Lecture 19

Model-Based Reinforcement Learning

We've spent a significant amount of time discussing methods for model-based and model-free policy search. Most methods for model-based reinforcement learning tend to be of the policy search variety. Instead of operating on real-world data, these methods often leverage a simulator (or model). Other methods leverage an approximate model and use structure in the model itself to compute gradients.

19.1. PILCO

PILCO (probabilistic inference for learning control) is a model-based reinforcement learning approach that attempts to learn a feedback policy by learning a Gaussian process model of the dynamics.

The overall approach is

- Compute dynamics model
- Compute long term predictions
- Policy Improvement
- Apply the controller

PILCO considers dynamics of the form

$$x_t = f(x_{t-1}, u_{t-1}) \tag{19.1}$$

where the policy is given as $\pi(x)$ and the policy minimizes

$$J^\pi = \sum_{t=0}^{T} \mathbb{E}_{x_t}[c(x_t)]. \tag{19.2}$$

To train the $\mathcal{GP}$, we use the training inputs $\tilde{x}_{t-1} = [x_{t-1}^T u_{t-1}^T]^T$ and the training outputs $\Delta_t = f(x_{t-1}) - x_{t-1} + \epsilon = x_t - x_{t-1} + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$ The $\mathcal{GP}$ is meant to provide one-step predictions

$$p(x_t | x_{t-1}, u_{t-1}) = \mathcal{N}(x_t | \mu_t, \Sigma_t). \tag{19.3}$$

The prior mean is $m = 0$, and the prior covariance function is

$$k(\tilde{x}, \tilde{x}') = \alpha^2 \exp(-\frac{1}{2}(\tilde{x} - \tilde{x}')^T \Lambda^{-1} (\tilde{x} - \tilde{x}')) \tag{19.4}$$

The Gaussian process hyperparameters can be learned as we discussed previously. Given some posterior predictive distribution for some set of test inputs $\tilde{x}_*$, we have

$$\mathbb{E}_f[\Delta_*] = k_*^T (K + \sigma_\epsilon^2 I)^{-1} y \tag{19.5}$$

and

$$\text{var}_f[\Delta_*] = k_{**} - k_*^T(K + \sigma_\epsilon^2 I)^{-1}k_* \tag{19.6}$$

where $k_* = k(\tilde{X}, \tilde{x}_*)$, $k_{**} = k(\tilde{x}_*, \tilde{x}_*)$, and y is a vector of training targets Δ_t.

PILCO trains conditionally independent $\mathcal{GP}$ for each target dimension.
Once the model is trained, we would then like to evaluate the policy J^π. To do this, we compute the following

$$\mathbb{E}[c(x_t)] = \int c(x_t)\mathcal{N}(x_t|\mu_t, \Sigma_t)dx_t, \forall t = 0...T \tag{19.7}$$

where $c(x_t)$ is the cost. To compute this, we must compute the cascaded one-step predictions $p(x_1), ..p(x_T)$. That is, we must compute $p(x_t)$ from $p(x_{t-1})$. The process to compute these probability distributions requires computing the following distributions in sequence:

$$p(x_{t-1}) \to p(u_{t-1}) \to p(x_{t-1}, u_{t-1}) \to p(\Delta_t) \to p(x_t). \tag{19.8}$$

To compute each one of these distributions, PILCO assumes

- Gaussian uncertainty distributions
- Special form of the policy $\pi(x_t)$

Deisenroth derives two special approximations using Gaussian Processes. Consider the case where $x_* \sim \mathcal{N}(\mu, \Sigma)$ is passed through a nonlinear function. The predictive distribution is given as

$$p(h(x_*)|\mu, \Sigma) = \int p(h(x_*)|x_*)p(x_*|\mu, \Sigma)dx_*. \tag{19.9}$$

In general, this can not be computed analytically, nor is it in general Gaussian. For a $\mathcal{GP}$, if we are willing to approximate the output distribution as a Gaussian distribution, we can exactly fit the mean and variance (moment matching) of the output distribution for a $\mathcal{GP}$ using the law of iterated expectations (Fubini's theorem).

