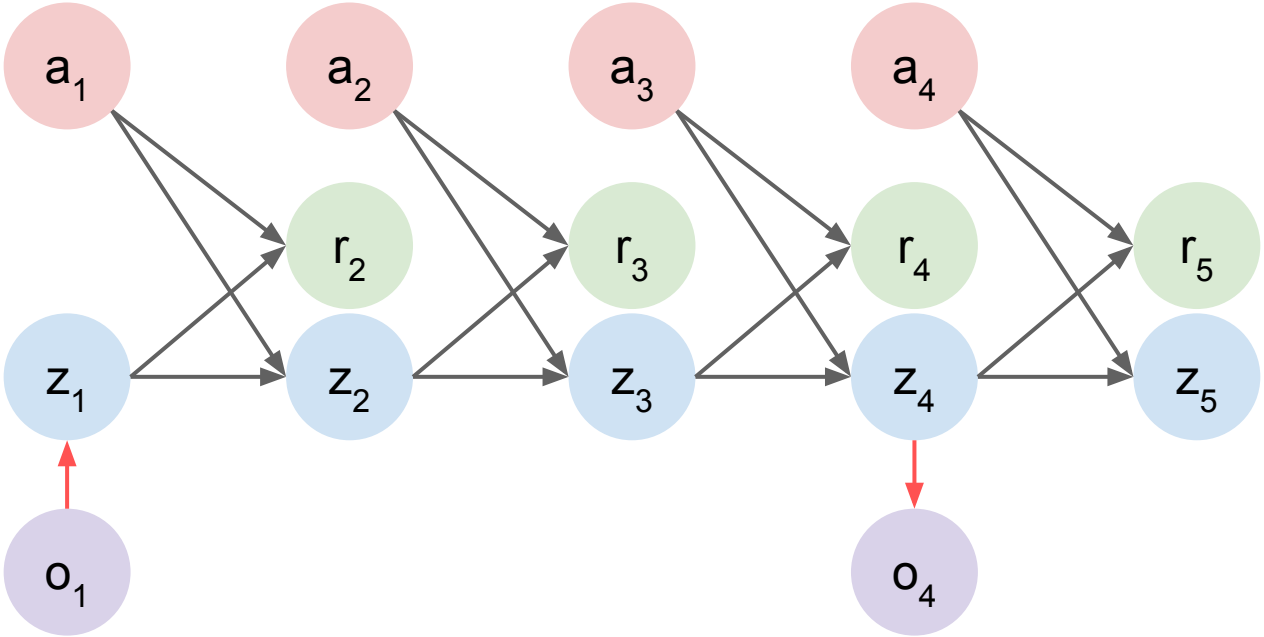


Latent state-transition models (a.k.a state-space models)

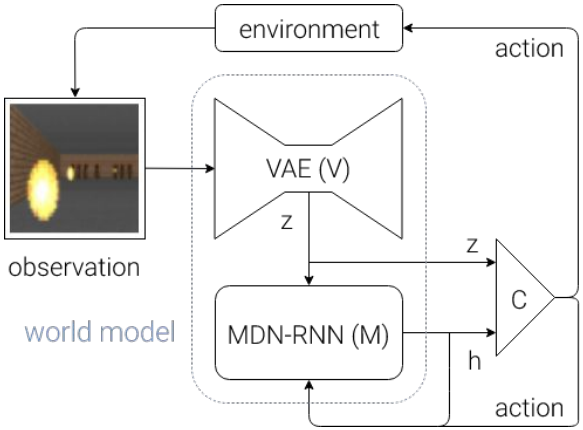
Reconstructing high dimensional observations is computationally expensive!

Embed initial observation, and then roll out in the latent space.

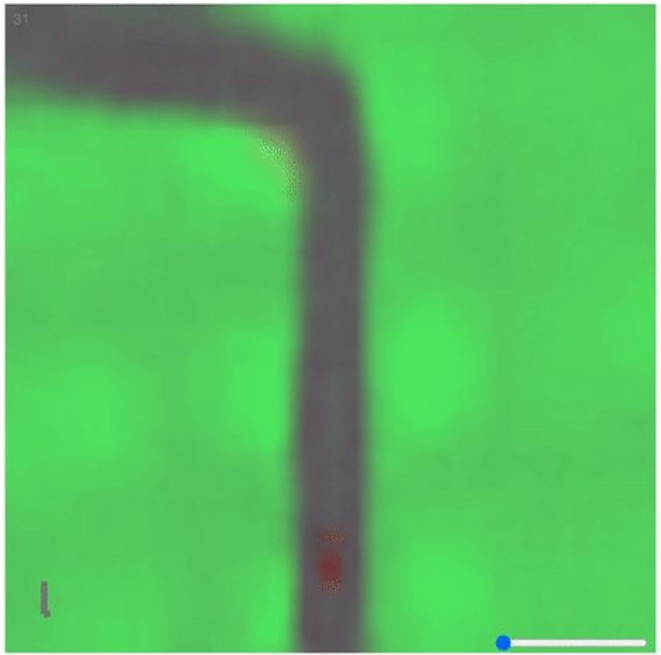
Reconstruct future frames as desired.



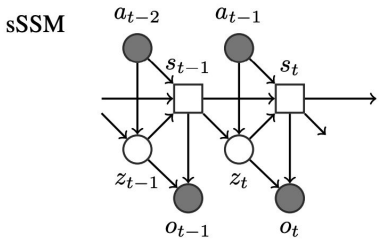
Latent state-transition models (a.k.a state-space models)



Ha & Schmidhuber (2018). World Models. NeurIPS 2018.



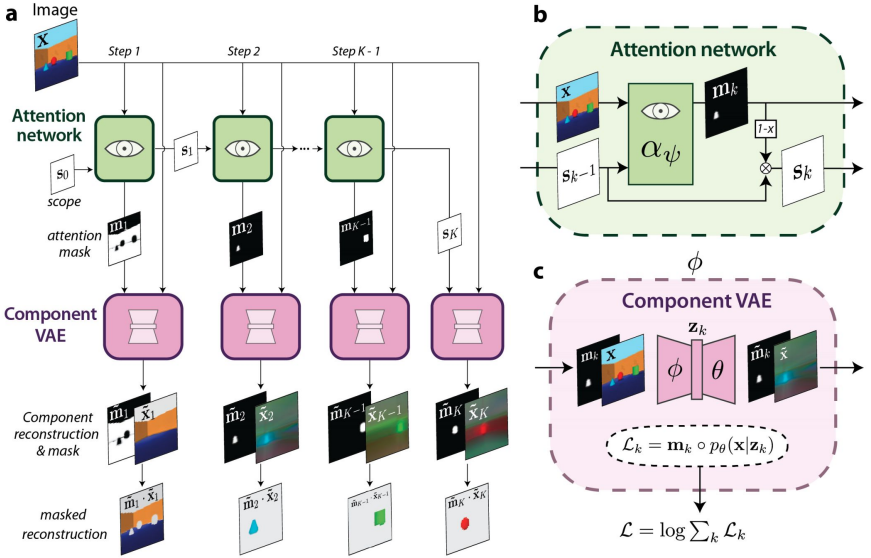
Latent state-transition models (a.k.a state-space models)

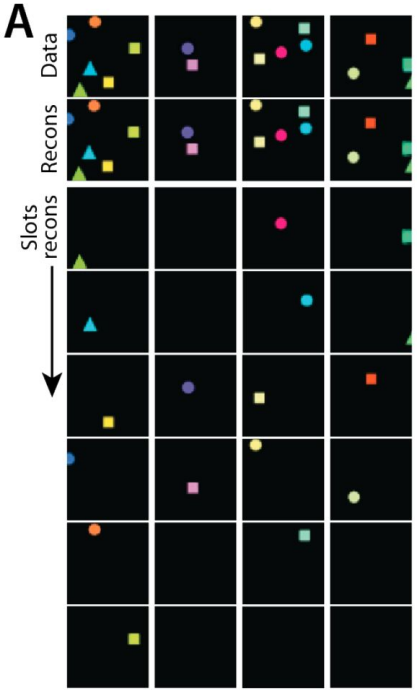


Buesing et al. (2018). Learning and Querying Fast Generative Models for Reinforcement Learning. ICML 2018.

Note: stochastic latent states lead to crisper predictions!

Structured latent state-transition models

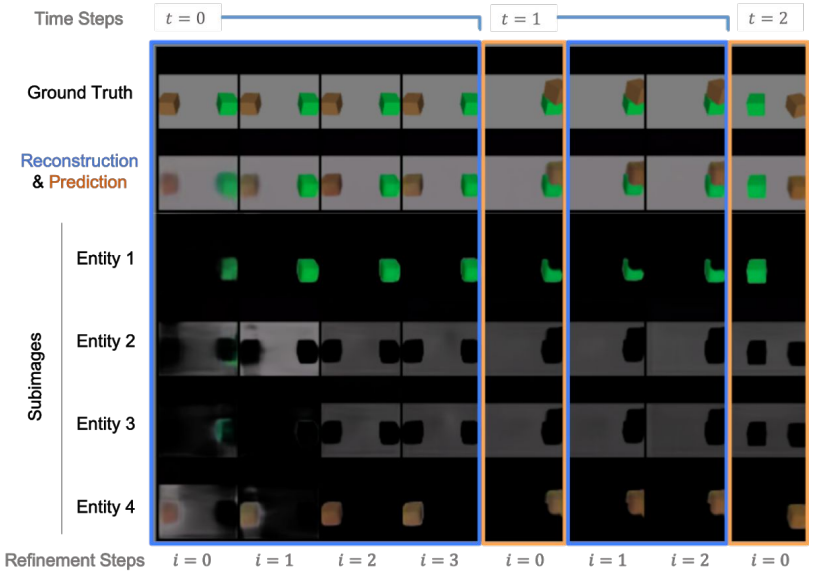




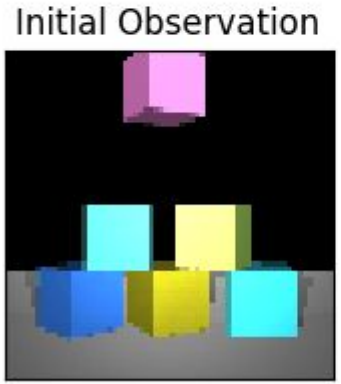
Burgess et al. (2019). MONet: Unsupervised Scene Decomposition and Representation. arXiv.

Watters et al. (2019). COBRA: Data-Efficient Model-Based RL through Unsupervised Object Discovery and Curiosity-Driven Exploration. arXiv.

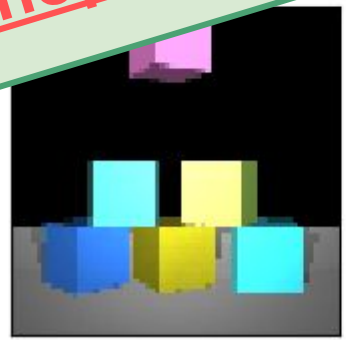
Structured latent state-transition models



Veerapaneni, Co-Reyes*, Chang*, et al. (2019). Entity Abstraction in Visual Model-Based Reinforcement Learning. CoRL 2019.* https://sites.google.com/corp/view/op3website/



Structured latent state-transition models



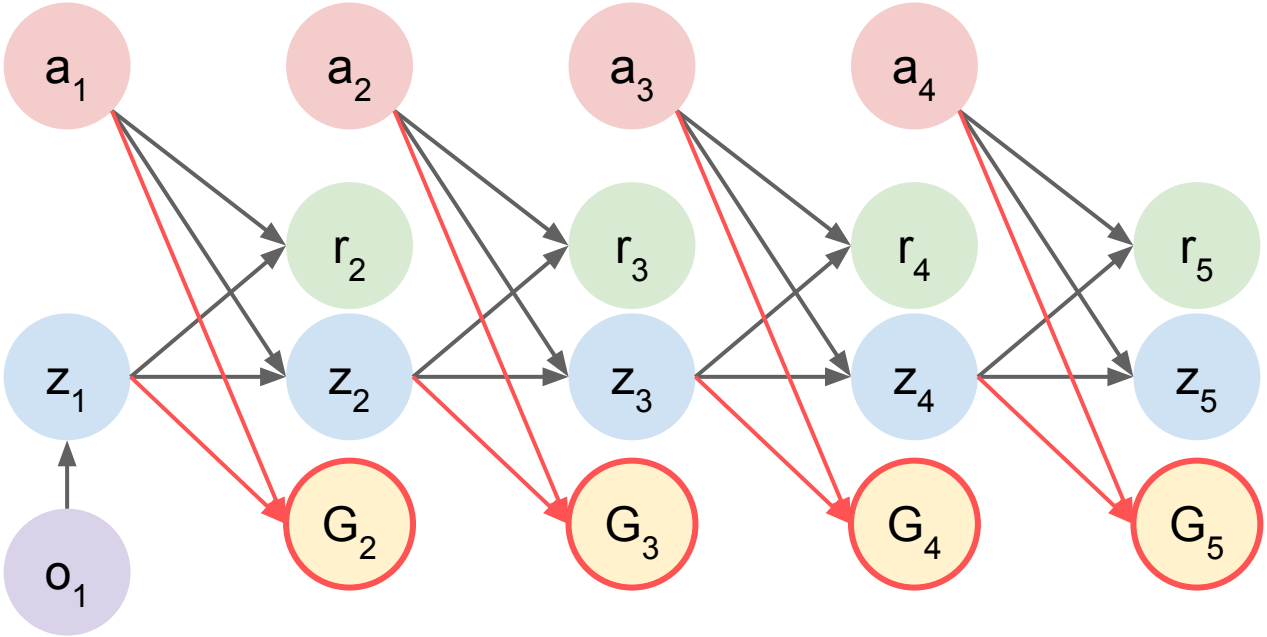
Check out the workshop on Object-Oriented Learning! https://oolworkshop.github.io/

Veerap… …ehang, et al. (2019). Entity Abstraction … …Model-Based Reinforcement Learning. CoRL 2019.* https://sites.google.com/corp/view/op3website/

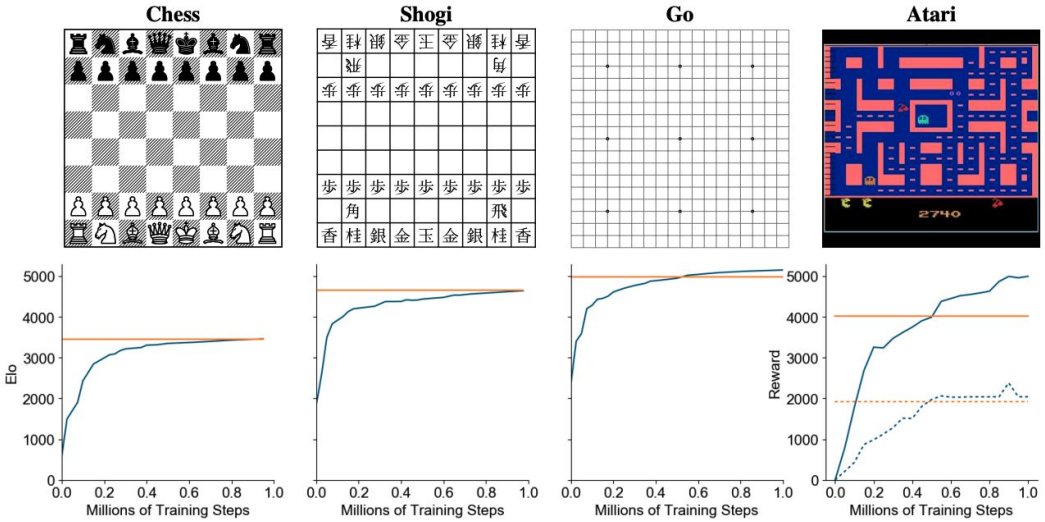
Recurrent value models

Predict expected return *G* ***and*** reward (and policy, not shown)

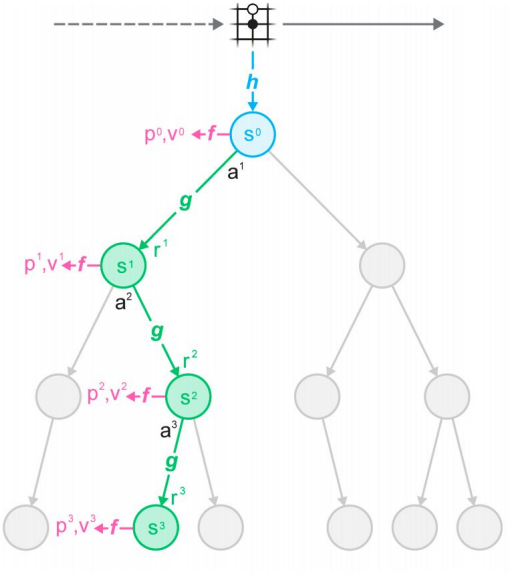
Blurs the boundary between model-free and model-based!



Recurrent value models



Schrittwieser et al. (2019). Mastering Atari, Go, Chess and Shogi by planning with a learned model. arXiv.



Unroll in latent space to predict reward, value, and policy, use with MCTS.

Never need to predict future observations!

Trade-offs for ways to represent states

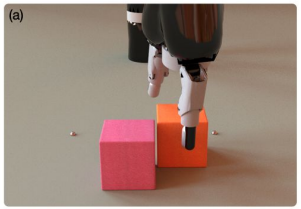
Name	Features	Speed of learning	Speed of predictions	Domain knowledge	Long-term accuracy
Dynamical system	States	Fast	Fast	High	High
MLP	States	Med	Fast	Low	Med
GNN	States	Med	Med	Med	High
Locally linear	States	Fast	Fast	Med	Med
Observation	Observations	Slow	Slow	Low	Low
State-space models	Latent States	Slow	Fast	Low	Med
Object-centric	Latent States	Slow	Med	Med	High
Recurrent value	Latent States	Med	Fast	Low	Med

Dimensions of learned models

States vs. **Observations** vs. **Latent States**

$$s = [x, \dot{x}, \phi]$$

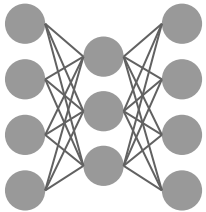
$$o =$$



$$z \in \mathbb{R}^{d_z}$$

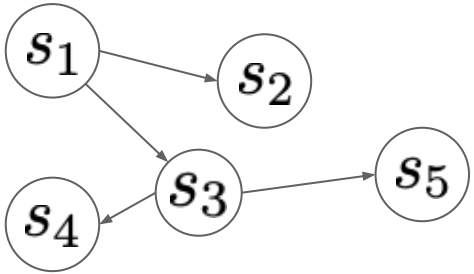
---

s_t
a_t



s_{t+1}

Parametric vs. **Non-Parametric**

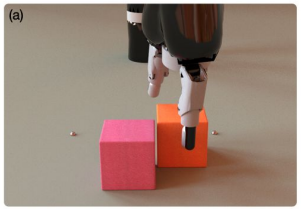


Dimensions of learned models

States vs. **Observations** vs. **Latent States**

$$s = [x, \dot{x}, \phi]$$

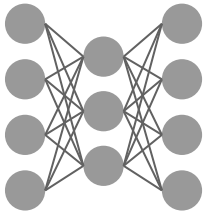
$$o =$$



$$z \in \mathbb{R}^{d_z}$$

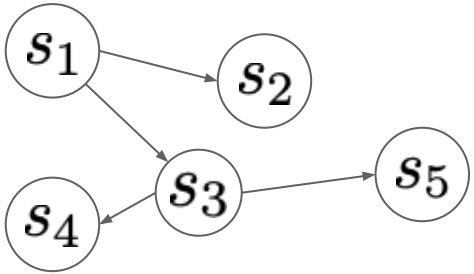
---

s_t
a_t

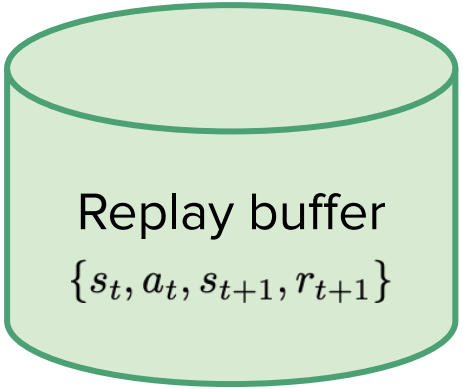


s_{t+1}

Parametric vs. **Non-Parametric**

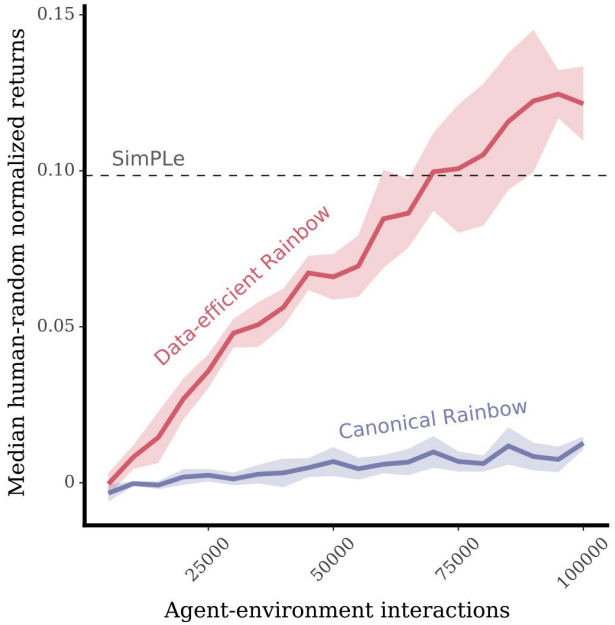


Non-parametric models: episodic memory



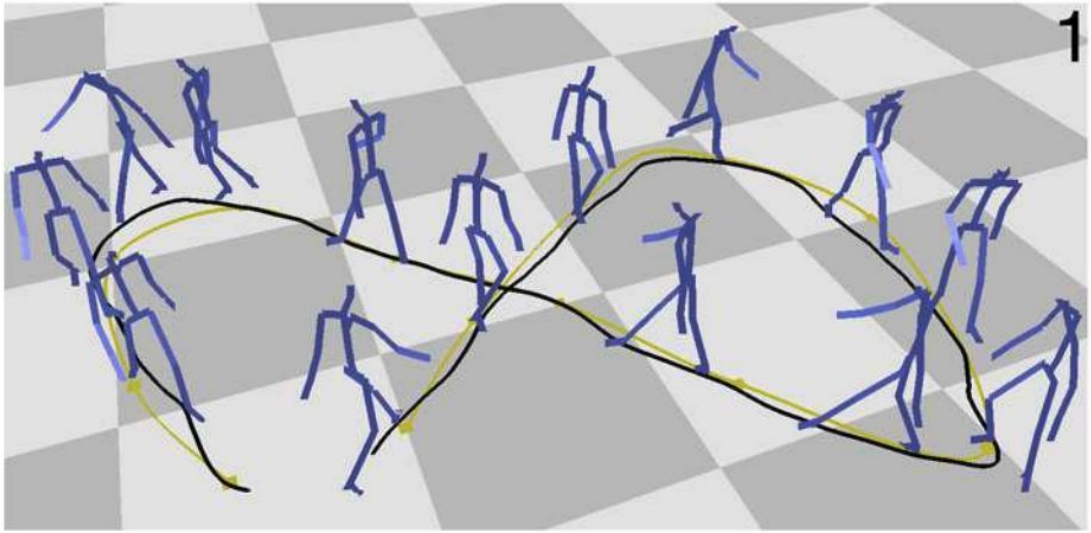
With enough data, the empirical samples of transitions models the full distribution.

Equivalent to a parametric model over the same distribution which just interpolates!



van Hasselt, Hessel, & Aslanides (2019). When to use parametric models in reinforcement learning? NeurIPS 2019.

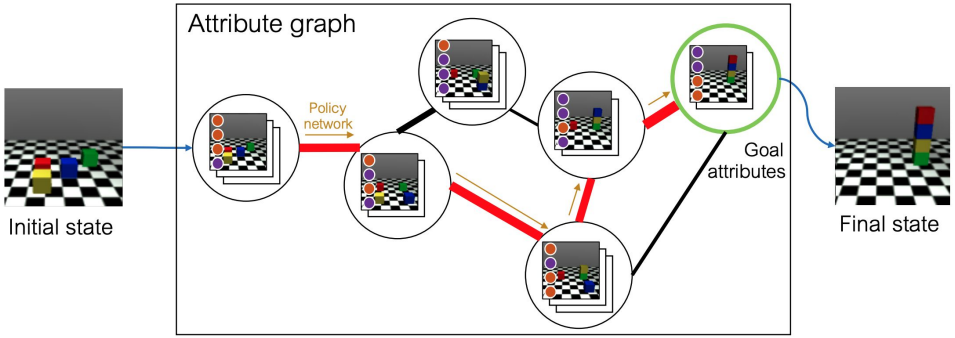
Non-parametric models: transition graphs



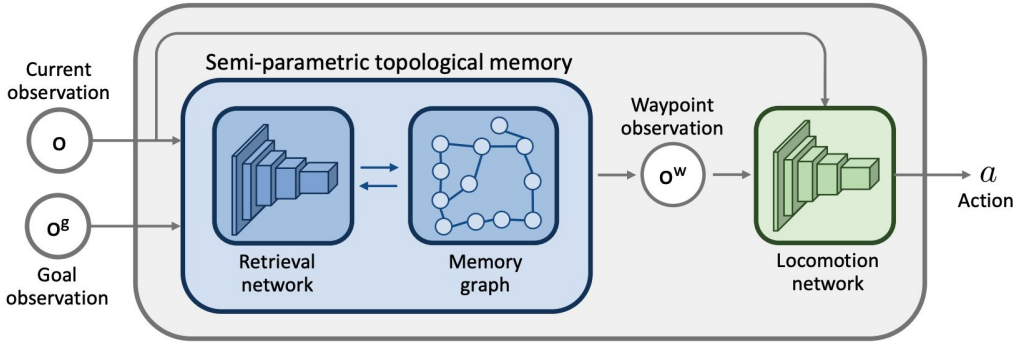
Kovar, Gleicher, & Pighin (2002). Motion graphs. ACM Transactions on Graphics, 21(3).

Idea: Identify transitions between stored data points, and interpolate between them.

Non-parametric models: transition graphs



Zhang, Lerer, et al. (2018). Composable Planning with Attributes. ICML 2018.



Savinov, Dosovitskiy, & Koltun (2018). Semi-parametric topological memory for navigation. ICLR 2018.

Non-parametric models: symbolic descriptions

Action/Operator:

Description:	The robot can move from x to y.
Precondition:	`ROOM(`x`)`, `ROOM(`y`)` and `at-robby(`x`)` are true.
Effect:	`at-robby(`y`)` becomes true. `at-robby(`x`)` becomes false. Everything else doesn't change.

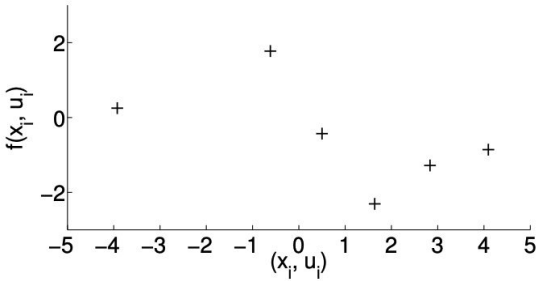
In PDDL:

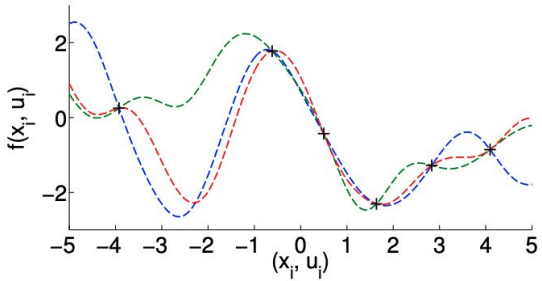
```
(:action move :parameters (?x ?y)
   :precondition (and (ROOM ?x) (ROOM ?y)
                      (at-robby ?x))
   :effect       (and (at-robby ?y)
                      (not (at-robby ?x))))
```

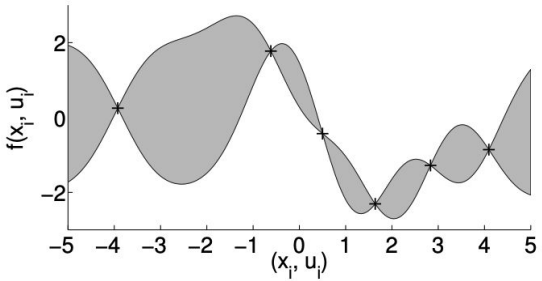
Konidaris, Kaelbling, & Lozano-Pérez (2018). From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. Journal of Artificial Intelligence Research, 61.

https://www.cs.toronto.edu/~sheila/2542/s14/A1/introtopddl2.pdf

Non-parametric models: Gaussian processes







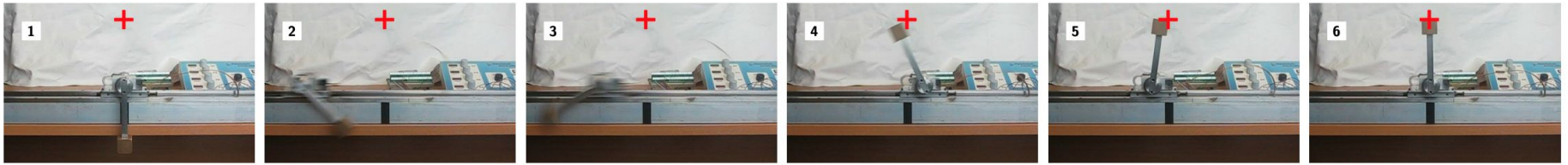


Figure 3. Real cart-pole system. Snapshots of a controlled trajectory of 20 s length after having learned the task. To solve the swing-up plus balancing, PILCO required only 17.5 s of interaction with the physical system.

Deisenroth & Rasmussen (2011). PILCO: A Model-Based and Data-Efficient Approach to Policy Search. ICML 2011.

Trade-offs for ways to model transitions

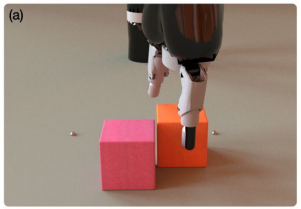
Name	Speed of learning	Speed of predictions	Domain knowledge	Long-term accuracy	Memory
Neural network	Slow	Fast	Low	Low	Low
Episodic memory	Slow	Fast	Low	Low	High
Transition graphs	Slow	Med	Low	Med	High
Symbolic description	n/a	Med	High	High	Low
Gaussian processes	Fast	Fast	Med	Med	Med

Dimensions of learned models

States vs. **Observations** vs. **Latent States**

$$s = [x, \dot{x}, \phi]$$

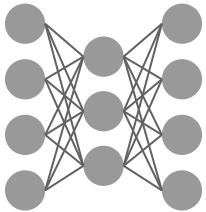
$$o =$$



$$z \in \mathbb{R}^{d_z}$$

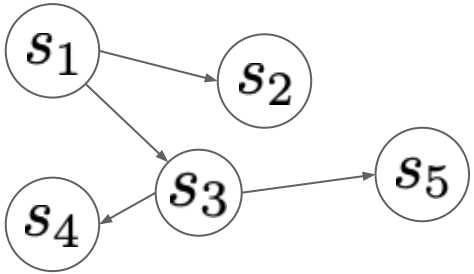
---

s_t
a_t



s_{t+1}

Parametric vs. **Non-Parametric**



Model-Based Methods in Reinforcement Learning

Part 2: Model-based Control

Igor Mordatch (Google) & Jessica Hamrick (DeepMind)

ICML 2020 Tutorial

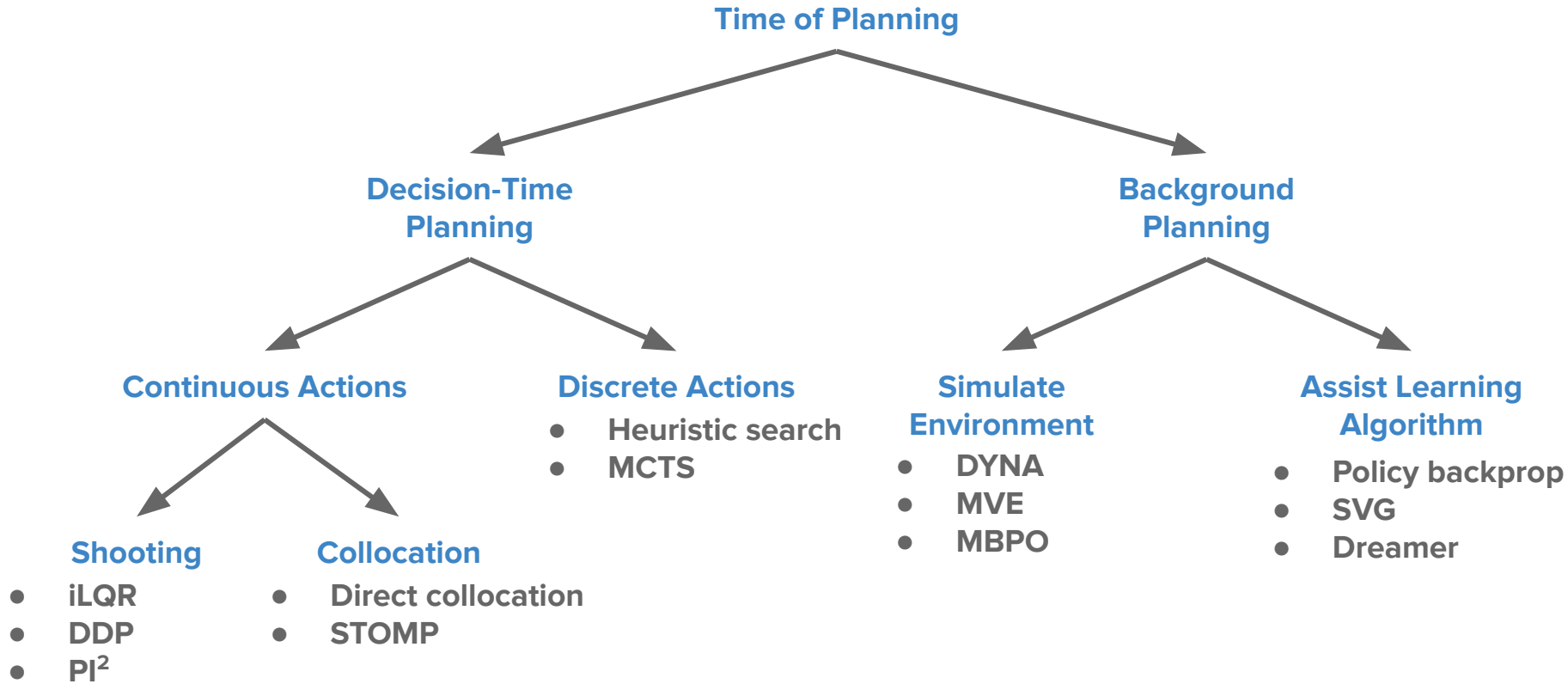
Outline

1. Introduction and motivation
2. Problem statement
3. What is a “model”?
4. **What is model-based control?**
5. Model-based control in the loop
6. What else can models be used for?
7. What’s missing from model-based methods?
8. Conclusion

Aims to understand

- How models are used in reinforcement learning
- The difference between background and decision-time planning
- The difference between discrete and continuous planning
- Variety and motivations of continuous planning methods

Landscape of planning methods



Aims to understand

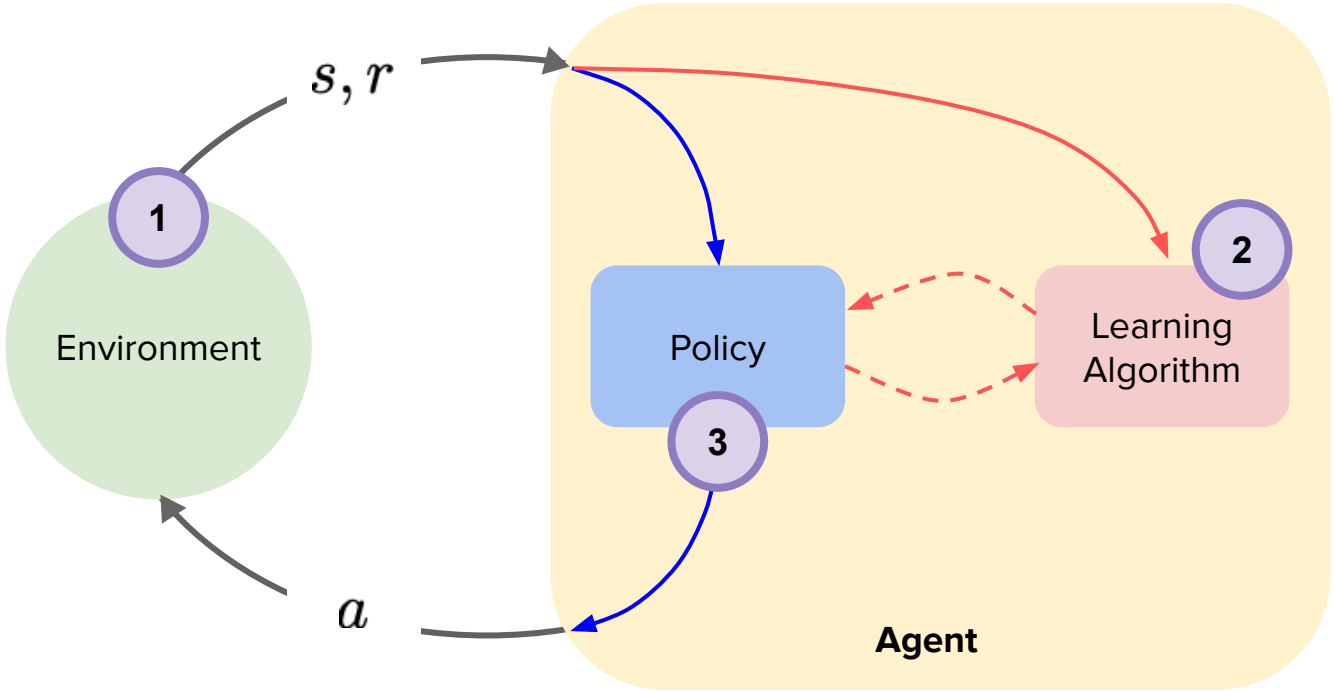
- **How models are used in reinforcement learning**
- The difference between background and decision-time planning
- The difference between discrete and continuous planning
- Variety and motivations of continuous planning methods

How models are used in reinforcement learning

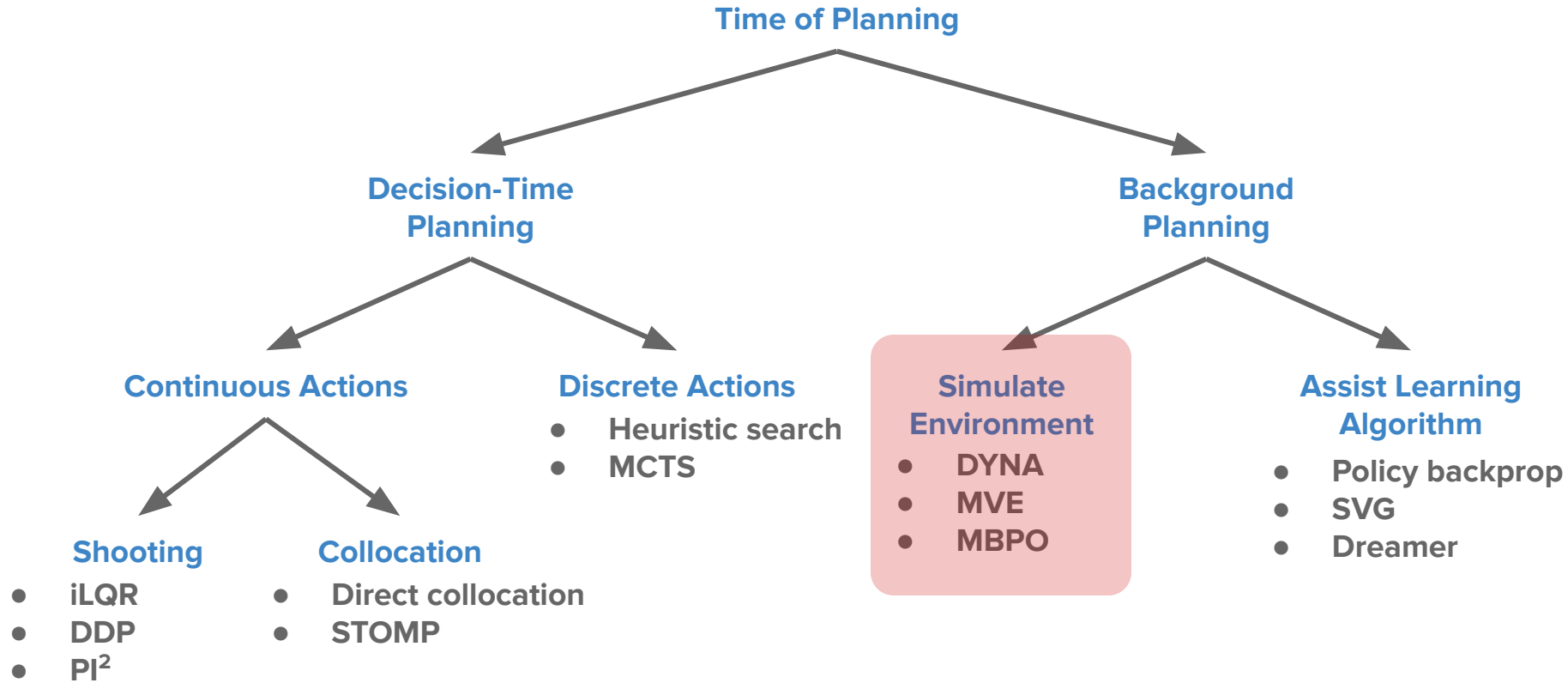
1. Simulating the environment

2. Assisting the learning algorithm

3. Strengthening the policy



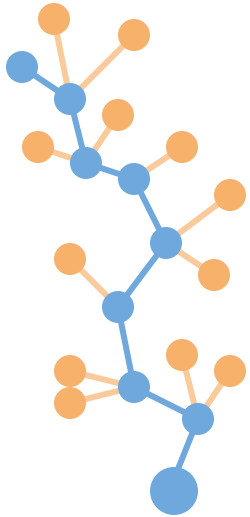
Landscape of planning methods



Simulate the Environment

Mix real and model-generated experience & apply traditional model-free methods

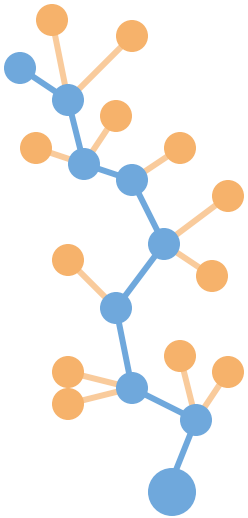
- Q-Learning
- Policy Gradient
- ...



Model offers larger, augmented training datasets

Simulate the Environment

Mix real and model-generated experience & apply traditional model-free methods



Q-Learning:

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S,Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

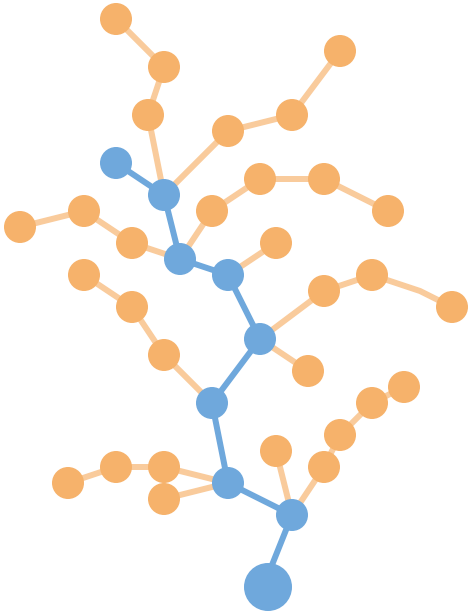
Update Q values on real transitions

& model transitions

Sutton (1990). Dyna, an integrated architecture for learning, planning, and reacting.

