Nonlinear Models

Numerical Methods for Deep Learning

Lars Ruthotto

Departments of Mathematics and Computer Science, Emory University

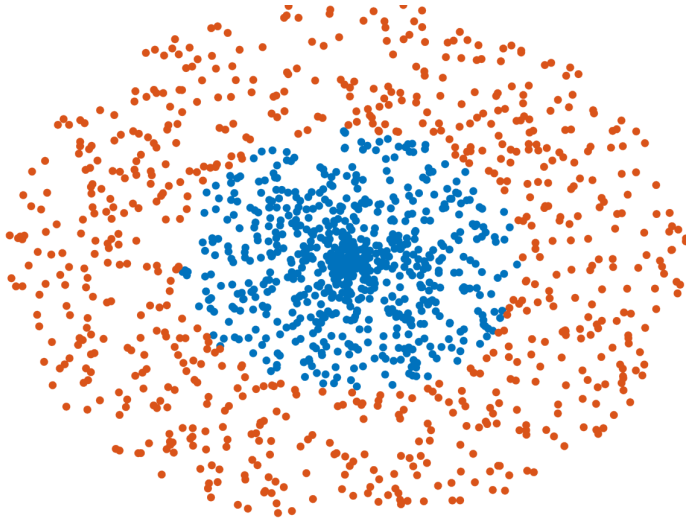
Course Overview

- Part 1: Linear Models
  1. Introduction and Applications
  2. Linear Models: Least-Squares and Logistic Regression
- Part 2: Neural Networks
  1. Introduction to Nonlinear Models
  2. Parametric Models, Convolutions
  3. Single Layer Neural Networks
  4. Training Algorithms for Single Layer Neural Networks
  5. Neural Networks and Residual Neural Networks (ResNets)
- Part 3: Neural Networks as Differential Equations
  1. ResNets as ODEs
  2. Residual CNNs and their relation to PDEs

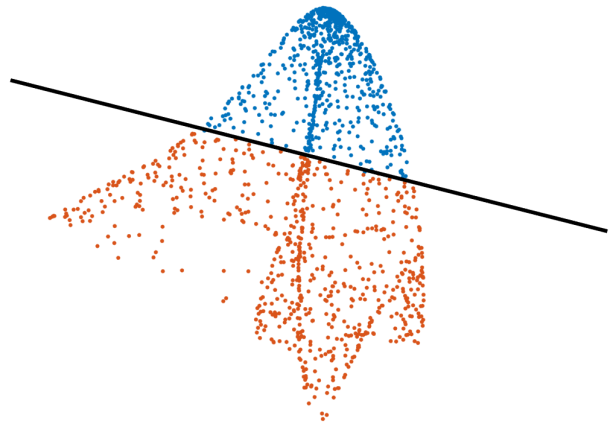
Introducion

Motivation: Nonlinear Models

In general, impossible to find a linear separator between classes



input features



transformed features

Goal/Trick

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

Example: Linear Fitting

Assume $\mathbf{C} \in \mathbb{R}^{n_c \times n}$, $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ and $n \gg n_f$. Goal: Find $\mathbf{W} \in \mathbb{R}^{n_c \times n_f}$ such that

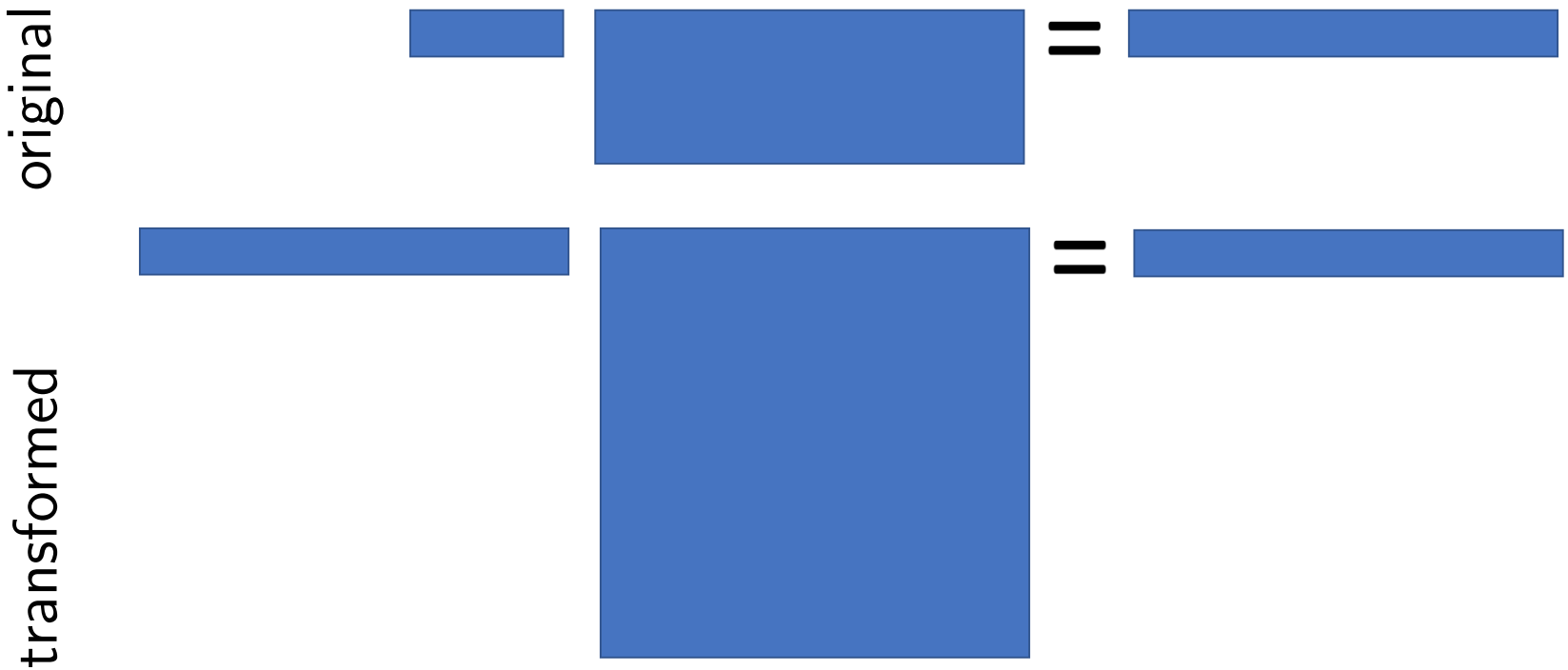
$$\mathbf{C} = \mathbf{WY}$$

If $\operatorname{rank}(\mathbf{Y}) < n$, there may be no solution.

