Lecture #7: Model-Based Reinforcement Learning, Advanced Model Learning and Images

Lee Yong Ler, Lin Qian
Department of Computer Science
National University of Singapore
Singapore, S117417

1 Introduction

In reinforcement learning RL, we maximize the rewards for our actions. By simply looking at the equation below, rewards depend on the policy and the system dynamics (model).

Model-based reinforcement learning requires less samples and allow the agent to plan by thinking ahead. However, it involves complex optimization methods and more assumptions and approximations.

2 Summary of Model-free RL vs Model-based RL

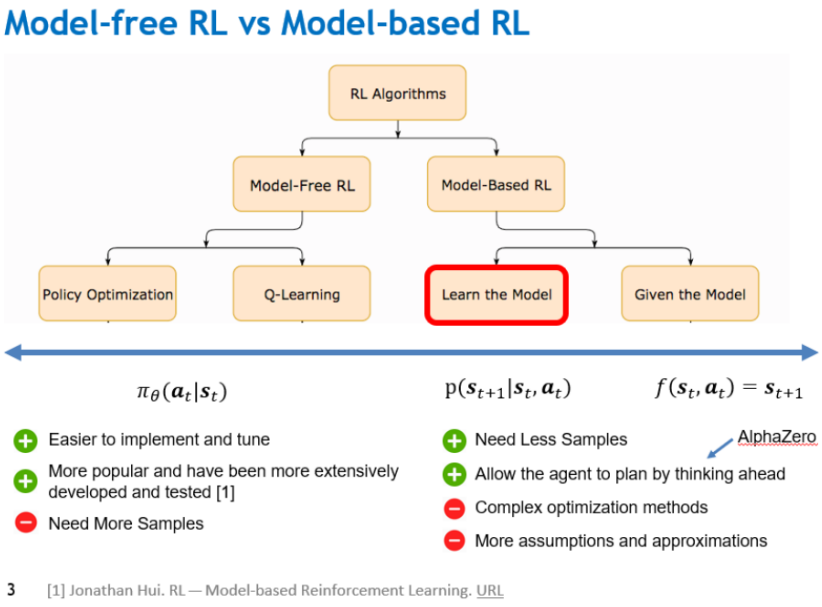


Figure 1: Overview: Model-free RL vs Model-based RL

DAgger First you collect data with initial policy then you get more data then you obtain another initial policy.

– To try to mitigate the variation when you do not follow the same trajectory

Version 1.5 Instead of doing all actions, we execute the first action and observe the resulting state and re-plan.

- Planning not cheap

Version 2.0

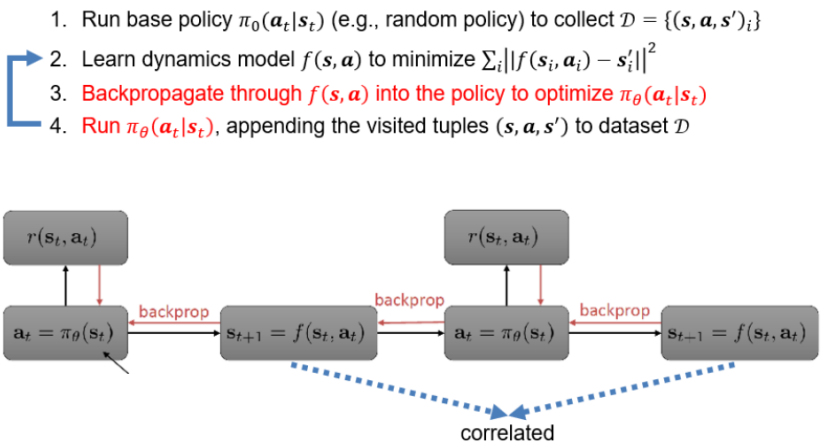


Figure 2: Version 2.0 Typo in image, a_{t+1} at the third step instead of a_t in between correlated states

Backpropagate to optimize

- Face exploding or vanishing gradient problem due to correlation

3 System Dynamics representation models

3.1 Gaussian (~Model-based 2.0)

Gaussian: learn Gaussian Process model probability $p(s'|s, a)$ to maximize the sum of probability

- Perform BP on probability to optimize the policy
- Performance: data-efficient, not great with non-smooth dynamics

