Trustworthy AI Autonomy
M2-2: Model-based decision making

Ding Zhao
Assistant Professor
Carnegie Mellon University

2022 @ Ding Zhao
Plan for today

• Model-based control
  • LQR, iLQR, MPC

• Model-based reinforcement learning
  • Neural network based method
  • Local (linearized) model
  • Planning: Cross Entropy Method
  • Gaussian process-based Reinforcement learning (next lecture)
Recap: On-policy vs off-policy

- Policy optimization is almost always performed on-policy, which means that each update only uses data collected while acting according to the most recent version of the policy. The historical data collected with very old policy is not used. They can be used with both continuous and discrete states. Using gradient, they converge to a local minima of $J(\theta)$

- Q-learning, e.g., DQN, is almost always performed off-policy, which means that each update can use data collected during the whole training history, regardless of what policy the agent was choosing to explore the environment. Therefore, it is more sampling efficient. No guarantee of convergence.

https://jonathan-hui.medium.com/rl-model-based-reinforcement-learning-3c2b6f0aa323

(Lec 8)
Recap: MDP/Reinforcement Learning

- Instead of asking for demos, we only request a single digit number $r_t$ to indicate the level of happiness - reward.

$$s_{t+1} \sim p \left( \cdot \mid s_t, a_t \right)$$

$$a_t \sim \pi \left( \cdot \mid s_t \right)$$

$$r_t \sim r \left( \cdot \mid s_t, a_t \right)$$

Here $p(s_{t+1} \mid s_t, a_t)$ is called the **model**.
How to get the model?

- Often we do know the dynamics
- Well-studied systems, e.g., automotive
- Optimal control

\[ \dot{x} = f(x, u, t) \quad \dot{x} = Ax + Bu \]

\[ y = g(x, u, t) \quad y = Cx + Du \]

\[
\frac{d}{dt} s_1 = \begin{bmatrix} \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{2C_\alpha}{m_\psi} & 0 \\ 0 & \frac{2C_\alpha}{m_\psi} \end{bmatrix} \begin{bmatrix} y \\ \psi \end{bmatrix} + \begin{bmatrix} 0 \\ -\dot{\psi} + \frac{8C_\alpha^2 (t - l_f)}{m_\psi^2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_\alpha}{m_\psi} \end{bmatrix} [F]
\]

\[
\frac{d}{dt} s_2 = \begin{bmatrix} \dot{x} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{m_\psi} \end{bmatrix} [\delta] + \begin{bmatrix} 0 \\ \psi \dot{\psi} - fg \end{bmatrix}
\]

Figure 1: Bicycle model[2]
Where to get the model?

- Often we do know the dynamics
  - Well-studied systems, e.g., automotive
  - Optimal control
- We know the structure of the dynamics but need to fit some parameters
  - System identification: fit unknown parameters of a known model structure, e.g., estimation of the road friction, abrupt changes
  - Adaptive control: the model may not be accurately estimated but the control error vanishes

Where to get the model?

- **We do know the dynamics**
  - Well-studied systems, e.g., automotive
  - Optimal control

- **We know the structure of the dynamics but need to fit some parameters**
  - System identification – fit unknown parameters of a known model, e.g. estimation of the road friction, abrupt changes
  - Adaptive control: the model may not be accurate but the control error vanishes

- **We can learn the dynamics**
  - Model-based reinforcement learning: Fit a general-purpose model for
    \[ p(s_{t+1} \mid s_t, a_t) \]
Aside: notation

\[ s_t - \text{state} \]
\[ a_t - \text{action} \]
\[ r(s, a) - \text{reward function} \]

\[ r(s, a) = -c(x, u) \]

\[ x_t - \text{state} \]
\[ u_t - \text{action} \]
\[ c(x, u) - \text{cost function} \]
(Finite Horizon Discrete-Time) Linear Quadratic Regulator (LQR)

• Design control policy to minimize the cost function.

\[ J_{0,N} = \frac{1}{2} x(N)^T S_N x(N) + \frac{1}{2} \sum_{k=0}^{N-1} (x(k)^T Q x(k) + u(k)^T R u(k)) \]

where \( S_N, Q, R \geq 0 \), subject to the system dynamics

\[ x(k + 1) = A x(k) + B u(k) \]