Two options:

1. Regression: Solve $\min_{\mathbf{W}} \|\mathbf{WY} - \mathbf{C}\|_F^2 \rightsquigarrow$ always has solutions, but residual might be large
2. Nonlinear Model: Replace $\mathbf{Y}$ by $\sigma(\mathbf{KY})$ in regression, where σ is element-wise function (aka activation) and $\mathbf{K} \in \mathbb{R}^{m \times n_f}$ where $m \gg n_f$

Illustrating Nonlinear Models



Remarks

- instead of $\mathbf{WY} = \mathbf{C}$ solve $\hat{\mathbf{W}}\sigma(\mathbf{KY}) = \mathbf{C}$
- solve bigger problem $\rightsquigarrow$ memory, computation, ...
- what happens to $\mathrm{rank}(\sigma(\mathbf{KY}))$ when $\sigma(x) = x$?

Conjecture: Universal Approximation Properties

Given the data $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ and $\mathbf{C} \in \mathbb{R}^{n_c \times n}$ with $n \gg n_f$, there is nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$, a matrix $\mathbf{K} \in \mathbb{R}^{m \times n_f}$, and a bias $\mathbf{b} \in \mathbb{R}^m$ such that

$$\operatorname{rank}(\sigma(\mathbf{KY} + \mathbf{b})) = n.$$

Therefore, possible [2, 7] to find $\mathbf{W} \in \mathbb{R}^{n_c \times m}$

$$\mathbf{W}\sigma(\mathbf{KY} + \mathbf{b}) = \mathbf{C}.$$

Choosing Nonlinear Model

$$\mathbf{W}\sigma(\mathbf{K}\mathbf{Y} + \mathbf{b}) = \mathbf{C}$$

- how to choose σ?
  - early days: motivated by neurons
  - popular choice: $\sigma(x) = \tanh(x)$ (smooth, bounded, ...)
  - nowadays: $\sigma(x) = \max(x, 0)$ (aka ReLU, rectified linear unit, non-differentiable, not bounded, simple)
- how to choose $\mathbf{K}$ and $\mathbf{b}$?
  - pick randomly $\rightsquigarrow$ branded as *extreme learning machines* [8]
  - train (optimize) $\rightsquigarrow$ done for most neural network
  - *deep learning* when neural network has many layers

First Experiment: Random Transformation

Select activation function and choose $\mathbf{K}$ and $\mathbf{b}$ randomly and solve the least-squares/classification problem

The Pros:

- universal approximation theorem: can interpolate any function
- very(!) easy to program
- can serve as a benchmark to more sophisticated methods

Some concerns:

- may require very large $\mathbf{K}$ (scale with n, number of examples)
- may not generalize well
- large dense linear algebra

`EELM_Peaks.m`

Parametric Models

Motivation

Recall single layer

$$\mathbf{Z} = \sigma(\mathbf{K}\mathbf{Y} + \mathbf{b}),$$

where $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$, $\mathbf{K} \in \mathbb{R}^{m \times n_f}$, $\mathbf{b} \in \mathbb{R}^m$, and σ element-wise activation.

We saw that $m \gg n_f$ needed to fit training data.
Conservative example: Consider MNIST ($n_f = 28^2$) and use $m = n_f \rightsquigarrow 614,656$ unknowns for a single layer. Famous quote:

> *With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.*

Possible remedies:

- **Regularization:** penalize $\mathbf{K}$
- **Parametric model:** $\mathbf{K}(\boldsymbol{\theta})$ where $\boldsymbol{\theta} \in \mathbb{R}^p$ with $p \ll m \cdot n_f$.

Some Simple Parametric Models

- Diagonal scaling:

$$\mathbf{K}(\boldsymbol{\theta}) = \mathrm{diag}(\boldsymbol{\theta}) \in \mathbb{R}^{n_f \times n_f}$$

  Advantage: preserves size and structure of data.

- Antisymmetric kernel

$$\mathbf{K}(\boldsymbol{\theta}) = \begin{pmatrix} 0 & \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 \\ -\boldsymbol{\theta}_1 & 0 & \boldsymbol{\theta}_3 \\ -\boldsymbol{\theta}_2 & -\boldsymbol{\theta}_3 & 0 \end{pmatrix}$$

  Advantage?: $\mathrm{real}(\lambda_i(\mathbf{K}(\boldsymbol{\theta}))) = 0$.

- M-matrix

$$\mathbf{K}(\boldsymbol{\theta}) = \begin{pmatrix} \boldsymbol{\theta}_1 + \boldsymbol{\theta}_2 & -\boldsymbol{\theta}_1 & -\boldsymbol{\theta}_2 \\ -\boldsymbol{\theta}_3 & \boldsymbol{\theta}_3 + \boldsymbol{\theta}_4 & -\boldsymbol{\theta}_4 \\ -\boldsymbol{\theta}_5 & -\boldsymbol{\theta}_6 & \boldsymbol{\theta}_5 + \boldsymbol{\theta}_6 \end{pmatrix} \qquad \boldsymbol{\theta} \geq 0$$

  Advantage: like differential operator

Differentiating Parametric Models

Need derivatives of model to optimize $\boldsymbol{\theta}$ in

$$E(\mathbf{W}\sigma(\mathbf{K}(\boldsymbol{\theta})\mathbf{Y} + \mathbf{b}), \mathbf{C})$$

(we can re-use previous derivatives and use chain rule)

Note that all previous models are linear in the following sense

$$\mathbf{K}(\boldsymbol{\theta}) = \mathrm{mat}(\mathbf{Q}\,\boldsymbol{\theta})$$

Therefore, matrix-vector products with the Jacobian simply are

$$\mathbf{J}_{\boldsymbol{\theta}}(\mathbf{K}(\boldsymbol{\theta}))\mathbf{v} = \mathrm{mat}(\mathbf{Q}\ \mathbf{v}) \quad \text{and} \quad \mathbf{J}_{\boldsymbol{\theta}}(\mathbf{K}(\boldsymbol{\theta}))^\top \mathbf{w} = \mathbf{Q}^\top \mathbf{w}$$

where $\mathbf{v} \in \mathbb{R}^p$ and $\mathbf{w} \in \mathbb{R}^m$.

