Optimal Control

Numerical Methods for Deep Learning

Learning Objective: Optimal Control

In this module we discuss optimal control methods for ResNet training

Learning tasks:

- regression
- segmentation
- classification

Concepts from optimal control

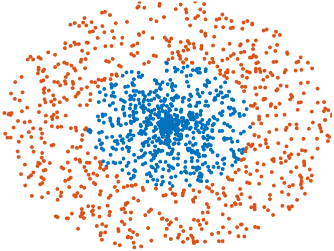
- optimize-then-discretize vs. discretize-then-optimize [3]
- backpropagation vs. adjoint equations

Residual Network as a Path Planning Problem

Change in notation: Moving forward it is more convenient to define $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ (transpose data matrix) and $\mathbf{C} \in \mathbb{R}^{n_c \times n}$.

$$\partial_t \mathbf{Y}(t) = \sigma(\mathbf{K}(t)\mathbf{Y}(t) + b(t)) \quad \mathbf{Y}(0) = \mathbf{Y}_0$$

The goal is to plan a path such that the transformed features, $\mathbf{Y}(T)$, can be linearly separated.



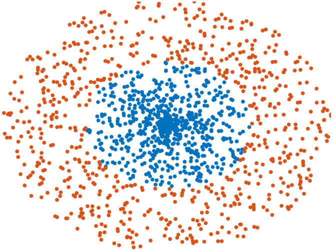
input features, $\mathbf{Y}(0)$

Residual Network as a Path Planning Problem

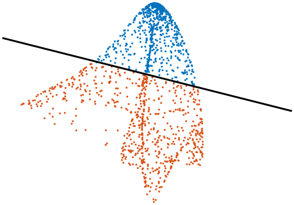
Change in notation: Moving forward it is more convenient to define $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ (transpose data matrix) and $\mathbf{C} \in \mathbb{R}^{n_c \times n}$.

$$\partial_t \mathbf{Y}(t) = \sigma(\mathbf{K}(t)\mathbf{Y}(t) + b(t)) \quad \mathbf{Y}(0) = \mathbf{Y}_0$$

The goal is to plan a path such that the transformed features, $\mathbf{Y}(T)$, can be linearly separated.



input features, $\mathbf{Y}(0)$



transformed features $\mathbf{Y}(T)$

Example: The Adjoint Equation

Simplified learning problem: one example $(\mathbf{y}_0, \mathbf{c})$, no weights for classifier, no regularizer, $\mathbf{y}(0, \boldsymbol{\theta}) = \mathbf{y}_0$

$$\min_{\boldsymbol{\theta}} \operatorname{loss}(\mathbf{y}(1, \boldsymbol{\theta}), \mathbf{c}) \quad \text{with} \quad \partial_t \mathbf{y}(t, \boldsymbol{\theta}) = f(\mathbf{y}(t), \boldsymbol{\theta}(t)).$$

Example: The Adjoint Equation

Simplified learning problem: one example $(\mathbf{y}_0, \mathbf{c})$, no weights for classifier, no regularizer, $\mathbf{y}(0, \boldsymbol{\theta}) = \mathbf{y}_0$

$$\min_{\boldsymbol{\theta}} \mathrm{loss}(\mathbf{y}(1, \boldsymbol{\theta}), \mathbf{c}) \;\text{ with }\; \partial_t \mathbf{y}(t, \boldsymbol{\theta}) = f(\mathbf{y}(t), \boldsymbol{\theta}(t)).$$

