Applying Higher-Order Runge-Kutta Methods to Neural Networks

Derek Onken and Lars Ruthotto Department of Mathematics and Computer Science, Emory University

EMORY COLLEGE OF ARTS AND SCIENCES

Objectives

Broader Goals: Model training of deep neural networks (DNNs) as optimal control problem.

1. simplify design of DNNs
   ($\approx$ discretize a PDE)
2. analyze stablity and generalization
   ($\approx$ vanishing/exploding gradients)
3. develop variational framework
   ($\rightsquigarrow$ multilevel and multiscale learning)
4. design reversible dynamics
   ($\rightsquigarrow$ memory-free learning)

Current focus:

1. **research:** model order reduction, efficient optimization, stable dynamics, time-integrators [1]
2. **community:** free MATLAB/Julia software
3. **accessibility:** building models in pyTorch

DNNs meet Optimal Control

Goal: Find a function $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ and its parameter $\theta \in \mathbb{R}^p$ such that $f(\mathbf{y}_k, \theta) \approx \mathbf{c}_k$ for training data $\mathbf{y}_1, \dots, \mathbf{y}_s \in \mathbb{R}^n$ and labels $\mathbf{c}_1, \dots, \mathbf{c}_s \in \mathbb{R}^m$.

Model $\mathbf{y}_k^N = f(\mathbf{y}_k, \theta)$ as output of Residual Neural Network (ResNN) with N layers. Let $\mathbf{y}_k^0 = \mathbf{y}_k$ and

$$\mathbf{y}_k^{i+1} = \mathbf{y}_k^i + hg(\mathbf{y}_k^i, \theta^i), \quad \forall i = 0, \dots, N-1.$$

(g transforms features, e.g., $g(\mathbf{y}, \theta) = \tanh(\mathbf{K}(\theta)\mathbf{y})$)

Note that ResNN is a forward Euler discretization [2] of the initial value problem ($t \in [0, T]$)

$$\partial_t y_k(t, \theta) = g(y_k(t, \theta), \theta(t)), \quad y_k(0, \theta) = \mathbf{y}_k$$

Learning: Find θ and weights of classifier by solving

$$\min_{\theta, \mathbf{W}} \frac{1}{s} \sum_{k=1}^{s} \text{loss}(y_k(T, \theta)\mathbf{W}, \mathbf{c}_k) + \text{regularizer}(\theta, \mathbf{W}).$$

learning $\approx$ mass transport, trajectory planning

References

[1] Chen et al. *Neural Ordinary Differential Equations.*. NeurIPS, 2018.

[2] E Haber, L Ruthotto *Stable Architectures for Deep Neural Networks.* Inverse Problems, 2017.

[3] L Ruthotto, E Haber *Deep Neural Networks Motivated by Partial Differential Equations.* arXiv, 2018.

Motivation

Since the community recognizes the effectiveness of Resnets and their skip connections (shown to be equivalent to Forward Euler), wouldn't higher-order Runge-Kutta schemes assist in training?

Runge-Kutta Schemes

Goal: Improve training by maintaining few parameters and controlling conditioning

Recall the Fourth-Order Runge-Kutta
Defining the length of the j-th time interval by

$$h_j = t_{j+1} - t_j,$$

the update scheme reads

$$\mathbf{u}_{j+1} = \mathbf{u}_j + \frac{h_j}{6}\left(f(\boldsymbol{\theta}(t_j), \mathbf{z}_1) + 2f(\boldsymbol{\theta}(t_{j+1/2}), \mathbf{z}_2) \right.$$
$$\left. + 2f(\boldsymbol{\theta}(t_{j+1/2}), \mathbf{z}_3) + f(\boldsymbol{\theta}_{j+1}, \mathbf{z}_4)\right)$$

where f is the primary layer in the dynamic unit as a function of the controls $\boldsymbol{\theta}(t_k)$ and intermediate states $\mathbf{z}_i$ that are computed as follows

$$\mathbf{z}_1 = \mathbf{u}_j$$
$$\mathbf{z}_2 = \mathbf{u}_j + \frac{h_j}{2} f(\boldsymbol{\theta}(t_j), \mathbf{u}_j)$$
$$\mathbf{z}_3 = \mathbf{u}_j + \frac{h_j}{2} f(\boldsymbol{\theta}(t_{j+1/2}), \mathbf{z}_1)$$
$$\mathbf{z}_4 = \mathbf{u}_j + h_j f(\boldsymbol{\theta}(t_{j+1/2}), \mathbf{z}_2)$$

