Dynamical System Inspired Adaptive Time Stepping Controller for Residual Network Families

Yibo Yang,[1,2,*] **Jianlong Wu,**[2,3,*] **Hongyang Li,**[2] **Xia Li,**[2,4] **Tiancheng Shen,**[1,2] **Zhouchen Lin**[2,5,†]
[1]Center for Data Science, Academy for Advanced Interdisciplinary Studies, Peking University
[2]Key Laboratory of Machine Perception (MOE), School of EECS, Peking University
[3]School of Computer Science and Technology, Shandong University
[4]Key Laboratory of Machine Perception, Shenzhen Graduate School, Peking University
[5]Samsung Research China – Beijing (SRC-B)
{ibo, jlwu1992, lhy_ustb, ethanlee, tianchengshen, zlin}@pku.edu.cn

Abstract

The correspondence between residual networks and dynamical systems motivates researchers to unravel the physics of ResNets with well-developed tools in numeral methods of ODE systems. The Runge-Kutta-Fehlberg method is an adaptive time stepping that renders a good trade-off between the stability and efficiency. Can we also have an adaptive time stepping for ResNets to ensure both stability and performance? In this study, we analyze the effects of time stepping on the Euler method and ResNets. We establish a stability condition for ResNets with step sizes and weight parameters, and point out the effects of step sizes on the stability and performance. Inspired by our analyses, we develop an adaptive time stepping controller that is dependent on the parameters of the current step, and aware of previous steps. The controller is jointly optimized with the network training so that variable step sizes and evolution time can be adaptively adjusted. We conduct experiments on ImageNet and CIFAR to demonstrate the effectiveness. It is shown that our proposed method is able to improve both stability and accuracy without introducing additional overhead in inference phase.

Introduction

Currently, the structure of neural network is mainly developed by hand-crafted design (Simonyan and Zisserman 2015; Szegedy et al. 2015) or neural architecture searching (Baker et al. 2017). A theoretical guidance is still lacking for understanding deep network behaviors. One of the most successful architectures, residual network (ResNet) (He et al. 2016), introduces identity mappings to enable training a very deep model. ResNets are also used as the base model for a series of computer vision tasks such as scene segmentation (Chen et al. 2017), and action recognition (Tran et al. 2018). Despite the huge success, the understanding of ResNets is mainly supported by empirical analyses and experimental evidences, other than some attempts from an optimization view (Li et al. 2018). Recently, the connection between ResNet and dynamical system has inspired researchers to unravel the physics of residual networks using the rich theories and techniques in differential equations (E 2017; Haber and Ruthotto 2017).

In (E 2017), it is noted that the Euler method for ordinary differential equations (ODEs) has the same formulation as ResNet iterative updates, and ResNet is viewed as a discrete dynamical system. In this way, the parameter learning in neural networks is translated into its continuous counterpart as an optimal control problem (Chen et al. 2018; Li and Hao 2018; Li et al. 2017; Behrmann, Duvenaud, and Jacobsen 2019). Based on the Euler method, some studies introduce multi-step or higher-oder discretization (Lu et al. 2018; He et al. 2019), and fractional optimal control (Jia et al. 2019) to construct more powerful network structures for different tasks. Other studies analyze the stability of residual networks and propose more stable and robust structures (Haber and Ruthotto 2017; Chang et al. 2018a; Ruthotto and Haber 2018; Haber et al. 2019).

An appropriate time stepping for discretization methods of ODEs is crucial for the stability and efficiency (Ascher and Petzold 1998). A small step size is able to render an accurate solution, but requires more steps for a fixed evolution time. Chang *et al.* adopt a multi-level strategy to adjust the time step size for ResNet training (Chang et al. 2018b). In a recent study, a small step size is suggested for more stable and robust ResNets (Zhang et al. 2019). However, an overly small step size would smooth the feature learning. From a dynamical system view, the evolution time for network with a fixed depth and a small step size is too short for the system to evolve from the initial state to the final linearly separable state. In numerical methods for ODEs, adaptive time stepping strategies, such as the Runge-Kutta-Fehlberg (RKF) method (Hairer, Nørsett, and Wanner 1991) as shown in Figure 1, are able to attain a good trade-off between stability and cost. Can we also design an adaptive time stepping for ResNets to ensure both stability and performance?

In this study, we analyze the effects of time stepping on the stability and performance of residual networks, and point out that each step size should be aware of previous steps and the weight parameters in the current step. We develop an adaptive controller, which connects different steps as an LSTM, and takes the parameters of each step as input, to output a set of coefficients that decide the current step size.

[*]Equal Contribution
[†]Corresponding Author

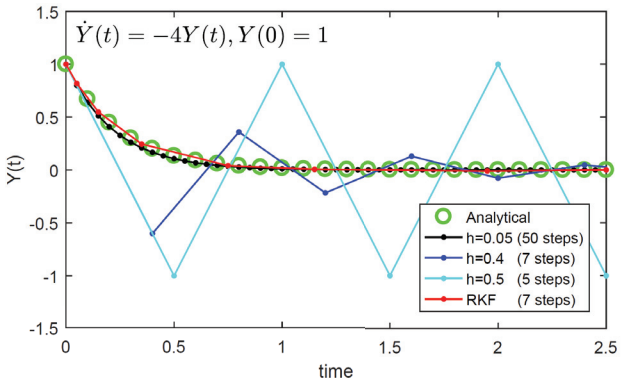

Figure 1: An initial value problem with different discretization step sizes h. The RKF method is able to attain a good trade-off between stability and efficiency, offering a stable solution, while using a small number of steps.

