Towards Adaptive Residual Network Training: A Neural-ODE Perspective

Chengyu Dong [1] Liyuan Liu [2] Zichao Li [1] Jingbo Shang [1]

Abstract

In pursuit of resource-economical machine learning, attempts have been made to dynamically adjust computation workloads in different training stages, i.e., starting with a shallow network and gradually increasing the model depth (and computation workloads) during training. However, there is neither guarantee nor guidance on designing such network grow, due to the lack of its theoretical underpinnings. In this work, to explore the theory behind, we conduct theoretical analyses from an ordinary differential equation perspective. Specifically, we illustrate the dynamics of network growth and propose a novel performance measure specific to the depth increase. Illuminated by our analyses, we move towards theoretically sound growing operations and schedulers, giving rise to an adaptive training algorithm for residual networks, LipGrow, which automatically increases network depth thus accelerates training. In our experiments, it achieves comparable performance while reducing $\sim 50\%$ of training time.

1. Introduction

Residual networks have been advancing the state of the art with deeper and deeper models (He et al., 2016a;b; Devlin et al., 2019; Liu et al., 2020). The increasingly heavy training cost is impeding efficient iterations in both research and industrial applications, thus ensuing as a major challenge. To accelerate the training, various strategies have been proposed (Chen et al., 2015; Huang et al., 2016; Chang et al., 2017; Istrate et al., 2018) but mostly are heuristic.

In this paper, we present adaptive residual network training, which starts from optimizing a shallow network, and gradually increases the depth as the training proceeds. The network growing scheme in a residual architecture setting is not new (Chang et al., 2017; Wen et al., 2019), but the dynamics of growing is poorly understood. In particular, the following questions, of our major interests but not exhaustively enumerated, remain unclear: (1) Why the functions in a shallow network can be successfully inherited by a deeper network? In other words, why the network growing can be feasible in the first place? (2) How the increasing depth and continual optimization simultaneously affect the growing performance? (3) What consequences will different transformations of the network make? (4) When is an appropriate timing to grow the network?

In this work, we conduct theoretical analyses to explore the dynamics of network grow, and make a step forward to clarify these intriguing questions. Specifically, we view a network with N residual blocks as an ordinary differential equation (ODE) solver with N time steps. Inspired by the global truncation error and embedded error introduced in classic ODE theory (Ascher & Petzold, 1998), we align a finite and discrete residual network with the optimal ODE solution, and with a deeper network respectively, in order to bound the growing performance both globally and locally. Analyses of the bounds shed insights on the design of effective growing operator and scheduler.

In light of our analysis, we propose LipGrow, an adaptive training algorithm for residual networks. After each epoch, it calculates the Lipschitz constant of the residual blocks and then decides whether it is the right timing to increase the depth. Such process is completely automated and requires no excessive and meticulous parameter tuning. In our experiments on the CIFAR-10, CIFAR-100, and Tiny-ImageNet, LipGrow[1] reduces $\sim 50\%$ of training time, while achieving competitive or even better accuracy.

In summary, our major contributions are:

- We conduct theoretical analyses on the dynamics of network growing, describe a novel performance measure, and clarify the reason that such growing can be feasible.
- We propose an effective adaptive training algorithm for residual networks, LipGrow, in light of our analyses;
- We empirically verified that LipGrow leads to about 50% training time reduction while retaining the competitive or even better performance.

[1]University of California, San Diego [2]University of Illinois at Urbana-Champaign. Correspondence to: Jingbo Shang <jshang@eng.ucsd.edu>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

[1]`https://github.com/shwinshaker/LipGrow`

2. Network Grow in Neural ODE Perspective

Aiming to reduce the computation workload for model training, network grow adjusts the network structure while trying to retain the performance of the final model. Although it could be straightforward to design the structure transformation, analyzing the corresponding performance change is challenging. Specifically, it requires us to describe the relationship between models before and after the transformation, while these models have not only different parameters but also different architectures.

Fortunately, the neural ODE perspective, as introduced below, implies the existence of an "optimal" network, which can be interpreted as a functional limit that finite residual networks can converge to. Such convergence coincides with the network sequence produced by the grow, and allows us to conduct rigorous analysis and propose adaptive residual network training. Furthermore, the performance metric can now be defined in the functional space, allowing direct analyses of the effect of functional transformation induced by the grow, and functional state of the network engendered by the optimization.

In this section, we first briefly review the neural ODE perspective, whereby the picture of network grow is exhibited.

2.1. Neural ODE

The residual network can be constructed as

$$\boldsymbol{z}_{n+1}^{(N)} = \boldsymbol{z}_n^{(N)} + h^{(N)} f_n^{(N)}(\boldsymbol{z}_n^{(N)}), \tag{1}$$

where N denotes the depth of the residual network, $n \in \{0, ..., N-1\}$ is the index of the residual block, $h^{(N)}$ is a fixed step size, $\boldsymbol{z}$ represents the feature map, $f_n^{(N)}$ represents the n-th residual block function.

Equation 1 resembles the Euler method for solving an ordinary differential equation (ODE) (Haber & Ruthotto, 2017; Weinan, 2017). Here $f_n^{(N)}(\cdot)$ is only defined at discrete time points $t_n^{(N)} = t_s + (t_e - t_s)(n/N)$, where t_s and t_e are the start and end time respectively. But we can always employ a relaxation and define the residual functions continuously along the time dimension (specific construction is elaborated in the Appendix). We formulate the yielded ODE as

$$\frac{d\boldsymbol{z}^{(N)}(t)}{dt} = f^{(N)}(t, \boldsymbol{z}^{(N)}(t)). \tag{2}$$

