Collaborators and Acknowledgments

Xingjian Li
Emory

Lars Ruthotto
Emory

Samy Wu Fung
UCLA

Levon Nurbekyan
UCLA

Stan Osher
UCLA

Funding:

Special thanks: Organizers and staff of IPAM Long Program MLP 2019 and NVIDIA.
Overview

- **Background**
  - Problem
  - Pontryagin Maximum Principle (PMP)
  - Hamilton–Jacobi–Bellman Partial Differential Equation (HJB)

- **Mathematical Formulation**
  - Shock-Robustness
  - HJB Penalizers

- **Neural Networks (NNs)**
  - Model Formulation
  - Numerics

- **Results**
  - 150-Dimensional Swarm Trajectory Planning
  - Quadcopter with Complicated Dynamics

- **Conclusion**
Optimal Control (OC) Problem

Corridor Problem

Consider two centrally-controlled agents that navigate through a corridor/valley between two hills to fixed targets

Assume

- We have control over the agents’ velocities (the control)

Want

- Shortest paths, e.g. the geodesics (optimality)
- No collisions
- Agents to reach targets at final time
Multi-Agent Formulation

Consider \( n \) agents initially at \( x_1, \ldots, x_n \in \mathbb{R}^q \Rightarrow x = (x_1, \ldots, x_n) \in \mathbb{R}^d \)

Agents follow trajectories \( z_x(t) \) during time \( t \in [0, T] \)

\[
\begin{align*}
z_x(0) &= x = \begin{bmatrix} -2 \\ -2 \\ 2 \\ -2 \end{bmatrix} \quad \text{agent 1} \\
&= \begin{bmatrix} 2 \\ 2 \\ -2 \\ 2 \end{bmatrix} \quad \text{agent 2}
y &= \begin{bmatrix} 2 \\ 2 \\ -2 \\ 2 \end{bmatrix}
\end{align*}
\]

Terminal Cost

\[
G(z_x(T)) = \frac{\alpha_1}{2} \| z_x(T) - y \|^2
\]

for multiplier \( \alpha_1 \in \mathbb{R} \)
Trajectories Governed by Differential Equation

The state $z_x$ depends on the control $u_x$ and previous state via the system

$$\partial_t z_x(t) = f(t, z_x(t), u_x(t)), \quad z_x(0) = x$$

For Corridor: \quad $u_x(t)$ (the velocity)

where

- time $t \in [0, T]$
- initial state $x \in \mathbb{R}^d$
- admissible controls $U \subset \mathbb{R}^a$
- $f: [0, T] \times \mathbb{R}^d \times U \to \mathbb{R}^d$ models the evolution of the state $z_x: [0, T] \to \mathbb{R}^d$ in response to the control $u_x: [0, T] \to U$
Running Cost

Running costs where $z_i$ and $u_i$ are the state and control for the $i$th agent, respectively

$$L(t, z(t), u(t)) = E(z(t), u(t)) + \alpha_2 Q(z(t), u(t)) + \alpha_3 W(z(t), u(t))$$

$$= \sum_{i=1}^{n} E_i(z_i(t), u_i(t)) + \alpha_2 \sum_{i=1}^{n} Q_i(z_i(t), u_i(t)) + \alpha_3 \sum_{j \neq i} W_{ij}(z_i(t), z_j(t))$$

For Corridor:

$$\frac{1}{2} \| u_i(t) \|^2$$

sum of Gaussians
cpiecewise Gaussian repulsion

for multipliers $\alpha_2, \alpha_3 \in \mathbb{R}$ and

- $E_i$ is the energy of an agent,
- $Q_i$ represents any obstacles or terrain,
- $W_{ij}$ are the interaction costs between homogeneous agents $i$ and $j$ with radius $r$

$$W_{ij}(z_i, z_j) = \begin{cases} 
\exp \left(-\frac{\|z_i - z_j\|^2}{2r^2}\right), & \|z_i - z_j\| < 2r \\
0, & \text{otherwise}
\end{cases}$$
Optimal Control (OC) Problem

Running Cost: \( L(s, \cdot) = E(\cdot) + \alpha_2 Q(\cdot) + \alpha_3 W(\cdot) \)

Terminal Cost: \( G(z_x(T)) = \frac{\alpha_1}{2} \| z_x(T) - y \|^2 \)

**Goal:** Find the control that incurs minimal cost\(^1\)

\[
\Phi(t, x) = \inf_{u_x} \left\{ \int_t^T L(s, z_x(s), u_x(s)) \, ds + G(z_x(T)) \right\}
\] (2)