Example: Derivative of M-matrix

$$\mathbf{K}(\boldsymbol{\theta}) = \begin{pmatrix} \boldsymbol{\theta}_1 + \boldsymbol{\theta}_2 & -\boldsymbol{\theta}_1 & -\boldsymbol{\theta}_2 \\ -\boldsymbol{\theta}_3 & \boldsymbol{\theta}_3 + \boldsymbol{\theta}_4 & -\boldsymbol{\theta}_4 \\ -\boldsymbol{\theta}_5 & -\boldsymbol{\theta}_6 & \boldsymbol{\theta}_5 + \boldsymbol{\theta}_6 \end{pmatrix} \quad \boldsymbol{\theta} \geq 0$$

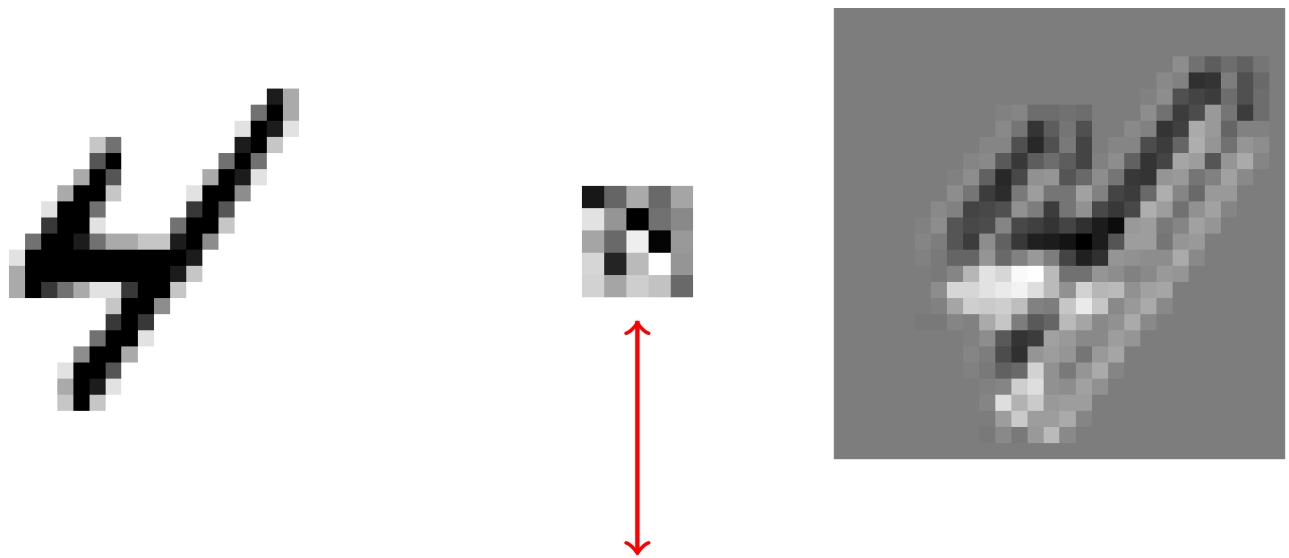
verify that this can be written as $K(\boldsymbol{\theta}) = \mathrm{mat}(\mathbf{Q}\,\boldsymbol{\theta})$ where

$$\mathbf{Q} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \in \mathbb{R}^{9\times 6}$$

Note: not efficient to construct $\mathbf{Q}$ when p large but helpful when computing derivatives

Convolutional Neural Networks [9]

$\mathbf{y} \in \mathbb{R}^{28\times 28}$ input features $\quad \mathbf{z} \in \mathbb{R}^{28\times 28}$ output features



$\boldsymbol{\theta} \in \mathbb{R}^{5\times 5}$ convolution kernel

- useful for speech, images, videos, ...
- efficient parameterization, efficient codes (GPUs, ...)
- later: CNNs as parametric model and PDEs, simple code
- see `E13Conv2D.m`

Convolutions in 1D

Let $y, z, \theta : \mathbb{R} \to \mathbb{R}$, $z : \mathbb{R} \to \mathbb{R}$ be continuous functions then

$$z(x) = (\theta * y)(x) = \int_{-\infty}^{\infty} \theta(x - t) y(t) dt.$$

Assume $\theta(x) \neq 0$ only in interval $[-a, a]$ (compact support).

A few properties

- $\theta * y = \mathcal{F}^{-1}((\mathcal{F}\theta)(\mathcal{F}y))$, $\mathcal{F}$ is Fourier transform
- $\theta * y = y * \theta$

Discrete Convolutions in 1D

Let $\boldsymbol{\theta} \in \mathbb{R}^{2k+1}$ be stencil, $\mathbf{y} \in \mathbb{R}^{n_f}$ grid function

$$\mathbf{z}_i = (\boldsymbol{\theta} * \mathbf{y})_i = \sum_{j=-k}^{k} \boldsymbol{\theta}_j \mathbf{y}_{i-1}.$$

Example: Discretize $\boldsymbol{\theta} \in \mathbb{R}^3$ (non-zeros only), $\mathbf{y}, \mathbf{z} \in \mathbb{R}^4$ on regular grid

$$\begin{aligned}
\mathbf{z}_1 &= \boldsymbol{\theta}_3 \mathbf{w}_1 + \boldsymbol{\theta}_2 \mathbf{x}_1 + \boldsymbol{\theta}_1 \mathbf{x}_2 \\
\mathbf{z}_2 &= \boldsymbol{\theta}_3 \mathbf{x}_1 + \boldsymbol{\theta}_2 \mathbf{x}_2 + \boldsymbol{\theta}_1 \mathbf{x}_3 \\
\mathbf{z}_3 &= \boldsymbol{\theta}_3 \mathbf{x}_2 + \boldsymbol{\theta}_2 \mathbf{x}_3 + \boldsymbol{\theta}_1 \mathbf{x}_4 \\
\mathbf{z}_4 &= \boldsymbol{\theta}_3 \mathbf{x}_3 + \boldsymbol{\theta}_2 \mathbf{x}_4 + \boldsymbol{\theta}_1 \mathbf{w}_2
\end{aligned}$$

where $\mathbf{w}_1, \mathbf{w}_2$ are used to implement different boundary conditions (right choice? depends . . .).

Structured Matrices - 1

$$\begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \\ \mathbf{z}_4 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 & & \\ & \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 & \\ & & \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 \\ & & & \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 \end{pmatrix} \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \\ \mathbf{w}_2 \end{pmatrix}$$

Different boundary conditions lead to different structures

- Zero boundary conditions: $\mathbf{w}_1 = \mathbf{w}_2 = 0$

$$\begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \\ \mathbf{z}_4 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 & & \\ \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 & \\ & \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 \\ & & \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \end{pmatrix}$$

This is a *Toeplitz matrix* (constant along diagonals).

