Model Based RL

Aaron Lou

June 2018

1 Introduction

Last lecture covered various Model Based Optimization and Planning methods. However, if the model dynamics are not known, we have two possible methods for learning the dynamics: global and local.

Note that previous methods such as Q-learning and policy gradient are off-model, since we don't actually use the model and instead just sample.

Once we learn these ways, we can use the previous methods (monte carlo tree search, iLQR) to find an optimal policy. In particular, we need to find

$$\frac{df}{dx_t}, \frac{df}{du_t}$$

2 Global RL Algorithms

We can fit a global rl algorithms. Our first model is quite simple.

Result: The optimal policy through Model Based RL
1. run base policy $\pi_0(a_t \mid s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$.
2. learn dynamics model $f(s, a)$ through minimizing $\sum_i ||f(s_i, a_i) - s_i'||^2$.
3. Plan using $f(s, a)$ to choose actions.

Algorithm 1: Model-based RL ver 0.5

This sometimes works. However, it is hard to design a good base policy. It's useful in the sense that we can learn some model parameters, such as some physics. However, in general, this exacerbates the distribution mismatch problem as we use expressive models.

We can improve this model by collecting data (DAgger) using π_f ie the policy we make using the transition dynamics. It is given as

Result: The optimal policy through Model Based RL
1. run base policy $\pi_0(a_t \mid s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$.
while *until convergence* **do**
  2. learn dynamics model $f(s, a)$ through minimizing $\sum_i ||f(s_i, a_i) - s'_i||^2$.
  3. plan policy through $f(s, a)$.
  4. execute policy and add $\{(x, u, x')_i\}$ to $\mathcal{D}$.
end

Algorithm 2: Model-based RL ver 1.0

If we make a mistake, we can replan. This gives us a new model using Model Predictive Control (MPC). It is enumearted as follows

Result: The optimal policy through Model Based RL
1. run base policy $\pi_0(a_t \mid s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$.
while *every N steps* **do**
  2. learn dynamics model $f(s, a)$ through minimizing $\sum_i ||f(s_i, a_i) - s'_i||^2$.
  while *for some steps* **do**
    3. plan policy through $f(s, a)$.
    4. execute the first planned action and observe resulting state s' MPC.
    5. append (s, a, s') to dataset $\mathcal{D}$.
  end
end

Algorithm 3: Model-based Rl ver 1.5

However, this is a lot of calculations. If we could backpropogate directly into the policy, that might be more effective. This is possible because the structure of the data is like and RNN, with s_{t+1} and $r(s_t, a_t)$ being able to propagate into it. The algorithm is given as

Result: The optimal policy through Model Based RL
1. run base policy $\pi_0(a_t \mid s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$.
while *until convergence* **do**
  2. learn dynamics model $f(s, a)$ through minimizing $\sum_i ||f(s_i, a_i) - s'_i||^2$.
  3. propagate through $f(s, a)$ into the policy to optimize $\pi_\theta(a_t \mid s_t)$.
  4. run $\pi_\theta(a_t \mid s_t)$ and append the visited tuples (s, a, s') to $\mathcal{D}$.
end

Algorithm 4: Model-based Rl ver 2.0

A summary of pros and cons is given below

- Version 0.5: collect random samples, train dynamics, plan.
  - + simple, no iterative procedure.
  - − distribution mismatch problem.
- Version 1.0: iteratively collect data, replan, collect data.
  - + simple, solved distribution mismatch problem.
  - − open loop plan might perform poorly, especially in stochastic domains.
- Version 1.5: iteratively collect data using MPC (Model Predictive Control).
  - + robust to small errors.
  - − computationally expensive.
- Version 2.0: backpropogate directly into policy.
  - + computationally cheap.
  - − can be numerically unstable, especially in stochastic domains.

The model that we choose for our model. They are as follows

1. Gaussian Process: take in (s, a) and output s'.
   - + very data-efficient.
   - − not great with non-smooth dynamics and get's slower O(n).
2. Neural Network: take in (s, a) and output s'.

