On the stability of robot learning models

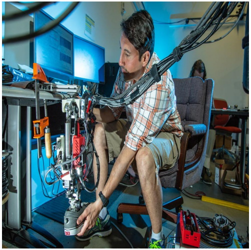
Lekan Ogunmolu

Preferred Networks, Tokyo, Japan

July 05, 2018

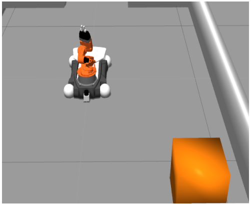
Hand Engineered Control Laws

- Deriving control laws by hand $\rightarrow$
  - finding stable robust control laws often require high discipline expertise
  - hand-designed controllers will only go so far for controlling complex nonlinear systems
  - It's 2018. Robot Learning is <u>real</u>.



Robot Learning Models

- Learning control laws/policies from data
  - often meets performance requirements
  - quite often done with black box basis functions e.g. neural networks
  - no stability guarantee
- "A robot may not harm humanity, or, by inaction, allow humanity to come to harm." – Asimov's zeroth law



Research Outlook

- Q-Learning Pros
  - model-free approach
  - often uses parametric Lipschitz basis functions to parameterize an unknown DS
  - examples: Q-learning, Batch Q-learning, QT-Opt etc

- Q-Learning Cons:
  - over-parameterized policies/overly sensitive policies
  - gradient-based optimization of a reward function often noisy

Policy Search

- Policy Search Approaches:
  - allows self-discovery of policies
  - directly operate in parameter space of the parameterized policies

- Policy Search Approaches:
  - do not fare well in the presence of uncertainty [Ogunmolu et al, NIPS 2017, IROS 2017]
  - need extra robustness design to assure perturbations adaptation [Ogunmolu et al, IROS 2018]

Estimating a Dynamical System's 'Stability from Data

- Stable estimator of Dynamical Systems, Khansari-Zadeh, 2011
  - Idea: parameterize reaching motions with a GMM
    - learned model will be almost unstable.
  - What to do?
    - with Lyapunov analysis, derive globally asymptotically stable control law
    - use Lyapunov redesign to correct unstable trajectories in learned model
    - then execute learned trajectory on robot

Advantages of SEDs

- Reactivity: Instant adaptation to dynamic environments
- Robustness: Mitigates parametric disturbance in the face of uncertainty
- Steady State: Guaranteed convergence to steady state

- Goal: Harness Lyapunov stability analysis on robot learning models
  - use parametric models (because, why not?), stability is provable (fyi, this is why not ☺)

Data-driven robot control and stability

- Idea: Show robot motion to be executed at a later future time
  - for example, a set of N kinesthetic demonstrations governed by a first-order ode $\dot{\zeta}^{t,n} = f(\zeta^{t,n})$
  - $\zeta^{t,n}$, $\dot{\zeta}^{t,n} \in \mathbb{R}^d$ are state vectors that completely describe the robot's motion
  - $f(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$ is a nonlinear continuous and continuously differentiable function with a single equilibrium point $\dot{\zeta}^\star = f(\zeta^\star) = 0$
- from the N demonstrations construct an estimate, $\hat{f}$ of f

Learning Control Lyapunov Function (CLF)

- Lyapunov functions help us analyze the stability of the equilibrium points of a differential equation such as

$$\dot{\zeta} = f(\zeta, t), \quad \zeta(t_0) = \zeta_0 \tag{1}$$

- Lyapunov functions also used to design stable feedback control laws ($\leftarrow$ Lyapunov redesign)
  - essentially involve the selection of a suitable function $V(\zeta)$ such that $\dot{V}(\zeta) \leq -Q(\zeta)$
  - $Q(\zeta)$ is a SPD function (LaSalle's Invariance Theorem)

Definition

A smooth positive definite and radially unbounded function $V(\zeta)$ is called a CLF for (1) if