The learning objective can be defined as

$$\varepsilon(f^{(N)}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\Phi(\mathcal{F}^{(N)}(\boldsymbol{x}), \boldsymbol{y}(\boldsymbol{x})),$$

where Φ is a smooth criterion function. We wish to find an appropriate solution $f^{*(N)}$, such that

$$f^{*(N)} = \arg\min_{f^{(N)}} \varepsilon(f^{(N)}).$$

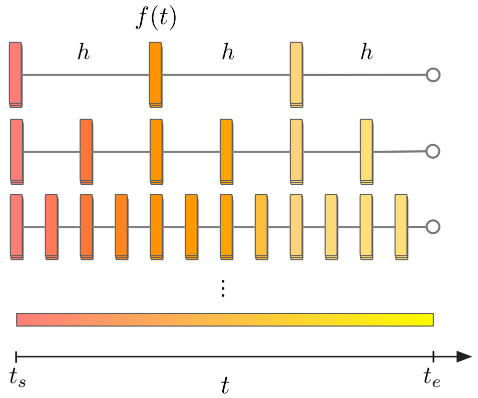

Figure 1: Convergence of a sequence of residual networks. A residual network can be viewed as a cascade of residual functions exerted at discrete time points evenly spaced at a step size of h. As the depth approaches infinity, it converges to a functional limit that is smoothly defined in the time dimension.

The network function $\mathcal{F}^{(N)}$ is connected to the ODE by

$$\mathcal{F}^{(N)}(\boldsymbol{x}) \equiv \boldsymbol{z}^{(N)}(t_e) = \boldsymbol{z}^{(N)}(t_s) + \int_{t_s}^{t_e} f^{(N)}(t, \boldsymbol{z}^{(N)}(t))dt.$$

2.2. Optimal ODE

As the depth of the ResNet goes to infinity, we assume it approaches a continuous limit, which has been analyzed in previous works. Specifically, Avelin & Nyström (2019) prove that a ResNet with shared weights across blocks converges to an autonomous ODE. Müller (2019) prove that the continuous ODE can be approximated by a ResNet with finite depth. Lu et al. (2020) provides a new continuous limit that guarantees the optimality of minimizers. In a network grow scheme, the behaviour of the minimizer as the depth increases is of particular interest. Thorpe & van Gennip (2018) prove that, under specific regularity conditions, a sequence of minimizers of finite ResNets converges to a continuous limit, which itself minimizes $\varepsilon(f^{(\infty)})$.

Theorem 1 (Convergence of a sequence of minimizers)**.** *Let $f^{*(N)}$ be the minimizer of $\varepsilon(f^{(N)})$, and $\{f^{*(N)}\}_{N\in\mathbb{N}}$ be a sequence of such minimizers. If*

$$\sup_N \left(\left\| f_0^{*(N)} \right\|^2 + N \sum_n \left\| f_n^{*(N)} - f_{n+1}^{*(N)} \right\|^2 \right) < +\infty,$$

then

$$\lim_{N\to\infty} f^{*(N)}(t) = f^*(t), \ \forall\, t,$$

and f^* is the minimizer of $\varepsilon(f^{(\infty)})$.

We now refer to the minimizer of a residual network in the continuous limit as the optimal ODE. Theorem 1 shows the convergence of discrete residual functions to the optimal ODE, as illustrated in Figure 1. An important implication of such convergence is that, the minimizers of two deep residual networks are not too far from each other, which demonstrates the feasibility of effective network grows.

Note that the regularity condition here requires adjacent residual blocks to be similar, which is not trivial as the depth goes to infinity. An effective network grow ought to adhere to such regularity to benefit from the convergence.

2.3. Growing Performance Evaluation

The optimal ODE reasonably bounds the performance of a network in a residual architecture given the maximized capacity [2]. Note that the optimal ODE is not necessarily identical to the Bayesian optimal model as the input-output relation may not exactly follow an ODE and the global optimal may be impossible to find by any realistic optimizer. Therefore, to investigate the performance of a residual network with finite depth, it is reasonable to align it with the optimal ODE. The learning objective can be rewritten as

$$\tilde{\varepsilon}(f^{(N)}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\Phi(\mathcal{F}^{(N)}(\boldsymbol{x}), \mathcal{F}^{*}(\boldsymbol{x})),$$

where the outputs are compared to the optimal ODE instead of the true labels.

Given Φ as a smooth l-Lipschitz criterion function, it is more convenient to investigate such difference in the feature map space with any norm $\|\cdot\|$. Strictly speaking, since

$$\Phi(\mathcal{F}^{(N)}(\boldsymbol{x}), \mathcal{F}^{*}(\boldsymbol{x})) \leq l\cdot\|\mathcal{F}^{(N)}(\boldsymbol{x}) - \mathcal{F}^{*}(\boldsymbol{x})\|,$$

the learning objective can be alternatively defined as

$$e^{(N)} \equiv \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\|\mathcal{F}^{(N)}(\boldsymbol{x}) - \mathcal{F}^{*}(\boldsymbol{x})\|.$$

Here, we refer $e^{(N)}$ as the global error to be minimized.

2.4. Co-optimization of Parameter and Depth

We now wish to properly bound the global error. A deep network is a cascade of function blocks. With identical inputs, the global error is an accumulation of the functional difference induced in each block.

Theorem 2 (Upper Bound of Global Error)**.** *Assume the maximum L^{∞} distance (denoted as $\|\cdot\|_{\infty}$) between $f^{(N)}$ and f^{*} is bounded. Namely,*

$$\sup_{t}\left\|f^{*}(t) - f^{(N)}(t)\right\|_{\infty} = C^{(N,*)},$$

then the global error is bounded by

$$e^{(N)} \leq \frac{\exp\left[\mathcal{L}\left(f^{*}\right)\left(t_e - t_s\right)\right] - 1}{\mathcal{L}\left(f^{*}\right)}\left[\frac{M(\boldsymbol{z}^{*})}{2}h^{(N)} + C^{(N,*)}\right], \tag{3}$$

where $M(\boldsymbol{z}^{*}) = \sup_t \|\boldsymbol{z}^{*\prime\prime}(t)\|$, and $\mathcal{L}(f^{*})$ is the maximum Lipschitz constant of f^{*}.

