Neural Networks with Cheap Differential Operators

Ricky T. Q. Chen and David Duvenaud

University of Toronto, Vector Institute

Overview

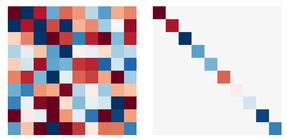
Given a function $f : \mathbb{R}^d \to \mathbb{R}^d$, we seek to obtain a vector containing its *dimension-wise* k-th order derivatives,

$$D^k_{\text{dim}} f(x) := \left[\frac{\partial^k f_1(x)}{\partial x_1^k} \quad \cdots \quad \frac{\partial^k f_d(x)}{\partial x_d^k} \right]^T \in \mathbb{R}^d \tag{1}$$

using only k evaluations of automatic differentiation regardless of the dimension d.

This has applications in:

I. Solving differential equations.
II. Continuous Normalizing Flows.
III. Learning stochastic differential equations.



Full Jacobian of $f(x)$ — $D_{\text{dim}} f(x)$ (diag of Jac)

Reverse-mode Automatic Differentiation

Deep learning software rely on automatic differentiation (AD) to compute gradients. More generally, AD computes vector-Jacobian products.

$$v^T \frac{\partial f(x)}{\partial x} = \sum_{i=1}^{d} v_i \frac{\partial f_i(x)}{\partial x} \tag{2}$$

However, **computing the Jacobian trace (ie. $\sum D_{\text{dim}}$) is as expensive as the full Jacobian.** One evaluation of AD can only compute one row of the Jacobian.

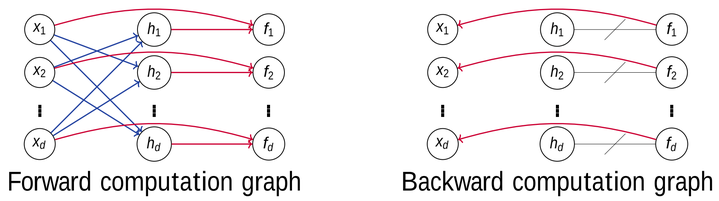
Forward: Network Structure

Our approach:
Dimension-wise derivatives can be efficiently computed for *restricted* architectures with simple *modifications to the AD procedure* in the backward pass.

Conditioner $h_i = c_i(x_{\setminus i})$. The i-th hidden dimension depends on all inputs except the i-th input dimension. Can be computed in parallel using masked neural networks (e.g. MADE, PixelCNN).

Transformer $f_i(x) = \tau_i(x_i; h_i)$. $\tau_i : \mathbb{R}^d \to \mathbb{R}$ outputs the i-th dimension given the concatenated vector. Can be computed in parallel if composed of matrix multiplication and element-wise operations.

Backward: Modified Computation Graph



Forward computation graph — Backward computation graph

Let $\hat{h}$ be `stop_gradient`(h) and $\hat{f} = \tau(x; \hat{h})$, so

$$\frac{\partial \hat{f}_i(x)}{\partial x_j} = \frac{\partial \tau_i(x_i; \hat{h}_i)}{\partial x_j} = \begin{cases} \frac{\partial f_i(x)}{\partial x_i} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{3}$$

Dimension-wise derivatives.

$$\mathbf{1}^T \frac{\partial \hat{f}(x)}{\partial x} = D_{\text{dim}} \hat{f} = \left[\frac{\partial f_1(x)}{\partial x_1} \quad \cdots \quad \frac{\partial f_d(x)}{\partial x_d} \right]^T = D_{\text{dim}} f \tag{4}$$

Higher orders.

$$\mathbf{1}^T \frac{\partial D^{k-1}_{\text{dim}} \hat{f}(x)}{\partial x} = D^k_{\text{dim}} \hat{f}(x) = D^k_{\text{dim}} f(x) \tag{5}$$

Backpropagating through dim-wise derivatives.

