Reinforcement Learning

Lecture 10: Model-based RL (Continuous Actions)

Christopher Mutschler

Fraunhofer IIS

Fraunhofer-Institut für Integrierte Schaltungen IIS

Online Planning with Continuous Actions
Outline

- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - Background Planning
    - Environment data augmentation / simulation
    - Sample efficient policy learning
  - **Online Planning**
    - Discrete Actions
    - **Continuous Actions**
  - Auxiliary tasks
- Real-world application

Online Planning with Continuous Actions

The Objective

- Imagine this everyday situation….

$$s_t \longleftarrow a_t$$

$$\min_{\mathbf{a}_1,\dots,\mathbf{a}_T} \log p(\text{killed by Jason } | \mathbf{a}_1, \dots, \mathbf{a}_T)$$

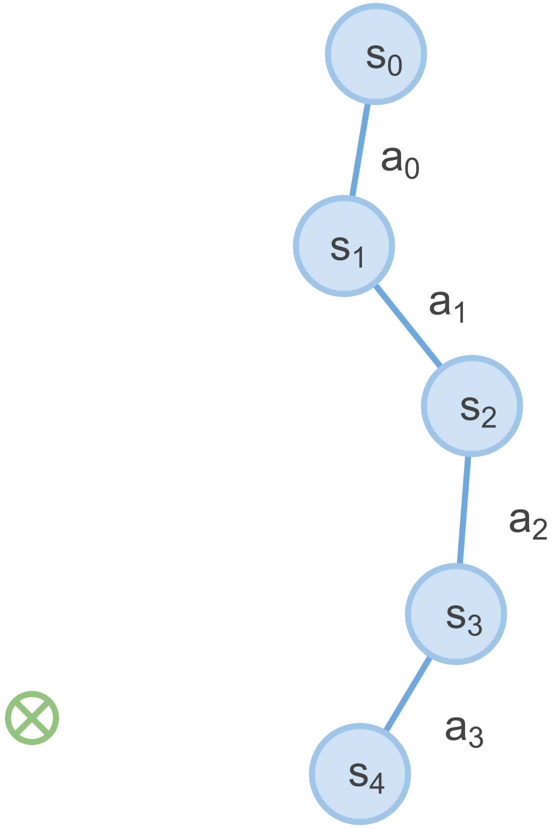
$$\max_{\mathbf{a}_1,\dots,\mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \quad \text{s.t.} \quad s_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\dots,\mathbf{a}_T} \underbrace{J(\mathbf{a}_1, \dots, \mathbf{a}_T)}_{\text{don't care what this is}}, \qquad \mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A})$$

- Simplest method:
  1. pick $\mathbf{A}_1, \dots, \mathbf{A}_N$ from some distribution (e.g., uniform)
  2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

Online Planning with Continuous Actions

Trajectory Optimization w/ Derivatives



Random shooting:

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

2. Expansion: execute actions $a_0, \dots, a_H$ to get states $s_1, \dots, s_H$

3. Evaluation: get trajectory reward $J(\mathrm{a}) = \sum_{t=0}^{H} r_t$

4. Back-propagation: get recursive gradients

Remember:

$$\max_{\mathbf{a}_1,\dots,\mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \quad \text{s.t.} \quad s_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

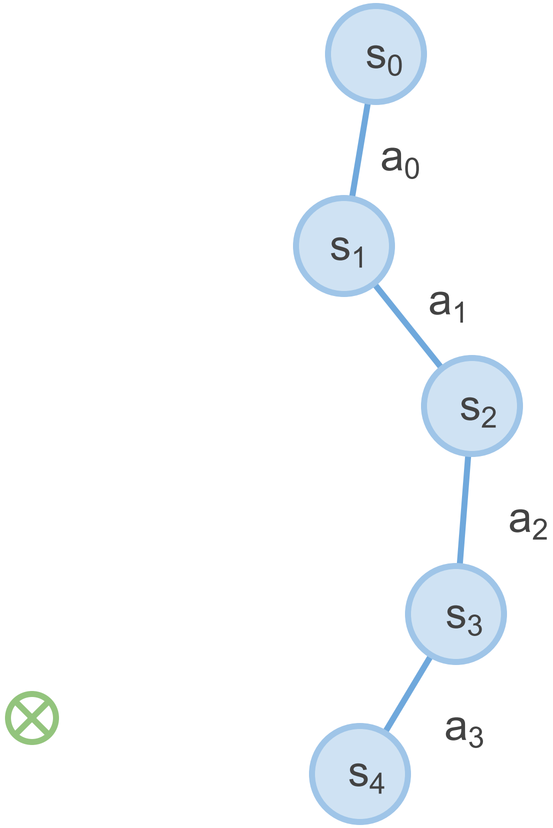
This is:

$$\max_{\mathbf{a}_1,\dots,\mathbf{a}_T} r(s_1, a_1) + r(f(s_1, a_1), a_2) + \dots + r(f(f(\dots)\dots), a_t)$$

https://sites.google.com/view/mbrl-tutorial

Online Planning with Continuous Actions

Trajectory Optimization w/ Derivatives



Random shooting:

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

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

reward model derivatives

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

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

transition model derivatives

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

computed recursively

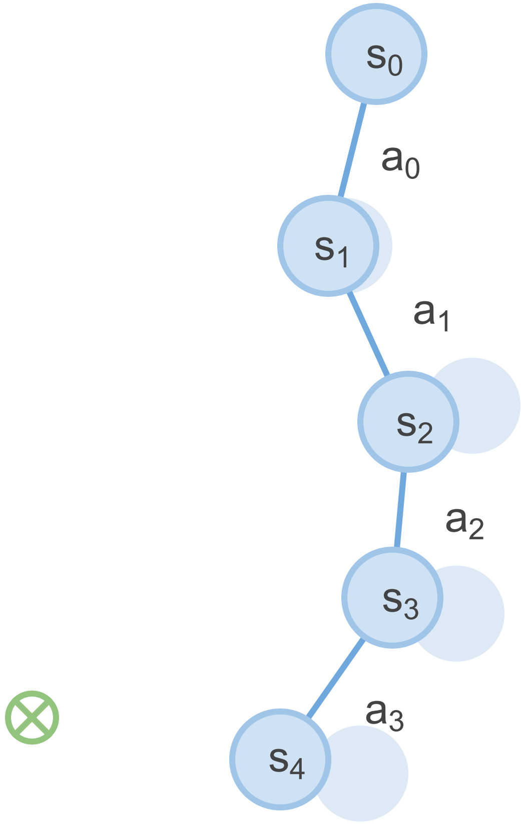
Easily available via auto-diff!

https://sites.google.com/view/mbrl-tutorial

Fraunhofer IIS

Online Planning with Continuous Actions

Trajectory Optimization w/ Derivatives



Random shooting:

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

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

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

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

5. Update actions via gradient ascent and repeat steps 2-5

https://sites.google.com/view/mbrl-tutorial

Online Planning with Continuous Actions

Trajectory Optimization w/ Derivatives

Shooting methods vs. collocation

- Shooting methods only optimize over actions:

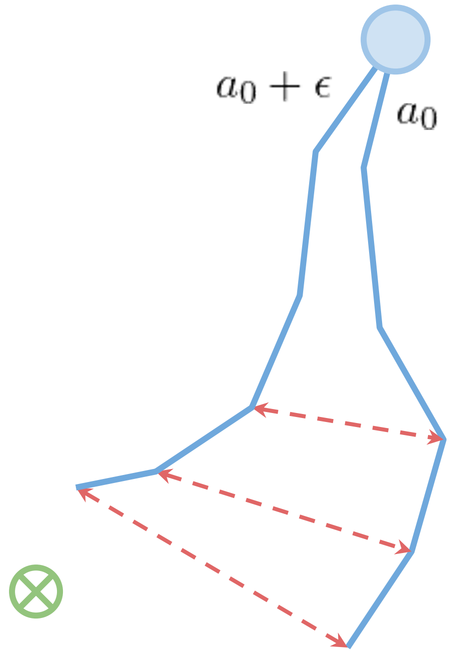
$$\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \underbrace{r(s_1,a_1)+r(f(s_1,a_1),a_2)+\cdots+r(f(f(\ldots)\ldots),a_t)}_{\text{Same issue as exploding/vanishing gradients in RNN training (but cannot change transition function here)}}$$

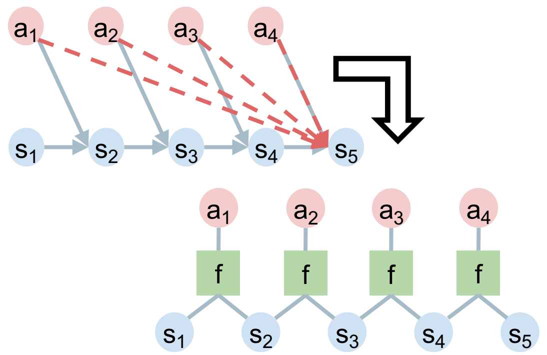
- This leads to high sensitivity → poorly conditioned
  - small changes in early actions lead to large state changes downstream
- **Collocation:** optimize for states and/or actions directly

$$\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t) \quad \text{s.t.} \quad s_{t+1} = f(\mathbf{s}_t,\mathbf{a}_t)$$

