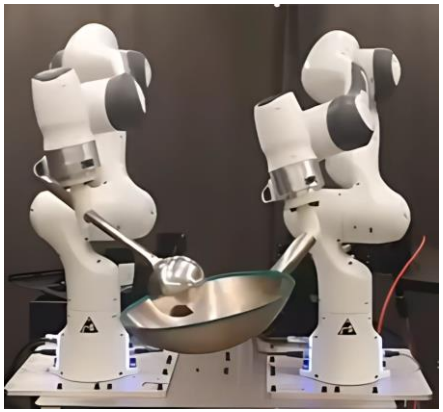
ECE7121 Learning-based control – 2025 Fall

# Model-based RL


INHA UNIVERSITY

# What's wrong with known dynamics?




# Can we learn a simulator?

- Simulator : predictive model of $s_{t+1}$ given $s_t, a_t$
  = transition dynamics $f(s_t, a_t) = s_{t+1}$

- How to apply RL algorithm in the real-world?
  - collect data $D = \{(s, a, s')_i\}$ using some base policy
  - $f(s, a)$ to minimize $\sum_i \| f(s_i, a_i) - s'_i \|^2$
  - run your favorite RL method inside simulator $f$

- Issues
  - Data coverage matters a lot
  - How to learn a simulator (model) efficiently?
  - Need to account for model inaccuracies

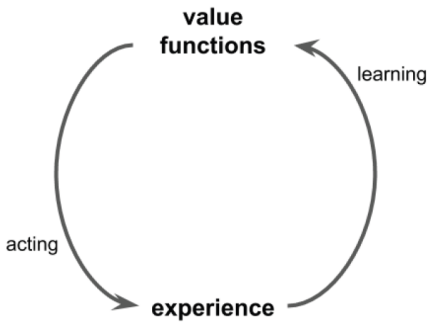
# Can we learn a simulator?

- Simulators in brief
  - you have it already
    - games (e.g., go, chess)
    - derived from the first principle (e.g., rigid body dynamics)
    - simulated environments (robot simulators)
  - you approximately know most of it (e.g. certain physical models)
    - can fit the unknown parameters using the data – system identification
  - you don’t know it (most scenarios in practice)
    - learn it end-to-end
    - learn a (low-dim) state representation, then learn model over representations

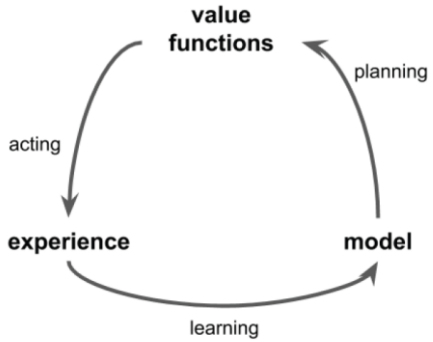
# Can we exploit a simulator?

- If we knew $f(s_t, a_t) = s_{t+1}$, we could use the tools from last week

- Model-based RL v.0.5
  - run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
  - learn dynamics model $f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s_i'\|^2$
  - plan through $f(s, a)$ to choose actions




model-free RL

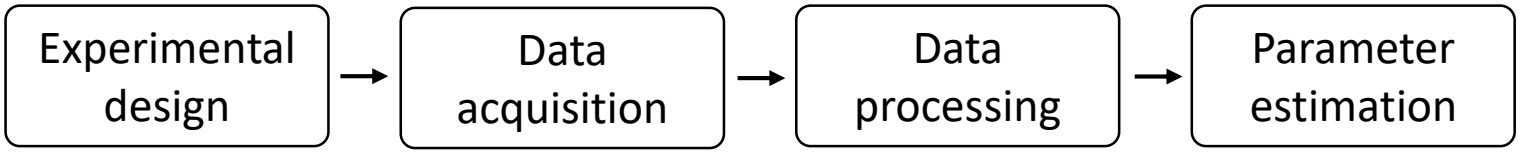



model-based RL

# Can we exploit a simulator?

- Model-based RL v.0.5, does it work well?
  - essentially how system identification works in classical robotics
  - some care should be taken to design a good base policy
  - particularly effective if we can hand-engineer a dynamics model using the domain knowledge, and fist just a few parameters

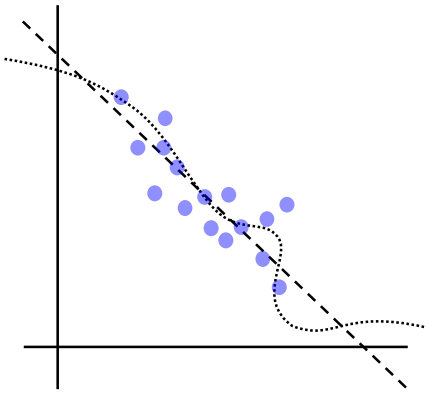
- System identification procedure



# Can we exploit a simulator?

- Model-based RL v.0.5, does it work well?
  - not for the deep RL approaches
  - distribution mismatch problem becomes exacerbated as we use more expressive model classes

