Symplectic Adjoint Method for Exact Gradient of Neural ODE with Minimal Memory

Takashi Matsubara*, Yuto Miyatake*, Takaharu Yaguchi†

*Osaka University, †Kobe University

Motivation

Neural ODE [1]

- is a neural network that approximates an ordinary differential equation (ODE) $\dot{x} = f(x)$ via a numerical integration.
- can learn continuous-time dynamics such as physical systems.
- can learn probabilistic distributions by the change-of-variables.

Existing Methods and Their Difficulties:

- The adjoint method obtains a gradient by a numerical integration backward in time without the backpropagation over time; it needs a high computation cost to suppress numerical errors.
- The backpropagation algorithm over an integration consumes an extremely large memory.
- This is true even after a checkpointing scheme divides an integration into several subparts.

Methods	Memory for Checkpoints	Memory for Backprop.	Computation
Adjoint method	M	$M+L$	$N+2\tilde{N}$
Backpropagation	–	$MNsL$	$2N$
Checkpoint per component	M	NsL	$3N$
Checkpoint per step (ACA) [2]	MN	sL	$3N$
Symplectic adjoint method	$MN+s$	L	$4N$

$\times MsL$

M: # of neural ODE components
L: # of layers in a neural network
$N, \tilde{N}$: # of steps in the forward/backward integrations
s: # of function evaluations per step

Proposal: Symplectic Adjoint Method

- Using a symplectic integrator, the adjoint method obtains an exact gradient (up to rounding errors) of a neural ODE.
- A new checkpointing scheme inside a numerical integrator

References

[1] Chen+. (2018). Neural Ordinary Differential Equations. In NeurIPS.
[2] Zhuang+. (2020). Adaptive Checkpoint Adjoint Method for Gradient Estimation in Neural ODE. In ICML.
[3] Sanz-Serna, (2016). Symplectic Runge-Kutta schemes for adjoint equations, automatic differentiation, optimal control, and more. SIAM Review.
[4] Grathwohl+ (2018). FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. In ICLR.
[5] Matsubara+. (2020). Deep Energy-Based Modeling of Discrete-Time Physics. In NeurIPS.

Theory: Adjoint Method

Systems

- Main system $\dot{x}(t) = f(x(t), t), x(0) = x_0$
- Variational system $\dot{\delta}(t) = \frac{\partial f}{\partial x}(x(t), t)\delta(t), \delta(0) = I$, then $\delta(t) = \frac{\partial x(t)}{\partial x_0}$.
- Adjoint system $\dot{\lambda}(t) = -\frac{\partial f}{\partial x}(x(t), t)^T \lambda(t), \lambda(T) = \left(\frac{\partial \mathcal{L}}{\partial x(T)}\right)^\mathsf{T}$
- For parameters θ, $\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x \\ \theta \end{bmatrix} = \begin{bmatrix} f(x(t), t) \\ 0 \end{bmatrix}, \begin{bmatrix} x \\ \theta \end{bmatrix}(0) = \begin{bmatrix} x_0 \\ \theta \end{bmatrix}$

Remarks in Continuous Time

- The bilinear quantity $\lambda^T \delta$ is preserved.
- $\lambda(t)$ represents the gradient $\left(\frac{\partial \mathcal{L}}{\partial x(t)}\right)^\mathsf{T}$ through a backward integration.

Proposal: Symplectic Adjoint Method

Discretized Systems

- The main system is discretized by a Runge-Kutta method

$$x_{n+1} = x_n + h_n \sum_{i=1}^s b_i k_{n,i}$$
$$k_{n,i} \coloneqq f(X_{n,i}, t_n + c_i h_n),\ X_{n,i} \coloneqq x_n + h_n \sum_{j=1}^s a_{i,j} k_{n,j}$$

- The variational system is discretized by the same method.
- The adjoint system is discretized by another Runge-Kutta method

$$\lambda_{n+1} = \lambda_n + h_n \sum_{i=1}^s B_i l_{n,i}$$
$$l_{n,i} \coloneqq -\frac{\partial f(X_{n,i}, t_n + C_i h_n)}{\partial X_{n,i}}^T \Lambda_{n,i},\ \Lambda_{n,i} \coloneqq \lambda_n + h_n \sum_{j=1}^s A_{i,j} l_{n,j}$$

Theorem in Discrete Time [3]

- If $B_i = b_i \neq 0, C_i = c_i,\ b_i A_{i,j} + B_j a_{j,i} - b_i B_j = 0$, $\lambda^T \delta$ is conserved in discrete time, and *the adjoint method obtains an exact gradient.*
- The condition is for a paired of Runge-Kutta methods to be symplectic.
- In general, a symplectic integrator conserves the bilinear quantity $\lambda^T \delta$, associated with the symplectic structure.

