Reinforcement Learning - Week 6

Model-based reinforcement learning
Optimizing a trajectory using iLQR

Antonio Chiappetta

November 2019

1 Definition of the cost function

1.1 System dynamics

In the inverted pendulum problem, state and and action are defined respectively in terms of angle position and velocity and in terms of the applied force:

$$x = (\theta,\ \dot{\theta})^T \tag{1}$$

$$u = (f)^T \tag{2}$$

The system dynamics can be represented with a non-linear function in the form:

$$x_{t+1} = F(x_t, u_t) = (\theta_{t+1}, \dot{\theta}_{t+1}) \tag{3}$$

$$\dot{\theta}_{t+1} = \dot{\theta}_t + (-\frac{3g}{2l} \cdot (\theta_t + \pi) + \frac{3}{2ml^2} \cdot u_t) \cdot dt \tag{4}$$

$$\theta_{t+1} = \theta_t + \dot{\theta}_{t+1} \cdot dt \tag{5}$$

Thanks to the KRR model we can avoid worrying about this non-linearity, and work with an approximated linear model.

1.2 Cost function (Task 1)

Considering a sequence of actions $\boldsymbol{u} = (u_1, ..., u_{T-1})^T$ and of corresponding states $\boldsymbol{x} = (x_1, ..., x_T)^T$ generated by this sequence, a quadratic cost function, representing the cost incurred until the horizon is reached, is:

$$J_0(x, u) = (x_T - x^*)^T Q_F (x_T - x^*) + \sum_{t=1}^{T-1} (x_t^T Q x_t + u_t^T R u_t), \tag{6}$$

where $Q \in \mathbb{R}^{2x2}$ and $Q_F \in \mathbb{R}^{2x2}$ determine state costs and $R \in R^{1x1}$ determine action costs, and $x^* = (\frac{\pi}{2}, 0)$ is the target state. We can represent these matrices in terms of costs in this way:

$$Q_F = Q = \begin{bmatrix} \frac{1}{2}c_\theta & 0 \\ 0 & \frac{1}{2}c_{\dot{\theta}} \end{bmatrix} \tag{7}$$

$$R = \begin{bmatrix} c_f \end{bmatrix} \tag{8}$$

To adapt the function to a single step inside the time horizon, it can be expressed as:

$$J_0(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t, \tag{9}$$

1.3 Hyper-parameters selection (Question 2)

The choice for state costs and action costs depend on how much we want to penalize each of the involved variables. The goal of the inverted pendulum is to reach the unstable equilibrium at $\theta = \frac{\pi}{2}$, using as little force as possible. The angle is the parameter that most of all identifies how close we are to the objective; the angular velocity needs to be slightly penalized to avoid getting to the objective too quickly and miss it; the force determines the angular acceleration and needs to be penalized because we need the pendulum to accelerate a lot when it is far from the objective, and then decelerate a lot when it is getting close. Thus, the considerations to do are:

- Angle needs to be highly penalized
- Angular velocity needs to be slightly penalized
- Force needs to be penalized

To match these requirements, and still try out many combinations of these hyper-parameters, we will use factorial design with the following sets of values, for a total of 36 combinations:

- $c_\theta \in [0.5; 1.5]$, with a 0.5 step
- $c_{\dot{\theta}} \in [0.05; 0.15]$, with a 0.05 step
- $c_f \in [0.1; 0.5]$, with a 0.1 step

1.4 Gradient and Hessian of the cost function (Task 1)

After having calculated the matrix products in equation (9), the first and second order derivatives of the cost function can be calculated as follows:

$$J(x_t, u_t) = c_\theta \cdot \theta_t^2 + c_{\dot{\theta}_t} \cdot \dot{\theta}^2 + c_f \cdot f_t^2 \tag{10}$$

$$\dot{J}(x,u) = \begin{bmatrix} \frac{\partial J(x,u)}{\partial \theta} \\ \frac{\partial J(x,u)}{\partial \dot{\theta}} \\ \frac{\partial J(x,u)}{\partial f} \end{bmatrix} = \begin{bmatrix} 2 \cdot c_\theta \cdot \theta_t \\ 2 \cdot c_{\dot{\theta}} \cdot \dot{\theta}_t \\ 2 \cdot c_f \cdot f_t \end{bmatrix} \tag{11}$$