From this RK4 scheme for f, we build a dynamic unit as part of a simple model to compare different time-steppings for when f is a layer of type:

Double / ResNN: $\sigma_2 \circ \mathcal{N}_2 \circ K_{\theta_2} \circ \sigma_1 \circ \mathcal{N}_1 \circ K_{\theta_1}(Y)$

Preactivated Double: $\mathcal{N}_2 \circ K_{\theta_2} \circ \sigma_2 \circ \mathcal{N}_1 \circ K_{\theta_1} \circ \sigma_1(Y)$

Double Sym / Parabolic [3]: $-K_\theta^\top \circ \sigma \circ \mathcal{N} \circ K_\theta(Y)$

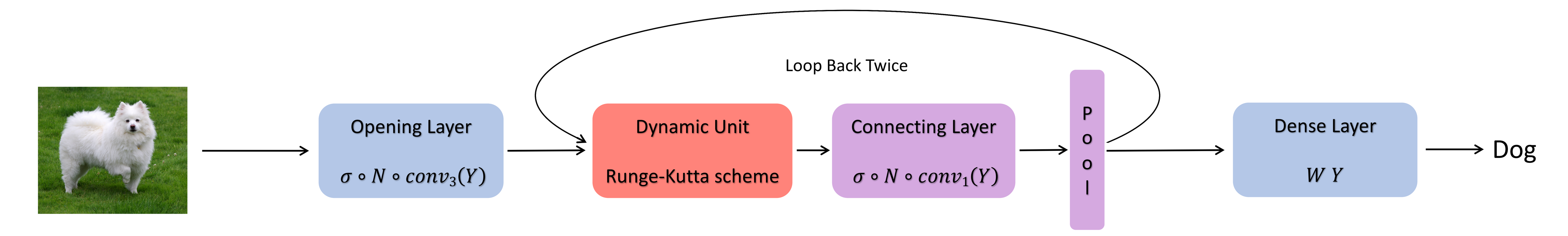
for activation functions σ, normalizations $\mathcal{N}$, and convolution operators K defined by weights θ

Software

Github:

- Meganet.m: academic and teaching tool
- Meganet.jl: high-performance distributed computing
- PyTorch implementations in the works