[2] In an augmented sense. Practical realizations of continuous ODE (Neural ODE), and discretized ODE (ResNet) always include leading layers which lift the dimension (Dupont et al., 2019).

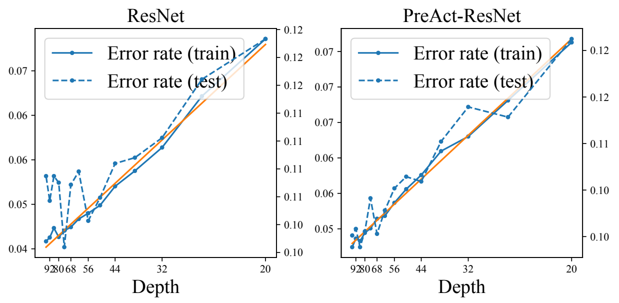

Figure 2: Relationship between Step Size and the Performance of ResNet and Pre-Activation ResNet on the CIFAR-10 dataset.

The functional difference term included in the theorem validates our above intuition. A residual block in a finite network is an impulse exerted at a specific time point that approximates the cumulative effect of the optimal ODE within a time interval. The approximation error is caused by the relative functional difference and accumulatively leads to the output difference considering all residual blocks.

Another interesting finding is that the global error is linearly correlated with the step size. The fact that $h^{(N)} \propto 1/N$ immediately implies that the model performance consistently improves as the network becomes deeper. This justifies the crucial impact of depth in residual network learning. We empirically verify this correlation on the CIFAR-10 dataset in Figure 2. Note that for pre-activation ResNet (He et al., 2016b), both the training and test error linearly correlates with stepsize. On the other hand, for the original ResNet (He et al., 2016a), the correlation is less significant, since it applies an additional non-linear transformation after the addition of residual thus not strictly subject to the discrete ODE formula in Equation 1.

Theorem 2 demonstrates that the global error is an appropriate evaluation metric for the network growing performance. The effect of capacity augmentation, as revealed by the step size, and the effect of parameter optimization, as revealed by the functional error, are disentangled in the upper bound, which implies the co-optimization nature of the network growing. It sheds lights to effective growing method design.

3. Towards Effective and Automated Grow

Inspired by our analysis before, we propose to conduct adaptive training in an automated manner.

3.1. Generic Adaptive Training

Algorithm 1 presents a generic adaptive training setup. In each epoch, the model is first updated in the conventional training manner. After that, if the growing scheduler $\mathcal{S}$ decides to increase the depth of the network, the growing operator $\mathcal{G}$ will initialize a deeper network based on the current network for continual training. Following this setup, the adaptive training algorithm design is nailed down to specific choices of $\mathcal{S}$ and $\mathcal{G}$.

Algorithm 1 Generic Adaptive Training

```
Require: Growing scheduler $\mathcal{S}$, Growing operator $\mathcal{G}$,
    Dataset $\mathcal{D}$, Optimizer (w. loss) $\mathcal{A}$.
1: while within training budget do
2:   for $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ do
3:     $f := \mathcal{A}(f, \boldsymbol{x}, \boldsymbol{y})$ (Conventional Parameter Update)
4:     if $\mathcal{S}(f)$ then
5:       $f := \mathcal{G}(f)$ (Increase the Depth of the Network)
6:     end if
7:   end for
8: end while
Return: Final network $f$
```

In our problem, for a desired deep model, we are interested in maximizing its performance through adaptive training, or equivalently minimizing the training cost while the performance is comparable. Therefore, it is different from Neural Architecture Search, which aims at selecting the final architecture.

3.2. How to Grow – Minimize Functional Difference

The specific initialization method in the growing operator $\mathcal{G}$ is important to ensure an effective training. In the broad topic of network transformation, multiple strategies are employed, including identity initialization (Chen et al., 2015) or its generalized version known as network morphism (Wei et al., 2016; Jin et al., 2018), random initialization (Wen et al., 2019), initialization with partial training (Istrate et al., 2018), and duplication (Chang et al., 2017).

In terms of the performance of the final model, and for residual network particularly, the bound of global error provided by Theorem 2 implies that it is important to investigate the functional difference induced by the initialization. Specifically, the residual functions of a deeper network $f^{(N^+)}$ after growing differ from the optimal ODE by

$$C^{(N^+,*)} = \sup_t \|f^*(t) - f^{(N^+)}(t)\|_\infty.$$

Towards minimizing such functional difference, we propose a simple cloning method, which clones the residual blocks of the previous network to the new network. However, the residual functions in a finite depth network are defined at a set of discrete time points. Directly cloning from the corresponding time points may only be trivial since the sets of time points in two networks are not necessarily aligned.

The best choice is then cloning from the nearest time points, namely for every desired time point $t^+ \in \{t_n^{(N^+)}\}$ in the new network, a non-trivial time point $t \in \{t_n^{(N)}\}$ is found in the previous network such that $t \equiv \chi(t^+) = \arg\min_t |t - t^+|$, and the corresponding residual function is cloned as $f^{(N^+)}(t^+) := f^{(N)}(\chi(t^+))$.

Now the functional difference can be bounded, since

$$C^{(N^+,*)} \leq C_t^* + C^{(N,*)},$$

where the second term $C^{(N,*)}$ is the functional difference carried over from the previous network, which only depends on the optimizer $\mathcal{A}$, and should be close to 0 if the optimization of the previous network is effective. The first term $C_t^* \equiv \sup_t \|f^*(t) - f^*(\chi(t))\|_\infty$ should also be close to 0, if the network is sufficiently deep. Therefore it can be concluded, as the network becomes increasingly deep, the residual functions of the network after growth are guaranteed to be close to the optimal ODE.

Such cloning method also maintains the regularity condition in Theorem 2. Since all of the residual functions are cloned from previous network, the sum of the differences between adjacent residual functions will at most be equal, which ensures efficient continual optimization after the grow given relative close network minimizers.