Simulate the Environment

Mix real and model-generated experience & apply traditional model-free methods



Policy Learning:

Algorithm 2 Model-Based Policy Optimization with Deep Reinforcement Learning

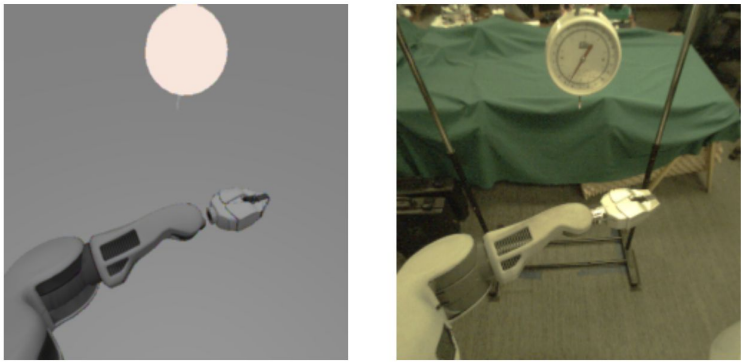
Initialize policy π_ϕ, predictive model p_θ, environment dataset $\mathcal{D}_{\text{env}}$, model dataset $\mathcal{D}_{\text{model}}$
for N epochs **do**
  Train model p_θ on $\mathcal{D}_{\text{env}}$ via maximum likelihood
  for E steps **do**
    Take action in environment according to π_ϕ; add to $\mathcal{D}_{\text{env}}$
    for M model rollouts **do**
      Sample s_t uniformly from $\mathcal{D}_{\text{env}}$
      Perform k-step model rollout starting from s_t using policy π_ϕ; add to $\mathcal{D}_{\text{model}}$
    for G gradient updates **do**
      Update policy parameters on model data: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$

Janner et al (2019). When to Trust Your Model: Model-Based Policy Optimization.

Simulate the Environment

Mix real and model-generated experience & apply traditional model-free methods

Popularized in robotics as sim2real research



Tzeng et al (2017). Adapting Deep Visuomotor Representations with Weak Pairwise Constraints.

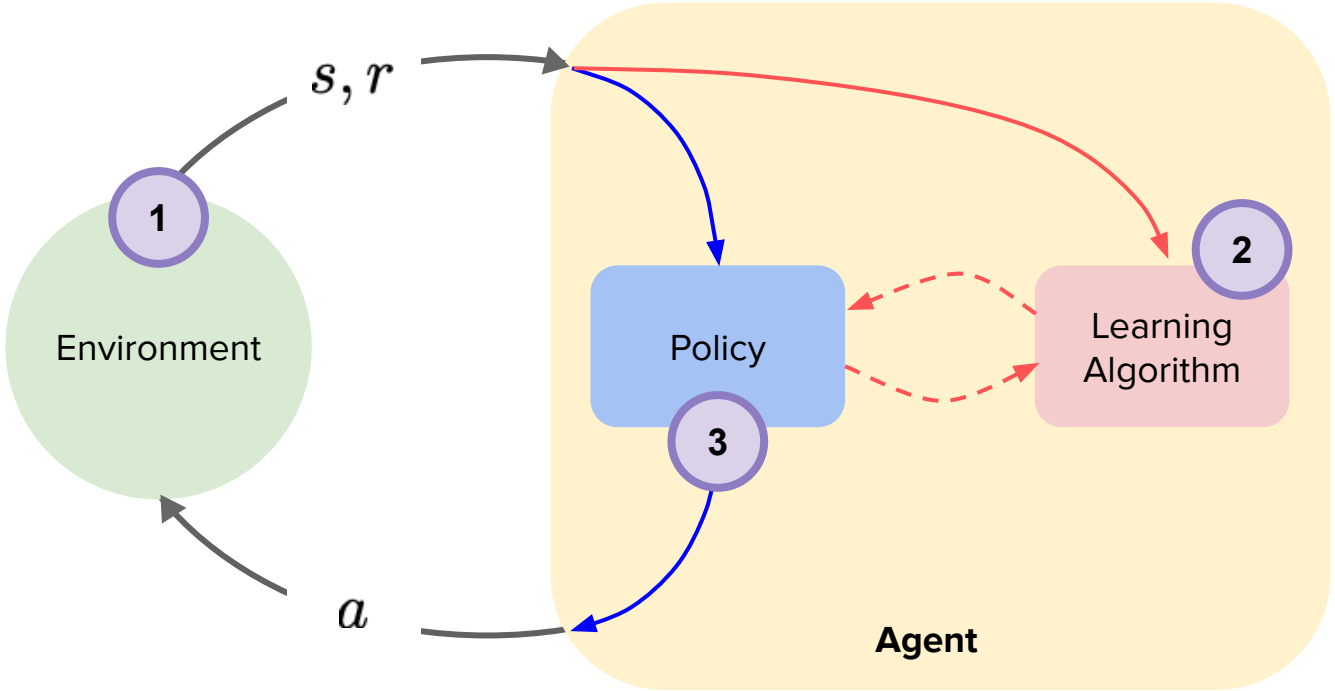
OpenAI et al (2019). Learning Dexterous In-Hand Manipulation.

How models are used in reinforcement learning

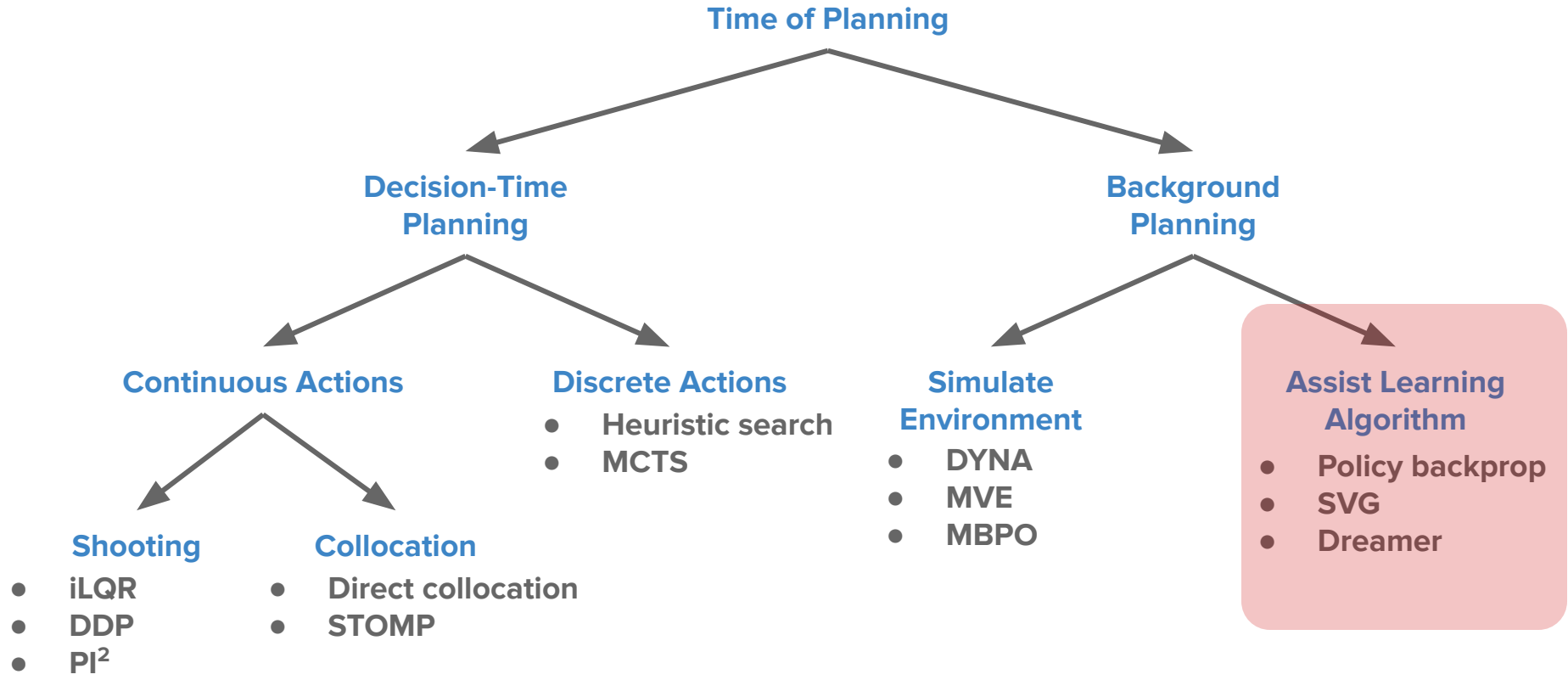
1. Simulating the environment

2. Assisting the learning algorithm

3. Strengthening the policy

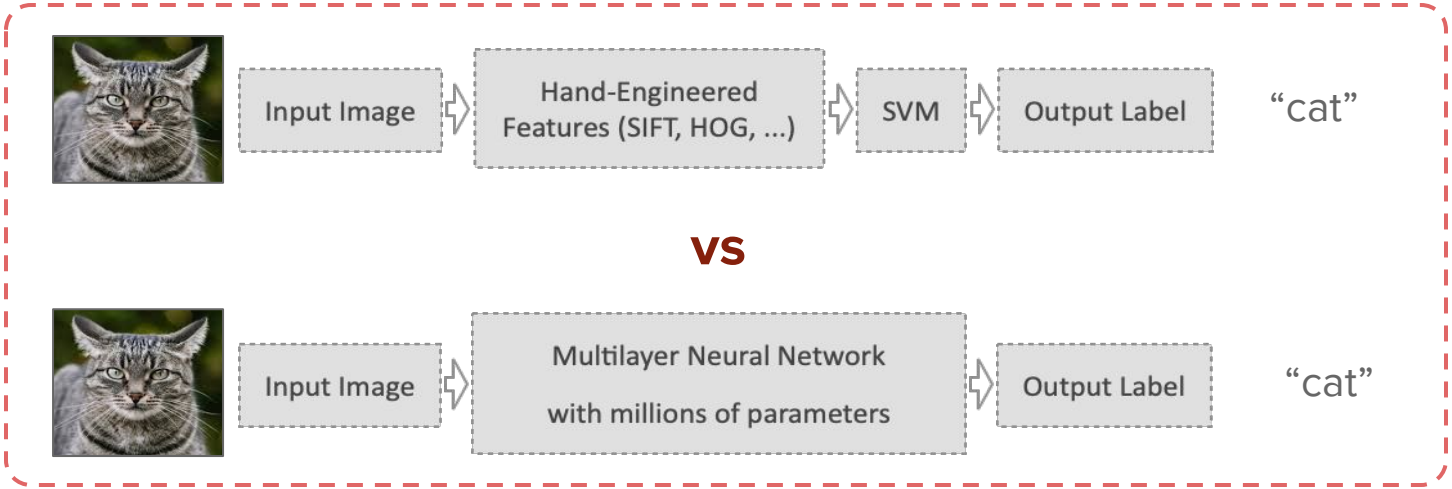


Landscape of planning methods



End-to-end training with models

Very successful in improving and simplifying supervised learning in computer vision, speech recognition, language, ...



Can we apply the same approach in reinforcement learning?

Example: policy gradient

Goal: maximize reward of parametric policy:

$$J(\theta) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t), \quad a_t = \pi_\theta(s_t), \quad s_{t+1} = T(s_, ta_t)$$

Just apply gradient ascent on policy gradient $\nabla_\theta J$

How to calculate $\nabla_\theta J$?

Example: policy gradient

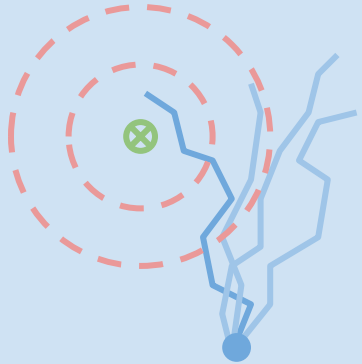
Goal: maximize reward of parametric policy:

$$J(\theta) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t), \quad a_t = \pi_\theta(s_t), \quad s_{t+1} = T(s_, ta_t)$$

Just apply gradient ascent on policy gradient $\nabla_\theta J$ **How to calculate $\nabla_\theta J$?**

Sampling (Likelihood ratio / REINFORCE)

- High-variance
- Requires stochastic policy

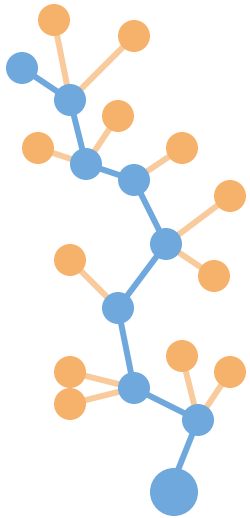


Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning.

End-to-end training with models

Aside from imaginary experience, (smooth) models offer

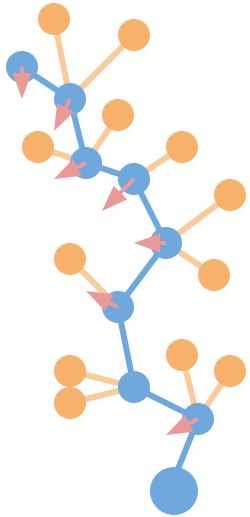
$$s_{t+1} = f_s(s_t, a_t) \quad r_t = f_r(s_t, a_t)$$



Schmidhuber (1990). Making the world differentiable.

End-to-end training with models

Aside from imaginary experience, (smooth) models offer derivatives



$$s_{t+1} = f_s(s_t, a_t) \quad r_t = f_r(s_t, a_t)$$

$$\nabla_{s_t}(s_{t+1}), \nabla_{a_t}(s_{t+1}), \nabla_{s_t}(r_t), \nabla_{a_t}(r_t), \ldots$$

i.e. "how do small changes in action change next state?"

Why is this useful?

Allows end-to-end differentiation of the learning algorithms via **back-propagation**

Example: policy gradient

Goal: maximize reward of parametric policy:

$$J(\theta) \approx \sum_{t=0}^{H} \gamma^t R(s_t, a_t), \quad \pi_\theta(s_t), \quad f_s(s_t, a_t), \quad f_r(s_t, a_t)$$

Just apply gradient ascent on policy gradient $\nabla_\theta J$

How to calculate $\nabla_\theta J$?

1. Approximate transitions and rewards with differentiable models
2. Calculate policy gradient via back-propagation-through-time (BPTT):

$$\nabla_\theta J = \sum_{t=0}^{H} \gamma^t \nabla_s f_r(s_t, a_t) \nabla_\theta s_t + \gamma^t \nabla_a f_r(s_t, a_t) \nabla_\theta \pi_\theta(s_t)$$

$$\nabla_\theta s_t = \nabla_s f_s(s_{t-1}, a_{t-t}) \nabla_\theta s_{t-1} + \nabla_a f_s(s_{t-1}, a_{s-t}) \nabla_\theta \pi_\theta(s_{t-1})$$

- Just chain rule
- Calculated recursively backwards in time (i.e. RNNs)
- Easy with auto-differentiation

Example: policy gradient

Goal: maximize reward of parametric policy:

$$J(\theta) \approx \sum_{t=0}^{H} \gamma^t r_t, \quad a_t = \pi_\theta(s_t), \quad s_{t+1} = f_s(s_t, a_t), \quad r_t = f_r(s_t, a_t)$$

Just apply gradient ascent on policy gradient $\nabla_\theta J$ **How to calculate $\nabla_\theta J$?**

1. Approximate transitions and rewards with differentiable models
2. Calculate policy gradient via back-propagation-through-time (BPTT):

+

- Deterministic (no variance)
- Long-term credit assignment

-

- Prone to local minima
- Poor conditioning (vanishing/exploding gradients)

End-to-end training with models

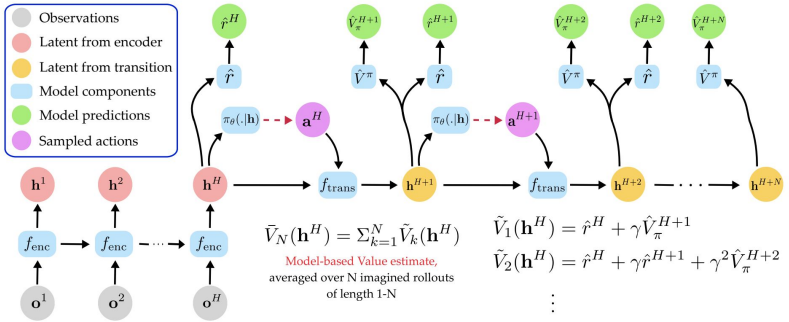
Models allow differentiation wrt any function (policy, value, ...)

Model-based back-propagation either along real or model-generated trajectories

Heess et al (2015). Learning Continuous Control Policies by Stochastic Value Gradients.

Hafner et al (2020). Dream to Control: Learning Behaviors by Latent Imagination.

Byravan et al (2020). Imagined Value Gradients.

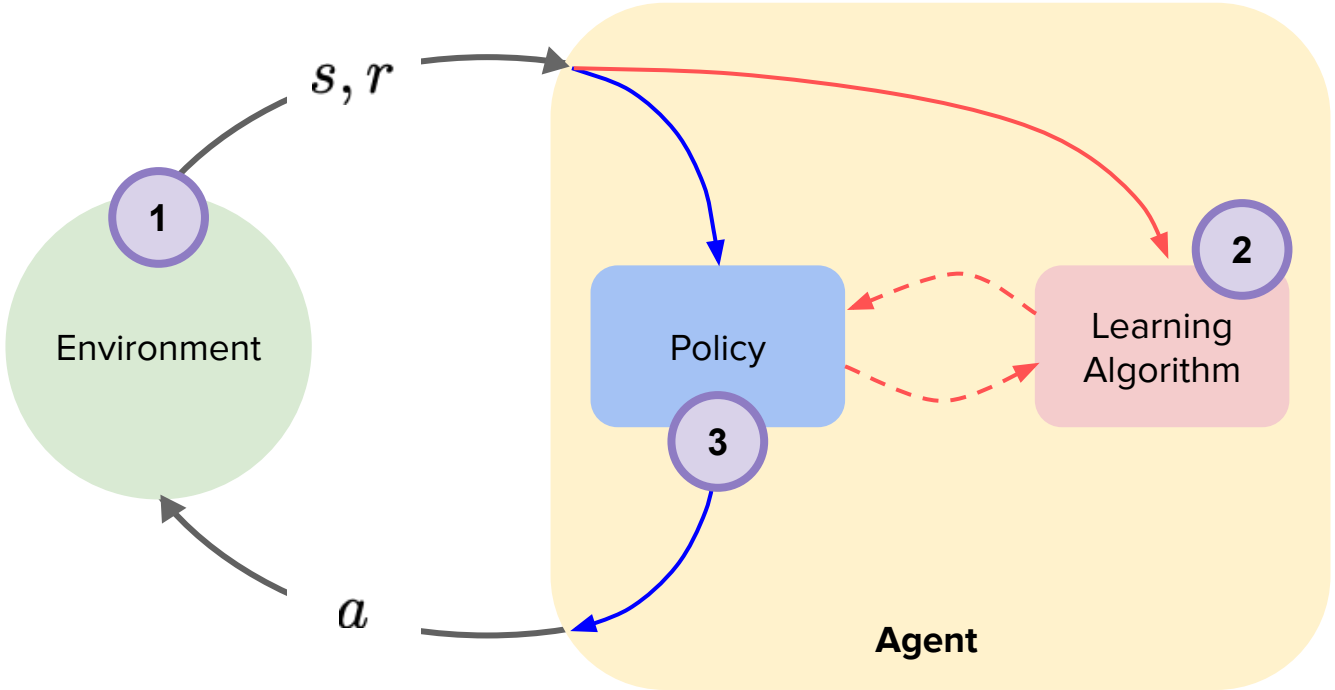


How models are used in reinforcement learning

1. Simulating the environment

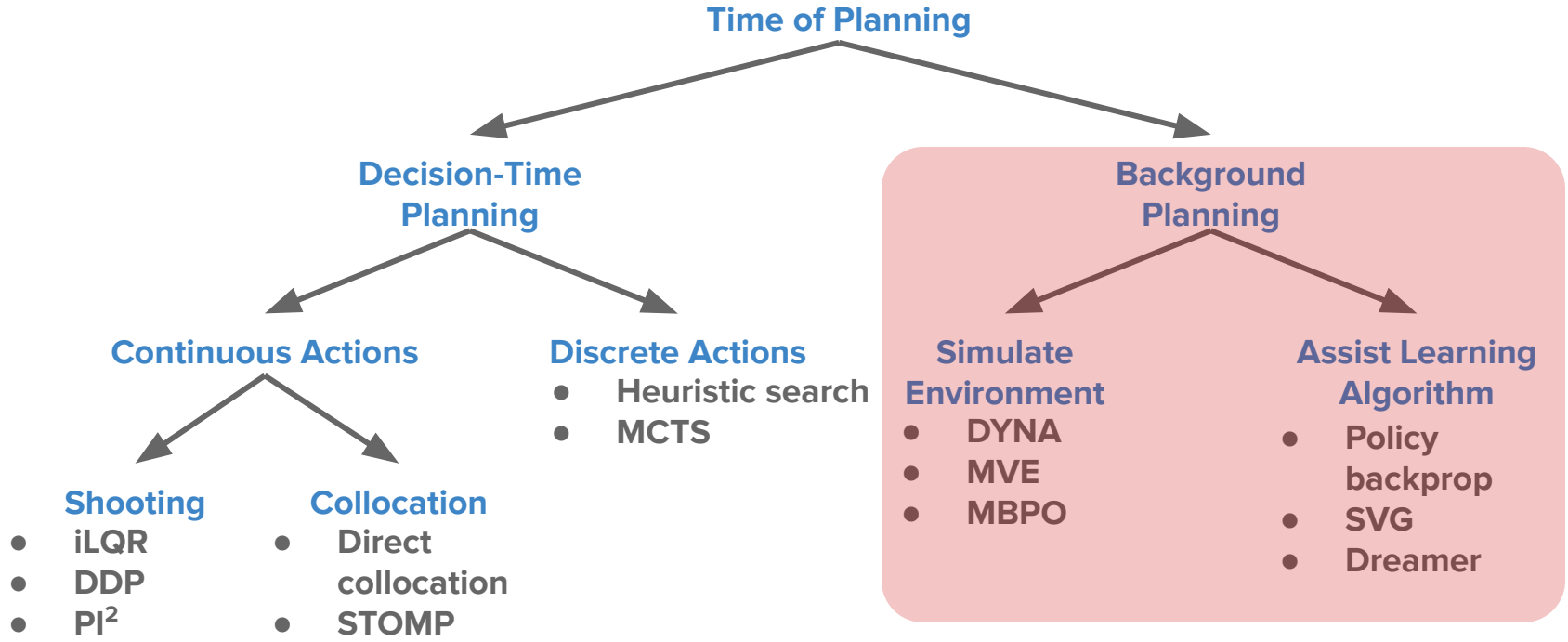
2. Assisting the learning algorithm

3. Strengthening the policy



Background vs decision-time planning

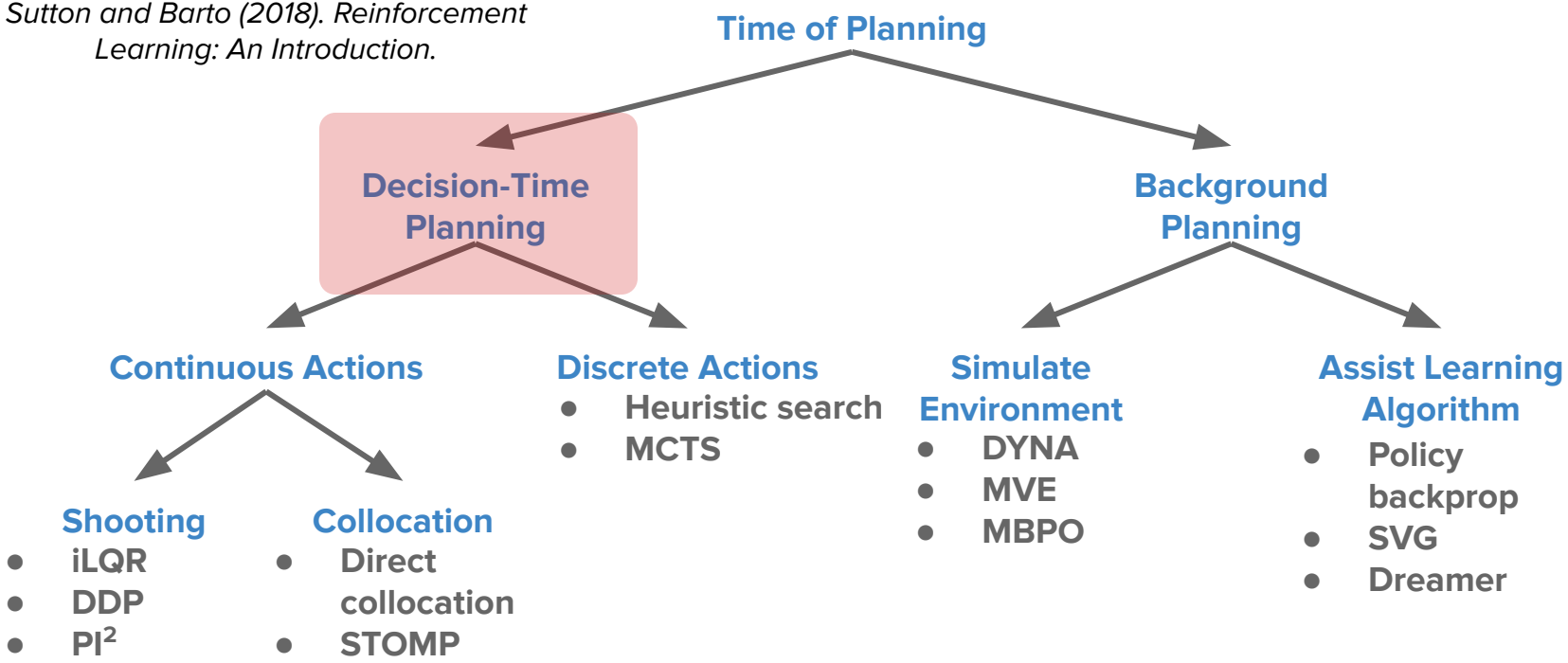
So far looked at *background planning*



Background vs decision-time planning

Decision-time planning is additional unique option for model-based setting

Sutton and Barto (2018). Reinforcement Learning: An Introduction.

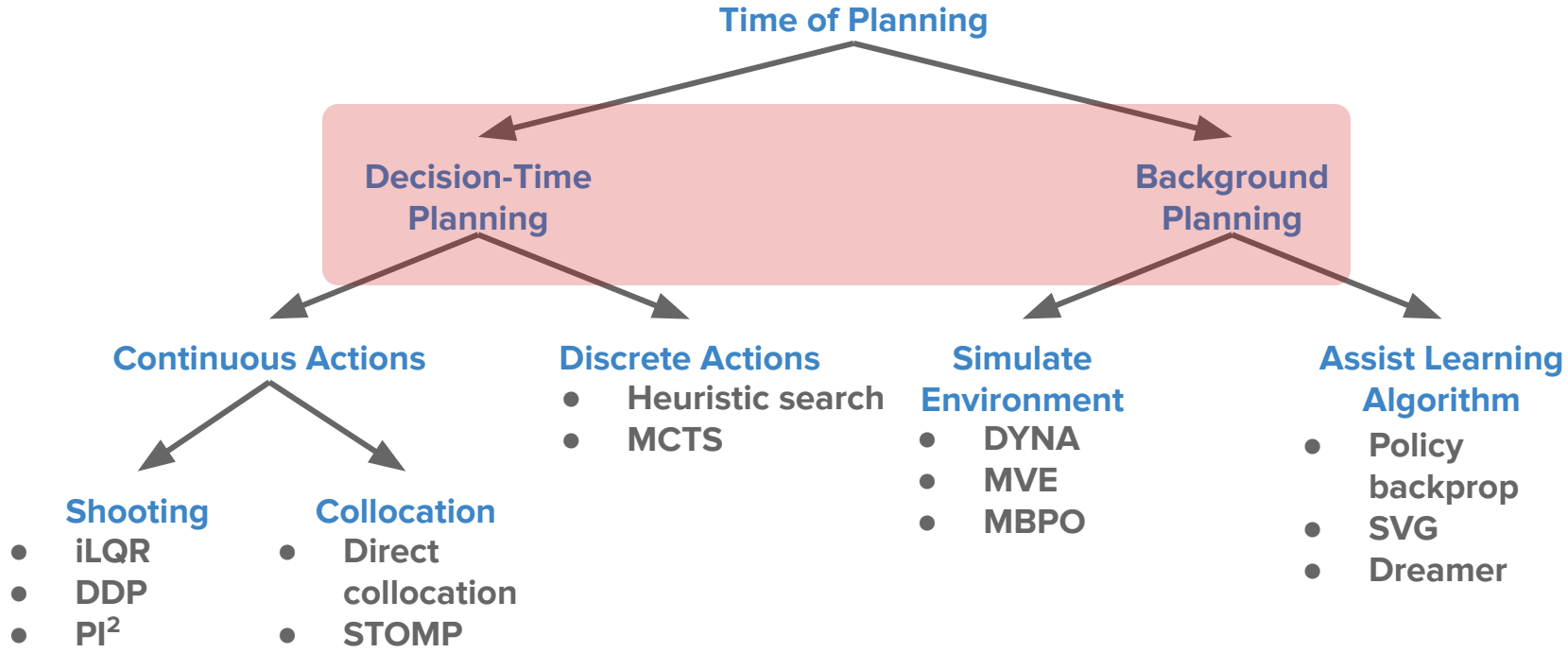


Aims to understand

- How models are used in reinforcement learning
- **The difference between background and decision-time planning**
- The difference between discrete and continuous planning
- Variety and motivations of continuous planning methods

Background vs decision-time planning

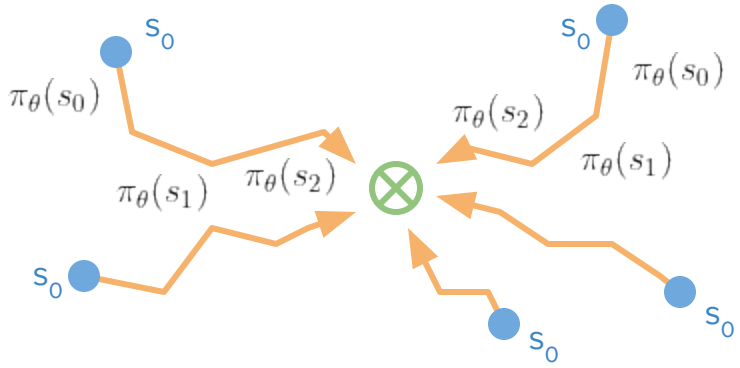
Let's look at the distinction between the two



Background vs decision-time planning

Background Planning

*"Learn how to act for **any** situation"*

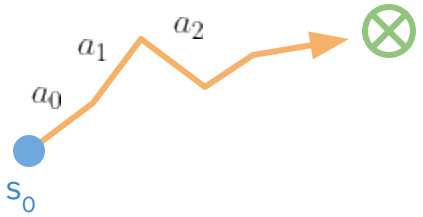


Optimization variables: θ
(parameters of policy, or value, or ...)

ex: $J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t], \quad a_t = \pi_\theta(s_t)$

Decision-Time Planning

*"Find best sequence of actions for my **current** situation"*



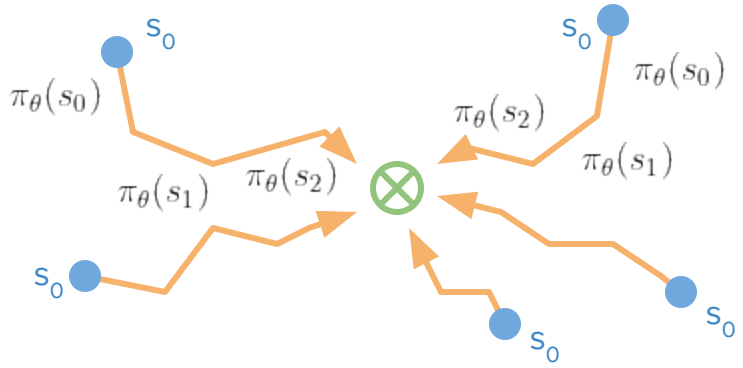
Optimization variables: $a_0, a_1, \ldots, a_H$
(sequence of actions and/or states)

ex: $J(a_0, \ldots, a_H) = \sum_{t=0}^{H} \gamma^t r_t$

Background vs decision-time planning

Background Planning

*"Learn how to act for **any** situation"*

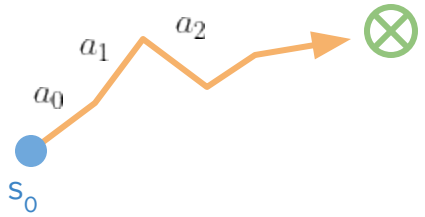


Optimization variables: θ
(parameters of policy, or value, or ...)

Habit

Decision-Time Planning

*"Find best sequence of actions for my **current** situation"*



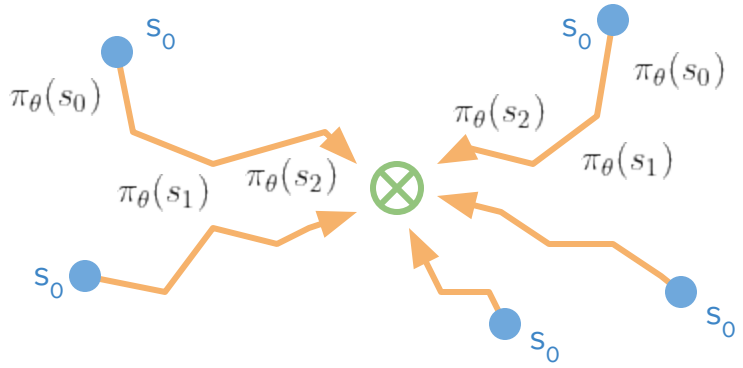
Optimization variables: $a_0, a_1, \ldots, a_H$
(sequence of actions and/or states)

Improvisation

Background vs decision-time planning

Background Planning

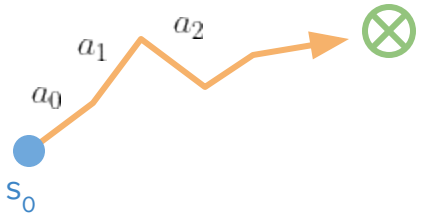
*"Learn how to act for **any** situation"*



Optimization variables: θ
(parameters of policy, or value, or ...)

Decision-Time Planning

*"Find best sequence of actions for my **current** situation"*



Optimization variables: $a_0, a_1, \ldots, a_H$
(sequence of actions and/or states)

Fast

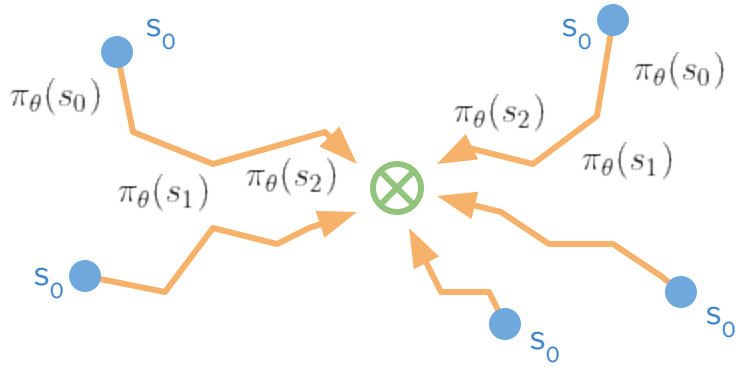
Kahneman (2011). Thinking, Fast and Slow.

Slow

Background vs decision-time planning

Background Planning

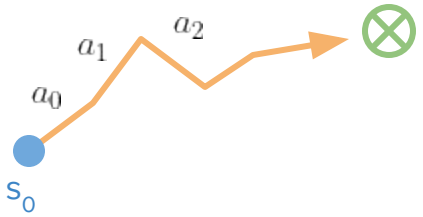
*"Learn how to act for **any** situation"*



Optimization variables: θ
(parameters of policy, or value, or ...)

Decision-Time Planning

*"Find best sequence of actions for my **current** situation"*



Optimization variables: $a_0, a_1, \ldots, a_H$
(sequence of actions and/or states)

Why use one over the other?

Background vs decision-time planning

Background	Decision-Time	
		Act on most recent state of the world
		Act without any learning
		Competent in unfamiliar situations
		Independent of observation space

Background

$$J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t], \quad a_t = \pi_\theta(s_t)$$

joint angles?
pixels?
graphs?

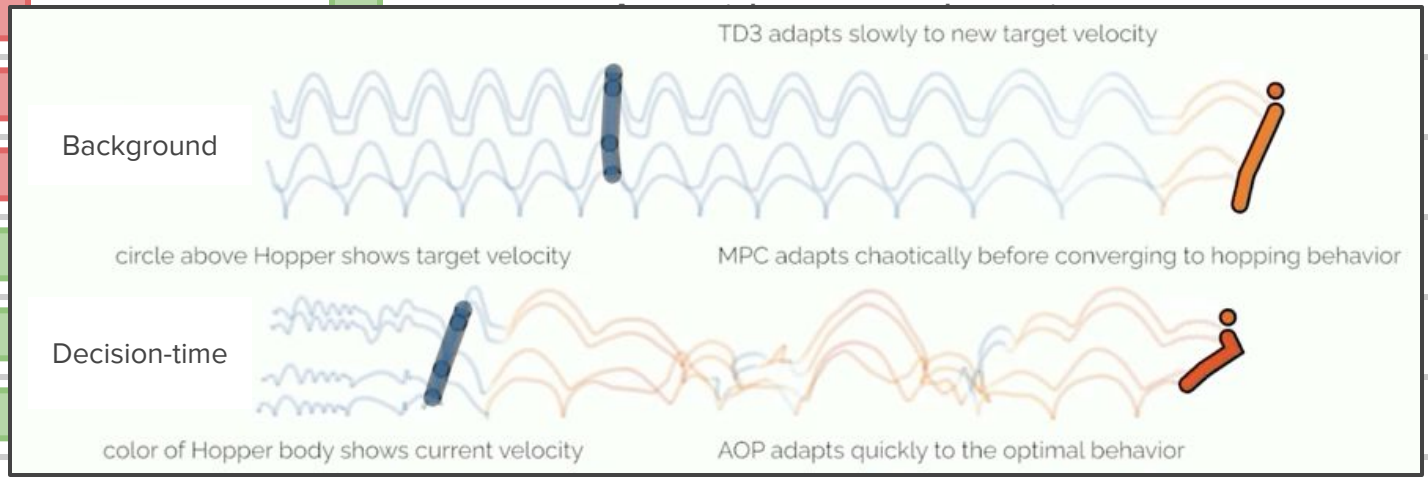
Decision-Time

$$J(a_0, ..., a_H) = \sum_{t=0}^{H} \gamma^t r_t$$

Background vs decision-time planning

Background Decision-Time

Act on most recent state of the world



Lu et al (2019). Adaptive Online Planning for Continual Lifelong Learning.

Background vs decision-time planning

Background	Decision-Time	
		Act on most recent state of the world
		Act without any learning
		Competent in unfamiliar situations
		Independent of observation space
		Partial observability
		Fast computation at deployment
		Predictability and coherence
		Same for discrete and continuous actions

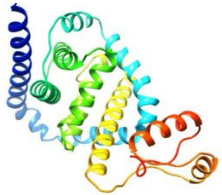
Can mix both (see next section)

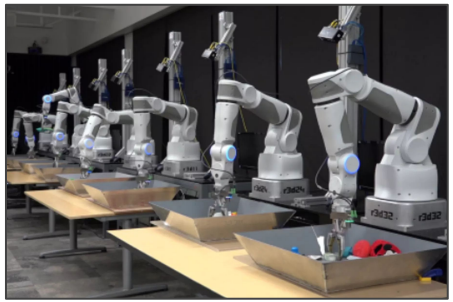
Aims to understand

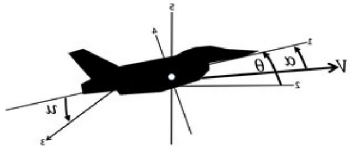
- How models are used in reinforcement learning
- The difference between background and decision-time planning
- **The difference between discrete and continuous planning**
- Variety and motivations of continuous planning methods

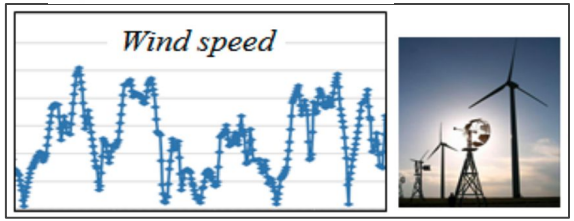
Continuous vs discrete actions

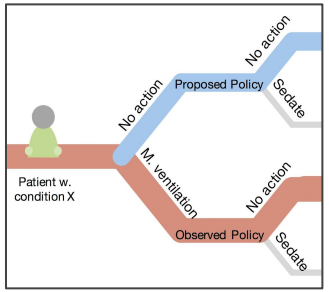
Choice depends on your problem





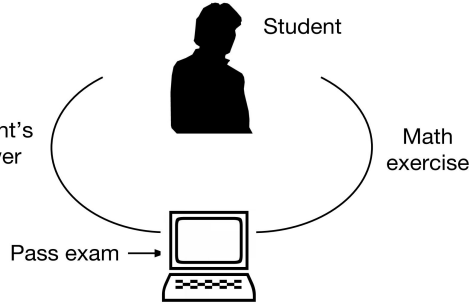




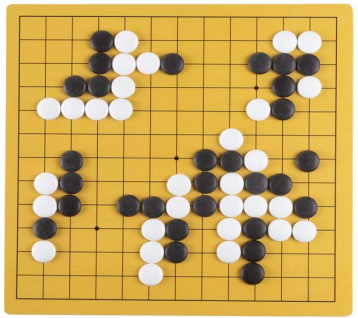


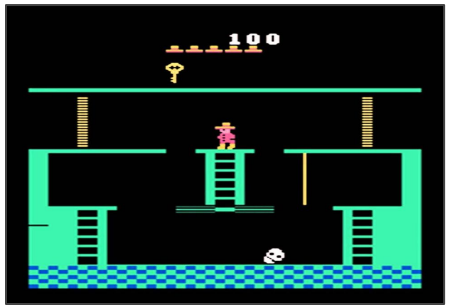
Oberst and Sontag (2019).

Student's answer



Brunskill (2017).





Continuous vs discrete actions

- Not a significant distinction for background planning
  - Learn stochastic policies that sample from discrete or continuous distributions

$$a \sim \pi(\cdot|s)$$ ← Gaussian, categorical, ...

Continuous vs discrete actions

- Not a significant distinction for background planning
  - Learn stochastic policies that sample from discrete or continuous distributions
  - Backpropagation still possible via reparameterization

Jang et al (2016). Categorical Reparameterization with Gumbel-Softmax.

Maddison et al (2016). The Concrete Distribution.