3.2 Neural Network

Input: current status and action
Output: next status
Euclidean training loss (complex) corresponds to Gaussian probability $p(s'|s, a)$
Performance: Very expressive (data-consuming), not great when data size is small

3.3 Gaussian Mixture Model

3.4 Problem with global models

- Big neural network or GP
- Planner will go to places where model erroneously thinks things are better than what they really are
- Generally hard to find a good model converge to an accurate policy in most of the state space
- he model could be too complex comparing with policy

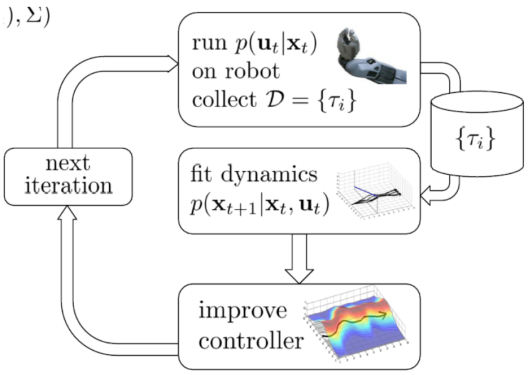


Figure 3: Overview: Local models

3.5 Local models

Q: Why do we add a Gaussian noise?

- If deterministic, you won't be able to find a correlation of current state with your next state. With noise you would be able to find a link between the current state and the next state.
- Without noise, all of the state would be at the same point. With noise, you would be able to do a linear regression on the same point. (can get a piecewise linear approximation of the function)
- Will be able to see more next state.

Problem

- What controller to execute
- How to stay close to old controller

3.6 Version 2.0 Controller

Version 2.0: $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$

Set $\Sigma_t = \mathbf{Q}^{-1}_{\mathbf{u}_t,\mathbf{u}_t}$

$Q(\mathbf{x}_t, \mathbf{u}_t)$ is the cost to go: total cost we get after taking an action

Standard LQR solves $\min \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t)$

Linear-Gaussian solution solves $\min \sum_{t=1}^{T} E_{(\mathbf{x}_t,\mathbf{u}_t)\sim p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t, \mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$

This is the *maximum entropy* solution: act as randomly as possible while minimizing cost

Injecting more noise when the action is not that important, less when it matters.

Q: Are we adding noise to two places, in the transition and the policy?

- Yes. Transition noise is to help to you get whether this action matters.
- Expectation under $x(t)$ of the expectation under $u(t)|x(t)$ is the same
- Expectation under $x(t)$ with probability theory

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}\left[-\log \bar{p}(\mathbf{u}_t|\mathbf{x}_t)\right] + E_{p(\mathbf{x}_t)}[\underbrace{E_{p(\mathbf{u}_t|\mathbf{x}_t)}[\log p(\mathbf{u}_t|\mathbf{x}_t)]}_{\text{negative entropy}}]$$

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}\left[-\log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))\right]$$