- \( \Phi(t, x) \in \mathbb{R} \) is the **value function** (i.e., optimal cost-to-go)
- solution \( u_x^* \) is the **optimal control**
- **optimal trajectory** \( z_x^* \) dictated by \( u_x^* \)

Pontryagin Maximum Principle (PMP)

Existing Approach

Solve the forward-backward system\(^2\) for \(0 \leq t \leq T\)

\[
\begin{align*}
\partial_t z^*_x(t) &= -\nabla_p H(t, z^*_x(t), p_x(t)), \\
\partial_t p_x(t) &= \nabla_x H(t, z^*_x(t), p_x(t)), \\
z^*_x(0) &= x, \quad p_x(T) = \nabla_x G(z^*_x(T)),
\end{align*}
\]

(3)

where

- Hamiltonian \(H(t, x, p_x) = \sup_{u_x \in U} \{-p_x \cdot f(t, x, u_x) - L(t, x, u_x)\}\)
- adjoint \(p_x : [0, T] \rightarrow \mathbb{R}^d\)

then notation-wise, we have \(u^*_x(t) = u^*(t, z^*_x(t), p_x(t))\)

Pontryagin Maximum Principle (PMP)

Existing Approach

Solve the forward-backward system\(^2\) for \(0 \leq t \leq T\)

\[
\begin{aligned}
\frac{\partial}{\partial t} z^*_x(t) &= -\nabla_p H(t, z^*_x(t), p_x(t)), \\
\frac{\partial}{\partial t} p_x(t) &= \nabla_x H(t, z^*_x(t), p_x(t)), \\
z^*_x(0) &= x, \quad p_x(T) = \nabla_x G(z^*_x(T)),
\end{aligned}
\tag{3}
\]

where

- Hamiltonian \(H(t, x, p_x) = \sup_{u_x \in U} \{-p_x \cdot f(t, x, u_x) - L(t, x, u_x)\}\)
- adjoint \(p_x : [0, T] \rightarrow \mathbb{R}^d\)

then notation-wise, we have \(u^*_x(t) = u^*(t, z^*_x(t), p_x(t))\)

Comments

- \textit{Local} solution method
  - Solved for a single \(x\)
  - For a new \(x\), need to resolve (3)

- Solving the system is difficult and depends on the initial guess \(p_x(0)\) (if using a shooting method)

Hamilton-Jacobi-Bellman (HJB)

Existing Approach

Solve the HJB PDE\(^3\) (also called *dynamic programming* equations)

\[
\begin{align*}
-\partial_t \Phi(t, x) &= -H(t, x, \nabla_x \Phi(t, x)), \\
\Phi(T, x) &= G(x)
\end{align*}
\]  
(4)

arises from correspondence

\[
p_x(t) = \nabla_x \Phi(t, z^*_x(t))
\]  
(5)

Hamilton-Jacobi-Bellman (HJB)

Existing Approach

Solve the HJB PDE\(^3\) (also called *dynamic programming* equations)

\[
\begin{cases}
-\partial_t \Phi(t, x) = -H(t, x, \nabla_x \Phi(t, x)), \\
\Phi(T, x) = G(x)
\end{cases}
\]

arises from correspondence

\[
p_x(t) = \nabla_x \Phi(t, z^*_x(t))
\]


**Comments**

- *Global* solution method
  - Solved for all \(x\)
  - For a new \(x\), no recomputation
- Need grids to solve (4), which scale poorly to high-dimensions
Our Approach

Motivation

Want:

- Semi-global solution method (from HJB)
  ⇒ one model useful for many initial conditions
  ⇒ method is robust to shocks/disturbances
- High-dimensional (from PMP)
  ⇒ multi-agent problems provide high dimensionality and are easy to visualize
Semi-Global Solution Method
Robust to Shocks

Want: semi-global $\Phi$ (value function)

How to obtain:

- Solve for Hamiltonian $H$
- Replace adjoint $p$ with $\nabla_x \Phi$ using (5)
- Use initial states sampled from Gaussian distribution
- Solve

$$\min_{\Phi} \mathbb{E}_{x \sim N(\mu, \Sigma)} \left\{ \int_0^T L(s, z_x(s), u_x(s)) \, ds + G(z_x(T)) \right\}$$

s.t.