Continuous vs discrete actions

- Not a significant distinction for background planning
  - Learn stochastic policies that sample from discrete or continuous distributions
  - Backpropagation still possible via reparameterization
  - In either case, optimization problem is smooth wrt policy parameters

$$J(\theta) = \mathbb{E}_{\pi}[\sum_{t} r_t], \quad a_t \sim \pi(\cdot|s_t, \theta)$$

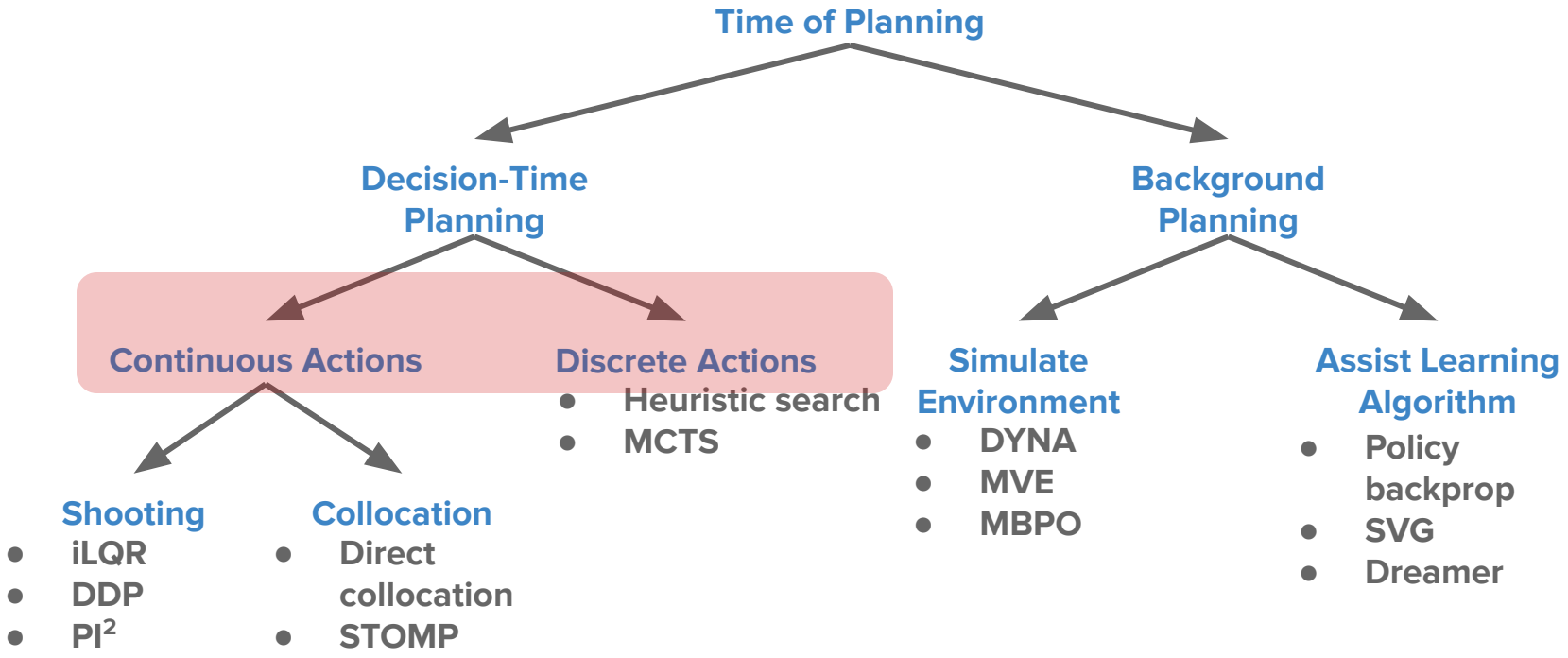
Continuous vs discrete actions

- Not a significant distinction for background planning

- But leads to specialized methods for decision-time planning
  - discrete search vs continuous trajectory optimization

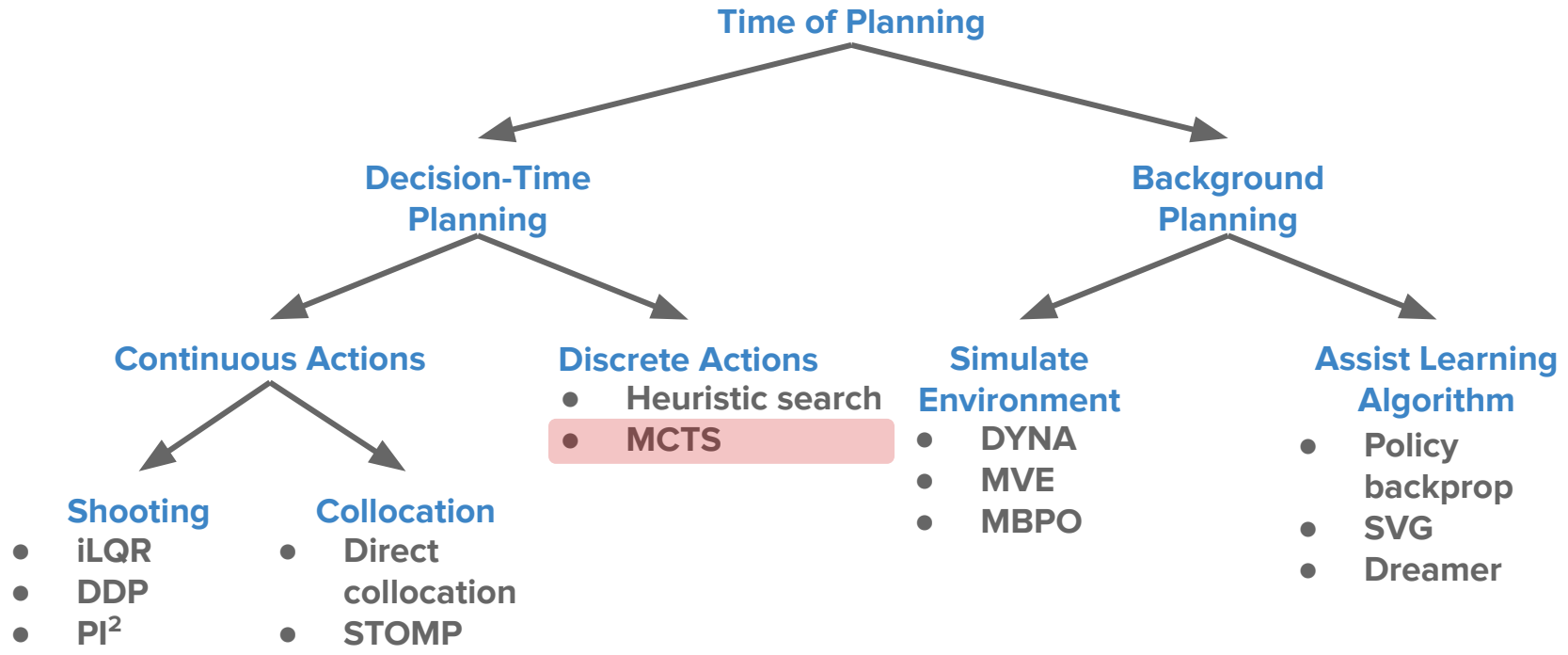
Continuous vs discrete actions

Let's look at an example method for each setting and compare/contrast



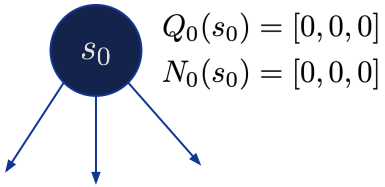
Continuous vs discrete actions

Let's look at an example method for each setting and compare/contrast



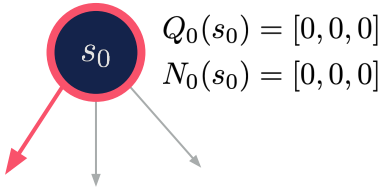
Monte-Carlo Tree Search

1. Initialize $Q_0(s,a) = 0,\ N_0(s,a) = 0,\ k = 0$



Silver et al (2017). Mastering the game of Go without human knowledge.

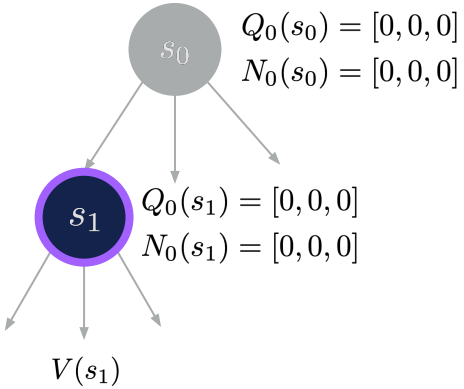
Monte-Carlo Tree Search



1. Initialize $Q_0(s, a) = 0,\ N_0(s, a) = 0,\ k = 0$
2. **Expansion**: Expand nodes according to a search policy:
   $\pi_k(s) = Q_k(s, a)$

Silver et al (2017). Mastering the game of Go without human knowledge.

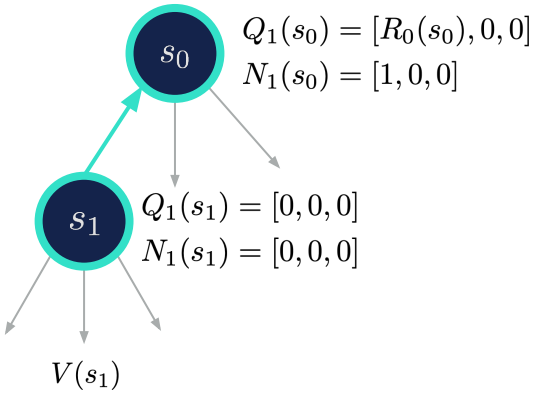
Monte-Carlo Tree Search



1. Initialize $Q_0(s, a) = 0,\ N_0(s, a) = 0,\ k = 0$
2. **Expansion:** Expand nodes according to a search policy:
   $$\pi_k(s) = Q_k(s, a)$$
3. **Evaluation:** When a new node is reached, estimate its long-term value using Monte-Carlo rollouts

Silver et al (2017). Mastering the game of Go without human knowledge.

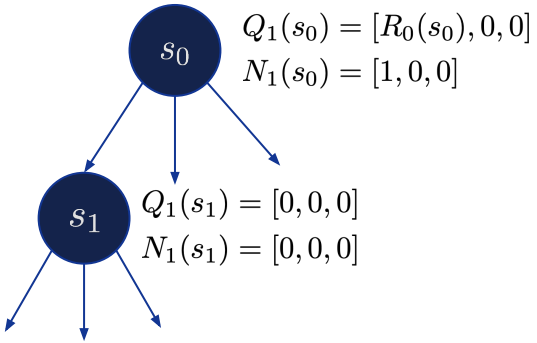
Monte-Carlo Tree Search



1. Initialize $Q_0(s,a) = 0,\ N_0(s,a) = 0,\ k = 0$
2. **Expansion:** Expand nodes according to a search policy:
   $$\pi_k(s) = Q_k(s,a)$$
3. **Evaluation:** When a new node is reached, estimate its long-term value using Monte-Carlo rollouts
4. **Backup:** For all parent nodes:
   $$Q_{k+1}(s,a) = \frac{Q_k(s,a) \cdot N_k(s,a) + R}{N_k(s,a) + 1}$$
   $$N_{k+1}(s,a) = N_k(s,a) + 1$$

Silver et al (2017). Mastering the game of Go without human knowledge.

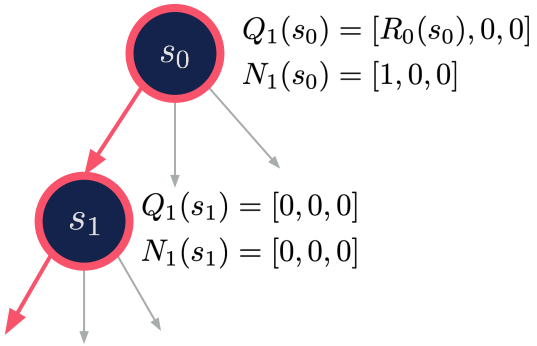
Monte-Carlo Tree Search



1. Initialize $Q_0(s,a) = 0,\ N_0(s,a) = 0,\ k = 0$
2. **Expansion:** Expand nodes according to a search policy:
   $$\pi_k(s) = Q_k(s,a)$$
3. **Evaluation:** When a new node is reached, estimate its long-term value using Monte-Carlo rollouts
4. **Backup:** For all parent nodes:
   $$Q_{k+1}(s,a) = \frac{Q_k(s,a) \cdot N_k(s,a) + R}{N_k(s,a) + 1}$$
   $$N_{k+1}(s,a) = N_k(s,a) + 1$$
5. Repeat Steps 2-4 until search budget is exhausted.
   $$k = k + 1$$

Silver et al (2017). Mastering the game of Go without human knowledge.

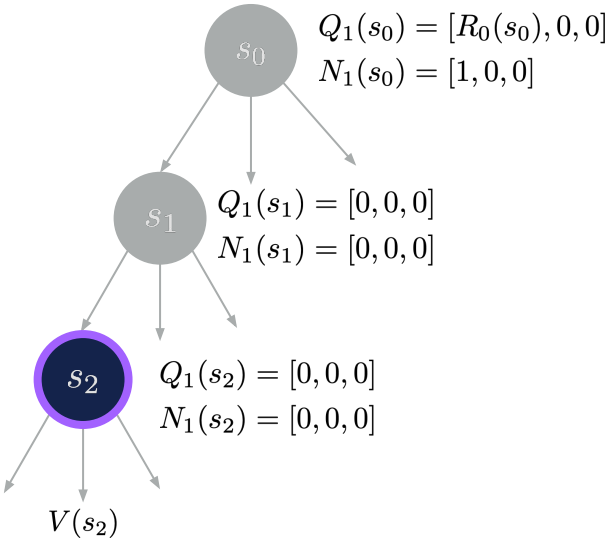
Monte-Carlo Tree Search



1. Initialize $Q_0(s,a) = 0,\ N_0(s,a) = 0,\ k = 0$
2. **Expansion:** Expand nodes according to a search policy:
   $$\pi_k(s) = Q_k(s,a)$$
3. **Evaluation:** When a new node is reached, estimate its long-term value using Monte-Carlo rollouts
4. **Backup:** For all parent nodes:
   $$Q_{k+1}(s,a) = \frac{Q_k(s,a) \cdot N_k(s,a) + R}{N_k(s,a) + 1}$$
   $$N_{k+1}(s,a) = N_k(s,a) + 1$$
5. Repeat Steps 2-4 until search budget is exhausted.
   $$k = k + 1$$

Silver et al (2017). Mastering the game of Go without human knowledge.

Monte-Carlo Tree Search



1. Initialize $Q_0(s,a) = 0,\ N_0(s,a) = 0,\ k = 0$

2. **Expansion:** Expand nodes according to a search policy:

   $$\pi_k(s) = Q_k(s,a)$$

3. **Evaluation:** When a new node is reached, estimate its long-term value using Monte-Carlo rollouts

4. **Backup:** For all parent nodes:

   $$Q_{k+1}(s,a) = \frac{Q_k(s,a) \cdot N_k(s,a) + R}{N_k(s,a) + 1}$$

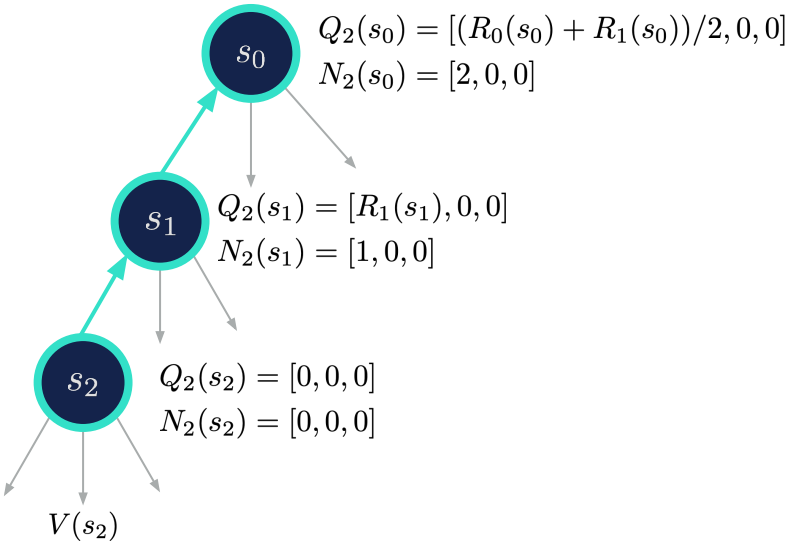
   $$N_{k+1}(s,a) = N_k(s,a) + 1$$

5. Repeat Steps 2-4 until search budget is exhausted.

   $$k = k + 1$$

Silver et al (2017). Mastering the game of Go without human knowledge.

Monte-Carlo Tree Search



1. Initialize $Q_0(s,a) = 0,\ N_0(s,a) = 0,\ k = 0$

2. **Expansion:** Expand nodes according to a search policy:

   $$\pi_k(s) = Q_k(s,a)$$

3. **Evaluation:** When a new node is reached, estimate its long-term value using Monte-Carlo rollouts

4. **Backup:** For all parent nodes:

   $$Q_{k+1}(s,a) = \frac{Q_k(s,a) \cdot N_k(s,a) + R}{N_k(s,a) + 1}$$

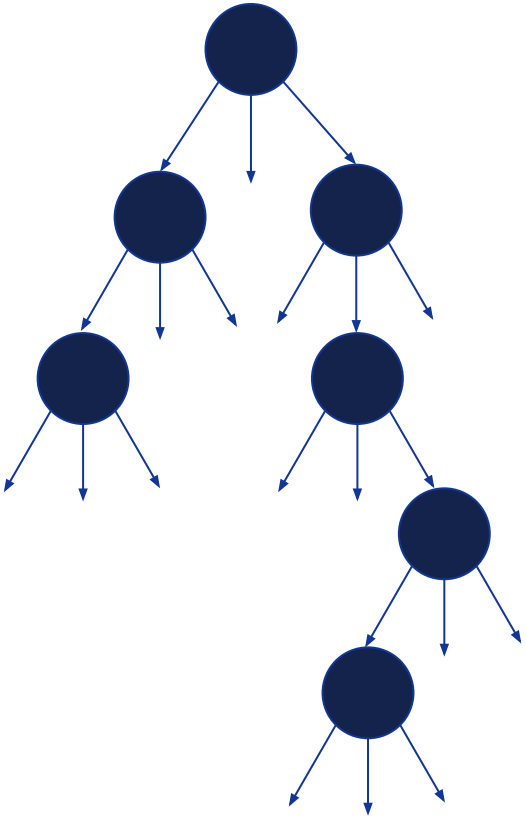
   $$N_{k+1}(s,a) = N_k(s,a) + 1$$

5. Repeat Steps 2-4 until search budget is exhausted.

   $$k = k + 1$$

Silver et al (2017). Mastering the game of Go without human knowledge.

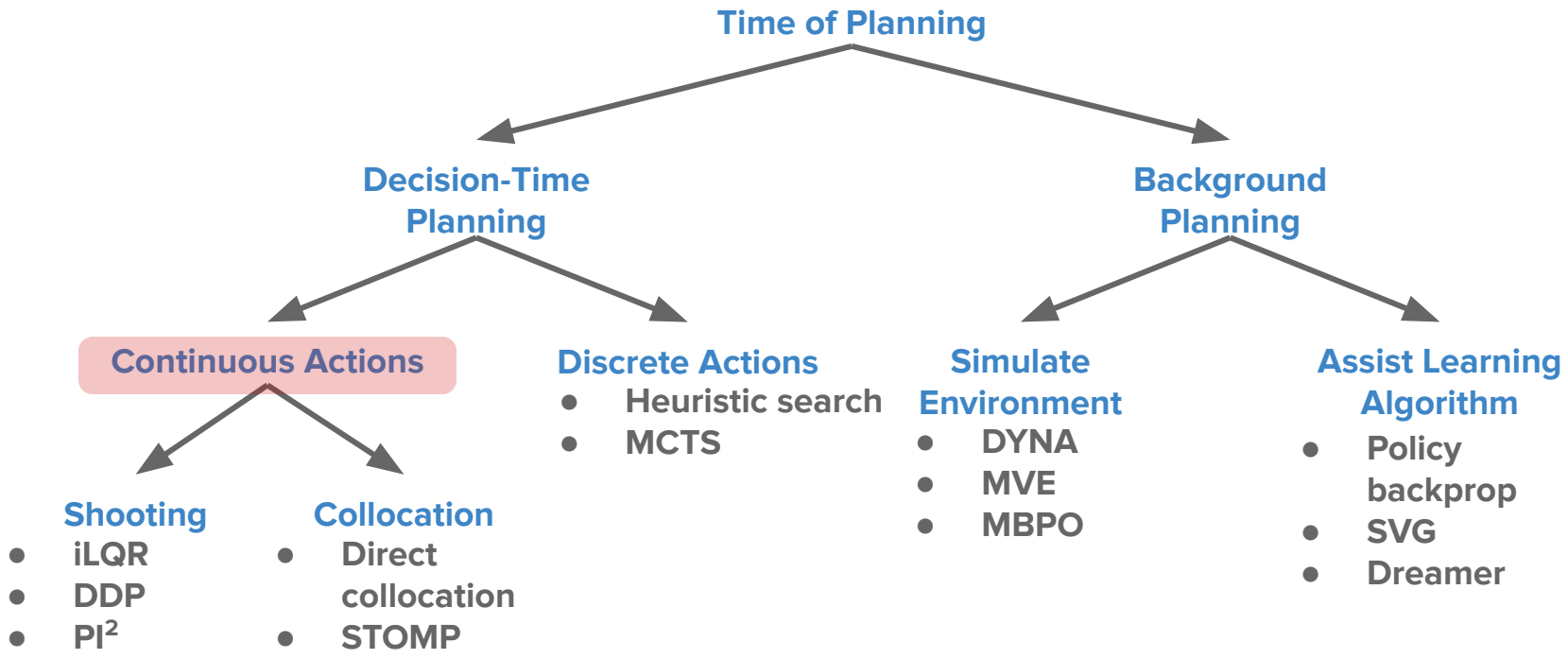
Monte-Carlo Tree Search



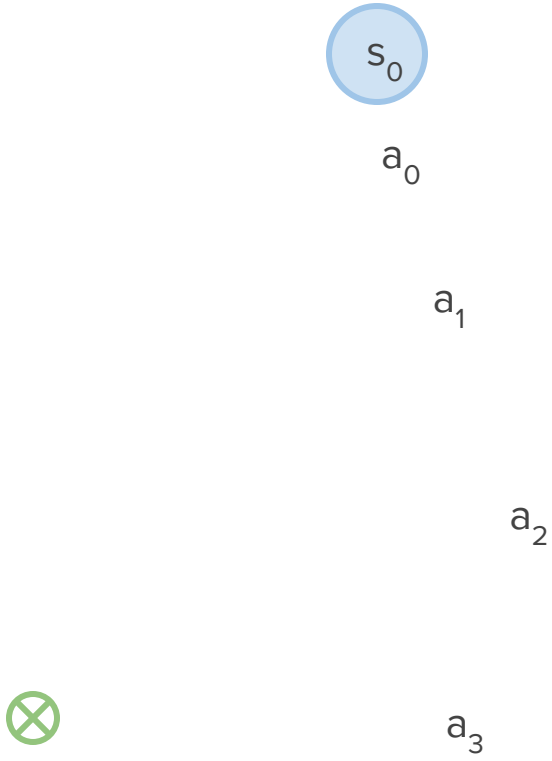
1. Initialize $Q_0(s,a) = 0,\ N_0(s,a) = 0,\ k = 0$
2. **Expansion:** Expand nodes according to a search policy:
   $$\pi_k(s) = Q_k(s,a)$$
3. **Evaluation:** When a new node is reached, estimate its long-term value using Monte-Carlo rollouts
4. **Backup:** For all parent nodes:
   $$Q_{k+1}(s,a) = \frac{Q_k(s,a) \cdot N_k(s,a) + R}{N_k(s,a) + 1}$$
   $$N_{k+1}(s,a) = N_k(s,a) + 1$$
5. Repeat Steps 2-4 until search budget is exhausted.
   $$k = k + 1$$

Silver et al (2017). Mastering the game of Go without human knowledge.

Trajectory Optimization

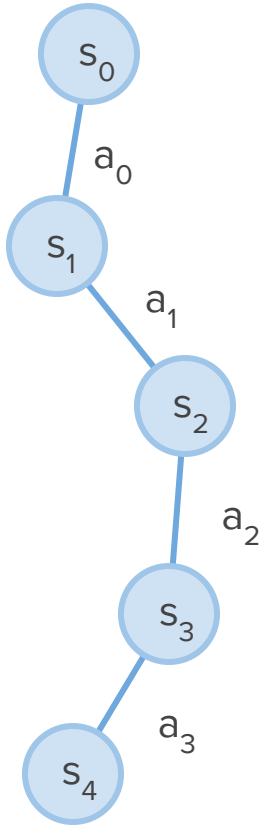


Trajectory Optimization



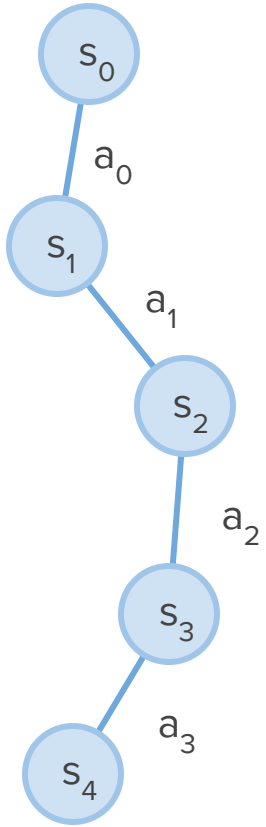
1. Initialize $a_0, ..., a_H$ from guess

Trajectory Optimization



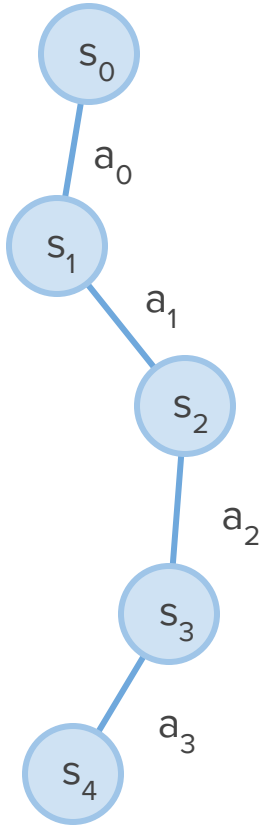
1. Initialize $a_0, ..., a_H$ from guess
2. **Expansion:** Execute actions $\mathbf{a} = a_0, ..., a_H$ to get states: $s_1, ..., s_H$

Trajectory Optimization



1. Initialize $a_0, ..., a_H$ from guess
2. **Expansion:** Execute actions $\mathbf{a} = a_0, ..., a_H$ to get states: $s_1, ..., s_H$
3. **Evaluation:** get trajectory reward $J(\mathbf{a}) = \sum_{t=0}^{H} r_t$

Trajectory Optimization



1. Initialize $a_0, ..., a_H$ from guess
2. **Expansion:** Execute actions $\mathbf{a} = a_0, ..., a_H$ to get states: $s_1, ..., s_H$
3. **Evaluation:** get trajectory reward $J(\mathbf{a}) = \sum_{t=0}^{H} r_t$
4. **Back-propagation:** get recursive gradients

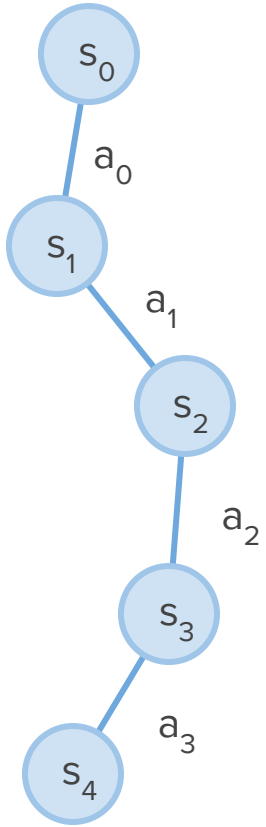
$$\nabla_{\mathbf{a}} J = \sum_{t=0}^{H} \nabla_{\mathbf{a}} r_t$$

$$\nabla_{\mathbf{a}} r_t = \nabla_s f_r(s_t, a_t) \nabla_{\mathbf{a}} s_t + \nabla_a f_r(s_t, a_t)$$

$$\nabla_{\mathbf{a}} s_t = \nabla_a f_s(s_{t-1}, a_{t-1}) + \nabla_s f_s(s_{t-1}, a_{t-1}) \nabla_{\mathbf{a}} s_{t-1}$$

$$\nabla_{\mathbf{a}} s_{t-1} = ...$$

Trajectory Optimization



1. Initialize $a_0, ..., a_H$ from guess
2. **Expansion:** Execute actions $\mathbf{a} = a_0, ..., a_H$ to get states: $s_1, ..., s_H$
3. **Evaluation:** get trajectory reward $J(\mathbf{a}) = \sum_{t=0}^{H} r_t$
4. **Back-propagation:** get recursive gradients

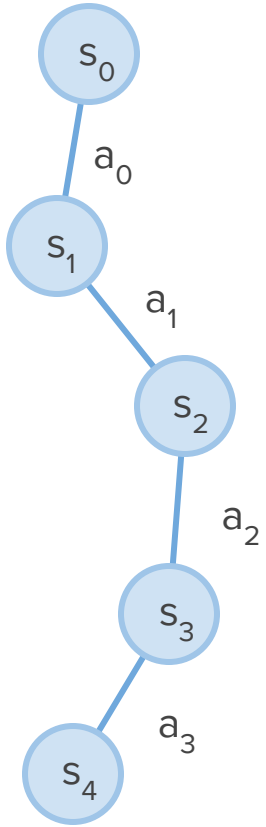
$$\nabla_{\mathbf{a}} J = \sum_{t=0}^{H} \nabla_{\mathbf{a}} r_t$$ *reward model derivatives*

$$\nabla_{\mathbf{a}} r_t = \nabla_s f_r(s_t, a_t) \nabla_{\mathbf{a}} s_t + \nabla_a f_r(s_t, a_t)$$

$$\nabla_{\mathbf{a}} s_t = \nabla_a f_s(s_{t-1}, a_{t-1}) + \nabla_s f_s(s_{t-1}, a_{t-1}) \nabla_{\mathbf{a}} s_{t-1}$$

$$\nabla_{\mathbf{a}} s_{t-1} = \ldots$$ *transition model derivatives*

Trajectory Optimization



1. Initialize $a_0, ..., a_H$ from guess
2. **Expansion:** Execute actions $\mathbf{a} = a_0, ..., a_H$ to get states: $s_1, ..., s_H$
3. **Evaluation:** get trajectory reward $J(\mathbf{a}) = \sum_{t=0}^{H} r_t$
4. **Back-propagation:** get recursive gradients

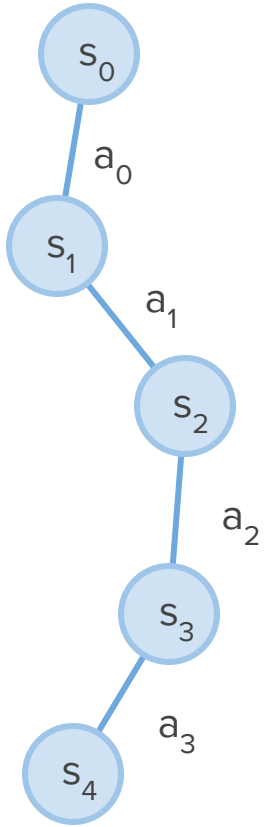
$$\nabla_{\mathbf{a}} J = \sum_{t=0}^{H} \nabla_{\mathbf{a}} r_t$$

$$\nabla_{\mathbf{a}} r_t = \nabla_s f_r(s_t, a_t) \nabla_{\mathbf{a}} s_t + \nabla_a f_r(s_t, a_t)$$

$$\nabla_{\mathbf{a}} s_t = \nabla_a f_s(s_{t-1}, a_{t-1}) + \nabla_s f_s(s_{t-1}, a_{t-1}) \nabla_{\mathbf{a}} s_{t-1}$$

$$\nabla_{\mathbf{a}} s_{t-1} = \ldots$$ *(computed recursively)*

Trajectory Optimization



1. Initialize $a_0, ..., a_H$ from guess
2. **Expansion:** Execute actions $\mathbf{a} = a_0, ..., a_H$ to get states: $s_1, ..., s_H$
3. **Evaluation:** get trajectory reward $J(\mathbf{a}) = \sum_{t=0}^{H} r_t$
4. **Back-propagation:** get recursive gradients

$$\nabla_{\mathbf{a}} J = \sum_{t=0}^{H} \nabla_{\mathbf{a}} r_t$$

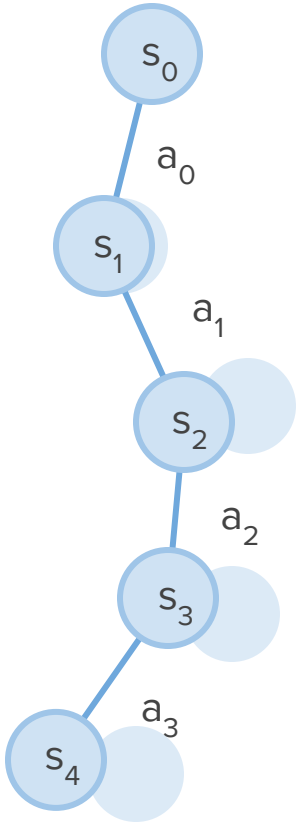
$$\nabla_{\mathbf{a}} r_t = \nabla_s f_r(s_t, a_t) \nabla_{\mathbf{a}} s_t + \nabla_a f_r(s_t, a_t)$$

$$\nabla_{\mathbf{a}} s_t = \nabla_a f_s(s_{t-1}, a_{t-1}) + \nabla_s f_s(s_{t-1}, a_{t-1}) \nabla_{\mathbf{a}} s_{t-1}$$

$$\nabla_{\mathbf{a}} s_{t-1} = ...$$

Easily available with auto-differentiation

Trajectory Optimization



1. Initialize $a_0, ..., a_H$ from guess
2. **Expansion:** Execute actions $\mathbf{a} = a_0, ..., a_H$ to get states: $s_1, ..., s_H$
3. **Evaluation:** get trajectory reward $J(\mathbf{a}) = \sum_{t=0}^{H} r_t$
4. **Back-propagation:** get recursive gradients

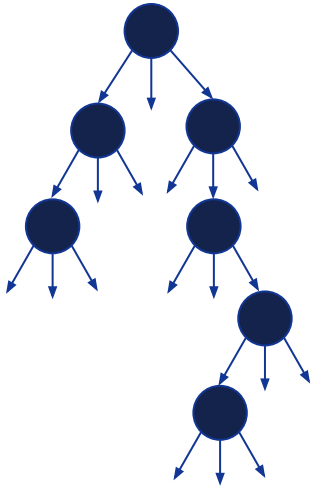
$$\nabla_{\mathbf{a}} J = \sum_{t=0}^{H} \nabla_{\mathbf{a}} r_t$$

$$\nabla_{\mathbf{a}} r_t = \nabla_s f_r(s_t, a_t) \nabla_{\mathbf{a}} s_t + \nabla_a f_r(s_t, a_t)$$

$$\nabla_{\mathbf{a}} s_t = \nabla_a f_s(s_{t-1}, a_{t-1}) + \nabla_s f_s(s_{t-1}, a_{t-1}) \nabla_{\mathbf{a}} s_{t-1}$$

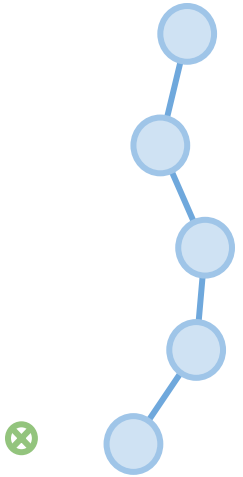
$$\nabla_{\mathbf{a}} s_{t-1} = ...$$

5. Update all actions via gradient ascent $\mathbf{a} \leftarrow \mathbf{a} + \nabla_{\mathbf{a}} J$ and repeat Steps 2-5.



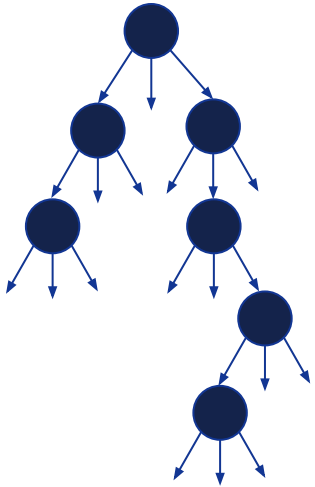
Discrete: Monte-Carlo tree search

- Consider multiple action paths and store their returns



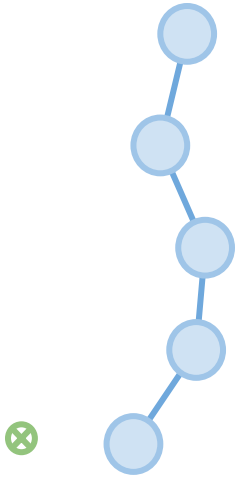
Continuous: Trajectory optimization

- Consider only one action paths and store its return



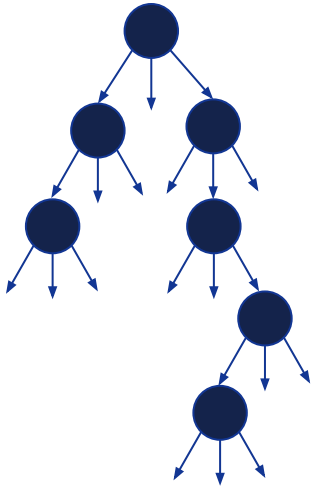
Discrete: Monte-Carlo tree search

- Consider multiple action paths and store their returns
- Store and update tree of Q values
- Expand tree by selecting actions via policy / Q values



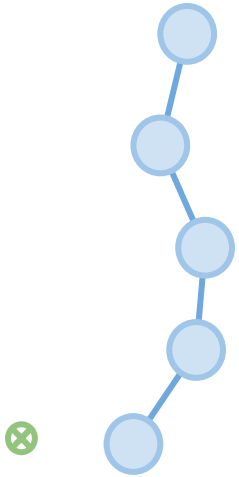
Continuous: Trajectory optimization

- Consider only one action paths and store its return
- Store and update optimal action sequence
- Update action sequence by gradient ascent



Discrete: Monte-Carlo tree search

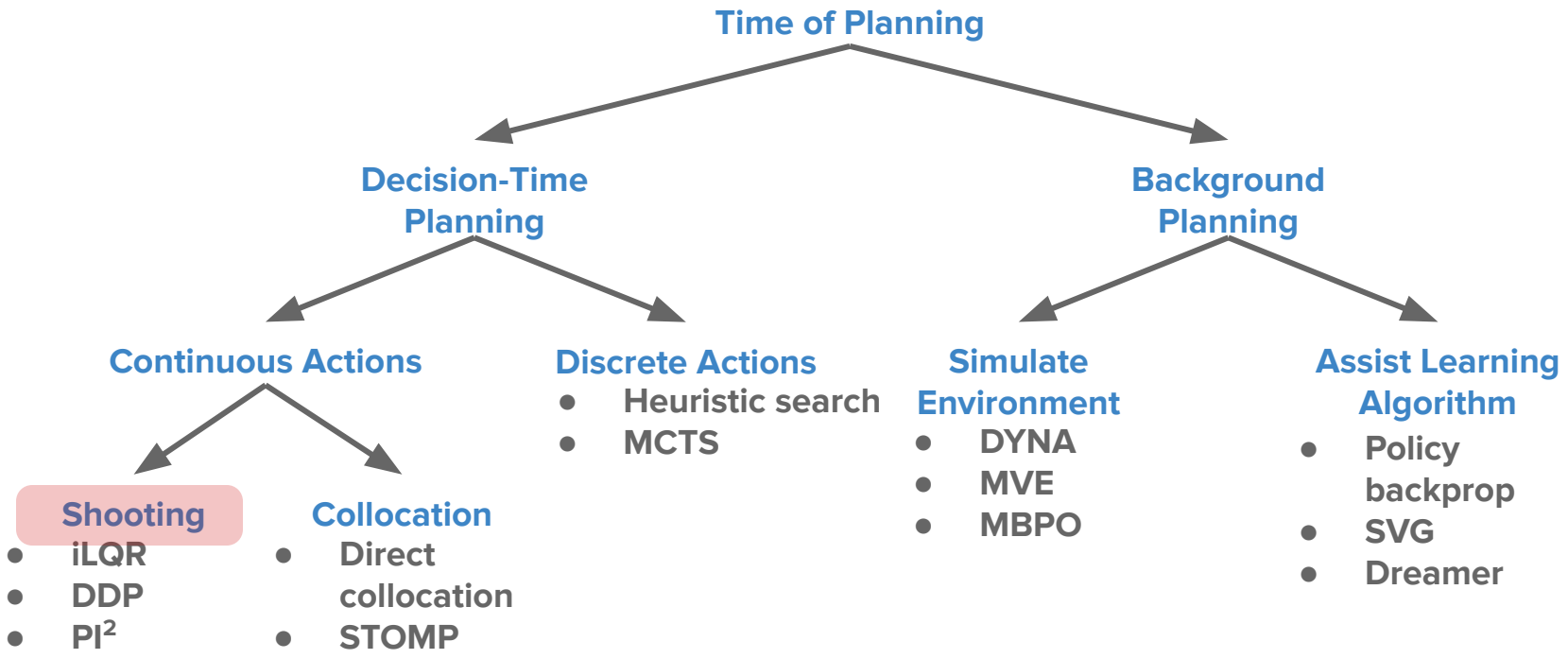
- Consider multiple action paths and store their returns
- Store and update tree of Q values
- Expand tree by selecting actions via policy / Q values
- Typically cannot expand entire tree (myopic solution)



Continuous: Trajectory optimization

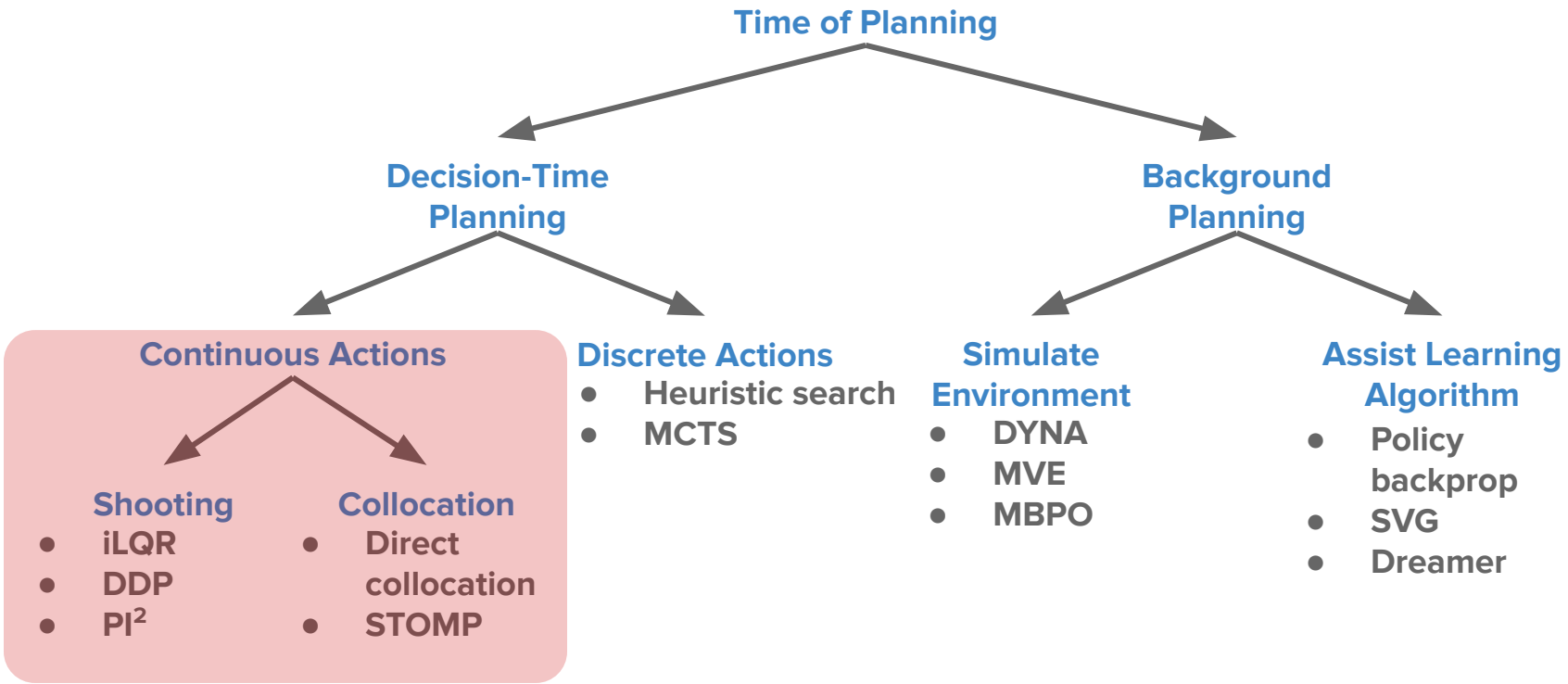
- Consider only one action paths and store its return
- Store and update optimal action sequence
- Update action sequence by gradient ascent
- Only guaranteed to reach local maximum

Continuous trajectory optimization



Continuous trajectory optimization

Broad range of methods



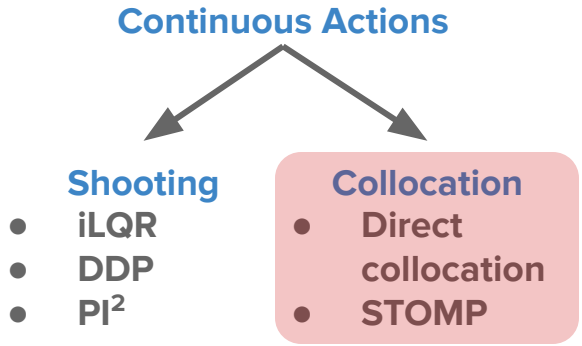
Aims to understand

- How models are used in reinforcement learning
- The difference between background and decision-time planning
- The difference between discrete and continuous planning
- **Variety and motivations of continuous planning methods**

Why so many methods?

