

Parametric Models

Numerical Methods for Deep Learning

Motivation

Recall single layer

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

where $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$, $\mathbf{K} \in \mathbb{R}^{m \times n_f}$, $b \in \mathbb{R}$, 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 $m = n_f \leadsto 614,656$ 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:** $\mathbf{K}(\theta)$ where $\theta \in \mathbb{R}^p$ with $p \ll m \cdot n_f$.

Some Simple Parametric Models

- ▶ Diagonal scaling:

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

Advantage: preserves size and structure of data.

- ▶ Antisymmetric kernel

$$\mathbf{K}(\theta) = \begin{pmatrix} 0 & \theta_1 & \theta_2 \\ -\theta_1 & 0 & \theta_3 \\ -\theta_2 & -\theta_3 & 0 \end{pmatrix}$$

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

- ▶ M -matrix

$$\mathbf{K}(\theta) = \begin{pmatrix} \theta_1 + \theta_2 & -\theta_1 & -\theta_2 \\ -\theta_3 & \theta_3 + \theta_4 & -\theta_4 \\ -\theta_5 & -\theta_6 & \theta_5 + \theta_6 \end{pmatrix} \quad \theta \geq 0$$

Advantage: like differential operator

Differentiating Parametric Models

Need derivatives of model to optimize θ in

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

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

Note that all previous models are linear in the following sense

$$\mathbf{K}(\theta) = \text{mat}(\mathbf{Q} \theta)$$

Therefore, matrix-vector products with the Jacobian simply are

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

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

Example: Derivative of M-matrix

$$\mathbf{K}(\theta) = \begin{pmatrix} \theta_1 + \theta_2 & -\theta_1 & -\theta_2 \\ -\theta_3 & \theta_3 + \theta_4 & -\theta_4 \\ -\theta_5 & -\theta_6 & \theta_5 + \theta_6 \end{pmatrix} \quad \theta \geq 0$$

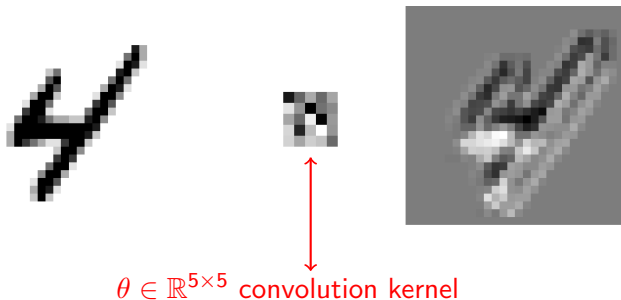
verify that this can be written as $K(\theta) = \text{mat}(\mathbf{Q} \theta)$ where

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

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

Convolutional Neural Networks [2]

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



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

Convolutions in 1D

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

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

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

$$\mathbf{z}_1 = \theta_3 \mathbf{w}_1 + \theta_2 \mathbf{x}_1 + \theta_1 \mathbf{x}_2$$

$$\mathbf{z}_2 = \theta_3 \mathbf{x}_1 + \theta_2 \mathbf{x}_2 + \theta_1 \mathbf{x}_3$$

$$\mathbf{z}_3 = \theta_3 \mathbf{x}_2 + \theta_2 \mathbf{x}_3 + \theta_1 \mathbf{x}_4$$

$$\mathbf{z}_4 = \theta_3 \mathbf{x}_3 + \theta_2 \mathbf{x}_4 + \theta_1 \mathbf{w}_2$$

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 $\mathbf{w}_2 = \mathbf{x}_1$

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

this is a *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

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

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

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

Coding: 1D Convolution using FFTs

Let $\theta \in \mathbb{R}^3$ 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

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

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

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

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

Need to differentiate eigenvalues w.r.t. θ . Note linearity

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

Derivatives of 1D Convolution - 2

Assume we have

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

Then mat-vecs with Jacobian are easy to compute

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

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

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

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

Extension 1: Many Examples

Let $n > 1$. In MATLAB must avoid for-loop over examples.

$$\mathbf{K}(\theta)\mathbf{Y} = \text{real}(\mathbf{F}^* \text{diag}(\lambda(\theta))\mathbf{F}\mathbf{Y})$$

$$\mathbf{K}(\theta)^\top \mathbf{Z} = \text{real}(\mathbf{F} \text{diag}(\lambda(\theta))\mathbf{F}^* \mathbf{Z})$$

