A Neural Network Approach Applied to Multi-Agent Optimal Control

European Control Conference
June 2021

Derek Onken
Emory University
derekonken.com
Collaborators and Acknowledgments

Levon Nurbekyan
UCLA

Xingjian Li
Emory

Samy Wu Fung
UCLA

Lars Ruthotto
Emory

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**
  - Two-Agent Corridor Problem
  - 150-Dimensional Swarm Trajectory Planning

- **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 \( x_1, \ldots, x_n \in \mathbb{R}^q \Rightarrow \mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^d \)

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

\[
\begin{align*}
\mathbf{z}_x(0) = \mathbf{x} &= \begin{bmatrix} -2 \\ -2 \\ 2 \\ -2 \end{bmatrix} & \text{agent 1} \\
&\begin{bmatrix} -2 \\ -2 \\ 2 \\ -2 \end{bmatrix} & \text{agent 2}
\end{align*}
\]

\[
\mathbf{y} = \begin{bmatrix} 2 \\ 2 \\ -2 \\ 2 \end{bmatrix}
\]

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

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

\[
\begin{align*}
\partial_t z_x(t) &= f(t, z_x(t), u_x(t)), \quad z_x(0) = x \\
\text{For Corridor:} & \quad = u_x(t) \text{ (the velocity)}
\end{align*}
\]

where

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

Goal: Find the control that incurs minimal cost

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

- \( \Phi(t, x) \in \mathbb{R} \) 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 \(0 \leq t \leq T\)

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

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 \(0 \leq t \leq T\)
\[
\begin{aligned}
\frac{\partial_t z_x^*(t)}{} &= -\nabla_p H(t, z_x^*(t), p_x(t)), \\
\frac{\partial_t p_x(t)}{} &= \nabla_x H(t, z_x^*(t), p_x(t)), \\
z_x^*(0) &= x, \quad p_x(T) = \nabla G(z_x^*(T)),
\end{aligned}
\]

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
- **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)

\[
\begin{aligned}
-\partial_t \Phi(t, x) &= -H(t, x, \nabla \Phi(t, x)), \\
\Phi(T, x) &= G(x)
\end{aligned}
\]  

(4)

arises from correspondence

\[
p_x(t) = \nabla \Phi(t, x^*_x(t))
\]  

(5)

Hamilton-Jacobi-Bellman (HJB)

Existing Approach

Solve the HJB PDE\(^3\)
(also called *dynamic programming* equations)
\[
\begin{align*}
    -\partial_t \Phi(t, x) &= -H(t, x, \nabla \Phi(t, x)), \\
    \Phi(T, x) &= G(x)
\end{align*}
\]
(4)

arises from correspondence
\[
p_x(t) = \nabla \Phi(t, z^*_x(t))
\]
(5)

**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 $\nabla \Phi$ 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.

$$\partial_t z_x(t) = -\nabla_p H(t, z_x(t), \nabla \Phi(t, z_x(t))) = -\nabla \Phi(t, z_x(t))$$

For Corridor

Example:

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

Recall the HJB equations

\[-\partial_t \Phi(t, z_x(t)) = -H(t, z_x(t), \nabla \Phi(t, z_x(t))), \]

\[\Phi(T, z_x(T)) = G(z_x(T))\]

Make penalizers

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

\[c_{HJfin,x} = \left| \Phi(T, z_x(T)) - G(z_x(T)) \right|\]

\[c_{HJgrad,x} = \left| \nabla \Phi(T, z_x(T)) - \nabla G(z_x(T)) \right|\]

\[HJt\text{ penalizer} \Rightarrow \text{few time steps}^{4,5}\]

---


Formulation

Rewrite time-integrals as part of the ODE

$$\min_{\Phi} \mathbb{E}_{x \sim \mathcal{N}(\mu, \Sigma)} c_L, x(T) + G(z_x(T)) + \beta_1 c_{HJ_t, x}(T) + \beta_2 c_{HJ_{\text{fin}}, x} + \beta_3 c_{HJ_{\text{grad}}, x},$$

subject to

$$\partial_t \begin{pmatrix} z_x(t) \\ c_L, x(t) \\ c_{HJ_t, x}(t) \end{pmatrix} = \begin{pmatrix} -\nabla_p H(t, z_x(t), \nabla \Phi(t, z_x(t))) \\ L_x(t) \\ \partial_t \Phi(t, z_x(t)) - H(t, z_x(t), \nabla \Phi(t, z_x(t))) \end{pmatrix}, \begin{pmatrix} z_x(0) \\ c_L, x(0) \\ c_{HJ_t, x}(0) \end{pmatrix} = \begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix}.$$

where, by the envelope formula,

$$L_x(t) = \nabla \Phi(t, z_x(t)) \cdot \nabla_p H(t, z_x(t), \nabla \Phi(t, z_x(t))) - H(t, z_x(t), \nabla \Phi(t, z_x(t)))$$

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.
We parameterize the value function

\[ a_0 = \sigma(K_0s + b_0), \]

- space-time inputs \( s = (x, t) \in \mathbb{R}^{d+1} \)
- element-wise activation function \( \sigma(x) = \log(\exp(x) + \exp(-x)) \)

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

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

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

\( \theta \) 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 \( b_0, b_1 \in \mathbb{R}^m \).

\[ ^6 \text{He et al. “Deep Residual Learning for Image Recognition”. 2016.} \]
Our Network
A Brief Look Under the Hood

We parameterize the value function

\[ N(s) = a_0 + \sigma(K_1 a_0 + b_1), \]
\[ a_0 = \sigma(K_0 s + b_0), \]

and

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

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

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

subject to

\[
\partial_t \begin{pmatrix} z_{x}(t) \\ c_{L,x}(t) \\ c_{HJt,x}(t) \end{pmatrix} = \begin{pmatrix} -\nabla p H(t, z_{x}(t), \nabla \Phi(t, z_{x}(t))) \\ L_{x}(t) \\ \partial_t \Phi(t, z_{x}(t)) - H(t, z_{x}(t), \nabla \Phi(t, z_{x}(t))) \end{pmatrix}, \begin{pmatrix} z_{x}(0) \\ c_{L,x}(0) \\ c_{HJt,x}(0) \end{pmatrix}, = \begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix}.
\]
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

$$\partial_t \begin{pmatrix} z_{x}(t) \\ c_{L,x}(t) \\ c_{HJt,x}(t) \end{pmatrix} = F(t, z_{x}(t), \nabla \Phi(t, z_{x}(t); \theta)), \quad \begin{pmatrix} z_{x}(0) \\ c_{L,x}(0) \\ c_{HJt,x}(0) \end{pmatrix}, = \begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix}.$$
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 $\Rightarrow$ 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:

$$J(\theta) = \mathbb{E}_{x \sim \mathcal{N}(\mu, \Sigma)} c_{L, x}(T) + G(z_{x}(T)) + \beta_1 c_{HJ_t,x}(T) + \beta_2 c_{HJfin,x} + \beta_3 c_{HJgrad,x}$$
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 $\theta$

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
Baseline

Corridor

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. \quad 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 local approach, whereas the NN is global.
Swarm Trajectory Planning

50 3-dimensional agents with obstacles\textsuperscript{11}

In 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**

- **Demonstrate shock-robustness**

---

**Other Work:**

- D Onken, L Nurbekyan, X Li, S Wu Fung, S Osher, L Ruthotto
  - *A Neural Network Approach for High-Dimensional Optimal Control*

  - Code: github.com/donken/NeuralOC
  - Simulations: imgur.com/a/eWr6sUb


References