Use adjoint method to compute gradient of objective w.r.t. $\boldsymbol{\theta}$

$$\frac{\partial \mathrm{loss}}{\partial \boldsymbol{\theta}}(t) = \left(\frac{\partial f}{\partial \boldsymbol{\theta}}(\mathbf{y}(t, \boldsymbol{\theta}), \boldsymbol{\theta}(t)) \right)^\top \mathbf{z}(t)$$

where $\mathbf{z}$ satisfies the adjoint method ($-\partial_t \leadsto$ backward in time)

$$-\partial_t \mathbf{z}(t, \boldsymbol{\theta}) = \left(\frac{\partial f}{\partial \mathbf{y}}(\mathbf{y}(t, \boldsymbol{\theta}), \boldsymbol{\theta}(t)) \right)^\top \mathbf{z}(t),$$

$$\mathbf{z}(1, \boldsymbol{\theta}) = \frac{\partial \mathrm{loss}}{\partial \mathbf{y}}(\mathbf{y}(1, \boldsymbol{\theta}), \mathbf{c}).$$

Example: The Adjoint Equation

Simplified learning problem: one example $(\mathbf{y}_0, \mathbf{c})$, no weights for classifier, no regularizer, $\mathbf{y}(0, \boldsymbol{\theta}) = \mathbf{y}_0$

$$\min_{\boldsymbol{\theta}} \mathrm{loss}(\mathbf{y}(1, \boldsymbol{\theta}), \mathbf{c}) \text{ with } \partial_t \mathbf{y}(t, \boldsymbol{\theta}) = f(\mathbf{y}(t), \boldsymbol{\theta}(t)).$$

Use adjoint method to compute gradient of objective w.r.t. $\boldsymbol{\theta}$

$$\frac{\partial \mathrm{loss}}{\partial \boldsymbol{\theta}}(t) = \left(\frac{\partial f}{\partial \boldsymbol{\theta}}(\mathbf{y}(t, \boldsymbol{\theta}), \boldsymbol{\theta}(t)) \right)^\top \mathbf{z}(t)$$

where $\mathbf{z}$ satisfies the adjoint method ($-\partial_t \rightsquigarrow$ backward in time)

$$-\partial_t \mathbf{z}(t, \boldsymbol{\theta}) = \left(\frac{\partial f}{\partial \mathbf{y}}(\mathbf{y}(t, \boldsymbol{\theta}), \boldsymbol{\theta}(t)) \right)^\top \mathbf{z}(t),$$

$$\mathbf{z}(1, \boldsymbol{\theta}) = \frac{\partial \mathrm{loss}}{\partial \mathbf{y}}(\mathbf{y}(1, \boldsymbol{\theta}), \mathbf{c}).$$

note: $\mathbf{y}(t)$ needed to solve adjoint equation (memory!)

Diff→Disc vs. Disc→Diff [3]

$$\min_{\boldsymbol{\theta}} \mathrm{loss}(\mathbf{Y}(1,\boldsymbol{\theta}),\mathbf{C}) \quad \text{with} \quad \partial_t \mathbf{Y}(t,\boldsymbol{\theta}) = f(\mathbf{Y}(t),\boldsymbol{\theta}(t)).$$

Diff→Disc vs. Disc→Diff [3]

$$\min_{\boldsymbol{\theta}} \operatorname{loss}(\mathbf{Y}(1,\boldsymbol{\theta}),\mathbf{C}) \;\text{ with }\; \partial_t \mathbf{Y}(t,\boldsymbol{\theta}) = f(\mathbf{Y}(t),\boldsymbol{\theta}(t)).$$

First-Differentiate-then-Discretize (Diff→Disc)

- Keep $\boldsymbol{\theta}, \mathbf{Y}$ continuous in time
- Euler-Lagrange-Equations $\rightsquigarrow$ adjoint equation ($\approx$ backprop)
- flexible choice of ODE solver in forward and adjoint
- gradients only useful if fwd and adjoint solved well
- use optimization to obtain discrete solution of ELE

Diff→Disc vs. Disc→Diff [3]

$$\min_{\boldsymbol{\theta}} \operatorname{loss}(\mathbf{Y}(1,\boldsymbol{\theta}),\mathbf{C}) \;\text{ with }\; \partial_t \mathbf{Y}(t,\boldsymbol{\theta}) = f(\mathbf{Y}(t),\boldsymbol{\theta}(t)).$$

First-Differentiate-then-Discretize (Diff→Disc)

