Adaptive learning for optimal feedback control of high-dimensional nonlinear systems

Tenavi Nakamura-Zimmerer

Applied Mathematics Department
University of California, Santa Cruz

October 11, 2019
Optimal control
A practical problem

\[
\begin{align*}
\text{minimize} & \quad u \left[ \text{distance to Merrill} + \text{sweatiness} \right] \\
\text{subject to} & \quad t_f = 15 \text{ minutes}, \quad \text{position}(0) = \text{Baskin} \\
& \quad \frac{d}{dt} \text{position} = f(\text{steep hills, walking speed, direction}) \\
& \quad \text{sweatiness} = \int_0^{t_f} (\text{steep hills}) \times (\text{walking speed}) \, dt
\end{align*}
\]
Optimal control
Problem statement

(OCP) \[
\begin{align*}
\text{minimize} \quad & F(x(t_f)) + \int_0^{t_f} L(t, x, u) \, dt, \\
\text{subject to} \quad & \dot{x}(t) = f(t, x, u), \\
& x(0) = x_0.
\end{align*}
\]

state: \( x(t) : [0, t_f] \to X \subseteq \mathbb{R}^n \)
control: \( u(t, x) : [0, t_f] \times X \to U \subseteq \mathbb{R}^m \)
dynamics: \( f(t, x, u) : [0, t_f] \times X \times U \to \mathbb{R}^n \)
terminal cost: \( F(x(t_f)) : X \to \mathbb{R} \)
running cost: \( L(t, x, u) : [0, t_f] \times X \times U \to \mathbb{R} \)
Some applications
Two kinds of solutions

Open-loop:

Closed-loop/feedback:
Hamilton-Jacobi-Bellman equation

For closed-loop control, solve a *family* of OCPs over all \((t, x)\):

\[
\begin{align*}
\text{(OCP2)} \quad \begin{cases} 
\text{minimize} & F(y(t_f)) + \int_t^{t_f} L(\tau, y, u) d\tau, \\
\text{subject to} & \dot{y}(\tau) = f(\tau, y, u), \\
& y(t) = x.
\end{cases}
\end{align*}
\]

Define the *value function*

\[
V(t, x) := \inf_{u \in \mathcal{U}} \left\{ F(y(t_f)) + \int_t^{t_f} L(\tau, y, u) d\tau \right\}
\]

Value function satisfies **Hamilton-Jacobi-Bellman** PDE

\[
\begin{cases} 
- V_t(t, x) = \min_{u \in \mathcal{U}} \left\{ L(t, x, u) + [V_x(t, x)]^T f(t, x, u) \right\}, \\
V(t_f, x) = F(x).
\end{cases}
\]
Hamilton-Jacobi-Bellman equation

How do we use this? Define the Hamiltonian

\[ H(t, x, V_x, u) := L(t, x, u) + V_x^T f(t, x, u) \]

Optimal control minimizes the Hamiltonian:

\[ u^* = u^*(t, x; V_x) = \arg \min_{u \in U} H(t, x, V_x, u) \]

\[ \implies H^*(t, x, V_x) := H(t, x, V_x, u^*) \]

\[ \text{(HJB)} \begin{cases} -V_t(t, x) = H^*(t, x, V_x), \\ V(t_f, x) = F(x). \end{cases} \]

1. Solve HJB offline for \( V(t, x) \)
2. Evaluate \( u^* = u^*(t, x; V_x) \) online
3. Go home
CURSE OF DIMENSIONALITY
HJB is a nonlinear PDE in $n$ spatial dimensions plus time

- Standard discretization out the window for $n \geq 4$
- Traditionally considered extremely challenging for $n \geq 5$
- Except in special cases considered intractable for $n \geq 10$

For context:

- Simplest model of a car has $n = 3$
- Rigid body model of satellite rotation has $n = 6$
- Quadcopter drone model has $n = 12$
- Systems in computational biology have $n \gg 1$
Computational methods for HJB

- level set methods (Osher & Sethian *JCP* 1988)
- max-plus algebra (McEneany *SICON* 2007)
- patchy dynamic programming (Navasca & Krener 2007)
- semi-Lagrangian methods (Falcone & Ferreti 2013)
- tree-structures (Alla et al. *SISC* 2018)
- polynomial Galerkin method (Kalise & Kunisch *SISC* 2018)

**Some drawbacks:**

- limited to moderate dimensions and/or special structure
- solution may be valid only in a small neighborhood of a nominal trajectory
- accuracy of the solution is hard to verify for general systems
Computational methods for HJB
Machine learning

The new hotness!

