Differentiating the Residual Neural Network

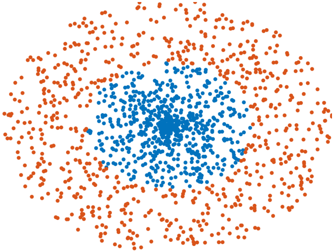
Numerical Methods for Deep Learning

Residual Network as a Path Planning Problem

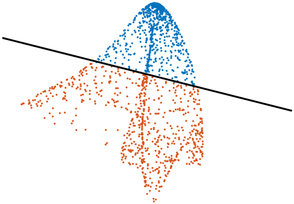
Change in notation: Moving forward it is more convenient to define $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ (transpose data matrix) and $\mathbf{C} \in \mathbb{R}^{n_c \times n}$.

$$\dot{\mathbf{Y}}(t) = \sigma(\mathbf{K}(t)\mathbf{Y}(t) + b(t)) \quad \mathbf{Y}(0) = \mathbf{Y}_0$$

The goal is to plan a path such that the transformed features, $\mathbf{Y}(T)$, can be linearly separated.



input features, $\mathbf{Y}(0)$



transformed features $\mathbf{Y}(T)$

Residual Network - Forward Propagation

Idea: Obtain forward propagation by discretizing the ODE

$$\dot{\mathbf{Y}} = \sigma(\mathbf{K}\mathbf{Y} + b) \quad \mathbf{Y}(0) = \mathbf{Y}_0$$

Example: Use forward Euler method

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

Here: $\mathbf{Y}_j$ is called the *state*, $\mathbf{K}_j, b_j$ are *controls*, and $h > 0$ is time step size.

More general forward propagation

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

Allows for changing resolution and width (and classical neural networks).

Residual Network - Classification Problem

Classification with final state by solving

$$\min_{\mathbf{W},\mathbf{K}_{0,\dots,N-1},b_{0,\dots,N-1}} E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_{0,\dots,N-1}, b_{0,\dots,N-1}), \mathbf{C}^{\mathrm{obs}}\right)$$

Need to differentiate

- E w.r.t $\mathbf{W}$ (linear classifier $\rightsquigarrow$ Lecture 3)
- $\mathcal{S}$ w.r.t $\mathbf{Y}_N$ (single layer $\rightsquigarrow$ Lecture 8)
- $\mathbf{Y}_N$ w.r.t control variables $(\mathbf{K}_{0,\dots,N-1}, b_{0,\dots,N-1})$

Having these, apply chain rule to get, e.g.,

$$\nabla_{\mathbf{K}_j} E = \left(\mathbf{J}_{\mathbf{K}_j}\mathbf{Y}_N\right)^\top \nabla_{\mathbf{Y}_N} E$$

How? Adjoint method [**?** **?**] (more general than back propagation [**?**])

Computing Derivatives - Sensitivity Equation

Idea: Differentiate the forward propagation (forward Euler) with respect to $\mathbf{K}_i$ for fixed $0 \le i \le N$. Note that

$$\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_j = 0, \quad \text{for} \quad j \le i.$$

Next, note that

$$\mathbf{J}_{\mathbf{K}_j}\mathbf{Y}_{i+1} = h\mathrm{diag}(\sigma'(\mathbf{K}_i\mathbf{Y}_i + b_i))(\mathbf{Y}_i^\top \otimes \mathbf{I})$$

Continuing like this, gives for the final state:

$$\begin{aligned} \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N &= \mathbf{P}_{N-1}\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_{N-1} \\ &+ h\mathrm{diag}(\sigma'(\cdots))\left((\mathbf{I} \otimes \mathbf{K}_{N-1})\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_{N-1}\right) \end{aligned}$$

Next: Write this as a block triangular **linear** system.