- Keep $\boldsymbol{\theta}, \mathbf{Y}$ continuous in time
- Euler-Lagrange-Equations $\rightsquigarrow$ adjoint equation ($\approx$ backprop)
- flexible choice of ODE solver in forward and adjoint
- gradients only useful if fwd and adjoint solved well
- use optimization to obtain discrete solution of ELE

First-Discretize-then-Differentiate (Disc→Diff)

- Discretize $\boldsymbol{\theta}, \mathbf{Y}$ in time (could use different grids)
- Differentiate objective (e.g., use automatic differentiation)
- gradients related to adjoints but no choice of solver
- gradients useful even if discretization is inaccurate
- use nonlinear optimization tools to approximate minimizer

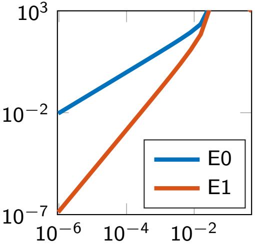
Example: Gradient Test Disc→Diff

Goal: Find weights of neural network $F(\mathbf{u}, \theta)$ such that

$$\partial_t \mathbf{u} = F(\mathbf{u}, \theta), \quad \mathbf{u}(0) = \mathbf{u}_0$$

fits true ODE at $0 < t_1 < t_2 < \cdots < t_n \leq 1.5$; details Sec. 8 from paper below.

Question: How does accuracy of ODE solvers impact the quality of gradient?



$\epsilon_{\mathrm{rel}} = 10^{-7}, \epsilon_{\mathrm{abs}} = 10^{-9}$

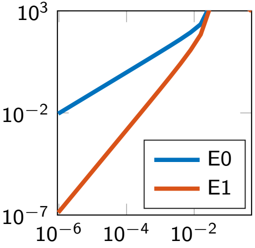
Example: Gradient Test Disc→Diff

Goal: Find weights of neural network $F(\mathbf{u}, \theta)$ such that

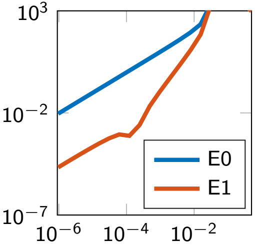
$$\partial_t \mathbf{u} = F(\mathbf{u}, \theta), \quad \mathbf{u}(0) = \mathbf{u}_0$$

fits true ODE at $0 < t_1 < t_2 < \cdots < t_n \leq 1.5$; details Sec. 8 from paper below.

Question: How does accuracy of ODE solvers impact the quality of gradient?