⬇

$$\max_{\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t) \quad \text{s.t.} \quad \|s_{t+1} - f(s_t,a_t)\| = 0$$





https://sites.google.com/view/mbrl-tutorial

Fraunhofer IIS

Online Planning with Continuous Actions

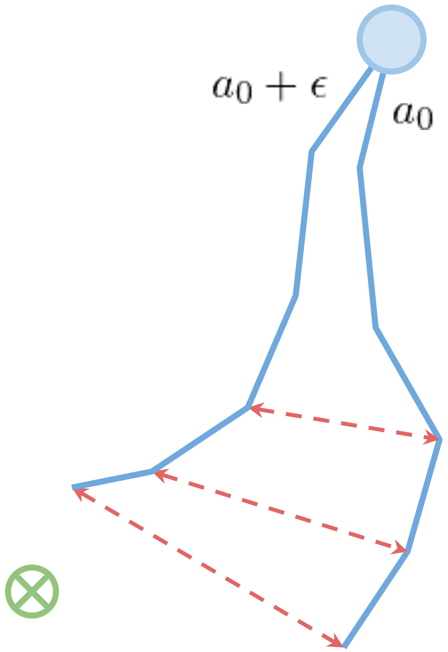
Trajectory Optimization w/ Derivatives

Shooting methods vs. collocation

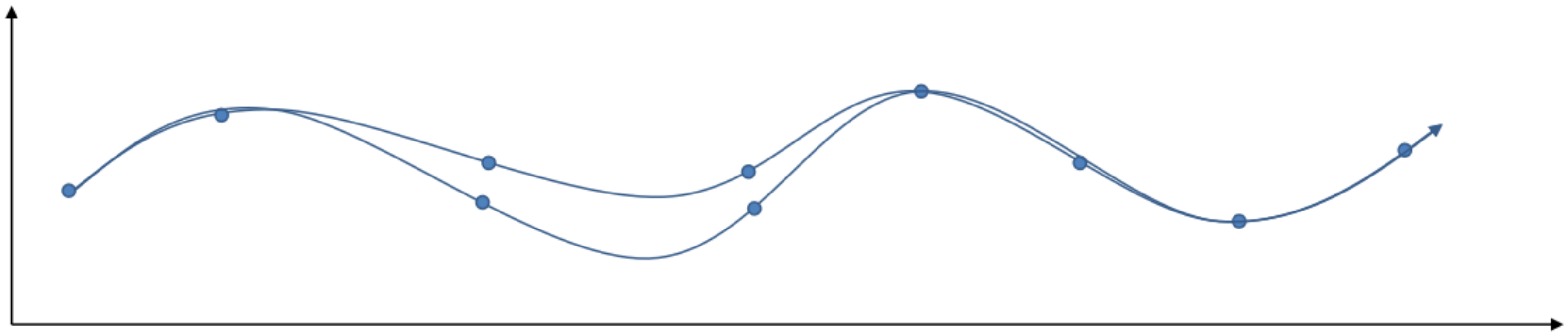
- Shooting methods only optimize over actions:

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \underbrace{r(s_1,a_1) + r(f(s_1,a_1),a_2) + \cdots + r(f(f(\ldots)\ldots),a_t)}_{\text{Same issue as exploding/vanishing gradients in RNN training (but cannot change transition function here)}}$$

- This leads to high sensitivity → poorly conditioned
  - small changes in early actions lead to large state changes downstream



- **Collocation:** optimize for states and/or actions directly



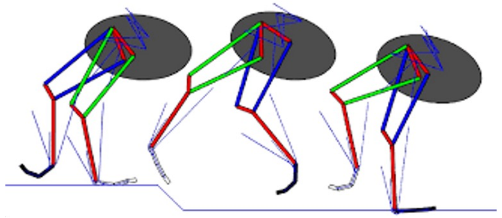
Online Planning with Continuous Actions

Trajectory Optimization w/ Derivatives

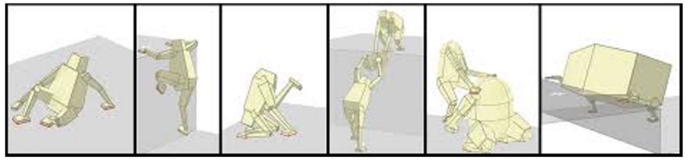
Shooting methods vs. collocation

Summary:

- Well-conditioned optimization problem
  - Changing $s_1 a_1$ becomes similar to changing $s_T a_T$
- Larger, but easier to optimize search space
  - good for contact-rich problems



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



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

Online Planning with Continuous Actions

Cross Entropy Maximization

- → Simplest method:
  1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)
  2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

- Was this really a good idea? Can we do better?

- Yes, we can! → **Cross Entropy Maximization (CEM)**
  Conceptually:

1. Sample $A_1, \ldots, A_N$ from $p(A)$
2. Evaluate $J(A_1), \ldots, J(A_N)$
3. Pick best ones $A_{i_1}, \ldots, A_{i_M}$ with $M < N$
4. Refit $p(A)$ around the best ones

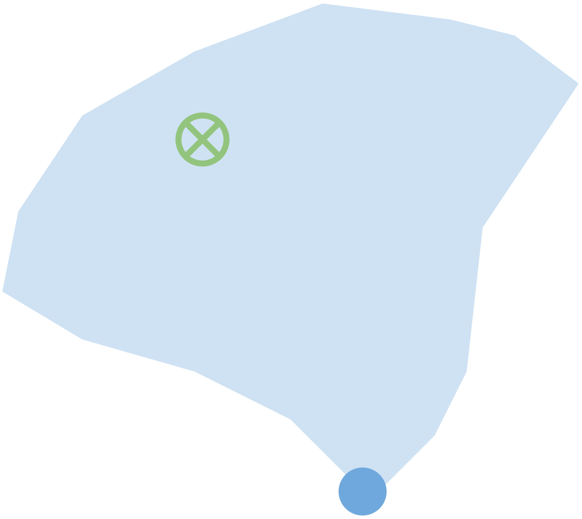
Online Planning with Continuous Actions

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

- Gradient-free
- Population-based (like e.g., Genetic Algorithms), can escape local optima

Maintain optimal trajectory
mean and covariance



Cross-Entropy Method (one iteration)

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

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

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

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

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

https://sites.google.com/view/mbrl-tutorial

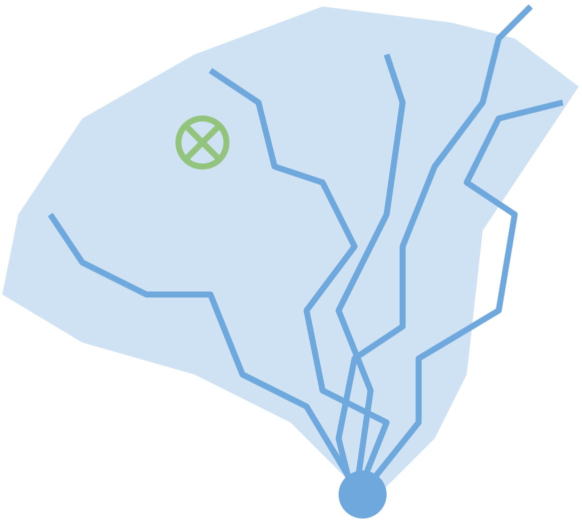
Stulp et al (2012). Path Integral Policy Improvement with Covariance Matrix Adaptation.

Online Planning with Continuous Actions

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

- Gradient-free
- Population-based (like e.g., Genetic Algorithms), can escape local optima



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

Use Gaussian to sample around current parameter mean

Cross-Entropy Method (one iteration)

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

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

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

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

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

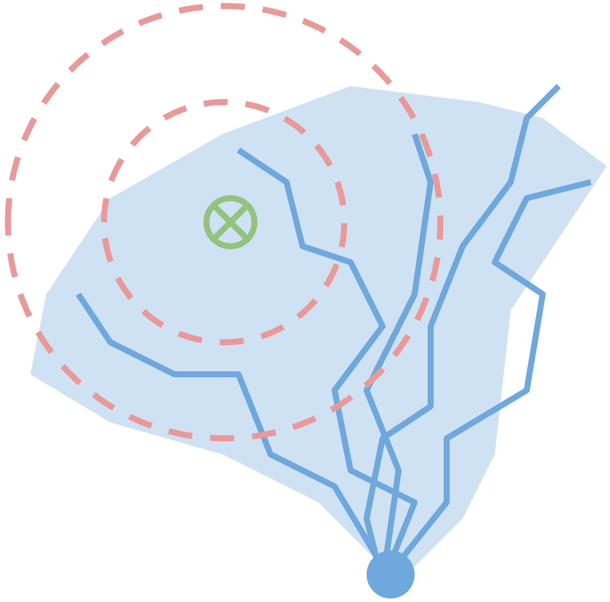
https://sites.google.com/view/mbrl-tutorial

Online Planning with Continuous Actions

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

- Gradient-free
- Population-based (like e.g., Genetic Algorithms), can escape local optima



Evaluate **(using the model)** the sampled parameters and keep the top K samples

Cross-Entropy Method (one iteration)

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

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

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

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

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

https://sites.google.com/view/mbrl-tutorial

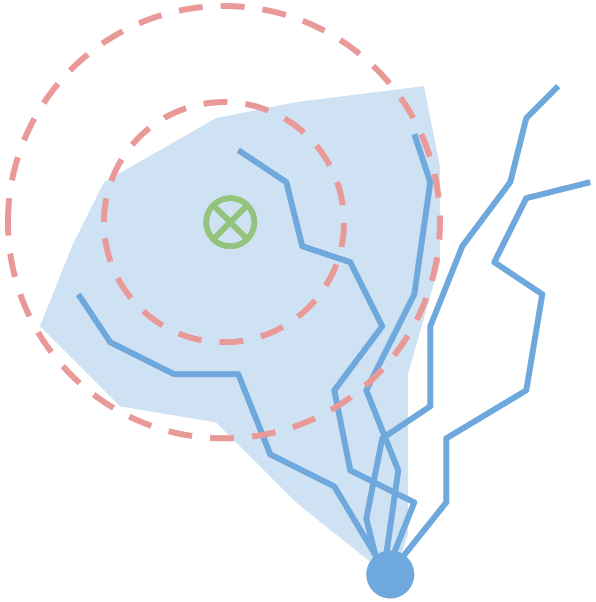
Stulp et al (2012). Path Integral Policy Improvement with Covariance Matrix Adaptation.