Model

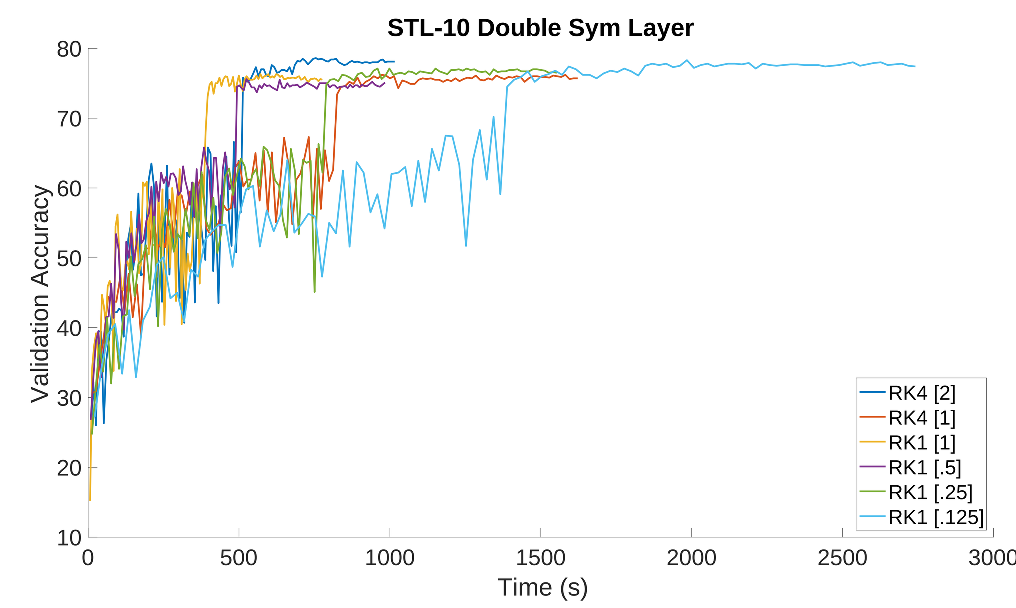


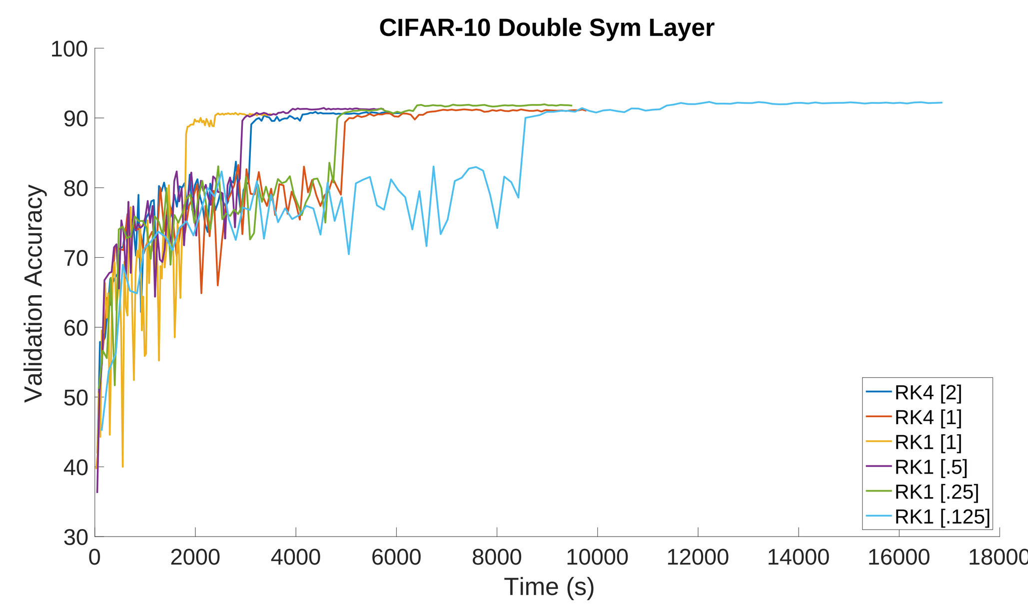
Numerical Results

We train a simple model of a convolutional opening layer, three blocks containing the RK scheme doubling the channels each pass, and one fully connected layer.

The dynamic unit is the only portion that we vary.

Our learning strategy uses 120 epochs of SGD with momentum with initial learning rate of 0.1 which reduces by a factor of 10 after epochs 60, 80, 100.





Team

- Eldad Haber (UBC, Vancouver)
- Eran Treister (Ben Gurion, Israel)
- Simion Novikov (Ben Gurion, Israel)

Funding

Supported by the National Science Foundation awards DMS 1522599 and CAREER DMS 1751636 and by NVIDIA Corporation.