• It is found that the optimal control solution follows an elegant format

\[ u^*(k) = K_k x(k) \]

\[ \min J_{k,N} = J_{k,N}^* = \frac{1}{2} x(k)^T S_k x(k) \]

• where \( K_k \) is a constant only dependent on \( A, B, S, Q, R \),

\[ S_N \Rightarrow K_{N-1} \Rightarrow S_{N-1} \Rightarrow K_{N-2} \Rightarrow S_{N-2} \Rightarrow \ldots \Rightarrow S_0 ( = J_{0,N}^* ) \]

\[ K_k = -(R + B^T S_{k+1} B)^{-1} B^T S_{k+1} A, S_k = (A + B K_k)^T S_{k+1} (A + B K_k) + Q + K_k^T R K_k \]
(FH-DT) LQR vs MPC

- (Linear) Modal Predictive Control or "Receding Horizon Control"
- Calculate $u^*(k : k + N)$, but only use $u^*(k)$ and recalculate $u^*(k + 1 : k + N + 1)$ in the next step. Essentially, it is a closed loop version of LQR, therefore, it could be more robust by increasing computation budget.
Atlas uses its whole body -- legs, arms, torso -- to perform a sequence of dynamic maneuvers that form a gymnastic routine. We created the maneuvers using new techniques that streamline the development process. First, an optimization algorithm transforms high-level descriptions of each maneuver into dynamically-feasible reference motions. Then Atlas tracks the motions using a model predictive controller that smoothly blends from one maneuver to the next. Using this approach, we developed the routine significantly faster than previous Atlas routines, with a performance success rate of about 80%.
iterative LQR (iLQR)

https://www.youtube.com/watch?v=anIsw2-Lbco
iterative LQR (iLQR)

- Approximate a nonlinear system as a linear-quadratic system at $\tilde{x}_t$, $\tilde{u}_t$ with Taylor expansion

$\begin{align*}
x_{t+1} &= f(x_t, u_t) \approx f(\tilde{x}_t, \tilde{u}_t) + \nabla_{x_t,u_t} f(\tilde{x}_t, \tilde{u}_t) \left[ x_t - \tilde{x}_t \right] \\
c(x_t, u_t) &\approx c(\tilde{x}_t, \tilde{u}_t) + \nabla_{x_t,u_t} c(\tilde{x}_t, \tilde{u}_t) \left[ x_t - \tilde{x}_t \right] + \frac{1}{2} \left[ x_t - \tilde{x}_t \right]^T \nabla^2_{x_t,u_t} c(\tilde{x}_t, \tilde{u}_t) \left[ x_t - \tilde{x}_t \right] \\
\end{align*}$

$\delta x_t = x_t - \tilde{x}_t$,  $\delta x_{t+1} = f(x_t, u_t) - f(\tilde{x}_t, \tilde{u}_t)$

$\delta u_t = u_t - \tilde{u}_t$

- Run LQR with state $\delta x_t$ and action $\delta u_t$. Then rerun the linearization to update the model.
Case study: nonlinear model-predictive control with iLQR

Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization

Yuval Tassa, Tom Erez and Emanuel Todorov
University of Washington

every time step:

observe the state $x_t$

use iLQR to plan $u_t, \ldots, u_T$ to minimize $\sum_{t'=t}^{t+T} c(x_{t'}, u_{t'})$

execute action $u_t$, discard $u_{t+1}, \ldots, u_{t+T}$
Model-based Reinforcement Learning

Sensing
\[ \mathcal{D} = \{s_t, a_t\}_i \]

Environment

Modeling
Estimate \( p(s_{t+1} | s_t, a_t) \)
Supervised learning/ regression

Action
Optimize \( a_t \sim \pi_\theta( \cdot | s_t) \)
Backpropogation with \( p \)
What kind of models can we learn?