In doing so, the network is trained with variable step sizes and evolution time, so that our time stepping is optimized jointly to render the network a better stability and performance. More importantly, because the controller is data-independent, our performance gains come with no additional cost in inference phase.

The contributions of this study can be listed as follows:

- We analyze the correspondence between ODEs and ResNets, establish a stability condition for ResNets with step sizes and weight parameters, and point out the effects of step sizes on the stability and performance.
- Based on our analyses, we develop a self-adaptive time stepping controller to enable optimizing variable step sizes and evolution time jointly with the network training.
- Experiments on ImageNet and CIFAR demonstrate that our method is able to improve both stability and accuracy. The improvements come with no additional cost in inference phase. We also test the application of our method to two non-residual network structures.

View ResNet as a Dynamical System

The forward propagation in a residual network block (He et al. 2016) can be written as:

$$\mathbf{y}_{j+1} = \mathbf{y}_j + h\mathcal{F}(\mathbf{y}_j, \mathbf{W}_j), \quad j = 0, 1, ..., D-1, \quad (1)$$

where $\mathcal{F}$ is the residual function for each step, and D is the network depth. Here we add the $h > 0$ in a multiplicative way with the residual branch. Usually a unit of $\mathcal{F}$ has the form of $\mathcal{F} = \sigma(BN(\mathbf{W}_j\mathbf{y}_j))$, where σ is the non-linear activation. When $h = 1$, it reduces to the original form of ResNets. Regarding h as a fixed step size, we see that Eq. (1) can be interpreted as the forward Euler method discretization for the following initial value problem (IVP) (E 2017; Haber and Ruthotto 2017):

$$\dot{\mathbf{y}}(t) = \mathcal{F}(\mathbf{y}(t), \mathbf{W}(t)), \quad \mathbf{y}(0) = \mathbf{y}_0, \quad (2)$$

where features $\mathbf{y}(t)$ and parameters $\mathbf{W}(t)$ are viewed in their continuous limit as a function of time $t \in [0, T]$. The evolution time T corresponds to the network depth D. In doing so, residual networks are interpreted as the discrete counterpart of dynamical systems, and parameter learning is equivalent to solving an optimal control problem with respect to the ODE system in Eq. (2) (Chen et al. 2018; Li and Hao 2018; Haber et al. 2018). Related studies use the stability condition of the forward Euler method to analyze the stability of ResNets and propose better structures (Chang et al. 2018a; Haber et al. 2019). We show that time stepping is crucial for the stability and performance of ResNets.

Time Stepping for the Euler method

Given a linear problem $\dot{y}(t) = \lambda y(t), t \in [0, T]$, we have its forward Euler's discretization as $y_{j+1} = y_j + h\lambda y_j, j = 0, ..., N-1$, where h is the fixed step size and $T = Nh$. Assuming that $y_0^\epsilon = y_0 + \epsilon$ is the initial value suffered from a perturbation $\epsilon > 0$, we have:

$$|y_N^\epsilon - y_N| = |1 + h\lambda|^N \epsilon, \quad (3)$$

which indicates that when $N \to \infty$, the perturbation is controllable if $|1+h\lambda| \leq 1$. As a more general case, the forward Euler method for non-linear system Eq. (2) is stable when the following condition holds (Ascher and Petzold 1998):

$$\max |1 + h\lambda_i(\mathbf{J}(t))| \leq 1, \quad (4)$$

where λ_i denotes the i-th eigenvalue of the Jacobian matrix defined as $\mathbf{J}(t) = \nabla_{\mathbf{y}}\mathcal{F}(\mathbf{y}(t), \mathbf{W}(t))$. From the stability condition, we can see that a small step size h is required to obtain a stable solution. In practical implementations, the step size h should satisfy a stable solution, while being as large as possible to reduce the amount of iterative steps for a fixed evolution time T. Thus, the choice for a time stepping scheme is crucial for both stability and efficiency.

The Runge-Kutta-Fehlberg (RKF) method (Hairer, Nørsett, and Wanner 1991) as an adaptive time stepping is able to attain a good trade-off between the stability and efficiency. It uses the p-th (usually p=4) order Runge-Kutta method to compute the current solution y_{j+1}, and the local truncation error is:

$$|y_{j+1} - y(t_{j+1})| = O(\Delta t_j^{p+1}), \quad (5)$$

where Δt_j is the current step size. The $y(t_{j+1})$ is approximated by the $p+1$-th order form, denoted as $\hat{y}_{j+1}$. Then the new step size can be adjusted as:

$$\Delta t_{j+1} = k \times \Delta t_j \times \left(\frac{Tol}{|y_{j+1} - \hat{y}_{j+1}|}\right)^{1/(p+1)}, \quad (6)$$

where k is a factor and Tol is a tolerance error. The method adaptively increases or reduces the next step size according to the agreement between y_{j+1} and $\hat{y}_{j+1}$.

As a simple example, we consider the problem , $\dot{y}(t) = -4y(t), y(0) = 1$, whose analytical solution can be easily derived as $y(t) = \exp(-4t)$. As shown in Figure 1, the adaptive method RKF is able to offer a stable solution, but requires significantly less number of steps than a small step size for the evolution period. If we use a large step size to reduce the number of steps, the solution will be unstable. Thus, an adaptive time stepping scheme is crucial for the stability and efficiency of solution to ODE systems.

Time Stepping for ResNets

As a discrete counterpart of dynamical systems, ResNet has similar behaviors to the discretization method for ODEs. We show that time stepping causes similar effects on the stability and performance of ResNets.