Noisy Stochastic Shifts

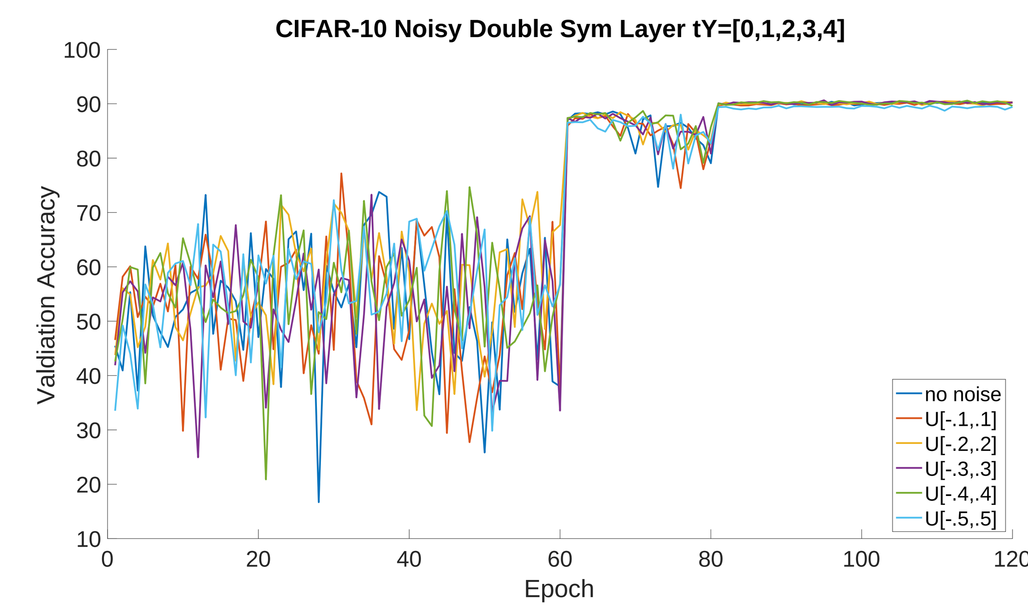
Goal: Analyze the network when the time-stepping is varied every epoch

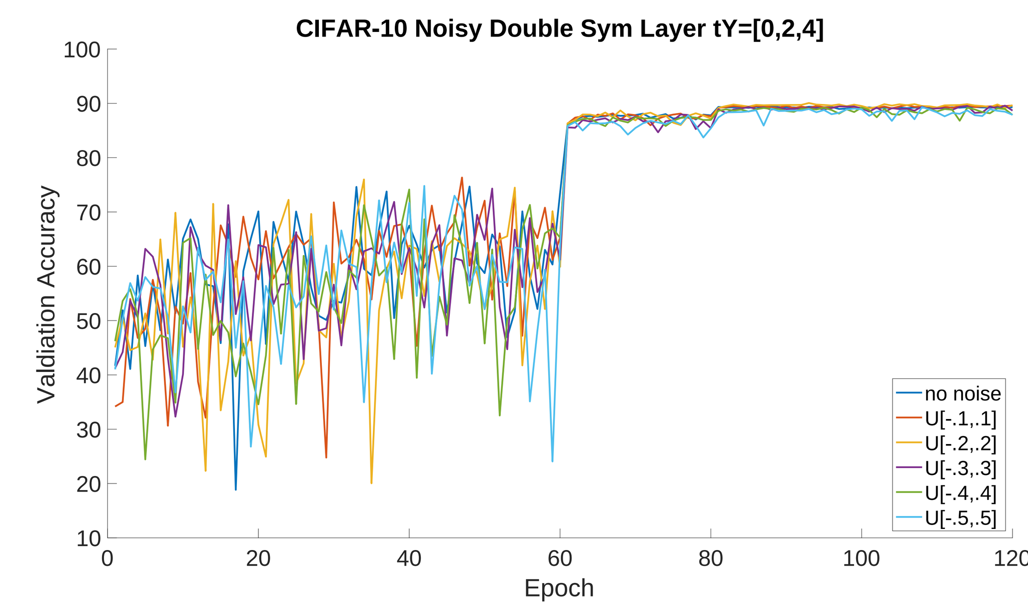
Fixing the control time steps $t_{\boldsymbol{\theta}} = [0, 1, 2, 3, 4]$ and state time steps $t_{\mathbf{Y}} = [0, 1, 2, 3, 4]$ or $[0, 2, 4]$, at every epoch, draw noise $\boldsymbol{\epsilon}$ from a uniform distribution.

This varies the interpolation of the control weights to obtain different state weights.

Results:

For a Double Sym Layer in the dynamic unit





Future Directions

- Loss Landscape Analysis
- Adversarial Vulnerability Analysis
- Adaptive Time-Stepping
- Adams-Bashforth Methods