Online Planning with Continuous Actions

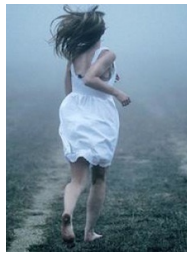
Christopher Mutschler

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

The Objective

- Imagine this everyday situation….

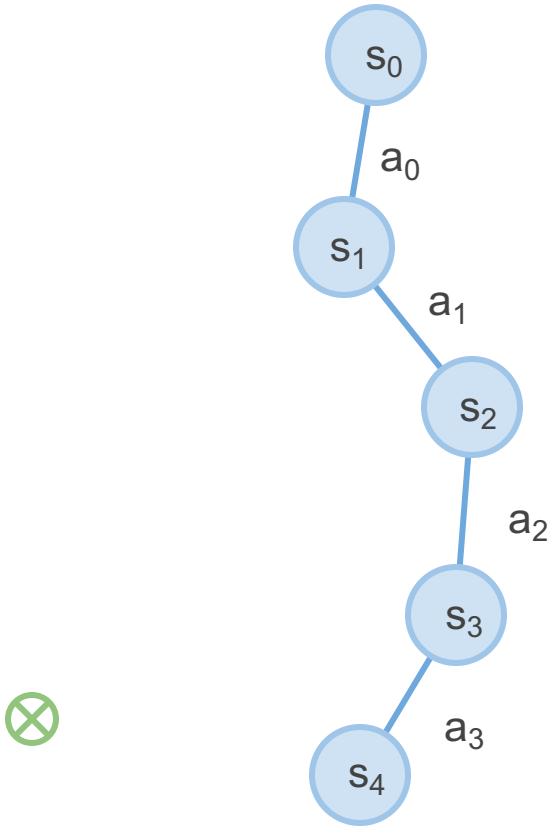

s_t ← $\min_{\mathbf{a}_1,\dots,\mathbf{a}_T} \log p(\text{killed by Jason} \mid \mathbf{a}_1, \dots, \mathbf{a}_T)$ — a_t

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

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

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

Trajectory Optimization w/ Derivatives



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

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

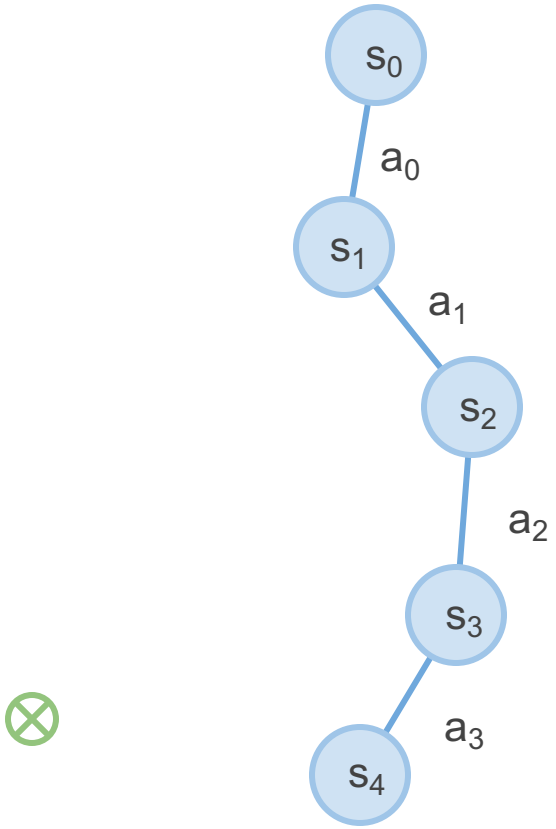
Remember:

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

This is:

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

Trajectory Optimization w/ Derivatives



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

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

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

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

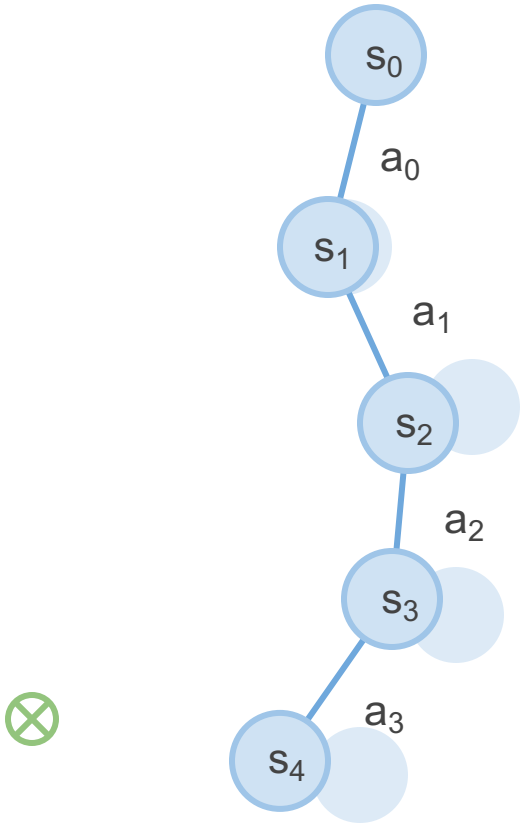
transition model derivatives

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

computed recursively

Easily available via auto-diff!

Trajectory Optimization w/ Derivatives



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

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

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

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

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

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

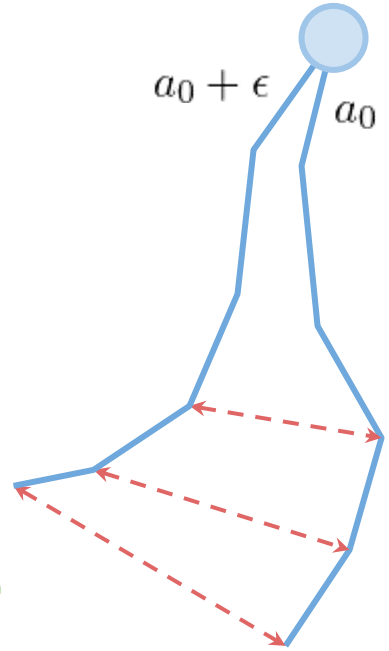
Trajectory Optimization w/ Derivatives

Shooting methods vs. collocation

- Shooting methods only optimize over actions:

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

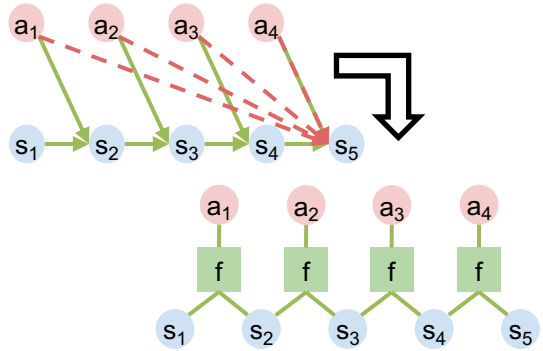
Same issue as exploding/vanishing gradients in RNN training
(but cannot change transition function here)



- 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

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



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

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

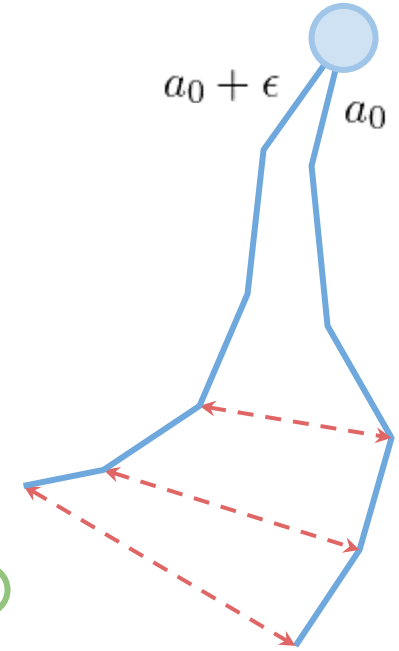
Trajectory Optimization w/ Derivatives

Shooting methods vs. collocation

- Shooting methods only optimize over actions:

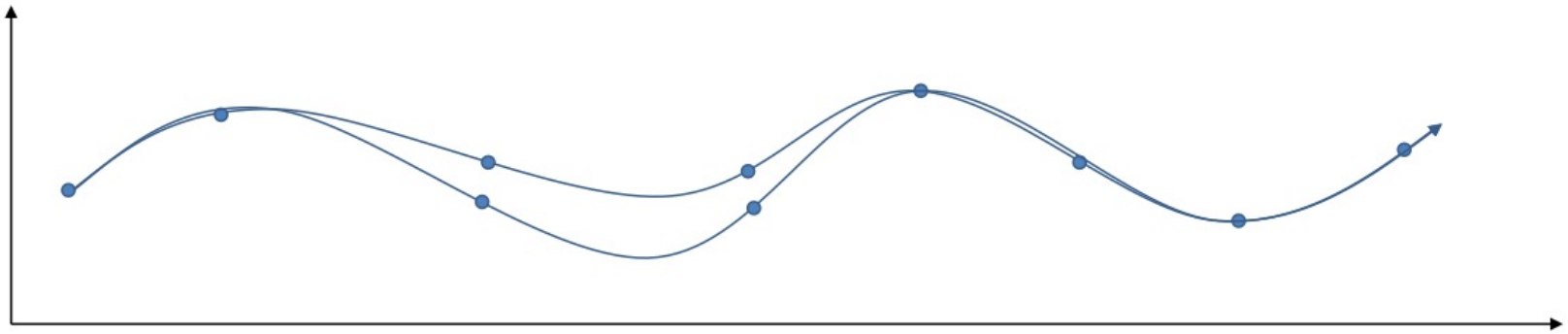
$$\min_{\mathbf{a}_1,\dots,\mathbf{a}_T} \underbrace{r(s_1,a_1)+r(f(s_1,a_1),a_2)+\cdots+r(f(f(\dots)\dots),a_t)}$$

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

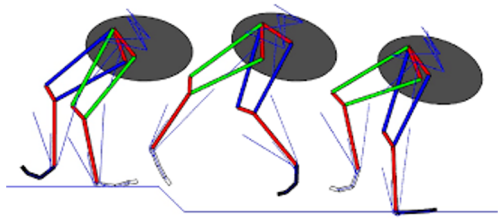


Trajectory Optimization w/ Derivatives

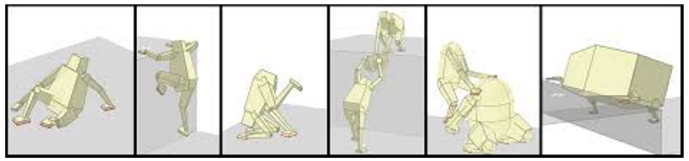
Shooting methods vs. collocation

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.

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?

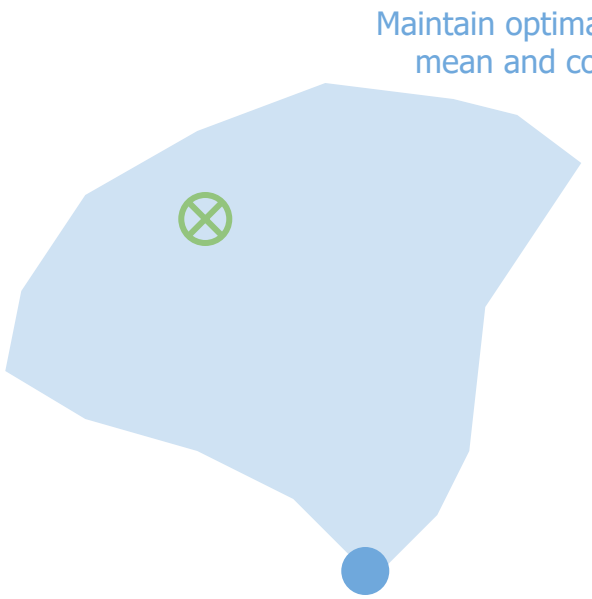
- Yes, we can! → **Cross Entropy Maximization (CEM)**
  Conceptually:

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

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

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



Cross-Entropy Method (one iteration)

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

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

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

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

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

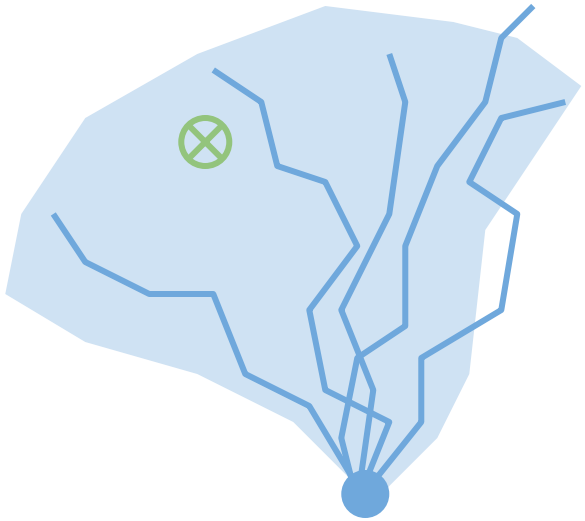
https://sites.google.com/view/mbrl-tutorial

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

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

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



Use Gaussian to sample around current parameter mean

Cross-Entropy Method (one iteration)

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

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

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

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

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

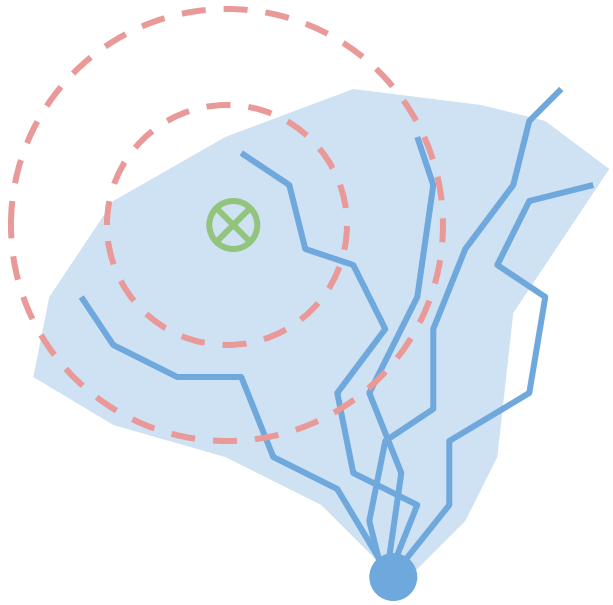
https://sites.google.com/view/mbrl-tutorial

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

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)

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

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

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

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

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

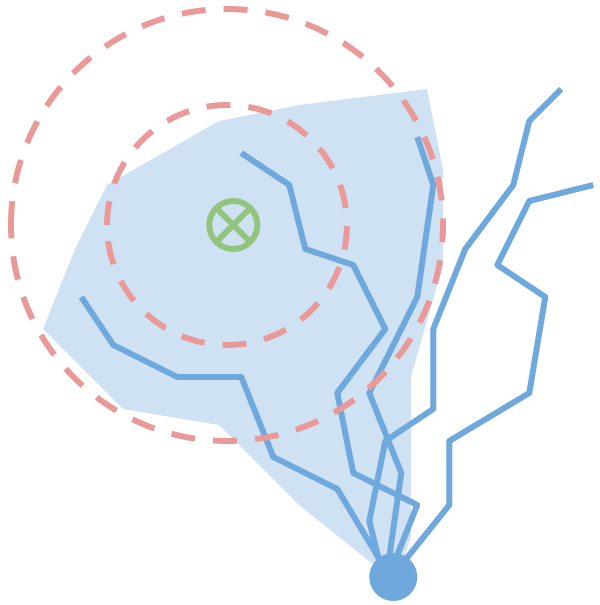
https://sites.google.com/view/mbrl-tutorial