Linear regression $y = ax + b$



Collected data

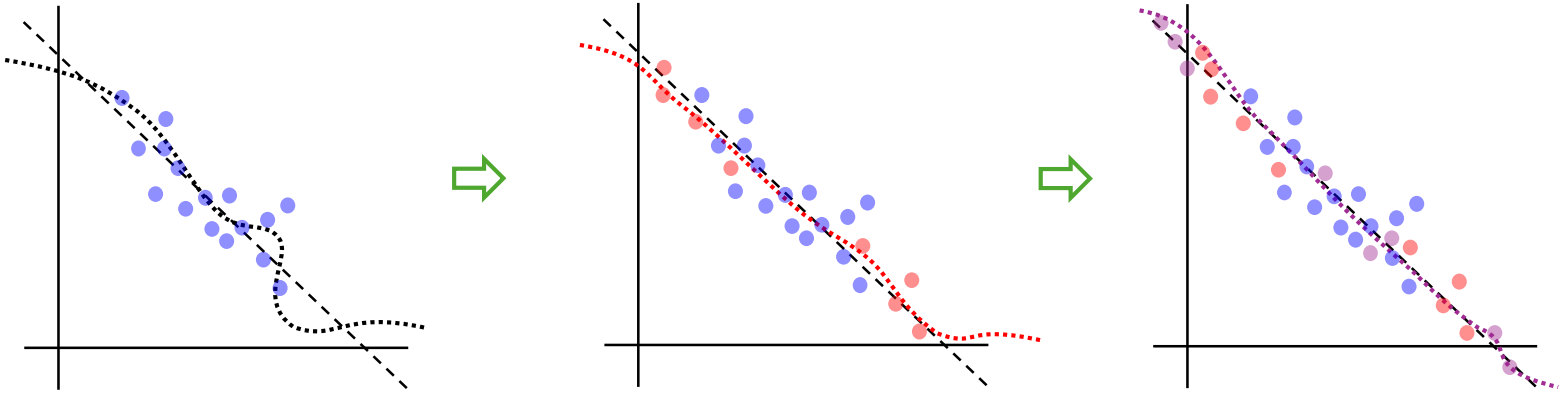
Model-based estimate

NN estimate

# Model-based RL

- Model-based RL v.1.0
  - run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
  - learn dynamics model $f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s_i'\|^2$
  - plan through $f(s, a)$ to choose actions
  - execute those actions and add the resulting data $\{(s, a, s')_i\}$ to $\mathcal{D}$

Linear regression $y = ax + b$



Collected data

Newly collected data

Model-based estimate

NN estimate

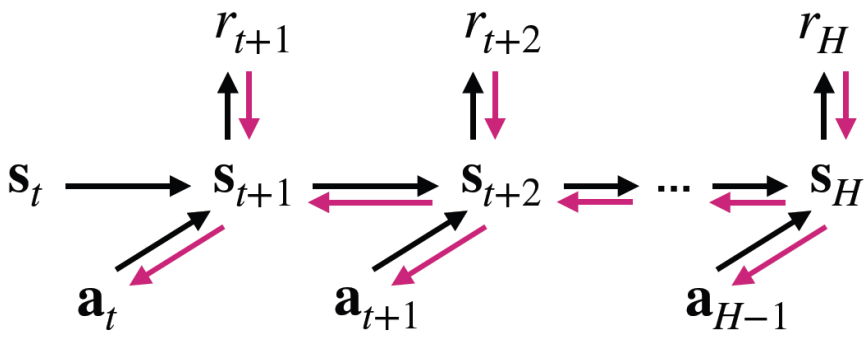
# Model-based RL

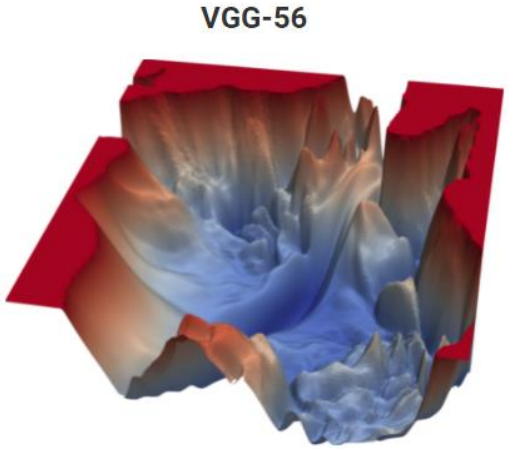
- Model-based RL v.1.1
  - run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
  - learn dynamics model $f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s'_i\|^2$
  - plan through $f(s, a)$ to choose actions
  - execute the first action and add the resulting data $\{(s, a, s')_i\}$ to $\mathcal{D}$

- We use MPC framework for planning
  - Replanning helps with model errors
  - The more you replan, the less perfect each individual plan needs to be
  - but computation become intensive (appropriate for short horizons)

# Planning with the learned model

- Plan through $f(s, a)$ to choose actions (= trajectory optimization)
- 1) gradient-based optimization



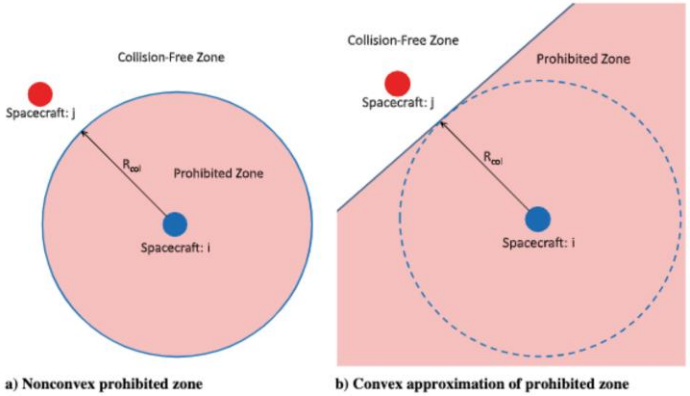

- It is a highly non-convex problem
- backpropagation uses the chain rule to compute gradients
- iterative gradient descent (basic, not efficient)
- only possible to differential simulators

# Planning with the learned model

> 1) gradient-based optimization
  - iterative convexification and optimization (SCP/SQP)

  - SQP Algorithm
    - Initialize a trajectory
    - affinized the dynamics
    - quadratic approximation of the cost
    - affinized the constraint
    - solve the QP

until convergence



a) Nonconvex prohibited zone

b) Convex approximation of prohibited zone

  - SCP uses arbitrary convex approximations (more general)
  - SCP and SQP are heuristic
    - it can fail to find an optimal (feasible) solution
    - highly dependent on initialization