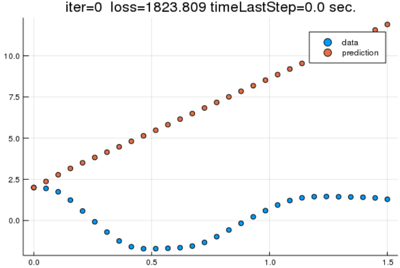


$\epsilon_{\mathrm{rel}} = 10^{-7}, \epsilon_{\mathrm{abs}} = 10^{-9}$

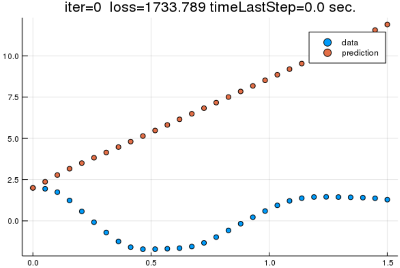


$\epsilon_{\mathrm{rel}} = 10^{-2}, \epsilon_{\mathrm{abs}} = 10^{-2}$

Example: Training Disc→Diff

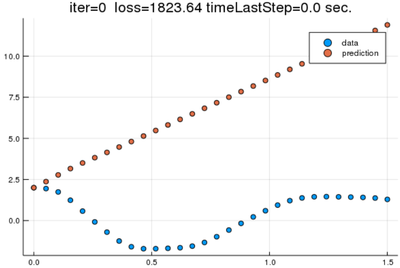


Neural ODE,
$\epsilon_{\text{rel}} = 10^{-7}, \epsilon_{\text{abs}} = 10^{-9}$

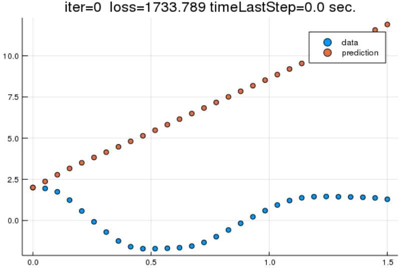


Disc→Diff, RK4, 30 steps

Training: ADAM with default setting, same initialization

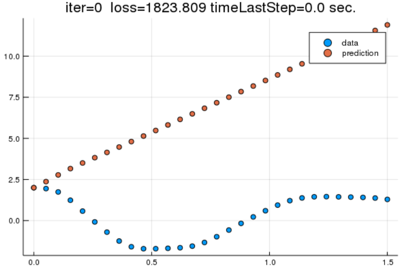


Neural ODE,
$\epsilon_{\text{rel}} = 10^{-7}, \epsilon_{\text{abs}} = 10^{-9}$

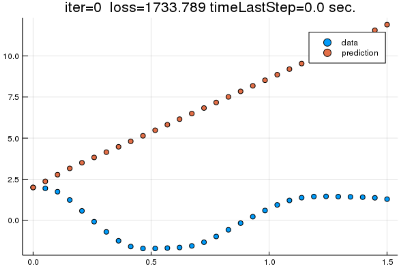


Disc→Diff, RK4, 30 steps

Training: ADAM with default setting, same initialization



Neural ODE,
$\epsilon_{\mathrm{rel}} = 10^{-2}, \epsilon_{\mathrm{abs}} = 10^{-2}$



Disc$\rightarrow$Diff, RK4, 30 steps

Training: ADAM with default setting, same initialization

Residual Network - Forward Propagation

Idea: Obtain forward propagation by discretizing the ODE

$$\partial_t \mathbf{Y} = \sigma(\mathbf{K}\mathbf{Y} + b) \quad \mathbf{Y}(0) = \mathbf{Y}_0$$

Residual Network - Forward Propagation

Idea: Obtain forward propagation by discretizing the ODE

$$\partial_t \mathbf{Y} = \sigma(\mathbf{K}\mathbf{Y} + b) \quad \mathbf{Y}(0) = \mathbf{Y}_0$$

Example: Use forward Euler method

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma(\mathbf{K}_j \mathbf{Y}_j + b_j)$$

Here: $\mathbf{Y}_j$ is called the *state*, $\mathbf{K}_j, b_j$ are *controls*, and $h > 0$ is time step size.

Residual Network - Forward Propagation

Idea: Obtain forward propagation by discretizing the ODE

$$\partial_t \mathbf{Y} = \sigma(\mathbf{K}\mathbf{Y} + b) \quad \mathbf{Y}(0) = \mathbf{Y}_0$$

Example: Use forward Euler method

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma(\mathbf{K}_j\mathbf{Y}_j + b_j)$$

Here: $\mathbf{Y}_j$ is called the *state*, $\mathbf{K}_j, b_j$ are *controls*, and $h > 0$ is time step size.

More general forward propagation

$$\mathbf{Y}_{j+1} = \mathbf{P}_j\mathbf{Y}_j + h\sigma(\mathbf{K}_j\mathbf{Y}_j + b_j), \qquad \mathbf{P}_j \text{ fixed.}$$

Allows for changing resolution and width (and classical neural networks).