Proposition 1 *Consider a ResNet with D residual blocks and variable step sizes Δt_j for each step. Let ϵ be the perturbation coming from noise or adversary and satisfies $||\mathbf{y}_0^\epsilon - \mathbf{y}_0|| = \epsilon$. We have:*

$$||\mathbf{y}_D^\epsilon - \mathbf{y}_D|| \leq \epsilon \cdot \prod_{j=0}^{D-1} (1 + ||\mathbf{W}_j||_2 \Delta t_j), \tag{7}$$

where $||\mathbf{W}_j||_2$ denotes the spectral norm of weight matrix in each residual block.

Proof 1 *See Appendix A for its proof.* ■

We note that the robustness of ResNets to perturbation is affected by the network depth, spectral norm of each weight matrix, and each step size. Eq. (7) shows that the stability of ResNets is conditioned on each layer in a stacking way, which is consistent with previous findings (Veit, Wilber, and Belongie 2016). For each layer, it is suggested that the weight matrix should have a small spectral norm. Since $||\mathbf{W}_j||_2^2 \leq ||\mathbf{W}_j||_F^2$, weight decay that regularizes the Frobenius norm is effective to train models with robustness to input perturbations. However, it shrinks the weight matrix in all directions, and discards information of input features. Some studies propose spectral norm regularizer (Yoshida and Miyato 2017; Miyato et al. 2018) or Jacobian regularizer (Sokolić et al. 2017) to improve the stability.

The term in Eq. (7) has a similar form to Eq. (4), and also indicates that a small step size Δt_j should be chosen for ResNets' stability. Nevertheless, as pointed out by (Zhang et al. 2019), an overly small step size would smooth the feature learning. Denoting loss function as L, in ResNets with variable step sizes Δt, we have the gradient backpropagated to layer $\mathbf{y}_n$ as:

$$\frac{\partial L}{\partial \mathbf{y}_n} = \frac{\partial L}{\partial \mathbf{y}_D} \left[\mathbf{1} + \frac{\partial}{\partial \mathbf{y}_n} \sum_{i=n}^{D-1} \mathcal{F}(\mathbf{y}_i, \mathbf{W}_i) \Delta t_i \right], \tag{8}$$

which shows that the backpropagated information for each layer comes from two terms. When the step size Δt_i is too small, the second term would vanish, and gradients for each layer would be the same as $\frac{\partial L}{\partial \mathbf{y}_D}$. This would make the network inefficient and lacking in representation power. From the dynamical system perspective, if the network with a fixed depth has a small step size, its corresponding optimal control problem would have a short period of evolution time, which increases the difficulty of transforming the data space from the initial state to the expected linearly separable state.

Therefore, similar to the discretization for ODE systems, ResNets also need an adaptive time stepping to enable a good trade-off between the stability and performance. A related study (Zhang and Laura 2018) proposes to optimize step sizes Δt_j as explicit parameters. We note that independent step sizes are not self-aware and cannot be adjusted adaptively. Inspired by our analyses, we propose a self-adaptive time stepping controller that is dependent on the weight matrices and aware of previous steps.

Proposed Methods

In this section, we first introduce our design in the optimal control view, and then describe the components of our self-adaptive time stepping controller. Finally, we analyze the complexity of our method and show implementation details.

The Optimal Control View

In our analyses, we note that the product of step size and spectral norm of weight matrix in each layer decides the stability. Directly calculating the spectral norm requires the SVD decomposition, which makes the training inefficient. Here we introduce a controller that outputs the current step size Δt_j dependent on the convolution parameters $\mathbf{w}_j$ in this layer. Besides, similar to the design of RKF method, the new step size should remember previous step sizes to avoid sharp increment or reduction. In line with these views, denoting the controller as Θ parameterized by $\{\theta_d\}_{d=1}^{D-1}$, we have the corresponding optimal control problem as:

$$\begin{aligned} \min_{\mathbf{w},\theta} \quad & J = \frac{1}{S} \sum_{s=1}^{S} \Phi(\mathbf{y}_s(T), \mathbf{y}_s^*) + \sum_{d=1}^{D-1} R(\mathbf{w}(t_d), \theta(t_d)), \\ s.t. \quad & \mathbf{y}_s(t_{d+1}) = \mathbf{y}_s(t_d) + \mathcal{F}\left(\mathbf{y}_s(t_d), \mathbf{w}(t_d)\right) \Delta t_d \\ & \Delta t_d = \Theta\left(\mathbf{w}(t_d); \Delta t_1, ..., \Delta t_{d-1}; \theta(t_d)\right) \\ & t_0 = 0,\ t_{d+1} = t_d + \Delta t_d,\ \mathbf{y}_s(0) = \mathbf{y}_s \\ & T = t_D = \sum_{d=0}^{D-1} \Delta t_d,\ d = 0, 1, ..., D-1 \end{aligned} \tag{9}$$

where Φ is the loss function, R is the regularization, $\mathbf{y}_s^*$ is the label of input image $\mathbf{y}_s$, and S is number of samples. The optimal control problem in discrete time (Kwakernaak and Sivan 1972) looks for the best control parameters $\{\mathbf{w}, \theta\}$ for this dynamical system that aims to minimize the cost J. From Eq. (9) we can see that, the system has variable step sizes Δt_j and evolution time T. In implementations, the controller is jointly optimized with the network training, so that an optimal time stepping can be searched to render the network better stability and performance.