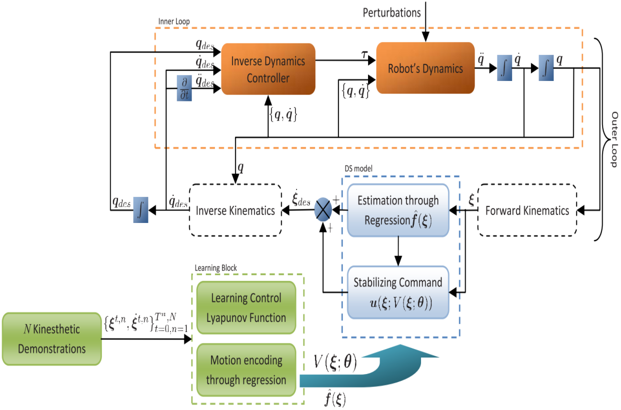
$$\inf_u \left[\frac{\partial V}{\partial \zeta}(\zeta) f(\zeta, u)\right] \leq -Q(\zeta) \quad \zeta \neq 0 \tag{2}$$

Learning Control Lyapunov Function

- Note that $V(\zeta)$ must be 0 everywhere at the target attractor $\zeta^\star$ and positive everywhere else i.e. $V(\zeta) > 0 \, \forall \, \zeta \in \mathfrak{R}^d \backslash \zeta^\star$ and $V(\zeta^\star) = 0$

- Lyapunov functions are by definition difficult to hand-engineer for complex plants/system dynamics
  - successful deployment usually guided by the Engineer's experience/intuition about the problem
  - fun fact: Lyapunov doesn't know the correct way to handpick a Lyapunov function for a system

Khansari's Algorithm



Lyapunov Function: Definition

- Define a class $\mathcal{C}^1$ smooth $V(\cdot)$ e.g.

$$V(\zeta;\theta) = (\zeta - \zeta^\star)^T P^0 (\zeta - \zeta^\star)^T + \sum_{\ell=1}^{\mathcal{L}} \beta^\ell (\zeta - \zeta^\star)^T P^\ell (\zeta - \mu^\ell - \zeta^\star) \tag{3}$$

  - ℓ is a user defined quantity, specifying the number of asymmetric quadratic functions
  - $\mu^\ell \in \mathbb{R}^d$ are the vectors influencing the asymmetric shape of $V(\cdot)$
  - $P^\ell \in \mathbb{R}^{d \times d}$ are positive definite matrices, and the coefficients $\beta^\ell(\zeta;\theta)$ are

$$\beta^\ell(\zeta;\theta) = \begin{cases} 1, \ \forall\, \zeta : (\zeta - \zeta^\star)^T P^\ell (\zeta - \mu^\ell - \zeta^\star) \geq 0 \\ 0, \ \forall\, \zeta : (\zeta - \zeta^\star)^T P^\ell (\zeta - \mu^\ell - \zeta^\star) < 0 \end{cases} \tag{4}$$

- Note that $V(\zeta)$ must be 0 everywhere at the target

Lyapunov Function: Definition

- Turns out that (3) helps fulfill the stability constraints for the CLF function $V(\cdot)$
- Helps scale up to higher dimensional dataset
- (3) can be paramerized by a vector θ,
- obtained by optimizing an appropriate cost function as a box-constrained convex optimization problem e.g. with the L-BFGS algorithm

$$\min_{\theta} J(\theta) = \sum_{n=1}^{N} \sum_{t=1}^{T^n} \frac{(1+\bar{w})\ \mathrm{sign}(\psi^{t,n})}{2} (\psi^{t,n})^2 + \ldots \\ \frac{(1-\bar{w})}{2} (\psi^{t,n})^2 \\ \text{subject to } P^{\ell} \succ 0 \ \forall\ \ell > 0, \ldots, L \tag{5}$$

Convex Optimization of Lyapunov Function

- $\psi^{t,n}$ is defined as

$$\psi^{t,n} = \psi(\zeta^{t,n}, \dot{\zeta}^{t,n}; \theta) = \frac{(\nabla_\zeta V(\zeta^{t,n}; \theta))^T \dot{\zeta}^{t,n}}{\|\nabla_\zeta V(\zeta^{t,n}; \theta)\| \|\dot{\zeta}^{t,n}\|} \tag{6}$$

