A Numerical Analysis Perspective on Deep Neural Networks

Machine Learning for Physics and the Physics of Learning

Los Angeles, September, 2019

Lars Ruthotto

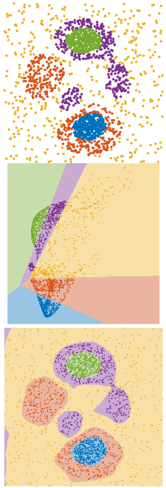
Departments of Mathematics and Computer Science, Emory University

lruthotto@emory.edu

@lruthotto

Agenda: Numerical Analysis of Deep Neural Networks

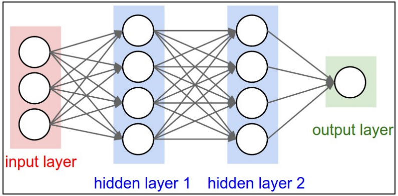
- **Notation: Deep Learning**
- **Case 1: Invertibility**
- **Case 2: Time Integrators**
  - Example: higher order or conservative?
- **Case 3: Discretize-then-Optimize**
  - Example: Neural ODEs
- **Case 4: Ill-conditioning**
  - Example: Single layer neural network
- **Conclusion and Summary**



Key question: What can numerical analysts do in the age of ML?

Deep Learning

Deep Learning Revolution (?)

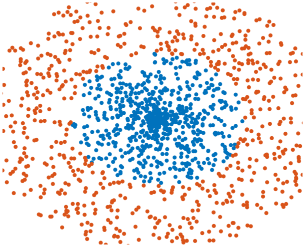


$$\begin{cases} \mathbf{Y}_{j+1} &= \sigma(\mathbf{K}_j\mathbf{Y}_j + \mathbf{b}_j) \\ \mathbf{Y}_{j+1} &= \mathbf{Y}_j + \sigma(\mathbf{K}_j\mathbf{Y}_j + \mathbf{b}_j) \\ \mathbf{Y}_{j+1} &= \mathbf{Y}_j + \sigma\left(\mathbf{K}_{j,2}\sigma(\mathbf{K}_{j,1}\mathbf{Y}_j + \mathbf{b}_{j,1}) + \mathbf{b}_{j,2}\right) \\ &\vdots \end{cases}$$

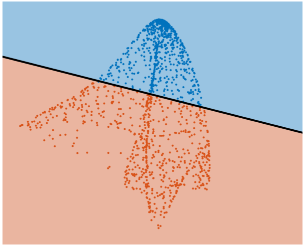
(Notation: $\mathbf{Y}_j$: features, $\mathbf{K}_j, \mathbf{b}_j$: weights, σ : activation)

- deep learning: use neural networks (from $\approx$ 1950's) with many hidden layers
- able to ”learn” complicated patterns from data
- applications: image classification, face recognition, segmentation, driverless cars, ...
- recent success fueled by: massive data sets, computing power
- A few recent references:
  - **A radical new neural network design could overcome big challenges in AI**, MIT Tech Review '18
  - Data Scientist: Sexiest Job of the 21st Century, Harvard Business Rev '17

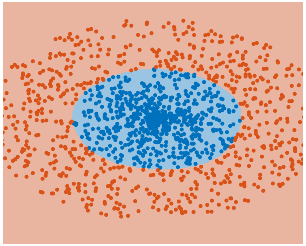
Optimal Control Framework for Deep Learning



training data, $\mathbf{Y}_0, \mathbf{C}$



prop. features, $\mathbf{Y}(T), \mathbf{C}$



classification result

Supervised Deep Learning Problem

Given training data, $\mathbf{Y}_0$, and labels, $\mathbf{C}$, find **network parameters** θ and **classification weights** $\mathbf{W}$, μ such that the DNN predicts the data-label relationship (and generalizes to new data), i.e., solve

$$\text{minimize}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \qquad \text{loss}[g(\mathbf{W}+\boldsymbol{\mu}), \mathbf{C}] + \text{regularizer}[\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}]$$

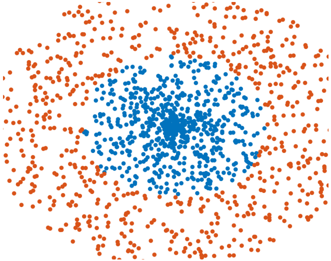
Deep Residual Neural Networks (simplified)

