

Convolutional Neural Networks and PDEs

Numerical Methods for Deep Learning

Recall - Deep Network

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

Problems/Challenges

- ▶ The state can grow (explosion) $\Rightarrow \mathbf{Y}_N$ very sensitive to \mathbf{Y}_0 and \mathbf{K}_j (exploding gradients)
- ▶ The state can go to 0 $\Rightarrow \mathbf{Y}_N$ independent to \mathbf{Y}_0 and to the \mathbf{K}_j 's (vanishing gradients)
- ▶ In both cases the optimization can fail

Today: Deep Convolutional Neural Networks

Deep network with convolution operators parameterized by stencils. Common

$$\mathbf{Y}_{j+1} = \mathbf{P}_j \mathbf{Y}_j + h \mathbf{K}_{j,2} \sigma(\mathbf{K}_{j,1} \mathbf{Y}_j + b_j)$$

Motivation for second convolution?

- ▶ approximation properties of double layer
- ▶ if $\sigma(x) = \max(x, 0)$ entries in \mathbf{Y} can only grow.

Learning tasks:

- ▶ image classification
- ▶ semantic segmentation

Convolutions and PDEs

Let \mathbf{y} be 1D grid function, $\mathbf{y} \leftrightarrow y$ (n cells of width $h_x = 1/n$)

$$\begin{aligned} K(\boldsymbol{\theta})\mathbf{y} &= [\boldsymbol{\theta}_1 \ \boldsymbol{\theta}_2 \ \boldsymbol{\theta}_3] * \mathbf{y} \\ &= \left(\frac{\beta_1}{4} [1 \ 2 \ 1] + \frac{\beta_2}{2h_x} [-1 \ 0 \ 1] + \frac{\beta_3}{h_x^2} [-1 \ 2 \ -1] \right) * \mathbf{y} \end{aligned}$$

Relation between β and $\boldsymbol{\theta}$ given by

$$\underbrace{\begin{pmatrix} 1/4 & -1/(2h_x) & -1/h_x^2 \\ 1/2 & 0 & 2/h_x^2 \\ 1/4 & 1/(2h_x) & -1/h_x^2 \end{pmatrix}}_{=\mathbf{A}(h_x)} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \end{pmatrix}.$$

In the limit $h_x \rightarrow 0$ this gives

$$K(\boldsymbol{\theta}(t)) = \beta_1(t) + \beta_2(t)\partial_x + \beta_3(t)\partial_x^2.$$

Scaling Convolution Operators: Rediscretization

Let \mathbf{y}_c and \mathbf{y}_f be 1D grid functions $\mathbf{y} \leftrightarrow y$ (grid: $n_c = n$ and $n_f = 2n$ cells of width $h_c = 1/n_c$ and $h_f = 1/n_f$, respectively)

Ex: Let $y(x) = (\cos(2\pi x^4)) + x - 0.8(x - 0.5)^2$, $n = 8$, and

$$\boldsymbol{\theta}_{\text{coarse}} = (-67, 129, -59)^\top.$$

Find corresponding kernel, $\boldsymbol{\theta}_{\text{fine}}$ on fine mesh.

Strategy:

1. setup $\mathbf{A}(h_c)$ and compute β
2. setup $\mathbf{A}(h_f)$ and compute $\boldsymbol{\theta}_{\text{fine}}$

E15CoarseToFineConv1D.m

Convolution and PDEs - 2D Case

Similar arguments apply in two and more dimensions. Let $\theta \in \mathbb{R}^{3 \times 3}$ be a given stencil then

$$K(\theta) = \beta_1 + \beta_2 \partial_x + \beta_3 \partial_y + \beta_4 \partial_{xy} + \beta_5 \partial_{xx} \\ + \beta_6 \partial_{yy} + \beta_7 \partial_{x^2 y} + \beta_8 \partial_{xy^2} + \beta_9 \partial_{x^2 y^2}$$

How to get the coefficients? Let $h_x > 0$ pixel size, use

$$\begin{pmatrix} 1 & -1 & -1 & | & 1 & 1 & -1 & | & -1 & -1 & -1 \\ 2 & -2 & 0 & | & 2 & -2 & 0 & | & 2 & 0 & 2 \\ 1 & -1 & 1 & | & 1 & 1 & 1 & | & -1 & 1 & -1 \\ 2 & 0 & -2 & | & -2 & 2 & 0 & | & 0 & 2 & 2 \\ 4 & 0 & 0 & | & -4 & -4 & 0 & | & 0 & 0 & -4 \\ 2 & 0 & 2 & | & -2 & -2 & 0 & | & 0 & -2 & 2 \\ 1 & 1 & -1 & | & 1 & 1 & 1 & | & 1 & -1 & -1 \\ 2 & 2 & 0 & | & 2 & -2 & 0 & | & -2 & 0 & 2 \\ 1 & 1 & 1 & | & 1 & 1 & -1 & | & 1 & 1 & -1 \end{pmatrix} \cdot \text{diag} \begin{pmatrix} 1/16. \\ 1/6h_x \\ 1/6h_x \\ 1/4h^2 \\ 1/4h^2 \\ 1/h^2 \\ 1/2h^3 \\ 1/2h^3 \\ 1/2h^3 \end{pmatrix}$$

