Convolutional Neural Networks

Numerical Methods for Deep Learning

Motivation

Recall single layer

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

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 $p \ll m \cdot n_f$.

Learning Objective: Convolutional Neural Networks

In this module we discuss parametric models, particularly CNNs.

Learning tasks:

- image classification
- image segmentation
- PDE solvers(?)

Numerical methods:

- structured matrix computation
- PDE-based regularizers
- parallel computing (GPUs,...)

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

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

  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

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

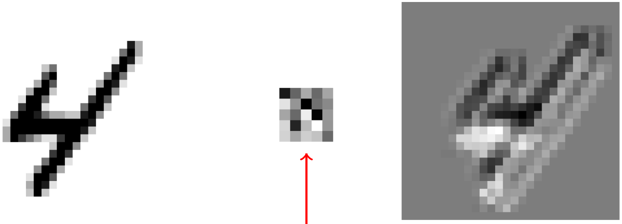
verify that this can be written as $K(\boldsymbol{\theta}) = \mathrm{mat}(\mathbf{Q}\,\boldsymbol{\theta})$ where

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

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

Convolutional Neural Networks [4]

$\mathbf{y} \in \mathbb{R}^{28\times 28}$ 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

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

$$
\begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \\ \mathbf{z}_4 \end{pmatrix} = \begin{pmatrix} \theta_3 & \theta_2 & \theta_1 & & & \\ & \theta_3 & \theta_2 & \theta_1 & & \\ & & \theta_3 & \theta_2 & \theta_1 & \\ & & & \theta_3 & \theta_2 & \theta_1 \end{pmatrix} \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \\ \mathbf{w}_2 \end{pmatrix}
$$

Different boundary conditions lead to different structures

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

$$
\begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \\ \mathbf{z}_4 \end{pmatrix} = \begin{pmatrix} \theta_2 & \theta_1 & & \\ \theta_3 & \theta_2 & \theta_1 & \\ & \theta_3 & \theta_2 & \theta_1 \\ & & \theta_3 & \theta_2 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \end{pmatrix}
$$

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

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

Extension: 2D Convolution

Example: Let $\mathbf{y}, \mathbf{z}, \boldsymbol{\theta} \in \mathbb{R}^{3\times 3}$ and assume periodic BCs then

$$\begin{aligned}\mathbf{z}_{21} &= \boldsymbol{\theta}_{33}\mathbf{y}_{13} + \boldsymbol{\theta}_{32}\mathbf{y}_{11} + \boldsymbol{\theta}_{31}\mathbf{y}_{12}\\ &\quad + \boldsymbol{\theta}_{23}\mathbf{y}_{23} + \boldsymbol{\theta}_{22}\mathbf{y}_{21} + \boldsymbol{\theta}_{21}\mathbf{y}_{22}\\ &\quad + \boldsymbol{\theta}_{13}\mathbf{y}_{33} + \boldsymbol{\theta}_{12}\mathbf{y}_{31} + \boldsymbol{\theta}_{11}\mathbf{y}_{32}\end{aligned}$$