Forward-backward SDEs: represent $V(t, x)$ with a neural network (NN), solve SDEs related to stochastic HJB (Han et al. *PNAS* 2018, Raissi *arXiv:1804.07010* 2018)

Least-squares methods: represent $V(t, x)$ with a NN, minimize residual of HJB at collocation points (Tassa & Erez *IEEE Trans. NN* 2007, Sirignano & Spiliopoulos *JCP* 2018)

Approximate dynamic programming: e.g. reinforcement learning (Lewis & Liu 2013)
Characteristics of $V(t, x)$ evolve according to Pontryagin’s Minimum Principle, a two-point BVP\(^1\):

$$u^* = \arg\min_{u \in U} H(t, x, \lambda, u), \quad \lambda(t) : [0, t_f] \rightarrow \mathbb{R}^n$$

\[
\begin{aligned}
\dot{x}(t) &= f(t, x, u^*), \quad x(0) = x_0, \\
\dot{\lambda}(t) &= -H_x(t, x, \lambda, u^*), \quad \lambda(t_f) = F_x(x(t_f)), \\
\dot{v}(t) &= -L(t, x, u^*), \quad v(t_f) = F(x(t_f)).
\end{aligned}
\]

Solutions of BVP are open-loop, but along the characteristic $x = x(t; x_0)$ we have

$$u^*(t, x) = u^*(t; x_0), \quad V(t, x) = v(t; x_0), \quad V_x(t, x) = \lambda(t; x_0)$$

\(^1\)As long as the value function is $C^1$. Optimality of BVP solutions can be guaranteed under some convexity conditions.
Characteristics-based methods for HJB

BVPs are not easy to solve, but easier than HJB and are causality-free: they can be solved independently

- embarassingly parallelizable
- don’t need a grid
- enables semi-global solutions

Lax/Hopf formulation with convex optimization (Chow et al. JCP 2019)

Proposed method

Generate data → Physics-informed learning → Adaptive data generation → Convergence test → Accuracy validation → Online optimal feedback control

Generality:
- Grid-free
- NN representation
- Empirical accuracy validation

Data efficiency:
- Adaptive sampling
- NN warm start
- Physics-informed learning
Neural network modeling

Model complicated functions by composing simpler functions:

\[ V(t, x) \approx V^{NN}(t, x) = g_M \circ g_{M-1} \circ \cdots \circ g_1(t, x), \]

where each layer \( g_m(y) = \sigma_m(W_m y + b_m) \):

- \( W_m \): weight matrix
- \( b_m \): bias vector
- \( \sigma_m(\cdot) \): activation function (tanh for hidden layers, identity for output layer)
Physics-informed machine learning

Data set from solving BVPs:

\[
D = \left\{ \begin{array}{l}
\text{inputs: } (t^{(i)}, x^{(i)}) \\
\text{outputs: } V^{(i)} := V(t^{(i)}, x^{(i)}), \quad \lambda^{(i)} := \lambda(t^{(i)}, x^{(i)})
\end{array} \right\}_{i=1}^{N_d}
\]

Collection of all free parameters of the NN:

\[
\theta := \{ \mathbf{W}_m, \mathbf{b}_m \}_{m=1}^{M}
\]

Physics-informed learning problem:

\[
\begin{aligned}
\text{minimize } \theta & \quad \mathcal{L}(\theta; D) := \text{loss}_V(\theta; D) + \mu \cdot \text{loss}_\lambda(\theta; D), \\
\text{where } & \quad \text{loss}_V(\theta; D) := \frac{1}{N_d} \sum_{i=1}^{N_d} \left[ V^{(i)} - V^{\text{NN}}(t^{(i)}, x^{(i)}; \theta) \right]^2, \\
& \quad \text{loss}_\lambda(\theta; D) := \frac{1}{N_d} \sum_{i=1}^{N_d} \left\| \lambda^{(i)} - \lambda^{\text{NN}}(t^{(i)}, x^{(i)}; \theta) \right\|^2, \\
& \quad \mu \geq 0 \text{ is a scalar weight}
\end{aligned}
\]
Physics-informed machine learning