For a $\mathcal{GP}$, we can also compute Σ_{xh} using the same process, which is given as

$$p(x_*, h(x_*)|\mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix} \begin{bmatrix} \Sigma & \Sigma_{xh} \\ \Sigma_{xh}^T & \Sigma_* \end{bmatrix} \right). \tag{19.10}$$

Assume we are given $p(x_{t-1})$. If $p(x_{t-1})$ is Gaussian, to compute $p(u_t)$, we must compute $p(\pi(x_t))$. If π is a radial basis function network, we can apply moment matching to compute the distribution. A Radial Basis Function policy can be given as

$$\pi(x_*) = \sum_{s=1}^{N} \beta_s k_\pi(x_s, x_*) = \beta_\pi^T k_\pi(X_\pi, x_*) \tag{19.11}$$

where k_π is the squared exponential kernel and $\beta_\pi = (K_\pi + \sigma_\pi^2 I)^{-1} y_\pi$ and y_π are the outputs of the policy. We can effectively interpret this as a deterministic $\mathcal{GP}$.

To calculate $p(x_{t-1}, u_{t-1})$, where $p(\tilde{x}_{t-1}) = p(x_{t-1}, u_{t-1})$, we can make use of our ability compute the covariances via moment matching.

We can then compute

$$p(\Delta_t) = \int p(f(\tilde{x}_{t-1})|\tilde{x}_{t-1})p(\tilde{x}_{t-1})d\tilde{x}_{t-1} \tag{19.12}$$

by approximating $p(\Delta_t)$ using exact moment matching.

Finally, we can use our ability to compute the Gaussian joint distributions to compute $p(x_t)$. Assuming we can compute μ_Δ and Σ_Δ, we can write $\mathcal{N}(x_t|\mu_t, \Sigma_t)$, where

$$\mu_t = \mu_{t-1} + \mu_\Delta \tag{19.13}$$
$$\Sigma_t = \Sigma_{t-1} + \Sigma_\Delta + \text{cov}[x_{t-1}, \Delta_t] + \text{cov}[\Delta_t, x_{t-1}] \tag{19.14}$$

Returning back to

$$\mathbb{E}[c(x_t)] = \int c(x_t)\mathcal{N}(x_t|\mu_t, \Sigma_t)dx_t, \quad t = 0, ..., T, \tag{19.15}$$

we can choose the cost c to solve the integral analytically. For instance

$$c(x) = 1 - \exp(-||x - x_d||^2/\sigma_c^2). \tag{19.16}$$

Both the expected cost and the gradients of the cost with respect to the policy parameters can be computed analytically. Typically NLP solvers can be used to compute θ.

19.2. Guided Policy Search

Typically, our objective is to minimize the expectation $\mathbb{E}_{p(\tau)}[\ell(\tau)]$ over trajectories $\tau = \{x_0, u_0, x_1, u_1, ...u_{N-1}, x_N\}$. $\ell(\tau) = \sum_{t=1}^N \ell(x_t, u_t)$ is the total cost and $p(\tau) = p(x_0)\prod_{t=0}^N p(x_{t+1}|x_t, u_t)p(u_t|x_t)$.

Guided policy search can be succinctly stated as

- Use stochastic policy to "roll-out" trajectories
- Use set of trajectories to fit a model
- Compute an updated policy using the model

Let us first consider an approach that only attempts to fit a local linear model of the form

$$p(u_t|x_t) = \mathcal{N}(K_t x_t + k_t, C_t). \tag{19.17}$$

This controller structure should now look familiar (e.g., LQR, CCMs), except now our control law is stochastic (for exploration). Let's also restrict ourselves to

$$p(x_{t+1}|x_t, u_t) = \mathcal{N}(A_t x_t + B_t u_t + c_t, \Sigma_t) \tag{19.18}$$

where

$$A_t = \frac{\partial f(x_t, u_t)}{\partial x}, \quad B_t = \frac{\partial f(x_t, u_t)}{\partial u} \tag{19.19}$$

To solve this we can leverage iterative LQR. Remember the optimal control law is

$$u_t = \hat{u}_t + k_t + K_t(x_t - \hat{x}_t) \tag{19.20}$$

where $k_t = -Q_{uu}^{-1}Q_u^T$ and $K_t = -Q_{uu}^{-1}Q_{ux}$.

If we use a time-varying linear Gaussian controller $p(u_t|x_t)$ and let $C_t = Q_{uu}^{-1}$, then the stochastic policy optimizes the following objective

$$\min_{p(\tau)\in\mathcal{N}(\tau)} \sum_{t=0}^N \mathbb{E}_p[\ell(x_t, u_t)] - \mathcal{H}(p(u_t|x_t)) \tag{19.21}$$

where $\mathcal{H}$ is the differential entropy given as $\mathcal{H}(p(u_t|x_t)) = -\sum_{u_t|x_t} p(u_t|x_t)\log(p(u_t|x_t)) = \mathbb{E}[-\log(p(u_t|x_t))]$.