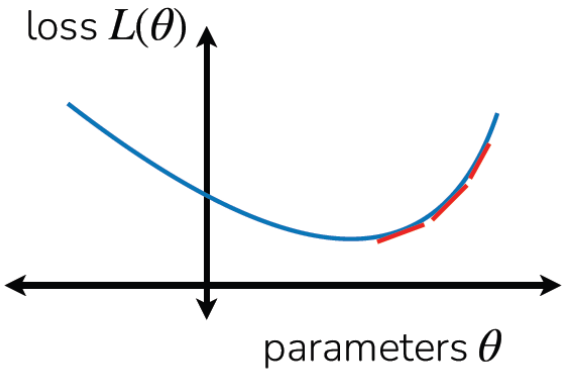
# Planning with the learned model

- 1) gradient-based optimization
  - iterative linear quadratic regulator (iLQR)
    - linearize the dynamics and makes a quadratic approximation of cost
    - solve using DP (Riccati equation)
    - don't directly deal with constraints (use penalty methods)
  - Differential dynamic programming (DDP)
    - use a full second-order method (need a second-order approximation of dyn.)
  - Both can be viewed as (approximated) Newton's method
  - Compare to QP, complexity increases linearly as horizon increases

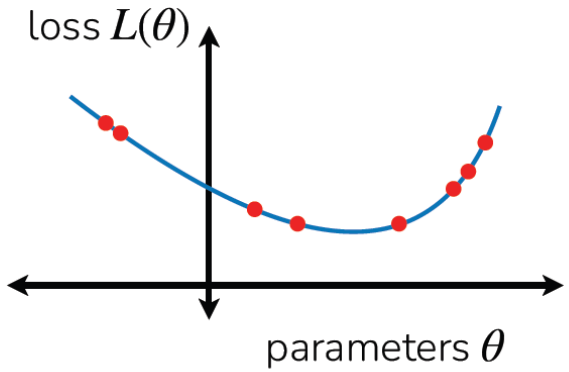
# Planning with the learned model

> 2) sampling-based optimization

Gradient-based (1st order)



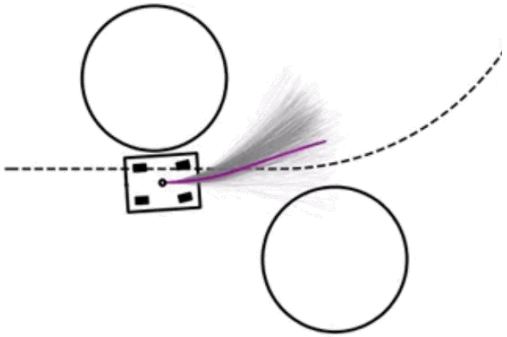

Sampling-based (0th order)




- requires no gradient information (gradient-free optimization)
- use randomness to explore the space
- often better at escaping local minima than gradient-based methods
- parallelizable
- scales poorly to high dimensions

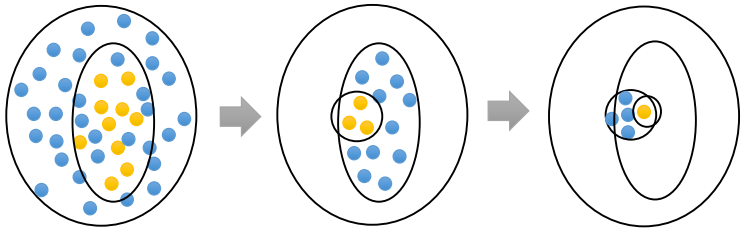
# Planning with the learned model

- 2) sampling-based optimization
  - Random shooting
    - Guess and check
    - sample many action sequences ($A = \{a_t, \dots, a_{t+H}\}$) $A_1, \dots, A_n$ from some distribution (e.g., uniform)
    - choose the best $A_i$ (max return)



    - can we improve the sampling distribution? (prior knowledge)
  - Cross-entropy method
    - sample many action sequences $A_1, \dots, A_n$ from $p(A)$
    - evaluate and pick elites $A_{i_1}, \dots, A_{i_m}$
    - refit $p(A)$ to the elites $A_{i_1}, \dots, A_{i_m}$

until convergence

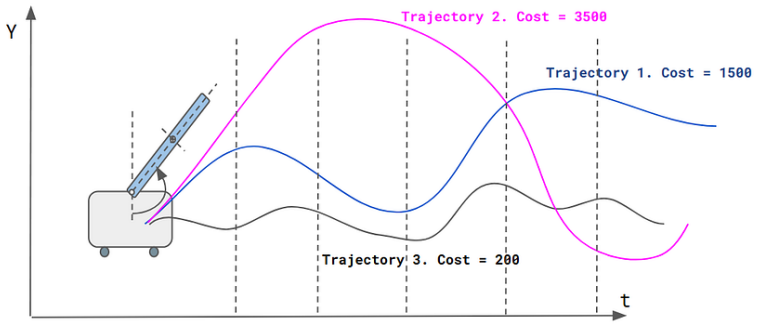


# Planning with the learned model

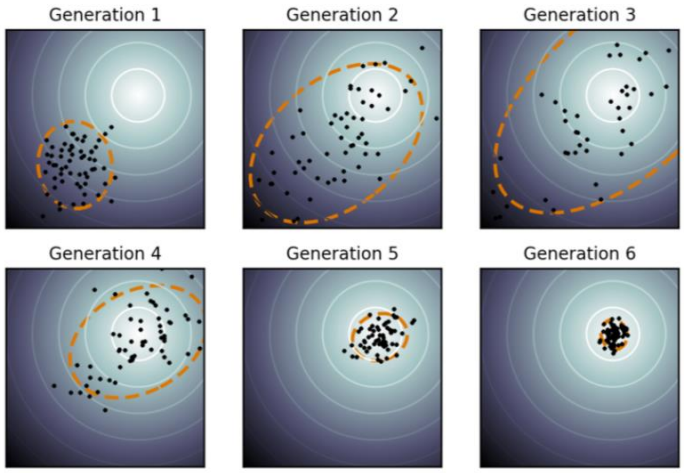
- 2) sampling-based optimization
  - model predictive path integral control (MPPI)
    - sample many action sequences $A_1, \dots, A_n$ from $p(A) \sim \mathcal{N}(\mu, \Sigma)$
    - evaluate corresponding costs $C_1, \dots, C_n$
    - compute weights based on costs: $w_i = \frac{\exp\left(-\frac{1}{\lambda} C_i\right)}{\sum_j \exp\left(-\frac{1}{\lambda} C_j\right)}$
    - update $\mu \leftarrow \sum_i w_i A_i$
  - covariance matrix adaptation evolutionary strategy (CMA-ES)
    - sample many action sequences $A_1, \dots, A_n$ from $p(A) \sim \mathcal{N}(\mu, \Sigma)$
    - evaluate corresponding costs $C_1, \dots, C_n$
    - compute weights based on costs: $w_i$ (e.g., exponential, top-K, ...)
    - update $\mu \leftarrow (1 - \gamma_\mu)\mu + \gamma_\mu \sum_i w_i A_i, \quad \Sigma \leftarrow (1 - \gamma_\Sigma)\Sigma + \gamma_\Sigma \sum_i (A_i - \mu)(A_i - \mu)^\top$
  - Even more general: not necessarily Gaussian, parametrized distribution