In matrix form, this gives

$$\begin{pmatrix} \mathbf{z}_{11}\\ \mathbf{z}_{21}\\ \mathbf{z}_{31}\\ \mathbf{z}_{12}\\ \mathbf{z}_{22}\\ \mathbf{z}_{32}\\ \mathbf{z}_{13}\\ \mathbf{z}_{23}\\ \mathbf{z}_{33} \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc}
\boldsymbol{\theta}_{22} & \boldsymbol{\theta}_{12} & \boldsymbol{\theta}_{32} & \boldsymbol{\theta}_{21} & \boldsymbol{\theta}_{11} & \boldsymbol{\theta}_{31} & \boldsymbol{\theta}_{23} & \boldsymbol{\theta}_{13} & \boldsymbol{\theta}_{33}\\
\boldsymbol{\theta}_{32} & \boldsymbol{\theta}_{22} & \boldsymbol{\theta}_{12} & \boldsymbol{\theta}_{31} & \boldsymbol{\theta}_{21} & \boldsymbol{\theta}_{11} & \boldsymbol{\theta}_{33} & \boldsymbol{\theta}_{23} & \boldsymbol{\theta}_{13}\\
\boldsymbol{\theta}_{12} & \boldsymbol{\theta}_{32} & \boldsymbol{\theta}_{22} & \boldsymbol{\theta}_{11} & \boldsymbol{\theta}_{31} & \boldsymbol{\theta}_{21} & \boldsymbol{\theta}_{13} & \boldsymbol{\theta}_{33} & \boldsymbol{\theta}_{23}\\ \hline
\boldsymbol{\theta}_{23} & \boldsymbol{\theta}_{13} & \boldsymbol{\theta}_{33} & \boldsymbol{\theta}_{22} & \boldsymbol{\theta}_{12} & \boldsymbol{\theta}_{32} & \boldsymbol{\theta}_{21} & \boldsymbol{\theta}_{11} & \boldsymbol{\theta}_{31}\\
\boldsymbol{\theta}_{33} & \boldsymbol{\theta}_{23} & \boldsymbol{\theta}_{13} & \boldsymbol{\theta}_{32} & \boldsymbol{\theta}_{22} & \boldsymbol{\theta}_{12} & \boldsymbol{\theta}_{31} & \boldsymbol{\theta}_{21} & \boldsymbol{\theta}_{11}\\
\boldsymbol{\theta}_{13} & \boldsymbol{\theta}_{33} & \boldsymbol{\theta}_{23} & \boldsymbol{\theta}_{12} & \boldsymbol{\theta}_{32} & \boldsymbol{\theta}_{22} & \boldsymbol{\theta}_{11} & \boldsymbol{\theta}_{31} & \boldsymbol{\theta}_{21}\\ \hline
\boldsymbol{\theta}_{21} & \boldsymbol{\theta}_{11} & \boldsymbol{\theta}_{31} & \boldsymbol{\theta}_{23} & \boldsymbol{\theta}_{13} & \boldsymbol{\theta}_{33} & \boldsymbol{\theta}_{22} & \boldsymbol{\theta}_{12} & \boldsymbol{\theta}_{32}\\
\boldsymbol{\theta}_{31} & \boldsymbol{\theta}_{21} & \boldsymbol{\theta}_{11} & \boldsymbol{\theta}_{33} & \boldsymbol{\theta}_{23} & \boldsymbol{\theta}_{13} & \boldsymbol{\theta}_{32} & \boldsymbol{\theta}_{22} & \boldsymbol{\theta}_{12}\\
\boldsymbol{\theta}_{11} & \boldsymbol{\theta}_{31} & \boldsymbol{\theta}_{21} & \boldsymbol{\theta}_{13} & \boldsymbol{\theta}_{33} & \boldsymbol{\theta}_{23} & \boldsymbol{\theta}_{12} & \boldsymbol{\theta}_{32} & \boldsymbol{\theta}_{22}
\end{array}\right) \begin{pmatrix} \mathbf{y}_{11}\\ \mathbf{y}_{21}\\ \mathbf{y}_{31}\\ \mathbf{y}_{12}\\ \mathbf{y}_{22}\\ \mathbf{y}_{32}\\ \mathbf{y}_{13}\\ \mathbf{y}_{23}\\ \mathbf{y}_{33} \end{pmatrix}$$

good news: this matrix is BCCB (block circulant with circulant blocks)

2D Convolution using FFTs

Since the 2D convolution operator is BCCB, we still have that

$$\mathbf{K}(\boldsymbol{\theta}) = \mathbf{F}^* \mathrm{diag}(\lambda(\boldsymbol{\theta}))\mathbf{F}, \quad \lambda(\boldsymbol{\theta}) = \mathbf{F}(\mathbf{K}(\boldsymbol{\theta})\mathbf{u}_1).$$

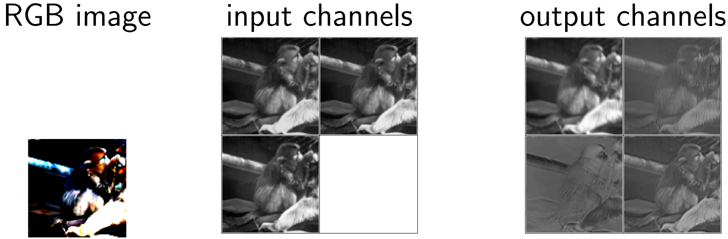
Differences:

- $\mathbf{F}$ and $\mathbf{F}^*$ now refer to 2D Fourier transform and its inverse (`fft2` and `ifft2`), respectively.
- need to find an efficient way to build first column of $\mathbf{K}(\boldsymbol{\theta})$ and encode that using $\mathbf{Q}$.