The deficiency of other initialization methods can be qualitatively analyzed in the same way (See Appendix).

3.3. When to Grow – Bound Temporal Error

In this section, we explore to determine a proper timing of the grow. Theorem 2 bounds the performance of a network w.r.t. the optimal ODE, which disentangles the effects of capacity increase and parameter optimization, however, it is not a practical guide to the grow schedule since the optimal ODE is never known beforehand. In fact, in a growing scheme, the only known network is the one that is currently being optimized. It is necessary to bound the performance of a future network $\mathcal{F}^{(N^+)}$ w.r.t. the current network $\mathcal{F}^{(N)}$, where $N^+ > N$. It can be formulated as

$$e^{(N^+)} \leq e^{(N)} + e^{(N,N^+)},$$

where

$$e^{(N,N^+)} \equiv \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}} \|\mathcal{F}^{(N^+)}(\boldsymbol{x}) - \mathcal{F}^{(N)}(\boldsymbol{x})\|.$$

$e^{(N,N^+)}$ is the error introduced in the outputs due to the grow, which we refer as temporal error.

Theorem 3 (Upper Bound of Temporal Error). *Assume the maximum L^∞ distance (denoted as $\|\cdot\|_\infty$) between $f^{(N)}$ and $f^{(N^+)}$ is bounded. Namely,*

$$\sup_t \left\| f^{(N)}(t) - f^{(N^+)}(t) \right\|_\infty = C^{(N,N^+)}, \qquad (4)$$

then the temporal error is bounded by

$$e^{(N,N^+)} \leq \frac{\exp\left[\mathcal{L}(f^{(N)})(t_e - t_s)\right] - 1}{\mathcal{L}(f^{(N)})} \cdot \Big[C^{(N,N^+)} + \tag{5}$$
$$0.25 \cdot M(f^{(N^+)}) \cdot \mathcal{L}(f^{(N)}) \cdot h^{(N)}\Big],$$

where $M(f^{(N)}) = \sup_t \|f^{(N)}(t)\|_\infty$, and $\mathcal{L}(f^{(N)})$ is the maximum Lipschitz constant of $f^{(N)}$.

It can be observed that the temporal error depends on the functional difference, which can be further bound by

$$C^{(N,N^+)} \leq C^{(N,N^+ \leftarrow N)} + C^{(N^+,N^+ \leftarrow N)},$$

where $C^{(N,N^+ \leftarrow N)} \equiv \sup_t \|f^{(N)}(t) - f^{(N^+ \leftarrow N)}(t)\|_\infty$, which is the relative functional error introduced by the growth and should be 0 when using our cloning initialization; $C^{(N^+,N^+ \leftarrow N)} \equiv \sup_t \|f^{(N^+)}(t) - f^{(N^+ \leftarrow N)}(t)\|_\infty$, which is introduced by the optimizer after the growth. and $\mathcal{F}^{(N^+ \leftarrow N)}$ is the network right after the growth.

Our cloning initialization does introduce part of the temporal error due to the accumulated difference between corresponding feature maps, which is controlled by the maximum Lipschitz constant $\mathcal{L}(f^{(N)})$. Theorem 3 shows that the bound of the temporal error monotonously correlated with $\mathcal{L}(f^{(N)})$. Intuitively, the temporal error measures the stability of the network functionality w.r.t. the growing transformation. A large Lipschitz constant weakens the stability as it amplifies the error propagating through the network, although it does benefit the expressivity of the network. Empirical evidence of the correlation between growing performance and the Lipschitz constant is presented in Appendix.

In summary, the Lipschitz constant serves as an important indicator of the performance in a network grow scheme. The growing scheduler can be designed to control the Lipschitz constant such that it will not become too large.

4. LipGrow

Here, we elaborate the implementation details of LipGrow, which is summarized in Algorithm 2.

4.1. Cloning Initialization as Growing Operator

To initialize a residual network with depth N^+ from a network with depth N, the mapping χ is first captured by a one-dimensional nearest neighbour search, which can be easily calculated. Specifically, $\chi(t_{n^+}^{(N^+)}) = t_n^{(N)}$, where $n = \lfloor N(t_{n^+}^{(N^+)} - t_s)/(t_e - t_s) + 1/2 \rfloor$. The corresponding residual blocks are then cloned. The first residual block in each subnetwork will only be cloned to its counterpart in the new network, since it contains a downsampling layer. Other residual blocks that are assigned to the first residual block will be cloned from the block right behind it.

Implicit Step Size. To ensure a roughly constant magnitude of the sum of the residuals, we have to scale the step size after the growth. To avoid significant modification to the model architecture, we propose to scale the weights and bias of `conv` and `bn` layers by N/N^+, after the block cloning. Typically, layers are constructed as `conv-bn-relu-conv-bn-relu` in a basic residual block.

This implementation is equivalent to introducing an explicit step size and scaling the residual output of each block, as long as the activation function is ReLU. Note that based on this way, nothing needs to be changed for the architecture setting, which eases the adaptation to existing models. Moreover, we observe that this way is slightly better than introducing explicit step size in our experiments. Note that, within a block the scaling effect of previous layers will be cancelled out by subsequent batch normalization due to the standard deviation term. But intuitively we desire an equal scaling of all weights. Implementations thus may vary for specific layer orders.

Choice of N^+. In practice, the total number of training epochs is typically around a few hundreds. The number of growths is thus limited if the network needs to be sufficiently optimized in each stage. To facilitate the efficiency of the training and balance the optimization of each residual function, we choose the depth to be doubled in each grow and thus the number of blocks exponentially increase during the adaptive training. In this case, our initialization degenerates to the interpolation introduced by Chang et al. (2017), where every residual block will have a clone after it.

4.2. Lipschitz Tolerated Growing Scheduler

In our scenario, Lipschitz constant serves as an upper bound for the growing risk. In numerical analysis, the typical way to control such a risk is referring to an user-defined tolerance. Inspired by this idea, we define a risk tolerance as r_{tol}. Each time the scaled Lipschitz (see below) exceeds r_{tol}, the grow will be triggered. Consequently, network depth will be increased, and the stepsize will be scaled.