Online Planning with Continuous Actions

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

- Gradient-free
- Population-based (like e.g., Genetic Algorithms), can escape local optima



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

Re-fit the sampling Gaussian using the top K samples

Cross-Entropy Method (one iteration)

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

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

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

$$\boxed{\begin{aligned} \boldsymbol{\theta}^{new} &= \sum_{k=1}^{K_e} \frac{1}{K_e} \boldsymbol{\theta}_k && \textbf{update} \quad (4) \\ \Sigma^{new} &= \sum_{k=1}^{K_e} \frac{1}{K_e} (\boldsymbol{\theta}_k - \boldsymbol{\theta})(\boldsymbol{\theta}_k - \boldsymbol{\theta})^\intercal && \textbf{update} \quad (5) \end{aligned}}$$

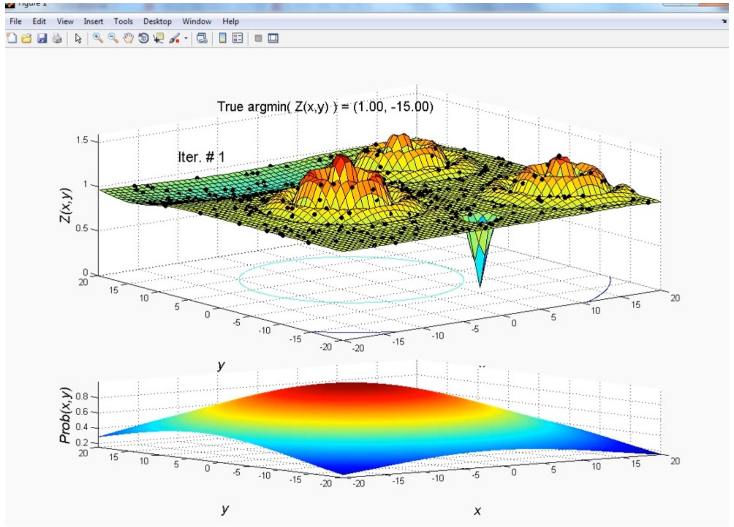
https://sites.google.com/view/mbrl-tutorial

Online Planning with Continuous Actions

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

- Gradient-free
- Population-based (like e.g., Genetic Algorithms), can escape local optima



https://www.youtube.com/watch?v=tNAIHEse7Ms

Re-fit the sampling Gaussian
using the top K samples

Cross-Entropy Method (one iteration)

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

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

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

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

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

https://sites.google.com/view/mbrl-tutorial

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

Fraunhofer IIS

Online Planning with Continuous Actions

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

- Gradient-free
- Population-based (like e.g., Genetic Algorithms), can escape local optima

- Advantages
  - Super-simple to implement
  - Easy to parallelize

- Limitations
  - Fails for high-dimensional action spaces
  - Gradient-free → increased sample complexity

Cross-Entropy Method (one iteration)

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

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

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

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

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

https://sites.google.com/view/mbrl-tutorial

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

Online Planning with Continuous Actions

Linear-Quadratic Regulator (LQR)

Analytical methods → 2nd order optimization and iLQR

- Iterative linearization of the dynamics
- Explores gradient information → fast convergence
- Very complicated method

Approximate transitions with linear functions and rewards with quadratics:

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

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

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.

https://sites.google.com/view/mbrl-tutorial

Online Planning with Continuous Actions

Outline

- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - Background Planning
    - Environment data augmentation / simulation
    - Sample efficient policy learning
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - **Auxiliary tasks**
- Real-world application

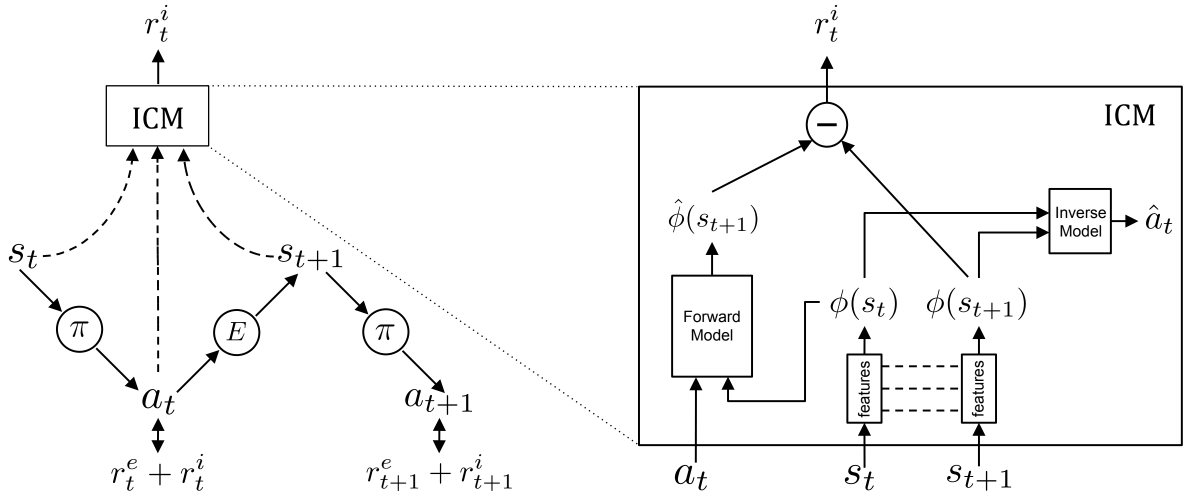
Online Planning with Continuous Actions

Auxiliary Tasks

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*
- Close to semi-supervised learning ideas

→ See more in Lecture on Exploration Strategies



Jürgen Schmidhuber: Curious model-building control systems. IJCNN.1991.
Pathak et al.: Curiosity-driven exploration by self-supervised prediction. ICML. 2017.

Real-World Application: Uncertainty in Model-based RL

Outline

- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- How can we use a model?
  - Background Planning
    - Environment data augmentation / simulation
    - Sample efficient policy learning
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- **Real-world application**

Real-World Application: Uncertainty in Model-based RL

Real-world application

Problem: Model is never perfect

- Aleatory/process uncertainty (risk): the process itself has noise
- Epistemic/Model uncertainty: our model is not perfect

Solutions:

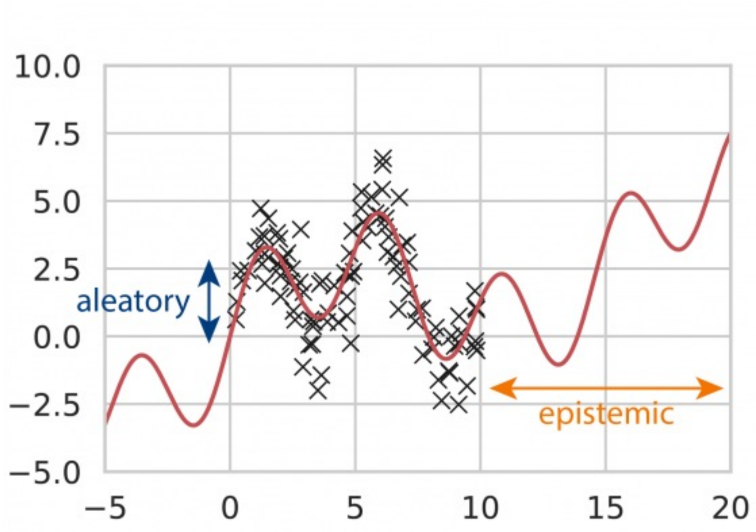
A. Act under imperfect models: use online re-planning (Model Predictive Control)
B. Estimate model uncertainty and use it for safe and efficient planning (MPC does not propagate uncertainty)
C. Combine A and B

Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

We will always have model errors

- Aleatory/process uncertainty (risk): the process itself has noise
- Epistemic/Model uncertainty: our model is not perfect



https://www.inovex.de/blog/uncertainty-quantification-deep-learning

Real-World Application: Uncertainty in Model-based RL

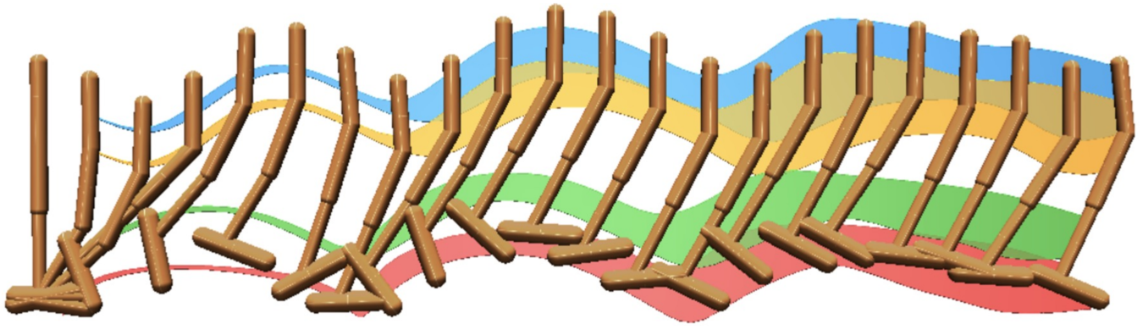
Model Uncertainty

We will always have model errors

- Aleatory/process uncertainty (risk): the process itself has noise
- Epistemic/Model uncertainty: our model is not perfect
- Model errors are additive: the prediction error will become larger as we try to predict further into the future

  → Small errors propagate and compound

  - Planner might even exploit such model errors!
  - Longer rollouts are less reliable



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

Real-World Application: Uncertainty in Model-based RL
Model Uncertainty

Model Calibration

- Big neural networks exhibit superior predictive power
  - But are not well calibrated in practice!
- Calibration affected by
  - depth & width of the network (the bigger the more overconfident)
  - weight decay & batch normalization
- Most calibration methods are post-hoc methods
  - Such methods do not work in RL settings!

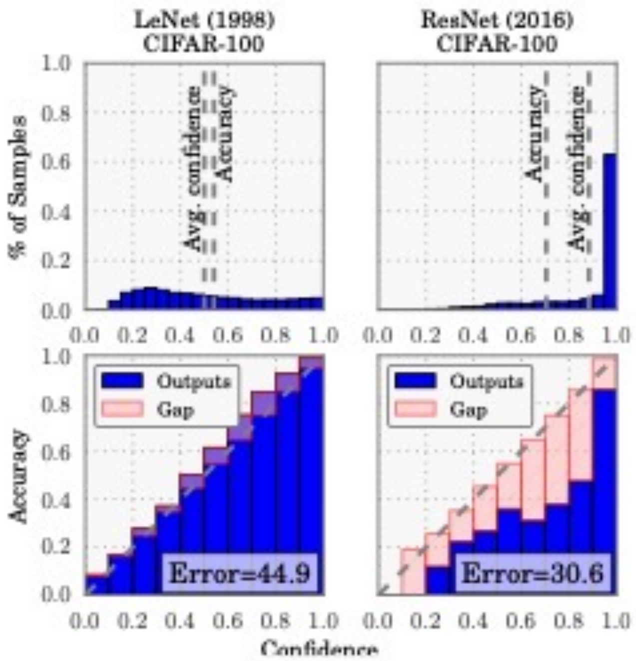


Figure 1. Confidence histograms (top) and reliability diagrams (bottom) for a 5-layer LeNet (left) and a 110-layer ResNet (right) on CIFAR-100. Refer to the text below for detailed illustration.