Residual Network - Optimization Problem

Note: Only final state used in loss

$$\min_{\mathbf{W},\mathbf{K}_{0,\ldots,N-1},b_{0,\ldots,N-1}} E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_{0,\ldots,N-1}, b_{0,\ldots,N-1}), \mathbf{C}^{\mathrm{obs}}\right)$$

Residual Network - Optimization Problem

Note: Only final state used in loss

$$\min_{\mathbf{W},\mathbf{K}_{0,\ldots,N-1},b_{0,\ldots,N-1}} E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_{0,\ldots,N-1}, b_{0,\ldots,N-1}), \mathbf{C}^{\mathrm{obs}}\right)$$

Need to differentiate

- E w.r.t $\mathbf{W}$
- $\mathcal{S}$ w.r.t $\mathbf{Y}_N$
- $\mathbf{Y}_N$ w.r.t control variables $(\mathbf{K}_{0,\ldots,N-1}, b_{0,\ldots,N-1})$

Residual Network - Optimization Problem

Note: Only final state used in loss

$$\min_{\mathbf{W},\mathbf{K}_{0,\ldots,N-1},b_{0,\ldots,N-1}} E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_{0,\ldots,N-1}, b_{0,\ldots,N-1}), \mathbf{C}^{\mathrm{obs}}\right)$$

Need to differentiate

- E w.r.t $\mathbf{W}$
- $\mathcal{S}$ w.r.t $\mathbf{Y}_N$
- $\mathbf{Y}_N$ w.r.t control variables $(\mathbf{K}_{0,\ldots,N-1}, b_{0,\ldots,N-1})$

Having these, apply chain rule to get, e.g.,

$$\nabla_{\mathbf{K}_j} E = \left(\mathbf{J}_{\mathbf{K}_j}\mathbf{Y}_N\right)^\top \nabla_{\mathbf{Y}_N} E$$

How? Adjoint method [1, 2] (more general than back propagation [4])

Computing Derivatives - Sensitivity Equation

Idea: Differentiate the forward propagation (forward Euler) with respect to $\mathbf{K}_i$ for fixed $0 \leq i \leq N$. Note that

$$\mathbf{J}_{\mathbf{K}_i} \mathbf{Y}_j = 0, \quad \text{for} \quad j \leq i.$$

Computing Derivatives - Sensitivity Equation

Idea: Differentiate the forward propagation (forward Euler) with respect to $\mathbf{K}_i$ for fixed $0 \le i \le N$. Note that

$$\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_j = 0, \quad \text{for} \quad j \le i.$$

Next, note that

$$\mathbf{J}_{\mathbf{K}_j}\mathbf{Y}_{i+1} = h\mathrm{diag}(\sigma'(\mathbf{K}_i\mathbf{Y}_i + b_i))(\mathbf{Y}_i^\top \otimes \mathbf{I})$$

Computing Derivatives - Sensitivity Equation

Idea: Differentiate the forward propagation (forward Euler) with respect to $\mathbf{K}_i$ for fixed $0 \leq i \leq N$. Note that

$$\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_j = 0, \quad \text{for} \quad j \leq i.$$

Next, note that

$$\mathbf{J}_{\mathbf{K}_j}\mathbf{Y}_{i+1} = h\mathrm{diag}(\sigma'(\mathbf{K}_i\mathbf{Y}_i + b_i))(\mathbf{Y}_i^\top \otimes \mathbf{I})$$

Continuing like this, gives for the final state:

$$\begin{aligned}
\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N &= \mathbf{P}_{N-1}\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_{N-1} \\
&+ h\mathrm{diag}(\sigma'(\cdots))\left((\mathbf{I} \otimes \mathbf{K}_{N-1})\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_{N-1}\right)
\end{aligned}$$

Computing Derivatives - Sensitivity Equation

Idea: Differentiate the forward propagation (forward Euler) with respect to $\mathbf{K}_i$ for fixed $0 \leq i \leq N$. Note that

$$\mathbf{J}_{\mathbf{K}_i} \mathbf{Y}_j = 0, \quad \text{for} \quad j \leq i.$$

Next, note that

$$\mathbf{J}_{\mathbf{K}_j} \mathbf{Y}_{i+1} = h \mathrm{diag}(\sigma'(\mathbf{K}_i \mathbf{Y}_i + b_i))(\mathbf{Y}_i^\top \otimes \mathbf{I})$$