# Planning with the learned model

- 2) sampling-based optimization
  - model predictive path integral control (MPPI)

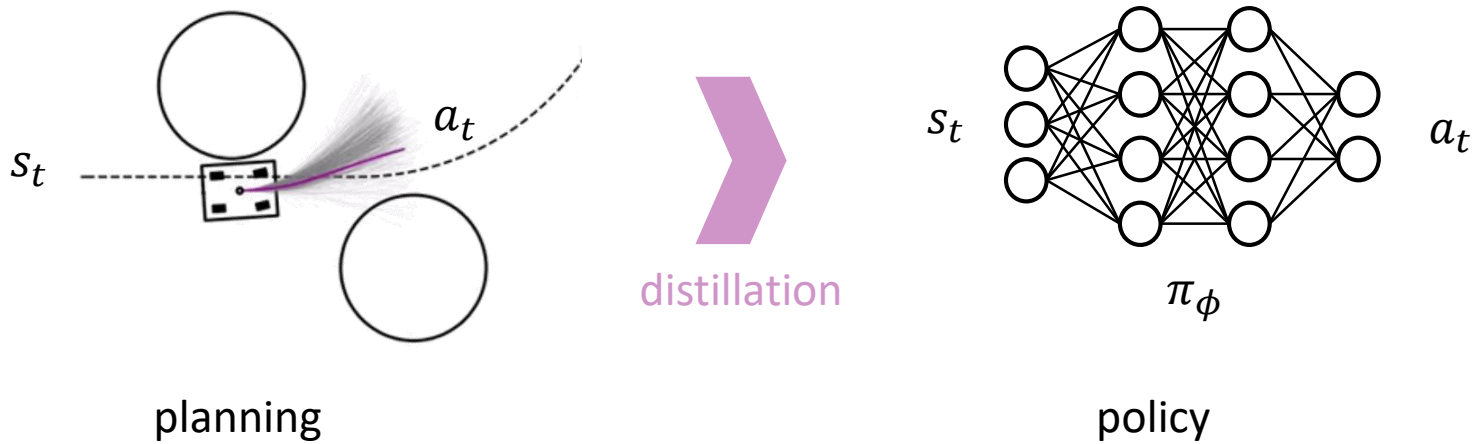



  - covariance matrix adaptation evolutionary strategy (CMA-ES)



# Planning with the learned model

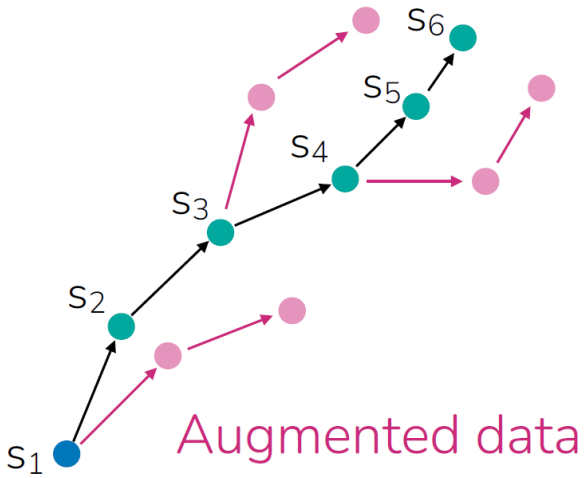
- Planning at test time can be computationally intensive
- Option 1: distill planner's actions into a policy
  - still limited to short-horizon problems



# Model-based policy optimization

> Option 2: augment model-free RL methods with data from model
  - generate full trajectories from initial states?
    - model may not be accurate for long horizons
  - generate partial trajectories from initial states?
    - model may not get good coverage of later states

  - generate partial trajectories from all states in the data
    - add augmented dataset and update policy



# Model learning

- Model-based RL v.0.5
  - run **base policy** $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
  - learn **dynamics model** $f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s_i'\|^2$
  - plan through $f(s, a)$ to choose actions

- Issues
  - (1) Domain shift, over-fitting, and generalization
    - what policy we should use to generate data?
  - (2) good model might be bad for control
    - what function class we should use to learn?
  - (3) real-world models are not time-invariant
  - (4) high-dimensional state/action or rich observation settings
  - (5) under-fitting and compounding errors

# Regularized model learning (1), (2)

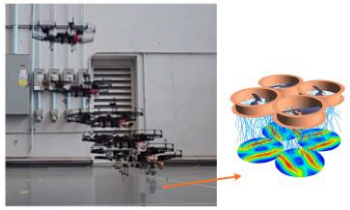
- Not only minimize the regression error, also design regularization
  - $f_\theta$ has some good properties for control (Lipschitz, smooth, controllable, …)
  - $f_\theta$ matches the physics (Lagrangian systems, symmetric, invariance …)

  - $\sum_i \|f_\theta(s_i, a_i) - s_i'\|^2 + R(\theta)$ or $\sum_i \|f_\theta(s_i, a_i) - s_i'\|^2,\ s.t\ \theta \in \Theta$

  - Examples
    - Residual dynamics learning use spectral normalization $\|f_\theta(x,u)\|_{Lip} \leq \gamma$

**Neural Lander: Stable Drone Landing Control Using Learned Dynamics**

Guanya Shi*,1, Xichen Shi*,1, Michael O'Connell*,1, Rose Yu2, Kamyar Azizzadenesheli3, Animashree Anandkumar1, Yisong Yue1, and Soon-Jo Chung1