Motivated by issues with a baseline method we looked at

- Sensitivity and poor conditioning
- Only reaches local optimum
- Slow convergence

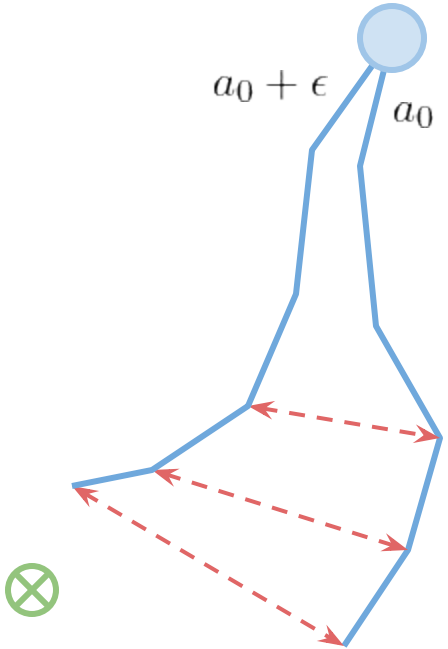


Addressing each leads to a different class of methods

Sensitivity and poor conditioning

Shooting: small changes in early actions lead to large state changes downstream

$$\min_{a_0,\ldots,a_H} \sum_{t=0}^{H} r(s_t, a_t), \quad s_{t+1} = f(s_t, a_t)$$

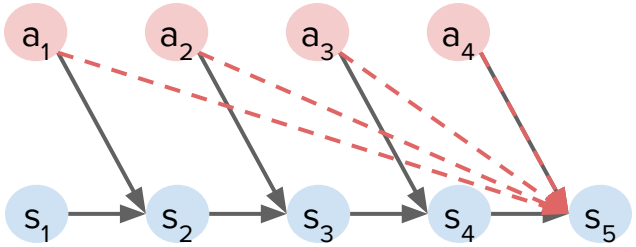


Sensitivity and poor conditioning

Shooting: small changes in early actions lead to large state changes downstream

$$\min_{a_0,\ldots,a_H} \sum_{t=0}^{H} r(s_t, a_t), \quad s_{t+1} = f(s_t, a_t)$$

$$r(s_0, a_0) + r(f(s_0, a_0), a_1) + \ldots + r(f(f(\ldots), \ldots), a_H)$$

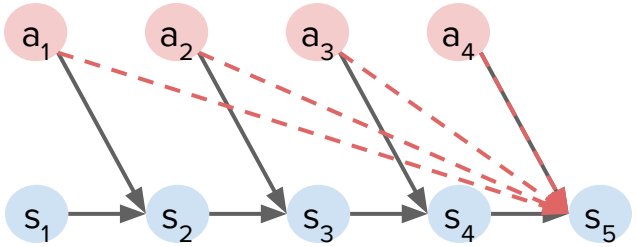


Same issue as exploding/vanishing gradients in RNN training
(but cannot change transition function here)

Sensitivity and poor conditioning

Shooting: small changes in early actions lead to large state changes downstream

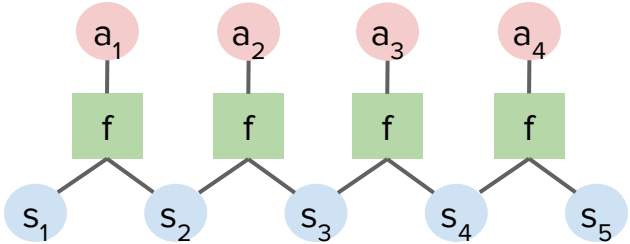
$$\min_{a_0,\ldots,a_H} \sum_{t=0}^{H} r(s_t, a_t), \quad s_{t+1} = f(s_t, a_t)$$



Collocation: optimize for states and/or actions directly

$$\min_{s_0,a_0,\ldots,s_H,a_H} \sum_{t=0}^{H} r(s_t, a_t), \quad ||s_{t+1} - f(s_t, a_t)|| = 0$$

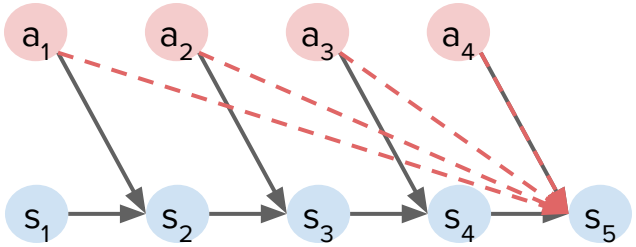
explicit optimization constraint



Sensitivity and poor conditioning

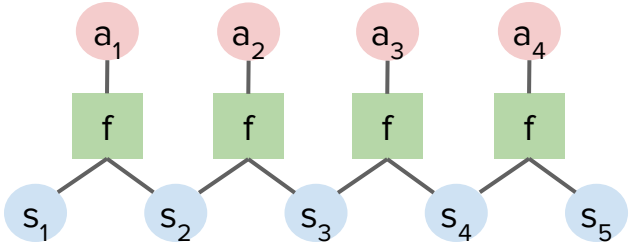
Shooting: small changes in early actions lead to large state changes downstream

$$\min_{a_0,\ldots,a_H} \sum_{t=0}^{H} r(s_t, a_t), \quad s_{t+1} = f(s_t, a_t)$$



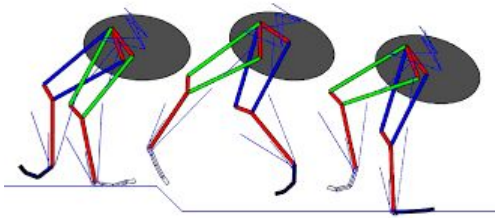
Collocation: optimize for states and/or actions directly

$$\min_{s_0,a_0,\ldots,s_H,a_H} \sum_{t=0}^{H} r(s_t, a_t), \quad ||s_{t+1} - f(s_t, a_t)|| = 0$$



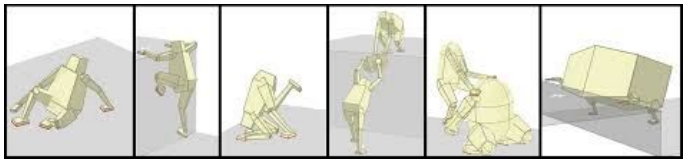
only pairwise dependencies

Sensitivity and poor conditioning



Posa et al (2014). A Direct Method for Trajectory Optimization of Rigid Bodies Through Contact.

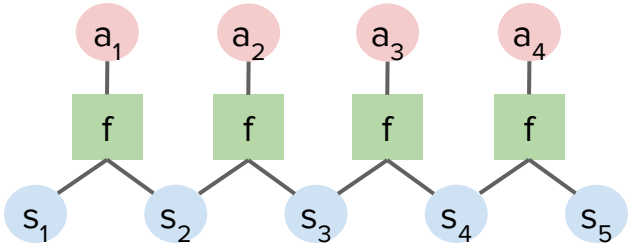
- Good conditioning
  - changing $s_0 a_0$ similar to changing $s_H a_H$
- Larger, but easier to optimize search space
  - good for contact-rich problems



Mordatch et al (2012). Discovery of Complex Behaviors through Contact-Invariant Optimization.

Collocation: optimize for states and/or actions directly

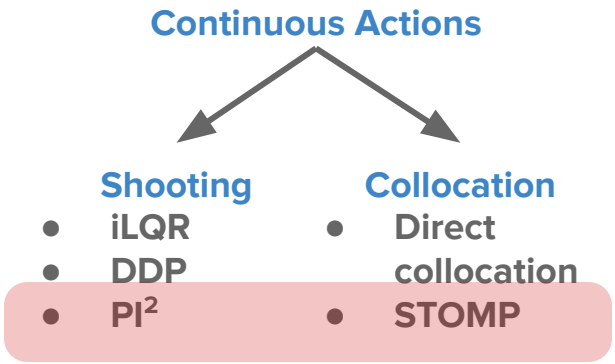
$$\min_{s_0,a_0,\ldots,s_H,a_H} \sum_{t=0}^{H} r(s_t,a_t), \quad ||s_{t+1} - f(s_t,a_t)|| = 0$$



Why so many methods?

Challenges of continuous trajectory optimization:

- Sensitivity and poor conditioning
- Only reaches local optimum
- Slow convergence



Avoiding local optima

- Sampling-based methods can escape local optima
- Cross-Entropy Method (CEM) and PI^2 are common

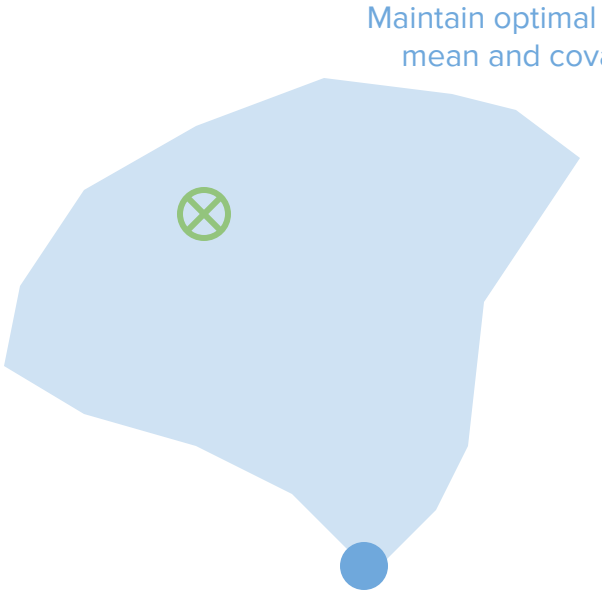
Mannor et al (2003). The Cross-Entropy Method for fast policy search.

*Theodorou et al (2010). **P**olicy **I**mprovement with **P**ath **I**ntegrals.*

Kalakrishnan et al (2011). STOMP: Stochastic trajectory optimization for motion planning.

Avoiding local optima

- Sampling-based methods can escape local optima
- Cross-Entropy Method (CEM) and PI^2 are common



Cross-Entropy Method (one iteration)

$$\boldsymbol{\theta}_{k=1\dots K} \sim \mathcal{N}(\boldsymbol{\theta}, \Sigma) \qquad \textbf{sample} \quad (1)$$

$$J_k = J(\boldsymbol{\theta}_k) \qquad \textbf{eval.} \quad (2)$$

$$\boldsymbol{\theta}_{k=1\dots K} \leftarrow \texttt{sort}\ \boldsymbol{\theta}_{k=1\dots K}\ \texttt{w.r.t}\ J_{k=1\dots K} \qquad \textbf{sort} \quad (3)$$

$$\boldsymbol{\theta}^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} \boldsymbol{\theta}_k \qquad \textbf{update} \quad (4)$$

$$\Sigma^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} (\boldsymbol{\theta}_k - \boldsymbol{\theta})(\boldsymbol{\theta}_k - \boldsymbol{\theta})^\intercal \qquad \textbf{update} \quad (5)$$

Stulp et al (2012). Path Integral Policy Improvement with Covariance Matrix Adaptation.

Avoiding local optima

- Sampling-based methods can escape local optima
- Cross-Entropy Method (CEM) and PI^2 are common

Sample actions around mean trajectory

Cross-Entropy Method (one iteration)

$$\boldsymbol{\theta}_{k=1\dots K} \sim \mathcal{N}(\boldsymbol{\theta}, \Sigma) \qquad \textbf{sample} \quad (1)$$

$$J_k = J(\boldsymbol{\theta}_k) \qquad \textbf{eval.} \quad (2)$$

$$\boldsymbol{\theta}_{k=1\dots K} \leftarrow \texttt{sort}\ \boldsymbol{\theta}_{k=1\dots K}\ \texttt{w.r.t}\ J_{k=1\dots K} \qquad \textbf{sort} \quad (3)$$

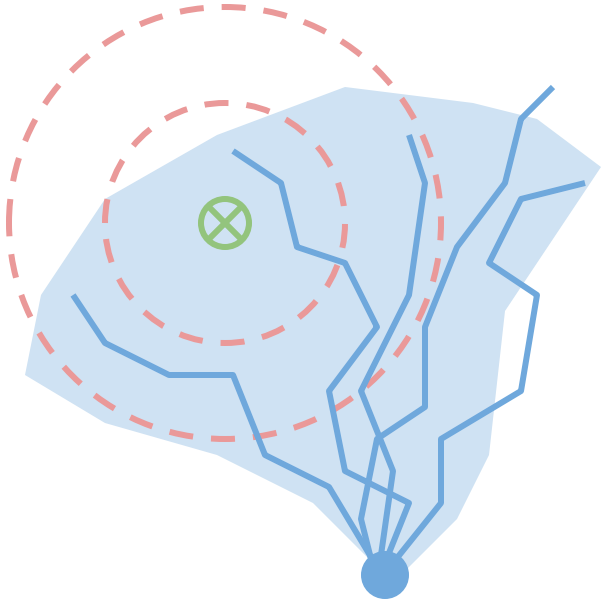
$$\boldsymbol{\theta}^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} \boldsymbol{\theta}_k \qquad \textbf{update} \quad (4)$$

$$\Sigma^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} (\boldsymbol{\theta}_k - \boldsymbol{\theta})(\boldsymbol{\theta}_k - \boldsymbol{\theta})^\intercal \qquad \textbf{update} \quad (5)$$

Stulp et al (2012). Path Integral Policy Improvement with Covariance Matrix Adaptation.

Avoiding local optima

- Sampling-based methods can escape local optima
- Cross-Entropy Method (CEM) and PI^2 are common



Evaluate fitness of samples

Cross-Entropy Method (one iteration)

$$\boldsymbol{\theta}_{k=1\dots K} \sim \mathcal{N}(\boldsymbol{\theta}, \Sigma) \qquad \textbf{sample} \quad (1)$$

$$J_k = J(\boldsymbol{\theta}_k) \qquad \textbf{eval.} \quad (2)$$

$$\boldsymbol{\theta}_{k=1\dots K} \leftarrow \texttt{sort}\ \boldsymbol{\theta}_{k=1\dots K}\ \texttt{w.r.t}\ J_{k=1\dots K} \qquad \textbf{sort} \quad (3)$$

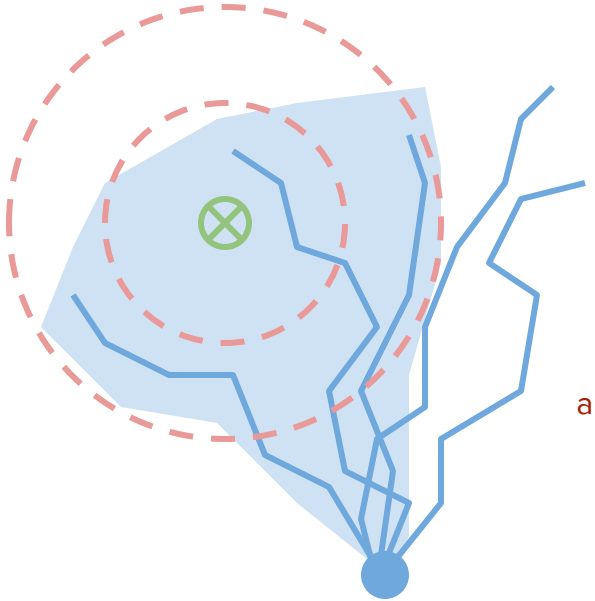
$$\boldsymbol{\theta}^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} \boldsymbol{\theta}_k \qquad \textbf{update} \quad (4)$$

$$\Sigma^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} (\boldsymbol{\theta}_k - \boldsymbol{\theta})(\boldsymbol{\theta}_k - \boldsymbol{\theta})^\intercal \qquad \textbf{update} \quad (5)$$

Stulp et al (2012). Path Integral Policy Improvement with Covariance Matrix Adaptation.

Avoiding local optima

- Sampling-based methods can escape local optima
- Cross-Entropy Method (CEM) and PI^2 are common



Refit optimal trajectory mean and covariance to best samples

Cross-Entropy Method (one iteration)

$$\boldsymbol{\theta}_{k=1\ldots K} \sim \mathcal{N}(\boldsymbol{\theta}, \Sigma) \qquad \textbf{sample} \quad (1)$$

$$J_k = J(\boldsymbol{\theta}_k) \qquad \textbf{eval.} \quad (2)$$

$$\boldsymbol{\theta}_{k=1\ldots K} \leftarrow \texttt{sort}\ \boldsymbol{\theta}_{k=1\ldots K}\ \texttt{w.r.t}\ J_{k=1\ldots K} \qquad \textbf{sort} \quad (3)$$

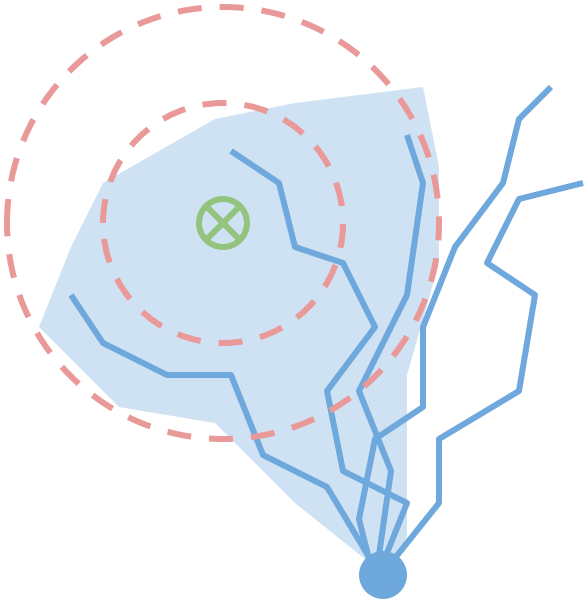
$$\boldsymbol{\theta}^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} \boldsymbol{\theta}_k \qquad \textbf{update} \quad (4)$$

$$\Sigma^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e} (\boldsymbol{\theta}_k - \boldsymbol{\theta})(\boldsymbol{\theta}_k - \boldsymbol{\theta})^\intercal \qquad \textbf{update} \quad (5)$$

Stulp et al (2012). Path Integral Policy Improvement with Covariance Matrix Adaptation.

Avoiding local optima

- Sampling-based methods can escape local optima
- Cross-Entropy Method (CEM) and PI^2 are common



Why does this work?

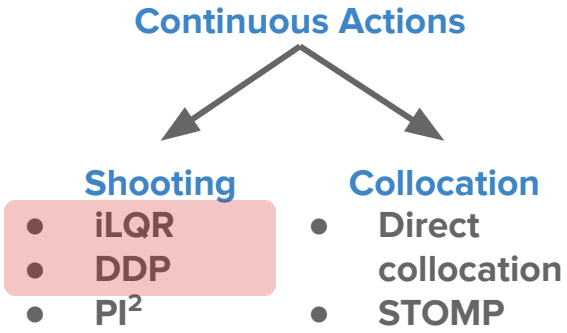
- Search space of decision-time plans much smaller than space of policy parameters
  - *ex.* 30x32 vs $32x64^4x32$
- More feasible plans than policy parameters

Choromanski et al (2019). Provably Robust Blackbox Optimization for Reinforcement Learning.

Why so many methods?

Challenges of continuous trajectory optimization:

- Sensitivity and poor conditioning
- Only reaches local optimum
- Slow convergence



Slow convergence and 2nd-order optimization

- Gradient descent is slow to converge
  - Can wait thousands-millions of iterations to train a policy, but too long for a one-time plan
- Can we do something like Newton's method for trajectory optimization?

Slow convergence and 2nd-order optimization

- Gradient descent is slow to converge
  - Can wait thousands-millions of iterations to train a policy, but too long for a one-time plan
- Can we do something like Newton's method for trajectory optimization?

Approximate transitions with linear functions and rewards with quadratics:

$$\min_{a_0,\ldots,a_H} \sum_{t=0}^{H} r_t, \quad s_{t+1} = f_s(s_t, a_t), \quad r_t = f_r(s_t, a_t)$$

$$f_s(s_t, a_t) \approx As_t + Ba_t, \quad f_r(s_t, a_t) \approx s_t^T Q s_t + a_t^T R a_t$$

Becomes *Linear-Quadratic Regulator (LQR)* problem and can be solved exactly

Kwakernaak et al (1972). Linear Optimal Control Systems.

Slow convergence and 2nd-order optimization

- Gradient descent is slow to converge
  - Can wait thousands-millions of iterations to train a policy, but too long for a one-time plan
- Can we do something like Newton's method for trajectory optimization?

Approximate transitions with linear functions and rewards with quadratics:

$$\min_{a_0,\ldots,a_H} \sum_{t=0}^{H} r_t, \quad s_{t+1} = f_s(s_t, a_t), \quad r_t = f_r(s_t, a_t)$$

$$f_s(s_t, a_t) \approx As_t + Ba_t, \quad f_r(s_t, a_t) \approx s_t^T Q s_t + a_t^T R a_t$$

Becomes *Linear-Quadratic Regulator (LQR)* problem and can be solved exactly

Locally approximate the model around current solution, solve LQR problem to update solution, and repeat

Todorov and Li (2005). A generalized iterative LQG method.

Slow convergence and 2nd-order optimization

- Gradient descent is slow to converge
  - Can wait thousands-millions of iterations to train a policy, but too long for a one-time plan
- Can we do something like Newton's method for trajectory optimization?

Approximate transitions with linear functions and rewards with quadratics:

$$\min_{a_0,\ldots,a_H} \sum_{t=0}^{H} r_t, \quad s_{t+1} = f_s(s_t, a_t), \quad r_t = f_r(s_t, a_t)$$

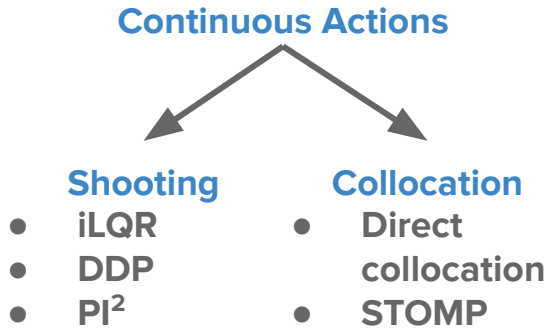
$$f_s(s_t, a_t) \approx As_t + Ba_t, \quad f_r(s_t, a_t) \approx s_t^T Q s_t + a_t^T R a_t$$

Differential dynamic programming (DDP) is similar, but with higher-order expansion of f_s

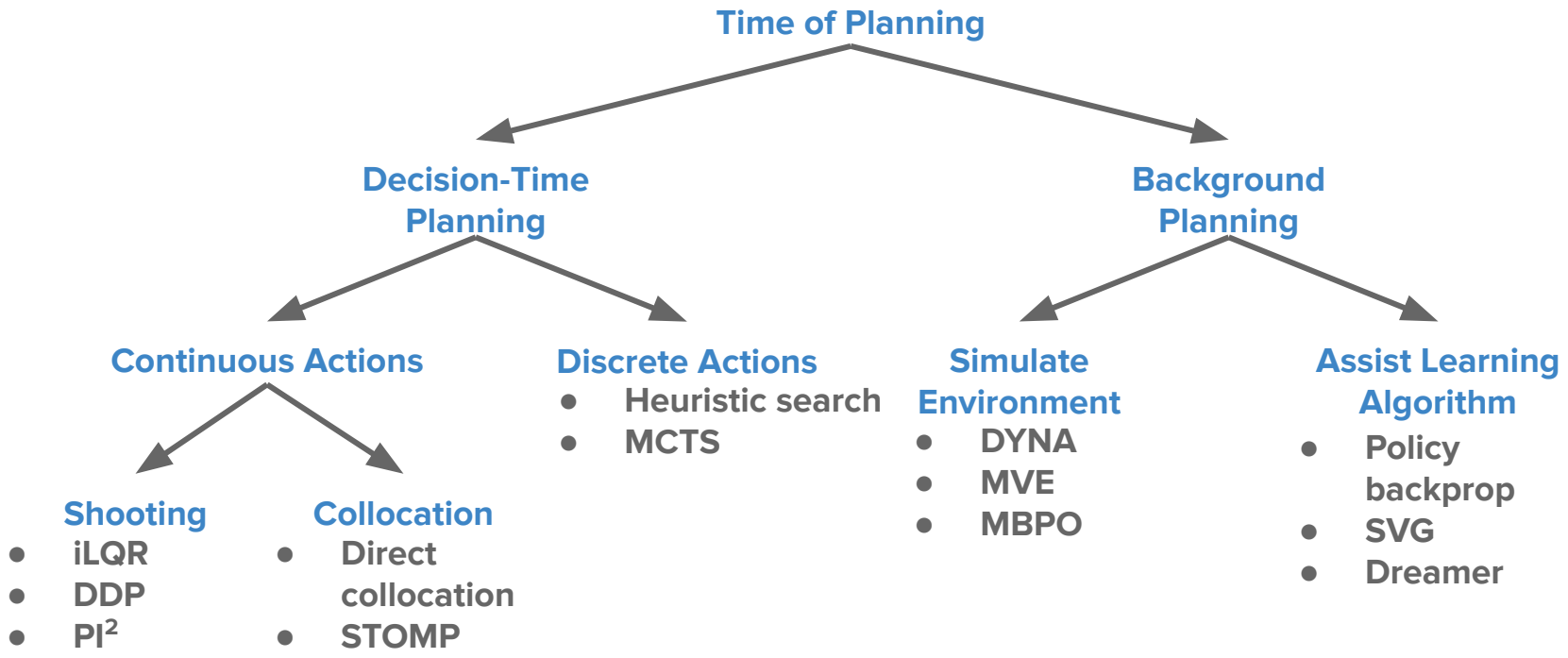
Jacobson and Mayne (1970). Differential Dynamic Programming.

Why so many methods?

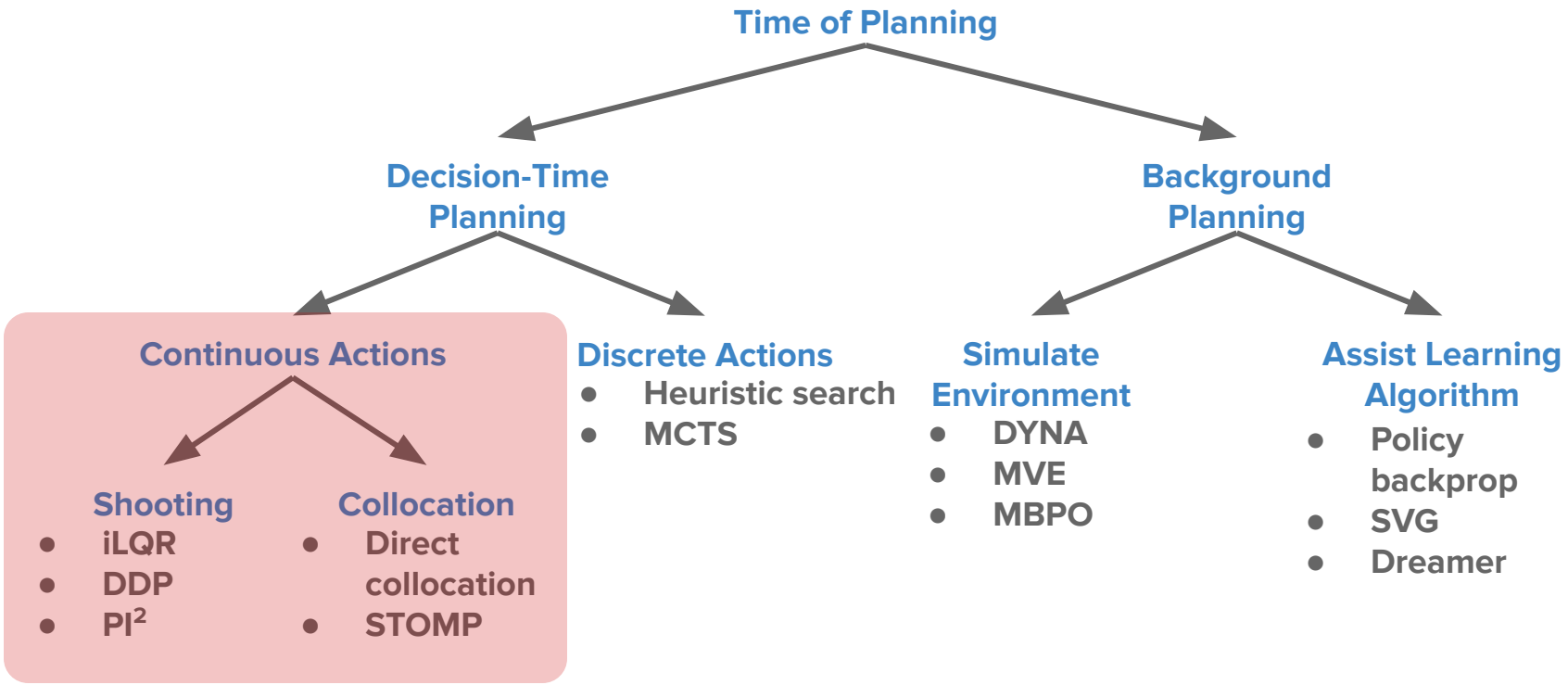
- Sensitivity and poor conditioning
- Only reaches local optimum
- Slow convergence



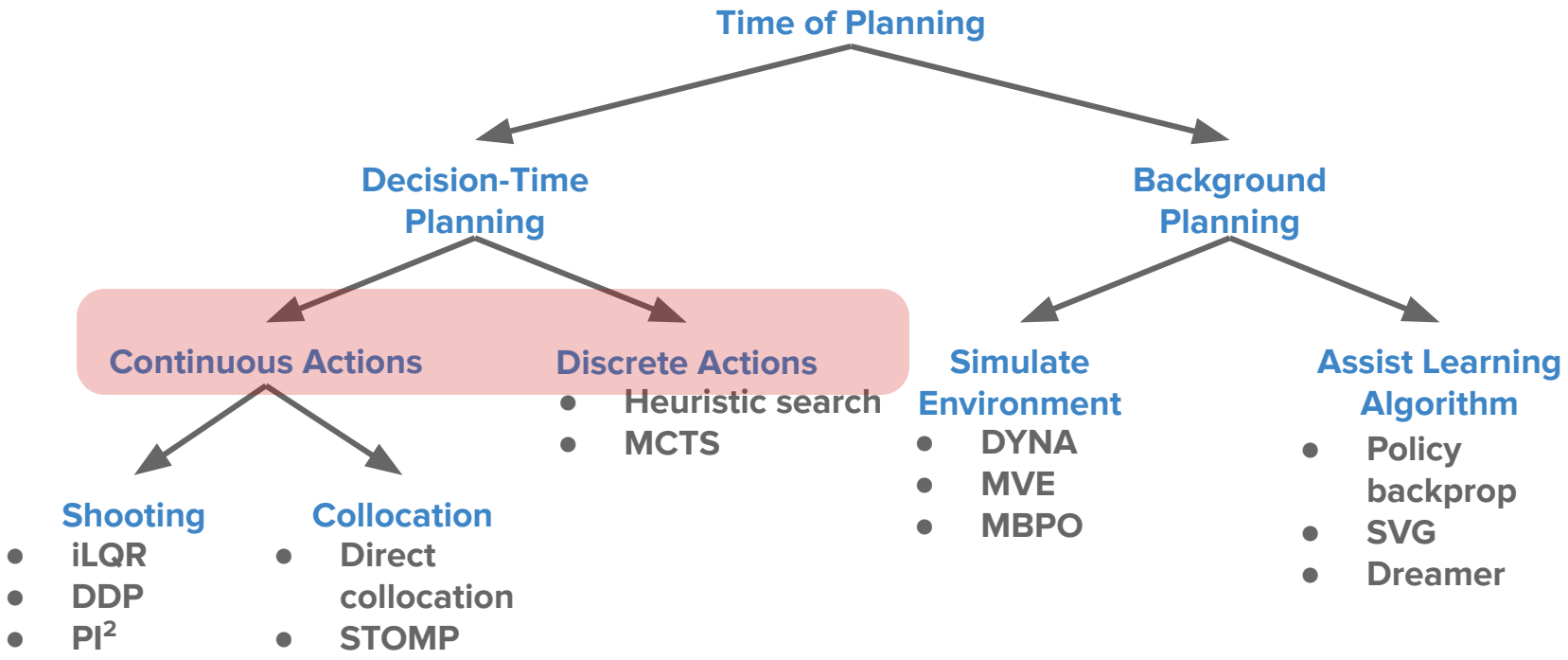
Landscape of model-based methods



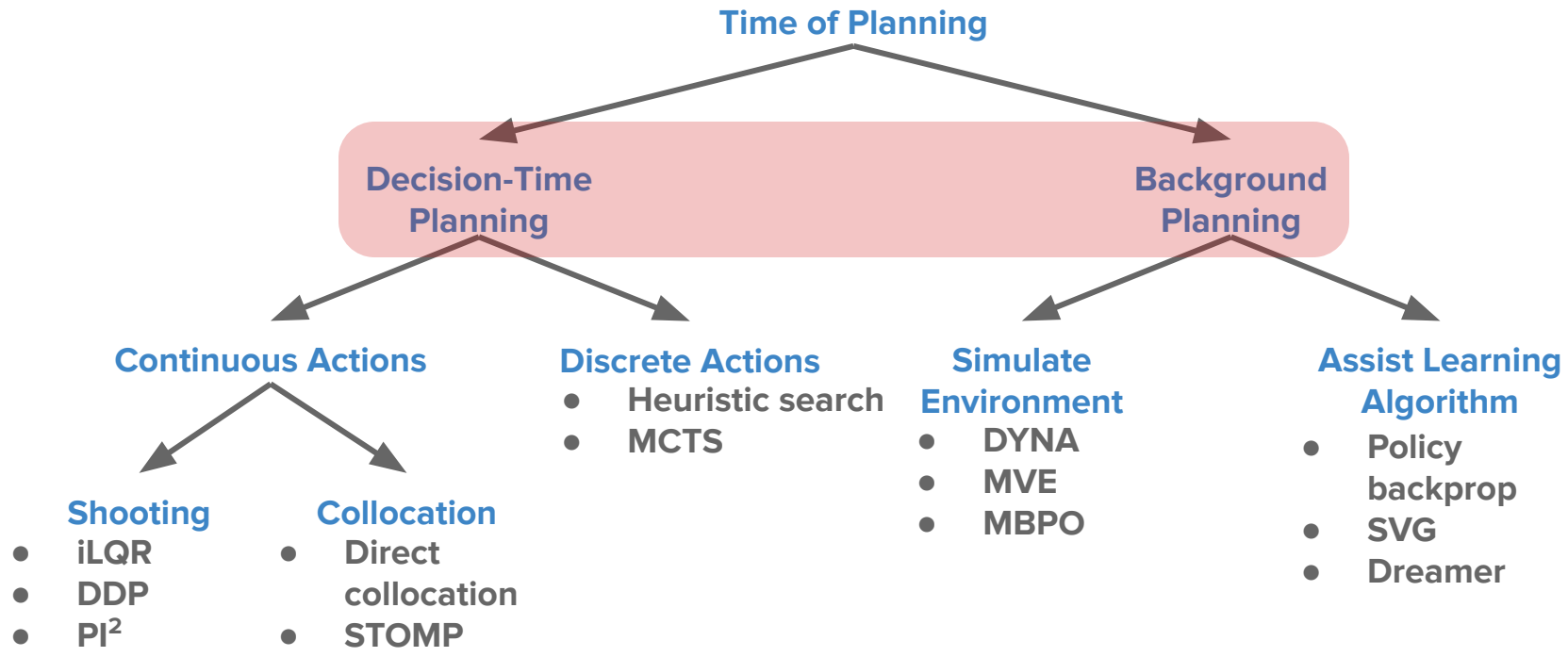
Landscape of model-based methods



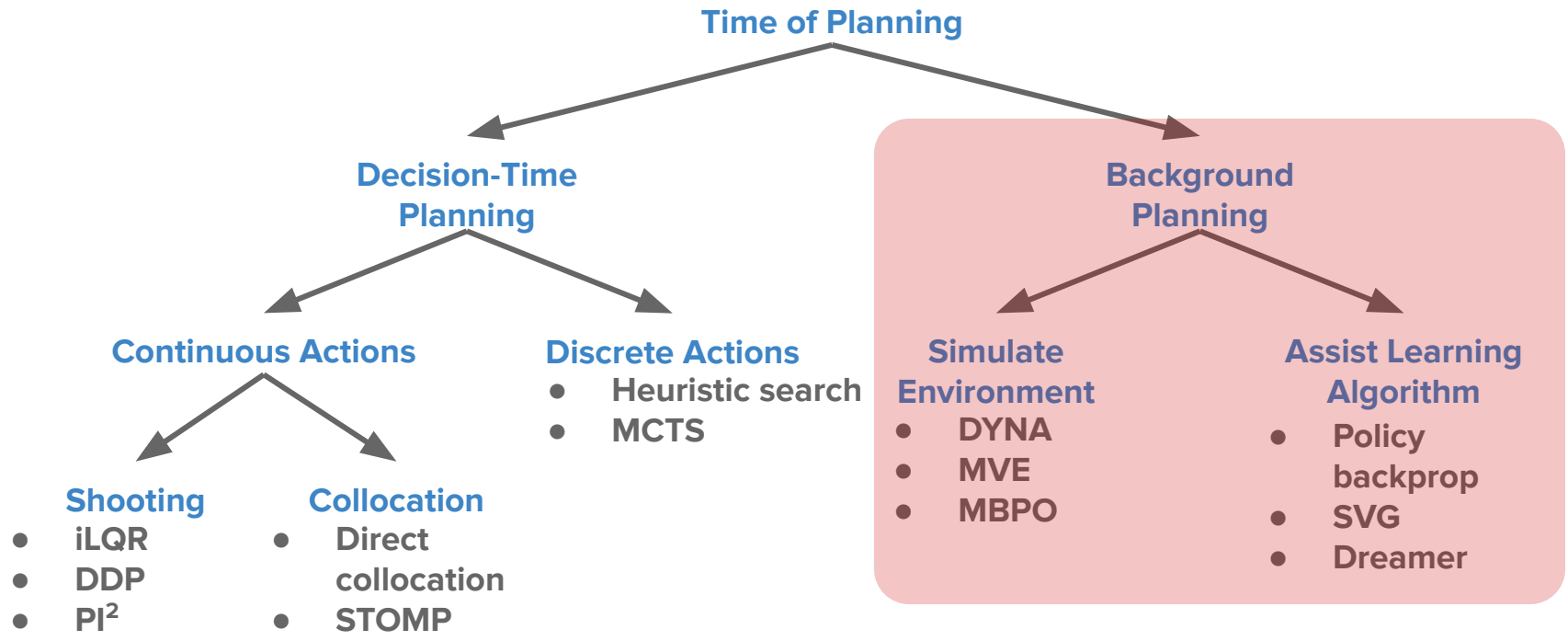
Landscape of model-based methods



Landscape of model-based methods



Landscape of model-based methods



Next part:

Model-based control in the loop and practical issues that arise

Model-Based Methods in Reinforcement Learning

Part 3: Model-based Control in the Loop

Igor Mordatch (Google) & Jessica Hamrick (DeepMind)

ICML 2020 Tutorial

Outline

1. Introduction and motivation
2. Problem statement
3. What is a “model”?
4. What is model-based control?
5. **Model-based control in the loop**
6. What else can models be used for?
7. What’s missing from model-based methods?
8. Conclusion

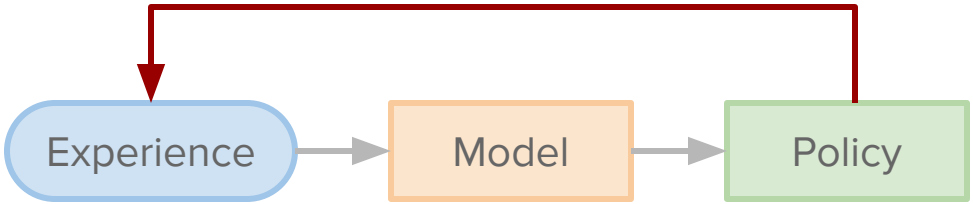
Aims to understand

- Where does **data** to train the models come from?
- Can we act with **imperfect** models?
- Why do we need model **uncertainty** and how to estimate it?
- Can we **combine** background and decision-time planning?

Gathering data to train the model

- *“Chicken or the egg”* problem

  Bad policy leads to bad experience, leads to bad model, leads to bad policy ...



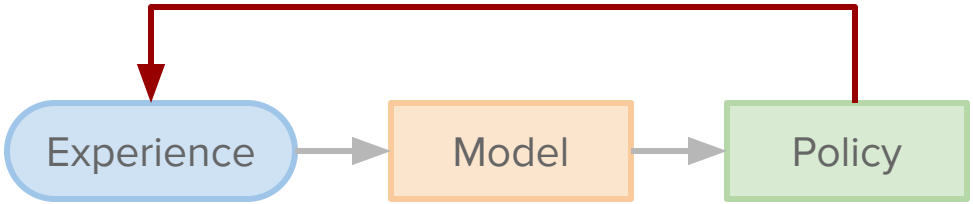
Gathering data to train the model

- *“Chicken or the egg”* problem

  Bad policy leads to bad experience, leads to bad model, leads to bad policy ...

- Training stability issues in practice
  - Game theory can provide criteria for stability:

Rajeswaran et al (2020). A Game Theoretic Framework for Model Based Reinforcement Learning.



Gathering data to train the model: **Fixed Experience**

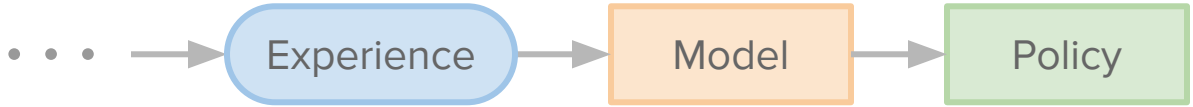
- Learn only from static datasets
  - Human demonstrations
  - Manually-engineered policy rollouts
  - Another (sub-optimal) policy

- Recently popular topic of *model-based offline reinforcement learning:*

  Kidambi et al (2020). MOReL: Model-Based Offline Reinforcement Learning.

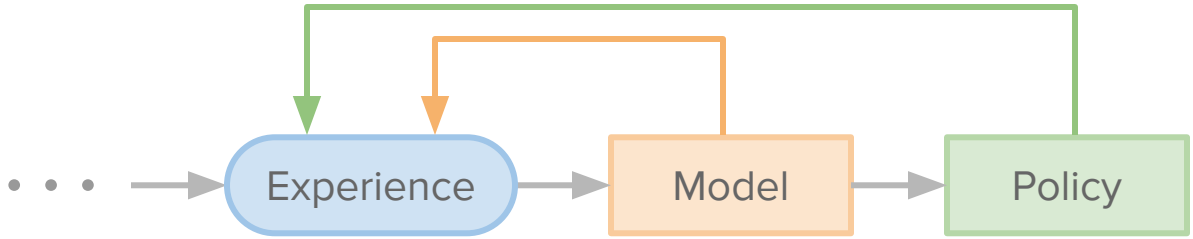
  Yu et al (2020). MOPO: Model-based Offline Policy Optimization.