$$\partial_t z_x(t) = u_x(t) = -\nabla_x \Phi(t, z_x(t))$$

Example:

$$\mu = \begin{bmatrix} -2 \\ -2 \\ 2 \\ -2 \end{bmatrix}, \quad \Sigma = I$$
Penalizers

Recall the HJB equations

\[-\partial_t \Phi(t, z_x(t)) = -H(t, z_x(t), \nabla_x \Phi(t, z_x(t))),\]
\[\Phi(T, z_x(T)) = G(z_x(T))\]

Make penalizers

\[c_{HJt, x}(t) = \int_0^t \left| \partial_s \Phi(s, z_x(s)) - H(s, z_x(s), \nabla_x \Phi(s, z_x(s))) \right| \, ds\]
\[c_{HJfin, x} = |\Phi(T, z_x(T)) - G(z_x(T))|\]
\[c_{HJgrad, x} = |\nabla_x \Phi(T, z_x(T)) - \nabla_x G(z_x(T))|\]

HJt penalizer ⇒ few time steps\(^4,5\)

Rewrite time-integrals as part of the ODE

\[
\min_{\Phi} \mathbb{E}_{x \sim \mathcal{N}(\mu, \Sigma)} c_L, x(T) + G(z_x(T)) + \beta_1 c_{\text{HJt}, x}(T) + \beta_2 c_{\text{HJfin}, x} + \beta_3 c_{\text{HJgrad}, x},
\]

subject to

\[
\partial_t \begin{pmatrix}
  z_x(t) \\
  c_L, x(t) \\
  c_{\text{HJt}, x}(t)
\end{pmatrix} = \begin{pmatrix}
  -\nabla_x \Phi(t, z_x(t)) \\
  L(t, z_x(t), \nabla_x \Phi(t, z_x(t))) \\
  \partial_t \Phi(t, z_x(t)) - H(t, z_x(t), \nabla_x \Phi(t, z_x(t)))
\end{pmatrix},
\]

initialized with \( z_x(0) = x \) and \( c_L, x(0) = c_{\text{HJt}, x}(0) = 0 \)

Scalars \( \beta_1, \beta_2, \beta_3 \) are weighted multipliers (NN hyperparameters)
How do we solve this PDE-constrained optimization problem?
How do we solve this PDE-constrained optimization problem?

Blend Neural Networks and Differential Equations

Choose your buzzword: Neural ODEs, Physics-Informed Neural Networks, etc.
Neural Network (NN) Basics

Consider a parameterized function:

\[ C = g(z; \theta) \]

where

- \( z \in \mathbb{R}^d \) is an input item (e.g., the state of the system)
- \( C \in \mathbb{R} \) is the corresponding output (e.g., the value from \( \Phi \))
- \( \theta \in \mathbb{R}^p \) are the parameters/weights of the model \( g \)

Think: Manifold Projection

Motivation: Nonlinear Models

In general, impossible to find a linear separator between points

Goal/Trick

Embed the point in higher dimension or move the points to make them linearly separable

Input Features Transformed (Hidden) Features Output
Single-Layer Example

- $d$ - # features
- $m$ - width

**Features**
$z \in \mathbb{R}^d$

**Weights** ($\theta$)
- $K \in \mathbb{R}^{m \times d}$$
- $w \in \mathbb{R}^m$
- bias $b \in \mathbb{R}$

**Outputs**
- $C \in \mathbb{R}$

**Nonlinearity** $\sigma$
- tanh, sigmoid, etc.

![Diagram of a single-layer neural network](image)

\[ a = \sigma(Kz + b) \]

\[ w^\top a = C \]
Our Network
A Brief Look Under the Hood

We parameterize the value function

\[ a_0 = \sigma(K_0 s + b_0), \]

- space-time inputs \( s = (x, t) \in \mathbb{R}^{d+1} \)

\[ \Phi(s; \theta) = w^\top N(s) + \frac{1}{2} s^\top (A^\top A) s + b^\top s + c, \]

for \( \theta = (w, A, b, c, K_0, K_1, b_0, b_1) \)

\[ N(s) = a_0 + \sigma(K_1 a_0 + b_1) \]

and

\[ a_0 = \sigma(K_0 s + b_0) \]