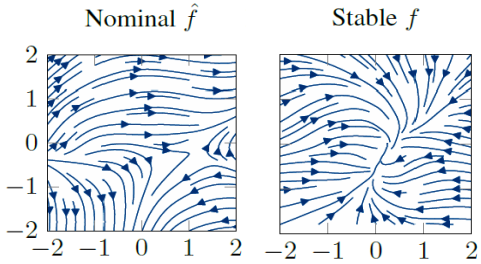




- Learning stabilizable nonlinear dynamics

Learning Stabilizable Nonlinear Dynamics with Contraction-Based Regularization

Sumeet Singh1, Spencer M. Richards1, Vikas Sindhwani2, Jean-Jacques E. Slotine3, and Marco Pavone1



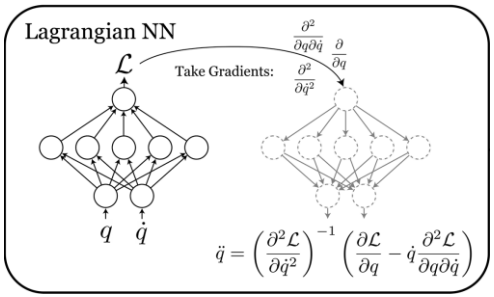
# Regularized model learning (1), (2)

> Examples

- deep Lagrangian networks

LAGRANGIAN NEURAL NETWORKS

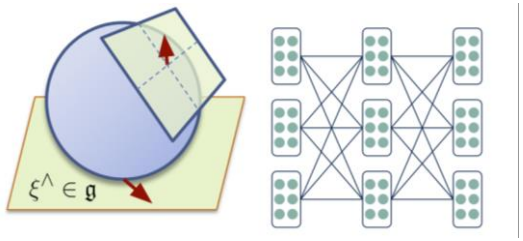

**Miles Cranmer** Princeton University mcranmer @princeton.edu
**Sam Greydanus** Oregon State University greydanus.17 @gmail.com
**Stephan Hoyer** Google Research shoyer @google.com
**Peter Battaglia** DeepMind peterbattaglia @google.com
**David Spergel*** Flatiron Institute davidspergel @flatironinstitute.org
**Shirley Ho†** Flatiron Institute shirleyho @flatironinstitute.org


- Invariance and equivariance (symmetry)

**Lie Neurons: Adjoint-Equivariant Neural Networks for Semisimple Lie Algebras**


Tzu-Yuan Lin [*1] Minghan Zhu [*1] Maani Ghaffari [1]

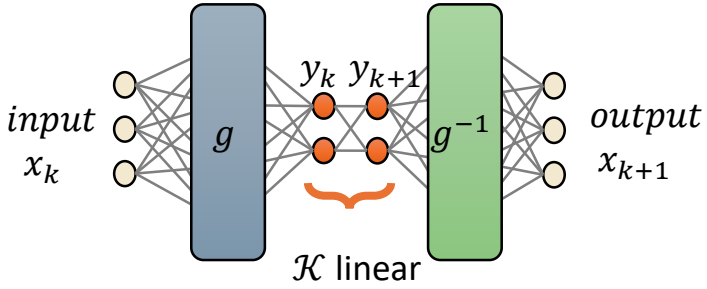




- Deep Koopman

K-mixup: Data augmentation for offline reinforcement learning using mixup in a Koopman invariant subspace


Junwoo Jang [a,1], Jungwoo Han [b,1], Jinwhan Kim [c,*]

[a] Department of Naval Architecture and Marine Engineering, University of Michigan, Ann Arbor, MI, 48109, USA
[b] Hyundai Motor Company, Seoul, 06182, South Korea
[c] Department of Mechanical Engineering, KAIST, Daejeon, 34141, South Korea

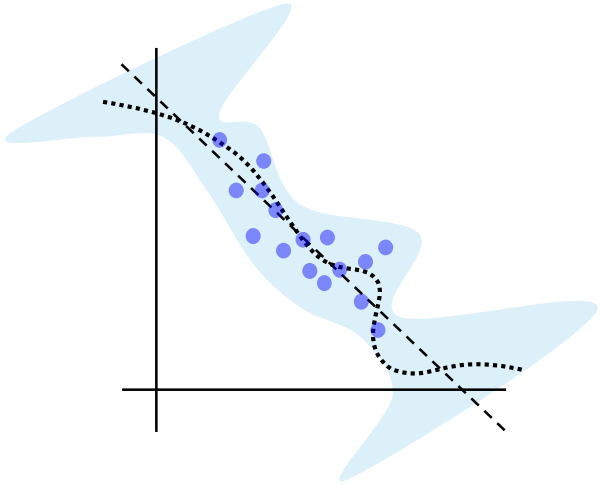



# Model learning with uncertainties (1), (2)

- Make $f_\theta$ as a probabilistic model to predict uncertainties
  - discourage overconfidence by jointly learning the uncertainties
  - use the uncertainties for the control (and data collection)

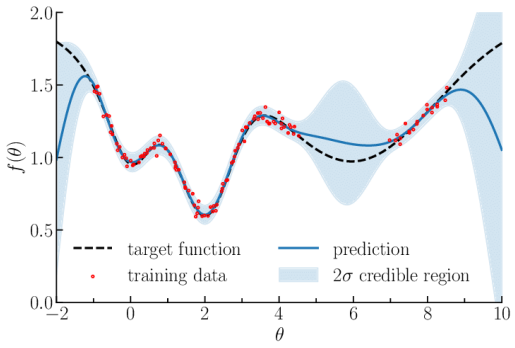
  - distribution mismatch

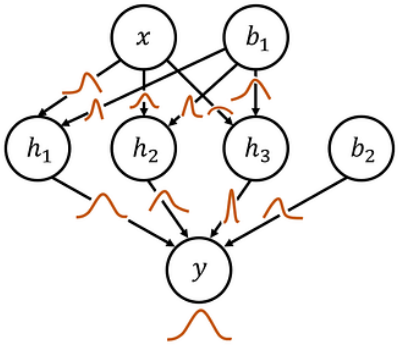
  - training time: explore the uncertain space
  - test time: exploit the certain space