Since the magnitudes of Lipschitz constants at different depths are distinct, we will scale the Lipschitz by the value obtained at the first epoch of training. After growing, the scaling basis will be reset since the depth changes. We average the Lipschitz across blocks, and use a smoothing window of 10 epochs to reduce noise. A certain number of epochs [3] will be reserved for the final model if the desired depth is not reached yet (sometimes the Lipschitz is low all

[3] 30 epochs for a total of 164 epochs (CIFAR-like); 20 epochs for a total of 90 epochs (ImageNet-like)

Algorithm 2 Our LipGrow Algorithm

```
Require: Dataset $\mathcal{D}$, Optimizer (w. loss) $\mathcal{A}$, Tolerance $r_{\text{tol}}$.
 1: $L_0 = \mathcal{L}(\mathcal{F})$ (Calculate the Lipschitz constant of $\mathcal{F}$)
 2: while within training budget do
 3:   for $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ do
 4:     $\mathcal{F} = \mathcal{A}(\mathcal{F}, \boldsymbol{x}, \boldsymbol{y})$ (Conventional Parameter Update)
 5:     if $\mathcal{L}(\mathcal{F})/L_0 > r_{\text{tol}}$ then
 6:       $\mathcal{F} := \text{Clone}(\mathcal{F})$ (Initialize a Deeper Network w. Cloning)
 7:       $L_0 = \mathcal{L}(\mathcal{F})$ (Calculate the Lipschitz constant of $\mathcal{F}$)
 8:     end if
 9:   end for
10: end while
Return: Final network $\mathcal{F}$
```

the time, and we grow very late to maximize the efficiency).

4.3. Efficient Calculation of Lipschitz

We first show how to calculate the Lipschitz constant of residual networks and then discuss its overhead.

A residual network contains a few residual blocks, and each residual block have a few layers. As these layers are composed sequentially, the Lipschitz constant of the composed function is simply the multiplication of the Lipschitz constant of its every component. Since a residual block contains (1) convolutional layers, (2) activation layer, typically ReLU, and (3) batch normalization, here, we show the calculations of these three types as follows.

Convolutional Layers. Convolutional layers are essentially linear operations, whose Lipschitz constants should be approached by their operator norms.

$$\text{Lip}(f_{\text{conv}}) = \max_{X, X \neq 0} \frac{\|W_f X\|}{\|X\|} \tag{6}$$

Exact calculation of operator norm involves SVD and Fourier transformation (Sedghi et al., 2018), but it will introduce 10 times overhead in our experiments. The precision is after all not our major concern. Therefore, we perform the power iteration method (Yoshida & Miyato, 2017; Tsuzuku et al., 2018; Gouk et al., 2018) to estimate the operator norm approximately.

ReLU. Its Lipschitz constant is 1 (Tsuzuku et al., 2018).

Batch Normalization. Batch normalization performs the following linear transformation

$$f_{\text{BN}}(x_i) := \gamma_i \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta_i.$$

Therefore the Lipschitz constant is simply calculated as (Tsuzuku et al., 2018)

$$\text{Lip}(f_{\text{BN}}) = \max_i \frac{|\gamma_i|}{\sqrt{\sigma_i^2 + \epsilon}}.$$

Overhead. The majority of the overhead is introduced by the iteration method for convolutional layers. In our implementation, we run 100 iterations[4] for every convolution filter in every epoch. Such overhead is negligible compared to the total training time (Gouk et al., 2018).

In our experiments, we observe that the empirical overhead for each epoch is only a few seconds, as shown in Table 1. It is worth mentioning that the overhead is only related to the number and dimension of convolution filters, and is not relevant to the dataset size.

4.4. Adaptive Learning Rate Scheduler

To incorporate adaptive growing, the learning rate scheduler needs some special design. In Chang et al. (2017), *cosine annealing scheduler with restarts* (Loshchilov & Hutter, 2016) is adopted to facilitate multi-depth training, where the restart epochs are aligned with the manually selected epochs when the depth is increased. However, since we never know the growing epochs beforehand in our adaptive scenario, we propose a variant as follows.

Adaptive Cosine Annealing Learning Rate Scheduler. After each growth, the cycle in a standard cosine learning rate scheduler is reset as the number of epochs left, namely

$$\eta = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{T_{\text{cur}} - T_{\text{grow}}}{T_{\text{tot}} - T_{\text{grow}}} \pi\right)\right),$$

where $\eta_{\min}$ and $\eta_{\max}$ are the minimum and maximum learning rates, respectively. T_{cur} is the current epoch, T_{tot} is the total epochs, and T_{grow} refers to the epoch of the last growth. At the beginning of the training, $T_{\text{grow}} = 0$.

Experiments are conducted comparing our proposed scheduler, cosine annealing and cosine annealing with restarts in Appendix. Generally they all have similar performance.

5. Experiments

We conduct experiments on three benchmark datasets, CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009) [5], and Tiny-ImageNet[6]. We follow their standard train, validation, and test splits in our experiments.

[4]One can greatly reduce the number of iterations if the overhead is really a concern. As mentioned in (Yoshida & Miyato, 2017), 1 iteration is often sufficient.

[5]`www.cs.toronto.edu/~kriz/cifar.html`

[6]`www.kaggle.com/c/tiny-imagenet`

Table 1: Overhead Caused by Lipschitz Constant Calculations.

# Convolution Filters	18	24	30	36	42	48	54	60	66	72
Overhead(secs)/Epoch	0.4	0.6	0.7	0.9	0.9	1.0	1.1	1.4	1.5	1.7

Table 2: Evaluation Results on the CIFAR datasets.