Award-winning forward propagation

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\mathbf{K}_{j,2}\sigma(\mathbf{K}_{j,1}\mathbf{Y}_j + \mathbf{b}_j), \quad \forall\, j = 0, 1, \ldots, N-1.$$

ResNet is forward Euler discretization of

$$\partial_t \mathbf{y}(t) = \mathbf{K}_2(t)\sigma\left(\mathbf{K}_1(t)\mathbf{y}(t) + \mathbf{b}(t)\right), \qquad \mathbf{y}(0) = \mathbf{y}_0.$$

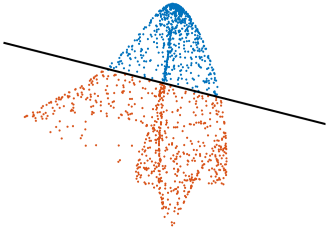


input features, $\mathbf{Y}_0$

Notation: $\boldsymbol{\theta}(t) = (\mathbf{K}_1(t), \mathbf{K}_2(t), \mathbf{b}(t))$ and

$$\partial_t \mathbf{y}(t) = f(\mathbf{y}, \boldsymbol{\theta}(t)), \quad \mathbf{y}(0) = \mathbf{y}_0$$

where $\quad f(\mathbf{y}, \boldsymbol{\theta}) = \mathbf{K}_2(t)\sigma\left(\mathbf{K}_1(t)\mathbf{y}(t) + \mathbf{b}(t)\right).$



propagated features, $\mathbf{Y}_N$

K. He, X. Zhang, S. Ren, and J. Sun
Deep residual learning for image recognition.
IEEE Conf. on CVPR, 770–778, 2016.

(Some) Related Work

DNNs as (stochastic) Dynamical Systems

- **Weinan E**, *Proposal on ML via Dynamical Systems*, Commun. Math. Stat., 5(1), 2017.
- **E Haber, LR**, *Stable Architectures for DNNs*, Inverse Problems, 2017.
- **Q. Li, L. Chen, C. Tai, Weinan E**, *Maximum Principle Based Algorithms*, arXiv, 2017.
- **B. Wang, B. Yuan, Z. Shi, S. Osher**, *ResNets Ensemble via the Feynman-Kac Formalism*, arXiv, 2018.

Numerical Time Integrators

- **Y. Lu, A. Zhong, Q. Li, B. Dong**, *Beyond Finite Layer DNNs*, arXiv, 2017.
- **B. Chang, L. Meng, E. Haber, LR, D. Begert, E. Holtham**, *Reversible architectures for DNNs*, AAAI, 2018.
- **T. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud**, *Neural ODEs*, NeurIPS, 2018.
- **E. Haber, K. Lensink, E. Treister, LR**, *IMEXnet: Forward Stable DNN*. ICML, 2019.

Optimal Control

- **S. Günther, LR, J.B. Schroder, E.C. Cyr, N.R. Gauger**, *Layer-parallel training of ResNets*, arXiv, 2018.
- **A. Gholami, K. Keutzer, G. Biros**, *ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs*, arXiv, 2019.
- **T. Zhang, Z. Yao, A. Gholami, K. Keutzer, J. Gonzalez, G. Biros, M. Mahoney**, *ANODEV2: A Coupled Neural ODE Evolution Framework*, arXiv, 2019.

PDE-motivated Approaches

- **E. Haber, LR, E. Holtham**, *Learning across scales - Multiscale CNNs*, AAAI, 2018.
- **LR, E. Haber**, *DNNs motivated by PDEs*, arXiv, 2018.

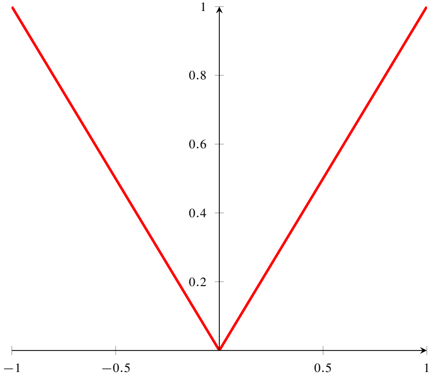
Numerical Methods for Deep Learning

An (almost perfectly) true statement

$$\text{backpropagation} + \text{GPU} + \begin{cases} \text{TensorFlow} \\ \text{Caffe} \\ \text{Torch} \\ \vdots \end{cases} \Rightarrow \text{success}$$

So, why study numeric methods for deep learning?