Chuan Guo et al.: On Calibration of Model Neural Networks. ICML. 2017.

Fraunhofer IIS

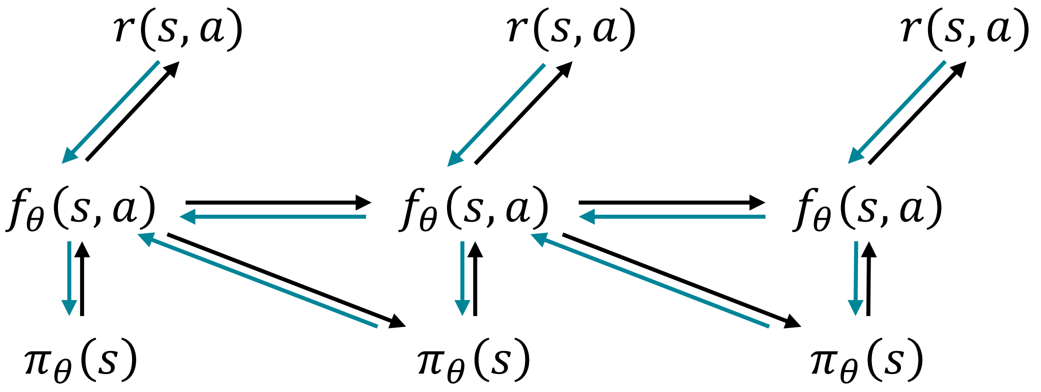
Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Recap: Background Planning with Policy Backpropagation

Better Algorithm:

1. Run a base policy $\pi_0(a_t|s_t)$ (e.g., a random policy) to collect data samples $\mathcal{D}\{(s,a,s')_i\}$
2. Learn a dynamics model $f_\theta(s,a)$ by minimizing $\sum_i \|f_\theta(s_i,a_i) - s_i'\|^2$
3. Backpropagate through $f_\theta(s,a)$ into policy to optimize $\pi_\theta(a_t|s_t)$
4. Run $\pi_\theta(a_t|s_t)$
5. Append visited tuples (s,a,s') to $\mathcal{D}$



Backprop:

$$\max_\theta \sum_t \gamma^t r(s_t,a_t)$$

Remember: we make mistakes!

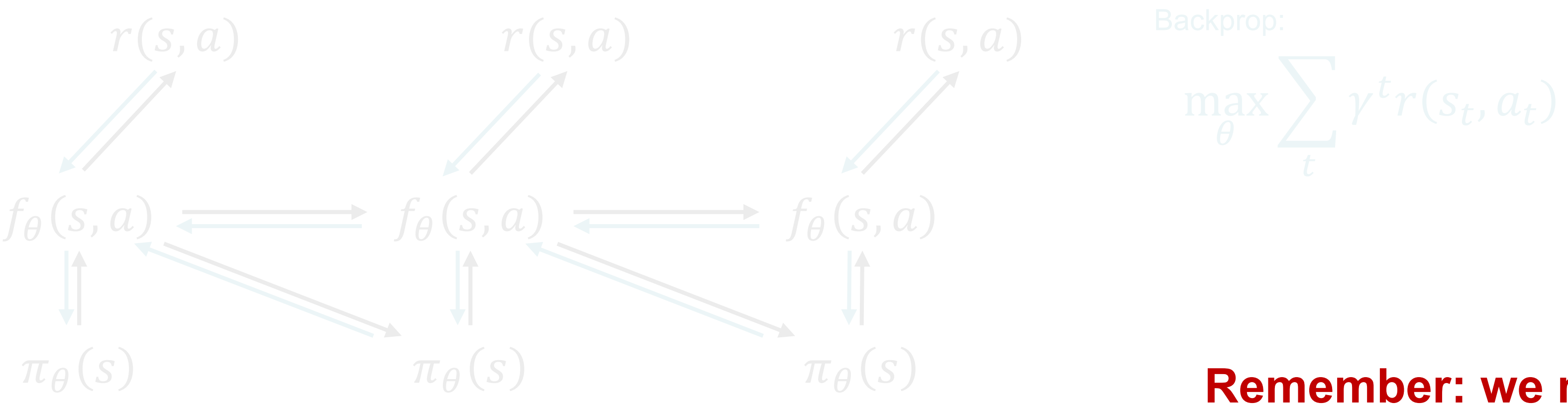
Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Recap: ~~Background Planning~~ with Policy Backpropagation

Better Algorithm for online planning:

1. Run a base policy $\pi_0(a_t|s_t)$ (e.g., a random policy) to collect data samples $\mathcal{D}\{(s,a,s')_i\}$
2. Learn a dynamics model $f_\theta(s,a)$ by minimizing $\sum_i \|f_\theta(s_i,a_i) - s_i'\|^2$
3. Plan through $f_\theta(s,a)$ to choose actions
4. Execute those actions
5. Append visited tuples (s,a,s') to $\mathcal{D}$



Remember: we make mistakes!

Real-World Application: Uncertainty in Model-based RL

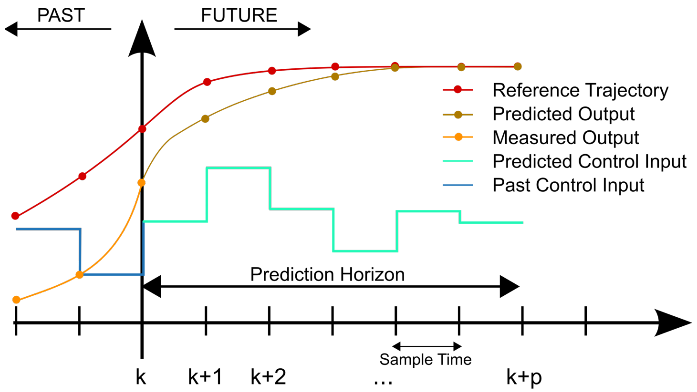
Model Uncertainty

Idea: don't commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)

Every N episodes

1. Run a base policy $\pi_0(a_t|s_t)$ (e.g., a random policy) to collect data samples $\mathcal{D}\{(s,a,s')_i\}$
2. Learn a dynamics model $f_\theta(s,a)$ by minimizing $\sum_i \|f_\theta(s_i,a_i) - s_i'\|^2$
3. Plan until time horizon $H < T$ using the model
4. Apply only first action of the plan
5. Append visited tuples (s,a,s') to $\mathcal{D}$



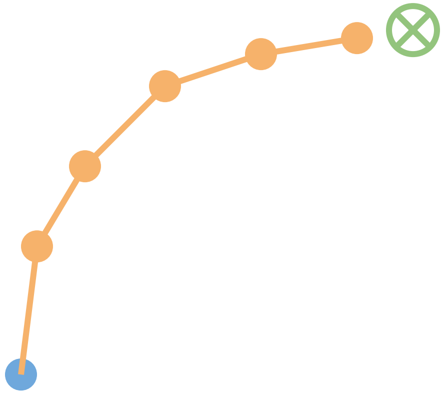
https://de.wikipedia.org/wiki/Model_Predictive_Control

Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Idea: don't commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)



Optimize trajectory

Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, ...\mathbf{u}_{N-1})$: 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;

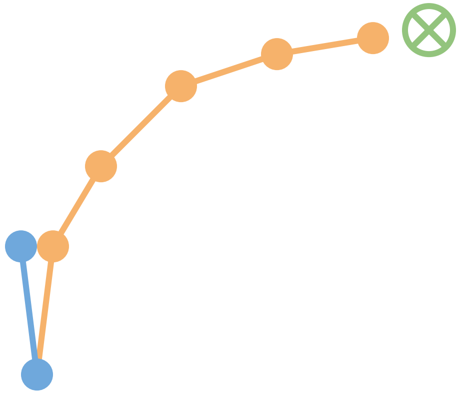
while *task not completed* **do**
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} = \mathbf{x}_{t_0}$;
    for $i \leftarrow 1$ **to** $N-1$ **do**
      $\mathbf{x}_{i+1} = \mathbf{x}_i + (\mathbf{f} + \mathbf{G}(\mathbf{u}_i + \delta\mathbf{u}_{i,k}))\Delta t$;
      $\tilde{S}(\tau_{i+1,k}) = \tilde{S}(\tau_{i,k}) + \tilde{q}$;
  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;

Example taken from https://sites.google.com/view/mbrl-tutorial

Real-World Application: Uncertainty in Model-based RL
Model Uncertainty

Idea: don't commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)



Apply first optimal action

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;

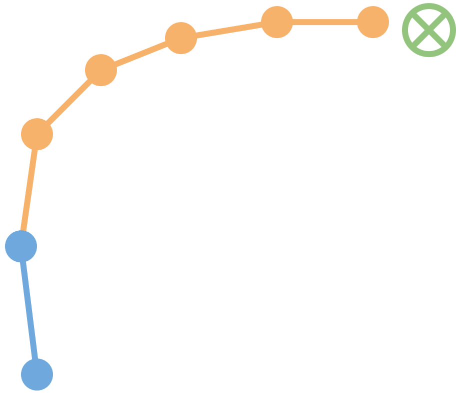
Example taken from https://sites.google.com/view/mbrl-tutorial

Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Idea: don’t commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)



Optimize trajectory

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;

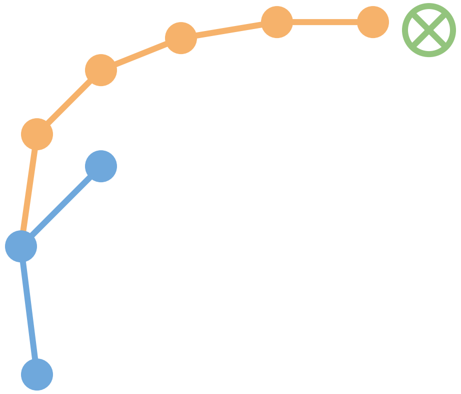
Example taken from https://sites.google.com/view/mbrl-tutorial

Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Idea: don't commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)



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;