Structured Matrices - 2

- Periodic boundary conditions: $\mathbf{w}_1 = \mathbf{x}_4$ and $\mathbf{w}_2 = \mathbf{x}_1$

$$\begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \\ \mathbf{z}_4 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 & & \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 & \\ & \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_1 & & \boldsymbol{\theta}_3 & \boldsymbol{\theta}_2 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \end{pmatrix}$$

  this is a *circulant matrix* (each row/column is periodic shift of previous row/column)

An attractive property of a circulant matrix is that we can efficiently compute its eigendecomposition

$$\mathbf{K}(\boldsymbol{\theta}) = \mathbf{F}^* \mathrm{diag}(\boldsymbol{\lambda}) \mathbf{F}$$

where $\mathbf{F}$ is the discrete Fourier transform and the eigenvalues, $\boldsymbol{\lambda} \in \mathbb{C}^4$, can be computed using first column

$$\boldsymbol{\lambda} = \mathbf{F}(\mathbf{K}(\boldsymbol{\theta})\mathbf{u}_1) \quad \text{where} \quad \mathbf{u}_1 = (1, 0, 0, 0)^\top.$$

Coding: 1D Convolution using FFTs

Let $\boldsymbol{\theta} \in \mathbb{R}^3$ be some stencil and $n_f = m = 16$

1. build a sparse matrix $\mathbf{K}$ for computing the convolution with periodic boundary conditions. Hint: `spdiags`
2. compute the eigenvalues of $\mathbf{K}$ using `eig(full(K))` and using `fft` and first column of $\mathbf{K}$. Compare!
3. verify that `norm(K*y - real(ifft(lam.*fft(y))))` is small.
4. repeat previous item for transpose.
5. write code that computes eigenvalues for arbitrary stencil size without building $\mathbf{K}$. Hint: `circshift`

Derivatives of 1D Convolution - 1

Recall that we need a way to compute

$$\mathbf{J}_{\boldsymbol{\theta}}(\mathbf{K}(\boldsymbol{\theta})\mathbf{Y})\mathbf{v} \quad \text{and} \quad \mathbf{J}_{\boldsymbol{\theta}}(\mathbf{K}(\boldsymbol{\theta})\mathbf{Y})^{\top}\mathbf{w}, \quad (\mathbf{J}_{\boldsymbol{\theta}} \in \mathbb{R}^{m \times p})$$

(note that we put $\mathbf{Y}$ inside the bracket to avoid tensors)

Assume single example, $\mathbf{y}$. Since we have periodic boundary conditions

$$\begin{aligned}\mathbf{K}(\boldsymbol{\theta})\mathbf{y} &= \operatorname{real}(\mathbf{F}^{*}(\boldsymbol{\lambda}(\boldsymbol{\theta}) \odot \mathbf{F}\mathbf{y})) \\ &= \operatorname{real}(\mathbf{F}^{*} \operatorname{diag}(\mathbf{F}\mathbf{y})\, \boldsymbol{\lambda}(\boldsymbol{\theta})), \quad \boldsymbol{\lambda}(\boldsymbol{\theta}) = \mathbf{F}(\mathbf{K}(\boldsymbol{\theta})\mathbf{u}_1).\end{aligned}$$

Need to differentiate eigenvalues w.r.t. $\boldsymbol{\theta}$. Note linearity

$$\mathbf{K}(\boldsymbol{\theta})\mathbf{u}_1 = \mathbf{Q}\boldsymbol{\theta}, \quad \mathbf{Q} = ?$$

Derivatives of 1D Convolution - 2

Assume we have

$$\mathbf{K}(\boldsymbol{\theta})\mathbf{y} = \mathrm{real}(\mathbf{F}^* \ \mathrm{diag}(\mathbf{F}\mathbf{y}) \ \mathbf{F}\mathbf{Q}\boldsymbol{\theta}))$$

Then mat-vecs with Jacobian are easy to compute

$$\mathbf{J}_{\boldsymbol{\theta}}(\mathbf{K}(\boldsymbol{\theta})\mathbf{y})\mathbf{v} = \mathrm{real}(\mathbf{F}^*(\mathrm{diag}(\mathbf{F}\mathbf{y})\mathbf{F}\mathbf{Q}\mathbf{v}))$$

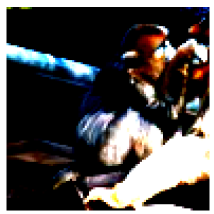
and (note that $\mathbf{F}^\top = \mathbf{F}$ and $(\mathbf{F}^*)^\top = \mathbf{F}^*$)

$$\mathbf{J}_{\boldsymbol{\theta}}(\mathbf{K}(\boldsymbol{\theta})\mathbf{y})^\top\mathbf{w} = \mathrm{real}(\mathbf{Q}^\top\mathbf{F}\mathrm{diag}(\mathbf{F}\mathbf{y})\mathbf{F}^*\mathbf{w})$$

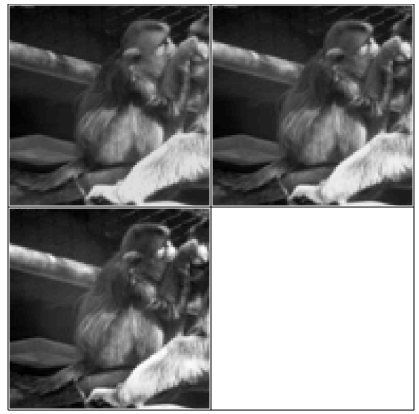
Code this and check Jacobian and its transpose using `conv1D.m`!