Method	Final Model	CIFAR-10			CIFAR-100		
		Val	Test	PPE ($\times 10^6$)	Val	Test	PPE ($\times 10^6$)
Vanilla	ResNet-14	91.57 ± 0.10	91.57 ± 0.25	0.18	67.39 ± 0.31	67.50 ± 0.74	0.18
	ResNet-20	92.50 ± 0.26	92.22 ± 0.62	0.27	68.68 ± 0.45	69.75 ± 0.13	0.26
	ResNet-50	93.23 ± 0.24	93.59 ± 0.30	0.76	70.86 ± 0.60	71.40 ± 0.58	0.76
	ResNet-74	93.25 ± 0.15	93.76 ± 0.26	1.15	72.61 ± 0.38	73.16 ± 0.40	1.15
LipGrow	ResNet-50	92.89 ± 0.26	92.99 ± 0.33	0.33 ± 0.02	70.63 ± 0.26	71.70 ± 0.67	0.41 ± 0.03
	ResNet-74	93.53 ± 0.31	93.46 ± 0.69	0.54 ± 0.06	72.47 ± 0.18	72.75 ± 0.59	0.54 ± 0.01

Table 3: Evaluation Results on the Tiny-ImageNet dataset.

Method	Tiny ImageNet (ResNet-66 model)		
	Val	Test	PPE ($\times 10^6$)
Vanilla	50.13 ± 0.77	48.18 ± 0.21	48.90
LipGrow	50.54 ± 0.16	48.87 ± 0.40	25.54 ± 0.66

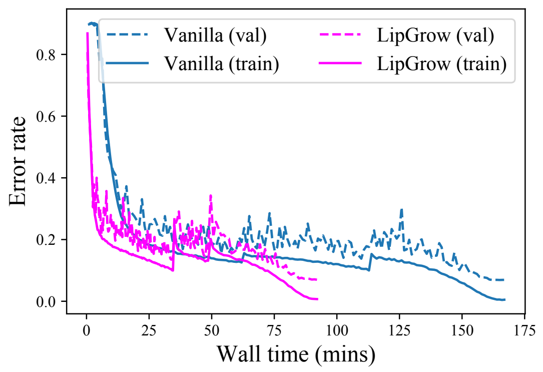

Figure 3: Training and Validation Error Rates w.r.t. Wall Training Time. The model is ResNet-74 trained on the CIFAR-10. We compare our LipGrow with vanilla training. All timings are based on a single Nvidia GeForce GTX 1080 Ti GPU.

5.1. Compared Methods

We mainly compare *LipGrow* with the *vanilla-ResNets* (referred as *Vanilla*), which are trained from scratch without adjusting its depth (*i.e.*, it directly trains a deep residual network from scratch). More experiments are conducted to compare *LipGrow* with a hand-tuned growing scheduler (referred as *Hand-Tuned*) in Section 5.5.

5.2. Experimental Settings

For the CIFAR datasets, we employ the ResNet model, which consists of 3 subnetworks[7], while for the Tiny-ImageNet dataset, we use the one consisting of 4 subnetworks. Accordingly, we experimented ResNet-12, ResNet-20, ResNet-50 and ResNet-74 on the CIFAR10 and CIFAR100 datasets, and ResNet-66 on the Tiny-ImageNet dataset.

All models on the CIFAR datasets are trained for 164 epochs, and evaluated on a single Nvidia GeForce GTX 1080 Ti GPU. All models on the Tiny-ImageNet dataset are trained for 90 epochs, and evaluated on a single Nvidia Quadro RTX 8000 GPU. We use a batch size of 128 for training, and 100 for validation. Weight decay and momentum are set to be 2×10^{-4} and 0.9, respectively. All statistics in experiments are based on 3 runs.

Towards a fair comparison with our method, we apply our proposed learning rate scheduler as well as implicit initialization to both Vanilla and Hand-Tuned.

In our adaptive growing strategy, the tolerance r_{tol} is the only hyper-parameter that needs to be tuned. It is typically chosen around 1.4, with only a marginal dependence on dataset[8]. Further experiments are conducted to explore the sensitivity of r_{tol}. According to our experiments, this tolerance value is largely universal (detailed discussions are included in the appendix).

5.3. Evaluation Results

In addition to the validation and test accuracy, towards efficiency evaluation, we introduce the parameters per epoch (*PPE*) metric. It is defined as the number of model parameters per epoch. For example, a fixed model of 1M parameters will result in 1M PPE. This is one of the fairest metrics reflecting the computation load (memory and processor) yet independent to hardware settings and utilization, since we desire to train a model with as few parameters as

[7]A subnetwork denotes a cascade of layers in a residual network where the output activations are of the same dimension.

[8]1.4 for CIFAR and 1.3 for Tiny-Imagenet

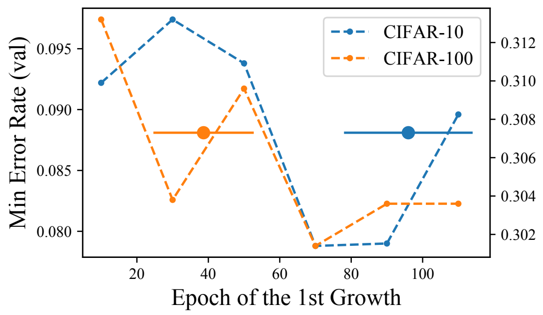

Figure 4: Grid Search vs. LipGrow in finding epochs for the first growth. Dashed lines refer to the minimum final error rate achieved by the corresponding scheduler. Solid lines describe the first epochs found by LipGrow (mean and std for 3 runs). X-axis is the epoch index of the growing timing of different growing schedulers. The model is ResNet-50 trained on the CIFAR-10 and CIFAR-100 datasets.

possible within a fixed budget of training epochs.

Results are presented in Tables 2 and 3. The model in the table denotes the final depth after grow. Each network will be grown for exactly 2 times, and each time the depth will be doubled. Therefore the growing path can be easily inferred by $N = 2^m(N_0 - 2) + 2$, where N_0 is the initial depth, and m is the number of grows.