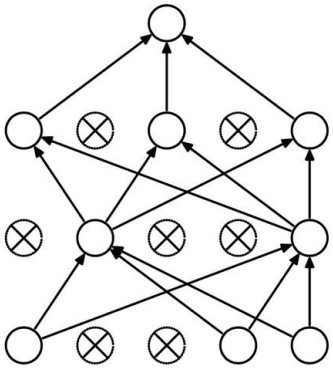


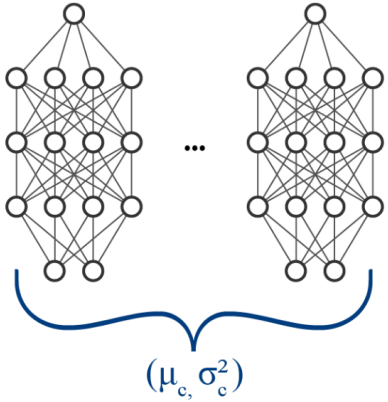
# Model learning with uncertainties (1), (2)

- How to obtain the uncertainties?
  - Gaussian process -> PILCO (2011)
  - Bayesian neural networks
  - Monte-Carlo dropout
  - Model ensemble -> PETS (2018), MBPO (2019)
  - Latent stochasticity -> Dreamer (2021)
  - Evidential dynamics



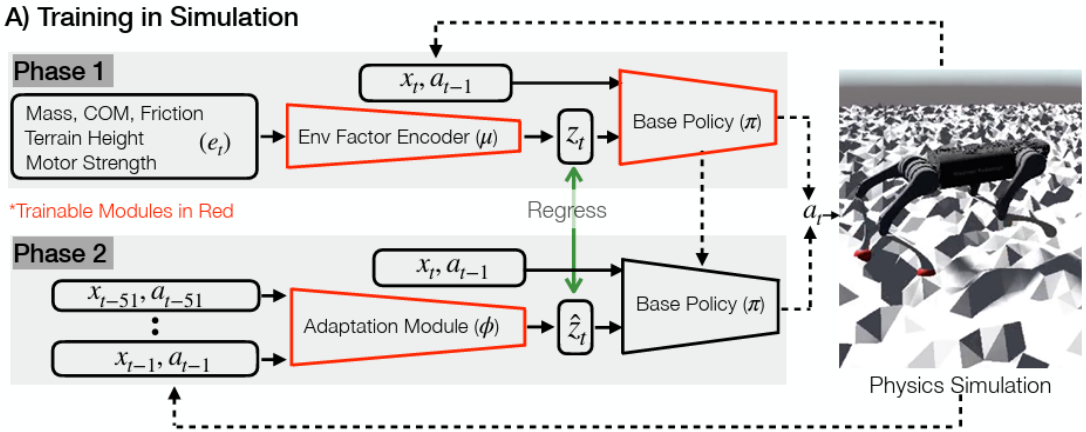



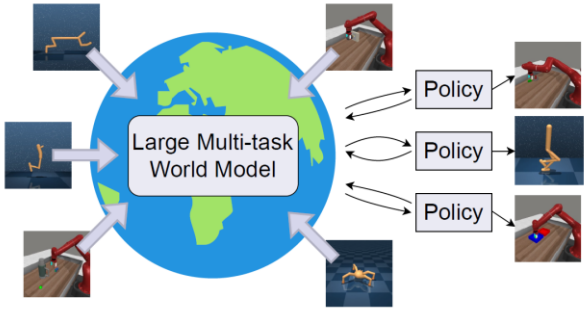


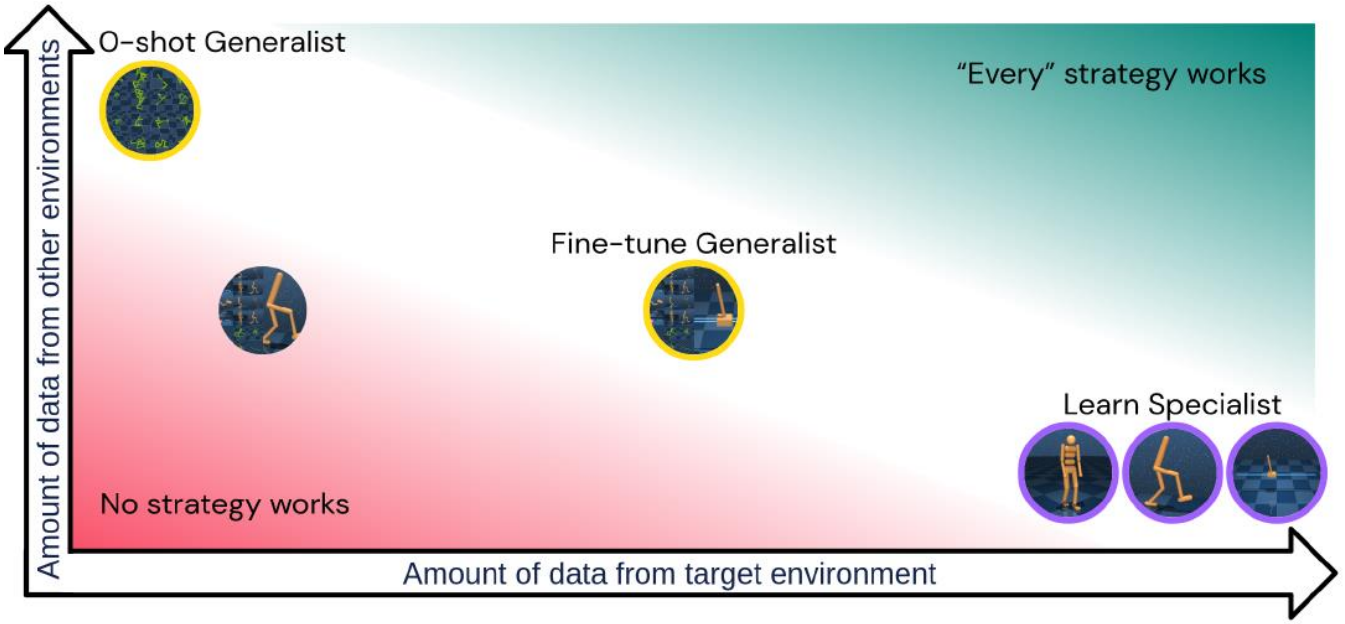
# Meta-learning dynamics, foundation model (3)