- Finding the stabilizing command may consist of solving the optimization problem

$$\min_{u(\zeta)} \quad \frac{1}{2} u(\zeta)^T u(\zeta)$$

$$\text{subjet to} \quad (\nabla_\zeta V(\zeta))^T \left(\hat{f}(\zeta) + u(\zeta)\right) \le -\rho(\|\zeta\|), \quad \forall \zeta \in$$

$$\hat{f}(\zeta) + u(\zeta) = 0, \qquad \zeta = \zeta^\star \tag{7}$$

- where $\rho(\|\zeta\|)$ is a class $\mathcal{K}$ function, that assures decrease in the energy function

Convex Optimization of Lyapunov Function

- As in Khansari, 2011, set

$$\rho(\|\zeta\|) = \rho_0(1 - e^{-\kappa_0\|\zeta\|}) \tag{8}$$

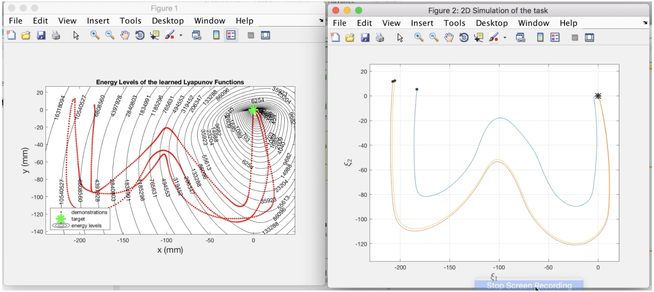
- where $\rho_0 > 0$ and $\kappa_0 > 0$.
- (7) is convex with solution:

$$u^\star(\zeta) = \begin{cases} 0 \quad \forall\, \zeta \in \mathbb{R}^d \backslash \zeta^\star, \nabla_\zeta V(\zeta)^T \hat{f}(\zeta) + \rho(\|\zeta\|) \leq 0 \\ -\hat{f}(\zeta) \quad \zeta = \zeta^\star \\ -\left(\dfrac{\nabla_\zeta V(\zeta)^T \hat{f}(\zeta) + \rho(\|\zeta\|)}{\nabla_\zeta V(\zeta)^T \nabla_\zeta V(\zeta)}\right) \nabla_\zeta V(\zeta) \quad \text{elsewhere} \end{cases}$$

CLFDM Algorithm Snapshot

- Parametrize $V(\cdot)$ with a vector θ, and learn the parameters θ using (5)
- Learn $\hat{f}(\zeta)$ from the dataset $\mathcal{D} = \{\zeta^{t,n}, \dot{\zeta}^{t,n}\}_{t=0,n=1}^{T^n,N}$
- **Robot Execution:** With $\hat{f}(\zeta), V(\zeta;\theta^\star), \rho(\|\zeta\|)$, initial condition ζ^0, targert $\zeta^\star$, accuracy tolerance ϵ, and integration time step δt
- set i $= 0$
- **while** $\|\zeta^i - \zeta^\star\| > \epsilon$ **do**
  - Compute $\hat{f}(\zeta^i)$
  - Compute $u^\star(\zeta^i)$ using (7)
  - update $\dot{\zeta}^i \leftarrow \hat{f}(\zeta^i) + u^\star(\zeta^i)$
  - Compute next state $\zeta^{i+1} \leftarrow \zeta^i + \dot{\zeta}^i\,\delta t$
  - $i \leftarrow i + 1$

Video Simulation



Example on the Torobo Arm

Comments on SEDS and CLFDM

- Because we use Gaussian Mixture Models, the model is not robust to high dimensional-data
- To assure robustness to higher dimensional data, we could:
  - borrow ideas from nonparametric reinforcement learning, e.g. model ensembles (Kurutach et. al. ICLR 2018)
  - or adopt ideas from gain scheduling in adaptive control (Sastry & Bodson, Adaptive Control, 1989)
- idea is to separate the dynamics into separate nonlinear envelopes
- linearize each envelope, and design control laws for each linearized envelope
- then interpolate the controllers to adapt to the model when running the system