- **Exact gradients** $V_{x}^{NN}(t, x)$ via automatic differentiation
- Improved data efficiency, more physically-consistent models
- $u^{*} = u^{*}(t, x; V_{x})$, hence better costate prediction accuracy
  $\implies$ better controls

**Training:**

![Diagram of the training process](image-url)
Rigid body satellite model

\[ \mathbf{v} = (\phi \ \theta \ \psi)^T \]

(roll, pitch, yaw)

\[ \mathbf{\omega} = (\omega_1 \ \omega_2 \ \omega_3)^T \]

angular momenta

\[ \mathbf{u} = (u_1 \ u_2 \ u_3)^T \]

control (reaction wheels)

\[ \psi \ (yaw) \]

\[ \theta \ (pitch) \]

\[ \phi \ (roll) \]

Rebel scum
Rigid body dynamics

\[
\begin{align*}
\begin{pmatrix}
\dot{v} \\
J \dot{\omega}
\end{pmatrix}
&= 
\begin{pmatrix}
E(v) \omega \\
S(\omega)R(v)h + Bu
\end{pmatrix}
\end{align*}
\]

\[
E(v) = 
\begin{pmatrix}
1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\
0 & \cos \phi & -\sin \phi \\
0 & \sin \phi / \cos \theta & \cos \phi / \cos \theta
\end{pmatrix},
S(\omega) = 
\begin{pmatrix}
0 & \omega_3 & -\omega_2 \\
-\omega_3 & 0 & \omega_1 \\
\omega_2 & -\omega_1 & 0
\end{pmatrix}
\]

\[
R(v) = 
\begin{pmatrix}
\cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\
\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi & \sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi & \cos \theta \sin \phi \\
\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi & \cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi & \cos \theta \cos \phi
\end{pmatrix}
\]

\[
B = 
\begin{pmatrix}
1 & 1/20 & 1/10 \\
1/15 & 1 & 1/10 \\
1/10 & 1/15 & 1
\end{pmatrix},
J = 
\begin{pmatrix}
2 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & 4
\end{pmatrix},
\]

\[
h = 
\begin{pmatrix}
1 \\
1
\end{pmatrix}
\]

\[
\begin{aligned}
\text{minimize} \quad & u(\cdot) \\
\text{subject to} \quad & \int_t^{t_f} L(v, \omega, u) \, d\tau + \frac{W_4}{2} \| v(t_f) \|^2 + \frac{W_5}{2} \| \omega(t_f) \|^2, \\
& \dot{v} = E(v)\omega, \\
& J\dot{\omega} = S(\omega)R(v)h + Bu, \\
& L(v, \omega, u) = \frac{W_1}{2} \| v \|^2 + \frac{W_2}{2} \| \omega \|^2 + \frac{W_3}{2} \| u \|^2, \\
& W_1 = 1, \; W_2 = 10, \; W_3 = \frac{1}{2}, \\
& W_4 = 1, \; W_5 = 1, \; t_f = 20. \\
\end{aligned}
\]

Initial conditions in the domain

\[
\{ \mathbf{v}, \omega \in \mathbb{R}^3 \mid -\frac{\pi}{3} \leq \phi, \theta, \psi \leq \frac{\pi}{3} \text{ and } -\frac{\pi}{4} \leq \omega_1, \omega_2, \omega_3 \leq \frac{\pi}{4} \}
\]

Analytical optimal feedback law:

\[
u^*(t, v, \omega; V_v, V_\omega) = -\frac{1}{W_3} \left[J^{-1}B \right]^T V_\omega(t, v, \omega)
\]
Implementation details

- To compare w/ sparse grid method, learn $V(0, \mathbf{v}, \omega)$ only
- Validation data set has $|\mathcal{D}_{\text{val}}| = 1000$ points (at $t = 0$)
- Sparse grid w/ 44,698 points has a mean absolute error (MAE) of $3.7 \times 10^{-3}$ on this data set
- Feedforward NN with 3 hidden layers with 64 neurons each
- NVIDIA RTX 2018Ti GPU; TensorFlow in Python; optimize with L-BFGS
Learning the value function