**Neural networks**

\[ s_{t+1} = f_\theta(s_t, a_t) \]

Pro: very expressive, can take the advantage of rich data
Con: not so good in low data regimes/rare events, lack of interpretation

**Stochastic functions (Gaussian Processes)**

\[ s_{t+1} \sim \mathcal{N}(\cdot | s_t, a_t, \mathcal{D}) \]

Pro: data efficient
Con: hard to model non-smooth dynamics, slower than NN when dataset is big

**Hierarchical /modular structures**

Pro: good interpretation, data efficient
Con: hard to train

Nonparametric
Reinforcement Learning - NN model-based

1. Run base policy $\pi_{\theta(0)}(a_t | s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s_t, a_t, s_{t+1})_{t=1:D}\}_k$

2. Learn model $f_{\phi(i)}$ by minimizing $\sum_t \|f_{\phi(i)}(s_t, a_t) - s_{t+1}\|^2$

3. Optimize $\pi_{\theta(k)}(a_t | s_t)$ using $f_{\phi(i)}$ via backpropagate

4. Execute with policy $\pi_{\theta(k)}(a_t | s_t)$, append new data $\mathcal{D}_{k+1}$ to $\mathcal{D}$

MB-NN-RL-1.0

Issue: we may over-rely on the model, which could have safety issues.

Planning helps to make the model more trustworthy
1. Run base policy $\pi_{\theta^{(0)}}(a_t | s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s_t, a_t, s_{t+1})_{t=1:D}\}_k$ 
2. Learn model $f_{\phi^{(i)}}$ by minimizing $\sum_t \|f_{\phi^{(i)}}(s_t, a_t) - s_{t+1}\|^2$ 
3. Plan through $f_{\phi^{(i)}}(s_t, a_t)$ to choose actions 
4. Execute the first planned action, observe results states $s_{t+1}$ 
5. Append $(s_t, a_t, s_{t+1})$ to $\mathcal{D}$
How to do planning (for multi-steps)?

- Planning with linearized models (local model)
  - i-LQR
- Planning with sampling based methods
  - CEM, PETS
Case study: local models and iLQR

Learning Contact-Rich Manipulation Skills with Guided Policy Search

Sergey Levine, Nolan Wajener, Pieter Abbeel

Abstract— Autonomous learning of object manipulation skills can enable robots to acquire rich behavioral repertoires that scale to the variety of objects found in the real world. However, current motion skill learning methods typically restrict the behavior to a compact, low-dimensional representation, limiting its expressiveness and generality. In this paper, we extend a recently developed policy search method [1] and use it to learn a range of dynamic manipulation behaviors with highly general policy representations, without using known models or example demonstrations. Our approach learns a set of trajectories for the desired motion skill by iteratively refining time-varying linear models, and then refines those trajectories into a single control policy that can generalize to new situations. To enable this method to run on a real robot, we introduce several improvements that reduce the sample count and automate parameter selection. We show that our method can acquire manipulation policies for challenging tasks, including placing an object on a peg, stacking light-fitting lego bricks, placing wooden rings onto tight-fitting gaps, inserting a shoe into a shoe, and serving bottle caps onto bottles.

I. INTRODUCTION

Autonomous acquisition of manipulation skills has the potential to dramatically improve the ease of deployment of robotic platforms, in domains ranging from manufacturing to household robotics, and the fluency and speed of the robot’s motion. It is often much easier to specify what a robot should do, by means of a compact cost function, than
Cross Entropy Method (Random Shooting)

Optimal planning:

\[ a_1, \ldots, a_T = \arg \max J(a_1, \ldots, a_T), A = \arg \max J(A) \]

Simplest method: randomly sample and pick the top actions

1. Pick \( A_1, \ldots, A_N \) from some distribution (e.g., uniform)

2. Choose \( A_i \) based on \( \arg \max J(A) \)
Case study: CEM with MPC

Algorithm 1 Our model-based MPC algorithm ‘PETS’:

1: Initialize data $\mathbb{D}$ with a random controller for one trial.
2: for Trial $k = 1$ to $K$ do
3:     Train a PE dynamics model $\tilde{f}$ given $\mathbb{D}$.
4:     for Time $t = 0$ to TaskHorizon do
5:         for Actions sampled $a_{t:t+T} \sim \text{CEM}(\cdot)$, 1 to NSamples do
6:             Propagate state particles $s^p_t$ using TS and $\tilde{f}\{\mathbb{D}, a_{t:t+T}\}$.
7:             Evaluate actions as $\sum_{t'=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} r(s^p_{t'}, a_{t'})$.
8:             Update CEM($\cdot$) distribution.
9:         Execute first action $a^*_t$ (only) from optimal actions $a^*_{t:t+T}$.
10: Record outcome: $\mathbb{D} \leftarrow \mathbb{D} \cup \{s_t, a^*_t, s_{t+1}\}$. 