Apply first optimal action

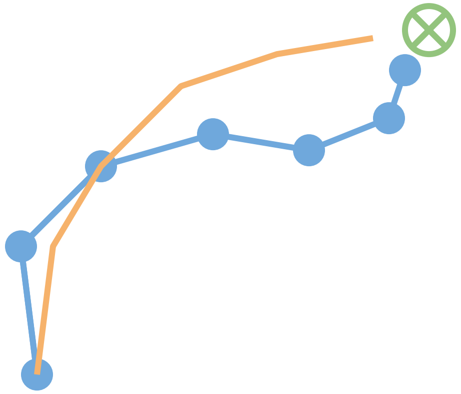
Example taken from https://sites.google.com/view/mbrl-tutorial

Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Idea: don't commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)



Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, ...\mathbf{u}_{N-1})$: 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;

Example taken from https://sites.google.com/view/mbrl-tutorial

Fraunhofer IIS

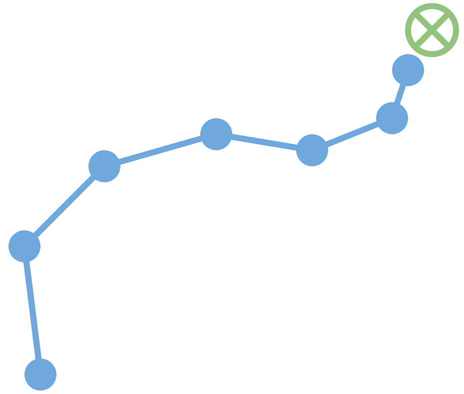
Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Idea: don't commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)

- Errors don't accumulate
- Don't need a perfect model, just one pointing in the right direction



Algorithm 1: Model Predictive Path Integral Control

Given: K: Number of samples;
N: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, ...\mathbf{u}_{N-1})$: 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;

Example taken from https://sites.google.com/view/mbrl-tutorial

Fraunhofer IIS

Real-World Application: Uncertainty in Model-based RL
Model Uncertainty

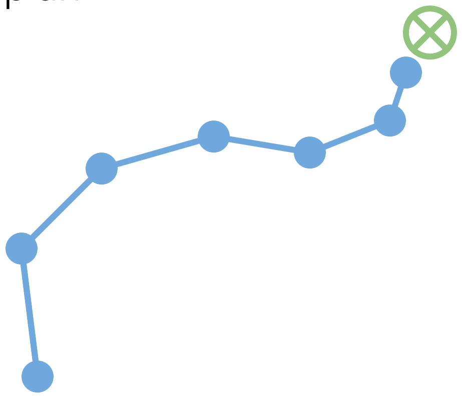
Idea: don't commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)

- Errors don't accumulate
- Don't need a perfect model, just one pointing in the right direction

This sounds expensive?

- Reuse solution from previous step as initial guess for next plan



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;

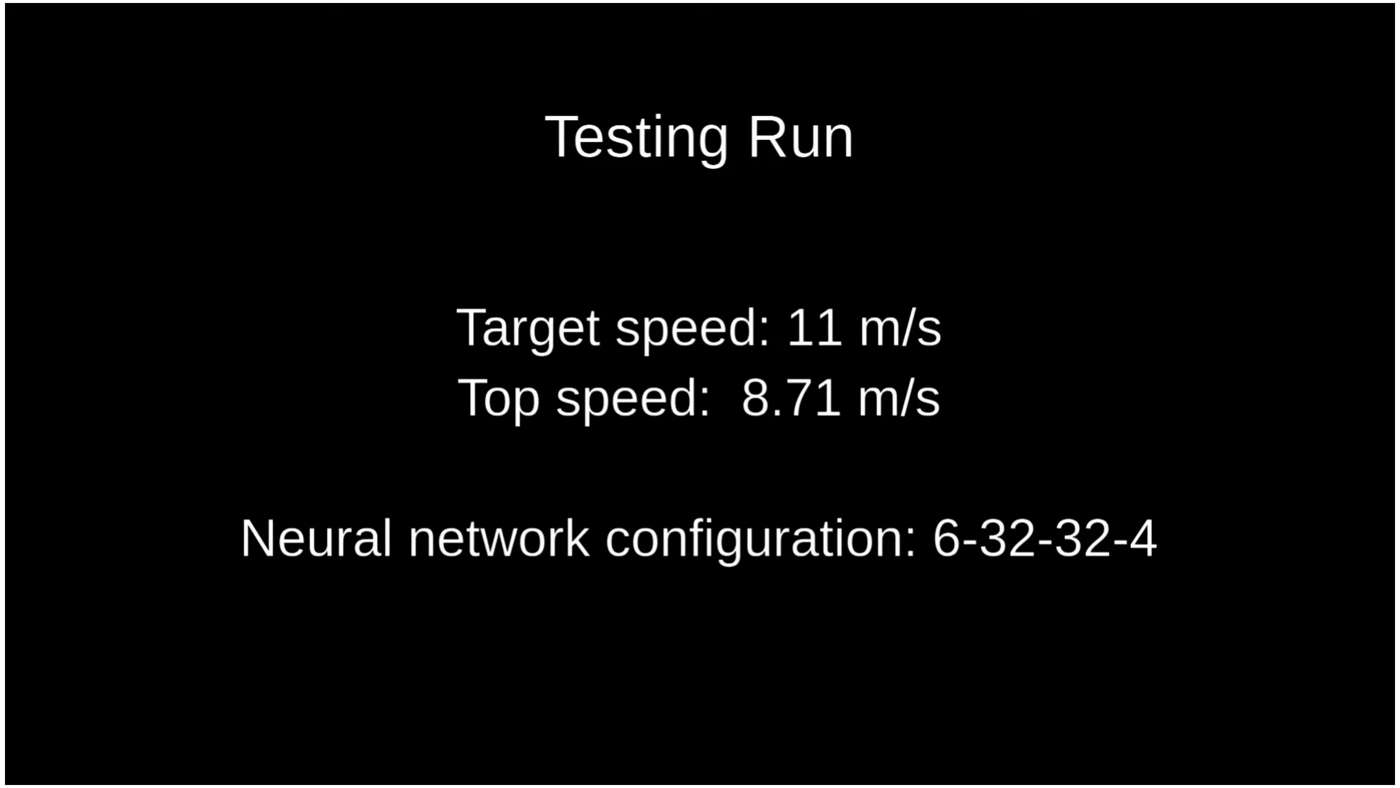
Example taken from https://sites.google.com/view/mbrl-tutorial

Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Idea: don’t commit to a plan bullheadedly but re-plan online

Model Predictive Control (MPC)



Williams et al.: Information Theoretic MPC for Model-Based Reinforcement Learning. ICRA. 2017.

Real-World Application: Uncertainty in Model-based RL

Model Uncertainty

Plan Conservatively: Model Uncertainty Propagation

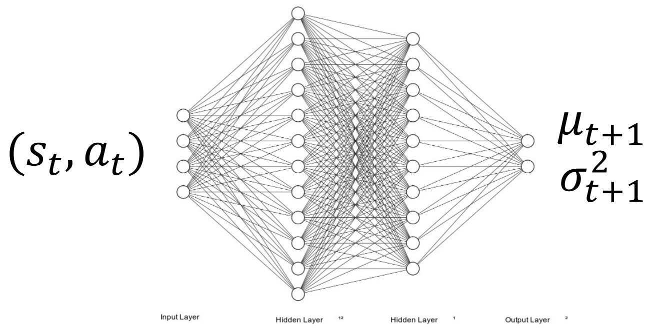
- Estimate/Quantify Model Uncertainty
  - Gaussian Processes
  - Monte Carlo Dropout
  - Probabilistic Neural Networks
  - Model Ensembles
- Propagate for multiple steps in the future

Real-World Application: Uncertainty in Model-based RL

Uncertainty Estimation

Probabilistic Neural Networks

- Capture Aleatoric/Process Uncertainty



- Loss function for training
  - Negative log prediction probability
  - Assume: dataset $\mathcal{D}$ with pairs of example data $\{(s_i, a_i), s_{i+1}\}$

$$\mathcal{L}(w) = -\sum_{i=1}^{n} \log \hat{f}(s_{i+1} |\, s_i, a_i; w)$$

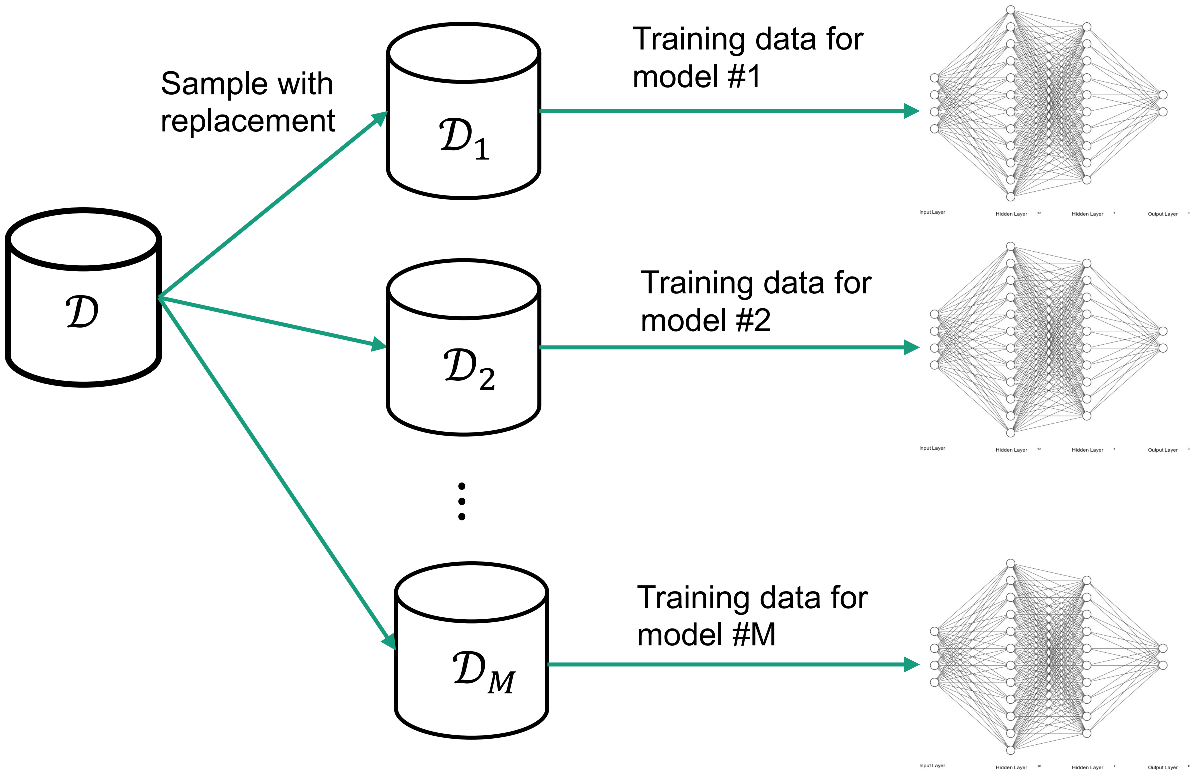
$$\mathcal{L}_{Gauss}(w) = \sum_{i=1}^{n} [\mu(s_i, a_i) - s_{i+1}]\, \Sigma^{-1}(s_i, a_i)[\mu(s_i, a_i) - s_{i+1}] + \log|\Sigma(s_i, a_i)|$$

Real-World Application: Uncertainty in Model-based RL

Uncertainty Estimation

Ensembles of Probabilistic Neural Networks

- Capture Epistemic/model Uncertainty



Real-World Application: Uncertainty in Model-based RL

Uncertainty Propagation

How do we propagate uncertainty to several future time-steps?