Our Network

A Brief Look Under the Hood

We parameterize the value function

$$V(s; \theta) = w^\top N(s) + \frac{1}{2} s^\top (A^\top A) s + b^\top s + c,$$

for $\theta = (w, A, b, c, K_0, K_1, b_0, b_1)$

where $N(s) = a_0 + \sigma(K_1 a_0 + b_1)$,

$$a_0 = \sigma(K_0 s + b_0),$$

and

- space-time inputs $s = (x, t) \in \mathbb{R}^{d+1}$
- $N(s): \mathbb{R}^{d+1} \to \mathbb{R}^m$ is a residual neural network (ResNet)$^6$
- element-wise activation function $\sigma(x) = \log(\exp(x) + \exp(-x))$

---

Our Network
A Brief Look Under the Hood

We parameterize the value function with

$$\Phi(s; \theta) = w^\top N(s) + \frac{1}{2} s^\top (A^\top A) s + b^\top s + c,$$

for $$\theta = (w, A, b, c, K_0, K_1, b_0, b_1)$$

where $$N(s) = a_0 + \sigma(K_1 a_0 + b_1),$$

$$a_0 = \sigma(K_0 s + b_0),$$

and

- space-time inputs $$s = (x, t) \in \mathbb{R}^{d+1}$$
- $$N(s): \mathbb{R}^{d+1} \to \mathbb{R}^m$$ is a residual neural network (ResNet)\(^6\)
- element-wise activation function $$\sigma(x) = \log(\exp(x) + \exp(-x))$$
- $$\theta$$ contains the trainable weights: $$w \in \mathbb{R}^m, A \in \mathbb{R}^{10 \times (d+1)}, b \in \mathbb{R}^{d+1}, c \in \mathbb{R}, K_0 \in \mathbb{R}^{m \times (d+1)}, K_1 \in \mathbb{R}^{m \times m},$$ and $$b_0, b_1 \in \mathbb{R}^m.$$

Differential Equations

**Recall:** We are solving

\[
\min_{\Phi} \mathbb{E}_{x \sim \mathcal{N}(\mu, \Sigma)} c_{L,x}(T) + G(z_x(T)) + \beta_1 c_{HJt,x}(T) + \beta_2 c_{HJfin,x} + \beta_3 c_{HJgrad,x},
\]

subject to

\[
\begin{pmatrix}
    z_x(t) \\
    c_{L,x}(t) \\
    c_{HJt,x}(t)
\end{pmatrix}
= \begin{pmatrix}
    -\nabla_x \Phi(t, z_x(t)) \\
    L(t, z_x(t), \nabla_x \Phi(t, z_x(t))) \\
    \partial_t \Phi(t, z_x(t)) - H(t, z_x(t), \nabla_x \Phi(t, z_x(t)))
\end{pmatrix},
\]

initialized with \( z_x(0) = x \) and \( c_{L,x}(0) = c_{HJt,x}(0) = 0 \).
Differential Equations

Which is the same as training the neural ODE

$$\min_{\theta} \mathbb{E}_{x \sim \mathcal{N}(\mu, \Sigma)} c_{L,x}(T) + G(z_x(T)) + \beta_1 c_{HJt,x}(T) + \beta_2 c_{HJfin,x} + \beta_3 c_{HJgrad,x}$$

subject to

$$\partial_t \begin{pmatrix} z_x(t) \\ c_{L,x}(t) \\ c_{HJt,x}(t) \end{pmatrix} = F(t, z_x(t), \nabla_x \Phi(t, z_x(t); \theta)),$$

initialized with $z_x(0) = x$ and $c_{L,x}(0) = c_{HJt,x}(0) = 0$
Training and Numerics

Solving the Minimization / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\theta$
Training and Numerics

Solving the Minimization / Training the Neural ODE:

Iterate through
1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\theta$

ODE solver:
Runge-Kutta 4 ⇒ efficient and accurate

Discretize-then-Optimize Approach:\(^7,^8\)
First, discretize the ODE at time points, then optimize over that discretization
As opposed to optimize-then-discretize, e.g., solve Karush-Kuhn-Tucker then discretize

\(^7\)Gholaminejad, Keutzer, and Biros. “ANODE: Unconditionally Accurate Memory-Efficient…”. 2019.
Training and Numerics

Solving the Minimization / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\theta$

Loss / Objective Function:

$$J(\theta) = \mathbb{E}_{x \sim N(\mu, \Sigma)} c_L, x(T) + G(z, x(T)) + \beta_1 c_{HJt}, x(T) + \beta_2 c_{HJfin}, x + \beta_3 c_{HJgrad}, x$$
Training and Numerics

Solving the Minimiziation / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\theta$

Compute gradient with respect to parameters (chain rule)