  See also: *Levine et al (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.*



Gathering data to train the model: **Data Augmentation**

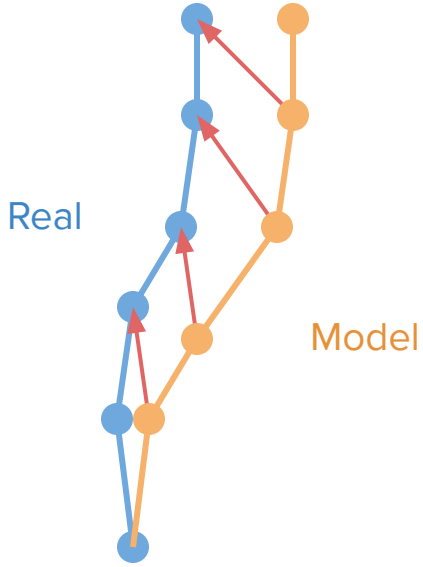
- Use model to generate data to train itself

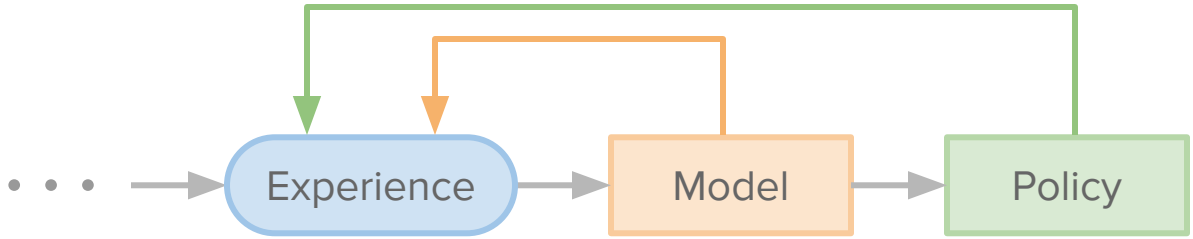


Gathering data to train the model: **Data Augmentation**

- Use model to generate data to train itself
- *Example:* Roll out the model and train to pull its predicted next states to true next states

Venkatraman et al (2014). Data as Demonstrator.



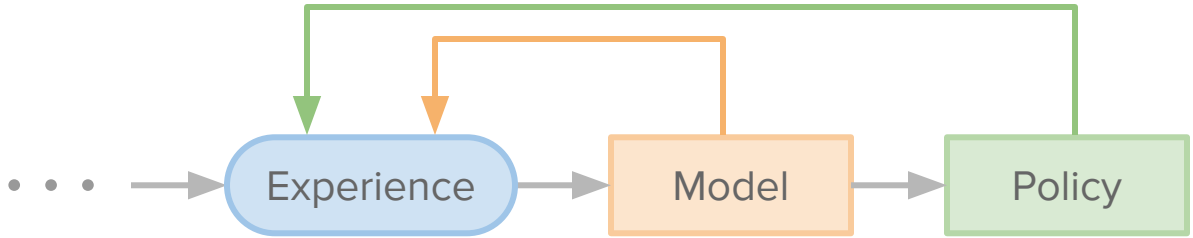


Gathering data to train the model: **Data Augmentation**

- Use model to generate data to train itself
- *Example:* Roll out the model and train to pull its predicted next states to true next states
- Adversarial ways to generate data to self-audit the model

Lin et al (2020). Model-based Adversarial Meta-Reinforcement Learning.

Du et al (2019). Model-Based Planning with Energy Models.

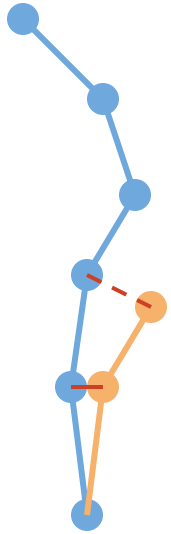


Models are not going to be perfect

- Won’t have experience everywhere
- Will have model function approximation error

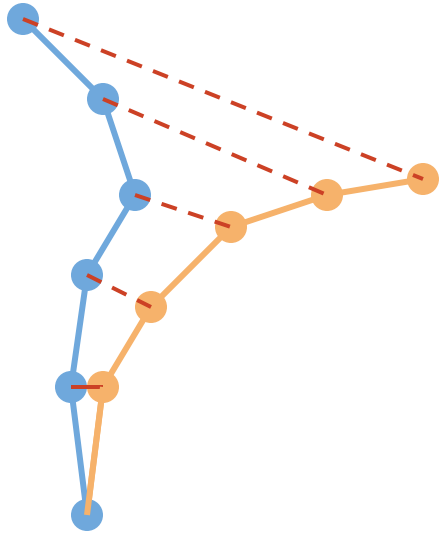
Models are not going to be perfect

- Won’t have experience everywhere
- Will have model function approximation error
- Small model errors propagate and compound



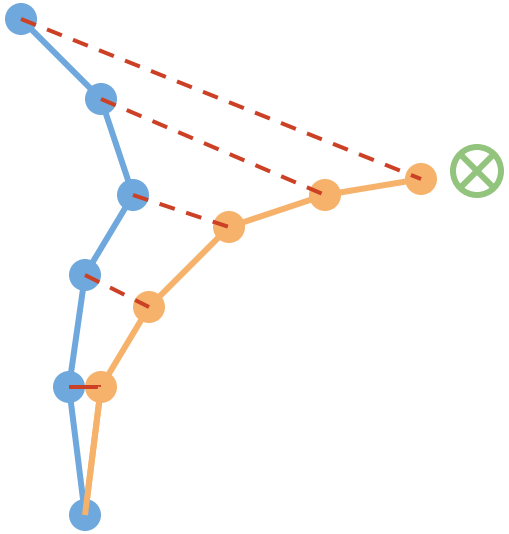
Models are not going to be perfect

- Won’t have experience everywhere
- Will have model function approximation error
- Small model errors propagate and compound



Models are not going to be perfect

- Won’t have experience everywhere
- Will have model function approximation error
- Small model errors propagate and compound
- Planner may *exploit* model errors

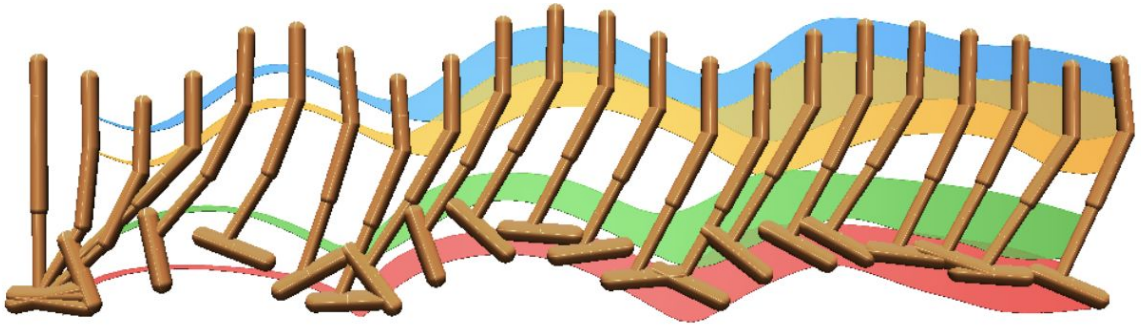


Models are not going to be perfect

- Won't have experience everywhere
- Will have model function approximation error
- Small model errors propagate and compound
- Planner may *exploit* model errors
- Longer model rollouts less reliable

policy performance bound worse with length k:

$$\eta[\pi] \geq \hat{\eta}[\pi] - 2r_{\max}\left[\frac{\gamma^{k+1}\epsilon_\pi}{(1-\gamma)^2} + \frac{\gamma^{k}\epsilon_\pi}{(1-\gamma)} + \frac{k}{1-\gamma}(\epsilon_{m'})\right]$$



Janner et al (2019). When to Trust Your Model: Model-Based Policy Optimization.

Models are not going to be perfect

- Won’t have experience everywhere
- Will have model function approximation error
- Small model errors propagate and compound
- Planner may *exploit* model errors
- Longer model rollouts less reliable
- **Can we still act with imperfect models?**

Outline of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

Estimating model uncertainty

- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

Combining planning and learning

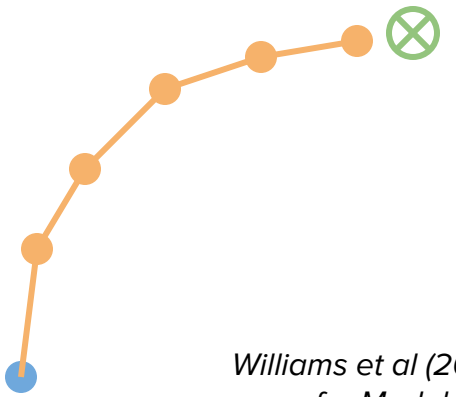
- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

Acting under imperfect models: **replan**

- Don’t **commit to a plan**, but **continually re-plan**

Acting under imperfect models: **replan**

- Don’t commit to a plan, but continually re-plan



Optimize trajectory

Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.

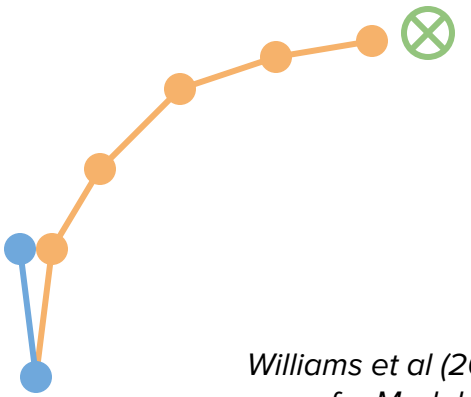
Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{N-1})$: Initial control sequence;
$\Delta t, \mathbf{x}_{t_0}, \mathbf{f}, \mathbf{G}, \mathbf{B}, \nu$: System/sampling dynamics;
$\phi, q, \mathbf{R}, \lambda$: Cost parameters;
$\mathbf{u}_{\text{init}}$: Value to initialize new controls to;

while *task not completed* **do**
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} = \mathbf{x}_{t_0}$;
    for $i \leftarrow 1$ **to** $N-1$ **do**
      $\mathbf{x}_{i+1} = \mathbf{x}_i + \left(\mathbf{f} + \mathbf{G}\left(\mathbf{u}_i + \delta\mathbf{u}_{i,k}\right)\right)\Delta t$;
      $\tilde{S}(\tau_{i+1,k}) = \tilde{S}(\tau_{i,k}) + \tilde{q}$;
  for $i \leftarrow 0$ **to** $N-1$ **do**
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \left[\sum_{k=1}^{K}\left(\frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)\delta\mathbf{u}_{i,k}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)}\right)\right]$;
  send to actuators$(\mathbf{u}_0)$;
  for $i \leftarrow 0$ **to** $N-2$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$;
  $\mathbf{u}_{N-1} = \mathbf{u}_{\text{init}}$
  Update the current state after receiving feedback;
  check for task completion;

Acting under imperfect models: **replan**

- Don’t commit to a plan, but continually re-plan



Apply first optimal action

Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.

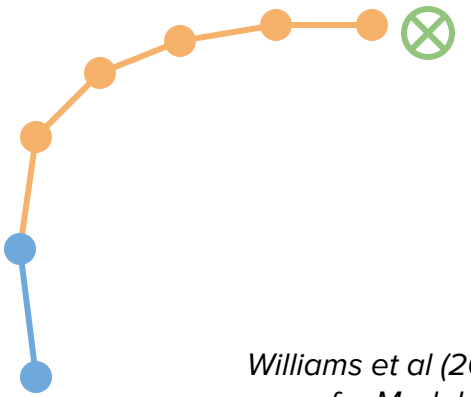
Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{N-1})$: Initial control sequence;
$\Delta t, \mathbf{x}_{t_0}, \mathbf{f}, \mathbf{G}, \mathbf{B}, \nu$: System/sampling dynamics;
$\phi, q, \mathbf{R}, \lambda$: Cost parameters;
$\mathbf{u}_{\text{init}}$: Value to initialize new controls to;

while *task not completed* **do**
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} = \mathbf{x}_{t_0}$;
    for $i \leftarrow 1$ **to** $N-1$ **do**
      $\mathbf{x}_{i+1} = \mathbf{x}_i + (\mathbf{f} + \mathbf{G}(\mathbf{u}_i + \delta\mathbf{u}_{i,k}))\Delta t$;
      $\tilde{S}(\tau_{i+1,k}) = \tilde{S}(\tau_{i,k}) + \tilde{q}$;
  for $i \leftarrow 0$ **to** $N-1$ **do**
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \left[\sum_{k=1}^{K}\left(\frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)\delta\mathbf{u}_{i,k}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)}\right)\right]$;
  send to actuators$(\mathbf{u}_0)$;
  for $i \leftarrow 0$ **to** $N-2$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$;
  $\mathbf{u}_{N-1} = \mathbf{u}_{\text{init}}$
  Update the current state after receiving feedback;
  check for task completion;

Acting under imperfect models: **replan**

- Don’t commit to a plan, but continually re-plan



Optimize trajectory

Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.

Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{N-1})$: Initial control sequence;
$\Delta t, \mathbf{x}_{t_0}, \mathbf{f}, \mathbf{G}, \mathbf{B}, \nu$: System/sampling dynamics;
$\phi, q, \mathbf{R}, \lambda$: Cost parameters;
$\mathbf{u}_{\text{init}}$: Value to initialize new controls to;

while *task not completed* **do**
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} = \mathbf{x}_{t_0}$;
    for $i \leftarrow 1$ **to** $N-1$ **do**
      $\mathbf{x}_{i+1} = \mathbf{x}_i + \left(\mathbf{f} + \mathbf{G}\left(\mathbf{u}_i + \delta\mathbf{u}_{i,k}\right)\right)\Delta t$;
      $\tilde{S}(\tau_{i+1,k}) = \tilde{S}(\tau_{i,k}) + \tilde{q}$;
  for $i \leftarrow 0$ **to** $N-1$ **do**
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \left[\sum_{k=1}^{K}\left(\frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)\delta\mathbf{u}_{i,k}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)}\right)\right]$;
  send to actuators$(\mathbf{u}_0)$;
  for $i \leftarrow 0$ **to** $N-2$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$;
  $\mathbf{u}_{N-1} = \mathbf{u}_{\text{init}}$
  Update the current state after receiving feedback;
  check for task completion;

Acting under imperfect models: **replan**

- Don’t commit to a plan, but continually re-plan

Apply first optimal action

Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.

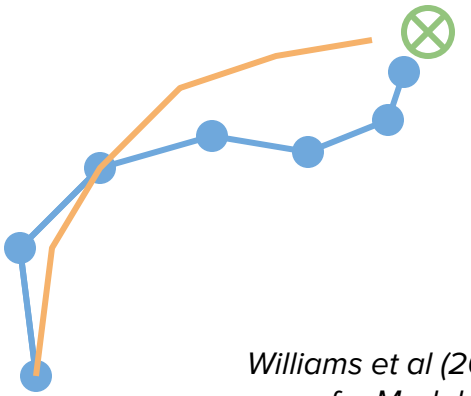
Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{N-1})$: Initial control sequence;
$\Delta t, \mathbf{x}_{t_0}, \mathbf{f}, \mathbf{G}, \mathbf{B}, \nu$: System/sampling dynamics;
$\phi, q, \mathbf{R}, \lambda$: Cost parameters;
$\mathbf{u}_{\text{init}}$: Value to initialize new controls to;

while *task not completed* **do**
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} = \mathbf{x}_{t_0}$;
    for $i \leftarrow 1$ **to** $N-1$ **do**
      $\mathbf{x}_{i+1} = \mathbf{x}_i + \left(\mathbf{f} + \mathbf{G}\left(\mathbf{u}_i + \delta\mathbf{u}_{i,k}\right)\right)\Delta t$;
      $\tilde{S}(\tau_{i+1,k}) = \tilde{S}(\tau_{i,k}) + \tilde{q}$;
  for $i \leftarrow 0$ **to** $N-1$ **do**
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \left[\sum_{k=1}^{K}\left(\frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)\delta\mathbf{u}_{i,k}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)}\right)\right]$;
  send to actuators$(\mathbf{u}_0)$;
  for $i \leftarrow 0$ **to** $N-2$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$;
  $\mathbf{u}_{N-1} = \mathbf{u}_{\text{init}}$
  Update the current state after receiving feedback;
  check for task completion;

Acting under imperfect models: **replan**

- Don’t commit to a plan, but continually re-plan



Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.

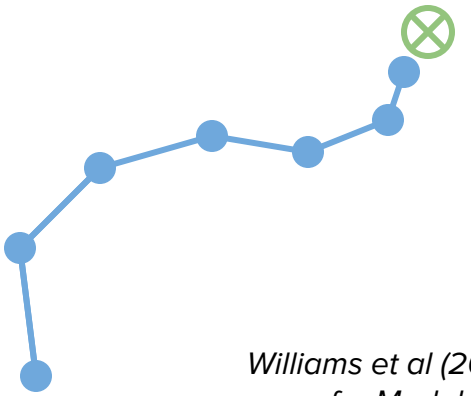
Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{N-1})$: Initial control sequence;
$\Delta t, \mathbf{x}_{t_0}, \mathbf{f}, \mathbf{G}, \mathbf{B}, \nu$: System/sampling dynamics;
$\phi, q, \mathbf{R}, \lambda$: Cost parameters;
$\mathbf{u}_{\text{init}}$: Value to initialize new controls to;

while *task not completed* **do**
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} = \mathbf{x}_{t_0}$;
    for $i \leftarrow 1$ **to** $N-1$ **do**
      $\mathbf{x}_{i+1} = \mathbf{x}_i + \left(\mathbf{f} + \mathbf{G}\left(\mathbf{u}_i + \delta\mathbf{u}_{i,k}\right)\right)\Delta t$;
      $\tilde{S}(\tau_{i+1,k}) = \tilde{S}(\tau_{i,k}) + \tilde{q}$;
  for $i \leftarrow 0$ **to** $N-1$ **do**
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \left[\sum_{k=1}^{K}\left(\frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)\delta\mathbf{u}_{i,k}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)}\right)\right]$;
  send to actuators$(\mathbf{u}_0)$;
  for $i \leftarrow 0$ **to** $N-2$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$;
  $\mathbf{u}_{N-1} = \mathbf{u}_{\text{init}}$
  Update the current state after receiving feedback;
  check for task completion;

Acting under imperfect models: **replan**

- Don’t commit to a plan, but continually re-plan
- Errors don’t accumulate
  - Don’t need a perfect model, just one pointing in the right direction



Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.

Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{N-1})$: Initial control sequence;
$\Delta t, \mathbf{x}_{t_0}, \mathbf{f}, \mathbf{G}, \mathbf{B}, \nu$: System/sampling dynamics;
$\phi, q, \mathbf{R}, \lambda$: Cost parameters;
$\mathbf{u}_{\text{init}}$: Value to initialize new controls to;

while *task not completed* **do**
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} = \mathbf{x}_{t_0}$;
    for $i \leftarrow 1$ **to** $N-1$ **do**
      $\mathbf{x}_{i+1} = \mathbf{x}_i + \left(\mathbf{f} + \mathbf{G}\left(\mathbf{u}_i + \delta\mathbf{u}_{i,k}\right)\right)\Delta t$;
      $\tilde{S}(\tau_{i+1,k}) = \tilde{S}(\tau_{i,k}) + \tilde{q}$;
  for $i \leftarrow 0$ **to** $N-1$ **do**
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \left[\sum_{k=1}^{K}\left(\frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)\delta\mathbf{u}_{i,k}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)}\right)\right]$;
  send to actuators$(\mathbf{u}_0)$;
  for $i \leftarrow 0$ **to** $N-2$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$;
  $\mathbf{u}_{N-1} = \mathbf{u}_{\text{init}}$
  Update the current state after receiving feedback;
  check for task completion;

Acting under imperfect models: **replan**

- Don’t commit to a plan, but continually re-plan
- Errors don’t accumulate
  - Don’t need a perfect model, just one pointing in the right direction
- Isn’t re-planning expensive?
  - Reuse solution from previous step as initial guess for next plan

Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.

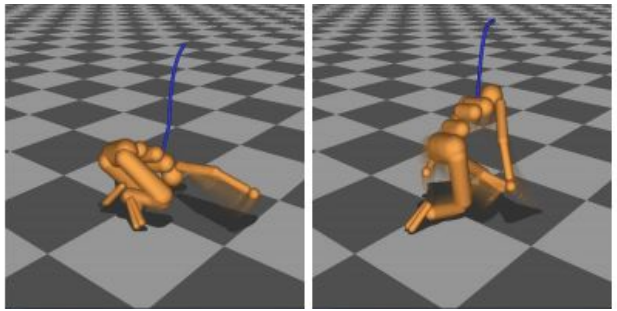
Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{N-1})$: Initial control sequence;
$\Delta t, \mathbf{x}_{t_0}, \mathbf{f}, \mathbf{G}, \mathbf{B}, \nu$: System/sampling dynamics;
$\phi, q, \mathbf{R}, \lambda$: Cost parameters;
$\mathbf{u}_{\text{init}}$: Value to initialize new controls to;

while *task not completed* **do**
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} = \mathbf{x}_{t_0}$;
    for $i \leftarrow 1$ **to** $N-1$ **do**
      $\mathbf{x}_{i+1} = \mathbf{x}_i + (\mathbf{f} + \mathbf{G}(\mathbf{u}_i + \delta\mathbf{u}_{i,k}))\Delta t$;
      $\tilde{S}(\tau_{i+1,k}) = \tilde{S}(\tau_{i,k}) + \tilde{q}$;
  for $i \leftarrow 0$ **to** $N-1$ **do**
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \left[\sum_{k=1}^{K} \left(\frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)\delta\mathbf{u}_{i,k}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_{i,k})\right)} \right)\right]$;
  send to actuators$(\mathbf{u}_0)$;
  for $i \leftarrow 0$ **to** $N-2$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$;
  $\mathbf{u}_{N-1} = \mathbf{u}_{\text{init}}$
  Update the current state after receiving feedback;
  check for task completion;

Acting under imperfect models: **replan**

- Applicable for real-time control of complex systems



Tassa et al (2012). Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization.

Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.

Nagabandi et al (2019). Deep Dynamics Models for Learning Dexterous Manipulation.

Outline of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

Estimating model uncertainty

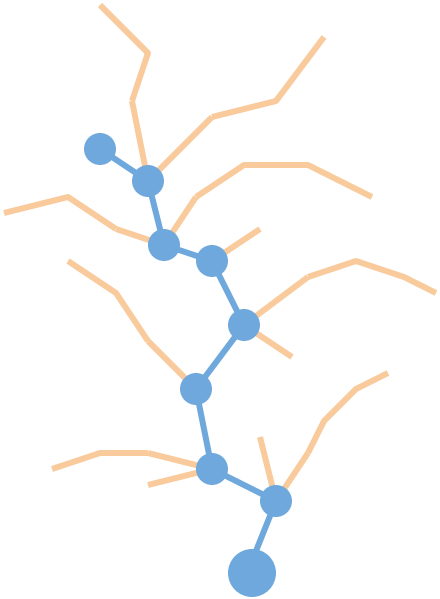
- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

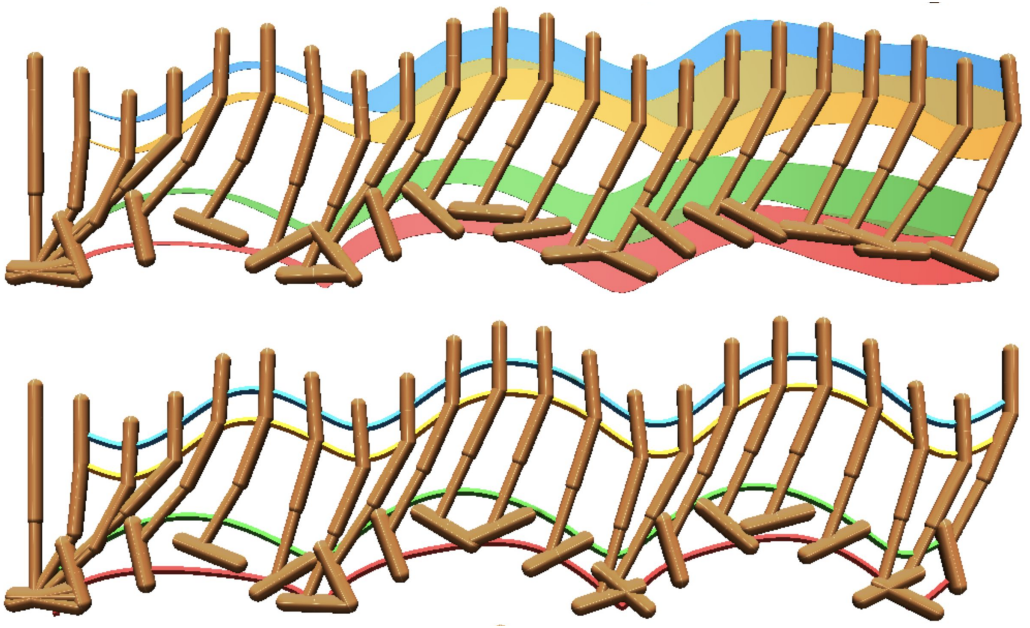
Combining planning and learning

- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

Acting under imperfect models: **plan conservatively**

- Keep model rollouts short





Janner et al (2019). When to Trust Your Model: Model-Based Policy Optimization.

Acting under imperfect models: **plan conservatively**

- Keep model rollouts short
- Plan for average or worst case wrt model uncertainty

$$\max_\theta \mathbb{E}_{f\sim F}[\sum_t \gamma^t r_t], \quad a_t = \pi_\theta(s_t), \quad s_{t+1} = f_s(s_t, a_t), \quad r_t = f_r(s_t, a_t)$$

Acting under imperfect models: **plan conservatively**

- Keep model rollouts short
- Plan for average or worst case wrt model uncertainty

Sample N perturbed models

Plan with worst performing models

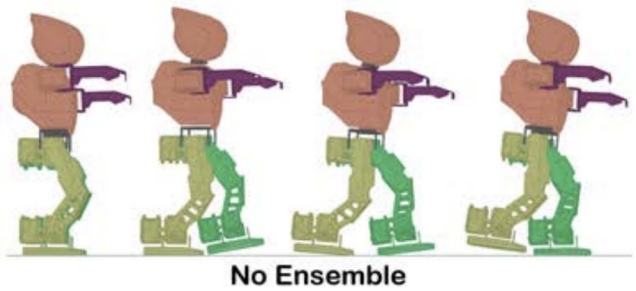
Algorithm 1: EPOpt–ϵ for Robust Policy Search

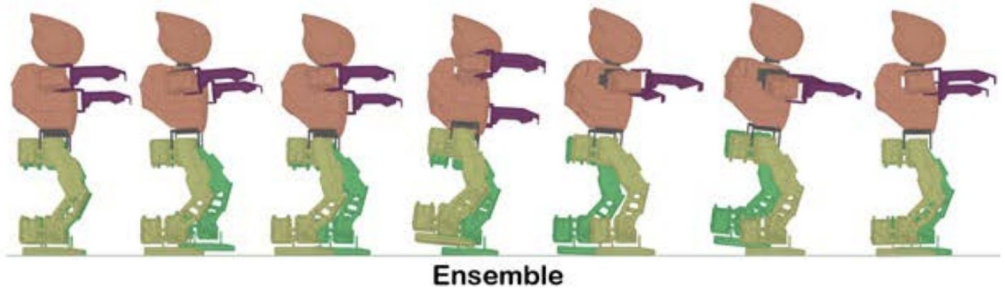
Input: ψ, θ_0, $niter$, N, ϵ
for *iteration* $i = 0, 1, 2, \ldots niter$ **do**
  for $k = 1, 2, \ldots N$ **do**
    sample model parameters $p_k \sim \mathcal{P}_\psi$
    sample a trajectory $\tau_k = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T-1}$ from $\mathcal{M}(p_k)$ using policy $\pi(\theta_i)$
  end
  compute $Q_\epsilon = \epsilon$ percentile of $\{R(\tau_k)\}_{k=1}^{N}$
  select sub-set $\mathbb{T} = \{\tau_k : R(\tau_k) \leq Q_\epsilon\}$
  Update policy: $\theta_{i+1} = \text{BatchPolOpt}(\theta_i, \mathbb{T})$
end

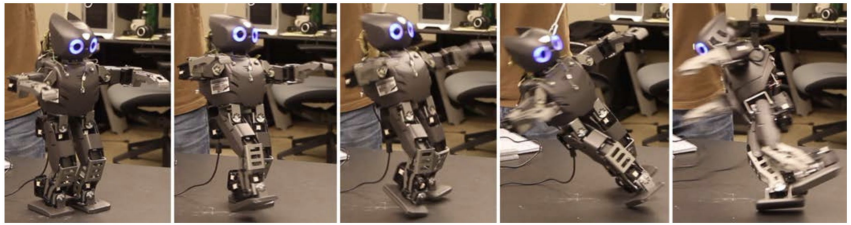
Rajeswaran et al (2017). EPOpt: Learning Robust Neural Network Policies Using Model Ensembles.

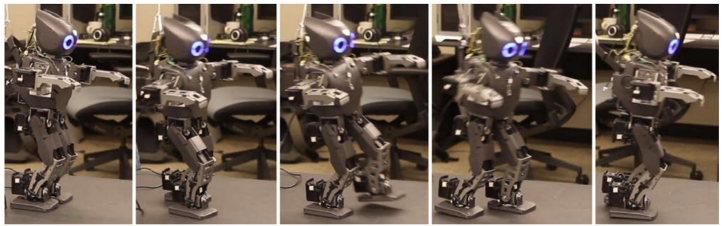
Acting under imperfect models: **plan conservatively**

- Keep model rollouts short
- Plan for average or worst case wrt model uncertainty









Mordatch et al (2015). Ensemble-CIO.

Acting under imperfect models: **plan conservatively**

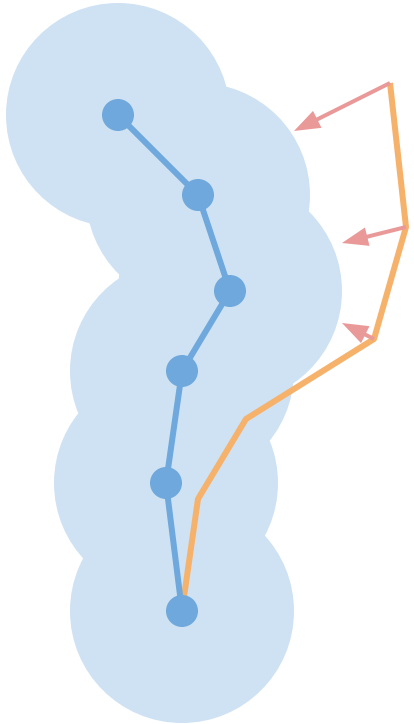
- Keep model rollouts short
- Plan for average or worst case wrt model uncertainty
- Act to stay close to states where model is certain
  - **Implicitly:** stay close to past policy

    Peters et al (2012). Relative Entropy Policy Search

    Levine et al (2014). Guided Policy Search under Unknown Dynamics.

  - **Explicitly:** add penalty to going into unknown region

    Kidambi et al (2020). MOReL: Model-Based Offline Reinforcement Learning.



Acting under imperfect models: **plan conservatively**

- Keep model rollouts short
- Plan for average or worst case wrt model **uncertainty**
- Act to stay close to states where model is **certain**

Acting under imperfect models: **plan conservatively**

- Keep model rollouts short
- Plan for average or worst case wrt model uncertainty
- Act to stay close to states where model is certain
- But how do we get model uncertainty?

Outline of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

Estimating model uncertainty

- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

Combining planning and learning

- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

Estimating model uncertainty

- Important for safe planning, but has other applications (see Jessica's next section)
- At least two distinct sources of uncertainty:

Epistemic uncertainty

- Model's lack of knowledge about the world
- Distribution over beliefs
- Reducible
- Changes with learning

Aleatoric uncertainty / Risk

- World's inherent stochasticity
- Distribution over outcomes
- Irreducible
- Static

Estimating model uncertainty

- Important for safe planning, but has other applications (see Jessica's next section)
- At least two distinct sources of uncertainty:

Epistemic uncertainty

- Model's lack of knowledge about the world
- Distribution over beliefs
- Reducible
- Changes with learning

Aleatoric uncertainty / Risk

- World's inherent stochasticity
- Distribution over outcomes
- Irreducible
- Static

Estimating model uncertainty

- Important for safe planning, but has other applications (see Jessica's next section)
- At least two distinct sources of uncertainty:

Epistemic uncertainty

- Model's lack of knowledge about the world
- Distribution over beliefs
- Reducible
- Changes with learning

Aleatoric uncertainty / Risk

- World's inherent stochasticity
- Distribution over outcomes
- Irreducible
- Static

Estimating model uncertainty

- Important for safe planning, but has other applications (see Jessica's next section)
- At least two distinct sources of uncertainty:

Epistemic uncertainty

- Model's lack of knowledge about the world
- Distribution over beliefs
- Reducible
- Changes with learning

Aleatoric uncertainty / Risk

- World's inherent stochasticity
- Distribution over outcomes
- Irreducible
- Static

How do we estimate these?

Estimating model uncertainty

- Multiple approaches, and an active research area in machine learning

  - Probabilistic neural networks
  - Bayesian neural networks
  - Gaussian processes
  - Pseudo-counts
  - Ensembles

For discussion in the context of reinforcement learning see:

Osband et al (2018). Randomized Prior Functions for Deep Reinforcement Learning.

Estimating model uncertainty: **probabilistic networks**

- Model explicitly outputs means and variances (typically Gaussian)

$$p(s_{t+1}|s_t, a_t) = \mathcal{N}(\mu_\theta(s_t, a_t), \sigma_\theta(s_t, a_t))$$

- Simple and reliable (supervised learning)
- Only captures aleatoric uncertainty / risk
- No guarantees for reasonable outputs outside of training data

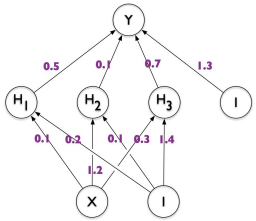
Estimating model uncertainty: **Bayesian neural nets**

- Model has distribution over neural network weights

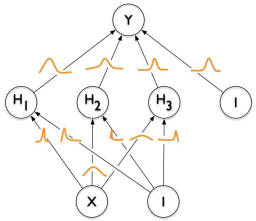
$$p(s_{t+1}|s_t, a_t) = \mathbb{E}_\theta[p(s_{t+1}|s_t, a_t, \theta)]$$

Neal (1995). Bayesian Learning for Neural Networks.

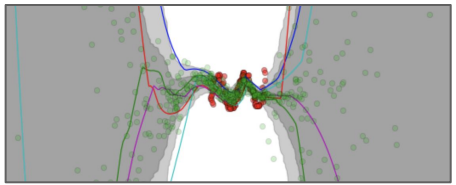
deterministic



factorized Gaussian



Blundell et al (2015). Weight Uncertainty in Neural Networks.



Houthooft et al (2017). VIME: Variational Information Maximizing Exploration.

- Captures epistemic and aleatoric uncertainty
- Factorized approximations can underestimate uncertainty
- Can be hard to train (but an active research area)

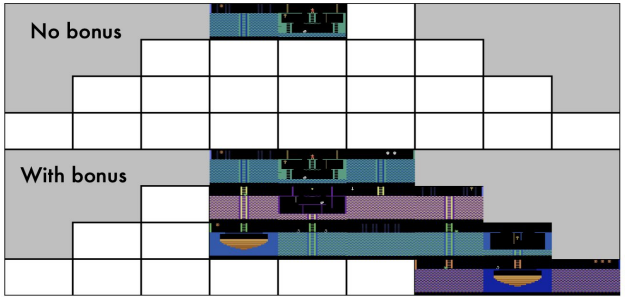
Estimating model uncertainty: **Gaussian processes**

- ...

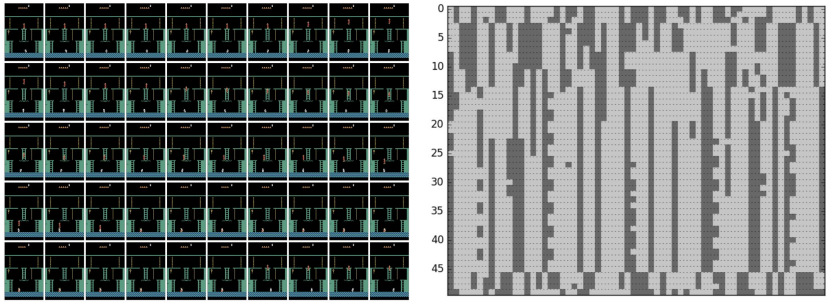
- Captures epistemic uncertainty
- Explicitly control state distance metric
- Can be hard to scale (but an active research area)

Estimating model uncertainty: **pseudo-count**

- Count or hash states you already visited



Bellemare et al (2016). Unifying count-based exploration and intrinsic motivation.



Tang et al (2017). #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning.

- Captures epistemic uncertainty
- Can be sensitive to state space in which you count

Estimating model uncertainty: **ensembles**

- Train multiple models independently and combine predictions across models

- Captures epistemic uncertainty
- Simple to implement and applicable in many contexts
- Can be sensitive to state space and network architecture

Estimating model uncertainty

- Multiple approaches, and an active research area in machine learning

  - Probabilistic neural networks
  - Bayesian neural networks
  - Gaussian processes
  - Pseudo-counts
  - Ensembles

- Ensembles currently popular due to simplicity and flexibility

Outline of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

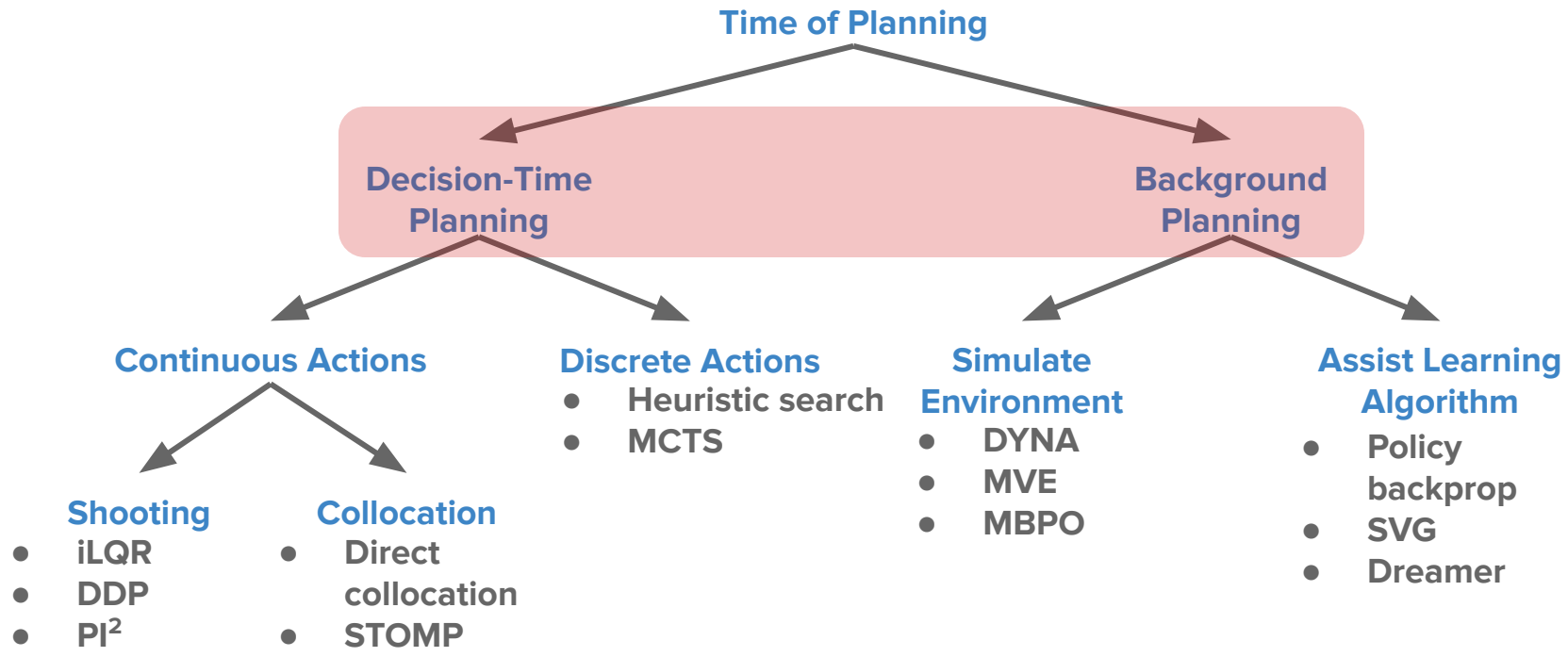
Estimating model uncertainty

- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

Combining planning and learning

- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

Landscape of model-based methods



Combining planning and learning

- Background and decision-time planning have complementary strengths and weaknesses

Combining planning and learning

- Background and decision-time planning have complementary strengths and weaknesses

Background	Decision-Time	
🟥	🟩	Observation space confound
🟥	🟩	Act on most recent state of the world
🟥	🟩	Act without any learning
🟥	🟩	Out of distribution behaviors
🟩	🟨	Partial observability
🟩	🟥	Computation at deployment
🟩	🟥	Predictability
🟩	🟥	Coherence

Combining planning and learning

- Background and decision-time planning have complementary strengths and weaknesses
- Can we combine both types of methods?