Extension: Width of CNNs

RGB image



input channels



output channels

Width of CNN can be controlled by number of input and output channels of each layer. Let $\mathbf{y} = (\mathbf{y}_R, \mathbf{y}_G, \mathbf{y}_B)$, then we might compute

$$\begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \\ \mathbf{z}_4 \end{pmatrix} = \begin{pmatrix} \mathbf{K}^{11}(\boldsymbol{\theta}^{11}) & \mathbf{K}^{12}(\boldsymbol{\theta}^{12}) & \mathbf{K}^{11}(\boldsymbol{\theta}^{13}) \\ \mathbf{K}^{21}(\boldsymbol{\theta}^{21}) & \mathbf{K}^{22}(\boldsymbol{\theta}^{22}) & \mathbf{K}^{23}(\boldsymbol{\theta}^{23}) \\ \mathbf{K}^{31}(\boldsymbol{\theta}^{31}) & \mathbf{K}^{32}(\boldsymbol{\theta}^{32}) & \mathbf{K}^{33}(\boldsymbol{\theta}^{33}) \\ \mathbf{K}^{41}(\boldsymbol{\theta}^{41}) & \mathbf{K}^{42}(\boldsymbol{\theta}^{42}) & \mathbf{K}^{43}(\boldsymbol{\theta}^{43}) \end{pmatrix} \begin{pmatrix} \mathbf{y}_R \\ \mathbf{y}_G \\ \mathbf{y}_B \end{pmatrix},$$

where $\mathbf{K}^{ij}$ is a 2D convolution operator with stencil $\boldsymbol{\theta}^{ij}$

Outlook: Possible Extensions

For now, we just introduced the very basic convolution layer. CNNs used in practice also use the following components

- *pooling:* reduce image resolution (e.g. average over patches)
- *stride:* Example: stride of two reduces image resolution by computing $\mathbf{z}$ only at every other pixel.

Build your own parametric model

- $M-$matrix for convolution
- cheaper convolution models: separable kernels, doubly symmetric kernels
- Wavelet, ...
- other sparsity patterns

Single Layer Neural Networks

Learning the Weights

Assume that the number of examples, n, is very large.
Using random weights, $\mathbf{K}$ might need to be very large to fit training data.
Solution may not generalize well to test data.

Idea: Learn $\mathbf{K}$ and b from the data (in addition to $\mathbf{W}$)

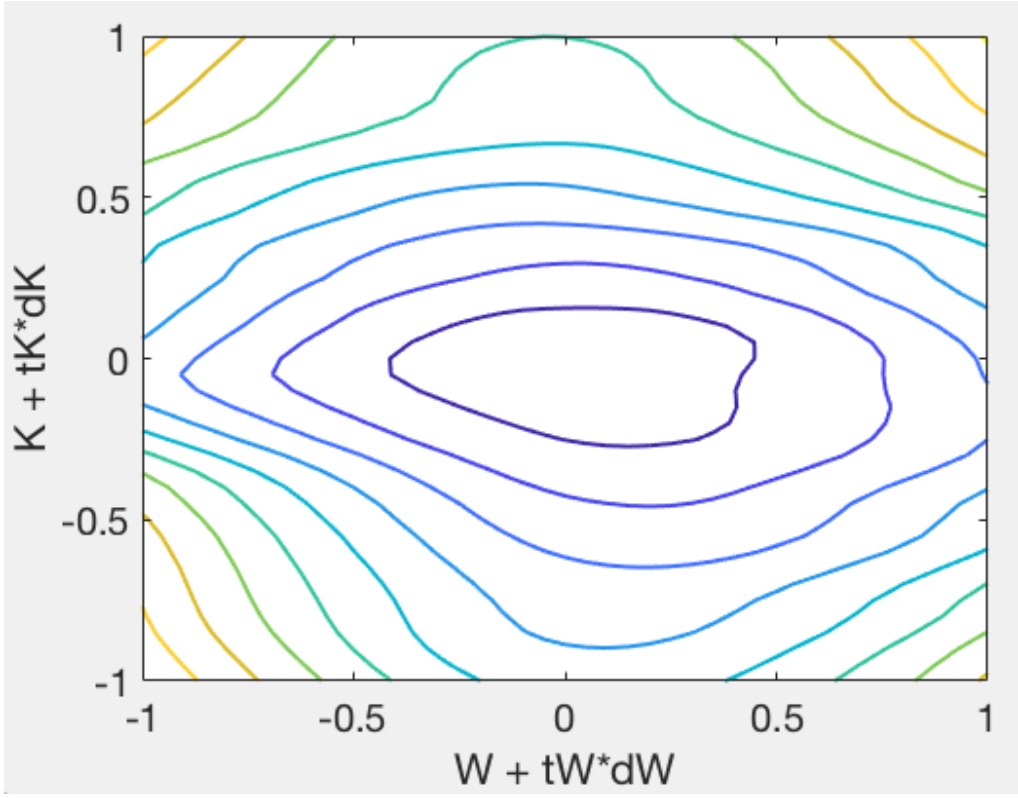
$$\min_{\mathbf{K},\mathbf{W},b} E(\mathbf{W}\sigma(\mathbf{K}\mathbf{Y} + \mathbf{b}), \mathbf{C}^{\text{obs}}) + \lambda R(\mathbf{W}, \mathbf{K}, \mathbf{b})$$

About this optimization problem:

- more unknowns $\mathbf{K} \in \mathbb{R}^{m \times n_f}$, $\mathbf{W} \in \mathbb{R}^{n_c \times m}$, $\mathbf{b} \in \mathbb{R}^m$
- non-convex problem $\rightsquigarrow$ local minima, careful initialization
- need to compute derivatives w.r.t. $\mathbf{K}, \mathbf{b}$

Non-Convexity

The optimization problem is non-convex. Simple illustration of cross-entropy along two random directions $d\mathbf{K}$ and $d\mathbf{W}$



(see `ESingleLayer_PlotObjective`)

Expect worse when number of layers grows!

Optimization for Single Layer Neural Networks

Gauss-Newton Method

Goal: Use curvature information for fast convergence

$$\nabla_{\mathbf{K}} E(\mathbf{K}, \mathbf{b}, \mathbf{W}) = (\mathbf{J}_{\mathbf{K}}\mathbf{Z})^\top \nabla_{\mathbf{Z}} E(\mathbf{W}\sigma(\mathbf{K}\mathbf{Y} + \mathbf{b}), \mathbf{C}),$$

where $\mathbf{J}_{\mathbf{K}}\mathbf{Z} = \nabla_{\mathbf{K}}\sigma(\mathbf{K}\mathbf{Y} + \mathbf{b})^\top$. This means that Hessian is

$$\nabla^2_{\mathbf{K}} E(\mathbf{K}) = (\mathbf{J}_{\mathbf{K}}\mathbf{Z})^\top \nabla^2_{\mathbf{Z}} E(\mathbf{C}, \mathbf{Z}, \mathbf{W}) \mathbf{J}_{\mathbf{K}}\mathbf{Z} + \sum_{i=1}^{n}\sum_{j=1}^{m} \nabla^2_{\mathbf{K}}\sigma(\mathbf{K}\mathbf{Y} + \mathbf{b})_{ij} \nabla_{\mathbf{Z}} E(\mathbf{C}, \mathbf{Z}, \mathbf{W})_{ij}$$

First term is spsd and we can compute it.
We neglect second term since

- can be indefinite and difficult to compute
- small if transformation is roughly linear or close to solution (easy to see for least-squares)

do the same for $\mathbf{b}$ and use full Hessian for $\mathbf{W} \rightsquigarrow$ ignore coupling!