- Adapt model parameters in real time
  - meta-learning dynamics + real-time dynamics adaptation
  - meta-learning dynamics + adaptive control
  - policy conditioned on model parameters
- Learn a generalized model (foundation model)
- Examples
  - Learning to adapt in dynamic, real-world environments through meta-RL
  - Neural-fly enables rapid learning for agile flight in strong winds
  - RMA: rapid motor adaptation for legged robots
  - GNM: a general navigation model to drive any robot
  - Drive anywhere: Generalizable end-to-end autonomous driving with multi-modal foundation models
  - A generalist dynamics model for control

# Meta-learning dynamics, foundation model (3)

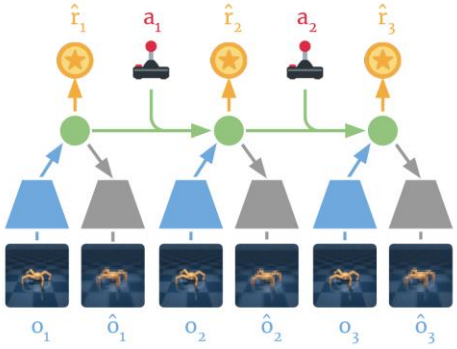


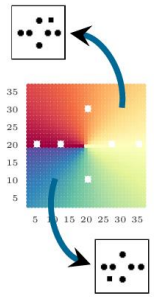



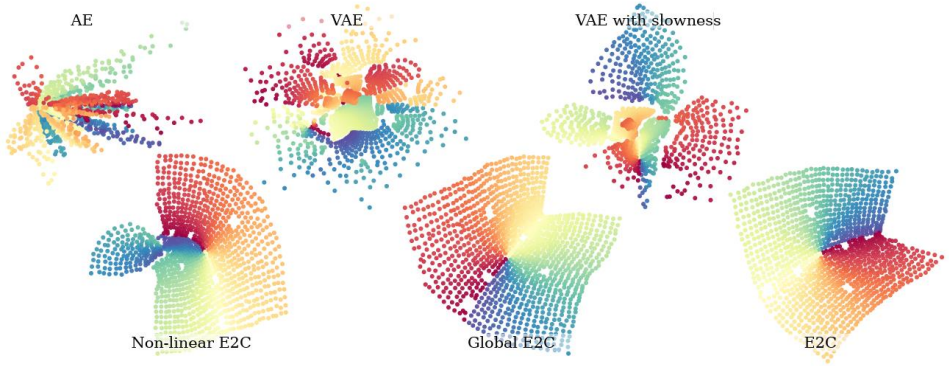



# Rich observations (4)

- learning world model from high-dimensional sensory input (e.g., images)
  - Intersection between CV and MBRL
  - Representation/CV techniques (e.g., contrastive learning, diffusion)



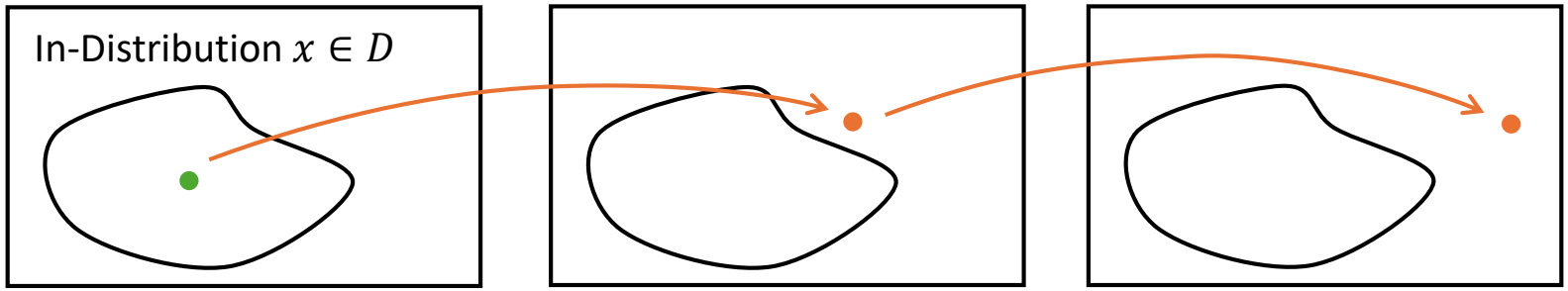

- Examples
  - Embed to control: a locally linear latent dynamics model for control from raw images
  - Planet: learning latent dynamics for planning from pixels
  - dreamer v1, v2, v3
  - TD-MPC 1, 2

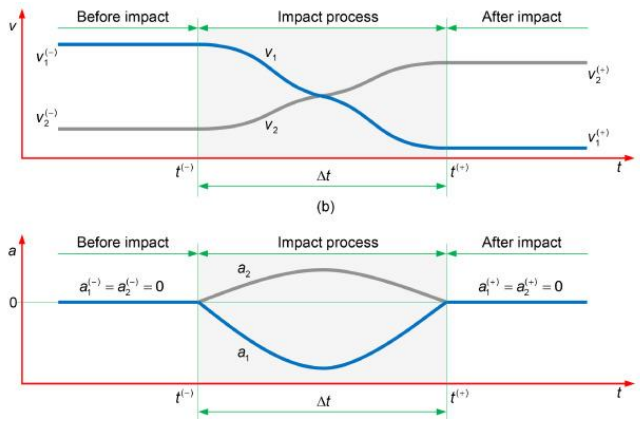
# Underfitting and compounding errors (5)