Outline of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

Estimating model uncertainty

- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

Combining planning and learning

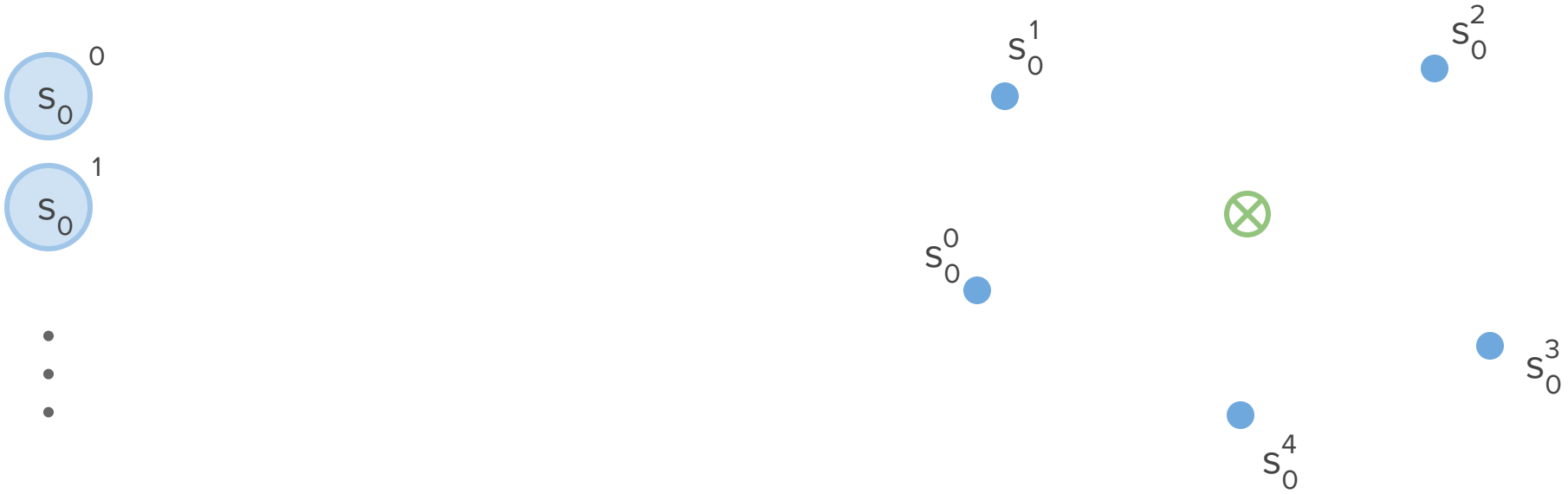
- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

Combining planning and learning: **distillation**

- Generate decision-time plans for multiple situations and distill into a policy/value

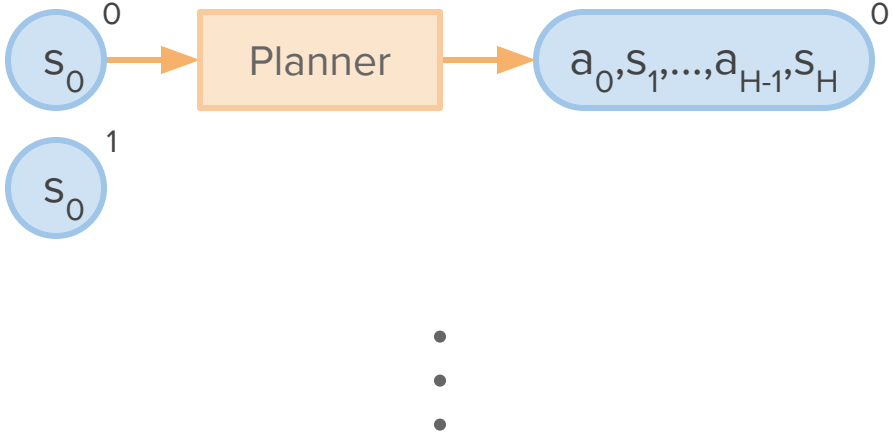
Combining planning and learning: **distillation**

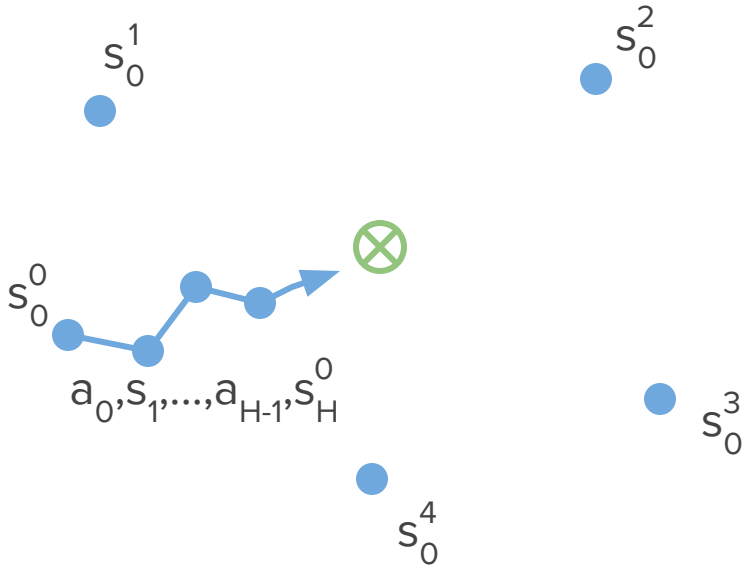
- Generate decision-time plans for multiple situations and distill into a policy/value



Combining planning and learning: **distillation**

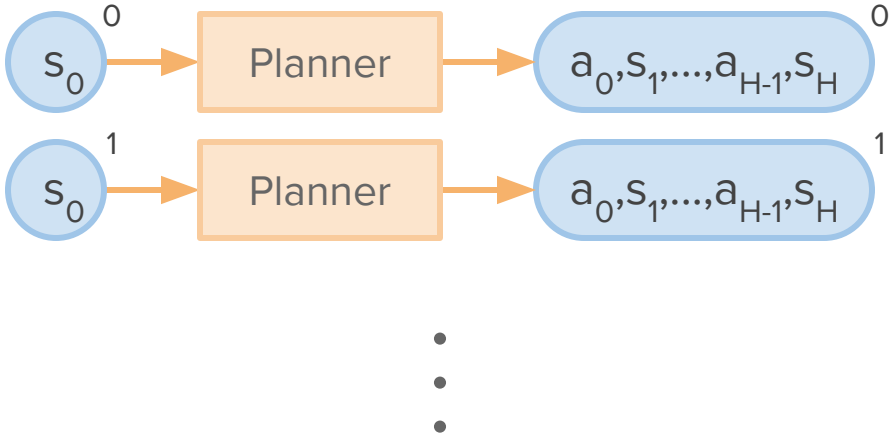
- Generate decision-time plans for multiple situations and distill into a policy/value

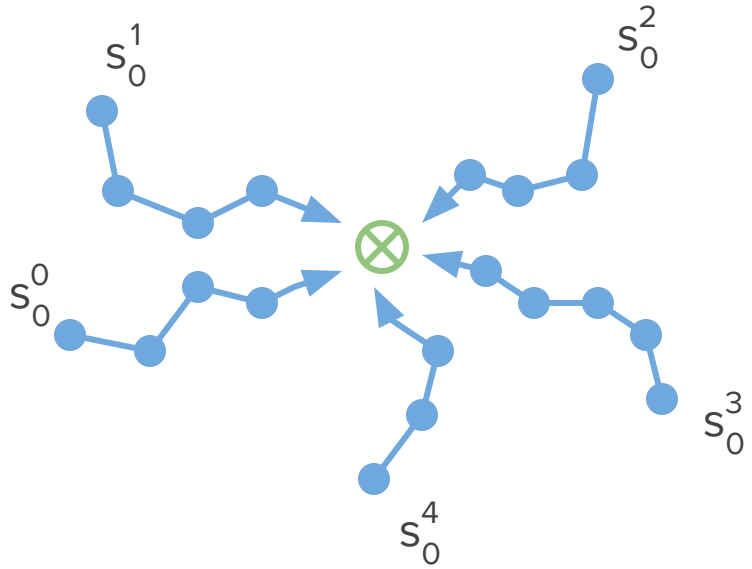




Combining planning and learning: **distillation**

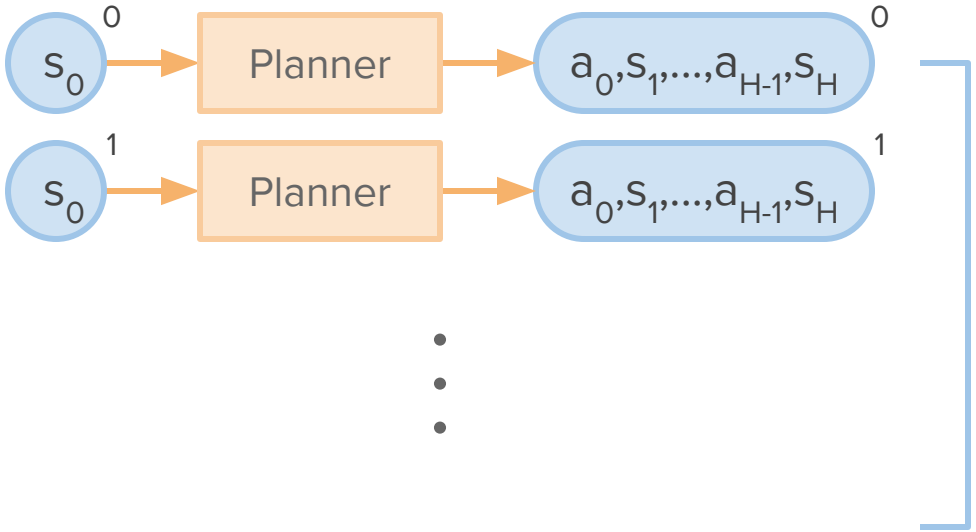
- Generate decision-time plans for multiple situations and distill into a policy/value





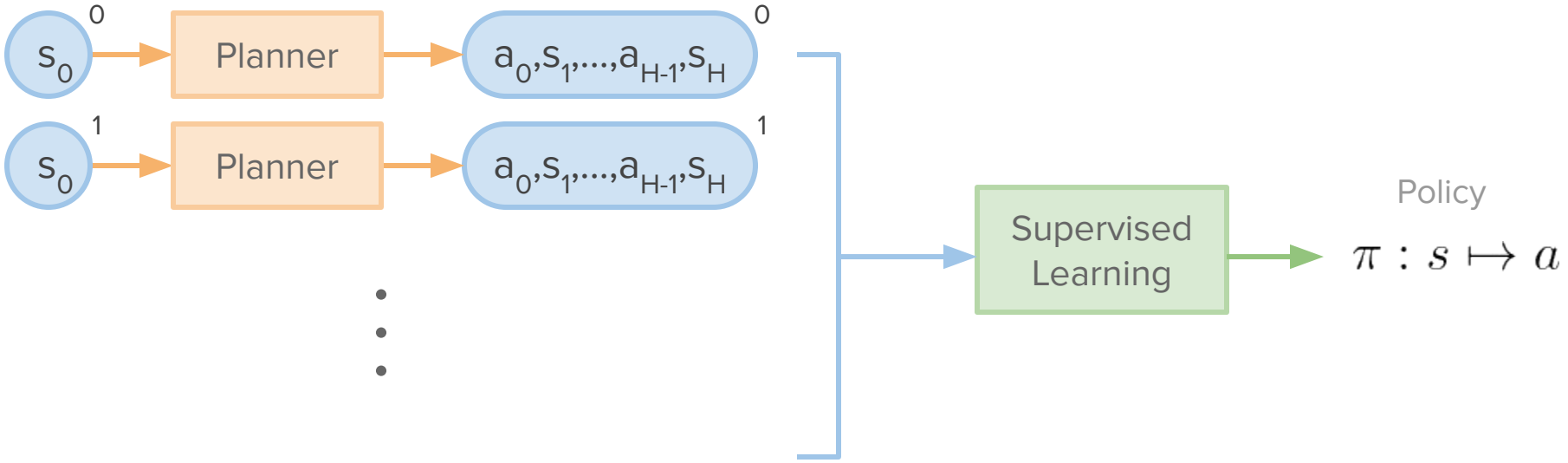
Combining planning and learning: **distillation**

- Generate decision-time plans for multiple situations and distill into a policy/value



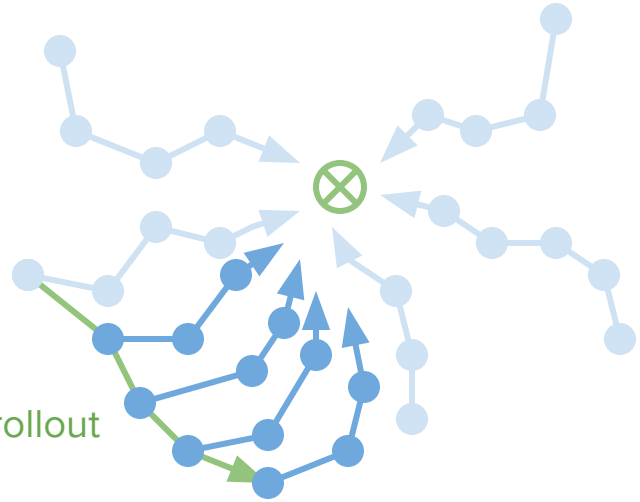
Combining planning and learning: **distillation**

- Generate decision-time plans for multiple situations and distill into a policy/value



Combining planning and learning: **distillation**

- Generate decision-time plans for multiple situations and distill into a policy/value
- What if learned policies have compounding errors?



Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

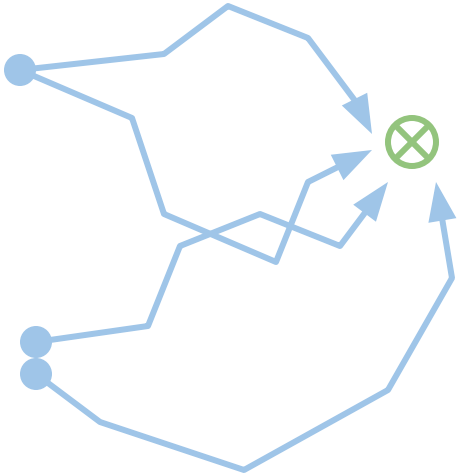
Algorithm 3.1: DAGGER Algorithm.

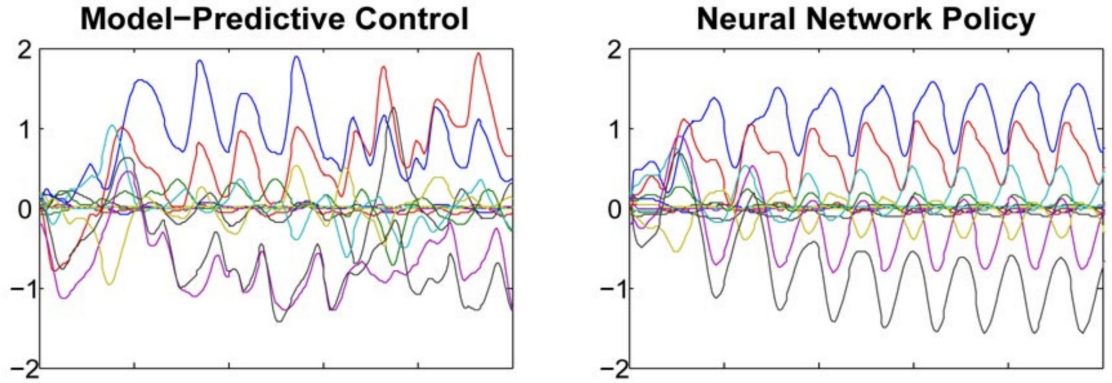
- Create plans from states visited by policy
- Add planned trajectories to distillation dataset (expands dataset where policy makes errors)

Ross et al (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning.

Combining planning and learning: **distillation**

- Generate decision-time plans for multiple situations and distill into a policy/value
- What if plans are inconsistent?

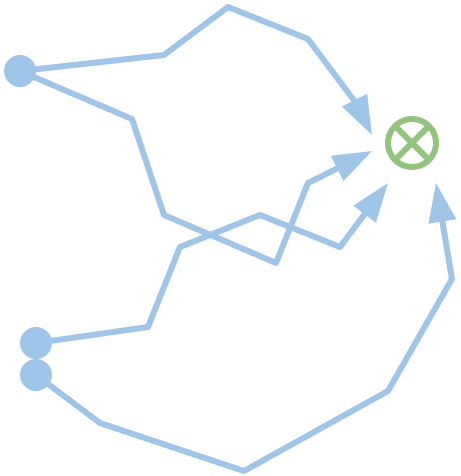


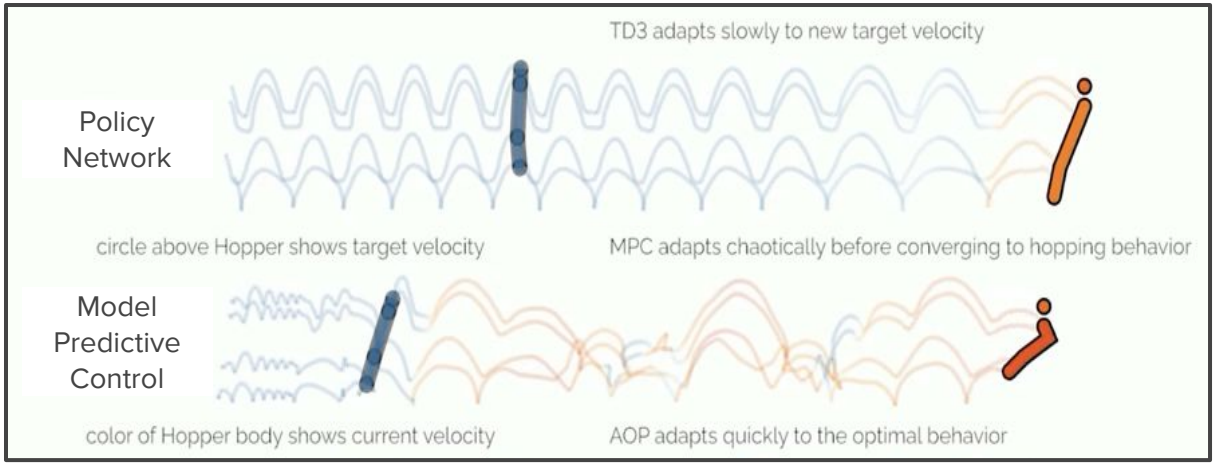


Mordatch et al (2015). Interactive Control of Diverse Complex Characters with Neural Networks.

Combining planning and learning: **distillation**

- Generate decision-time plans for multiple situations and distill into a policy/value
- What if plans are inconsistent?

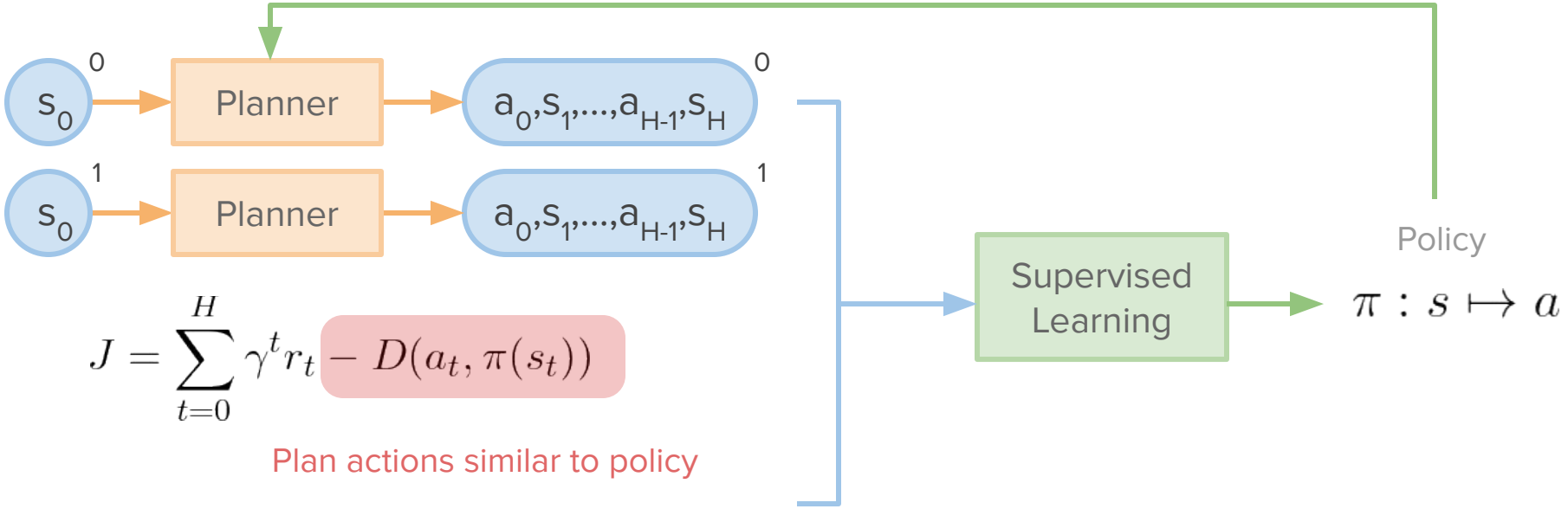




Lu et al (2019). Adaptive Online Planning for Continual Lifelong Learning.

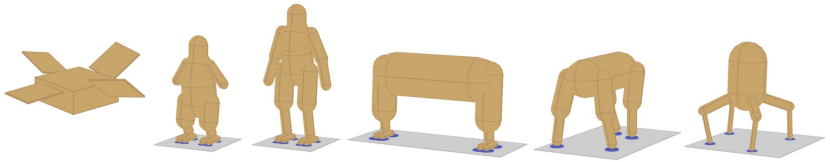
Combining planning and learning: **distillation**

- Generate decision-time plans for multiple situations and distill into a policy/value
- What if plans are inconsistent?

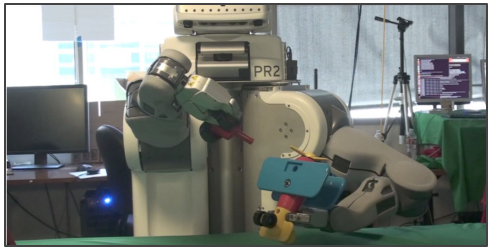


Combining planning and learning: **distillation**

- Generate decision-time plans for multiple situations and distill into a policy/value
- What if plans are inconsistent?



Mordatch et al (2015). Interactive Control of Diverse Complex Characters with Neural Networks.



Levine et al (2015). Learning Contact-Rich Manipulation Skills with Guided Policy Search.

$$J = \sum_{t=0}^{H} \gamma^t r_t - D(a_t, \pi(s_t))$$

Plan actions similar to policy

Algorithm 1 Expert Iteration

1: $\hat{\pi}_0$ = initial_policy()
2: π_0^* = build_expert($\hat{\pi}_0$)
3: **for** i = 1; i $\leq$ max_iterations; i++ **do**
4: S_i = sample_self_play($\hat{\pi}_{i-1}$)
5: $D_i = \{(s, \text{imitation\_learning\_target}(\pi^*_{i-1}(s)))|s \in S_i\}$
6: $\hat{\pi}_i$ = train_policy(D_i)
7: π_i^* = build_expert($\hat{\pi}_i$)
8: **end for**

Anthony et al (2017). Thinking Fast and Slow with Deep Learning and Tree Search.

Outline of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

Estimating model uncertainty

- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

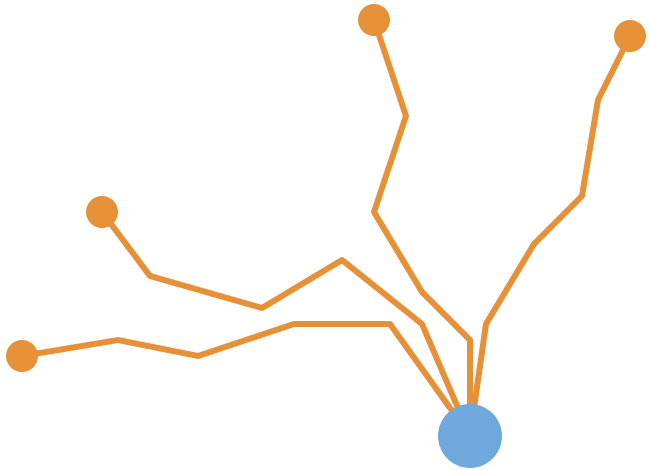
Combining planning and learning

- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

Combining planning and learning: **terminal value**

- **Problem:** trajectory optimization horizon is finite

$$J^H = \sum_{t=0}^{H} \gamma^t r_t$$

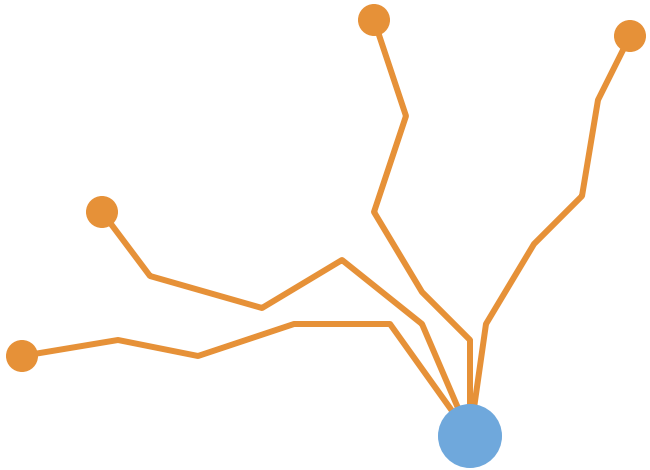


Combining planning and learning: **terminal value**

- **Problem:** trajectory optimization horizon is finite

$$J^H = \sum_{t=0}^{H} \gamma^t r_t$$

- Can lead to myopic behavior



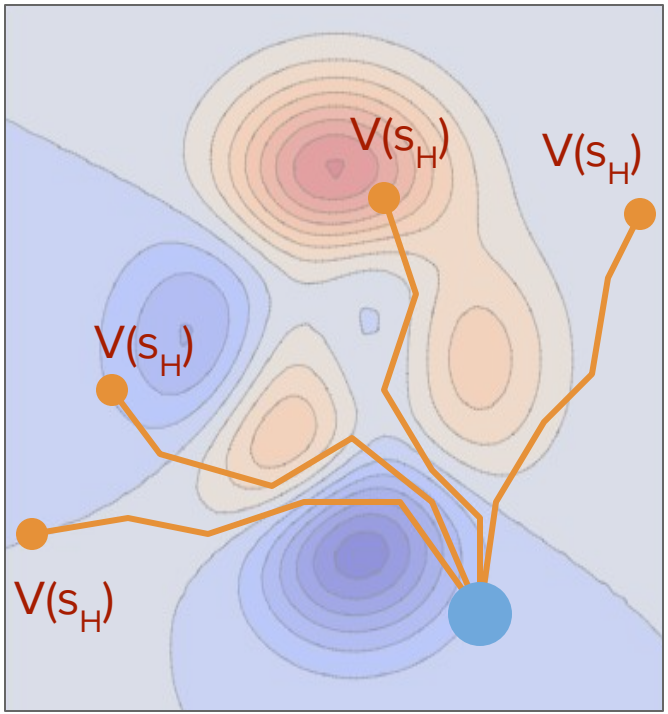
Combining planning and learning: **terminal value**

- **Problem:** trajectory optimization horizon is finite

$$J^H = \sum_{t=0}^{H} \gamma^t r_t$$

- Can lead to myopic behavior
- **Solution:** Append value function at terminal state

$$J^\infty = \sum_{t=0}^{\infty} \gamma^t r_t = \sum_{t=0}^{H} \gamma^t r_t + \gamma^H V(s_H)$$



Combining planning and learning: **terminal value**

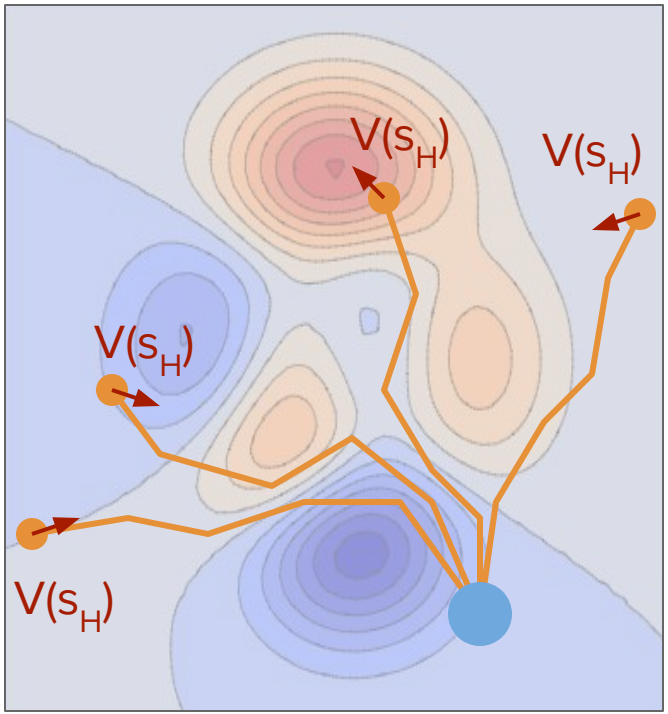
- **Problem:** trajectory optimization horizon is finite

$$J^H = \sum_{t=0}^{H} \gamma^t r_t$$

- Can lead to myopic behavior
- **Solution:** Append value function at terminal state

$$J^\infty = \sum_{t=0}^{\infty} \gamma^t r_t = \sum_{t=0}^{H} \gamma^t r_t + \gamma^H V(s_H)$$

- Learned value function guides plans to long-term good states



Combining planning and learning: **terminal value**

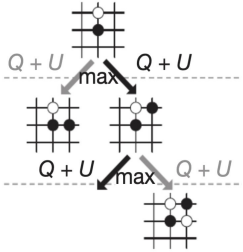
- **Problem:** trajectory optimization horizon is finite

$$J^H = \sum_{t=0}^{H} \gamma^t r_t$$

- Can lead to myopic behavior
- **Solution:** Append value function at terminal state

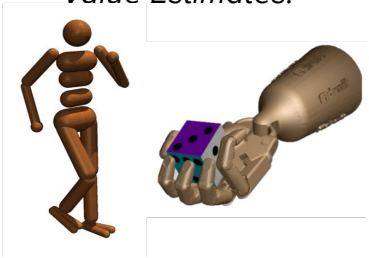
$$J^\infty = \sum_{t=0}^{\infty} \gamma^t r_t = \sum_{t=0}^{H} \gamma^t r_t + \gamma^H V(s_H)$$

- Applicable in discrete and continuous settings



Silver et al (2017). Mastering the game of Go without human knowledge.

Hamrick et al (2020). Combining Q-Learning and Search with Amortized Value Estimates.



Lowrey et al (2019). Plan Online, Learn Offline.

Outline of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

Estimating model uncertainty

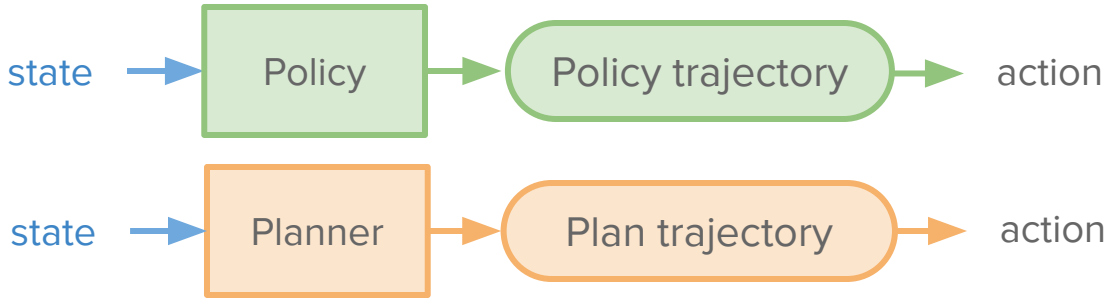
- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

Combining planning and learning

- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

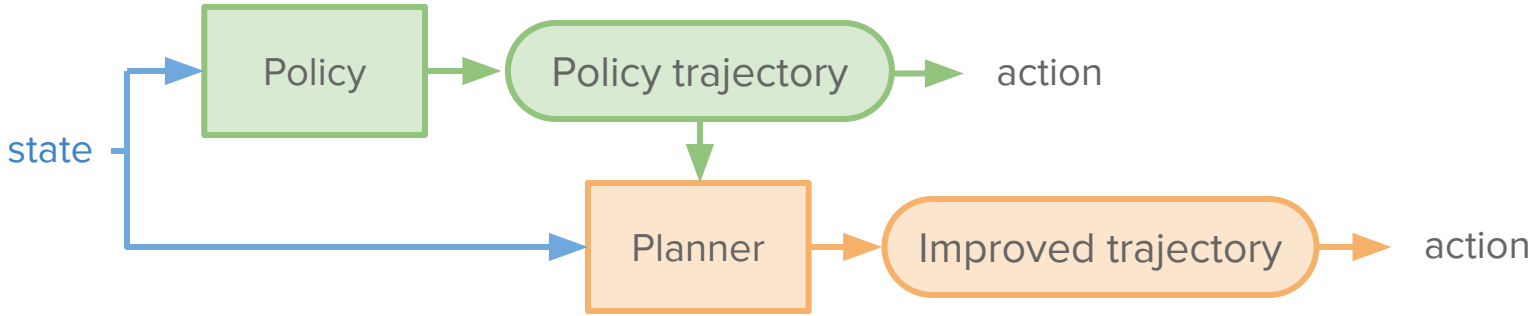
Combining planning and learning: **policy improvement**

- Previously either used policy (background) or decision-time planner to act



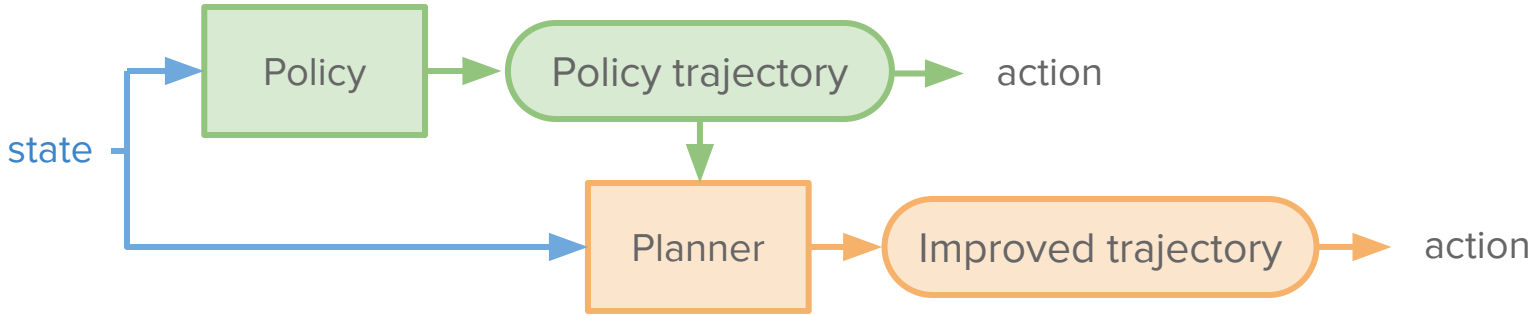
Combining planning and learning: **policy improvement**

- Apply planning as an operator to improve policy



Combining planning and learning: **policy improvement**

- Apply planning as an operator to improve policy
- Train policy so improvement operator has no effect



Silver et al (2017). Mastering the game of Go without human knowledge.

Levine et al (2014). Guided Policy Search under Unknown Dynamics.

Anthony et al (2017). Thinking Fast and Slow with Deep Learning and Tree Search.

Outline of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

Estimating model uncertainty

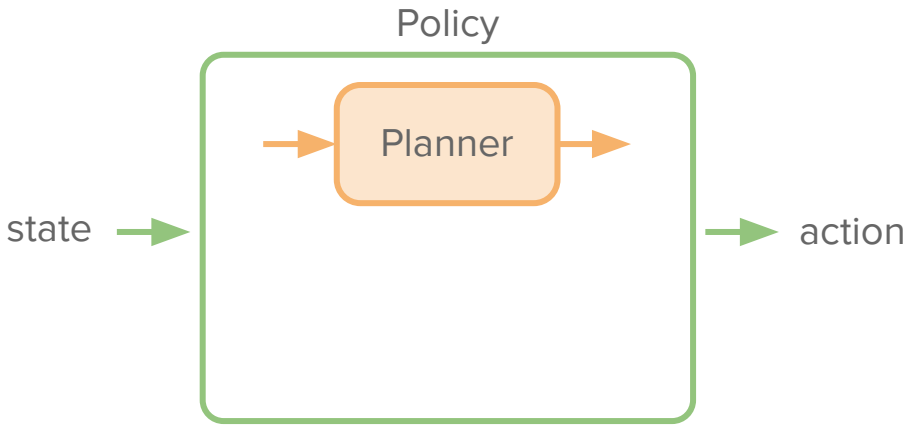
- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

Combining planning and learning

- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

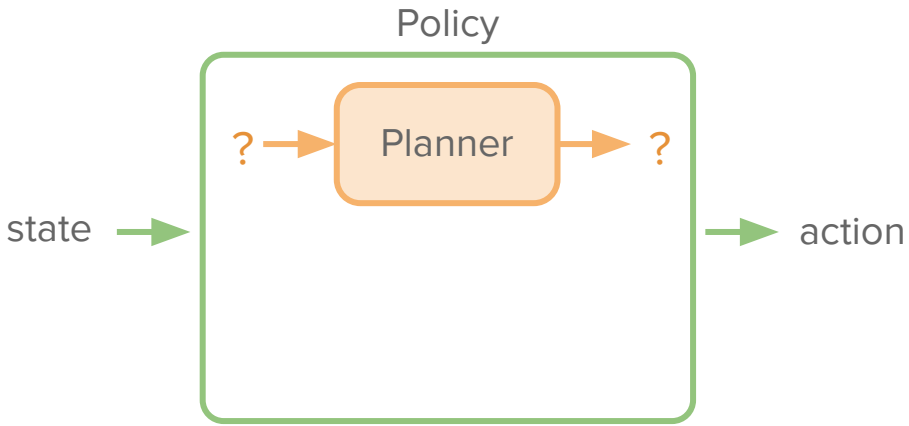
Combining planning and learning: **implicit planning**

- Insert planner as a component *inside* policy network and train end-to-end



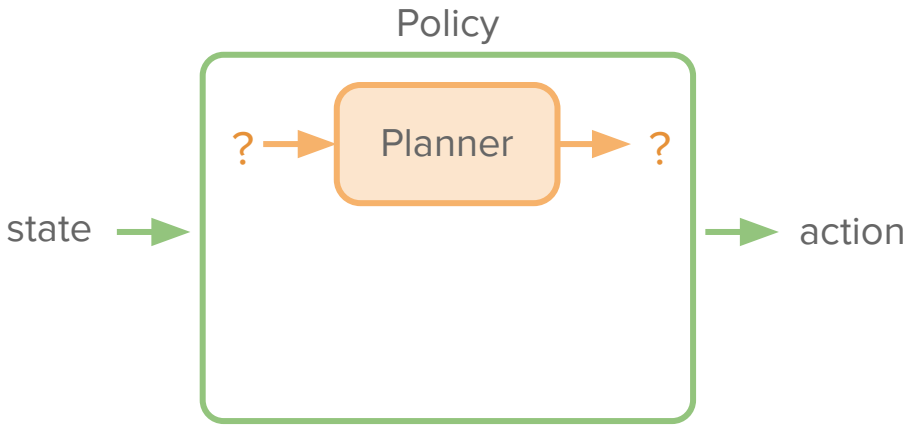
Combining planning and learning: **implicit planning**

- Insert planner as a component *inside* policy network and train end-to-end
- Policy network dictates abstract state/action spaces to plan in



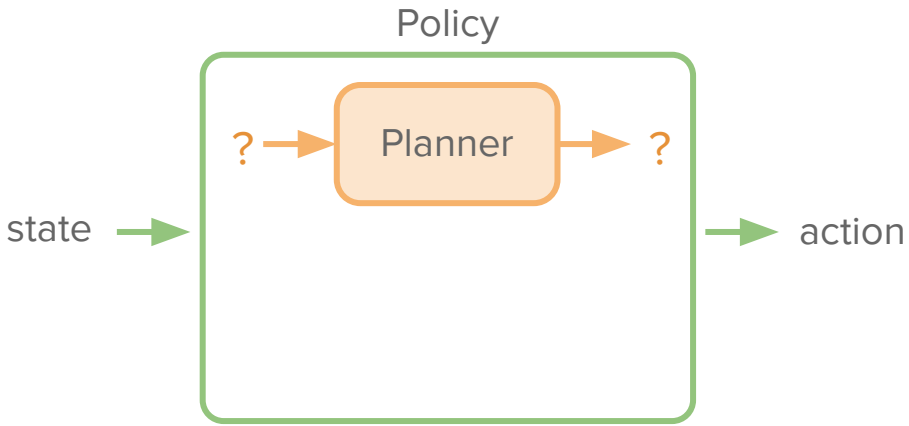
Combining planning and learning: **implicit planning**

- Insert planner as a component *inside* policy network and train end-to-end
- Policy network dictates abstract state/action spaces to plan in
- Requires differentiating through the planning algorithm



Combining planning and learning: **implicit planning**

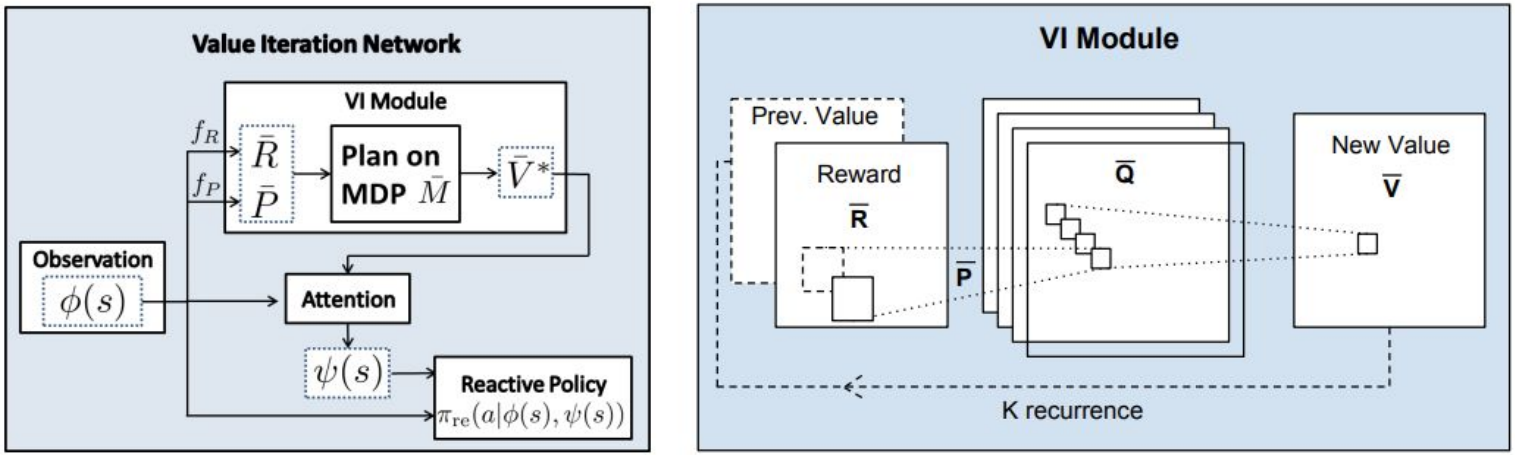
- Insert planner as a component *inside* policy network and train end-to-end
- Policy network dictates abstract state/action spaces to plan in
- Requires differentiating through the planning algorithm
- Multiple algorithms we've seen have been integrated



Combining planning and learning: **implicit planning**

- Insert planner as a component *inside* policy network and train end-to-end

Planner = Value iteration

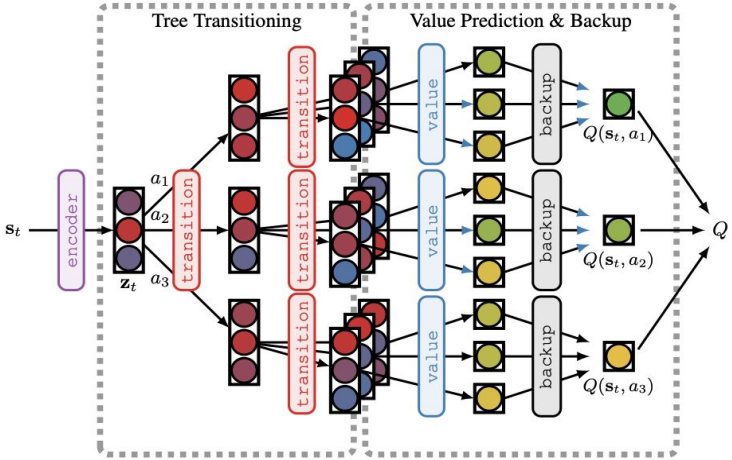


Tamar et al (2016). Value iteration networks.

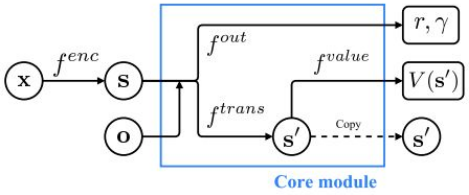
Combining planning and learning: **implicit planning**

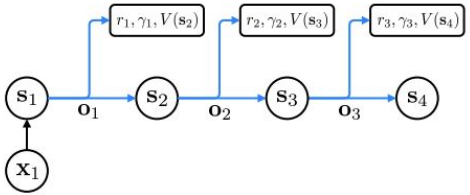
- Insert planner as a component *inside* policy network and train end-to-end

Planner = Tree search



Farquhar et al (2017). TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning.



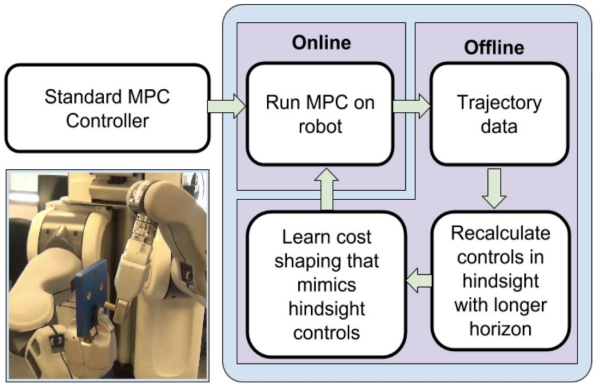


Oh et al (2017). Value Prediction Network.

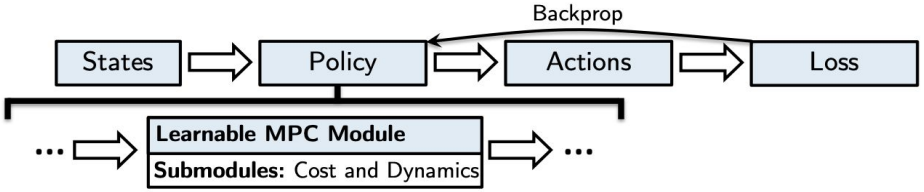
Combining planning and learning: **implicit planning**

- Insert planner as a component *inside* policy network and train end-to-end

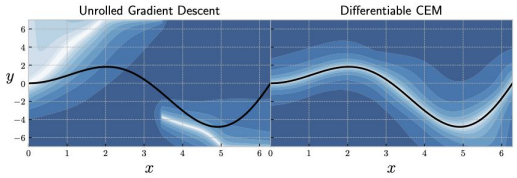
Planner = Continuous trajectory optimization



Tamar et al (2017). Learning from the Hindsight Plan – Episodic MPC Improvement.



Amos et al (2019). Differentiable MPC for End-to-end Planning and Control.