   The Euclidean Training loss corresponds with to Gaussian $p(s' \mid s, a)$. More complex losses output parameters of Gaussian mixture.
   - + very expressive, and use lots of data
   - − not great in low data areas.
3. GMM: train to get $p(s'|s, a)$ over tuples (s, a, s').

3 Local RL Algorithms

One idea could be to just first $\frac{df}{dx_t}, \frac{df}{du_t}$ around the current trajectory or policy. LQR gives us the controller. The general idea is

Result: The optimal controller
while *until convergence* **do**
1. run $p(a_t x_t)$ on robot and collect $\mathcal{D} = \{\tau_i\}$.
2. fit dynamics $p(x_{t+1} \mid x_t, u_t)$.
3. improve the controller.

end

If we have a stochastic controller $p(x_{t+1} \mid x_t, u_t) = \mathcal{N}(f(x_t, u_t), \Sigma)$. If we approximate f as a linear function, we would have $f(x_t, u_t) \approx A_t x_t + B_t u_t$ and

$$A_t = \frac{df}{dx_t} \quad B_t = \frac{df}{du_t}$$

3.1 How to fit dynamics

Version 1.0: fit $p(x_{t+1} \mid x_t, u_t)$ at each time using linear regression.

$$p(x_{t+1} \mid x_t, u_t) = \mathcal{N}(A_t x_t + B_t u_t + c, N_t)$$

Version 2.0: fit $p(x_{t+1} \mid x_t, u_t)$ using Bayesian Linear Regression. In particular, use a global model as a prior.

3.2 How to improve controller

Using iLQR, we can update the controller. iLQR produces $\hat{x}_t, \hat{u}_t, K_t, k_t$. Where

$$u_t = K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t$$

is the action we'll take. Some algorithms are given as

- Version 0.5: $p(u_t \mid x_t) = \delta(u_t = \hat{u}_t)$. But, this doesn't correct deviations.
- Version 1.0: $p(u_t \mid x_t) = \delta(u_t = K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t)$. This is too good and will not have any deviation.
- Version 2.0: $p(u_t \mid x_t) = \mathcal{N}(K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t, \Sigma_t)$, where we add noise.

For version 2.0, we can set the noise $\Sigma_t = Q^{-1}_{u_t, u_t}$. The intuition is based on the equation

$$Q(x_t, u_t) = \text{const} + \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T Q_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T q_t$$

and we see that Q_{u_t,u_t} is inversely correlated with sensitivity. This Linear-Gaussian solution solves

$$\min \sum_{t=1}^{T} E_{(x_t,u_t)\sim p(x_t,u_t)}[c(x_t,u_t) - \mathcal{H}(p(u_t \mid x_t))]$$

and this is the maximizing entropy solution: act randomly while minimizing cost. However we need to keep our controller close to the old controller, or else it's not a local model. In particular, given

$$p(u_t \mid x_t) = \mathcal{N}(K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t, \Sigma_t)$$
$$p(\tau) = p(x_1) \prod_{t=1}^{T} p(u_t \mid x_t) p(x_{t+1} \mid x_t, u_t)$$

If the trajectory distribution $p(\tau)$ is close to the old $\overline{p}(\tau)$, then so will the dynamics be. We use KL convergence

$$D_{KL}(p(\tau) \mid\mid \overline{p}(\tau)) < \epsilon$$

The KL-divergence is given by

$$D_{KL}(p(\tau) \mid\mid \overline{p}(\tau)) = E_{p(\tau)}[\log p(\tau) - \log \overline{p}(\tau)]$$
$$p(\tau) = p(x_1) \prod_{t=1}^{T} p(u_t \mid x_t) p(x_{t+1} \mid x_t, u_t)$$
$$\overline{p}(\tau) = p(x_1) \prod_{t=1}^{T} \overline{p}(u_t \mid x_t) p(x_{t+1} \mid x_t, u_t)$$

since the dynamics and initial states are the same, so we can calculate

$$\log p(\tau) - \log \overline{p}(\tau) = \sum_{t=1^T} \log p(u_t \mid x_t) - \log \overline{p}(u_t \mid x_t)$$