A Simple Example



Predict the output of:

```
x = param(0.0)
f = abs(x)
Tracker.back!(f)
Tracker.grad(x)
```

Case 1: Reversibility

Reversibility: Continuous vs. Discrete

Goal: If $\mathbf{Y} = \mathrm{NN}(\mathbf{X}, \theta)$, want $\mathbf{X} = \mathrm{NN}^{-1}(\mathbf{Y}, \theta)$!

Idea 1: ResNet

$$\partial_t \mathbf{Y} = \tanh(\mathbf{K}(t)\mathbf{Y} + \mathbf{b}(t))$$

- discretize: RK4, 16 time steps
- 4 channels, pad inputs with 0
- inverse: integrate backward in time

Idea 2: Hamiltonian NN

$$\partial_t \begin{pmatrix} \mathbf{Y} \\ \mathbf{Z} \end{pmatrix} = \begin{pmatrix} \tanh(\mathbf{K}(t)\mathbf{Z} + \mathbf{b}(t)) \\ -\tanh(\mathbf{K}(t)^\top \mathbf{Y} + \mathbf{b}(t)) \end{pmatrix}$$

- discretize: Verlet, 32 time steps
- no padding, trivial inverse

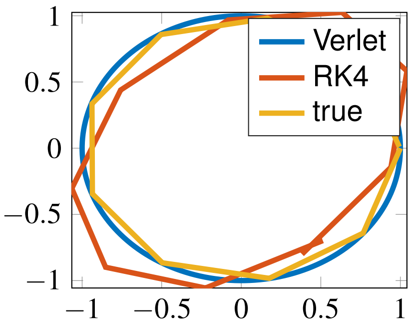
Case 2: Black-box

Example: High-order vs. Symplectic?

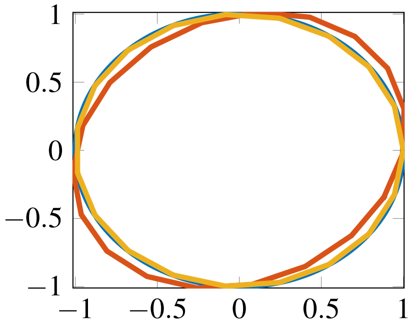
Consider linear harmonic oscillator

$$\partial_t \begin{pmatrix} q \\ p \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} q \\ p \end{pmatrix} \quad \text{associated with} \quad H(p,q) = p^2 + q^2$$

h large



h small



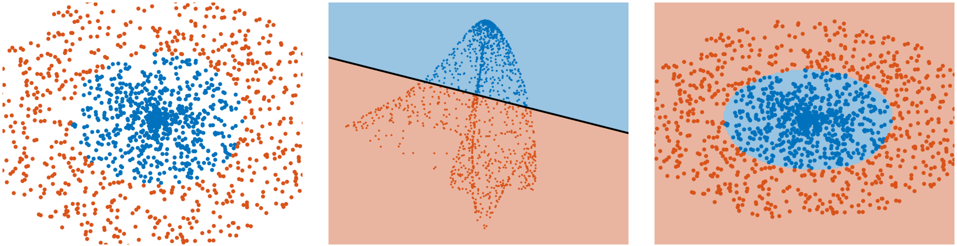
Use the right (in this case lower-order) integrators $\rightsquigarrow$ more bang for the buck!

U Ascher
Numerical methods for evolutionary differential equations.
SIAM, 2008

S Greydanus, M Dzamba, J Yosinski
Hamiltonian Neural Networks.
arXiv:1906.01563

Case 3: Discrete vs. Continuous

Optimal Control Framework for Deep Learning



training data, $\mathbf{Y}_0, \mathbf{C}$ prop. features, $\mathbf{Y}(T), \mathbf{C}$ classification result

Supervised Deep Learning Problem

Given training data, $\mathbf{Y}_0$, and labels, $\mathbf{C}$, find **network parameters** θ and **classification weights** $\mathbf{W}$, μ such that the DNN predicts the data-label relationship (and generalizes to new data), i.e., solve

$$\text{minimize}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \quad \text{loss}[g(\mathbf{W}+\boldsymbol{\mu}), \mathbf{C}] + \text{regularizer}[\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}]$$