Implementation with Nested Checkpointing Scheme

- Forward integration of the state x_n
  1. while retaining each step x_n as a checkpoint and discarding the computation graph.
- Backward integration of the adjoint variable λ_n
  1. Integrate the state x_n forward in time by a step from a checkpoint while retaining each stage $X_{n,i}$ as an internal checkpoint.
  2. Integrate the adjoint variable λ_n backward in time by a step while loading internal checkpoints $X_{n,i}$, recomputing f, and computing $\partial f / \partial X_{n,i}$.

Results

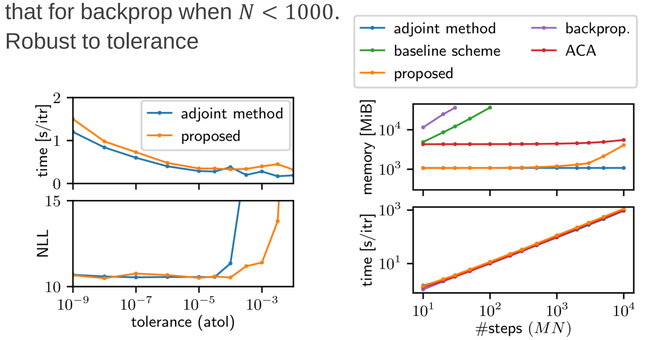
Results on FFJORD [4]

- Consumes the lowest memory in most cases.
- Works relatively fast in many cases.
- More robust to rounding errors.

Method	MINIBOONE NLL	MINIBOONE mem	MINIBOONE Time	GAS NLL	GAS mem	GAS time	POWER NLL	POWER mem	POWER time
adjoint method	10.59	*170*	*0.74*	-10.53	*24*	4.82	**-0.31**	**8.1**	6.33
backpropagation	*10.54*	4436	0.91	-9.53	4479	12	-0.24	1710.9	10.64
baseline scheme	*10.54*	4457	1.1	-9.53	1858	5.48	-0.24	515.2	**4.37**
ACA	10.57	306	**0.77**	*-10.65*	73	**3.98**	**-0.31**	29.5	*5.08*
proposed	**10.49**	**95**	0.84	**-10.89**	**20**	*4.39*	**-0.31**	*9.2*	5.73

Method	HEPMASS NLL	HEPMASS mem	HEPMASS Time	BSDS300 NLL	BSDS300 mem	BSDS300 time	MNIST NLL	MNIST mem	MNIST time
adjoint method	16.49	*40*	4.19	**-152.04**	*577*	11.7	*0.918*	*1086*	10.12
backpropagation	17.03	5254	11.82	-	-	-	-	-	-
baseline scheme	17.03	1102	4.40	-	-	-	-	-	-
ACA	**16.41**	88	**3.67**	*-151.27*	757	**6.97**	0.919	4332	**7.94**
proposed	*16.48*	**35**	*4.15*	-151.17	**283**	*8.07*	**0.917**	**1079**	*9.42*

- Memory consumption for checkpoints is negligible compared to that for backprop when $N < 1000$.
- Robust to tolerance



Results on Physical Systems [5]

- A similar tendency

Method	KdV equation NLL	KdV equation mem	KdV equation Time	Cahn-Hilliard system NLL	Cahn-Hilliard system mem	Cahn-Hilliard system time
adjoint method	1.61	*93.7*	276	5.58	*93.7*	942
backpropagation	1.61	693.9	**105**	4.68	3047.1	**425**
ACA	1.61	647.8	*137*	5.82	648.0	*484*
proposed	1.61	**79.8**	162	5.47	**80.3**	568