Self-adaptive Time Stepping Controller

Since the time stepping controller takes the convolution parameters as input and is aware of previous steps, we parametrize the controller as an LSTM that connects different steps. In implementations, we split the step size as a vector $\Delta \mathbf{t}$ with the same channel number as the feature in current step. The product between residual branch and step size is replaced with a channel-wise multiplication. We find this helps to improve the training stability and accuracy.

An illustration of our method is shown in Figure 2. Denote the convolution parameters of the d-th layer as $\mathbf{w}_d \in \mathbb{R}^{k_1 \times k_2 \times C_1 \times C_2}$, where k_1, k_2 are the kernel sizes, and C_1, C_2 are the number of channels for the input and output, respectively. In order to acquire representative information of

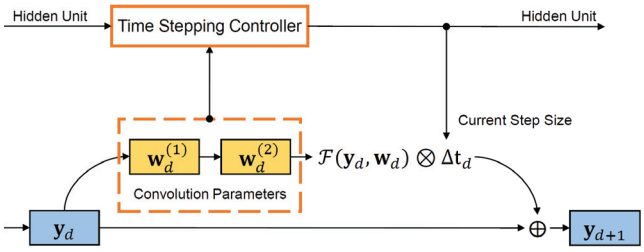

Figure 2: An illustration of our proposed time stepping controller, which is dependent on the convolution parameters, and connects different steps to be aware of previous steps.

the residual function, we average $\mathbf{w}_d$ by projecting along the input dimension and get $\bar{\mathbf{w}}_d \in \mathbb{R}^{k_1 k_2 C_2}$ after reshaping. We concatenate them if there are multiple convolution layers in the residual branch. The $\bar{\mathbf{w}}_d$ first goes through a transformation layer to reduce the dimension into a reduction of r of the channels in the current layer:

$$\mathbf{x}_d = g_{in}(W_{in} \cdot \bar{\mathbf{w}}_d + b_{in}) \in \mathbb{R}^{\frac{C_2}{r}}, \tag{10}$$

where g_{in} refers to the ReLU function, W_{in} is the transformation matrix, b_{in} is the bias vector, and $\mathbf{x}_d$ is the input for LSTM. The hidden unit of LSTM has the same dimension as $\mathbf{x}_d$, *i.e.* $\mathbf{h}^{l-1} \in \mathbb{R}^{\frac{C_2}{r}}$. The interaction between inner states and gates at d-th layer goes through the following steps:

$$\begin{aligned}
\mathbf{i}_d &= \sigma\left(W_i \cdot [\mathbf{h}_{d-1}, \mathbf{x}_d] + b_i\right), \\
\mathbf{f}_d &= \sigma\left(W_f \cdot [\mathbf{h}_{d-1}, \mathbf{x}_d] + b_f\right), \\
\mathbf{g}_d &= \tanh\left(W_g \cdot [\mathbf{h}_{d-1}, \mathbf{x}_d] + b_g\right), \\
\mathbf{o}_d &= \sigma\left(W_o \cdot \left[\mathbf{h}^{d-1}, \mathbf{x}_d\right] + b_o\right), \\
\mathbf{c}_d &= \mathbf{f}_d \odot \mathbf{c}_{d-1} + \mathbf{i}_d \odot \mathbf{g}_d, \\
\mathbf{h}_d &= \mathbf{o}_d \odot \tanh\left(\mathbf{c}_d\right),
\end{aligned} \tag{11}$$

where σ refers to the sigmoid function and $\odot$ denotes element-wise multiplication. After the above processes, another fully connected layer transforms the hidden unit $\mathbf{h}_d$ into the step size vector $\Delta \mathbf{t}_d$:

$$\Delta \mathbf{t}_d = g_{out}(W_{out} \cdot \mathbf{h}_d + b_{out}), \tag{12}$$

where $W_{out} \in \mathbb{R}^{C_2 \times \frac{C_2}{r}}$, $b_{out} \in \mathbb{R}^{C_2}$, and g_{out} denotes the non-linear sigmoid function that restricts the elements in the current step size between range $(0, 1)$. The forward propagation in the current step is:

$$\mathbf{y}_{d+1} = \mathbf{y}_d + \mathcal{F}(\mathbf{y}_d, \mathbf{w}_d) \otimes \Delta \mathbf{t}_d, \tag{13}$$

where $\otimes$ represents the channel-wise multiplication. We have one controller for each size stage in ResNet, and all parameters above keep shared in the same size stage.

We note that the channel-wise attention technique (Hu, Shen, and Sun 2018) has a similar formulation to Eq. (13). Our method differs from theirs in that our controller is data independent and does not rely on the feature space. What our time stepping aims to optimize is part of the structural information. When training finishes, our method discards the controller and has no considerable addition cost in inference phase (except a little calculation of multiplying step sizes). This cannot be realized by attention methods that are feature dependent. Besides, our experiments show that our method is compatible with the attention method.

Table 1: Parameter complexity introduced by the three methods in training and inference phases. k_1 and k_2 denote the kernel sizes of the convolution parameters.