Optimal Control Background: Diff→Disc vs. Disc→Diff

$$\begin{aligned} \text{minimize}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \quad & \text{loss}[g(\mathbf{W}\mathbf{Y}(T)+\boldsymbol{\mu}),\mathbf{C}] + \text{regularizer}[\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}] \\ \text{subject to} \quad & \partial_t \mathbf{Y}(t) = f\left(\mathbf{Y}(t),\boldsymbol{\theta}(t)\right), \ \mathbf{Y}(0)=\mathbf{Y}_0. \end{aligned}$$

- **First-Differentiate-then-Discretize (Diff→Disc)**
  - Keep $\boldsymbol{\theta}, \mathbf{b}, \mathbf{Y}$ 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 $\boldsymbol{\theta}, \mathbf{b}, \mathbf{Y}$ 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

MD Gunzburger
Perspectives in flow control and optimization.
SIAM, 2013.

TQ Chen et al.,
Neural Ordinary Differential Equations.
NeurIPS, 2018.

A Gholami, K Keutzer, G Biros
ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs.
arXiv:1902.10298

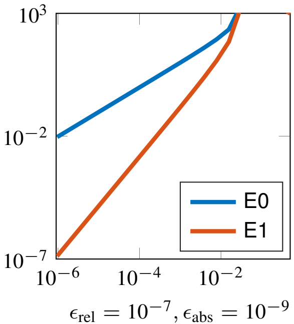
Example: Gradient Test Disc→Diff

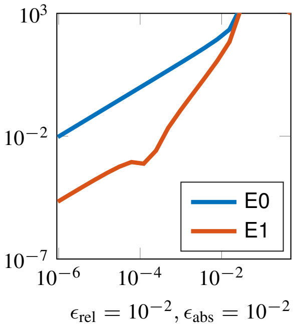
Goal: Find weights of neural network $F(\mathbf{u}, \theta)$ such that

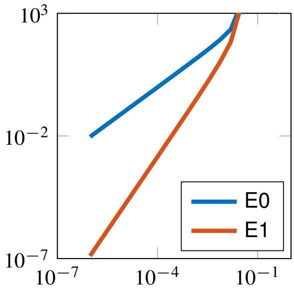
$$\partial_t \mathbf{u} = F(\mathbf{u}, \theta), \quad \mathbf{u}(0) = \mathbf{u}_0$$

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

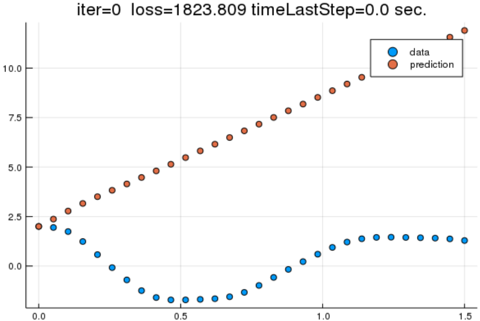
Question: How does accuracy of ODE solvers impact the quality of gradient?

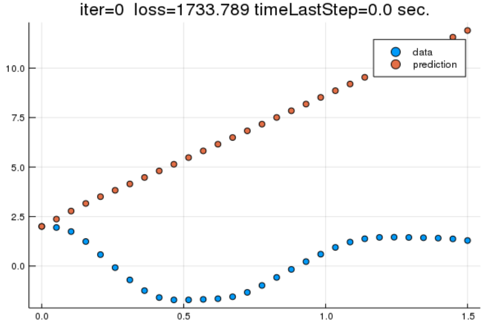






C Rackauckas, M Innes, Y Ma, J Bettencourt, L White, V Dixit
DiffEqFlux.jl - A Julia Library for Neural Differential Equations.
arXiv:1902.02376