Exploiting Gaussian Processes for Higher Dimensional Data

- GPs can maintain uncertainty over predictions
  - big plus for parametric learning-based models compared against neural networks which hardly model uncertainty
  - + stability can be maintained along the trajectories of the error dynamics
  - + infinite representation power with plenty data
- Catch
  - the curse of dimensionality is an ever-present trouble
  - + so far they've been relatively limited to low-dimensional settings

Proposal: Gaussian Processes + Differential Dynamic Programming

- what if we could learn a stable estimator of a DS as we gather data during imitation learning/roll-outs?
  - continually learn a Gaussian Mixture Model based on data samples gathered during exploration
  - stabilize the learnt model $\hat{f}(\zeta)$ using SEDS
  - since model learned will be crude, leverage on DDP to continually solve an optimal control problem (e.g. see Ogunmolu et al, IROS18)
- **DDP**:
  - an incremental step-by-step approach for controlling a nonlinear system
  - idea: suppose we have a discrete, stochastic dynamical system $\dot{x} = f(x, u, w)$

Differential Dynamic Programming

- DDP constructs a locally linear approximation to x, by applying (an erstwhile found) nominal control law, $\bar{u}$, to x; this yields $\bar{x}$
- when we apply the nominal control, the dynamics become $\delta x = x - \bar{x}$
- how does DDP work?
  - starting at a future time, T, we will apply the control sequence $\{\bar{u}\}_{t=0}^{T}$ in order to obtain open and closed-loop feedback gains k, K respectively
  - in a forward pass, we will repeatedly apply the feedback policy $\pi(x)$ which we will run on the system
- the process above is repeated until convergence; typically occurs in $4 - 5$ iterations (see Jacoson and Mayne)

Assuring stability of learning-based reaching motions with GPs + SEDS + DDP

- Goal: achieving stable complex motion control
  - leveraging on data efficiency ✓ GMM
  - avoid the black box optimization that deep neural networks provide ✓
  - can we assure piecewise stability while trying to achioeve these reaching motions?
    - turns out that with SEDs, yes we can ✓
    - **CATCH:** finding region of attraction is computationally costly and a non-trivial task in higher dimensions
    - **Solution:** Construct an iterative method to find a GMM in order to ensure the asymptotic stability when we reach the target ✓

Proposal: High-Dimensional DP-based Parametric Policy Optimization

- Given a trajectory, x, with cartesian velocity $\dot{x}$, our goal is to learn the discrete time, stochastic dynamics $\dot{x} = \hat{f}(x, u, w)$
  - x is the n-dimensional state vector, u is the p-dimensional control vector, and w may be considered a p-dimensional random disturbance vector
- we will take $x \in \mathbb{R}^d$, where d is at least ≥ 7
  - note that SEDs and traditional Gaussian processes fail when we scale up dimensions such as this
- we also want to learn a policy Π which contains the admissible policies $\{\pi_1, \pi_2, \ldots, \pi_n\}$

- such that each control law in the policy sequence constitutes a stable feedback controller that executes a desired sequential reaching motion

Proposal: High-Dimensional DP-based Parametric Policy Optimization

- Given a trajectory, x, with cartesian velocity $\dot{x}$, our goal is to learn the discrete time, stochastic dynamics $\dot{x} = \hat{f}(x, u, w)$
- we will take $x \in \mathbb{R}^d$, where d is at least ≥ 7
  - note that SEDs and traditional Gaussian processes fail when we scale up dimensions such as this
- we also want to learn a policy Π which contains the admissible policies $\{\pi_1, \pi_2, \ldots, \pi_n\}$

- such that each control law in the policy sequence constitutes a stable feedback controller that executes a desired sequential reaching motion