These require almost no change to the code. For the Jacobians, we need to re-order slightly and get

$$\mathbf{J}_\theta(\mathbf{K}(\theta)\mathbf{Y}) = \text{real}(\mathbf{F}^* \text{diag}(\mathbf{F}\mathbf{Q}\mathbf{v})\mathbf{F}\mathbf{Y})$$

and for the transpose we need to sum over examples

$$\mathbf{J}_\theta(\mathbf{K}(\theta)\mathbf{Y})^\top \mathbf{W} = \text{real}(\mathbf{Q}^\top \mathbf{F}((\mathbf{F}\mathbf{Y}) \odot (\mathbf{F}^* \mathbf{W})\mathbf{e}_n))$$

Extension 2: 2D Convolution

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

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

In matrix form, this gives

$$\begin{pmatrix} \mathbf{z}_{11} \\ \mathbf{z}_{21} \\ \mathbf{z}_{31} \\ \mathbf{z}_{12} \\ \mathbf{z}_{22} \\ \mathbf{z}_{32} \\ \mathbf{z}_{13} \\ \mathbf{z}_{23} \\ \mathbf{z}_{33} \end{pmatrix} = \begin{pmatrix} \theta_{22} & \theta_{12} & \theta_{32} & \theta_{21} & \theta_{11} & \theta_{31} & \theta_{23} & \theta_{13} & \theta_{33} \\ \theta_{32} & \theta_{22} & \theta_{12} & \theta_{31} & \theta_{21} & \theta_{11} & \theta_{33} & \theta_{23} & \theta_{13} \\ \theta_{12} & \theta_{32} & \theta_{22} & \theta_{11} & \theta_{31} & \theta_{21} & \theta_{13} & \theta_{33} & \theta_{23} \\ \theta_{23} & \theta_{13} & \theta_{33} & \theta_{22} & \theta_{12} & \theta_{32} & \theta_{21} & \theta_{11} & \theta_{31} \\ \theta_{33} & \theta_{23} & \theta_{13} & \theta_{32} & \theta_{22} & \theta_{12} & \theta_{31} & \theta_{21} & \theta_{11} \\ \theta_{13} & \theta_{33} & \theta_{23} & \theta_{12} & \theta_{32} & \theta_{22} & \theta_{11} & \theta_{31} & \theta_{21} \\ \theta_{21} & \theta_{11} & \theta_{31} & \theta_{23} & \theta_{13} & \theta_{33} & \theta_{22} & \theta_{12} & \theta_{32} \\ \theta_{31} & \theta_{21} & \theta_{11} & \theta_{33} & \theta_{23} & \theta_{13} & \theta_{32} & \theta_{22} & \theta_{12} \\ \theta_{11} & \theta_{31} & \theta_{21} & \theta_{13} & \theta_{33} & \theta_{23} & \theta_{12} & \theta_{32} & \theta_{22} \end{pmatrix} \begin{pmatrix} \mathbf{y}_{11} \\ \mathbf{y}_{21} \\ \mathbf{y}_{31} \\ \mathbf{y}_{12} \\ \mathbf{y}_{22} \\ \mathbf{y}_{32} \\ \mathbf{y}_{13} \\ \mathbf{y}_{23} \\ \mathbf{y}_{33} \end{pmatrix}$$

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

Extension 2: 2D Convolution using FFTs

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

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

Differences:

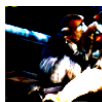
- ▶ \mathbf{F} and \mathbf{F}^* now refer to 2D Fourier transform and its inverse (`fft2` and `ifft2`), respectively.
- ▶ need to find an efficient way to build first column of $\mathbf{K}(\theta)$ and encode that using \mathbf{Q} .

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

For more details on convolutions and structured matrices see [1]. For FFT-based implementations of CNNs see [3, 4].

Extension 3: 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

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

where \mathbf{K}^{ij} is a 2D convolution operator with stencil θ^{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 (ideas for projects)

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

References

- [1] 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.
- [2] 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.
- [3] M. Mathieu, M. Henaff, and Y. LeCun. Fast Training of Convolutional Networks through FFTs. *arxiv preprint 1312.5851*, 2013.
- [4] 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.