Algebraic Prolongation of Convolution Kernels

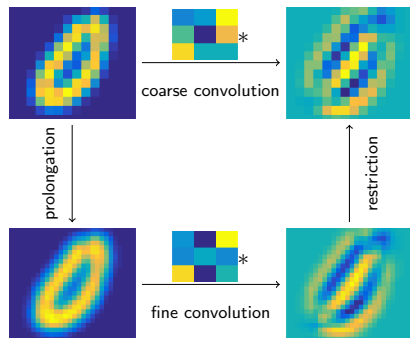
$$\mathbf{K}_H = \mathbf{R}\mathbf{K}_h\mathbf{P},$$

where

- ▶ \mathbf{K}_h fine mesh operator (given)
- ▶ \mathbf{R} restriction (e.g., averaging)
- ▶ \mathbf{P} prolongation (e.g., interpolation)

Remarks:

- ▶ Galerkin: $\mathbf{R} = \gamma\mathbf{P}^\top$
- ▶ Coarse \rightarrow Fine: unique if kernel size constant.
- ▶ only small linear solve required



Scaling Convolution Operators: Algebraic

Let $n_c = 3$, $n_f = 6$ (see E15CoarseToFineConv1D.m)

Prolongation and restriction

$$\mathbf{P} = \begin{pmatrix} 1 & & & & & \\ 3/4 & 1/4 & & & & \\ 1/4 & 3/4 & & & & \\ & 3/4 & 1/4 & & & \\ & 1/4 & 3/4 & & & \\ & & & 1 & & \end{pmatrix},$$
$$\mathbf{R} = \begin{pmatrix} 1/2 & 1/2 & & & & \\ & & 1/2 & 1/2 & & \\ & & & & 1/2 & 1/2 \end{pmatrix}$$

1. build $\mathbf{K}_H = \mathbf{R}\mathbf{K}_h\mathbf{D}$ for j th unit vector as stencil
2. extract a column associated with interior node
3. reshape and get patch around node
4. vectorize and use as one column

Stability of Convolutional ResNets

Residual Neural Network is discretization of

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

To analyze stability need to look at eigenvalues of

$$\mathbf{J}(t) = \mathbf{K}_2(t) \text{diag}(\sigma'(\mathbf{K}_1(t) \mathbf{y}(t) + b(t))) \mathbf{K}_1(t)$$

Difficult to analyze eigenvalues for general \mathbf{K}_2 and \mathbf{K}_1 .

Easy option: Set $\mathbf{K}_2 = -\mathbf{K}_1^\top$ (and drop subscript index).

Doing so gives:

$$\mathbf{J}_{\text{sym}}(t) = -\mathbf{K}(t)^\top \text{diag}(\sigma'(\mathbf{K}_1(t) \mathbf{y}(t) + b(t))) \mathbf{K}(t)$$

symmetric negative semidefinite \leadsto stability if $\partial_t \mathbf{K}$ small

E15ParabolicCNN.m

Parabolic CNN

In original Residual Net choose $\mathbf{K}_2 = -\mathbf{K}_1^\top = \mathbf{K}^\top$. This gives parabolic CNN

$$\partial_t \mathbf{Y}(t) = -\mathbf{K}(t)^\top \sigma(\mathbf{K}(t)\mathbf{Y}(t) + \mathbf{b}(t)), \quad \mathbf{Y}(0) = \mathbf{Y}_0.$$

Theorem

If σ is monotonically non-decreasing, then the forward propagation is stable, i.e., there is a $M > 0$ such that for all T

$$\|\mathbf{Y}(T) - \mathbf{Y}_\epsilon(T)\|_F \leq M \|\mathbf{Y}(0) - \mathbf{Y}_\epsilon(0)\|_F,$$

where \mathbf{Y} and \mathbf{Y}_ϵ are solutions for different initial values.

Use forward Euler discretization with h small enough

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j - h\mathbf{K}_j^\top \sigma(\mathbf{K}_j\mathbf{Y} + \mathbf{b}_j), \quad j = 0, 1, \dots, N-1$$

Similar to anisotropic diffusion (popular in image processing) [1]

Proof: Stability of Parabolic Networks

For ease of notation, assume no bias. We show that

$$\partial_t \|\mathbf{Y}(t) - \mathbf{Y}_\epsilon(t)\|_F^2 \leq 0.$$

Integrating this over $[0, T]$ yields the stability result.