$$\begin{aligned} \ddot{J}(x,u) &= \begin{bmatrix} \frac{\partial \dot{J}(x,u)_\theta}{\partial \theta} & \frac{\partial \dot{J}(x,u)_\theta}{\partial \dot{\theta}} & \frac{\partial \dot{J}(x,u)_\theta}{\partial t} \\ \frac{\partial \dot{J}(x,u)_{\dot{\theta}}}{\partial \theta} & \frac{\partial \dot{J}(x,u)_{\dot{\theta}}}{\partial \dot{\theta}} & \frac{\partial \dot{J}(x,u)_{\dot{\theta}}}{\partial t} \\ \frac{\partial \dot{J}(x,u)_f}{\partial \theta} & \frac{\partial \dot{J}(x,u)_f}{\partial \dot{\theta}} & \frac{\partial \dot{J}(x,u)_f}{\partial t} \end{bmatrix} \\ &= \begin{bmatrix} 2 \cdot c_\theta & 0 & 0 \\ 0 & 2 \cdot c_{\dot{\theta}} & 0 \\ 0 & 0 & 2 \cdot c_f \end{bmatrix} \end{aligned} \tag{12}$$

1.5 Implementation

The following graph shows the result generated by the implementation of the cost function described before, using the following hyper-parameters:

- $c_\theta = 1.0$
- $c_{\dot{\theta}} = 0.1$
- $c_f = 0.5$

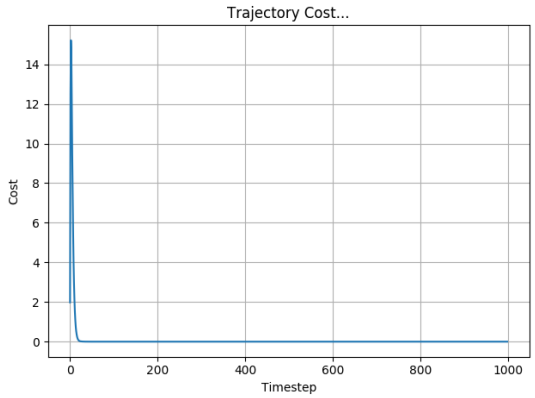

Figure 1: iLQR with KRR approximated model

2 iLQR for non-linear cost functions (Question 1)

Yes, it can handle a non-linear cost function. Actually, iLQR works with quadratic cost functions, that are not linear.

The reason behind that choice is that quadratic costs are needed to have at every iteration a positive cost function, that is easier to minimize and that always assigns the same importance to positive and negative deviations from the target.

3 Use of system dynamics (Task 4)

3.1 Implementation

The following figure shows the usage of the system dynamics with the cost function implemented before.

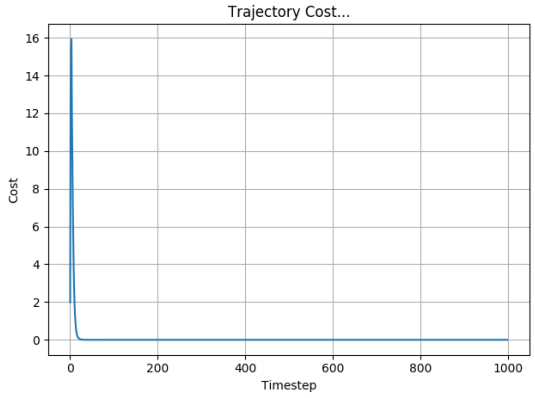

Figure 2: iLQR with system dynamics

4 Account for approximated model uncertainty (Question 3)

A possible improvement could be generated by the use of a Kalman filter to control to uncertainty of the estimates. The estimates is updated using a state transition model with a weighted average, assigning more weight being to estimates with higher certainty. The system can then be controlled using LQR with mean predicted state.

The additional predictions needed by the Kalman filter can be quite slow using KRR. An alternative to speed up predictions by having a very similar learned function is Support Vector Regression. SVR takes more time to train but learns a sparse model by using way less support vectors, so it can later generate predictions at a higher speed.

5 Horizon effect on learning (Question 4)

The minimum length of the horizon is a problem-dependent quantity which must be found by trial-and-error, and affects the way the model learns a policy with a good performance. Performance degrade for horizons shorter than a certain quantity, but do not qualitatively improve for longer ones. [1]

One way to speed up online trajectory optimization is to provide an approximation to the optimal value function, and use it as final cost applied at the horizon. Specifically, this makes it possible to use shorter horizons while avoiding myopic behavior.

References

[1] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. *IEEE International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012.