and so the KL-divergence is given by

$$D_{KL}(p(\tau) \mid\mid \overline{p}(\tau)) = \sum_{t=1}^{T} E_{p(x_t,u_t)}[\log p(u_t \mid x_t) - \log \overline{p}(u_t \mid x_t)]$$
$$= \sum_{t=1}^{T} E_{p(x_t,u_t)}[-\log \overline{p}(u_t \mid x_t) - \mathcal{H}(p(u_t \mid x_t))]$$

since we note that

$$\log p(u_t \mid x_t) = E_{p(x_t)}[E_{p(u_t|x_t)}[\log p(u_t \mid x_t)] = \mathcal{H}(p(u_t \mid x_t))$$

This is really similar to Linear-Gaussian, as that solves

$$\min \sum_{t=1}^{T} E_{p(x_t,u_t)}[c(x_t,u_t) - \mathcal{H}(p(u_t \mid x_t))]$$

And so we want to get D_{KL} into the cost using iLQR with the constraint $D_{KL}(p(\tau) \mid\mid \overline{p}(\tau)) \leq \epsilon$.

A small aside for Dual Gradient Descent. Notice if we wish to find $\min_x f(x)$ s.t. $C(x) = 0$. We can calculate

$$\mathcal{L}(x,\lambda) = f(x) + \lambda C(x) \quad g(\lambda) = \inf_x \mathcal{L}(x,\lambda) \quad \lambda \leftarrow \arg\max_\lambda g(\lambda)$$

and use gradient ascent. However, if we are given a $x^\star$ s.t. $x^\star = \arg\min_x \mathcal{L}(x,\lambda)$ and have

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

The full algorithm is given by

while *until convergence* **do**
1. Find $x^\star \leftarrow \arg\min_x \mathcal{L}(x,\lambda)$.
2. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(x^\star,\lambda)$.
3. $\lambda \leftarrow \lambda + \alpha\frac{dg}{d\lambda}$

end

Algorithm 5: Dual Gradient Descent (Ascent)

The constrained optimization problems is what we want to solve for Dual Gradient Descent (DGD) with iLQR. It is given by

$$\min_p \sum_{t=1}^{T} E_{p(x_t,u_t)}[c(x_t,u_t)] \quad D_{KL}(p(\tau) \mid\mid \overline{p}(\tau)) \leq \epsilon$$

$$D_{KL}(p(\tau) \mid\mid \overline{p}(\tau)) = \sum_{t=1}^{T} E_{p(x_t,u_t)}[-\log \overline{p}(u_t \mid x_t) - \mathcal{H}(p(u_t \mid x_t))]$$

$$\mathcal{L}(p,\lambda) = \sum_{t=1}^{T} E_{p(x_t,u_t)}[c(x_t,u_t) - \lambda \log \overline{p}(u_t \mid x_t) - \lambda\mathcal{H}(p(u_t \mid x_t))] - \lambda_\epsilon$$

We find $p^\star \leftarrow \arg\min_p \mathcal{L}(p,\lambda)$ and follow the Dual Gradient Descent algorithm. Using LQR, we can just use $\tilde{c}(x_t,u_t) = \frac{1}{\lambda}c(x_t,u_t) - \log \overline{p}(u_t \mid x_t)$. The full algorithm is given below

while *until convergence* **do**
  1. Set $\tilde{c}(x_t, u_t) = \frac{1}{\lambda} c(x_t, u_t) - \log \overline{p}(u_t \mid x_t)$.
  2. Use LQR to find $p^\star(u_t \mid x_t)$ using $\tilde{c}$. 3.
  $\lambda \leftarrow \lambda + \alpha(D_{KL}(p(\tau) \mid\mid \overline{p}(\tau)) - \epsilon)$.

end

Algorithm 6: DGD with iLQR

This technique of using bounded KL-divergence is useful between two policies or controllers. Using this on policies is the same as trajectory distributions, and we'll use this for model-free RL.