Continuing like this, gives for the final state:

$$\begin{aligned} \mathbf{J}_{\mathbf{K}_i} \mathbf{Y}_N &= \mathbf{P}_{N-1} \mathbf{J}_{\mathbf{K}_i} \mathbf{Y}_{N-1} \\ &+ h \mathrm{diag}(\sigma'(\cdots)) \left((\mathbf{I} \otimes \mathbf{K}_{N-1}) \mathbf{J}_{\mathbf{K}_i} \mathbf{Y}_{N-1} \right) \end{aligned}$$

Next: Write this as a block triangular **linear** system.

Computing Derivatives - Sensitivity Equations

Block triangular **linear** system for the gradients

$$\begin{pmatrix} \mathbf{I} & & & & \\ -\mathbf{T}_{i+1} & \mathbf{I} & & & \\ & \ddots & \ddots & & \\ & & & -\mathbf{T}_{N-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_{i+1} \\ \\ \\ \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{R}_i \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\mathbf{T}_j = \mathbf{P}_j + h\mathrm{diag}(\sigma'(\mathbf{K}_j\mathbf{Y}_j + b_j))(\mathbf{I} \otimes \mathbf{K}_j)$$

and

$$\mathbf{R}_i = h\mathrm{diag}(\sigma'(\mathbf{K}_i\mathbf{Y}_i + b_i))(\mathbf{Y}_i^\top \otimes \mathbf{I}).$$

Computing Derivatives - Sensitivity Equation

Block triangular **linear** system for the gradients

$$\underbrace{\begin{pmatrix} \mathbf{I} & & & & \\ -\mathbf{T}_{i+1} & \mathbf{I} & & & \\ & \ddots & \ddots & & \\ & & & -\mathbf{T}_{N-1} & \mathbf{I} \end{pmatrix}}_{=\mathbf{T}} \underbrace{\begin{pmatrix} \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_{i+1} \\ \\ \\ \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N \end{pmatrix}}_{=\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}} = \underbrace{\begin{pmatrix} \mathbf{R}_i \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{=\mathbf{R}}$$

To compute matrix-vector product $(\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N)\,\mathbf{v}$

- Multiply $\mathbf{R}\,\mathbf{v}$
- Solve (forward propagate) $\mathbf{T}\,\mathbf{J}_{\mathbf{K}_i}\mathbf{Y} = \mathbf{R}\,\mathbf{v}$
- Extract the last time step

Σ: Optimal Control

Biggest question: Continuous vs. discrete

$$\min_{\mathbf{W},\mathbf{Y}(T,\boldsymbol{\theta})} E\left(\mathbf{WY}(T,\boldsymbol{\theta}),\mathbf{C}^{\mathrm{obs}}\right) \quad \text{vs.} \quad \min_{\mathbf{W},\mathbf{Y}_N(\boldsymbol{\theta})} E\left(\mathbf{WY}_N(\boldsymbol{\theta}),\mathbf{C}^{\mathrm{obs}}\right)$$

Continuous model

+ can help initialization (easy to add layers)
+ simplifies analysis and insight
+ inspires better architectures (discrete!)
- high accuracy needs high computational costs
- meaningful (dynamics not derived from 1st principles?)

Discrete model

+ back propagation easier than solving adjoint equations
+ accurate gradients even for large time steps
+ computationally more efficient
- may 'overfit' on a given discretization
- need careful discretization

References

[1] G. A. Bliss. The use of adjoint systems in the problem of differential corrections for trajectories. *JUS Artillery*, 51:296–311, 1919.

[2] A. Borzì and V. Schulz. *Computational optimization of systems governed by partial differential equations*, volume 8. SIAM, Philadelphia, PA, 2012.

[3] D. Onken and L. Ruthotto. Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows. *arXiv.org*, May 2020.

[4] D. Rumelhart, G. Hinton, and J. Williams, R. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.