PE: Probabilistic Ensembles

TS: Trajectory Sampling
Case study: planning with CEM
Safe RL with non-stationary environment (a shaking head)
What kind of models can we learn?

**Neural networks**

\[ s_{t+1} = f_\theta(s_t, a_t) \]

- Pro: very expressive, can take the advantage of rich data
- Con: not so good in low data regimes/rare events, lack of interpretation

**Stochastic functions (Gaussian Processes)**

\[ s_{t+1} \sim \mathcal{N}(\cdot | s_t, a_t, \mathcal{D}) \]

- Pro: data efficient
- Con: hard to model non-smooth dynamics, slower than NN when dataset is big

**Hierarchical /modular structures**

- Pro: good interpretation, data efficient
- Con: hard to train

**Nonparametric**
Neural network vs Gaussian processes

• Neural network
  • very powerful to approximate nonlinear functions
  • Efficient training
  • overfitting issues

• Gaussian processes
  • approximate nonlinear functions
  • provide sensible uncertainties
  • a probability distribution upon a set of functions
  • adjust complexity with data size: nonparametric
  • may suffer from the curse of dimension
A Visual Exploration of Gaussian Processes

How to turn a collection of small building blocks into a versatile tool for solving regression problems.

Regression is used to find a function (line) that represents a set of data points as closely as possible.

A Gaussian process is a probabilistic method that gives a confidence (shaded) for the predicted function.
Effect of model errors and benefit of GP

• The main reason why model-based RL are not widely used in real-world application is that they can suffer severely from model errors, i.e., they inherently assume that the learned model resembles the real environment sufficiently accurately.

• Given a small data set of observed transitions (left), multiple transition functions plausibly could have generated them (center).

![Diagram showing effect of model errors and multiple plausible transition functions.](image-url)

Fig. 1. Effect of model errors. Left: Small data set of observed transitions from an idealized one-dimensional representations of states and actions \((x_t, u_t)\) to the next state \(x_{t+1} = f(x_t, u_t)\). Center: Multiple plausible deterministic models. Right: Probabilistic model. The probabilistic model describes the uncertainty about the latent function by a probability distribution on the set of all plausible transition functions. Predictions with deterministic models are claimed with full confidence, while the probabilistic model expresses its predictive uncertainty by a probability distribution.
Effect of model errors and benefit of GP

- Fig. 1. Effect of model errors. Left: Small data set of observed transitions from an idealized one-dimensional representations of states and actions \((x_t, u_t)\) to the next state \(x_{t+1} = f(x_t, u_t)\). Center: Multiple plausible deterministic models. Right: Probabilistic model. The probabilistic model describes the uncertainty about the latent function by a probability distribution on the set of all plausible transition functions. Predictions with deterministic models are claimed with full confidence, while the probabilistic model expresses its predictive uncertainty by a probability distribution.

- Choosing a single deterministic model has severe consequences: Long-term predictions often leave the range of the training data in which case the predictions become essentially arbitrary. However, the deterministic model claims them with full confidence! By contrast, a probabilistic model places a posterior distribution on plausible transition functions (right) and expresses the level of uncertainty about the model itself.
PILCO algorithm

Algorithm 1 PILCO

1: **init**: Sample controller parameters $\theta \sim N(0, I)$. Apply random control signals and record data.
2: **repeat**
3: Learn probabilistic (GP) dynamics model, see Sec. 3.1, using all data
4: **repeat**
5: Approximate inference for policy evaluation, see Sec. 3.2: get $J^\pi(\theta)$, Eq. (9)–(11)
6: Gradient-based policy improvement, see Sec. 3.3: get $\nabla J^\pi(\theta) / \nabla\theta$, Eq. (12)–(16)
7: Update parameters $\theta$ (e.g., CG or L-BFGS).
8: **until** convergence; **return** $\theta^*$
9: Set $\pi^* \leftarrow \pi(\theta^*)$
10: **until** $\pi^*$ to system and record data
11: **until** task learned