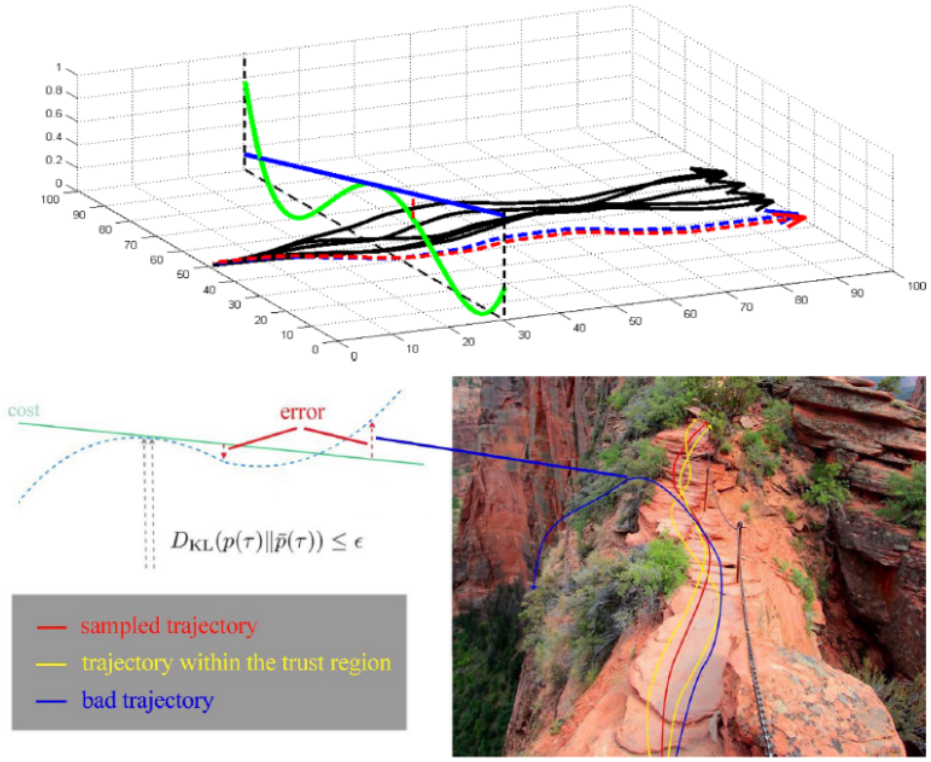


Figure 4: If we stray too far from old controller, may veer too far from the trajectory.

3.7 How to stay close to old controller

Use KL divergence to find trust region

$$\min_{p} \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t)] \text{ s.t. } \underbrace{D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon}_{\text{trust region}}$$

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}\left[-\log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))\right]$$

Dual Gradient Descent

- Transform the objective into a Lagrange dual function which can be optimized iteratively.

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$g(\lambda) = \mathcal{L}(\mathbf{x}^\star(\lambda), \lambda)$$

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$$

$$\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

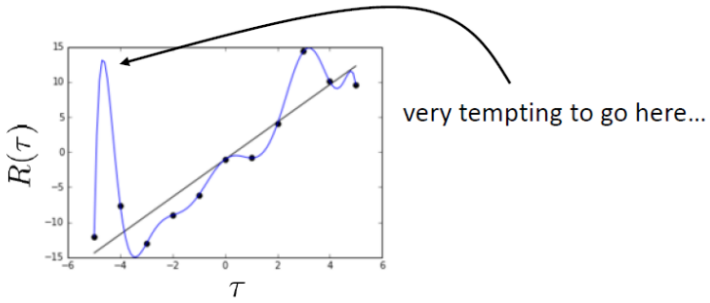
1. Find $\mathbf{x}^\star \leftarrow \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$
2. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

Figure 5: Dual gradient descent

4 Managing overfitting in model-based RL

Problem: Performance gap (pure model-based vs model-free) Example: model based RL version 1.5

- Trapped by a sudden increasing on reward



How to get uncertainty-aware model?

1. Use output entropy from a NN which output probability of next state given the current state and action
   (a) Will try to fit noise with data generation but there is still uncertainty cos of your model (there are two type of uncertainty statistical uncertainty and model uncertainty)
2. Estimation of $p(\theta|D)$ where θ are the parameters

Bootstrap Ensemble

- Averaging of different network trained on a dataset made with sample with replacement
- Normal bootstrap used in machine learning
- In practice do not need resampling cos initialization of NN makes models independent anyway

Q: How does dropout differ from bootstrapping ?

- Variance from different model low, we can trust the models but if variance really high, we would not trust any of the model. Drop out would not allow us to know the variance.

Algorithm to evaluate models (Figure 6)
Run step 1-3 in k neural network and average at step 4 so that you will know how good this action sequence is.

5 Usage of images in Model-based RL (Learning latent space)

1. Use bottleneck of autoencoder and feed into policy to output action

In general, for candidate action sequence $\mathbf{a}_1, \ldots, \mathbf{a}_H$:

Step 1: sample $\theta \sim p(\theta|\mathcal{D})$

Step 2: at each time step t, sample $\mathbf{s}_{t+1} \sim p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

Step 3: calculate $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

Step 4: repeat steps 1 to 3 and accumulate the average reward

Figure 6: Evaluation steps

2. Learning embedding then learn latent space
3. Learning both embedding and image together (learning interested version of image only)
4. Directly learning the next observation using another model

Q: What is spatial *softmax*?

- *Softmax* across the different channels
- Tensorflow definition compute the *softmax* for each channel, get three pairs of (x, y), location of them.

References