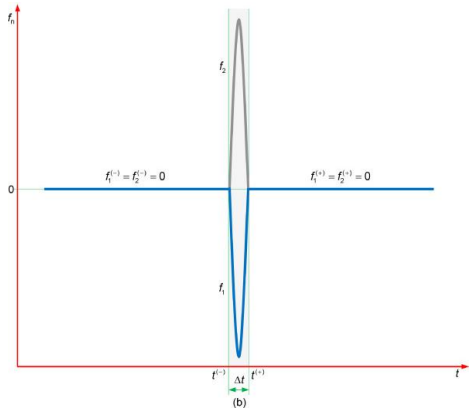
- Underfitting: hard to model complex/non-smooth dynamics (e.g., contacts)
- Compounding error: in multi-step prediction (for planning), the iterative predictions produce large errors

Data space $x \in R^n$

*Prediction must be always in distribution*



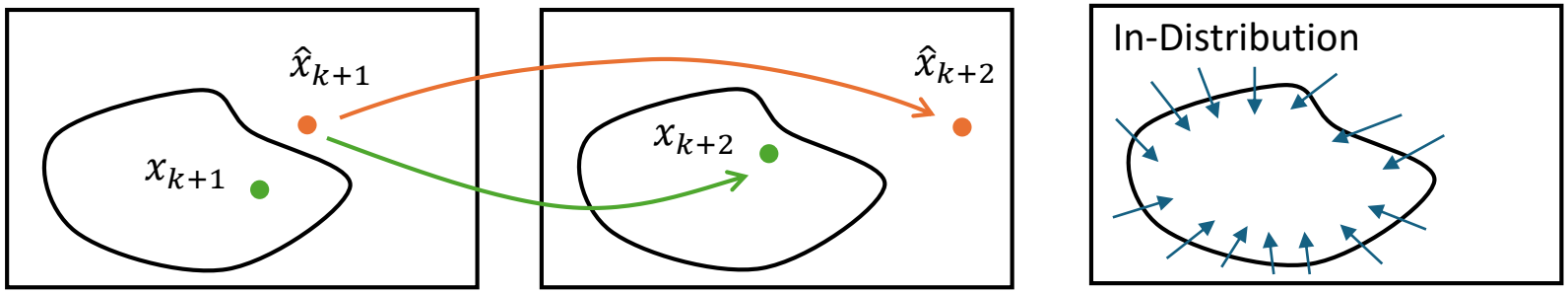



# Underfitting and compounding errors (5)

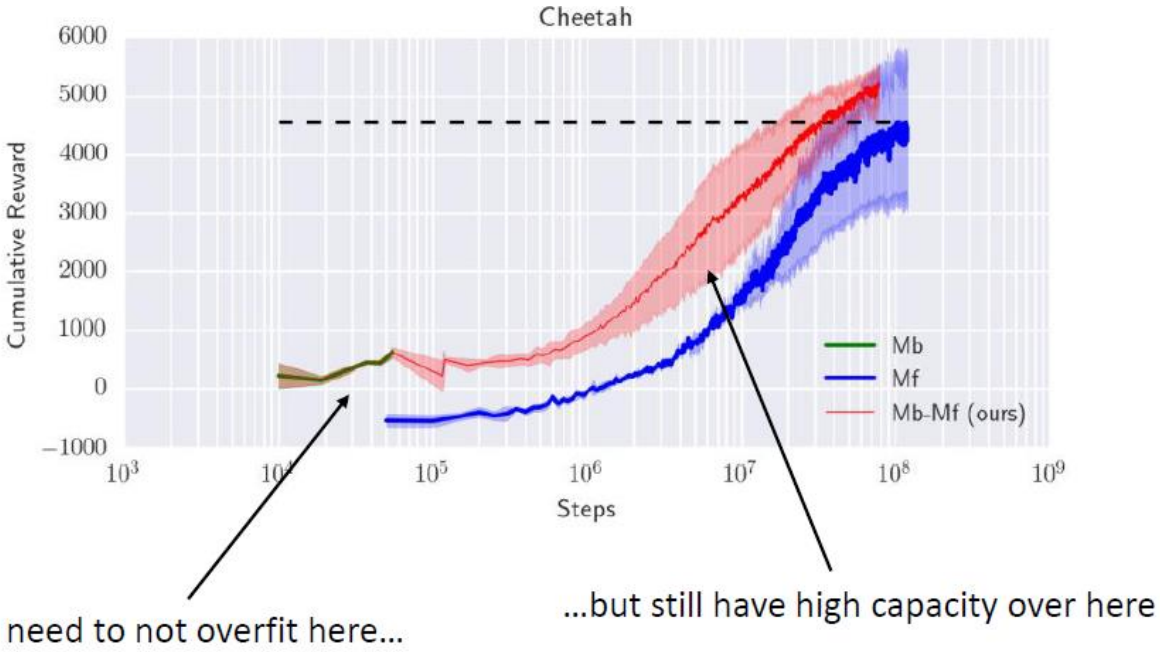
- Solution
  - Keep in-distribution (better representation)
  - hierarchical or hybrid model
  - curriculum learning
  - fine-tuned by model-free RL

- Examples
  - latent dynamics for contact-rich control (hybrid latent model)
  - hydrodynamic modeling of a robotic surface vehicle using representation learning for long-term prediction



# Underfitting and compounding errors (5)

- Example
  - Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning



# Model learning

- Other kinds of models
  - so far: modeling $p(s_{t+1}|s_t, a_t)$ or $p(z_{t+1}|z_t, a_t)$ and $p(z_t|s_{t-n:t})$

- Many alternatives
  - inverse model $p(a_t|s_t, s_{t+1})$
  - multi-step inverse model $p(a_{t:t+n}|s_t, s_{t+n})$
  - future prediction without actions $p(s_{t+1:t+n}|s_t)$
  - video interpolation $p(s_{t+1:t+n}|s_t, s_{t+n+1})$
  - transition distribution $p(s_t, a_t, s_{t+1})$

# When to use model-based RL?

- Big upsides and big downsides
  - immensely useful, far more data efficient if model is easy to learn
  - model can be trained on data without reward labels (fully self-supervised)
  - model is somewhat task-agnostic (can be transferred to other tasks)

  - model don’t optimize for task performance
  - sometimes harder to learn than a policy
  - another thing to train, more hyperparameters, more compute intensive