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

Cross Entropy Maximization

Sampling methods → Cross Entropy Maximization

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



Re-fit the sampling Gaussian
using the top K samples

Cross-Entropy Method (one iteration)

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

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

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

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

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

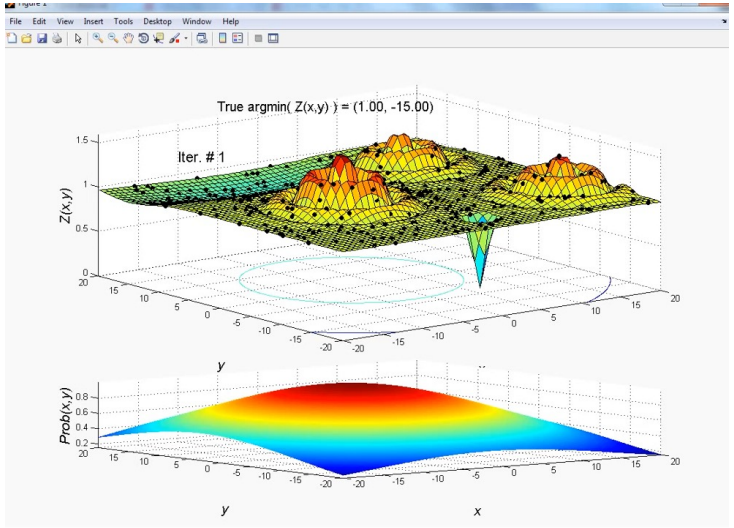
https://sites.google.com/view/mbrl-tutorial

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

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.

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.

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:

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

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

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.

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

Outline

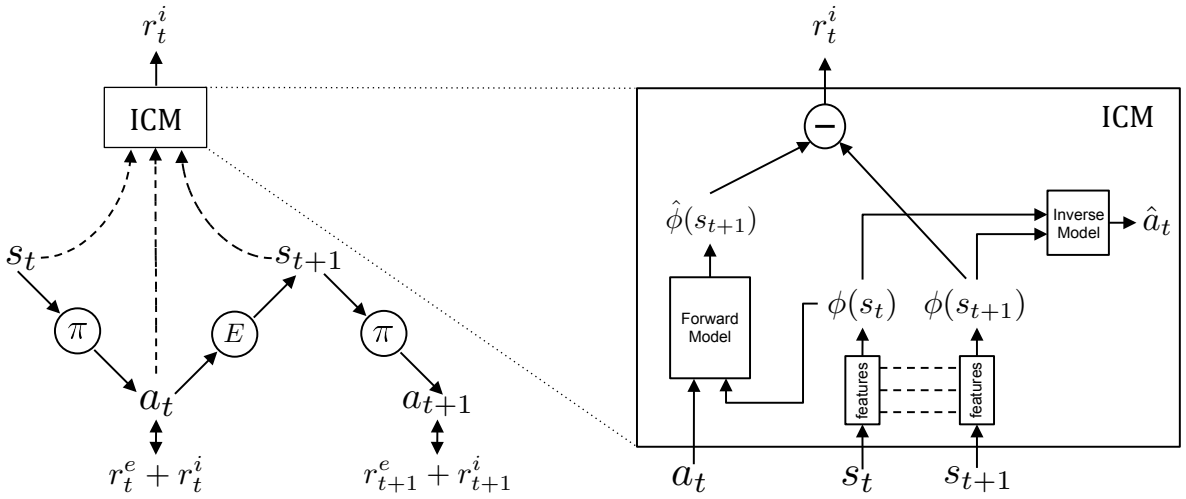
- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - Background Planning
    - Environment data augmentation / simulation
    - Sample efficient policy learning
  - Online Planning
    - Discrete Actions
    - Continuous Actions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

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.

References

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