Variable Projection - 1

Idea: Treat learning problem as coupled optimization problem with blocks $\boldsymbol{\theta} = (\mathbf{K}, \mathbf{b})$ and $\mathbf{W}$.
Simple illustration for coupled least-squares problem [4, 3, 11]

$$\min_{\boldsymbol{\theta},\mathbf{w}} \phi(\boldsymbol{\theta}, \mathbf{w}) = \frac{1}{2}\|\mathbf{A}(\boldsymbol{\theta})\mathbf{w} - \mathbf{c}\|^2 + \frac{\lambda}{2}\|\mathbf{L}\mathbf{w}\|^2 + \frac{\beta}{2}\|\mathbf{M}\boldsymbol{\theta}\|^2$$

Note that for given $\boldsymbol{\theta}$ the problem becomes a standard least-squares problem. Define:

$$\mathbf{w}(\boldsymbol{\theta}) = \left(\mathbf{A}(\boldsymbol{\theta})^\top \mathbf{A}(\boldsymbol{\theta}) + \lambda \mathbf{L}^\top \mathbf{L}\right)^{-1} \mathbf{A}(\boldsymbol{\theta})^\top \mathbf{c}$$

This gives optimization problem in $\boldsymbol{\theta}$ only (aka *reduced/projected problem*)

$$\min_{\boldsymbol{\theta}} \tilde{\phi}(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{A}(\boldsymbol{\theta})\mathbf{w}(\boldsymbol{\theta}) - \mathbf{c}\|^2 + \frac{\lambda}{2}\|\mathbf{L}\mathbf{w}(\boldsymbol{\theta})\|^2 + \frac{\beta}{2}\|\mathbf{M}\boldsymbol{\theta}\|^2$$

Variable Projection - 2

$$\min_{\boldsymbol{\theta}} \tilde{\phi}(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{A}(\boldsymbol{\theta})\mathbf{w}(\boldsymbol{\theta}) - \mathbf{c}\|^2 + \frac{\lambda}{2}\|\mathbf{L}\mathbf{w}(\boldsymbol{\theta})\|^2 + \frac{\beta}{2}\|\mathbf{M}\boldsymbol{\theta}\|^2$$

Optimality condition:

$$\nabla\tilde{\phi}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}\phi(\boldsymbol{\theta}, \mathbf{w}) + \nabla_{\boldsymbol{\theta}}\mathbf{w}(\boldsymbol{\theta})\nabla_{\mathbf{w}}\phi(\boldsymbol{\theta}, \mathbf{w}) \stackrel{!}{=} 0.$$

Less complicated than it seems since

$$\nabla_{\mathbf{w}}\phi(\boldsymbol{\theta}, \mathbf{w}(\boldsymbol{\theta})) = \mathbf{A}(\boldsymbol{\theta})^\top(\mathbf{A}(\boldsymbol{\theta})\mathbf{w}(\boldsymbol{\theta}) - \mathbf{c}) + \lambda\mathbf{L}^\top\mathbf{L}\mathbf{w}(\boldsymbol{\theta}) = 0$$

Discussion:

- ignore second term in gradient computation
- apply gradient descent/NLCG/BFGS to minimize $\tilde{\phi}$
- solve least-squares problem in each evaluation of $\tilde{\phi}$
- gradient is only correct if LS problem is solved exactly

Variable Projection for Single Layer

$$\min_{\boldsymbol{\theta},\mathbf{W}} E(\mathbf{W}\sigma(\mathbf{K}(\boldsymbol{\theta})\mathbf{Y}), \mathbf{C}) + \lambda R(\boldsymbol{\theta}, \mathbf{W})$$

Assume that the regularizer is separable, i.e.,

$$R(\boldsymbol{\theta}, \mathbf{W}) = R_1(\boldsymbol{\theta}) + R_2(\mathbf{W})$$

and that R_2 is convex and smooth. Hence, the projection requires solving the regularized classification problem

$$\mathbf{W}(\boldsymbol{\theta}) = \arg\min_{\mathbf{W}} E(\mathbf{W}\sigma(\mathbf{K}(\boldsymbol{\theta})\mathbf{Y}), \mathbf{C}) + \lambda R_2(\mathbf{W})$$

practical considerations:

- solve for $\mathbf{W}(\boldsymbol{\theta})$ using Newton (need accuracy)
- need good solver to approximate gradient w.r.t. $\boldsymbol{\theta}$ well
- use Gauss-Newton or steepest descent to solve for $\boldsymbol{\theta}$

Stochastic Optimization

Assume that each $\mathbf{y}_i$, $\mathbf{c}_i$ pair is drawn from some (unknown probability distribution).
Then, we can interpret the learning problem as minimizing the expected value of the cross entropy, e.g., in linear regression

$$E(\boldsymbol{\theta}, \mathbf{W}) = \mathbb{E}\left(\frac{1}{2}\|\mathbf{W}\sigma(\mathbf{K}(\boldsymbol{\theta})\mathbf{y} + \mathbf{b}) - \mathbf{c}\|^2\right)$$

This is a stochastic optimization problem [1]. One idea:
Stochastic Approximation: Design iteration $(\boldsymbol{\theta}_k, \mathbf{W}_k) \to (\boldsymbol{\theta}^*, \mathbf{W}^*)$ so that expected value decreases.
Example: Stochastic Gradient Descent, ADAM, ...
Pro: sample can be small (*mini batch*)
Con: how to monitor objective, linesearch, descent, ...

Stochastic Average Approximation

Alternative way to solve stochastic optimization problem

$$E(\mathbf{W}) = \mathbb{E}\left(\frac{1}{2}\|\mathbf{W}\sigma(\mathbf{K}(\boldsymbol{\theta})\mathbf{y} + \mathbf{b}) - \mathbf{c}\|^2\right)$$

Pick relatively large sample $S \subset \{1, \ldots, n\}$ and use deterministic optimization method to solve

$$\min_{\boldsymbol{\theta}, \mathbf{W}} \frac{1}{2|S|} \sum_{s \in S} \|\mathbf{W}\sigma(\mathbf{K}(\boldsymbol{\theta})\mathbf{y}_s + \mathbf{b}) - \mathbf{c}_s^\top\|^2.$$

Pro: use your favorite solver, linesearch, stopping...
Con: large batches needed

Note: Sample stays fixed during iteration.