Methods	Training	Inference
TSC_{indp}	$\sum_{b=1}^{B} L_b C_b$	$\sum_{b=1}^{B} L_b C_b$
TSC_{2fc}	$\sum_{b=1}^{B} L_b \frac{C_b^2}{r}(1 + k_1 k_2)$	$\sum_{b=1}^{B} L_b C_b$
TSC_{LSTM}	$\sum_{b=1}^{B} \frac{C_b^2}{r}(1 + \frac{8}{r} + k_1 k_2)$	$\sum_{b=1}^{B} L_b C_b$

Complexity Analysis

We denote the **t**ime **s**tepping **c**ontroller using LSTM as TSC_{LSTM}. In addition to this structure, we also consider two other versions as its counterparts. We analyze their complexities in this subsection and compare their performance in experiments. The first one removes the LSTM layers Eq. (11), and only keeps the input and output transformation layers but does not share their parameters. It is similar to the two fully-connected layers module in (Hu, Shen, and Sun 2018). We denote this version as TSC_{2fc}. This structure is dependent on the convolution parameters but not aware of previous steps. The other one does not use a controller, and only introduces the step sizes $\{\Delta \mathbf{t}_d\}_{d=0}^{D-1}$ as explicit parameters that are independent of weight matrices and previous steps. This version is denoted as TSC_{indp}.

When training finishes, only the optimized step sizes should be stored and the controller can be removed, which leads to little additional cost in inference phase. As for training complexity, TSC_{indp} has the same cost since it does not have a controller. As for TSC_{LSTM} and TSC_{2fc}, TSC_{LSTM} consumes less parameters because of the weight-sharing in LSTM, while TSC_{2fc} consumes less computation because it does not have the LSTM layers in Eq. (11). Assuming that the network has B feature size stages (as an example, $B = 3$ for CIFAR and 4 for ImageNet). There are L_b layers in the b-th stage, and C_b is the number of channels per layer. We compare the three methods' parameter complexity of training and inference phase in Table 1. We will compare their performance and overhead in experiments.

Implementations

In experimental section, we test our proposed methods on the ResNet and its variants. Here we show the details of our implementation. For ResNets structures without bottleneck, there are two convolution layers in each residual block, and we have $k_1 = k_2 = 3$ for the kernel sizes. We concatenate these two parts of projected parameters in layer d by $\bar{\mathbf{w}}_d = [\bar{\mathbf{w}}_d^{(1)}, \bar{\mathbf{w}}_d^{(2)}] \in \mathbb{R}^{2k_1 k_2 C_2}$, where C_2 denotes the output channels in this layer. For ResNets structures with bottleneck, there are three convolution layers in each residual block, and the kernel size is 1 for the first and third layers, and 3 for the bottleneck layer. We only concatenate the

Table 2: Comparison between different controllers on ImageNet. They are all based on ResNet-50 structure. Our methods do not have much additional cost in inference phase.

Methods	top-1 (err.)	Params(M) *(train / infer)*	GFLOPs *(train / infer)*
ResNet (reported)	24.70	25.56	3.86
ResNet (ours)	24.42	25.56	3.86
TSC$_{indp}$-ResNet	24.12	25.57/25.57	3.86/3.86
TSC$_{2fc}$-ResNet	23.90	31.58/25.57	3.88/3.86
TSC$_{LSTM}$-ResNet	**23.63**	27.83/25.57	3.89/3.86

first and third projected parameters by $\bar{\mathbf{w}}_d = [\bar{\mathbf{w}}_d^{(1)}, \bar{\mathbf{w}}_d^{(3)}] \in \mathbb{R}^{C_2+C_2/4}$, where $C_2/4$ is the number of bottleneck channels. We set the reduction r to 4 for ResNets without bottleneck, and 8 for ResNets with bottleneck. An illustration of our method on ResNet-34 (without bottleneck) and ResNet-50 (with bottleneck) is shown in Appendix B.

Experiments

We conduct experiments on CIFAR-10, CIFAR-100, and ImageNet to validate our time stepping controller on ResNet and its variants. We also test the application of our method to two non-residual network structures.

Datasets and Training Details

Datasets For training sets of the ImageNet dataset, we adopt the standard data augmentation scheme (He et al. 2016). A 224×224 crop is randomly sampled from the image or its horizontal flip. The input images are normalized by mean and standard deviation for each channel. All models are trained on the training set and we report the single center crop error rate on the validation set. For CIFAR-10 and CIFAR-100, we adopt a standard data augmentation scheme by padding the images 4 pixels filled with 0 on each side and then randomly sampling a 32×32 crop from each image or its horizontal flip. The images are normalized by mean and standard deviation.

Training Details We train our models using stochastic gradient descent (SGD) with the Nesterov momentum 0.9 and weight decay 10^{-4}. The parameters are initialized following (He et al. 2015). For the ImageNet dataset, we train for 100 epochs with an initial learning rate of 0.1, and drop the learning rate every 30 epochs. A mini-batch has 256 images among 8 GPUs. For CIFAR-10 and CIFAR-100, we train for 300 epochs with a mini-batch of 64 images. The learning rate is set to 0.1 and divided by 10 at 50% and 75% of the training procedure. For our results on CIFAR, we run for 3 times with different seeds and report mean values.

Ablation Study

In order to test the effectiveness of our proposed LSTM time stepping controller TSC$_{LSTM}$, we conduct experiments on ImageNet and compare with the two counterparts, TSC$_{indp}$ and TSC$_{2fc}$. We perform our methods with ResNet-50. As shown in Table 2, our re-implementation has a slightly better performance than reported. When armed with TSC$_{indp}$,

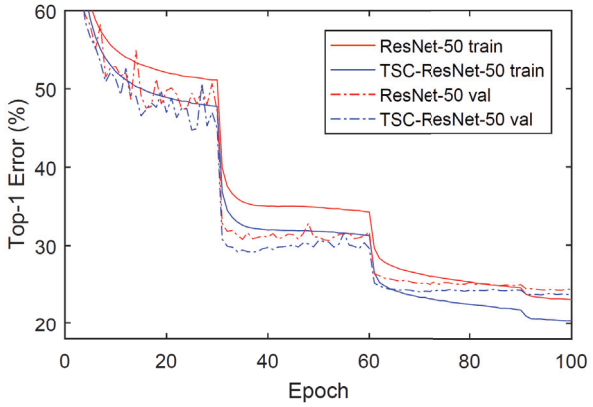

Figure 3: Top-1 error rate training curves of the baseline ResNet-50 and our TSC$_{LSTM}$-ResNet-50 on ImageNet.

