DISSECTING NEURAL ODES

Massaroli et al.

Léon Migus

SCIENCES SORBONNE UNIVERSITÉ

LIP6

MLIA Machine Learning & Deep Learning for Information Access

Neural ODE

Formulation

Output of forward propagation comes from solving the ODE in equation 1.

$$\begin{aligned} \frac{d\mathbf{z}(t)}{dt} &= f(\mathbf{z}(t), t, \theta) \\ \mathbf{z}(0) &= x \end{aligned} \tag{1}$$

In practice, output of forward propagation comes from a solver at time T, and is shown equation 2.

$$z(T) = \text{ODESolve}(z(t_0), f, t_0, T, \theta) \tag{2}$$

In order to train, we optimize the loss at time the final time T and we use the adjoint method to get the gradients.

Loss function

Loss

Loss is only taken at final time T. But latent space evolves through a continuum of layers. Why not consider a loss distributed on the whole depth domain as in equation 3?

$$\mathrm{I} := L(\mathbf{z}(T)) + \int_S l(\tau, \mathbf{z}(\tau)) d\tau \tag{3}$$

Training is done by modifying equations of the adjoint method.
In practice, is it really better?

Depth-variance

Depth-variance

SCIENCES SORBONNE UNIVERSITÉ

Depth-variance

- In Neural ODE, weight sharing
- But θ should depend on t.
- So new problem is solving equation 4.

$$\begin{aligned} \frac{d\mathbf{z}(t)}{dt} &= f(\mathbf{z}(t), t, \theta(t)) \\ \mathbf{z}(0) &= x \end{aligned} \tag{4}$$

Solutions

SCIENCES SORBONNE UNIVERSITÉ

Depth-variance approaches

- **Gradient descent**

$$\theta_{k+1}(t) = \theta_k(t) - \eta \frac{\partial l_k}{\partial \theta(t)} \tag{5}$$

If $\theta(t)$ is $L2$, then we have the sensitivity with the adjoint equation. In order to implement it, we need to choose a finite dimensional approximation of the solution.

- **Galerkin Neural ODEs**

$$\theta(t) = \sum_{j=1}^{m} \alpha_j \odot \psi_j(t) \tag{6}$$

- **Stacked Neural ODEs**: same as taking θ piece-wise constant.

Augmentation strategies

Augmentation strategies

SCIENCES SORBONNE UNIVERSITÉ

Augmentation strategies

Solve the IVP in a higher dimensional space $(n_x + n_\alpha)$ to simplify the flows and allow the network to learn functions it couldn't have learned.

- **Dupont approach**: 0-augmentation

$$\begin{bmatrix} \mathbf{h}(0) \\ \mathbf{a}(0) \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix} \tag{7}$$

- **Input-layer augmentation**

$$\mathbf{z}(0) = h_x(\mathbf{x}) \tag{8}$$

- **Higher-order Neural ODEs**: we solve a higher order ODE to increase dimension (e.g. second order $n_x = n_\alpha$) with less parameters.

Augmentation strategies

Higher-order Neural ODEs

Higher-order Neural ODEs, with $\mathbf{z}(t) = [\mathbf{z}_q(t), \mathbf{z}_p(t)]$:

$$\begin{aligned} \mathbf{z}_q''(t) &= f(\mathbf{z}(t), t, \theta(t)) \\ &\equiv \\ \mathbf{z}_q'(t) &= \mathbf{z}_p(t),\ \mathbf{z}_p'(t) = f(\mathbf{z}_q(t), \mathbf{z}_p(t), t, \theta(t)) \end{aligned} \tag{9}$$

	NODE		ANODE		IL-NODE		2nd–Ord.	
	MNIST	CIFAR	MNIST	CIFAR	MNIST	CIFAR	MNIST	CIFAR
Test Acc.	96.8	58.9	98.9	70.8	99.1	**73.4**	**99.2**	72.8
NFE	98	93	71	169	44	65	**43**	**59**
Param.[K]	21.4	37.1	20.4	35.0	20.7	36.1	**20.0**	**34.6**

Table 1: Mean test results across 10 runs on MNIST and CIFAR. We report the mean NFE at convergence. Input layer and higher order augmentation improve task performance and preserve low NFEs at convergence.

Figure 1: Augmentation strategies results

Beyond augmentation

Beyond augmentation

Beyond augmentation

- **Data-controlled Neural ODEs**: Learning the ODE with the data in it (e.g. sum of $\mathbf{h}$ and $\mathbf{x}$).

$$\begin{aligned} \frac{d\mathbf{z}(t)}{dt} &= f(\mathbf{z}(t), \mathbf{x}, t, \theta(t)) \\ \mathbf{z}(0) &= h_x(\mathbf{x}) \end{aligned} \tag{10}$$

- **Adaptive-depth Neural ODEs**: avoid problem of crossing trajectories by changing for each sample the final time step, "i.e." the depth.
  Using a hypernetwork g to learn the depth and then using the regular architecture.

Thanks!

SCIENCES SORBONNE UNIVERSITÉ

Thanks for your attention!