Experiment: Peaks

Compare the three approaches for training a single layer neural network

- `ESingleLayer_PeaksSGD.m` - stochastic gradient descent
- `ESingleLayer_PeaksNewtonCG.m` - Newton CG with block-diagonal Hessian approximation
- `ESingleLayer_PeaksVarPro.m` - Fully coupled solver. Eliminate $\boldsymbol{\theta}$ and use steepest descent/Newton CG for reduced problem.

Example: Conditioning of Single Layer Training

Consider the regression problem with a single neural network layer

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

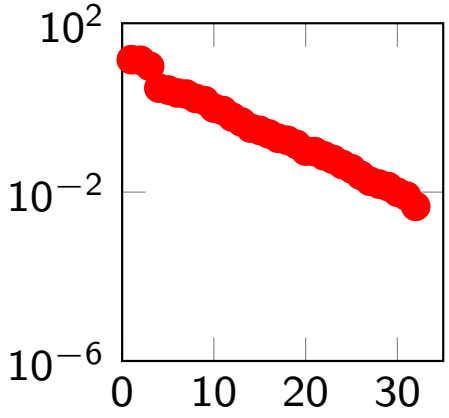
The problem above is a non-linear least squares problem (NNLS). Common to look at the Jacobian of $\mathbf{r} = \mathrm{vec}(\mathbf{R})$, i.e., $\mathbf{J} = [\mathbf{J}_\mathbf{W}\ \mathbf{J}_\mathbf{K}]$ where

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

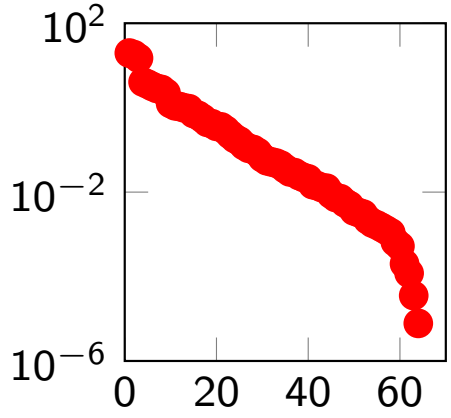
(here, we vectorized $\mathbf{R}$, $\mathbf{I}$ is identity, and $\otimes$ is the Kronecker product)

Example: Condition Numbers

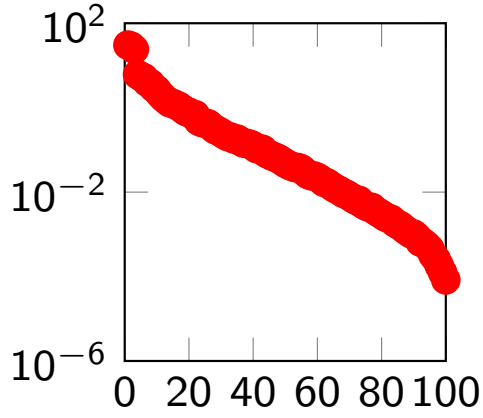
sing. vals. $m = 8$



sing. vals. $m = 16$



sing. vals. $m = 32$



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

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

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

Deep Neural Networks

Why Deep Networks?

- Universal approximation theorem of NN suggests that we can approximate **any** function by two layers.
- But - The width of the layer can be very large $\mathcal{O}(n \cdot n_f)$
- Deeper architectures can lead to more efficient descriptions of the problem.
  (No real proof but lots of practical experience)

Deep Neural Networks

How deep is deep?
We will answer this question later ...

Until recently, the standard architecture was

$$\begin{aligned} \mathbf{Y}_1 &= \sigma(\mathbf{K}_0\mathbf{Y}_0 + \mathbf{b}_0) \\ \vdots &= \vdots \\ \mathbf{Y}_N &= \sigma(\mathbf{K}_{N-1}\mathbf{Y}_{N-1} + \mathbf{b}_{N-1}) \end{aligned}$$

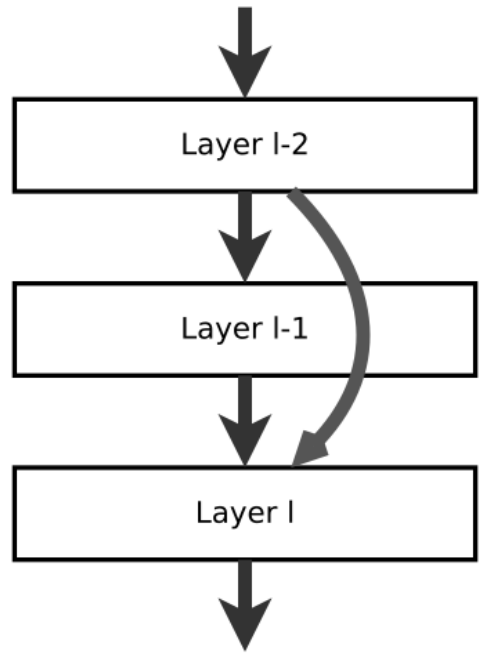
And use $\mathbf{Y}_N$ to classify. This leads to the optimization problem

$$\min_{\mathbf{K}_{0,\ldots,N-1},\mathbf{b}_{0,\ldots,N-1},\mathbf{W}} E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_1,\ldots,\mathbf{K}_{N-1},\mathbf{b}_1,\ldots,\mathbf{b}_{N-1}),\mathbf{C}^{\text{obs}}\right)$$

Deep Neural Networks in Practice

(Some) challenges:

- Computational costs (architecture have millions or billions of parameters)
- difficult to design
- difficult to train (exploding/vanishing gradients)
- unpredictable performance



In 2015, He et al. [5, 6] came with a new architecture that solves many of the problems

Residual Neural Networks

Simplified Residual Neural Network

Residual Network

$$\begin{aligned}
\mathbf{Y}_1 &= \mathbf{Y}_0 + \sigma(\mathbf{K}_0\mathbf{Y}_0 + \mathbf{b}_0) \\
\vdots &= \vdots \\
\mathbf{Y}_N &= \mathbf{Y}_{N-1} + \sigma(\mathbf{K}_{N-1}\mathbf{Y}_{N-1} + \mathbf{b}_{N-1})
\end{aligned}$$