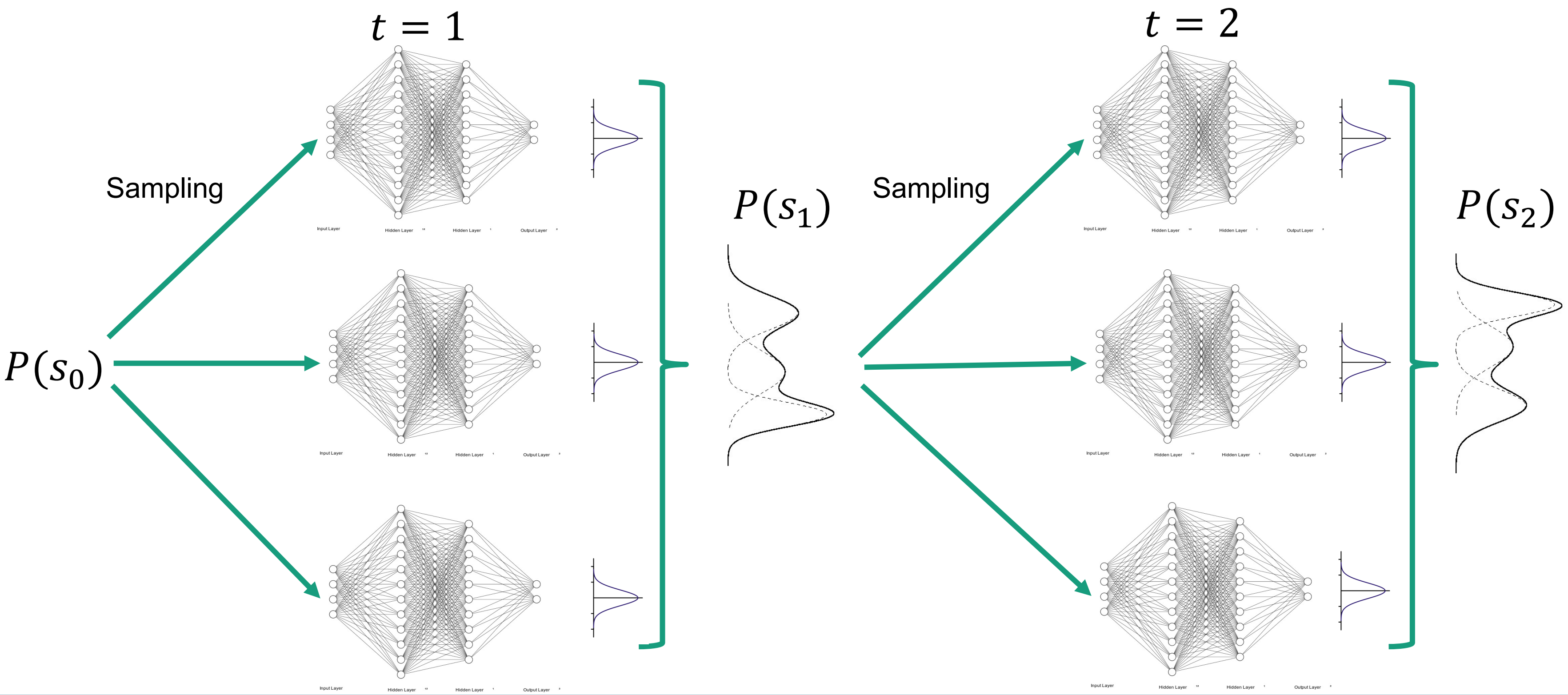
$$\begin{tikzcd}
s_0 \arrow[dr] & & s_1 \arrow[dr] & & s_2 \arrow[dr] & \\
& \hat{f} \arrow[ur] & & \hat{f} \arrow[ur] & & \hat{f} \\
a_0 \arrow[ur] & & a_1 \arrow[ur] & & a_2 \arrow[ur] &
\end{tikzcd}$$

Real-World Application: Uncertainty in Model-based RL

Uncertainty Propagation

How do we propagate uncertainty to several future time-steps?

