ACRL (16-899, S09) Lecture #24 (Tuesday April 21)

Adaptive Control - An Anthropological View

Lecturer: Chris Atkeson *Scribe: Forrest Rogers-Marcovitz*

What can we learn from cross-fertilization between Adaptive Control and Machine Learning?

1 Traditional Adaptive Control

We present three different traditional Adaptive Control (AC) camps. There is also a camp that dismisses AC and focuses on making a robust controller for all conditions.

1.1 Switch between known controllers

Gain scheduling is a common form with local gains and dynamics; a different controller is created for each environment. Table look-up is one implementation. Planes uses this and change the the controller based on altitude and speed. One common problem encountered is chatter around state transitions that can lead to instability.

1.2 Indirect AC

This technique learns the system model and then design the controller. The typical linear model consists of a feature vector, θ, and parameters, ϕ:

$$x_{t+1} = \theta\phi \tag{1}$$

We talked about three methods.

- Pole Placement:
  Changing the goal matrix, K, moves the poles to the desired position and delay constants. This is a form of imitation learning.

$$\begin{aligned} \vec{u} &= -K\vec{x} && (2)\\ \dot{\vec{x}} &= \mathbf{A}\vec{x} + \mathbf{B}\vec{u} && (3)\\ &= (\mathbf{A} - \mathbf{B}K)\vec{x} && (4) \end{aligned}$$

- Model Reference AC (MRAC):
  This methods involves zeros and poles placement. It can not handle zeros in the right hand plane, non-minimum phase system.
- LQR:

$$c(x, u) = x^\top Qx + u^\top Ru \tag{5}$$

1.3 Direct AC

This technique learns the controller with out a system model. Chris claims this is no different than indirect AC because it indirectly learns the system model but doesn't perform as well.

2 Example: Pole Placement

Given the system,

$$\dot{x} = ax + bu + c \tag{6}$$

we want to shift the poles from a to d,

$$\dot{x} = dx + bu \tag{7}$$

$$u = -k_1 x - k_2 \tag{8}$$

$$k_1 = \frac{a-d}{b}, \quad k_2 = \frac{c}{b} \tag{9}$$

3 System Model Learning

Given your basic linear dynamic system:

$$x_{k+1} = Ax_k + Bu_k \tag{10}$$

We can learn the A matrix by setting the control, $u = 0$, to zero and plotting the response. Similarly, stabilize the state and whack it; watch the response to learn B. Now that we don't live in the sixties and have computers, we have other methods.

$$A\theta = b \tag{11}$$

$$\begin{pmatrix} x_0 & u_0 & 1 \\ x_1 & u_1 & 1 \\ \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} a \\ b \\ \vdots \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix} \tag{12}$$

Three methods were presented.

3.1 Least-Square Regression

This can be done with a pseudo-inverse:

$$\hat{\theta} = (A^\top A)^{-1} A^\top b \tag{13}$$

Single Value Decomposition (SVD) works better. This is a batch method only.

3.2 Gradient Descent

$$
\begin{aligned}
score : S &= (a_i\theta - b_i)^\top (a_i\theta - b_i) && (14)\\
gradient : \frac{\partial S}{\partial \theta} &\propto (a_i\theta - b_i)a_i && (15)\\
update : \Delta\theta &= -\epsilon\frac{\partial S}{\partial \theta} = -\epsilon(a_i\theta - b_i)a_i && (16)
\end{aligned}
$$

3.3 Sherman-Morrison-Woodbury

This is a second order incremental least squares update. a_i is a column vector, b_i is an element of b, and A_{j-1}, is the first j-1 rows of A.

$$
\begin{aligned}
N_{j-1} &= (A_{j-1}^\top A_{j-1})^{-1} && (17)\\
N_j^{-1} &= (N_{j-1} + a_j a_j^\top)^{-1} && (18)\\
&= N_{j-1}^{-1} - \frac{N_{j-1}^{-1} a_j a_j^\top N_{j-1}^{-1}}{1 + a_j^\top N_{j-1}^{-1} a_j} && (19)
\end{aligned}
$$

4 Problems

Unfortunitly these methods suck most of the time; it's easy to violate the underlying assumptions. Here are some problems.

4.1 Slow Gradients

Slow gradients lead to really slow convergence.

4.2 Noise

When there noise is louder than the signal, θ learns the correlation in noise. There are two methods to correct this:

- Persistence of Excitation - "Do something": add energy to all modes, make signal bigger than noise
- Dead Zone - "Do nothing": set signal to zero if close zero

4.3 Outliers

Outliers have an unproportional affect on the system. We need a way of rejecting them. Here are a few hack-of-the-week methods for outliers rejection:

- forgetting or add decay constant
- trust regions
- normalization: $\Delta\theta = -\frac{\epsilon}{m^2}\frac{\partial S}{\partial \theta}$, $m^2 = 1 + \phi_i^\top P \phi_i$, where ϕ_i is the state radius.
- introduce a bias: $(A^\top A + \lambda I)^{-1}$, you can also add a prior weight.

4.4 Unmodeled Dynamics

They cause all kinds of problems. It is especially bad when combined with noise. A MATLAB simulated system was used to demonstrate. The system model used by the filter was a single rigid body, but the actual dynamics where two elements, each with half the mass, combined with a spring and damper. The natural frequency was set close to the excitation frequency. With a very small amount of noise, the learned mass converges to zero as there was no correlation to the modeled and acutal system.

5 Conclusion

The majority of the Machine Learning community ignores these problems, while the Adaptive Control community is not flexible. They should learn from each other.