All else stays the same and extends also to higher dimensions (like for videos).

For more details on convolutions and structured matrices see [3]. For FFT-based implementations of CNNs see [5, 6]. Note that FFTs enable also to compute $\mathbf{K}(\boldsymbol{\theta})^{-1}\mathbf{Y}$ efficiently.

The Width of a CNN



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

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

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

Other Operators for Imaging Tasks

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. maximum or 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

Regularizers for Image Classification

Focus on last layer: Goal is to train weights $\mathbf{W} \in \mathbb{R}^{n_c \times m}$ to express the relation between $\mathbf{Z} = \sigma(\mathbf{KY} + \mathbf{b}\mathbf{e}_n^\top)$ and labels $\mathbf{C} \in \mathbb{R}^{n_c \times n}$ by solving

$$\min_{\mathbf{W}} E(\mathbf{W}) = E(\mathbf{C}, \mathbf{W}, \mathbf{Z})$$

Recall: $\mathrm{rank}\mathbf{Z} \leq \min\{m, n\}$

- $n < m$: No unique solution
- $n > m$: $\mathbf{Z}$ may still be rank-deficient/ ill-posed

Opportunities/challenges in image classification:

- when using convolutions, $\mathbf{Z}$ looks like filtered images
- data is high dimensional ($m \approx$ number of pixels/voxels/frames)
- higher resolution $\rightsquigarrow$ need more examples?
- higher resolution $\rightsquigarrow$ larger rank?

Regularization

If Hessian $\nabla^2 E$ highly ill-conditioned, regularization is needed.

- Symptom: weights are large or oscillatory.
- Alternative: Estimate condition number (costly!), early stopping

Solution: Use regularization to obtain a well-posed problem

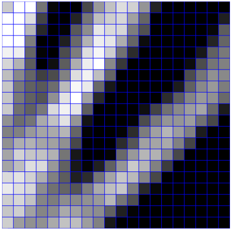
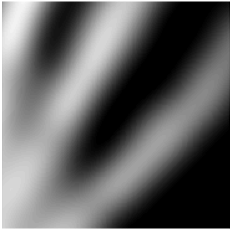
$$\min_{W} \ \phi(\mathbf{W}) = E(\mathbf{W}) + \lambda R(\mathbf{W}),$$

where

- R is a regularizer, $R(\mathbf{W})$ large when $\mathbf{W}$ is irregular and small otherwise
- λ is a regularization parameter (needs to be chosen)
- Mathematically: R makes sure $\mathbf{W}^*$ lies in desired function space (and is sufficiently *regular*).

Excellent references include [1, 2, 7].

What is an Image?

Digital images are arrays $\mathbf{U} = \mathbb{R}^{m_1 \times m_2 \times c}$ ($c = 1 \rightsquigarrow$ grey only).

- perhaps most common interpretation in image processing

Continuous point of view: Images are functions supported on a domain $\Omega \in \mathbb{R}^2$ $u : \Omega \to \mathbb{R}^c$.

- choose function space (e.g., continuous, differentiable)
- discretize on regular grids $\rightsquigarrow$ digital image
- apply operators to images (e.g., gradient in edge detection)

Type of Regularization

Classical Tikhonov (aka weight decay)

$$R(\mathbf{W}) = \frac{1}{2}\|\mathbf{W}\|_F^2$$

requires elements to be small.