ResNet has a small performance improvement, due to the introduced step sizes as explicit parameters. It reveals that a trainable step size benefits the ResNet performance. TSC$_{2fc}$ and TSC$_{LSTM}$ have larger improvements, which shows that the controller dependent on the convolution parameters contributes to a better performance. TSC$_{LSTM}$ is further aware of previous steps, because of the memory brought by LSTM, and has an improvement of 0.79% top-1 accuracy than baseline. The training curves of our TSC$_{LSTM}$ and baseline are compared in Figure 3. It is shown that our method has a superiority during the whole training procedure. The ablation study demonstrates the effectiveness of our design that the step sizes should be dependent on convolution parameters and aware of previous steps. We use the LSTM controller TSC$_{LSTM}$ for our later experiments.

Besides, the optimized step sizes belong to part of the structural information and are not data dependent. Thus, when training finishes, only the step sizes should be stored, and the additional overhead introduced by our controller can be spared in inference. As shown in Table 2, compared with baseline, these three methods have little additional cost in inference. For training, TSC$_{indp}$ has the same parameters and computation as inference because it does not use a parameterized controller. TSC$_{LSTM}$ consumes less parameters but more computation than TSC$_{2fc}$ in training. It is consistent with our analysis in prior section.

Improving the Performance

We add our time stepping controller on ResNet families with different depths and different variants, including ResNeXt (Xie et al. 2017) and SENet (Hu, Shen, and Sun 2018), to validate the ability of improving performance. As shown in Table 3, for fair comparison, we re-implement the baseline methods and most of our re-implementation performance are superior to the reported numbers.

When armed with our time stepping controller, it is shown that ResNets in different depths consistently have a 0.6%-0.8% improvement on performance. ResNet-50 has the largest accuracy gain. We see that our methods introduce a small number of parameters and computation for training,

Table 3: Top-1 error rates on ImageNet with and without our proposed time stepping controller. The numbers in brackets show the performance improvement over the re-implemented baselines. The left and right numbers of Params or GFLOPs columns show the overhead of our methods during training and inference phase respectively. All baseline methods are not able to reduce the overhead during inference phase, so there is a single value on the corresponding columns.

Models	re-implementation			with our time stepping controller		
	Error. (%)	Params(M)	GFLOPs	Error. (%) (gain)	Params(M) *(train / infer)*	GFLOPs *(train / infer)*
ResNet-18	29.41	11.69	1.81	28.80 **(0.61)**	13.52 / **11.69**	1.81 / **1.81**
ResNet-34	26.03	21.80	3.66	25.39 **(0.64)**	23.63 / **21.80**	3.66 / **3.66**
ResNet-50	24.42	25.56	3.86	23.63 **(0.79)**	27.83 / **25.57**	3.89 / **3.86**
ResNet-101	22.94	44.55	7.58	22.24 **(0.70)**	46.82 / **44.58**	7.64 / **7.58**
ResNeXt-50	22.84	25.03	3.77	22.23 **(0.61)**	27.30 / **25.04**	3.80 / **3.77**
ResNeXt-101	21.88	44.18	7.51	21.17 **(0.71)**	46.45 / **44.21**	7.57 / **7.51**
SENet-50	23.27	28.09	3.87	22.75 **(0.52)**	30.36 / **28.10**	3.90 / **3.87**
SENet-101	22.37	49.33	7.60	21.82 **(0.55)**	51.60 / **49.36**	7.66 / **7.60**

and nearly no considerable extra cost for inference.

We also add our time stepping controller on ResNet variants to test the scalability. The implementation of ResNeXt is similar to ResNet. It is shown that our method is also effective to ResNeXts. For ResNeXt-50, it has a top-1 accuracy improvement of 0.61%, while ResNeXt-101 has a larger gain of 0.71%.

Compared with feature operating modules, such as the channel-wise attention in SENet, our method may not have strong advantages for improving the performance, because the attention methods are data dependent, while ours are searching for adaptive step sizes, which are independent from features and belong to structural information. In spite of this, we note that TSC-ResNet-101 (22.24% top-1 error rate) has surpassed the performance of SENet-101 (22.37% top-1 error rate) using less parameters and computation. Our performance gain has little extra cost in inference, which cannot be realized by the feature dependent method SENet. We also show that our time stepping controller is compatible with SENets, even if they share a similar propagation formulation as Eq. (13). Our method reduces a top-1 error rate of 0.52% for SENet-50, and 0.55 % for SENet-101.

From Table 3, we observe that a deeper model benefits more from our time stepping controller in general. We believe that the reason is that a deeper network suffers more from the effects of step sizes. As indicated by Eq. (7), when depth increases, the cumulative influence of the spectral norm of weight matrices and step sizes become larger. In this case, an inappropriate step size would heavily impede the stability and performance of the network. It is in line with our intuition that deeper networks have more difficulties of training. Our method has an adaptive time stepping controller to adjust the steps sizes jointly with the network training, and thus helps more for deeper networks.