Computing Derivatives - Sensitivity Equations

Block triangular **linear** system for the gradients

$$\begin{pmatrix} \mathbf{I} & & & & \\ -\mathbf{T}_{i+1} & \mathbf{I} & & & \\ & \ddots & \ddots & & \\ & & & -\mathbf{T}_{N-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_{i+1} \\ \\ \\ \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{R}_i \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\mathbf{T}_j = \mathbf{P}_j + h\mathrm{diag}(\sigma'(\mathbf{K}_j\mathbf{Y}_j + b_j))(\mathbf{I} \otimes \mathbf{K}_j)$$

and

$$\mathbf{R}_i = h\mathrm{diag}(\sigma'(\mathbf{K}_i\mathbf{Y}_i + b_i))(\mathbf{Y}_i^\top \otimes \mathbf{I}).$$

Computing Derivatives - Sensitivity Equation

Block triangular **linear** system for the gradients

$$\underbrace{\begin{pmatrix} \mathbf{I} & & & & \\ -\mathbf{T}_{i+1} & \mathbf{I} & & & \\ & \ddots & \ddots & & \\ & & & -\mathbf{T}_{N-1} & \mathbf{I} \end{pmatrix}}_{=\mathbf{T}} \underbrace{\begin{pmatrix} \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_{i+1} \\ \\ \\ \mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N \end{pmatrix}}_{=\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}} = \underbrace{\begin{pmatrix} \mathbf{R}_i \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{=\mathbf{R}}$$

To compute matrix-vector product $(\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N)\,\mathbf{v}$

- Multiply $\mathbf{R}\,\mathbf{v}$
- Solve (forward propagate) $\mathbf{T}\,\mathbf{J}_{\mathbf{K}_i}\mathbf{Y} = \mathbf{R}\,\mathbf{v}$
- Extract the last time step

The Sensitivity Equation

Symbolically

$$\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N = \mathbf{Q}\mathbf{T}^{-1}\mathbf{R}$$

where

$$\mathbf{Q} = [0, \ldots, \mathbf{I}].$$

The transpose

$$(\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N)^\top = \mathbf{R}^\top\mathbf{T}^{-T}\mathbf{Q}^\top$$

The Sensitivity Equation

$$(\nabla_{\mathbf{K}_i}\mathbf{Y}_N)^\top = \mathbf{R}^\top \mathbf{T}^{-T}\mathbf{Q}^\top$$

$$\begin{pmatrix} \mathbf{R}_i^\top & 0 & \dots & 0 \end{pmatrix} \begin{pmatrix} \mathbf{I} & -\mathbf{T}_{i+1}^\top & & & \\ & \mathbf{I} & -\mathbf{T}_{i+2}^\top & & \\ & & \ddots & \ddots & \\ & & & \mathbf{I} & -\mathbf{T}_N \\ & & & & \mathbf{I} \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ \\ \vdots \\ \\ \mathbf{I} \end{pmatrix}$$

To multiply by the transpose

- Initialize with last step
- **solve backward** in time
- Extract the first step and multiply by $\mathbf{R}_i^\top$

More about the sensitivity equation

To compute $(\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N)^\top$ for all i's note that the same quantities are recomputed. Can be evaluated in $\mathcal{O}(N)$ steps

For gradient based method the transpose is sufficient

Newton based methods require both forward sensitivities and adjoint.

Testing Derivatives

Task 1: Programming the derivative test
as usual
Task 2: Programming the adjoint - the adjoint test
Code a - Computes $(\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N)\mathbf{v}$
Code b - Computes $(\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N)^\top\mathbf{u}$

Testing - for random $\mathbf{u}, \mathbf{v}$

$$\mathbf{u}^\top\left((\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N)\mathbf{v}\right) = \mathbf{v}^\top\left((\mathbf{J}_{\mathbf{K}_i}\mathbf{Y}_N)^\top\mathbf{u}\right)$$

References

[1] G. A. Bliss. The use of adjoint systems in the problem of differential corrections for trajectories. *JUS Artillery*, 51:296–311, 1919.

[2] A. Borzì and V. Schulz. *Computational optimization of systems governed by partial differential equations*, volume 8. SIAM, Philadelphia, PA, 2012.

[3] D. Rumelhart, G. Hinton, and J. Williams, R. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.