Neural Networks with Cheap Differential Operators

Ricky T. Q. Chen and David Duvenaud

University of Toronto, Vector Institute

Overview

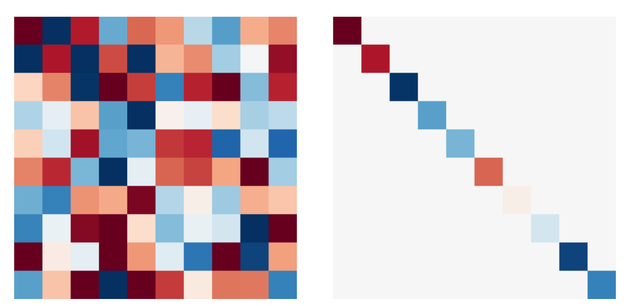
Given $f : \mathbb{R}^d \to \mathbb{R}^d$, we want its *dimension-wise* k-th order derivatives:

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

- Example: divergence operator $\nabla \cdot f := \text{tr}(\mathcal{D}_{\text{dim}} f)$.
- Cost scales with D for general networks (same as full Jacobian), because backprop only gives one row at a time.
- We introduce HollowNets, which let us evaluate dimensionwise derivatives for same cost **regardless of the dimension d**.

Applications in:

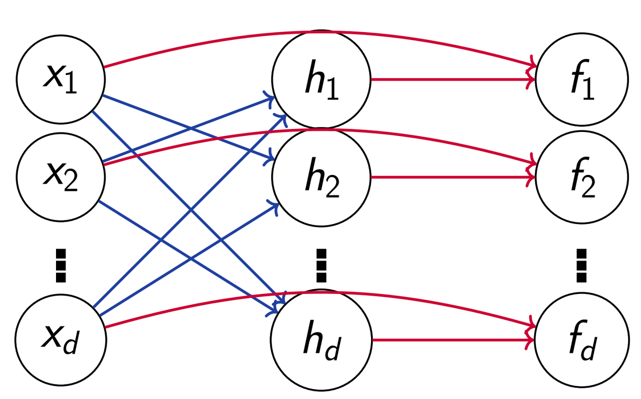
I. Solving fixed-point problems.
II. Continuous-time generative modeling.
III. Partial differential equations.



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

HollowNet Architecture

Dimension-wise derivatives can be efficiently computed for **restricted** architectures:



Forward computation graph

Interactions

$h_i = c_i(x_{-i})$. $c_i : \mathbb{R} \to \mathbb{R}^{d_h}$

The i-th hidden state depends on all inputs except the i-th input dimension.

The conditioner network has a **hollow** *Jacobian.*

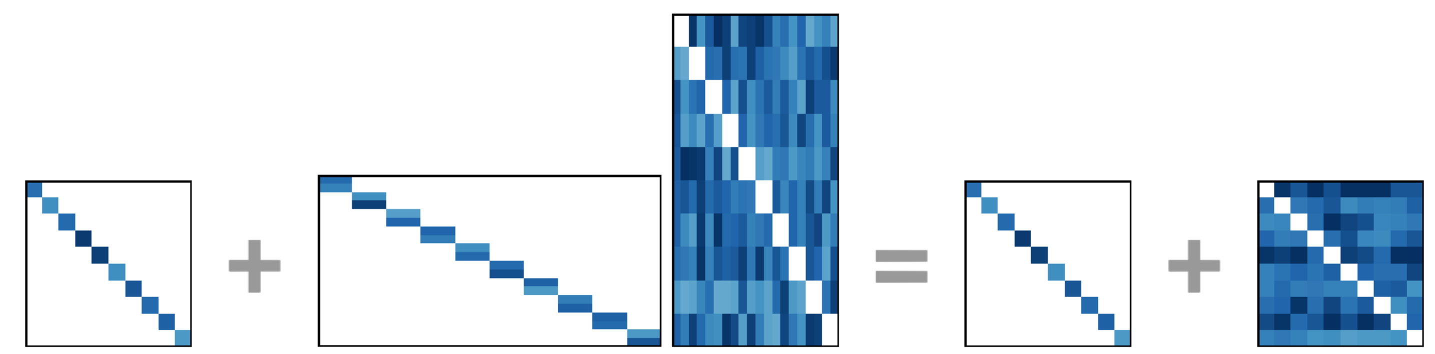
Per-Dim Transform

$f_i(x) = \tau_i(x_i, h_i)$. $\tau_i : \mathbb{R}^{d_h+1} \to \mathbb{R}$ outputs the i-th dimension given the concatenated vector.