$$\frac{\partial D^k_{\text{dim}} \hat{f}}{\partial w} + \frac{\partial D^k_{\text{dim}} \hat{f}}{\partial \hat{h}} \frac{\partial h}{\partial w} = \frac{\partial D^k_{\text{dim}} f}{\partial w} \tag{6}$$

App I: Linear Multistep ODE Solvers

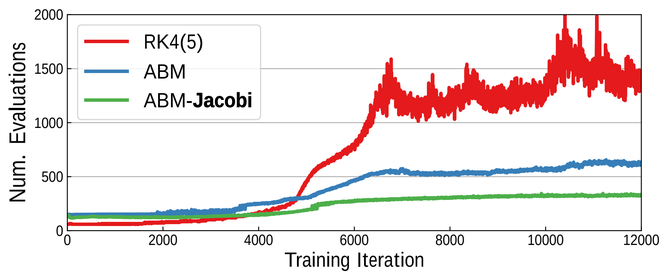
Implicit ODE solvers need to solve an optimization sub-problem in the inner loop.

General idea: replace Newton-Raphson

$$y^{(k+1)} = y^{(k)} - \left[\frac{\partial F(y^{(k)})}{\partial y^{(k)}} \right]^{-1} F(y^{(k)}) \tag{7}$$

with Jacobi-Newton

$$y^{(k+1)} = y^{(k)} - [D_{\text{dim}} F(y)]^{-1} F(y^{(k)}) \tag{8}$$



App II: Continuous Normalizing Flows

If $\frac{dx}{dt} = f(t; x)$, then $\frac{\partial \log p(x)}{\partial t} = -\text{tr}\left(\frac{\partial f}{\partial x}\right)$. (See "Neural ODEs")

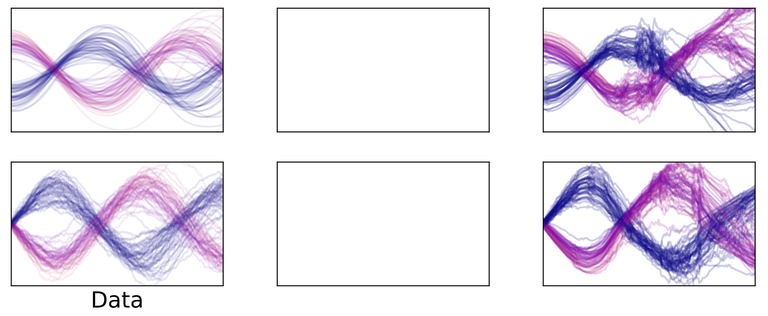
Model	MNIST		Omniglot	
	ELBO ↑	NLL ↓	ELBO ↑	NLL ↓
VAE	-86.55	82.14	-104.28	97.25
Planar	-86.06	81.91	-102.65	96.04
IAF	-84.20	80.79	-102.41	96.08
Sylvester	-83.32	**80.22**	-99.00	**93.77**
FFJORD	-82.82		-98.33	
Ode Op-CNF	**-82.37**	**80.22**	**-97.42**	**93.90**

Table: Evidence lower bound and negative log-likelihood for static MNIST and Omniglot.

Exact trace vs stochastic trace. Converges faster and results in easier to solve dynamical systems.

App III: Stochastic Differential Eqs

$$dx(t) = f(x(t); t)dt + g(x(t); t)dW \tag{9}$$



Data

Idea: match left- and right-hand-side of the Fokker-Planck equation, which describes the change in density.

$$\frac{\partial p(t; x)}{\partial t} = \sum_{i=1}^{d} \frac{\partial}{\partial x_i} [f_i(t; x) p(t; x)] + \frac{1}{2} \sum_{i=1}^{d} \frac{\partial^2}{\partial x_i^2} g_{ii}^2(t; x) p(t; x) \tag{10}$$

References

Germain *et al.* "MADE: Masked autoencoder for distribution estimation." (2015)

Huang *et al.* "Neural Autoregressive Flows." (2018)

Chen *et al.* "Neural ODEs." (2018)