Improving the Stability

In order to test the ability of our time stepping controller to improve the stability, we conduct experiments to show the controller's robustness to perturbations and increasing depths.

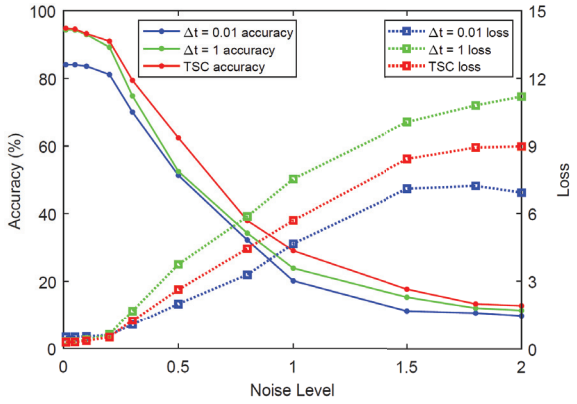

Figure 4: Under different noise level, the inference accuracy and loss of ResNets with step sizes 1, 0.01, and our time stepping controller (TSC) on CIFAR-10 test set.

We train ResNet-56 on CIFAR-10 with different step sizes (0.01 and 1), and our time stepping controller. After training, we inject different level Gaussian noise to the input for inference on the test set. The level of perturbation is decided by the standard deviation of the synthetic Gaussian noise. As shown in Figure 4, when the noise level increases, the accuracies of $\Delta t = 0.01$ and $\Delta t = 0.01$ both drop quickly. The loss of $\Delta t = 1$ has a sharp increment compared with that of $\Delta t = 0.01$. It is in line with our analysis in Eq. (7) that a small step size in each layer helps to bound the adverse effect caused by perturbations. However, the performance of $\Delta t = 0.01$ is significantly worse than $\Delta t = 1$. As a comparison, our time stepping controller offers adaptive step sizes. It is shown that TSC has a moderate loss increment with the noise level rising. Although TSC has a higher loss than $\Delta t = 0.01$ when noise level is larger than 0.5, the accuracies of TSC are consistently better than $\Delta t = 1$ and $\Delta t = 0.01$. This demonstrates that our stepping controller improves the ResNet robustness to perturbations, and offers a good trade-off between the stability and performance.

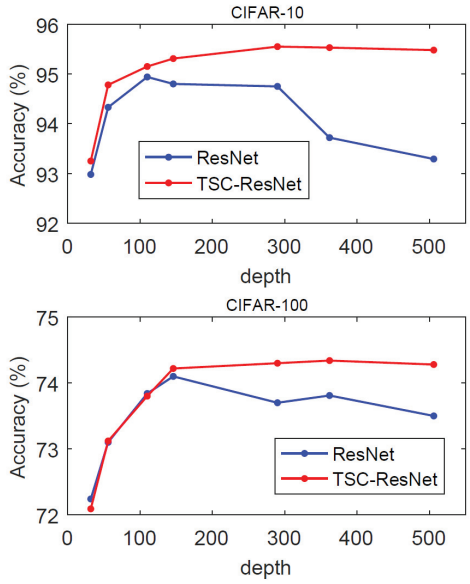

Figure 5: Comparison of ResNets and TSC-ResNets on CIFAR-10 and CIFAR-100 with increasing depths (32, 56, 110, 146, 290, 362 and 506 layers).

We further test the robustness of our method to increasing depths. As shown in Figure 5, we train ResNet and TSC-ResNet with different depths on the CIFAR-10 and CIFAR-100 datasets. For shallow residual networks, TSC-ResNet and ResNet have similar performances. When depth increases, the accuracies of ResNets approach a plateau, and then have a drop for very deep networks. A similar result is also reported in (Zhang et al. 2019). As a comparison, the performance of TSC-ResNet is more stable. It keeps a slow increment in accuracy for large depths. We note that the performance gap between TSC-ResNet and ResNet is larger for deeper networks in general, which is consistent with our observation on the ImageNet experiments in the prior section. This demonstrates our analysis that deeper networks suffer from an accumulated instability, and thus gain more benefits from our adaptive time stepping controller.

Analysis

As shown in Table 4, we average the optimized time step size vectors of TSC-ResNet-50 and TSC-ResNet-101 in different layers. We found that the step sizes in shallow layers of both TSC-ResNet-50 and TSC-ResNet-101 are centered around 0.5. For the layers in the last size stage (Conv-5_1, 5_2, 5_3), the step sizes are approaching 1. Conv-3_1 and 4_1 keep the initial state and are mildly affected by our method, because each of our time stepping controller consider previous steps but they correspond to the first time step in each size stage. Besides, the shallow layers should have small step sizes to avoid accumulated instability. But for layers Conv-5_1, 5_2, and 5_3, they should enlarge step sizes to achieve strong feature transformations for the final representation.

We also observe that, deeper layers in Conv-5_1, 5_2, and 5_3 converge to larger step sizes. For the same layers in Conv-5_1, 5_2, and 5_3, TSC-ResNet-101 converges to a larger step size. It reveals that deeper layer or deeper model requires a larger step size. It is in line with our experimental results that deeper models gain more benefits from our method in general. We believe that the reason lies in that deeper network corresponds to longer evolution time and suffer more from inappropriate time stepping. We also conjecture that our adaptive time stepping's effects on shallow layers mainly ensure stability, while the adjustments for deep layers contribute to the performance gains.

Table 4: The final optimized time step sizes (averaged along the channel dimension) of TSC-ResNet-50 and TSC-ResNet-101 in different layers. "C-3_1" denotes the first residual block in the Conv-3 size stage.