LipGrow can reach comparable validation and test accuracy as Vanilla, while reducing the computation load by more than 50%. Moreover, LipGrow allows an automatic choice of the growing epochs, thus saving vast time to explore proper decisions, which may not be less computationally heavy than training a deep network from scratch. Section 5.5 offers more comparisons with *Hand-Tuned*.

To reflect the real-world training time, we visualize the learning curve w.r.t. the wall clock time of training in Figure 3, given the same computational environment. LipGrow demonstrates significant training speedup, while achieving similar performance in the end. This observation is consistent with our PPE measurements in Tables 2 and 3, which further verifies the effectiveness of our proposed method.

5.4. Effectiveness of LipGrow Growing Scheduler

To explore the effectiveness of LipGrow growing scheduler, we did grid search to evaluate the growing timing decided by LipGrow. Specifically, we conduct experiments with ResNet-50 on CIFAR-10 and CIFAR-100 and first run LipGrow multiple times and record both its first growing time and the final performance. Then, we conduct grid search on the growing time and record the corresponding final performance. The results are visualized in Figure 4. The first growth epochs chosen by LipGrow are nearly the optimal epochs based on the grid search results, although it could be some local optimal. Specifically, on CIFAR-10, the growing timing decided by LipGrow is near optimal, while on CIFAR-100, the growing timing decided by LipGrow nears a local optimal and fails to reach a better timing. This behavior is reasonable as LipGrow decides the growing on-the-fly, which leads to better training efficiency (*i.e.*, it does not require to conduct training multiple times for growing scheduler tuning). It also shows that the growth epochs chosen by LipGrow across different runs are relatively stable.

5.5. Adaptive vs. Hand-Tuned

In the end, we compare LipGrow with *Hand-Tuned* growing scheduler. Specifically, we first sample a set of different growing schedulers, evaluate the model performance trained with these sampled schedulers, and visualize their required training cost (*i.e.*, PPE) and best accuracy in Figure 5. Besides, we also list the *Hand-Tuned* scheduler recommended by Chang et al. (2017), which achieves a better performance with more training cost. It is worth mentioning that, all these compared methods require a trial-and-error approach, which contradicts to the goal of adaptive training (*i.e.*, cost of tuning growing scheduler is enough to train with a static depth). As depth balances model performance and training cost, choosing a growing scheduler can be viewed as a multi-objective optimization problem and all points achieving Pareto-efficiency are Pareto-optimal (*i.e.*, no other scheduler can achieve better performance with less training cost). Specifically, although *LipGrow* and *Hand-Tuned* achieve comparable performance (*i.e.*, *LipGrow* is more efficient while *Hand-Tuned* performs better), *LipGrow* achieves Pareto-optimal and Hand-Tuned can be further improved. It verifies the potential of deciding growing timing in a greedy manner and the effectiveness of LipGrow.

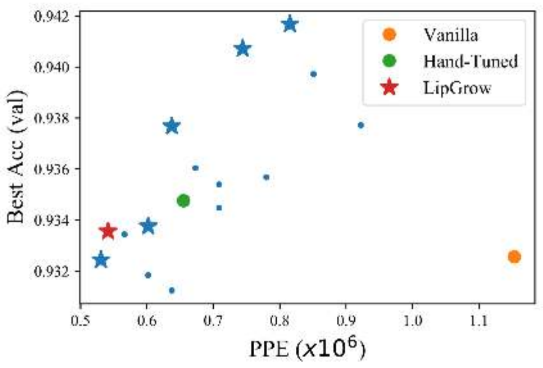

Figure 5: Validation Accuracy on CIFAR10 vs. Required PPE based on Different Choices of the Growing Epochs. Pareto-optimal schedulers (balancing cost and performance) are marked as star. Specifically, *LipGrow* achieves Pareto-optimal and Hand-Tuned can be further improved.

6. Related work

Here, we review the literature of three related topics.

Dynamic System View of Residual Network. Deep residual network can be viewed as a discretized ODE (Haber & Ruthotto, 2017; Weinan, 2017; Ruthotto & Haber, 2018). To this end, various perspectives and methods from numerical analysis are employed to improve the architecture (Lu et al., 2017; Ciccone et al., 2018), reduce training time (Chang et al., 2017), reduce memory overload (Chen et al., 2018), enable inversability (Chang et al., 2018), and adapt to other models (e.g., recurrent neural networks (Chang et al., 2019), Transformers (Lu et al., 2019)).

In particular, Chen et al. (2018) explicitly parameterize time in the residual architecture and use ODE solver to perform transformations. This modification can adaptively trade off precision for efficiency by controlling the number of evaluations. Nevertheless, applications to practical learning tasks will be limited since the blackbox ODE solver and typical downsample layers are mutually incompatible, which requires a completely new design of the model architecture. In contrast, our adaptive method views the trade-off *in situ*, thus can be seamlessly applied to state-of-the-art models.

Network Grow and Neural Architecture Search. Adaption of network architecture under specific situations is of constant interest. Early work has already proven the difficulty of training very deep neural networks (Glorot & Bengio, 2010). To circumvent this problem and improve training stability, easy shallow networks are often first trained, and then merged into deep networks (Simonyan & Zisserman, 2014). DropIn (Smith et al., 2016) further explores this idea by incrementally including new layers, making the train of deeper network possible and achieve better performance. As a counterpart, Huang et al. (2016) suggests to randomly disable network blocks during training, which can be interpreted as a DropOut on the network level. This trick significantly reduces the training time and improves the performance at the same time, though careful hyper-parameter engineering is required. Multi-level residual network (Chang et al., 2017) is the closest one linked to our work, which explores the possibility of augmenting network depth in a dynamic system of view, whereas proper time to the perform the augmentation is unknown beforehand. AutoGrow (Wen et al., 2019) attempts to automate the discover of proper depth to achieve near-optimal performance on different datasets. Several growing strategies are tested, yet the preferred one is a manually-tuned periodic setting.