- Moment Matching

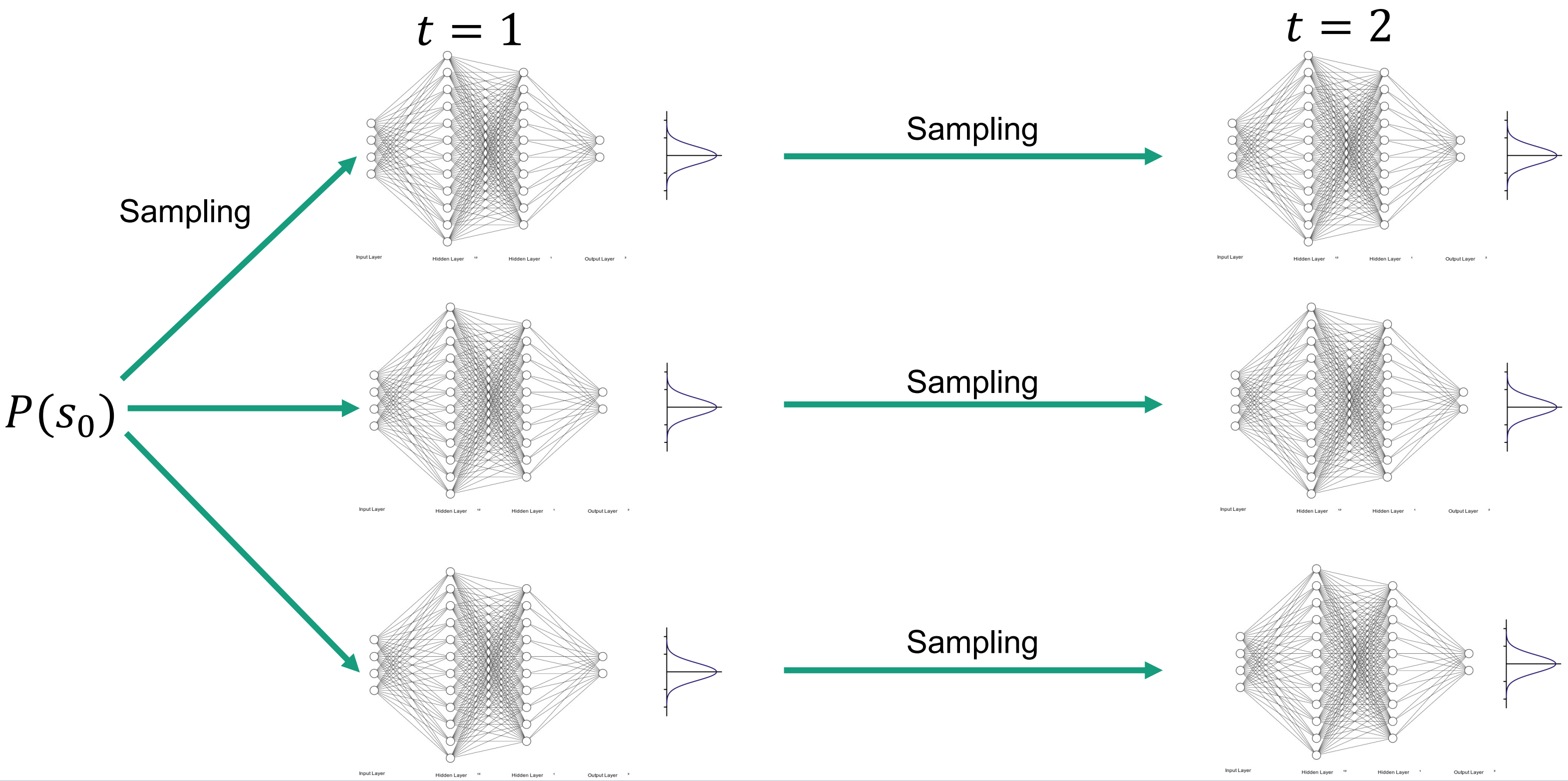


Real-World Application: Uncertainty in Model-based RL

Uncertainty Propagation

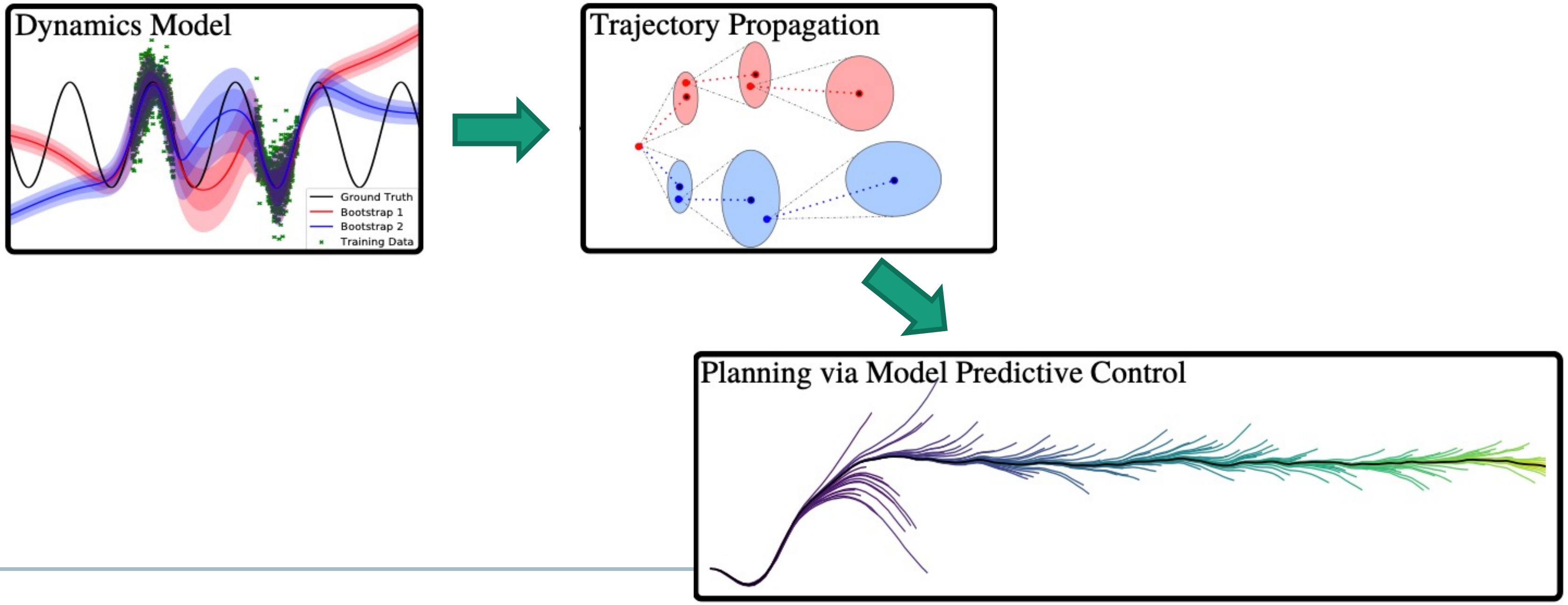
How do we propagate uncertainty to several future time-steps?

- Trajectory Sampling



Real-World Application: Uncertainty in Model-based RL
Uncertainty Estimation

Ensembles of Probabilistic Neural Networks
- Capture Epistemic/model Uncertainty
- Sample Trajectories
- Plan via MPC



Kurtland Chua et al.: Deep reinforcement learning in a handful of trials using probabilistic dynamics models. NIPS 2018.

Real-World Application: Uncertainty in Model-based RL

Use Uncertainty information

Plan conservatively:
Closed-Loop control with Model Uncertainty Propagation

- PILCO *(probabilistic inference for learning control)*
  - Background planning
  - Gaussian Processes as state-space model
  - Moment-matching for posterior (next state) distribution (uncertainty propagation)

Marc Deisenroth et al.: PILCO: A model-based and data-efficient approach to policy search. ICML 2011.

Real-World Application: Uncertainty in Model-based RL

Use Uncertainty information

Plan conservatively:
Closed-Loop control with Model Uncertainty Propagation

- PILCO *(probabilistic inference for learning control)*
  - Background planning
  - Gaussian Processes as state-space model
  - Moment-matching for posterior (next state) distribution (uncertainty propagation)
- Why GP?
  1. GPs have low extrapolation error by design (they fall back to prior distribution)

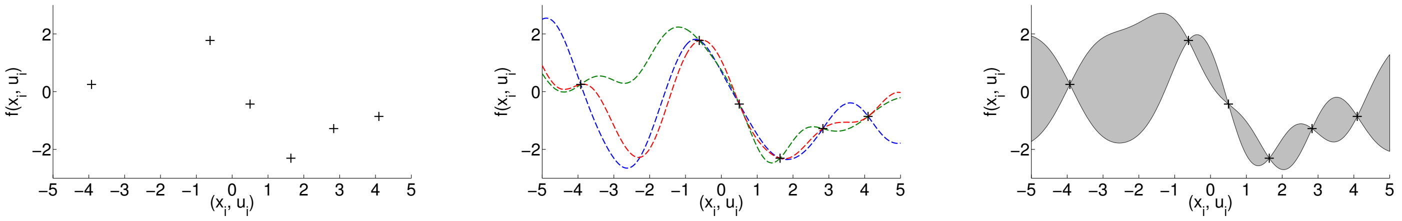


Figure 1. Small data set of observed transitions (left), multiple plausible deterministic function approximators (center), probabilistic function approximator (right). The probabilistic approximator models uncertainty about the latent function.

Marc Deisenroth et al.: PILCO: A model-based and data-efficient approach to policy search. ICML 2011.

Real-World Application: Uncertainty in Model-based RL

Use Uncertainty information

Plan conservatively:
Closed-Loop control with Model Uncertainty Propagation

- PILCO *(probabilistic inference for learning control)*
  - Background planning
  - Gaussian Processes as state-space model
  - Moment-matching for posterior (next state) distribution (uncertainty propagation)
- Why GP?
  1. GPs have low extrapolation error by design (they fall back to prior distribution)
  2. Moment-matching for RBF kernel can be calculated analytically

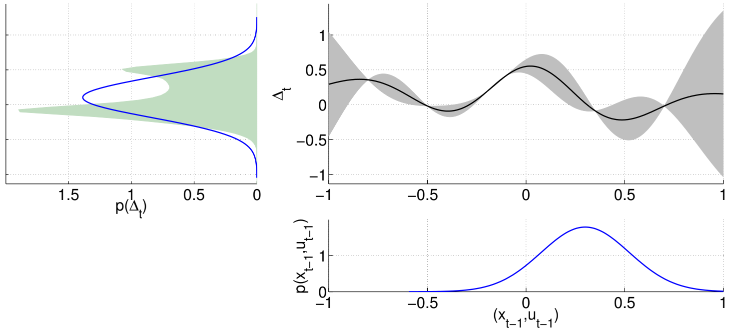


Figure 2. GP prediction at an uncertain input. The input distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ is assumed Gaussian (lower right panel). When propagating it through the GP model (upper right panel), we obtain the shaded distribution $p(\Delta_t)$, upper left panel. We approximate $p(\Delta_t)$ by a Gaussian with the exact mean and variance (upper left panel).

Marc Deisenroth et al.: PILCO: A model-based and data-efficient approach to policy search. ICML 2011.

Real-World Application: Uncertainty in Model-based RL

Use Uncertainty information

Plan conservatively:
Closed-Loop control with Model Uncertainty Propagation

- PILCO *(probabilistic inference for learning control)*
  - Background planning
  - Gaussian Processes as state-space model
  - Moment-matching for posterior (next state) distribution (uncertainty propagation)

Algorithm 1 PILCO

```
init: Sample controller parameters θ ∼ N(0, I).
Apply random control signals and record data.
repeat
    Learn probabilistic (GP) dynamics model, see Sec. 2.1, using all data.
    Model-based policy search, see Sec. 2.2–2.3.
    repeat
        Approximate inference for policy evaluation, see Sec. 2.2: get J^π(θ), Eqs. (10)–(12), (24).
        Gradient-based policy improvement, see Sec. 2.3: get dJ^π(θ)/dθ, Eqs. (26)–(30).
        Update parameters θ (e.g., CG or L-BFGS).
    until convergence; return θ*
    Set π* ← π(θ*).
    Apply π* to system (single trial/episode) and record data.
until task learned
```

https://www.youtube.com/watch?v=XiigTGKZfks

Marc Deisenroth et al.: PILCO: A model-based and data-efficient approach to policy search. ICML 2011.

Real-World Application: Uncertainty in Model-based RL

Use Uncertainty information

Plan conservatively:
Closed-Loop control with Model Uncertainty Propagation

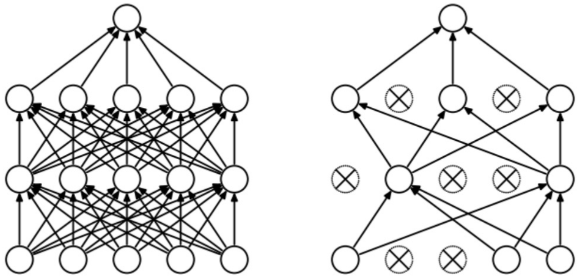
- PILCO *(probabilistic inference for learning control)*
  - Background planning
  - Gaussian Processes as state-space model
  - Moment-matching for posterior (next state) distribution (uncertainty propagation)
- Advantages:
  - \+ Unprecedent sample efficiency
  - \+ Robust to small model errors
- Shortcomings:
  - \- GPs scale cubically with the number of training data samples
  - \- Only specific classes of policies and reward functions are supported
  - \- Assumes/requires problems with very smooth dynamics

Real-World Application: Uncertainty in Model-based RL

Use Uncertainty information

Plan conservatively:
Closed-Loop control with Model Uncertainty Propagation

- Deep PILCO
  - Background planning
  - Bayesian neural network (MC Dropout/Ensemble as state-space model
  - Moment-matching for posterior (next state) distribution (uncertainty propagation)
  - **Addresses all PILCO shortcomings**
- Advantages:
  - \+ Scaling to large datasets + leveraging GPU processing
  - \+ Very flexible policies
  - \+ Robust to small model errors
- Shortcomings:
  - \- Still high execution times (e.g., for 40 learning iterations in Cartpole PILCO needed 20.7 hours (!!!) and Deep PILCO (GPU) needed 5.8 hours)



(a) Standard Neural Net (b) After applying dropout.

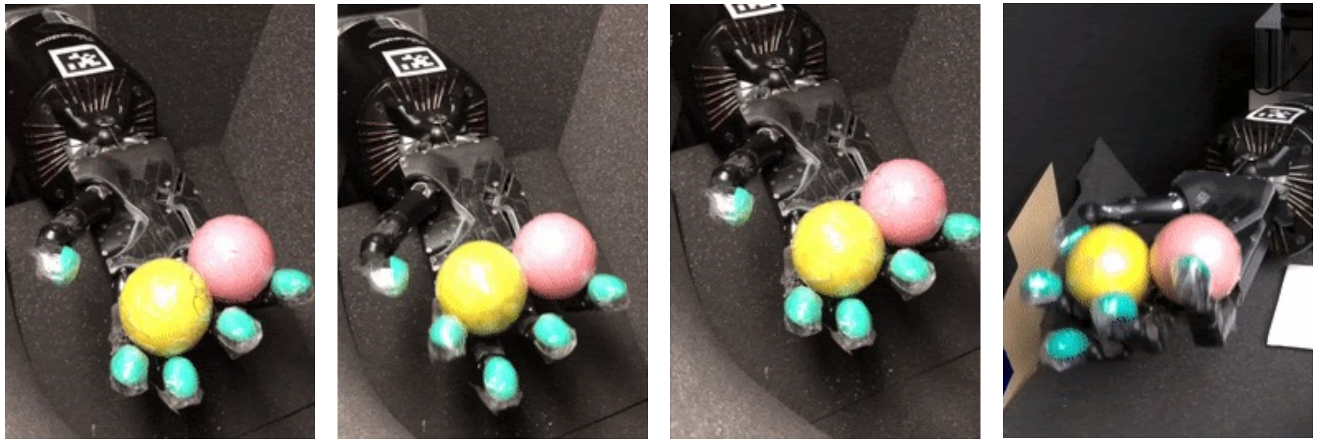
Nitish Srivastava et al.: Dropout: a simple way to prevent neural networks from overfitting. JMLR. 2014.