- Pure regression ($\mu = 0$) doesn’t compare w/ sparse grid
- Can match sparse grid accuracy using 40 times fewer samples
- Fast training - *most of the cost is in data generation*
Closed loop simulations

\[
u_{NN}(t, x) = u^*(t, x; V_{x}^{NN}(t, x)) = \arg \min_{u \in \mathcal{U}} H(t, x, V_{x}^{NN}(t, x), u)
\]

In this case learned \(V(0, v, \omega)\) only, so implement model predictive feedback control (MPC): \(u = u_{NN}(t = 0, v, \omega)\)

Each evaluation of the control takes only about a millisecond
Closed loop simulations

\[ u^{NN}(t, x) = u^*(t, x; V_x^{NN}(t, x)) = \arg \min_{u \in \mathcal{U}} H(t, x, V_x^{NN}(t, x), u) \]

In this case learned \( V(0, \nu, \omega) \) only, so implement model predictive feedback control (MPC):

\[ u = u^{NN}(t = 0, \nu, \omega) \]

Each evaluation of the control takes only about a millisecond
Another look at optimization
Progressive batching

How to improve data generation? How much data to generate?

Usually in machine learning, BIG DATA...

- mini batches
- gradient (+ momentum)
- slow convergence
- mystical generalization properties

Progressive data generation strategy

Our problem: small data sets but we can generate more on the fly

⇒ How much data to generate? We propose a straightforward adaptive data generation and model refinement strategy:

\[
\sum_{m=1}^{M} \text{Var} \left[ \frac{\partial L}{\partial \theta_m} (\theta; (t^{(i)}, x^{(i)})) \right] \leq c |D_{\text{train}}^{r} | \| L_\theta (\theta; D_{\text{train}}^{r}) \|_1
\]

Optimization

Convergence test

Generate new data according to

\[
|D_{\text{train}}^{r+1} | \geq \frac{\sum_{m=1}^{M} \text{Var} \left[ \frac{\partial L}{\partial \theta_m} (\theta; (t^{(i)}, x^{(i)})) \right]}{c \| L_\theta (\theta; D_{\text{train}}^{r}) \|_1}
\]

Pass

Fail
Adaptive sampling scheme

Now we know how much data to generate... but we also have the freedom to choose where...

Pick initial conditions where predicted value gradient is large

1: while need more data do
2: Sample initial conditions $x_0^{(i)}$, $i = 1, \ldots, N_c$, from $X$
3: Predict gradient norms
   \[ \left\{ \left\| V_x^{NN} \left( 0, x_0^{(i)} \right) \right\| \right\}_{i=1}^{N_c} \]
4: Choose initial condition with largest gradient norm and solve the BVP
5: Add resulting time series to $D_{\text{train}}^{r+1}$
6: end while
Efficient data generation with NN warm start

Solving the BVP is easier than HJB, but still hard - highly sensitive to initial guess...

**NN warm start:**

1. Use a small data set to quickly train a low-fidelity NN model
2. Simulate the closed-loop dynamics with a NN controller
3. Predict \( \lambda(t) \approx V_x^{NN}(t, x) \) along trajectory
4. Simulation provides a good guess for the BVP solver
Training with adaptive data generation

Back to the rigid body problem

- Gradient loss weight $\mu = 10$; tolerance $\epsilon = 0.1$
- Training data set updated adaptively $\implies$ more useful data
- Compare to training on fixed data set with 8192 points

![Graph showing training data set sizes and optimization iterations](image-url)
Numerical results - NN warm start

- 1000 “difficult” initial conditions (largest predicted gradient norm, $\|V_{x}^{NN}(\cdot)\|$ picked from $10^6$ candidates)
- NNs have different levels of accuracy in predicting the gradient $V_{x}^{NN} \approx \lambda$, measured as mean relative $L^2$ error (MRL$^2$)

<table>
<thead>
<tr>
<th>NN warm start</th>
<th>gradient MRL$^2$</th>
<th>% BVP conv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0$</td>
<td>$3.7 \times 10^{-1}$</td>
<td>90%</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>$4.0 \times 10^{-2}$</td>
<td>99.6%</td>
</tr>
<tr>
<td>$10^1$</td>
<td>$3.5 \times 10^{-2}$</td>
<td>100%</td>
</tr>
</tbody>
</table>
High-dimensional application: PDE control problem