Network Morphism (Wei et al., 2016; 2017) is another line of work that manages to transform a layer to multiple layers with the represented function intact. Net2net (Chen et al., 2015) is a successful application of this idea to knowledge transfer. Nevertheless, when applied to optimization procedure, it is unclear whether network morphism can preserve general optimization flow. Degraded performance is reported when comes to practical training (Wen et al., 2019).

Similar ideas can also be discovered in many network architectures, including progressive growing of GAN (Karras et al., 2017), Adaptive Computation Time (Graves, 2016; Jernite et al., 2016) for RNN, etc. Neural architecture search (NAS) (Stanley & Miikkulainen, 2002; Zoph & Le, 2016) is also a generic way to conduct architecture optimization. Typically it aims to improve the inference performance at the cost of more expensive training, while our proposed LipGrow aims to accelerate training.

Lipschitz Constant in Deep Neural Networks. Lipschitz constant is widely discussed in learning theory, especially for deep learning which often incorporates a large composition of functions. Towards Lipschitz continuity, general stability is often a major concern, which involves adversarial robustness (Cisse et al., 2017), generalizability (Bartlett et al., 2017), stability of GAN (Qi, 2019), hype-parameter insensitivity (Gouk et al., 2018), to name a few.

Specific to residual networks, Behrmann et al. (2018) enforce the Lipschitz constraint so as to extend ResNet to the generative model. This work offers a similar view of ResNet as an ODE discretization.

7. Conclusions and Future Work

In this paper, we study the depth of residual networks and explore to increase it during training in an adaptive way, so that we can reduce the total training time while retaining the model capacity and performance. From a neural ODE perspective, we discuss the global error with respect to the optimal ODE, and elucidate the contributions of capacity augmentation and parameter optimization in a network grow scheme. We also present the temporal error as the signal of network growth to control the growing risk. Thereby we propose LipGrow, leveraging the theoretical analyses to guide the network growth. Extensive experiments demonstrate that LipGrow can achieve better or comparable performance while reducing $\sim 50\%$ of training time.

For future work, potential improvements of our algorithm can be explored. For example, although the decision of LipGrow is based on theoretical derivation, it is still a greedy algorithm focusing on the temporal risk and can be trapped in the local optimal. Thus, it would be beneficial to defining an unified objective incorporating both the performance measure and training cost, upon which an algorithm that maximizes the performance yield bounded by pre-specified training budgets is possible. Last but not least, theoretical guidance presented in our work can be extended to other practical applications involving functional transformation such as Neural Architecture Search and network pruning. Hopefully we can step towards better understanding of the current heuristics in AutoML community, and provides theoretically sound suggestions to optimize the performance.

References

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

Avelin, B. and Nyström, K. Neural odes as the deep limit of resnets with constant weights, 2019.

Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pp. 6240–6249, 2017.

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2018.

Chang, B., Meng, L., Haber, E., Tung, F., and Begert, D. Multi-level residual networks from dynamical systems view. *ArXiv*, abs/1710.10348, 2017.

Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Chang, B., Chen, M., Haber, E., and Chi, E. H. Antisymmetricrnn: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.

Chen, T., Goodfellow, I., and Shlens, J. Net2net: Accelerating learning via knowledge transfer, 2015.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Ciccone, M., Gallieri, M., Masci, J., Osendorfer, C., and Gomez, F. Nais-net: Stable deep networks from non-autonomous differential equations. In *Advances in Neural Information Processing Systems*, pp. 3025–3035, 2018.

Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., and Usunier, N. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 854–863. JMLR. org, 2017.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes. In *NeurIPS*, 2019.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Gouk, H., Frank, E., Pfahringer, B., and Cree, M. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368*, 2018.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Istrate, R., Malossi, A. C. I., Bekas, C., and Nikolopoulos, D. Incremental training of deep convolutional neural networks. *arXiv preprint arXiv:1803.10232*, 2018.

Jernite, Y., Grave, E., Joulin, A., and Mikolov, T. Variable computation in recurrent neural networks. *arXiv preprint arXiv:1611.06188*, 2016.

Jin, H., Song, Q., and Hu, X. Auto-keras: Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 2018.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Liu, L., Liu, X., Gao, J., Chen, W., and Han, J. Understanding the difficulty of training transformers. *ArXiv*, abs/2004.08249, 2020.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Lu, Y., Zhong, A., Li, Q., and Dong, B. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*, 2017.

Lu, Y., Li, Z., He, D., Sun, Z., Dong, B., Qin, T., Wang, L., and Liu, T.-Y. Understanding and improving transformer from a multi-particle dynamic system point of view. *arXiv preprint arXiv:1906.02762*, 2019.

Lu, Y., Ma, C., Lu, Y., Lu, J., and Ying, L. A mean-field analysis of deep resnet and beyond: Towards provable optimization via overparameterization from depth, 2020.

Müller, J. On the space-time expressivity of resnets, 2019.

Qi, G.-J. Loss-sensitive generative adversarial networks on lipschitz densities. *International Journal of Computer Vision*, pp. 1–23, 2019.

Ruthotto, L. and Haber, E. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pp. 1–13, 2018.

Sedghi, H., Gupta, V., and Long, P. M. The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*, 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Smith, L. N., Hand, E. M., and Doster, T. Gradual dropin of layers to train very deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4763–4771, 2016.

Stanley, K. O. and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Thorpe, M. and van Gennip, Y. Deep limits of residual neural networks, 2018.

Tsuzuku, Y., Sato, I., and Sugiyama, M. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 6541–6550, 2018.

Wei, T., Wang, C., Rui, Y., and Chen, C. W. Network morphism. In *International Conference on Machine Learning*, pp. 564–572, 2016.

Wei, T., Wang, C., and Chen, C. W. Modularized morphing of neural networks. *arXiv preprint arXiv:1701.03281*, 2017.

Weinan, E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

Wen, W., Yan, F., and Li, H. Autogrow: Automatic layer growing in deep convolutional networks, 2019.

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

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.