When $\mathbf{Z}$ contains images, also columns in $\mathbf{W}$ can be seen as images

$$\mathbf{w}^\top \mathbf{y} \approx \int_\Omega w(\boldsymbol{\xi}) y(\boldsymbol{\xi}) d\boldsymbol{\xi}.$$

General Tikhonov: Let $\mathbf{L}$ be a given matrix

$$R(\mathbf{W}) = \frac{1}{2}\|\mathbf{LW}\|_F^2$$

If $\mathbf{L}$ is discrete derivative operator, entries need to be smooth.

Discretization of ∇^2

Idea: Ensure classifier is smooth by using Laplacian $\mathbf{L} \approx \nabla^2$.

Finite difference in 1D: Let $\mathbf{u} \in \mathbb{R}^N$ be discretization of $u : [0, 1] \to \mathbb{R}$ on regular grid with pixel size $h = 1/N$

$$\nabla^2 u(x_j) \approx \frac{1}{h^2}(-2\mathbf{u}_j + \mathbf{u}_{j-1} + \mathbf{u}_{j+1}).$$

Code in 1D

```
L1D = @(N,h) 1/h^2 *...
 spdiags(ones(n,1)  * [1  -2  1],-1:1,N,N)
```

Finite difference in 2D: Let $\mathbf{U} \in \mathbb{R}^{N \times N}$ be discretization of $u : [0, 1]^2 \to \mathbb{R}$ on regular grid with pixel size $h = 1/N$

$$\nabla^2 I(x_{ij}) \approx \frac{1}{h^2}(-4\mathbf{I}_{ij} + \mathbf{I}_{i-1j} + \mathbf{I}_{i+1j} + \mathbf{I}_{ij-1} + \mathbf{I}_{ij+1}).$$

Discretization of ∇^2

In 2D $\quad \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$

Use Kroneker products

$$\mathrm{vec}(\mathbf{LUI}) = (\mathbf{I}^\top \otimes \mathbf{L})\mathrm{vec}(\mathbf{U}).$$

Code in 2D

```
L = kron(speye(m2), L1D(m1,h1)) + ...
      kron(L1D(m2,h2),speye(m1) );
```

More about discrete ∇^2

Note that $\mathbf{L}$ can also be written as a convolution

$$\mathbf{L} = \frac{1}{h^2} \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} * \mathbf{U}.$$

In general - any differential operator with constant coefficients can be written as convolution and vice versa.

Continuous interpretation allows re-computing a convolution kernel for different image resolutions.

Σ: Convolutional Neural Networks

Idea: Use image structure in forward propagation and classification.

$$\min_{\mathbf{W},\boldsymbol{\theta}} \ \phi(\mathbf{W}) = E(\mathbf{W}, \mathbf{Z}(\boldsymbol{\theta})) + \lambda R(\mathbf{W}, \boldsymbol{\theta}),$$

where $\mathbf{Z}(\boldsymbol{\theta}) = \sigma(\mathbf{K}(\boldsymbol{\theta})\mathbf{Y} + \mathbf{b}(\boldsymbol{\theta}))$.

Discussion:

- note that convolutions are not the only option to parameterize a neural network layer
- options for convolutions: write own code (full flexibility but slow!), use library (fast, but not flexible), FFT (good trade-off, can compute inverses)
- regularization: use image structure to enforce smoothness

References

[1] P. C. Hansen. *Rank-deficient and discrete ill-posed problems*. SIAM Monographs on Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998.

[2] P. C. Hansen. *Discrete inverse problems*, volume 7 of *Fundamentals of Algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2010.

[3] P. C. Hansen, J. G. Nagy, and D. P. O'Leary. *Deblurring Images: Matrices, Spectra and Filtering*. Matrices, Spectra, and Filtering. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006.

[4] 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.

[5] M. Mathieu, M. Henaff, and Y. LeCun. Fast Training of Convolutional Networks through FFTs. *arxiv preprint 1312.5851*, 2013.

[6] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. Fast Convolutional Nets With fbfft: A GPU Performance Evaluation. *arxiv preprint [cs.LG] 1412.7580v3*, 2014.

[7] C. R. Vogel. *Computational Methods for Inverse Problems*. SIAM, Philadelphia, 2002.