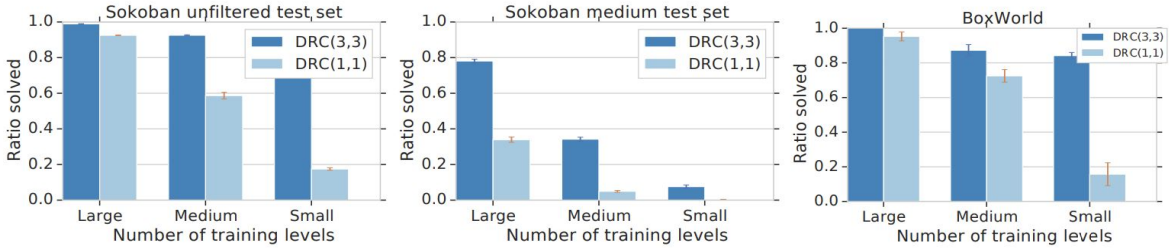


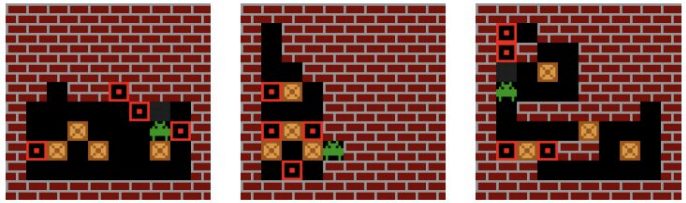
Amos et al (2020). The Differentiable Cross-Entropy Method.

Combining planning and learning: **implicit planning**

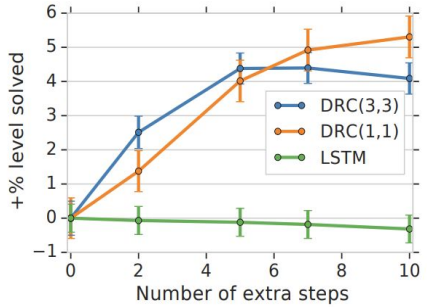
- Insert planner as a component *inside* policy network and train end-to-end
- Or could planning *emerge* in generic policies and model-free training?

Generic, **recurrently-applied** networks exhibit generalization similar to model-based planners





Utilize additional computation steps



Guez et al (2019). An Investigation of Model-Free Planning.

Summary of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation

Acting under imperfect models

- Replan via model-predictive control
- Plan conservatively

Estimating model uncertainty

- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

Combining planning and learning

- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

Next part:

What else can models be used for and what's missing from current methods?

Model-Based Methods in Reinforcement Learning

Part 4: Beyond Vanilla MBRL

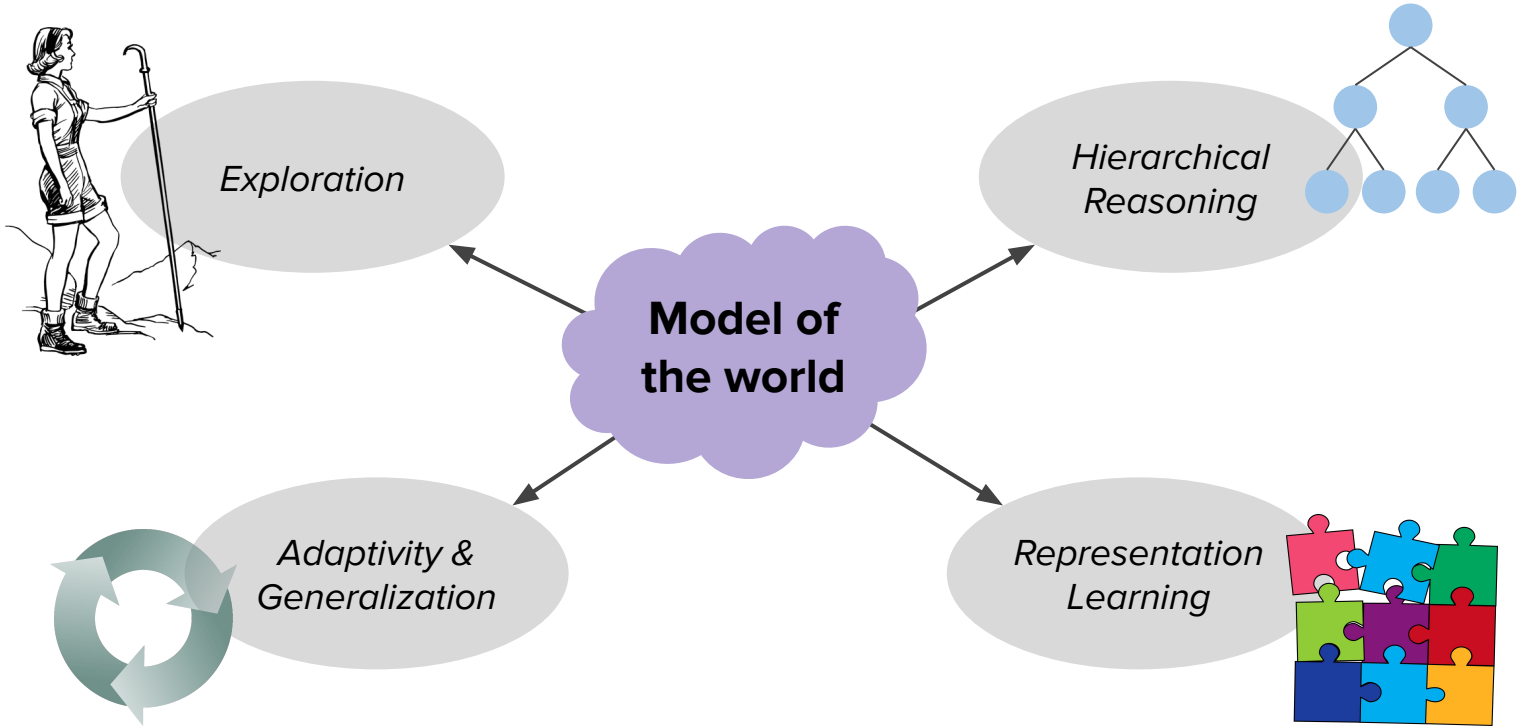
Igor Mordatch (Google) & Jessica Hamrick (DeepMind)

ICML 2020 Tutorial

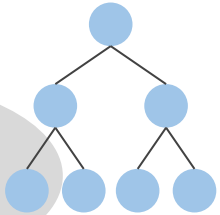
Outline

1. Introduction and motivation
2. Problem statement
3. What is a “model”?
4. What is model-based control?
5. Model-based control in the loop
6. **What else can models be used for?**
7. What’s missing from model-based methods?
8. Conclusion

What else are models good for?



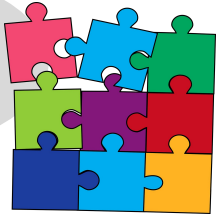
What else are models good for?



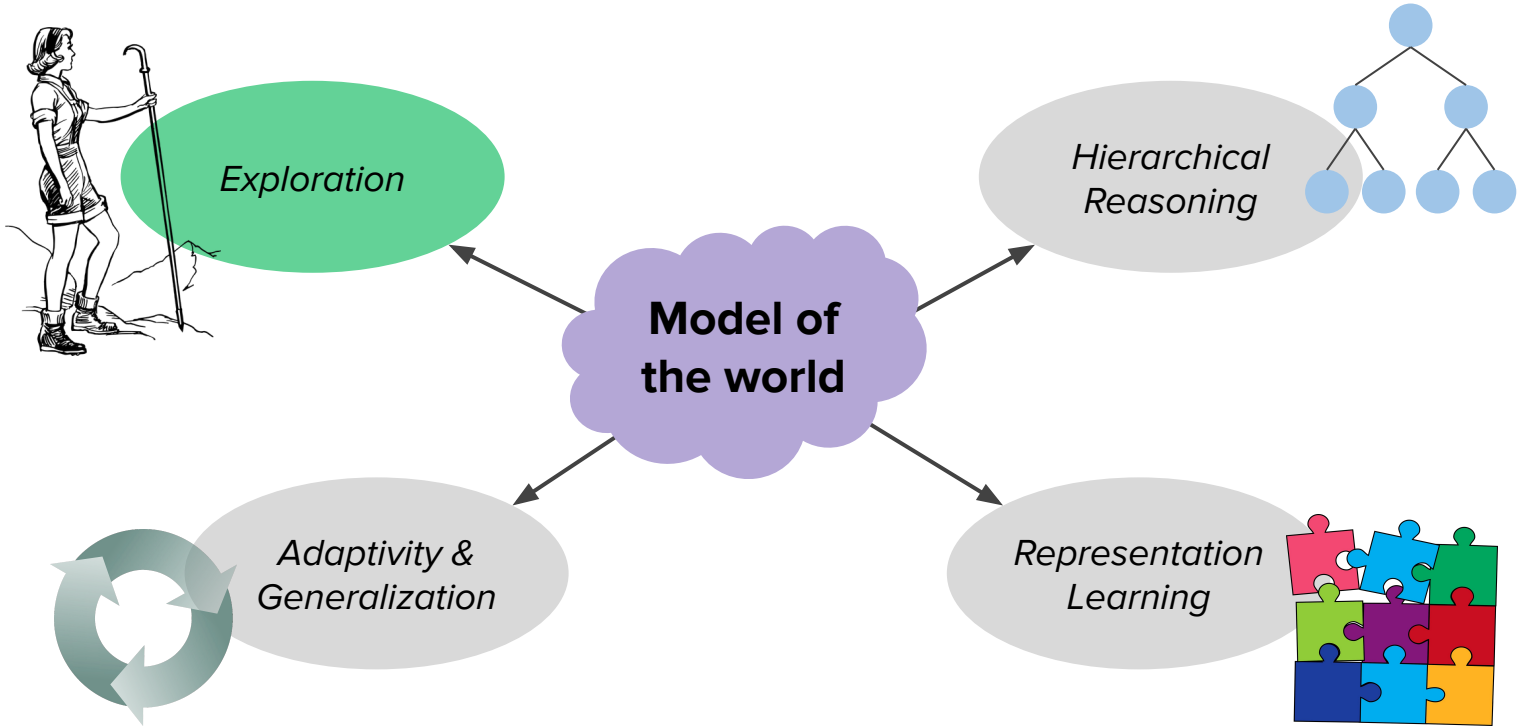
Also... reasoning about other agents, dealing with partial observability, language understanding, commonsense reasoning, and more!

Generalization

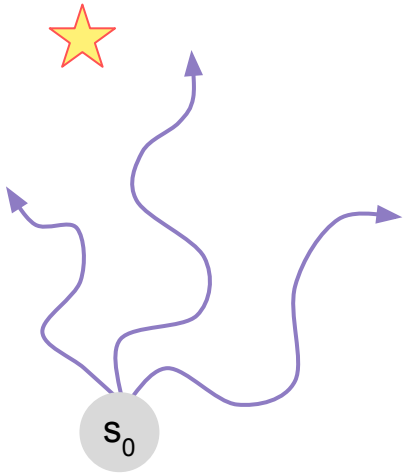
Learning



What else are models good for?

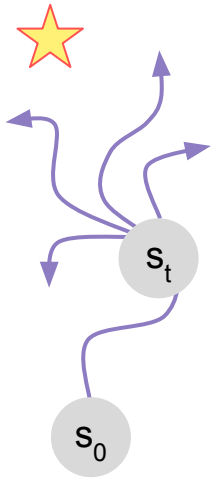


Exploiting “resettability”



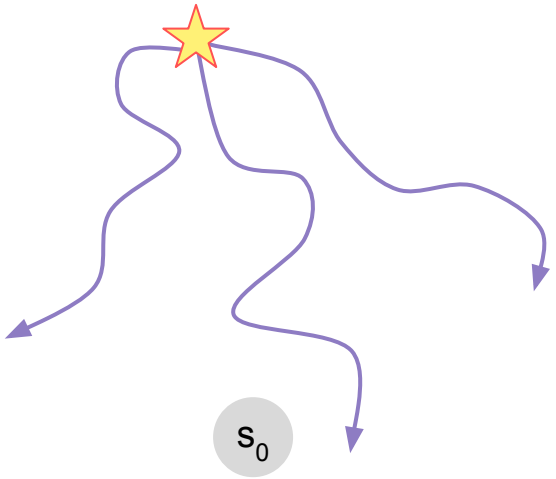
Forward exploration

Note: only works when the environment itself is resettable!



Return, then explore

Ecoffet et al. (2019). Go-explore: a new approach for hard-exploration problems. arXiv.

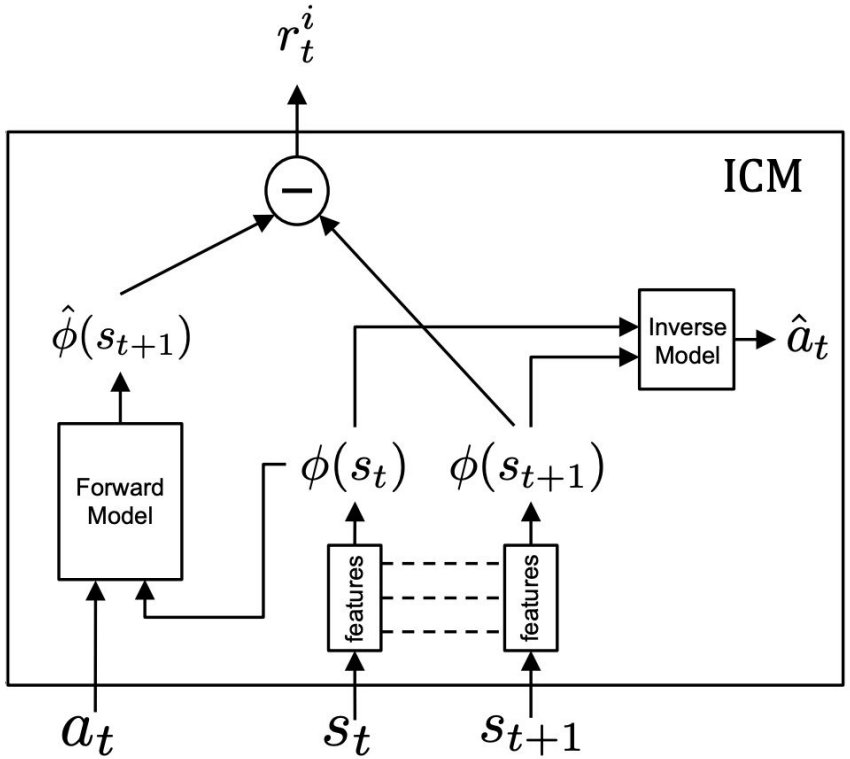


Backward exploration

Edwards, Downs, & Davidson (2018). Forward-Backward Reinforcement Learning. arXiv.

Agostinelli et al. (2019). Solving the Rubik's Cube with Deep Reinforcement Learning and Search. Nature Machine Intelligence.

Curiosity-based exploration

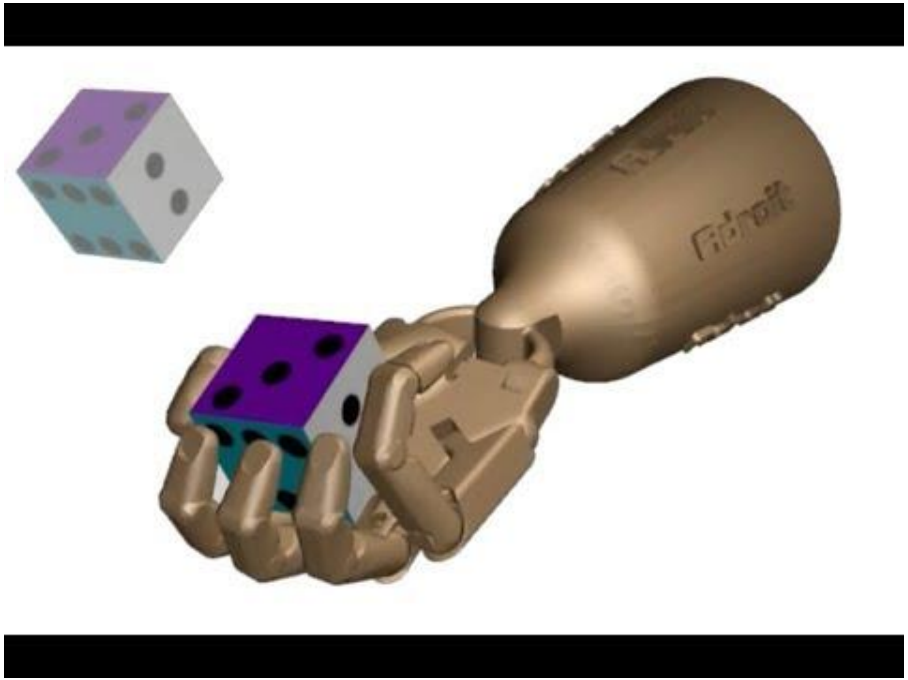


- Use forward model prediction error as an intrinsic reward
- Train policy to maximize intrinsic reward
- Encourages the agent to **revisit** states that are novel or unexpected

Schmidhuber (1991). Curious model-building control systems. IJCNN.

Pathak et al. (2017). Curiosity-driven exploration by self-supervised prediction. ICML.

Planning to explore



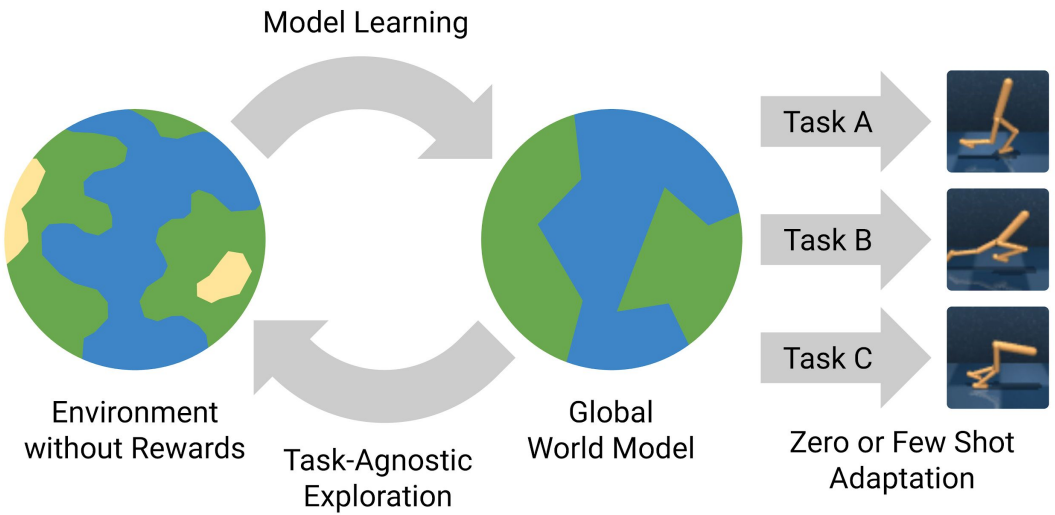
- Compute intrinsic reward **during (decision-time) planning** to direct the agent into new regions of state space

- Intrinsic reward = softmax across an ensemble of value functions

$$\hat{V}(s) = \log\left(\sum_{k=1}^{K} \exp\left(\kappa\, \hat{V}_{\theta_k}(s)\right)\right)$$

Lowrey et al. (2019). Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control. ICLR 2019.

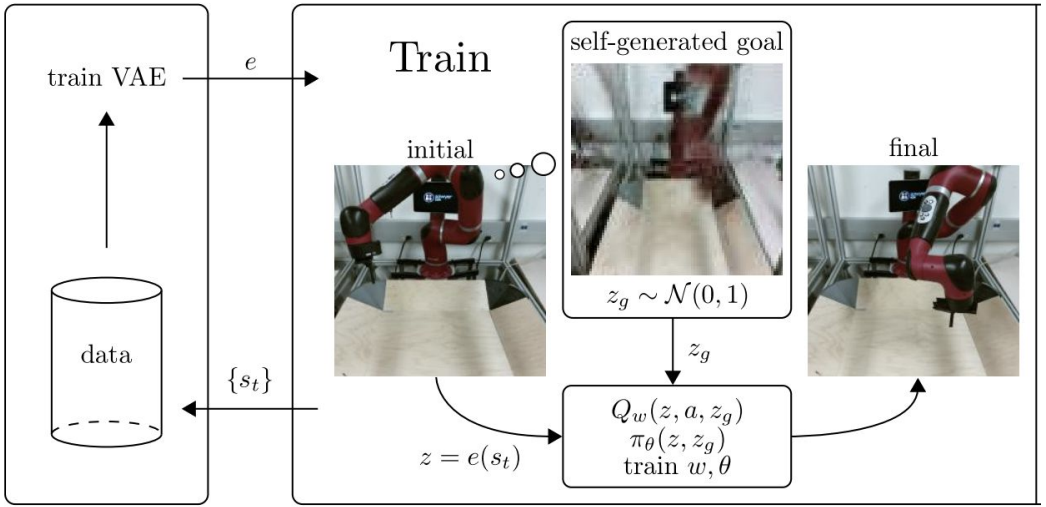
Planning to explore



- Compute intrinsic reward **during (background) planning** to direct the agent into new regions of state space
- Intrinsic reward = disagreement across forward model predictions
- Task agnostic, which enables rapid few-shot learning & adaptation

Sekar et al. (2020). Planning to Explore via Self-Supervised World Models. ICML.

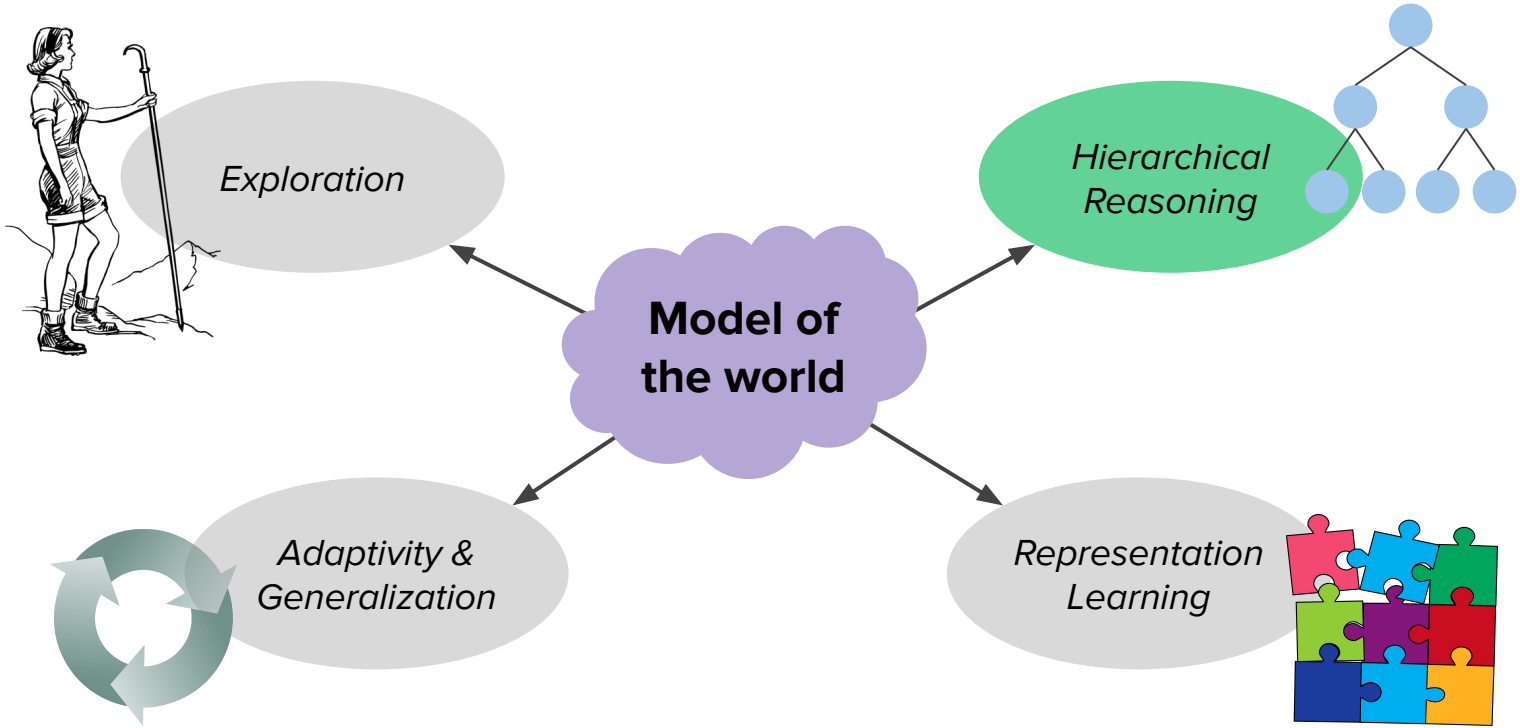
Goal-directed exploration



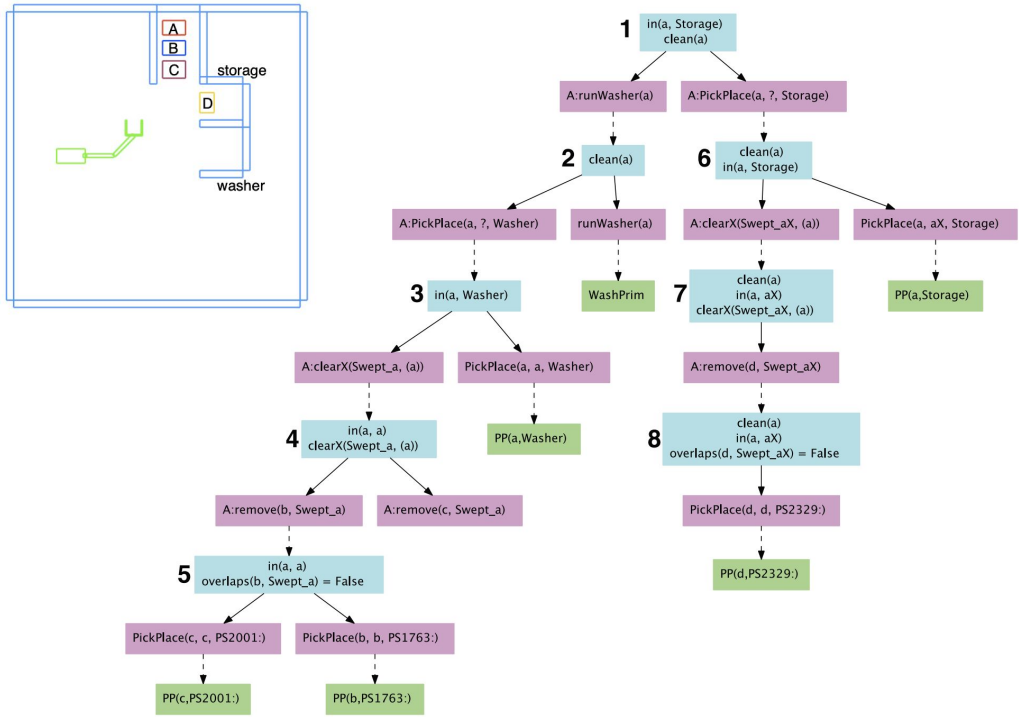
- Learn a density model over states, and sample goals from this model
- Train a policy to achieve the imagined goals
- At test time, new goal images can be provided

Nair, Pong, et al. (2018). Visual Reinforcement Learning with Imagined Goals. NeurIPS.

What else are models good for?



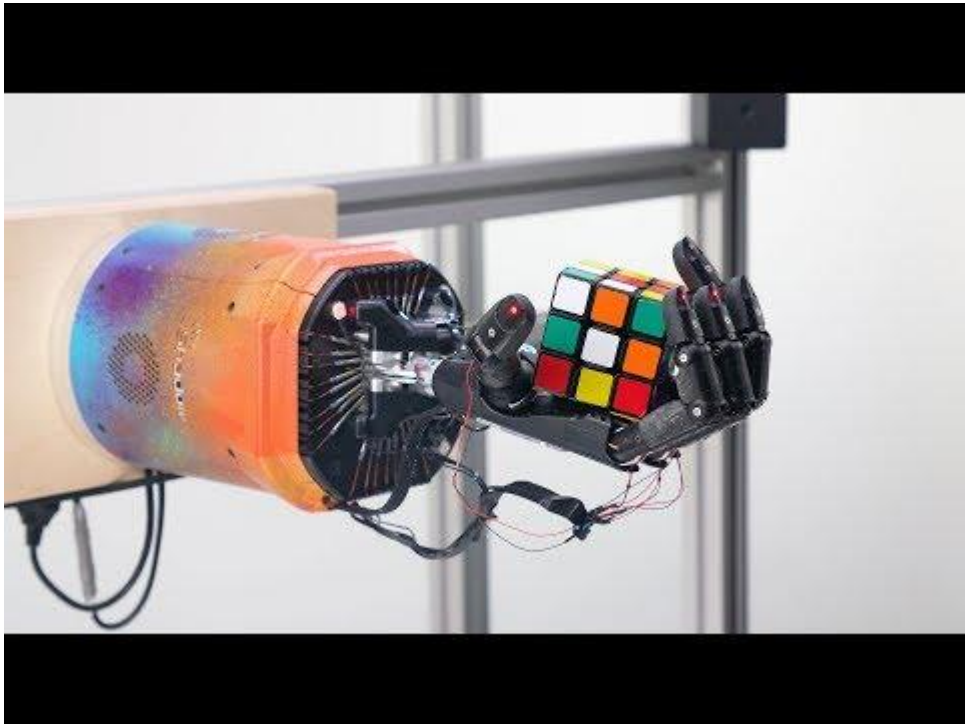
Task and motion planning (TAMP)



- Perform symbolic task planning at the high level to select high-level goals and/or skills
- Perform motion planning at the low level to achieve goals or execute skills
- Enables solving very long-horizon problems

Kaelbling & Lozano-Pérez (2011). Hierarchical Task and Motion Planning in the Now. ICRA.

Task planning: Rubik’s cube

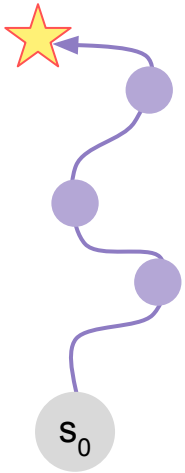


- Use Kociemba’s algorithm to generate a solution (plan) of high-level actions (e.g. rotate face)
- Low-level NN policy generates the controls needed to achieve high-level goals

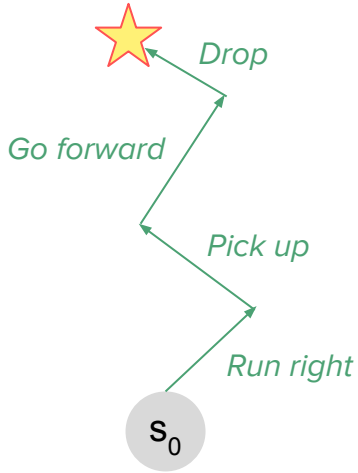
OpenAI et al. (2019). Solving Rubik's Cube with a Robot Hand. arXiv.

Where does the high-level state space come from?

Subgoal-based approaches



Skill / option-based approaches



Lots of work in model-free hierarchical RL breaks down into these approaches too!

Here: focus on recent MBRL approaches in HRL

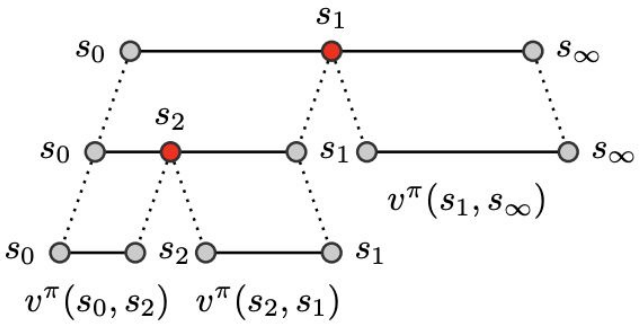
(Note: these approaches are two sides of the same coin! See: Konidaris et al. (2018). From Skills to Symbols)

Subgoal-based approaches

- Rather than learning a transition model, learn a universal value function approximator *V(s, g)*
  (see: Schaul et al. (2015), ICML)

- A plan of length *k* is then given by maximizing:

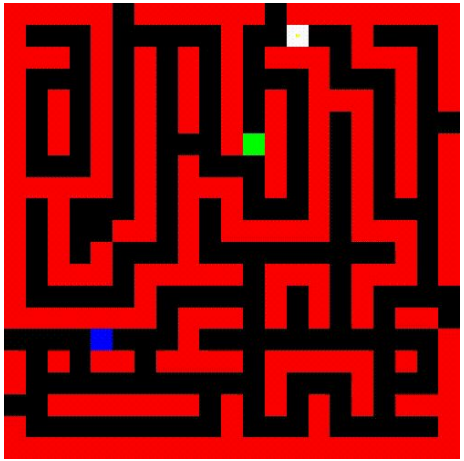
$$\arg\max_{\{s_i\}_{i=1}^{k}} \left(V(s_0, s_1) + V(s_k, s_g) + \sum_{i=1}^{k-1} V(s_i, s_{i+1}) \right)$$



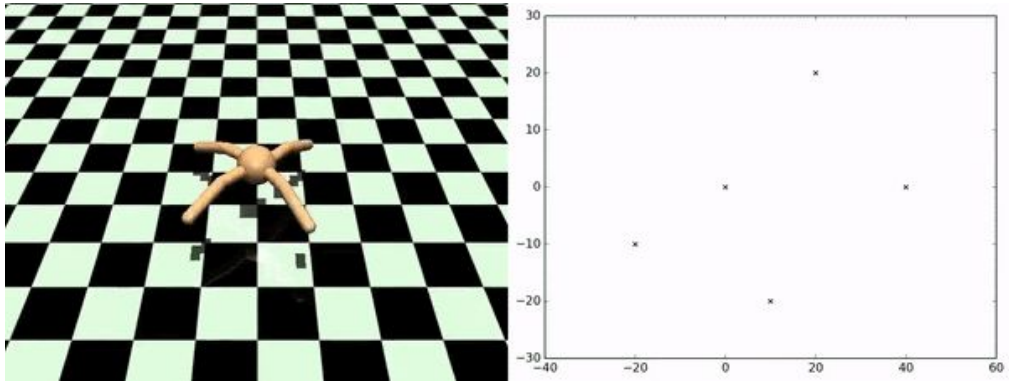
Nasiriany et al. (2019). Planning with Goal-Conditioned Policies. NeurIPS.

Jurgenson et al. (2019). Sub-Goal Trees -- A Framework for Goal-Directed Trajectory Prediction and Optimization. arXiv.

Parascandolo, Buesing, et al. (2020). Divide-and-Conquer Monte Carlo Tree Search For Goal-Directed Planning. arXiv.



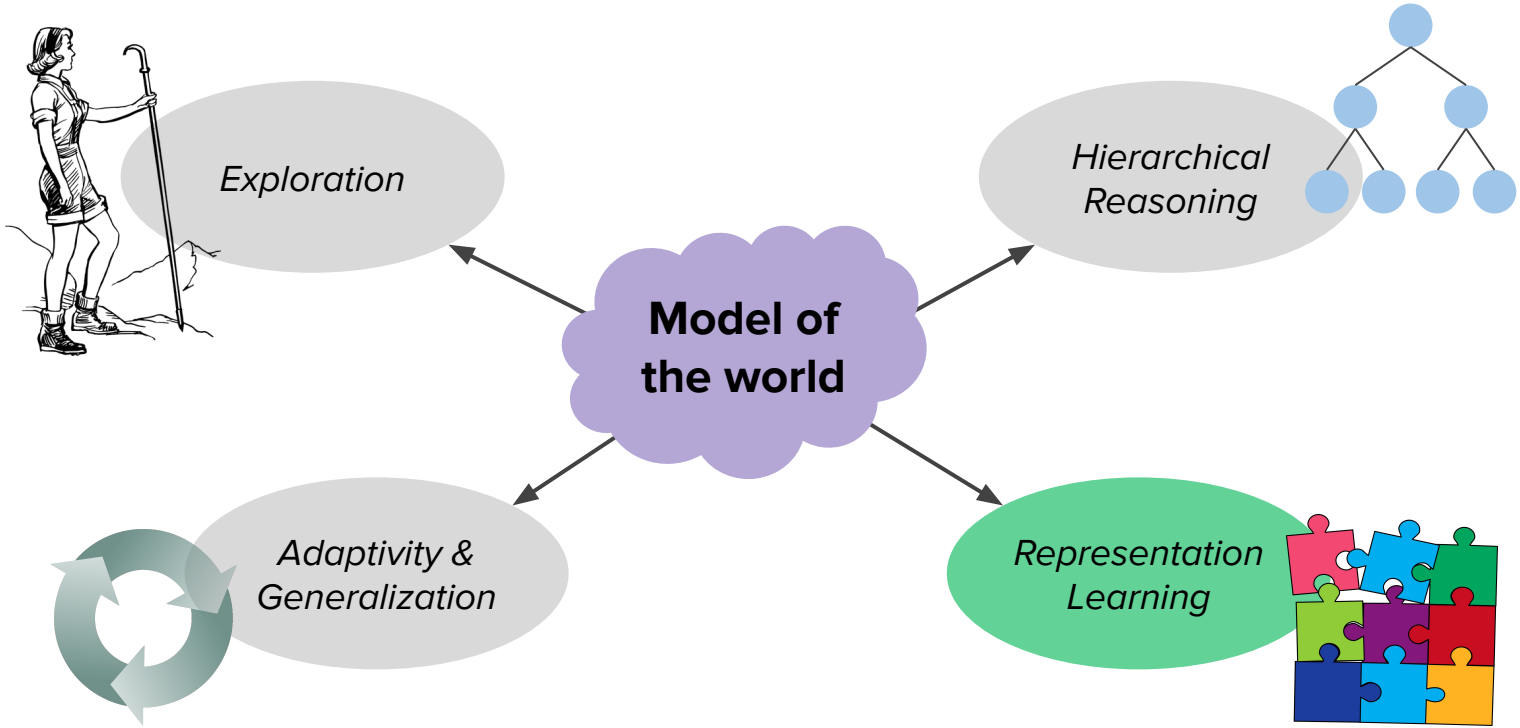
Skill-based approaches



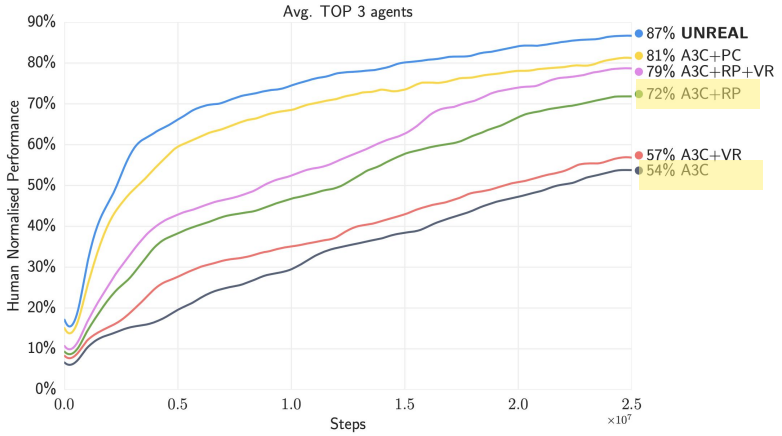
- Learn set of skills whose outcomes are (1) easy to predict and (2) diverse
- Learn dynamics model over skills, and plan with MPC
- Can solve long-horizon sequences of high-level goals with no additional learning

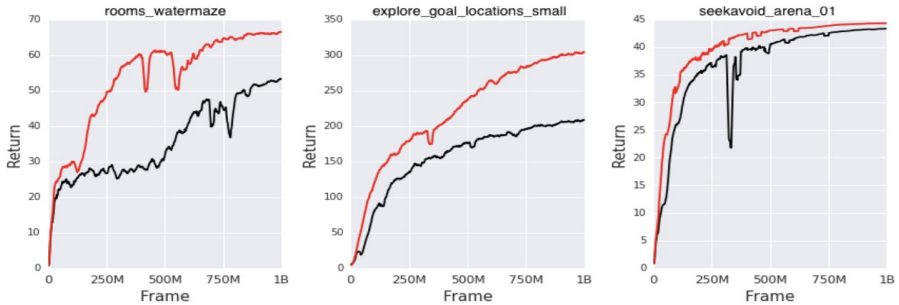
Sharma et al. (2020). Dynamics-Aware Unsupervised Discovery of Skills. ICLR.

What else are models good for?



Model learning as an auxiliary loss



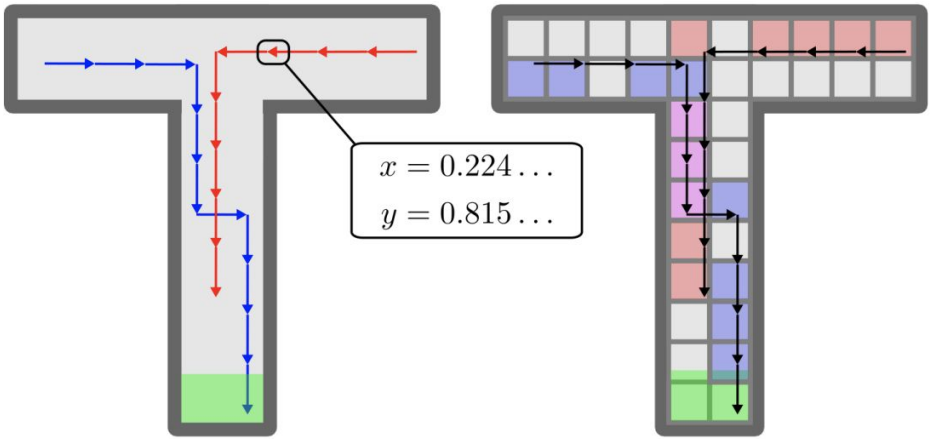


- Learn to predict features of the world such as reward or future observations
- Features are not used for planning, only for representation learning
- Features which capture the structure of the world are useful!

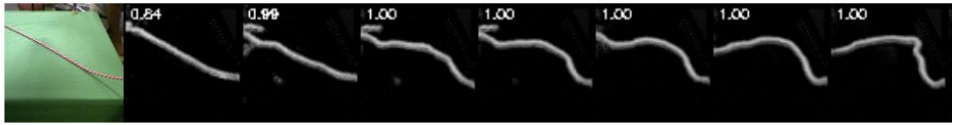
Jaderberg et al. (2017). Reinforcement learning with unsupervised auxiliary tasks. ICLR 2017.

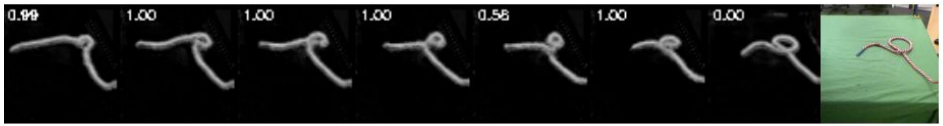
van den Oord, Li, & Vinyals (2019). Representation Learning with Contrastive Predictive Coding. arXiv.