Use automatic differentiation\(^9\) to compute $\nabla_{\theta} J$

Training and Numerics

Solving the Minimization / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\theta$

Use ADAM\(^{10}\)

A stochastic subgradient method with momentum
Empirically, ADAM works well in noisy high-dimensional spaces

Results

Small Shock

Large Shock
Discrete optimization approach via forward Euler

\[
\min_{\{u^{(k)}\}} \quad G\left(z^{(n_t)}\right) + h \sum_{k=0}^{n_t-1} L\left(t^{(k)}, z^{(k)}, u^{(k)}\right)
\]

s.t. \[ z^{(k+1)} = z^{(k)} + h f\left(t^{(k)}, z^{(k)}, u^{(k)}\right), \]
\[ z^{(0)} = x \]

where \( h = T / n_t \). We use \( T = 1 \) and \( n_t = 50 \).

This is a \textit{local} approach, whereas the NN is \textit{global}.
Swap Experiments

Two agents swap positions with hard corridor\textsuperscript{11}

Twelve agents swap positions\textsuperscript{11}

Addressing Curse of Dimensionality\textsuperscript{12}

Setup:
- Take subproblems of the 12-agent swap experiment (2, 3, 4, 5, and 6 pairs of agents)
- Train the smallest NN we can that achieves a fixed suboptimality (relative to baseline)

The number of parameters grows linearly with problem dimension $d$

\textsuperscript{12}Bellman. \textit{Dynamic Programming}. 1957.
Swarms Trajectory Planning

50 3-dimensional agents with obstacles

Quadcopter Problem

More complicated dynamics

Controls: thrust $u$, torques $\tau_\psi, \tau_\theta, \tau_\phi$

\[
\dot{x} = v_x \\
\dot{y} = v_y \\
\dot{z} = v_z \\
\dot{\psi} = v_\psi \\
\dot{\theta} = v_\theta \\
\dot{\phi} = v_\phi \\
\dot{v}_x = \frac{u}{m} f_7(\psi, \theta, \phi) \\
\dot{v}_y = \frac{u}{m} f_8(\psi, \theta, \phi) \\
\dot{v}_z = \frac{u}{m} f_9(\theta, \phi) - g \\
\dot{v}_\psi = \tau_\psi \\
\dot{v}_\theta = \tau_\theta \\
\dot{v}_\phi = \tau_\phi
\]

where

\[
\begin{align*}
    f_7(\psi, \theta, \phi) &= \sin(\psi) \sin(\phi) + \cos(\psi) \sin(\theta) \cos(\phi), \\
    f_8(\psi, \theta, \phi) &= -\cos(\psi) \sin(\phi) + \sin(\psi) \sin(\theta) \cos(\phi), \\
    f_9(\theta, \phi) &= \cos(\theta) \cos(\phi).
\end{align*}
\]

Quadcopter Comparison with Baseline

![Graph showing comparison between NN and Baseline controls over time](image)

- Control: NN $u$, Baseline $u$
- Control: NN $\tau_{\psi}$, Baseline $\tau_{\psi}$
- Control: NN $\tau_{\theta}$, Baseline $\tau_{\theta}$
- Control: NN $\tau_{\phi}$, Baseline $\tau_{\phi}$

Time ($n_t = 50$)
Review

- **Want to solve**
  - High-Dimensional Control Problems
  - Semi-Globally

- Combine Pontryagin Maximum Principle and Hamilton-Jacobi-Bellman approaches

- Parameterize the value function $\Phi$ with a neural network

- Solve trajectory problem in 150 dimensions

- Solve quadcopter problem with complicated dynamics

- Demonstrate shock-robustness
Conclusions

- **Parameterizing** $\Phi \\
  \Rightarrow$ **extrapolation capabilities**

- **HJB penalizers improve training**

- **Lagrangian coordinates (no grids) help scalability**

---

Coming Soon:

- DO, L Nurbekyan, X Li, S Wu Fung, S Osher, L Ruthotto
  
  *A Neural Network Approach Applied to Multi-Agent Optimal Control*
  

- Code: github.com/EmoryMLIP/NeuralOC
  
  Simulations: imgur.com/a/eWr6sUb
Future Work

- More rigorous experiments with many 12-d quadcopters
- Deployment on actual quadcopters
- Combination with existing methods and sensors
Future Work

- More rigorous experiments with many 12-d quadcopters
- Deployment on actual quadcopters
- Combination with existing methods and sensors

Questions?


References II


Onken, Derek et al. (2020). “OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport”. In: AAAI.