Summary

Real-world systems

MPC (CEM-like) + Ensemble of Probabilistic Models

- Online Planning (MPC)
- Trajectory sampling for uncertainty propagation
- Ensembles of probabilistic neural networks for modeling



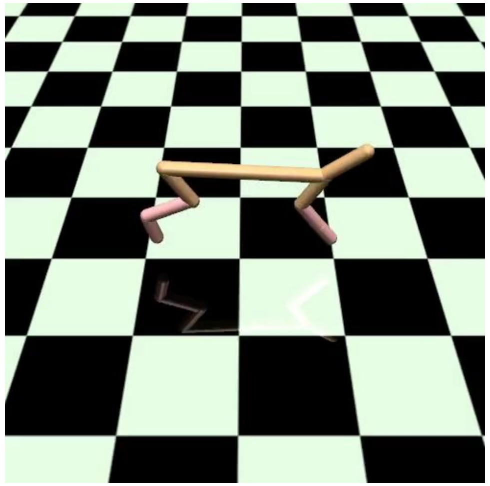
Training progress on the ShadowHand hardware.
From left to right: 0-0.25 hours, 0.25-0.5 hours, 0.5-1.5 hours, ~2 hours.

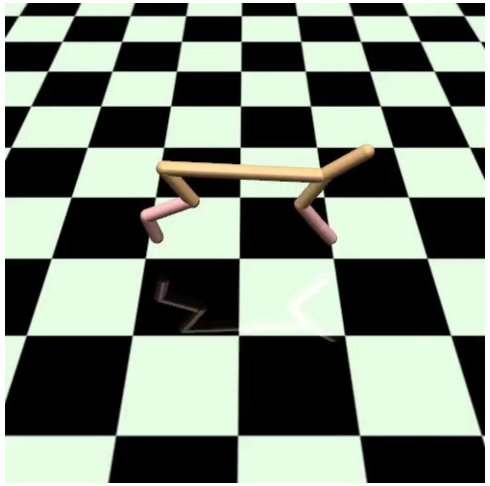
A. Nagabandi et al.: Deep dynamics models for learning dexterous manipulation. Conf. Robot Learning. 2020.
https://bair.berkeley.edu/blog/2019/09/30/deep-dynamics/

Summary
Real-world systems

Sample efficiency benchmarks

- Example Domain: MuJoCo HalfCheetah





http://ai.berkeley.edu/lecture_slides.html

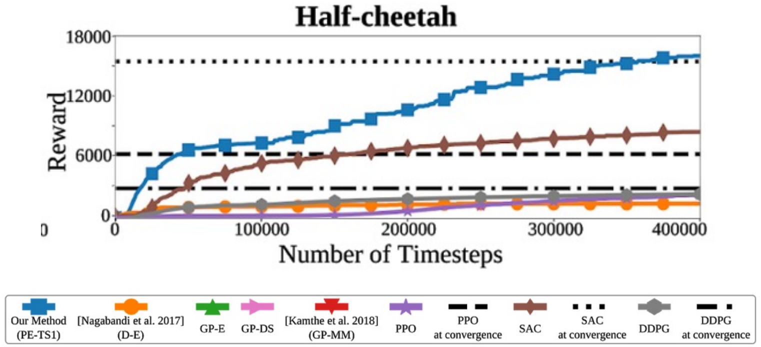
Summary

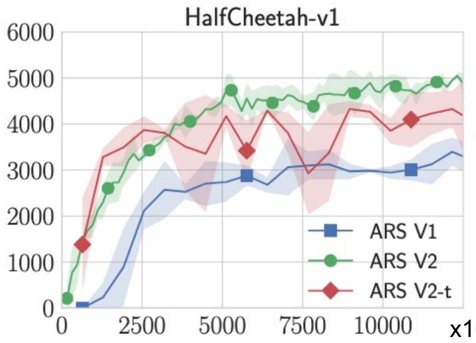
Real-world systems

Sample efficiency benchmarks

- Example Domain: MuJoCo HalfCheetah

FAIL!	Model-based RL (e.g., PILCO)
$k \cdot 10^5$	Model-based Deep RL (e.g., PETS)
$k \cdot 10^6$	Value-based Off-Policy Methods (e.g., SAC)
$k/2 \cdot 10^7$	Policy Gradient Methods (e.g., TRPO)
$k/2 \cdot 10^8$	Fully Online Methods (e.g., A3C)
$k \cdot 10^8$	Gradient-free Methods (e.g., ARS)





x1000 timesteps!

References

- https://sites.google.com/view/mbrl-tutorial
- Fazeli et al. (2019). See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. Science Robotics, 4(26).
- Fisac et al. (2019). A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems. IEEE Transactions on Automatic Control.
- Sadigh et al. (2016). Planning for autonomous cars that leverage effects on human actions. RSS 2016.
- Silver et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484.
- Segler, Preuss, & Waller (2018). Planning chemical syntheses with deep neural networks and symbolic AI. Nature, 555(7698).
- Salas & Powell (2013). Benchmarking a scalable approximate dynamic programming algorithm for stochastic control of multidimensional energy storage problems.
- Warren Powell's 2017 ECSO tutorial, "A Unified Framework for Optimization under Uncertainty"
- https://de.mathworks.com/help/ident/ug/modeling-a-vehicle-dynamics-system.html
- M. I. Palmqvist et al.: Model predictive control for autonomous driving of a truck. KTH Royal Institute of Technology School of Electrical Engineering. 2016.
- https://de.wikipedia.org/wiki/Deep_Learning
- https://en.wikipedia.org/wiki/Long_short-term_memory
- Ebert, Finn, et al. (2018); Finn & Levine (2017); Finn, Goodfellow, & Levine (2016)
- https://bair.berkeley.edu/blog/2018/11/30/visual-rl/
- Lange et al.: Batch reinforcement learning. In Reinforcement learning (pp. 45-73). Springer, Berlin, Heidelberg. 2012.
- Kaiser et al.: Model-based reinforcement learning for atari. arXiv preprint arXiv:1903.00374. 2019.
- https://worldmodels.github.io
- Ha & Schmidhuber: World Models. NeurIPS 2018.
- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
- Janner et al.: When to Trust Your Model: Model-Based Policy Optimization. NeurIPS 2019.

References (cont'd)

- Andrychowicz, OpenAI: Marcin, et al. "Learning dexterous in-hand manipulation." The International Journal of Robotics Research 39.1 (2020): 3-20.
- https://deepmind.com/research/case-studies/alphago-the-story-so-far
- https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules
- Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
- Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.
- https://deepmind.com/blog/article/alphago-zero-starting-scratch
- Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." Science 362.6419 (2018): 1140-1144.
- Schrittwieser, Julian, et al. "Mastering atari, go, chess and shogi by planning with a learned model." Nature 588.7839 (2020): 604-609.
- https://www.youtube.com/watch?v=tNAIHEse7Ms
- Todorov and Li (2005). A generalized iterative LQG method.
- Jürgen Schmidhuber: Curious model-building control systems. IJCNN.1991.
- Pathak et al.: Curiosity-driven exploration by self-supervised prediction. ICML. 2017.
- https://www.inovex.de/blog/uncertainty-quantification-deep-learning
- Janner et al (2019). When to Trust Your Model: Model-Based Policy Optimization.
- Chuan Guo et al.: On Calibration of Model Neural Networks. ICML. 2017.
- https://de.wikipedia.org/wiki/Model_Predictive_Control
- Williams et al.: Information Theoretic MPC for Model-Based Reinforcement Learning. ICRA. 2017.
- Kurtland Chua et al.: Deep reinforcement learning in a handful of trials using probabilistic dynamics models. NIPS 2018.

Fraunhofer IIS

References (cont’d)

- Marc Deisenroth et al.: PILCO: A model-based and data-efficient approach to policy search. ICML. 2011.
- https://www.youtube.com/watch?v=XiigTGKZfks
- Nitish Srivastava et al.: Dropout: a simple way to prevent neural networks from overfitting. JMLR. 2014.
- A. Nagabandi et al.: Deep dynamics models for learning dexterous manipulation. Conf. Robot Learning. 2020. https://bair.berkeley.edu/blog/2019/09/30/deep-dynamics/
- http://ai.berkeley.edu/lecture_slides.html
- Sergey Levine: CS285 Deep Reinforcement Learning
- Posa et al (2014). A Direct Method for Trajectory Optimization of Rigid Bodies Through Contact.
- Mordatch et al (2012). Discovery of Complex Behaviors through Contact-Invariant Optimization.
- Stulp et al (2012). Path Integral Policy Improvement with Covariance Matrix Adaptation.

Further reading & watching:

- Marc Deisenroth: The Role of Uncertainty in Model-based Reinforcement Learning. Workshop on Uncertainty Propagation in Composite Models, Munich, 2019. https://deisenroth.cc/talks/2019-10-10-munich.pdf
- Igor Mordatch and Jessica Hamrick: Tutorial on Model-Based Methods in Reinforcement Learning. Presented at International Conference on Machine Learning (ICML) 2020. https://sites.google.com/view/mbrl-tutorial
- Sergey Levine: CS 285 at UC Berkeley – Deep Reinforcement Learning. Lectures “Lecture 10: Model-based Planning”, “Lecture 11: Model-based Reinforcement Learning”, and “Lecture 11: Model-based Policy Learning”, http://rail.eecs.berkeley.edu/deeprlcourse/