PILCO: Design policy to minimize the cost function $J^\pi(\theta) = \sum_{t=0}^{T} \mathbb{E}_{x_t}[c(x_t)], \quad x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ subject to the unknown system dynamics $f$ and noise $w$: $x_{t+1} = f(x_t, u_t) + w, \quad w \sim \mathcal{N}(0, \Sigma_w)$

- Model Learning: we use tuples $(x_t, u_t) \in \mathbb{R}^{D+F}$ denoted as $\tilde{x} := [x^\top u^\top]^\top$ as training inputs and differences $\Delta_t = x_{t+1} - x_t \in \mathbb{R}^D$ as training outputs (targets). The posterior GP is a one-step prediction model, and the predicted successor state $x_{t+1}$ is Gaussian distributed

$$p \left( x_{t+1} \mid x_t, u_t \right) = \mathcal{N} \left( x_{t+1} \mid \mu_{t+1}, \Sigma_{t+1} \right), \quad \mu_{t+1} = x_t + \mathbb{E}_f[\Delta_t], \quad \Sigma_{t+1} = \text{var}_f[\Delta_t]$$

where the mean and variance of the GP prediction are

$$\mathbb{E}_f[\Delta_t] = m_f(\tilde{x}_t) = k_{\ast}^\top \left( K + \sigma_w^2 I \right)^{-1} y, \quad \text{var}_f[\Delta_t] = k_{\ast\ast} - k_{\ast}^\top \left( K + \sigma_w^2 I \right)^{-1} k_{\ast}$$

respectively, with $k_{\ast} := k(\tilde{X}, \tilde{x}_t)$, $k_{\ast\ast} := k(\tilde{x}_t, \tilde{x}_t)$, $\tilde{X} = [\tilde{x}_1, \ldots, \tilde{x}_n]$, $y = [\Delta_1, \ldots, \Delta_n]^\top$, $K$ is the kernel matrix with entries $K_{ij} = k(\tilde{x}_i, \tilde{x}_j)$.

- Kernel: a positive semidefinite covariance function

$$k(\tilde{x}_p, \tilde{x}_q) = \sigma_f^2 \exp \left( -\frac{1}{2} (\tilde{x}_p - \tilde{x}_q)^\top \Lambda^{-1} (\tilde{x}_p - \tilde{x}_q) \right) + \delta_{pq} \sigma_w^2$$

With parameters length-scales $\ell_i$, signal variance $\sigma_f^2$, and noise variance $\sigma_w^2$ learned by max likelihood
GP-based RL in a real-world application
Challenges in model learning

- Under-fitting: If the model class is restricted (e.g., linear function or gaussian process) we have under-modeling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth. As a result, though we learn faster than model free in the beginning, MBRL ends up having worse asymptotic performance than model-free methods, that do not suffer from model bias.

- Over-fitting: If the model class is very expressive (e.g., neural networks) the model will overfit, especially in the beginning of training, where we have very few samples.

- Uncertainty/errors propagated and amplified through planning
Model-based vs Model free

- **Model-based**
  + data efficient in training
  + Possibility to transfer across tasks
  + Increase interoperability
  - Do not optimize directly over performance
    - Usually need domain knowledge (overfitting/under-fitting)
    - Maybe hard to learn policy

- **Model-free**
  + Need little assumption
  + Efficient for learning complex policy
  - Require a lot training data
  - Not transferable and lack of interoperability

Widely used in safety-critical applications
Combine model-based and model-free
Popular RL algorithms

- Policy Optimization
  - Policy Gradient: A2C / A3C, PPO, TRPO
  - Off-Policy: DDPG, TD3, SAC

- Q-Learning
  - DQN, C51, QR-DQN, HER

- Learn the Model
  - World Models: I2A, MBMF, MBVE

- Given the Model
  - AlphaZero

Summary

• Model-based RL

• Choose the format of the models

• Learn $p(s_{t+1} \mid s_t, a_t)$ with new data and pre-defined models

• Replanning at each step (iLQR or CEM)
Worth reading


• a github repo https://github.com/anassinator/ilqr

• A Visual Exploration of Gaussian Processes
  • https://distill.pub/2019/visual-exploration-gaussian-processes/