The transformer network has **diagonal** *partial Jacobians.*

Modified Backward Computation Graph

Full Jacobian factors into diagonal and hollow matrices:

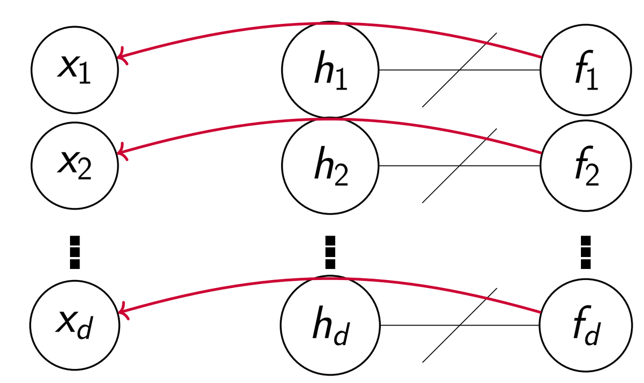


$$\frac{d}{dx} f = \frac{\partial \tau}{\partial x} + \frac{\partial \tau}{\partial h}\frac{\partial h}{\partial x} = \text{(diagonal)} + \text{(hollow)}$$

Can get exact gradients efficiently with modified backwards pass:

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

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



Backward computation graph

App I: Implicit ODE Solvers for Stiff Equations

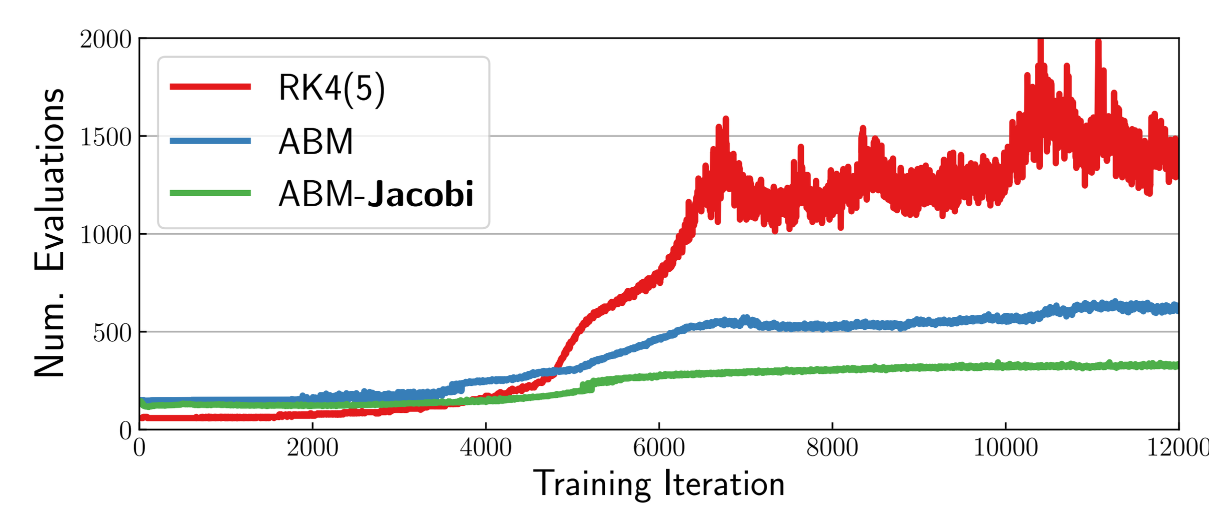
Implicit ODE solvers need to solve an optimization in inner loop.

General idea: replace Newton-Raphson

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

with **Jacobi-Newton:**

$$y^{(k+1)} = y^{(k)} - \left[\mathcal{D}_{\text{dim}} F(y)\right]^{-1} \odot F(y^{(k)})$$



Jacobi-Newton Implicit Solver > Implicit Solver > Explicit Solver

App II: Continuous Normalizing Flows

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

(See *Continuous Normalizing Flows* section in "Neural ODEs")

Table: Evidence lower bound and negative log-likelihood.

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 (Stochastic trace)	-82.82	–	-98.33	–
Hollow-CNF (Ours; exact trace)	**-82.37**	**80.22**	**-97.42**	**93.90**

Exact trace computation converges faster and results in easier to solve dynamical systems than stochastic trace estimation.

App III: Learning Stochastic Differential Equations

$$dx(t) = f(x(t), t)dt + g(x(t), t)dW$$

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

$$\frac{\partial p(t,x)}{\partial t} = -\sum_{i=1}^{d} \frac{\partial}{\partial x_i}\left[f_i(t,x)p(t,x)\right] + \frac{1}{2}\sum_{i=1}^{d}\frac{\partial^2}{\partial x_i^2}\left[g_{ii}^2(t,x)p(t,x)\right]$$

References

Germain *et al.* "MADE: Masked Autoencoder for Distribution Estimation." (2015)

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

Chen *et al.* "Neural Ordinary Differential Equations." (2018)