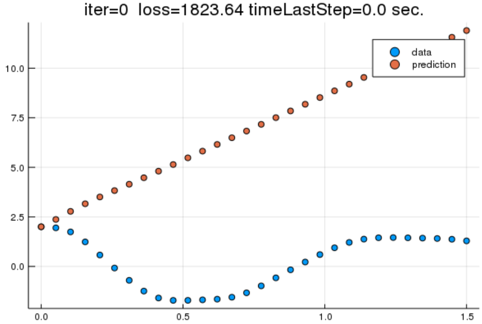


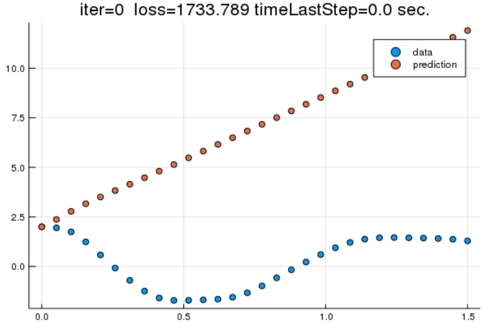


Neural ODE, $\epsilon_{\text{rel}} = 10^{-7}, \epsilon_{\text{abs}} = 10^{-9}$

Disc$\rightarrow$Diff, RK4, 30 steps

Training: ADAM with default setting, same initialization

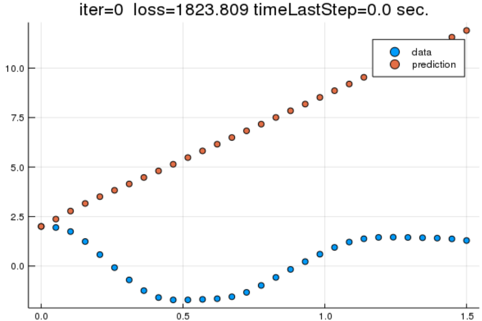


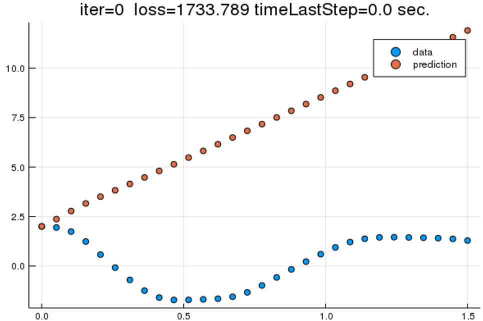


Neural ODE, $\epsilon_{\text{rel}} = 10^{-7}$, $\epsilon_{\text{abs}} = 10^{-9}$

Disc$\rightarrow$Diff, RK4, 30 steps

Training: ADAM with default setting, same initialization





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

Disc→Diff, RK4, 30 steps

Training: ADAM with default setting, same initialization

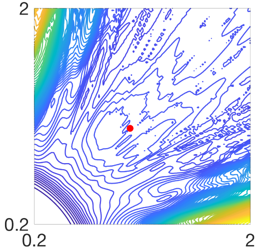
Impact of Network Architecture on Optimization - 1

$$\min_{\theta} \frac{1}{2}\|\mathbf{Y}_N(\theta) - \mathbf{C}\|_F^2 \qquad \mathbf{Y}_{j+1}(\theta) = \mathbf{Y}_j(\theta) + \frac{10}{N}\tanh\left(\mathbf{K}\mathbf{Y}_j(\theta)\right)$$

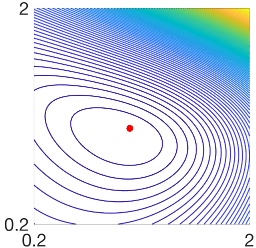
where $\mathbf{C} = \mathbf{Y}_{200}(1,1)$, $\mathbf{Y}_0 \sim \mathcal{N}(0,1)$, and

$$\mathbf{K}(\theta) = \begin{pmatrix} -\theta_1 - \theta_2 & \theta_1 & \theta_2 \\ \theta_2 & -\theta_1 - \theta_2 & \theta_1 \\ \theta_1 & \theta_2 & -\theta_1 - \theta_2 \end{pmatrix}$$