Why? Note that for all $t \in [0, T]$ taking derivative gives

$$\begin{aligned} & (-\mathbf{K}(t)^\top \sigma(\mathbf{K}(t)\mathbf{Y}) + \mathbf{K}(t)^\top \sigma(\mathbf{K}(t)\mathbf{Y}_\epsilon), \mathbf{Y} - \mathbf{Y}_\epsilon) \\ & - (\sigma(\mathbf{K}(t)\mathbf{Y}) - \sigma(\mathbf{K}(t)\mathbf{Y}_\epsilon), \mathbf{K}(t)(\mathbf{Y} - \mathbf{Y}_\epsilon)) \leq 0. \end{aligned}$$

Where (\cdot, \cdot) is inner product and the inequality follows from the monotonicity of the activation function.

Relation to Total Variation

Note that the parabolic network can be seen as a gradient flow for

$$\phi(\mathbf{Y}) = s(\mathbf{K}(t)\mathbf{Y}(t) + b(t)),$$

where $\sigma(x) = s'(x)$. What is s ?

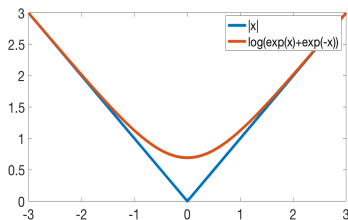
► $\sigma(x) = \max(x, 0)$:

$$s(x) = \max(x, 0)^2$$

► $\sigma(x) = \tanh(x)$:

$$s(x) = \log(\exp(x) - \exp(-x))$$

For $\mathbf{K}(t) = \nabla_{h_x}$, this shows a relation to total variation [3].



thanks to Stanley Osher for providing this example

Normalization

Goal: Make sure $\|\mathbf{Y}_j\|$ does not change too much.

1. Compute

$$\mathbf{Z}_j = \mathbf{K}_j \mathbf{Y}_j$$

2. “Normalize”

$$\hat{\mathbf{Z}}_j = N(\mathbf{Z}_j)$$

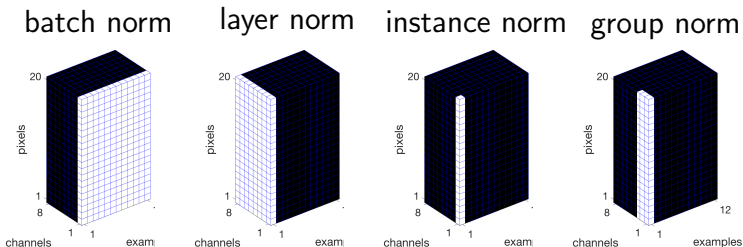
3. Update

$$\mathbf{Y}_{j+1} = \mathbf{P}_j \mathbf{Y}_j - h \mathbf{K}_j^\top \sigma(\alpha_j \hat{\mathbf{Z}}_j + \beta_j),$$

where α_j, β_j are trainable weights.

How to normalize?

Normalization



Represent image data as a multidimensional array

$$\underbrace{Height \times Width}_{pixels} \times Channels \times Examples$$

We can normalize over each one of those.

Normalization

Normalization along `dir` is done by

- ▶ Reducing the mean

$$Y = Y - \text{mean}(Y, \text{dir})$$

- ▶ Dividing by the standard deviation

$$Y = Y ./ \text{sqrt}(Y.^2 + \text{eps})$$

where $\text{eps} > 0$ is a conditioning parameter.

Questions:

- ▶ How to compute derivatives?
- ▶ The effect of batch size?
- ▶ How well does this work?

Batch Normalization

- ▶ the first normalization suggested [2]
- ▶ coupling across examples. Intuitive?
- ▶ sensitive to batch size
- ▶ works very well (i.e., better performance of SGD)

E15BatchNorm.m

Instance Normalization

Suggested recently [4] but has much older roots
Equivalent to total variation in image denoising [3]

$$TV(\mathbf{Y}) = \frac{|\nabla \mathbf{Y}|}{\sqrt{|\nabla \mathbf{Y}|^2 + \epsilon}}$$

Can we utilize this to do better?

E15InstanceNorm.m

References

- [1] Y. Chen and T. Pock. Trainable Nonlinear Reaction Diffusion: A Flexible Framework for Fast and Effective Image Restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1256–1272, June 2017.
- [2] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint [cs.LG] 1502.03167v3*, 2015.
- [3] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.
- [4] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *arxiv preprint [cs.CV] 1607.08022v3*, 2016.