If you remember back to iLQR, the algorithm leverages a linesearch to update the policy. Typically, this involves "rolling-out" the new policy with a different scaling factor and computing the cost. The scaling factor is reduced until the new cost is less than the old cost. However, for deployment on real systems, this linesearch is not practical.

Instead, GPS solves

$$\min_{p(u_t|x_t)} \mathbb{E}_{p(\tau)}[\ell(\tau)] \quad s.t. \quad D_{KL}(p(\tau)||\hat{p}(\tau)) \leq \epsilon. \tag{19.22}$$

Since the new and old trajectory distribution dynamics are the same we have

$$D_{KL}(p(\tau)||\hat{p}(\tau)) = \sum_{\tau} p(\tau)\log\frac{p(\tau)}{\hat{p}(\tau)} = \sum_{t=0}^{N} \mathbb{E}_{p(x_t,u_t)}[\log\hat{p}(u_t|x_t)] - \mathcal{H}(p(u_t|x_t)) \tag{19.23}$$

The Lagrangian is given as

$$\mathcal{L}(p,\eta) = E_p[\ell(\tau)] + \eta[D_{KL}(p(\tau)||\hat{p}(\tau)) - \epsilon] \tag{19.24}$$

$$= \sum_{t=0}^{N} \mathbb{E}_{p(x_t,u_t)}[\ell(x_t,u_t) - \eta\log\hat{p}(u_t|x_t)] - \eta\mathcal{H}(p(u_t|x_t)) - \eta\epsilon \tag{19.25}$$

Divide through by η and we can use iLQR to update the augmented cost function

$$\tilde{\ell}(x_t,u_t) = \frac{1}{\eta}\ell(x_t,u_t) - \log\hat{p}(u_t|x_t) \tag{19.26}$$

19.2.1 Dual Gradient Descent

Dual gradient descent is used to solve constrained optimization problems. Consider

$$\min_x f(x) \tag{19.27}$$

$$s.t. \quad c(x) = 0 \tag{19.28}$$

We can then write the Lagrangian

$$\mathcal{L}(x,\lambda) = f(x) + \lambda c(x). \tag{19.29}$$

The dual function is given as

$$g(\lambda) = \inf\mathcal{L}(x,\lambda) = \mathcal{L}(x^*,\lambda) \tag{19.30}$$

where $x^* = \arg\min_x \mathcal{L}(x,\lambda)$.

The Dual Gradient Descent Algorithm is

- Compute $x^* = \arg\min_x \mathcal{L}(x,\lambda)$
- Compute the gradient of the dual function $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{dx^*}\frac{dx^*}{d\lambda} + \frac{d\mathcal{L}}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}$
- Update the dual variable $\lambda = \lambda + \alpha\frac{dg}{d\lambda}$

19.2.2 Gaussian Mixture Models

GMMs can be used to fit the local data and then be used to compute the local Gaussian dynamics for executing iterative LQR. The GMM attempts to fit a model of the form $x_{t+1} = f(x_t,u_t)$

19.3. Neural Network Policies

Ideally, we would like to leverage a neural network to represent our policy, not just a local policy computed by iterative LQR. The hope is that this neural network policy generalizes better.

We now try to optimize our neural network policy as follows:

$$\min_{\theta, p(\tau)} \mathbb{E}[\ell(\tau)] \quad s.t. \quad D_{KL}(p(x_t)\pi_\theta(u_t|x_t)||p(x_t, u_t)) = 0 \tag{19.31}$$

$$\mathcal{L}(\theta, p, \lambda) = \mathbb{E}_{p(\tau)}[\ell(\tau)] + \sum_{t=0}^{N} \lambda_t D_{KL}(p(x_t)\pi_\theta(u_t|x_t)||p(x_t, u_t)) \tag{19.32}$$

This optimization is also solved using dual gradient descent. However, it occurs in an "outer" loop. The algorithm proceeds as follows

- Generate samples from linear-Gaussian Controller
- Fit the dynamics to the samples
- Minimize the KL divergence between the neural network policy and the local controller using supervised learning
- Update the local policy
- Take one step to optimize the dual variables

Bibliography

[1] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[2] Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

[3] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. *Advances in neural information processing systems*, 27, 2014.