loss, $N = 5$

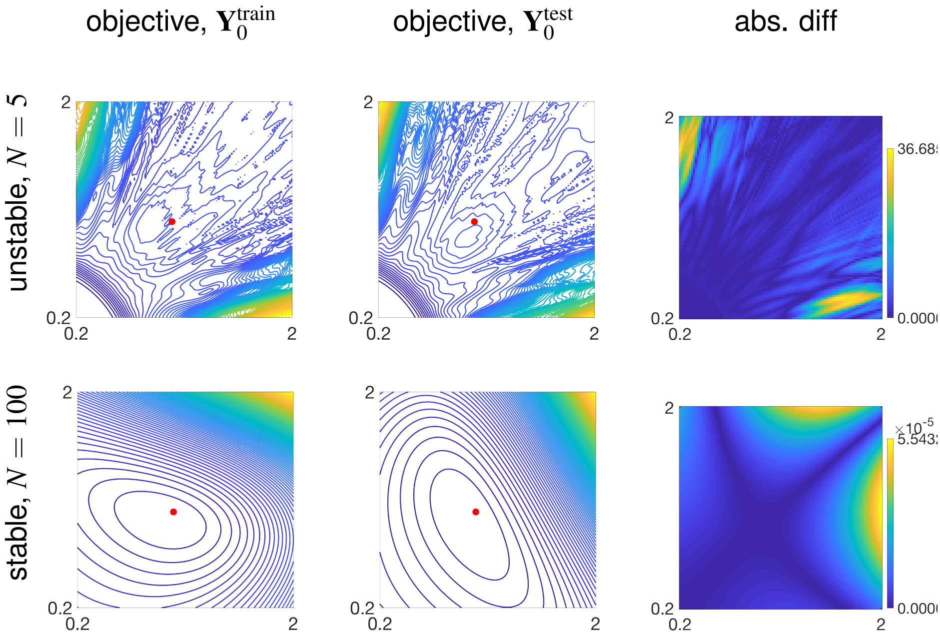


loss, $N = 100$



Next: Compare examples for different inputs $\sim$ generalization

Impact of Network Architecture on Optimization - 2



Case 4: Conditioning

Conditioning of the Learning Problem

Consider the regression problem with a single neural network layer

$$\min_{\mathbf{W},\mathbf{K}} \frac{1}{2s}\|\mathbf{R}(\mathbf{W},\mathbf{K})\|^2, \quad \text{where} \quad \mathbf{R}(\mathbf{W},\mathbf{K}) = \mathbf{W}\sigma(\mathbf{K}\mathbf{Y}) - \mathbf{C}$$

- $\mathbf{Y} \in \mathbb{R}^{d\times s}$ - input features
- $\mathbf{C} \in \mathbb{R}^{n\times s}$ - output features
- $\mathbf{K} \in \mathbb{R}^{m\times d}, \mathbf{W} \in \mathbb{R}^{n\times m}$ - weights for fully-connected transformation
- $\sigma : \mathbb{R} \to \mathbb{R}$ - activation function (applied to each element)

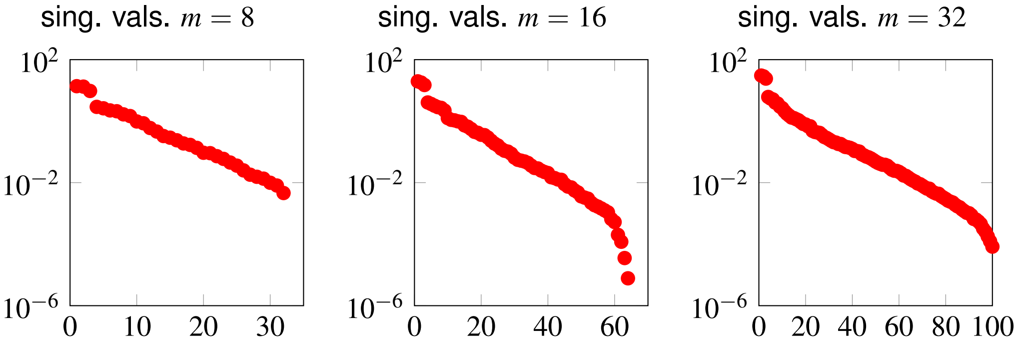
The problem above is a non-linear least squares problem (NNLS). Common to look at the Jacobian of $\mathbf{r}$, i.e., $\mathbf{J} = [\mathbf{J}_\mathbf{W}\ \mathbf{J}_\mathbf{K}]$ where

$$\mathbf{J}_\mathbf{W} = \sigma(\mathbf{K}\mathbf{Y})^\top \otimes \mathbf{I}, \quad \text{and} \quad \mathbf{J}_\mathbf{K} = (\mathbf{I}\otimes\mathbf{W})\,\mathrm{diag}(\sigma'(\mathbf{K}\mathbf{Y}))\,(\mathbf{Y}^\top \otimes \mathbf{I})$$

(here, we vectorized $\mathbf{R}$, $\mathbf{I}$ is identity, and $\otimes$ is the Kronecker product)

Q: What are the properties of $\mathbf{J}$?