And use $\mathbf{Y}_N$ to classify. This leads to the optimization problem

$$\min_{\mathbf{K}_{0,\ldots,N-1},\mathbf{b}_{0,\ldots,N-1},\mathbf{W}} E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_1,\ldots,\mathbf{K}_{N-1},\mathbf{b}_1,\ldots,\mathbf{b}_{N-1}),\mathbf{C}^{\text{obs}}\right)$$

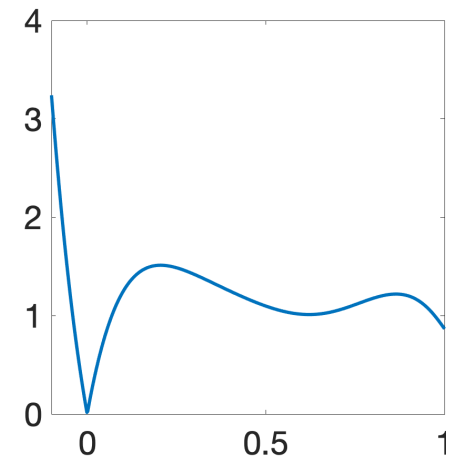
Leads to smoother objective function [10].

Neural Net vs. ResNet

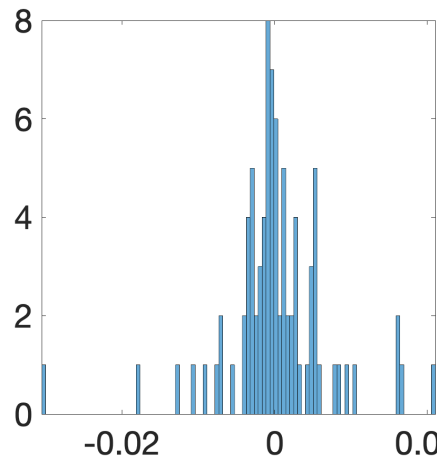
Rough comparison: Use $n_f = 4$, $n_c = 1$ and build neural net with 5 hidden layers of 4.

1. f_{NN}, neural net: $\mathbf{Y}_{l+1} = \sigma(\mathbf{K}_l \mathbf{Y}_l)$
2. f_{Res}, ResNet: $\mathbf{Y}_{l+1} = \mathbf{Y}_l + \sigma(\mathbf{K}_l \mathbf{Y}_l)$

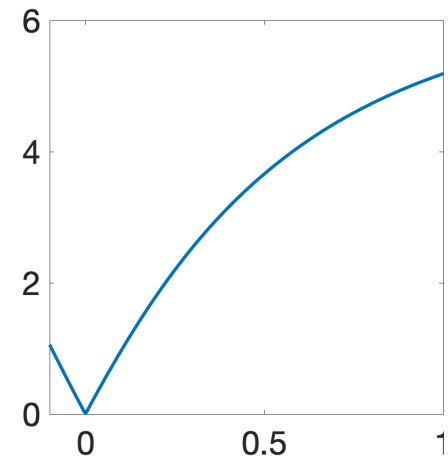
Generate two sets of random weights $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_0$ and define $\boldsymbol{\theta}(t) = t\boldsymbol{\theta}_0 + (1-t)\boldsymbol{\theta}_1$.



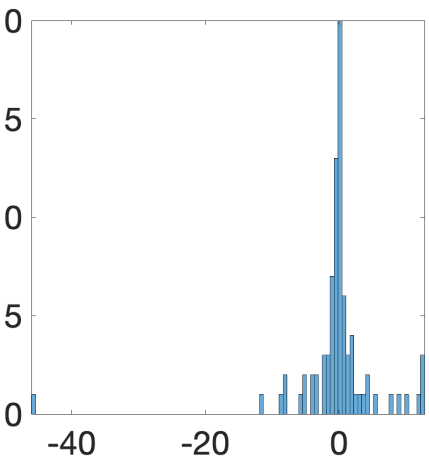
$\frac{\|f_{\mathrm{NN}}(\mathbf{Y},\boldsymbol{\theta}(t)) - f_{\mathrm{NN}}(\mathbf{Y},\boldsymbol{\theta}(0)\|}{\|f_{\mathrm{NN}}(\mathbf{Y},\boldsymbol{\theta}(0))\|}$



$\mathrm{hist}(\nabla J_{\mathrm{NN}})$



$\frac{\|f_{\mathrm{Res}}(\mathbf{Y},\boldsymbol{\theta}(t)) - f_{\mathrm{Res}}(\mathbf{Y},\boldsymbol{\theta}(0)\|}{\|f_{\mathrm{Res}}(\mathbf{Y},\boldsymbol{\theta}(0))\|}$



$\mathrm{hist}(\nabla J_{\mathrm{Res}})$

see `EReseNet_vs_NeuralNet.m`

Experiment: Peaks

Compare the three approaches for training a residual neural network

- `EResNet_PeaksSGD.m` - stochastic gradient descent
- `EResNet_PeaksNewtonCG.m` - Newton CG with block-diagonal Hessian approximation
- `EResNet_PeaksVarPro.m` - Fully coupled solver. Eliminate $\boldsymbol{\theta}$ and use steepest descent/Newton CG for reduced problem.

Part 2 Summary: Nonlinear Models

- main idea: overcome limitations of linear model by (learned) feature transformation
- single layer neural networks
  - illustration of universal approximation theorem
  - random transformation $\leadsto$ extreme learning
  - optimize transformation $\leadsto$ non-convex optimization
- Parametric models
  - design, e.g., to enable fast computation
  - most famous: convolutional neural network
- deep neural networks
  - multilayer perceptron: vanishing gradients
  - residual neural networks: smoother objective
  - ResNets can be very deep (∞ as we see next)

References

[1] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *arXiv preprint [stat.ML] (1606.04838v1)*, 2016.

[2] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[3] G. Golub and V. Pereyra. Separable nonlinear least squares: the variable projection method and its applications. *Inverse Problems*, 19:R1–R26, 2003.

[4] G. H. Golub and V. Pereyra. The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. *SIAM Journal on Numerical Analysis*, 10(2):413–432, 1973.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[7] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, Dec. 2006.

[9] Y. LeCun, B. E. Boser, and J. S. Denker. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

References (cont.)

[10] H. Li, Z. Xu, G. Taylor, and T. Goldstein. Visualizing the Loss Landscape of Neural Nets. 2017.

[11] D. P. O'Leary and B. W. Rust. Variable projection for nonlinear least squares problems. *Computational Optimization and Applications. An International Journal*, 54(3):579–593, 2013.