Models	C-3_1	C-4_1	C-4_2	C-5_1	C-5_2	C-5_3
TSC-50	0.510	0.500	0.731	0.835	0.901	0.926
TSC-101	0.508	0.513	0.657	0.854	0.927	0.950

Extension to Non-residual Networks

Our analyses and the development of our time stepping controller are based on the correspondence between residual networks and discrete dynamical systems. In order to test the scalability of our method to other networks, we add the controller to two non-residual network structures, DenseNet (Huang et al. 2017) and CliqueNet (Yang et al. 2018). The results are shown in Appendix C.

Conclusion

In this study, we use the correspondence between residual networks and discrete dynamical systems to unravel the physics of ResNets. We analyze the stability condition of the Euler method and ResNet propagation, and point out the effects of step sizes on the stability and performance of ResNets. Inspired by the adaptive time stepping in numerical methods of ODEs, we develop an adaptive time stepping controller that is dependent on the parameters of the network and aware of previous steps to adaptively adjust the step sizes and evolution time. Experiments on ImageNet, CIFAR-10, and CIFAR-100 show that our method is able to improve both performance and stability, without introducing much overhead in inference phase. Our method can also be applied to other non-residual network structures.

Ackonwledgement

Z. Lin is supported by NSF China (grant no.s 61625301 and 61731018), Major Scientific Research Project of Zhejiang Lab (grant no.s 2019KB0AC01 and 2019KB0AB02), and Beijing Academy of Artificial Intelligence. J. Wu is supported by the Fundamental Research Funds of Shandong University and SenseTime Research Fund for Young Scholars.

References

Ascher, U. M., and Petzold, L. R. 1998. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam.

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2017. Designing neural network architectures using reinforcement learning. In *ICLR*.

Behrmann, J.; Duvenaud, D.; and Jacobsen, J.-H. 2019. Invertible residual networks. In *ICML*.

Chang, B.; Meng, L.; Haber, E.; Ruthotto, L.; Begert, D.; and Holtham, E. 2018a. Reversible architectures for arbitrarily deep residual neural networks. In *AAAI*.

Chang, B.; Meng, L.; Haber, E.; Tung, F.; and Begert, D. 2018b. Multi-level residual networks from dynamical systems view. In *ICLR*.

Chen, L.-C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; and Yuille, A. L. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI* 40(4):834–848.

Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. In *NeurIPS*, 6571–6583.

E, W. 2017. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*.

Haber, E., and Ruthotto, L. 2017. Stable architectures for deep neural networks. *Inverse Problems* 34(1):014004.

Haber, E.; Ruthotto, L.; Holtham, E.; and Jun, S.-H. 2018. Learning across scales—multiscale methods for convolution neural networks. In *AAAI*.

Haber, E.; Lensink, K.; Triester, E.; and Ruthotto, L. 2019. Imexnet A forward stable deep neural network. In *ICML*.

Hairer, E.; Nørsett, S. P.; and Wanner, G. 1991. *Solving ordinary differential equations. 1, Nonstiff problems*. Springer-Vlg.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 1026–1034.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

He, X.; Mo, Z.; Wang, P.; Liu, Y.; Yang, M.; and Cheng, J. 2019. Ode-inspired network design for single image super-resolution. In *CVPR*, 1732–1741.

Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-excitation networks. In *CVPR*.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *CVPR*.

Jia, X.; Liu, S.; Feng, X.; and Zhang, L. 2019. Focnet: A fractional optimal control network for image denoising. In *CVPR*, 6054–6063.

Kwakernaak, H., and Sivan, R. 1972. *Linear optimal control systems*, volume 1. Wiley-interscience New York.

Li, Q., and Hao, S. 2018. An optimal control approach to deep learning and applications to discrete-weight neural networks. In *ICML*.

Li, Q.; Chen, L.; Tai, C.; and Weinan, E. 2017. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research*.

Li, H.; Yang, Y.; Chen, D.; and Lin, Z. 2018. Optimization algorithm inspired deep neural network structure design. *arXiv preprint arXiv:1810.01638*.

Lu, Y.; Zhong, A.; Li, Q.; and Dong, B. 2018. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *ICML*.

Miyato, T.; Kataoka, T.; Koyama, M.; and Yoshida, Y. 2018. Spectral normalization for generative adversarial networks. In *ICLR*.

Ruthotto, L., and Haber, E. 2018. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*.

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Sokolić, J.; Giryes, R.; Sapiro, G.; and Rodrigues, M. R. 2017. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing* 65(16):4265–4280.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*, 1–9.

Tran, D.; Wang, H.; Torresani, L.; Ray, J.; LeCun, Y.; and Paluri, M. 2018. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*, 6450–6459.

Veit, A.; Wilber, M. J.; and Belongie, S. 2016. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, 550–558.

Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; and He, K. 2017. Aggregated residual transformations for deep neural networks. In *CVPR*.

Yang, Y.; Zhong, Z.; Shen, T.; and Lin, Z. 2018. Convolutional neural networks with alternately updated clique. In *CVPR*.

Yoshida, Y., and Miyato, T. 2017. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*.

Zhang, J., and Laura, W. 2018. Smooth inter-layer propagation of stabilized neural networks for classification. *arXiv preprint arXiv:1809.10315*.

Zhang, J.; Han, B.; Wynter, L.; Low, K. H.; and Kankanhalli, M. 2019. Towards robust resnet: A small step but a giant leap. In *IJCAI*.