Example: Condition Numbers



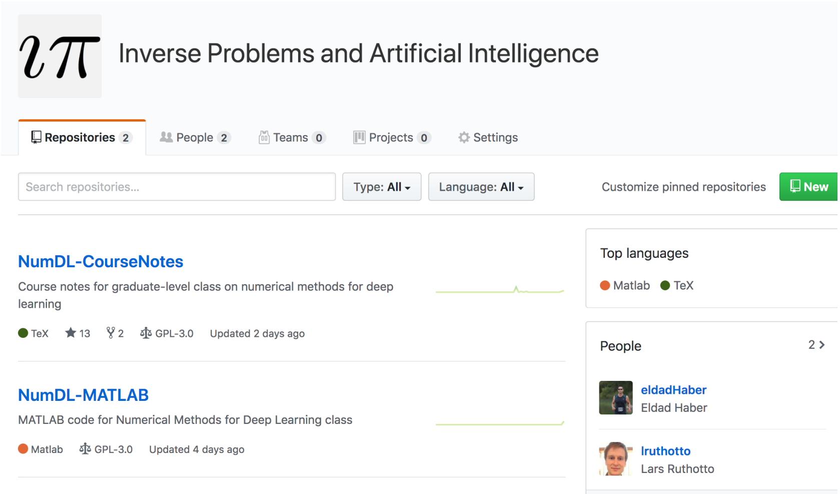
$$\mathbf{R}(\mathbf{K}, \mathbf{W}) = \mathbf{W}\sigma(\mathbf{KY}) - \mathbf{C}$$

- $d = 3/n = 1$ input/output features
- $s = 100$ examples $\sim \mathcal{U}([-1, 1]^d)$
- $m = \{8, 16, 32\}$ width of network
- $\sigma = \tanh$
- $\mathbf{K}, \mathbf{W} \sim \mathcal{N}(0, 1)$

Discussion:

- problem is ill-posed $\rightsquigarrow$ regularize!
- $\mathrm{cond}(\mathbf{J})$ large $\rightsquigarrow$ smart LinAlg
- how about single/half precision?
- NNLS solvers will not be effective
- need better initialization / method

Conclusion



- course launched Spring 18 at Emory and UBC
- slides + simple MATLAB codes available (pyTorch to come)
- next offerings: Fall '19 at UBC and Spring '20 at Emory

check it out: `https://github.com/IPAIopen`

Numerical Methods for Deep Learning

An (almost perfectly) true statement

$$\text{backpropagation} + \text{GPU} + \left\{ \begin{array}{c} \text{TensorFlow} \\ \text{Caffe} \\ \text{Torch} \\ \vdots \end{array} \right. \Rightarrow \text{success}$$

So, why study numeric methods for deep learning?

Transfer Learning

- DL is similar to path planning, optimal control, differential equations . . .

Do More With Less

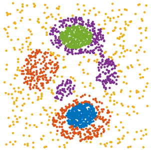
- Better modeling and algorithms $\rightsquigarrow$ process more data, use less resources
- How about 3D images and videos?

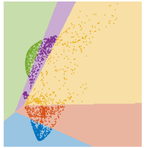
Power Of Abstraction

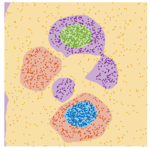
- Use continuous interpretation to design/relate architectures

Σ: Numerical Analysis of Deep Neural Networks

- **Case 1: Invertibility**
  - often but not always important
  - it does not come for free
- **Case 2: Time integrators**
  - conserving physical quantities can simplify numerics
- **Case 3: Discretize-then-Optimize**
  - Neural ODE: need high accuracy to obtain good gradients
  - Disc→Diff: can be more efficient/predictable
- **Case 4: Ill-conditioning**
  - even simple learning problems can be ill-posed
  - need more analysis, especially in low-precisions







There are a lot of challenges in ML for computational and applied mathematicians