Discretized PDE-constrained optimal control problem\(^3\):

\[
\begin{align*}
\text{minimize } & \quad \int_t^{t_f} \frac{1}{2} \left[ w^T x^2(\tau) + W_1 u^2(\tau, x) \right] d\tau + \frac{W_2}{2} w^T x^2(t_f), \\
\text{subject to } & \quad \dot{x} = x \odot Dx + \nu D^2 x + \alpha x \odot e^{\beta x} + I_{\Omega} u, \\
& \quad \nu = 0.2, \alpha = 1.5, \beta = -0.1, \\
& \quad W_1 = 0.1, W_2 = 1, t_f = 8
\end{align*}
\]

\(D, D^2 \in \mathbb{R}^{n \times n}\): Chebyshev differentiation matrices

\(\odot\): element-wise multiplication (\(*\) in MATLAB)

\(I_{\Omega} = \) indicator function

---

\(^3\) An infinite horizon version of this problem was solved in dimension \(n = 12\) in Kalise & Kunisch *SISC* 2018.
Learning high-dimensional value functions

We test $n = 10, 20, \text{ and } 30$. Initial conditions sampled in

$$\mathcal{X}_0 = \{ x \in \mathbb{R}^n | -2 \leq x_j \leq 2, j = 1, 2, \ldots, n \}$$

- 50 trajectories in $\mathcal{D}_{\text{val}}$; start with 30 trajectories in $\mathcal{D}_{\text{train}}$
- Each trajectory contains a few hundred data points
- Generate more data adaptively during training
- Same NN architecture, but now take time $t$ as an input

<table>
<thead>
<tr>
<th>$n$</th>
<th>num. trajectories</th>
<th>training time</th>
<th>value MAE</th>
<th>gradient MRL$^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>132</td>
<td>10.1 min</td>
<td>$2.4 \times 10^{-3}$</td>
<td>$1.8 \times 10^{-2}$</td>
</tr>
<tr>
<td>20</td>
<td>60</td>
<td>9.2 min</td>
<td>$8.9 \times 10^{-4}$</td>
<td>$2.1 \times 10^{-2}$</td>
</tr>
<tr>
<td>30</td>
<td>59</td>
<td>13.3 min</td>
<td>$5.0 \times 10^{-4}$</td>
<td>$2.0 \times 10^{-2}$</td>
</tr>
</tbody>
</table>
Closed loop simulations

\[ X(0, \xi) = 2 \sin(\pi \xi) \]

\[ X(0, \xi) = -2 \sin(\pi \xi) \]
Summary remarks

1. Introduced a new data-driven framework for synthesizing nonlinear optimal feedback controls

2. Works for many kinds of systems, is data-efficient, and allows semi-global solutions and empirical accuracy validation

3. Demonstrated potential for use in solving practical and high-dimensional problems
Future work
State and control constraints, viscosity solutions, and stochastic control

Hard constraints:

Constraints appear in most practical problems

\[ \Rightarrow \text{discontinuous controls} \]
\[ \Rightarrow \text{non-smooth value functions} \]
\[ \Rightarrow \text{method of characteristics fails} \]

HJB still has *viscosity solution* $\varepsilon$ := limit of smooth solutions $V^\varepsilon(\cdot)$ of

\[
(\varepsilon\text{HJB}) \begin{cases} 
- [V_t^\varepsilon + H^*(t, x, V_x^\varepsilon)] = \varepsilon \cdot \text{Trace}(V_{xx}^\varepsilon), \\
V^\varepsilon(t_f, x) = F(x). 
\end{cases}
\]

Future work
Optimal feedback control under uncertainty

Uncertain optimal control problem:\(^4\)

\[
\begin{align*}
\text{minimize} & \quad \mathbb{E}_{x_0} \left[ F(x(t_f)) + \int_{t}^{t_f} L(\tau, x, u) \, d\tau \right], \\
\text{subject to} & \quad \dot{x}(\tau) = f(\tau, x, u), \\
& \quad x(t) \sim p(x_0),
\end{align*}
\]

Control depends on the PDF of the state: \( u = u(t, [p(t, x)]) \)

## Timetable of future research

<table>
<thead>
<tr>
<th>Fall '19</th>
<th>Winter '20</th>
<th>Spring '20</th>
<th>Summer '20</th>
<th>Fall '20</th>
<th>Winter '21</th>
<th>Spring '21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constrained problems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Viscosity solutions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncertain problems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thesis writing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.
References