“Plannable” representations



CIGAN



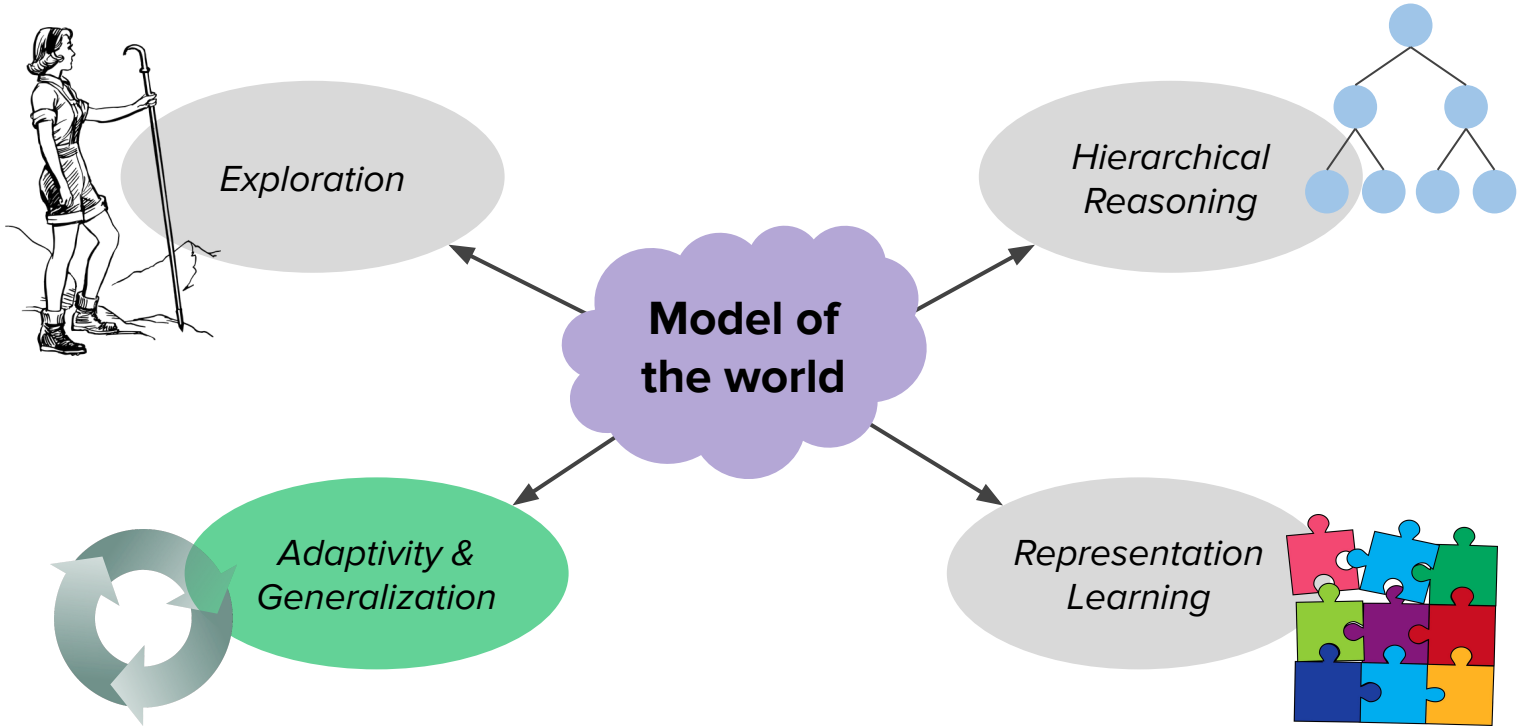


- Learn an embedding of states that is easier to plan in, e.g.
  - Discretized
  - States that can be transitioned between should be near to each other in latent space!
- Related to notions in hierarchical RL (state abstraction)

Corneil et al. (2018). Efficient Model-Based Deep Reinforcement Learning with Variational State Tabulation. ICML.

Kurutach et al. (2018). Learning Plannable Representations with Causal InfoGAN. NeurIPS.

What else are models good for?

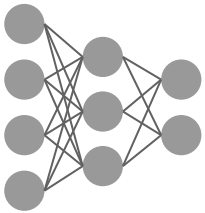


Different ways to adapt to new scenarios

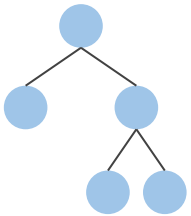
Change in rewards

Change in dynamics

Adapt the policy
(SLOW)

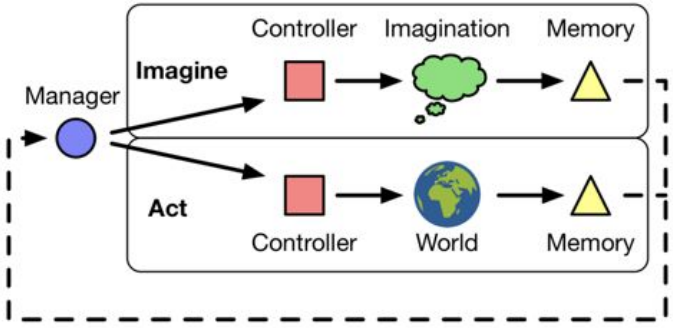


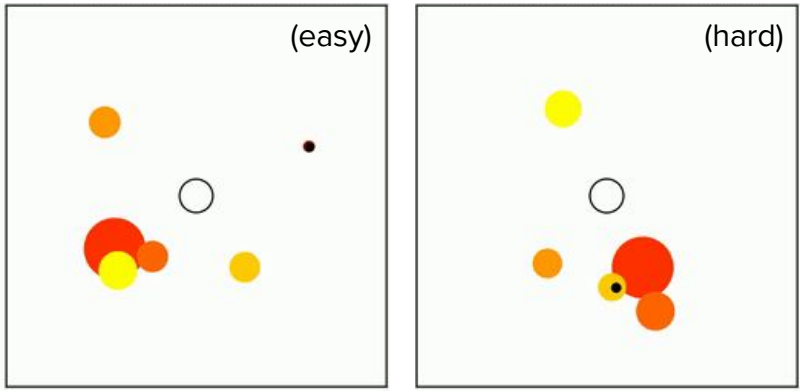
Adapt the planner
(FAST)



Adapt the model
(FAST)

Adapting the planner in new states



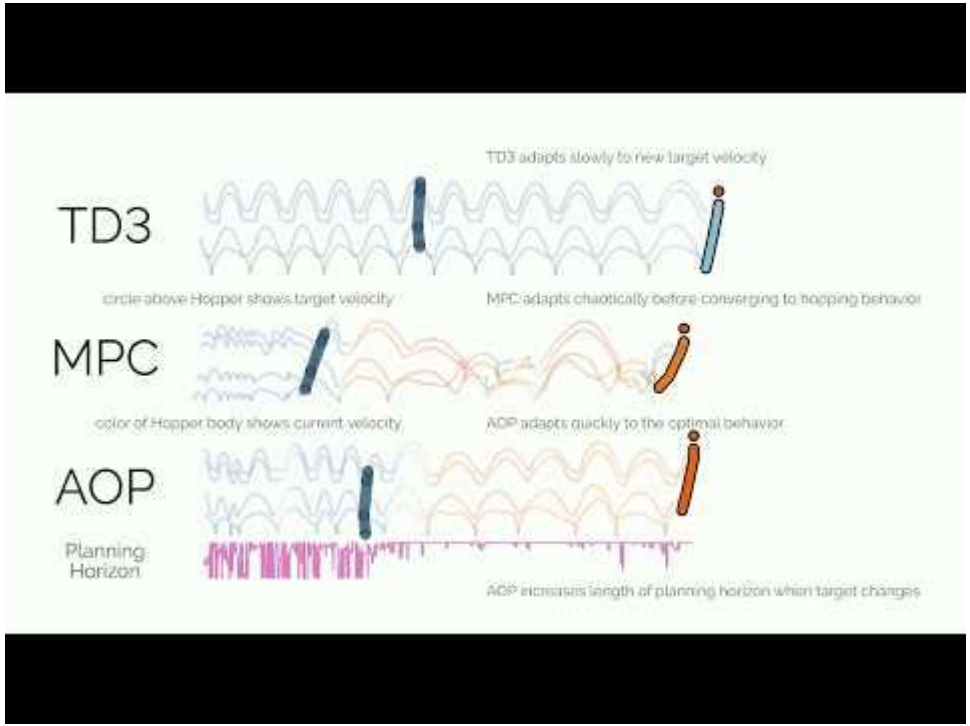


- Policy may not generalize to all states (especially in combinatorial spaces)
- Detect when planning is required, and adapt the amount of planning depending on the difficulty of the task

Hamrick et al. (2017). Metacontrol for adaptive imagination-based optimization. ICLR 2017.

Pascanu, Li, et al. (2017). Learning model-based planning from scratch. arXiv.

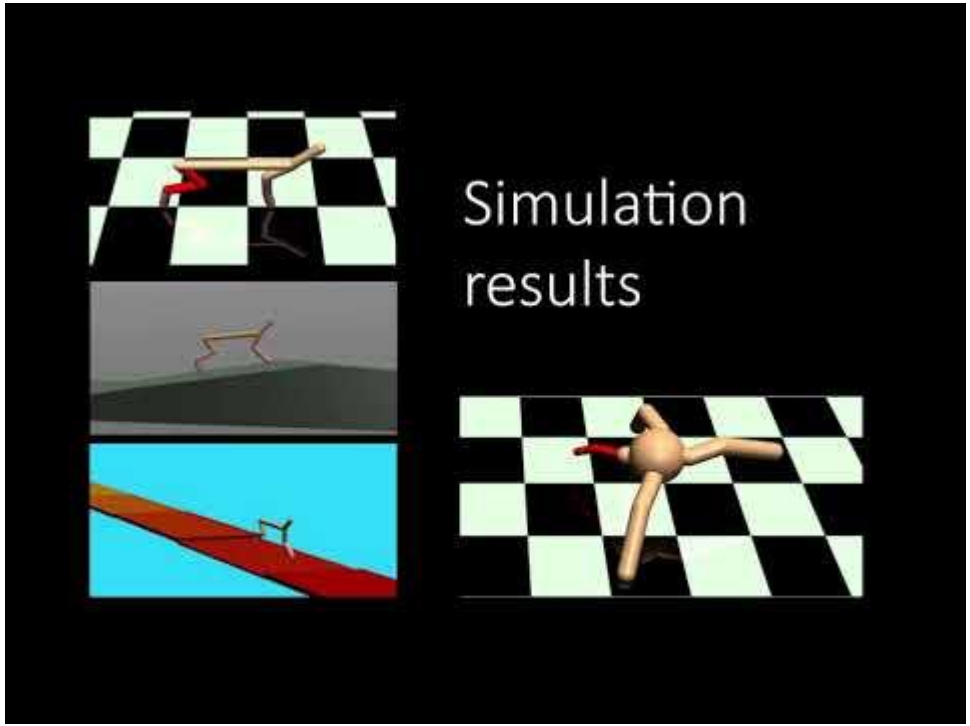
Adapting the planner to new rewards



- Lifelong learning setup with observed or unobserved changes in the reward
- Changes in reward ➔ more planning, because the policy prior is less reliable

Lu, Mordatch, & Abbeel (2019). Adaptive Online Planning for Continual Lifelong Learning. NeurIPS Deep RL Workshop.

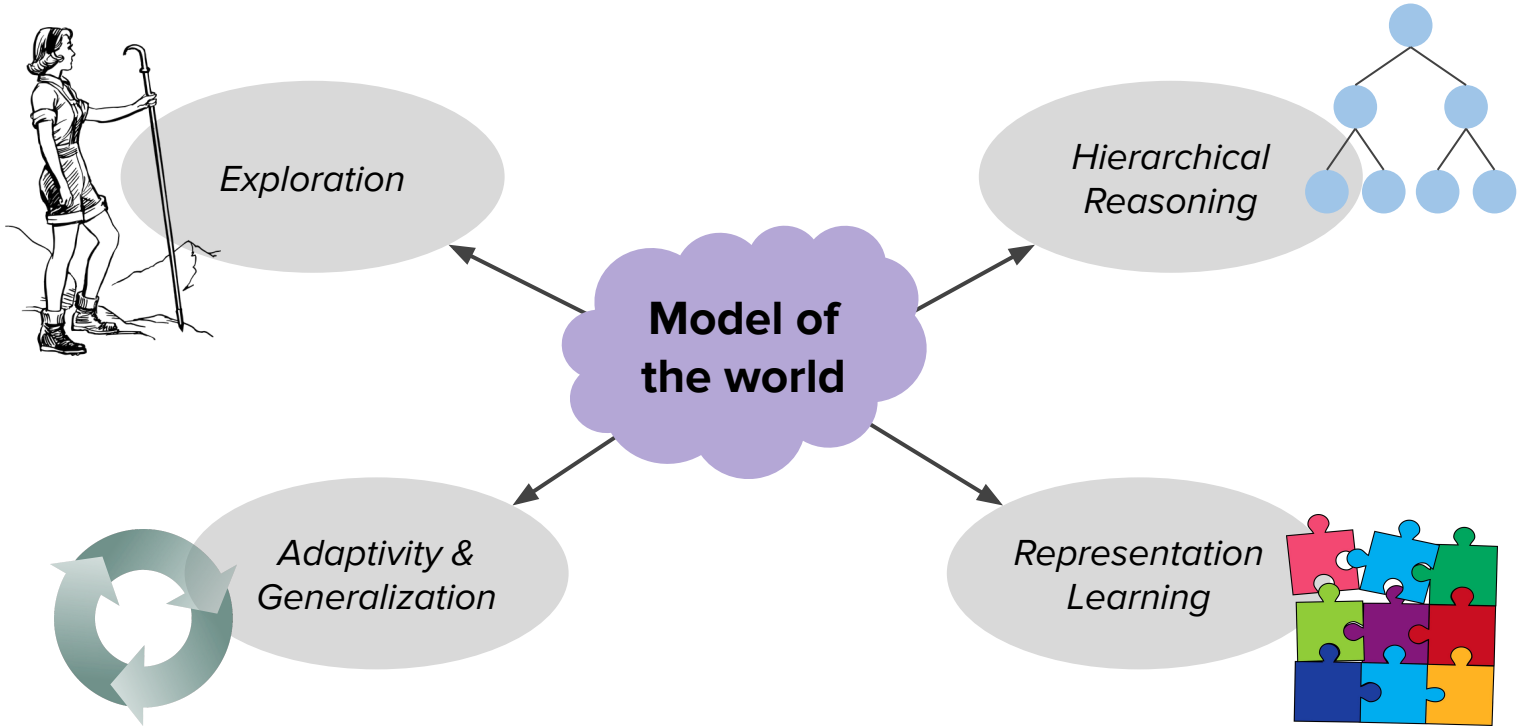
Adapting the model to new dynamics



- Train the model with a meta-learning objective
- At test time, adapt the model to new dynamics (change in terrain, body)

Nagabandi et al. (2019). Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning. ICLR.

What else are models good for?



Outline

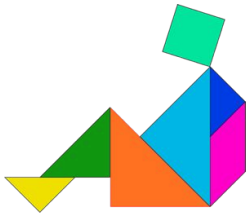
1. Introduction and motivation
2. Problem statement
3. What is a “model”?
4. What is model-based control?
5. Model-based control in the loop
6. What else can models be used for?
7. **What’s missing from model-based methods?**
8. Conclusion

Humans are the ultimate model-based reasoners!

- **Motor control:** forward kinematics models in the cerebellum
- **Language comprehension:** models of what is communicated
- **Pragmatics:** models of listener & speaker beliefs
- **Theory of mind:** models of other agents' beliefs and behavior
- **Decision making:** model-based reinforcement learning
- **Intuitive physics**: forward models of physical dynamics
- **Scientific reasoning:** mental models of scientific phenomena
- **Creativity:** being able to imagine novel combinations of things
- … and much more!

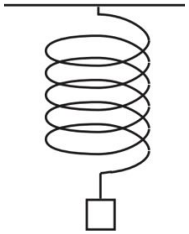
Markman, Klein, & Suhr (2008). Handbook of Imagination and Mental Simulation.
Abraham (2020). The Cambridge Handbook of the Imagination.

Compositionality

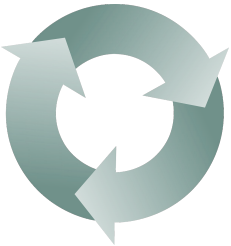


Causality

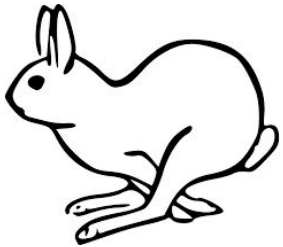
Incompleteness



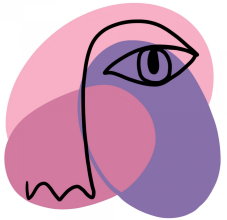
Adaptivity



Efficiency



Abstraction



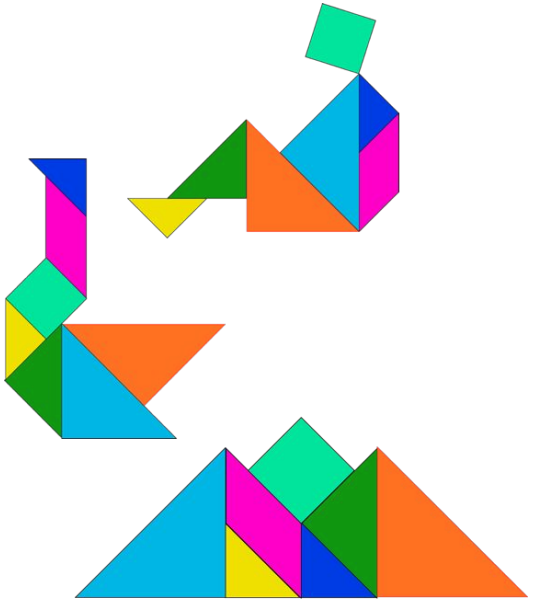
Faster planning

Higher tolerance to model error

Scalability to harder problems

More robust real-world applications & better commonsense reasoning!

Compositionality



- Humans understand the the world in terms of objects, events, & relations
- We can break down the world into these pieces & then manipulate those pieces in our imaginations.

Spelke & Kinzler (2007). Core knowledge. Developmental science, 10(1).

Dubey, Agrawal, Pathak, Griffiths, & Efros (2018). Investigating human priors for playing video games. ICML 2018.

Finke & Slayton (1988). Explorations of creative visual synthesis in mental imagery. Memory & Cognition, 16(3).

Causality

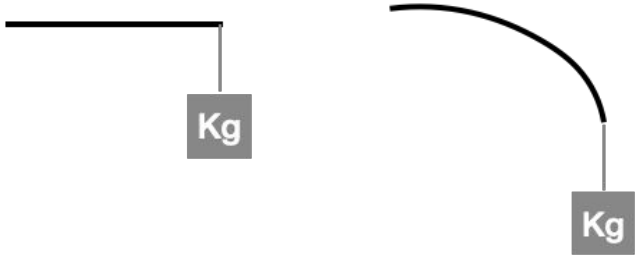
- Our models of the world are not just compositional, they are also *causal*.
- We know how intervening on some variables might (or might not) affect other variables.
- This allows us to reason about **things that have never occurred (and never will!)**

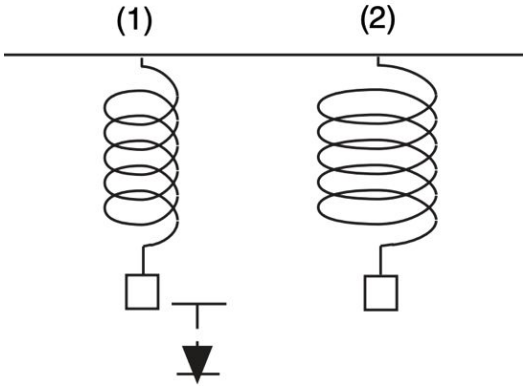
Griffiths & Tenenbaum (2009). Theory-based causal induction. *Psychological Review*, *116*(4).

This paper explores the physics of the what-if question **“what if the entire Earth was instantaneously replaced with an equal volume of closely packed, but uncompressed blueberries?”** While the assumption may be absurd, the consequences can be explored rigorously using elementary physics. The result is not entirely dissimilar to a small ocean-world exoplanet.

Sandberg (2018). Blueberry Earth. arXiv.

Incompleteness

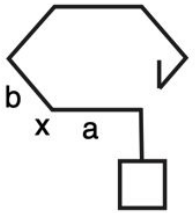




Which spring will stretch further?

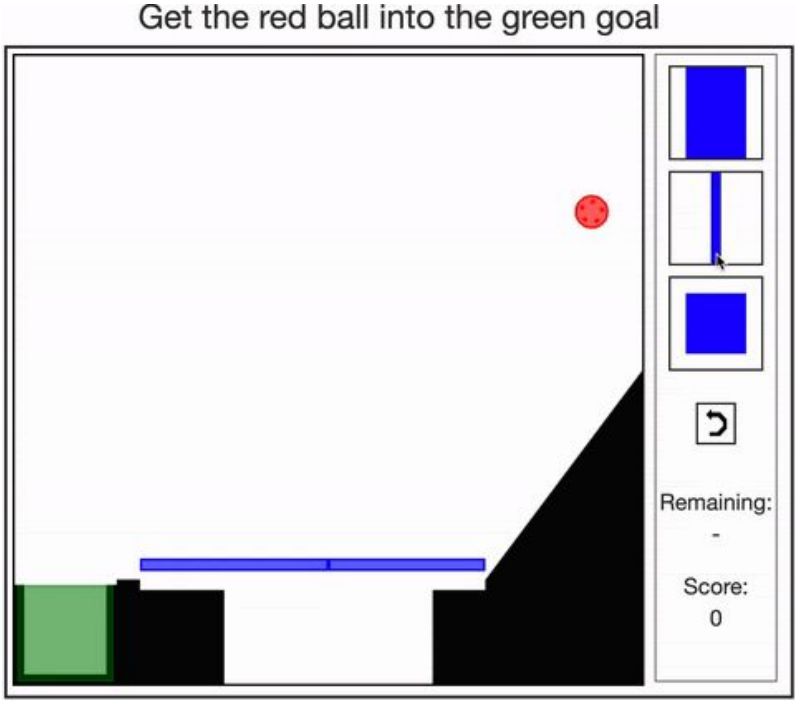
"But then it occurs to me that there's **something clearly wrong with that [bending rod] metaphor**, because… you get a spring which stretches more and more at the bottom… But that is not the case…they're uniform all the way around."

"Aha!! **Maybe the behavior of the spring has something to do with twist** (*makes twisting motion with right hand*) forces as well as bend forces. That's a real interesting idea."



Clement (2009). The role of imagistic simulation in scientific thought experiments. Topics in Cognitive Science, 1(4), 686-710.

Adaptivity

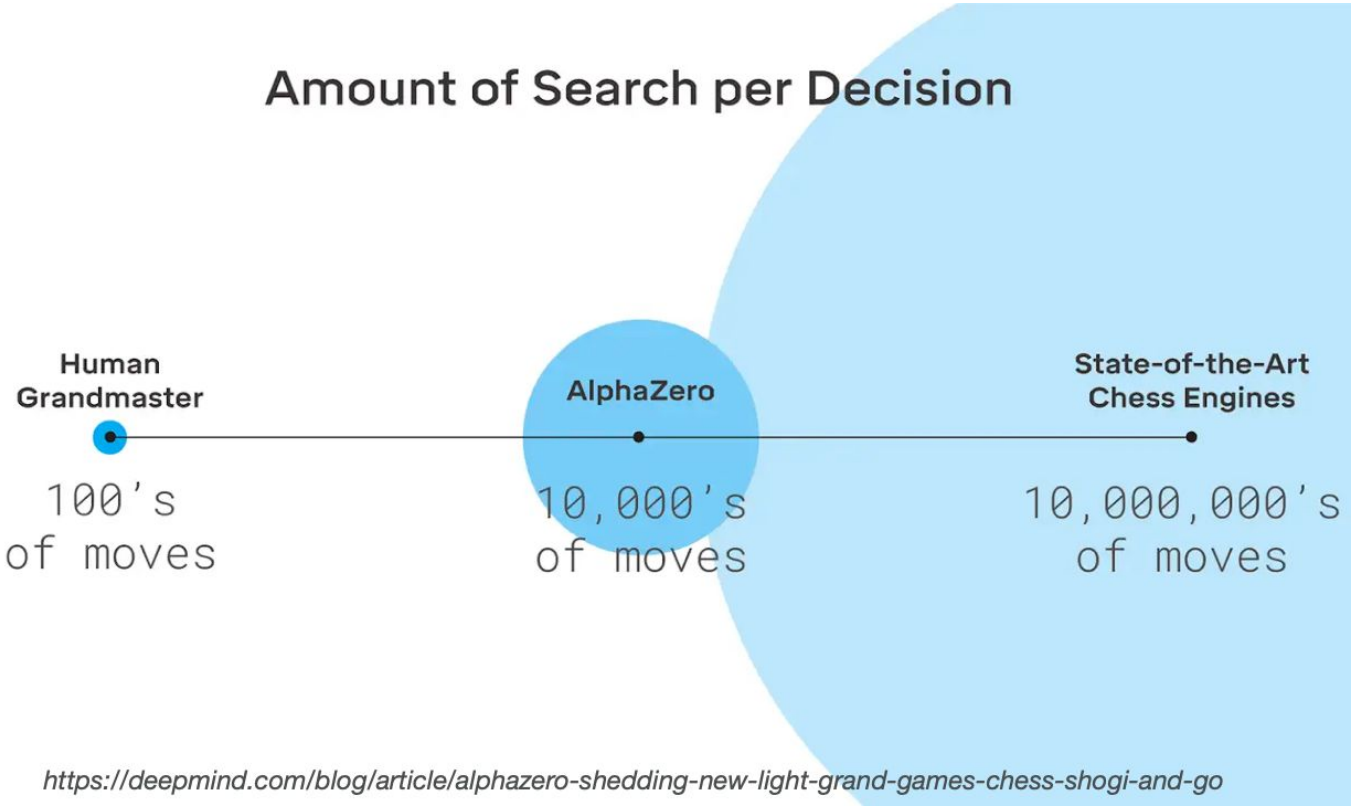


- We can rapidly assemble our compositional knowledge into on-the-fly models that are adapted to the current task
- Then we quickly solve these models, leveraging both mental simulation & (carefully chosen) real experience

Allen, Smith, & Tenenbaum (2019). The tools challenge: Rapid trial-and-error learning in physical problem solving. CogSci 2019.

Dasgupta, Smith, Schulz, Tenenbaum, & Gershman (2018). Learning to act by integrating mental simulations and physical experiments. CogSci 2018.

Efficiency



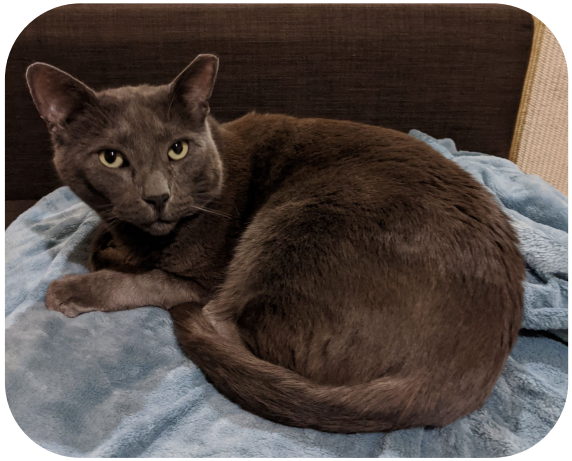
https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go

Abstraction

- State abstraction:
  - By immediate area (e.g. table vs. floor)
  - By room (e.g. kitchen vs. bedroom)
  - By building (e.g. home vs. office)
  - Relations: in, on, at, in between, next to, etc.

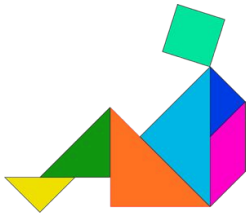
- Temporal abstraction:
  - Seconds, Minutes, Hours, Days, Weeks
  - Relations: before, after, during, begin, end

- Feeding the cat:
  - Go home from the office *(minutes)*
  - Go into the kitchen *(seconds)*
  - Move my arm to the drawer *(seconds)*
  - Grasp the drawer with my hand *(seconds)*
  - Pull out a can of food with my other hand *(seconds)*
  - Tear off the cover with my fingers *(seconds)*



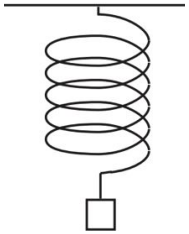
Kumulus

Compositionality

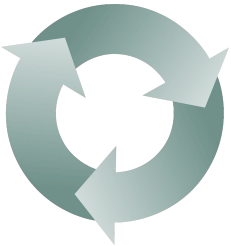


Causality

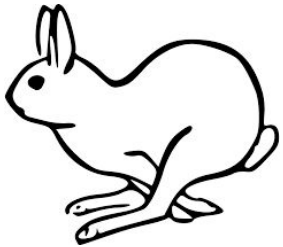
Incompleteness



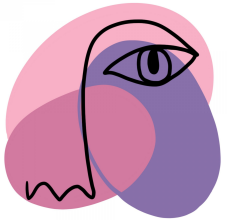
Adaptivity



Efficiency



Abstraction



Faster planning

Higher tolerance to model error

Scalability to harder problems

More robust real-world applications & better commonsense reasoning!

Outline

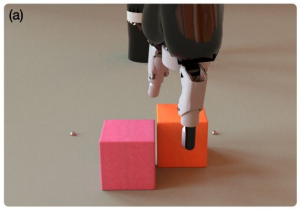
1. Introduction and motivation
2. Problem statement
3. What is a “model”?
4. What is model-based control?
5. Model-based control in the loop
6. What else can models be used for?
7. What’s missing from model-based methods?
8. **Conclusion**

Dimensions of learned models

States vs. **Observations** vs. **Latent States**

$$s = [x, \dot{x}, \phi]$$

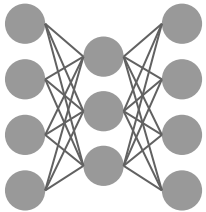
$$o =$$



$$z \in \mathbb{R}^{d_z}$$

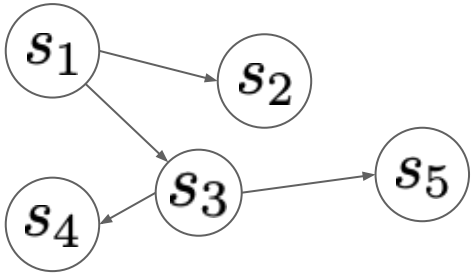
---

s_t
a_t



s_{t+1}

Parametric vs. **Non-Parametric**

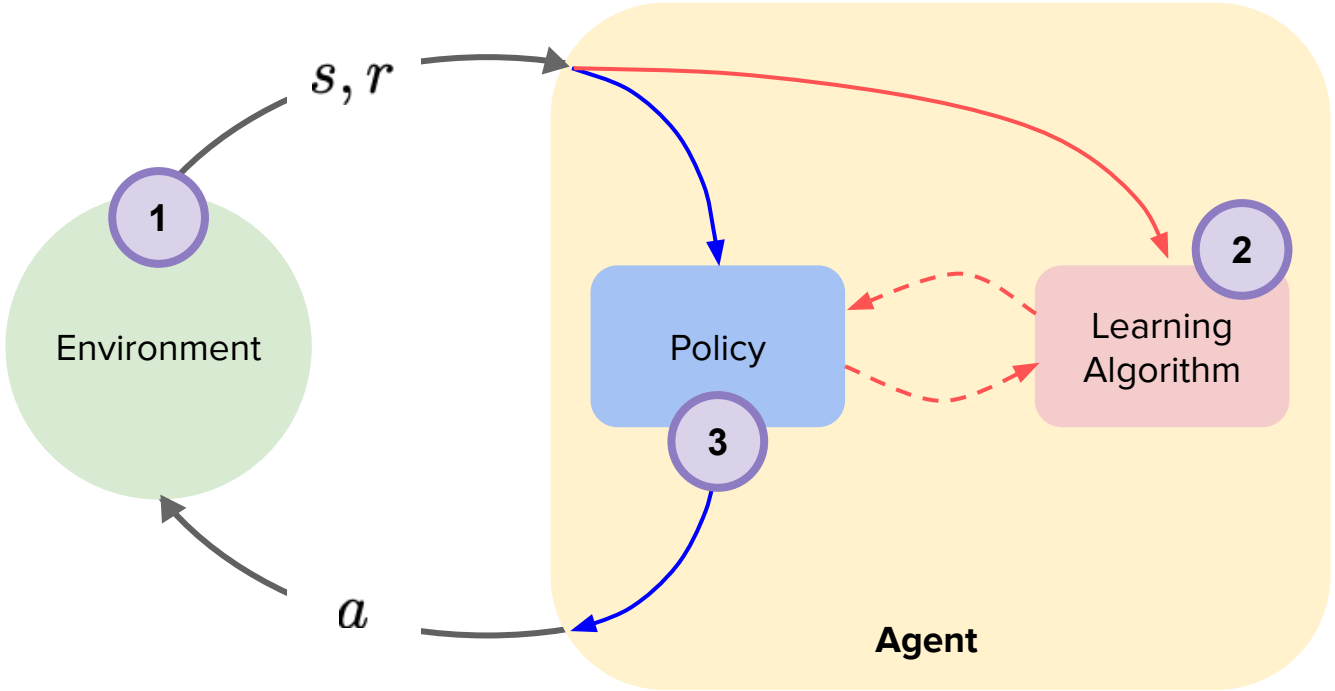


Where does the model fit into the picture?

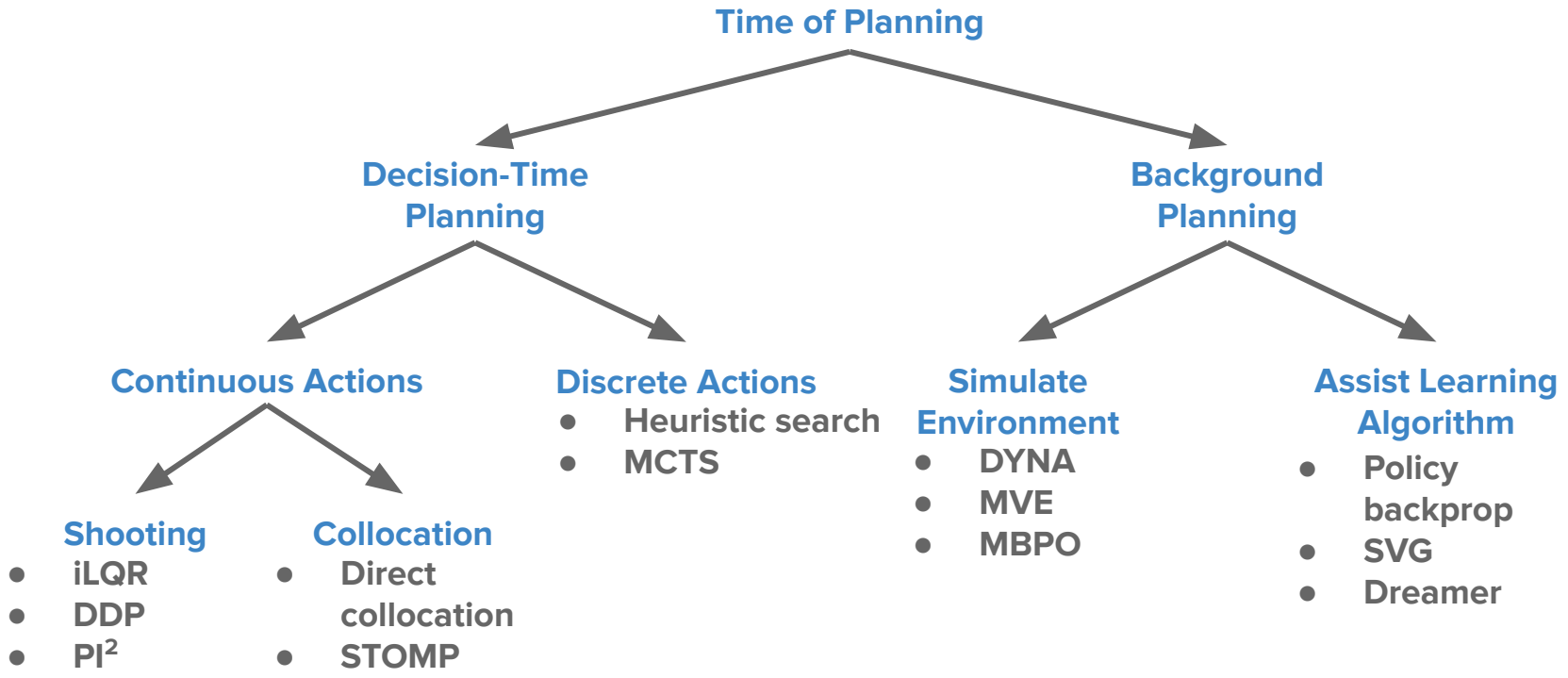
1. Simulating the environment

2. Assisting the learning algorithm

3. Strengthening the policy



Landscape of model-based methods



Summary of practical considerations

Gathering data to train models

- Fixed off-line datasets
- Data augmentation
- Conservative model or policy updates

Acting under imperfect models

- Replan via model-predictive control
- Plan close to familiar states

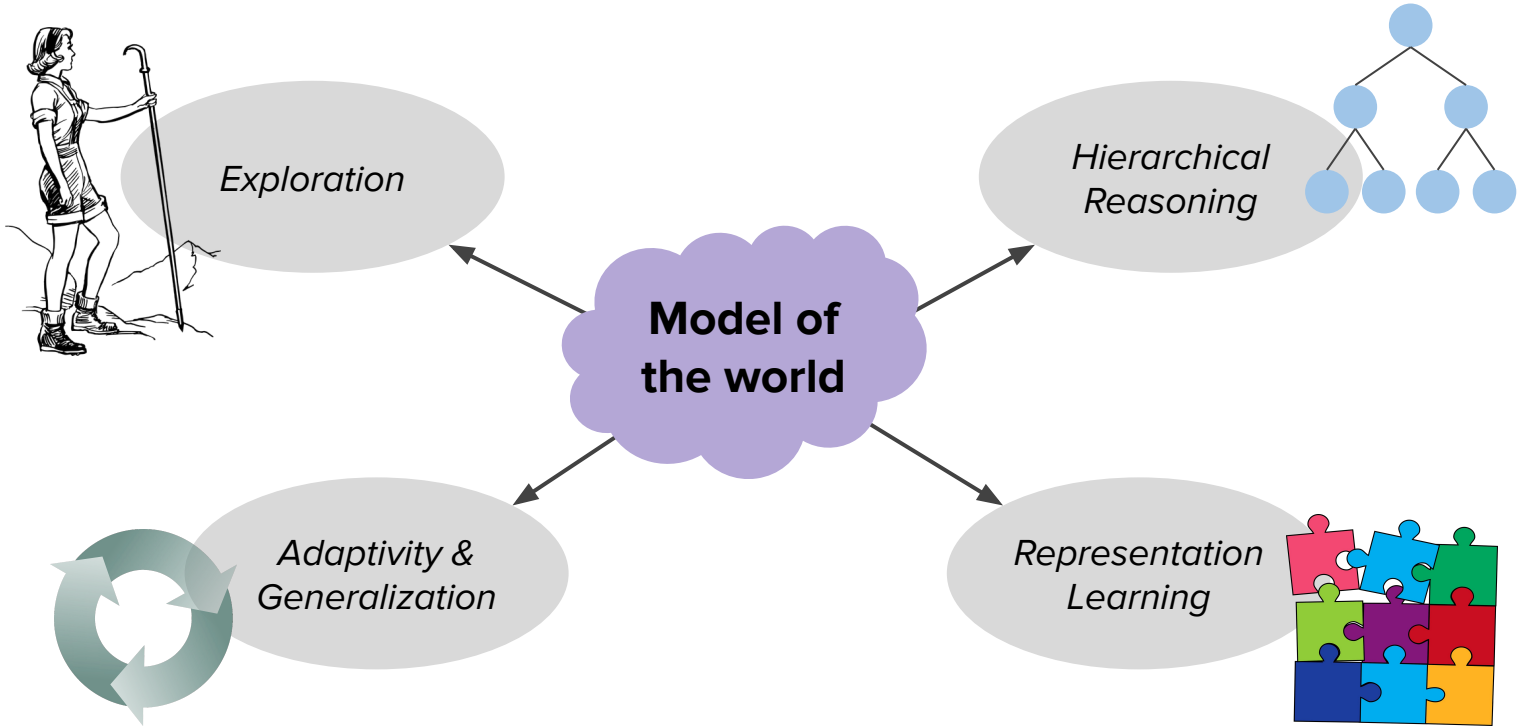
Estimating model uncertainty

- Gaussian processes
- Bayesian neural networks
- Pseudo-counts
- Ensembles

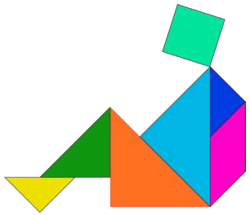
Combining planning and learning

- Distillation
- Terminal value functions
- Planning as policy improvement
- Implicit planning

What else are models good for?

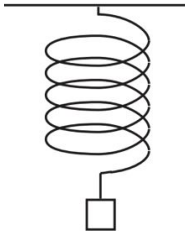


Compositionality

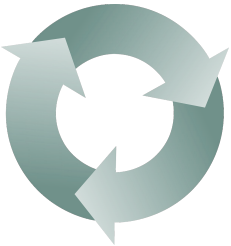


Causality

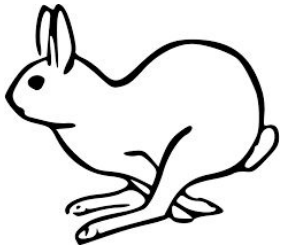
Incompleteness



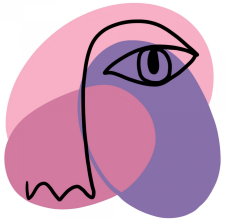
Adaptivity



Efficiency



Abstraction



Faster planning

Higher tolerance to model error

Scalability to harder problems

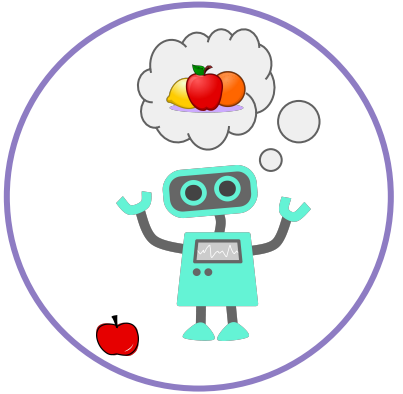
More robust real-world applications & better commonsense reasoning!

Robotics

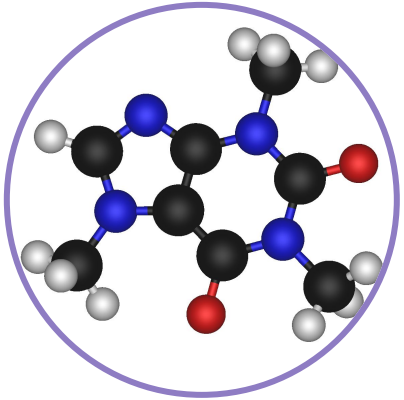
Human/AI Interaction

Operations & Logistics

Commonsense Reasoning



Games



Physical Sciences

Climate & Energy

Ethical and broader impacts

✘ Improper use of automated decision making (e.g., discriminatory pricing, hiring, etc.)

✘ Problematic components lead to bad decisions (e.g., discriminatory vision models)

→ Think critically about the choice of model, planner, and ways in which the overall system could be used ←

Timnit Gebru & Emily Denton's CVPR Tutorial on Fairness, Accountability, Transparency, and Ethics in Computer Vision:
https://sites.google.com/corp/view/fatecv-tutorial

✔ Improved data efficiency → decreased energy & computational cost

✔ Potentially more interpretable decision-making (e.g., can inspect the model to generate explanations)

→ Many opportunities to develop MBRL methods with multiple objectives in mind, beyond outperforming model-free RL ←

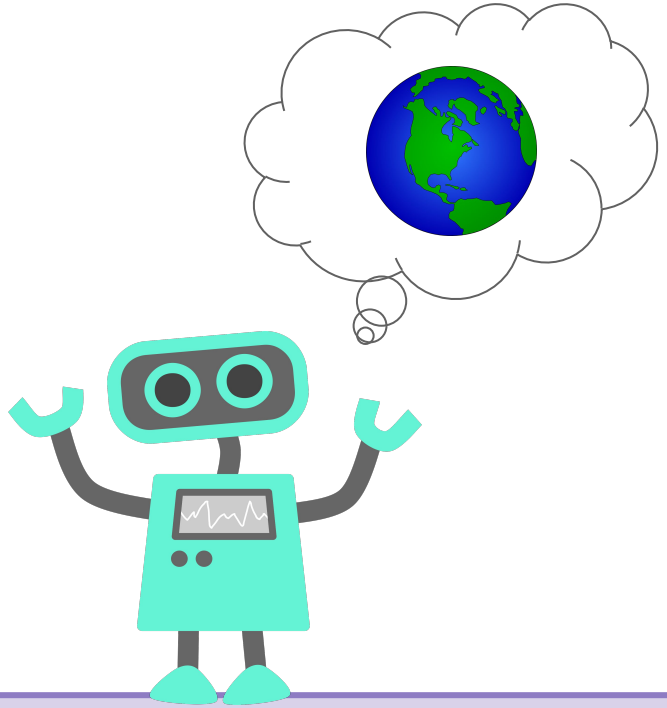
XXAI: Extending Explainable AI Beyond Deep Models and Classifiers
http://interpretable-ml.org/icml2020workshop/

Participatory Approaches to Machine Learning
https://participatoryml.github.io/

Thanks for listening!

Thanks to:

- Aravind Rajeswaran
- Vikash Kumar
- Theophane Weber
- Lars Buesing
- Tobias Pfaff
- Sarah Seiter



For references, slides, resources, or to contact us with questions or feedback:
http://sites.google.com/view/mbrl-tutorial